Makefile


```
  1: # fiwix/Makefile
  2: #
  3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
  4: # Distributed under the terms of the Fiwix License.
  5: #
  6:
  7: TOPDIR  := $(shell if [ "$$PWD" != "" ] ; then echo $$PWD ; else pwd ; fi)
  8: INCLUDE = $(TOPDIR)/include
  9:
 10: ARCH = -m32
 11: CPU = -march=i386
 12:
 13: DEBUG = -D__DEBUG__ #-D__2DEBUG__
 14:
 15: CC = $(CROSS_COMPILE)gcc $(ARCH) $(CPU) #$(DEBUG)
 16: LD = $(CROSS_COMPILE)ld
 17:
 18: CFLAGS = -I$(INCLUDE) -O2 -ffreestanding -Wall -Wstrict-prototypes #-Wextra
 19: LDFLAGS = -m elf_i386 -nostartfiles -nostdlib -nodefaultlibs -nostdinc
 20:
 21: DIRS = kernel kernel/syscalls mm fs drivers/block drivers/char lib
 22: OBJS = kernel/kernel.o kernel/syscalls/syscalls.o mm/mm.o fs/fs.o \
 23:        drivers/block/block.o drivers/char/char.o lib/lib.o
 24:
 25: export CC LD CFLAGS LDFLAGS INCLUDE
 26:
 27: all:
 28:         @echo "#define UTS_VERSION \"`date`\"" > include/fiwix/version.h
 29:         @for n in $(DIRS) ; do (cd $$n ; $(MAKE)) ; done
 30:         $(LD) -N -T fiwix.ld $(LDFLAGS) $(OBJS) -o fiwix
 31:         nm fiwix | sort | gzip -9c > System.map.gz
 32:
 33: clean:
 34:         @for n in $(DIRS) ; do (cd $$n ; $(MAKE) clean) ; done
 35:         rm -f *.o fiwix System.map.gz
 36:
 37: floppy:
 38:         mkfs.ext2 -m 0 /dev/fd0
 39:         mount -t ext2 /dev/fd0 /mnt/floppy
 40:         @mkdir -p /mnt/floppy/boot/grub
 41:         @echo "(fd0)   /dev/fd0" > /mnt/floppy/boot/grub/device.map
 42:         @grub-install --root-directory=/mnt/floppy /dev/fd0
 43:         @tools/MAKEBOOTDISK
 44:         @cp -prf tools/etc/* /mnt/floppy/etc
 45:         @cp fiwix /mnt/floppy/boot
 46:         @cp System.map.gz /mnt/floppy/boot
 47:         @cp tools/install.sh /mnt/floppy/sbin
 48:         umount /mnt/floppy
 49:
 50: floppy_update:
 51:         mount -t ext2 /dev/fd0 /mnt/floppy
 52:         cp fiwix /mnt/floppy/boot
 53:         cp System.map.gz /mnt/floppy/boot
 54:         umount /mnt/floppy
 55:
 56:
```

fiwix.ld


```
/*
 * Linker script for the Fiwix kernel (3GB user / 1GB kernel).
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

OUTPUT_FORMAT("elf32-i386")
OUTPUT_ARCH(i386)
ENTRY(start)            /* entry point */
vaddr = 0xC0000000;     /* virtual base address at 3GB */
paddr = 0x100000;       /* physical address at 1MB */

/* define output sections */
SECTIONS
{
        . = paddr;

        /* kernel setup code */
        .setup ALIGN(4096) :
        {
                *(.setup)
        }

        . += vaddr;

        /* kernel code */
        .text : AT(ADDR(.text) - vaddr)
        {
                *(.text)
        }
        _etext = .;

        /* initialized data */
        .data ALIGN(4096) : AT(ADDR(.data) - vaddr)
        {
                *(.data)
                *(.rodata*)
        }
        _edata = .;

        /* uninitialized data */
        .bss ALIGN(4096) : AT(ADDR(.bss) - vaddr)
        {
                *(COMMON*)
                *(.bss*)
        }
        _end = .;

        /* remove information not needed */
        /DISCARD/ :
        {
                *(.eh_frame)
        }
}
```

kernel/boot.S


```
/*
 * fiwix/kernel/boot.S
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/const.h>

#define ASM     1               /* GRUB stuff */
#include <fiwix/multiboot.h>

#define CR0_MP  0x00000002      /* CR0 bit-01 MP (Monitor Coprocessor) */
#define CR0_NE  0x00000020      /* CR0 bit-05 NE (Numeric Error) */
#define CR0_WP  0x00010000      /* CR0 bit-16 WP (Write Protect) */
#define CR0_AM  0x00040000      /* CR0 bit-18 AM (Alignment Mask) */
#define CR0_PG  0x80000000      /* CR0 bit-31 PG (Paging) */

.section .setup, "a"            /* "a" attribute means Allocatable section */

.align 4
tmp_gdtr:
        .word ((3 * 8) - 1)
        .long tmp_gdt

.align 4
tmp_gdt:
        /* NULL DESCRIPTOR */
        .word   0x0000
        .word   0x0000
        .word   0x0000
        .word   0x0000

        /* KERNEL CODE */
        .word   0xFFFF          /* segment limit 15-00 */
        .word   0x0000          /* base address 15-00 */
        .byte   0x00            /* base address 23-16 */
        .byte   0x9A            /* P=1 DPL=00 S=1 TYPE=1010 (exec/read) */
        .byte   0xCF            /* G=1 DB=1 0=0 AVL=0 SEGLIM=1111 */
        .byte   0x40            /* base address 31-24 */

        /* KERNEL DATA */
        .word   0xFFFF          /* segment limit 15-00 */
        .word   0x0000          /* base address 15-00 */
        .byte   0x00            /* base address 23-16 */
        .byte   0x92            /* P=1 DPL=00 S=1 TYPE=0010 (read/write) */
        .byte   0xCF            /* G=1 DB=1 0=0 AVL=0 SEGLIM=1111 */
        .byte   0x40            /* base address 31-24 */


.text

.globl start; start:
        cli
        jmp     multiboot_entry

.align 4
multiboot_header:                       /* multiboot header */
        .long   MULTIBOOT_HEADER_MAGIC  /* magic */
        .long   MULTIBOOT_HEADER_FLAGS  /* flags */
        /* checksum */
        .long   -(MULTIBOOT_HEADER_MAGIC + MULTIBOOT_HEADER_FLAGS)

#ifndef __ELF__
        .long   multiboot_header        /* header_addr */
        .long   _start                  /* load_addr */
        .long   _edata                  /* load_end_addr */
```

kernel/boot.S


```
   68:         .long   _end                    /* bss_end_addr */
   69:         .long   multiboot_entry         /* entry_addr */
   70: #endif /* ! __ELF__ */
   71:
   72: /*
   73:  * We use the CX register in order to keep intact the values in AX and BX
   74:  * registers, since they are holding the Multiboot values 'magic' and 'info'
   75:  * respectively.
   76:  */
   77: multiboot_entry:
   78:         lgdt    tmp_gdtr                /* load GDTR with the temporary GDT */
   79:         movw    $KERNEL_DS, %cx
   80:         movw    %cx, %ds
   81:         movw    %cx, %es
   82:         movw    %cx, %fs
   83:         movw    %cx, %gs
   84:         movw    %cx, %ss
   85:         ljmp    $KERNEL_CS, $1f
   86: 1:
   87:
   88:
   89: /*
   90:  * WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING !!!
   91:  * ---------------------------------------------------------------------------
   92:  * The minimal page directory of 4MB only works if the in-memory size of the
   93:  * kernel is lesser than 3MB. If you need more space go to the setup_minmem()
   94:  * function and set the 'mb4' variable accordingly.
   95:  *
   96:  * In order to know the current size of the Fiwix kernel, just follow this:
   97:  *
   98:  * # readelf -l fiwix
   99:  *   Elf file type is EXEC (Executable file)
  100:  *   Entry point 0xc0100020
  101:  *   There are 2 program headers, starting at offset 52
  102:  *
  103:  *   Program Headers:
  104:  *     Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  105:  *     LOAD           0x000074 0x00100000 0x00100000 0x00020 0x00020 R   0x4
  106:  *     LOAD           0x0000a0 0xc0100020 0x00100020 0x33f8c 0x859a0 RWE 0x20
  107:  *                                        check this value --> ^^^^^^^
  108:  */
  109:         movl    $0xC0010000, %esp       /* default stack address */
  110:         pushl   $0                      /* reset EFLAGS */
  111:         popf
  112:
  113:         pushl   %eax                    /* save Multiboot magic value */
  114:         call    setup_minmem            /* setup a minimal page directory */
  115:         movl    %eax, %cr3
  116:
  117:         movl    %cr0, %eax
  118:         andl    $0x00000011, %eax       /* disable all, preserve ET & PE (GRUB)
*/
  119:         orl     $CR0_PG, %eax           /* enable PG (Paging) */
  120:         orl     $CR0_AM, %eax           /* enable AM (Alignment Mask) */
  121:         orl     $CR0_WP, %eax           /* enable WP (Write Protect) */
  122:         orl     $CR0_NE, %eax           /* enable NE (Numeric Error) */
  123:         orl     $CR0_MP, %eax           /* enable MP (Monitor Coprocessor) */
  124:         movl    %eax, %cr0
  125:
  126:         call    bss_init                /* initialize BSS segment */
  127:         call    gdt_init                /* setup and load the definitive GDT */
  128:
  129:         pushl   %ebx                    /* save Multiboot info structure */
  130:         call    get_last_boot_addr
  131:         add     $4, %esp
  132:         popl    %ecx                    /* restore Multiboot magic value */
  133:         andl    $0xFFFFF000, %eax       /* page aligned */
```

kernel/boot.S


```
134:         addl    $0x3000, %eax           /* 2 whole pages for kernel stack */
135:         subl    $4, %eax
136:         movl    %eax, %esp              /* set kernel stack */
137:
138:         pushl   %esp                    /* save kernel stack address */
139:         pushl   %ebx                    /* save Multiboot info structure */
140:         pushl   %ecx                    /* save Multiboot magic value */
141:         call    start_kernel
142:
143: .align 4
144: .globl cpu_idle; cpu_idle:
145:         hlt
146:         jmp     cpu_idle
147:
148: .align 4
149: .org 0x1000
150: .globl  _fdc_transfer_area
151: _fdc_transfer_area:     .fill 512*2*18,1,0
```

kernel/cmos.c


```
 1: /*
 2:  * fiwix/kernel/cmos.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/asm.h>
 9: #include <fiwix/cmos.h>
10:
11: int cmos_update_in_progress(void)
12: {
13:         return(cmos_read(CMOS_STATA) & CMOS_STATA_UIP);
14: }
15:
16: unsigned char cmos_read_date(unsigned char addr)
17: {
18:         /* make sure an update isn't in progress */
19:         while(cmos_update_in_progress());
20:
21:         if(!(cmos_read(CMOS_STATB) & CMOS_STATB_DM)) {
22:                 return BCD2BIN(cmos_read(addr));
23:         }
24:         return cmos_read(addr);
25: }
26:
27: void cmos_write_date(unsigned char addr, unsigned char value)
28: {
29:         /* make sure an update isn't in progress */
30:         while(cmos_update_in_progress());
31:
32:         if(!(cmos_read(CMOS_STATB) & CMOS_STATB_DM)) {
33:                 cmos_write(addr, BIN2BCD(value));
34:         }
35:         cmos_write(addr, value);
36: }
37:
38: unsigned char cmos_read(unsigned char addr)
39: {
40:         unsigned long int flags;
41:
42:         SAVE_FLAGS(flags); CLI();
43:         outport_b(CMOS_INDEX, addr);
44:         RESTORE_FLAGS(flags);
45:
46:         return inport_b(CMOS_DATA);
47: }
48:
49: void cmos_write(unsigned char addr, unsigned char value)
50: {
51:         unsigned long int flags;
52:
53:         SAVE_FLAGS(flags); CLI();
54:         outport_b(CMOS_INDEX, addr);
55:         outport_b(CMOS_DATA, value);
56:         RESTORE_FLAGS(flags);
57: }
```

kernel/core386.S


```
    1: /*
    2:  * fiwix/kernel/core386.S
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/const.h>
    9: #include <fiwix/unistd.h>
   10:
   11: #define CR0_MP  ~(0x00000002)   /* CR0 bit-01 MP (Monitor Coprocessor) */
   12: #define CR0_EM  0x00000004      /* CR0 bit-02 EM (Emulation) */
   13:
   14: #define SS_RPL3         0x03    /* Request Privilege Level 3 */
   15:
   16: #define GS              0x00
   17: #define FS              0x04
   18: #define ES              0x08
   19: #define DS              0x0C
   20: #define EDI             0x10    /* \                            */
   21: #define ESI             0x14    /* |                            */
   22: #define EBP             0x18    /* |                            */
   23: #define ESP             0x1C    /* | saved by                   */
   24: #define EBX             0x20    /* | 'pusha'                    */
   25: #define EDX             0x24    /* |                            */
   26: #define ECX             0x28    /* |                            */
   27: #define EAX             0x2C    /* /                            */
   28: #define ERR             0x30    /*   error code or padding      */
   29: #define EIP             0x34    /* \                            */
   30: #define CS              0x38    /* | saved by processor         */
   31: #define FLAGS           0x3C    /* /                            */
   32: #define OLDESP          0x40    /* \ saved by processor on      */
   33: #define OLDSS           0x44    /* / privilege level change     */
   34:
   35: #define SAVE_ALL                                                \
   36:         pushal                                                  ;\
   37:         pushl   %ds                                             ;\
   38:         pushl   %es                                             ;\
   39:         pushl   %fs                                             ;\
   40:         pushl   %gs
   41:
   42: #define EXCEPTION(exception)                                    \
   43:         pushl   $exception                                      ;\
   44:         call    trap_handler                                    ;\
   45:         addl    $4, %esp
   46:
   47: #define IRQ(irq)                                                \
   48:         pushl   $irq                                            ;\
   49:         call    irq_handler                                     ;\
   50:         addl    $4, %esp
   51:
   52: /*
   53:  * Check only for signals if we are returning from user-mode. issig() function
   54:  * returns 1 if there are signals or 0 otherwise. If there are signals psig()
   55:  * function is called with the stack as the first argument.
   56:  */
   57: #define CHECK_SIGNALS                                           \
   58:         cmpw    $KERNEL_CS, CS(%esp)                            ;\
   59:         je      1f                                              ;\
   60:         call    issig                                           ;\
   61:         cmpl    $0, %eax                                        ;\
   62:         je      1f                                              ;\
   63:         movl    %esp, %eax                                      ;\
   64:         pushl   %eax                                            ;\
   65:         call    psig                                            ;\
   66:         addl    $4, %esp                                        ;\
   67: 1:
```

kernel/core386.S


```
 68:
 69: #define SCHEDULE                                                    \
 70:         cmpl    $0, need_resched                                    ;\
 71:         je      1f                                                  ;\
 72:         call    do_sched                                            ;\
 73: 1:
 74:
 75: #define BOTTOM_HALVES                                               \
 76: /*      sti                                                 */      ;\
 77:         call    do_bh
 78:
 79: #define RESTORE_ALL                                                 \
 80:         popl    %gs                                                 ;\
 81:         popl    %fs                                                 ;\
 82:         popl    %es                                                 ;\
 83:         popl    %ds                                                 ;\
 84:         popal                                                       ;\
 85:         addl    $4, %esp        # suppress error code (or padding) from stack
 86:
 87:
 88: .text
 89:
 90: .align 4
 91: .globl except0; except0:        # DIVIDE ERROR
 92:         pushl   $0              # save simulated error code to stack
 93:         SAVE_ALL
 94:         EXCEPTION(0x0)
 95:         CHECK_SIGNALS
 96:         SCHEDULE
 97:         BOTTOM_HALVES
 98:         RESTORE_ALL
 99:         iret
100:
101: .align 4
102: .globl except1; except1:        # DEBUG
103:         pushl   $0              # save simulated error code to stack
104:         SAVE_ALL
105:         EXCEPTION(0x1)
106:         CHECK_SIGNALS
107:         SCHEDULE
108:         BOTTOM_HALVES
109:         RESTORE_ALL
110:         iret
111:
112: .align 4
113: .globl except2; except2:        # NMI INTERRUPT
114:         pushl   $0              # save simulated error code to stack
115:         SAVE_ALL
116:         EXCEPTION(0x2)
117:         CHECK_SIGNALS
118:         SCHEDULE
119:         BOTTOM_HALVES
120:         RESTORE_ALL
121:         iret
122:
123: .align 4
124: .globl except3; except3:        # BREAKPOINT INT3
125:         pushl   $0              # save simulated error code to stack
126:         SAVE_ALL
127:         EXCEPTION(0x3)
128:         CHECK_SIGNALS
129:         SCHEDULE
130:         BOTTOM_HALVES
131:         RESTORE_ALL
132:         iret
133:
134: .align 4
```

kernel/core386.S


```
135: .globl except4; except4:        # OVERFLOW
136:         pushl   $0              # save simulated error code to stack
137:         SAVE_ALL
138:         EXCEPTION(0x4)
139:         CHECK_SIGNALS
140:         SCHEDULE
141:         BOTTOM_HALVES
142:         RESTORE_ALL
143:         iret
144:
145: .align 4
146: .globl except5; except5:        # BOUND
147:         pushl   $0              # save simulated error code to stack
148:         SAVE_ALL
149:         EXCEPTION(0x5)
150:         CHECK_SIGNALS
151:         SCHEDULE
152:         BOTTOM_HALVES
153:         RESTORE_ALL
154:         iret
155:
156: .align 4
157: .globl except6; except6:        # INVALID OPCODE
158:         pushl   $0              # save simulated error code to stack
159:         SAVE_ALL
160:         EXCEPTION(0x6)
161:         CHECK_SIGNALS
162:         SCHEDULE
163:         BOTTOM_HALVES
164:         RESTORE_ALL
165:         iret
166:
167: .align 4
168: .globl except7; except7:        # NO MATH COPROCESSOR
169:         pushl   $0              # save simulated error code to stack
170:         SAVE_ALL
171:         EXCEPTION(0x7)
172:         clts                    # floating-opcode cached!
173:         CHECK_SIGNALS
174:         SCHEDULE
175:         BOTTOM_HALVES
176:         RESTORE_ALL
177:         iret
178:
179: .align 4
180: .globl except8; except8:        # DOUBLE FAULT
181:         SAVE_ALL
182:         EXCEPTION(0x8)
183:         CHECK_SIGNALS
184:         SCHEDULE
185:         BOTTOM_HALVES
186:         RESTORE_ALL
187:         iret
188:
189: .align 4
190: .globl except9; except9:        # COPROCESSOR SEGMENT OVERRUN
191:         pushl   $0              # save simulated error code to stack
192:         SAVE_ALL
193:         EXCEPTION(0x9)
194:         CHECK_SIGNALS
195:         SCHEDULE
196:         BOTTOM_HALVES
197:         RESTORE_ALL
198:         iret
199:
200: .align 4
201: .globl exceptA; exceptA:        # INVALID TSS
```

kernel/core386.S


```
202:         SAVE_ALL
203:         EXCEPTION(0xA)
204:         CHECK_SIGNALS
205:         SCHEDULE
206:         BOTTOM_HALVES
207:         RESTORE_ALL
208:         iret
209:
210: .align 4
211: .globl exceptB; exceptB:        # SEGMENT NOT PRESENT
212:         SAVE_ALL
213:         EXCEPTION(0xB)
214:         CHECK_SIGNALS
215:         SCHEDULE
216:         BOTTOM_HALVES
217:         RESTORE_ALL
218:         iret
219:
220: .align 4
221: .globl exceptC; exceptC:        # STACK SEGMENT FAULT
222:         SAVE_ALL
223:         EXCEPTION(0xC)
224:         CHECK_SIGNALS
225:         SCHEDULE
226:         BOTTOM_HALVES
227:         RESTORE_ALL
228:         iret
229:
230: .align 4
231: .globl exceptD; exceptD:        # GENERAL PROTECTION FAULT
232:         SAVE_ALL
233:         EXCEPTION(0xD)
234:         CHECK_SIGNALS
235:         SCHEDULE
236:         BOTTOM_HALVES
237:         RESTORE_ALL
238:         iret
239:
240: .align 4
241: .globl exceptE; exceptE:        # PAGE FAULT
242:         SAVE_ALL
243:         EXCEPTION(0xE)
244:         CHECK_SIGNALS
245:         SCHEDULE
246:         BOTTOM_HALVES
247:         RESTORE_ALL
248:         iret
249:
250: .align 4
251: .globl exceptF; exceptF:        # INTEL RESERVED
252:         pushl   $0              # save simulated error code to stack
253:         SAVE_ALL
254:         EXCEPTION(0xF)
255:         CHECK_SIGNALS
256:         SCHEDULE
257:         BOTTOM_HALVES
258:         RESTORE_ALL
259:         iret
260:
261: .globl except10; except10:      # FLOATING POINT ERROR
262:         pushl   $0              # save simulated error code to stack
263:         SAVE_ALL
264:         EXCEPTION(0x10)
265:         CHECK_SIGNALS
266:         SCHEDULE
267:         BOTTOM_HALVES
268:         RESTORE_ALL
```

kernel/core386.S


```
269:         iret
270:
271: .globl except11; except11:      # ALIGNMENT CHECK
272:         EXCEPTION(0x11)
273:         SAVE_ALL
274:         CHECK_SIGNALS
275:         SCHEDULE
276:         BOTTOM_HALVES
277:         RESTORE_ALL
278:         iret
279:
280: .globl except12; except12:      # MACHINE CHECK
281:         pushl   $0              # save simulated error code to stack
282:         SAVE_ALL
283:         EXCEPTION(0x12)
284:         CHECK_SIGNALS
285:         SCHEDULE
286:         BOTTOM_HALVES
287:         RESTORE_ALL
288:         iret
289:
290: .globl except13; except13:      # SIMD FLOATING POINT
291:         pushl   $0              # save simulated error code to stack
292:         SAVE_ALL
293:         EXCEPTION(0x13)
294:         CHECK_SIGNALS
295:         SCHEDULE
296:         BOTTOM_HALVES
297:         RESTORE_ALL
298:         iret
299:
300: .globl except14; except14:      # INTEL RESERVED
301: .globl except15; except15:      # INTEL RESERVED
302: .globl except16; except16:      # INTEL RESERVED
303: .globl except17; except17:      # INTEL RESERVED
304: .globl except18; except18:      # INTEL RESERVED
305: .globl except19; except19:      # INTEL RESERVED
306: .globl except1A; except1A:      # INTEL RESERVED
307: .globl except1B; except1B:      # INTEL RESERVED
308: .globl except1C; except1C:      # INTEL RESERVED
309: .globl except1D; except1D:      # INTEL RESERVED
310: .globl except1E; except1E:      # INTEL RESERVED
311: .globl except1F; except1F:      # INTEL RESERVED
312:         pushl   $0              # save simulated error code to stack
313:         SAVE_ALL
314:         EXCEPTION(0x14)
315:         CHECK_SIGNALS
316:         SCHEDULE
317:         BOTTOM_HALVES
318:         RESTORE_ALL
319:         iret
320:
321: .align 4
322: .globl irq0; irq0:              # TIMER
323:         pushl   $0              # save simulated error code to stack
324:         SAVE_ALL
325:         IRQ(0)
326:         CHECK_SIGNALS
327:         SCHEDULE
328:         BOTTOM_HALVES
329:         RESTORE_ALL
330:         iret
331:
332: .align 4
333: .globl irq1; irq1:              # KEYBOARD
334:         pushl   $0              # save simulated error code to stack
335:         SAVE_ALL
```

kernel/core386.S


```
336:         IRQ(1)
337:         CHECK_SIGNALS
338:         SCHEDULE
339:         BOTTOM_HALVES
340:         RESTORE_ALL
341:         iret
342:
343: .align 4
344: .globl irq2; irq2:             # CASCADE
345:         pushl   $0             # save simulated error code to stack
346:         SAVE_ALL
347:         IRQ(2)
348:         CHECK_SIGNALS
349:         SCHEDULE
350:         BOTTOM_HALVES
351:         RESTORE_ALL
352:         iret
353:
354: .align 4
355: .globl irq3; irq3:             # SERIAL
356:         pushl   $0             # save simulated error code to stack
357:         SAVE_ALL
358:         IRQ(3)
359:         CHECK_SIGNALS
360:         SCHEDULE
361:         BOTTOM_HALVES
362:         RESTORE_ALL
363:         iret
364:
365: .align 4
366: .globl irq4; irq4:             # SERIAL
367:         pushl   $0             # save simulated error code to stack
368:         SAVE_ALL
369:         IRQ(4)
370:         CHECK_SIGNALS
371:         SCHEDULE
372:         BOTTOM_HALVES
373:         RESTORE_ALL
374:         iret
375:
376: .align 4
377: .globl irq5; irq5:
378:         pushl   $0             # save simulated error code to stack
379:         SAVE_ALL
380:         IRQ(5)
381:         CHECK_SIGNALS
382:         SCHEDULE
383:         BOTTOM_HALVES
384:         RESTORE_ALL
385:         iret
386:
387: .align 4
388: .globl irq6; irq6:             # FLOPPY
389:         pushl   $0             # save simulated error code to stack
390:         SAVE_ALL
391:         IRQ(6)
392:         CHECK_SIGNALS
393:         SCHEDULE
394:         BOTTOM_HALVES
395:         RESTORE_ALL
396:         iret
397:
398: .align 4
399: .globl irq7; irq7:
400:         pushl   $0             # save simulated error code to stack
401:         SAVE_ALL
402:         IRQ(7)
```

kernel/core386.S


```
403:         CHECK_SIGNALS
404:         SCHEDULE
405:         BOTTOM_HALVES
406:         RESTORE_ALL
407:         iret
408:
409: .align 4
410: .globl irq8; irq8:
411:         pushl   $0              # save simulated error code to stack
412:         SAVE_ALL
413:         IRQ(8)
414:         CHECK_SIGNALS
415:         SCHEDULE
416:         BOTTOM_HALVES
417:         RESTORE_ALL
418:         iret
419:
420: .align 4
421: .globl irq9; irq9:
422:         pushl   $0              # save simulated error code to stack
423:         SAVE_ALL
424:         IRQ(9)
425:         CHECK_SIGNALS
426:         SCHEDULE
427:         BOTTOM_HALVES
428:         RESTORE_ALL
429:         iret
430:
431: .align 4
432: .globl irq10; irq10:
433:         pushl   $0              # save simulated error code to stack
434:         SAVE_ALL
435:         IRQ(10)
436:         CHECK_SIGNALS
437:         SCHEDULE
438:         BOTTOM_HALVES
439:         RESTORE_ALL
440:         iret
441:
442: .align 4
443: .globl irq11; irq11:
444:         pushl   $0              # save simulated error code to stack
445:         SAVE_ALL
446:         IRQ(11)
447:         CHECK_SIGNALS
448:         SCHEDULE
449:         BOTTOM_HALVES
450:         RESTORE_ALL
451:         iret
452:
453: .align 4
454: .globl irq12; irq12:
455:         pushl   $0              # save simulated error code to stack
456:         SAVE_ALL
457:         IRQ(12)
458:         CHECK_SIGNALS
459:         SCHEDULE
460:         BOTTOM_HALVES
461:         RESTORE_ALL
462:         iret
463:
464: .align 4
465: .globl irq13; irq13:
466:         pushl   $0              # save simulated error code to stack
467:         SAVE_ALL
468:         IRQ(13)
469:         CHECK_SIGNALS
```

kernel/core386.S


```
  470:         SCHEDULE
  471:         BOTTOM_HALVES
  472:         RESTORE_ALL
  473:         iret
  474:
  475: .align 4
  476: .globl irq14; irq14:           # IDE Primary
  477:         pushl   $0             # save simulated error code to stack
  478:         SAVE_ALL
  479:         IRQ(14)
  480:         CHECK_SIGNALS
  481:         SCHEDULE
  482:         BOTTOM_HALVES
  483:         RESTORE_ALL
  484:         iret
  485:
  486: .align 4
  487: .globl irq15; irq15:           # IDE Secondary
  488:         pushl   $0             # save simulated error code to stack
  489:         SAVE_ALL
  490:         IRQ(15)
  491:         CHECK_SIGNALS
  492:         SCHEDULE
  493:         BOTTOM_HALVES
  494:         RESTORE_ALL
  495:         iret
  496:
  497: .align 4
  498: .globl unknown_irq; unknown_irq:
  499:         pushl   $0             # save simulated error code to stack
  500:         SAVE_ALL
  501:         IRQ(-1)
  502:         RESTORE_ALL
  503:         iret
  504:
  505: .align 4
  506: .globl switch_to_user_mode; switch_to_user_mode:
  507:         cli
  508:         xorl    %eax, %eax             # initialize %eax
  509:         movl    %eax, %ebx             # initialize %ebx
  510:         movl    %eax, %ecx             # initialize %ecx
  511:         movl    %eax, %edx             # initialize %edx
  512:         movl    %eax, %esi             # initialize %esi
  513:         movl    %eax, %edi             # initialize %edi
  514:         movl    %eax, %ebp             # initialize %ebp
  515:         movl    $(USER_DS | SS_RPL3), %eax
  516:         movw    %ax, %ds
  517:         movw    %ax, %es
  518:         movw    %ax, %fs
  519:         movw    %ax, %gs
  520:         pushl   %eax
  521:         pushl   $KERNEL_BASE_ADDR - 4   # user stack address
  522:         pushl   $0x202                  # initialize eflags (Linux 2.2 = 0x292)
  523:         popfl
  524:         pushfl
  525:         movl    $(USER_CS | SS_RPL3), %eax
  526:         pushl   %eax
  527:         pushl   $KERNEL_BASE_ADDR - 0x1000      # go to init_trampoline() in use
r mode
  528:         iret
  529:
  530: .align 4
  531: .globl sighandler_trampoline; sighandler_trampoline:
  532:         pushl   %eax
  533:         call    *%ecx
  534:         popl    %ebx
  535:
```

kernel/core386.S


```
536:         movl    $SYS_sigreturn, %eax
537:         int     $0x80
538:
539:         # never reached, otherwise call sys_exit()
540:         movl    $SYS_exit, %eax
541:         int     $0x80
542:         ret
543: .align 4
544: .globl end_sighandler_trampoline; end_sighandler_trampoline:
545:         nop
546:
547: .align 4
548: .globl syscall; syscall:                # SYSTEM CALL ENTRY
549:         pushl   %eax                    # save the system call number
550:         SAVE_ALL
551:
552:         pushl   %edi                    # \ 5th parameter
553:         pushl   %esi                    # | 4th parameter
554:         pushl   %edx                    # | 3rd parameter
555:         pushl   %ecx                    # | 2nd parameter
556:         pushl   %ebx                    # / 1st parameter
557:         pushl   %eax                    # system call number
558:         call    do_syscall
559:         addl    $24, %esp               # suppress all 6 pushl from the stack
560:         movl    %eax, EAX(%esp)         # save the return value
561:
562:         CHECK_SIGNALS
563:         SCHEDULE
564:         BOTTOM_HALVES
565: .align 4
566: .globl return_from_syscall; return_from_syscall:
567:         RESTORE_ALL
568:         iret
569:
570: .align 4
571: .globl do_switch; do_switch:
572:         movl    %esp, %ebx
573:         pushal
574:         pushfl
575:         movl    0x4(%ebx), %eax         # save ESP to 'prev->tss.esp'
576:         movl    %esp, (%eax)
577:         movl    0x8(%ebx), %eax         # save EIP to 'prev->tss.eip'
578:         movl    $1f, (%eax)
579:         movl    0xC(%ebx), %esp         # load 'next->tss.esp' into ESP
580:         pushl   0x10(%ebx)              # push 'next->tss.eip' into ESP
581:         movl    0x14(%ebx), %eax        # load 'next->tss.cr3' into CR3
582:         ltr     0x18(%ebx)              # load TSS
583:         movl    %eax, %cr3
584:         ret
585: 1:
586:         popfl
587:         popal
588:
589: .align 4
590: .globl cpuid; cpuid:
591:         pushl   %ebp
592:         movl    %esp, %ebp
593:         pushl   %edi
594:         pushl   %esi
595:         pushl   %ebx
596:
597:         pushf
598:         pop     %eax                    # put original EFLAGS in EAX
599:         mov     %eax, %ecx              # save original EFLAGS in ECX
600:         xor     $0x200000, %eax         # change bit 21 (ID) in EFLAGS
601:         push    %eax                    # save new EFLAGS on stack
602:         popf                            # replace current EFLAGS
```

kernel/core386.S


```
603:          pushf
604:          pop     %eax                    # put EFLAGS in EAX
605:          cmp     %ecx, %eax              # compare if both EFLAGS are equal
606:
607:          je      test386                 # can't toggle ID bit, no CPUID
608:          xor     %ebx, %ebx              # CPUID available, will return 0
609:          jmp     end_cpuid
610:
611: test386:
612:          mov     %ecx, %eax              # get original EFLAGS
613:          xor     $0x40000, %eax          # change bit 18 (AC) in EFLAGS
614:          push    %eax                    # save new EFLAGS on stack
615:          popf                            # replace current EFLAGS
616:          pushf
617:          pop     %eax
618:          cmp     %ecx, %eax              # compare if both EFLAGS are equal
619:          movb    $3, %bl                 # looks like an i386, return 3
620:          je      end_cpuid
621:          movb    $4, %bl                 # otherwise is an old i486, return 4
622:
623: end_cpuid:
624:          push    %ecx                    # push original EFLAGS
625:          popf                            # restore original EFLAGS
626:          xor     %eax, %eax
627:          movb    %bl, %al                # put return value to AL
628:
629:          popl    %ebx
630:          popl    %esi
631:          popl    %edi
632:          popl    %ebp
633:          ret
634:
635: .align 4
636: .globl getfpu; getfpu:
637:          pushl   %ebp
638:          movl    %esp, %ebp
639:          pushl   %edi
640:          pushl   %esi
641:          pushl   %ebx
642:
643:          fninit
644:          movl    $0x5a5a, _fpstatus
645:          fnstsw  _fpstatus
646:          movl    _fpstatus, %eax
647:          cmp     $0, %al
648:          movl    $0, _fpstatus
649:          jne     end_getfpu
650:
651: check_control_word:
652:          fnstcw  _fpstatus
653:          movl    _fpstatus, %eax
654:          andl    $0x103f, %eax
655:          cmp     $0x3f, %ax
656:          movl    $0, _fpstatus
657:          jne     end_getfpu
658:          movl    $1, _fpstatus
659:
660: end_getfpu:
661:          movl    _fpstatus, %eax
662:          cmp     $0, %al
663:          jne     1f                      # return if there is a coprocessor
664:          movl    %cr0, %eax              # otherwise (no math processor):
665:          orl     $CR0_EM, %eax           # - set   EM (Emulation)
666:          andl    $CR0_MP, %eax           # - clear MP (Monitor Coprocessor)
667:          movl    %eax, %cr0
668:          movl    $0, %eax
669: 1:
```

kernel/core386.S


```
670:          popl    %ebx
671:          popl    %esi
672:          popl    %edi
673:          popl    %ebp
674:          ret
675:
676: .align 4
677: .globl vendor_id; vendor_id:
678:          pushl   %ebp
679:          movl    %esp, %ebp
680:          pushl   %ebx
681:          pushl   %edx
682:          pushl   %ecx
683:
684:          mov     $0, %eax
685:          cpuid
686:          movl    %ebx, _vendorid         # save the 12 bytes of vendor ID string
687:          movl    %edx, _vendorid+4
688:          movl    %ecx, _vendorid+8
689:
690:          popl    %ecx
691:          popl    %edx
692:          popl    %ebx
693:          popl    %ebp
694:          ret                             # EAX returns the highest CPUID value
695:
696: .align 4
697: .globl signature_flags; signature_flags:
698:          pushl   %ebp
699:          movl    %esp, %ebp
700:          pushl   %edi
701:          pushl   %esi
702:          pushl   %ebx
703:          pushl   %edx
704:
705:          mov     $1, %eax
706:          cpuid
707:          movl    %eax, _cpusignature     # signature (model and stepping)
708:          movl    %ebx, _brandid          # misc. information
709:          movl    %edx, _cpuflags         # feature flags
710:          shrl    $8, %eax
711:          andl    $0xF, %eax
712:          movl    %eax, _cputype          # family
713:
714:          popl    %edx
715:          popl    %ebx
716:          popl    %esi
717:          popl    %edi
718:          popl    %ebp
719:          ret
720:
721: .align 4
722: .globl brand_str; brand_str:
723:          pushl   %ebp
724:          movl    %esp, %ebp
725:          pushl   %edi
726:          pushl   %esi
727:          pushl   %ebx
728:
729:          movl    $0x80000000, %eax
730:          cpuid
731:          cmp     $0x80000000, %eax       # check if brand string is supported
732:          jbe     no_brand_str
733:          movl    $0x80000002, %eax       # get first 16 bytes of brand string
734:          cpuid
735:          movl    %eax, _brandstr
736:          movl    %ebx, _brandstr+4
```

kernel/core386.S


```
737:          movl    %ecx, _brandstr+8
738:          movl    %edx, _brandstr+12
739:          movl    $0x80000003, %eax       # get more 16 bytes of brand string
740:          cpuid
741:          movl    %eax, _brandstr+16
742:          movl    %ebx, _brandstr+20
743:          movl    %ecx, _brandstr+24
744:          movl    %edx, _brandstr+28
745:          movl    $0x80000004, %eax       # get last 16 bytes of brand string
746:          cpuid
747:          movl    %eax, _brandstr+32
748:          movl    %ebx, _brandstr+36
749:          movl    %ecx, _brandstr+40
750:          movl    %edx, _brandstr+44
751:          jmp     end_brand_str
752:
753: no_brand_str:
754:          movl    $1, %eax
755:
756: end_brand_str:
757:          movl    $0, %eax
758:          popl    %ebx
759:          popl    %esi
760:          popl    %edi
761:          popl    %ebp
762:          ret
763:
764: .align 4
765: .globl tlbinfo; tlbinfo:
766:          pushl   %edx
767:          pushl   %ecx
768:          mov     $2, %eax
769:          cpuid
770:          movl    %eax, _tlbinfo_eax      # store cache information
771:          movl    %ebx, _tlbinfo_ebx
772:          movl    %edx, _tlbinfo_ecx
773:          movl    %ecx, _tlbinfo_edx
774:          popl    %ecx
775:          popl    %edx
776:          ret
777:
778: .align 4
779: .globl inport_b; inport_b:
780:          pushl   %ebp
781:          movl    %esp, %ebp
782:
783:          movw    0x08(%ebp), %dx         # port addr
784:          inb     %dx, %al
785:
786:          jmp     1f                      # recovery time
787: 1:       jmp     1f                      # recovery time
788: 1:       popl    %ebp
789:          ret
790:
791: .align 4
792: .globl inport_w; inport_w:
793:          pushl   %ebp
794:          movl    %esp, %ebp
795:
796:          movw    0x08(%ebp), %dx         # port addr
797:          inw     %dx, %ax
798:
799:          jmp     1f                      # recovery time
800: 1:       jmp     1f                      # recovery time
801: 1:       popl    %ebp
802:          ret
803:
```

kernel/core386.S


```
804: .align 4
805: .globl inport_sw; inport_sw:
806:         pushl   %ebp
807:         movl    %esp, %ebp
808:         pushl   %edx
809:         pushl   %edi
810:         pushl   %ecx
811:
812:         cld
813:         mov     0x8(%ebp), %edx         # port addr
814:         mov     0xC(%ebp), %edi         # dest
815:         mov     0x10(%ebp), %ecx        # count
816:         rep
817:         insw
818:
819:         popl    %ecx
820:         popl    %edi
821:         popl    %edx
822:         popl    %ebp
823:         ret
824:
825: .align 4
826: .globl outport_b; outport_b:
827:         pushl   %ebp
828:         movl    %esp, %ebp
829:
830:         movw    0x8(%ebp), %dx          # port addr
831:         movb    0xC(%ebp), %al          # data
832:         outb    %al, %dx
833:
834:         jmp     1f                      # recovery time
835: 1:      jmp     1f                      # recovery time
836: 1:      popl    %ebp
837:         ret
838:
839: .align 4
840: .globl outport_w; outport_w:
841:         pushl   %ebp
842:         movl    %esp, %ebp
843:
844:         movw    0x8(%ebp), %dx          # port addr
845:         movw    0xC(%ebp), %ax          # data
846:         outw    %ax, %dx
847:
848:         jmp     1f                      # recovery time
849: 1:      jmp     1f                      # recovery time
850: 1:      popl    %ebp
851:         ret
852:
853: .align 4
854: .globl outport_sw; outport_sw:
855:         pushl   %ebp
856:         movl    %esp, %ebp
857:         pushl   %edx
858:         pushl   %esi
859:         pushl   %ecx
860:
861:         cld
862:         mov     0x8(%ebp), %edx         # port addr
863:         mov     0xC(%ebp), %esi         # src
864:         mov     0x10(%ebp), %ecx        # count
865:         rep
866:         outsw
867:
868:         popl    %ecx
869:         popl    %esi
870:         popl    %edx
```

kernel/core386.S


```
871:         popl    %ebp
872:         ret
873:
874: .align 4
875: .globl load_gdt; load_gdt:
876:         movl    0x4(%esp), %eax
877:         lgdt    (%eax)
878:         movw    $KERNEL_DS, %ax
879:         movw    %ax, %ds
880:         movw    %ax, %es
881:         movw    %ax, %fs
882:         movw    %ax, %gs
883:         movw    %ax, %ss
884:         ljmp    $KERNEL_CS, $1f
885: 1:
886:         ret
887:
888: .align 4
889: .globl load_idt; load_idt:
890:         movl    0x4(%esp), %eax
891:         lidt    (%eax)
892:         ret
893:
894: .align 4
895: .globl activate_kpage_dir; activate_kpage_dir:
896:         movl    kpage_dir, %eax
897:         movl    %eax, %cr3
898:         ret
899:
900: .align 4
901: .globl load_tr; load_tr:
902:         mov     0x4(%esp), %ax
903:         ltr     %ax
904:         ret
905:
906: .align 4
907: .globl get_rdtsc; get_rdtsc:
908:         cpuid
909:         rdtsc
910:         ret
911:
912: .align 4
913: .globl invalidate_tlb; invalidate_tlb:
914:         movl    %cr3, %eax
915:         movl    %eax, %cr3
916:         ret
917:
918:
919: .data
920:
921: .globl  _cputype
922: .globl  _cpusignature
923: .globl  _cpuflags
924: .globl  _fpstatus
925: .globl  _brandid
926: .globl  _vendorid
927: .globl  _brandstr
928: .globl  _tlbinfo_eax
929: .globl  _tlbinfo_ebx
930: .globl  _tlbinfo_ecx
931: .globl  _tlbinfo_edx
932:
933: _cputype:       .int    0
934: _cpusignature:  .int    0
935: _cpuflags:      .int    0
936: _fpstatus:      .int    0
937: _brandid:       .int    0
```

kernel/core386.S


```
938: _vendorid:      .fill   13,1,0
939: _brandstr:      .fill   49,1,0
940: _tlbinfo_eax:   .int    0
941: _tlbinfo_ebx:   .int    0
942: _tlbinfo_ecx:   .int    0
943: _tlbinfo_edx:   .int    0
```

kernel/cpu.c


```
    1: /*
    2:  * fiwix/kernel/cpu.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/utsname.h>
   11: #include <fiwix/pic.h>
   12: #include <fiwix/pit.h>
   13: #include <fiwix/cpu.h>
   14: #include <fiwix/timer.h>
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/string.h>
   17:
   18: char UTS_MACHINE[_UTSNAME_LENGTH];
   19:
   20: static struct cpu_type intel[] = {
   21:         { 4,
   22:           { "i486 DX", "i486 DX", "i486 SX", "i486 DX/2",
   23:             "i486 SL", "i486 SX/2", NULL, "i486 DX/2 WBE",
   24:             "i486 DX/4", NULL, NULL, NULL, NULL, NULL, NULL, NULL }
   25:         },
   26:         { 5,
   27:           { NULL, "Pentium 60/66", "Pentium 75-200", "Pentium ODfor486",
   28:             "PentiumMMX", NULL, NULL, NULL, NULL, NULL, NULL, NULL,
   29:             NULL, NULL, NULL }
   30:         },
   31:         { 6,
   32:           { NULL, "Pentium Pro", NULL, "Pentium II", NULL, "Pentium II",
   33:             "Intel Celeron", "Pentium III", "Pentium III", NULL,
   34:             "Pentium III Xeon", "Pentium III", NULL, NULL, NULL, NULL }
   35:         }
   36: };
   37:
   38: static const char *cpu_flags[] = {
   39:         "FPU", "VME", "DE", "PSE", "TSC", "MSR", "PAE", "MCE", "CX8", "APIC",
   40:         "10", "SEP", "MTRR", "PGE", "MCA", "CMOV", "PAT", "PSE-36", "PSN",
   41:         "CLFSH", "20", "DS", "ACPI", "MMX", "FXSR", "SSE", "SSE2", "SS",
   42:         "HTT", "TM", "30", "PBE"
   43: };
   44:
   45: static unsigned long int detect_cpuspeed(void)
   46: {
   47:         unsigned long long int tsc1, tsc2;
   48:
   49:         outport_b(MODEREG, SEL_CHAN2 | LSB_MSB | TERM_COUNT | BINARY_CTR);
   50:         outport_b(CHANNEL2, (OSCIL / HZ) & 0xFF);
   51:         outport_b(CHANNEL2, (OSCIL / HZ) >> 8);
   52:         outport_b(PS2_SYSCTRL_B, inport_b(PS2_SYSCTRL_B) | ENABLE_SDATA | ENABLE
_TMR2G);
   53:
   54:         tsc1 = 0;
   55:         tsc1 = get_rdtsc();
   56:
   57:         while(!(inport_b(PS2_SYSCTRL_B) & 0x20));
   58:
   59:         tsc2 = 0;
   60:         tsc2 = get_rdtsc();
   61:
   62:         outport_b(PS2_SYSCTRL_B, inport_b(PS2_SYSCTRL_B) & ~(ENABLE_SDATA | ENAB
LE_TMR2G));
   63:
   64:         return (tsc2 - tsc1) * HZ;
   65: }
```

kernel/cpu.c


```
 66:
 67: /*
 68:  * These are the 2nd and 3rd level cache values according to Intel Processor
 69:  * Identification and the CPUID Instruction.
 70:  * Application Note 485. Document Number: 241618-031. September 2006.
 71:  */
 72: static void show_cache(int value)
 73: {
 74:         switch(value) {
 75:                 /* 2nd level cache */
 76:                 case 0x39:
 77:                 case 0x3B:
 78:                 case 0x41:
 79:                 case 0x79:
 80:                         cpu_table.cache = "128KB L2";
 81:                         break;
 82:                 case 0x3A:
 83:                         cpu_table.cache = "192KB L2";
 84:                         break;
 85:                 case 0x3C:
 86:                 case 0x42:
 87:                 case 0x7A:
 88:                 case 0x82:
 89:                         cpu_table.cache = "256KB L2";
 90:                         break;
 91:                 case 0x3D:
 92:                         cpu_table.cache = "384KB L2";
 93:                         break;
 94:                 case 0x3E:
 95:                 case 0x43:
 96:                 case 0x7B:
 97:                 case 0x7F:
 98:                 case 0x83:
 99:                 case 0x86:
100:                         cpu_table.cache = "512KB L2";
101:                         break;
102:                 case 0x44:
103:                 case 0x78:
104:                 case 0x7C:
105:                 case 0x84:
106:                 case 0x87:
107:                         cpu_table.cache = "1MB L2";
108:                         break;
109:                 case 0x45:
110:                 case 0x7D:
111:                 case 0x85:
112:                         cpu_table.cache = "2MB L2";
113:                         break;
114:
115:                 /* 3rd level cache */
116:                 case 0x22:
117:                         cpu_table.cache = "512KB L3";
118:                         break;
119:                 case 0x23:
120:                         cpu_table.cache = "1MB L3";
121:                         break;
122:                 case 0x25:
123:                         cpu_table.cache = "2MB L3";
124:                         break;
125:                 case 0x29:
126:                 case 0x46:
127:                         cpu_table.cache = "4MB L3";
128:                         break;
129:                 case 0x49:
130:                         cpu_table.cache = "4MB L3 & L2";
131:                         break;
132:                 case 0x4A:
```

kernel/cpu.c


```
  133:                          cpu_table.cache = "6MB L3";
  134:                          break;
  135:                  case 0x47:
  136:                  case 0x4B:
  137:                          cpu_table.cache = "8MB L3";
  138:                          break;
  139:                  case 0x4C:
  140:                          cpu_table.cache = "12MB L3";
  141:                          break;
  142:                  case 0x4D:
  143:                          cpu_table.cache = "16MB L3";
  144:                          break;
  145:                  default:
  146:                          break;
  147:          }
  148: }
  149:
  150: static void check_cache(int maxcpuid)
  151: {
  152:          int n, maxcpuids;
  153:
  154:          maxcpuids = 1;
  155:          if(maxcpuid >= 2) {
  156:                  for(n = 0; n < maxcpuids; n++) {
  157:                          tlbinfo();
  158:                          maxcpuids = _tlbinfo_eax & 0xFF;
  159:                          show_cache((_tlbinfo_eax >> 8) & 0xFF);
  160:                          show_cache((_tlbinfo_eax >> 16) & 0xFF);
  161:                          show_cache((_tlbinfo_eax >> 24) & 0xFF);
  162:                          if(!(_tlbinfo_ebx & RESERVED_DESC)) {
  163:                                  show_cache(_tlbinfo_ebx & 0xFF);
  164:                                  show_cache((_tlbinfo_ebx >> 8) & 0xFF);
  165:                                  show_cache((_tlbinfo_ebx >> 16) & 0xFF);
  166:                                  show_cache((_tlbinfo_ebx >> 24) & 0xFF);
  167:                          }
  168:                          if(!(_tlbinfo_ecx & RESERVED_DESC)) {
  169:                                  show_cache(_tlbinfo_ecx & 0xFF);
  170:                                  show_cache((_tlbinfo_ecx >> 8) & 0xFF);
  171:                                  show_cache((_tlbinfo_ecx >> 16) & 0xFF);
  172:                                  show_cache((_tlbinfo_ecx >> 24) & 0xFF);
  173:                          }
  174:                          if(!(_tlbinfo_edx & RESERVED_DESC)) {
  175:                                  show_cache(_tlbinfo_edx & 0xFF);
  176:                                  show_cache((_tlbinfo_edx >> 8) & 0xFF);
  177:                                  show_cache((_tlbinfo_edx >> 16) & 0xFF);
  178:                                  show_cache((_tlbinfo_edx >> 24) & 0xFF);
  179:                          }
  180:                  }
  181:          }
  182: }
  183:
  184: int get_cpu_flags(char *buffer, int offset)
  185: {
  186:          int n, size;
  187:          unsigned int mask;
  188:
  189:          size = sprintk(buffer + offset, "flags           :");
  190:          for(n = 0, mask = 1; n < 32; n++, mask <<= 1) {
  191:                  if(_cpuflags & mask) {
  192:                          size += sprintk(buffer + offset + size, " %s", cpu_flags
[n]);
  193:                  }
  194:          }
  195:          size += sprintk(buffer + offset + size, "\n");
  196:          return size;
  197: }
  198:
```

kernel/cpu.c


```
  199: void cpu_init(void)
  200: {
  201:         unsigned int n;
  202:         int maxcpuid;
  203:
  204:         memset_b(&cpu_table, NULL, sizeof(cpu_table));
  205:         cpu_table.model = -1;
  206:         cpu_table.stepping = -1;
  207:
  208:         printk("cpu       -                -\t");
  209:         cpu_table.family = cpuid();
  210:         if(!cpu_table.family) {
  211:                 cpu_table.has_cpuid = 1;
  212:                 maxcpuid = vendor_id();
  213:                 cpu_table.vendor_id = _vendorid;
  214:                 if(maxcpuid >= 1) {
  215:                         signature_flags();
  216:                         cpu_table.family = _cputype;
  217:                         cpu_table.flags = _cpuflags;
  218:                         sprintk(UTS_MACHINE, "i%c86", _cputype <= 6 ? ('0' + _cp
utype) : '6');
  219:                         strncpy(sys_utsname.machine, UTS_MACHINE, _UTSNAME_LENGT
H);
  220:                         if(!strcmp((char *)_vendorid, "GenuineIntel")) {
  221:                                 printk("Intel ");
  222:                                 for(n = 0; n < sizeof(intel) / sizeof(struct cpu
_type); n++) {
  223:                                         if(intel[n].cpu == _cputype) {
  224:                                                 cpu_table.model_name = !intel[n]
.name[(((int)_cpusignature >> 4) & 0xF)] ? NULL : intel[n].name[(((int)_cpusignature >>
 4) & 0xF)];
  225:                                                 break;
  226:                                         }
  227:                                 }
  228:                                 if(cpu_table.model_name) {
  229:                                         printk("%s", cpu_table.model_name);
  230:                                 } else {
  231:                                         printk("processor");
  232:                                 }
  233:                         } else if(!strcmp((char *)_vendorid, "AuthenticAMD")) {
  234:                                 printk("AMD processor");
  235:                         } else {
  236:                                 printk("x86");
  237:                         }
  238:                         if(_cpuflags & CPU_TSC) {
  239:                                 cpu_table.hz = detect_cpuspeed();
  240:                                 printk(" at %d.%d Mhz", (cpu_table.hz / 1000000)
, ((cpu_table.hz % 1000000) / 100000));
  241:                                 check_cache(maxcpuid);
  242:                                 if(cpu_table.cache) {
  243:                                         printk(" (%s)", cpu_table.cache);
  244:                                 }
  245:                         }
  246:                         printk("\n");
  247:                         printk("\t\t\t\tvendorid=%s ", _vendorid);
  248:                         cpu_table.model = (_cpusignature >> 4) & 0xF;
  249:                         cpu_table.stepping = _cpusignature & 0xF;
  250:                         printk("model=%d stepping=%d\n", cpu_table.model, cpu_ta
ble.stepping);
  251:                 }
  252:                 if(!brand_str()) {
  253:                         cpu_table.model_name = _brandstr;
  254:                         if(cpu_table.model_name[0]) {
  255:                                 printk("\t\t\t\t%s\n", cpu_table.model_name);
  256:                         }
  257:                 }
  258:         } else {
```

kernel/cpu.c


```
259:                 printk("80%d86\n", cpu_table.family);
260:                 cpu_table.has_cpuid = 0;
261:         }
262:         cpu_table.has_fpu = getfpu();
263: }
```

kernel/gdt.c


```
    1: /*
    2:  * fiwix/kernel/gdt.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/const.h>
   10: #include <fiwix/types.h>
   11: #include <fiwix/segments.h>
   12: #include <fiwix/process.h>
   13: #include <fiwix/limits.h>
   14: #include <fiwix/string.h>
   15:
   16: struct seg_desc gdt[NR_GDT_ENTRIES];
   17:
   18: struct desc_r gdtr = {
   19:         sizeof(gdt) - 1,
   20:         (unsigned int)&gdt
   21: };
   22:
   23: static void gdt_set_entry(int num, unsigned int base_addr, unsigned int limit, c
har loflags, char hiflags)
   24: {
   25:         num /= sizeof(struct seg_desc);
   26:         gdt[num].sd_lolimit = limit & 0xFFFF;
   27:         gdt[num].sd_lobase = base_addr & 0xFFFFFF;
   28:         gdt[num].sd_loflags = loflags;
   29:         gdt[num].sd_hilimit = (limit >> 16) & 0x0F;
   30:         gdt[num].sd_hiflags = hiflags;
   31:         gdt[num].sd_hibase = (base_addr >> 24) & 0xFF;
   32: }
   33:
   34: void gdt_init(void)
   35: {
   36:         unsigned char loflags;
   37:
   38:         gdt_set_entry(0, 0, 0, 0, 0);   /* null descriptor */
   39:
   40:         loflags = SD_CODE | SD_CD | SD_DPL0 | SD_PRESENT;
   41:         gdt_set_entry(KERNEL_CS, 0, 0xFFFFFFFF, loflags, SD_OPSIZE32 | SD_PAGE4K
B);
   42:         loflags = SD_DATA | SD_CD | SD_DPL0 | SD_PRESENT;
   43:         gdt_set_entry(KERNEL_DS, 0, 0xFFFFFFFF, loflags, SD_OPSIZE32 | SD_PAGE4K
B);
   44:
   45:         loflags = SD_CODE | SD_CD | SD_DPL3 | SD_PRESENT;
   46:         gdt_set_entry(USER_CS, 0, 0xFFFFFFFF, loflags, SD_OPSIZE32 | SD_PAGE4KB)
;
   47:         loflags = SD_DATA | SD_CD | SD_DPL3 | SD_PRESENT;
   48:         gdt_set_entry(USER_DS, 0, 0xFFFFFFFF, loflags, SD_OPSIZE32 | SD_PAGE4KB)
;
   49:
   50:         loflags = SD_TSSPRESENT;
   51:         gdt_set_entry(TSS, 0, sizeof(struct proc) - 1, loflags, SD_OPSIZE32);
   52:
   53:         load_gdt((unsigned int)&gdtr);
   54: }
```

kernel/idt.c


```
    1: /*
    2:  * fiwix/kernel/idt.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/const.h>
   10: #include <fiwix/types.h>
   11: #include <fiwix/segments.h>
   12: #include <fiwix/string.h>
   13:
   14: struct gate_desc idt[NR_IDT_ENTRIES];
   15:
   16: struct desc_r idtr = {
   17:         sizeof(idt) - 1,
   18:         (unsigned int)idt
   19: };
   20:
   21: static void set_idt_entry(int num, __off_t handler, unsigned int flags)
   22: {
   23:         idt[num].gd_looffset = handler & 0x0000FFFF;
   24:         idt[num].gd_selector = KERNEL_CS;
   25:         idt[num].gd_flags = flags << 8;
   26:         idt[num].gd_hioffset = handler >> 16;
   27: }
   28:
   29: void idt_init(void)
   30: {
   31:         int n;
   32:
   33:         memset_b(idt, NULL, sizeof(idt));
   34:         for(n = 0; n < NR_IDT_ENTRIES; n++) {
   35:                 set_idt_entry(n, (__off_t)&unknown_irq, SD_32INTRGATE | SD_PRESE
NT);
   36:         }
   37:
   38:         /* FIXME: must be SD_32TRAPGATE for true multitasking */
   39:         set_idt_entry(0x00, (__off_t)&except0, SD_32INTRGATE | SD_PRESENT);
   40:         set_idt_entry(0x01, (__off_t)&except1, SD_32INTRGATE | SD_PRESENT);
   41:         set_idt_entry(0x02, (__off_t)&except2, SD_32INTRGATE | SD_PRESENT);
   42:         set_idt_entry(0x03, (__off_t)&except3, SD_32INTRGATE | SD_PRESENT);
   43:         set_idt_entry(0x04, (__off_t)&except4, SD_32INTRGATE | SD_PRESENT);
   44:         set_idt_entry(0x05, (__off_t)&except5, SD_32INTRGATE | SD_PRESENT);
   45:         set_idt_entry(0x06, (__off_t)&except6, SD_32INTRGATE | SD_PRESENT);
   46:         set_idt_entry(0x07, (__off_t)&except7, SD_32INTRGATE | SD_PRESENT);
   47:         set_idt_entry(0x08, (__off_t)&except8, SD_32INTRGATE | SD_PRESENT);
   48:         set_idt_entry(0x09, (__off_t)&except9, SD_32INTRGATE | SD_PRESENT);
   49:         set_idt_entry(0x0A, (__off_t)&exceptA, SD_32INTRGATE | SD_PRESENT);
   50:         set_idt_entry(0x0B, (__off_t)&exceptB, SD_32INTRGATE | SD_PRESENT);
   51:         set_idt_entry(0x0C, (__off_t)&exceptC, SD_32INTRGATE | SD_PRESENT);
   52:         set_idt_entry(0x0D, (__off_t)&exceptD, SD_32INTRGATE | SD_PRESENT);
   53:         set_idt_entry(0x0E, (__off_t)&exceptE, SD_32INTRGATE | SD_PRESENT);
   54:         set_idt_entry(0x0F, (__off_t)&exceptF, SD_32INTRGATE | SD_PRESENT);
   55:         set_idt_entry(0x10, (__off_t)&except10, SD_32INTRGATE | SD_PRESENT);
   56:         set_idt_entry(0x11, (__off_t)&except11, SD_32INTRGATE | SD_PRESENT);
   57:         set_idt_entry(0x12, (__off_t)&except12, SD_32INTRGATE | SD_PRESENT);
   58:         set_idt_entry(0x13, (__off_t)&except13, SD_32INTRGATE | SD_PRESENT);
   59:         set_idt_entry(0x14, (__off_t)&except14, SD_32INTRGATE | SD_PRESENT);
   60:         set_idt_entry(0x15, (__off_t)&except15, SD_32INTRGATE | SD_PRESENT);
   61:         set_idt_entry(0x16, (__off_t)&except16, SD_32INTRGATE | SD_PRESENT);
   62:         set_idt_entry(0x17, (__off_t)&except17, SD_32INTRGATE | SD_PRESENT);
   63:         set_idt_entry(0x18, (__off_t)&except18, SD_32INTRGATE | SD_PRESENT);
   64:         set_idt_entry(0x19, (__off_t)&except19, SD_32INTRGATE | SD_PRESENT);
   65:         set_idt_entry(0x1A, (__off_t)&except1A, SD_32INTRGATE | SD_PRESENT);
   66:         set_idt_entry(0x1B, (__off_t)&except1B, SD_32INTRGATE | SD_PRESENT);
```

kernel/idt.c


```
   67:         set_idt_entry(0x1C, (__off_t)&except1C, SD_32INTRGATE | SD_PRESENT);
   68:         set_idt_entry(0x1D, (__off_t)&except1D, SD_32INTRGATE | SD_PRESENT);
   69:         set_idt_entry(0x1E, (__off_t)&except1E, SD_32INTRGATE | SD_PRESENT);
   70:         set_idt_entry(0x1F, (__off_t)&except1F, SD_32INTRGATE | SD_PRESENT);
   71:
   72:         set_idt_entry(0x20, (__off_t)&irq0, SD_32INTRGATE | SD_PRESENT);
   73:         set_idt_entry(0x21, (__off_t)&irq1, SD_32INTRGATE | SD_PRESENT);
   74:         set_idt_entry(0x22, (__off_t)&irq2, SD_32INTRGATE | SD_PRESENT);
   75:         set_idt_entry(0x23, (__off_t)&irq3, SD_32INTRGATE | SD_PRESENT);
   76:         set_idt_entry(0x24, (__off_t)&irq4, SD_32INTRGATE | SD_PRESENT);
   77:         set_idt_entry(0x25, (__off_t)&irq5, SD_32INTRGATE | SD_PRESENT);
   78:         set_idt_entry(0x26, (__off_t)&irq6, SD_32INTRGATE | SD_PRESENT);
   79:         set_idt_entry(0x27, (__off_t)&irq7, SD_32INTRGATE | SD_PRESENT);
   80:         set_idt_entry(0x28, (__off_t)&irq8, SD_32INTRGATE | SD_PRESENT);
   81:         set_idt_entry(0x29, (__off_t)&irq9, SD_32INTRGATE | SD_PRESENT);
   82:         set_idt_entry(0x2A, (__off_t)&irq10, SD_32INTRGATE | SD_PRESENT);
   83:         set_idt_entry(0x2B, (__off_t)&irq11, SD_32INTRGATE | SD_PRESENT);
   84:         set_idt_entry(0x2C, (__off_t)&irq12, SD_32INTRGATE | SD_PRESENT);
   85:         set_idt_entry(0x2D, (__off_t)&irq13, SD_32INTRGATE | SD_PRESENT);
   86:         set_idt_entry(0x2E, (__off_t)&irq14, SD_32INTRGATE | SD_PRESENT);
   87:         set_idt_entry(0x2F, (__off_t)&irq15, SD_32INTRGATE | SD_PRESENT);
   88:
   89:         /* FIXME: must be SD_32TRAPGATE for true multitasking */
   90:         set_idt_entry(0x80, (__off_t)&syscall, SD_32INTRGATE | SD_DPL3 | SD_PRES
ENT);
   91:
   92:         load_idt((unsigned int)&idtr);
   93: }
```

kernel/init.c


```
    1: /*
    2:  * fiwix/kernel/init.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/system.h>
   11: #include <fiwix/mm.h>
   12: #include <fiwix/timer.h>
   13: #include <fiwix/sched.h>
   14: #include <fiwix/fcntl.h>
   15: #include <fiwix/stat.h>
   16: #include <fiwix/process.h>
   17: #include <fiwix/syscalls.h>
   18: #include <fiwix/unistd.h>
   19: #include <fiwix/stdio.h>
   20: #include <fiwix/string.h>
   21:
   22: #define INIT_TRAMPOLINE_SIZE    128     /* max. size of init_trampoline() */
   23:
   24: char *init_argv[] = { INIT_PROGRAM, NULL, NULL };
   25: char *init_envp[] = { "HOME=/", "TERM=linux", NULL };
   26:
   27: static void init_trampoline(void)
   28: {
   29:         USER_SYSCALL(SYS_open, "/dev/console", O_RDWR, 0);      /* stdin */
   30:         USER_SYSCALL(SYS_dup, 0, NULL, NULL);                   /* stdout */
   31:         USER_SYSCALL(SYS_dup, 0, NULL, NULL);                   /* stderr */
   32:         USER_SYSCALL(SYS_execve, INIT_PROGRAM, init_argv, init_envp);
   33:
   34:         /* only reached in case of error in sys_execve() */
   35:         USER_SYSCALL(SYS_exit, NULL, NULL, NULL);
   36: }
   37:
   38: void init_init(void)
   39: {
   40:         int n;
   41:         unsigned int page;
   42:         struct inode *i;
   43:         unsigned int *pgdir;
   44:         struct proc *init;
   45:
   46:         if(namei(INIT_PROGRAM, &i, NULL, FOLLOW_LINKS)) {
   47:                 PANIC("can't find %s.\n", INIT_PROGRAM);
   48:         }
   49:         if(!S_ISREG(i->i_mode)) {
   50:                 PANIC("%s is not a regular file.\n", INIT_PROGRAM);
   51:         }
   52:         iput(i);
   53:
   54:         /* INIT slot was already created in main.c */
   55:         init = &proc_table[INIT];
   56:
   57:         /* INIT process starts with the current (kernel) Page Directory */
   58:         if(!(pgdir = (void *)kmalloc())) {
   59:                 goto init_init__die;
   60:         }
   61:         init->rss++;
   62:         memcpy_b(pgdir, kpage_dir, PAGE_SIZE);
   63:         init->tss.cr3 = V2P((unsigned int)pgdir);
   64:
   65:         memset_b(init->vma, NULL, sizeof(init->vma));
   66:         init->ppid = 0;
   67:         init->pgid = 0;
```

kernel/init.c


```
   68:         init->sid = 0;
   69:         init->flags = 0;
   70:         init->children = 0;
   71:         init->priority = DEF_PRIORITY;
   72:         init->start_time = CURRENT_TICKS;
   73:         init->sleep_address = NULL;
   74:         init->uid = init->gid = 0;
   75:         init->euid = init->egid = 0;
   76:         init->suid = init->sgid = 0;
   77:         memset_b(init->fd, NULL, sizeof(init->fd));
   78:         memset_b(init->fd_flags, NULL, sizeof(init->fd_flags));
   79:         init->root = current->root;
   80:         init->pwd = current->pwd;
   81:         strcpy(init->argv0, init_argv[0]);
   82:         init_argv[1] = init_args;
   83:         sprintk(init->pidstr, "%d", init->pid);
   84:         init->sigpending = 0;
   85:         init->sigblocked = 0;
   86:         init->sigexecuting = 0;
   87:         memset_b(init->sigaction, NULL, sizeof(init->sigaction));
   88:         memset_b(&init->usage, NULL, sizeof(struct rusage));
   89:         memset_b(&init->cusage, NULL, sizeof(struct rusage));
   90:         init->timeout = 0;
   91:         for(n = 0; n < RLIM_NLIMITS; n++) {
   92:                 init->rlim[n].rlim_cur = init->rlim[n].rlim_max = RLIM_INFINITY;
   93:         }
   94:         init->rlim[RLIMIT_NOFILE].rlim_cur = OPEN_MAX;
   95:         init->rlim[RLIMIT_NOFILE].rlim_max = NR_OPENS;
   96:         init->rlim[RLIMIT_NPROC].rlim_cur = CHILD_MAX;
   97:         init->rlim[RLIMIT_NPROC].rlim_cur = NR_PROCS;
   98:         init->umask = 0022;
   99:
  100:         /* setup the stack */
  101:         if(!(init->tss.esp0 = kmalloc())) {
  102:                 goto init_init__die;
  103:         }
  104:         init->tss.esp0 += PAGE_SIZE - 4;
  105:         init->rss++;
  106:         init->tss.ss0 = KERNEL_DS;
  107:
  108:         /* setup the init_trampoline */
  109:         page = map_page(init, KERNEL_BASE_ADDR - PAGE_SIZE, 0, PROT_READ | PROT_
WRITE);
  110:         memcpy_b((void *)page, init_trampoline, INIT_TRAMPOLINE_SIZE);
  111:
  112:         init->tss.eip = (unsigned int)switch_to_user_mode;
  113:         init->tss.esp = page + PAGE_SIZE - 4;
  114:
  115:         init->state = PROC_RUNNING;
  116:         nr_processes++;
  117:         return;
  118:
  119: init_init__die:
  120:         PANIC("unable to run init process.\n");
  121: }
```

kernel/main.c


```
 1: /*
 2:  * fiwix/kernel/main.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/asm.h>
 9: #include <fiwix/kernel.h>
10: #include <fiwix/mm.h>
11: #include <fiwix/system.h>
12: #include <fiwix/timer.h>
13: #include <fiwix/sched.h>
14: #include <fiwix/sleep.h>
15: #include <fiwix/cpu.h>
16: #include <fiwix/pic.h>
17: #include <fiwix/fs.h>
18: #include <fiwix/devices.h>
19: #include <fiwix/console.h>
20: #include <fiwix/keyboard.h>
21: #include <fiwix/ramdisk.h>
22: #include <fiwix/version.h>
23: #include <fiwix/limits.h>
24: #include <fiwix/segments.h>
25: #include <fiwix/syscalls.h>
26: #include <fiwix/stdio.h>
27: #include <fiwix/string.h>
28: #include <fiwix/kparms.h>
29: #include <fiwix/i386elf.h>
30: #include <fiwix/bios.h>
31:
32: /*
33:  * check if the bit BIT in Multiboot FLAGS is set
34:  * ----------------------------------------------
35:  * bit 11 -> vbe_*
36:  * bit 10 -> apm_table
37:  * bit  9 -> boot_loader_name
38:  * bit  8 -> config_table
39:  * bit  7 -> drives_length and drives_addr
40:  * bit  6 -> mmap_length and mmap_addr
41:  * bit  5 -> ELF symbols
42:  * bit  4 -> a.out symbols
43:  * bit  3 -> mods_count and mods_addr
44:  * bit  2 -> cmdline
45:  * bit  1 -> boot_device
46:  * bit  0 -> mem_lower and mem_upper values
47:  */
48: #define CHECK_FLAG(flags,bit)   ((flags) & (1 << (bit)))
49:
50: Elf32_Shdr *symtab, *strtab;
51: unsigned int _last_data_addr;
52: int _memsize;
53: int _extmemsize;
54: int _rootdev;
55: int _noramdisk;
56: int _ramdisksize;
57: char _rootfstype[10];
58: char _rootdevname[DEVNAME_MAX + 1];
59: char _initrd[DEVNAME_MAX + 1];
60: int _syscondev;
61: char *init_args;
62:
63: char cmdline[NAME_MAX + 1];
64:
65: struct new_utsname sys_utsname = {
66:         UTS_SYSNAME,
67:         UTS_NODENAME,
```

kernel/main.c


```
   68:         UTS_RELEASE,
   69:         UTS_VERSION,
   70:         "",
   71:         UTS_DOMAINNAME,
   72: };
   73:
   74: struct kernel_stat kstat;
   75:
   76: /*
   77:  * This function returns the last address used by kernel symbols or the value
   78:  * of 'mod_end' (in the module structure) of the last module loaded by GRUB.
   79:  *
   80:  * This is intended to setup the kernel stack beyond all these addresses.
   81:  */
   82: unsigned int get_last_boot_addr(unsigned int info)
   83: {
   84:         multiboot_info_t *mbi;
   85:         Elf32_Shdr *shdr;
   86:         elf_section_header_table_t *hdr;
   87:         struct module *mod;
   88:         unsigned short int n;
   89:         unsigned int addr;
   90:
   91:         symtab = strtab = NULL;
   92:         mbi = (multiboot_info_t *)info;
   93:         hdr = &(mbi->u.elf_sec);
   94:         for(n = 0; n < hdr->num; n++) {
   95:                 shdr = (Elf32_Shdr *)(hdr->addr + (n * hdr->size));
   96:                 if(shdr->sh_type == SHT_SYMTAB) {
   97:                         symtab = shdr;
   98:                 }
   99:                 if(shdr->sh_type == SHT_STRTAB) {
  100:                         strtab = shdr;
  101:                 }
  102:         }
  103:
  104:         addr = strtab->sh_addr + strtab->sh_size;
  105:
  106:         /*
  107:          * https://www.gnu.org/software/grub/manual/multiboot/multiboot.html
  108:          *
  109:          * Check if GRUB has loaded some modules and, if so, get the last
  110:          * address used by the last one.
  111:          */
  112:         if(CHECK_FLAG(mbi->flags, 3)) {
  113:                 mod = (struct module *)mbi->mods_addr;
  114:                 for(n = 0; n < mbi->mods_count; n++, mod++) {
  115:                         addr = mod->mod_end;
  116:                 }
  117:         }
  118:
  119:         return P2V(addr);
  120: }
  121:
  122: /* check the validity of a command line parameter */
  123: static int check_parm(struct kparms *parm, const char *value)
  124: {
  125:         int n;
  126:
  127:         if(!strcmp(parm->name, "root=")) {
  128:                 for(n = 0; parm->value[n]; n++) {
  129:                         if(!strcmp(parm->value[n], value)) {
  130:                                 _rootdev = parm->sysval[n];
  131:                                 strncpy(_rootdevname, value, DEVNAME_MAX);
  132:                                 return 0;
  133:                         }
  134:                 }
```

kernel/main.c


```
  135:                  return 1;
  136:          }
  137:          if(!strcmp(parm->name, "noramdisk")) {
  138:                  _noramdisk = 1;
  139:                  return 0;
  140:          }
  141:          if(!strcmp(parm->name, "ramdisksize=")) {
  142:                  int size = atoi(value);
  143:                  if(!size || size > RAMDISK_MAXSIZE) {
  144:                          printk("WARNING: 'ramdisksize' value is out of limits, d
efaulting to 4096KB.\n");
  145:                          _ramdisksize = 0;
  146:                  } else {
  147:                          _ramdisksize = size;
  148:                  }
  149:                  return 0;
  150:          }
  151:          if(!strcmp(parm->name, "initrd=")) {
  152:                  if(value[0]) {
  153:                          strncpy(_initrd, value, DEVNAME_MAX);
  154:                          return 0;
  155:                  }
  156:          }
  157:          if(!strcmp(parm->name, "rootfstype=")) {
  158:                  for(n = 0; parm->value[n]; n++) {
  159:                          if(!strcmp(parm->value[n], value)) {
  160:                                  strncpy(_rootfstype, value, sizeof(_rootfstype))
;
  161:                                  return 0;
  162:                          }
  163:                  }
  164:                  return 1;
  165:          }
  166:          if(!strcmp(parm->name, "console=")) {
  167:                  for(n = 0; parm->value[n]; n++) {
  168:                          if(!strcmp(parm->value[n], value)) {
  169:                                  _syscondev = parm->sysval[n];
  170:                                  return 0;
  171:                          }
  172:                  }
  173:                  return 1;
  174:          }
  175:          printk("WARNING: the parameter '%s' looks valid but it's not defined!\n"
, parm->name);
  176:          return 0;
  177: }
  178:
  179: static int parse_arg(const char *arg)
  180: {
  181:          int n;
  182:
  183:          /* '--' marks the beginning of the init arguments */
  184:          if(!strcmp(arg, "--")) {
  185:                  return 1;
  186:          }
  187:
  188:          for(n = 0; parm_table[n].name; n++) {
  189:                  if(!strncmp(arg, parm_table[n].name, strlen(parm_table[n].name))
) {
  190:                          arg += strlen(parm_table[n].name);
  191:                          if(check_parm(&parm_table[n], arg)) {
  192:                                  printk("WARNING: invalid value '%s' in the '%s'
parameter.\n", arg, parm_table[n].name);
  193:                          }
  194:                          return 0;
  195:                  }
  196:          }
```

kernel/main.c


```
  197:         printk("WARNING: invalid cmdline parameter: '%s'.\n", arg);
  198:         return 0;
  199: }
  200:
  201: static char * parse_cmdline(const char *str)
  202: {
  203:         char *from, *to;
  204:         char arg[CMDL_ARG_LEN];
  205:         char c;
  206:
  207:         from = to = (char *)str;
  208:         for(;;) {
  209:                 c = *(str++);
  210:                 if(c == ' ' || !c) {
  211:                         if(to - from < CMDL_ARG_LEN) {
  212:                                 memcpy_b(arg, from, to - from);
  213:                                 arg[to - from] = NULL;
  214:                                 if(arg[0] != NULL) {
  215:                                         if(parse_arg(arg)) {
  216:                                                 while(*(from++)) {
  217:                                                         if(*from != '-' && *from
 != ' ') {
  218:                                                                 break;
  219:                                                         }
  220:                                                 }
  221:                                                 return from;
  222:                                         }
  223:                                 }
  224:                         } else {
  225:                                 memcpy_b(arg, from, CMDL_ARG_LEN);
  226:                                 arg[CMDL_ARG_LEN - 1] = NULL;
  227:                                 printk("WARNING: invalid length of the cmdline p
arameter '%s'.\n", arg);
  228:                         }
  229:                         from = ++to;
  230:                         if(!c) {
  231:                                 break;
  232:                         }
  233:                         continue;
  234:                 }
  235:                 to++;
  236:         }
  237:
  238:         return NULL;
  239: }
  240:
  241: void start_kernel(unsigned long magic, unsigned long info, unsigned int stack)
  242: {
  243:         struct proc *init, *p_kswapd;
  244:         multiboot_info_t mbi;
  245:
  246:         /* default kernel values */
  247:         strcpy(_rootfstype, "ext2");            /* filesystem is ext2 */
  248:         _syscondev = MKDEV(VCONSOLES_MAJOR, 0); /* console is /dev/tty0 */
  249:
  250:         pic_init();
  251:         idt_init();
  252:         dev_init();
  253:         tty_init();
  254:
  255:         printk("                                Welcome to %s\n", UTS_SYSNAME);
  256:         printk("                    Copyright (c) 2018-2020, Jordi Sanfeliu\n")
;
  257:         printk("\n");
  258:         printk("                     kernel v%s for i386 architecture\n", UTS_R
ELEASE);
  259:         printk("                (GCC %s, built on %s)\n", __VERSION__, UTS_VERSIO
```

kernel/main.c


```
N);
  260:          printk("\n");
  261:          printk("DEVICE    ADDRESS         IRQ   COMMENT\n");
  262:          printk("-------------------------------------------------------------
--------------\n");
  263:
  264:          if(magic != MULTIBOOT_BOOTLOADER_MAGIC) {
  265:                  printk("WARNING: invalid multiboot-bootloader magic number: 0x%x
.\n\n", (unsigned long int)magic);
  266:                  memset_b(&mbi, NULL, sizeof(struct multiboot_info));
  267:          } else {
  268:                  memcpy_b(&mbi, (void *)info, sizeof(struct multiboot_info));
  269:          }
  270:
  271:          memset_b(&kstat, NULL, sizeof(kstat));
  272:
  273:          cpu_init();
  274:
  275:          /* check if a command line was supplied */
  276:          if(CHECK_FLAG(mbi.flags, 2)) {
  277:                  int n, len;
  278:                  char c;
  279:                  char *p;
  280:
  281:                  p = (char *)mbi.cmdline;
  282:                  len = strlen(p);
  283:                  /* suppress 'fiwix' */
  284:                  for(n = 0; n < len; n++) {
  285:                          c = *(p++);
  286:                          if(c == ' ') {
  287:                                  break;
  288:                          }
  289:                  }
  290:                  strcpy(cmdline, p);
  291:                  init_args = parse_cmdline(cmdline);
  292:          } else {
  293:                  printk("WARNING: no cmdline detected!\n");
  294:          }
  295:
  296:          printk("kernel    0x%08X       -    cmdline='%s'\n", KERNEL_ENTRY_ADDR,
cmdline);
  297:
  298:          timer_init();
  299:          vconsole_init();
  300:          keyboard_init();
  301:
  302:          if(CHECK_FLAG(mbi.flags, 3)) {
  303:                  int n;
  304:                  struct module *mod;
  305:
  306:                  mod = (struct module *)mbi.mods_addr;
  307:                  for(n = 0; n < mbi.mods_count; n++, mod++) {
  308:                          if(!strcmp((char *)mod->string, _initrd)) {
  309:                                  printk("initrd    0x%08X-0x%08X file='%s' size=%
dKB\n", mod->mod_start, mod->mod_end, mod->string, (mod->mod_end - mod->mod_start) / 10
24);
  310:                                  ramdisk_table[0].addr = (char *)mod->mod_start;
  311:                          }
  312:                  }
  313:          }
  314:
  315:          if(!CHECK_FLAG(mbi.flags, 0)) {
  316:                  printk("WARNING: values in mem_lower and mem_upper are not valid
!\n");
  317:          }
  318:          _memsize = (unsigned int)mbi.mem_lower;
  319:          _extmemsize = (unsigned int)mbi.mem_upper;
```

kernel/main.c


```
320:
321:         if(CHECK_FLAG(mbi.flags, 6)) {
322:                 bios_map_init((memory_map_t *)mbi.mmap_addr, mbi.mmap_length);
323:         } else {
324:                 bios_map_init(NULL, 0);
325:         }
326:
327:         _last_data_addr = stack - KERNEL_BASE_ADDR;
328:         mem_init();
329:         proc_init();
330:
331:         if(!(CHECK_FLAG(mbi.flags, 5))) {
332:                 printk("WARNING: ELF section header table is not valid!\n");
333:         }
334:
335:         /* IDLE is now the current process */
336:         set_tss(current);
337:         load_tr(TSS);
338:         current->tss.cr3 = (unsigned int)kpage_dir;
339:         current->flags |= PF_KPROC;
340:
341:         /* reserve the slot 1 for the INIT process */
342:         init = get_proc_free();
343:         proc_slot_init(init);
344:         init->pid = get_unused_pid();
345:
346:         /* create and setup kswapd process */
347:         p_kswapd = kernel_process(kswapd);
348:
349:         /* kswapd will take over the rest of the kernel initialization */
350:         p_kswapd->state = PROC_RUNNING;
351:         need_resched = 1;
352:
353:         STI();          /* let's rock! */
354:         cpu_idle();
355: }
356:
357: void stop_kernel(void)
358: {
359:         struct proc *p;
360:
361:         printk("\n");
362:         printk("**    Safe to Power Off    **\n");
363:         printk("            -or-\n");
364:         printk("** Press Any Key to Reboot **\n");
365:         any_key_to_reboot = 1;
366:
367:         /* put all processes to sleep and reset all pending signals */
368:         FOR_EACH_PROCESS(p) {
369:                 p->state = PROC_SLEEPING;
370:                 p->sigpending = 0;
371:         }
372:
373:         /* TODO: disable all interrupts */
374:         CLI();
375:         disable_irq(TIMER_IRQ);
376:
377:         /* switch to IDLE process */
378:         if(current) {
379:                 do_sched();
380:         }
381:
382:         STI();
383:         enable_irq(KEYBOARD_IRQ);
384:         cpu_idle();
385: }
```

kernel/Makefile


```
  1: # fiwix/kernel/Makefile
  2: #
  3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
  4: # Distributed under the terms of the Fiwix License.
  5: #
  6:
  7: .S.o:
  8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
  9: .c.o:
 10:         $(CC) $(CFLAGS) -c -o $@ $<
 11:
 12: OBJS = boot.o core386.o main.o init.o gdt.o idt.o syscalls.o pic.o pit.o \
 13:        traps.o cpu.o cmos.o timer.o sched.o sleep.o signal.o process.o
 14:
 15: kernel: $(OBJS)
 16:         $(LD) $(LDFLAGS) -r $(OBJS) -o kernel.o
 17:
 18: clean:
 19:         rm -f *.o
 20:
```

kernel/pic.c


```
    1: /*
    2:  * fiwix/kernel/pic.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/config.h>
   11: #include <fiwix/limits.h>
   12: #include <fiwix/errno.h>
   13: #include <fiwix/pic.h>
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/string.h>
   16: #include <fiwix/sigcontext.h>
   17:
   18: /* interrupt vector base addresses */
   19: #define IRQ0_ADDR       0x20
   20: #define IRQ8_ADDR       0x28
   21:
   22: /*
   23:  * pic.c implements a bottom half table using a singly linked list.
   24:  *
   25:  *  head                            tail
   26:  * +---------+  ----------+  ...  ----------+
   27:  * |data|next|  |data|next|  ...  |data|next|
   28:  * |    |  -->  |    |  -->  ...  |    |  / |
   29:  * +---------+  ----------+  ...  ----------+
   30:  *  (bh)         (bh)              (bh)
   31:  */
   32:
   33: struct bh bh_pool[NR_BH];
   34: struct bh *bh_pool_head;
   35: struct bh *bh_head;
   36: struct bh *bh_tail;
   37:
   38: static struct bh *get_free_bh(void)
   39: {
   40:         struct bh *new;
   41:
   42:         new = NULL;
   43:         if(bh_pool_head) {
   44:                 new = bh_pool_head;
   45:                 bh_pool_head = bh_pool_head->next;
   46:                 new->next = NULL;
   47:         }
   48:         return new;
   49: }
   50:
   51: static void put_free_bh(struct bh *old)
   52: {
   53:         old->next = bh_pool_head;
   54:         bh_pool_head = old;
   55: }
   56:
   57: /*
   58:  * This sends the command OCW3 to PIC (master or slave) to obtain the register
   59:  * values. Slave is chained and represents IRQs 8-15. Master represents IRQs
   60:  * 0-7, with IRQ 2 being the chain.
   61:  */
   62: static unsigned short int pic_get_irq_reg(int ocw3)
   63: {
   64:         outport_b(PIC_MASTER, ocw3);
   65:         outport_b(PIC_SLAVE, ocw3);
   66:         return (inport_b(PIC_SLAVE) << 8) | inport_b(PIC_MASTER);
   67: }
```

kernel/pic.c


```
 68:
 69: void add_bh(void (*fn)(void))
 70: {
 71:         unsigned long int flags;
 72:         struct bh *b;
 73:
 74:         SAVE_FLAGS(flags); CLI();
 75:
 76:         if(!(b = get_free_bh())) {
 77:                 RESTORE_FLAGS(flags);
 78:                 PANIC("no more bottom half slots!\n");
 79:         }
 80:
 81:         /* initialize bh */
 82:         memset_b(b, NULL, sizeof(struct bh));
 83:         b->fn = fn;
 84:
 85:         if(!bh_tail) {
 86:                 bh_head = bh_tail = b;
 87:         } else {
 88:                 bh_tail->next = b;
 89:                 bh_tail = b;
 90:         }
 91:
 92:         RESTORE_FLAGS(flags);
 93:         return;
 94: }
 95:
 96: void del_bh(void)
 97: {
 98:         unsigned long int flags;
 99:         struct bh *b;
100:
101:         if(!bh_head) {
102:                 return;
103:         }
104:
105:         SAVE_FLAGS(flags); CLI();
106:
107:         b = bh_head;
108:         if(bh_head == bh_tail) {
109:                 bh_head = bh_tail = NULL;
110:         } else {
111:                 bh_head = bh_head->next;
112:         }
113:         put_free_bh(b);
114:
115:         RESTORE_FLAGS(flags);
116:         return;
117: }
118:
119: void enable_irq(int irq)
120: {
121:         int addr;
122:
123:         addr = (irq > 7) ? PIC_SLAVE + DATA : PIC_MASTER + DATA;
124:         irq &= 0x0007;
125:
126:         outport_b(addr, inport_b(addr) & ~(1 << irq));
127: }
128:
129: void disable_irq(int irq)
130: {
131:         int addr;
132:
133:         addr = (irq > 7) ? PIC_SLAVE + DATA : PIC_MASTER + DATA;
134:         irq &= 0x0007;
```

kernel/pic.c


```
  135:
  136:         outport_b(addr, inport_b(addr) | (1 << irq));
  137: }
  138:
  139: int register_irq(int irq, char *name, void *addr)
  140: {
  141:         if(irq < 0  || irq >= NR_IRQS) {
  142:                 return -EINVAL;
  143:         }
  144:
  145:         if(irq_table[irq].registered) {
  146:                 printk("WARNING: %s(): interrupt %d already registered!\n", __FU
NCTION__, irq);
  147:                 return -EINVAL;
  148:         }
  149:         irq_table[irq].ticks = 0;
  150:         irq_table[irq].name = name;
  151:         irq_table[irq].registered = 1;
  152:         irq_table[irq].handler = addr;
  153:         return 0;
  154: }
  155:
  156: int unregister_irq(int irq)
  157: {
  158:         if(irq < 0  || irq >= NR_IRQS) {
  159:                 return -EINVAL;
  160:         }
  161:
  162:         if(!irq_table[irq].registered) {
  163:                 printk("WARNING: %s(): trying to unregister an unregistered inte
rrupt %d.\n", __FUNCTION__, irq);
  164:                 return -EINVAL;
  165:         }
  166:         memset_b(&irq_table[irq], NULL, sizeof(struct interrupts));
  167:         return 0;
  168: }
  169:
  170: /* each ISR points to this function */
  171: void irq_handler(int irq, struct sigcontext sc)
  172: {
  173:         int real;
  174:
  175:         /* this should help to detect hardware problems */
  176:         if(irq == -1) {
  177:                 printk("Unknown IRQ received!\n");
  178:                 return;
  179:         }
  180:
  181:         /* spurious interrupt treatment */
  182:         if(!irq_table[irq].handler) {
  183:                 real = pic_get_irq_reg(PIC_READ_ISR);
  184:                 if(!real) {
  185:                         /*
  186:                          * If IRQ was not real and came from slave, then send
  187:                          * an EOI to master because it doesn't know if the IRQ
  188:                          * was a spurious interrupt from slave.
  189:                          */
  190:                         if(irq > 7) {
  191:                                 outport_b(PIC_MASTER, EOI);
  192:                         }
  193:                         if(kstat.sirqs < MAX_SPU_NOTICES) {
  194:                                 printk("WARNING: spurious interrupt detected (un
registered IRQ %d).\n", irq);
  195:                         } else if(kstat.sirqs == MAX_SPU_NOTICES) {
  196:                                 printk("WARNING: too many spurious interrupts; n
ot logging any more.\n");
  197:                         }
```

kernel/pic.c

```
198:                        kstat.sirqs++;
199:                        return;
200:                }
201:                if(irq > 7) {
202:                        outport_b(PIC_SLAVE, EOI);
203:                }
204:                outport_b(PIC_MASTER, EOI);
205:                return;
206:        }
207:
208:        disable_irq(irq);
209:        if(irq > 7) {
210:                outport_b(PIC_SLAVE, EOI);
211:        }
212:        outport_b(PIC_MASTER, EOI);
213:
214:        kstat.irqs++;
215:        irq_table[irq].ticks++;
216:        irq_table[irq].handler(&sc);
217:        enable_irq(irq);
218: }
219:
220: /* do bottom halves (interrupts are (FIXME) enabled) */
221: void do_bh(void)
222: {
223:        struct bh *b;
224:        void (*fn)(void);
225:
226:        if((b = bh_head)) {
227:                while(b) {
228:                        fn = b->fn;
229:                        b = b->next;
230:                        del_bh();
231:                        (*fn)();
232:                }
233:        }
234: }
235:
236: void pic_init(void)
237: {
238:        int n;
239:        struct bh *b;
240:
241:        memset_b(irq_table, NULL, sizeof(irq_table));
242:        memset_b(bh_pool, NULL, sizeof(bh_pool));
243:
244:        /* bh free list initialization */
245:        bh_pool_head = NULL;
246:        n = NR_BH;
247:        while(n--) {
248:                b = &bh_pool[n];
249:                put_free_bh(b);
250:        }
251:        bh_head = bh_tail = NULL;
252:
253:        /* remap interrupts for PIC1 */
254:        outport_b(PIC_MASTER, ICW1_RESET);
255:        outport_b(PIC_MASTER + DATA, IRQ0_ADDR);        /* ICW2 */
256:        outport_b(PIC_MASTER + DATA, 1 << CASCADE_IRQ); /* ICW3 */
257:        outport_b(PIC_MASTER + DATA, ICW4_8086EOI);
258:
259:        /* remap interrupts for PIC2 */
260:        outport_b(PIC_SLAVE, ICW1_RESET);
261:        outport_b(PIC_SLAVE + DATA, IRQ8_ADDR);         /* ICW2 */
262:        outport_b(PIC_SLAVE + DATA, CASCADE_IRQ);       /* ICW3 */
263:        outport_b(PIC_SLAVE + DATA, ICW4_8086EOI);
264:
```

kernel/pic.c


```
265:         /* mask all IRQs except cascade */
266:         outport_b(PIC_MASTER + DATA, ~(1 << CASCADE_IRQ));
267:
268:         /* mask all IRQs */
269:         outport_b(PIC_SLAVE + DATA, OCW1);
270: }
```

kernel/pit.c


```
    1: /*
    2:  * fiwix/kernel/pit.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/pit.h>
   10:
   11: void pit_beep_on(void)
   12: {
   13:         outport_b(MODEREG, SEL_CHAN2 | LSB_MSB | SQUARE_WAVE | BINARY_CTR);
   14:         outport_b(CHANNEL2, (OSCIL / BEEP_FREQ) & 0xFF);        /* LSB */
   15:         outport_b(CHANNEL2, (OSCIL / BEEP_FREQ) >> 8);          /* MSB */
   16:         outport_b(PS2_SYSCTRL_B, inport_b(PS2_SYSCTRL_B) | ENABLE_SDATA | ENABLE
_TMR2G);
   17: }
   18:
   19: void pit_beep_off(unsigned int unused)
   20: {
   21:         outport_b(PS2_SYSCTRL_B, inport_b(PS2_SYSCTRL_B) & ~(ENABLE_SDATA | ENAB
LE_TMR2G));
   22: }
   23:
   24: void pit_init(unsigned short int hertz)
   25: {
   26:         outport_b(MODEREG, SEL_CHAN0 | LSB_MSB | RATE_GEN | BINARY_CTR);
   27:         outport_b(CHANNEL0, (OSCIL / hertz) & 0xFF);    /* LSB */
   28:         outport_b(CHANNEL0, (OSCIL / hertz) >> 8);      /* MSB */
   29: }
```

kernel/process.c


```
    1: /*
    2:  * fiwix/kernel/process.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/mm.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/process.h>
   12: #include <fiwix/timer.h>
   13: #include <fiwix/sched.h>
   14: #include <fiwix/sleep.h>
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/string.h>
   17:
   18: struct proc *proc_table;
   19: struct proc *current;
   20:
   21: struct proc *proc_pool_head;
   22: struct proc *proc_table_head;
   23: struct proc *proc_table_tail;
   24: unsigned int free_proc_slots = 0;
   25:
   26: static struct resource slot_resource = { NULL, NULL };
   27: static struct resource pid_resource = { NULL, NULL };
   28:
   29: int nr_processes = 0;
   30: __pid_t lastpid = 0;
   31:
   32: int kill_pid(__pid_t pid, __sigset_t signum)
   33: {
   34:         struct proc *p;
   35:
   36:         FOR_EACH_PROCESS(p) {
   37:                 if(p->pid == pid && (p->state && p->state != PROC_ZOMBIE)) {
   38:                         return send_sig(p, signum);
   39:                 }
   40:         }
   41:         return -ESRCH;
   42: }
   43:
   44: int kill_pgrp(__pid_t pgid, __sigset_t signum)
   45: {
   46:         struct proc *p;
   47:         int found;
   48:
   49:         found = 0;
   50:         FOR_EACH_PROCESS(p) {
   51:                 if(p->pgid == pgid && (p->state && p->state != PROC_ZOMBIE)) {
   52:                         send_sig(p, signum);
   53:                         found = 1;
   54:                 }
   55:         }
   56:
   57:         if(!found) {
   58:                 return -ESRCH;
   59:         }
   60:         return 0;
   61: }
   62:
   63: /* sum up child (and its children) statistics */
   64: void add_crusage(struct proc *p, struct rusage *cru)
   65: {
   66:         cru->ru_utime.tv_sec = p->usage.ru_utime.tv_sec + p->cusage.ru_utime.tv_
sec;
```

kernel/process.c


```
   67:         cru->ru_utime.tv_usec = p->usage.ru_utime.tv_usec + p->cusage.ru_utime.t
v_usec;
   68:         if(cru->ru_utime.tv_usec >= 1000000) {
   69:                 cru->ru_utime.tv_sec++;
   70:                 cru->ru_utime.tv_usec -= 1000000;
   71:         }
   72:         cru->ru_stime.tv_sec = p->usage.ru_stime.tv_sec + p->cusage.ru_stime.tv_
sec;
   73:         cru->ru_stime.tv_usec = p->usage.ru_stime.tv_usec + p->cusage.ru_stime.t
v_usec;
   74:         if(cru->ru_stime.tv_usec >= 1000000) {
   75:                 cru->ru_stime.tv_sec++;
   76:                 cru->ru_stime.tv_usec -= 1000000;
   77:         }
   78:         cru->ru_maxrss = p->usage.ru_maxrss + p->cusage.ru_maxrss;
   79:         cru->ru_ixrss = p->usage.ru_ixrss + p->cusage.ru_ixrss;
   80:         cru->ru_idrss = p->usage.ru_idrss + p->cusage.ru_idrss;
   81:         cru->ru_isrss = p->usage.ru_isrss + p->cusage.ru_isrss;
   82:         cru->ru_minflt = p->usage.ru_minflt + p->cusage.ru_minflt;
   83:         cru->ru_majflt = p->usage.ru_majflt + p->cusage.ru_majflt;
   84:         cru->ru_nswap = p->usage.ru_nswap + p->cusage.ru_nswap;
   85:         cru->ru_inblock = p->usage.ru_inblock + p->cusage.ru_inblock;
   86:         cru->ru_oublock = p->usage.ru_oublock + p->cusage.ru_oublock;
   87:         cru->ru_msgsnd = p->usage.ru_msgsnd + p->cusage.ru_msgsnd;
   88:         cru->ru_msgrcv = p->usage.ru_msgrcv + p->cusage.ru_msgrcv;
   89:         cru->ru_nsignals = p->usage.ru_nsignals + p->cusage.ru_nsignals;
   90:         cru->ru_nvcsw = p->usage.ru_nvcsw + p->cusage.ru_nvcsw;
   91:         cru->ru_nivcsw = p->usage.ru_nivcsw + p->cusage.ru_nivcsw;
   92: }
   93:
   94: void get_rusage(struct proc *p, struct rusage *ru)
   95: {
   96:         struct rusage cru;
   97:
   98:         /*
   99:          * Conforms with SUSv3 which specifies that if SIGCHLD is being ignored
  100:          * then the child statistics should not be added to the values returned
  101:          * by RUSAGE_CHILDREN.
  102:          */
  103:         if(current->sigaction[SIGCHLD - 1].sa_handler == SIG_IGN) {
  104:                 return;
  105:         }
  106:
  107:         add_crusage(p, &cru);
  108:         memcpy_b(ru, &cru, sizeof(struct rusage));
  109: }
  110:
  111: /* add child statistics to parent */
  112: void add_rusage(struct proc *p)
  113: {
  114:         struct rusage cru;
  115:
  116:         add_crusage(p, &cru);
  117:         current->cusage.ru_utime.tv_sec += cru.ru_utime.tv_sec;
  118:         current->cusage.ru_utime.tv_usec += cru.ru_utime.tv_usec;
  119:         if(current->cusage.ru_utime.tv_usec >= 1000000) {
  120:                 current->cusage.ru_utime.tv_sec++;
  121:                 current->cusage.ru_utime.tv_usec -= 1000000;
  122:         }
  123:         current->cusage.ru_stime.tv_sec += cru.ru_stime.tv_sec;
  124:         current->cusage.ru_stime.tv_usec += cru.ru_stime.tv_usec;
  125:         if(current->cusage.ru_stime.tv_usec >= 1000000) {
  126:                 current->cusage.ru_stime.tv_sec++;
  127:                 current->cusage.ru_stime.tv_usec -= 1000000;
  128:         }
  129:         current->cusage.ru_maxrss += cru.ru_maxrss;
  130:         current->cusage.ru_ixrss += cru.ru_ixrss;
```

kernel/process.c


```
131:         current->cusage.ru_idrss += cru.ru_idrss;
132:         current->cusage.ru_isrss += cru.ru_isrss;
133:         current->cusage.ru_minflt += cru.ru_minflt;
134:         current->cusage.ru_majflt += cru.ru_majflt;
135:         current->cusage.ru_nswap += cru.ru_nswap;
136:         current->cusage.ru_inblock += cru.ru_inblock;
137:         current->cusage.ru_oublock += cru.ru_oublock;
138:         current->cusage.ru_msgsnd += cru.ru_msgsnd;
139:         current->cusage.ru_msgrcv += cru.ru_msgrcv;
140:         current->cusage.ru_nsignals += cru.ru_nsignals;
141:         current->cusage.ru_nvcsw += cru.ru_nvcsw;
142:         current->cusage.ru_nivcsw += cru.ru_nivcsw;
143: }
144:
145: struct proc * get_next_zombie(struct proc *parent)
146: {
147:         struct proc *p;
148:
149:         if(proc_table_head == NULL) {
150:                 PANIC("process table is empty!\n");
151:         }
152:
153:         FOR_EACH_PROCESS(p) {
154:                 if(p->ppid == parent->pid && p->state == PROC_ZOMBIE) {
155:                         return p;
156:                 }
157:         }
158:
159:         return NULL;
160: }
161:
162: __pid_t remove_zombie(struct proc *p)
163: {
164:         __pid_t pid;
165:
166:         pid = p->pid;
167:         kfree(p->tss.esp0);             current->rss--;
168:         kfree(P2V(p->tss.cr3));         current->rss--;
169:         release_proc(p);
170:         current->children--;
171:         return pid;
172: }
173:
174: /*
175:  * An orphaned process group is a process group in which the parent of every
176:  * member is either itself a member of the group or is not a member of the
177:  * group's session.
178:  */
179: int is_orphaned_pgrp(__pid_t pgid)
180: {
181:         struct proc *p, *pp;
182:
183:         FOR_EACH_PROCESS(p) {
184:                 if(p->pgid != pgid) {
185:                         continue;
186:                 }
187:                 if(p->state && p->state != PROC_ZOMBIE) {
188:                         pp = get_proc_by_pid(p->ppid);
189:                         /* return if only one is found that breaks the rule */
190:                         if(pp->pgid != pgid || pp->sid == p->sid) {
191:                                 return 0;
192:                         }
193:                 }
194:         }
195:         return 1;
196: }
197:
```

kernel/process.c


```
  198: struct proc * get_proc_free(void)
  199: {
  200:         struct proc *p = NULL;
  201:
  202:         if(free_proc_slots <= SAFE_SLOTS && !IS_SUPERUSER) {
  203:                 printk("WARNING: %s(): the remaining slots are only for root use
r!\n", __FUNCTION__);
  204:                 return NULL;
  205:         }
  206:
  207:         lock_resource(&slot_resource);
  208:
  209:         if(proc_pool_head) {
  210:
  211:                 /* get (remove) a process slot from the free list */
  212:                 p = proc_pool_head;
  213:                 proc_pool_head = proc_pool_head->next;
  214:
  215:                 free_proc_slots--;
  216:         } else {
  217:                 printk("WARNING: %s(): no more slots free in proc table!\n", __F
UNCTION__);
  218:         }
  219:
  220:         unlock_resource(&slot_resource);
  221:         return p;
  222: }
  223:
  224: void release_proc(struct proc *p)
  225: {
  226:         lock_resource(&slot_resource);
  227:
  228:         /* remove a process from the proc_table */
  229:         if(p == proc_table_tail) {
  230:                 if(proc_table_head == proc_table_tail) {
  231:                         proc_table_head = proc_table_tail = NULL;
  232:                 } else {
  233:                         proc_table_tail = proc_table_tail->prev;
  234:                         proc_table_tail->next = NULL;
  235:                 }
  236:         } else {
  237:                 p->prev->next = p->next;
  238:                 p->next->prev = p->prev;
  239:         }
  240:
  241:         /* initialize and put a process slot back in the free list */
  242:         memset_b(p, NULL, sizeof(struct proc));
  243:         p->next = proc_pool_head;
  244:         proc_pool_head = p;
  245:         free_proc_slots++;
  246:
  247:         unlock_resource(&slot_resource);
  248: }
  249:
  250: int get_unused_pid(void)
  251: {
  252:         short int loop;
  253:         struct proc *p;
  254:
  255:         loop = 0;
  256:         lock_resource(&pid_resource);
  257:
  258: loop:
  259:         lastpid++;
  260:         if(lastpid > MAX_PID_VALUE) {
  261:                 loop++;
  262:                 lastpid = INIT;
```

kernel/process.c


```
  263:         }
  264:         if(loop > 1) {
  265:                 printk("WARNING: %s(): system ran out of PID numbers!\n");
  266:                 return 0;
  267:         }
  268:         FOR_EACH_PROCESS(p) {
  269:                 if(p->state != PROC_UNUSED) {
  270:                         /*
  271:                          * Make sure the kernel never reuses active pid, pgid
  272:                          * or sid values.
  273:                          */
  274:                         if(lastpid == p->pid || lastpid == p->pgid || lastpid ==
 p->sid) {
  275:                                 goto loop;
  276:                         }
  277:                 }
  278:         }
  279:
  280:         unlock_resource(&pid_resource);
  281:         return lastpid;
  282: }
  283:
  284: struct proc * get_proc_by_pid(__pid_t pid)
  285: {
  286:         struct proc *p;
  287:
  288:         FOR_EACH_PROCESS(p) {
  289:                 if(p->state && p->pid == pid) {
  290:                         return p;
  291:                 }
  292:         }
  293:
  294:         PANIC("would return NULL! (current->pid=%d pid=%d)\n", current->pid, pid
);
  295:         return NULL;
  296: }
  297:
  298: int get_new_user_fd(int fd)
  299: {
  300:         int n;
  301:
  302:         for(n = fd; n < OPEN_MAX && n < current->rlim[RLIMIT_NOFILE].rlim_cur; n
++) {
  303:                 if(current->fd[n] == 0) {
  304:                         current->fd[n] = -1;
  305:                         current->fd_flags[n] = 0;
  306:                         return n;
  307:                 }
  308:         }
  309:
  310:         return -EMFILE;
  311: }
  312:
  313: void release_user_fd(int ufd)
  314: {
  315:         current->fd[ufd] = 0;
  316: }
  317:
  318: struct proc * kernel_process(int (*fn)(void))
  319: {
  320:         struct proc *p;
  321:
  322:         p = get_proc_free();
  323:         proc_slot_init(p);
  324:         p->pid = get_unused_pid();
  325:         p->ppid = 0;
  326:         p->flags |= PF_KPROC;
```

kernel/process.c


```
327:         p->priority = DEF_PRIORITY;
328:         if(!(p->tss.esp0 = kmalloc())) {
329:                 release_proc(p);
330:                 return NULL;
331:         }
332:         p->tss.esp0 += PAGE_SIZE - 4;
333:         p->rss++;
334:         p->tss.cr3 = (unsigned int)kpage_dir;
335:         p->tss.eip = (unsigned int)fn;
336:         p->tss.esp = p->tss.esp0;
337:         sprintk(p->pidstr, "%d", p->pid);
338:         p->state = PROC_RUNNING;
339:         return p;
340: }
341:
342: void proc_slot_init(struct proc *p)
343: {
344:         int n;
345:
346:         /* insert process at the end of proc_table */
347:         lock_resource(&slot_resource);
348:         if(proc_table_head == NULL) {
349:                 p->prev = NULL;
350:                 p->next = NULL;
351:                 proc_table_head = proc_table_tail = p;
352:         } else {
353:                 p->prev = proc_table_tail;
354:                 p->next = NULL;
355:                 proc_table_tail->next = p;
356:                 proc_table_tail = p;
357:         }
358:         p->sleep_prev = p->sleep_next = NULL;
359:         unlock_resource(&slot_resource);
360:
361:         memset_b(&p->tss, NULL, sizeof(struct i386tss));
362:         p->tss.io_bitmap_addr = (unsigned int)&p->io_bitmap;
363:
364:         /*
365:          * At the moment, all io_bitmap bits are setup to 0, which means full
366:          * access. This must be changed to 1 once we have fixed the ioperm()
367:          * system call.
368:          */
369:         for(n = 0; n < IO_BITMAP_SIZE + 1; n++) {
370:                 p->io_bitmap[n] = 0;    /* FIXME: change it to 1 */
371:         }
372:         p->state = PROC_IDLE;
373: }
374:
375: void proc_init(void)
376: {
377:         int n;
378:         struct proc *p;
379:
380:         memset_b(proc_table, NULL, proc_table_size);
381:
382:         /* free list initialization */
383:         proc_pool_head = NULL;
384:         n = (proc_table_size / sizeof(struct proc)) - 1;
385:         do {
386:                 p = &proc_table[n];
387:
388:                 /* fill the free list */
389:                 p->next = proc_pool_head;
390:                 proc_pool_head = p;
391:                 free_proc_slots++;
392:         } while(n--);
393:         proc_table_head = proc_table_tail = NULL;
```

kernel/process.c


```
394:
395:         /* slot 0 is for the IDLE process */
396:         current = get_proc_free();
397:         proc_slot_init(current);
398: }
```

kernel/sched.c


```
    1: /*
    2:  * fiwix/kernel/sched.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/const.h>
   11: #include <fiwix/sched.h>
   12: #include <fiwix/process.h>
   13: #include <fiwix/segments.h>
   14: #include <fiwix/timer.h>
   15: #include <fiwix/pic.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: extern struct seg_desc gdt[NR_GDT_ENTRIES];
   20: int need_resched = 0;
   21:
   22: static void context_switch(struct proc *next)
   23: {
   24:         struct proc *prev;
   25:
   26:         kstat.ctxt++;
   27:         prev = current;
   28:         set_tss(next);
   29:         current = next;
   30:         do_switch(&prev->tss.esp, &prev->tss.eip, next->tss.esp, next->tss.eip,
next->tss.cr3, TSS);
   31: }
   32:
   33: void set_tss(struct proc *p)
   34: {
   35:         struct seg_desc *g;
   36:
   37:         g = &gdt[TSS / sizeof(struct seg_desc)];
   38:
   39:         g->sd_lobase = (unsigned int)p;
   40:         g->sd_loflags = SD_TSSPRESENT;
   41:         g->sd_hibase = (char)(((unsigned int)p) >> 24);
   42: }
   43:
   44: /* Round Robin algorithm */
   45: void do_sched(void)
   46: {
   47:         int count;
   48:         struct proc *p, *selected;
   49:
   50:         /* let the current running process consume its time slice */
   51:         if(!need_resched && current->state == PROC_RUNNING && current->cpu_count
 > 0) {
   52:                 return;
   53:         }
   54:
   55:         need_resched = 0;
   56:         for(;;) {
   57:                 count = -1;
   58:                 selected = &proc_table[IDLE];
   59:
   60:                 FOR_EACH_PROCESS(p) {
   61:                         if(p->state == PROC_RUNNING && p->cpu_count > count) {
   62:                                 count = p->cpu_count;
   63:                                 selected = p;
   64:                         }
   65:                 }
```

kernel/sched.c


```
66:                 if(count) {
67:                         break;
68:                 }
69:
70:                 /* reassigns new quantum to all processes */
71:                 FOR_EACH_PROCESS(p) {
72:                         if(p->state) {
73:                                 p->cpu_count = p->priority;
74:                         }
75:                 }
76:         }
77:         if(current != selected) {
78:                 context_switch(selected);
79:         }
80: }
81:
82: void sched_init(void)
83: {
84:         get_system_time();
85:
86:         /* this should be more unpredictable */
87:         kstat.random_seed = CURRENT_TIME;
88: }
```

kernel/signal.c


```
    1: /*
    2:  * fiwix/kernel/signal.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/process.h>
   12: #include <fiwix/signal.h>
   13: #include <fiwix/sigcontext.h>
   14: #include <fiwix/sleep.h>
   15: #include <fiwix/sched.h>
   16: #include <fiwix/syscalls.h>
   17: #include <fiwix/mm.h>
   18: #include <fiwix/stdio.h>
   19: #include <fiwix/string.h>
   20:
   21: int send_sig(struct proc *p, __sigset_t signum)
   22: {
   23:         struct proc *z;
   24:
   25:         if(signum > NSIG || !p) {
   26:                 return -EINVAL;
   27:         }
   28:
   29:         if(!IS_SUPERUSER && current->euid != p->euid && current->sid != p->sid)
{
   30:                 return -EPERM;
   31:         }
   32:
   33:         /* kernel processes can't receive signals */
   34:         if(p->flags & PF_KPROC) {
   35:                 return 0;
   36:         }
   37:
   38:         switch(signum) {
   39:                 case 0:
   40:                         return 0;
   41:                 case SIGKILL:
   42:                 case SIGCONT:
   43:                         if(p->state == PROC_STOPPED) {
   44:                                 p->state = PROC_RUNNING;
   45:                         }
   46:                         /* discard all pending stop signals */
   47:                         p->sigpending &= SIG_MASK(SIGSTOP);
   48:                         p->sigpending &= SIG_MASK(SIGTSTP);
   49:                         p->sigpending &= SIG_MASK(SIGTTIN);
   50:                         p->sigpending &= SIG_MASK(SIGTTOU);
   51:                         break;
   52:                 case SIGSTOP:
   53:                 case SIGTSTP:
   54:                 case SIGTTIN:
   55:                 case SIGTTOU:
   56:                         /* discard all pending SIGCONT signals */
   57:                         p->sigpending &= SIG_MASK(SIGCONT);
   58:                         break;
   59:                 default:
   60:                         break;
   61:         }
   62:
   63:         if(p->sigaction[signum - 1].sa_handler == SIG_IGN && signum != SIGCHLD)
{
   64:                 return 0;
   65:         }
```

kernel/signal.c


```
   66:
   67:         /* SIGCHLD is ignored by default */
   68:         if(p->sigaction[signum - 1].sa_handler == SIG_DFL) {
   69:                 /*
   70:                  * INIT process is special, it only gets signals that have the
   71:                  * signal handler installed. This avoids to bring down the
   72:                  * system accidentally.
   73:                  */
   74:                 if(p->pid == INIT) {
   75:                         return 0;
   76:                 }
   77:                 if(signum == SIGCHLD) {
   78:                         return 0;
   79:                 }
   80:         }
   81:
   82:         /* if SIGCHLD is ignored reap its children (prevent zombies) */
   83:         if(p->sigaction[signum - 1].sa_handler == SIG_IGN) {
   84:                 if(signum == SIGCHLD) {
   85:                         while((z = get_next_zombie(p))) {
   86:                                 remove_zombie(z);
   87:                         }
   88:                         return 0;
   89:                 }
   90:         }
   91:
   92:         p->sigpending |= 1 << (signum - 1);
   93:         p->usage.ru_nsignals++;
   94:
   95:         /* wake up the process only if that signal is not blocked */
   96:         if(!(p->sigblocked & (1 << (signum - 1)))) {
   97:                 wakeup_proc(p);
   98:         }
   99:
  100:         return 0;
  101: }
  102:
  103: int issig(void)
  104: {
  105:         __sigset_t signum;
  106:         unsigned int mask;
  107:         struct proc *p;
  108:
  109:         if(!(current->sigpending & ~current->sigblocked)) {
  110:                 return 0;
  111:         }
  112:
  113:         for(signum = 1, mask = 1; signum < NSIG; signum++, mask <<= 1) {
  114:                 if(current->sigpending & mask) {
  115:                         if(signum == SIGCHLD) {
  116:                                 if(current->sigaction[signum - 1].sa_handler ==
SIG_IGN) {
  117:                                         /* this process ignores SIGCHLD */
  118:                                         while((p = get_next_zombie(current))) {
  119:                                                 remove_zombie(p);
  120:                                         }
  121:                                 } else {
  122:                                         if(current->sigaction[signum - 1].sa_han
dler != SIG_DFL) {
  123:                                                 return signum;
  124:                                         }
  125:                                 }
  126:                         } else {
  127:                                 if(current->sigaction[signum - 1].sa_handler !=
SIG_IGN) {
  128:                                         return signum;
  129:                                 }
```

kernel/signal.c


```
  130:                          }
  131:                          current->sigpending &= ~mask;
  132:                  }
  133:          }
  134:          return 0;
  135: }
  136:
  137: void psig(unsigned int stack)
  138: {
  139:          int len;
  140:          __sigset_t signum;
  141:          unsigned int mask;
  142:          struct sigcontext *sc;
  143:
  144:          sc = (struct sigcontext *)stack;
  145:          for(signum = 1, mask = 1; signum < NSIG; signum++, mask <<= 1) {
  146:                  if(current->sigpending & mask) {
  147:                          current->sigpending &= ~mask;
  148:
  149:                          if((unsigned int)current->sigaction[signum - 1].sa_handl
er) {
  150:                                  current->sigexecuting = mask;
  151:                                  if(!(current->sigaction[signum - 1].sa_flags & S
A_NODEFER)) {
  152:                                          current->sigblocked |= mask;
  153:                                  }
  154:
  155:                                  /* save the current sigcontext */
  156:                                  memcpy_b(&current->sc[signum - 1], sc, sizeof(st
ruct sigcontext));
  157:                                  /* setup the jump to the user signal handler */
  158:                                  len = ((int)end_sighandler_trampoline - (int)sig
handler_trampoline);
  159:                                  sc->oldesp -= len;
  160:                                  sc->oldesp -= 4;
  161:                                  sc->oldesp &= ~3;       /* round up */
  162:                                  memcpy_b((void *)sc->oldesp, sighandler_trampoli
ne, len);
  163:                                  sc->ecx = (unsigned int)current->sigaction[signu
m - 1].sa_handler;
  164:                                  sc->eax= signum;
  165:                                  sc->eip = sc->oldesp;
  166:
  167:                                  if(current->sigaction[signum - 1].sa_flags & SA_
RESETHAND) {
  168:                                          current->sigaction[signum - 1].sa_handle
r = SIG_DFL;
  169:                                  }
  170:                                  return;
  171:                          }
  172:                          if(current->sigaction[signum - 1].sa_handler == SIG_DFL)
 {
  173:                                  switch(signum) {
  174:                                          case SIGCONT:
  175:                                                  current->state = PROC_RUNNING;
  176:                                                  need_resched = 1;
  177:                                                  break;
  178:                                          case SIGSTOP:
  179:                                          case SIGTSTP:
  180:                                          case SIGTTIN:
  181:                                          case SIGTTOU:
  182:                                                  current->exit_code = signum;
  183:                                                  current->state = PROC_STOPPED;
  184:                                                  if(!(current->sigaction[signum -
 1].sa_flags & SA_NOCLDSTOP)) {
  185:                                                          send_sig(get_proc_by_pid
(current->ppid), SIGCHLD);
```

kernel/signal.c


```
  186:                                                    /* needed for job contro
l */
  187:                                                    wakeup(&sys_wait4);
  188:                                            }
  189:                                            need_resched = 1;
  190:                                            break;
  191:                                    case SIGCHLD:
  192:                                            break;
  193:                                    default:
  194:                                            do_exit(signum);
  195:                                }
  196:                        }
  197:                }
  198:        }
  199:
  200:        /* coming from a system call that needs to be restarted */
  201:        if(sc->err > 0) {
  202:                if(sc->eax == -ERESTART) {
  203:                        sc->eax = sc->err;      /* syscall was saved in 'err' */
  204:                        sc->eip -= 2;           /* point again to 'int 0x80' */
  205:                }
  206:        }
  207: }
```

kernel/sleep.c


```
    1: /*
    2:  * fiwix/kernel/sleep.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/limits.h>
   11: #include <fiwix/sleep.h>
   12: #include <fiwix/sched.h>
   13: #include <fiwix/signal.h>
   14: #include <fiwix/process.h>
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/string.h>
   17:
   18: #define NR_BUCKETS              (NR_PROCS * 10) / 100   /* 10% of NR_PROCS */
   19: #define SLEEP_HASH(addr)        ((addr) % NR_BUCKETS)
   20:
   21: struct proc *sleep_hash_table[NR_BUCKETS];
   22: static unsigned int area = 0;
   23:
   24: int sleep(void *address, int state)
   25: {
   26:         unsigned long int flags;
   27:         struct proc **h;
   28:         int signum, i;
   29:
   30:         /* return if it has signals */
   31:         if(state == PROC_INTERRUPTIBLE) {
   32:                 if((signum = issig())) {
   33:                         return signum;
   34:                 }
   35:         }
   36:
   37:         if(current->state == PROC_SLEEPING) {
   38:                 printk("WARNING: %s(): process with pid '%d' is already sleeping
!\n", __FUNCTION__, current->pid);
   39:                 return 0;
   40:         }
   41:
   42:         SAVE_FLAGS(flags); CLI();
   43:         i = SLEEP_HASH((unsigned int)address);
   44:         h = &sleep_hash_table[i];
   45:
   46:         /* insert process in the head */
   47:         if(!*h) {
   48:                 *h = current;
   49:                 (*h)->sleep_prev = (*h)->sleep_next = NULL;
   50:         } else {
   51:                 current->sleep_prev = NULL;
   52:                 current->sleep_next = *h;
   53:                 (*h)->sleep_prev = current;
   54:                 *h = current;
   55:         }
   56:         current->sleep_address = address;
   57:         current->state = PROC_SLEEPING;
   58:
   59:         do_sched();
   60:
   61:         signum = 0;
   62:         if(state == PROC_INTERRUPTIBLE) {
   63:                 signum = issig();
   64:         }
   65:
   66:         RESTORE_FLAGS(flags);
```

kernel/sleep.c


```
 67:         return signum;
 68: }
 69:
 70: void wakeup(void *address)
 71: {
 72:         unsigned long int flags;
 73:         struct proc **h;
 74:         int i;
 75:
 76:         SAVE_FLAGS(flags); CLI();
 77:         i = SLEEP_HASH((unsigned int)address);
 78:         h = &sleep_hash_table[i];
 79:
 80:         while(*h) {
 81:                 if((*h)->sleep_address == address) {
 82:                         (*h)->sleep_address = NULL;
 83:                         (*h)->state = PROC_RUNNING;
 84:                         need_resched = 1;
 85:                         if((*h)->sleep_next) {
 86:                                 (*h)->sleep_next->sleep_prev = (*h)->sleep_prev;
 87:                         }
 88:                         if((*h)->sleep_prev) {
 89:                                 (*h)->sleep_prev->sleep_next = (*h)->sleep_next;
 90:                         }
 91:                         if(h == &sleep_hash_table[i]) { /* if it's the head */
 92:                                 *h = (*h)->sleep_next;
 93:                                 continue;
 94:                         }
 95:                 }
 96:                 if(*h) {
 97:                         h = &(*h)->sleep_next;
 98:                 }
 99:         }
100:         RESTORE_FLAGS(flags);
101: }
102:
103: void wakeup_proc(struct proc *p)
104: {
105:         unsigned long int flags;
106:         struct proc **h;
107:         int i;
108:
109:         if(p->state != PROC_SLEEPING && p->state != PROC_STOPPED) {
110:                 return;
111:         }
112:
113:         SAVE_FLAGS(flags); CLI();
114:
115:         /* stopped processes don't have sleep address */
116:         if(p->sleep_address) {
117:                 if(p->sleep_next) {
118:                         p->sleep_next->sleep_prev = p->sleep_prev;
119:                 }
120:                 if(p->sleep_prev) {
121:                         p->sleep_prev->sleep_next = p->sleep_next;
122:                 }
123:
124:                 i = SLEEP_HASH((unsigned int)p->sleep_address);
125:                 h = &sleep_hash_table[i];
126:                 if(*h == p) {   /* if it's the head */
127:                         *h = (*h)->sleep_next;
128:                 }
129:         }
130:         p->sleep_address = NULL;
131:         p->state = PROC_RUNNING;
132:         need_resched = 1;
133:
```

kernel/sleep.c


```
134:         RESTORE_FLAGS(flags);
135:         return;
136: }
137:
138: void lock_resource(struct resource *resource)
139: {
140:         unsigned long int flags;
141:
142:         for(;;) {
143:                 SAVE_FLAGS(flags); CLI();
144:                 if(resource->locked) {
145:                         resource->wanted = 1;
146:                         RESTORE_FLAGS(flags);
147:                         sleep(resource, PROC_UNINTERRUPTIBLE);
148:                 } else {
149:                         break;
150:                 }
151:         }
152:         resource->locked = 1;
153:         RESTORE_FLAGS(flags);
154: }
155:
156: void unlock_resource(struct resource *resource)
157: {
158:         unsigned long int flags;
159:
160:         SAVE_FLAGS(flags); CLI();
161:         resource->locked = 0;
162:         if(resource->wanted) {
163:                 resource->wanted = 0;
164:                 wakeup(resource);
165:         }
166:         RESTORE_FLAGS(flags);
167: }
168:
169: int lock_area(unsigned int flag)
170: {
171:         unsigned long int flags;
172:         int retval;
173:
174:         SAVE_FLAGS(flags); CLI();
175:         retval = area & flag;
176:         area |= flag;
177:         RESTORE_FLAGS(flags);
178:
179:         return retval;
180: }
181:
182: int unlock_area(unsigned int flag)
183: {
184:         unsigned long int flags;
185:         int retval;
186:
187:         SAVE_FLAGS(flags); CLI();
188:         retval = area & flag;
189:         area &= ~flag;
190:         RESTORE_FLAGS(flags);
191:
192:         return retval;
193: }
194:
195: void sleep_init(void)
196: {
197:         memset_b(sleep_hash_table, NULL, sizeof(sleep_hash_table));
198: }
```

kernel/syscalls.c


```
 1: /*
 2:  * fiwix/kernel/syscalls.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/syscalls.h>
10: #include <fiwix/mm.h>
11: #include <fiwix/errno.h>
12: #include <fiwix/string.h>
13:
14: #ifdef __DEBUG__
15: #include <fiwix/stdio.h>
16: #endif /*__DEBUG__ */
17:
18: void free_name(const char *name)
19: {
20:         kfree((unsigned int)name);
21: }
22:
23: /*
24:  * This function has two objectives:
25:  * 1. to check the memory address validity of the char pointer supplied by the
26:  *    user, while at the same time limit its length to PAGE_SIZE (4096) bytes.
27:  * 2. to create a copy of 'filename' in the kernel data space before using it.
28:  */
29: int malloc_name(const char *filename, char **name)
30: {
31:         struct vma *vma;
32:         unsigned int start;
33:         short int n, len;
34:         char *b;
35:
36:         /*
37:          * Verifies if the 'vma' array of that process is not empty. It can
38:          * only be empty during the initialization of INIT, when it calls to
39:          * sys_execve and sys_open without having yet a proper setup.
40:          */
41:         if(current->vma[0].s_type != 0) {
42:                 if(!filename) {
43:                         return -EFAULT;
44:                 }
45:                 start = (unsigned int)filename;
46:                 if(!(vma = find_vma_region(start))) {
47:                         return -EFAULT;
48:                 }
49:                 if(!(vma->prot & PROT_READ)) {
50:                         return -EFAULT;
51:                 }
52:                 len = MIN(vma->end - start, PAGE_SIZE);
53:                 if(len < PAGE_SIZE) {
54:                         if((vma = find_vma_region(vma->end))) {
55:                                 if(vma->prot & PROT_READ) {
56:                                         len = PAGE_SIZE;
57:                                 }
58:                         }
59:                 }
60:         } else {
61:                 len = PAGE_SIZE;
62:         }
63:         if(!(b = (char *)kmalloc())) {
64:                 return -ENOMEM;
65:         }
66:         *name = b;
67:         for(n = 0; n < len; n++) {
```

kernel/syscalls.c


```
68:                 if(!(*b = *filename)) {
69:                         return 0;
70:                 }
71:                 b++;
72:                 filename++;
73:         }
74:
75:         free_name(*name);
76:         return -ENAMETOOLONG;
77: }
78:
79: int check_user_permission(struct inode *i)
80: {
81:         if(!IS_SUPERUSER) {
82:                 if(current->euid != i->i_uid) {
83:                         return 1;
84:                 }
85:         }
86:         return 0;
87: }
88:
89: int check_group(struct inode *i)
90: {
91:         int n;
92:         __gid_t gid;
93:
94:         if(current->flags & PF_USEREAL) {
95:                 gid = current->gid;
96:         } else {
97:                 gid = current->egid;
98:         }
99:
100:        if(i->i_gid == gid) {
101:                return 0;
102:        }
103:
104:        for(n = 0; n < NGROUPS_MAX; n++) {
105:                if(current->groups[n] == -1) {
106:                        break;
107:                }
108:                if(current->groups[n] == i->i_gid) {
109:                        return 0;
110:                }
111:        }
112:        return 1;
113: }
114:
115: int check_user_area(int type, const void *addr, unsigned int size)
116: {
117:         struct vma *vma;
118:         unsigned int start;
119:
120:         /*
121:          * Verifies if the 'vma' array of that process is not empty. It can
122:          * only be empty during the initialization of INIT, when it calls to
123:          * sys_execve and sys_open without having yet a proper setup.
124:          */
125:         if(current->vma[0].s_type != 0) {
126:                 start = (unsigned int)addr;
127:                 if(!(vma = find_vma_region(start))) {
128:                         return -EFAULT;
129:                 }
130:
131:                 for(;;) {
132:                         if(type == VERIFY_WRITE) {
133:                                 if(!(vma->prot & PROT_WRITE)) {
134:                                         return -EFAULT;
```

kernel/syscalls.c


```
135:                                    }
136:                            } else {
137:                                    if(!(vma->prot & PROT_READ)) {
138:                                            return -EFAULT;
139:                                    }
140:                            }
141:                            if(start + size <= vma->end) {
142:                                    break;
143:                            }
144:                            if(!(vma = find_vma_region(vma->end))) {
145:                                    return -EFAULT;
146:                            }
147:                    }
148:            }
149:
150:            return 0;
151: }
152:
153: int check_permission(int mask, struct inode *i)
154: {
155:            __uid_t uid;
156:
157:            if(current->flags & PF_USEREAL) {
158:                    uid = current->uid;
159:            } else {
160:                    uid = current->euid;
161:            }
162:
163:            if(uid == 0) {
164:                    return 0;
165:            }
166:            if(i->i_uid == uid) {
167:                    if((((i->i_mode >> 6) & 7) & mask) == mask) {
168:                            return 0;
169:                    }
170:            }
171:            if(!check_group(i)) {
172:                    if((((i->i_mode >> 3) & 7) & mask) == mask) {
173:                            return 0;
174:                    }
175:            }
176:            if(((i->i_mode & 7) & mask) == mask) {
177:                    return 0;
178:            }
179:
180:            return -EACCES;
181: }
182:
183: /* Linux 2.0.40 ABI system call (some from 2.2.26) */
184: void *syscall_table[] = {
185:            NULL,                           /* 0 */ /* sys_setup (-ENOSYS) */
186:            sys_exit,
187:            sys_fork,
188:            sys_read,
189:            sys_write,
190:            sys_open,                       /* 5 */
191:            sys_close,
192:            sys_waitpid,
193:            sys_creat,
194:            sys_link,
195:            sys_unlink,                     /* 10 */
196:            sys_execve,
197:            sys_chdir,
198:            sys_time,
199:            sys_mknod,
200:            sys_chmod,                      /* 15 */
201:            sys_chown,
```

kernel/syscalls.c


```
202:          NULL,                                   /* sys_break (-ENOSYS) */
203:          sys_stat,
204:          sys_lseek,
205:          sys_getpid,                     /* 20 */
206:          sys_mount,
207:          sys_umount,
208:          sys_setuid,
209:          sys_getuid,
210:          sys_stime,                      /* 25 */
211:          NULL,   // sys_ptrace
212:          sys_alarm,
213:          sys_fstat,
214:          sys_pause,
215:          sys_utime,                      /* 30 */
216:          NULL,                             /* sys_stty (-ENOSYS) */
217:          NULL,                             /* sys_gtty (-ENOSYS) */
218:          sys_access,
219:          NULL,   // sys_nice
220:          sys_ftime,                      /* 35 */
221:          sys_sync,
222:          sys_kill,
223:          sys_rename,
224:          sys_mkdir,
225:          sys_rmdir,                      /* 40 */
226:          sys_dup,
227:          sys_pipe,
228:          sys_times,
229:          NULL,   // sys_prof
230:          sys_brk,                        /* 45 */
231:          sys_setgid,
232:          sys_getgid,
233:          sys_signal,
234:          sys_geteuid,
235:          sys_getegid,                    /* 50 */
236:          NULL,   // sys_acct
237:          sys_umount2,
238:          NULL,                             /* sys_lock (-ENOSYS) */
239:          sys_ioctl,
240:          sys_fcntl,                      /* 55 */
241:          NULL,                             /* sys_mpx (-ENOSYS) */
242:          sys_setpgid,
243:          NULL,                             /* sys_ulimit (-ENOSYS) */
244:          sys_olduname,
245:          sys_umask,                      /* 60 */
246:          sys_chroot,
247:          sys_ustat,
248:          sys_dup2,
249:          sys_getppid,
250:          sys_getpgrp,                    /* 65 */
251:          sys_setsid,
252:          sys_sigaction,
253:          sys_sgetmask,
254:          sys_ssetmask,
255:          sys_setreuid,                   /* 70 */
256:          sys_setregid,
257:          sys_sigsuspend,
258:          sys_sigpending,
259:          sys_sethostname,
260:          sys_setrlimit,                  /* 75 */
261:          sys_getrlimit,
262:          sys_getrusage,
263:          sys_gettimeofday,
264:          sys_settimeofday,
265:          sys_getgroups,                  /* 80 */
266:          sys_setgroups,
267:          old_select,
268:          sys_symlink,
```

kernel/syscalls.c


```
269:         sys_lstat,
270:         sys_readlink,                  /* 85 */
271:         NULL,   // sys_uselib
272:         NULL,   // sys_swapon
273:         sys_reboot,
274:         NULL,   // old_readdir
275:         old_mmap,                      /* 90 */
276:         sys_munmap,
277:         sys_truncate,
278:         sys_ftruncate,
279:         sys_fchmod,
280:         sys_fchown,                    /* 95 */
281:         NULL,   // sys_getpriority
282:         NULL,   // sys_setpriority
283:         NULL,                                 /* sys_profil (-ENOSYS) */
284:         sys_statfs,
285:         sys_fstatfs,                   /* 100 */
286:         sys_ioperm,
287:         sys_socketcall, // sys_socketcall XXX
288:         NULL,   // sys_syslog
289:         sys_setitimer,
290:         sys_getitimer,                 /* 105 */
291:         sys_newstat,
292:         sys_newlstat,
293:         sys_newfstat,
294:         sys_uname,
295:         sys_iopl,                      /* 110 */
296:         NULL,   // sys_vhangup
297:         NULL,                                 /* sys_idle (-ENOSYS) */
298:         NULL,   // sys_vm86old
299:         sys_wait4,
300:         NULL,   // sys_swapoff         /* 115 */
301:         sys_sysinfo,
302:         NULL,   // sys_ipc
303:         sys_fsync,
304:         sys_sigreturn,
305:         NULL,   // sys_clone           /* 120 */
306:         sys_setdomainname,
307:         sys_newuname,
308:         NULL,   // sys_modify_ldt
309:         NULL,   // sys_adjtimex
310:         sys_mprotect,                  /* 125 */
311:         sys_sigprocmask,
312:         NULL,   // sys_create_module
313:         NULL,   // sys_init_module
314:         NULL,   // sys_delete_module
315:         NULL,   // sys_get_kernel_syms  /* 130 */
316:         NULL,   // sys_quotactl
317:         sys_getpgid,
318:         sys_fchdir,
319:         NULL,   // sys_bdflush
320:         NULL,   // sys_sysfs           /* 135 */
321:         sys_personality,
322:         NULL,                                 /* afs_syscall (-ENOSYS) */
323:         sys_setfsuid,
324:         sys_setfsgid,
325:         sys_llseek,                    /* 140 */
326:         sys_getdents,
327:         sys_select,
328:         sys_flock,
329:         NULL,   // sys_msync
330:         NULL,   // sys_readv           /* 145 */
331:         NULL,   // sys_writev
332:         sys_getsid,
333:         sys_fdatasync,
334:         NULL,   // sys_sysctl
335:         NULL,   // sys_mlock           /* 150 */
```

kernel/syscalls.c


```
  336:          NULL,   // sys_munlock
  337:          NULL,   // sys_mlockall
  338:          NULL,   // sys_munlockall
  339:          NULL,   // sys_sched_setparam
  340:          NULL,   // sys_sched_getparam   /* 155 */
  341:          NULL,   // sys_sched_setscheduler
  342:          NULL,   // sys_sched_getscheduler
  343:          NULL,   // sys_sched_yield
  344:          NULL,   // sys_sched_get_priority_max
  345:          NULL,   // sys_sched_get_priority_min   /* 160 */
  346:          NULL,   // sys_sched_rr_get_interval
  347:          sys_nanosleep,
  348:          NULL,   // sys_mremap
  349:
  350:          NULL,
  351:          NULL,
  352:          NULL,
  353:          NULL,
  354:          NULL,
  355:          NULL,
  356:          NULL,
  357:          NULL,
  358:          NULL,
  359:          NULL,
  360:          NULL,
  361:          NULL,
  362:          NULL,
  363:          NULL,
  364:          NULL,
  365:          NULL,
  366:          NULL,
  367:          NULL,
  368:          sys_chown,                      /* 182 */
  369:          sys_getcwd,
  370:          NULL,
  371:          NULL,
  372:          NULL,
  373:          NULL,
  374:          NULL,
  375:          NULL,
  376:          sys_fork,                       /* 190 (sys_vfork) */
  377: };
  378:
  379: static void do_bad_syscall(unsigned int num)
  380: {
  381: #ifdef __DEBUG__
  382:          printk("***** (pid %d) system call %d not supported yet *****\n", curren
t->pid, num);
  383: #endif /*__DEBUG__ */
  384: }
  385:
  386: /*
  387:  * The argument 'struct sigcontext' is needed because there are some system
  388:  * calls (such as sys_iopl and sys_fork) that need to get information from
  389:  * certain registers (EFLAGS and ESP). The rest of system calls will ignore
  390:  * such extra argument.
  391:  */
  392: int do_syscall(unsigned int num, int arg1, int arg2, int arg3, int arg4, int arg
5, struct sigcontext sc)
  393: {
  394:          int (*sys_func)(int, ...);
  395:
  396:          if(num > NR_SYSCALLS) {
  397:                  do_bad_syscall(num);
  398:                  return -ENOSYS;
  399:          }
  400:          sys_func = syscall_table[num];
```

kernel/syscalls.c


```
401:         if(!sys_func) {
402:                 do_bad_syscall(num);
403:                 return -ENOSYS;
404:         }
405:         current->sp = (unsigned int)&sc;
406:         return sys_func(arg1, arg2, arg3, arg4, arg5, &sc);
407: }
```

kernel/timer.c


```
    1: /*
    2:  * fiwix/kernel/timer.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/const.h>
   11: #include <fiwix/cmos.h>
   12: #include <fiwix/pit.h>
   13: #include <fiwix/timer.h>
   14: #include <fiwix/time.h>
   15: #include <fiwix/pic.h>
   16: #include <fiwix/sched.h>
   17: #include <fiwix/pic.h>
   18: #include <fiwix/cmos.h>
   19: #include <fiwix/signal.h>
   20: #include <fiwix/process.h>
   21: #include <fiwix/sleep.h>
   22: #include <fiwix/errno.h>
   23: #include <fiwix/stdio.h>
   24: #include <fiwix/string.h>
   25:
   26: /*
   27:  * timer.c implements a callout table using a singly linked list.
   28:  *
   29:  *  head
   30:  * +---------+  ----------+  ...  ----------+
   31:  * |data|next|  |data|next|  ...  |data|next|
   32:  * |    |  -->  |    |  -->  ...  |    |  / |
   33:  * +---------+  ----------+  ...  ----------+
   34:  *  (callout)    (callout)         (callout)
   35:  */
   36:
   37: struct callout callout_pool[NR_CALLOUTS];
   38: struct callout *callout_pool_head;
   39: struct callout *callout_head;
   40:
   41: static char month[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
   42: unsigned int avenrun[3] = { 0, 0, 0 };
   43:
   44: static unsigned int count_active_procs(void)
   45: {
   46:         int counter;
   47:         struct proc *p;
   48:
   49:         counter = 0;
   50:         FOR_EACH_PROCESS(p) {
   51:                 if(p->state == PROC_RUNNING) {
   52:                         counter += FIXED_1;
   53:                 }
   54:         }
   55:         return counter;
   56: }
   57:
   58: static void calc_load(void)
   59: {
   60:         unsigned int active_procs;
   61:         static int count = LOAD_FREQ;
   62:
   63:         if(count-- > 0) {
   64:                 return;
   65:         }
   66:
   67:         count = LOAD_FREQ;
```

kernel/timer.c


```
68:         active_procs = count_active_procs();
69:         CALC_LOAD(avenrun[0], EXP_1, active_procs);
70:         CALC_LOAD(avenrun[1], EXP_5, active_procs);
71:         CALC_LOAD(avenrun[2], EXP_15, active_procs);
72: }
73:
74: static struct callout *get_free_callout(void)
75: {
76:         struct callout *new;
77:
78:         new = NULL;
79:         if(callout_pool_head) {
80:                 new = callout_pool_head;
81:                 callout_pool_head = callout_pool_head->next;
82:                 new->next = NULL;
83:         }
84:         return new;
85: }
86:
87: static void put_free_callout(struct callout *old)
88: {
89:         old->next = callout_pool_head;
90:         callout_pool_head = old;
91: }
92:
93: static void do_del_callout(struct callout *c)
94: {
95:         struct callout **tmp;
96:
97:         if(callout_head) {
98:                 tmp = &callout_head;
99:                 while(*tmp) {
100:                         if((*tmp) == c) {
101:                                 if((*tmp)->next != NULL) {
102:                                         *tmp = (*tmp)->next;
103:                                         (*tmp)->expires += c->expires;
104:                                 } else {
105:                                         *tmp = NULL;
106:                                 }
107:                                 put_free_callout(c);
108:                                 break;
109:                         }
110:                         tmp = &(*tmp)->next;
111:                 }
112:         }
113:         return;
114: }
115:
116: void add_callout(struct callout_req *creq, unsigned int ticks)
117: {
118:         unsigned long int flags;
119:         struct callout *c, **tmp;
120:
121:         del_callout(creq);
122:         SAVE_FLAGS(flags); CLI();
123:
124:         if(!(c = get_free_callout())) {
125:                 printk("WARNING: %s(): no more callout slots!\n", __FUNCTION__);
126:                 RESTORE_FLAGS(flags);
127:                 return;
128:         }
129:
130:         /* setup the new callout */
131:         memset_b(c, NULL, sizeof(struct callout));
132:         c->expires = ticks;
133:         c->fn = creq->fn;
134:         c->arg = creq->arg;
```

kernel/timer.c


```
135:
136:         if(!callout_head) {
137:                 callout_head = c;
138:         } else {
139:                 tmp = &callout_head;
140:                 while(*tmp) {
141:                         if((*tmp)->expires > c->expires) {
142:                                 (*tmp)->expires -= c->expires;
143:                                 c->next = *tmp;
144:                                 break;
145:                         }
146:                         c->expires -= (*tmp)->expires;
147:                         tmp = &(*tmp)->next;
148:                 }
149:                 *tmp = c;
150:         }
151:         RESTORE_FLAGS(flags);
152:         return;
153: }
154:
155: void del_callout(struct callout_req *creq)
156: {
157:         unsigned long int flags;
158:         struct callout *c;
159:
160:         SAVE_FLAGS(flags); CLI();
161:         c = callout_head;
162:         while(c) {
163:                 if(c->fn == creq->fn && c->arg == creq->arg) {
164:                         do_del_callout(c);
165:                         break;
166:                 }
167:                 c = c->next;
168:         }
169:         RESTORE_FLAGS(flags);
170:         return;
171: }
172:
173: void do_timer(struct sigcontext *sc)
174: {
175:         if((++kstat.ticks % HZ) == 0) {
176:                 CURRENT_TIME++;
177:                 kstat.uptime++;
178:         }
179:
180:         add_bh(timer_bh);
181:
182:         /* FIXME: put this in 'timer_bh' */
183:         if(sc->cs == KERNEL_CS) {
184:                 current->usage.ru_stime.tv_usec += TICK;
185:                 if(current->usage.ru_stime.tv_usec >= 1000000) {
186:                         current->usage.ru_stime.tv_sec++;
187:                         current->usage.ru_stime.tv_usec -= 1000000;
188:                 }
189:                 if(current->pid != IDLE) {
190:                         kstat.cpu_system++;
191:                 }
192:         } else {
193:                 current->usage.ru_utime.tv_usec += TICK;
194:                 if(current->usage.ru_utime.tv_usec >= 1000000) {
195:                         current->usage.ru_utime.tv_sec++;
196:                         current->usage.ru_utime.tv_usec -= 1000000;
197:                 }
198:                 if(current->pid != IDLE) {
199:                         kstat.cpu_user++;
200:                 }
201:                 if(current->it_virt_value > 0) {
```

kernel/timer.c


```
  202:                          current->it_virt_value--;
  203:                          if(!current->it_virt_value) {
  204:                                  current->it_virt_value = current->it_virt_interv
al;
  205:                                  send_sig(current, SIGVTALRM);
  206:                          }
  207:                  }
  208:          }
  209: }
  210:
  211: unsigned long int tv2ticks(const struct timeval *tv)
  212: {
  213:          return((tv->tv_sec * HZ) + tv->tv_usec * HZ / 1000000);
  214: }
  215:
  216: void ticks2tv(long int ticks, struct timeval *tv)
  217: {
  218:          tv->tv_sec = ticks / HZ;
  219:          tv->tv_usec = (ticks % HZ) * 1000000 / HZ;
  220:          return;
  221: }
  222:
  223: int setitimer(int which, const struct itimerval *new_value, struct itimerval *ol
d_value)
  224: {
  225:          switch(which) {
  226:                  case ITIMER_REAL:
  227:                          if((unsigned int)old_value) {
  228:                                  ticks2tv(current->it_real_interval, &old_value->
it_interval);
  229:                                  ticks2tv(current->it_real_value, &old_value->it_
value);
  230:                          }
  231:                          current->it_real_interval = tv2ticks(&new_value->it_inte
rval);
  232:                          current->it_real_value = tv2ticks(&new_value->it_value);
  233:                          break;
  234:                  case ITIMER_VIRTUAL:
  235:                          if((unsigned int)old_value) {
  236:                                  ticks2tv(current->it_virt_interval, &old_value->
it_interval);
  237:                                  ticks2tv(current->it_virt_value, &old_value->it_
value);
  238:                          }
  239:                          current->it_virt_interval = tv2ticks(&new_value->it_inte
rval);
  240:                          current->it_virt_value = tv2ticks(&new_value->it_value);
  241:                          break;
  242:                  case ITIMER_PROF:
  243:                          if((unsigned int)old_value) {
  244:                                  ticks2tv(current->it_prof_interval, &old_value->
it_interval);
  245:                                  ticks2tv(current->it_prof_value, &old_value->it_
value);
  246:                          }
  247:                          current->it_prof_interval = tv2ticks(&new_value->it_inte
rval);
  248:                          current->it_prof_value = tv2ticks(&new_value->it_value);
  249:                          break;
  250:                  default:
  251:                          return -EINVAL;
  252:          }
  253:
  254:          return 0;
  255: }
  256:
  257: unsigned long int mktime(struct mt *mt)
```

kernel/timer.c


```
  258: {
  259:         int n, total_days;
  260:         unsigned long int seconds;
  261:
  262:         total_days = 0;
  263:
  264:         for(n = UNIX_EPOCH; n < mt->mt_year; n++) {
  265:                 total_days += DAYS_PER_YEAR(n);
  266:         }
  267:         for(n = 0; n < (mt->mt_month - 1); n++) {
  268:                 total_days += month[n];
  269:                 if(n == 1) {
  270:                         total_days += LEAP_YEAR(mt->mt_year) ? 1 : 0;
  271:                 }
  272:         }
  273:
  274:         total_days += (mt->mt_day - 1);
  275:         seconds = total_days * SECS_PER_DAY;
  276:         seconds += mt->mt_hour * SECS_PER_HOUR;
  277:         seconds += mt->mt_min * SECS_PER_MIN;
  278:         seconds += mt->mt_sec;
  279:         return seconds;
  280: }
  281:
  282: void timer_bh(void)
  283: {
  284:         struct proc *p;
  285:
  286:         if(current->usage.ru_utime.tv_sec + current->usage.ru_stime.tv_sec > cur
rent->rlim[RLIMIT_CPU].rlim_cur) {
  287:                 send_sig(current, SIGXCPU);
  288:         }
  289:
  290:         if(current->it_prof_value > 0) {
  291:                 current->it_prof_value--;
  292:                 if(!current->it_prof_value) {
  293:                         current->it_prof_value = current->it_prof_interval;
  294:                         send_sig(current, SIGPROF);
  295:                 }
  296:         }
  297:
  298:         calc_load();
  299:         FOR_EACH_PROCESS(p) {
  300:                 if(!p->state) {
  301:                         continue;
  302:                 }
  303:                 if(p->timeout > 0 && p->timeout < INFINITE_WAIT) {
  304:                         p->timeout--;
  305:                         if(!p->timeout) {
  306:                                 wakeup_proc(p);
  307:                         }
  308:                 }
  309:                 if(p->it_real_value > 0) {
  310:                         p->it_real_value--;
  311:                         if(!p->it_real_value) {
  312:                                 p->it_real_value = p->it_real_interval;
  313:                                 send_sig(p, SIGALRM);
  314:                         }
  315:                 }
  316:         }
  317:
  318:         /* callouts */
  319:         if(callout_head) {
  320:                 if(callout_head->expires > 0) {
  321:                         callout_head->expires--;
  322:                         if(!callout_head->expires) {
  323:                                 add_bh(callouts_bh);
```

kernel/timer.c

```
324:                        }
325:                } else {
326:                        printk("%s(): callout losing ticks.\n", __FUNCTION__);
327:                        add_bh(callouts_bh);
328:                }
329:        }
330:
331:        if(current->pid > IDLE && --current->cpu_count <= 0) {
332:                current->cpu_count = 0;
333:                need_resched = 1;
334:        }
335: }
336:
337: void callouts_bh(void)
338: {
339:        struct callout *c;
340:        void (*fn)(unsigned int);
341:        unsigned int arg;
342:
343:        while(callout_head) {
344:                if(callout_head->expires) {
345:                        break;
346:                }
347:                if(lock_area(AREA_CALLOUT)) {
348:                        continue;
349:                }
350:                fn = callout_head->fn;
351:                arg = callout_head->arg;
352:                c = callout_head;
353:                callout_head = callout_head->next;
354:                put_free_callout(c);
355:                unlock_area(AREA_CALLOUT);
356:                fn(arg);
357:        }
358: }
359:
360: void get_system_time(void)
361: {
362:        short int cmos_century;
363:        struct mt mt;
364:
365:        /* read date and time from CMOS */
366:        mt.mt_sec = cmos_read_date(CMOS_SEC);
367:        mt.mt_min = cmos_read_date(CMOS_MIN);
368:        mt.mt_hour = cmos_read_date(CMOS_HOUR);
369:        mt.mt_day = cmos_read_date(CMOS_DAY);
370:        mt.mt_month = cmos_read_date(CMOS_MONTH);
371:        mt.mt_year = cmos_read_date(CMOS_YEAR);
372:        cmos_century = cmos_read_date(CMOS_CENTURY);
373:        mt.mt_year += cmos_century * 100;
374:
375:        kstat.boot_time = CURRENT_TIME = mktime(&mt);
376: }
377:
378: void set_system_time(__time_t t)
379: {
380:        int sec, spm, min, hour, d, m, y;
381:
382:        sec = t;
383:        y = 1970;
384:        while(sec >= (DAYS_PER_YEAR(y) * SECS_PER_DAY)) {
385:                sec -= (DAYS_PER_YEAR(y) * SECS_PER_DAY);
386:                y++;
387:        }
388:
389:        m = 0;
390:        while(sec > month[m] * SECS_PER_DAY) {
```

kernel/timer.c


```
  391:                  spm = month[m] * SECS_PER_DAY;
  392:                  if(m == 1) {
  393:                          spm = LEAP_YEAR(y) ? spm + SECS_PER_DAY : spm;
  394:                  }
  395:                  sec -= spm;
  396:                  m++;
  397:          }
  398:          m++;
  399:
  400:          d = 1;
  401:          while(sec >= SECS_PER_DAY) {
  402:                  sec -= SECS_PER_DAY;
  403:                  d++;
  404:          }
  405:
  406:          hour = 0;
  407:          while(sec >= SECS_PER_HOUR) {
  408:                  sec -= SECS_PER_HOUR;
  409:                  hour++;
  410:          }
  411:
  412:          min = 0;
  413:          while(sec >= SECS_PER_MIN) {
  414:                  sec -= SECS_PER_MIN;
  415:                  min++;
  416:          }
  417:
  418:          /* write date and time to CMOS */
  419:          cmos_write_date(CMOS_SEC, sec);
  420:          cmos_write_date(CMOS_MIN, min);
  421:          cmos_write_date(CMOS_HOUR, hour);
  422:          cmos_write_date(CMOS_DAY, d);
  423:          cmos_write_date(CMOS_MONTH, m);
  424:          cmos_write_date(CMOS_YEAR, y % 100);
  425:          cmos_write_date(CMOS_CENTURY, (y - (y % 100)) / 100);
  426:
  427:          CURRENT_TIME = t;
  428: }
  429:
  430: void timer_init(void)
  431: {
  432:          int n;
  433:          struct callout *c;
  434:
  435:          pit_init(HZ);
  436:          memset_b(callout_pool, NULL, sizeof(callout_pool));
  437:
  438:          /* callout free list initialization */
  439:          callout_pool_head = NULL;
  440:          n = NR_CALLOUTS;
  441:          while(n--) {
  442:                  c = &callout_pool[n];
  443:                  put_free_callout(c);
  444:          }
  445:          callout_head = NULL;
  446:
  447:          printk("clock     -                %d    type=PIT Hz=%d\n", TIMER_IRQ, H
Z);
  448:          if(!register_irq(TIMER_IRQ, "timer", do_timer)) {
  449:                  enable_irq(TIMER_IRQ);
  450:          }
  451: }
```

kernel/traps.c


```
    1: /*
    2:  * fiwix/kernel/traps.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/traps.h>
   11: #include <fiwix/cpu.h>
   12: #include <fiwix/mm.h>
   13: #include <fiwix/process.h>
   14: #include <fiwix/signal.h>
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/string.h>
   17: #include <fiwix/sched.h>
   18:
   19: /*
   20:  * PS/2 System Control Port B
   21:  * --------------------------------------
   22:  * bit 7 -> system board RAM parity check
   23:  * bit 6 -> channel check
   24:  * bit 5 -> timer 2 (speaker time) output
   25:  * bit 4 -> refresh request (toggle)
   26:  * bit 3 -> channel check status
   27:  * bit 2 -> parity check status
   28:  * bit 1 -> speaker data status
   29:  * bit 0 -> timer 2 gate to speaker status
   30:  */
   31: #define PS2_SYSCTRL_B   0x61    /* PS/2 system control port B (read) */
   32:
   33: struct traps traps_table[NR_EXCEPTIONS] = {
   34:         { "Divide Error", do_divide_error, 0 },
   35:         { "Debug", do_debug, 0 },
   36:         { "NMI Interrupt", do_nmi_interrupt, 0 },
   37:         { "Breakpoint", do_breakpoint, 0 },
   38:         { "Overflow" , do_overflow, 0 },
   39:         { "BOUND Range Exceeded", do_bound, 0 },
   40:         { "Invalid Opcode", do_invalid_opcode, 0 },
   41:         { "Device Not Available (No Math Coprocessor)", do_no_math_coprocessor,
0 },
   42:         { "Double Fault", do_double_fault, 1 },
   43:         { "Coprocessor Segment Overrun", do_coprocessor_segment_overrun, 0 },
   44:         { "Invalid TSS", do_invalid_tss, 1 },
   45:         { "Segment Not Present", do_segment_not_present, 1 },
   46:         { "Stack-Segment Fault", do_stack_segment_fault, 1 },
   47:         { "General Protection", do_general_protection, 1 },
   48:         { "Page Fault", do_page_fault, 1 },
   49:         { "Intel reserved", do_reserved, 0 },
   50:         { "x87 FPU Floating-Point Error", do_floating_point_error, 0 },
   51:         { "Alignment Check", do_alignment_check, 1 },
   52:         { "Machine Check", do_machine_check, 0 },
   53:         { "SIMD Floating-Point Exception", do_simd_fault, 0 },
   54:         { "Intel reserved", do_reserved, 0 },
   55:         { "Intel reserved", do_reserved, 0 },
   56:         { "Intel reserved", do_reserved, 0 },
   57:         { "Intel reserved", do_reserved, 0 },
   58:         { "Intel reserved", do_reserved, 0 },
   59:         { "Intel reserved", do_reserved, 0 },
   60:         { "Intel reserved", do_reserved, 0 },
   61:         { "Intel reserved", do_reserved, 0 },
   62:         { "Intel reserved", do_reserved, 0 },
   63:         { "Intel reserved", do_reserved, 0 },
   64:         { "Intel reserved", do_reserved, 0 },
   65:         { "Intel reserved", do_reserved, 0 }
   66: };
```

kernel/traps.c


```
 67:
 68: void do_divide_error(unsigned int trap, struct sigcontext *sc)
 69: {
 70:         if(dump_registers(trap, sc)) {
 71:                 PANIC("");
 72:         }
 73:         send_sig(current, SIGFPE);
 74:         return;
 75: }
 76:
 77: void do_debug(unsigned int trap, struct sigcontext *sc)
 78: {
 79:         if(dump_registers(trap, sc)) {
 80:                 PANIC("");
 81:         }
 82:         send_sig(current, SIGTRAP);
 83:         return;
 84: }
 85:
 86: void do_nmi_interrupt(unsigned int trap, struct sigcontext *sc)
 87: {
 88:         unsigned char error;
 89:
 90:         error = inport_b(PS2_SYSCTRL_B);
 91:
 92:         printk("NMI received: ", error);
 93:         switch(error) {
 94:                 case 0x80:
 95:                         printk("parity check occurred. Defective RAM chips?\n");
 96:                         break;
 97:                 default:
 98:                         printk("unknown error 0x%x\n", error);
 99:                         break;
100:         }
101:
102:         if(dump_registers(trap, sc)) {
103:                 PANIC("");
104:         }
105:         send_sig(current, SIGSEGV);
106:         return;
107: }
108:
109: void do_breakpoint(unsigned int trap, struct sigcontext *sc)
110: {
111:         if(dump_registers(trap, sc)) {
112:                 PANIC("");
113:         }
114:         send_sig(current, SIGTRAP);
115:         return;
116: }
117:
118: void do_overflow(unsigned int trap, struct sigcontext *sc)
119: {
120:         if(dump_registers(trap, sc)) {
121:                 PANIC("");
122:         }
123:         send_sig(current, SIGSEGV);
124:         return;
125: }
126:
127: void do_bound(unsigned int trap, struct sigcontext *sc)
128: {
129:         if(dump_registers(trap, sc)) {
130:                 PANIC("");
131:         }
132:         send_sig(current, SIGSEGV);
133:         return;
```

kernel/traps.c


```
134: }
135:
136: void do_invalid_opcode(unsigned int trap, struct sigcontext *sc)
137: {
138:         if(dump_registers(trap, sc)) {
139:                 PANIC("");
140:         }
141:         send_sig(current, SIGILL);
142:         return;
143: }
144:
145: void do_no_math_coprocessor(unsigned int trap, struct sigcontext *sc)
146: {
147:         /* floating-point emulation would go here */
148:
149:         if(dump_registers(trap, sc)) {
150:                 PANIC("No coprocessor/emulation found.\n");
151:         }
152:         send_sig(current, SIGILL);
153:         return;
154: }
155:
156: void do_double_fault(unsigned int trap, struct sigcontext *sc)
157: {
158:         if(dump_registers(trap, sc)) {
159:                 PANIC("");
160:         }
161:         send_sig(current, SIGSEGV);
162:         return;
163: }
164:
165: void do_coprocessor_segment_overrun(unsigned int trap, struct sigcontext *sc)
166: {
167:         if(dump_registers(trap, sc)) {
168:                 PANIC("");
169:         }
170:         send_sig(current, SIGFPE);
171:         return;
172: }
173:
174: void do_invalid_tss(unsigned int trap, struct sigcontext *sc)
175: {
176:         if(dump_registers(trap, sc)) {
177:                 PANIC("");
178:         }
179:         send_sig(current, SIGSEGV);
180:         return;
181: }
182:
183: void do_segment_not_present(unsigned int trap, struct sigcontext *sc)
184: {
185:         if(dump_registers(trap, sc)) {
186:                 PANIC("");
187:         }
188:         send_sig(current, SIGBUS);
189:         return;
190: }
191:
192: void do_stack_segment_fault(unsigned int trap, struct sigcontext *sc)
193: {
194:         if(dump_registers(trap, sc)) {
195:                 PANIC("");
196:         }
197:         send_sig(current, SIGBUS);
198:         return;
199: }
200:
```

kernel/traps.c


```
201: void do_general_protection(unsigned int trap, struct sigcontext *sc)
202: {
203:         if(dump_registers(trap, sc)) {
204:                 PANIC("");
205:         }
206:         send_sig(current, SIGSEGV);
207:         return;
208: }
209:
210: /* do_page_fault() resides in mm/fault.c */
211:
212: void do_reserved(unsigned int trap, struct sigcontext *sc)
213: {
214:         if(dump_registers(trap, sc)) {
215:                 PANIC("");
216:         }
217:         send_sig(current, SIGSEGV);
218:         return;
219: }
220:
221: void do_floating_point_error(unsigned int trap, struct sigcontext *sc)
222: {
223:         if(dump_registers(trap, sc)) {
224:                 PANIC("");
225:         }
226:         send_sig(current, SIGFPE);
227:         return;
228: }
229:
230: void do_alignment_check(unsigned int trap, struct sigcontext *sc)
231: {
232:         if(dump_registers(trap, sc)) {
233:                 PANIC("");
234:         }
235:         send_sig(current, SIGSEGV);
236:         return;
237: }
238:
239: void do_machine_check(unsigned int trap, struct sigcontext *sc)
240: {
241:         if(dump_registers(trap, sc)) {
242:                 PANIC("");
243:         }
244:         send_sig(current, SIGSEGV);
245:         return;
246: }
247:
248: void do_simd_fault(unsigned int trap, struct sigcontext *sc)
249: {
250:         if(dump_registers(trap, sc)) {
251:                 PANIC("");
252:         }
253:         send_sig(current, SIGSEGV);
254:         return;
255: }
256:
257: void trap_handler(unsigned int trap, struct sigcontext sc)
258: {
259:         traps_table[trap].handler(trap, &sc);
260:
261:         /* avoids confusion with -RESTART return value */
262:         sc.err = -sc.err;
263: }
264:
265: static const char * elf_lookup_symbol(unsigned int addr)
266: {
267:         Elf32_Sym *sym;
```

kernel/traps.c


```
  268:          unsigned int n;
  269:
  270:          sym = (Elf32_Sym *)symtab->sh_addr;
  271:          for(n = 0; n < symtab->sh_size / sizeof(Elf32_Sym); n++, sym++) {
  272:                  if(ELF32_ST_TYPE(sym->st_info) != STT_FUNC) {
  273:                          continue;
  274:                  }
  275:                  if(addr >= sym->st_value && addr < (sym->st_value + sym->st_size
)) {
  276:                          return (const char *)strtab->sh_addr + sym->st_name;
  277:                  }
  278:          }
  279:          return NULL;
  280: }
  281:
  282: int dump_registers(unsigned int trap, struct sigcontext *sc)
  283: {
  284:          unsigned int cr2, addr, n;
  285:          unsigned int *sp;
  286:          const char *str;
  287:
  288:          printk("\n");
  289:          if(trap == 14) {        /* Page Fault */
  290:                  GET_CR2(cr2);
  291:                  printk("%s at 0x%08x (%s) with error code 0x%08x (0b%b)\n", trap
s_table[trap].name, cr2, sc->err & PFAULT_W ? "writing" : "reading", sc->err, sc->err);
  292:          } else {
  293:                  printk("EXCEPTION: %s", traps_table[trap].name);
  294:                  if(traps_table[trap].errcode) {
  295:                          printk(": error code 0x%08x (0b%b)", sc->err, sc->err);
  296:                  }
  297:                  printk("\n");
  298:          }
  299:
  300:          printk("Process '%s' with pid %d", current->argv0, current->pid);
  301:          if(sc->cs == KERNEL_CS) {
  302:                  printk(" in '%s()'.", elf_lookup_symbol(sc->eip));
  303:          }
  304:          printk("\n");
  305:
  306:          printk(" cs: 0x%08x\teip: 0x%08x\tefl: 0x%08x\t ss: 0x%08x\tesp: 0x%08x\
n", sc->cs, sc->eip, sc->eflags, sc->oldss, sc->oldesp);
  307:          printk("eax: 0x%08x\tebx: 0x%08x\tecx: 0x%08x\tedx: 0x%08x\n", sc->eax,
sc->ebx, sc->ecx, sc->edx);
  308:          printk("esi: 0x%08x\tedi: 0x%08x\tesp: 0x%08x\tebp: 0x%08x\n", sc->esi,
sc->edi, sc->esp, sc->ebp);
  309:          printk(" ds: 0x%08x\t es: 0x%08x\t fs: 0x%08x\t gs: 0x%08x\n", sc->ds, s
c->es, sc->fs, sc->gs);
  310:
  311:          if(sc->cs == KERNEL_CS) {
  312:                  printk("Stack:\n");
  313:                  GET_ESP(sp);
  314:                  sp = (unsigned int *)sp;
  315:                  for(n = 1; n <= 32; n++) {
  316:                          printk(" %08x", *sp);
  317:                          sp++;
  318:                          if(!(n % 8)) {
  319:                                  printk("\n");
  320:                          }
  321:                  }
  322:                  printk("Backtrace:\n");
  323:                  GET_ESP(sp);
  324:                  sp = (unsigned int *)sp;
  325:                  for(n = 0; n < 128; n++) {
  326:                          addr = *sp;
  327:                          str = elf_lookup_symbol(addr);
  328:                          if(str) {
```

kernel/traps.c


```
329:                                    printk("<0x%08x> %s()\n", addr, str);
330:                            }
331:                            sp++;
332:                    }
333:            }
334:
335:            /* panics if the exception has been in kernel mode */
336:            if(current->flags & PF_KPROC || sc->cs == KERNEL_CS) {
337:                    return 1;
338:            }
339:
340:            return 0;
341: }
```

kernel/syscalls/access.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/access.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/stat.h>
   11: #include <fiwix/errno.h>
   12: #include <fiwix/string.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/process.h>
   17: #endif /*__DEBUG__ */
   18:
   19: int sys_access(const char *filename, __mode_t mode)
   20: {
   21:         struct inode *i;
   22:         char *tmp_name;
   23:         int errno;
   24:
   25: #ifdef __DEBUG__
   26:         printk("(pid %d) sys_access('%s', %d)", current->pid, filename, mode);
   27: #endif /*__DEBUG__ */
   28:
   29:         if((mode & S_IRWXO) != mode) {
   30:                 return -EINVAL;
   31:         }
   32:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   33:                 return errno;
   34:         }
   35:         current->flags |= PF_USEREAL;
   36:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
   37:                 current->flags &= ~PF_USEREAL;
   38:                 free_name(tmp_name);
   39:                 return errno;
   40:         }
   41:         if(mode & TO_WRITE) {
   42:                 if(S_ISREG(i->i_mode) || S_ISDIR(i->i_mode) || S_ISLNK(i->i_mode
)) {
   43:                         if(IS_RDONLY_FS(i)) {
   44:                                 current->flags &= ~PF_USEREAL;
   45:                                 iput(i);
   46:                                 free_name(tmp_name);
   47:                                 return -EROFS;
   48:                         }
   49:                 }
   50:         }
   51:         errno = check_permission(mode, i);
   52:
   53: #ifdef __DEBUG__
   54:         printk(" -> returning %d\n", errno);
   55: #endif /*__DEBUG__ */
   56:
   57:         current->flags &= ~PF_USEREAL;
   58:         iput(i);
   59:         free_name(tmp_name);
   60:         return errno;
   61: }
```

kernel/syscalls/alarm.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/alarm.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/time.h>
10:
11: #ifdef __DEBUG__
12: #include <fiwix/stdio.h>
13: #include <fiwix/process.h>
14: #endif /*__DEBUG__ */
15:
16: int sys_alarm(unsigned int secs)
17: {
18:         struct itimerval value, ovalue;
19:
20: #ifdef __DEBUG__
21:         printk("(pid %d) sys_alarm(%d)\n", current->pid, secs);
22: #endif /*__DEBUG__ */
23:
24:         value.it_interval.tv_sec = 0;
25:         value.it_interval.tv_usec = 0;
26:         value.it_value.tv_sec = secs;
27:         value.it_value.tv_usec = 0;
28:         setitimer(ITIMER_REAL, &value, &ovalue);
29:
30:         /*
31:          * If there are still some usecs left and since the return value has
32:          * not enough granularity for them, then just add 1 second to it.
33:          */
34:         if(ovalue.it_value.tv_usec) {
35:                 ovalue.it_value.tv_sec++;
36:         }
37:
38:         return ovalue.it_value.tv_sec;
39: }
```

kernel/syscalls/brk.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/brk.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9: #include <fiwix/mm.h>
10: #include <fiwix/mman.h>
11: #include <fiwix/errno.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #endif /*__DEBUG__ */
16:
17: int sys_brk(unsigned int brk)
18: {
19:         unsigned int newbrk;
20:
21: #ifdef __DEBUG__
22:         printk("(pid %d) sys_brk(0x%08x) -> ", current->pid, brk);
23: #endif /*__DEBUG__ */
24:
25:         if(!brk || brk < current->brk_lower) {
26: #ifdef __DEBUG__
27:                 printk("0x%08x\n", current->brk);
28: #endif /*__DEBUG__ */
29:                 return current->brk;
30:         }
31:
32:         newbrk = PAGE_ALIGN(brk);
33:         if(newbrk == current->brk || newbrk < current->brk_lower) {
34: #ifdef __DEBUG__
35:                 printk("0x%08x\n", current->brk);
36: #endif /*__DEBUG__ */
37:                 return brk;
38:         }
39:
40:         if(brk < current->brk) {
41:                 do_munmap(newbrk, current->brk - newbrk);
42:                 current->brk = brk;
43:                 return brk;
44:         }
45:         if(!expand_heap(newbrk)) {
46:                 current->brk = brk;
47:         } else {
48:                 return -ENOMEM;
49:         }
50: #ifdef __DEBUG__
51:         printk("0x%08x\n", current->brk);
52: #endif /*__DEBUG__ */
53:         return current->brk;
54: }
```

kernel/syscalls/chdir.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/chdir.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/stat.h>
10: #include <fiwix/errno.h>
11: #include <fiwix/string.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #include <fiwix/process.h>
16: #endif /*__DEBUG__ */
17:
18: int sys_chdir(const char *dirname)
19: {
20:         struct inode *i;
21:         char *tmp_name;
22:         int errno;
23:
24: #ifdef __DEBUG__
25:         printk("(pid %d) sys_chdir('%s')\n", current->pid, dirname);
26: #endif /*__DEBUG__ */
27:
28:         if((errno = malloc_name(dirname, &tmp_name)) < 0) {
29:                 return errno;
30:         }
31:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
32:                 free_name(tmp_name);
33:                 return errno;
34:         }
35:         if(!S_ISDIR(i->i_mode)) {
36:                 iput(i);
37:                 free_name(tmp_name);
38:                 return -ENOTDIR;
39:         }
40:         if((errno = check_permission(TO_EXEC, i))) {
41:                 iput(i);
42:                 free_name(tmp_name);
43:                 return errno;
44:         }
45:         iput(current->pwd);
46:         current->pwd = i;
47:         free_name(tmp_name);
48:         return 0;
49: }
```

kernel/syscalls/chmod.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/chmod.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/kernel.h>
10: #include <fiwix/fs.h>
11: #include <fiwix/stat.h>
12: #include <fiwix/errno.h>
13: #include <fiwix/string.h>
14:
15: #ifdef __DEBUG__
16: #include <fiwix/stdio.h>
17: #include <fiwix/process.h>
18: #endif /*__DEBUG__ */
19:
20: int sys_chmod(const char *filename, __mode_t mode)
21: {
22:         struct inode *i;
23:         char *tmp_name;
24:         int errno;
25:
26: #ifdef __DEBUG__
27:         printk("(pid %d) sys_chmod('%s', %d)\n", current->pid, filename, mode);
28: #endif /*__DEBUG__ */
29:
30:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
31:                 return errno;
32:         }
33:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
34:                 free_name(tmp_name);
35:                 return errno;
36:         }
37:
38:         if(IS_RDONLY_FS(i)) {
39:                 iput(i);
40:                 free_name(tmp_name);
41:                 return -EROFS;
42:         }
43:         if(check_user_permission(i)) {
44:                 iput(i);
45:                 free_name(tmp_name);
46:                 return -EPERM;
47:         }
48:
49:         i->i_mode &= S_IFMT;
50:         i->i_mode |= mode & ~S_IFMT;
51:         i->i_ctime = CURRENT_TIME;
52:         i->dirty = 1;
53:         iput(i);
54:         free_name(tmp_name);
55:         return 0;
56: }
```

kernel/syscalls/chown.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/chown.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/stat.h>
   12: #include <fiwix/errno.h>
   13: #include <fiwix/string.h>
   14:
   15: #ifdef __DEBUG__
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/process.h>
   18: #endif /*__DEBUG__ */
   19:
   20: int sys_chown(const char *filename, __uid_t owner, __gid_t group)
   21: {
   22:         struct inode *i;
   23:         char *tmp_name;
   24:         int errno;
   25:
   26: #ifdef __DEBUG__
   27:         printk("(pid %d) sys_chown('%s', %d, %d)\n", current->pid, filename, own
er, group);
   28: #endif /*__DEBUG__ */
   29:
   30:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   31:                 return errno;
   32:         }
   33:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
   34:                 free_name(tmp_name);
   35:                 return errno;
   36:         }
   37:
   38:         if(IS_RDONLY_FS(i)) {
   39:                 iput(i);
   40:                 free_name(tmp_name);
   41:                 return -EROFS;
   42:         }
   43:         if(check_user_permission(i)) {
   44:                 iput(i);
   45:                 free_name(tmp_name);
   46:                 return -EPERM;
   47:         }
   48:
   49:         if(owner == (__uid_t)-1) {
   50:                 owner = i->i_uid;
   51:         } else {
   52:                 i->i_mode &= ~(S_ISUID);
   53:                 i->i_ctime = CURRENT_TIME;
   54:         }
   55:         if(group == (__gid_t)-1) {
   56:                 group = i->i_gid;
   57:         } else {
   58:                 i->i_mode &= ~(S_ISGID);
   59:                 i->i_ctime = CURRENT_TIME;
   60:         }
   61:
   62:         i->i_uid = owner;
   63:         i->i_gid = group;
   64:         i->dirty = 1;
   65:         iput(i);
   66:         free_name(tmp_name);
```

kernel/syscalls/chown.c


```
67:         return 0;
68: }
```

kernel/syscalls/chroot.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/chroot.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/stat.h>
10: #include <fiwix/errno.h>
11: #include <fiwix/string.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #include <fiwix/process.h>
16: #endif /*__DEBUG__ */
17:
18: int sys_chroot(const char *dirname)
19: {
20:         struct inode *i;
21:         char *tmp_name;
22:         int errno;
23:
24: #ifdef __DEBUG__
25:         printk("(pid %d) sys_chroot('%s')\n", current->pid, dirname);
26: #endif /*__DEBUG__ */
27:
28:         if((errno = malloc_name(dirname, &tmp_name)) < 0) {
29:                 return errno;
30:         }
31:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
32:                 free_name(tmp_name);
33:                 return errno;
34:         }
35:         if(!S_ISDIR(i->i_mode)) {
36:                 iput(i);
37:                 free_name(tmp_name);
38:                 return -ENOTDIR;
39:         }
40:         iput(current->root);
41:         current->root = i;
42:         free_name(tmp_name);
43:         return 0;
44: }
```

kernel/syscalls/close.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/close.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/syscalls.h>
    9: #include <fiwix/locks.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/stdio.h>
   12:
   13: int sys_close(unsigned int ufd)
   14: {
   15:         unsigned int fd;
   16:         struct inode *i;
   17:
   18: #ifdef __DEBUG__
   19:         printk("(pid %d) sys_close(%d)\n", current->pid, ufd);
   20: #endif /*__DEBUG__ */
   21:
   22:         CHECK_UFD(ufd);
   23:         fd = current->fd[ufd];
   24:         release_user_fd(ufd);
   25:
   26:         if(--fd_table[fd].count) {
   27:                 return 0;
   28:         }
   29:         i = fd_table[fd].inode;
   30:         flock_release_inode(i);
   31:         if(i->fsop && i->fsop->close) {
   32:                 i->fsop->close(i, &fd_table[fd]);
   33:                 release_fd(fd);
   34:                 iput(i);
   35:                 return 0;
   36:         }
   37:         printk("WARNING: %s(): ufd %d without the close() method!\n", __FUNCTION
__, ufd);
   38:         return -EINVAL;
   39: }
```

kernel/syscalls/creat.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/creat.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/syscalls.h>
10: #include <fiwix/fcntl.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #include <fiwix/process.h>
15: #endif /*__DEBUG__ */
16:
17: int sys_creat(const char *filename, __mode_t mode)
18: {
19: #ifdef __DEBUG__
20:         printk("(pid %d) sys_creat('%s', %d)\n", current->pid, filename, mode);
21: #endif /*__DEBUG__ */
22:         return sys_open(filename, O_CREAT | O_WRONLY | O_TRUNC, mode);
23: }
```

kernel/syscalls/dup2.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/dup2.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/syscalls.h>
 9: #include <fiwix/process.h>
10: #include <fiwix/errno.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #endif /*__DEBUG__ */
15:
16: int sys_dup2(int old_ufd, int new_ufd)
17: {
18: #ifdef __DEBUG__
19:         printk("(pid %d) sys_dup2(%d, %d)", current->pid, old_ufd, new_ufd);
20: #endif /*__DEBUG__ */
21:
22:         CHECK_UFD(old_ufd);
23:         if(new_ufd < 0 || new_ufd > OPEN_MAX) {
24:                 return -EINVAL;
25:         }
26:         if(old_ufd == new_ufd) {
27:                 return new_ufd;
28:         }
29:         if(current->fd[new_ufd]) {
30:                 sys_close(new_ufd);
31:         }
32:         if((new_ufd = get_new_user_fd(new_ufd)) < 0) {
33:                 return new_ufd;
34:         }
35:         current->fd[new_ufd] = current->fd[old_ufd];
36:         fd_table[current->fd[new_ufd]].count++;
37: #ifdef __DEBUG__
38:         printk(" --> returning %d\n", new_ufd);
39: #endif /*__DEBUG__ */
40:         return new_ufd;
41: }
```

kernel/syscalls/dup.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/dup.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/syscalls.h>
10: #include <fiwix/errno.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #include <fiwix/process.h>
15: #endif /*__DEBUG__ */
16:
17: int sys_dup(unsigned int ufd)
18: {
19:         int new_ufd;
20:
21: #ifdef __DEBUG__
22:         printk("(pid %d) sys_dup(%d)", current->pid, ufd);
23: #endif /*__DEBUG__ */
24:
25:         CHECK_UFD(ufd);
26:         if((new_ufd = get_new_user_fd(0)) < 0) {
27:                 return new_ufd;
28:         }
29:
30: #ifdef __DEBUG__
31:         printk(" -> %d\n", new_ufd);
32: #endif /*__DEBUG__ */
33:
34:         current->fd[new_ufd] = current->fd[ufd];
35:         fd_table[current->fd[new_ufd]].count++;
36:         return new_ufd;
37: }
```

kernel/syscalls/execve.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/execve.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/syscalls.h>
    9: #include <fiwix/stat.h>
   10: #include <fiwix/buffer.h>
   11: #include <fiwix/mm.h>
   12: #include <fiwix/process.h>
   13: #include <fiwix/fcntl.h>
   14: #include <fiwix/errno.h>
   15: #include <fiwix/string.h>
   16:
   17: #ifdef __DEBUG__
   18: #include <fiwix/stdio.h>
   19: #endif /*__DEBUG__ */
   20:
   21: static int initialize_barg(struct binargs *barg, char *argv[], char *envp[])
   22: {
   23:         int n, errno;
   24:
   25:         for(n = 0; n < ARG_MAX; n++) {
   26:                 barg->page[n] = 0;
   27:         }
   28:         barg->argv_len = barg->envp_len = 0;
   29:
   30:         for(n = 0; argv[n]; n++) {
   31:                 if((errno = check_user_area(VERIFY_READ, argv[n], sizeof(char *)
))) {
   32:                         return errno;
   33:                 }
   34:                 barg->argv_len += strlen(argv[n]) + 1;
   35:         }
   36:         barg->argc = n;
   37:
   38:         for(n = 0; envp[n]; n++) {
   39:                 if((errno = check_user_area(VERIFY_READ, envp[n], sizeof(char *)
))) {
   40:                         return errno;
   41:                 }
   42:                 barg->envp_len += strlen(envp[n]) + 1;
   43:         }
   44:         barg->envc = n;
   45:
   46:         return 0;
   47: }
   48:
   49: static void free_barg_pages(struct binargs *barg)
   50: {
   51:         int n;
   52:
   53:         for(n = 0; n < ARG_MAX; n++) {
   54:                 if(barg->page[n]) {
   55:                         kfree(barg->page[n]);
   56:                 }
   57:         }
   58: }
   59:
   60: static int add_strings(struct binargs *barg, char *filename, char *interpreter,
char *args)
   61: {
   62:         int n, p, offset;
   63:         unsigned int ae_str_len;
   64:         char *page;
```

kernel/syscalls/execve.c


```
 65:
 66:         /*
 67:          * For a script we need to substitute the saved argv[0] by the original
 68:          * 'filename' supplied in execve(), otherwise the interpreter won't be
 69:          * able to find the script file.
 70:          */
 71:         offset = barg->offset;
 72:         p = ARG_MAX - 1;
 73:         ae_str_len = barg->argv_len + barg->envp_len + 4;
 74:         p -= ae_str_len / PAGE_SIZE;
 75:         offset = PAGE_SIZE - (ae_str_len % PAGE_SIZE);
 76:         if(offset == PAGE_SIZE) {
 77:                 offset = 0;
 78:                 p++;
 79:         }
 80:         page = (char *)barg->page[p];
 81:         while(*(page + offset)) {
 82:                 offset++;
 83:                 barg->argv_len--;
 84:                 if(offset == PAGE_SIZE) {
 85:                         p++;
 86:                         offset = 0;
 87:                         page = (char *)barg->page[p];
 88:                 }
 89:         }
 90:         barg->argv_len--;
 91:
 92:
 93:         p = ARG_MAX - 1;
 94:         barg->argv_len += strlen(interpreter) + 1;
 95:         barg->argv_len += strlen(args) ? strlen(args) + 1 : 0;
 96:         barg->argv_len += strlen(filename) + 1;
 97:         barg->argc++;
 98:         if(*args) {
 99:                 barg->argc++;
100:         }
101:         ae_str_len = barg->argv_len + barg->envp_len + 4;
102:         p -= ae_str_len / PAGE_SIZE;
103:         offset = PAGE_SIZE - (ae_str_len % PAGE_SIZE);
104:         if(offset == PAGE_SIZE) {
105:                 offset = 0;
106:                 p++;
107:         }
108:         barg->offset = offset;
109:         for(n = p; n < ARG_MAX; n++) {
110:                 if(!barg->page[n]) {
111:                         if(!(barg->page[n] = kmalloc())) {
112:                                 free_barg_pages(barg);
113:                                 return -ENOMEM;
114:                         }
115:                 }
116:         }
117:
118:         /* interpreter */
119:         page = (char *)barg->page[p];
120:         while(*interpreter) {
121:                 *(page + offset) = *interpreter;
122:                 offset++;
123:                 interpreter++;
124:                 if(offset == PAGE_SIZE) {
125:                         p++;
126:                         offset = 0;
127:                         page = (char *)barg->page[p];
128:                 }
129:         }
130:         *(page + offset++) = NULL;
131:         if(offset == PAGE_SIZE) {
```

kernel/syscalls/execve.c


```
132:                 p++;
133:                 offset = 0;
134:         }
135:
136:         /* args */
137:         page = (char *)barg->page[p];
138:         if(*args) {
139:                 while(*args) {
140:                         *(page + offset) = *args;
141:                         offset++;
142:                         args++;
143:                         if(offset == PAGE_SIZE) {
144:                                 p++;
145:                                 offset = 0;
146:                                 page = (char *)barg->page[p];
147:                         }
148:                 }
149:                 *(page + offset++) = NULL;
150:                 if(offset == PAGE_SIZE) {
151:                         p++;
152:                         offset = 0;
153:                 }
154:         }
155:
156:         /* original script ('filename' with path) at argv[0] */
157:         page = (char *)barg->page[p];
158:         while(*filename) {
159:                 *(page + offset) = *filename;
160:                 offset++;
161:                 filename++;
162:                 if(offset == PAGE_SIZE) {
163:                         p++;
164:                         offset = 0;
165:                         page = (char *)barg->page[p];
166:                 }
167:         }
168:         *(page + offset) = NULL;
169:
170:         return 0;
171: }
172:
173: static int copy_strings(struct binargs *barg, char *argv[], char *envp[])
174: {
175:         int n, p, offset;
176:         unsigned int ae_str_len;
177:         char *page, *str;
178:
179:         p = ARG_MAX - 1;
180:         ae_str_len = barg->argv_len + barg->envp_len + 4;
181:         p -= ae_str_len / PAGE_SIZE;
182:         offset = PAGE_SIZE - (ae_str_len % PAGE_SIZE);
183:         if(offset == PAGE_SIZE) {
184:                 offset = 0;
185:                 p++;
186:         }
187:         barg->offset = offset;
188:         for(n = p; n < ARG_MAX; n++) {
189:                 if(!(barg->page[n] = kmalloc())) {
190:                         free_barg_pages(barg);
191:                         return -ENOMEM;
192:                 }
193:         }
194:         for(n = 0; n < barg->argc; n++) {
195:                 str = argv[n];
196:                 page = (char *)barg->page[p];
197:                 while(*str) {
198:                         *(page + offset) = *str;
```

kernel/syscalls/execve.c


```
  199:                          offset++;
  200:                          str++;
  201:                          if(offset == PAGE_SIZE) {
  202:                                  p++;
  203:                                  offset = 0;
  204:                                  page = (char *)barg->page[p];
  205:                          }
  206:                  }
  207:                  *(page + offset++) = NULL;
  208:                  if(offset == PAGE_SIZE) {
  209:                          p++;
  210:                          offset = 0;
  211:                  }
  212:          }
  213:          for(n = 0; n < barg->envc; n++) {
  214:                  str = envp[n];
  215:                  page = (char *)barg->page[p];
  216:                  while(*str) {
  217:                          *(page + offset) = *str;
  218:                          offset++;
  219:                          str++;
  220:                          if(offset == PAGE_SIZE) {
  221:                                  p++;
  222:                                  offset = 0;
  223:                                  page = (char *)barg->page[p];
  224:                          }
  225:                  }
  226:                  *(page + offset++) = NULL;
  227:                  if(offset == PAGE_SIZE) {
  228:                          p++;
  229:                          offset = 0;
  230:                  }
  231:          }
  232:
  233:          return 0;
  234: }
  235:
  236: static int do_execve(const char *filename, char *argv[], char *envp[], struct si
gcontext *sc)
  237: {
  238:          char interpreter[NAME_MAX + 1], args[NAME_MAX + 1], name[NAME_MAX + 1];
  239:          __blk_t block;
  240:          struct buffer *buf;
  241:          struct inode *i;
  242:          struct binargs barg;
  243:          char *data, *tmp_name;
  244:          int errno;
  245:
  246:          if((errno = initialize_barg(&barg, &(*argv), &(*envp))) < 0) {
  247:                  return errno;
  248:          }
  249:
  250:          /* save 'argv' and 'envp' into the kernel address space */
  251:          if((errno = copy_strings(&barg, &(*argv), &(*envp)))) {
  252:                  return errno;
  253:          }
  254:
  255:          if(!(data = (void *)kmalloc())) {
  256:                  return -ENOMEM;
  257:          }
  258:
  259:          if((errno = malloc_name(filename, &tmp_name)) < 0) {
  260:                  kfree((unsigned int)data);
  261:                  free_barg_pages(&barg);
  262:                  return errno;
  263:          }
  264:          strcpy(name, tmp_name);
```

kernel/syscalls/execve.c


```
  265:          free_name(tmp_name);
  266:
  267:
  268: loop:
  269:          if((errno = namei(name, &i, NULL, FOLLOW_LINKS))) {
  270:                  free_barg_pages(&barg);
  271:                  kfree((unsigned int)data);
  272:                  return errno;
  273:          }
  274:
  275:          if(!S_ISREG(i->i_mode)) {
  276:                  iput(i);
  277:                  free_barg_pages(&barg);
  278:                  kfree((unsigned int)data);
  279:                  return -EACCES;
  280:          }
  281:          if(check_permission(TO_EXEC, i) < 0) {
  282:                  iput(i);
  283:                  free_barg_pages(&barg);
  284:                  kfree((unsigned int)data);
  285:                  return -EACCES;
  286:          }
  287:
  288:          if((block = bmap(i, 0, FOR_READING)) < 0) {
  289:                  iput(i);
  290:                  free_barg_pages(&barg);
  291:                  kfree((unsigned int)data);
  292:                  return block;
  293:          }
  294:          if(!(buf = bread(i->dev, block, i->sb->s_blocksize))) {
  295:                  iput(i);
  296:                  free_barg_pages(&barg);
  297:                  kfree((unsigned int)data);
  298:                  return -EIO;
  299:          }
  300:
  301:          /*
  302:           * The contents of the buffer is copied and then freed immediately to
  303:           * make sure that it won't conflict while zeroing the BSS fractional
  304:           * page, in case that the same block is requested during the page fault.
  305:           */
  306:          memcpy_b(data, buf->data, i->sb->s_blocksize);
  307:          brelse(buf);
  308:
  309:          errno = elf_load(i, &barg, sc, data);
  310:          if(errno == -ENOEXEC) {
  311:                  /* OK, looks like it was not an ELF binary; let's see if it is a
 script */
  312:                  memset_b(interpreter, 0, NAME_MAX + 1);
  313:                  memset_b(args, 0, NAME_MAX + 1);
  314:                  errno = script_load(interpreter, args, data);
  315:                  if(!errno) {
  316:                          /* yes, it is! */
  317:                          iput(i);
  318:                          if((errno = add_strings(&barg, name, interpreter, args))
) {
  319:                                  free_barg_pages(&barg);
  320:                                  kfree((unsigned int)data);
  321:                                  return errno;
  322:                          }
  323:                          strcpy(name, interpreter);
  324:                          goto loop;
  325:                  }
  326:          }
  327:
  328:          if(!errno) {
  329:                  if(i->i_mode & S_ISUID) {
```

kernel/syscalls/execve.c


```
  330:                          current->euid = i->i_uid;
  331:                  }
  332:                  if(i->i_mode & S_ISGID) {
  333:                          current->egid = i->i_gid;
  334:                  }
  335:          }
  336:
  337:          iput(i);
  338:          free_barg_pages(&barg);
  339:          kfree((unsigned int)data);
  340:          return errno;
  341: }
  342:
  343: int sys_execve(const char *filename, char *argv[], char *envp[], int arg4, int a
rg5, struct sigcontext *sc)
  344: {
  345:          char argv0[NAME_MAX + 1];
  346:          int n, errno;
  347:
  348: #ifdef __DEBUG__
  349:          printk("(pid %d) sys_execve('%s', ...)\n", current->pid, filename);
  350: #endif /*__DEBUG__ */
  351:
  352:          /* copy filename into kernel address space */
  353:          strncpy(argv0, argv[0], NAME_MAX);
  354:          if((errno = do_execve(filename, &(*argv), &(*envp), sc))) {
  355:                  return errno;
  356:          }
  357:
  358:          strncpy(current->argv0, argv0, NAME_MAX);
  359:          for(n = 0; n < OPEN_MAX; n++) {
  360:                  if(current->fd[n] && (current->fd_flags[n] & FD_CLOEXEC)) {
  361:                          sys_close(n);
  362:                  }
  363:          }
  364:
  365:          current->suid = current->euid;
  366:          current->sgid = current->egid;
  367:          current->sigpending = 0;
  368:          current->sigexecuting = 0;
  369:          for(n = 0; n < NSIG; n++) {
  370:                  current->sigaction[n].sa_mask = 0;
  371:                  current->sigaction[n].sa_flags = 0;
  372:                  if(current->sigaction[n].sa_handler != SIG_IGN) {
  373:                          current->sigaction[n].sa_handler = SIG_DFL;
  374:                  }
  375:          }
  376:          current->sleep_address = NULL;
  377:          current->flags |= PF_PEXEC;
  378:          return 0;
  379: }
```

kernel/syscalls/exit.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/exit.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/asm.h>
 9: #include <fiwix/kernel.h>
10: #include <fiwix/syscalls.h>
11: #include <fiwix/process.h>
12: #include <fiwix/sched.h>
13: #include <fiwix/mman.h>
14: #include <fiwix/sleep.h>
15: #include <fiwix/stdio.h>
16: #include <fiwix/string.h>
17:
18: void do_exit(int exit_code)
19: {
20:         int n;
21:         struct proc *p, *init;
22:
23: #ifdef __DEBUG__
24:         printk("\n");
25:         printk("sys_exit(pid %d, ppid %d)\n", current->pid, current->ppid);
26:         printk("------------------------------\n");
27: #endif /*__DEBUG__ */
28:
29:         CLI();
30:         current->state = PROC_ZOMBIE;
31:
32:         release_binary();
33:         current->argv = NULL;
34:         current->envp = NULL;
35:         current->sigpending = 0;
36:         current->sigblocked = 0;
37:         current->sigexecuting = 0;
38:         for(n = 0; n < NSIG; n++) {
39:                 current->sigaction[n].sa_mask = 0;
40:                 current->sigaction[n].sa_flags = 0;
41:                 current->sigaction[n].sa_handler = SIG_IGN;
42:         }
43:
44:         init = get_proc_by_pid(INIT);
45:         FOR_EACH_PROCESS(p) {
46:                 if(SESS_LEADER(current)) {
47:                         if(p->sid == current->sid && p->state != PROC_ZOMBIE) {
48:                                 p->pgid = 0;
49:                                 p->sid = 0;
50:                                 p->ctty = NULL;
51:                                 send_sig(p, SIGHUP);
52:                                 send_sig(p, SIGCONT);
53:                         }
54:                 }
55:
56:                 /* make INIT inherit the children of this exiting process */
57:                 if(p->state && p->ppid == current->pid) {
58:                         p->ppid = INIT;
59:                         init->children++;
60:                         if(p->state == PROC_ZOMBIE) {
61:                                 send_sig(init, SIGCHLD);
62:                         }
63:                 }
64:         }
65:
66:         if(SESS_LEADER(current)) {
67:                 disassociate_ctty(current->ctty);
```

kernel/syscalls/exit.c


```
   68:         }
   69:
   70:         for(n = 0; n < OPEN_MAX; n++) {
   71:                 if(current->fd[n]) {
   72:                         sys_close(n);
   73:                 }
   74:         }
   75:
   76:         iput(current->root);
   77:         current->root = NULL;
   78:         iput(current->pwd);
   79:         current->pwd = NULL;
   80:         current->exit_code = exit_code;
   81:         if(!--nr_processes) {
   82:                 printk("\n");
   83:                 printk("WARNING: the last user process has exited. The kernel wi
ll stop itself.\n");
   84:                 stop_kernel();
   85:         }
   86:
   87:         /* notify the parent about the child's death */
   88:         if((p = get_proc_by_pid(current->ppid))) {
   89:                 send_sig(p, SIGCHLD);
   90:                 if(p->sleep_address == &sys_wait4) {
   91:                         wakeup_proc(p);
   92:                 }
   93:         }
   94:
   95:         need_resched = 1;
   96:
   97:         /* make sure to recover if the process returns from the death (!?) */
   98:         for(;;) {
   99:                 current->state = PROC_ZOMBIE;
  100:                 do_sched();
  101:         }
  102: }
  103:
  104: int sys_exit(int exit_code)
  105: {
  106: #ifdef __DEBUG__
  107:         printk("(pid %d) sys_exit()\n", current->pid);
  108: #endif /*__DEBUG__ */
  109:
  110:         /* exit code in the second byte.
  111:          *  15               8 7               0
  112:          * +-------------------+-------------------+
  113:          * | exit code (0-255) |         0         |
  114:          * +-------------------+-------------------+
  115:          */
  116:         do_exit((exit_code & 0xFF) << 8);
  117:         return 0;
  118: }
```

kernel/syscalls/fchdir.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/fchdir.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/process.h>
10: #include <fiwix/stat.h>
11: #include <fiwix/errno.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #endif /*__DEBUG__ */
16:
17: int sys_fchdir(unsigned int ufd)
18: {
19:         struct inode *i;
20:
21: #ifdef __DEBUG__
22:         printk("(pid %d) sys_fchdir(%d)\n", current->pid, ufd);
23: #endif /*__DEBUG__ */
24:
25:         CHECK_UFD(ufd);
26:         i = fd_table[current->fd[ufd]].inode;
27:         if(!S_ISDIR(i->i_mode)) {
28:                 return -ENOTDIR;
29:         }
30:         iput(current->pwd);
31:         current->pwd = i;
32:         current->pwd->count++;
33:         return 0;
34: }
```

kernel/syscalls/fchmod.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/fchmod.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/kernel.h>
 9: #include <fiwix/types.h>
10: #include <fiwix/fs.h>
11: #include <fiwix/stat.h>
12: #include <fiwix/errno.h>
13:
14: #ifdef __DEBUG__
15: #include <fiwix/stdio.h>
16: #include <fiwix/process.h>
17: #endif /*__DEBUG__ */
18:
19: int sys_fchmod(int ufd, __mode_t mode)
20: {
21:         struct inode *i;
22:
23: #ifdef __DEBUG__
24:         printk("(pid %d) sys_fchmod(%d, %d)\n", current->pid, ufd, mode);
25: #endif /*__DEBUG__ */
26:
27:         CHECK_UFD(ufd);
28:         i = fd_table[current->fd[ufd]].inode;
29:
30:         if(IS_RDONLY_FS(i)) {
31:                 return -EROFS;
32:         }
33:         if(check_user_permission(i)) {
34:                 return -EPERM;
35:         }
36:
37:         i->i_mode &= S_IFMT;
38:         i->i_mode |= mode & ~S_IFMT;
39:         i->i_ctime = CURRENT_TIME;
40:         i->dirty = 1;
41:         return 0;
42: }
```

kernel/syscalls/fchown.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/fchown.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/types.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/stat.h>
   12: #include <fiwix/process.h>
   13: #include <fiwix/errno.h>
   14:
   15: #ifdef __DEBUG__
   16: #include <fiwix/stdio.h>
   17: #endif /*__DEBUG__ */
   18:
   19: int sys_fchown(int ufd, __uid_t owner, __gid_t group)
   20: {
   21:         struct inode *i;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_fchown(%d, %d, %d)\n", current->pid, ufd, owner, gr
oup);
   25: #endif /*__DEBUG__ */
   26:
   27:         CHECK_UFD(ufd);
   28:         i = fd_table[current->fd[ufd]].inode;
   29:
   30:         if(IS_RDONLY_FS(i)) {
   31:                 return -EROFS;
   32:         }
   33:         if(check_user_permission(i)) {
   34:                 return -EPERM;
   35:         }
   36:
   37:         if(owner == (__uid_t)-1) {
   38:                 owner = i->i_uid;
   39:         } else {
   40:                 i->i_mode &= ~(S_ISUID);
   41:         }
   42:         if(group == (__gid_t)-1) {
   43:                 group = i->i_gid;
   44:         } else {
   45:                 i->i_mode &= ~(S_ISGID);
   46:         }
   47:
   48:         i->i_uid = owner;
   49:         i->i_gid = group;
   50:         i->i_ctime = CURRENT_TIME;
   51:         i->dirty = 1;
   52:         return 0;
   53: }
```

kernel/syscalls/fcntl.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/fcntl.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/syscalls.h>
    9: #include <fiwix/fcntl.h>
   10: #include <fiwix/locks.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_fcntl(int ufd, int cmd, unsigned long int arg)
   19: {
   20:         int new_ufd, errno;
   21:
   22: #ifdef __DEBUG__
   23:         printk("(pid %d) sys_fcntl(%d, %d, 0x%08x)\n", current->pid, ufd, cmd, a
rg);
   24: #endif /*__DEBUG__ */
   25:
   26:         CHECK_UFD(ufd);
   27:         switch(cmd) {
   28:                 case F_DUPFD:
   29:                         CHECK_UFD(ufd);
   30:                         if(arg >= OPEN_MAX) {
   31:                                 return -EINVAL;
   32:                         }
   33:                         if((new_ufd = get_new_user_fd(arg)) < 0) {
   34:                                 return new_ufd;
   35:                         }
   36:                         current->fd[new_ufd] = current->fd[ufd];
   37:                         fd_table[current->fd[new_ufd]].count++;
   38: #ifdef __DEBUG__
   39:                         printk("\t--> returning %d\n", new_ufd);
   40: #endif /*__DEBUG__ */
   41:                         return new_ufd;
   42:                 case F_GETFD:
   43:                         return (current->fd_flags[ufd] & FD_CLOEXEC);
   44:                 case F_SETFD:
   45:                         current->fd_flags[ufd] = (arg & FD_CLOEXEC);
   46:                         break;
   47:                 case F_GETFL:
   48:                         return fd_table[current->fd[ufd]].flags;
   49:                 case F_SETFL:
   50:                         fd_table[current->fd[ufd]].flags &= ~(O_APPEND | O_NONBL
OCK);
   51:                         fd_table[current->fd[ufd]].flags |= arg & (O_APPEND | O_
NONBLOCK);
   52:                         break;
   53:                 case F_GETLK:
   54:                 case F_SETLK:
   55:                 case F_SETLKW:
   56:                         if((errno = check_user_area(VERIFY_READ, (void *)arg, si
zeof(struct flock)))) {
   57:                                 return errno;
   58:                         }
   59:                         return posix_lock(ufd, cmd, (struct flock *)arg);
   60:                 default:
   61:                         return -EINVAL;
   62:         }
   63:         return 0;
```

kernel/syscalls/fcntl.c


```
64: }
```

kernel/syscalls/fdatasync.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/fdatasync.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/syscalls.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #include <fiwix/process.h>
13: #endif /*__DEBUG__ */
14:
15: int sys_fdatasync(int ufd)
16: {
17: #ifdef __DEBUG__
18:         printk("(pid %d) sys_fdatasync(%d)\n", current->pid, ufd);
19: #endif /*__DEBUG__ */
20:
21:         return sys_fsync(ufd);
22: }
```

kernel/syscalls/flock.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/flock.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/process.h>
10: #include <fiwix/locks.h>
11: #include <fiwix/errno.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #endif /*__DEBUG__ */
16:
17: int sys_flock(int ufd, int op)
18: {
19:         struct inode *i;
20:
21: #ifdef __DEBUG__
22:         printk("(pid %d) sys_flock(%d, %d)\n", current->pid, ufd, op);
23: #endif /*__DEBUG__ */
24:
25:         CHECK_UFD(ufd);
26:         i = fd_table[current->fd[ufd]].inode;
27:         return flock_inode(i, op);
28: }
```

kernel/syscalls/fork.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/fork.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/types.h>
   11: #include <fiwix/const.h>
   12: #include <fiwix/sigcontext.h>
   13: #include <fiwix/process.h>
   14: #include <fiwix/sched.h>
   15: #include <fiwix/mm.h>
   16: #include <fiwix/errno.h>
   17: #include <fiwix/stdio.h>
   18: #include <fiwix/string.h>
   19:
   20: int sys_fork(int arg1, int arg2, int arg3, int arg4, int arg5, struct sigcontext
*sc)
   21: {
   22:         int count, pages;
   23:         unsigned int n;
   24:         unsigned int *child_pgdir;
   25:         struct sigcontext *stack;
   26:         struct proc *child, *p;
   27:         struct vma *vma;
   28:         __pid_t pid;
   29:
   30: #ifdef __DEBUG__
   31:         printk("(pid %d) sys_fork()\n", current->pid);
   32: #endif /*__DEBUG__ */
   33:
   34:         /* check the number of processes already allocated by this UID */
   35:         count = 0;
   36:         FOR_EACH_PROCESS(p) {
   37:                 if(p->state && p->uid == current->uid) {
   38:                         count++;
   39:                 }
   40:         }
   41:         if(count > current->rlim[RLIMIT_NPROC].rlim_cur) {
   42:                 printk("WARNING: %s(): RLIMIT_NPROC exceeded.\n", __FUNCTION__);
   43:                 return -EAGAIN;
   44:         }
   45:
   46:         if(!(pid = get_unused_pid())) {
   47:                 return -EAGAIN;
   48:         }
   49:         if(!(child = get_proc_free())) {
   50:                 return -EAGAIN;
   51:         }
   52:
   53:         /*
   54:          * This memcpy() will overwrite the prev and next pointers, so that's
   55:          * the reason why proc_slot_init() is separated from get_proc_free().
   56:          */
   57:         memcpy_b(child, current, sizeof(struct proc));
   58:
   59:         proc_slot_init(child);
   60:         child->pid = pid;
   61:         memset_b(&child->tss, NULL, sizeof(struct i386tss));
   62:         sprintk(child->pidstr, "%d", child->pid);
   63:         child->state = PROC_IDLE;
   64:
   65:         if(!(child_pgdir = (void *)kmalloc())) {
   66:                 release_proc(child);
```

kernel/syscalls/fork.c


```
   67:                 return -ENOMEM;
   68:         }
   69:         child->rss++;
   70:         memcpy_b(child_pgdir, kpage_dir, PAGE_SIZE);
   71:         child->tss.cr3 = V2P((unsigned int)child_pgdir);
   72:
   73:         child->ppid = current->pid;
   74:         child->flags = 0;
   75:         child->children = 0;
   76:         child->cpu_count = child->priority;
   77:         child->start_time = CURRENT_TICKS;
   78:         child->sleep_address = NULL;
   79:
   80:         memcpy_b(child->vma, current->vma, sizeof(child->vma));
   81:         vma = child->vma;
   82:         for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
   83:                 if(vma->inode) {
   84:                         vma->inode->count++;
   85:                 }
   86:         }
   87:
   88:         child->sigpending = 0;
   89:         child->sigexecuting = 0;
   90:         memset_b(&child->sc, NULL, sizeof(struct sigcontext));
   91:         memset_b(&child->usage, NULL, sizeof(struct rusage));
   92:         memset_b(&child->cusage, NULL, sizeof(struct rusage));
   93:         child->it_real_interval = 0;
   94:         child->it_real_value = 0;
   95:         child->it_virt_interval = 0;
   96:         child->it_virt_value = 0;
   97:         child->it_prof_interval = 0;
   98:         child->it_prof_value = 0;
   99:
  100:         if(!(child->tss.esp0 = kmalloc())) {
  101:                 kfree((unsigned int)child_pgdir);
  102:                 kfree((unsigned int)child->vma);
  103:                 release_proc(child);
  104:                 return -ENOMEM;
  105:         }
  106:
  107:         if(!(pages = clone_pages(child))) {
  108:                 printk("WARNING: %s(): not enough memory, can't clone pages.\n",
 __FUNCTION__);
  109:                 free_page_tables(child);
  110:                 kfree((unsigned int)child_pgdir);
  111:                 kfree((unsigned int)child->vma);
  112:                 release_proc(child);
  113:                 return -ENOMEM;
  114:         }
  115:         child->rss += pages;
  116:         invalidate_tlb();
  117:
  118:         child->tss.esp0 += PAGE_SIZE - 4;
  119:         child->rss++;
  120:         child->tss.ss0 = KERNEL_DS;
  121:
  122:         memcpy_b((unsigned int *)(child->tss.esp0 & PAGE_MASK), (void *)((unsign
ed int)(sc) & PAGE_MASK), PAGE_SIZE);
  123:         stack = (struct sigcontext *)((child->tss.esp0 & PAGE_MASK) + ((unsigned
 int)(sc) & ~PAGE_MASK));
  124:
  125:         child->tss.eip = (unsigned int)return_from_syscall;
  126:         child->tss.esp = (unsigned int)stack;
  127:         stack->eax = 0;         /* child returns 0 */
  128:
  129:         child->state = PROC_RUNNING;
  130:
```

kernel/syscalls/fork.c


```
131:         /* increase file descriptors usage */
132:         for(n = 0; n < OPEN_MAX; n++) {
133:                 if(current->fd[n]) {
134:                         fd_table[current->fd[n]].count++;
135:                 }
136:         }
137:         if(current->root) {
138:                 current->root->count++;
139:         }
140:         if(current->pwd) {
141:                 current->pwd->count++;
142:         }
143:
144:         kstat.processes++;
145:         nr_processes++;
146:         current->children++;
147:
148:         return child->pid;      /* parent returns child's PID */
149: }
```

kernel/syscalls/fstat.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/fstat.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/syscalls.h>
   10: #include <fiwix/statbuf.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_fstat(unsigned int ufd, struct old_stat *statbuf)
   19: {
   20:         struct inode *i;
   21:         int errno;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_fstat(%d, 0x%08x) -> returning structure\n", curren
t->pid, ufd, (unsigned int )statbuf);
   25: #endif /*__DEBUG__ */
   26:
   27:         CHECK_UFD(ufd);
   28:         if((errno = check_user_area(VERIFY_WRITE, statbuf, sizeof(struct old_sta
t)))) {
   29:                 return errno;
   30:         }
   31:         i = fd_table[current->fd[ufd]].inode;
   32:         statbuf->st_dev = i->dev;
   33:         statbuf->st_ino = i->inode;
   34:         statbuf->st_mode = i->i_mode;
   35:         statbuf->st_nlink = i->i_nlink;
   36:         statbuf->st_uid = i->i_uid;
   37:         statbuf->st_gid = i->i_gid;
   38:         statbuf->st_rdev = i->rdev;
   39:         statbuf->st_size = i->i_size;
   40:         statbuf->st_atime = i->i_atime;
   41:         statbuf->st_mtime = i->i_mtime;
   42:         statbuf->st_ctime = i->i_ctime;
   43:         return 0;
   44: }
```

kernel/syscalls/fstatfs.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/fstatfs.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/statfs.h>
   10: #include <fiwix/errno.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_fstatfs(unsigned int ufd, struct statfs *statfsbuf)
   18: {
   19:         struct inode *i;
   20:         int errno;
   21:
   22: #ifdef __DEBUG__
   23:         printk("(pid %d) sys_fstatfs(%d, 0x%08x)\n", current->pid, ufd, (unsigne
d int)statfsbuf);
   24: #endif /*__DEBUG__ */
   25:
   26:         CHECK_UFD(ufd);
   27:         if((errno = check_user_area(VERIFY_WRITE, statfsbuf, sizeof(struct statf
s))) {
   28:                 return errno;
   29:         }
   30:         i = fd_table[current->fd[ufd]].inode;
   31:         if(i->sb && i->sb->fsop && i->sb->fsop->statfs) {
   32:                 i->sb->fsop->statfs(i->sb, statfsbuf);
   33:                 return 0;
   34:         }
   35:         return -ENOSYS;
   36: }
```

kernel/syscalls/fsync.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/fsync.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/filesystems.h>
10: #include <fiwix/process.h>
11: #include <fiwix/stat.h>
12: #include <fiwix/buffer.h>
13: #include <fiwix/errno.h>
14:
15: #ifdef __DEBUG__
16: #include <fiwix/stdio.h>
17: #endif /*__DEBUG__ */
18:
19: int sys_fsync(int ufd)
20: {
21:         struct inode *i;
22:
23: #ifdef __DEBUG__
24:         printk("(pid %d) sys_fsync(%d)\n", current->pid, ufd);
25: #endif /*__DEBUG__ */
26:
27:         CHECK_UFD(ufd);
28:         i = fd_table[current->fd[ufd]].inode;
29:         if(!S_ISREG(i->i_mode)) {
30:                 return -EINVAL;
31:         }
32:         if(IS_RDONLY_FS(i)) {
33:                 return -EROFS;
34:         }
35:         sync_superblocks(i->dev);
36:         sync_inodes(i->dev);
37:         sync_buffers(i->dev);
38:         return 0;
39: }
```

kernel/syscalls/ftime.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/ftime.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/kernel.h>
 9: #include <fiwix/fs.h>
10: #include <fiwix/timeb.h>
11: #include <fiwix/timer.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #include <fiwix/process.h>
16: #endif /*__DEBUG__ */
17:
18: int sys_ftime(struct timeb *tp)
19: {
20:         int errno;
21:
22: #ifdef __DEBUG__
23:         printk("(pid %d) sys_ftime()\n", current->pid);
24: #endif /*__DEBUG__ */
25:
26:         if((errno = check_user_area(VERIFY_WRITE, tp, sizeof(struct timeb)))) {
27:                 return errno;
28:         }
29:         tp->time = CURRENT_TIME;
30:         tp->millitm = ((kstat.ticks % HZ) * 1000000) / HZ;
31:         /* FIXME: 'timezone' and 'dstflag' fields are not used */
32:
33:         return 0;
34: }
```

kernel/syscalls/ftruncate.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/ftruncate.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/fs.h>
10: #include <fiwix/fcntl.h>
11: #include <fiwix/stat.h>
12: #include <fiwix/errno.h>
13:
14: #ifdef __DEBUG__
15: #include <fiwix/stdio.h>
16: #include <fiwix/process.h>
17: #endif /*__DEBUG__ */
18:
19: int sys_ftruncate(int ufd, __off_t length)
20: {
21:         struct inode *i;
22:         int errno;
23:
24: #ifdef __DEBUG__
25:         printk("(pid %d) sys_ftruncate(%d, %d)\n", current->pid, ufd, length);
26: #endif /*__DEBUG__ */
27:
28:         CHECK_UFD(ufd);
29:         i = fd_table[current->fd[ufd]].inode;
30:         if((fd_table[current->fd[ufd]].flags & O_ACCMODE) == O_RDONLY) {
31:                 return -EINVAL;
32:         }
33:         if(S_ISDIR(i->i_mode)) {
34:                 return -EISDIR;
35:         }
36:         if(IS_RDONLY_FS(i)) {
37:                 return -EROFS;
38:         }
39:         if(check_permission(TO_WRITE, i) < 0) {
40:                 return -EPERM;
41:         }
42:         if(length == i->i_size) {
43:                 return 0;
44:         }
45:
46:         errno = 0;
47:         if(i->fsop && i->fsop->truncate) {
48:                 inode_lock(i);
49:                 errno = i->fsop->truncate(i, length);
50:                 inode_unlock(i);
51:         }
52:         return errno;
53: }
```

kernel/syscalls/getcwd.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/getcwd.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/errno.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_getcwd(char *buf, __size_t size)
   18: {
   19:         int errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_getcwd(0x%08x, %d)\n", current->pid, (unsigned int)
buf, size);
   23: #endif /*__DEBUG__ */
   24:
   25:         if((errno = check_user_area(VERIFY_WRITE, buf, size))) {
   26:                 return errno;
   27:         }
   28:         return -ENOSYS;
   29: }
```

kernel/syscalls/getdents.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/getdents.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/dirent.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/stat.h>
   12: #include <fiwix/errno.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_getdents(unsigned int ufd, struct dirent *dirent, unsigned int count)
   19: {
   20:         struct inode *i;
   21:         int errno;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_getdents(%d, 0x%08x, %d)", current->pid, ufd, (unsi
gned int)dirent, count);
   25: #endif /*__DEBUG__ */
   26:
   27:         CHECK_UFD(ufd);
   28:         if((errno = check_user_area(VERIFY_WRITE, dirent, sizeof(struct dirent))
)) {
   29:                 return errno;
   30:         }
   31:         i = fd_table[current->fd[ufd]].inode;
   32:
   33:         if(!S_ISDIR(i->i_mode)) {
   34:                 return -ENOTDIR;
   35:         }
   36:
   37:         if(i->fsop && i->fsop->readdir) {
   38:                 errno = i->fsop->readdir(i, &fd_table[current->fd[ufd]], dirent,
 count);
   39:         #ifdef __DEBUG__
   40:                 printk(" -> returning %d\n", errno);
   41:         #endif /*__DEBUG__ */
   42:                 return errno;
   43:         }
   44:         return -EINVAL;
   45: }
```

kernel/syscalls/getegid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/getegid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #endif /*__DEBUG__ */
13:
14: int sys_getegid(void)
15: {
16: #ifdef __DEBUG__
17:         printk("(pid %d) sys_getegid() -> %d\n", current->pid, current->egid);
18: #endif /*__DEBUG__ */
19:         return current->egid;
20: }
```

kernel/syscalls/geteuid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/geteuid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #endif /*__DEBUG__ */
13:
14: int sys_geteuid(void)
15: {
16: #ifdef __DEBUG__
17:         printk("(pid %d) sys_geteuid() -> %d\n", current->pid, current->euid);
18: #endif /*__DEBUG__ */
19:         return current->euid;
20: }
```

kernel/syscalls/getgid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/getgid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #endif /*__DEBUG__ */
13:
14: int sys_getgid(void)
15: {
16: #ifdef __DEBUG__
17:         printk("(pid %d) sys_getgid() -> %d\n", current->pid, current->gid);
18: #endif /*__DEBUG__ */
19:         return current->gid;
20: }
```

kernel/syscalls/getgroups.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/getgroups.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_getgroups(__ssize_t size, __gid_t *list)
   18: {
   19:         int n, errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_getgroups(%d, 0x%08x)\n", current->pid, size, (unsi
gned int)list);
   23: #endif /*__DEBUG__ */
   24:
   25:         /*
   26:          * If size is 0, sys_getgroups() shall return the number of group IDs
   27:          * that it would otherwise return without modifying the array pointed
   28:          * to by list.
   29:          */
   30:         if(!size) {
   31:                 for(n = 0; n < NGROUPS_MAX; n++) {
   32:                         if(current->groups[n] == -1) {
   33:                                 break;
   34:                         }
   35:                 }
   36:                 return n;
   37:         }
   38:
   39:         if((errno = check_user_area(VERIFY_WRITE, list, sizeof(__gid_t)))) {
   40:                 return errno;
   41:         }
   42:         for(n = 0; n < NGROUPS_MAX; n++) {
   43:                 if(current->groups[n] == -1) {
   44:                         break;
   45:                 }
   46:                 if(size) {
   47:                         if(n > size) {
   48:                                 return -EINVAL;
   49:                         }
   50:                         list[n] = (__gid_t)current->groups[n];
   51:                 }
   52:         }
   53:         return n;
   54: }
```

kernel/syscalls/getitimer.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/getitimer.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/time.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_getitimer(int which, struct itimerval *curr_value)
   18: {
   19:         int errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_getitimer(%d, 0x%08x) -> \n", current->pid, which,
(unsigned int)curr_value);
   23: #endif /*__DEBUG__ */
   24:
   25:         if((unsigned int)curr_value) {
   26:                 if((errno = check_user_area(VERIFY_WRITE, curr_value, sizeof(str
uct itimerval)))) {
   27:                         return errno;
   28:                 }
   29:         }
   30:
   31:         switch(which) {
   32:                 case ITIMER_REAL:
   33:                         ticks2tv(current->it_real_interval, &curr_value->it_inte
rval);
   34:                         ticks2tv(current->it_real_value, &curr_value->it_value);
   35:                         break;
   36:                 case ITIMER_VIRTUAL:
   37:                         ticks2tv(current->it_virt_interval, &curr_value->it_inte
rval);
   38:                         ticks2tv(current->it_virt_value, &curr_value->it_value);
   39:                         break;
   40:                 case ITIMER_PROF:
   41:                         ticks2tv(current->it_prof_interval, &curr_value->it_inte
rval);
   42:                         ticks2tv(current->it_prof_value, &curr_value->it_value);
   43:                         break;
   44:                 default:
   45:                         return -EINVAL;
   46:         }
   47:         return 0;
   48: }
```

kernel/syscalls/getpgid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/getpgid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/process.h>
10: #include <fiwix/sched.h>
11: #include <fiwix/errno.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #endif /*__DEBUG__ */
16:
17: int sys_getpgid(__pid_t pid)
18: {
19:         struct proc *p;
20:
21: #ifdef __DEBUG__
22:         printk("(pid %d) sys_getpgid(%d)\n", current->pid, pid);
23: #endif /*__DEBUG__ */
24:
25:         if(pid < 0) {
26:                 return -EINVAL;
27:         }
28:         if(!pid) {
29:                 return current->pgid;
30:         }
31:         FOR_EACH_PROCESS(p) {
32:                 if(p->state != PROC_UNUSED && p->pid == pid) {
33:                         return p->pgid;
34:                 }
35:         }
36:         return -ESRCH;
37: }
```

kernel/syscalls/getpgrp.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/getpgrp.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #endif /*__DEBUG__ */
13:
14: int sys_getpgrp(void)
15: {
16: #ifdef __DEBUG__
17:         printk("(pid %d) sys_getpgrp() -> %d\n", current->pid, current->pgid);
18: #endif /*__DEBUG__ */
19:         return current->pgid;
20: }
```

kernel/syscalls/getpid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/getpid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #endif /*__DEBUG__ */
13:
14: int sys_getpid(void)
15: {
16: #ifdef __DEBUG__
17:         printk("(pid %d) sys_getpid() -> %d\n", current->pid, current->pid);
18: #endif /*__DEBUG__ */
19:         return current->pid;
20: }
```

kernel/syscalls/getppid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/getppid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #endif /*__DEBUG__ */
13:
14: int sys_getppid(void)
15: {
16: #ifdef __DEBUG__
17:         printk("(pid %d) sys_getppid() -> %d\n", current->pid, current->ppid);
18: #endif /*__DEBUG__ */
19:         return current->ppid;
20: }
```

kernel/syscalls/getrlimit.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/getrlimit.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/resource.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_getrlimit(int resource, struct rlimit *rlim)
   18: {
   19:         int errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_getrlimit(%d, 0x%08x)\n", current->pid, resource, (
unsigned int)rlim);
   23: #endif /*__DEBUG__ */
   24:
   25:         if((errno = check_user_area(VERIFY_WRITE, rlim, sizeof(struct rlimit))))
 {
   26:                 return errno;
   27:         }
   28:         if(resource < 0 || resource >= RLIM_NLIMITS) {
   29:                 return -EINVAL;
   30:         }
   31:
   32:         rlim->rlim_cur = current->rlim[resource].rlim_cur;
   33:         rlim->rlim_max = current->rlim[resource].rlim_max;
   34:         return 0;
   35: }
```

kernel/syscalls/getrusage.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/getrusage.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/resource.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12: #include <fiwix/string.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_getrusage(int who, struct rusage *usage)
   19: {
   20:         int errno;
   21:
   22: #ifdef __DEBUG__
   23:         printk("(pid %d) sys_getrusage(%d, 0x%08x)\n", current->pid, who, (unsig
ned int)usage);
   24: #endif /*__DEBUG__ */
   25:
   26:         if((errno = check_user_area(VERIFY_WRITE, usage, sizeof(struct rusage)))
) {
   27:                 return errno;
   28:         }
   29:         switch(who) {
   30:                 case RUSAGE_SELF:
   31:                         memcpy_b(usage, &current->usage, sizeof(struct rusage));
   32:                         break;
   33:                 case RUSAGE_CHILDREN:
   34:                         memcpy_b(usage, &current->cusage, sizeof(struct rusage))
;
   35:                         break;
   36:                 default:
   37:                         return -EINVAL;
   38:         }
   39:         return 0;
   40: }
```

kernel/syscalls/getsid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/getsid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/process.h>
10: #include <fiwix/sched.h>
11: #include <fiwix/errno.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #endif /*__DEBUG__ */
16:
17: int sys_getsid(__pid_t pid)
18: {
19:         struct proc *p;
20:
21: #ifdef __DEBUG__
22:         printk("(pid %d) sys_getsid(%d)\n", current->pid, pid);
23: #endif /*__DEBUG__ */
24:
25:         if(pid < 0) {
26:                 return -EINVAL;
27:         }
28:         if(!pid) {
29:                 return current->sid;
30:         }
31:
32:         FOR_EACH_PROCESS(p) {
33:                 if(p->state != PROC_UNUSED && p->pid == pid) {
34:                         return p->sid;
35:                 }
36:         }
37:         return -ESRCH;
38: }
```

kernel/syscalls/gettimeofday.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/gettimeofday.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/time.h>
   12: #include <fiwix/timer.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_gettimeofday(struct timeval *tv, struct timezone *tz)
   19: {
   20:         int errno;
   21:
   22: #ifdef __DEBUG__
   23:         printk("(pid %d) sys_gettimeofday()\n", current->pid);
   24: #endif /*__DEBUG__ */
   25:
   26:         if(tv) {
   27:                 if((errno = check_user_area(VERIFY_WRITE, tv, sizeof(struct time
val)))) {
   28:                         return errno;
   29:                 }
   30:                 tv->tv_sec = CURRENT_TIME;
   31:                 tv->tv_usec = ((kstat.ticks % HZ) * 1000000) / HZ;
   32:         }
   33:         if(tz) {
   34:                 if((errno = check_user_area(VERIFY_WRITE, tz, sizeof(struct time
zone)))) {
   35:                         return errno;
   36:                 }
   37:                 tz->tz_minuteswest = kstat.tz_minuteswest;
   38:                 tz->tz_dsttime = kstat.tz_dsttime;
   39:         }
   40:         return 0;
   41: }
```

kernel/syscalls/getuid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/getuid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #endif /*__DEBUG__ */
13:
14: int sys_getuid(void)
15: {
16: #ifdef __DEBUG__
17:         printk("(pid %d) sys_getuid() -> %d\n", current->pid, current->uid);
18: #endif /*__DEBUG__ */
19:
20:         return current->uid;
21: }
```

kernel/syscalls/ioctl.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/ioctl.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/process.h>
    9: #include <fiwix/errno.h>
   10:
   11: #ifdef __DEBUG__
   12: #include <fiwix/stdio.h>
   13: #endif /*__DEBUG__ */
   14:
   15: int sys_ioctl(unsigned int fd, int cmd, unsigned long int arg)
   16: {
   17:         int errno;
   18:         struct inode *i;
   19:
   20: #ifdef __DEBUG__
   21:         printk("(pid %d) sys_ioctl(%d, 0x%x, 0x%08x) -> ", current->pid, fd, cmd
, arg);
   22: #endif /*__DEBUG__ */
   23:
   24:         CHECK_UFD(fd);
   25:         i = fd_table[current->fd[fd]].inode;
   26:         if(i->fsop && i->fsop->ioctl) {
   27:                 errno = i->fsop->ioctl(i, cmd, arg);
   28:
   29: #ifdef __DEBUG__
   30:                 printk("%d\n", errno);
   31: #endif /*__DEBUG__ */
   32:
   33:                 return errno;
   34:         }
   35:
   36: #ifdef __DEBUG__
   37:         printk("%d\n", -ENOTTY);
   38: #endif /*__DEBUG__ */
   39:
   40:         return -ENOTTY;
   41: }
```

kernel/syscalls/ioperm.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/ioperm.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/process.h>
    9: #include <fiwix/errno.h>
   10:
   11: #ifdef __DEBUG__
   12: #include <fiwix/stdio.h>
   13: #endif /*__DEBUG__ */
   14:
   15: int sys_ioperm(unsigned long int from, unsigned long int num, int turn_on)
   16: {
   17: #ifdef __DEBUG__
   18:         printk("(pid %d) sys_ioperm(0x%08x, 0x%08x, 0x%08x)\n", current->pid, fr
om, num, turn_on);
   19: #endif /*__DEBUG__ */
   20:
   21:         if(!IS_SUPERUSER) {
   22:                 return -EPERM;
   23:         }
   24:
   25:         /* FIXME: to be implemented */
   26:
   27:         return 0;
   28: }
```

kernel/syscalls/iopl.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/iopl.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: /*
    9:  * Chapter Input/Output of IA-32 Intel(R) Architecture Software Developer's
   10:  * Manual Volume 1 Basic Architecture, says the processor permits applications
   11:  * to access I/O ports in either of two ways: by using I/O address space or by
   12:  * using memory-mapped I/O. Linux 2.0 and Fiwix uses the first one.
   13:  *
   14:  * This system call sets the IOPL field in the EFLAGS register to the value of
   15:  * 'level' (which is pressumably zero), so the current process will have
   16:  * privileges to use any port, even if that port is beyond of the default size
   17:  * of the I/O bitmap in TSS (which is IO_BITMAP_SIZE = 32). Otherwise the
   18:  * processor checks the I/O permission bit map to determine if access to a
   19:  * specific I/O port is allowed.
   20:  *
   21:  * So, we leave it here as in Linux 2.0. That means, leaving to I/O bit map to
   22:  * control the ports up to 0x3FF, and the rest of ports will be controlled by
   23:  * using this system call.
   24:  */
   25:
   26: #include <fiwix/process.h>
   27: #include <fiwix/segments.h>
   28: #include <fiwix/sigcontext.h>
   29: #include <fiwix/errno.h>
   30:
   31: #ifdef __DEBUG__
   32: #include <fiwix/stdio.h>
   33: #endif /*__DEBUG__ */
   34:
   35: int sys_iopl(int level, int arg2, int arg3, int arg4, int arg5, struct sigcontex
t *sc)
   36: {
   37: #ifdef __DEBUG__
   38:         printk("(pid %d) sys_iopl(%d) -> ", current->pid, level);
   39: #endif /*__DEBUG__ */
   40:         if(level > USR_PL) {
   41: #ifdef __DEBUG__
   42:         printk("-EINVAL\n");
   43: #endif /*__DEBUG__ */
   44:                 return -EINVAL;
   45:         }
   46:         if(!IS_SUPERUSER) {
   47: #ifdef __DEBUG__
   48:         printk("-EPERM\n");
   49: #endif /*__DEBUG__ */
   50:                 return -EPERM;
   51:         }
   52:
   53:         sc->eflags = (sc->eflags & 0xFFFFCFFF) | (level << EF_IOPL);
   54: #ifdef __DEBUG__
   55:         printk("0\n");
   56: #endif /*__DEBUG__ */
   57:         return 0;
   58: }
```

kernel/syscalls/kill.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/kill.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/process.h>
10: #include <fiwix/errno.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #endif /*__DEBUG__ */
15:
16: int sys_kill(__pid_t pid, __sigset_t signum)
17: {
18:         int count;
19:         struct proc *p;
20:
21: #ifdef __DEBUG__
22:         printk("(pid %d) sys_kill(%d, %d)\n", current->pid, pid, signum);
23: #endif /*__DEBUG__ */
24:
25:         if(signum < 1 || signum > NSIG) {
26:                 return -EINVAL;
27:         }
28:         if(pid == -1) {
29:                 count = 0;
30:                 FOR_EACH_PROCESS(p) {
31:                         if(p->pid > 1 && p != current) {
32:                                 count++;
33:                                 send_sig(p, signum);
34:                         }
35:                 }
36:                 return count ? 0 : -ESRCH;
37:         }
38:         if(!pid) {
39:                 return kill_pgrp(current->pgid, signum);
40:         }
41:         if(pid < 1) {
42:                 return kill_pgrp(-pid, signum);
43:         }
44:
45:         return kill_pid(pid, signum);
46: }
```

kernel/syscalls/link.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/link.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/stat.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/string.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_link(const char *oldname, const char *newname)
   19: {
   20:         struct inode *i, *dir, *i_new, *dir_new;
   21:         char *tmp_oldname, *tmp_newname, *basename;
   22:         int errno;
   23:
   24: #ifdef __DEBUG__
   25:         printk("(pid %d) sys_link('%s', '%s')\n", current->pid, oldname, newname
);
   26: #endif /*__DEBUG__ */
   27:
   28:         if((errno = malloc_name(oldname, &tmp_oldname)) < 0) {
   29:                 return errno;
   30:         }
   31:         if((errno = malloc_name(newname, &tmp_newname)) < 0) {
   32:                 free_name(tmp_oldname);
   33:                 return errno;
   34:         }
   35:
   36:         if((errno = namei(tmp_oldname, &i, &dir, !FOLLOW_LINKS))) {
   37:                 if(dir) {
   38:                         iput(dir);
   39:                 }
   40:                 free_name(tmp_oldname);
   41:                 free_name(tmp_newname);
   42:                 return errno;
   43:         }
   44:         if(S_ISDIR(i->i_mode)) {
   45:                 iput(i);
   46:                 iput(dir);
   47:                 free_name(tmp_oldname);
   48:                 free_name(tmp_newname);
   49:                 return -EPERM;
   50:         }
   51:         if(IS_RDONLY_FS(i)) {
   52:                 iput(i);
   53:                 iput(dir);
   54:                 free_name(tmp_oldname);
   55:                 free_name(tmp_newname);
   56:                 return -EROFS;
   57:         }
   58:         if(i->i_nlink == LINK_MAX) {
   59:                 iput(i);
   60:                 iput(dir);
   61:                 free_name(tmp_oldname);
   62:                 free_name(tmp_newname);
   63:                 return -EMLINK;
   64:         }
   65:
   66:         basename = get_basename(tmp_newname);
```

kernel/syscalls/link.c


```
 67:         if((errno = namei(tmp_newname, &i_new, &dir_new, !FOLLOW_LINKS))) {
 68:                 if(!dir_new) {
 69:                         iput(i);
 70:                         iput(dir);
 71:                         free_name(tmp_oldname);
 72:                         free_name(tmp_newname);
 73:                         return errno;
 74:                 }
 75:         }
 76:         if(!errno) {
 77:                 iput(i);
 78:                 iput(dir);
 79:                 iput(i_new);
 80:                 iput(dir_new);
 81:                 free_name(tmp_oldname);
 82:                 free_name(tmp_newname);
 83:                 return -EEXIST;
 84:         }
 85:         if(i->dev != dir_new->dev) {
 86:                 iput(i);
 87:                 iput(dir);
 88:                 iput(dir_new);
 89:                 free_name(tmp_oldname);
 90:                 free_name(tmp_newname);
 91:                 return -EXDEV;
 92:         }
 93:         if(check_permission(TO_EXEC | TO_WRITE, dir_new) < 0) {
 94:                 iput(i);
 95:                 iput(dir);
 96:                 iput(dir_new);
 97:                 free_name(tmp_oldname);
 98:                 free_name(tmp_newname);
 99:                 return -EACCES;
100:         }
101:
102:         if(dir_new->fsop && dir_new->fsop->link) {
103:                 errno = dir_new->fsop->link(i, dir_new, basename);
104:         } else {
105:                 errno = -EPERM;
106:         }
107:         iput(i);
108:         iput(dir);
109:         iput(dir_new);
110:         free_name(tmp_oldname);
111:         free_name(tmp_newname);
112:         return errno;
113: }
```

kernel/syscalls/llseek.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/llseek.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12: #include <fiwix/string.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_llseek(unsigned int ufd, unsigned long int offset_high, unsigned long in
t offset_low, __loff_t *result, unsigned int whence)
   19: {
   20:         struct inode *i;
   21:         __loff_t offset;
   22:         __loff_t new_offset;
   23:         int errno;
   24:
   25: #ifdef __DEBUG__
   26:         printk("(pid %d) sys_llseek(%d, %d, %d, %08x, %d)", current->pid, ufd, o
ffset_high, offset_low, result, whence);
   27: #endif /*__DEBUG__ */
   28:
   29:         CHECK_UFD(ufd);
   30:         if((errno = check_user_area(VERIFY_WRITE, result, sizeof(__loff_t)))) {
   31:                 return errno;
   32:         }
   33:         i = fd_table[current->fd[ufd]].inode;
   34:         offset = (__loff_t) (((__loff_t)offset_high << 32) | offset_low);
   35:         switch(whence) {
   36:                 case SEEK_SET:
   37:                         new_offset = offset;
   38:                         break;
   39:                 case SEEK_CUR:
   40:                         new_offset = fd_table[current->fd[ufd]].offset + offset;
   41:                         break;
   42:                 case SEEK_END:
   43:                         new_offset = i->i_size + offset;
   44:                         break;
   45:                 default:
   46:                         return -EINVAL;
   47:         }
   48:         fd_table[current->fd[ufd]].offset = new_offset;
   49:
   50: #ifdef __DEBUG__
   51:         printk(" -> returning %d\n", new_offset);
   52: #endif /*__DEBUG__ */
   53:
   54:         memcpy_b(result, &new_offset, sizeof(__loff_t));
   55:         return 0;
   56: }
```

kernel/syscalls/lseek.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/lseek.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/syscalls.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_lseek(unsigned int ufd, __off_t offset, unsigned int whence)
   19: {
   20:         struct inode *i;
   21:         __off_t new_offset;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_lseek(%d, %d, %d)", current->pid, ufd, offset, when
ce);
   25: #endif /*__DEBUG__ */
   26:
   27:         CHECK_UFD(ufd);
   28:
   29:         i = fd_table[current->fd[ufd]].inode;
   30:         switch(whence) {
   31:                 case SEEK_SET:
   32:                         new_offset = offset;
   33:                         break;
   34:                 case SEEK_CUR:
   35:                         new_offset = fd_table[current->fd[ufd]].offset + offset;
   36:                         break;
   37:                 case SEEK_END:
   38:                         new_offset = i->i_size + offset;
   39:                         break;
   40:                 default:
   41:                         return -EINVAL;
   42:         }
   43:         if((int)new_offset < 0) {
   44:                 return -EINVAL;
   45:         }
   46:         if(i->fsop && i->fsop->lseek) {
   47:                 fd_table[current->fd[ufd]].offset = new_offset;
   48:                 new_offset = i->fsop->lseek(i, new_offset);
   49:         } else {
   50:                 return -EPERM;
   51:         }
   52:
   53: #ifdef __DEBUG__
   54:         printk(" -> returning %d\n", new_offset);
   55: #endif /*__DEBUG__ */
   56:
   57:         return new_offset;
   58: }
```

kernel/syscalls/lstat.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/lstat.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/stat.h>
   10: #include <fiwix/string.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_lstat(const char *filename, struct old_stat *statbuf)
   18: {
   19:         struct inode *i;
   20:         char *tmp_name;
   21:         int errno;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_lstat('%s', 0x%08x) -> returning structure\n", curr
ent->pid, filename, (unsigned int )statbuf);
   25: #endif /*__DEBUG__ */
   26:
   27:         if((errno = check_user_area(VERIFY_WRITE, statbuf, sizeof(struct old_sta
t)))) {
   28:                 return errno;
   29:         }
   30:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   31:                 return errno;
   32:         }
   33:         if((errno = namei(tmp_name, &i, NULL, !FOLLOW_LINKS))) {
   34:                 free_name(tmp_name);
   35:                 return errno;
   36:         }
   37:         statbuf->st_dev = i->dev;
   38:         statbuf->st_ino = i->inode;
   39:         statbuf->st_mode = i->i_mode;
   40:         statbuf->st_nlink = i->i_nlink;
   41:         statbuf->st_uid = i->i_uid;
   42:         statbuf->st_gid = i->i_gid;
   43:         statbuf->st_rdev = i->rdev;
   44:         statbuf->st_size = i->i_size;
   45:         statbuf->st_atime = i->i_atime;
   46:         statbuf->st_mtime = i->i_mtime;
   47:         statbuf->st_ctime = i->i_ctime;
   48:         iput(i);
   49:         free_name(tmp_name);
   50:         return 0;
   51: }
```

kernel/syscalls/Makefile


```
 1: # fiwix/kernel/syscalls/Makefile
 2: #
 3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
 4: # Distributed under the terms of the Fiwix License.
 5: #
 6:
 7: .S.o:
 8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
 9: .c.o:
10:         $(CC) $(CFLAGS) -c -o $@ $<
11:
12: SRC = $(wildcard *.c)
13: OBJS = $(patsubst %.c,%.o,$(SRC))
14:
15: lib:    $(OBJS)
16:         $(LD) $(LDFLAGS) -r $(OBJS) -o syscalls.o
17:
18: clean:
19:         rm -f *.o
20:
```

kernel/syscalls/mkdir.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/mkdir.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/fs.h>
10: #include <fiwix/stat.h>
11: #include <fiwix/errno.h>
12: #include <fiwix/string.h>
13:
14: #ifdef __DEBUG__
15: #include <fiwix/stdio.h>
16: #include <fiwix/process.h>
17: #endif /*__DEBUG__ */
18:
19: int sys_mkdir(const char *dirname, __mode_t mode)
20: {
21:         struct inode *i, *dir;
22:         char *tmp_dirname, *basename;
23:         int errno;
24:
25: #ifdef __DEBUG__
26:         printk("(pid %d) sys_mkdir('%s', %o)\n", current->pid, dirname, mode);
27: #endif /*__DEBUG__ */
28:
29:         if((errno = malloc_name(dirname, &tmp_dirname)) < 0) {
30:                 return errno;
31:         }
32:         basename = remove_trailing_slash(tmp_dirname);
33:         if((errno = namei(basename, &i, &dir, !FOLLOW_LINKS))) {
34:                 if(!dir) {
35:                         free_name(tmp_dirname);
36:                         return errno;
37:                 }
38:         }
39:         if(!errno) {
40:                 iput(i);
41:                 iput(dir);
42:                 free_name(tmp_dirname);
43:                 return -EEXIST;
44:         }
45:         if(IS_RDONLY_FS(dir)) {
46:                 iput(dir);
47:                 free_name(tmp_dirname);
48:                 return -EROFS;
49:         }
50:
51:         if(check_permission(TO_EXEC | TO_WRITE, dir) < 0) {
52:                 iput(dir);
53:                 free_name(tmp_dirname);
54:                 return -EACCES;
55:         }
56:
57:         basename = get_basename(basename);
58:         if(dir->fsop && dir->fsop->mkdir) {
59:                 errno = dir->fsop->mkdir(dir, basename, mode);
60:         } else {
61:                 errno = -EPERM;
62:         }
63:         iput(dir);
64:         free_name(tmp_dirname);
65:         return errno;
66: }
```

kernel/syscalls/mknod.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/mknod.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/stat.h>
   11: #include <fiwix/errno.h>
   12: #include <fiwix/string.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/process.h>
   17: #endif /*__DEBUG__ */
   18:
   19: int sys_mknod(const char *pathname, __mode_t mode, __dev_t dev)
   20: {
   21:         struct inode *i, *dir;
   22:         char *tmp_name, *basename;
   23:         int errno;
   24:
   25: #ifdef __DEBUG__
   26:         printk("(pid %d) sys_mknod('%s', %d, %x)\n", current->pid, pathname, mod
e, dev);
   27: #endif /*__DEBUG__ */
   28:
   29:         if(!S_ISCHR(mode) && !S_ISBLK(mode) && !S_ISFIFO(mode)) {
   30:                 return -EINVAL;
   31:         }
   32:         if(!S_ISFIFO(mode) && !IS_SUPERUSER) {
   33:                 return -EPERM;
   34:         }
   35:
   36:         if((errno = malloc_name(pathname, &tmp_name)) < 0) {
   37:                 return errno;
   38:         }
   39:         basename = get_basename(tmp_name);
   40:         if((errno = namei(tmp_name, &i, &dir, !FOLLOW_LINKS))) {
   41:                 if(!dir) {
   42:                         free_name(tmp_name);
   43:                         return errno;
   44:                 }
   45:         }
   46:         if(!errno) {
   47:                 iput(i);
   48:                 iput(dir);
   49:                 free_name(tmp_name);
   50:                 return -EEXIST;
   51:         }
   52:         if(IS_RDONLY_FS(dir)) {
   53:                 iput(dir);
   54:                 free_name(tmp_name);
   55:                 return -EROFS;
   56:         }
   57:         if(check_permission(TO_EXEC | TO_WRITE, dir) < 0) {
   58:                 iput(dir);
   59:                 free_name(tmp_name);
   60:                 return -EACCES;
   61:         }
   62:
   63:         if(dir->fsop && dir->fsop->mknod) {
   64:                 errno = dir->fsop->mknod(dir, basename, mode, dev);
   65:         } else {
   66:                 errno = -EPERM;
```

kernel/syscalls/mknod.c


```
67:         }
68:         iput(dir);
69:         free_name(tmp_name);
70:         return errno;
71: }
```

kernel/syscalls/mount.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/mount.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/stat.h>
   11: #include <fiwix/buffer.h>
   12: #include <fiwix/mman.h>
   13: #include <fiwix/filesystems.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/errno.h>
   16: #include <fiwix/string.h>
   17:
   18: #ifdef __DEBUG__
   19: #include <fiwix/stdio.h>
   20: #include <fiwix/process.h>
   21: #endif /*__DEBUG__ */
   22:
   23: int sys_mount(const char *source, const char *target, const char *fstype, unsign
ed long int flags, const void *data)
   24: {
   25:         struct inode *i_source, *i_target;
   26:         struct mount *mt;
   27:         struct filesystems *fs;
   28:         struct vma *vma;
   29:         char *tmp_source, *tmp_target, *tmp_fstype;
   30:         __dev_t dev;
   31:         int len, errno;
   32:
   33: #ifdef __DEBUG__
   34:         printk("(pid %d) sys_mount(%s, %s, %s, 0x%08x, 0x%08x\n", current->pid,
source, target, (int)fstype ? fstype : "<NULL>", flags, data);
   35: #endif /* __DEBUG__ */
   36:
   37:         if(!IS_SUPERUSER) {
   38:                 return -EPERM;
   39:         }
   40:
   41:         if((errno = malloc_name(target, &tmp_target)) < 0) {
   42:                 return errno;
   43:         }
   44:         if((errno = namei(tmp_target, &i_target, NULL, FOLLOW_LINKS))) {
   45:                 free_name(tmp_target);
   46:                 return errno;
   47:         }
   48:         if(!S_ISDIR(i_target->i_mode)) {
   49:                 iput(i_target);
   50:                 free_name(tmp_target);
   51:                 return -ENOTDIR;
   52:         }
   53:         if((flags & MS_MGC_VAL) == MS_MGC_VAL) {
   54:                 flags &= ~MS_MGC_MSK;
   55:         }
   56:
   57:         if(flags & MS_REMOUNT) {
   58:                 if(!(mt = get_mount_point(i_target))) {
   59:                         iput(i_target);
   60:                         free_name(tmp_target);
   61:                         return -EINVAL;
   62:                 }
   63:                 fs = mt->fs;
   64:                 if(fs->fsop && fs->fsop->remount_fs) {
   65:                         if((errno = fs->fsop->remount_fs(&mt->sb, flags))) {
```

kernel/syscalls/mount.c


```
66:                                 iput(i_target);
67:                                 free_name(tmp_target);
68:                                 return errno;
69:                         }
70:                 } else {
71:                         iput(i_target);
72:                         free_name(tmp_target);
73:                         return -EINVAL;
74:                 }
75:
76:                 /* switching from RW to RO */
77:                 if(flags & MS_RDONLY && !(mt->sb.flags & MS_RDONLY)) {
78:                         dev = mt->dev;
79:                         /*
80:                          * FIXME: if there are files opened in RW mode then
81:                          * we can't continue and must return -EBUSY.
82:                          */
83:                         if(fs->fsop && fs->fsop->release_superblock) {
84:                                 fs->fsop->release_superblock(&mt->sb);
85:                         }
86:                         sync_superblocks(dev);
87:                         sync_inodes(dev);
88:                         sync_buffers(dev);
89:                 }
90:
91:                 mt->sb.flags &= ~MS_RDONLY;
92:                 mt->sb.flags |= (flags & MS_RDONLY);
93:                 iput(i_target);
94:                 free_name(tmp_target);
95:                 return 0;
96:         }
97:
98:         if(i_target->mount_point) {
99:                 iput(i_target);
100:                free_name(tmp_target);
101:                return -EBUSY;
102:        }
103:
104:        /* check the validity of fstype */
105:        if(!(vma = find_vma_region((unsigned int)fstype))) {
106:                iput(i_target);
107:                free_name(tmp_target);
108:                return -EFAULT;
109:        }
110:        if(!(vma->prot & PROT_READ)) {
111:                iput(i_target);
112:                free_name(tmp_target);
113:                return -EFAULT;
114:        }
115:        len = MIN(vma->end - (unsigned int)fstype, PAGE_SIZE - 1);
116:        if(!(tmp_fstype = (char *)kmalloc())) {
117:                iput(i_target);
118:                free_name(tmp_target);
119:                return -ENOMEM;
120:        }
121:        memcpy_b(tmp_fstype, fstype, len);
122:
123:        if(!(fs = get_filesystem(fstype))) {
124:                iput(i_target);
125:                free_name(tmp_target);
126:                free_name(tmp_fstype);
127:                return -ENODEV;
128:        }
129:        dev = fs->fsop->fsdev;
130:
131:        if((errno = malloc_name(source, &tmp_source)) < 0) {
132:                iput(i_target);
```

kernel/syscalls/mount.c


```
133:                    free_name(tmp_target);
134:                    free_name(tmp_fstype);
135:                    return errno;
136:            }
137:            if(fs->fsop->flags == FSOP_REQUIRES_DEV) {
138:                    if((errno = namei(tmp_source, &i_source, NULL, FOLLOW_LINKS))) {
139:                            iput(i_target);
140:                            free_name(tmp_target);
141:                            free_name(tmp_fstype);
142:                            free_name(tmp_source);
143:                            return errno;
144:                    }
145:                    if(!S_ISBLK(i_source->i_mode)) {
146:                            iput(i_target);
147:                            iput(i_source);
148:                            free_name(tmp_target);
149:                            free_name(tmp_fstype);
150:                            free_name(tmp_source);
151:                            return -ENOTBLK;
152:                    }
153:                    if(i_source->fsop && i_source->fsop->open) {
154:                            if((errno = i_source->fsop->open(i_source, NULL))) {
155:                                    iput(i_target);
156:                                    iput(i_source);
157:                                    free_name(tmp_target);
158:                                    free_name(tmp_fstype);
159:                                    free_name(tmp_source);
160:                                    return errno;
161:                            }
162:                    } else {
163:                            iput(i_target);
164:                            iput(i_source);
165:                            free_name(tmp_target);
166:                            free_name(tmp_fstype);
167:                            free_name(tmp_source);
168:                            return -EINVAL;
169:                    }
170:                    dev = i_source->rdev;
171:            }
172:
173:            if(!(mt = get_free_mount_point(dev))) {
174:                    if(fs->fsop->flags == FSOP_REQUIRES_DEV) {
175:                            i_source->fsop->close(i_source, NULL);
176:                            iput(i_source);
177:                    }
178:                    iput(i_target);
179:                    free_name(tmp_target);
180:                    free_name(tmp_fstype);
181:                    free_name(tmp_source);
182:                    return -EBUSY;
183:            }
184:
185:            mt->sb.flags = flags;
186:            if(fs->fsop && fs->fsop->read_superblock) {
187:                    if((errno = fs->fsop->read_superblock(dev, &mt->sb))) {
188:                            i_source->fsop->close(i_source, NULL);
189:                            if(fs->fsop->flags == FSOP_REQUIRES_DEV) {
190:                                    iput(i_source);
191:                            }
192:                            iput(i_target);
193:                            release_mount_point(mt);
194:                            free_name(tmp_target);
195:                            free_name(tmp_fstype);
196:                            free_name(tmp_source);
197:                            return errno;
198:                    }
199:            } else {
```

kernel/syscalls/mount.c


```
200:                 if(fs->fsop->flags == FSOP_REQUIRES_DEV) {
201:                         iput(i_source);
202:                 }
203:                 iput(i_target);
204:                 release_mount_point(mt);
205:                 free_name(tmp_target);
206:                 free_name(tmp_fstype);
207:                 free_name(tmp_source);
208:                 return -EINVAL;
209:         }
210:
211:         mt->dev = dev;
212:         strncpy(mt->devname, tmp_source, DEVNAME_MAX);
213:         strcpy(mt->dirname, tmp_target);
214:         mt->sb.dir = i_target;
215:         mt->fs = fs;
216:         i_target->mount_point = mt->sb.root;
217:         free_name(tmp_target);
218:         free_name(tmp_fstype);
219:         free_name(tmp_source);
220:         return 0;
221: }
```

kernel/syscalls/mprotect.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/mprotect.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/mman.h>
   10: #include <fiwix/mm.h>
   11: #include <fiwix/fcntl.h>
   12: #include <fiwix/errno.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/process.h>
   17: #endif /*__DEBUG__ */
   18:
   19: int sys_mprotect(unsigned int addr, __size_t length, int prot)
   20: {
   21:         struct vma *vma;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_mprotect(0x%08x, %d, %d)\n", current->pid, addr, le
ngth, prot);
   25: #endif /*__DEBUG__ */
   26:
   27:         if((addr & ~PAGE_MASK) || length < 0) {
   28:                 return -EINVAL;
   29:         }
   30:         if(prot > (PROT_READ | PROT_WRITE | PROT_EXEC)) {
   31:                 return -EINVAL;
   32:         }
   33:         if(!(vma = find_vma_region(addr))) {
   34:                 return -ENOMEM;
   35:         }
   36:         length = PAGE_ALIGN(length);
   37:         if((addr + length) > vma->end) {
   38:                 return -ENOMEM;
   39:         }
   40:         if(vma->inode && (vma->flags & MAP_SHARED)) {
   41:                 if(prot & PROT_WRITE) {
   42:                         if(!(vma->o_mode & (O_WRONLY | O_RDWR))) {
   43:                                 return -EACCES;
   44:                         }
   45:                 }
   46:         }
   47:
   48:         return do_mprotect(vma, addr, length, prot);
   49: }
```

kernel/syscalls/munmap.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/munmap.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/mman.h>
10:
11: #ifdef __DEBUG__
12: #include <fiwix/stdio.h>
13: #include <fiwix/process.h>
14: #endif /*__DEBUG__ */
15:
16: int sys_munmap(unsigned int addr, __size_t length)
17: {
18: #ifdef __DEBUG__
19:         printk("(pid %d) sys_munmap(0x%08x, %d)\n", current->pid, addr, length);
20: #endif /*__DEBUG__ */
21:         return do_munmap(addr, length);
22: }
```

kernel/syscalls/nanosleep.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/nanosleep.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/time.h>
   10: #include <fiwix/timer.h>
   11: #include <fiwix/process.h>
   12: #include <fiwix/sched.h>
   13: #include <fiwix/sleep.h>
   14: #include <fiwix/errno.h>
   15:
   16: #ifdef __DEBUG__
   17: #include <fiwix/stdio.h>
   18: #endif /*__DEBUG__ */
   19:
   20: int sys_nanosleep(const struct timespec *req, struct timespec *rem)
   21: {
   22:         int errno;
   23:
   24: #ifdef __DEBUG__
   25:         printk("(pid %d) sys_nanosleep(0x%08x, 0x%08x)\n", current->pid, (unsign
ed int)req, (unsigned int)rem);
   26: #endif /*__DEBUG__ */
   27:
   28:         if((errno = check_user_area(VERIFY_READ, req, sizeof(struct timespec))))
 {
   29:                 return errno;
   30:         }
   31:         if(req->tv_sec < 0 || req->tv_nsec >= 1000000000L || req->tv_nsec < 0) {
   32:                 return -EINVAL;
   33:         }
   34:
   35:         current->timeout = (req->tv_sec * HZ) + (req->tv_nsec * HZ / 1000000000L
);
   36:         if(current->timeout) {
   37:                 sleep(&sys_nanosleep, PROC_INTERRUPTIBLE);
   38:                 if(current->timeout) {
   39:                         if(rem) {
   40:                                 if((errno = check_user_area(VERIFY_WRITE, rem, s
izeof(struct timespec)))) {
   41:                                         return errno;
   42:                                 }
   43:                                 rem->tv_sec = current->timeout / HZ;
   44:                                 rem->tv_nsec = (current->timeout % HZ) * 1000000
000L / HZ;
   45:                         }
   46:                         return -EINTR;
   47:                 }
   48:         }
   49:         return 0;
   50: }
```

kernel/syscalls/newfstat.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/newfstat.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/statbuf.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_newfstat(unsigned int ufd, struct new_stat *statbuf)
   18: {
   19:         struct inode *i;
   20:         int errno;
   21:
   22: #ifdef __DEBUG__
   23:         printk("(pid %d) sys_newfstat(%d, 0x%08x) -> returning structure\n", cur
rent->pid, ufd, (unsigned int )statbuf);
   24: #endif /*__DEBUG__ */
   25:
   26:         CHECK_UFD(ufd);
   27:         if((errno = check_user_area(VERIFY_WRITE, statbuf, sizeof(struct new_sta
t)))) {
   28:                 return errno;
   29:         }
   30:         i = fd_table[current->fd[ufd]].inode;
   31:         statbuf->st_dev = i->dev;
   32:         statbuf->__pad1 = 0;
   33:         statbuf->st_ino = i->inode;
   34:         statbuf->st_mode = i->i_mode;
   35:         statbuf->st_nlink = i->i_nlink;
   36:         statbuf->st_uid = i->i_uid;
   37:         statbuf->st_gid = i->i_gid;
   38:         statbuf->st_rdev = i->rdev;
   39:         statbuf->__pad2 = 0;
   40:         statbuf->st_size = i->i_size;
   41:         statbuf->st_blksize = i->sb->s_blocksize;
   42:         statbuf->st_blocks = i->i_blocks;
   43:         if(!i->i_blocks) {
   44:                 statbuf->st_blocks = (i->i_size / i->sb->s_blocksize * 2);
   45:                 statbuf->st_blocks++;
   46:         }
   47:         statbuf->st_atime = i->i_atime;
   48:         statbuf->__unused1 = 0;
   49:         statbuf->st_mtime = i->i_mtime;
   50:         statbuf->__unused2 = 0;
   51:         statbuf->st_ctime = i->i_ctime;
   52:         statbuf->__unused3 = 0;
   53:         statbuf->__unused4 = 0;
   54:         statbuf->__unused5 = 0;
   55:         return 0;
   56: }
```

kernel/syscalls/newlstat.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/newlstat.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/statbuf.h>
   10: #include <fiwix/string.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_newlstat(const char *filename, struct new_stat *statbuf)
   18: {
   19:         struct inode *i;
   20:         char *tmp_name;
   21:         int errno;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_newlstat('%s', 0x%08x) -> returning structure\n", c
urrent->pid, filename, (unsigned int )statbuf);
   25: #endif /*__DEBUG__ */
   26:
   27:         if((errno = check_user_area(VERIFY_WRITE, statbuf, sizeof(struct new_sta
t)))) {
   28:                 return errno;
   29:         }
   30:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   31:                 return errno;
   32:         }
   33:         if((errno = namei(tmp_name, &i, NULL, !FOLLOW_LINKS))) {
   34:                 free_name(tmp_name);
   35:                 return errno;
   36:         }
   37:         statbuf->st_dev = i->dev;
   38:         statbuf->__pad1 = 0;
   39:         statbuf->st_ino = i->inode;
   40:         statbuf->st_mode = i->i_mode;
   41:         statbuf->st_nlink = i->i_nlink;
   42:         statbuf->st_uid = i->i_uid;
   43:         statbuf->st_gid = i->i_gid;
   44:         statbuf->st_rdev = i->rdev;
   45:         statbuf->__pad2 = 0;
   46:         statbuf->st_size = i->i_size;
   47:         statbuf->st_blksize = i->sb->s_blocksize;
   48:         statbuf->st_blocks = i->i_blocks;
   49:         if(!i->i_blocks) {
   50:                 statbuf->st_blocks = (i->i_size / i->sb->s_blocksize) * 2;
   51:                 statbuf->st_blocks++;
   52:         }
   53:         statbuf->st_atime = i->i_atime;
   54:         statbuf->__unused1 = 0;
   55:         statbuf->st_mtime = i->i_mtime;
   56:         statbuf->__unused2 = 0;
   57:         statbuf->st_ctime = i->i_ctime;
   58:         statbuf->__unused3 = 0;
   59:         statbuf->__unused4 = 0;
   60:         statbuf->__unused5 = 0;
   61:         iput(i);
   62:         free_name(tmp_name);
   63:         return 0;
   64: }
```

kernel/syscalls/newstat.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/newstat.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/statbuf.h>
   10: #include <fiwix/string.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_newstat(const char *filename, struct new_stat *statbuf)
   18: {
   19:         struct inode *i;
   20:         char *tmp_name;
   21:         int errno;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_newstat('%s', 0x%08x) -> returning structure\n", cu
rrent->pid, filename, (unsigned int )statbuf);
   25: #endif /*__DEBUG__ */
   26:
   27:         if((errno = check_user_area(VERIFY_WRITE, statbuf, sizeof(struct new_sta
t)))) {
   28:                 return errno;
   29:         }
   30:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   31:                 return errno;
   32:         }
   33:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
   34:                 free_name(tmp_name);
   35:                 return errno;
   36:         }
   37:         statbuf->st_dev = i->dev;
   38:         statbuf->__pad1 = 0;
   39:         statbuf->st_ino = i->inode;
   40:         statbuf->st_mode = i->i_mode;
   41:         statbuf->st_nlink = i->i_nlink;
   42:         statbuf->st_uid = i->i_uid;
   43:         statbuf->st_gid = i->i_gid;
   44:         statbuf->st_rdev = i->rdev;
   45:         statbuf->__pad2 = 0;
   46:         statbuf->st_size = i->i_size;
   47:         statbuf->st_blksize = i->sb->s_blocksize;
   48:         statbuf->st_blocks = i->i_blocks;
   49:         if(!i->i_blocks) {
   50:                 statbuf->st_blocks = (i->i_size / i->sb->s_blocksize) * 2;
   51:                 statbuf->st_blocks++;
   52:         }
   53:         statbuf->st_atime = i->i_atime;
   54:         statbuf->__unused1 = 0;
   55:         statbuf->st_mtime = i->i_mtime;
   56:         statbuf->__unused2 = 0;
   57:         statbuf->st_ctime = i->i_ctime;
   58:         statbuf->__unused3 = 0;
   59:         statbuf->__unused4 = 0;
   60:         statbuf->__unused5 = 0;
   61:         iput(i);
   62:         free_name(tmp_name);
   63:         return 0;
   64: }
```

kernel/syscalls/newuname.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/newuname.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/utsname.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/string.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_newuname(struct new_utsname *uname)
   19: {
   20:         int errno;
   21:
   22: #ifdef __DEBUG__
   23:         printk("(pid %d) sys_newuname(0x%08x)\n", current->pid, (int)uname);
   24: #endif /*__DEBUG__ */
   25:
   26:         if((errno = check_user_area(VERIFY_WRITE, uname, sizeof(struct new_utsna
me)))) {
   27:                 return errno;
   28:         }
   29:         if(!uname) {
   30:                 return -EFAULT;
   31:         }
   32:         memcpy_b(uname, &sys_utsname, sizeof(struct new_utsname));
   33:         return 0;
   34: }
```

kernel/syscalls/old_mmap.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/old_mmap.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/mman.h>
   10: #include <fiwix/mm.h>
   11: #include <fiwix/fcntl.h>
   12: #include <fiwix/errno.h>
   13: #include <fiwix/string.h>
   14:
   15: #ifdef __DEBUG__
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/process.h>
   18: #endif /*__DEBUG__ */
   19:
   20: int old_mmap(struct mmap *mmap)
   21: {
   22:         unsigned int page;
   23:         struct inode *i;
   24:         char flags;
   25:         int errno;
   26:
   27: #ifdef __DEBUG__
   28:         printk("(pid %d) old_mmap(0x%08x, %d, 0x%02x, 0x%02x, %d, 0x%08x) -> ",
current->pid, mmap->start, mmap->length, mmap->prot, mmap->flags, mmap->fd, mmap->offse
t);
   29: #endif /*__DEBUG__ */
   30:
   31:         if((errno = check_user_area(VERIFY_READ, mmap, sizeof(struct mmap)))) {
   32:                 return errno;
   33:         }
   34:         if(!mmap->length) {
   35:                 return -EINVAL;
   36:         }
   37:
   38:         i = NULL;
   39:         flags = 0;
   40:         if(!(mmap->flags & MAP_ANONYMOUS)) {
   41:                 CHECK_UFD(mmap->fd);
   42:                 if(!(i = fd_table[current->fd[mmap->fd]].inode)) {
   43:                         return -EBADF;
   44:                 }
   45:                 flags = fd_table[current->fd[mmap->fd]].flags & O_ACCMODE;
   46:         }
   47:         page = do_mmap(i, mmap->start, mmap->length, mmap->prot, mmap->flags, mm
ap->offset, P_MMAP, flags);
   48: #ifdef __DEBUG__
   49:         printk("0x%08x\n", page);
   50: #endif /*__DEBUG__ */
   51:         return page;
   52: }
```

kernel/syscalls/old_select.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/old_select.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/syscalls.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/sleep.h>
   12: #include <fiwix/sched.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/process.h>
   17: #endif /*__DEBUG__ */
   18:
   19: int old_select(unsigned long int *params)
   20: {
   21:         int nfds;
   22:         fd_set *readfds;
   23:         fd_set *writefds;
   24:         fd_set *exceptfds;
   25:         struct timeval *timeout;
   26:         int errno;
   27:
   28: #ifdef __DEBUG__
   29:         printk("(pid %d) old_select(0x%08x)\n", current->pid, (int)params);
   30: #endif /*__DEBUG__ */
   31:
   32:         if((errno = check_user_area(VERIFY_READ, (void *)params, sizeof(unsigned
int) * 5))) {
   33:                 return errno;
   34:         }
   35:         nfds = *(int *)params;
   36:         readfds = *(fd_set **)(params + 1);
   37:         writefds = *(fd_set **)(params + 2);
   38:         exceptfds = *(fd_set **)(params + 3);
   39:         timeout = *(struct timeval **)(params + 4);
   40:
   41:         return sys_select(nfds, readfds, writefds, exceptfds, timeout);
   42: }
```

kernel/syscalls/olduname.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/olduname.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/utsname.h>
   10: #include <fiwix/string.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_olduname(struct oldold_utsname *uname)
   18: {
   19:         int errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_olduname(0x%0x)", current->pid, uname);
   23: #endif /*__DEBUG__ */
   24:
   25:         if((errno = check_user_area(VERIFY_WRITE, uname, sizeof(struct oldold_ut
sname)))) {
   26:                 return errno;
   27:         }
   28:         memcpy_b(&uname->sysname, &sys_utsname.sysname, _OLD_UTSNAME_LENGTH);
   29:         memset_b(&uname->sysname + _OLD_UTSNAME_LENGTH, NULL, 1);
   30:         memcpy_b(&uname->nodename, &sys_utsname.nodename, _OLD_UTSNAME_LENGTH);
   31:         memset_b(&uname->nodename + _OLD_UTSNAME_LENGTH, NULL, 1);
   32:         memcpy_b(&uname->release, &sys_utsname.release, _OLD_UTSNAME_LENGTH);
   33:         memset_b(&uname->release + _OLD_UTSNAME_LENGTH, NULL, 1);
   34:         memcpy_b(&uname->version, &sys_utsname.version, _OLD_UTSNAME_LENGTH);
   35:         memset_b(&uname->version + _OLD_UTSNAME_LENGTH, NULL, 1);
   36:         memcpy_b(&uname->machine, &sys_utsname.machine, _OLD_UTSNAME_LENGTH);
   37:         memset_b(&uname->machine + _OLD_UTSNAME_LENGTH, NULL, 1);
   38:         return 0;
   39: }
```

kernel/syscalls/open.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/open.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/syscalls.h>
    9: #include <fiwix/stat.h>
   10: #include <fiwix/types.h>
   11: #include <fiwix/fcntl.h>
   12: #include <fiwix/errno.h>
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/string.h>
   15:
   16: int sys_open(const char *filename, int flags, __mode_t mode)
   17: {
   18:         int fd, ufd;
   19:         struct inode *i, *dir;
   20:         char *tmp_name, *basename;
   21:         int errno, follow_links, perms;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_open('%s', %o, %o)\n", current->pid, filename, flag
s, mode);
   25: #endif /*__DEBUG__ */
   26:
   27:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   28:                 return errno;
   29:         }
   30:
   31:         basename = get_basename(tmp_name);
   32:         follow_links = flags & O_NOFOLLOW ? !FOLLOW_LINKS : FOLLOW_LINKS;
   33:         if((errno = namei(tmp_name, &i, &dir, follow_links))) {
   34:                 if(!dir) {
   35:                         free_name(tmp_name);
   36:                         if(flags & O_CREAT) {
   37:                                 return -ENOENT;
   38:                         }
   39:                         return errno;
   40:                 }
   41:         }
   42:
   43: #ifdef __DEBUG__
   44:         printk("\t(inode = %d)\n", i ? i->inode : -1);
   45: #endif /*__DEBUG__ */
   46:
   47:         if(flags & O_CREAT) {
   48:                 if(!errno && (flags & O_EXCL)) {
   49:                         iput(i);
   50:                         iput(dir);
   51:                         free_name(tmp_name);
   52:                         return -EEXIST;
   53:                 }
   54:                 if(check_permission(TO_EXEC | TO_WRITE, dir) < 0) {
   55:                         iput(i);
   56:                         iput(dir);
   57:                         free_name(tmp_name);
   58:                         return -EACCES;
   59:                 }
   60:                 if(errno) {     /* assumes -ENOENT */
   61:                         if(dir->fsop && dir->fsop->create) {
   62:                                 errno = dir->fsop->create(dir, basename, mode, &
i);
   63:                                 if(errno) {
   64:                                         iput(dir);
   65:                                         free_name(tmp_name);
```

kernel/syscalls/open.c


```
   66:                                         return errno;
   67:                                 }
   68:                         } else {
   69:                                 iput(dir);
   70:                                 free_name(tmp_name);
   71:                                 return -EACCES;
   72:                         }
   73:                 }
   74:         } else {
   75:                 if(errno) {
   76:                         iput(dir);
   77:                         free_name(tmp_name);
   78:                         return errno;
   79:                 }
   80:                 if(S_ISDIR(i->i_mode) && (flags & (O_RDWR | O_WRONLY | O_TRUNC))
) {
   81:                         iput(i);
   82:                         iput(dir);
   83:                         free_name(tmp_name);
   84:                         return -EISDIR;
   85:                 }
   86:                 mode = 0;
   87:         }
   88:
   89:         if((flags & O_ACCMODE) == O_RDONLY) {
   90:                 perms = TO_READ;
   91:         } else if((flags & O_ACCMODE) == O_WRONLY) {
   92:                 perms = TO_WRITE;
   93:         } else {
   94:                 perms = TO_READ | TO_WRITE;
   95:         }
   96:         if((errno = check_permission(perms, i))) {
   97:                 iput(i);
   98:                 iput(dir);
   99:                 free_name(tmp_name);
  100:                 return errno;
  101:         }
  102:         if((fd = get_new_fd(i)) < 0) {
  103:                 iput(i);
  104:                 iput(dir);
  105:                 free_name(tmp_name);
  106:                 return fd;
  107:         }
  108:         if((ufd = get_new_user_fd(0)) < 0) {
  109:                 release_fd(fd);
  110:                 iput(i);
  111:                 iput(dir);
  112:                 free_name(tmp_name);
  113:                 return ufd;
  114:         }
  115:
  116: #ifdef __DEBUG__
  117:         printk("\t(ufd = %d)\n", ufd);
  118: #endif /*__DEBUG__ */
  119:
  120:         fd_table[fd].flags = flags;
  121:         current->fd[ufd] = fd;
  122:         if(i->fsop && i->fsop->open) {
  123:                 if((errno = i->fsop->open(i, &fd_table[fd])) < 0) {
  124:                         release_fd(fd);
  125:                         release_user_fd(ufd);
  126:                         iput(i);
  127:                         iput(dir);
  128:                         free_name(tmp_name);
  129:                         return errno;
  130:                 }
  131:                 iput(dir);
```

kernel/syscalls/open.c


```
  132:                 free_name(tmp_name);
  133:                 return ufd;
  134:         }
  135:
  136:         printk("WARNING: %s(): file '%s' (inode %d) without the open() method!\n
", __FUNCTION__, tmp_name, i->inode);
  137:         release_fd(fd);
  138:         release_user_fd(ufd);
  139:         iput(i);
  140:         iput(dir);
  141:         free_name(tmp_name);
  142:         return -EINVAL;
  143: }
```

kernel/syscalls/pause.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/pause.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/syscalls.h>
 9: #include <fiwix/sched.h>
10: #include <fiwix/sleep.h>
11: #include <fiwix/errno.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #include <fiwix/process.h>
16: #endif /*__DEBUG__ */
17:
18: int sys_pause(void)
19: {
20: #ifdef __DEBUG__
21:         printk("(pid %d) sys_pause()\n", current->pid);
22: #endif /*__DEBUG__ */
23:
24:         for(;;) {
25:                 if(sleep(&sys_pause, PROC_INTERRUPTIBLE)) {
26:                         break;
27:                 }
28:         }
29:         return -EINTR;
30: }
```

kernel/syscalls/personality.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/personality.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #ifdef __DEBUG__
    9: #include <fiwix/stdio.h>
   10: #include <fiwix/process.h>
   11: #endif /*__DEBUG__ */
   12:
   13: int sys_personality(unsigned long int persona)
   14: {
   15: #ifdef __DEBUG__
   16:         printk("(pid %d) sys_personality(%d) -> %d\n", current->pid, persona, pe
rsona);
   17: #endif /*__DEBUG__ */
   18:         return persona;
   19: }
```

kernel/syscalls/pipe.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/pipe.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/filesystems.h>
   10: #include <fiwix/fcntl.h>
   11: #include <fiwix/stat.h>
   12: #include <fiwix/errno.h>
   13: #include <fiwix/stdio.h>
   14:
   15: int sys_pipe(int pipefd[2])
   16: {
   17:         int rfd, rufd;
   18:         int wfd, wufd;
   19:         struct filesystems *fs;
   20:         struct inode *i;
   21:         int errno;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_pipe()", current->pid);
   25: #endif /*__DEBUG__ */
   26:
   27:         if(!(fs = get_filesystem("pipefs"))) {
   28:                 printk("WARNING: %s(): pipefs filesystem is not registered!\n",
__FUNCTION__);
   29:                 return -EINVAL;
   30:         }
   31:         if((errno = check_user_area(VERIFY_WRITE, pipefd, sizeof(int) * 2))) {
   32:                 return errno;
   33:         }
   34:         if(!(i = ialloc(&fs->mt->sb, S_IFIFO))) {
   35:                 return -EINVAL;
   36:         }
   37:         if((rfd = get_new_fd(i)) < 0) {
   38:                 iput(i);
   39:                 return -ENFILE;
   40:         }
   41:         if((wfd = get_new_fd(i)) < 0) {
   42:                 release_fd(rfd);
   43:                 iput(i);
   44:                 return -ENFILE;
   45:         }
   46:         if((rufd = get_new_user_fd(0)) < 0) {
   47:                 release_fd(rfd);
   48:                 release_fd(wfd);
   49:                 iput(i);
   50:                 return -EMFILE;
   51:         }
   52:         if((wufd = get_new_user_fd(0)) < 0) {
   53:                 release_fd(rfd);
   54:                 release_fd(wfd);
   55:                 release_user_fd(rufd);
   56:                 iput(i);
   57:                 return -EMFILE;
   58:         }
   59:
   60:         pipefd[0] = rufd;
   61:         pipefd[1] = wufd;
   62:         current->fd[rufd] = rfd;
   63:         current->fd[wufd] = wfd;
   64:         fd_table[rfd].flags = O_RDONLY;
   65:         fd_table[wfd].flags = O_WRONLY;
   66:
```

kernel/syscalls/pipe.c


```
   67: #ifdef __DEBUG__
   68:         printk(" -> inode=%d, rufd=%d wufd=%d (rfd=%d wfd=%d)\n", i->inode, rufd
, wufd, rfd, wfd);
   69: #endif /*__DEBUG__ */
   70:
   71:         return 0;
   72: }
```

kernel/syscalls/read.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/read.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/fcntl.h>
   10: #include <fiwix/errno.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_read(unsigned int ufd, char *buf, int count)
   18: {
   19:         struct inode *i;
   20:         int errno;
   21:
   22: #ifdef __DEBUG__
   23:         printk("(pid %d) sys_read(%d, 0x%08x, %d) -> ", current->pid, ufd, buf,
count);
   24: #endif /*__DEBUG__ */
   25:
   26:         CHECK_UFD(ufd);
   27:         if((errno = check_user_area(VERIFY_WRITE, buf, count))) {
   28:                 return errno;
   29:         }
   30:         if(fd_table[current->fd[ufd]].flags & O_WRONLY) {
   31:                 return -EBADF;
   32:         }
   33:         if(!count) {
   34:                 return 0;
   35:         }
   36:         if(count < 0) {
   37:                 return -EINVAL;
   38:         }
   39:
   40:         i = fd_table[current->fd[ufd]].inode;
   41:         if(i->fsop && i->fsop->read) {
   42:                 errno = i->fsop->read(i, &fd_table[current->fd[ufd]], buf, count
);
   43: #ifdef __DEBUG__
   44:                 printk("%d\n", errno);
   45: #endif /*__DEBUG__ */
   46:                 return errno;
   47:         }
   48:         return -EINVAL;
   49: }
```

kernel/syscalls/readlink.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/readlink.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/stat.h>
   11: #include <fiwix/errno.h>
   12: #include <fiwix/string.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/process.h>
   17: #endif /*__DEBUG__ */
   18:
   19: int sys_readlink(const char *filename, char *buffer, __size_t bufsize)
   20: {
   21:         struct inode *i;
   22:         char *tmp_name;
   23:         int errno;
   24:
   25: #ifdef __DEBUG__
   26:         printk("(pid %d) sys_readlink(%s, 0x%08x, %d)\n", current->pid, filename
, (unsigned int)buffer, bufsize);
   27: #endif /*__DEBUG__ */
   28:
   29:         if(bufsize <= 0) {
   30:                 return -EINVAL;
   31:         }
   32:         if((errno = check_user_area(VERIFY_WRITE, buffer, bufsize))) {
   33:                 return errno;
   34:         }
   35:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   36:                 return errno;
   37:         }
   38:         if((errno = namei(tmp_name, &i, NULL, !FOLLOW_LINKS))) {
   39:                 free_name(tmp_name);
   40:                 return errno;
   41:         }
   42:         if(!S_ISLNK(i->i_mode)) {
   43:                 iput(i);
   44:                 free_name(tmp_name);
   45:                 return -EINVAL;
   46:         }
   47:
   48:         if(i->fsop && i->fsop->readlink) {
   49:                 errno = i->fsop->readlink(i, buffer, bufsize);
   50:                 iput(i);
   51:                 free_name(tmp_name);
   52:                 return errno;
   53:         }
   54:         iput(i);
   55:         free_name(tmp_name);
   56:         return -EINVAL;
   57: }
```

kernel/syscalls/reboot.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/reboot.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/syscalls.h>
   10: #include <fiwix/reboot.h>
   11: #include <fiwix/signal.h>
   12: #include <fiwix/process.h>
   13: #include <fiwix/errno.h>
   14:
   15: #ifdef __DEBUG__
   16: #include <fiwix/stdio.h>
   17: #endif /*__DEBUG__ */
   18:
   19: int sys_reboot(int magic1, int magic2, int flag)
   20: {
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_reboot(0x%08x, %d, 0x%08x)\n", current->pid, magic1
, magic2, flag);
   23: #endif /*__DEBUG__ */
   24:
   25:         if(!IS_SUPERUSER) {
   26:                 return -EPERM;
   27:         }
   28:         if((magic1 != BMAGIC_1) || (magic2 != BMAGIC_2)) {
   29:                 return -EINVAL;
   30:         }
   31:         switch(flag) {
   32:                 case BMAGIC_SOFT:
   33:                         ctrl_alt_del = 0;
   34:                         break;
   35:                 case BMAGIC_HARD:
   36:                         ctrl_alt_del = 1;
   37:                         break;
   38:                 case BMAGIC_REBOOT:
   39:                         reboot();
   40:                         break;
   41:                 case BMAGIC_HALT:
   42:                         sys_kill(-1, SIGKILL);
   43:                         stop_kernel();
   44:                         break;
   45:                 default:
   46:                         return -EINVAL;
   47:         }
   48:         return 0;
   49: }
```

kernel/syscalls/rename.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/rename.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/stat.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/string.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_rename(const char *oldpath, const char *newpath)
   19: {
   20:         struct inode *i, *dir, *i_new, *dir_new;
   21:         char *tmp_oldpath, *tmp_newpath;
   22:         char *oldbasename, *newbasename;
   23:         int errno;
   24:
   25: #ifdef __DEBUG__
   26:         printk("(pid %d) sys_rename('%s', '%s')\n", current->pid, oldpath, newpa
th);
   27: #endif /*__DEBUG__ */
   28:
   29:         if((errno = malloc_name(oldpath, &tmp_oldpath)) < 0) {
   30:                 return errno;
   31:         }
   32:         if((errno = namei(tmp_oldpath, &i, &dir, !FOLLOW_LINKS))) {
   33:                 if(dir) {
   34:                         iput(dir);
   35:                 }
   36:                 free_name(tmp_oldpath);
   37:                 return errno;
   38:         }
   39:         if(IS_RDONLY_FS(i)) {
   40:                 iput(i);
   41:                 iput(dir);
   42:                 free_name(tmp_oldpath);
   43:                 return -EROFS;
   44:         }
   45:
   46:         if((errno = malloc_name(newpath, &tmp_newpath)) < 0) {
   47:                 iput(i);
   48:                 iput(dir);
   49:                 free_name(tmp_oldpath);
   50:                 return errno;
   51:         }
   52:         newbasename = remove_trailing_slash(tmp_newpath);
   53:         if((errno = namei(newbasename, &i_new, &dir_new, !FOLLOW_LINKS))) {
   54:                 if(!dir_new) {
   55:                         iput(i);
   56:                         iput(dir);
   57:                         free_name(tmp_oldpath);
   58:                         free_name(tmp_newpath);
   59:                         return errno;
   60:                 }
   61:         }
   62:         if(dir->dev != dir_new->dev) {
   63:                 errno = -EXDEV;
   64:                 goto end;
   65:         }
   66:         newbasename = get_basename(newbasename);
```

kernel/syscalls/rename.c


```
   67:         if((newbasename[0] == '.' && newbasename[1] == NULL) || (newbasename[0]
== '.' && newbasename[1] == '.' && newbasename[2] == NULL)) {
   68:                 errno = -EINVAL;
   69:                 goto end;
   70:         }
   71:
   72:         oldbasename = get_basename(tmp_oldpath);
   73:         oldbasename = remove_trailing_slash(oldbasename);
   74:         if((oldbasename[0] == '.' && oldbasename[1] == NULL) || (oldbasename[0]
== '.' && oldbasename[1] == '.' && oldbasename[2] == NULL)) {
   75:                 errno = -EINVAL;
   76:                 goto end;
   77:         }
   78:
   79:         if(i_new) {
   80:                 if(S_ISREG(i->i_mode)) {
   81:                         if(S_ISDIR(i_new->i_mode)) {
   82:                                 errno = -EISDIR;
   83:                                 goto end;
   84:                         }
   85:                 }
   86:                 if(S_ISDIR(i->i_mode)) {
   87:                         if(!S_ISDIR(i_new->i_mode)) {
   88:                                 errno = -ENOTDIR;
   89:                                 goto end;
   90:                         }
   91:                 }
   92:                 if(i->inode == i_new->inode) {
   93:                         errno = 0;
   94:                         goto end;
   95:                 }
   96:         }
   97:
   98:         if(check_permission(TO_EXEC | TO_WRITE, dir_new) < 0) {
   99:                 errno = -EACCES;
  100:                 goto end;
  101:         }
  102:
  103:         if(dir_new->fsop && dir_new->fsop->rename) {
  104:                 errno = dir_new->fsop->rename(i, dir, i_new, dir_new, oldbasenam
e, newbasename);
  105:         } else {
  106:                 errno = -EPERM;
  107:         }
  108:
  109: end:
  110:         iput(i);
  111:         iput(dir);
  112:         iput(i_new);
  113:         iput(dir_new);
  114:         free_name(tmp_oldpath);
  115:         free_name(tmp_newpath);
  116:         return errno;
  117: }
```

kernel/syscalls/rmdir.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/rmdir.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/stat.h>
10: #include <fiwix/errno.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #include <fiwix/process.h>
15: #endif /*__DEBUG__ */
16:
17: int sys_rmdir(const char *dirname)
18: {
19:         struct inode *i, *dir;
20:         char *tmp_dirname;
21:         int errno;
22:
23: #ifdef __DEBUG__
24:         printk("(pid %d) sys_rmdir(%s)\n", current->pid, dirname);
25: #endif /*__DEBUG__ */
26:
27:         if((errno = malloc_name(dirname, &tmp_dirname)) < 0) {
28:                 return errno;
29:         }
30:         if((errno = namei(tmp_dirname, &i, &dir, !FOLLOW_LINKS))) {
31:                 if(dir) {
32:                         iput(dir);
33:                 }
34:                 free_name(tmp_dirname);
35:                 return errno;
36:         }
37:         if(!S_ISDIR(i->i_mode)) {
38:                 iput(i);
39:                 iput(dir);
40:                 free_name(tmp_dirname);
41:                 return -ENOTDIR;
42:         }
43:         if(i == current->root || i->mount_point || i->count > 1) {
44:                 iput(i);
45:                 iput(dir);
46:                 free_name(tmp_dirname);
47:                 return -EBUSY;
48:         }
49:         if(IS_RDONLY_FS(i)) {
50:                 iput(i);
51:                 iput(dir);
52:                 free_name(tmp_dirname);
53:                 return -EROFS;
54:         }
55:         if(i == dir) {
56:                 iput(i);
57:                 iput(dir);
58:                 free_name(tmp_dirname);
59:                 return -EPERM;
60:         }
61:         if(check_permission(TO_EXEC | TO_WRITE, dir) < 0) {
62:                 iput(i);
63:                 iput(dir);
64:                 free_name(tmp_dirname);
65:                 return -EACCES;
66:         }
67:
```

kernel/syscalls/rmdir.c


```
68:         /* check sticky permission bit */
69:         if(dir->i_mode & S_ISVTX) {
70:                 if(check_user_permission(i)) {
71:                         iput(i);
72:                         iput(dir);
73:                         free_name(tmp_dirname);
74:                         return -EPERM;
75:                 }
76:         }
77:
78:         if(i->fsop && i->fsop->rmdir) {
79:                 errno = i->fsop->rmdir(dir, i);
80:         } else {
81:                 errno = -EPERM;
82:         }
83:         iput(i);
84:         iput(dir);
85:         free_name(tmp_dirname);
86:         return errno;
87: }
```

kernel/syscalls/select.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/select.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/timer.h>
   12: #include <fiwix/sched.h>
   13: #include <fiwix/sleep.h>
   14: #include <fiwix/errno.h>
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/string.h>
   17:
   18: static int check_fds(int nfds, fd_set *rfds, fd_set *wfds, fd_set *efds)
   19: {
   20:         int n, bit;
   21:         unsigned long int set;
   22:
   23:         n = bit = 0;
   24:         while(bit < nfds) {
   25:                 bit = n * __NFDBITS;
   26:                 set = rfds->fds_bits[n] | wfds->fds_bits[n] | efds->fds_bits[n];
   27:                 while(set) {
   28:                         if(__FD_ISSET(bit, rfds)) {
   29:                                 CHECK_UFD(bit);
   30:                         }
   31:                         set >>= 1;
   32:                         bit++;
   33:                 }
   34:                 n++;
   35:         }
   36:
   37:         return 0;
   38: }
   39:
   40: static int do_check(struct inode *i, int flag)
   41: {
   42:         if(i->fsop && i->fsop->select) {
   43:                 if(i->fsop->select(i, flag)) {
   44:                         return 1;
   45:                 }
   46:         }
   47:
   48:         return 0;
   49: }
   50:
   51: int do_select(int nfds, fd_set *rfds, fd_set *wfds, fd_set *efds, fd_set *res_rf
ds, fd_set *res_wfds, fd_set *res_efds)
   52: {
   53:         int n, count;
   54:         struct inode *i;
   55:
   56:         count = 0;
   57:         for(;;) {
   58:                 for(n = 0; n < nfds; n++) {
   59:                         if(!current->fd[n]) {
   60:                                 continue;
   61:                         }
   62:                         i = fd_table[current->fd[n]].inode;
   63:                         if(__FD_ISSET(n, rfds)) {
   64:                                 if(do_check(i, SEL_R)) {
   65:                                         __FD_SET(n, res_rfds);
   66:                                         count++;
```

kernel/syscalls/select.c


```
   67:                                 }
   68:                         }
   69:                         if(__FD_ISSET(n, wfds)) {
   70:                                 if(do_check(i, SEL_W)) {
   71:                                         __FD_SET(n, res_wfds);
   72:                                         count++;
   73:                                 }
   74:                         }
   75:                         if(__FD_ISSET(n, efds)) {
   76:                                 if(do_check(i, SEL_E)) {
   77:                                         __FD_SET(n, res_efds);
   78:                                         count++;
   79:                                 }
   80:                         }
   81:                 }
   82:
   83:                 if(count || !current->timeout || current->sigpending & ~current-
>sigblocked) {
   84:                         break;
   85:                 }
   86:                 sleep(&do_select, PROC_INTERRUPTIBLE);
   87:         }
   88:
   89:         return count;
   90: }
   91:
   92: int sys_select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, s
truct timeval *timeout)
   93: {
   94:         unsigned long int t;
   95:         fd_set rfds, wfds, efds;
   96:         fd_set res_rfds, res_wfds, res_efds;
   97:         int errno;
   98:
   99: #ifdef __DEBUG__
  100:         printk("(pid %d) sys_select(%d, 0x%08x, 0x%08x, 0x%08x, 0x%08x [%d])\n",
 current->pid, nfds, (int)readfds, (int)writefds, (int)exceptfds, (int)timeout, (int)ti
meout ? tv2ticks(timeout): 0);
  101: #endif /*__DEBUG__ */
  102:
  103:         if(nfds < 0) {
  104:                 return -EINVAL;
  105:         }
  106:         if(nfds > MIN(__FD_SETSIZE, NR_OPENS)) {
  107:                 nfds = MIN(__FD_SETSIZE, NR_OPENS);
  108:         }
  109:
  110:         if(readfds) {
  111:                 if((errno = check_user_area(VERIFY_WRITE, readfds, sizeof(fd_set
)))) {
  112:                         return errno;
  113:                 }
  114:                 memcpy_b(&rfds, readfds, sizeof(fd_set));
  115:         } else {
  116:                 __FD_ZERO(&rfds);
  117:         }
  118:         if(writefds) {
  119:                 if((errno = check_user_area(VERIFY_WRITE, writefds, sizeof(fd_se
t)))) {
  120:                         return errno;
  121:                 }
  122:                 memcpy_b(&wfds, writefds, sizeof(fd_set));
  123:         } else {
  124:                 __FD_ZERO(&wfds);
  125:         }
  126:         if(exceptfds) {
  127:                 if((errno = check_user_area(VERIFY_WRITE, exceptfds, sizeof(fd_s
```

kernel/syscalls/select.c


```
et)))) {
  128:                          return errno;
  129:                  }
  130:                  memcpy_b(&efds, exceptfds, sizeof(fd_set));
  131:          } else {
  132:                  __FD_ZERO(&efds);
  133:          }
  134:
  135:          /* check the validity of all fds */
  136:          if((errno = check_fds(nfds, &rfds, &wfds, &efds)) < 0) {
  137:                  return errno;
  138:          }
  139:
  140:          if(timeout) {
  141:                  t = tv2ticks(timeout);
  142:          } else {
  143:                  t = INFINITE_WAIT;
  144:          }
  145:
  146:          __FD_ZERO(&res_rfds);
  147:          __FD_ZERO(&res_wfds);
  148:          __FD_ZERO(&res_efds);
  149:
  150:          current->timeout = t;
  151:          if((errno = do_select(nfds, &rfds, &wfds, &efds, &res_rfds, &res_wfds, &
res_efds)) < 0) {
  152:                  return errno;
  153:          }
  154:          current->timeout = 0;
  155:
  156:          if(readfds) {
  157:                  memcpy_b(readfds, &res_rfds, sizeof(fd_set));
  158:          }
  159:          if(writefds) {
  160:                  memcpy_b(writefds, &res_wfds, sizeof(fd_set));
  161:          }
  162:          if(exceptfds) {
  163:                  memcpy_b(exceptfds, &res_efds, sizeof(fd_set));
  164:          }
  165:          return errno;
  166: }
```

kernel/syscalls/setdomainname.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/setdomainname.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/utsname.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/string.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_setdomainname(const char *name, int length)
   19: {
   20:         int errno;
   21:
   22: #ifdef __DEBUG__
   23:         printk("(pid %d) sys_setdomainname('%s', %d)\n", current->pid, name, len
gth);
   24: #endif /*__DEBUG__ */
   25:
   26:         if((errno = check_user_area(VERIFY_READ, name, length))) {
   27:                 return errno;
   28:         }
   29:         if(!IS_SUPERUSER) {
   30:                 return -EPERM;
   31:         }
   32:         if(length < 0 || length > _UTSNAME_LENGTH) {
   33:                 return -EINVAL;
   34:         }
   35:         memcpy_b(&sys_utsname.domainname, name, length);
   36:         sys_utsname.domainname[length] = NULL;
   37:         return 0;
   38: }
```

kernel/syscalls/setfsgid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/setfsgid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #include <fiwix/process.h>
13: #endif /*__DEBUG__ */
14:
15: int sys_setfsgid(__gid_t fsgid)
16: {
17: #ifdef __DEBUG__
18:         printk("(pid %d) sys_setfsgid(%d) -> %d\n", current->pid, fsgid);
19: #endif /*__DEBUG__ */
20:         return 0;
21: }
```

kernel/syscalls/setfsuid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/setfsuid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #include <fiwix/process.h>
13: #endif /*__DEBUG__ */
14:
15: int sys_setfsuid(__uid_t fsuid)
16: {
17: #ifdef __DEBUG__
18:         printk("(pid %d) sys_setfsuid(%d) -> %d\n", current->pid, fsuid);
19: #endif /*__DEBUG__ */
20:         return 0;
21: }
```

kernel/syscalls/setgid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/setgid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/process.h>
10: #include <fiwix/errno.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #endif /*__DEBUG__ */
15:
16: int sys_setgid(__gid_t gid)
17: {
18: #ifdef __DEBUG__
19:         printk("(pid %d) sys_setgid(%d)\n", current->pid, gid);
20: #endif /*__DEBUG__ */
21:
22:         if(IS_SUPERUSER) {
23:                 current->gid = current->egid = current->sgid = gid;
24:         } else {
25:                 if((current->gid == gid) || (current->sgid == gid)) {
26:                         current->egid = gid;
27:                 } else {
28:                         return -EPERM;
29:                 }
30:         }
31:         return 0;
32: }
```

kernel/syscalls/setgroups.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/setgroups.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_setgroups(__ssize_t size, const __gid_t *list)
   18: {
   19:         int n, errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_setgroups(%d, 0x%08x)\n", current->pid, size, (unsi
gned int)list);
   23: #endif /*__DEBUG__ */
   24:
   25:         if((errno = check_user_area(VERIFY_READ, list, sizeof(__gid_t)))) {
   26:                 return errno;
   27:         }
   28:         if(!IS_SUPERUSER) {
   29:                 return -EPERM;
   30:         }
   31:         if(size > NGROUPS_MAX) {
   32:                 return -EINVAL;
   33:         }
   34:         for(n = 0; n < NGROUPS_MAX; n++) {
   35:                 current->groups[n] = -1;
   36:                 if(n < size) {
   37:                         current->groups[n] = list[n];
   38:                 }
   39:         }
   40:         return 0;
   41: }
```

kernel/syscalls/sethostname.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/sethostname.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/utsname.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12: #include <fiwix/string.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_sethostname(const char *name, int length)
   19: {
   20:         int errno;
   21:         char *tmp_name;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_sethostname('%s', %d)\n", current->pid, name, lengt
h);
   25: #endif /*__DEBUG__ */
   26:
   27:         if((errno = malloc_name(name, &tmp_name)) < 0) {
   28:                 return errno;
   29:         }
   30:         if(!IS_SUPERUSER) {
   31:                 free_name(tmp_name);
   32:                 return -EPERM;
   33:         }
   34:         if(length < 0 || length > _UTSNAME_LENGTH) {
   35:                 free_name(tmp_name);
   36:                 return -EINVAL;
   37:         }
   38:         memcpy_b(&sys_utsname.nodename, tmp_name, length);
   39:         sys_utsname.nodename[length] = NULL;
   40:         free_name(tmp_name);
   41:         return 0;
   42: }
```

kernel/syscalls/setitimer.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/setitimer.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/time.h>
   10:
   11: #ifdef __DEBUG__
   12: #include <fiwix/stdio.h>
   13: #include <fiwix/process.h>
   14: #endif /*__DEBUG__ */
   15:
   16: int sys_setitimer(int which, const struct itimerval *new_value, struct itimerval
 *old_value)
   17: {
   18:         int errno;
   19:
   20: #ifdef __DEBUG__
   21:         printk("(pid %d) sys_setitimer(%d, 0x%08x, 0x%08x) -> \n", current->pid,
 which, (unsigned int)new_value, (unsigned int)old_value);
   22: #endif /*__DEBUG__ */
   23:
   24:         if((unsigned int)old_value) {
   25:                 if((errno = check_user_area(VERIFY_WRITE, old_value, sizeof(stru
ct itimerval)))) {
   26:                         return errno;
   27:                 }
   28:         }
   29:
   30:         return setitimer(which, new_value, old_value);
   31: }
```

kernel/syscalls/setpgid.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/setpgid.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/process.h>
   10: #include <fiwix/sched.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_setpgid(__pid_t pid, __pid_t pgid)
   18: {
   19:         struct proc *p;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_setpgid(%d, %d)", current->pid, pid, pgid);
   23: #endif /*__DEBUG__ */
   24:
   25:         if(pid < 0 || pgid < 0) {
   26:                 return -EINVAL;
   27:         }
   28:         if(!pid) {
   29:                 pid = current->pid;
   30:         }
   31:
   32:         p = get_proc_by_pid(pid);
   33:         if(!pgid) {
   34:                 pgid = p->pid;
   35:         }
   36:
   37:         if(p != current && (p->state == PROC_UNUSED || p->ppid != current->pid))
{
   38:                 return -ESRCH;
   39:         }
   40:         if(p->sid == p->pid || p->sid != current->sid) {
   41:                 return -EPERM;
   42:         }
   43:
   44:         {
   45:                 struct proc *p;
   46:
   47:                 FOR_EACH_PROCESS(p) {
   48:                         if(p->state != PROC_UNUSED) {
   49:                                 if(p->pgid == pgid && p->sid != current->sid) {
   50:                                         return -EPERM;
   51:                                 }
   52:                         }
   53:                 }
   54:         }
   55:
   56:         if(p != current && p->flags & PF_PEXEC) {
   57:                 return -EACCES;
   58:         }
   59:
   60:         p->pgid = pgid;
   61:
   62: #ifdef __DEBUG__
   63:         printk(" -> 0\n");
   64: #endif /*__DEBUG__ */
   65:
   66:         return 0;
```

kernel/syscalls/setpgid.c


```
67: }
```

kernel/syscalls/setregid.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/setregid.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/process.h>
   10: #include <fiwix/errno.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #endif /*__DEBUG__ */
   15:
   16: int sys_setregid(__gid_t gid, __gid_t egid)
   17: {
   18: #ifdef __DEBUG__
   19:         printk("(pid %d) sys_setregid(%d, %d) -> ", current->pid, gid, egid);
   20: #endif /*__DEBUG__ */
   21:
   22:         if(IS_SUPERUSER) {
   23:                 if(egid >= 0) {
   24:                         if(gid >= 0 || (current->egid >= 0 && current->gid != eg
id)) {
   25:                                 current->sgid = egid;
   26:                         }
   27:                         current->egid = egid;
   28:                 }
   29:                 if(gid >= 0) {
   30:                         current->gid = gid;
   31:                 }
   32:         } else {
   33:                 if(egid >= 0 && (current->gid == egid || current->egid == egid |
| current->sgid == egid)) {
   34:                         if(gid >= 0 || (current->egid >= 0 && current->gid != eg
id)) {
   35:                                 current->sgid = egid;
   36:                         }
   37:                         current->egid = egid;
   38:                 } else {
   39:                         return -EPERM;
   40:                 }
   41:                 if(gid >= 0 && (current->gid == gid || current->egid == gid)) {
   42:                         current->gid = gid;
   43:                 } else {
   44:                         return -EPERM;
   45:                 }
   46:         }
   47:
   48:         return 0;
   49: }
```

kernel/syscalls/setreuid.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/setreuid.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/process.h>
   10: #include <fiwix/errno.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #endif /*__DEBUG__ */
   15:
   16: int sys_setreuid(__uid_t uid, __uid_t euid)
   17: {
   18: #ifdef __DEBUG__
   19:         printk("(pid %d) sys_setreuid(%d, %d) -> ", current->pid, uid, euid);
   20: #endif /*__DEBUG__ */
   21:
   22:         if(IS_SUPERUSER) {
   23:                 if(euid >= 0) {
   24:                         if(uid >= 0 || (current->euid >= 0 && current->uid != eu
id)) {
   25:                                 current->suid = euid;
   26:                         }
   27:                         current->euid = euid;
   28:                 }
   29:                 if(uid >= 0) {
   30:                         current->uid = uid;
   31:                 }
   32:         } else {
   33:                 if(euid >= 0 && (current->uid == euid || current->euid == euid |
| current->suid == euid)) {
   34:                         if(uid >= 0 || (current->euid >= 0 && current->uid != eu
id)) {
   35:                                 current->suid = euid;
   36:                         }
   37:                         current->euid = euid;
   38:                 } else {
   39:                         return -EPERM;
   40:                 }
   41:                 if(uid >= 0 && (current->uid == uid || current->euid == uid)) {
   42:                         current->uid = uid;
   43:                 } else {
   44:                         return -EPERM;
   45:                 }
   46:         }
   47:
   48:         return 0;
   49: }
```

kernel/syscalls/setrlimit.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/setrlimit.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/resource.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12: #include <fiwix/string.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_setrlimit(int resource, const struct rlimit *rlim)
   19: {
   20:         int errno;
   21:
   22: #ifdef __DEBUG__
   23:         printk("(pid %d) sys_setrlimit(%d, 0x%08x)\n", current->pid, resource, (
unsigned int)rlim);
   24: #endif /*__DEBUG__ */
   25:
   26:         if((errno = check_user_area(VERIFY_READ, rlim, sizeof(struct rlimit))))
{
   27:                 return errno;
   28:         }
   29:         if(resource < 0 || resource >= RLIM_NLIMITS) {
   30:                 return -EINVAL;
   31:         }
   32:         if(rlim->rlim_cur > rlim->rlim_max) {
   33:                 return -EINVAL;
   34:         }
   35:         if(!IS_SUPERUSER) {
   36:                 if(rlim->rlim_max > current->rlim[resource].rlim_max) {
   37:                         return -EPERM;
   38:                 }
   39:         }
   40:         memcpy_b(&current->rlim[resource], rlim, sizeof(struct rlimit));
   41:         return 0;
   42: }
```

kernel/syscalls/setsid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/setsid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9: #include <fiwix/errno.h>
10: #include <fiwix/string.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #endif /*__DEBUG__ */
15:
16: int sys_setsid(void)
17: {
18:         struct proc *p;
19:
20: #ifdef __DEBUG__
21:         printk("(pid %d) sys_setsid()\n", current->pid);
22: #endif /*__DEBUG__ */
23:
24:         if(PG_LEADER(current)) {
25:                 return -EPERM;
26:         }
27:         FOR_EACH_PROCESS(p) {   /* POSIX ANSI/IEEE Std 1003.1-1996 4.3.2 */
28:                 if(p != current && p->pgid == current->pid) {
29:                         return -EPERM;
30:                 }
31:         }
32:
33:         current->sid = current->pgid = current->pid;
34:         current->ctty = NULL;
35:         return current->sid;
36: }
```

kernel/syscalls/settimeofday.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/settimeofday.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/time.h>
   11: #include <fiwix/timer.h>
   12: #include <fiwix/process.h>
   13: #include <fiwix/errno.h>
   14:
   15: #ifdef __DEBUG__
   16: #include <fiwix/stdio.h>
   17: #endif /*__DEBUG__ */
   18:
   19: int sys_settimeofday(const struct timeval *tv, const struct timezone *tz)
   20: {
   21:         int errno;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_settimeofday()\n", current->pid);
   25: #endif /*__DEBUG__ */
   26:
   27:         if(!IS_SUPERUSER) {
   28:                 return -EPERM;
   29:         }
   30:
   31:         if(tv) {
   32:                 if((errno = check_user_area(VERIFY_READ, tv, sizeof(struct timev
al)))) {
   33:                         return errno;
   34:                 }
   35:                 CURRENT_TIME = tv->tv_sec;
   36:                 set_system_time(CURRENT_TIME);
   37:         }
   38:         if(tz) {
   39:                 if((errno = check_user_area(VERIFY_READ, tz, sizeof(struct timez
one)))) {
   40:                         return errno;
   41:                 }
   42:                 kstat.tz_minuteswest = tz->tz_minuteswest;
   43:                 kstat.tz_dsttime = tz->tz_dsttime;
   44:         }
   45:         return 0;
   46: }
```

kernel/syscalls/setuid.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/setuid.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/process.h>
10: #include <fiwix/errno.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #endif /*__DEBUG__ */
15:
16: int sys_setuid(__uid_t uid)
17: {
18: #ifdef __DEBUG__
19:         printk("(pid %d) sys_setuid(%d)\n", current->pid, uid);
20: #endif /*__DEBUG__ */
21:
22:         if(IS_SUPERUSER) {
23:                 current->uid = current->suid = uid;
24:         } else {
25:                 if((current->uid != uid) && (current->suid != uid)) {
26:                         return -EPERM;
27:                 }
28:         }
29:         current->euid = uid;
30:         return 0;
31: }
```

kernel/syscalls/sgetmask.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/sgetmask.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9:
10: #ifdef __DEBUG__
11: #include <fiwix/stdio.h>
12: #endif /*__DEBUG__ */
13:
14: int sys_sgetmask(void)
15: {
16: #ifdef __DEBUG__
17:         printk("(pid %d) sys_sgetmask() -> \n", current->pid);
18: #endif /*__DEBUG__ */
19:         return current->sigblocked;
20: }
```

kernel/syscalls/sigaction.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/sigaction.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/signal.h>
   10: #include <fiwix/errno.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_sigaction(__sigset_t signum, const struct sigaction *newaction, struct s
igaction *oldaction)
   18: {
   19:         int errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_sigaction(%d, 0x%08x, 0x%08x)\n", current->pid, sig
num, (unsigned int)newaction, (unsigned int)oldaction);
   23: #endif /*__DEBUG__ */
   24:
   25:         if(signum < 1 || signum > NSIG) {
   26:                 return -EINVAL;
   27:         }
   28:         if(signum == SIGKILL || signum == SIGSTOP) {
   29:                 return -EINVAL;
   30:         }
   31:         if(oldaction) {
   32:                 if((errno = check_user_area(VERIFY_WRITE, oldaction, sizeof(stru
ct sigaction)))) {
   33:                         return errno;
   34:                 }
   35:                 *oldaction = current->sigaction[signum - 1];
   36:         }
   37:         if(newaction) {
   38:                 if((errno = check_user_area(VERIFY_READ, newaction, sizeof(struc
t sigaction)))) {
   39:                         return errno;
   40:                 }
   41:                 current->sigaction[signum - 1] = *newaction;
   42:                 if(current->sigaction[signum - 1].sa_handler == SIG_IGN) {
   43:                         if(signum != SIGCHLD) {
   44:                                 current->sigpending &= SIG_MASK(signum);
   45:                         }
   46:                 }
   47:                 if(current->sigaction[signum - 1].sa_handler == SIG_DFL) {
   48:                         if(signum != SIGCHLD) {
   49:                                 current->sigpending &= SIG_MASK(signum);
   50:                         }
   51:                 }
   52:         }
   53:         return 0;
   54: }
```

kernel/syscalls/signal.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/signal.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/syscalls.h>
    9: #include <fiwix/signal.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12: #include <fiwix/string.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #endif /*__DEBUG__ */
   17:
   18: unsigned int sys_signal(__sigset_t signum, void(* sighandler)(int))
   19: {
   20:         struct sigaction s;
   21:         int errno;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_signal()\n", current->pid);
   25: #endif /*__DEBUG__ */
   26:
   27:         if(signum < 1 || signum > NSIG) {
   28:                 return -EINVAL;
   29:         }
   30:         if(signum == SIGKILL || signum == SIGSTOP) {
   31:                 return -EINVAL;
   32:         }
   33:         if(sighandler != SIG_DFL && sighandler != SIG_IGN) {
   34:                 if((errno = check_user_area(VERIFY_READ, sighandler, sizeof(void
))))) {
   35:                         return errno;
   36:                 }
   37:         }
   38:
   39:         memset_b(&s, 0, sizeof(struct sigaction));
   40:         s.sa_handler = sighandler;
   41:         s.sa_mask = 0;
   42:         s.sa_flags = SA_RESETHAND;
   43:         sighandler = current->sigaction[signum - 1].sa_handler;
   44:         current->sigaction[signum - 1] = s;
   45:         if(current->sigaction[signum - 1].sa_handler == SIG_IGN) {
   46:                 if(signum != SIGCHLD) {
   47:                         current->sigpending &= SIG_MASK(signum);
   48:                 }
   49:         }
   50:         if(current->sigaction[signum - 1].sa_handler == SIG_DFL) {
   51:                 if(signum != SIGCHLD) {
   52:                         current->sigpending &= SIG_MASK(signum);
   53:                 }
   54:         }
   55:         return (unsigned int)sighandler;
   56: }
```

kernel/syscalls/sigpending.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/sigpending.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/signal.h>
10: #include <fiwix/process.h>
11: #include <fiwix/string.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #endif /*__DEBUG__ */
16:
17: int sys_sigpending(__sigset_t *set)
18: {
19:         int errno;
20:
21: #ifdef __DEBUG__
22:         printk("(pid %d) sys_sigpending(0x%08x) -> ", current->pid, set);
23: #endif /*__DEBUG__ */
24:
25:         if((errno = check_user_area(VERIFY_WRITE, set, sizeof(__sigset_t)))) {
26:                 return errno;
27:         }
28:         memcpy_b(set, &current->sigpending, sizeof(__sigset_t));
29:         return 0;
30: }
```

kernel/syscalls/sigprocmask.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/sigprocmask.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/signal.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/errno.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_sigprocmask(int how, const __sigset_t *set, __sigset_t *oldset)
   18: {
   19:         int errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_sigprocmask(%d, 0x%08x, 0x%08x)\n", current->pid, h
ow, set, oldset);
   23: #endif /*__DEBUG__ */
   24:
   25:         if(oldset) {
   26:                 if((errno = check_user_area(VERIFY_WRITE, oldset, sizeof(__sigse
t_t)))) {
   27:                         return errno;
   28:                 }
   29:                 *oldset = current->sigblocked;
   30:         }
   31:
   32:         if(set) {
   33:                 if((errno = check_user_area(VERIFY_READ, set, sizeof(__sigset_t)
))) {
   34:                         return errno;
   35:                 }
   36:                 switch(how) {
   37:                         case SIG_BLOCK:
   38:                                 current->sigblocked |= (*set & SIG_BLOCKABLE);
   39:                                 break;
   40:                         case SIG_UNBLOCK:
   41:                                 current->sigblocked &= ~(*set & SIG_BLOCKABLE);
   42:                                 break;
   43:                         case SIG_SETMASK:
   44:                                 current->sigblocked = (*set & SIG_BLOCKABLE);
   45:                                 break;
   46:                         default:
   47:                                 return -EINVAL;
   48:                 }
   49:         }
   50:         return 0;
   51: }
```

kernel/syscalls/sigreturn.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/sigreturn.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/process.h>
    9: #include <fiwix/sigcontext.h>
   10: #include <fiwix/string.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #endif /*__DEBUG__ */
   15:
   16: int sys_sigreturn(unsigned int signum, int arg2, int arg3, int arg4, int arg5, s
truct sigcontext *sc)
   17: {
   18: #ifdef __DEBUG__
   19:         printk("(pid %d) sys_sigreturn(0x%08x)\n", current->pid, signum);
   20: #endif /*__DEBUG__ */
   21:
   22:         current->sigblocked &= ~current->sigexecuting;
   23:         current->sigexecuting = 0;
   24:         memcpy_b(sc, &current->sc[signum - 1], sizeof(struct sigcontext));
   25:
   26:         /*
   27:          * We return here the value that the syscall was returning when it was
   28:          * interrupted by a signal.
   29:          */
   30:         return current->sc[signum - 1].eax;
   31: }
```

kernel/syscalls/sigsuspend.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/sigsuspend.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/syscalls.h>
   10: #include <fiwix/signal.h>
   11: #include <fiwix/process.h>
   12: #include <fiwix/errno.h>
   13:
   14: #ifdef __DEBUG__
   15: #include <fiwix/stdio.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_sigsuspend(__sigset_t *mask)
   19: {
   20:         __sigset_t old_mask;
   21:         int errno;
   22:
   23: #ifdef __DEBUG__
   24:         printk("(pid %d) sys_sigsuspend(0x%08x) -> ", current->pid, mask);
   25: #endif /*__DEBUG__ */
   26:
   27:         old_mask = current->sigblocked;
   28:         if(mask) {
   29:                 if((errno = check_user_area(VERIFY_READ, mask, sizeof(__sigset_t
))))) {
   30:                         return errno;
   31:                 }
   32:                 current->sigblocked = (int)*mask & SIG_BLOCKABLE;
   33:         } else {
   34:                 current->sigblocked = 0 & SIG_BLOCKABLE;
   35:         }
   36:         sys_pause();
   37:         current->sigblocked = old_mask;
   38:
   39: #ifdef __DEBUG__
   40:         printk("-EINTR\n");
   41: #endif /*__DEBUG__ */
   42:
   43:         return -EINTR;
   44: }
```

kernel/syscalls/socketcall.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/socketcall.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/errno.h>
    9:
   10: #ifdef __DEBUG__
   11: #include <fiwix/stdio.h>
   12: #include <fiwix/process.h>
   13: #endif /*__DEBUG__ */
   14:
   15: int sys_socketcall(int call, unsigned long int *args)
   16: {
   17: #ifdef __DEBUG__
   18:         printk("(pid %d) sys_socketcall(%d, 0x%08x) -> ENOENT\n", current->pid,
call, args);
   19: #endif /*__DEBUG__ */
   20:
   21:         /* FIXME: to be implemented */
   22:
   23:         return -ENOENT;
   24: }
```

kernel/syscalls/ssetmask.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/ssetmask.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/process.h>
 9: #include <fiwix/signal.h>
10:
11: #ifdef __DEBUG__
12: #include <fiwix/stdio.h>
13: #endif /*__DEBUG__ */
14:
15: int sys_ssetmask(int newmask)
16: {
17:         int oldmask;
18:
19: #ifdef __DEBUG__
20:         printk("(pid %d) sys_ssetmask(0x%08x) -> \n", current->pid, newmask);
21: #endif /*__DEBUG__ */
22:
23:         oldmask = current->sigblocked;
24:         current->sigblocked = newmask & SIG_BLOCKABLE;
25:         return oldmask;
26: }
```

kernel/syscalls/stat.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/stat.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/statbuf.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/string.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_stat(const char *filename, struct old_stat *statbuf)
   19: {
   20:         struct inode *i;
   21:         char *tmp_name;
   22:         int errno;
   23:
   24: #ifdef __DEBUG__
   25:         printk("(pid %d) sys_stat(%s, 0x%08x) -> returning structure\n", current
->pid, filename, (unsigned int )statbuf);
   26: #endif /*__DEBUG__ */
   27:
   28:         if((errno = check_user_area(VERIFY_WRITE, statbuf, sizeof(struct old_sta
t)))) {
   29:                 return errno;
   30:         }
   31:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   32:                 return errno;
   33:         }
   34:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
   35:                 free_name(tmp_name);
   36:                 return errno;
   37:         }
   38:         statbuf->st_dev = i->dev;
   39:         statbuf->st_ino = i->inode;
   40:         statbuf->st_mode = i->i_mode;
   41:         statbuf->st_nlink = i->i_nlink;
   42:         statbuf->st_uid = i->i_uid;
   43:         statbuf->st_gid = i->i_gid;
   44:         statbuf->st_rdev = i->rdev;
   45:         statbuf->st_size = i->i_size;
   46:         statbuf->st_atime = i->i_atime;
   47:         statbuf->st_mtime = i->i_mtime;
   48:         statbuf->st_ctime = i->i_ctime;
   49:         iput(i);
   50:         free_name(tmp_name);
   51:         return 0;
   52: }
```

kernel/syscalls/statfs.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/statfs.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/statfs.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/string.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_statfs(const char *filename, struct statfs *statfsbuf)
   19: {
   20:         struct inode *i;
   21:         char *tmp_name;
   22:         int errno;
   23:
   24: #ifdef __DEBUG__
   25:         printk("(pid %d) sys_statfs('%s', 0x%08x)\n", current->pid, filename, (u
nsigned int)statfsbuf);
   26: #endif /*__DEBUG__ */
   27:
   28:         if((errno = check_user_area(VERIFY_WRITE, statfsbuf, sizeof(struct statf
s)))) {
   29:                 return errno;
   30:         }
   31:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   32:                 return errno;
   33:         }
   34:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
   35:                 free_name(tmp_name);
   36:                 return errno;
   37:         }
   38:         if(i->sb && i->sb->fsop && i->sb->fsop->statfs) {
   39:                 i->sb->fsop->statfs(i->sb, statfsbuf);
   40:                 iput(i);
   41:                 free_name(tmp_name);
   42:                 return 0;
   43:         }
   44:         iput(i);
   45:         free_name(tmp_name);
   46:         return -ENOSYS;
   47: }
```

kernel/syscalls/stime.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/stime.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/fs.h>
10: #include <fiwix/timer.h>
11: #include <fiwix/errno.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #include <fiwix/process.h>
16: #endif /*__DEBUG__ */
17:
18: int sys_stime(__time_t *t)
19: {
20:         int errno;
21:
22: #ifdef __DEBUG__
23:         printk("(pid %d) sys_stime(0x%08x)\n", current->pid, (unsigned int)t);
24: #endif /*__DEBUG__ */
25:
26:         if(!IS_SUPERUSER) {
27:                 return -EPERM;
28:         }
29:         if((errno = check_user_area(VERIFY_READ, t, sizeof(__time_t)))) {
30:                 return errno;
31:         }
32:
33:         set_system_time(*t);
34:         return 0;
35: }
```

kernel/syscalls/symlink.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/symlink.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/stat.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/string.h>
   12:
   13: #ifdef __DEBUG__
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/process.h>
   16: #endif /*__DEBUG__ */
   17:
   18: int sys_symlink(const char *oldpath, const char *newpath)
   19: {
   20:         struct inode *i, *dir;
   21:         char *tmp_oldpath, *tmp_newpath, *basename;
   22:         int errno;
   23:
   24: #ifdef __DEBUG__
   25:         printk("(pid %d) sys_symlink('%s', '%s')\n", current->pid, oldpath, newp
ath);
   26: #endif /*__DEBUG__ */
   27:
   28:         if((errno = malloc_name(oldpath, &tmp_oldpath)) < 0) {
   29:                 return errno;
   30:         }
   31:         if((errno = malloc_name(newpath, &tmp_newpath)) < 0) {
   32:                 free_name(tmp_oldpath);
   33:                 return errno;
   34:         }
   35:         basename = get_basename(tmp_newpath);
   36:         if((errno = namei(tmp_newpath, &i, &dir, !FOLLOW_LINKS))) {
   37:                 if(!dir) {
   38:                         free_name(tmp_oldpath);
   39:                         free_name(tmp_newpath);
   40:                         return errno;
   41:                 }
   42:         }
   43:         if(!errno) {
   44:                 iput(i);
   45:                 iput(dir);
   46:                 free_name(tmp_oldpath);
   47:                 free_name(tmp_newpath);
   48:                 return -EEXIST;
   49:         }
   50:         if(IS_RDONLY_FS(dir)) {
   51:                 iput(dir);
   52:                 free_name(tmp_oldpath);
   53:                 free_name(tmp_newpath);
   54:                 return -EROFS;
   55:         }
   56:
   57:         if(check_permission(TO_EXEC | TO_WRITE, dir) < 0) {
   58:                 iput(dir);
   59:                 free_name(tmp_oldpath);
   60:                 free_name(tmp_newpath);
   61:                 return -EACCES;
   62:         }
   63:
   64:         if(dir->fsop && dir->fsop->symlink) {
   65:                 errno = dir->fsop->symlink(dir, basename, tmp_oldpath);
   66:         } else {
```

kernel/syscalls/symlink.c


```
67:                 errno = -EPERM;
68:         }
69:         iput(dir);
70:         free_name(tmp_oldpath);
71:         free_name(tmp_newpath);
72:         return errno;
73: }
```

kernel/syscalls/sync.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/sync.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/buffer.h>
10: #include <fiwix/filesystems.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #include <fiwix/process.h>
15: #endif /*__DEBUG__ */
16:
17: void sys_sync(void)
18: {
19: #ifdef __DEBUG__
20:         printk("(pid %d) sys_sync()\n", current->pid);
21: #endif /*__DEBUG__ */
22:
23:         sync_superblocks(0);    /* in all devices */
24:         sync_inodes(0);         /* in all devices */
25:         sync_buffers(0);        /* in all devices */
26:         return;
27: }
```

kernel/syscalls/sysinfo.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/sysinfo.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/system.h>
   11: #include <fiwix/sched.h>
   12: #include <fiwix/mm.h>
   13: #include <fiwix/string.h>
   14:
   15: #ifdef __DEBUG__
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/process.h>
   18: #endif /*__DEBUG__ */
   19:
   20: int sys_sysinfo(struct sysinfo *info)
   21: {
   22:         struct sysinfo tmp_info;
   23:         struct proc *p;
   24:         int errno;
   25:
   26: #ifdef __DEBUG__
   27:         printk("(pid %d) sys_sysinfo(0x%08x)\n ", current->pid, (unsigned int)in
fo);
   28: #endif /*__DEBUG__ */
   29:
   30:         if((errno = check_user_area(VERIFY_WRITE, info, sizeof(struct sysinfo)))
) {
   31:                 return errno;
   32:         }
   33:         memset_b(&tmp_info, NULL, sizeof(struct sysinfo));
   34:         tmp_info.loads[0] = avenrun[0] << (SI_LOAD_SHIFT - FSHIFT);
   35:         tmp_info.loads[1] = avenrun[1] << (SI_LOAD_SHIFT - FSHIFT);
   36:         tmp_info.loads[2] = avenrun[2] << (SI_LOAD_SHIFT - FSHIFT);
   37:         tmp_info.uptime = kstat.uptime;
   38:         tmp_info.totalram = kstat.total_mem_pages << PAGE_SHIFT;
   39:         tmp_info.freeram = kstat.free_pages << PAGE_SHIFT;
   40:         tmp_info.sharedram = 0;
   41:         tmp_info.bufferram = kstat.buffers * 1024;
   42:         tmp_info.totalswap = 0;
   43:         tmp_info.freeswap = 0;
   44:         FOR_EACH_PROCESS(p) {
   45:                 if(p->state) {
   46:                         tmp_info.procs++;
   47:                 }
   48:         }
   49:
   50:         memcpy_b(info, &tmp_info, sizeof(struct sysinfo));
   51:         return 0;
   52: }
```

kernel/syscalls/time.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/time.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/fs.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_time(__time_t *tloc)
   18: {
   19:         int errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_time() -> ", current->pid);
   23: #endif /*__DEBUG__ */
   24:
   25:         if(tloc) {
   26:                 if((errno = check_user_area(VERIFY_WRITE, tloc, sizeof(__time_t)
))) {
   27:                         return errno;
   28:                 }
   29:                 *tloc = CURRENT_TIME;
   30:         }
   31:
   32: #ifdef __DEBUG__
   33:         printk("%d\n", CURRENT_TIME);
   34: #endif /*__DEBUG__ */
   35:
   36:         return CURRENT_TIME;
   37: }
```

kernel/syscalls/times.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/times.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/kernel.h>
 9: #include <fiwix/types.h>
10: #include <fiwix/syscalls.h>
11: #include <fiwix/times.h>
12:
13: #ifdef __DEBUG__
14: #include <fiwix/stdio.h>
15: #include <fiwix/process.h>
16: #endif /*__DEBUG__ */
17:
18: int sys_times(struct tms *buf)
19: {
20:         int errno;
21:
22: #ifdef __DEBUG__
23:         printk("(pid %d) sys_times(0x%08x) -> ", (unsigned int )buf);
24: #endif /*__DEBUG__ */
25:
26:         if((errno = check_user_area(VERIFY_WRITE, buf, sizeof(struct tms)))) {
27:                 return errno;
28:         }
29:         if(buf) {
30:                 buf->tms_utime = tv2ticks(&current->usage.ru_utime);
31:                 buf->tms_stime = tv2ticks(&current->usage.ru_stime);
32:                 buf->tms_cutime = tv2ticks(&current->cusage.ru_utime);
33:                 buf->tms_cstime = tv2ticks(&current->cusage.ru_stime);
34:         }
35:
36:         return kstat.ticks;
37: }
```

kernel/syscalls/truncate.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/truncate.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/fs.h>
10: #include <fiwix/stat.h>
11: #include <fiwix/errno.h>
12: #include <fiwix/string.h>
13:
14: #ifdef __DEBUG__
15: #include <fiwix/stdio.h>
16: #include <fiwix/process.h>
17: #endif /*__DEBUG__ */
18:
19: int sys_truncate(const char *path, __off_t length)
20: {
21:         struct inode *i;
22:         char *tmp_name;
23:         int errno;
24:
25: #ifdef __DEBUG__
26:         printk("(pid %d) sys_truncate(%s, %d)\n", current->pid, path, length);
27: #endif /*__DEBUG__ */
28:
29:         if((errno = malloc_name(path, &tmp_name)) < 0) {
30:                 return errno;
31:         }
32:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
33:                 free_name(tmp_name);
34:                 return errno;
35:         }
36:         if(S_ISDIR(i->i_mode)) {
37:                 iput(i);
38:                 free_name(tmp_name);
39:                 return -EISDIR;
40:         }
41:         if(IS_RDONLY_FS(i)) {
42:                 iput(i);
43:                 free_name(tmp_name);
44:                 return -EROFS;
45:         }
46:         if(check_permission(TO_WRITE, i) < 0) {
47:                 iput(i);
48:                 free_name(tmp_name);
49:                 return -EACCES;
50:         }
51:         if(length == i->i_size) {
52:                 iput(i);
53:                 free_name(tmp_name);
54:                 return 0;
55:         }
56:
57:         errno = 0;
58:         if(i->fsop && i->fsop->truncate) {
59:                 inode_lock(i);
60:                 errno = i->fsop->truncate(i, length);
61:                 inode_unlock(i);
62:         }
63:         iput(i);
64:         free_name(tmp_name);
65:         return errno;
66: }
```

kernel/syscalls/umask.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/umask.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/process.h>
10: #include <fiwix/stat.h>
11:
12: #ifdef __DEBUG__
13: #include <fiwix/stdio.h>
14: #endif /*__DEBUG__ */
15:
16: int sys_umask(__mode_t mask)
17: {
18:         __mode_t old_umask;
19:
20: #ifdef __DEBUG__
21:         printk("(pid %d) sys_umask(%d)\n", current->pid, mask);
22: #endif /*__DEBUG__ */
23:
24:         old_umask = current->umask;
25:         current->umask = mask & (S_IRWXU | S_IRWXG | S_IRWXO);
26:         return old_umask;
27: }
```

kernel/syscalls/umount2.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/umount2.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/filesystems.h>
   11: #include <fiwix/stat.h>
   12: #include <fiwix/sleep.h>
   13: #include <fiwix/devices.h>
   14: #include <fiwix/buffer.h>
   15: #include <fiwix/errno.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: static struct resource umount_resource = { NULL, NULL };
   20:
   21: int sys_umount2(const char *target, int flags)
   22: {
   23:         struct inode *i_target;
   24:         struct mount *mt = NULL;
   25:         struct filesystems *fs;
   26:         struct device *d;
   27:         struct inode dummy_i;
   28:         struct superblock *sb;
   29:         char *tmp_target;
   30:         __dev_t dev;
   31:         int errno;
   32:
   33: #ifdef __DEBUG__
   34:         printk("(pid %d) sys_umount2(%s, 0x%08x)\n", current->pid, target, flags
);
   35: #endif /*__DEBUG__ */
   36:
   37:         if(!IS_SUPERUSER) {
   38:                 return -EPERM;
   39:         }
   40:         if((errno = malloc_name(target, &tmp_target)) < 0) {
   41:                 return errno;
   42:         }
   43:         if((errno = namei(tmp_target, &i_target, NULL, FOLLOW_LINKS))) {
   44:                 free_name(tmp_target);
   45:                 return errno;
   46:         }
   47:         if(!S_ISBLK(i_target->i_mode) && !S_ISDIR(i_target->i_mode)) {
   48:                 iput(i_target);
   49:                 free_name(tmp_target);
   50:                 return -EINVAL;
   51:         }
   52:
   53:         if(!(mt = get_mount_point(i_target))) {
   54:                 iput(i_target);
   55:                 free_name(tmp_target);
   56:                 return -EINVAL;
   57:         }
   58:         if(S_ISBLK(i_target->i_mode)) {
   59:                 dev = i_target->rdev;
   60:         } else {
   61:                 dev = i_target->sb->dev;
   62:         }
   63:
   64:         if(!(sb = get_superblock(dev))) {
   65:                 printk("WARNING: %s(): unable to get superblock from device %d,%
d\n", __FUNCTION__, MAJOR(dev), MINOR(dev));
```

kernel/syscalls/umount2.c


```
   66:                 iput(i_target);
   67:                 free_name(tmp_target);
   68:                 return -EINVAL;
   69:         }
   70:
   71:         /*
   72:          * We must free now the inode in order to avoid having its 'count' to 2
   73:          * when calling check_fs_busy(), specially if sys_umount() was called
   74:          * using the mount-point instead of the device.
   75:          */
   76:         iput(i_target);
   77:         free_name(tmp_target);
   78:
   79:         if(check_fs_busy(dev, sb->root)) {
   80:                 return -EBUSY;
   81:         }
   82:
   83:         lock_resource(&umount_resource);
   84:
   85:         fs = mt->fs;
   86:         if(fs->fsop && fs->fsop->release_superblock) {
   87:                 fs->fsop->release_superblock(sb);
   88:         }
   89:         if(sb->fsop->flags & FSOP_REQUIRES_DEV) {
   90:                 if(!(d = get_device(BLK_DEV, MAJOR(dev)))) {
   91:                         printk("WARNING: %s(): block device %d,%d not registered
!\n", __FUNCTION__, MAJOR(dev), MINOR(dev));
   92:                         unlock_resource(&umount_resource);
   93:                         return -EINVAL;
   94:                 }
   95:                 memset_b(&dummy_i, 0, sizeof(struct inode));
   96:                 dummy_i.dev = dummy_i.rdev = dev;
   97:                 if(d && d->fsop && d->fsop->close) {
   98:                         d->fsop->close(&dummy_i, NULL);
   99:                 }
  100:         }
  101:
  102:         sb->dir->mount_point = NULL;
  103:         iput(sb->root);
  104:         iput(sb->dir);
  105:
  106:         sync_superblocks(dev);
  107:         sync_inodes(dev);
  108:         sync_buffers(dev);
  109:         invalidate_buffers(dev);
  110:         invalidate_inodes(dev);
  111:
  112:         release_mount_point(mt);
  113:         unlock_resource(&umount_resource);
  114:         return 0;
  115: }
```

kernel/syscalls/umount.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/umount.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/syscalls.h>
 9: #include <fiwix/process.h>
10:
11: #ifdef __DEBUG__
12: #include <fiwix/stdio.h>
13: #endif /*__DEBUG__ */
14:
15: int sys_umount(const char *target)
16: {
17: #ifdef __DEBUG__
18:         printk("(pid %d) sys_umount(%s)\n", current->pid, target);
19: #endif /*__DEBUG__ */
20:
21:         return sys_umount2(target, 0);
22: }
```

kernel/syscalls/uname.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/uname.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/utsname.h>
   10: #include <fiwix/string.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_uname(struct old_utsname *uname)
   18: {
   19:         int errno;
   20:
   21: #ifdef __DEBUG__
   22:         printk("(pid %d) sys_uname(0x%08x) -> returning ", current->pid, (unsign
ed int)uname);
   23: #endif /*__DEBUG__ */
   24:
   25:         if((errno = check_user_area(VERIFY_WRITE, uname, sizeof(struct old_utsna
me)))) {
   26:                 return errno;
   27:         }
   28:         memcpy_b(&uname->sysname, &sys_utsname.sysname, sizeof(sys_utsname.sysna
me));
   29:         memcpy_b(&uname->nodename, &sys_utsname.nodename, sizeof(sys_utsname.nod
ename));
   30:         memcpy_b(&uname->release, &sys_utsname.release, sizeof(sys_utsname.relea
se));
   31:         memcpy_b(&uname->version, &sys_utsname.version, sizeof(sys_utsname.versi
on));
   32:         memcpy_b(&uname->machine, &sys_utsname.machine, sizeof(sys_utsname.machi
ne));
   33:         return 0;
   34: }
```

kernel/syscalls/unlink.c


```
 1: /*
 2:  * fiwix/kernel/syscalls/unlink.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/fs.h>
 9: #include <fiwix/syscalls.h>
10: #include <fiwix/stat.h>
11: #include <fiwix/errno.h>
12: #include <fiwix/string.h>
13:
14: #ifdef __DEBUG__
15: #include <fiwix/stdio.h>
16: #include <fiwix/process.h>
17: #endif /*__DEBUG__ */
18:
19: int sys_unlink(const char *filename)
20: {
21:         struct inode *i, *dir;
22:         char *tmp_name, *basename;
23:         int errno;
24:
25: #ifdef __DEBUG__
26:         printk("(pid %d) sys_unlink('%s')\n", current->pid, filename);
27: #endif /*__DEBUG__ */
28:
29:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
30:                 return errno;
31:         }
32:         if((errno = namei(tmp_name, &i, &dir, !FOLLOW_LINKS))) {
33:                 if(dir) {
34:                         iput(dir);
35:                 }
36:                 free_name(tmp_name);
37:                 return errno;
38:         }
39:         if(S_ISDIR(i->i_mode)) {
40:                 iput(i);
41:                 iput(dir);
42:                 free_name(tmp_name);
43:                 return -EPERM;  /* Linux returns -EISDIR */
44:         }
45:         if(IS_RDONLY_FS(i)) {
46:                 iput(i);
47:                 iput(dir);
48:                 free_name(tmp_name);
49:                 return -EROFS;
50:         }
51:         if(check_permission(TO_EXEC | TO_WRITE, dir) < 0) {
52:                 iput(i);
53:                 iput(dir);
54:                 free_name(tmp_name);
55:                 return -EACCES;
56:         }
57:
58:         /* check sticky permission bit */
59:         if(dir->i_mode & S_ISVTX) {
60:                 if(check_user_permission(i)) {
61:                         iput(i);
62:                         iput(dir);
63:                         free_name(tmp_name);
64:                         return -EPERM;
65:                 }
66:         }
67:
```

kernel/syscalls/unlink.c


```
68:         basename = get_basename(filename);
69:         if(dir->fsop && dir->fsop->unlink) {
70:                 errno = dir->fsop->unlink(dir, i, basename);
71:         } else {
72:                 errno = -EPERM;
73:         }
74:         iput(i);
75:         iput(dir);
76:         free_name(tmp_name);
77:         return errno;
78: }
```

kernel/syscalls/ustat.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/ustat.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/filesystems.h>
   11: #include <fiwix/ustat.h>
   12: #include <fiwix/statfs.h>
   13: #include <fiwix/errno.h>
   14: #include <fiwix/string.h>
   15:
   16: #ifdef __DEBUG__
   17: #include <fiwix/stdio.h>
   18: #include <fiwix/process.h>
   19: #endif /*__DEBUG__ */
   20:
   21: int sys_ustat(__dev_t dev, struct ustat *ubuf)
   22: {
   23:         struct superblock *sb;
   24:         struct statfs statfsbuf;
   25:         int errno;
   26:
   27: #ifdef __DEBUG__
   28:         printk("(pid %d) sys_ustat(%d, 0x%08x)\n", current->pid, dev, (int)ubuf)
;
   29: #endif /*__DEBUG__ */
   30:         if((errno = check_user_area(VERIFY_WRITE, ubuf, sizeof(struct ustat))))
{
   31:                 return errno;
   32:         }
   33:         if(!(sb = get_superblock(dev))) {
   34:                 return -EINVAL;
   35:         }
   36:         if(sb->fsop && sb->fsop->statfs) {
   37:                 sb->fsop->statfs(sb, &statfsbuf);
   38:                 memset_b(ubuf, NULL, sizeof(struct ustat));
   39:                 ubuf->f_tfree = statfsbuf.f_bfree;
   40:                 ubuf->f_tinode = statfsbuf.f_ffree;
   41:                 return 0;
   42:         }
   43:         return -ENOSYS;
   44: }
```

kernel/syscalls/utime.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/utime.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/utime.h>
   11: #include <fiwix/stat.h>
   12: #include <fiwix/errno.h>
   13: #include <fiwix/string.h>
   14:
   15: #ifdef __DEBUG__
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/process.h>
   18: #endif /*__DEBUG__ */
   19:
   20: int sys_utime(const char *filename, struct utimbuf *times)
   21: {
   22:         struct inode *i;
   23:         char *tmp_name;
   24:         int errno;
   25:
   26: #ifdef __DEBUG__
   27:         printk("(pid %d) sys_utime('%s', 0x%08x)\n", current->pid, filename, (in
t)times);
   28: #endif /*__DEBUG__ */
   29:
   30:         if((errno = malloc_name(filename, &tmp_name)) < 0) {
   31:                 return errno;
   32:         }
   33:         if((errno = namei(tmp_name, &i, NULL, FOLLOW_LINKS))) {
   34:                 free_name(tmp_name);
   35:                 return errno;
   36:         }
   37:
   38:         if(IS_RDONLY_FS(i)) {
   39:                 iput(i);
   40:                 free_name(tmp_name);
   41:                 return -EROFS;
   42:         }
   43:
   44:         if(!times) {
   45:                 if(check_user_permission(i) || check_permission(TO_WRITE, i)) {
   46:                         iput(i);
   47:                         free_name(tmp_name);
   48:                         return -EACCES;
   49:                 }
   50:                 i->i_atime = CURRENT_TIME;
   51:                 i->i_mtime = CURRENT_TIME;
   52:         } else {
   53:                 if((errno = check_user_area(VERIFY_READ, times, sizeof(struct ut
imbuf)))) {
   54:                         iput(i);
   55:                         free_name(tmp_name);
   56:                         return errno;
   57:                 }
   58:                 if(check_user_permission(i)) {
   59:                         iput(i);
   60:                         free_name(tmp_name);
   61:                         return -EPERM;
   62:                 }
   63:                 i->i_atime = times->actime;
   64:                 i->i_mtime = times->modtime;
   65:         }
```

kernel/syscalls/utime.c


```
66:
67:         i->i_ctime = CURRENT_TIME;
68:         i->dirty = 1;
69:         iput(i);
70:         free_name(tmp_name);
71:         return 0;
72: }
```

kernel/syscalls/wait4.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/wait4.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/resource.h>
   11: #include <fiwix/signal.h>
   12: #include <fiwix/sched.h>
   13: #include <fiwix/sleep.h>
   14: #include <fiwix/errno.h>
   15:
   16: #ifdef __DEBUG__
   17: #include <fiwix/stdio.h>
   18: #include <fiwix/process.h>
   19: #endif /*__DEBUG__ */
   20:
   21: int sys_wait4(__pid_t pid, int *status, int options, struct rusage *ru)
   22: {
   23:         struct proc *p;
   24:         int flag, signum, errno;
   25:
   26: #ifdef __DEBUG__
   27:         printk("(pid %d) sys_wait4(%d, status, %d)\n", current->pid, pid, option
s);
   28: #endif /*__DEBUG__ */
   29:
   30:         if(ru) {
   31:                 if((errno = check_user_area(VERIFY_WRITE, ru, sizeof(struct rusa
ge)))) {
   32:                         return errno;
   33:                 }
   34:         }
   35:         while(current->children) {
   36:                 flag = 0;
   37:                 FOR_EACH_PROCESS(p) {
   38:                         if(p->ppid != current->pid) {
   39:                                 continue;
   40:                         }
   41:                         if(pid > 0) {
   42:                                 if(p->pid == pid) {
   43:                                         flag = 1;
   44:                                 }
   45:                         }
   46:                         if(!pid) {
   47:                                 if(p->pgid == current->pgid) {
   48:                                         flag = 1;
   49:                                 }
   50:                         }
   51:                         if(pid < -1) {
   52:                                 if(p->pgid == -pid) {
   53:                                         flag = 1;
   54:                                 }
   55:                         }
   56:                         if(pid == -1) {
   57:                                 flag = 1;
   58:                         }
   59:                         if(flag) {
   60:                                 if(p->state == PROC_STOPPED) {
   61:                                         if(!p->exit_code) {
   62:                                                 continue;
   63:                                         }
   64:                                         if(status) {
   65:                                                 *status = (p->exit_code << 8) |
```

kernel/syscalls/wait4.c


```
0x7F;
   66:                                          }
   67:                                          p->exit_code = 0;
   68:                                          if(ru) {
   69:                                                  get_rusage(p, ru);
   70:                                          }
   71:                                          return p->pid;
   72:                                  }
   73:                                  if(p->state == PROC_ZOMBIE) {
   74:                                          add_rusage(p);
   75:                                          if(status) {
   76:                                                  *status = p->exit_code;
   77:                                          }
   78:                                          if(ru) {
   79:                                                  get_rusage(p, ru);
   80:                                          }
   81:                                          return remove_zombie(p);
   82:                                  }
   83:                          }
   84:                          flag = 0;
   85:                  }
   86:                  if(options & WNOHANG) {
   87:                          if(flag) {
   88:                                  return 0;
   89:                          }
   90:                          break;
   91:                  }
   92:                  if((signum = sleep(&sys_wait4, PROC_INTERRUPTIBLE))) {
   93:                          return signum;
   94:                  }
   95:                  current->sigpending &= SIG_MASK(SIGCHLD);
   96:          }
   97:          return -ECHILD;
   98: }
```

kernel/syscalls/waitpid.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/waitpid.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/syscalls.h>
   10: #include <fiwix/string.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_waitpid(__pid_t pid, int *status, int options)
   18: {
   19: #ifdef __DEBUG__
   20:         printk("(pid %d) sys_waitpid(%d, 0x%08x, %d)\n", current->pid, pid,  *st
atus, options);
   21: #endif /*__DEBUG__ */
   22:         return sys_wait4(pid, status, options, NULL);
   23: }
```

kernel/syscalls/write.c


```
    1: /*
    2:  * fiwix/kernel/syscalls/write.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/fs.h>
    9: #include <fiwix/fcntl.h>
   10: #include <fiwix/errno.h>
   11:
   12: #ifdef __DEBUG__
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/process.h>
   15: #endif /*__DEBUG__ */
   16:
   17: int sys_write(unsigned int ufd, const char *buf, int count)
   18: {
   19:         struct inode *i;
   20:         int errno;
   21:
   22: #ifdef __DEBUG__
   23: /*      printk("(pid %d) sys_write(%d, '%s', %d)\n", current->pid, ufd, buf, cou
nt);*/
   24:         printk("(pid %d) sys_write(%d, 0x%08x, %d) -> ", current->pid, ufd, buf,
 count);
   25: #endif /*__DEBUG__ */
   26:
   27:         CHECK_UFD(ufd);
   28:         if((errno = check_user_area(VERIFY_READ, buf, count))) {
   29:                 return errno;
   30:         }
   31:         if(fd_table[current->fd[ufd]].flags & O_RDONLY) {
   32:                 return -EBADF;
   33:         }
   34:         if(!count) {
   35:                 return 0;
   36:         }
   37:         if(count < 0) {
   38:                 return -EINVAL;
   39:         }
   40:         i = fd_table[current->fd[ufd]].inode;
   41:         if(i->fsop && i->fsop->write) {
   42:                 errno = i->fsop->write(i, &fd_table[current->fd[ufd]], buf, coun
t);
   43: #ifdef __DEBUG__
   44:                 printk("%d\n", errno);
   45: #endif /*__DEBUG__ */
   46:                 return errno;
   47:         }
   48:         return -EINVAL;
   49: }
```

drivers/block/dma.c


```
/*
 * fiwix/drivers/block/dma.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/asm.h>
#include <fiwix/dma.h>
#include <fiwix/string.h>

/*
 *  DMA Channel  Page  Address  Count
 *  --------------------------------
 *  0 (8 bit)     87h       0h     1h
 *  1 (8 bit)     83h       2h     3h
 *  2 (8 bit)     81h       4h     5h
 *  3 (8 bit)     82h       6h     7h
 *  4 (16 bit)    8Fh      C0h    C2h
 *  5 (16 bit)    8Bh      C4h    C6h
 *  6 (16 bit)    89h      C8h    CAh
 *  7 (16 bit)    8Ah      CCh    CEh
 */

#define LOW_BYTE(addr)  (addr & 0x00FF)
#define HIGH_BYTE(addr) ((addr & 0xFF00) >> 8)

unsigned char dma_mask[DMA_CHANNELS] =
        { 0x0A, 0x0A, 0x0A, 0x0A, 0xD4, 0xD4, 0xD4, 0xD4 };
unsigned char dma_mode[DMA_CHANNELS] =
        { 0x0B, 0x0B, 0x0B, 0x0B, 0xD6, 0xD6, 0xD6, 0xD6 };
unsigned char dma_clear[DMA_CHANNELS] =
        { 0x0C, 0x0C, 0x0C, 0x0C, 0xD8, 0xD8, 0xD8, 0xD8 };
unsigned char dma_page[DMA_CHANNELS] =
        { 0x87, 0x83, 0x81, 0x82, 0x8F, 0x8B, 0x89, 0x8A };
unsigned char dma_address[DMA_CHANNELS] =
        { 0x00, 0x02, 0x04, 0x06, 0xC0, 0xC4, 0xC8, 0xCC };
unsigned char dma_count[DMA_CHANNELS] =
        { 0x01, 0x03, 0x05, 0x07, 0xC2, 0xC6, 0xCA, 0xCE };


void start_dma(int channel, void *address, unsigned int count, int mode)
{
        /* setup (mask) the DMA channel */
        outport_b(dma_mask[channel], DMA_MASK_CHANNEL | channel);

        /* clear any data transfers that are currently executing */
        outport_b(dma_clear[channel], 0);

        /* set the specified mode */
        outport_b(dma_mode[channel], mode | channel);

        /* set the offset address */
        outport_b(dma_address[channel], LOW_BYTE((unsigned int)address));
        outport_b(dma_address[channel], HIGH_BYTE((unsigned int)address));

        /* set the physical page */
        outport_b(dma_page[channel], (unsigned int)address >> 16);

        /* the true (internal) length sent to the DMA is actually length + 1 */
        count--;

        /* set the length of the data */
        outport_b(dma_count[channel], LOW_BYTE(count));
        outport_b(dma_count[channel], HIGH_BYTE(count));

        /* clear the mask */
```

drivers/block/dma.c


```
68:         outport_b(dma_mask[channel], DMA_UNMASK_CHANNEL | channel);
69: }
70:
71: int dma_register(int channel, char *dev_name)
72: {
73:         if(dma_resources[channel]) {
74:                 return 1;
75:         }
76:         dma_resources[channel] = dev_name;
77:         return 0;
78: }
79:
80: int dma_unregister(int channel)
81: {
82:         if(!dma_resources[channel]) {
83:                 return 1;
84:         }
85:
86:         dma_resources[channel] = NULL;
87:         return 0;
88: }
89:
90: void dma_init(void)
91: {
92:         memset_b(dma_resources, NULL, sizeof(dma_resources));
93: }
```

drivers/block/floppy.c


```
/*
 * fiwix/drivers/block/floppy.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/asm.h>
#include <fiwix/floppy.h>
#include <fiwix/ioctl.h>
#include <fiwix/devices.h>
#include <fiwix/part.h>
#include <fiwix/fs.h>
#include <fiwix/buffer.h>
#include <fiwix/sleep.h>
#include <fiwix/timer.h>
#include <fiwix/sched.h>
#include <fiwix/errno.h>
#include <fiwix/pic.h>
#include <fiwix/cmos.h>
#include <fiwix/dma.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

#define WAIT_MOTOR_OFF  (3 * HZ)        /* time waiting to turn the motor off */
#define WAIT_FDC        WAIT_MOTOR_OFF

#define INVALID_TRACK   -1

#define DEV_TYPE_SHIFT  2               /* right shift to match with the floppy
                                           type when minor > 3 */

static int need_reset = 0;
static int fdc_wait_interrupt = 0;
static int fdc_timeout = 0;
static unsigned char fdc_results[MAX_FDC_RESULTS];
static struct resource floppy_resource = { NULL, NULL };

static struct fddt fdd_type[] = {
/*
 * R (data rate): 0 = 500Kb/s, 2 = 250Kb/s, 3 = 1Mb/s
 * SPEC(IFY) 0xAF: SRT = 6ms, HUT = 240ms (500Kb/s)
 * SPEC(IFY) 0xD7: SRT = 6ms, HUT = 240ms (250Kb/s)
 * SPEC(IFY) 0xDF: SRT = 3ms, HUT = 240ms (500Kb/s)
 * Head Load Time 0x02: HLT = 4ms (500Kb/s), Non-DMA = 0 (DMA enabled)
 *
 *        SIZE    KB   T   S  H  G_RW  G_FM  R  SPEC  HLT   NAME
 *        ------------------------------------------------------------- */
        {    0,    0,  0,  0, 0, 0x00, 0x00, 0, 0x00, 0x00, NULL           },
        {  720,  360, 40,  9, 2, 0x2A, 0x50, 2, 0xD7, 0x02, "360KB 5.25\"" },
        { 2400, 1200, 80, 15, 2, 0x2A, 0x50, 0, 0xAF, 0x02, "1.2MB 5.25\"" },
        { 1440,  720, 80,  9, 2, 0x1B, 0x54, 2, 0xD7, 0x02, "720KB 3.5\""  },
        { 2880, 1440, 80, 18, 2, 0x1B, 0x54, 0, 0xAF, 0x02, "1.44MB 3.5\"" },
/*      { 5760, 2880, 80, 36, 2, 0x38, 0x53, 3, 0xDF, 0x02, "2.88MB 3.5\"" },*/
};

/* maximum size of a track for floppy types of 1.44MB */
extern char _fdc_transfer_area[BPS * 2 * 18];

struct fdd_status {
        char type;              /* floppy disk drive type */
        char motor;
        char recalibrated;
        char current_track;
};

static struct fdd_status fdd_status[] = {
```

drivers/block/floppy.c


```
  68:         { 0, 0, 0, INVALID_TRACK },
  69:         { 0, 0, 0, INVALID_TRACK },
  70: };
  71:
  72: static unsigned char current_fdd = 0;
  73: static struct fddt *current_fdd_type;
  74: static unsigned int fdd_sizes[256];
  75:
  76: static struct fs_operations fdc_driver_fsop = {
  77:         0,
  78:         0,
  79:
  80:         fdc_open,
  81:         fdc_close,
  82:         NULL,                   /* read */
  83:         NULL,                   /* write */
  84:         fdc_ioctl,
  85:         fdc_lseek,
  86:         NULL,                   /* readdir */
  87:         NULL,                   /* mmap */
  88:         NULL,                   /* select */
  89:
  90:         NULL,                   /* readlink */
  91:         NULL,                   /* followlink */
  92:         NULL,                   /* bmap */
  93:         NULL,                   /* lockup */
  94:         NULL,                   /* rmdir */
  95:         NULL,                   /* link */
  96:         NULL,                   /* unlink */
  97:         NULL,                   /* symlink */
  98:         NULL,                   /* mkdir */
  99:         NULL,                   /* mknod */
 100:         NULL,                   /* truncate */
 101:         NULL,                   /* create */
 102:         NULL,                   /* rename */
 103:
 104:         fdc_read,
 105:         fdc_write,
 106:
 107:         NULL,                   /* read_inode */
 108:         NULL,                   /* write_inode */
 109:         NULL,                   /* ialloc */
 110:         NULL,                   /* ifree */
 111:         NULL,                   /* statfs */
 112:         NULL,                   /* read_superblock */
 113:         NULL,                   /* remount_fs */
 114:         NULL,                   /* write_superblock */
 115:         NULL                    /* release_superblock */
 116: };
 117:
 118: static struct device floppy_device = {
 119:         "floppy",
 120:         FLOPPY_IRQ,
 121:         FDC_MAJOR,
 122:         { 0, 0, 0, 0, 0, 0, 0, 0 },
 123:         BLKSIZE_1K,
 124:         &fdd_sizes,
 125:         &fdc_driver_fsop,
 126: };
 127:
 128: static int fdc_in(void)
 129: {
 130:         int n;
 131:         unsigned char status;
 132:
 133:         if(need_reset) {
 134:                 return -1;
```

drivers/block/floppy.c


```
  135:         }
  136:
  137:         for(n = 0; n < 10000; n++) {
  138:                 status = inport_b(FDC_MSR) & (FDC_RQM | FDC_DIO);
  139:                 if(status == FDC_RQM) {
  140:                         return 0;
  141:                 }
  142:                 if(status == (FDC_RQM | FDC_DIO)) {
  143:                         return inport_b(FDC_DATA);
  144:                 }
  145:         }
  146:         need_reset = 1;
  147:         printk("WARNING: %s(): fd%d: timeout on %s.\n", __FUNCTION__, current_fd
d, floppy_device.name);
  148:         return -1;
  149: }
  150:
  151: static void fdc_out(unsigned char value)
  152: {
  153:         int n;
  154:         unsigned char status;
  155:
  156:         if(need_reset) {
  157:                 return;
  158:         }
  159:
  160:         for(n = 0; n < 10000; n++) {
  161:                 status = inport_b(FDC_MSR) & (FDC_RQM | FDC_DIO);
  162:                 if(status == FDC_RQM) {
  163:                         outport_b(FDC_DATA, value);
  164:                         return;
  165:                 }
  166:         }
  167:
  168:         need_reset = 1;
  169:         printk("WARNING: %s(): fd%d: unable to send byte 0x%x on %s.\n", __FUNCT
ION__, current_fdd, value, floppy_device.name);
  170: }
  171:
  172: static void fdc_get_results(void)
  173: {
  174:         int n;
  175:
  176:         memset_b(fdc_results, 0, sizeof(fdc_results));
  177:         for(n = 0; n < MAX_FDC_RESULTS; n++) {
  178:                 fdc_results[n] = fdc_in();
  179:         }
  180:         return;
  181: }
  182:
  183: static int fdc_motor_on(void)
  184: {
  185:         struct callout_req creq;
  186:         int errno;
  187:
  188:         if(fdd_status[current_fdd].motor) {
  189:                 return 0;
  190:         }
  191:
  192:         /* select floppy disk drive and turn on its motor */
  193:         outport_b(FDC_DOR, (FDC_DRIVE0 << current_fdd) | FDC_DMA_ENABLE | FDC_EN
ABLE | current_fdd);
  194:         fdd_status[current_fdd].motor = 1;
  195:         fdd_status[!current_fdd].motor = 0;
  196:
  197:         /* fixed spin-up time of 500ms for 3.5" and 5.25" */
  198:         creq.fn = fdc_timer;
```

drivers/block/floppy.c


```
  199:          creq.arg = FDC_TR_MOTOR;
  200:          add_callout(&creq, HZ / 2);
  201:          sleep(&fdc_motor_on, PROC_UNINTERRUPTIBLE);
  202:
  203:          errno = 0;
  204:
  205:          /* check for a disk change */
  206:          if(inport_b(FDC_DIR) & 0x80) {
  207:                  errno = 1;
  208:          }
  209:
  210:          return errno;
  211: }
  212:
  213: static void do_motor_off(unsigned int fdd)
  214: {
  215:          outport_b(FDC_DOR, FDC_DMA_ENABLE | FDC_ENABLE | fdd);
  216:          fdd_status[fdd].motor = 0;
  217:          fdd_status[0].motor = fdd_status[1].motor = 0;
  218: }
  219:
  220: static void fdc_motor_off(void)
  221: {
  222:          struct callout_req creq;
  223:
  224:          creq.fn = do_motor_off;
  225:          creq.arg = current_fdd;
  226:          add_callout(&creq, WAIT_FDC);
  227: }
  228:
  229: static void fdc_reset(void)
  230: {
  231:          int n;
  232:          struct callout_req creq;
  233:
  234:          need_reset = 0;
  235:
  236:          fdc_wait_interrupt = FDC_RESET;
  237:          outport_b(FDC_DOR, 0);                  /* enter in reset mode */
  238: /*       outport_b(FDC_DOR, FDC_DMA_ENABLE); */
  239:          for(n = 0; n < 1000; n++) {             /* recovery time */
  240:                  NOP();
  241:          }
  242:          outport_b(FDC_DOR, FDC_DMA_ENABLE | FDC_ENABLE);
  243:
  244:          creq.fn = fdc_timer;
  245:          creq.arg = FDC_TR_DEFAULT;
  246:          add_callout(&creq, WAIT_FDC);
  247:          sleep(&irq_floppy, PROC_UNINTERRUPTIBLE);
  248:          if(fdc_timeout) {
  249:                  need_reset = 1;
  250:                  printk("WARNING: %s(): fd%d: timeout on %s.\n", __FUNCTION__, cu
rrent_fdd, floppy_device.name);
  251:          }
  252:          del_callout(&creq);
  253:
  254:          fdd_status[0].motor = fdd_status[1].motor = 0;
  255:          fdd_status[current_fdd].recalibrated = 0;
  256:
  257:          /* assumes drive polling mode is ON (by default) */
  258:          for(n = 0; n < 4; n++) {
  259:                  fdc_out(FDC_SENSEI);
  260:                  fdc_get_results();
  261:          }
  262:
  263:          /* keeps controller informed on the drive about to use */
  264:          fdc_out(FDC_SPECIFY);
```

drivers/block/floppy.c


```
  265:         fdc_out(current_fdd_type->spec);
  266:         fdc_out(current_fdd_type->hlt);
  267:
  268:         /* set data rate */
  269:         outport_b(FDC_CCR, current_fdd_type->rate);
  270: }
  271:
  272: static int fdc_recalibrate(void)
  273: {
  274:         struct callout_req creq;
  275:
  276:         if(need_reset) {
  277:                 return 1;
  278:         }
  279:
  280:         fdc_wait_interrupt = FDC_RECALIBRATE;
  281:         fdc_motor_on();
  282:         fdc_out(FDC_RECALIBRATE);
  283:         fdc_out(current_fdd);
  284:
  285:         if(need_reset) {
  286:                 return 1;
  287:         }
  288:
  289:         creq.fn = fdc_timer;
  290:         creq.arg = FDC_TR_DEFAULT;
  291:         add_callout(&creq, WAIT_FDC);
  292:         sleep(&irq_floppy, PROC_UNINTERRUPTIBLE);
  293:         if(fdc_timeout) {
  294:                 need_reset = 1;
  295:                 printk("WARNING: %s(): fd%d: timeout on %s.\n", __FUNCTION__, cu
rrent_fdd, floppy_device.name);
  296:                 return 1;
  297:         }
  298:
  299:         del_callout(&creq);
  300:         fdc_out(FDC_SENSEI);
  301:         fdc_get_results();
  302:
  303:         /* PCN must be 0 indicating a successful position to track 0 */
  304:         if((fdc_results[ST0] & (ST0_IC | ST0_SE | ST0_UC | ST0_NR)) != ST0_RECAL
IBRATE || fdc_results[ST_PCN]) {
  305:                 need_reset = 1;
  306:                 printk("WARNING: %s(): fd%d: unable to recalibrate on %s.\n", __
FUNCTION__, current_fdd, floppy_device.name);
  307:                 return 1;
  308:         }
  309:
  310:         fdd_status[current_fdd].current_track = INVALID_TRACK;
  311:         fdd_status[current_fdd].recalibrated = 1;
  312:         fdc_motor_off();
  313:         return 0;
  314: }
  315:
  316: static int fdc_seek(int track, int head)
  317: {
  318:         struct callout_req creq;
  319:
  320:         if(need_reset) {
  321:                 return 1;
  322:         }
  323:
  324:         if(!fdd_status[current_fdd].recalibrated) {
  325:                 if(fdc_recalibrate()) {
  326:                         return 1;
  327:                 }
  328:         }
```

drivers/block/floppy.c


```
  329:
  330:          if(fdd_status[current_fdd].current_track == track) {
  331:                  return 0;
  332:          }
  333:
  334:          fdc_wait_interrupt = FDC_SEEK;
  335:          fdc_motor_on();
  336:          fdc_out(FDC_SEEK);
  337:          fdc_out((head << 2) | current_fdd);
  338:          fdc_out(track);
  339:
  340:          if(need_reset) {
  341:                  return 1;
  342:          }
  343:
  344:          creq.fn = fdc_timer;
  345:          creq.arg = FDC_TR_DEFAULT;
  346:          add_callout(&creq, WAIT_FDC);
  347:          sleep(&irq_floppy, PROC_UNINTERRUPTIBLE);
  348:          if(fdc_timeout) {
  349:                  need_reset = 1;
  350:                  printk("WARNING: %s(): fd%d: timeout on %s.\n", __FUNCTION__, cu
rrent_fdd, floppy_device.name);
  351:                  return 1;
  352:          }
  353:
  354:          del_callout(&creq);
  355:          fdc_out(FDC_SENSEI);
  356:          fdc_get_results();
  357:
  358:          if((fdc_results[ST0] & (ST0_IC | ST0_SE | ST0_UC | ST0_NR)) != ST0_SEEK
|| fdc_results[ST_PCN] != track) {
  359:                  need_reset = 1;
  360:                  printk("WARNING: %s(): fd%d: unable to seek on %s.\n", __FUNCTIO
N__, current_fdd, floppy_device.name);
  361:                  return 1;
  362:          }
  363:
  364:          fdc_motor_off();
  365:          fdd_status[current_fdd].current_track = track;
  366:          return 0;
  367: }
  368:
  369: static int fdc_get_chip(void)
  370: {
  371:          unsigned char version, fifo, id;
  372:
  373:          fdc_out(FDC_VERSION);
  374:          version = fdc_in();
  375:          fdc_out(FDC_LOCK);
  376:          fifo = fdc_in();
  377:          fdc_out(FDC_PARTID);
  378:          id = fdc_in();
  379:
  380:          if(version == 0x80) {
  381:                  if(fifo == 0x80) {
  382:                          printk("(NEC D765/Intel 8272A/compatible)\n");
  383:                          return 0;
  384:                  }
  385:                  if(fifo == 0) {
  386:                          printk("(Intel 82072)\n");
  387:                          return 0;
  388:                  }
  389:          }
  390:
  391:          if(version == 0x81) {
  392:                  printk("(Very Early Intel 82077/compatible)\n");
```

drivers/block/floppy.c


```
  393:                    return 0;
  394:            }
  395:
  396:            if(version == 0x90) {
  397:                    if(fifo == 0x80) {
  398:                            printk("(Old Intel 82077, no FIFO)\n");
  399:                            return 0;
  400:                    }
  401:                    if(fifo == 0) {
  402:                            if(id == 0x80) {
  403:                                    printk("(New Intel 82077)\n");
  404:                                    return 0;
  405:                            }
  406:                            if(id == 0x41) {
  407:                                    printk("(Intel 82078)\n");
  408:                                    return 0;
  409:                            }
  410:                            if(id == 0x73) {
  411:                                    printk("(National Semiconductor PC87306)\n");
  412:                                    return 0;
  413:                            }
  414:                            printk("(Intel 82078 compatible)\n");
  415:                            return 0;
  416:                    }
  417:                    printk("(NEC 72065B)\n");
  418:                    return 0;
  419:            }
  420:
  421:            if(version == 0xA0) {
  422:                    printk("(SMC FDC37c65C+)\n");
  423:                    return 0;
  424:            }
  425:            printk("(unknown controller chip)\n");
  426:            return 1;
  427: }
  428:
  429: static int fdc_block2chs(__blk_t block, int blksize, int *cyl, int *head, int *s
ector)
  430: {
  431:            int spb = blksize / FDC_SECTSIZE;
  432:
  433:            *cyl = (block * spb) / (current_fdd_type->spt * current_fdd_type->heads)
;
  434:            *head = ((block * spb) % (current_fdd_type->spt * current_fdd_type->head
s)) / current_fdd_type->spt;
  435:            *sector = (((block * spb) % (current_fdd_type->spt * current_fdd_type->h
eads)) % current_fdd_type->spt) + 1;
  436:
  437:            if(*cyl >= current_fdd_type->tracks || *head >= current_fdd_type->heads
|| *sector > current_fdd_type->spt) {
  438:                    return 1;
  439:            }
  440:
  441:            return 0;
  442: }
  443:
  444: static void set_current_fdd_type(int minor)
  445: {
  446:            current_fdd = minor & 1;
  447:
  448:            /* minors 0 and 1 are directly assigned */
  449:            if(minor < 2) {
  450:                    current_fdd_type = &fdd_type[(int)fdd_status[current_fdd].type];
  451:            } else {
  452:                    current_fdd_type = &fdd_type[minor >> DEV_TYPE_SHIFT];
  453:            }
  454: }
```

drivers/block/floppy.c


```
  455:
  456: void irq_floppy(void)
  457: {
  458:         if(!fdc_wait_interrupt) {
  459:                 printk("WARNING: %s(): fd%d: unexpected interrupt on %s.\n", __F
UNCTION__, current_fdd, floppy_device.name);
  460:                 need_reset = 1;
  461:         } else {
  462:                 fdc_timeout = fdc_wait_interrupt = 0;
  463:                 wakeup(&irq_floppy);
  464:         }
  465: }
  466:
  467: void fdc_timer(unsigned int reason)
  468: {
  469:         switch(reason) {
  470:                 case FDC_TR_DEFAULT:
  471:                         fdc_timeout = 1;
  472:                         fdc_wait_interrupt = 0;
  473:                         wakeup(&irq_floppy);
  474:                         break;
  475:                 case FDC_TR_MOTOR:
  476:                         wakeup(&fdc_motor_on);
  477:                         break;
  478:         }
  479: }
  480:
  481: int fdc_open(struct inode *i, struct fd *fd_table)
  482: {
  483:         unsigned char minor;
  484:
  485:         minor = MINOR(i->rdev);
  486:         if(!TEST_MINOR(floppy_device.minors, minor)) {
  487:                 return -ENXIO;
  488:         }
  489:
  490:         lock_resource(&floppy_resource);
  491:         set_current_fdd_type(minor);
  492:         unlock_resource(&floppy_resource);
  493:
  494:         return 0;
  495: }
  496:
  497: int fdc_close(struct inode *i, struct fd *fd_table)
  498: {
  499:         unsigned char minor;
  500:
  501:         minor = MINOR(i->rdev);
  502:         if(!TEST_MINOR(floppy_device.minors, minor)) {
  503:                 return -ENXIO;
  504:         }
  505:
  506:         lock_resource(&floppy_resource);
  507:         set_current_fdd_type(minor);
  508:         unlock_resource(&floppy_resource);
  509:
  510:         return 0;
  511: }
  512:
  513: int fdc_read(__dev_t dev, __blk_t block, char *buffer, int blksize)
  514: {
  515:         unsigned char minor;
  516:         unsigned int sectors_read;
  517:         int cyl, head, sector;
  518:         int retries;
  519:         struct callout_req creq;
  520:         struct device *d;
```

drivers/block/floppy.c


```
  521:
  522:          minor = MINOR(dev);
  523:          if(!TEST_MINOR(floppy_device.minors, minor)) {
  524:                  return -ENXIO;
  525:          }
  526:
  527:          if(!blksize) {
  528:                  if(!(d = get_device(BLK_DEV, MAJOR(dev)))) {
  529:                          return -EINVAL;
  530:                  }
  531:                  blksize = d->blksize;
  532:          }
  533:          blksize = blksize ? blksize : BLKSIZE_1K;
  534:
  535:          lock_resource(&floppy_resource);
  536:          set_current_fdd_type(minor);
  537:
  538:          if(fdc_block2chs(block, blksize, &cyl, &head, &sector)) {
  539:                  printk("WARNING: %s(): fd%d: invalid block number %d on %s devic
e %d,%d.\n", __FUNCTION__, current_fdd, block, floppy_device.name, MAJOR(dev), MINOR(de
v));
  540:                  unlock_resource(&floppy_resource);
  541:                  return -EINVAL;
  542:          }
  543:
  544:          for(retries = 0; retries < MAX_FDC_ERR; retries++) {
  545:                  if(need_reset) {
  546:                          fdc_reset();
  547:                  }
  548:                  if(fdc_motor_on()) {
  549:                          printk("%s(): %s disk was changed in device %d,%d!\n", _
_FUNCTION__, floppy_device.name, MAJOR(dev), MINOR(dev));
  550:                          invalidate_buffers(dev);
  551:                          fdd_status[current_fdd].recalibrated = 0;
  552:                  }
  553:
  554:                  if(fdc_seek(cyl, head)) {
  555:                          printk("WARNING: %s(): fd%d: seek error on %s device %d,
%d during read operation.\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(dev)
, MINOR(dev));
  556:                          continue;
  557:                  }
  558:
  559:                  start_dma(FLOPPY_DMA, _fdc_transfer_area, blksize, DMA_MODE_WRIT
E | DMA_MODE_SINGLE);
  560:
  561:                  /* send READ command */
  562:                  fdc_wait_interrupt = FDC_READ;
  563:                  fdc_out(FDC_READ);
  564:                  fdc_out((head << 2) | current_fdd);
  565:                  fdc_out(cyl);
  566:                  fdc_out(head);
  567:                  fdc_out(sector);
  568:                  fdc_out(2);     /* sector size is 512 bytes */
  569:                  fdc_out(current_fdd_type->spt);
  570:                  fdc_out(current_fdd_type->gpl1);
  571:                  fdc_out(0xFF);  /* sector size is 512 bytes */
  572:
  573:                  if(need_reset) {
  574:                          printk("WARNING: %s(): fd%d: needs reset on %s device %d
,%d during read operation.\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(dev
), MINOR(dev));
  575:                          continue;
  576:                  }
  577:                  creq.fn = fdc_timer;
  578:                  creq.arg = FDC_TR_DEFAULT;
  579:                  add_callout(&creq, WAIT_FDC);
```

drivers/block/floppy.c


```
  580:                 sleep(&irq_floppy, PROC_UNINTERRUPTIBLE);
  581:                 if(fdc_timeout) {
  582:                         need_reset = 1;
  583:                         printk("WARNING: %s(): fd%d: timeout on %s device %d,%d.
\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(dev), MINOR(dev));
  584:                         continue;
  585:                 }
  586:                 del_callout(&creq);
  587:                 fdc_get_results();
  588:                 if(fdc_results[ST0] & (ST0_IC | ST0_UC | ST0_NR)) {
  589:                         need_reset = 1;
  590:                         continue;
  591:                 }
  592:                 break;
  593:         }
  594:
  595:         if(retries >= MAX_FDC_ERR) {
  596:                 printk("WARNING: %s(): fd%d: error on %s device %d,%d during rea
d operation,\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(dev), MINOR(dev))
;
  597:                 printk("\tblock=%d, sector=%d, cylinder/head=%d/%d\n", block, se
ctor, cyl, head);
  598:                 unlock_resource(&floppy_resource);
  599:                 fdc_motor_off();
  600:                 return -EIO;
  601:         }
  602:
  603:         fdc_motor_off();
  604:         sectors_read = (fdc_results[ST_CYL] - cyl) * (current_fdd_type->heads *
current_fdd_type->spt);
  605:         sectors_read += (fdc_results[ST_HEAD] - head) * current_fdd_type->spt;
  606:         sectors_read += fdc_results[ST_SECTOR] - sector;
  607:         if(sectors_read * BPS != blksize) {
  608:                 printk("WARNING: %s(): fd%d: read error on %s device %d,%d (%d s
ectors read).\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(dev), MINOR(dev)
, sectors_read);
  609:                 printk("\tblock=%d, sector=%d, cylinder/head=%d/%d\n", block, se
ctor, cyl, head);
  610:                 unlock_resource(&floppy_resource);
  611:                 fdc_motor_off();
  612:                 return -EIO;
  613:         }
  614:
  615:         memcpy_b(buffer, (void *)_fdc_transfer_area, blksize);
  616:
  617:         unlock_resource(&floppy_resource);
  618:         return sectors_read * BPS;
  619: }
  620:
  621: int fdc_write(__dev_t dev, __blk_t block, char *buffer, int blksize)
  622: {
  623:         unsigned char minor;
  624:         unsigned int sectors_written;
  625:         int cyl, head, sector;
  626:         int retries;
  627:         struct callout_req creq;
  628:         struct device *d;
  629:
  630:         minor = MINOR(dev);
  631:         if(!TEST_MINOR(floppy_device.minors, minor)) {
  632:                 return -ENXIO;
  633:         }
  634:
  635:         if(!blksize) {
  636:                 if(!(d = get_device(BLK_DEV, MAJOR(dev)))) {
  637:                         return -EINVAL;
  638:                 }
```

drivers/block/floppy.c


```
  639:                  blksize = d->blksize;
  640:          }
  641:          blksize = blksize ? blksize : BLKSIZE_1K;
  642:
  643:          lock_resource(&floppy_resource);
  644:          set_current_fdd_type(minor);
  645:
  646:          if(fdc_block2chs(block, blksize, &cyl, &head, &sector)) {
  647:                  printk("WARNING: %s(): fd%d: invalid block number %d on %s devic
e %d,%d.\n", __FUNCTION__, current_fdd, block, floppy_device.name, MAJOR(dev), MINOR(de
v));
  648:                  unlock_resource(&floppy_resource);
  649:                  return -EINVAL;
  650:          }
  651:
  652:          for(retries = 0; retries < MAX_FDC_ERR; retries++) {
  653:                  if(need_reset) {
  654:                          fdc_reset();
  655:                  }
  656:                  if(fdc_motor_on()) {
  657:                          printk("%s(): %s disk was changed in device %d,%d!\n", _
_FUNCTION__, floppy_device.name, MAJOR(dev), MINOR(dev));
  658:                          invalidate_buffers(dev);
  659:                          fdd_status[current_fdd].recalibrated = 0;
  660:                  }
  661:
  662:                  if(fdc_seek(cyl, head)) {
  663:                          printk("WARNING: %s(): fd%d: seek error on %s device %d,
%d during write operation.\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(dev
), MINOR(dev));
  664:                          continue;
  665:                  }
  666:
  667:                  start_dma(FLOPPY_DMA, _fdc_transfer_area, blksize, DMA_MODE_READ
 | DMA_MODE_SINGLE);
  668:                  memcpy_b((void *)_fdc_transfer_area, buffer, blksize);
  669:
  670:                  /* send WRITE command */
  671:                  fdc_wait_interrupt = FDC_WRITE;
  672:                  fdc_out(FDC_WRITE);
  673:                  fdc_out((head << 2) | current_fdd);
  674:                  fdc_out(cyl);
  675:                  fdc_out(head);
  676:                  fdc_out(sector);
  677:                  fdc_out(2);     /* sector size is 512 bytes */
  678:                  fdc_out(current_fdd_type->spt);
  679:                  fdc_out(current_fdd_type->gpl1);
  680:                  fdc_out(0xFF);  /* sector size is 512 bytes */
  681:
  682:                  if(need_reset) {
  683:                          printk("WARNING: %s(): fd%d: needs reset on %s device %d
,%d during write operation.\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(de
v), MINOR(dev));
  684:                          continue;
  685:                  }
  686:                  creq.fn = fdc_timer;
  687:                  creq.arg = FDC_TR_DEFAULT;
  688:                  add_callout(&creq, WAIT_FDC);
  689:                  sleep(&irq_floppy, PROC_UNINTERRUPTIBLE);
  690:                  if(fdc_timeout) {
  691:                          need_reset = 1;
  692:                          printk("WARNING: %s(): fd%d: timeout on %s device %d,%d.
\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(dev), MINOR(dev));
  693:                          continue;
  694:                  }
  695:                  del_callout(&creq);
  696:                  fdc_get_results();
```

drivers/block/floppy.c


```
  697:                  if(fdc_results[ST1] & ST1_NW) {
  698:                          unlock_resource(&floppy_resource);
  699:                          fdc_motor_off();
  700:                          return -EROFS;
  701:                  }
  702:                  if(fdc_results[ST0] & (ST0_IC | ST0_UC | ST0_NR)) {
  703:                          need_reset = 1;
  704:                          continue;
  705:                  }
  706:                  break;
  707:          }
  708:
  709:          if(retries >= MAX_FDC_ERR) {
  710:                  printk("WARNING: %s(): fd%d: error on %s device %d,%d during wri
te operation,\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(dev), MINOR(dev)
);
  711:                  printk("\tblock=%d, sector=%d, cylinder/head=%d/%d\n", block, se
ctor, cyl, head);
  712:                  unlock_resource(&floppy_resource);
  713:                  fdc_motor_off();
  714:                  return -EIO;
  715:          }
  716:
  717:          fdc_motor_off();
  718:          sectors_written = (fdc_results[ST_CYL] - cyl) * (current_fdd_type->heads
 * current_fdd_type->spt);
  719:          sectors_written += (fdc_results[ST_HEAD] - head) * current_fdd_type->spt
;
  720:          sectors_written += fdc_results[ST_SECTOR] - sector;
  721:          if(sectors_written * BPS != blksize) {
  722:                  printk("WARNING: %s(): fd%d: write error on %s device %d,%d (%d
sectors written).\n", __FUNCTION__, current_fdd, floppy_device.name, MAJOR(dev), MINOR(
dev), sectors_written);
  723:                  printk("\tblock=%d, sector=%d, cylinder/head=%d/%d\n", block, se
ctor, cyl, head);
  724:                  unlock_resource(&floppy_resource);
  725:                  fdc_motor_off();
  726:                  return -EIO;
  727:          }
  728:
  729:          unlock_resource(&floppy_resource);
  730:          return sectors_written * BPS;
  731: }
  732:
  733: int fdc_ioctl(struct inode *i, int cmd, unsigned long int arg)
  734: {
  735:          unsigned char minor;
  736:          struct hd_geometry *geom;
  737:          int errno;
  738:
  739:          minor = MINOR(i->rdev);
  740:          if(!TEST_MINOR(floppy_device.minors, minor)) {
  741:                  return -ENXIO;
  742:          }
  743:
  744:          lock_resource(&floppy_resource);
  745:          set_current_fdd_type(minor);
  746:          unlock_resource(&floppy_resource);
  747:
  748:          switch(cmd) {
  749:                  case HDIO_GETGEO:
  750:                          if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(struct hd_geometry)))) {
  751:                                  return errno;
  752:                          }
  753:                          geom = (struct hd_geometry *)arg;
  754:                          geom->heads = current_fdd_type->heads;
```

drivers/block/floppy.c


```
  755:                          geom->sectors = current_fdd_type->spt;
  756:                          geom->cylinders = current_fdd_type->tracks;
  757:                          geom->start = 0;
  758:                          break;
  759:                  case BLKRRPART:
  760:                          break;
  761:                  case BLKGETSIZE:
  762:                          if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(unsigned int)))) {
  763:                                  return errno;
  764:                          }
  765:                          *(int *)arg = fdd_sizes[MINOR(i->rdev)] * 2;
  766:                          break;
  767:                  default:
  768:                          return -EINVAL;
  769:          }
  770:          return 0;
  771: }
  772:
  773: int fdc_lseek(struct inode *i, __off_t offset)
  774: {
  775:          unsigned char minor;
  776:
  777:          minor = MINOR(i->rdev);
  778:          if(!TEST_MINOR(floppy_device.minors, minor)) {
  779:                  return -ENXIO;
  780:          }
  781:
  782:          lock_resource(&floppy_resource);
  783:          set_current_fdd_type(minor);
  784:          unlock_resource(&floppy_resource);
  785:
  786:          return offset;
  787: }
  788:
  789: void floppy_init(void)
  790: {
  791:          short int cmosval, master, slave;
  792:
  793:          cmosval = cmos_read(CMOS_FDDTYPE);
  794:          set_current_fdd_type(0);        /* sets /dev/fd0 by default */
  795:
  796:          /* the high nibble describes the 'master' floppy drive */
  797:          master = cmosval >> 4;
  798:
  799:          /*
  800:           * Some BIOS may return the value 0x05 (for 2.88MB floppy type) which is
  801:           * not supported by Fiwix. This prevents from using an unexistent type
  802:           * in the fdd_type structure if this happens.
  803:           */
  804:          if(master > 4) {
  805:                  master = 4;
  806:          }
  807:
  808:          if(master) {
  809:                  if(!register_irq(FLOPPY_IRQ, floppy_device.name, irq_floppy)) {
  810:                          enable_irq(FLOPPY_IRQ);
  811:                  }
  812:                  printk("fd0       0x%04X-0x%04X   %2d    ", FDC_SRA, FDC_CCR, FL
OPPY_IRQ);
  813:                  printk("%s ", fdd_type[master].name);
  814:                  fdd_status[0].type = fdd_status[1].type = master;
  815:                  SET_MINOR(floppy_device.minors, 0);
  816:                  SET_MINOR(floppy_device.minors, 4);
  817:                  SET_MINOR(floppy_device.minors, 8);
  818:                  SET_MINOR(floppy_device.minors, 12);
  819:                  SET_MINOR(floppy_device.minors, 16);
```

drivers/block/floppy.c


```
  820:                  fdd_sizes[0] = fdd_type[master].sizekb;
  821:                  fdd_sizes[4] = fdd_type[1].sizekb;
  822:                  fdd_sizes[8] = fdd_type[2].sizekb;
  823:                  fdd_sizes[12] = fdd_type[3].sizekb;
  824:                  fdd_sizes[16] = fdd_type[4].sizekb;
  825:                  fdc_reset();
  826:                  fdc_get_chip();
  827:          }
  828:
  829:          /* the low nibble is for the 'slave' floppy drive */
  830:          slave = cmosval & 0x0F;
  831:          if(slave) {
  832:                  if(!master) {
  833:                          if(!register_irq(FLOPPY_IRQ, floppy_device.name, irq_flo
ppy)) {
  834:                                  enable_irq(FLOPPY_IRQ);
  835:                          }
  836:                  }
  837:                  printk("fd1       0x%04X-0x%04X   %2d    ", FDC_SRA, FDC_CCR, FL
OPPY_IRQ);
  838:                  printk("%s  ", fdd_type[slave].name);
  839:                  fdd_status[1].type = slave;
  840:                  SET_MINOR(floppy_device.minors, 1);
  841:                  SET_MINOR(floppy_device.minors, 5);
  842:                  SET_MINOR(floppy_device.minors, 9);
  843:                  SET_MINOR(floppy_device.minors, 13);
  844:                  SET_MINOR(floppy_device.minors, 17);
  845:                  fdd_sizes[1] = fdd_type[slave].sizekb;
  846:                  fdd_sizes[5] = fdd_type[1].sizekb;
  847:                  fdd_sizes[9] = fdd_type[2].sizekb;
  848:                  fdd_sizes[13] = fdd_type[3].sizekb;
  849:                  fdd_sizes[17] = fdd_type[4].sizekb;
  850:                  if(!master) {
  851:                          fdc_get_chip();
  852:                  } else {
  853:                          printk("\n");
  854:                  }
  855:          }
  856:
  857:          if(master || slave) {
  858:                  need_reset = 1;
  859:                  dma_init();
  860:                  if(dma_register(FLOPPY_DMA, floppy_device.name)) {
  861:                          printk("WARNING: %s(): fd%d: unable to register DMA chan
nel on %s.\n", __FUNCTION__, current_fdd, floppy_device.name);
  862:                  } else  {
  863:                          if(!register_device(BLK_DEV, &floppy_device)) {
  864:                                  do_motor_off(current_fdd);
  865:                          }
  866:                  }
  867:          }
  868: }
```

drivers/block/ide.c


```
    1: /*
    2:  * fiwix/drivers/block/ide.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/ide.h>
   10: #include <fiwix/ide_hd.h>
   11: #include <fiwix/ide_cd.h>
   12: #include <fiwix/devices.h>
   13: #include <fiwix/sleep.h>
   14: #include <fiwix/timer.h>
   15: #include <fiwix/sched.h>
   16: #include <fiwix/cpu.h>
   17: #include <fiwix/pic.h>
   18: #include <fiwix/fs.h>
   19: #include <fiwix/mm.h>
   20: #include <fiwix/errno.h>
   21: #include <fiwix/stdio.h>
   22: #include <fiwix/string.h>
   23:
   24: int ide0_need_reset = 0;
   25: int ide0_wait_interrupt = 0;
   26: int ide0_timeout = 0;
   27: int ide1_need_reset = 0;
   28: int ide1_wait_interrupt = 0;
   29: int ide1_timeout = 0;
   30:
   31: struct ide ide_table[NR_IDE_CTRLS] = {
   32:         { IDE_PRIMARY, IDE0_BASE, IDE0_CTRL, IDE0_IRQ,
   33:                 {
   34:                         { IDE_MASTER, "hda", IDE0_MAJOR, 0, -1, NULL, NULL, NULL
, NULL, NULL, { NULL }, {{ NULL }} },
   35:                         { IDE_SLAVE, "hdb", IDE0_MAJOR, 0, -1, NULL, NULL, NULL,
 NULL, NULL, { NULL }, {{ NULL }} }
   36:                 }
   37:         },
   38:         { IDE_SECONDARY, IDE1_BASE, IDE1_CTRL, IDE1_IRQ,
   39:                 {
   40:                         { IDE_MASTER, "hdc", IDE1_MAJOR, 0, -1, NULL, NULL, NULL
, NULL, NULL, { NULL }, {{ NULL }} },
   41:                         { IDE_SLAVE, "hdd", IDE1_MAJOR, 0, -1, NULL, NULL, NULL,
 NULL, NULL, { NULL }, {{ NULL }} }
   42:                 }
   43:         }
   44: };
   45:
   46: static char *ide_ctrl_name[] = { "primary", "secondary" };
   47: static char *ide_drv_name[] = { "master", "slave" };
   48:
   49: static unsigned int ide0_sizes[256];
   50: static unsigned int ide1_sizes[256];
   51:
   52: static struct fs_operations ide_driver_fsop = {
   53:         0,
   54:         0,
   55:
   56:         ide_open,
   57:         ide_close,
   58:         NULL,                   /* read */
   59:         NULL,                   /* write */
   60:         ide_ioctl,
   61:         NULL,                   /* lseek */
   62:         NULL,                   /* readdir */
   63:         NULL,                   /* mmap */
```

drivers/block/ide.c


```
   64:         NULL,                   /* select */
   65:
   66:         NULL,                   /* readlink */
   67:         NULL,                   /* followlink */
   68:         NULL,                   /* bmap */
   69:         NULL,                   /* lockup */
   70:         NULL,                   /* rmdir */
   71:         NULL,                   /* link */
   72:         NULL,                   /* unlink */
   73:         NULL,                   /* symlink */
   74:         NULL,                   /* mkdir */
   75:         NULL,                   /* mknod */
   76:         NULL,                   /* truncate */
   77:         NULL,                   /* create */
   78:         NULL,                   /* rename */
   79:
   80:         ide_read,
   81:         ide_write,
   82:
   83:         NULL,                   /* read_inode */
   84:         NULL,                   /* write_inode */
   85:         NULL,                   /* ialloc */
   86:         NULL,                   /* ifree */
   87:         NULL,                   /* statfs */
   88:         NULL,                   /* read_superblock */
   89:         NULL,                   /* remount_fs */
   90:         NULL,                   /* write_superblock */
   91:         NULL                    /* release_superblock */
   92: };
   93:
   94: static struct device ide0_device = {
   95:         "ide0",
   96:         IDE0_IRQ,
   97:         IDE0_MAJOR,
   98:         { 0, 0, 0, 0, 0, 0, 0, 0 },
   99:         0,
  100:         &ide0_sizes,
  101:         &ide_driver_fsop,
  102: };
  103:
  104: static struct device ide1_device = {
  105:         "ide1",
  106:         IDE1_IRQ,
  107:         IDE1_MAJOR,
  108:         { 0, 0, 0, 0, 0, 0, 0, 0 },
  109:         0,
  110:         &ide1_sizes,
  111:         &ide_driver_fsop,
  112: };
  113:
  114: static int ide_identify(struct ide *ide, int drive)
  115: {
  116:         short int status, *buffer;
  117:         struct callout_req creq;
  118:
  119:         if((status = ide_drvsel(ide, drive, IDE_CHS_MODE, 0))) {
  120:                 /* some controllers return 0xFF to indicate a non-drive conditio
n */
  121:                 if(status == 0xFF) {
  122:                         return status;
  123:                 }
  124:                 printk("WARNING: %s(): error on device '%s'.\n", __FUNCTION__, i
de->drive[drive].dev_name);
  125:                 ide_error(ide, status);
  126:                 return status;
  127:         }
  128:
```

drivers/block/ide.c


```
  129:          outport_b(ide->base + IDE_COMMAND, (ide->drive[drive].flags & DEVICE_IS_
ATAPI) ? ATA_IDENTIFY_PACKET : ATA_IDENTIFY);
  130:          if(ide->channel == IDE_PRIMARY) {
  131:                  ide0_wait_interrupt = ide->base;
  132:                  creq.fn = ide0_timer;
  133:                  creq.arg = 0;
  134:                  add_callout(&creq, WAIT_FOR_IDE);
  135:                  sleep(&irq_ide0, PROC_UNINTERRUPTIBLE);
  136:                  if(ide0_timeout) {
  137:                          status = inport_b(ide->base + IDE_STATUS);
  138:                          if((status & (IDE_STAT_RDY | IDE_STAT_DRQ)) != (IDE_STAT
_RDY | IDE_STAT_DRQ)) {
  139:                                  return 1;
  140:                          }
  141:                  }
  142:                  del_callout(&creq);
  143:          }
  144:          if(ide->channel == IDE_SECONDARY) {
  145:                  ide1_wait_interrupt = ide->base;
  146:                  creq.fn = ide1_timer;
  147:                  creq.arg = 0;
  148:                  add_callout(&creq, WAIT_FOR_IDE);
  149:                  sleep(&irq_ide1, PROC_UNINTERRUPTIBLE);
  150:                  if(ide1_timeout) {
  151:                          status = inport_b(ide->base + IDE_STATUS);
  152:                          if((status & (IDE_STAT_RDY | IDE_STAT_DRQ)) != (IDE_STAT
_RDY | IDE_STAT_DRQ)) {
  153:                                  return 1;
  154:                          }
  155:                  }
  156:                  del_callout(&creq);
  157:          }
  158:
  159:          status = inport_b(ide->base + IDE_STATUS);
  160:          if((status & (IDE_STAT_RDY | IDE_STAT_DRQ)) != (IDE_STAT_RDY | IDE_STAT_
DRQ)) {
  161:                  return 1;
  162:          }
  163:
  164:          if(!(buffer = (void *)kmalloc())) {
  165:                  return 1;
  166:          }
  167:
  168:          inport_sw(ide->base + IDE_DATA, (void *)buffer, IDE_HD_SECTSIZE / sizeof
(short int));
  169:          memcpy_b(&ide->drive[drive].ident, (void *)buffer, sizeof(struct ide_drv
_ident));
  170:          kfree((unsigned int)buffer);
  171:
  172:          if(ide->drive[drive].ident.gen_config == IDE_SUPPORTS_CFA) {
  173:                  ide->drive[drive].flags |= DEVICE_IS_CFA;
  174:          }
  175:
  176:          if(ide->drive[drive].flags & DEVICE_IS_ATAPI) {
  177:                  if(((ide->drive[drive].ident.gen_config >> 8) & 0x1F) == ATAPI_I
S_CDROM) {
  178:                          ide->drive[drive].flags |= DEVICE_IS_CDROM;
  179:                  }
  180:                  if(ide->drive[drive].ident.gen_config & 0x3) {
  181:                          printk("WARNING: %s(): packet size must be 16 bytes!\n")
;
  182:                  }
  183:          }
  184:
  185:          /* only bits 0-7 are relevant */
  186:          ide->drive[drive].ident.rw_multiple &= 0xFF;
  187:          return 0;
```

drivers/block/ide.c


```
  188: }
  189:
  190: static void get_device_size(struct ide_drv *drive)
  191: {
  192:         if(drive->ident.capabilities & IDE_HAS_LBA) {
  193:                 drive->lba_cyls = drive->ident.logic_cyls;
  194:                 drive->lba_heads = drive->ident.logic_heads;
  195:                 drive->lba_factor = 0;
  196:
  197:                 while(drive->lba_cyls > 1023) {
  198:                         if(drive->lba_heads < 255) {
  199:                                 drive->lba_cyls >>= 1;
  200:                                 drive->lba_heads <<= 1;
  201:                         } else {
  202:                                 break;
  203:                         }
  204:                         drive->lba_factor++;
  205:                 }
  206:                 drive->nr_sects = drive->ident.tot_sectors | (drive->ident.tot_s
ectors2 << 16);
  207:         }
  208:
  209:         /* some old disk drives (ATA or ATA2) don't specify total sectors */
  210:         if(!(drive->ident.capabilities & IDE_HAS_LBA)) {
  211:                 if(drive->nr_sects == 0) {
  212:                         drive->nr_sects = drive->ident.logic_cyls * drive->ident
.logic_heads * drive->ident.logic_spt;
  213:                 }
  214:         }
  215:
  216: }
  217:
  218: static int get_udma(struct ide *ide, int drive)
  219: {
  220:         int udma;
  221:
  222:         if(ide->drive[drive].ident.fields_validity & IDE_HAS_UDMA) {
  223:                 if((ide->drive[drive].ident.ultradma >> 13) & 1) {
  224:                         udma = 5;
  225:                 } else if((ide->drive[drive].ident.ultradma >> 12) & 1) {
  226:                         udma = 4;
  227:                 } else if((ide->drive[drive].ident.ultradma >> 11) & 1) {
  228:                         udma = 3;
  229:                 } else if((ide->drive[drive].ident.ultradma >> 10) & 1) {
  230:                         udma = 2;
  231:                 } else if((ide->drive[drive].ident.ultradma >> 9) & 1) {
  232:                         udma = 1;
  233:                 } else {
  234:                         udma = 0;
  235:                 }
  236:         } else {
  237:                 udma = -1;
  238:         }
  239:         return udma;
  240: }
  241:
  242: static void ide_results(struct ide *ide, int drive)
  243: {
  244:         unsigned int cyl, hds, sect;
  245:         __loff_t capacity;
  246:         int udma;
  247:         int udma_speed[] = { 16, 25, 33, 44, 66, 100 };
  248:
  249:         cyl = ide->drive[drive].ident.logic_cyls;
  250:         hds = ide->drive[drive].ident.logic_heads;
  251:         sect = ide->drive[drive].ident.logic_spt;
  252:
```

drivers/block/ide.c


```
  253:          udma = get_udma(ide, drive);
  254:          /*
  255:           * After knowing if the device is UDMA capable we could choose between
  256:           * the PIO transfer mode or the UDMA transfer mode.
  257:           * FIXME: Currently only PIO mode is supported.
  258:           */
  259:
  260:          capacity = (__loff_t)ide->drive[drive].nr_sects * BPS;
  261:          capacity = capacity / 1024 / 1024;
  262:
  263:          printk("%s       0x%04X-0x%04X   %2d    ", ide->drive[drive].dev_name, i
de->base, ide->base + IDE_BASE_LEN, ide->irq);
  264:          swap_asc_word(ide->drive[drive].ident.model_number, 40);
  265:          printk("%s %s ", ide_ctrl_name[ide->channel], ide_drv_name[ide->drive[dr
ive].drive]);
  266:
  267:          if(!(ide->drive[drive].flags & DEVICE_IS_ATAPI)) {
  268:                  printk("ATA");
  269:          } else {
  270:                  printk("ATAPI");
  271:          }
  272:
  273:          if(ide->drive[drive].flags & DEVICE_IS_CFA) {
  274:                  printk(" CFA");
  275:          }
  276:
  277:          if(ide->drive[drive].flags & DEVICE_IS_DISK) {
  278:                  printk(" DISK drive %dMB\n", (unsigned int)capacity);
  279:                  printk("                                model=%s\n", ide->drive[
drive].ident.model_number);
  280:                  if(ide->drive[drive].nr_sects < IDE_MIN_LBA) {
  281:                          printk("                                CHS=%d/%d/%d", c
yl, hds, sect);
  282:                  } else {
  283:                          ide->drive[drive].flags |= DEVICE_REQUIRES_LBA;
  284:                          printk("                                sectors=%d", ide
->drive[drive].nr_sects);
  285:                  }
  286:                  printk(" cache=%dKB", ide->drive[drive].ident.buffer_cache >> 1)
;
  287:          }
  288:
  289:          if(ide->drive[drive].flags & DEVICE_IS_CDROM) {
  290:                  printk(" CDROM drive\n");
  291:                  printk("                                model=%s\n", ide->drive[
drive].ident.model_number);
  292:                  printk("                                cache=%dKB", ide->drive[
drive].ident.buffer_cache >> 1);
  293:          }
  294:
  295:          if(udma >= 0) {
  296:                  printk(" UDMA%d(%d)", udma, udma_speed[udma]);
  297:          }
  298:          if(ide->drive[drive].ident.capabilities & IDE_HAS_LBA) {
  299:                  ide->drive[drive].flags |= DEVICE_REQUIRES_LBA;
  300:                  printk(" LBA");
  301:          }
  302:
  303:          printk("\n");
  304:
  305:          if(ide->drive[drive].ident.rw_multiple > 1) {
  306:                  ide->drive[drive].flags |= DEVICE_HAS_RW_MULTIPLE;
  307:          }
  308:
  309:          /*
  310:          printk("\n");
  311:          printk("%s -> %s\n", ide->drive[drive].dev_name, ide->drive[drive].flags
```

drivers/block/ide.c


```
 & DEVICE_IS_ATAPI ? "ATAPI" : "ATA");
  312:          printk("general conf  = %d (%b) (0x%x)\n", ide->drive[drive].ident.gen_c
onfig, ide->drive[drive].ident.gen_config, ide->drive[drive].ident.gen_config);
  313:          printk("logic_cyls    = %d (%b)\n", ide->drive[drive].ident.logic_cyls,
ide->drive[drive].ident.logic_cyls);
  314:          printk("reserved2     = %d (%b)\n", ide->drive[drive].ident.reserved2, i
de->drive[drive].ident.reserved2);
  315:          printk("logic_heads   = %d (%b)\n", ide->drive[drive].ident.logic_heads,
 ide->drive[drive].ident.logic_heads);
  316:          printk("retired4      = %d (%b)\n", ide->drive[drive].ident.retired4, id
e->drive[drive].ident.retired4);
  317:          printk("retired5      = %d (%b)\n", ide->drive[drive].ident.retired5, id
e->drive[drive].ident.retired5);
  318:          printk("logic_spt     = %d (%b)\n", ide->drive[drive].ident.logic_spt, i
de->drive[drive].ident.logic_spt);
  319:          printk("retired7      = %d (%b)\n", ide->drive[drive].ident.retired7, id
e->drive[drive].ident.retired7);
  320:          printk("retired8      = %d (%b)\n", ide->drive[drive].ident.retired8, id
e->drive[drive].ident.retired8);
  321:          printk("retired9      = %d (%b)\n", ide->drive[drive].ident.retired9, id
e->drive[drive].ident.retired9);
  322:          printk("serial number = '%s'\n", ide->drive[drive].ident.serial_number);
  323:          printk("vendor spec20 = %d (%b)\n", ide->drive[drive].ident.vendor_spec2
0, ide->drive[drive].ident.vendor_spec20);
  324:          printk("buffer cache  = %d (%b)\n", ide->drive[drive].ident.buffer_cache
, ide->drive[drive].ident.buffer_cache);
  325:          printk("vendor spec22 = %d (%b)\n", ide->drive[drive].ident.vendor_spec2
2, ide->drive[drive].ident.vendor_spec22);
  326:          printk("firmware rev  = '%s'\n", ide->drive[drive].ident.firmware_rev);
  327:          printk("model number  = '%s'\n", ide->drive[drive].ident.model_number);
  328:          printk("rw multiple   = %d (%b)\n", ide->drive[drive].ident.rw_multiple,
 ide->drive[drive].ident.rw_multiple);
  329:          printk("reserved48    = %d (%b)\n", ide->drive[drive].ident.reserved48,
ide->drive[drive].ident.reserved48);
  330:          printk("capabilities  = %d (%b)\n", ide->drive[drive].ident.capabilities
, ide->drive[drive].ident.capabilities);
  331:          printk("reserved50    = %d (%b)\n", ide->drive[drive].ident.reserved50,
ide->drive[drive].ident.reserved50);
  332:          printk("pio mode      = %d (%b)\n", ide->drive[drive].ident.pio_mode, id
e->drive[drive].ident.pio_mode);
  333:          printk("dma mode      = %d (%b)\n", ide->drive[drive].ident.dma_mode, id
e->drive[drive].ident.dma_mode);
  334:          printk("fields validi = %d (%b)\n", ide->drive[drive].ident.fields_valid
ity, ide->drive[drive].ident.fields_validity);
  335:          printk("cur log cyls  = %d (%b)\n", ide->drive[drive].ident.cur_log_cyls
, ide->drive[drive].ident.cur_log_cyls);
  336:          printk("cur log heads = %d (%b)\n", ide->drive[drive].ident.cur_log_head
s, ide->drive[drive].ident.cur_log_heads);
  337:          printk("cur log spt   = %d (%b)\n", ide->drive[drive].ident.cur_log_spt,
 ide->drive[drive].ident.cur_log_spt);
  338:          printk("cur capacity  = %d (%b)\n", ide->drive[drive].ident.cur_capacity
 | (ide->drive[drive].ident.cur_capacity2 << 16), ide->drive[drive].ident.cur_capacity
| (ide->drive[drive].ident.cur_capacity2 << 16));
  339:          printk("mult sect set = %d (%b)\n", ide->drive[drive].ident.mult_sect_se
t, ide->drive[drive].ident.mult_sect_set);
  340:          printk("tot sectors   = %d (%b)\n", ide->drive[drive].ident.tot_sectors
| (ide->drive[drive].ident.tot_sectors2 << 16), ide->drive[drive].ident.tot_sectors | (
ide->drive[drive].ident.tot_sectors2 << 16));
  341:          printk("singleword dma= %d (%b)\n", ide->drive[drive].ident.singleword_d
ma, ide->drive[drive].ident.singleword_dma);
  342:          printk("multiword dma = %d (%b)\n", ide->drive[drive].ident.multiword_dm
a, ide->drive[drive].ident.multiword_dma);
  343:          printk("adv pio modes = %d (%b)\n", ide->drive[drive].ident.adv_pio_mode
s, ide->drive[drive].ident.adv_pio_modes);
  344:          printk("min multiword = %d (%b)\n", ide->drive[drive].ident.min_multiwor
d, ide->drive[drive].ident.min_multiword);
  345:          printk("rec multiword = %d (%b)\n", ide->drive[drive].ident.rec_multiwor
```

drivers/block/ide.c


```
d, ide->drive[drive].ident.rec_multiword);
  346:          printk("min pio wo fc = %d (%b)\n", ide->drive[drive].ident.min_pio_wo_f
c, ide->drive[drive].ident.min_pio_wo_fc);
  347:          printk("min pio w fc  = %d (%b)\n", ide->drive[drive].ident.min_pio_w_fc
, ide->drive[drive].ident.min_pio_w_fc);
  348:          printk("reserved69    = %d (%b)\n", ide->drive[drive].ident.reserved69,
ide->drive[drive].ident.reserved69);
  349:          printk("reserved70    = %d (%b)\n", ide->drive[drive].ident.reserved70,
ide->drive[drive].ident.reserved70);
  350:          printk("reserved71    = %d (%b)\n", ide->drive[drive].ident.reserved71,
ide->drive[drive].ident.reserved71);
  351:          printk("reserved72    = %d (%b)\n", ide->drive[drive].ident.reserved72,
ide->drive[drive].ident.reserved72);
  352:          printk("reserved73    = %d (%b)\n", ide->drive[drive].ident.reserved73,
ide->drive[drive].ident.reserved73);
  353:          printk("reserved74    = %d (%b)\n", ide->drive[drive].ident.reserved74,
ide->drive[drive].ident.reserved74);
  354:          printk("queue depth   = %d (%b)\n", ide->drive[drive].ident.queue_depth,
 ide->drive[drive].ident.queue_depth);
  355:          printk("reserved76    = %d (%b)\n", ide->drive[drive].ident.reserved76,
ide->drive[drive].ident.reserved76);
  356:          printk("reserved77    = %d (%b)\n", ide->drive[drive].ident.reserved77,
ide->drive[drive].ident.reserved77);
  357:          printk("reserved78    = %d (%b)\n", ide->drive[drive].ident.reserved78,
ide->drive[drive].ident.reserved78);
  358:          printk("reserved79    = %d (%b)\n", ide->drive[drive].ident.reserved79,
ide->drive[drive].ident.reserved79);
  359:          printk("major version = %d (%b)\n", ide->drive[drive].ident.majorver, id
e->drive[drive].ident.majorver);
  360:          printk("minor version = %d (%b)\n", ide->drive[drive].ident.minorver, id
e->drive[drive].ident.minorver);
  361:          printk("cmdset1       = %d (%b)\n", ide->drive[drive].ident.cmdset1, ide
->drive[drive].ident.cmdset1);
  362:          printk("cmdset2       = %d (%b)\n", ide->drive[drive].ident.cmdset2, ide
->drive[drive].ident.cmdset2);
  363:          printk("cmdsf ext     = %d (%b)\n", ide->drive[drive].ident.cmdsf_ext, i
de->drive[drive].ident.cmdsf_ext);
  364:          printk("cmdsf enable1 = %d (%b)\n", ide->drive[drive].ident.cmdsf_enable
1, ide->drive[drive].ident.cmdsf_enable1);
  365:          printk("cmdsf enable2 = %d (%b)\n", ide->drive[drive].ident.cmdsf_enable
2, ide->drive[drive].ident.cmdsf_enable2);
  366:          printk("cmdsf default = %d (%b)\n", ide->drive[drive].ident.cmdsf_defaul
t, ide->drive[drive].ident.cmdsf_default);
  367:          printk("ultra dma     = %d (%b)\n", ide->drive[drive].ident.ultradma, id
e->drive[drive].ident.ultradma);
  368:          printk("reserved89    = %d (%b)\n", ide->drive[drive].ident.reserved89,
ide->drive[drive].ident.reserved89);
  369:          printk("reserved90    = %d (%b)\n", ide->drive[drive].ident.reserved90,
ide->drive[drive].ident.reserved90);
  370:          printk("current apm   = %d (%b)\n", ide->drive[drive].ident.curapm, ide-
>drive[drive].ident.curapm);
  371:          */
  372: }
  373:
  374: void irq_ide0(void)
  375: {
  376:          if(!ide0_wait_interrupt) {
  377:                  printk("WARNING: %s(): unexpected interrupt!\n", __FUNCTION__);
  378:                  ide0_need_reset = 1;
  379:          } else {
  380:                  ide0_timeout = ide0_wait_interrupt = 0;
  381:                  wakeup(&irq_ide0);
  382:          }
  383: }
  384:
  385: void irq_ide1(void)
  386: {
```

drivers/block/ide.c


```
387:         if(!ide1_wait_interrupt) {
388:                 printk("WARNING: %s(): unexpected interrupt!\n", __FUNCTION__);
389:                 ide1_need_reset = 1;
390:         } else {
391:                 ide1_timeout = ide1_wait_interrupt = 0;
392:                 wakeup(&irq_ide1);
393:         }
394: }
395:
396: void ide0_timer(unsigned int arg)
397: {
398:         ide0_timeout = 1;
399:         ide0_wait_interrupt = 0;
400:         wakeup(&irq_ide0);
401: }
402:
403: void ide1_timer(unsigned int arg)
404: {
405:         ide1_timeout = 1;
406:         ide1_wait_interrupt = 0;
407:         wakeup(&irq_ide1);
408: }
409:
410: void ide_error(struct ide *ide, int status)
411: {
412:         int error;
413:
414:         if(status & IDE_STAT_ERR) {
415:                 error = inport_b(ide->base + IDE_ERROR);
416:                 if(error) {
417:                         printk("error=0x%x [", error);
418:                 }
419:                 if(error & IDE_ERR_AMNF) {
420:                         printk("address mark not found, ");
421:                 }
422:                 if(error & IDE_ERR_TK0NF) {
423:                         printk("track 0 not found (no media) or media error, ");
424:                 }
425:                 if(error & IDE_ERR_ABRT) {
426:                         printk("command aborted, ");
427:                 }
428:                 if(error & IDE_ERR_MCR) {
429:                         printk("media change requested, ");
430:                 }
431:                 if(error & IDE_ERR_IDNF) {
432:                         printk("id mark not found, ");
433:                 }
434:                 if(error & IDE_ERR_MC) {
435:                         printk("media changer, ");
436:                 }
437:                 if(error & IDE_ERR_UNC) {
438:                         printk("uncorrectable data, ");
439:                 }
440:                 if(error & IDE_ERR_BBK) {
441:                         printk("bad block, ");
442:                 }
443:                 printk("]");
444:         }
445:         if(status & IDE_STAT_DWF) {
446:                 printk("device fault, ");
447:         }
448:         if(status & IDE_STAT_BSY) {
449:                 printk("device busy, ");
450:         }
451:         printk("\n");
452: }
453:
```

drivers/block/ide.c


```
454: void ide_delay(void)
455: {
456:         int n;
457:
458:         for(n = 0; n < 10000; n++) {
459:                 NOP();
460:         }
461: }
462:
463: void ide_wait400ns(struct ide *ide)
464: {
465:         int n;
466:
467:         /* wait 400ns */
468:         for(n = 0; n < 4; n++) {
469:                 inport_b(ide->ctrl + IDE_ALT_STATUS);
470:         }
471: }
472:
473: int ide_ready(struct ide *ide)
474: {
475:         int n, retries, status;
476:
477:         SET_IDE_RDY_RETR(retries);
478:         for(n = 0; n < retries; n++) {
479:                 status = inport_b(ide->ctrl + IDE_ALT_STATUS);
480:                 if(!(status & IDE_STAT_BSY)) {
481:                         return 0;
482:                 }
483:                 ide_delay();
484:         }
485:
486:         inport_b(ide->base + IDE_STATUS);       /* clear any pending interrupt *
/
487:         return status;
488: }
489:
490: int ide_drvsel(struct ide *ide, int drive, int mode, unsigned char lba24_head)
491: {
492:         int n;
493:         int status;
494:
495:         for(n = 0; n < MAX_IDE_ERR; n++) {
496:                 if((status = ide_ready(ide))) {
497:                         continue;
498:                 }
499:                 break;
500:         }
501:         if(status) {
502:                 return status;
503:         }
504:
505:         outport_b(ide->base + IDE_DRVHD, (mode + (drive << 4)) | lba24_head);
506:         ide_wait400ns(ide);
507:
508:         for(n = 0; n < MAX_IDE_ERR; n++) {
509:                 if((status = ide_ready(ide))) {
510:                         continue;
511:                 }
512:                 break;
513:         }
514:         return status;
515: }
516:
517: int ide_softreset(struct ide *ide)
518: {
519:         int error;
```

drivers/block/ide.c


```
  520:
  521:          error = 0;
  522:
  523:          outport_b(ide->base + IDE_DRVHD, IDE_CHS_MODE);
  524:          ide_delay();
  525:
  526:          outport_b(ide->ctrl + IDE_DEV_CTRL, IDE_DEVCTR_SRST | IDE_DEVCTR_NIEN);
  527:          ide_delay();
  528:          outport_b(ide->ctrl + IDE_DEV_CTRL, 0);
  529:          ide_delay();
  530:
  531:          outport_b(ide->base + IDE_DRVHD, IDE_CHS_MODE);
  532:          ide_delay();
  533:          if(ide_ready(ide)) {
  534:                  printk("WARNING: %s(): reset error on IDE(%d:0).\n", __FUNCTION_
_, ide->channel);
  535:                  error = 1;
  536:          } else {
  537:                  /* device is disk by default */
  538:                  ide->drive[IDE_MASTER].flags |= DEVICE_IS_DISK;
  539:
  540:                  /* check if it's an ATAPI device */
  541:                  if(inport_b(ide->base + IDE_SECCNT) == 1 && inport_b(ide->base +
 IDE_SECNUM) == 1) {
  542:                          if(inport_b(ide->base + IDE_LCYL) == 0x14 && inport_b(id
e->base + IDE_HCYL) == 0xEB) {
  543:                                  ide->drive[IDE_MASTER].flags &= ~DEVICE_IS_DISK;
  544:                                  ide->drive[IDE_MASTER].flags |= DEVICE_IS_ATAPI;
  545:                          }
  546:                  }
  547:          }
  548:
  549:          outport_b(ide->base + IDE_DRVHD, IDE_CHS_MODE + (1 << 4));
  550:          ide_delay();
  551:          if(ide_ready(ide)) {
  552:                  printk("WARNING: %s(): reset error on IDE(%d:1).\n", __FUNCTION_
_, ide->channel);
  553:                  outport_b(ide->base + IDE_DRVHD, IDE_CHS_MODE);
  554:                  ide_delay();
  555:                  ide_ready(ide);
  556:                  error |= (1 << 4);
  557:          }
  558:
  559:          outport_b(ide->ctrl + IDE_DEV_CTRL, 0);
  560:          ide_delay();
  561:          if(error > 1) {
  562:                  return error;
  563:          }
  564:
  565:          /* device is disk by default */
  566:          ide->drive[IDE_SLAVE].flags |= DEVICE_IS_DISK;
  567:
  568:          /* check if it's an ATAPI device */
  569:          if(inport_b(ide->base + IDE_SECCNT) == 1 && inport_b(ide->base + IDE_SEC
NUM) == 1) {
  570:                  if(inport_b(ide->base + IDE_LCYL) == 0x14 && inport_b(ide->base
+ IDE_HCYL) == 0xEB) {
  571:                          ide->drive[IDE_SLAVE].flags &= ~DEVICE_IS_DISK;
  572:                          ide->drive[IDE_SLAVE].flags |= DEVICE_IS_ATAPI;
  573:                  }
  574:          }
  575:
  576:          return error;
  577: }
  578:
  579: struct ide * get_ide_controller(__dev_t dev)
  580: {
```

drivers/block/ide.c


```
581:         int controller;
582:
583:         if(MAJOR(dev) == IDE0_MAJOR) {
584:                 controller = IDE_PRIMARY;
585:         } else {
586:                 if(MAJOR(dev) == IDE1_MAJOR) {
587:                         controller = IDE_SECONDARY;
588:                 } else {
589:                         return NULL;
590:                 }
591:         }
592:         return &ide_table[controller];
593: }
594:
595: int get_ide_drive(__dev_t dev)
596: {
597:         int drive;
598:
599:         drive = MINOR(dev);
600:         if(drive) {
601:                 if(drive & (1 << IDE_SLAVE_MSF)) {
602:                         drive = IDE_SLAVE;
603:                 } else {
604:                         drive = IDE_MASTER;
605:                 }
606:         }
607:         return drive;
608: }
609:
610: int ide_open(struct inode *i, struct fd *fd_table)
611: {
612:         int drive;
613:         struct ide *ide;
614:         struct device *d;
615:
616:         if(!(ide = get_ide_controller(i->rdev))) {
617:                 return -EINVAL;
618:         }
619:
620:         if(!(d = get_device(BLK_DEV, MAJOR(i->rdev)))) {
621:                 return -ENXIO;
622:         }
623:         if(!TEST_MINOR(d->minors, MINOR(i->rdev))) {
624:                 return -ENXIO;
625:         }
626:
627:         drive = get_ide_drive(i->rdev);
628:         if(ide->drive[drive].fsop && ide->drive[drive].fsop->open) {
629:                 return ide->drive[drive].fsop->open(i, fd_table);
630:         }
631:         return -EINVAL;
632: }
633:
634: int ide_close(struct inode *i, struct fd *fd_table)
635: {
636:         int drive;
637:         struct ide *ide;
638:         struct device *d;
639:
640:         if(!(ide = get_ide_controller(i->rdev))) {
641:                 return -EINVAL;
642:         }
643:
644:         if(!(d = get_device(BLK_DEV, MAJOR(i->rdev)))) {
645:                 return -ENXIO;
646:         }
647:         if(!TEST_MINOR(d->minors, MINOR(i->rdev))) {
```

drivers/block/ide.c


```
  648:                  return -ENXIO;
  649:          }
  650:
  651:          drive = get_ide_drive(i->rdev);
  652:          if(ide->drive[drive].fsop && ide->drive[drive].fsop->close) {
  653:                  return ide->drive[drive].fsop->close(i, fd_table);
  654:          }
  655:          return -EINVAL;
  656: }
  657:
  658: int ide_read(__dev_t dev, __blk_t block, char *buffer, int blksize)
  659: {
  660:          int drive;
  661:          struct ide *ide;
  662:          struct device *d;
  663:
  664:          if(!(ide = get_ide_controller(dev))) {
  665:                  printk("%s(): no ide controller!\n", __FUNCTION__);
  666:                  return -EINVAL;
  667:          }
  668:
  669:          if(!(d = get_device(BLK_DEV, MAJOR(dev)))) {
  670:                  return -ENXIO;
  671:          }
  672:          if(!TEST_MINOR(d->minors, MINOR(dev))) {
  673:                  return -ENXIO;
  674:          }
  675:
  676:          drive = get_ide_drive(dev);
  677:          if(ide->drive[drive].fsop && ide->drive[drive].fsop->read_block) {
  678:                  return ide->drive[drive].fsop->read_block(dev, block, buffer, bl
ksize);
  679:          }
  680:          printk("WARNING: %s(): device %d,%d does not have the read_block() metho
d!\n", __FUNCTION__, MAJOR(dev), MINOR(dev));
  681:          return -EINVAL;
  682: }
  683:
  684: int ide_write(__dev_t dev, __blk_t block, char *buffer, int blksize)
  685: {
  686:          int drive;
  687:          struct ide *ide;
  688:          struct device *d;
  689:
  690:          if(!(ide = get_ide_controller(dev))) {
  691:                  printk("%s(): no ide controller!\n", __FUNCTION__);
  692:                  return -EINVAL;
  693:          }
  694:
  695:          if(!(d = get_device(BLK_DEV, MAJOR(dev)))) {
  696:                  return -ENXIO;
  697:          }
  698:          if(!TEST_MINOR(d->minors, MINOR(dev))) {
  699:                  return -ENXIO;
  700:          }
  701:
  702:          drive = get_ide_drive(dev);
  703:          if(ide->drive[drive].fsop && ide->drive[drive].fsop->write_block) {
  704:                  return ide->drive[drive].fsop->write_block(dev, block, buffer, b
lksize);
  705:          }
  706:          printk("WARNING: %s(): device %d,%d does not have the write_block() meth
od!\n", __FUNCTION__, MAJOR(dev), MINOR(dev));
  707:          return -EINVAL;
  708: }
  709:
  710: int ide_ioctl(struct inode *i, int cmd, unsigned long int arg)
```

drivers/block/ide.c


```
711: {
712:         int drive;
713:         struct ide *ide;
714:         struct device *d;
715:
716:         if(!(ide = get_ide_controller(i->rdev))) {
717:                 return -EINVAL;
718:         }
719:
720:         if(!(d = get_device(BLK_DEV, MAJOR(i->rdev)))) {
721:                 return -ENXIO;
722:         }
723:         if(!TEST_MINOR(d->minors, MINOR(i->rdev))) {
724:                 return -ENXIO;
725:         }
726:
727:         drive = get_ide_drive(i->rdev);
728:         if(ide->drive[drive].fsop && ide->drive[drive].fsop->ioctl) {
729:                 return ide->drive[drive].fsop->ioctl(i, cmd, arg);
730:         }
731:         return -EINVAL;
732: }
733:
734: void ide_init(void)
735: {
736:         int devices, errno;
737:         struct ide *ide;
738:
739:         if(!register_irq(IDE0_IRQ, ide0_device.name, irq_ide0)) {
740:                 enable_irq(IDE0_IRQ);
741:         }
742:         devices = 0;
743:
744:         ide = &ide_table[IDE_PRIMARY];
745:         errno = ide_softreset(ide);
746:         if(!(errno & 1)) {
747:                 if(!(ide_identify(ide, IDE_MASTER))) {
748:                         get_device_size(&ide->drive[IDE_MASTER]);
749:                         ide_results(ide, IDE_MASTER);
750:                         register_device(BLK_DEV, &ide0_device);
751:                         if(ide->drive[IDE_MASTER].flags & DEVICE_IS_DISK) {
752:                                 if(!ide_hd_init(ide, IDE_MASTER)) {
753:                                         devices++;
754:                                 }
755:                         }
756:                         if(ide->drive[IDE_MASTER].flags & DEVICE_IS_CDROM) {
757:                                 if(!ide_cd_init(ide, IDE_MASTER)) {
758:                                         devices++;
759:                                 }
760:                         }
761:                 }
762:         }
763:         if(!(errno & 0x10)) {
764:                 if(!(ide_identify(ide, IDE_SLAVE))) {
765:                         get_device_size(&ide->drive[IDE_SLAVE]);
766:                         ide_results(ide, IDE_SLAVE);
767:                         if(!devices) {
768:                                 register_device(BLK_DEV, &ide0_device);
769:                         }
770:                         if(ide->drive[IDE_SLAVE].flags & DEVICE_IS_DISK) {
771:                                 if(!ide_hd_init(ide, IDE_SLAVE)) {
772:                                         devices++;
773:                                 }
774:                         }
775:                         if(ide->drive[IDE_SLAVE].flags & DEVICE_IS_CDROM) {
776:                                 if(!ide_cd_init(ide, IDE_SLAVE)) {
777:                                         devices++;
```

drivers/block/ide.c


```
778:                                    }
779:                            }
780:                    }
781:            }
782:            if(!devices) {
783:                    disable_irq(IDE0_IRQ);
784:                    unregister_irq(IDE0_IRQ);
785:            }
786:
787:            if(!register_irq(IDE1_IRQ, ide1_device.name, irq_ide1)) {
788:                    enable_irq(IDE1_IRQ);
789:            }
790:            devices = 0;
791:            ide = &ide_table[IDE_SECONDARY];
792:            errno = ide_softreset(ide);
793:            if(!(errno & 1)) {
794:                    if(!(ide_identify(ide, IDE_MASTER))) {
795:                            get_device_size(&ide->drive[IDE_MASTER]);
796:                            ide_results(ide, IDE_MASTER);
797:                            register_device(BLK_DEV, &ide1_device);
798:                            if(ide->drive[IDE_MASTER].flags & DEVICE_IS_DISK) {
799:                                    if(!ide_hd_init(ide, IDE_MASTER)) {
800:                                            devices++;
801:                                    }
802:                            }
803:                            if(ide->drive[IDE_MASTER].flags & DEVICE_IS_CDROM) {
804:                                    if(!ide_cd_init(ide, IDE_MASTER)) {
805:                                            devices++;
806:                                    }
807:                            }
808:                    }
809:            }
810:            if(!(errno & 0x10)) {
811:                    if(!(ide_identify(ide, IDE_SLAVE))) {
812:                            get_device_size(&ide->drive[IDE_SLAVE]);
813:                            ide_results(ide, IDE_SLAVE);
814:                            if(!devices) {
815:                                    register_device(BLK_DEV, &ide1_device);
816:                            }
817:                            if(ide->drive[IDE_SLAVE].flags & DEVICE_IS_DISK) {
818:                                    if(!ide_hd_init(ide, IDE_SLAVE)) {
819:                                            devices++;
820:                                    }
821:                            }
822:                            if(ide->drive[IDE_SLAVE].flags & DEVICE_IS_CDROM) {
823:                                    if(!ide_cd_init(ide, IDE_SLAVE)) {
824:                                            devices++;
825:                                    }
826:                            }
827:                    }
828:            }
829:            if(!devices) {
830:                    disable_irq(IDE1_IRQ);
831:                    unregister_irq(IDE1_IRQ);
832:            }
833: }
```

drivers/block/ide_cd.c


```
/*
 * fiwix/drivers/block/ide_cd.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/asm.h>
#include <fiwix/buffer.h>
#include <fiwix/ide.h>
#include <fiwix/ide_cd.h>
#include <fiwix/ioctl.h>
#include <fiwix/devices.h>
#include <fiwix/sleep.h>
#include <fiwix/timer.h>
#include <fiwix/sched.h>
#include <fiwix/cpu.h>
#include <fiwix/fs.h>
#include <fiwix/part.h>
#include <fiwix/process.h>
#include <fiwix/mm.h>
#include <fiwix/errno.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

/* default size of 1GB is enough to read a whole CDROM */
#define CDROM_DEFAULT_SIZE      (1024 * 1024)   /* in KBs */

static struct resource ide_cd_resource = { NULL, NULL };

static struct fs_operations ide_cd_driver_fsop = {
        0,
        0,

        ide_cd_open,
        ide_cd_close,
        NULL,                   /* read */
        NULL,                   /* write */
        ide_cd_ioctl,
        NULL,                   /* lseek */
        NULL,                   /* readdir */
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        NULL,                   /* bmap */
        NULL,                   /* lockup */
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        NULL,                   /* truncate */
        NULL,                   /* create */
        NULL,                   /* rename */

        ide_cd_read,
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statfs */
        NULL,                   /* read_superblock */
```

drivers/block/ide_cd.c


```
   68:         NULL,                   /* remount_fs */
   69:         NULL,                   /* write_superblock */
   70:         NULL                    /* release_superblock */
   71: };
   72:
   73: static char *sense_key_err[] = {
   74:         "NO SENSE",
   75:         "RECOVERED ERROR",
   76:         "NOT READY",
   77:         "MEDIUM ERROR",
   78:         "HARDWARE ERROR",
   79:         "ILLEGAL REQUEST",
   80:         "UNIT ATTENTION",
   81:         "DATA PROTECT",
   82:         "RESERVED",
   83:         "RESERVED",
   84:         "RESERVED",
   85:         "ABORTED COMMAND",
   86:         "MISCOMPARE",
   87:         "RESERVED"
   88: };
   89:
   90: enum {
   91:         RS_NO_SENSE,
   92:         RS_RECOVERED_ERROR,
   93:         RS_NOT_READY,
   94:         RS_MEDIUM_ERROR,
   95:         RS_HARDWARE_ERROR,
   96:         RS_ILLEGAL_REQUEST,
   97:         RS_UNIT_ATTENTION,
   98:         RS_DATA_PROTECT,
   99:         RS_RESERVED1,
  100:         RS_RESERVED2,
  101:         RS_RESERVED3,
  102:         RS_ABORTED_COMMAND,
  103:         RS_MISCOMPARE,
  104:         RS_RESERVED4
  105: };
  106:
  107: static int send_packet_command(unsigned char *pkt, struct ide *ide, int drive, i
nt blksize)
  108: {
  109:         int n, retries, status;
  110:
  111:         outport_b(ide->ctrl + IDE_DEV_CTRL, 0);
  112:         ide_delay();
  113:         outport_b(ide->base + IDE_DRVHD, IDE_CHS_MODE);
  114:         ide_delay();
  115:         if(ide_drvsel(ide, drive, IDE_CHS_MODE, 0)) {
  116:                 printk("WARNING: %s(): %s: drive not ready to receive PACKET com
mand.\n", __FUNCTION__, ide->drive[drive].dev_name);
  117:                 return 1;
  118:         }
  119:
  120:         CLI();
  121:         outport_b(ide->base + IDE_FEATURES, 0);
  122:         outport_b(ide->base + IDE_SECCNT, 0);
  123:         outport_b(ide->base + IDE_SECNUM, 0);
  124:         outport_b(ide->base + IDE_LCYL, blksize & 0xFF);
  125:         outport_b(ide->base + IDE_HCYL, blksize >> 8);
  126:         outport_b(ide->base + IDE_DRVHD, drive << 4);
  127:         outport_b(ide->base + IDE_COMMAND, ATA_PACKET);
  128:         ide_wait400ns(ide);
  129:
  130: /*
  131:  * NOTE: Some devices prior to ATA/ATAPI-4 assert INTRQ if enabled at this
  132:  * point. See IDENTIFY PACKET DEVICE, word 0, bits 5-6 to determine if an
```

drivers/block/ide_cd.c


```
  133:  * interrupt will occur.
  134:  */
  135:         SET_IDE_RDY_RETR(retries);
  136:
  137:         for(n = 0; n < retries; n++) {
  138:                 status = inport_b(ide->base + IDE_STATUS);
  139:                 if((status & (IDE_STAT_DRQ | IDE_STAT_BSY)) == IDE_STAT_DRQ) {
  140:                         break;
  141:                 }
  142:                 ide_delay();
  143:         }
  144:         if(n >= retries) {
  145:                 printk("WARNING: %s(): %s: drive not ready to receive command pa
cket (retries = %d).\n", __FUNCTION__, ide->drive[drive].dev_name, n);
  146:                 return 1;
  147:         }
  148:
  149:         outport_sw(ide->base + IDE_DATA, pkt, 12 / sizeof(short int));
  150:         return 0;
  151: }
  152:
  153: static int atapi_read_data(__dev_t dev, char *data, struct ide *ide, int blksize
, int offset)
  154: {
  155:         int errno, status;
  156:         char *buffer;
  157:         int retries, bytes;
  158:         struct callout_req creq;
  159:
  160:         for(retries = 0; retries < MAX_IDE_ERR; retries++) {
  161:                 if(ide->channel == IDE_PRIMARY) {
  162:                         ide0_wait_interrupt = ide->base;
  163:                         creq.fn = ide0_timer;
  164:                         creq.arg = 0;
  165:                         add_callout(&creq, WAIT_FOR_IDE);
  166:                         sleep(&irq_ide0, PROC_UNINTERRUPTIBLE);
  167:                         if(ide0_timeout) {
  168:                                 status = inport_b(ide->base + IDE_STATUS);
  169:                                 if((status & (IDE_STAT_RDY | IDE_STAT_DRQ)) != (
IDE_STAT_RDY | IDE_STAT_DRQ)) {
  170:                                         continue;
  171:                                 }
  172:                         }
  173:                         del_callout(&creq);
  174:                 }
  175:                 if(ide->channel == IDE_SECONDARY) {
  176:                         ide1_wait_interrupt = ide->base;
  177:                         creq.fn = ide1_timer;
  178:                         creq.arg = 0;
  179:                         add_callout(&creq, WAIT_FOR_IDE);
  180:                         sleep(&irq_ide1, PROC_UNINTERRUPTIBLE);
  181:                         if(ide1_timeout) {
  182:                                 status = inport_b(ide->base + IDE_STATUS);
  183:                                 if((status & (IDE_STAT_RDY | IDE_STAT_DRQ)) != (
IDE_STAT_RDY | IDE_STAT_DRQ)) {
  184:                                         continue;
  185:                                 }
  186:                         }
  187:                         del_callout(&creq);
  188:                 }
  189:                 status = inport_b(ide->base + IDE_STATUS);
  190:                 if(status & IDE_STAT_ERR) {
  191:                         continue;
  192:                 }
  193:
  194:                 if((status & (IDE_STAT_DRQ | IDE_STAT_BSY)) == 0) {
  195:                         break;
```

drivers/block/ide_cd.c


```
  196:                 }
  197:
  198:                 bytes = (inport_b(ide->base + IDE_HCYL) << 8) + inport_b(ide->ba
se + IDE_LCYL);
  199:                 if(!bytes || bytes > blksize) {
  200:                         break;
  201:                 }
  202:
  203:                 bytes = MAX(bytes, IDE_CD_SECTSIZE);    /* read more than 2048 b
ytes is not supported */
  204:                 buffer = data + offset;
  205:                 inport_sw(ide->base + IDE_DATA, (void *)buffer, bytes / sizeof(s
hort int));
  206:         }
  207:
  208:         if(status & IDE_STAT_ERR) {
  209:                 errno = inport_b(ide->base + IDE_ERROR);
  210:                 printk("WARNING: %s(): error on cdrom device %d,%d, status=0x%x
error=0x%x,\n", __FUNCTION__, MAJOR(dev), MINOR(dev), status, errno);
  211:                 return 1;
  212:         }
  213:
  214:         if(retries >= MAX_IDE_ERR) {
  215:                 printk("WARNING: %s(): timeout on cdrom device %d,%d, status=0x%
x.\n", __FUNCTION__, MAJOR(dev), MINOR(dev), status);
  216:                 /* a reset may be required at this moment */
  217:                 return 1;
  218:         }
  219:         return 0;
  220: }
  221:
  222: static int atapi_cmd_testunit(struct ide *ide, int drive)
  223: {
  224:         unsigned char pkt[12];
  225:
  226:         pkt[0] = ATAPI_TEST_UNIT;
  227:         pkt[1] = NULL;
  228:         pkt[2] = NULL;
  229:         pkt[3] = NULL;
  230:         pkt[4] = NULL;
  231:         pkt[5] = NULL;
  232:         pkt[6] = NULL;
  233:         pkt[7] = NULL;
  234:         pkt[8] = NULL;
  235:         pkt[9] = NULL;
  236:         pkt[10] = NULL;
  237:         pkt[11] = NULL;
  238:         return send_packet_command(pkt, ide, drive, 0);
  239: }
  240:
  241: static int atapi_cmd_reqsense(struct ide *ide, int drive)
  242: {
  243:         unsigned char pkt[12];
  244:
  245:         pkt[0] = ATAPI_REQUEST_SENSE;
  246:         pkt[1] = NULL;
  247:         pkt[2] = NULL;
  248:         pkt[3] = NULL;
  249:         pkt[4] = 252;   /* this command can send up to 252 bytes */
  250:         pkt[5] = NULL;
  251:         pkt[6] = NULL;
  252:         pkt[7] = NULL;
  253:         pkt[8] = NULL;
  254:         pkt[9] = NULL;
  255:         pkt[10] = NULL;
  256:         pkt[11] = NULL;
  257:         return send_packet_command(pkt, ide, drive, 0);
```

drivers/block/ide_cd.c


```
  258: }
  259:
  260: static int atapi_cmd_startstop(int action, struct ide *ide, int drive)
  261: {
  262:         unsigned char pkt[12];
  263:
  264:         pkt[0] = ATAPI_START_STOP;
  265:         pkt[1] = NULL;
  266:         pkt[2] = NULL;
  267:         pkt[3] = NULL;
  268:         pkt[4] = action;
  269:         pkt[5] = NULL;
  270:         pkt[6] = NULL;
  271:         pkt[7] = NULL;
  272:         pkt[8] = NULL;
  273:         pkt[9] = NULL;
  274:         pkt[10] = NULL;
  275:         pkt[11] = NULL;
  276:         return send_packet_command(pkt, ide, drive, 0);
  277: }
  278:
  279: static int atapi_cmd_mediumrm(int action, struct ide *ide, int drive)
  280: {
  281:         unsigned char pkt[12];
  282:
  283:         pkt[0] = ATAPI_MEDIUM_REMOVAL;
  284:         pkt[1] = NULL;
  285:         pkt[2] = NULL;
  286:         pkt[3] = NULL;
  287:         pkt[4] = action;
  288:         pkt[5] = NULL;
  289:         pkt[6] = NULL;
  290:         pkt[7] = NULL;
  291:         pkt[8] = NULL;
  292:         pkt[9] = NULL;
  293:         pkt[10] = NULL;
  294:         pkt[11] = NULL;
  295:         return send_packet_command(pkt, ide, drive, 0);
  296: }
  297:
  298: static int request_sense(char *buffer, __dev_t dev, struct ide *ide, int drive)
  299: {
  300:         int errcode;
  301:         int sense_key, sense_asc;
  302:
  303:         errcode = inport_b(ide->base + IDE_ERROR);
  304:         sense_key = (errcode & 0xF0) >> 4;
  305:         printk("\tSense Key code indicates a '%s' condition.\n", sense_key_err[s
ense_key & 0xF]);
  306:         errcode = atapi_cmd_reqsense(ide, drive);
  307:         printk("reqsense() returned %d\n", errcode);
  308:         errcode = atapi_read_data(dev, buffer, ide, BLKSIZE_2K, 0);
  309:         printk("atapi_read_data() returned %d\n", errcode);
  310:         errcode = (int)(buffer[0] & 0x7F);
  311:         sense_key = (int)(buffer[2] & 0xF);
  312:         sense_asc = (int)(buffer[12] & 0xFF);
  313:         printk("errcode = %x\n", errcode);
  314:         printk("sense_key = %x\n", sense_key);
  315:         printk("sense_asc = %x\n", sense_asc);
  316:         return errcode;
  317: }
  318:
  319: void ide_cd_timer(unsigned int arg)
  320: {
  321:         wakeup(&ide_cd_open);
  322: }
  323:
```

drivers/block/ide_cd.c


```
  324: int ide_cd_open(struct inode *i, struct fd *fd_table)
  325: {
  326:         int minor;
  327:         int drive;
  328:         char *buffer;
  329:         int errcode;
  330:         int sense_key, sense_asc;
  331:         int retries;
  332:         struct ide *ide;
  333:
  334:         if(!(ide = get_ide_controller(i->rdev))) {
  335:                 return -EINVAL;
  336:         }
  337:
  338:         minor = MINOR(i->rdev);
  339:         drive = get_ide_drive(i->rdev);
  340:         if(drive) {
  341:                 minor &= ~(1 << IDE_SLAVE_MSF);
  342:         }
  343:
  344:         CLI();
  345:         lock_resource(&ide_cd_resource);
  346:
  347:         if(!(buffer = (void *)kmalloc())) {
  348:                 unlock_resource(&ide_cd_resource);
  349:                 return -ENOMEM;
  350:         }
  351:
  352:         if((errcode = atapi_cmd_testunit(ide, drive))) {
  353:                 printk("WARNING: %s(): cdrom device %d,%d is not ready for TEST_
UNIT, error %d.\n", __FUNCTION__, MAJOR(i->rdev), MINOR(i->rdev), errcode);
  354:                 request_sense(buffer, i->rdev, ide, drive);
  355:         }
  356:
  357:         for(retries = 0; retries < MAX_CD_ERR; retries++) {
  358:                 if(!(errcode = atapi_cmd_startstop(CD_LOAD, ide, drive))) {
  359:                         break;
  360:                 }
  361:                 printk("WARNING: %s(): cdrom device %d,%d is not ready for CD_LO
AD, error %d.\n", __FUNCTION__, MAJOR(i->rdev), MINOR(i->rdev), errcode);
  362:                 atapi_read_data(i->rdev, buffer, ide, BLKSIZE_2K, 0);
  363:                 errcode = request_sense(buffer, i->rdev, ide, drive);
  364:                 sense_key = (errcode & 0xF0) >> 4;
  365:                 /* trying to eject on slim drives may lead to an illegal request
 */
  366:                 if(!sense_key || sense_key == RS_ILLEGAL_REQUEST) {
  367:                         break;
  368:                 }
  369:                 if(errcode == 0x70 || errcode == 0x71) {
  370:                         sense_key = (int)(buffer[2] & 0xF);
  371:                         sense_asc = (int)(buffer[12] & 0xFF);
  372:                         if(sense_key == RS_NOT_READY && sense_asc == ASC_NO_MEDI
UM) {
  373:                                 kfree((unsigned int)buffer);
  374:                                 unlock_resource(&ide_cd_resource);
  375:                                 return -ENOMEDIUM;
  376:                         }
  377:                 }
  378:         }
  379:
  380:         if(retries == MAX_CD_ERR) {
  381:                 if(sense_key == RS_NOT_READY) {
  382:                         kfree((unsigned int)buffer);
  383:                         unlock_resource(&ide_cd_resource);
  384:                         return -ENOMEDIUM;
  385:                 }
  386:         }
```

drivers/block/ide_cd.c


```
  387:
  388:          if(atapi_cmd_mediumrm(CD_LOCK_MEDIUM, ide, drive)) {
  389:                  printk("WARNING: %s(): error on cdrom device %d,%d while trying
to lock medium.\n", __FUNCTION__, MAJOR(i->rdev), MINOR(i->rdev), ATAPI_MEDIUM_REMOVAL)
;
  390:                  request_sense(buffer, i->rdev, ide, drive);
  391:          }
  392:
  393:          /* this line just to catch interrupt */
  394:          atapi_read_data(i->rdev, buffer, ide, BLKSIZE_2K, 0);
  395:          kfree((unsigned int)buffer);
  396:
  397:          unlock_resource(&ide_cd_resource);
  398:          return 0;
  399: }
  400:
  401: int ide_cd_close(struct inode *i, struct fd *fd_table)
  402: {
  403:          int drive;
  404:          char *buffer;
  405:          struct ide *ide;
  406:
  407:          if(!(ide = get_ide_controller(i->rdev))) {
  408:                  return -EINVAL;
  409:          }
  410:
  411:          if(!(buffer = (void *)kmalloc())) {
  412:                  return -ENOMEM;
  413:          }
  414:
  415:          drive = get_ide_drive(i->rdev);
  416:
  417:          /* FIXME: only if device usage == 0 */
  418:          invalidate_buffers(i->rdev);
  419:
  420:          if(atapi_cmd_mediumrm(CD_UNLOCK_MEDIUM, ide, drive)) {
  421:                  printk("WARNING: %s(): error on cdrom device %d,%d during 0x%x c
ommand.\n", __FUNCTION__, MAJOR(i->rdev), MINOR(i->rdev), ATAPI_MEDIUM_REMOVAL);
  422:          }
  423:
  424:          /* this line just to catch interrupt */
  425:          atapi_read_data(i->rdev, buffer, ide, BLKSIZE_2K, 0);
  426:          kfree((unsigned int)buffer);
  427:
  428:          return 0;
  429: }
  430:
  431: int ide_cd_read(__dev_t dev, __blk_t block, char *buffer, int blksize)
  432: {
  433:          int drive;
  434:          int sectors_to_read;
  435:          int n, retries;
  436:          unsigned char pkt[12];
  437:          struct ide *ide;
  438:
  439:          if(!(ide = get_ide_controller(dev))) {
  440:                  return -EINVAL;
  441:          }
  442:
  443:          drive = get_ide_drive(dev);
  444:          blksize = BLKSIZE_2K;
  445:          sectors_to_read = blksize / IDE_CD_SECTSIZE;
  446:
  447:          pkt[0] = ATAPI_READ10;
  448:          pkt[1] = NULL;
  449:          pkt[2] = (block >> 24) & 0xFF;
  450:          pkt[3] = (block >> 16) & 0xFF;
```

drivers/block/ide_cd.c


```
  451:          pkt[4] = (block >> 8) & 0xFF;
  452:          pkt[5] = block & 0xFF;
  453:          pkt[6] = NULL;
  454:          pkt[7] = (sectors_to_read >> 8) & 0xFF;
  455:          pkt[8] = sectors_to_read & 0xFF;
  456:          pkt[9] = NULL;
  457:          pkt[10] = NULL;
  458:          pkt[11] = NULL;
  459:
  460:          lock_resource(&ide_cd_resource);
  461:          for(n = 0; n < sectors_to_read; n++, block++) {
  462:                  for(retries = 0; retries < MAX_CD_ERR; retries++) {
  463:                          if(send_packet_command(pkt, ide, drive, blksize)) {
  464:                                  printk("\tblock=%d, offset=%d\n", block, block *
 blksize);
  465:                                  unlock_resource(&ide_cd_resource);
  466:                                  return -EIO;
  467:                          }
  468:                          if(atapi_read_data(dev, buffer, ide, blksize, n * IDE_CD
_SECTSIZE)) {
  469:                                  int errcode;
  470:                                  int sense_key;
  471:                                  errcode = inport_b(ide->base + IDE_ERROR);
  472:                                  sense_key = (errcode & 0xF0) >> 4;
  473:                                  printk("\tSense Key code indicates a ’%s’ condit
ion.\n", sense_key_err[sense_key & 0xF]);
  474:                                  if(sense_key) {
  475:                                          continue;
  476:                                  }
  477:                          }
  478:                          break;
  479:                  }
  480:                  if(retries == MAX_CD_ERR) {
  481:                          printk("\tblock=%d, offset=%d\n", block, block * blksize
);
  482:                          unlock_resource(&ide_cd_resource);
  483:                          return -EIO;
  484:                  }
  485:
  486:          }
  487:          unlock_resource(&ide_cd_resource);
  488:          return sectors_to_read * IDE_CD_SECTSIZE;
  489: }
  490:
  491: int ide_cd_ioctl(struct inode *i, int cmd, unsigned long int arg)
  492: {
  493:          struct ide *ide;
  494:
  495:          if(!(ide = get_ide_controller(i->rdev))) {
  496:                  return -EINVAL;
  497:          }
  498:
  499:          switch(cmd) {
  500:                  default:
  501:                          return -EINVAL;
  502:                          break;
  503:          }
  504:
  505:          return 0;
  506: }
  507:
  508: int ide_cd_init(struct ide *ide, int drive)
  509: {
  510:          struct device *d;
  511:
  512:          ide->drive[drive].fsop = &ide_cd_driver_fsop;
  513:
```

drivers/block/ide_cd.c


```
  514:          if(!(d = get_device(BLK_DEV, ide->drive[drive].major))) {
  515:                  return -EINVAL;
  516:          }
  517:          if(drive == IDE_MASTER) {
  518:                  ide->drive[drive].minor_shift = IDE_MASTER_MSF;
  519:                  SET_MINOR(d->minors, 0);
  520:                  ((unsigned int *)d->device_data)[0] = CDROM_DEFAULT_SIZE;
  521:          } else {
  522:                  ide->drive[drive].minor_shift = IDE_SLAVE_MSF;
  523:                  SET_MINOR(d->minors, 1 << IDE_SLAVE_MSF);
  524:                  ((unsigned int *)d->device_data)[1 << IDE_SLAVE_MSF] = CDROM_DEF
AULT_SIZE;
  525:          }
  526:
  527:          return 0;
  528: }
```

drivers/block/ide_hd.c


```
    1: /*
    2:  * fiwix/drivers/block/ide_hd.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/buffer.h>
   10: #include <fiwix/ide.h>
   11: #include <fiwix/ide_hd.h>
   12: #include <fiwix/ioctl.h>
   13: #include <fiwix/devices.h>
   14: #include <fiwix/sleep.h>
   15: #include <fiwix/timer.h>
   16: #include <fiwix/sched.h>
   17: #include <fiwix/cpu.h>
   18: #include <fiwix/fs.h>
   19: #include <fiwix/part.h>
   20: #include <fiwix/process.h>
   21: #include <fiwix/mm.h>
   22: #include <fiwix/errno.h>
   23: #include <fiwix/stdio.h>
   24: #include <fiwix/string.h>
   25:
   26: static struct resource ide_hd_resource = { NULL, NULL };
   27:
   28: static struct fs_operations ide_hd_driver_fsop = {
   29:         0,
   30:         0,
   31:
   32:         ide_hd_open,
   33:         ide_hd_close,
   34:         NULL,                   /* read */
   35:         NULL,                   /* write */
   36:         ide_hd_ioctl,
   37:         NULL,                   /* lseek */
   38:         NULL,                   /* readdir */
   39:         NULL,                   /* mmap */
   40:         NULL,                   /* select */
   41:
   42:         NULL,                   /* readlink */
   43:         NULL,                   /* followlink */
   44:         NULL,                   /* bmap */
   45:         NULL,                   /* lockup */
   46:         NULL,                   /* rmdir */
   47:         NULL,                   /* link */
   48:         NULL,                   /* unlink */
   49:         NULL,                   /* symlink */
   50:         NULL,                   /* mkdir */
   51:         NULL,                   /* mknod */
   52:         NULL,                   /* truncate */
   53:         NULL,                   /* create */
   54:         NULL,                   /* rename */
   55:
   56:         ide_hd_read,
   57:         ide_hd_write,
   58:
   59:         NULL,                   /* read_inode */
   60:         NULL,                   /* write_inode */
   61:         NULL,                   /* ialloc */
   62:         NULL,                   /* ifree */
   63:         NULL,                   /* statfs */
   64:         NULL,                   /* read_superblock */
   65:         NULL,                   /* remount_fs */
   66:         NULL,                   /* write_superblock */
   67:         NULL                    /* release_superblock */
```

drivers/block/ide_hd.c


```
   68: };
   69:
   70: static void assign_minors(__dev_t rdev, struct ide *ide, struct partition *part)
   71: {
   72:         int n;
   73:         int drive, minor;
   74:         struct device *d;
   75:
   76:         minor = 0;
   77:         drive = get_ide_drive(rdev);
   78:
   79:         if(ide->channel == IDE_PRIMARY) {
   80:                 if(!(d = get_device(BLK_DEV, IDE0_MAJOR))) {
   81:                         return;
   82:                 }
   83:         } else if(ide->channel == IDE_SECONDARY) {
   84:                 if(!(d = get_device(BLK_DEV, IDE1_MAJOR))) {
   85:                         return;
   86:                 }
   87:         } else {
   88:                 printk("WARNING: %s(): invalid device %d,%d.\n", __FUNCTION__, M
AJOR(rdev), MINOR(rdev));
   89:                 return;
   90:         }
   91:
   92:         for(n = 0; n < NR_PARTITIONS; n++) {
   93:                 if(drive == IDE_MASTER) {
   94:                         minor = (1 << ide->drive[drive].minor_shift) + n;
   95:                 }
   96:                 if(drive == IDE_SLAVE) {
   97:                         minor = (1 << ide->drive[drive].minor_shift) + n + 1;
   98:                 }
   99:                 CLEAR_MINOR(d->minors, minor);
  100:                 if(part[n].type) {
  101:                         SET_MINOR(d->minors, minor);
  102:                         ((unsigned int *)d->device_data)[minor] = part[n].nr_sec
ts / 2;
  103:                 }
  104:         }
  105: }
  106:
  107: static __off_t block2sector(__off_t offset, int blksize, struct partition *part,
 int minor)
  108: {
  109:         __off_t sector;
  110:
  111:         sector = offset * (blksize / IDE_HD_SECTSIZE);
  112:         if(minor) {
  113:                 sector += part[minor - 1].startsect;
  114:         }
  115:         return sector;
  116: }
  117:
  118: static void sector2chs(__off_t offset, int *cyl, int *head, int *sector, struct
ide_drv_ident *ident)
  119: {
  120:         *cyl = offset / (ident->logic_spt * ident->logic_heads);
  121:         *head = (offset / ident->logic_spt) % ident->logic_heads;
  122:         *sector = (offset % ident->logic_spt) + 1;
  123: }
  124:
  125: int ide_hd_open(struct inode *i, struct fd *fd_table)
  126: {
  127:         return 0;
  128: }
  129:
  130: int ide_hd_close(struct inode *i, struct fd *fd_table)
```

drivers/block/ide_hd.c


```
  131: {
  132:         sync_buffers(i->rdev);
  133:         return 0;
  134: }
  135:
  136: int ide_hd_read(__dev_t dev, __blk_t block, char *buffer, int blksize)
  137: {
  138:         int minor;
  139:         int drive;
  140:         int sectors_to_read, cmd;
  141:         int n, status, r, retries;
  142:         int cyl, head, sector;
  143:         __off_t offset;
  144:         struct ide *ide;
  145:         struct ide_drv_ident *ident;
  146:         struct partition *part;
  147:         struct callout_req creq;
  148:
  149:         if(!(ide = get_ide_controller(dev))) {
  150:                 return -EINVAL;
  151:         }
  152:
  153:         minor = MINOR(dev);
  154:         if((drive = get_ide_drive(dev))) {
  155:                 minor &= ~(1 << IDE_SLAVE_MSF);
  156:         }
  157:
  158:         SET_IDE_RDY_RETR(retries);
  159:
  160:         blksize = blksize ? blksize : BLKSIZE_1K;
  161:         sectors_to_read = MIN(blksize, PAGE_SIZE) / IDE_HD_SECTSIZE;
  162:
  163:         ident = &ide->drive[drive].ident;
  164:         part = ide->drive[drive].part_table;
  165:         offset = block2sector(block, blksize, part, minor);
  166:
  167:         CLI();
  168:         lock_resource(&ide_hd_resource);
  169:
  170:         n = 0;
  171:
  172:         while(n < sectors_to_read) {
  173:                 if(ide->drive[drive].flags & DEVICE_HAS_RW_MULTIPLE) {
  174:                         outport_b(ide->base + IDE_SECCNT, sectors_to_read);
  175:                         cmd = ATA_READ_MULTIPLE_PIO;
  176:                 } else {
  177:                         outport_b(ide->base + IDE_SECCNT, 1);
  178:                         cmd = ATA_READ_PIO;
  179:                 }
  180:
  181:                 if(ide->drive[drive].flags & DEVICE_REQUIRES_LBA) {
  182:                         outport_b(ide->base + IDE_SECNUM, offset & 0xFF);
  183:                         outport_b(ide->base + IDE_LCYL, (offset >> 8) & 0xFF);
  184:                         outport_b(ide->base + IDE_HCYL, (offset >> 16) & 0xFF);
  185:                         if(ide_drvsel(ide, drive, IDE_LBA_MODE, (offset >> 24) &
 0x0F)) {
  186:                                 printk("WARNING: %s(): %s: drive not ready.\n",
__FUNCTION__, ide->drive[drive].dev_name);
  187:                                 unlock_resource(&ide_hd_resource);
  188:                                 return -EIO;
  189:                         }
  190:                 } else {
  191:                         sector2chs(offset, &cyl, &head, &sector, ident);
  192:                         outport_b(ide->base + IDE_SECNUM, sector);
  193:                         outport_b(ide->base + IDE_LCYL, cyl);
  194:                         outport_b(ide->base + IDE_HCYL, (cyl >> 8));
  195:                         if(ide_drvsel(ide, drive, IDE_CHS_MODE, head)) {
```

drivers/block/ide_hd.c


```
  196:                                  printk("WARNING: %s(): %s: drive not ready.\n",
__FUNCTION__, ide->drive[drive].dev_name);
  197:                                  unlock_resource(&ide_hd_resource);
  198:                                  return -EIO;
  199:                          }
  200:                  }
  201:                  outport_b(ide->base + IDE_COMMAND, cmd);
  202:                  if(ide->channel == IDE_PRIMARY) {
  203:                          ide0_wait_interrupt = ide->base;
  204:                          creq.fn = ide0_timer;
  205:                          creq.arg = 0;
  206:                          add_callout(&creq, WAIT_FOR_IDE);
  207:                          sleep(&irq_ide0, PROC_UNINTERRUPTIBLE);
  208:                          if(!ide0_timeout) {
  209:                                  del_callout(&creq);
  210:                          }
  211:                  }
  212:                  if(ide->channel == IDE_SECONDARY) {
  213:                          ide1_wait_interrupt = ide->base;
  214:                          creq.fn = ide1_timer;
  215:                          creq.arg = 0;
  216:                          add_callout(&creq, WAIT_FOR_IDE);
  217:                          sleep(&irq_ide1, PROC_UNINTERRUPTIBLE);
  218:                          if(!ide1_timeout) {
  219:                                  del_callout(&creq);
  220:                          }
  221:                  }
  222:                  for(r = 0; r < retries; r++) {
  223:                          status = inport_b(ide->base + IDE_STATUS);
  224:                          if(!(status & IDE_STAT_BSY) && (status & IDE_STAT_DRQ))
{
  225:                                  break;
  226:                          }
  227:                          ide_delay();
  228:                  }
  229:                  if(status & IDE_STAT_ERR) {
  230:                          printk("WARNING: %s(): %s: error on hard disk dev %d,%d
during read.\n", __FUNCTION__, ide->drive[drive].dev_name, MAJOR(dev), MINOR(dev));
  231:                          printk("\tstatus=0x%x ", status);
  232:                          ide_error(ide, status);
  233:                          printk("\tblock %d, sector %d.\n", block, offset);
  234:                          inport_b(ide->base + IDE_STATUS);       /* clear any pen
ding interrupt */
  235:                          unlock_resource(&ide_hd_resource);
  236:                          return -EIO;
  237:                  }
  238:
  239:                  if(cmd == ATA_READ_MULTIPLE_PIO) {
  240:                          inport_sw(ide->base + IDE_DATA, (void *)buffer, (IDE_HD_
SECTSIZE * sectors_to_read) / sizeof(short int));
  241:                          break;
  242:                  }
  243:                  inport_sw(ide->base + IDE_DATA, (void *)buffer, IDE_HD_SECTSIZE
/ sizeof(short int));
  244:                  inport_b(ide->ctrl + IDE_ALT_STATUS);   /* ignore results */
  245:                  inport_b(ide->base + IDE_STATUS);       /* clear any pending int
errupt */
  246:                  n++;
  247:                  offset++;
  248:                  buffer += IDE_HD_SECTSIZE;
  249:          }
  250:          inport_b(ide->ctrl + IDE_ALT_STATUS);   /* ignore results */
  251:          inport_b(ide->base + IDE_STATUS);       /* clear any pending interrupt *
/
  252:          unlock_resource(&ide_hd_resource);
  253:          return sectors_to_read * IDE_HD_SECTSIZE;
  254: }
```

drivers/block/ide_hd.c


```
  255:
  256: int ide_hd_write(__dev_t dev, __blk_t block, char *buffer, int blksize)
  257: {
  258:         int minor;
  259:         int drive;
  260:         int sectors_to_write, cmd;
  261:         int n, status, r, retries;
  262:         int cyl, head, sector;
  263:         __off_t offset;
  264:         struct ide *ide;
  265:         struct ide_drv_ident *ident;
  266:         struct partition *part;
  267:         struct callout_req creq;
  268:
  269:         if(!(ide = get_ide_controller(dev))) {
  270:                 return -EINVAL;
  271:         }
  272:
  273:         minor = MINOR(dev);
  274:         if((drive = get_ide_drive(dev))) {
  275:                 minor &= ~(1 << IDE_SLAVE_MSF);
  276:         }
  277:
  278:         SET_IDE_RDY_RETR(retries);
  279:
  280:         blksize = blksize ? blksize : BLKSIZE_1K;
  281:         sectors_to_write = MIN(blksize, PAGE_SIZE) / IDE_HD_SECTSIZE;
  282:
  283:         ident = &ide->drive[drive].ident;
  284:         part = ide->drive[drive].part_table;
  285:         offset = block2sector(block, blksize, part, minor);
  286:
  287:         CLI();
  288:         lock_resource(&ide_hd_resource);
  289:
  290:         n = 0;
  291:
  292:         while(n < sectors_to_write) {
  293:                 if(ide->drive[drive].flags & DEVICE_HAS_RW_MULTIPLE) {
  294:                         outport_b(ide->base + IDE_SECCNT, sectors_to_write);
  295:                         cmd = ATA_WRITE_MULTIPLE_PIO;
  296:                 } else {
  297:                         outport_b(ide->base + IDE_SECCNT, 1);
  298:                         cmd = ATA_WRITE_PIO;
  299:                 }
  300:
  301:                 if(ide->drive[drive].flags & DEVICE_REQUIRES_LBA) {
  302:                         outport_b(ide->base + IDE_SECNUM, offset & 0xFF);
  303:                         outport_b(ide->base + IDE_LCYL, (offset >> 8) & 0xFF);
  304:                         outport_b(ide->base + IDE_HCYL, (offset >> 16) & 0xFF);
  305:                         if(ide_drvsel(ide, drive, IDE_LBA_MODE, (offset >> 24) &
 0x0F)) {
  306:                                 printk("WARNING: %s(): %s: drive not ready.\n",
__FUNCTION__, ide->drive[drive].dev_name);
  307:                                 unlock_resource(&ide_hd_resource);
  308:                                 return -EIO;
  309:                         }
  310:                 } else {
  311:                         sector2chs(offset, &cyl, &head, &sector, ident);
  312:                         outport_b(ide->base + IDE_SECNUM, sector);
  313:                         outport_b(ide->base + IDE_LCYL, cyl);
  314:                         outport_b(ide->base + IDE_HCYL, (cyl >> 8));
  315:                         if(ide_drvsel(ide, drive, IDE_CHS_MODE, head)) {
  316:                                 printk("WARNING: %s(): %s: drive not ready.\n",
__FUNCTION__, ide->drive[drive].dev_name);
  317:                                 unlock_resource(&ide_hd_resource);
  318:                                 return -EIO;
```

drivers/block/ide_hd.c


```
  319:                          }
  320:                  }
  321:                  outport_b(ide->base + IDE_COMMAND, cmd);
  322:                  for(r = 0; r < retries; r++) {
  323:                          status = inport_b(ide->base + IDE_STATUS);
  324:                          if(!(status & IDE_STAT_BSY) && (status & IDE_STAT_DRQ))
{
  325:                                  break;
  326:                          }
  327:                          ide_delay();
  328:                  }
  329:                  if(status & IDE_STAT_ERR) {
  330:                          printk("WARNING: %s(): %s: error on hard disk dev %d,%d
during write.\n", __FUNCTION__, ide->drive[drive].dev_name, MAJOR(dev), MINOR(dev));
  331:                          printk("\tstatus=0x%x ", status);
  332:                          ide_error(ide, status);
  333:                          printk("\tblock %d, sector %d.\n", block, offset);
  334:                          inport_b(ide->base + IDE_STATUS);       /* clear any pen
ding interrupt */
  335:                          unlock_resource(&ide_hd_resource);
  336:                          return -EIO;
  337:                  }
  338:
  339:                  if(cmd == ATA_WRITE_MULTIPLE_PIO) {
  340:                          outport_sw(ide->base + IDE_DATA, (void *)buffer, (IDE_HD
_SECTSIZE * sectors_to_write) / sizeof(short int));
  341:                  } else {
  342:                          outport_sw(ide->base + IDE_DATA, (void *)buffer, IDE_HD_
SECTSIZE / sizeof(short int));
  343:                  }
  344:                  if(ide->channel == IDE_PRIMARY) {
  345:                          ide0_wait_interrupt = ide->base;
  346:                          creq.fn = ide0_timer;
  347:                          creq.arg = 0;
  348:                          add_callout(&creq, WAIT_FOR_IDE);
  349:                          sleep(&irq_ide0, PROC_UNINTERRUPTIBLE);
  350:                          if(!ide0_timeout) {
  351:                                  del_callout(&creq);
  352:                          }
  353:                  }
  354:                  if(ide->channel == IDE_SECONDARY) {
  355:                          ide1_wait_interrupt = ide->base;
  356:                          creq.fn = ide1_timer;
  357:                          creq.arg = 0;
  358:                          add_callout(&creq, WAIT_FOR_IDE);
  359:                          sleep(&irq_ide1, PROC_UNINTERRUPTIBLE);
  360:                          if(!ide1_timeout) {
  361:                                  del_callout(&creq);
  362:                          }
  363:                  }
  364:
  365:                  inport_b(ide->ctrl + IDE_ALT_STATUS);   /* ignore results */
  366:                  inport_b(ide->base + IDE_STATUS);       /* clear any pending int
errupt */
  367:                  if(cmd == ATA_WRITE_MULTIPLE_PIO) {
  368:                          break;
  369:                  }
  370:                  n++;
  371:                  offset++;
  372:                  buffer += IDE_HD_SECTSIZE;
  373:          }
  374:          inport_b(ide->ctrl + IDE_ALT_STATUS);   /* ignore results */
  375:          inport_b(ide->base + IDE_STATUS);       /* clear any pending interrupt *
/
  376:          unlock_resource(&ide_hd_resource);
  377:          return sectors_to_write * IDE_HD_SECTSIZE;
  378: }
```

drivers/block/ide_hd.c


```
  379:
  380: int ide_hd_ioctl(struct inode *i, int cmd, unsigned long int arg)
  381: {
  382:         int minor;
  383:         int drive;
  384:         struct ide *ide;
  385:         struct ide_drv_ident *ident;
  386:         struct partition *part;
  387:         struct hd_geometry *geom;
  388:         int errno;
  389:
  390:         if(!(ide = get_ide_controller(i->rdev))) {
  391:                 return -EINVAL;
  392:         }
  393:
  394:         minor = MINOR(i->rdev);
  395:         drive = get_ide_drive(i->rdev);
  396:         if(drive) {
  397:                 minor &= ~(1 << IDE_SLAVE_MSF);
  398:         }
  399:
  400:         ident = &ide->drive[drive].ident;
  401:         part = ide->drive[drive].part_table;
  402:
  403:         switch(cmd) {
  404:                 case HDIO_GETGEO:
  405:                         if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(struct hd_geometry)))) {
  406:                                 return errno;
  407:                         }
  408:                         geom = (struct hd_geometry *)arg;
  409:                         geom->cylinders = ident->logic_cyls;
  410:                         geom->heads = (char)ident->logic_heads;
  411:                         geom->sectors = (char)ident->logic_spt;
  412:                         geom->start = 0;
  413:                         if(minor) {
  414:                                 geom->start = part[minor - 1].startsect;
  415:                         }
  416:                         break;
  417:                 case BLKGETSIZE:
  418:                         if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(unsigned int)))) {
  419:                                 return errno;
  420:                         }
  421:                         if(!minor) {
  422:                                 *(int *)arg = (unsigned int)ide->drive[drive].nr
_sects;
  423:                         } else {
  424:                                 *(int *)arg = (unsigned int)ide->drive[drive].pa
rt_table[minor - 1].nr_sects;
  425:                         }
  426:                         break;
  427:                 case BLKFLSBUF:
  428:                         sync_buffers(i->rdev);
  429:                         invalidate_buffers(i->rdev);
  430:                         break;
  431:                 case BLKRRPART:
  432:                         read_msdos_partition(i->rdev, part);
  433:                         assign_minors(i->rdev, ide, part);
  434:                         break;
  435:                 default:
  436:                         return -EINVAL;
  437:                         break;
  438:         }
  439:
  440:         return 0;
  441: }
```

drivers/block/ide_hd.c


```
  442:
  443: int ide_hd_init(struct ide *ide, int drive)
  444: {
  445:         int n;
  446:         __dev_t rdev;
  447:         struct device *d;
  448:         struct partition *part;
  449:
  450:         rdev = 0;
  451:         ide->drive[drive].fsop = &ide_hd_driver_fsop;
  452:         part = ide->drive[drive].part_table;
  453:
  454:         if(ide->channel == IDE_PRIMARY) {
  455:                 if(!(d = get_device(BLK_DEV, IDE0_MAJOR))) {
  456:                         return -EINVAL;
  457:                 }
  458:                 if(drive == IDE_MASTER) {
  459:                         rdev = MKDEV(IDE0_MAJOR, drive);
  460:                         ide->drive[drive].minor_shift = IDE_MASTER_MSF;
  461:                         SET_MINOR(d->minors, 0);
  462:                         ((unsigned int *)d->device_data)[0] = ide->drive[drive].
nr_sects / 2;
  463:                 } else {
  464:                         rdev = MKDEV(IDE0_MAJOR, 1 << IDE_SLAVE_MSF);
  465:                         ide->drive[drive].minor_shift = IDE_SLAVE_MSF;
  466:                         SET_MINOR(d->minors, 1 << IDE_SLAVE_MSF);
  467:                         ((unsigned int *)d->device_data)[1 << IDE_SLAVE_MSF] = i
de->drive[drive].nr_sects / 2;
  468:                 }
  469:         } else if(ide->channel == IDE_SECONDARY) {
  470:                 if(!(d = get_device(BLK_DEV, IDE1_MAJOR))) {
  471:                         return -EINVAL;
  472:                 }
  473:                 if(drive == IDE_MASTER) {
  474:                         rdev = MKDEV(IDE1_MAJOR, drive);
  475:                         ide->drive[drive].minor_shift = IDE_MASTER_MSF;
  476:                         SET_MINOR(d->minors, 0);
  477:                         ((unsigned int *)d->device_data)[0] = ide->drive[drive].
nr_sects / 2;
  478:                 } else {
  479:                         rdev = MKDEV(IDE1_MAJOR, 1 << IDE_SLAVE_MSF);
  480:                         ide->drive[drive].minor_shift = IDE_SLAVE_MSF;
  481:                         SET_MINOR(d->minors, 1 << IDE_SLAVE_MSF);
  482:                         ((unsigned int *)d->device_data)[1 << IDE_SLAVE_MSF] = i
de->drive[drive].nr_sects / 2;
  483:                 }
  484:         } else {
  485:                 printk("WARNING: %s(): invalid drive number %d.\n", __FUNCTION__
, drive);
  486:                 return 1;
  487:         }
  488:
  489:         read_msdos_partition(rdev, part);
  490:         assign_minors(rdev, ide, part);
  491:         printk("                                    partition summary: ");
  492:         for(n = 0; n < NR_PARTITIONS; n++) {
  493:                 if(part[n].type) {
  494:                         printk("%s%d ", ide->drive[drive].dev_name, n + 1);
  495:                 }
  496:         }
  497:         printk("\n");
  498:
  499:         outport_b(ide->ctrl + IDE_DEV_CTRL, IDE_DEVCTR_NIEN);
  500:         if(ide->drive[drive].flags & DEVICE_HAS_RW_MULTIPLE) {
  501:                 /* read/write in 4KB blocks (8 sectors) by default */
  502:                 outport_b(ide->base + IDE_SECCNT, PAGE_SIZE / IDE_HD_SECTSIZE);
  503:                 outport_b(ide->base + IDE_COMMAND, ATA_SET_MULTIPLE_MODE);
```

drivers/block/ide_hd.c


```
504:                    ide_wait400ns(ide);
505:                    while(inport_b(ide->base + IDE_STATUS) & IDE_STAT_BSY);
506:            }
507:            outport_b(ide->ctrl + IDE_DEV_CTRL, IDE_DEVCTR_DRQ);
508:
509:            return 0;
510: }
```

drivers/block/Makefile


```
 1: # fiwix/drivers/block/Makefile
 2: #
 3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
 4: # Distributed under the terms of the Fiwix License.
 5: #
 6:
 7: .S.o:
 8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
 9: .c.o:
10:         $(CC) $(CFLAGS) -c -o $@ $<
11:
12: OBJS = dma.o floppy.o part.o ide.o ide_hd.o ide_cd.o ramdisk.o
13:
14: block:  $(OBJS)
15:         $(LD) $(LDFLAGS) -r $(OBJS) -o block.o
16:
17: clean:
18:         rm -f *.o
19:
```

drivers/block/part.c


```
    1: /*
    2:  * fiwix/drivers/block/part.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/ide.h>
    9: #include <fiwix/ide_hd.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/part.h>
   12: #include <fiwix/mm.h>
   13: #include <fiwix/errno.h>
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/string.h>
   16:
   17: int read_msdos_partition(__dev_t dev, struct partition *part)
   18: {
   19:         char *buffer;
   20:
   21:         if(!(buffer = (void *)kmalloc())) {
   22:                 return -ENOMEM;
   23:         }
   24:
   25:         if(ide_hd_read(dev, PARTITION_BLOCK, buffer, BLKSIZE_1K) <= 0) {
   26:                 printk("WARNING: %s(): unable to read partition block in device
%d,%d.\n", __FUNCTION__, MAJOR(dev), MINOR(dev));
   27:                 kfree((unsigned int)buffer);
   28:                 return -EIO;
   29:         }
   30:
   31:         memcpy_b(part, (void *)(buffer + MBR_CODE_SIZE), sizeof(struct partition
) * NR_PARTITIONS);
   32:         kfree((unsigned int)buffer);
   33:         return 0;
   34: }
```

drivers/block/ramdisk.c


```
  1: /*
  2:  * fiwix/drivers/block/ramdisk.c
  3:  *
  4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
  5:  * Distributed under the terms of the Fiwix License.
  6:  */
  7:
  8: #include <fiwix/kernel.h>
  9: #include <fiwix/ramdisk.h>
 10: #include <fiwix/ioctl.h>
 11: #include <fiwix/devices.h>
 12: #include <fiwix/part.h>
 13: #include <fiwix/fs.h>
 14: #include <fiwix/errno.h>
 15: #include <fiwix/mm.h>
 16: #include <fiwix/stdio.h>
 17: #include <fiwix/string.h>
 18:
 19: static unsigned int rd_sizes[256];
 20:
 21: static struct fs_operations ramdisk_driver_fsop = {
 22:         0,
 23:         0,
 24:
 25:         ramdisk_open,
 26:         ramdisk_close,
 27:         NULL,                   /* read */
 28:         NULL,                   /* write */
 29:         ramdisk_ioctl,
 30:         ramdisk_lseek,
 31:         NULL,                   /* readdir */
 32:         NULL,                   /* mmap */
 33:         NULL,                   /* select */
 34:
 35:         NULL,                   /* readlink */
 36:         NULL,                   /* followlink */
 37:         NULL,                   /* bmap */
 38:         NULL,                   /* lockup */
 39:         NULL,                   /* rmdir */
 40:         NULL,                   /* link */
 41:         NULL,                   /* unlink */
 42:         NULL,                   /* symlink */
 43:         NULL,                   /* mkdir */
 44:         NULL,                   /* mknod */
 45:         NULL,                   /* truncate */
 46:         NULL,                   /* create */
 47:         NULL,                   /* rename */
 48:
 49:         ramdisk_read,
 50:         ramdisk_write,
 51:
 52:         NULL,                   /* read_inode */
 53:         NULL,                   /* write_inode */
 54:         NULL,                   /* ialloc */
 55:         NULL,                   /* ifree */
 56:         NULL,                   /* statfs */
 57:         NULL,                   /* read_superblock */
 58:         NULL,                   /* remount_fs */
 59:         NULL,                   /* write_superblock */
 60:         NULL                    /* release_superblock */
 61: };
 62:
 63: static struct device ramdisk_device = {
 64:         "ramdisk",
 65:         -1,
 66:         RAMDISK_MAJOR,
 67:         { 0, 0, 0, 0, 0, 0, 0, 0 },
```

drivers/block/ramdisk.c


```
 68:         BLKSIZE_1K,
 69:         &rd_sizes,
 70:         &ramdisk_driver_fsop,
 71: };
 72:
 73: static struct ramdisk * get_ramdisk(int minor)
 74: {
 75:         if(TEST_MINOR(ramdisk_device.minors, minor)) {
 76:                 return &ramdisk_table[minor];
 77:         }
 78:         return NULL;
 79: }
 80:
 81: int ramdisk_open(struct inode *i, struct fd *fd_table)
 82: {
 83:         if(!get_ramdisk(MINOR(i->rdev))) {
 84:                 return -ENXIO;
 85:         }
 86:         return 0;
 87: }
 88:
 89: int ramdisk_close(struct inode *i, struct fd *fd_table)
 90: {
 91:         if(!get_ramdisk(MINOR(i->rdev))) {
 92:                 return -ENXIO;
 93:         }
 94:         return 0;
 95: }
 96:
 97: int ramdisk_read(__dev_t dev, __blk_t block, char *buffer, int blksize)
 98: {
 99:         int size;
100:         __off_t offset;
101:         struct ramdisk *ramdisk;
102:
103:         if(!(ramdisk = get_ramdisk(MINOR(dev)))) {
104:                 return -ENXIO;
105:         }
106:
107:         size = rd_sizes[MINOR(dev)] * 1024;
108:         offset = block * blksize;
109:         blksize = MIN(blksize, size - offset);
110:         memcpy_b((void *)buffer, ramdisk->addr + offset, blksize);
111:         return blksize;
112: }
113:
114: int ramdisk_write(__dev_t dev, __blk_t block, char *buffer, int blksize)
115: {
116:         int size;
117:         __off_t offset;
118:         struct ramdisk *ramdisk;
119:
120:         if(!(ramdisk = get_ramdisk(MINOR(dev)))) {
121:                 return -ENXIO;
122:         }
123:
124:         size = rd_sizes[MINOR(dev)] * 1024;
125:         offset = block * blksize;
126:         blksize = MIN(blksize, size - offset);
127:         memcpy_b((void *)ramdisk->addr + offset, buffer, blksize);
128:         return blksize;
129: }
130:
131: int ramdisk_ioctl(struct inode *i, int cmd, unsigned long int arg)
132: {
133:         struct hd_geometry *geom;
134:         int errno;
```

drivers/block/ramdisk.c


```
  135:
  136:          if(!get_ramdisk(MINOR(i->rdev))) {
  137:                  return -ENXIO;
  138:          }
  139:
  140:          switch(cmd) {
  141:                  case HDIO_GETGEO:
  142:                          if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(struct hd_geometry)))) {
  143:                                  return errno;
  144:                          }
  145:                          geom = (struct hd_geometry *)arg;
  146:                          geom->heads = 63;
  147:                          geom->sectors = 16;
  148:                          geom->cylinders = rd_sizes[MINOR(i->rdev)] * 1024 / BPS;
  149:                          geom->cylinders /= (geom->heads * geom->sectors);
  150:                          geom->start = 0;
  151:                          break;
  152:                  case BLKRRPART:
  153:                          break;
  154:                  case BLKGETSIZE:
  155:                          if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(unsigned int)))) {
  156:                                  return errno;
  157:                          }
  158:                          *(int *)arg = rd_sizes[MINOR(i->rdev)] * 2;
  159:                          break;
  160:                  default:
  161:                          return -EINVAL;
  162:          }
  163:          return 0;
  164: }
  165:
  166: int ramdisk_lseek(struct inode *i, __off_t offset)
  167: {
  168:          return offset;
  169: }
  170:
  171: void ramdisk_init(void)
  172: {
  173:          int n;
  174:          struct ramdisk *ramdisk;
  175:
  176:          if(!_noramdisk) {
  177:                  for(n = 0; n < RAMDISK_MINORS; n++) {
  178:                          SET_MINOR(ramdisk_device.minors, n);
  179:                          rd_sizes[n] = _ramdisksize;
  180:                          ramdisk = get_ramdisk(n);
  181:                          printk("ram%d      0x%08x-0x%08x %d RAMdisk(s) of %dKB s
ize, %dKB blocksize\n", n, ramdisk->addr, ramdisk->addr + (_ramdisksize * 1024), RAMDIS
K_MINORS, _ramdisksize, BLKSIZE_1K / 1024);
  182:                  }
  183:                  register_device(BLK_DEV, &ramdisk_device);
  184:          }
  185: }
```

drivers/char/console.c


```
/*
 * fiwix/drivers/char/console.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/asm.h>
#include <fiwix/kernel.h>
#include <fiwix/ctype.h>
#include <fiwix/console.h>
#include <fiwix/devices.h>
#include <fiwix/tty.h>
#include <fiwix/keyboard.h>
#include <fiwix/sleep.h>
#include <fiwix/pit.h>
#include <fiwix/timer.h>
#include <fiwix/process.h>
#include <fiwix/sched.h>
#include <fiwix/kd.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

#define CSI_J_CUR2END   0       /* clear from cursor to end of screen */
#define CSI_J_STA2CUR   1       /* clear from start of screen to cursor */
#define CSI_J_SCREEN    2       /* clear entire screen */

#define CSI_K_CUR2END   0       /* clear from cursor to end of line */
#define CSI_K_STA2CUR   1       /* clear from start of line to cursor */
#define CSI_K_LINE      2       /* clear entire line */

#define CSE             vc->esc = 0     /* Code Set End */

#define ON              1
#define OFF             0

#define SCROLL_UP       1
#define SCROLL_DOWN     2

/* VT100 ID string generated by <ESC>Z or <ESC>[c */
#define VT100ID         "\033[?1;2c"

/* VT100 report status generated by <ESC>[5n */
#define DEVICE_OK       "\033[0n"
#define DEVICE_NOT_OK   "\033[3n"

/* ISO/IEC 8859-1:1998 (aka latin1, IBM819, CP819), same as in Linux */
static const char *iso8859 =
        "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
        "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
        " !\"#$%&'()*+,-./0123456789:;<=>?"
        "@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_"
        "`abcdefghijklmnopqrstuvwxyz{|}~\0"
        "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
        "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
        "\377\255\233\234\376\235\174\025\376\376\246\256\252\055\376\376"
        "\370\361\375\376\376\346\024\371\376\376\247\257\254\253\376\250"
        "\376\376\376\376\216\217\222\200\376\220\376\376\376\376\376\376"
        "\376\245\376\376\376\376\231\376\350\376\376\376\232\376\376\341"
        "\205\240\203\376\204\206\221\207\212\202\210\211\215\241\214\213"
        "\376\244\225\242\223\376\224\366\355\227\243\226\201\376\376\230"
;

unsigned short int *video_base_address;
short int current_cons;
unsigned char screen_is_off = 0;
int buf_y, buf_top;
```

drivers/char/console.c


```
 68:
 69: struct vconsole vc[NR_VCONSOLES + 1];
 70: unsigned short int vcbuf[VC_BUF_SIZE];
 71:
 72: static struct fs_operations tty_driver_fsop = {
 73:         0,
 74:         0,
 75:
 76:         tty_open,
 77:         tty_close,
 78:         tty_read,
 79:         tty_write,
 80:         tty_ioctl,
 81:         tty_lseek,
 82:         NULL,                   /* readdir */
 83:         NULL,                   /* mmap */
 84:         tty_select,
 85:
 86:         NULL,                   /* readlink */
 87:         NULL,                   /* followlink */
 88:         NULL,                   /* bmap */
 89:         NULL,                   /* lookup */
 90:         NULL,                   /* rmdir */
 91:         NULL,                   /* link */
 92:         NULL,                   /* unlink */
 93:         NULL,                   /* symlink */
 94:         NULL,                   /* mkdir */
 95:         NULL,                   /* mknod */
 96:         NULL,                   /* truncate */
 97:         NULL,                   /* create */
 98:         NULL,                   /* rename */
 99:
100:         NULL,                   /* read_block */
101:         NULL,                   /* write_block */
102:
103:         NULL,                   /* read_inode */
104:         NULL,                   /* write_inode */
105:         NULL,                   /* ialloc */
106:         NULL,                   /* ifree */
107:         NULL,                   /* statfs */
108:         NULL,                   /* read_superblock */
109:         NULL,                   /* remount_fs */
110:         NULL,                   /* write_superblock */
111:         NULL                    /* release_superblock */
112: };
113:
114: static struct device tty_device = {
115:         "vconsole",
116:         KEYBOARD_IRQ,
117:         VCONSOLES_MAJOR,
118:         { 0, 0, 0, 0, 0, 0, 0, 0 },
119:         0,
120:         NULL,
121:         &tty_driver_fsop,
122: };
123:
124: static struct device console_device = {
125:         "console",
126:         KEYBOARD_IRQ,
127:         SYSCON_MAJOR,
128:         { 0, 0, 0, 0, 0, 0, 0, 0 },
129:         0,
130:         NULL,
131:         &tty_driver_fsop,
132: };
133:
134: unsigned short int ansi_color_table[] = {
```

drivers/char/console.c


```
  135:         COLOR_BLACK,
  136:         COLOR_RED,
  137:         COLOR_GREEN,
  138:         COLOR_BROWN,
  139:         COLOR_BLUE,
  140:         COLOR_MAGENTA,
  141:         COLOR_CYAN,
  142:         COLOR_WHITE
  143: };
  144:
  145: static void update_curpos(struct vconsole *vc)
  146: {
  147:         unsigned short int curpos;
  148:
  149:         if(vc->has_focus) {
  150:                 curpos = (vc->y * vc->columns) + vc->x;
  151:                 outport_b(video_port + CRT_INDEX, CRT_CURSOR_POS_HI);
  152:                 outport_b(video_port + CRT_DATA, (curpos >> 8) & 0xFF);
  153:                 outport_b(video_port + CRT_INDEX, CRT_CURSOR_POS_LO);
  154:                 outport_b(video_port + CRT_DATA, (curpos & 0xFF));
  155:         }
  156: }
  157:
  158: static void show_cursor(int mode)
  159: {
  160:         int status;
  161:
  162:         switch(mode) {
  163:                 case ON:
  164:                         outport_b(video_port + CRT_INDEX, CRT_CURSOR_STR);
  165:                         status = inport_b(video_port + CRT_DATA);
  166:                         outport_b(video_port + CRT_DATA, status & CURSOR_MASK);
  167:                         break;
  168:                 case OFF:
  169:                         outport_b(video_port + CRT_INDEX, CRT_CURSOR_STR);
  170:                         status = inport_b(video_port + CRT_DATA);
  171:                         outport_b(video_port + CRT_DATA, status | CURSOR_DISABLE
);
  172:                         break;
  173:         }
  174: }
  175:
  176: static void get_curpos(struct vconsole *vc)
  177: {
  178:         unsigned short int curpos;
  179:
  180:         outport_b(video_port + CRT_INDEX, CRT_CURSOR_POS_HI);
  181:         curpos = inport_b(video_port + CRT_DATA) << 8;
  182:         outport_b(video_port + CRT_INDEX, CRT_CURSOR_POS_LO);
  183:         curpos |= inport_b(video_port + CRT_DATA);
  184:
  185:         vc->x = curpos % vc->columns;
  186:         vc->y = curpos / vc->columns;
  187: }
  188:
  189: static void adjust(struct vconsole *vc, int x, int y)
  190: {
  191:         if(x < 0) {
  192:                 x = 0;
  193:         }
  194:         if(x >= vc->columns) {
  195:                 x = vc->columns - 1;
  196:         }
  197:         if(y < 0) {
  198:                 y = 0;
  199:         }
  200:         if(y >= vc->bottom) {
```

drivers/char/console.c


```
  201:                 y = vc->bottom - 1;
  202:         }
  203:         vc->x = x;
  204:         vc->y = y;
  205: }
  206:
  207: static void delete_char(struct vconsole *vc)
  208: {
  209:         int n, from;
  210:
  211:         from = (vc->y * vc->columns) + vc->x;
  212:         n = vc->x;
  213:         while(++n < vc->columns) {
  214:                 memcpy_w(vc->vidmem + from, vc->vidmem + from + 1, 1);
  215:                 from++;
  216:         }
  217:         memset_w(vc->vidmem + from, BLANK_MEM, 1);
  218: }
  219:
  220: static void insert_char(struct vconsole *vc)
  221: {
  222:         int n, from;
  223:         unsigned short int tmp, last_char;
  224:
  225:         from = (vc->y * vc->columns) + vc->x;
  226:         n = vc->x + 1;
  227:         last_char = BLANK_MEM;
  228:         while(++n < vc->columns) {
  229:                 memcpy_w(&tmp, vc->vidmem + from, 1);
  230:                 memset_w(vc->vidmem + from, last_char, 1);
  231:                 last_char = tmp;
  232:                 from++;
  233:         }
  234: }
  235:
  236: static void scroll_screen(struct vconsole *vc, int top, int mode)
  237: {
  238:         int n, count, from;
  239:
  240:         if(!top) {
  241:                 top = vc->top;
  242:         }
  243:         switch(mode) {
  244:                 case SCROLL_UP:
  245:                         count = (vc->columns * (vc->bottom - top - 1)) * 2;
  246:                         from = top * vc->columns;
  247:                         top = (top + 1) * vc->columns;
  248:                         memcpy_b(vc->vidmem + from, vc->vidmem + top, count);
  249:                         memset_w(vc->vidmem + from + (count / 2), BLANK_MEM, (to
p * 2) / sizeof(unsigned short int));
  250:                         break;
  251:                 case SCROLL_DOWN:
  252:                         count = vc->columns * 2;
  253:                         for(n = vc->bottom - 1; n >= top; n--) {
  254:                                 memcpy_b(vc->vidmem + (vc->columns * (n + 1)), v
c->vidmem + (vc->columns * n), count);
  255:                         }
  256:                         memset_w(vc->vidmem + (top * vc->columns), BLANK_MEM, co
unt / sizeof(unsigned short int));
  257:                         break;
  258:         }
  259:         return;
  260: }
  261:
  262: static void cr(struct vconsole *vc)
  263: {
  264:         vc->x = 0;
```

drivers/char/console.c


```
  265: }
  266:
  267: static void lf(struct vconsole *vc)
  268: {
  269:         if(vc->y == vc->bottom) {
  270:                 scroll_screen(vc, 0, SCROLL_UP);
  271:         } else {
  272:                 vc->y++;
  273:         }
  274: }
  275:
  276: static void ri(struct vconsole *vc)
  277: {
  278:         if(vc->y == 0) {
  279:                 scroll_screen(vc, vc->y, SCROLL_DOWN);
  280:         } else {
  281:                 vc->y--;
  282:         }
  283: }
  284:
  285: static void csi_J(struct vconsole *vc, int mode)
  286: {
  287:         int from, count;
  288:
  289:         switch(mode) {
  290:                 case CSI_J_CUR2END:     /* Erase Down <ESC>[J */
  291:                         from = ((vc->y * vc->columns) + vc->x) * 2;
  292:                         count = (SCREEN_SIZE - from) / sizeof(unsigned short int
);
  293:                         break;
  294:                 case CSI_J_STA2CUR:     /* Erase Up <ESC>[1J */
  295:                         from = 0;
  296:                         count = (((vc->y * vc->columns) + vc->x) * 2) / sizeof(u
nsigned short int);
  297:                         break;
  298:                 case CSI_J_SCREEN:      /* Erase Screen <ESC>[2J */
  299:                         from = 0;
  300:                         count = SCREEN_SIZE / sizeof(unsigned short int);
  301:                         break;
  302:                 default:
  303:                         return;
  304:         }
  305:         memset_w(vc->vidmem + from, vc->color_attr, count);
  306: }
  307:
  308: static void csi_K(struct vconsole *vc, int mode)
  309: {
  310:         int from, count;
  311:
  312:         switch(mode) {
  313:                 case CSI_K_CUR2END:     /* Erase End of Line <ESC>[K */
  314:                         from = (vc->y * vc->columns) + vc->x;
  315:                         count = ((vc->columns - vc->x) * 2) / sizeof(unsigned sh
ort int);
  316:                         break;
  317:                 case CSI_K_STA2CUR:     /* Erase Start of Line <ESC>[1K */
  318:                         from = vc->y * vc->columns;
  319:                         count = (vc->x * 2) / sizeof(unsigned short int);
  320:                         break;
  321:                 case CSI_K_LINE:        /* Erase Line <ESC>[2K */
  322:                         from = vc->y * vc->columns;
  323:                         count = (vc->columns * 2) / sizeof(unsigned short int);
  324:                         break;
  325:                 default:
  326:                         return;
  327:         }
  328:         memset_w(vc->vidmem + from, vc->color_attr, count);
```

drivers/char/console.c


```
  329: }
  330:
  331: static void csi_L(struct vconsole *vc, int count)
  332: {
  333:         if(count > (vc->bottom - vc->top)) {
  334:                 count = vc->bottom - vc->top;
  335:         }
  336:         while(count--) {
  337:                 scroll_screen(vc, vc->y, SCROLL_DOWN);
  338:         }
  339: }
  340:
  341: static void csi_M(struct vconsole *vc, int count)
  342: {
  343:         if(count > (vc->bottom - vc->top)) {
  344:                 count = vc->bottom - vc->top;
  345:         }
  346:         while(count--) {
  347:                 scroll_screen(vc, vc->y, SCROLL_UP);
  348:         }
  349: }
  350:
  351: static void csi_P(struct vconsole *vc, int count)
  352: {
  353:         if(count > vc->columns) {
  354:                 count = vc->columns;
  355:         }
  356:         while(count--) {
  357:                 delete_char(vc);
  358:         }
  359: }
  360:
  361: static void csi_at(struct vconsole *vc, int count)
  362: {
  363:         if(count > vc->columns) {
  364:                 count = vc->columns;
  365:         }
  366:         while(count--) {
  367:                 insert_char(vc);
  368:         }
  369: }
  370:
  371: static void default_color_attr(struct vconsole *vc)
  372: {
  373:         vc->color_attr = DEF_MODE;
  374:         vc->bold = 0;
  375:         vc->underline = 0;
  376:         vc->blink = 0;
  377:         vc->reverse = 0;
  378: }
  379:
  380: static void csi_m(struct vconsole *vc)
  381: {
  382:         if(vc->reverse) {
  383:                 vc->color_attr = ((vc->color_attr & 0x7000) >> 4) | ((vc->color_
attr & 0x0700) << 4) | (vc->color_attr & 0x8800);
  384:         }
  385:
  386:         switch(vc->parmv1) {
  387:                 case COLOR_NORMAL:
  388:                         default_color_attr(vc);
  389:                         break;
  390:                 case COLOR_BOLD:
  391:                         vc->bold = 1;
  392:                         break;
  393:                 case COLOR_BOLD_OFF:
  394:                         vc->bold = 0;
```

drivers/char/console.c


```
  395:                          break;
  396:                  case COLOR_BLINK:
  397:                          vc->blink = 1;
  398:                          break;
  399:                  case COLOR_REVERSE:
  400:                          vc->reverse = 1;
  401:                          break;
  402:                  case 21:
  403:                  case 22:
  404:                          vc->bold = 1;
  405:                          break;
  406:                  case 25:
  407:                          vc->blink = 0;
  408:                          break;
  409:                  case 27:
  410:                          vc->reverse = 0;
  411:                          break;
  412:          }
  413:          if(vc->parmv1 >= 30 && vc->parmv1 <= 37) {
  414:                  vc->color_attr = (vc->color_attr & 0xF8FF) | (ansi_color_table[v
c->parmv1 - 30]);
  415:          }
  416:          if(vc->parmv1 >= 40 && vc->parmv1 <= 47) {
  417:                  vc->color_attr = (vc->color_attr & 0x8FFF) | ((ansi_color_table[
vc->parmv1 - 40]) << 4);
  418:          }
  419:          if(vc->parmv2 >= 30 && vc->parmv2 <= 37) {
  420:                  vc->color_attr = (vc->color_attr & 0xF8FF) | (ansi_color_table[v
c->parmv2 - 30]);
  421:          }
  422:          if(vc->parmv2 >= 40 && vc->parmv2 <= 47) {
  423:                  vc->color_attr = (vc->color_attr & 0x8FFF) | ((ansi_color_table[
vc->parmv2 - 40]) << 4);
  424:          }
  425:          if(vc->bold) {
  426:                  vc->color_attr |= 0x0800;
  427:          }
  428:          if(vc->blink) {
  429:                  vc->color_attr |= 0x8000;
  430:          }
  431:          if(vc->reverse) {
  432:                  vc->color_attr = ((vc->color_attr & 0x7000) >> 4) | ((vc->color_
attr & 0x0700) << 4) | (vc->color_attr & 0x8800);
  433:          }
  434: }
  435:
  436: static void init_vt(struct vconsole *vc)
  437: {
  438:          vc->vt_mode.mode = VT_AUTO;
  439:          vc->vt_mode.waitv = 0;
  440:          vc->vt_mode.relsig = 0;
  441:          vc->vt_mode.acqsig = 0;
  442:          vc->vt_mode.frsig = 0;
  443:          vc->vc_mode = KD_TEXT;
  444:          vc->tty->pid = 0;
  445:          vc->switchto_tty = -1;
  446: }
  447:
  448: static void insert_seq(struct tty *tty, char *buf, int count)
  449: {
  450:          while(count--) {
  451:                  tty_queue_putchar(tty, &tty->read_q, *(buf++));
  452:          }
  453:          tty->input(tty);
  454: }
  455:
  456: static void echo_char(struct vconsole *vc, unsigned char *buf, unsigned int coun
```

drivers/char/console.c


```
t)
  457: {
  458:         int n;
  459:         unsigned char ch;
  460:
  461:         if(vc->has_focus) {
  462:                 if(buf_top) {
  463:                         vconsole_restore(vc);
  464:                         show_cursor(ON);
  465:                         buf_top = 0;
  466:                 }
  467:         }
  468:
  469:         while(count--) {
  470:                 ch = *buf++;
  471:                 if(ch == NULL) {
  472:                         continue;
  473:
  474:                 } else if(ch == '\b') {
  475:                         if(vc->x) {
  476:                                 vc->x--;
  477:                         }
  478:
  479:                 } else if(ch == '\a') {
  480:                         vconsole_beep();
  481:
  482:                 } else if(ch == '\r') {
  483:                         cr(vc);
  484:
  485:                 } else if(ch == '\n') {
  486:                         cr(vc);
  487:                         vc->y++;
  488:                         if(vc->has_focus) {
  489:                                 buf_y++;
  490:                         }
  491:
  492:                 } else if(ch == '\t') {
  493:                         while(vc->x < (vc->columns - 1)) {
  494:                                 if(vc->tab_stop[++vc->x]) {
  495:                                         break;
  496:                                 }
  497:                         }
  498: /*                      vc->x += TAB_SIZE - (vc->x % TAB_SIZE); */
  499:                         vc->check_x = 1;
  500:
  501:                 } else {
  502:                         if((vc->x == vc->columns - 1) && vc->check_x) {
  503:                                 vc->x = 0;
  504:                                 vc->y++;
  505:                                 if(vc->has_focus) {
  506:                                         buf_y++;
  507:                                 }
  508:                         }
  509:                         if(vc->y >= vc->bottom) {
  510:                                 scroll_screen(vc, 0, SCROLL_UP);
  511:                                 vc->y--;
  512:                         }
  513:                         ch = iso8859[ch];
  514:                         vc->vidmem[(vc->y * vc->columns) + vc->x] = vc->color_at
tr | ch;
  515:                         if(vc->has_focus) {
  516:                                 vcbuf[(buf_y * vc->columns) + vc->x] = vc->color
_attr | ch;
  517:                         }
  518:                         if(vc->x < vc->columns - 1) {
  519:                                 vc->check_x = 0;
  520:                                 vc->x++;
```

drivers/char/console.c


```
  521:                          } else {
  522:                                  vc->check_x = 1;
  523:                          }
  524:                  }
  525:                  if(vc->y >= vc->bottom) {
  526:                          scroll_screen(vc, 0, SCROLL_UP);
  527:                          vc->y--;
  528:                  }
  529:                  if(vc->has_focus) {
  530:                          if(buf_y >= VC_BUF_LINES) {
  531:                                  memcpy_b(vcbuf, vcbuf + SCREEN_COLS, VC_BUF_SIZE
 - (SCREEN_COLS * 2));
  532:                                  for(n = (SCREEN_COLS * (VC_BUF_LINES - 1)); n <
(SCREEN_COLS * VC_BUF_LINES); n++) {
  533:                                          vcbuf[n] = BLANK_MEM;
  534:                                  }
  535:                                  buf_y--;
  536:                          }
  537:                  }
  538:          }
  539:          update_curpos(vc);
  540: }
  541:
  542: void vconsole_reset(struct tty *tty)
  543: {
  544:          int n;
  545:          struct vconsole *vc;
  546:
  547:          vc = (struct vconsole *)tty->driver_data;
  548:
  549:          vc->top = 0;
  550:          vc->bottom = SCREEN_LINES;
  551:          vc->columns = SCREEN_COLS;
  552:          vc->check_x = 0;
  553:          vc->led_status = 0;
  554:          set_leds(vc->led_status);
  555:          vc->scrlock = vc->numlock = vc->capslock = 0;
  556:          vc->esc = vc->sbracket = vc->semicolon = vc->question = 0;
  557:          vc->parmv1 = vc->parmv2 = 0;
  558:          default_color_attr(vc);
  559:          vc->insert_mode = 0;
  560:          vc->saved_x = vc->saved_y = 0;
  561:
  562:          for(n = 0; n < MAX_TAB_COLS; n++) {
  563:                  if(!(n % TAB_SIZE)) {
  564:                          vc->tab_stop[n] = 1;
  565:                  } else {
  566:                          vc->tab_stop[n] = 0;
  567:                  }
  568:          }
  569:
  570:          termios_reset(tty);
  571:          vc->tty->winsize.ws_row = vc->bottom - vc->top;
  572:          vc->tty->winsize.ws_col = vc->columns;
  573:          vc->tty->winsize.ws_xpixel = 0;
  574:          vc->tty->winsize.ws_ypixel = 0;
  575:          vc->tty->lnext = 0;
  576:
  577:          init_vt(vc);
  578:          vc->blanked = 0;
  579:          update_curpos(vc);
  580: }
  581:
  582: void vconsole_write(struct tty *tty)
  583: {
  584:          int n;
  585:          unsigned char ch;
```

drivers/char/console.c


```
  586:         int numeric;
  587:         struct vconsole *vc;
  588:
  589:         vc = (struct vconsole *)tty->driver_data;
  590:
  591:         if(buf_top) {
  592:                 vconsole_restore(vc);
  593:                 buf_top = 0;
  594:                 show_cursor(ON);
  595:                 update_curpos(vc);
  596:         }
  597:
  598:         numeric = 0;
  599:
  600:         while(!vc->scrlock && tty->write_q.count > 0) {
  601:                 ch = tty_queue_getchar(&tty->write_q);
  602:
  603:                 if(vc->esc) {
  604:                         if(vc->sbracket) {
  605:                                 if(IS_NUMERIC(ch)) {
  606:                                         numeric = 1;
  607:                                         if(vc->semicolon) {
  608:                                                 vc->parmv2 *= 10;
  609:                                                 vc->parmv2 += ch - '0';
  610:                                         } else {
  611:                                                 vc->parmv1 *= 10;
  612:                                                 vc->parmv1 += ch - '0';
  613:                                         }
  614:                                         continue;
  615:                                 }
  616:                                 switch(ch) {
  617:                                         case ';':
  618:                                                 vc->semicolon = 1;
  619:                                                 vc->parmv2 = 0;
  620:                                                 continue;
  621:                                         case '?':
  622:                                                 vc->question = 1;
  623:                                                 continue;
  624:                                         case 'A':       /* Cursor Up <ESC>[{COUN
T}A */
  625:                                                 vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
  626:                                                 adjust(vc, vc->x, vc->y - vc->pa
rmv1);
  627:                                                 CSE;
  628:                                                 continue;
  629:                                         case 'B':       /* Cursor Down <ESC>[{CO
UNT}B */
  630:                                                 vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
  631:                                                 adjust(vc, vc->x, vc->y + vc->pa
rmv1);
  632:                                                 CSE;
  633:                                                 continue;
  634:                                         case 'C':       /* Cursor Forward <ESC>[
{COUNT}C */
  635:                                                 vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
  636:                                                 adjust(vc, vc->x + vc->parmv1, v
c->y);
  637:                                                 CSE;
  638:                                                 continue;
  639:                                         case 'D':       /* Cursor Backward <ESC>
[{COUNT}D */
  640:                                                 vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
  641:                                                 adjust(vc, vc->x - vc->parmv1, v
```

drivers/char/console.c


```
c->y);
  642:                                          CSE;
  643:                                          continue;
  644:                                  case 'E':       /* Cursor Next Line(s) <
ESC>[{COUNT}E */
  645:                                          vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
  646:                                          adjust(vc, 0, vc->y + vc->parmv1
);
  647:                                          CSE;
  648:                                          continue;
  649:                                  case 'F':       /* Cursor Previous Line(
s) <ESC>[{COUNT}F */
  650:                                          vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
  651:                                          adjust(vc, 0, vc->y - vc->parmv1
);
  652:                                          CSE;
  653:                                          continue;
  654:                                  case 'G':       /* Cursor Horizontal Pos
ition <ESC>[{NUM1}G */
  655:                                  case '`':
  656:                                          vc->parmv1 = vc->parmv1 ? vc->pa
rmv1 - 1 : vc->parmv1;
  657:                                          adjust(vc, vc->parmv1, vc->y);
  658:                                          CSE;
  659:                                          continue;
  660:                                  case 'H':       /* Cursor Home <ESC>[{RO
W};{COLUMN}H */
  661:                                  case 'f':       /* Force Cursor Position
 <ESC>[{ROW};{COLUMN}f */
  662:                                          vc->parmv1 = vc->parmv1 ? vc->pa
rmv1 - 1 : vc->parmv1;
  663:                                          vc->parmv2 = vc->parmv2 ? vc->pa
rmv2 - 1 : vc->parmv2;
  664:                                          adjust(vc, vc->parmv2, vc->parmv
1);
  665:                                          CSE;
  666:                                          continue;
  667:                                  case 'J':       /* Erase (Down/Up/Screen
) <ESC>[J */
  668:                                          csi_J(vc, vc->parmv1);
  669:                                          CSE;
  670:                                          continue;
  671:                                  case 'K':       /* Erase (End of/Start o
f/) Line <ESC>[K */
  672:                                          csi_K(vc, vc->parmv1);
  673:                                          CSE;
  674:                                          continue;
  675:                                  case 'L':       /* Insert Line(s) <ESC>[
{COUNT}L */
  676:                                          vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
  677:                                          csi_L(vc, vc->parmv1);
  678:                                          CSE;
  679:                                          continue;
  680:                                  case 'M':       /* Delete Line(s) <ESC>[
{COUNT}M */
  681:                                          vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
  682:                                          csi_M(vc, vc->parmv1);
  683:                                          CSE;
  684:                                          continue;
  685:                                  case 'P':       /* Delete Character(s) <
ESC>[{COUNT}P */
  686:                                          vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
```

drivers/char/console.c


```
  687:                                                  csi_P(vc, vc->parmv1);
  688:                                                  CSE;
  689:                                                  continue;
  690:                                          case '@':       /* Insert Character(s) <
ESC>[{COUNT}@ */
  691:                                                  vc->parmv1 = !vc->parmv1 ? 1 : v
c->parmv1;
  692:                                                  csi_at(vc, vc->parmv1);
  693:                                                  CSE;
  694:                                                  continue;
  695:                                          case 'c':       /* Query Device Code <ES
C>[c */
  696:                                                  if(!numeric) {
  697:                                                          insert_seq(tty, VT100ID,
 7);
  698:                                                  }
  699:                                                  CSE;
  700:                                                  continue;
  701:                                          case 'd':       /* Cursor Vertical Posit
ion <ESC>[{NUM1}d */
  702:                                                  vc->parmv1 = vc->parmv1 ? vc->pa
rmv1 - 1 : vc->parmv1;
  703:                                                  adjust(vc, vc->x, vc->parmv1);
  704:                                                  CSE;
  705:                                                  continue;
  706:                                          case 'g':
  707:                                                  switch(vc->parmv1) {
  708:                                                          case 0: /* Clear Tab <ES
C>[g */
  709:                                                                  vc->tab_stop[vc-
>x] = 0;
  710:                                                                  break;
  711:                                                          case 3: /* Clear All Tab
s <ESC>[3g */
  712:                                                                  for(n = 0; n < M
AX_TAB_COLS; n++)
  713:                                                                          vc->tab_
stop[n] = 0;
  714:                                                                  break;
  715:                                                  }
  716:                                                  CSE;
  717:                                                  continue;
  718:                                          case 'h':
  719:                                                  if(vc->question) {
  720:                                                          switch(vc->parmv1) {
  721:                                                                  /* DEC modes */
  722:                                                                  case 25: /* Swit
ch Cursor Visible <ESC>[?25h */
  723:                                                                          show_cur
sor(ON);
  724:                                                                          break;
  725:                                                                  case 4:
  726:                                                                          vc->inse
rt_mode = ON; /* not used */
  727:                                                                          break;
  728:                                                          }
  729:                                                  }
  730:                                                  CSE;
  731:                                                  continue;
  732:                                          case 'l':
  733:                                                  if(vc->question) {
  734:                                                          switch(vc->parmv1) {
  735:                                                                  /* DEC modes */
  736:                                                                  case 25: /* Swit
ch Cursor Invisible <ESC>[?25l */
  737:                                                                          show_cur
sor(OFF);
```

drivers/char/console.c


```
  738:                                                                  break;
  739:                                                          case 4:
  740:                                                                  vc->inse
rt_mode = OFF; /* not used */
  741:                                                                  break;
  742:                                                          }
  743:                                                  }
  744:                                                  CSE;
  745:                                                  continue;
  746:                                          case 'm':       /* Character Attributes
<ESC>{NUM1}{NUM2}m */
  747:                                                  csi_m(vc);
  748:                                                  CSE;
  749:                                                  continue;
  750:                                          case 'n':
  751:                                                  if(!vc->question) {
  752:                                                          switch(vc->parmv1) {
  753:                                                                  case 5: /* Query
 Device Status <ESC>[5n */
  754:                                                                  insert_s
eq(tty, DEVICE_OK, 4);
  755:                                                                  break;
  756:                                                                  case 6: /* Query
 Cursor Position <ESC>[6n */
  757:                                                                  {
  758:
char curpos[8];
  759:
char len;
  760:
len = sprintk(curpos, "\033[%d;%dR", vc->y, vc->x);
  761:
insert_seq(tty, curpos, len);
  762:                                                                  }
  763:                                                                  break;
  764:                                                          }
  765:                                                  }
  766:                                                  CSE;
  767:                                                  continue;
  768:                                          case 'r':       /* Top and Bottom Margin
s <ESC>[r  / <ESC>[{start};{end}r */
  769:                                                  if(!vc->parmv1) {
  770:                                                          vc->parmv1++;
  771:                                                  }
  772:                                                  if(!vc->parmv2) {
  773:                                                          vc->parmv2 = SCREEN_LINE
S;
  774:                                                  }
  775:                                                  if(vc->parmv1 < vc->parmv2 && vc
->parmv2 <= SCREEN_LINES) {
  776:                                                          vc->top = vc->parmv1 - 1
;
  777:                                                          vc->bottom = vc->parmv2;
  778:                                                          adjust(vc, 0, 0);
  779:                                                  }
  780:                                                  CSE;
  781:                                                  continue;
  782:                                          case 's':       /* Save Cursor <ESC>[s *
/
  783:                                                  vc->saved_x = vc->x;
  784:                                                  vc->saved_y = vc->y;
  785:                                                  CSE;
  786:                                                  continue;
  787:                                          case 'u':       /* Restore Cursor <ESC>[
u */
  788:                                                  vc->x = vc->saved_x;
  789:                                                  vc->y = vc->saved_y;
```

drivers/char/console.c


```
  790:                                                    CSE;
  791:                                                    continue;
  792:                                            default:
  793:                                                    CSE;
  794:                                                    break;
  795:                                    }
  796:                            } else {
  797:                                    switch(ch) {
  798:                                            case '[':
  799:                                                    vc->sbracket = 1;
  800:                                                    vc->semicolon = 0;
  801:                                                    vc->question = 0;
  802:                                                    vc->parmv1 = vc->parmv2 = 0;
  803:                                                    continue;
  804:                                            case '7':       /* Save Cursor & Attrs <
ESC>7 */
  805:                                                    vc->saved_x = vc->x;
  806:                                                    vc->saved_y = vc->y;
  807:                                                    CSE;
  808:                                                    continue;
  809:                                            case '8':       /* Restore Cursor & Attr
s <ESC>8 */
  810:                                                    vc->x = vc->saved_x;
  811:                                                    vc->y = vc->saved_y;
  812:                                                    CSE;
  813:                                                    continue;
  814:                                            case 'D':       /* Scroll Down <ESC>D */
  815:                                                    lf(vc);
  816:                                                    CSE;
  817:                                                    continue;
  818:                                            case 'E':       /* Move To Next Line <ES
C>E */
  819:                                                    cr(vc);
  820:                                                    lf(vc);
  821:                                                    CSE;
  822:                                                    continue;
  823:                                            case 'H':       /* Set Tab <ESC>H */
  824:                                                    vc->tab_stop[vc->x] = 1;
  825:                                                    CSE;
  826:                                                    continue;
  827:                                            case 'M':       /* Scroll Up <ESC>M */
  828:                                                    ri(vc);
  829:                                                    CSE;
  830:                                                    continue;
  831:                                            case 'Z':       /* Identify Terminal <ES
C>Z */
  832:                                                    insert_seq(tty, VT100ID, 7);
  833:                                                    CSE;
  834:                                                    continue;
  835:                                            case 'c':       /* Reset Device <ESC>c *
/
  836:                                                    vconsole_reset(vc->tty);
  837:                                                    vc->x = vc->y = 0;
  838:                                                    csi_J(vc, CSI_J_SCREEN);
  839:                                                    CSE;
  840:                                                    continue;
  841:                                            default:
  842:                                                    CSE;
  843:                                                    break;
  844:                                    }
  845:                            }
  846:                    }
  847:                    switch(ch) {
  848:                            case '\033':
  849:                                    vc->esc = 1;
  850:                                    vc->sbracket = 0;
  851:                                    vc->semicolon = 0;
```

drivers/char/console.c


```
  852:                                    vc->question = 0;
  853:                                    vc->parmv1 = vc->parmv2 = 0;
  854:                                    continue;
  855:                            default:
  856:                                    echo_char(vc, &ch, 1);
  857:                                    continue;
  858:                    }
  859:            }
  860:            if(vc->vc_mode != KD_GRAPHICS) {
  861:                    update_curpos(vc);
  862:            }
  863:            wakeup(&tty_write);
  864: }
  865:
  866: void vconsole_select(int new_cons)
  867: {
  868:            new_cons++;
  869:            if(current_cons != new_cons) {
  870:                    if(vc[current_cons].vt_mode.mode == VT_PROCESS) {
  871:                            if(!kill_pid(vc[current_cons].tty->pid, vc[current_cons]
.vt_mode.acqsig)) {
  872:                                    vc[current_cons].switchto_tty = new_cons;
  873:                                    return;
  874:                            }
  875:                            init_vt(&vc[current_cons]);
  876:                    }
  877:                    if(vc[current_cons].vc_mode == KD_GRAPHICS) {
  878:                            return;
  879:                    }
  880:                    vconsole_select_final(new_cons);
  881:            }
  882: }
  883:
  884: void vconsole_select_final(int new_cons)
  885: {
  886:            if(current_cons != new_cons) {
  887:                    if(vc[new_cons].vt_mode.mode == VT_PROCESS) {
  888:                            if(kill_pid(vc[new_cons].tty->pid, vc[new_cons].vt_mode.
acqsig)) {
  889:                                    init_vt(&vc[new_cons]);
  890:                            }
  891:                    }
  892:                    if(buf_top) {
  893:                            vconsole_restore(&vc[current_cons]);
  894:                            buf_top = 0;
  895:                            update_curpos(&vc[current_cons]);
  896:                    }
  897:                    if(vc[current_cons].vc_mode != KD_GRAPHICS) {
  898:                            vconsole_save(&vc[current_cons]);
  899:                    }
  900:                    vc[current_cons].vidmem = vc[current_cons].scrbuf;
  901:                    vc[current_cons].has_focus = 0;
  902:                    vc[new_cons].vidmem = video_base_address;
  903:                    vc[new_cons].has_focus = 1;
  904:                    vconsole_restore(&vc[new_cons]);
  905:                    current_cons = new_cons;
  906:                    set_leds(vc[current_cons].led_status);
  907:                    update_curpos(&vc[current_cons]);
  908:
  909:                    buf_y = vc[current_cons].y;
  910:                    buf_top = 0;
  911:                    memset_w(vcbuf, BLANK_MEM, VC_BUF_SIZE / sizeof(unsigned short i
nt));
  912:                    memcpy_b(vcbuf, vc[current_cons].vidmem, SCREEN_SIZE);
  913:                    show_cursor(ON);
  914:            }
  915: }
```

drivers/char/console.c


```
916:
917: void vconsole_save(struct vconsole *vc)
918: {
919:         memcpy_b(vc->scrbuf, vc->vidmem, SCREEN_SIZE);
920: }
921:
922: void vconsole_restore(struct vconsole *vc)
923: {
924:         memcpy_b(vc->vidmem, vc->scrbuf, SCREEN_SIZE);
925: }
926:
927: void vconsole_buffer_scrl(int mode)
928: {
929:         int buf_line = buf_y;
930:
931:         if(buf_line <= SCREEN_LINES) {
932:                 return;
933:         }
934:         if(mode == VC_BUF_UP) {
935:                 if(buf_top < 0) {
936:                         return;
937:                 }
938:                 if(!buf_top) {
939:                         vconsole_save(&vc[current_cons]);
940:                         buf_top = (buf_line - SCREEN_LINES + 1) * SCREEN_COLS;
941:                         buf_top -= (SCREEN_LINES / 2) * SCREEN_COLS;
942:                 } else {
943:                         buf_top -= (SCREEN_LINES / 2) * SCREEN_COLS;
944:                 }
945:                 if(buf_top < 0) {
946:                         buf_top = 0;
947:                 }
948:                 memcpy_b(vc[current_cons].vidmem, vcbuf + buf_top, SCREEN_SIZE);
949:                 if(!buf_top) {
950:                         buf_top = -1;
951:                 }
952:                 show_cursor(OFF);
953:                 return;
954:         }
955:         if(mode == VC_BUF_DOWN) {
956:                 if(!buf_top) {
957:                         return;
958:                 }
959:                 if(buf_top == buf_line * SCREEN_COLS) {
960:                         return;
961:                 }
962:                 if(buf_top < 0) {
963:                         buf_top = 0;
964:                 }
965:                 buf_top += (SCREEN_LINES / 2) * SCREEN_COLS;
966:                 if(buf_top >= (buf_line - SCREEN_LINES + 1) * SCREEN_COLS) {
967:                         vconsole_restore(&vc[current_cons]);
968:                         buf_top = 0;
969:                         show_cursor(ON);
970:                         update_curpos(&vc[current_cons]);
971:                         return;
972:                 }
973:                 memcpy_b(vc[current_cons].vidmem, vcbuf + buf_top, SCREEN_SIZE);
974:                 return;
975:         }
976: }
977:
978: void blank_screen(struct vconsole *vc)
979: {
980:         if(vc->blanked) {
981:                 return;
982:         }
```

drivers/char/console.c


```
 983:         vconsole_save(vc);
 984:         memset_w(vc->vidmem, BLANK_MEM, SCREEN_SIZE / sizeof(short int));
 985:         vc->blanked = 1;
 986:         show_cursor(OFF);
 987: }
 988:
 989: void unblank_screen(struct vconsole *vc)
 990: {
 991:         if(!vc->blanked) {
 992:                 return;
 993:         }
 994:         vconsole_restore(vc);
 995:         vc->blanked = 0;
 996:         show_cursor(ON);
 997: }
 998:
 999: void screen_on(void)
1000: {
1001:         unsigned long int flags;
1002:         struct callout_req creq;
1003:
1004:         if(screen_is_off) {
1005:                 SAVE_FLAGS(flags); CLI();
1006:                 inport_b(INPUT_STAT1);
1007:                 inport_b(0x3BA);
1008:                 outport_b(ATTR_CONTROLLER, ATTR_CONTROLLER_PAS);
1009:                 RESTORE_FLAGS(flags);
1010:         }
1011:         creq.fn = screen_off;
1012:         creq.arg = 0;
1013:         add_callout(&creq, BLANK_INTERVAL);
1014: }
1015:
1016: void screen_off(unsigned int arg)
1017: {
1018:         unsigned long int flags;
1019:
1020:         screen_is_off = 1;
1021:         SAVE_FLAGS(flags); CLI();
1022:         inport_b(INPUT_STAT1);
1023:         inport_b(0x3BA);
1024:         outport_b(ATTR_CONTROLLER, 0);
1025:         RESTORE_FLAGS(flags);
1026: }
1027:
1028: void vconsole_start(struct tty *tty)
1029: {
1030:         struct vconsole *vc;
1031:
1032:         vc = (struct vconsole *)tty->driver_data;
1033:         if(!vc->scrlock) {
1034:                 return;
1035:         }
1036:         vc->led_status &= ~SCRLBIT;
1037:         vc->scrlock = 0;
1038:         set_leds(vc->led_status);
1039: }
1040:
1041: void vconsole_stop(struct tty *tty)
1042: {
1043:         struct vconsole *vc;
1044:
1045:         vc = (struct vconsole *)tty->driver_data;
1046:         if(vc->scrlock) {
1047:                 return;
1048:         }
1049:         vc->led_status |= SCRLBIT;
```

drivers/char/console.c


```
 1050:         vc->scrlock = 1;
 1051:         set_leds(vc->led_status);
 1052: }
 1053:
 1054: void vconsole_beep(void)
 1055: {
 1056:         struct callout_req creq;
 1057:
 1058:         pit_beep_on();
 1059:         creq.fn = pit_beep_off;
 1060:         creq.arg = 0;
 1061:         add_callout(&creq, HZ / 8);
 1062: }
 1063:
 1064: void vconsole_deltab(struct tty *tty)
 1065: {
 1066:         unsigned short int col, n;
 1067:         unsigned char count;
 1068:         struct vconsole *vc;
 1069:         struct cblock *cb;
 1070:         unsigned char ch;
 1071:
 1072:         vc = (struct vconsole *)tty->driver_data;
 1073:         cb = tty->cooked_q.head;
 1074:         col = count = 0;
 1075:
 1076:         while(cb) {
 1077:                 for(n = 0; n < cb->end_off; n++) {
 1078:                         if(n >= cb->start_off) {
 1079:                                 ch = cb->data[n];
 1080:                                 if(ch == '\t') {
 1081:                                         while(!vc->tab_stop[++col]);
 1082:                                 } else {
 1083:                                         col++;
 1084:                                         if(ISCNTRL(ch) && !ISSPACE(ch) && tty->t
ermios.c_lflag & ECHOCTL) {
 1085:                                                 col++;
 1086:                                         }
 1087:                                 }
 1088:                                 col %= vc->columns;
 1089:                         }
 1090:                 }
 1091:                 cb = cb->next;
 1092:         }
 1093:         count = vc->x - col;
 1094:
 1095:         while(count--) {
 1096:                 tty_queue_putchar(tty, &tty->write_q, '\b');
 1097:         }
 1098: }
 1099:
 1100: void console_flush_log_buf(char *buffer, unsigned int count)
 1101: {
 1102:         char *b;
 1103:         struct tty *tty;
 1104:
 1105:         tty = get_tty(_syscondev);
 1106:         b = buffer;
 1107:
 1108:         while(count) {
 1109:                 if(tty_queue_putchar(tty, &tty->write_q, *b) < 0) {
 1110:                         tty->output(tty);
 1111:                         continue;
 1112:                 }
 1113:                 count--;
 1114:                 b++;
 1115:         }
```

drivers/char/console.c


```
 1116:          tty->output(tty);
 1117: }
 1118:
 1119: void vconsole_init(void)
 1120: {
 1121:          int n;
 1122:          struct tty *tty;
 1123:
 1124:          printk("console");
 1125:          if((*(unsigned short int *)0x410 & 0x30) == 0x30) {
 1126:                  /* monochrome = 0x30 */
 1127:                  video_base_address = (void *)MONO_ADDR;
 1128:                  video_port = MONO_6845_ADDR;
 1129:                  printk("   0x%04X-0x%04X    -    VGA monochrome 80x25", video_po
rt, video_port + 1);
 1130:          } else {
 1131:                  /* color = 0x00 || 0x20 */
 1132:                  video_base_address = (void *)COLOR_ADDR;
 1133:                  video_port = COLOR_6845_ADDR;
 1134:                  printk("   0x%04X-0x%04X    -    VGA color 80x25", video_port, v
ideo_port + 1);
 1135:          }
 1136:
 1137:          printk(" (%d virtual consoles)\n", NR_VCONSOLES);
 1138:          screen_on();
 1139:
 1140:          for(n = 1; n < NR_VCONSOLES + 1; n++) {
 1141:                  if(!register_tty(MKDEV(VCONSOLES_MAJOR, n))) {
 1142:                          tty = get_tty(MKDEV(VCONSOLES_MAJOR, n));
 1143:                          tty->driver_data = (void *)&vc[n];
 1144:                          tty->stop = vconsole_stop;
 1145:                          tty->start = vconsole_start;
 1146:                          tty->deltab = vconsole_deltab;
 1147:                          tty->reset = vconsole_reset;
 1148:                          tty->input = do_cook;
 1149:                          tty->output = vconsole_write;
 1150:                          vc[n].tty = tty;
 1151:                          vc[n].vidmem = vc[n].scrbuf;
 1152:                          memset_w(vc[n].scrbuf, BLANK_MEM, SCREEN_SIZE / sizeof(s
hort int));
 1153:                          vconsole_reset(tty);
 1154:                          termios_reset(tty);
 1155:                          tty_queue_init(tty);
 1156:                  }
 1157:          }
 1158:          current_cons = 1;
 1159:          vc[current_cons].vidmem = video_base_address;
 1160:          vc[current_cons].has_focus = 1;
 1161:
 1162:          memcpy_b(vcbuf, vc[current_cons].vidmem, SCREEN_SIZE);
 1163:
 1164:          get_curpos(&vc[current_cons]);
 1165:          update_curpos(&vc[current_cons]);
 1166:          buf_y = vc[current_cons].y;
 1167:          buf_top = 0;
 1168:
 1169:          register_device(CHR_DEV, &console_device);
 1170:          register_device(CHR_DEV, &tty_device);
 1171:          register_console(console_flush_log_buf);
 1172: }
```

drivers/char/defkeymap.c


```
    1: /*
    2:  * fiwix/drivers/char/defkeymap.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/keyboard.h>
    9: #include <fiwix/string.h>
   10:
   11: #define BS      127     /* backspace */
   12:
   13: __key_t keymap[NR_MODIFIERS * NR_SCODES] = {
   14: /*
   15:  * Standard US keyboard (default keymap) with 16 modifiers
   16:  *
                                                        Shift
   17:  *
Shift                          Shift           Shift   AltGr   AltGr
   18:  * SCAN                                          Shift           Shift   AltGr
AltGr           Shift   AltGr   AltGr   Ctrl    Ctrl    Ctrl    Ctrl
   19:  * CODE  KEY             Base    Shift   AltGr   AltGr   Ctrl    Ctrl    Ctrl
Ctrl    Alt     Alt     Alt     Alt     Alt     Alt     Alt     Alt
   20:  * ================================================================================
================================================================ */
   21: /*  00 - NULL    */      0,      0,      0,      0,      0,      0,      0,
0,      0,      0,      0,      0,      0,      0,      0,      0,
   22: /*  01 - ESC     */      C('['), C('['), NULL,   NULL,   NULL,   0,      0,
0,      A('['), 0,      0,      0,      0,      0,      0,      0,
   23: /*  02 - 1       */      '1',    '!',    NULL,   NULL,   NULL,   0,      0,
0,      A('1'), 0,      0,      0,      0,      0,      0,      0,
   24: /*  03 - 2       */      '2',    '@',    '@',    NULL,   NULL,   0,      0,
0,      A('2'), 0,      0,      0,      0,      0,      0,      0,
   25: /*  04 - 3       */      '3',    '#',    NULL,   NULL,   C('['), 0,      0,
0,      A('3'), 0,      0,      0,      0,      0,      0,      0,
   26: /*  05 - 4       */      '4',    '$',    '$',    NULL,   C('\\'),0,      0,
0,      A('4'), 0,      0,      0,      0,      0,      0,      0,
   27: /*  06 - 5       */      '5',    '%',    NULL,   NULL,   C(']'), 0,      0,
0,      A('5'), 0,      0,      0,      0,      0,      0,      0,
   28: /*  07 - 6       */      '6',    '^',    NULL,   NULL,   C('^'), 0,      0,
0,      A('6'), 0,      0,      0,      0,      0,      0,      0,
   29: /*  08 - 7       */      '7',    '&',    '{',    NULL,   C('_'), 0,      0,
0,      A('7'), 0,      0,      0,      0,      0,      0,      0,
   30: /*  09 - 8       */      '8',    '*',    '[',    NULL,   BS,     0,      0,
0,      A('8'), 0,      0,      0,      0,      0,      0,      0,
   31: /*  10 - 9       */      '9',    '(',    ']',    NULL,   NULL,   0,      0,
0,      A('9'), 0,      0,      0,      0,      0,      0,      0,
   32: /*  11 - 0       */      '0',    ')',    '}',    NULL,   NULL,   0,      0,
0,      A('0'), 0,      0,      0,      0,      0,      0,      0,
   33: /*  12 - -_      */      '-',    '_',    '\\',   NULL,   C('_'), 0,      0,
0,      A('-'), 0,      0,      0,      0,      0,      0,      0,
   34: /*  13 - =+      */      '=',    '+',    NULL,   NULL,   NULL,   0,      0,
0,      A('='), 0,      0,      0,      0,      0,      0,      0,
   35: /*  14 - BS      */      BS,     BS,     NULL,   NULL,   NULL,   0,      0,
0,      A(BS),  0,      0,      0,      0,      0,      0,      0,
   36: /*  15 - TAB     */      '\t',   '\t',   NULL,   NULL,   NULL,   0,      0,
0,      A('\t'),0,      0,      0,      0,      0,      0,      0,
   37: /*  16 - q       */      L('q'), L('Q'), L('q'), L('Q'), C('Q'), 0,      0,
0,      A('q'), 0,      0,      0,      0,      0,      0,      0,
   38: /*  17 - w       */      L('w'), L('W'), L('w'), L('W'), C('W'), 0,      0,
0,      A('w'), 0,      0,      0,      0,      0,      0,      0,
   39: /*  18 - e       */      L('e'), L('E'), L('e'), L('E'), C('E'), 0,      0,
0,      A('e'), 0,      0,      0,      0,      0,      0,      0,
   40: /*  19 - r       */      L('r'), L('R'), L('r'), L('R'), C('R'), 0,      0,
0,      A('r'), 0,      0,      0,      0,      0,      0,      0,
   41: /*  20 - t       */      L('t'), L('T'), L('t'), L('T'), C('T'), 0,      0,
0,      A('t'), 0,      0,      0,      0,      0,      0,      0,
```

drivers/char/defkeymap.c


```
    42: /*  21 - y      */      L('y'), L('Y'), L('y'), L('Y'), C('Y'), 0,      0,
0,      A('y'), 0,      0,      0,      0,      0,      0,      0,
    43: /*  22 - u      */      L('u'), L('U'), L('u'), L('U'), C('U'), 0,      0,
0,      A('u'), 0,      0,      0,      0,      0,      0,      0,
    44: /*  23 - i      */      L('i'), L('I'), L('i'), L('I'), C('I'), 0,      0,
0,      A('i'), 0,      0,      0,      0,      0,      0,      0,
    45: /*  24 - o      */      L('o'), L('O'), L('o'), L('O'), C('O'), 0,      0,
0,      A('o'), 0,      0,      0,      0,      0,      0,      0,
    46: /*  25 - p      */      L('p'), L('P'), L('p'), L('P'), C('P'), 0,      0,
0,      A('p'), 0,      0,      0,      0,      0,      0,      0,
    47: /*  26 - [{     */      '[',    '{',    NULL,   NULL,   C('['), 0,      0,
0,      A('['), 0,      0,      0,      0,      0,      0,      0,
    48: /*  27 - ]}     */      ']',    '}',    '~',    NULL,   C(']'), 0,      0,
0,      A(']'), 0,      0,      0,      0,      0,      0,      0,
    49: /*  28 - CR     */      CR,     CR,     CR,     CR,     CR,     0,      0,
0,      A(CR),  0,      0,      0,      0,      0,      0,      0,
    50: /*  29 - CTRL   */      CTRL,   CTRL,   CTRL,   CTRL,   CTRL,   0,      0,
0,      CTRL,   0,      0,      0,      0,      0,      0,      0,
    51: /*  30 - a      */      L('a'), L('A'), L('a'), L('A'), C('A'), 0,      0,
0,      A('a'), 0,      0,      0,      0,      0,      0,      0,
    52: /*  31 - s      */      L('s'), L('S'), L('s'), L('S'), C('S'), 0,      0,
0,      A('s'), 0,      0,      0,      0,      0,      0,      0,
    53: /*  32 - d      */      L('d'), L('D'), L('d'), L('D'), C('D'), 0,      0,
0,      A('d'), 0,      0,      0,      0,      0,      0,      0,
    54: /*  33 - f      */      L('f'), L('F'), L('f'), L('F'), C('F'), 0,      0,
0,      A('f'), 0,      0,      0,      0,      0,      0,      0,
    55: /*  34 - g      */      L('g'), L('G'), L('g'), L('G'), C('G'), 0,      0,
0,      A('g'), 0,      0,      0,      0,      0,      0,      0,
    56: /*  35 - h      */      L('h'), L('H'), L('h'), L('H'), C('H'), 0,      0,
0,      A('h'), 0,      0,      0,      0,      0,      0,      0,
    57: /*  36 - j      */      L('j'), L('J'), L('j'), L('J'), C('J'), 0,      0,
0,      A('j'), 0,      0,      0,      0,      0,      0,      0,
    58: /*  37 - k      */      L('k'), L('K'), L('k'), L('K'), C('K'), 0,      0,
0,      A('k'), 0,      0,      0,      0,      0,      0,      0,
    59: /*  38 - l      */      L('l'), L('L'), L('l'), L('L'), C('L'), 0,      0,
0,      A('l'), 0,      0,      0,      0,      0,      0,      0,
    60: /*  39 - ;:     */      ';',    ':',    NULL,   NULL,   NULL,   0,      0,
0,      A(';'), 0,      0,      0,      0,      0,      0,      0,
    61: /*  40 - '"     */      '\'',   '"',    NULL,   NULL,   C('G'), 0,      0,
0,      A('\''),0,      0,      0,      0,      0,      0,      0,
    62: /*  41 - `~     */      '`',    '~',    NULL,   NULL,   NULL,   0,      0,
0,      A('`'), 0,      0,      0,      0,      0,      0,      0,
    63: /*  42 - LSHF   */      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  0,      0,
0,      SHIFT,  0,      0,      0,      0,      0,      0,      0,
    64: /*  43 - \|     */      '\\',   '|',    NULL,   NULL,   C('\\'),0,      0,
0,      A('\\'),0,      0,      0,      0,      0,      0,      0,
    65: /*  44 - z      */      L('z'), L('Z'), L('z'), L('Z'), C('Z'), 0,      0,
0,      A('z'), 0,      0,      0,      0,      0,      0,      0,
    66: /*  45 - x      */      L('x'), L('X'), L('x'), L('X'), C('X'), 0,      0,
0,      A('x'), 0,      0,      0,      0,      0,      0,      0,
    67: /*  46 - c      */      L('c'), L('C'), L('c'), L('C'), C('C'), 0,      0,
0,      A('c'), 0,      0,      0,      0,      0,      0,      0,
    68: /*  47 - v      */      L('v'), L('V'), L('v'), L('V'), C('V'), 0,      0,
0,      A('v'), 0,      0,      0,      0,      0,      0,      0,
    69: /*  48 - b      */      L('b'), L('B'), L('b'), L('B'), C('B'), 0,      0,
0,      A('b'), 0,      0,      0,      0,      0,      0,      0,
    70: /*  49 - n      */      L('n'), L('N'), L('n'), L('N'), C('N'), 0,      0,
0,      A('n'), 0,      0,      0,      0,      0,      0,      0,
    71: /*  50 - m      */      L('m'), L('M'), L('m'), L('M'), C('M'), 0,      0,
0,      A('m'), 0,      0,      0,      0,      0,      0,      0,
    72: /*  51 - ,<     */      ',',    '<',    NULL,   NULL,   NULL,   0,      0,
0,      A(','), 0,      0,      0,      0,      0,      0,      0,
    73: /*  52 - .>     */      '.',    '>',    NULL,   NULL,   NULL,   0,      0,
0,      A('.'), 0,      0,      0,      0,      0,      0,      0,
    74: /*  53 - /?     */      SLASH,  '?',    NULL,   NULL,   BS,     0,      0,
0,      A('/'), 0,      0,      0,      0,      0,      0,      0,
    75: /*  54 - RSHF   */      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  0,      0,
```

drivers/char/defkeymap.c

```
0,      SHIFT,  0,      0,      0,      0,      0,      0,      0,
   76: /*  55 - *      */      ASTSK,  ASTSK,  ASTSK,  ASTSK,  ASTSK,  0,      0,
0,      ASTSK,  0,      0,      0,      0,      0,      0,      0,
   77: /*  56 - ALT    */      ALT,    ALT,    ALT,    ALT,    ALT,    0,      0,
0,      ALT,    0,      0,      0,      0,      0,      0,      0,
   78: /*  57 - SPC    */      ' ',    ' ',    NULL,   NULL,   NULL,   0,      0,
0,      A(' '), 0,      0,      0,      0,      0,      0,      0,
   79: /*  58 - CAPS   */      CAPS,   CAPS,   CAPS,   CAPS,   CAPS,   0,      0,
0,      CAPS,   0,      0,      0,      0,      0,      0,      0,
   80: /*  59 - F1     */      F1,     SF1,    NULL,   NULL,   F1,     0,      0,
0,      AF1,    0,      0,      0,      0,      0,      0,      0,
   81: /*  60 - F2     */      F2,     SF2,    NULL,   NULL,   F2,     0,      0,
0,      AF2,    0,      0,      0,      0,      0,      0,      0,
   82: /*  61 - F3     */      F3,     SF3,    NULL,   NULL,   F3,     0,      0,
0,      AF3,    0,      0,      0,      0,      0,      0,      0,
   83: /*  62 - F4     */      F4,     SF4,    NULL,   NULL,   F4,     0,      0,
0,      AF4,    0,      0,      0,      0,      0,      0,      0,
   84: /*  63 - F5     */      F5,     SF5,    NULL,   NULL,   F5,     0,      0,
0,      AF5,    0,      0,      0,      0,      0,      0,      0,
   85: /*  64 - F6     */      F6,     SF6,    NULL,   NULL,   F6,     0,      0,
0,      AF6,    0,      0,      0,      0,      0,      0,      0,
   86: /*  65 - F7     */      F7,     SF7,    NULL,   NULL,   F7,     0,      0,
0,      AF7,    0,      0,      0,      0,      0,      0,      0,
   87: /*  66 - F8     */      F8,     SF8,    NULL,   NULL,   F8,     0,      0,
0,      AF8,    0,      0,      0,      0,      0,      0,      0,
   88: /*  67 - F9     */      F9,     SF9,    NULL,   NULL,   F9,     0,      0,
0,      AF9,    0,      0,      0,      0,      0,      0,      0,
   89: /*  68 - F10    */      F10,    SF10,   NULL,   NULL,   F10,    0,      0,
0,      AF10,   0,      0,      0,      0,      0,      0,      0,
   90: /*  69 - NUMS   */      NUMS,   NUMS,   NUMS,   NUMS,   NUMS,   0,      0,
0,      NUMS,   0,      0,      0,      0,      0,      0,      0,
   91: /*  70 - SCRL   */      SCRL,   SCRL3,  SCRL2,  NULL,   SCRL4,  0,      0,
0,      SCRL,   0,      0,      0,      0,      0,      0,      0,
   92: /*  71 - HOME/7 */      HOME,   HOME,   HOME,   HOME,   HOME,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
   93: /*  72 - UP  /8 */      UP,     UP,     UP,     UP,     UP,     0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
   94: /*  73 - PGUP/9 */      PGUP,   PGUP,   PGUP,   PGUP,   PGUP,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
   95: /*  74 - MINUS  */      MINUS,  MINUS,  MINUS,  MINUS,  MINUS,  0,      0,
0,      MINUS,  0,      0,      0,      0,      0,      0,      0,
   96: /*  75 - LEFT/4 */      LEFT,   LEFT,   LEFT,   LEFT,   LEFT,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
   97: /*  76 - MID /5 */      MID,    MID,    MID,    MID,    MID,    0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
   98: /*  77 - RIGH/6 */      RIGHT,  RIGHT,  RIGHT,  RIGHT,  RIGHT,  0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
   99: /*  78 - PLUS   */      PLUS,   PLUS,   PLUS,   PLUS,   PLUS,   0,      0,
0,      PLUS,   0,      0,      0,      0,      0,      0,      0,
  100: /*  79 - END /1 */      END,    END,    END,    END,    END,    0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  101: /*  80 - DOWN/2 */      DOWN,   DOWN,   DOWN,   DOWN,   DOWN,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  102: /*  81 - PGDN/3 */      PGDN,   PGDN,   PGDN,   PGDN,   PGDN,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  103: /*  82 - INS /0 */      INS,    INS,    INS,    INS,    INS,    0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  104: /*  83 - DEL /. */      DEL,    DEL,    DEL,    DEL,    DEL,    0,      0,
0,      DEL,    0,      0,      0,      0,      0,      0,      0,
  105: /*  84 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  106: /*  85 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  107: /*  86 - <>     */      '<',    '>',    '|',    NULL,   NULL,   0,      0,
0,      A('<'), 0,      0,      0,      0,      0,      0,      0,
  108: /*  87 - F11    */      SF1,    SF1,    NULL,   NULL,   F11,    0,      0,
0,      AF11,   0,      0,      0,      0,      0,      0,      0,
```

drivers/char/defkeymap.c


```
  109: /*  88 - F12    */      SF2,    SF2,    NULL,   NULL,   F12,    0,      0,
0,      AF12,   0,      0,      0,      0,      0,      0,      0,
  110: /*  89 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  111: /*  90 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  112: /*  91 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  113: /*  92 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  114: /*  93 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  115: /*  94 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  116: /*  95 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  117: /*  96 -        */      ENTER,  ENTER,  ENTER,  ENTER,  ENTER,  0,      0,
0,      ENTER,  0,      0,      0,      0,      0,      0,      0,
  118: /*  97 -        */      CTRL,   CTRL,   CTRL,   CTRL,   CTRL,   0,      0,
0,      CTRL,   0,      0,      0,      0,      0,      0,      0,
  119: /*  98 -        */      SLASH,  SLASH,  SLASH,  SLASH,  SLASH,  0,      0,
0,      SLASH,  0,      0,      0,      0,      0,      0,      0,
  120: /*  99 -        */      NULL,   NULL,   NULL,   NULL,   C('\\'),0,      0,
0,      C('\\'),0,      0,      0,      0,      0,      0,      0,
  121: /* 100 -        */      ALTGR,  ALTGR,  ALTGR,  ALTGR,  ALTGR,  0,      0,
0,      ALTGR,  0,      0,      0,      0,      0,      0,      0,
  122: /* 101 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  123: /* 102 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  124: /* 103 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  125: /* 104 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  126: /* 105 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  127: /* 106 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  128: /* 107 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  129: /* 108 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  130: /* 109 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  131: /* 110 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  132: /* 111 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  133: /* 112 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  134: /* 113 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  135: /* 114 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  136: /* 115 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  137: /* 116 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  138: /* 117 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  139: /* 118 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  140: /* 119 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  141: /* 120 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  142: /* 121 -        */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
```

drivers/char/defkeymap.c


```
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  143: /* 122 -          */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  144: /* 123 -          */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  145: /* 124 -          */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  146: /* 125 -          */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  147: /* 126 -          */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  148: /* 127 -          */      NULL,   NULL,   NULL,   NULL,   NULL,   0,      0,
0,      NULL,   0,      0,      0,      0,      0,      0,      0,
  149: };
```

drivers/char/keyboard.c


```
  1: /*
  2:  * fiwix/drivers/char/keyboard.c
  3:  *
  4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
  5:  * Distributed under the terms of the Fiwix License.
  6:  */
  7:
  8: #include <fiwix/asm.h>
  9: #include <fiwix/kernel.h>
 10: #include <fiwix/limits.h>
 11: #include <fiwix/keyboard.h>
 12: #include <fiwix/console.h>
 13: #include <fiwix/pic.h>
 14: #include <fiwix/signal.h>
 15: #include <fiwix/process.h>
 16: #include <fiwix/sleep.h>
 17: #include <fiwix/kd.h>
 18: #include <fiwix/stdio.h>
 19: #include <fiwix/string.h>
 20:
 21: #define KB_DATA         0x60    /* I/O data port */
 22: #define KBC_COMMAND     0x64    /* command/control port */
 23: #define KBC_STATUS      0x64    /* status register port */
 24:
 25: /*
 26:  * PS/2 System Control Port A
 27:  * --------------------------------
 28:  * bit 7 -> fixed disk activity led
 29:  * bit 6 -> fixed disk activity led
 30:  * bit 5 -> reserved
 31:  * bit 4 -> watchdog timer status
 32:  * bit 3 -> security lock latch
 33:  * bit 2 -> reserved
 34:  * bit 1 -> alternate gate A20
 35:  * bit 0 -> alternate hot reset
 36:  */
 37: #define PS2_SYSCTRL_A   0x92    /* PS/2 system control port A (write) */
 38:
 39: #define KB_CMD_RESET    0xFF    /* keyboard reset */
 40: #define KB_CMD_ENABLE   0xF4    /* keyboard enable scanning */
 41: #define KB_CMD_DISABLE  0xF5    /* keyboard disable scanning */
 42: #define KB_CMD_IDENTIFY 0xF2    /* keyboard identify (for PS/2 only) */
 43: #define KB_CMD_ECHO     0xEE    /* echo (for diagnostics only) */
 44:
 45: #define KBC_CMD_RECV_CONFIG     0x20    /* read controller's config byte */
 46: #define KBC_CMD_SEND_CONFIG     0x60    /* write controller's config byte */
 47: #define KBC_CMD_SELF_TEST       0xAA    /* self-test command */
 48: #define KBC_CMD_PS2_1_TEST      0xAB    /* first PS/2 interface test command */
 49: #define KBC_CMD_PS2_2_TEST      0xA9    /* second PS/2 interface test command */
 50: #define KBC_CMD_DISABLE_PS2_1   0xAD    /* disable first PS/2 port */
 51: #define KBC_CMD_ENABLE_PS2_1    0xAE    /* enable first PS/2 port */
 52: #define KBC_CMD_DISABLE_PS2_2   0xA7    /* disable second PS/2 port (if any) */
 53: #define KBC_CMD_ENABLE_PS2_2    0xA8    /* enable second PS/2 port (if any) */
 54: #define KBC_CMD_GET_IFACE       0xCA    /* get interface type (AT or MCA) */
 55: #define KBC_CMD_HOTRESET        0xFE    /* Hot Reset */
 56:
 57: /* flags of the status register */
 58: #define KB_STR_OUTBUSY  0x01    /* output buffer full, don't read yet */
 59: #define KB_STR_INBUSY   0x02    /* input buffer full, don't write yet */
 60: #define KB_STR_TXTMOUT  0x20    /* transmit time-out error */
 61: #define KB_STR_RXTMOUT  0x40    /* receive time-out error */
 62: #define KB_STR_PARERR   0X80    /* parity error */
 63: #define KB_STR_COMMERR  (KB_STR_TXTMOUT | KB_STR_RXTMOUT)
 64:
 65: #define KB_RESET_OK     0xAA    /* self-test passed */
 66: #define KB_ACK          0xFA    /* acknowledge */
 67: #define KB_SETLED       0xED    /* set/reset status indicators (LEDs) */
```

drivers/char/keyboard.c


```
   68: #define KB_RATE         0xF3    /* set typematic rate/delay */
   69: #define DELAY_250       0x00    /* typematic delay at 250ms (default) */
   70: #define DELAY_500       0x40    /* typematic delay at 500ms */
   71: #define DELAY_750       0x80    /* typematic delay at 750ms */
   72: #define DELAY_1000      0xC0    /* typematic delay at 1000ms */
   73: #define RATE_30         0x00    /* typematic rate at 30.0 reports/sec (default)
*/
   74:
   75: #define EXTKEY          0xE0    /* extended key (AltGr, Ctrl-Print, etc.) */
   76:
   77: __key_t *keymap_line;
   78:
   79: static unsigned char leds = 0;
   80: static unsigned char shift = 0;
   81: static unsigned char altgr = 0;
   82: static unsigned char ctrl = 0;
   83: static unsigned char alt = 0;
   84: static unsigned char extkey = 0;
   85: static unsigned char deadkey = 0;
   86:
   87: static unsigned char do_scrl_buf = 0;
   88: static char do_switch_console = -1;
   89:
   90: unsigned char kb_identify[2] = {0, 0};
   91: char ps2_active_ports = 0;
   92: char ps2_supp_ports = 0;
   93: char ps2_iface = 0;
   94: short int current_cons;
   95: char ctrl_alt_del = 1;
   96: char any_key_to_reboot = 0;
   97:
   98: struct diacritic *diacr;
   99: static char *diacr_chars = "''^ \"";
  100: struct diacritic grave_table[NR_DIACR] = {
  101:         { 'A', '\300' },
  102:         { 'E', '\310' },
  103:         { 'I', '\314' },
  104:         { 'O', '\322' },
  105:         { 'U', '\331' },
  106:         { 'a', '\340' },
  107:         { 'e', '\350' },
  108:         { 'i', '\354' },
  109:         { 'o', '\362' },
  110:         { 'u', '\371' },
  111: };
  112: struct diacritic acute_table[NR_DIACR] = {
  113:         { 'A', '\301' },
  114:         { 'E', '\311' },
  115:         { 'I', '\315' },
  116:         { 'O', '\323' },
  117:         { 'U', '\332' },
  118:         { 'a', '\341' },
  119:         { 'e', '\351' },
  120:         { 'i', '\355' },
  121:         { 'o', '\363' },
  122:         { 'u', '\372' },
  123: };
  124: struct diacritic circm_table[NR_DIACR] = {
  125:         { 'A', '\302' },
  126:         { 'E', '\312' },
  127:         { 'I', '\316' },
  128:         { 'O', '\324' },
  129:         { 'U', '\333' },
  130:         { 'a', '\342' },
  131:         { 'e', '\352' },
  132:         { 'i', '\356' },
  133:         { 'o', '\364' },
```

drivers/char/keyboard.c


```
134:         { 'u', '\373' },
135: };
136: struct diacritic diere_table[NR_DIACR] = {
137:         { 'A', '\304' },
138:         { 'E', '\313' },
139:         { 'I', '\317' },
140:         { 'O', '\326' },
141:         { 'U', '\334' },
142:         { 'a', '\344' },
143:         { 'e', '\353' },
144:         { 'i', '\357' },
145:         { 'o', '\366' },
146:         { 'u', '\374' },
147: };
148:
149: static char *pad_chars = "0123456789+-*/\015,.";
150:
151: static char *pad_seq[] = {
152:         "\033[2~",      /* INS */
153:         "\033[4~",      /* END */
154:         "\033[B" ,      /* DOWN */
155:         "\033[6~",      /* PGDN */
156:         "\033[D" ,      /* LEFT */
157:         "\033[G" ,      /* MID */
158:         "\033[C" ,      /* RIGHT */
159:         "\033[1~",      /* HOME */
160:         "\033[A" ,      /* UP */
161:         "\033[5~",      /* PGUP */
162:         "+",            /* PLUS */
163:         "-",            /* MINUS */
164:         "*",            /* ASTERISK */
165:         "/",            /* SLASH */
166:         "'\n'",         /* ENTER */
167:         ",",            /* COMMA */
168:         "\033[3~",      /* DEL */
169: };
170:
171: static char *fn_seq[] = {
172:         "\033[[A",      /* F1 */
173:         "\033[[B",      /* F2 */
174:         "\033[[C",      /* F3 */
175:         "\033[[D",      /* F4 */
176:         "\033[[E",      /* F5 */
177:         "\033[17~",     /* F6 */
178:         "\033[18~",     /* F7 */
179:         "\033[19~",     /* F8 */
180:         "\033[20~",     /* F9 */
181:         "\033[21~",     /* F10 */
182:         "\033[23~",     /* F11, SF1 */
183:         "\033[24~",     /* F12, SF2 */
184:         "\033[25~",     /* SF3 */
185:         "\033[26~",     /* SF4 */
186:         "\033[28~",     /* SF5 */
187:         "\033[29~",     /* SF6 */
188:         "\033[31~",     /* SF7 */
189:         "\033[32~",     /* SF8 */
190:         "\033[33~",     /* SF9 */
191:         "\033[34~",     /* SF10 */
192: };
193:
194: static void keyboard_delay(void)
195: {
196:         int n;
197:
198:         for(n = 0; n < 1000; n++) {
199:                 NOP();
200:         }
```

drivers/char/keyboard.c


```
  201: }
  202:
  203: /* wait controller input buffer to be clear (ready to write) */
  204: static int keyboard_wait_input(void)
  205: {
  206:         int n;
  207:
  208:         for(n = 0; n < 500000; n++) {
  209:                 if(!(inport_b(KBC_STATUS) & KB_STR_INBUSY)) {
  210:                         return 0;
  211:                 }
  212:         }
  213:         return 1;
  214: }
  215:
  216: static int keyboard_write(const unsigned char port, const unsigned char byte)
  217: {
  218:         if(!keyboard_wait_input()) {
  219:                 outport_b(port, byte);
  220:                 if(!keyboard_wait_input()) {
  221:                         return 0;
  222:                 }
  223:         }
  224:
  225:         return 1;
  226: }
  227:
  228: /* wait controller output buffer to be full (ready to read) */
  229: static int keyboard_wait_output(void)
  230: {
  231:         int n, value;
  232:
  233:         for(n = 0; n < 500000; n++) {
  234:                 if((value = inport_b(KBC_STATUS)) & KB_STR_OUTBUSY) {
  235:                         if(value & (KB_STR_COMMERR | KB_STR_PARERR)) {
  236:                                 continue;
  237:                         }
  238:                         return 0;
  239:                 }
  240:         }
  241:         return 1;
  242: }
  243:
  244: static int keyboard_wait_ack(void)
  245: {
  246:         int n;
  247:
  248:         keyboard_wait_output();
  249:         for(n = 0; n < 1000; n++) {
  250:                 if(inport_b(KB_DATA) == KB_ACK) {
  251:                         return 0;
  252:                 }
  253:                 keyboard_delay();
  254:         }
  255:         return 1;
  256: }
  257:
  258: static void keyboard_identify(void)
  259: {
  260:         /* disable */
  261:         keyboard_write(KB_DATA, KB_CMD_DISABLE);
  262:         if(keyboard_wait_ack()) {
  263:                 printk("WARNING: %s(): ACK not received on disable command!\n",
__FUNCTION__);
  264:         }
  265:
  266:         /* identify */
```

drivers/char/keyboard.c


```
  267:          keyboard_write(KB_DATA, KB_CMD_IDENTIFY);
  268:          if(keyboard_wait_ack()) {
  269:                  printk("WARNING: %s(): ACK not received on identify command!\n",
 __FUNCTION__);
  270:          }
  271:          if(!keyboard_wait_output()) {
  272:                  kb_identify[0] = inport_b(KB_DATA);
  273:          }
  274:          if(!keyboard_wait_output()) {
  275:                  kb_identify[1] = inport_b(KB_DATA);
  276:          }
  277:
  278:          /* enable */
  279:          keyboard_write(KB_DATA, KB_CMD_ENABLE);
  280:          if(keyboard_wait_ack()) {
  281:                  printk("WARNING: %s(): ACK not received on enable command!\n", _
_FUNCTION__);
  282:          }
  283:          keyboard_wait_output();
  284:          inport_b(KB_DATA);
  285:
  286:          /* get the interface type */
  287:          keyboard_write(KBC_COMMAND, KBC_CMD_GET_IFACE);
  288:          keyboard_wait_output();
  289:          ps2_iface = inport_b(KB_DATA);
  290: }
  291:
  292: static void keyboard_reset(void)
  293: {
  294:          int errno;
  295:          unsigned char config;
  296:
  297:          /* disable device(s) */
  298:          keyboard_write(KBC_COMMAND, KBC_CMD_DISABLE_PS2_1);
  299:          keyboard_write(KBC_COMMAND, KBC_CMD_DISABLE_PS2_2);
  300:
  301:          /* flush buffers */
  302:          while(!keyboard_wait_output()) {
  303:                  inport_b(KB_DATA);
  304:          }
  305:
  306:          /* get controller configuration */
  307:          keyboard_write(KBC_COMMAND, KBC_CMD_RECV_CONFIG);
  308:          keyboard_wait_output();
  309:          config = inport_b(KB_DATA);
  310:          ps2_active_ports = config & 0x01 ? 1 : 0;
  311:          ps2_active_ports += config & 0x02 ? 1 : 0;
  312:          ps2_supp_ports = 1 + (config & 0x20 ? 1 : 0);
  313:
  314:          /* set controller configuration (disabling IRQs) */
  315:          /*
  316:          keyboard_write(KBC_COMMAND, KBC_CMD_SEND_CONFIG);
  317:          keyboard_write(KB_DATA, config & ~(0x01 | 0x02 | 0x40));
  318:          */
  319:
  320:          /* PS/2 controller self-test */
  321:          keyboard_write(KBC_COMMAND, KBC_CMD_SELF_TEST);
  322:          keyboard_wait_output();
  323:          if((errno = inport_b(KB_DATA)) != 0x55) {
  324:                  printk("WARNING: %s(): keyboard returned 0x%x in self-test.\n",
__FUNCTION__, errno);
  325:          }
  326:
  327:          /*
  328:           * This sets again the controller configuration since the previous
  329:           * step may also reset the PS/2 controller to its power-on defaults.
  330:           */
```

drivers/char/keyboard.c


```
  331:          keyboard_write(KBC_COMMAND, KBC_CMD_SEND_CONFIG);
  332:          keyboard_write(KB_DATA, config);
  333:
  334:          /* first PS/2 interface test */
  335:          keyboard_write(KBC_COMMAND, KBC_CMD_PS2_1_TEST);
  336:          keyboard_wait_output();
  337:          if((errno = inport_b(KB_DATA)) != 0) {
  338:                  printk("WARNING: %s(): keyboard returned 0x%x in first PS/2 inte
rface test.\n", __FUNCTION__, errno);
  339:          }
  340:
  341:          if(ps2_supp_ports > 1) {
  342:                  /* second PS/2 interface test */
  343:                  keyboard_write(KBC_COMMAND, KBC_CMD_PS2_2_TEST);
  344:                  keyboard_wait_output();
  345:                  if((errno = inport_b(KB_DATA)) != 0) {
  346:                          printk("WARNING: %s(): keyboard returned 0x%x in second
PS/2 interface test.\n", __FUNCTION__, errno);
  347:                  }
  348:          }
  349:
  350:          /* enable device(s) */
  351:          keyboard_write(KBC_COMMAND, KBC_CMD_ENABLE_PS2_1);
  352:          keyboard_write(KBC_COMMAND, KBC_CMD_ENABLE_PS2_2);
  353:
  354:          /* reset device(s) */
  355:          keyboard_write(KB_DATA, KB_CMD_RESET);
  356:          if(keyboard_wait_ack()) {
  357:                  printk("WARNING: %s(): ACK not received on reset command!\n", __
FUNCTION__);
  358:          }
  359:          if(!keyboard_wait_output()) {
  360:                  if((errno = inport_b(KB_DATA)) != KB_RESET_OK) {
  361:                          printk("WARNING: %s(): keyboard returned 0x%x in reset.\
n", __FUNCTION__, errno);
  362:                  }
  363:          }
  364:
  365:          return;
  366: }
  367:
  368: static void putc(struct tty *tty, unsigned char ch)
  369: {
  370:          if(tty_queue_putchar(tty, &tty->read_q, ch) < 0) {
  371:                  if(tty->termios.c_iflag & IMAXBEL) {
  372:                          vconsole_beep();
  373:                  }
  374:          }
  375: }
  376:
  377: static void puts(struct tty *tty, char *seq)
  378: {
  379:          char ch;
  380:
  381:          while((ch = *(seq++))) {
  382:                  putc(tty, ch);
  383:          }
  384: }
  385:
  386: void reboot(void)
  387: {
  388:          CLI();
  389:          keyboard_write(PS2_SYSCTRL_A, 0x01);            /* Fast Hot Reset */
  390:          keyboard_write(KBC_COMMAND, KBC_CMD_HOTRESET);  /* Hot Reset */
  391:          HLT();
  392: }
  393:
```

drivers/char/keyboard.c


```
394: void set_leds(unsigned char leds)
395: {
396:         keyboard_write(KB_DATA, KB_SETLED);
397:         keyboard_wait_ack();
398:
399:         keyboard_write(KB_DATA, leds);
400:         keyboard_wait_ack();
401: }
402:
403: void irq_keyboard(void)
404: {
405:         __key_t key, type;
406:         unsigned char scode, mod;
407:         struct tty *tty;
408:         struct vconsole *vc;
409:         unsigned char c;
410:         int n;
411:
412:         tty = get_tty(MKDEV(VCONSOLES_MAJOR, current_cons));
413:         vc = (struct vconsole *)tty->driver_data;
414:
415:         scode = inport_b(KB_DATA);
416:
417:         screen_on();
418:         add_bh(keyboard_bh);
419:
420:         if(scode == KB_ACK) {
421:                 return;
422:         }
423:
424:         if(scode == EXTKEY) {
425:                 extkey = 1;
426:                 return;
427:         }
428:
429:         key = keymap[NR_MODIFIERS * (scode & 0x7F)];
430:
431:         /* a key has been released */
432:         if(scode & NR_SCODES) {
433:                 switch(key) {
434:                         case CTRL:
435:                                 ctrl = 0;
436:                                 break;
437:                         case ALT:
438:                                 if(!extkey) {
439:                                         alt = 0;
440:                                 } else {
441:                                         altgr = 0;
442:                                 }
443:                                 break;
444:                         case SHIFT:
445:                                 if(!extkey) {
446:                                         shift = 0;
447:                                 }
448:                                 break;
449:                         case CAPS:
450:                         case NUMS:
451:                         case SCRL:
452:                                 leds = 0;
453:                                 break;
454:                 }
455:                 extkey = 0;
456:                 return;
457:         }
458:
459:         switch(key) {
460:                 case CAPS:
```

drivers/char/keyboard.c


```
461:                        if(!leds) {
462:                                vc->led_status ^= CAPSBIT;
463:                                vc->capslock = !vc->capslock;
464:                                set_leds(vc->led_status);
465:                        }
466:                        leds = 1;
467:                        return;
468:                case NUMS:
469:                        if(!leds) {
470:                                vc->led_status ^= NUMSBIT;
471:                                vc->numlock = !vc->numlock;
472:                                set_leds(vc->led_status);
473:                        }
474:                        leds = 1;
475:                        return;
476:                case SCRL:
477:                        if(!leds) {
478:                                if(vc->scrlock) {
479:                                        tty->start(tty);
480:                                } else {
481:                                        tty->stop(tty);
482:                                }
483:                        }
484:                        leds = 1;
485:                        return;
486:                case CTRL:
487:                        ctrl = 1;
488:                        return;
489:                case ALT:
490:                        if(!extkey) {
491:                                alt = 1;
492:                        } else {
493:                                altgr = 1;
494:                        }
495:                        return;
496:                case SHIFT:
497:                        shift = 1;
498:                        extkey = 0;
499:                        return;
500:        }
501:
502:        if(ctrl && alt && key == DEL) {
503:                if(ctrl_alt_del) {
504:                        reboot();
505:                } else {
506:                        send_sig(&proc_table[INIT], SIGINT);
507:                }
508:                return;
509:        }
510:
511:        keymap_line = &keymap[(scode & 0x7F) * NR_MODIFIERS];
512:        mod = 0;
513:
514:        if(vc->capslock && (keymap_line[MOD_BASE] & LETTER_KEYS)) {
515:                mod = !vc->capslock ? shift : vc->capslock - shift;
516:        } else {
517:                if(shift && !extkey) {
518:                        mod = 1;
519:                }
520:        }
521:        if(altgr) {
522:                mod = 2;
523:        }
524:        if(ctrl) {
525:                mod = 4;
526:        }
527:        if(alt) {
```

drivers/char/keyboard.c


```
528:                    mod = 8;
529:            }
530:
531:            key = keymap_line[mod];
532:
533:            if(key >= AF1 && key <= AF12) {
534:                    do_switch_console = key - CONS_KEYS;
535:                    return;
536:            }
537:
538:            if(shift && (key == PGUP)) {
539:                    do_scrl_buf = VC_BUF_UP;
540:                    return;
541:            }
542:
543:            if(shift && (key == PGDN)) {
544:                    do_scrl_buf = VC_BUF_DOWN;
545:                    return;
546:            }
547:
548:            if(extkey && (scode == SLASH_NPAD)) {
549:                    key = SLASH;
550:            }
551:
552:            if(any_key_to_reboot) {
553:                    reboot();
554:            }
555:
556:            if(tty->count) {
557:                    type = key & 0xFF00;
558:                    c = key & 0xFF;
559:
560:                    switch(type) {
561:                            case FN_KEYS:
562:                                    puts(tty, fn_seq[c]);
563:                                    break;
564:
565:                            case SPEC_KEYS:
566:                                    switch(key) {
567:                                            case CR:
568:                                                    putc(tty, C('M'));
569:                                                    break;
570:                                    }
571:                                    break;
572:
573:                            case PAD_KEYS:
574:                                    if(!vc->numlock) {
575:                                            puts(tty, pad_seq[c]);
576:                                    } else {
577:                                            putc(tty, pad_chars[c]);
578:                                    }
579:                                    break;
580:
581:                            case DEAD_KEYS:
582:                                    if(!deadkey) {
583:                                            switch(c) {
584:                                                    case GRAVE ^ DEAD_KEYS:
585:                                                            deadkey = 1;
586:                                                            diacr = grave_table;
587:                                                            break;
588:                                                    case ACUTE ^ DEAD_KEYS:
589:                                                            deadkey = 2;
590:                                                            diacr = acute_table;
591:                                                            break;
592:                                                    case CIRCM ^ DEAD_KEYS:
593:                                                            deadkey = 3;
594:                                                            diacr = circm_table;
```

drivers/char/keyboard.c


```
595:                                                    break;
596:                                            case DIERE ^ DEAD_KEYS:
597:                                                    deadkey = 5;
598:                                                    diacr = diere_table;
599:                                                    break;
600:                                            }
601:                                            return;
602:                                    }
603:                                    c = diacr_chars[c];
604:                                    deadkey = 0;
605:                                    putc(tty, c);
606:
607:                                    break;
608:
609:                            case META_KEYS:
610:                                    putc(tty, '\033');
611:                                    putc(tty, c);
612:                                    break;
613:
614:                            case LETTER_KEYS:
615:                                    if(deadkey) {
616:                                            for(n = 0; n < NR_DIACR; n++) {
617:                                                    if(diacr[n].letter == c) {
618:                                                            c = diacr[n].code;
619:                                                    }
620:                                            }
621:                                    }
622:                                    putc(tty, c);
623:                                    break;
624:
625:                            default:
626:                                    if(deadkey && c == ' ') {
627:                                            c = diacr_chars[deadkey - 1];
628:                                    }
629:                                    putc(tty, c);
630:                                    break;
631:                    }
632:            }
633:            deadkey = 0;
634:            return;
635: }
636:
637: void keyboard_bh(void)
638: {
639:            int n;
640:            struct tty *tty;
641:
642:            if(do_switch_console >= 0) {
643:                    vconsole_select(do_switch_console);
644:                    do_switch_console = -1;
645:            }
646:
647:            if(do_scrl_buf) {
648:                    vconsole_buffer_scrl(do_scrl_buf);
649:                    do_scrl_buf = 0;
650:            }
651:
652:            tty = &tty_table[0];
653:            for(n = 0; n < NR_TTYS; n++, tty++) {
654:                    if(!tty->read_q.count) {
655:                            continue;
656:                    }
657:                    if(lock_area(AREA_TTY_READ)) {
658:                            continue;
659:                    }
660:                    tty->input(tty);
661:                    unlock_area(AREA_TTY_READ);
```

drivers/char/keyboard.c


```
  662:          }
  663: }
  664:
  665: void keyboard_init(void)
  666: {
  667:          keyboard_reset();
  668:
  669:          /* flush buffers */
  670:          while(!keyboard_wait_output()) {
  671:                  inport_b(KB_DATA);
  672:          }
  673:
  674:          keyboard_identify();
  675:
  676:          keyboard_write(KB_DATA, KB_RATE);
  677:          keyboard_wait_ack();
  678:          keyboard_write(KB_DATA, DELAY_250 | RATE_30);
  679:          keyboard_wait_ack();
  680:
  681:          printk("keyboard  0x%04X-0x%04X    %d    type=%s %s PS/2 devices=%d/%d\n
", 0x60, 0x64, KEYBOARD_IRQ, kb_identify[0] == 0xAB ? "MF2" : "unknown", ps2_iface & 0x
1 ? "MCA" : "AT", ps2_active_ports, ps2_supp_ports);
  682:
  683:          if(!register_irq(KEYBOARD_IRQ, "keyboard", irq_keyboard)) {
  684:                  enable_irq(KEYBOARD_IRQ);
  685:          }
  686: }
```

drivers/char/lp.c


```
/*
 * fiwix/drivers/char/lp.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/asm.h>
#include <fiwix/devices.h>
#include <fiwix/fs.h>
#include <fiwix/errno.h>
#include <fiwix/lp.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct lp lp_table[LP_MINORS];

static struct fs_operations lp_driver_fsop = {
        0,
        0,

        lp_open,
        lp_close,
        NULL,                   /* read */
        lp_write,
        NULL,                   /* ioctl */
        NULL,                   /* lseek */
        NULL,                   /* readdir */
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        NULL,                   /* bmap */
        NULL,                   /* lockup */
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        NULL,                   /* truncate */
        NULL,                   /* create */
        NULL,                   /* rename */

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statfs */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

static struct device lp_device = {
        "lp",
        -1,
        LP_MAJOR,
        { 0, 0, 0, 0, 0, 0, 0, 0 },
        0,
        NULL,
        &lp_driver_fsop,
```

drivers/char/lp.c


```
 68: };
 69:
 70: struct lp lp_table[LP_MINORS] = {
 71:         { LP0_ADDR, LP0_ADDR + 1, LP0_ADDR + 2, 0 }
 72: };
 73:
 74: static void lp_delay(void)
 75: {
 76:         int n;
 77:
 78:         for(n = 0; n < 10000; n++) {
 79:                 NOP();
 80:         }
 81: }
 82:
 83: static int lp_ready(int minor)
 84: {
 85:         int n;
 86:
 87:         for(n = 0; n < LP_RDY_RETR; n++) {
 88:                 if(inport_b(lp_table[minor].stat) & LP_STAT_BUS) {
 89:                         break;
 90:                 }
 91:                 lp_delay();
 92:         }
 93:         if(n == LP_RDY_RETR) {
 94:                 return 0;
 95:         }
 96:         return 1;
 97: }
 98:
 99: static int lp_probe(int minor)
100: {
101:         /* first check */
102:         outport_b(lp_table[minor].data, 0x55);
103:         lp_delay();
104:         if(inport_b(lp_table[minor].data) != 0x55) {
105:                 return 1;               /* did not retain data */
106:         }
107:
108:         /* second check */
109:         outport_b(lp_table[minor].data, 0xAA);
110:         lp_delay();
111:         if(inport_b(lp_table[minor].data) != 0xAA) {
112:                 return 1;               /* did not retain data */
113:         }
114:         return 0;
115: }
116:
117: static int lp_write_data(int minor, unsigned char c)
118: {
119:         unsigned char ctrl;
120:
121:         if(!lp_ready(minor)) {
122:                 return -EBUSY;
123:         }
124:         outport_b(lp_table[minor].data, c);
125:         ctrl = inport_b(lp_table[minor].ctrl);
126:         outport_b(lp_table[minor].ctrl, ctrl | LP_CTRL_STR);
127:         lp_delay();
128:         outport_b(lp_table[minor].ctrl, ctrl);
129:         if(!lp_ready(minor)) {
130:                 return -EBUSY;
131:         }
132:         return 1;
133: }
134:
```

drivers/char/lp.c


```
  135: int lp_open(struct inode *i, struct fd *fd_table)
  136: {
  137:         int minor;
  138:
  139:         minor = MINOR(i->rdev);
  140:         if(!TEST_MINOR(lp_device.minors, minor)) {
  141:                 return -ENXIO;
  142:         }
  143:         if(!(lp_table[minor].flags & LP_CTRL_SEL)) {
  144:                 return -ENXIO;
  145:         }
  146:         if(lp_table[minor].flags & LP_STAT_BUS) {
  147:                 return -EBUSY;
  148:         }
  149:         lp_table[minor].flags |= LP_STAT_BUS;
  150:         return 0;
  151: }
  152:
  153: int lp_close(struct inode *i, struct fd *fd_table)
  154: {
  155:         int minor;
  156:
  157:         minor = MINOR(i->rdev);
  158:         if(!TEST_MINOR(lp_device.minors, minor)) {
  159:                 return -ENXIO;
  160:         }
  161:         lp_table[minor].flags &= ~LP_STAT_BUS;
  162:         return 0;
  163: }
  164:
  165: int lp_write(struct inode *i, struct fd *fd_table, const char *buffer, __size_t
count)
  166: {
  167:         unsigned int n;
  168:         int bytes_written, total_written;
  169:         int minor;
  170:
  171:         minor = MINOR(i->rdev);
  172:         if(!TEST_MINOR(lp_device.minors, minor)) {
  173:                 return -ENXIO;
  174:         }
  175:
  176:         total_written = 0;
  177:         for(n = 0; n < count; n++) {
  178:                 bytes_written = lp_write_data(minor, buffer[n]);
  179:                 if(bytes_written != 1) {
  180:                         break;
  181:                 }
  182:                 total_written += bytes_written;
  183:         }
  184:
  185:         return total_written;
  186: }
  187:
  188: void lp_init(void)
  189: {
  190:         int n;
  191:         unsigned char ctrl;
  192:
  193:         for(n = 0; n < LP_MINORS; n++) {
  194:                 if(!lp_probe(n)) {
  195:                         ctrl = inport_b(lp_table[n].ctrl);
  196:                         ctrl &= ~LP_CTRL_AUT;   /* disable auto LF */
  197:                         ctrl |= LP_CTRL_INI;    /* initialize */
  198:                         ctrl |= LP_CTRL_SEL;    /* select in */
  199:                         ctrl &= ~LP_CTRL_IRQ;   /* disable IRQ */
  200:                         ctrl &= ~LP_CTRL_BID;   /* disable bidirectional mode */
```

drivers/char/lp.c


```
  201:                          outport_b(lp_table[n].ctrl, ctrl);
  202:                          lp_table[n].flags |= LP_CTRL_SEL;
  203:                          printk("lp%d       0x%04X-0x%04X    -           \n", n,
lp_table[n].data, lp_table[n].data + 2);
  204:                          SET_MINOR(lp_device.minors, n);
  205:                  }
  206:          }
  207:
  208:          for(n = 0; n < LP_MINORS; n++) {
  209:                  if(lp_table[n].flags & LP_CTRL_SEL) {
  210:                          if(register_device(CHR_DEV, &lp_device)) {
  211:                                  printk("WARNING: %s(): unable to register lp dev
ice.\n", __FUNCTION__);
  212:                          }
  213:                          break;
  214:                  }
  215:          }
  216: }
```

drivers/char/Makefile


```
 1: # fiwix/drivers/char/Makefile
 2: #
 3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
 4: # Distributed under the terms of the Fiwix License.
 5: #
 6:
 7: .S.o:
 8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
 9: .c.o:
10:         $(CC) $(CFLAGS) -c -o $@ $<
11:
12: OBJS = console.o tty.o tty_queue.o vt.o defkeymap.o keyboard.o memdev.o lp.o
13:
14: char:   $(OBJS)
15:         $(LD) $(LDFLAGS) -r $(OBJS) -o char.o
16:
17: clean:
18:         rm -f *.o
19:
```

drivers/char/memdev.c


```
  1: /*
  2:  * fiwix/drivers/char/memdev.c
  3:  *
  4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
  5:  * Distributed under the terms of the Fiwix License.
  6:  */
  7:
  8: #include <fiwix/asm.h>
  9: #include <fiwix/kernel.h>
 10: #include <fiwix/memdev.h>
 11: #include <fiwix/devices.h>
 12: #include <fiwix/fs.h>
 13: #include <fiwix/errno.h>
 14: #include <fiwix/mm.h>
 15: #include <fiwix/mman.h>
 16: #include <fiwix/bios.h>
 17: #include <fiwix/stdio.h>
 18: #include <fiwix/string.h>
 19:
 20: static struct fs_operations mem_driver_fsop = {
 21:         0,
 22:         0,
 23:
 24:         mem_open,
 25:         mem_close,
 26:         mem_read,
 27:         mem_write,
 28:         NULL,                   /* ioctl */
 29:         mem_lseek,
 30:         NULL,                   /* readdir */
 31:         mem_mmap,
 32:         NULL,                   /* select */
 33:
 34:         NULL,                   /* readlink */
 35:         NULL,                   /* followlink */
 36:         NULL,                   /* bmap */
 37:         NULL,                   /* lockup */
 38:         NULL,                   /* rmdir */
 39:         NULL,                   /* link */
 40:         NULL,                   /* unlink */
 41:         NULL,                   /* symlink */
 42:         NULL,                   /* mkdir */
 43:         NULL,                   /* mknod */
 44:         NULL,                   /* truncate */
 45:         NULL,                   /* create */
 46:         NULL,                   /* rename */
 47:
 48:         NULL,                   /* read_block */
 49:         NULL,                   /* write_block */
 50:
 51:         NULL,                   /* read_inode */
 52:         NULL,                   /* write_inode */
 53:         NULL,                   /* ialloc */
 54:         NULL,                   /* ifree */
 55:         NULL,                   /* statfs */
 56:         NULL,                   /* read_superblock */
 57:         NULL,                   /* remount_fs */
 58:         NULL,                   /* write_superblock */
 59:         NULL                    /* release_superblock */
 60: };
 61:
 62: static struct fs_operations kmem_driver_fsop = {
 63:         0,
 64:         0,
 65:
 66:         kmem_open,
 67:         kmem_close,
```

drivers/char/memdev.c


```
   68:         kmem_read,
   69:         kmem_write,
   70:         NULL,                   /* ioctl */
   71:         kmem_lseek,
   72:         NULL,                   /* readdir */
   73:         mem_mmap,
   74:         NULL,                   /* select */
   75:
   76:         NULL,                   /* readlink */
   77:         NULL,                   /* followlink */
   78:         NULL,                   /* bmap */
   79:         NULL,                   /* lockup */
   80:         NULL,                   /* rmdir */
   81:         NULL,                   /* link */
   82:         NULL,                   /* unlink */
   83:         NULL,                   /* symlink */
   84:         NULL,                   /* mkdir */
   85:         NULL,                   /* mknod */
   86:         NULL,                   /* truncate */
   87:         NULL,                   /* create */
   88:         NULL,                   /* rename */
   89:
   90:         NULL,                   /* read_block */
   91:         NULL,                   /* write_block */
   92:
   93:         NULL,                   /* read_inode */
   94:         NULL,                   /* write_inode */
   95:         NULL,                   /* ialloc */
   96:         NULL,                   /* ifree */
   97:         NULL,                   /* statfs */
   98:         NULL,                   /* read_superblock */
   99:         NULL,                   /* remount_fs */
  100:         NULL,                   /* write_superblock */
  101:         NULL                    /* release_superblock */
  102: };
  103:
  104: static struct fs_operations null_driver_fsop = {
  105:         0,
  106:         0,
  107:
  108:         null_open,
  109:         null_close,
  110:         null_read,
  111:         null_write,
  112:         NULL,                   /* ioctl */
  113:         null_lseek,
  114:         NULL,                   /* readdir */
  115:         NULL,                   /* mmap */
  116:         NULL,                   /* select */
  117:
  118:         NULL,                   /* readlink */
  119:         NULL,                   /* followlink */
  120:         NULL,                   /* bmap */
  121:         NULL,                   /* lockup */
  122:         NULL,                   /* rmdir */
  123:         NULL,                   /* link */
  124:         NULL,                   /* unlink */
  125:         NULL,                   /* symlink */
  126:         NULL,                   /* mkdir */
  127:         NULL,                   /* mknod */
  128:         NULL,                   /* truncate */
  129:         NULL,                   /* create */
  130:         NULL,                   /* rename */
  131:
  132:         NULL,                   /* read_block */
  133:         NULL,                   /* write_block */
  134:
```

drivers/char/memdev.c


```
  135:         NULL,                   /* read_inode */
  136:         NULL,                   /* write_inode */
  137:         NULL,                   /* ialloc */
  138:         NULL,                   /* ifree */
  139:         NULL,                   /* statfs */
  140:         NULL,                   /* read_superblock */
  141:         NULL,                   /* remount_fs */
  142:         NULL,                   /* write_superblock */
  143:         NULL                    /* release_superblock */
  144: };
  145:
  146: static struct fs_operations zero_driver_fsop = {
  147:         0,
  148:         0,
  149:
  150:         zero_open,
  151:         zero_close,
  152:         zero_read,
  153:         zero_write,
  154:         NULL,                   /* ioctl */
  155:         zero_lseek,
  156:         NULL,                   /* readdir */
  157:         NULL,                   /* mmap */
  158:         NULL,                   /* select */
  159:
  160:         NULL,                   /* readlink */
  161:         NULL,                   /* followlink */
  162:         NULL,                   /* bmap */
  163:         NULL,                   /* lockup */
  164:         NULL,                   /* rmdir */
  165:         NULL,                   /* link */
  166:         NULL,                   /* unlink */
  167:         NULL,                   /* symlink */
  168:         NULL,                   /* mkdir */
  169:         NULL,                   /* mknod */
  170:         NULL,                   /* truncate */
  171:         NULL,                   /* create */
  172:         NULL,                   /* rename */
  173:
  174:         NULL,                   /* read_block */
  175:         NULL,                   /* write_block */
  176:
  177:         NULL,                   /* read_inode */
  178:         NULL,                   /* write_inode */
  179:         NULL,                   /* ialloc */
  180:         NULL,                   /* ifree */
  181:         NULL,                   /* statfs */
  182:         NULL,                   /* read_superblock */
  183:         NULL,                   /* remount_fs */
  184:         NULL,                   /* write_superblock */
  185:         NULL                    /* release_superblock */
  186: };
  187:
  188: static struct fs_operations memdev_driver_fsop = {
  189:         0,
  190:         0,
  191:
  192:         memdev_open,
  193:         NULL,                   /* close */
  194:         NULL,                   /* read */
  195:         NULL,                   /* write */
  196:         NULL,                   /* ioctl */
  197:         NULL,                   /* lseek */
  198:         NULL,                   /* readdir */
  199:         NULL,                   /* mmap */
  200:         NULL,                   /* select */
  201:
```

drivers/char/memdev.c


```
202:         NULL,                  /* readlink */
203:         NULL,                  /* followlink */
204:         NULL,                  /* bmap */
205:         NULL,                  /* lockup */
206:         NULL,                  /* rmdir */
207:         NULL,                  /* link */
208:         NULL,                  /* unlink */
209:         NULL,                  /* symlink */
210:         NULL,                  /* mkdir */
211:         NULL,                  /* mknod */
212:         NULL,                  /* truncate */
213:         NULL,                  /* create */
214:         NULL,                  /* rename */
215:
216:         NULL,                  /* read_block */
217:         NULL,                  /* write_block */
218:
219:         NULL,                  /* read_inode */
220:         NULL,                  /* write_inode */
221:         NULL,                  /* ialloc */
222:         NULL,                  /* ifree */
223:         NULL,                  /* statfs */
224:         NULL,                  /* read_superblock */
225:         NULL,                  /* remount_fs */
226:         NULL,                  /* write_superblock */
227:         NULL                   /* release_superblock */
228: };
229:
230: static struct fs_operations urandom_driver_fsop = {
231:         0,
232:         0,
233:
234:         urandom_open,
235:         urandom_close,
236:         urandom_read,
237:         urandom_write,
238:         NULL,                  /* ioctl */
239:         urandom_lseek,
240:         NULL,                  /* readdir */
241:         NULL,                  /* mmap */
242:         NULL,                  /* select */
243:
244:         NULL,                  /* readlink */
245:         NULL,                  /* followlink */
246:         NULL,                  /* bmap */
247:         NULL,                  /* lockup */
248:         NULL,                  /* rmdir */
249:         NULL,                  /* link */
250:         NULL,                  /* unlink */
251:         NULL,                  /* symlink */
252:         NULL,                  /* mkdir */
253:         NULL,                  /* mknod */
254:         NULL,                  /* truncate */
255:         NULL,                  /* create */
256:         NULL,                  /* rename */
257:
258:         NULL,                  /* read_block */
259:         NULL,                  /* write_block */
260:
261:         NULL,                  /* read_inode */
262:         NULL,                  /* write_inode */
263:         NULL,                  /* ialloc */
264:         NULL,                  /* ifree */
265:         NULL,                  /* statfs */
266:         NULL,                  /* read_superblock */
267:         NULL,                  /* remount_fs */
268:         NULL,                  /* write_superblock */
```

drivers/char/memdev.c


```
  269:         NULL                    /* release_superblock */
  270: };
  271:
  272: static struct device memdev_device = {
  273:         "mem",
  274:         -1,
  275:         MEMDEV_MAJOR,
  276:         { 0, 0, 0, 0, 0, 0, 0, 0 },
  277:         0,
  278:         NULL,
  279:         &memdev_driver_fsop,
  280: };
  281:
  282: int mem_open(struct inode *i, struct fd *fd_table)
  283: {
  284:         return 0;
  285: }
  286:
  287: int mem_close(struct inode *i, struct fd *fd_table)
  288: {
  289:         return 0;
  290: }
  291:
  292: int mem_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t count)
  293: {
  294:         unsigned int physical_memory;
  295:
  296:         physical_memory = (kstat.physical_pages << PAGE_SHIFT);
  297:         if(fd_table->offset >= physical_memory) {
  298:                 return 0;
  299:         }
  300:         count = MIN(count, physical_memory - fd_table->offset);
  301:         memcpy_b(buffer, (void *)P2V(fd_table->offset), count);
  302:         fd_table->offset += count;
  303:         return count;
  304: }
  305:
  306: int mem_write(struct inode *i, struct fd *fd_table, const char *buffer, __size_t
count)
  307: {
  308:         unsigned int physical_memory;
  309:
  310:         physical_memory = (kstat.physical_pages << PAGE_SHIFT);
  311:         if(fd_table->offset >= physical_memory) {
  312:                 return 0;
  313:         }
  314:         count = MIN(count, physical_memory - fd_table->offset);
  315:         memcpy_b((void *)P2V(fd_table->offset), buffer, count);
  316:         fd_table->offset += count;
  317:         return count;
  318: }
  319:
  320: int mem_lseek(struct inode *i, __off_t offset)
  321: {
  322:         return offset;
  323: }
  324:
  325: int kmem_open(struct inode *i, struct fd *fd_table)
  326: {
  327:         return 0;
  328: }
  329:
  330: int kmem_close(struct inode *i, struct fd *fd_table)
  331: {
  332:         return 0;
  333: }
  334:
```

drivers/char/memdev.c


```
  335: int kmem_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t count
)
  336: {
  337:         memcpy_b(buffer, (void *)P2V(fd_table->offset), count);
  338:         fd_table->offset += count;
  339:         return count;
  340: }
  341:
  342: int kmem_write(struct inode *i, struct fd *fd_table, const char *buffer, __size_
t count)
  343: {
  344:         memcpy_b((void *)P2V(fd_table->offset), buffer, count);
  345:         fd_table->offset += count;
  346:         return count;
  347: }
  348:
  349: int kmem_lseek(struct inode *i, __off_t offset)
  350: {
  351:         return offset;
  352: }
  353:
  354: int null_open(struct inode *i, struct fd *fd_table)
  355: {
  356:         return 0;
  357: }
  358:
  359: int null_close(struct inode *i, struct fd *fd_table)
  360: {
  361:         return 0;
  362: }
  363:
  364: int null_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t count
)
  365: {
  366:         return 0;
  367: }
  368:
  369: int null_write(struct inode *i, struct fd *fd_table, const char *buffer, __size_
t count)
  370: {
  371:         return count;
  372: }
  373:
  374: int null_lseek(struct inode *i, __off_t offset)
  375: {
  376:         return offset;
  377: }
  378:
  379: int zero_open(struct inode *i, struct fd *fd_table)
  380: {
  381:         return 0;
  382: }
  383:
  384: int zero_close(struct inode *i, struct fd *fd_table)
  385: {
  386:         return 0;
  387: }
  388:
  389: int zero_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t count
)
  390: {
  391:         memset_b(buffer, NULL, count);
  392:         return count;
  393: }
  394:
  395: int zero_write(struct inode *i, struct fd *fd_table, const char *buffer, __size_
t count)
```

drivers/char/memdev.c


```
  396: {
  397:         return count;
  398: }
  399:
  400: int zero_lseek(struct inode *i, __off_t offset)
  401: {
  402:         return offset;
  403: }
  404:
  405: int urandom_open(struct inode *i, struct fd *fd_table)
  406: {
  407:         return 0;
  408: }
  409:
  410: int urandom_close(struct inode *i, struct fd *fd_table)
  411: {
  412:         return 0;
  413: }
  414:
  415: int urandom_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t co
unt)
  416: {
  417:         int n;
  418:
  419:         for(n = 0; n < count; n++) {
  420:                 kstat.random_seed = kstat.random_seed * 1103515245 + 12345;
  421:                 *buffer = (char)(unsigned int)(kstat.random_seed / 65536) % 256;
  422:                 buffer++;
  423:         }
  424:         return count;
  425: }
  426:
  427: int urandom_write(struct inode *i, struct fd *fd_table, const char *buffer, __si
ze_t count)
  428: {
  429:         return count;
  430: }
  431:
  432: int urandom_lseek(struct inode *i, __off_t offset)
  433: {
  434:         return offset;
  435: }
  436:
  437: int memdev_open(struct inode *i, struct fd *fd_table)
  438: {
  439:         unsigned char minor;
  440:
  441:         minor = MINOR(i->rdev);
  442:         switch(minor) {
  443:                 case MEMDEV_MEM:
  444:                         i->fsop = &mem_driver_fsop;
  445:                         break;
  446:                 case MEMDEV_KMEM:
  447:                         i->fsop = &kmem_driver_fsop;
  448:                         break;
  449:                 case MEMDEV_NULL:
  450:                         i->fsop = &null_driver_fsop;
  451:                         break;
  452:                 case MEMDEV_ZERO:
  453:                         i->fsop = &zero_driver_fsop;
  454:                         break;
  455:                 case MEMDEV_RANDOM:
  456:                         i->fsop = &urandom_driver_fsop;
  457:                         break;
  458:                 case MEMDEV_URANDOM:
  459:                         i->fsop = &urandom_driver_fsop;
  460:                         break;
```

drivers/char/memdev.c


```
  461:                 default:
  462:                         return -ENXIO;
  463:         }
  464:         if(i->fsop->open) {
  465:                 return i->fsop->open(i, fd_table);
  466:         }
  467:         return 0;
  468: }
  469:
  470: /*
  471:  * This function maps a range of physical addresses marked as not available for
  472:  * use in the BIOS memory map, like the video RAM.
  473:  */
  474: int mem_mmap(struct inode *i, struct vma *vma)
  475: {
  476:         unsigned int addr, length;
  477:
  478:         length = (vma->end - vma->start) & PAGE_MASK;
  479:
  480:         /* this breaks down the range in 4KB chunks */
  481:         for(addr = 0; addr < length; addr += PAGE_SIZE) {
  482:                 /* map the page only if is NOT available in the BIOS map */
  483:                 if(!addr_in_bios_map(vma->offset + addr)) {
  484:                         if(!map_page(current, (vma->start + addr) & PAGE_MASK, (
vma->offset + addr) & PAGE_MASK, PROT_READ | PROT_WRITE)) {
  485:                                 return -ENOMEM;
  486:                         }
  487:                 } else {
  488:                         printk("ERROR: %s(): mapping AVAILABLE pages in BIOS mem
ory map isn't supported.\n", __FUNCTION__);
  489:                         printk("\tinvalid mapping: 0x%08x -> 0x%08x\n", (vma->st
art + addr) & PAGE_MASK, (vma->offset + addr) & PAGE_MASK);
  490:                         return -EAGAIN;
  491:                 }
  492:         }
  493:         invalidate_tlb();
  494:         return 0;
  495: }
  496:
  497: void memdev_init(void)
  498: {
  499:         if(register_device(CHR_DEV, &memdev_device)) {
  500:                 printk("ERROR: %s(): unable to register memory devices.\n", __FU
NCTION__);
  501:                 return;
  502:         }
  503:         SET_MINOR(memdev_device.minors, MEMDEV_MEM);
  504:         SET_MINOR(memdev_device.minors, MEMDEV_KMEM);
  505:         SET_MINOR(memdev_device.minors, MEMDEV_NULL);
  506:         SET_MINOR(memdev_device.minors, MEMDEV_ZERO);
  507:         SET_MINOR(memdev_device.minors, MEMDEV_RANDOM);
  508:         SET_MINOR(memdev_device.minors, MEMDEV_URANDOM);
  509: }
```

drivers/char/tty.c


```
    1: /*
    2:  * fiwix/drivers/char/tty.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/ioctl.h>
   10: #include <fiwix/tty.h>
   11: #include <fiwix/ctype.h>
   12: #include <fiwix/console.h>
   13: #include <fiwix/pic.h>
   14: #include <fiwix/devices.h>
   15: #include <fiwix/fs.h>
   16: #include <fiwix/errno.h>
   17: #include <fiwix/sched.h>
   18: #include <fiwix/timer.h>
   19: #include <fiwix/sleep.h>
   20: #include <fiwix/process.h>
   21: #include <fiwix/fcntl.h>
   22: #include <fiwix/stdio.h>
   23: #include <fiwix/string.h>
   24:
   25: struct tty tty_table[NR_TTYS];
   26: extern short int current_cons;
   27:
   28: static void wait_vtime_off(unsigned int arg)
   29: {
   30:         unsigned int *fn = (unsigned int *)arg;
   31:
   32:         wakeup(fn);
   33: }
   34:
   35: static void get_termio(struct tty *tty, struct termio *termio)
   36: {
   37:         int n;
   38:
   39:         termio->c_iflag = tty->termios.c_iflag;
   40:         termio->c_oflag = tty->termios.c_oflag;
   41:         termio->c_cflag = tty->termios.c_cflag;
   42:         termio->c_lflag = tty->termios.c_lflag;
   43:         termio->c_line = tty->termios.c_line;
   44:         for(n = 0; n < NCC; n++) {
   45:                 termio->c_cc[n] = tty->termios.c_cc[n];
   46:         }
   47: }
   48:
   49: static void set_termio(struct tty *tty, struct termio *termio)
   50: {
   51:         int n;
   52:
   53:         tty->termios.c_iflag = termio->c_iflag;
   54:         tty->termios.c_oflag = termio->c_oflag;
   55:         tty->termios.c_cflag = termio->c_cflag;
   56:         tty->termios.c_lflag = termio->c_lflag;
   57:         tty->termios.c_line = termio->c_line;
   58:         for(n = 0; n < NCC; n++) {
   59:                 tty->termios.c_cc[n] = termio->c_cc[n];
   60:         }
   61: }
   62:
   63: static void out_char(struct tty *tty, unsigned char ch)
   64: {
   65:         if(ISCNTRL(ch) && !ISSPACE(ch) && (tty->termios.c_lflag & ECHOCTL)) {
   66:                 if(tty->lnext || (!tty->lnext && ch != tty->termios.c_cc[VEOF]))
 {
```

drivers/char/tty.c


```
   67:                         tty_queue_putchar(tty, &tty->write_q, '^');
   68:                         tty_queue_putchar(tty, &tty->write_q, ch + 64);
   69:                 }
   70:         } else {
   71:                 tty_queue_putchar(tty, &tty->write_q, ch);
   72:         }
   73: }
   74:
   75: static void erase_char(struct tty *tty, unsigned char erasechar)
   76: {
   77:         unsigned char ch;
   78:
   79:         if(erasechar == tty->termios.c_cc[VERASE]) {
   80:                 if((ch = tty_queue_unputchar(&tty->cooked_q))) {
   81:                         if(tty->termios.c_lflag & ECHO) {
   82:                                 tty_queue_putchar(tty, &tty->write_q, '\b');
   83:                                 tty_queue_putchar(tty, &tty->write_q, ' ');
   84:                                 tty_queue_putchar(tty, &tty->write_q, '\b');
   85:                                 if(ch == '\t') {
   86:                                         tty->deltab(tty);
   87:                                 }
   88:                                 if(ISCNTRL(ch) && !ISSPACE(ch) && tty->termios.c
_lflag & ECHOCTL) {
   89:                                         tty_queue_putchar(tty, &tty->write_q, '\
b');
   90:                                         tty_queue_putchar(tty, &tty->write_q, '
');
   91:                                         tty_queue_putchar(tty, &tty->write_q, '\
b');
   92:                                 }
   93:                         }
   94:                 }
   95:         }
   96:         if(erasechar == tty->termios.c_cc[VWERASE]) {
   97:                 unsigned char word_seen = 0;
   98:
   99:                 while(tty->cooked_q.count > 0) {
  100:                         ch = LAST_CHAR(&tty->cooked_q);
  101:                         if((ch == ' ' || ch == '\t') && word_seen) {
  102:                                 break;
  103:                         }
  104:                         if(ch != ' ' && ch != '\t') {
  105:                                 word_seen = 1;
  106:                         }
  107:                         erase_char(tty, tty->termios.c_cc[VERASE]);
  108:                 }
  109:         }
  110:         if(erasechar == tty->termios.c_cc[VKILL]) {
  111:                 while(tty->cooked_q.count > 0) {
  112:                         erase_char(tty, tty->termios.c_cc[VERASE]);
  113:                 }
  114:                 if(tty->termios.c_lflag & ECHOK && !(tty->termios.c_lflag & ECHO
E)) {
  115:                         tty_queue_putchar(tty, &tty->write_q, '\n');
  116:                 }
  117:         }
  118: }
  119:
  120: int register_tty(__dev_t dev)
  121: {
  122:         int n;
  123:
  124:         for(n = 0; n < NR_TTYS; n++) {
  125:                 if(tty_table[n].dev == dev) {
  126:                         printk("ERROR: %s(): tty device %d,%d already registered
!\n", __FUNCTION__, MAJOR(dev), MINOR(dev));
  127:                         return 1;
```

drivers/char/tty.c


```
128:                 }
129:                 if(!tty_table[n].dev) {
130:                         tty_table[n].dev = dev;
131:                         tty_table[n].count = 0;
132:                         return 0;
133:                 }
134:         }
135:         printk("ERROR: %s(): tty table is full!\n", __FUNCTION__);
136:         return 1;
137: }
138:
139: struct tty * get_tty(__dev_t dev)
140: {
141:         int n;
142:
143:         if(!dev) {
144:                 return NULL;
145:         }
146:
147:         /* /dev/console = system console */
148:         if(dev == MKDEV(SYSCON_MAJOR, 1)) {
149:                 dev = (__dev_t)_syscondev;
150:         }
151:
152:         /* /dev/tty0 = current virtual console */
153:         if(dev == MKDEV(VCONSOLES_MAJOR, 0)) {
154:                 dev = MKDEV(VCONSOLES_MAJOR, current_cons);
155:         }
156:
157:         /* /dev/tty = controlling TTY device */
158:         if(dev == MKDEV(SYSCON_MAJOR, 0)) {
159:                 if(!current->ctty) {
160:                         return NULL;
161:                 }
162:                 dev = current->ctty->dev;
163:         }
164:
165:         for(n = 0; n < NR_TTYS; n++) {
166:                 if(tty_table[n].dev != dev) {
167:                         continue;
168:                 }
169:                 return &tty_table[n];
170:         }
171:         return NULL;
172: }
173:
174: void disassociate_ctty(struct tty *tty)
175: {
176:         struct proc *p;
177:
178:         if(!tty) {
179:                 return;
180:         }
181:
182:         /* this tty is no longer the controlling tty of any session */
183:         tty->pgid = tty->sid = 0;
184:
185:         /* clear the controlling tty for all processes in the same SID */
186:         FOR_EACH_PROCESS(p) {
187:                 if((p->state != PROC_UNUSED) && (p->sid == current->sid)) {
188:                         p->ctty = NULL;
189:                 }
190:         }
191:         kill_pgrp(current->pgid, SIGHUP);
192:         kill_pgrp(current->pgid, SIGCONT);
193: }
194:
```

drivers/char/tty.c


```
  195: void termios_reset(struct tty *tty)
  196: {
  197:         tty->termios.c_iflag = ICRNL | IXON | IXOFF;
  198:         tty->termios.c_oflag = OPOST | ONLCR;
  199:         tty->termios.c_cflag = B38400 | CS8 | HUPCL | CREAD | CLOCAL;
  200:         tty->termios.c_lflag = ISIG | ICANON | ECHO | ECHOE | ECHOK | ECHOCTL |
ECHOKE | IEXTEN;
  201:         tty->termios.c_line = 0;
  202:         tty->termios.c_cc[VINTR]    = 3;        /* ^C */
  203:         tty->termios.c_cc[VQUIT]    = 28;       /* ^\ */
  204:         tty->termios.c_cc[VERASE]   = BS;       /* ^? (127) not '\b' (^H) */
  205:         tty->termios.c_cc[VKILL]    = 21;       /* ^U */
  206:         tty->termios.c_cc[VEOF]     = 4;        /* ^D */
  207:         tty->termios.c_cc[VTIME]    = 0;
  208:         tty->termios.c_cc[VMIN]     = 1;
  209:         tty->termios.c_cc[VSWTC]    = 0;
  210:         tty->termios.c_cc[VSTART]   = 17;       /* ^Q */
  211:         tty->termios.c_cc[VSTOP]    = 19;       /* ^S */
  212:         tty->termios.c_cc[VSUSP]    = 26;       /* ^Z */
  213:         tty->termios.c_cc[VEOL]     = '\n';     /* ^J */
  214:         tty->termios.c_cc[VREPRINT] = 18;       /* ^R */
  215:         tty->termios.c_cc[VDISCARD] = 15;       /* ^O */
  216:         tty->termios.c_cc[VWERASE]  = 23;       /* ^W */
  217:         tty->termios.c_cc[VLNEXT]   = 22;       /* ^V */
  218:         tty->termios.c_cc[VEOL2]    = 0;
  219: }
  220:
  221: void do_cook(struct tty *tty)
  222: {
  223:         int n;
  224:         unsigned char ch;
  225:         struct cblock *cb;
  226:
  227:         while(tty->read_q.count > 0) {
  228:                 ch = tty_queue_getchar(&tty->read_q);
  229:
  230:                 if((tty->termios.c_lflag & ISIG) && !tty->lnext) {
  231:                         if(ch == tty->termios.c_cc[VINTR]) {
  232:                                 if(!(tty->termios.c_lflag & NOFLSH)) {
  233:                                         tty_queue_flush(&tty->read_q);
  234:                                         tty_queue_flush(&tty->cooked_q);
  235:                                 }
  236:                                 if(tty->pgid > 0) {
  237:                                         kill_pgrp(tty->pgid, SIGINT);
  238:                                 }
  239:                                 break;
  240:                         }
  241:                         if(ch == tty->termios.c_cc[VQUIT]) {
  242:                                 if(tty->pgid > 0) {
  243:                                         kill_pgrp(tty->pgid, SIGQUIT);
  244:                                 }
  245:                                 break;
  246:                         }
  247:                         if(ch == tty->termios.c_cc[VSUSP]) {
  248:                                 if(tty->pgid > 0) {
  249:                                         kill_pgrp(tty->pgid, SIGTSTP);
  250:                                 }
  251:                                 break;
  252:                         }
  253:                 }
  254:
  255:                 if(tty->termios.c_iflag & ISTRIP) {
  256:                         ch = TOASCII(ch);
  257:                 }
  258:                 if(tty->termios.c_iflag & IUCLC) {
  259:                         if(ISUPPER(ch)) {
  260:                                 ch = TOLOWER(ch);
```

drivers/char/tty.c


```
  261:                          }
  262:                  }
  263:
  264:                  if(!tty->lnext) {
  265:                          if(ch == '\r') {
  266:                                  if(tty->termios.c_iflag & IGNCR) {
  267:                                          continue;
  268:                                  }
  269:                                  if(tty->termios.c_iflag & ICRNL) {
  270:                                          ch = '\n';
  271:                                  }
  272:                          } else {
  273:                                  if(ch == '\n') {
  274:                                          if(tty->termios.c_iflag & INLCR) {
  275:                                                  ch = '\r';
  276:                                          }
  277:                                  }
  278:                          }
  279:                  }
  280:
  281:                  if(tty->termios.c_lflag & ICANON && !tty->lnext) {
  282:                          if(ch == tty->termios.c_cc[VERASE] || ch == tty->termios
.c_cc[VWERASE] || ch == tty->termios.c_cc[VKILL]) {
  283:                                  erase_char(tty, ch);
  284:                                  continue;
  285:                          }
  286:
  287:                          if(ch == tty->termios.c_cc[VREPRINT]) {
  288:                                  out_char(tty, ch);
  289:                                  tty_queue_putchar(tty, &tty->write_q, '\n');
  290:                                  cb = tty->cooked_q.head;
  291:                                  while(cb) {
  292:                                          for(n = 0; n < cb->end_off; n++) {
  293:                                                  if(n >= cb->start_off) {
  294:                                                          out_char(tty, cb->data[n
]);
  295:                                                  }
  296:                                          }
  297:                                          cb = cb->next;
  298:                                  }
  299:                                  continue;
  300:                          }
  301:
  302:                          if(ch == tty->termios.c_cc[VLNEXT] && tty->termios.c_lfl
ag & IEXTEN) {
  303:                                  tty->lnext = 1;
  304:                                  if(tty->termios.c_lflag & ECHOCTL) {
  305:                                          tty_queue_putchar(tty, &tty->write_q, '^
');
  306:                                          tty_queue_putchar(tty, &tty->write_q, '\
b');
  307:                                  }
  308:                                  break;
  309:                          }
  310:
  311:                          if(tty->termios.c_iflag & IXON) {
  312:                                  if(ch == tty->termios.c_cc[VSTART]) {
  313:                                          tty->start(tty);
  314:                                          continue;
  315:                                  }
  316:                                  if(ch == tty->termios.c_cc[VSTOP]) {
  317:                                          tty->stop(tty);
  318:                                          continue;
  319:                                  }
  320:                                  if(tty->termios.c_iflag & IXANY) {
  321:                                          tty->start(tty);
  322:                                  }
```

drivers/char/tty.c


```
  323:                          }
  324:                  }
  325:
  326:                  /* using ISSPACE here makes LNEXT working incorrectly, FIXME */
  327:                  if(tty->termios.c_lflag & ICANON) {
  328:                          if(ISCNTRL(ch) && !ISSPACE(ch) && (tty->termios.c_lflag
& ECHOCTL)) {
  329:                                  out_char(tty, ch);
  330:                                  tty_queue_putchar(tty, &tty->cooked_q, ch);
  331:                                  tty->lnext = 0;
  332:                                  continue;
  333:                          }
  334:                          if(ch == '\n') {
  335:                                  tty->canon_data = 1;
  336:                          }
  337:                  }
  338:
  339:                  if(tty->termios.c_lflag & ECHO) {
  340:                          out_char(tty, ch);
  341:                  } else {
  342:                          if((tty->termios.c_lflag & ECHONL) && (ch == '\n')) {
  343:                                  out_char(tty, ch);
  344:                          }
  345:                  }
  346:                  tty_queue_putchar(tty, &tty->cooked_q, ch);
  347:                  tty->lnext = 0;
  348:          }
  349:          tty->output(tty);
  350:          wakeup(&tty_read);
  351:          if(!(tty->termios.c_lflag & ICANON) || ((tty->termios.c_lflag & ICANON)
&& tty->canon_data)) {
  352:                  wakeup(&do_select);
  353:          }
  354: }
  355:
  356: int tty_open(struct inode *i, struct fd *fd_table)
  357: {
  358:          int noctty_flag;
  359:          __dev_t dev;
  360:          struct tty *tty;
  361:
  362:          noctty_flag = fd_table->flags & O_NOCTTY;
  363:
  364:          dev = i->rdev;
  365:
  366:          if(MAJOR(i->rdev) == SYSCON_MAJOR && MINOR(i->rdev) == 0) {
  367:                  if(!current->ctty) {
  368:                          return -ENXIO;
  369:                  }
  370:                  dev = i->rdev;
  371:          }
  372:
  373:          if(MAJOR(dev) == VCONSOLES_MAJOR && MINOR(dev) == 0) {
  374:                  noctty_flag = 1;
  375:          }
  376:
  377:          if(!(tty = get_tty(dev))) {
  378:                  printk("%s(): oops! (%x)\n", __FUNCTION__, dev);
  379:                  printk("_syscondev = %x\n", _syscondev);
  380:                  return -ENXIO;
  381:          }
  382:          tty->count++;
  383:
  384:          if(SESS_LEADER(current) && !current->ctty && !noctty_flag && !tty->sid)
{
  385:                  current->ctty = tty;
  386:                  tty->sid = current->sid;
```

drivers/char/tty.c


```
  387:                 tty->pgid = current->pgid;
  388:         }
  389:         return 0;
  390: }
  391:
  392: int tty_close(struct inode *i, struct fd *fd_table)
  393: {
  394:         struct proc *p;
  395:         struct tty *tty;
  396:
  397:         if(!(tty = get_tty(i->rdev))) {
  398:                 printk("%s(): oops! (%x)\n", __FUNCTION__, i->rdev);
  399:                 return -ENXIO;
  400:         }
  401:
  402:         tty->count = tty->count ? tty->count - 1 : 0;
  403:         if(!tty->count) {
  404:                 tty->reset(tty);
  405:                 termios_reset(tty);
  406:                 tty->pgid = tty->sid = 0;
  407:
  408:                 /* this tty is no longer the controlling tty of any process */
  409:                 FOR_EACH_PROCESS(p) {
  410:                         if((p->state != PROC_UNUSED) && (p->ctty == tty)) {
  411:                                 p->ctty = NULL;
  412:                         }
  413:                 }
  414:         }
  415:         return 0;
  416: }
  417:
  418: int tty_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t count)
  419: {
  420:         unsigned int n, min;
  421:         unsigned char ch;
  422:         struct tty *tty;
  423:         struct callout_req creq;
  424:
  425:         if(!(tty = get_tty(i->rdev))) {
  426:                 printk("%s(): oops! (%x)\n", __FUNCTION__, i->rdev);
  427:                 return -ENXIO;
  428:         }
  429:
  430:         /* only the foreground process group is allowed to read from the tty */
  431:         if(i->rdev != MKDEV(VCONSOLES_MAJOR, 0)) {      /* /dev/tty0 */
  432:                 if(current->pgid != tty->pgid) {
  433:                         if(current->sigaction[SIGTTIN - 1].sa_handler == SIG_IGN
 || current->sigblocked & (1 << (SIGTTIN - 1)) || is_orphaned_pgrp(current->pgid)) {
  434:                                 return -EIO;
  435:                         }
  436:                         kill_pgrp(current->pgid, SIGTTIN);
  437:                         return -ERESTART;
  438:                 }
  439:         }
  440:
  441:         n = min = 0;
  442:         while(count > 0) {
  443:                 if(tty->termios.c_lflag & ICANON) {
  444:                         if((ch = LAST_CHAR(&tty->cooked_q))) {
  445:                                 if(ch == '\n' || ch == tty->termios.c_cc[VEOL] |
| ch == tty->termios.c_cc[VEOF] || (tty->termios.c_lflag & IEXTEN && ch == tty->termios
.c_cc[VEOL2] && tty->termios.c_cc[VEOL2] != 0)) {
  446:
  447:                                         tty->canon_data = 0;
  448:                                         /* EOF is not passed to the reading proc
ess */
  449:                                         if(ch == tty->termios.c_cc[VEOF]) {
```

drivers/char/tty.c


```
  450:                                                  tty_queue_unputchar(&tty->cooked
_q);
  451:                                          }
  452:
  453:                                          while(n < count) {
  454:                                                  if((ch = tty_queue_getchar(&tty-
>cooked_q))) {
  455:                                                          buffer[n++] = ch;
  456:                                                  } else {
  457:                                                          break;
  458:                                                  }
  459:                                          }
  460:                                          break;
  461:                                  }
  462:                          }
  463:                  } else {
  464:                          if(tty->termios.c_cc[VTIME] > 0) {
  465:                                  unsigned int ini_ticks = kstat.ticks;
  466:                                  unsigned int timeout;
  467:
  468:                                  if(!tty->termios.c_cc[VMIN]) {
  469:                                          /* VTIME is measured in tenths of second
 */
  470:                                          timeout = tty->termios.c_cc[VTIME] * (HZ
 / 10);
  471:
  472:                                          while(kstat.ticks - ini_ticks < timeout
&& !tty->cooked_q.count) {
  473:                                                  creq.fn = wait_vtime_off;
  474:                                                  creq.arg = (unsigned int)&tty->c
ooked_q;
  475:                                                  add_callout(&creq, timeout);
  476:                                                  if(fd_table->flags & O_NONBLOCK)
 {
  477:                                                          return -EAGAIN;
  478:                                                  }
  479:                                                  if(sleep(&tty_read, PROC_INTERRU
PTIBLE)) {
  480:                                                          return -EINTR;
  481:                                                  }
  482:                                          }
  483:                                          while(n < count) {
  484:                                                  if((ch = tty_queue_getchar(&tty-
>cooked_q))) {
  485:                                                          buffer[n++] = ch;
  486:                                                  } else {
  487:                                                          break;
  488:                                                  }
  489:                                          }
  490:                                          break;
  491:                                  } else {
  492:                                          if(tty->cooked_q.count > 0) {
  493:                                                  if(n < MIN(tty->termios.c_cc[VMI
N], count)) {
  494:                                                          ch = tty_queue_getchar(&
tty->cooked_q);
  495:                                                          buffer[n++] = ch;
  496:                                                  }
  497:                                                  if(n >= MIN(tty->termios.c_cc[VM
IN], count)) {
  498:                                                          del_callout(&creq);
  499:                                                          break;
  500:                                                  }
  501:                                                  timeout = tty->termios.c_cc[VTIM
E] * (HZ / 10);
  502:                                                  creq.fn = wait_vtime_off;
  503:                                                  creq.arg = (unsigned int)&tty->c
```

drivers/char/tty.c


```
ooked_q;
  504:                                                  add_callout(&creq, timeout);
  505:                                                  if(fd_table->flags & O_NONBLOCK)
 {
  506:                                                          return -EAGAIN;
  507:                                                  }
  508:                                                  if(sleep(&tty_read, PROC_INTERRU
PTIBLE)) {
  509:                                                          return -EINTR;
  510:                                                  }
  511:                                                  if(!tty->cooked_q.count) {
  512:                                                          break;
  513:                                                  }
  514:                                                  continue;
  515:                                          }
  516:                                  }
  517:                          } else {
  518:                                  if(tty->cooked_q.count > 0) {
  519:                                          if(min < tty->termios.c_cc[VMIN] || !tty
->termios.c_cc[VMIN]) {
  520:                                                  if(n < count) {
  521:                                                          ch = tty_queue_getchar(&
tty->cooked_q);
  522:                                                          buffer[n++] = ch;
  523:                                                  }
  524:                                                  min++;
  525:                                          }
  526:                                  }
  527:                                  if(min >= tty->termios.c_cc[VMIN]) {
  528:                                          break;
  529:                                  }
  530:                          }
  531:                  }
  532:                  if(fd_table->flags & O_NONBLOCK) {
  533:                          return -EAGAIN;
  534:                  }
  535:                  if(sleep(&tty_read, PROC_INTERRUPTIBLE)) {
  536:                          return -EINTR;
  537:                  }
  538:          }
  539:
  540:          if(n > 0) {
  541:                  i->i_atime = CURRENT_TIME;
  542:          }
  543:          return n;
  544: }
  545:
  546: int tty_write(struct inode *i, struct fd *fd_table, const char *buffer, __size_t
 count)
  547: {
  548:          unsigned int n;
  549:          unsigned char ch;
  550:          struct tty *tty;
  551:
  552:          if(!(tty = get_tty(i->rdev))) {
  553:                  printk("%s(): oops! (%x)\n", __FUNCTION__, i->rdev);
  554:                  return -ENXIO;
  555:          }
  556:
  557:          /* only the foreground process group is allowed to write to the tty */
  558:          if(i->rdev != MKDEV(VCONSOLES_MAJOR, 0)) {      /* /dev/tty0 */
  559:                  if(current->pgid != tty->pgid && tty->termios.c_lflag & TOSTOP)
{
  560:                          if(current->sigaction[SIGTTIN - 1].sa_handler != SIG_IGN
 && !(current->sigblocked & (1 << (SIGTTIN - 1)))) {
  561:                                  if(is_orphaned_pgrp(current->pgid)) {
  562:                                          return -EIO;
```

drivers/char/tty.c


```
563:                                    }
564:                                    kill_pgrp(current->pgid, SIGTTOU);
565:                                    return -ERESTART;
566:                            }
567:                    }
568:            }
569:
570:            n = 0;
571:            for(;;) {
572:                    if(current->sigpending & ~current->sigblocked) {
573:                            return -ERESTART;
574:                    }
575:                    while(count && n < count) {
576:                            ch = *(buffer + n);
577:                            /* FIXME: check if *(buffer + n) address is valid */
578:                            if(tty_queue_putchar(tty, &tty->write_q, ch) < 0) {
579:                                    break;
580:                            }
581:                            n++;
582:                    }
583:                    tty->output(tty);
584:                    if(n == count) {
585:                            break;
586:                    }
587:                    if(tty->write_q.count > 0) {
588:                            if(sleep(&tty_write, PROC_INTERRUPTIBLE)) {
589:                                    return -EINTR;
590:                            }
591:                    }
592:                    do_sched();
593:            }
594:
595:            if(n > 0) {
596:                    i->i_mtime = CURRENT_TIME;
597:            }
598:            return n;
599: }
600:
601: /* FIXME: http://www.lafn.org/~dave/linux/termios.txt (doc/termios.txt) */
602: int tty_ioctl(struct inode *i, int cmd, unsigned long int arg)
603: {
604:            struct proc *p;
605:            struct tty *tty;
606:            int errno;
607:
608:            if(!(tty = get_tty(i->rdev))) {
609:                    printk("%s(): oops! (%x)\n", __FUNCTION__, i->rdev);
610:                    return -ENXIO;
611:            }
612:
613:            switch(cmd) {
614:                    /*
615:                     * Fetch and store the current terminal parameters to a termios
616:                     * structure pointed to by the argument.
617:                     */
618:                    case TCGETS:
619:                            if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(struct termios)))) {
620:                                    return errno;
621:                            }
622:                            memcpy_b((void *)arg, &tty->termios, sizeof(struct termi
os));
623:                            break;
624:
625:                    /*
626:                     * Set the current terminal parameters according to the
627:                     * values in the termios structure pointed to by the argument.
```

drivers/char/tty.c


```
  628:                  */
  629:                 case TCSETS:
  630:                         if((errno = check_user_area(VERIFY_READ, (void *)arg, si
zeof(struct termios)))) {
  631:                                 return errno;
  632:                         }
  633:                         memcpy_b(&tty->termios, (void *)arg, sizeof(struct termi
os));
  634:                         break;
  635:
  636:                 /*
  637:                  * Same as TCSETS except it doesn't take effect until all
  638:                  * the characters queued for output have been transmitted.
  639:                  */
  640:                 case TCSETSW:
  641:                         if((errno = check_user_area(VERIFY_READ, (void *)arg, si
zeof(struct termios)))) {
  642:                                 return errno;
  643:                         }
  644:                         memcpy_b(&tty->termios, (void *)arg, sizeof(struct termi
os));
  645:                         break;
  646:
  647:                 /*
  648:                  * Same as TCSETSW except that all characters queued for
  649:                  * input are discarded.
  650:                  */
  651:                 case TCSETSF:
  652:                         if((errno = check_user_area(VERIFY_READ, (void *)arg, si
zeof(struct termios)))) {
  653:                                 return errno;
  654:                         }
  655:                         memcpy_b(&tty->termios, (void *)arg, sizeof(struct termi
os));
  656:                         tty_queue_flush(&tty->read_q);
  657:                         break;
  658:
  659:                 /*
  660:                  * Fetches and stores the current terminal parameters to a
  661:                  * termio structure pointed to by the argument.
  662:                  */
  663:                 case TCGETA:
  664:                         if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(struct termio)))) {
  665:                                 return errno;
  666:                         }
  667:                         get_termio(tty, (struct termio *)arg);
  668:                         break;
  669:
  670:                 /*
  671:                  * Set the current terminal parameters according to the
  672:                  * values in the termio structure pointed to by the argument.
  673:                  */
  674:                 case TCSETA:
  675:                         if((errno = check_user_area(VERIFY_READ, (void *)arg, si
zeof(struct termio)))) {
  676:                                 return errno;
  677:                         }
  678:                         set_termio(tty, (struct termio *)arg);
  679:                         break;
  680:
  681:                 /*
  682:                  * Same a TCSET except it doesn't take effect until all
  683:                  * the characters queued for output have been transmitted.
  684:                  */
  685:                 case TCSETAW:
  686:                         if((errno = check_user_area(VERIFY_READ, (void *)arg, si
```

drivers/char/tty.c


```
zeof(struct termio)))) {
  687:                                  return errno;
  688:                          }
  689:                          set_termio(tty, (struct termio *)arg);
  690:                          break;
  691:
  692:                  /*
  693:                   * Same as TCSETAW except that all characters queued for
  694:                   * input are discarded.
  695:                   */
  696:                  case TCSETAF:
  697:                          if((errno = check_user_area(VERIFY_READ, (void *)arg, si
zeof(struct termio)))) {
  698:                                  return errno;
  699:                          }
  700:                          set_termio(tty, (struct termio *)arg);
  701:                          break;
  702:
  703:                  case TCXONC:
  704:                          switch(arg) {
  705:                                  case TCOOFF:
  706:                                          tty->stop(tty);
  707:                                          break;
  708:                                  case TCOON:
  709:                                          tty->start(tty);
  710:                                          break;
  711:                                  default:
  712:                                          return -EINVAL;
  713:                          }
  714:                          break;
  715:                  case TCFLSH:
  716:                          switch(arg) {
  717:                                  case TCIFLUSH:
  718:                                          tty_queue_flush(&tty->read_q);
  719:                                          tty_queue_flush(&tty->cooked_q);
  720:                                          break;
  721:                                  case TCOFLUSH:
  722:                                          tty_queue_flush(&tty->write_q);
  723:                                          break;
  724:                                  case TCIOFLUSH:
  725:                                          tty_queue_flush(&tty->read_q);
  726:                                          tty_queue_flush(&tty->cooked_q);
  727:                                          tty_queue_flush(&tty->write_q);
  728:                                          break;
  729:                                  default:
  730:                                          return -EINVAL;
  731:                          }
  732:                          break;
  733:                  case TIOCSCTTY:
  734:                          if(SESS_LEADER(current) && (current->sid == tty->sid)) {
  735:                                  return 0;
  736:                          }
  737:                          if(!SESS_LEADER(current) || current->ctty) {
  738:                                  return -EPERM;
  739:                          }
  740:                          if(tty->sid) {
  741:                                  if((arg == 1) && IS_SUPERUSER) {
  742:                                          FOR_EACH_PROCESS(p) {
  743:                                                  if((p->state != PROC_UNUSED) &&
(p->ctty == tty)) {
  744:                                                          p->ctty = NULL;
  745:                                                  }
  746:                                          }
  747:                                  } else {
  748:                                          return -EPERM;
  749:                                  }
  750:                          }
```

drivers/char/tty.c


```
  751:                          current->ctty = tty;
  752:                          tty->sid = current->sid;
  753:                          tty->pgid = current->pgid;
  754:                          break;
  755:                  case TIOCGPGRP:
  756:                          if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(__pid_t)))) {
  757:                                  return errno;
  758:                          }
  759:                          memcpy_b((void *)arg, &tty->pgid, sizeof(__pid_t));
  760:                          break;
  761:                  case TIOCSPGRP:
  762:                          if(arg < 1) {
  763:                                  return -EINVAL;
  764:                          }
  765:                          if((errno = check_user_area(VERIFY_READ, (void *)arg, si
zeof(__pid_t)))) {
  766:                                  return errno;
  767:                          }
  768:                          memcpy_b(&tty->pgid, (void *)arg, sizeof(__pid_t));
  769:                          break;
  770:                  case TIOCGWINSZ:
  771:                          if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(struct winsize)))) {
  772:                                  return errno;
  773:                          }
  774:                          memcpy_b((void *)arg, &tty->winsize, sizeof(struct winsi
ze));
  775:                          break;
  776:                  case TIOCSWINSZ:
  777:                  {
  778:                          struct winsize *ws = (struct winsize *)arg;
  779:                          short int changed;
  780:
  781:                          if((errno = check_user_area(VERIFY_READ, (void *)arg, si
zeof(struct winsize)))) {
  782:                                  return errno;
  783:                          }
  784:                          changed = 0;
  785:                          if(tty->winsize.ws_row != ws->ws_row) {
  786:                                  changed = 1;
  787:                          }
  788:                          if(tty->winsize.ws_col != ws->ws_col) {
  789:                                  changed = 1;
  790:                          }
  791:                          if(tty->winsize.ws_xpixel != ws->ws_xpixel) {
  792:                                  changed = 1;
  793:                          }
  794:                          if(tty->winsize.ws_ypixel != ws->ws_ypixel) {
  795:                                  changed = 1;
  796:                          }
  797:                          tty->winsize.ws_row = ws->ws_row;
  798:                          tty->winsize.ws_col = ws->ws_col;
  799:                          tty->winsize.ws_xpixel = ws->ws_xpixel;
  800:                          tty->winsize.ws_ypixel = ws->ws_ypixel;
  801:                          if(changed) {
  802:                                  kill_pgrp(tty->pgid, SIGWINCH);
  803:                          }
  804:                  }
  805:                          break;
  806:                  case TIOCNOTTY:
  807:                          if(current->ctty != tty) {
  808:                                  return -ENOTTY;
  809:                          }
  810:                          if(SESS_LEADER(current)) {
  811:                                  disassociate_ctty(tty);
  812:                          }
```

drivers/char/tty.c


```
  813:                          break;
  814:                  case TIOCLINUX:
  815:                  {
  816:                          int val = *(unsigned char *)arg;
  817:                          if((errno = check_user_area(VERIFY_READ, (void *)arg, si
zeof(unsigned char)))) {
  818:                                  return errno;
  819:                          }
  820:                          switch(val) {
  821:                                  case 12:        /* get current console */
  822:                                          return current_cons;
  823:                                          break;
  824:                                  default:
  825:                                          return -EINVAL;
  826:                                          break;
  827:                          }
  828:                          break;
  829:                  }
  830:
  831:                  default:
  832:                          return vt_ioctl(tty, cmd, arg);
  833:          }
  834:          return 0;
  835: }
  836:
  837: int tty_lseek(struct inode *i, __off_t offset)
  838: {
  839:          return -ESPIPE;
  840: }
  841:
  842: int tty_select(struct inode *i, int flag)
  843: {
  844:          struct tty *tty;
  845:
  846:          if(!(tty = get_tty(i->rdev))) {
  847:                  printk("%s(): oops! (%x)\n", __FUNCTION__, i->rdev);
  848:                  return 0;
  849:          }
  850:
  851:          switch(flag) {
  852:                  case SEL_R:
  853:                          if(tty->cooked_q.count > 0) {
  854:                                  if(!(tty->termios.c_lflag & ICANON) || ((tty->te
rmios.c_lflag & ICANON) && tty->canon_data)) {
  855:                                          return 1;
  856:                                  }
  857:                          }
  858:                          break;
  859:                  case SEL_W:
  860:                          if(!tty->write_q.count) {
  861:                                  return 1;
  862:                          }
  863:                          break;
  864:          }
  865:          return 0;
  866: }
  867:
  868: void tty_init(void)
  869: {
  870:          memset_b(tty_table, NULL, sizeof(tty_table));
  871: }
```

drivers/char/tty_queue.c


```
/*
 * fiwix/drivers/char/tty_queue.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/asm.h>
#include <fiwix/kernel.h>
#include <fiwix/tty.h>
#include <fiwix/errno.h>
#include <fiwix/string.h>

/*
 * tty_queue.c implements a queue using a static-sized doubly linked list of a
 * central pool of buffers which covers all ttys.
 *
 *  head                                        tail
 * +--------------+  +--------------+  ...  +--------------+
 * |prev|data|next|  |prev|data|next|  ...  |prev|data|next|
 * | /  |    |  -->  <--  |    |  -->  ...  <--  |    |  / |
 * +--------------+  +--------------+  ...  +--------------+
 *  (cblock)          (cblock)                (cblock)
 */

struct cblock cblock_pool[CB_POOL_SIZE];
struct cblock *cblock_pool_head;

static struct cblock *get_free_cblock(void)
{
        struct cblock *new = NULL;

        if(cblock_pool_head) {
                new = cblock_pool_head;
                cblock_pool_head = cblock_pool_head->next;
                new->prev = new->next = NULL;
        }
        return new;
}

static void put_free_cblock(struct cblock *old)
{
        old->prev = NULL;
        old->next = cblock_pool_head;
        cblock_pool_head = old;
}

static struct cblock *insert_cblock_in_head(struct clist *q)
{
        struct cblock *cb;

        if(q->cb_num >= NR_CB_QUEUE) {
                return NULL;
        }
        if(!(cb = get_free_cblock())) {
                return NULL;
        }

        /* initialize cblock */
        cb->start_off = cb->end_off = 0;
        memset_b(cb->data, 0, CBSIZE);
        cb->prev = cb->next = NULL;
        q->cb_num++;

        if(!q->head) {
                q->head = q->tail = cb;
        } else {
```

drivers/char/tty_queue.c


```
 68:                 cb->prev = NULL;
 69:                 cb->next = q->head;
 70:                 q->head->prev = cb;
 71:                 q->head = cb;
 72:         }
 73:         return cb;
 74: }
 75:
 76: static struct cblock *insert_cblock_in_tail(struct clist *q)
 77: {
 78:         struct cblock *cb;
 79:
 80:         if(q->cb_num >= NR_CB_QUEUE) {
 81:                 return NULL;
 82:         }
 83:         if(!(cb = get_free_cblock())) {
 84:                 return NULL;
 85:         }
 86:
 87:         /* initialize cblock */
 88:         cb->start_off = cb->end_off = 0;
 89:         memset_b(cb->data, 0, CBSIZE);
 90:         cb->prev = cb->next = NULL;
 91:         q->cb_num++;
 92:
 93:         if(!q->tail) {
 94:                 q->head = q->tail = cb;
 95:         } else {
 96:                 cb->prev = q->tail;
 97:                 cb->next = NULL;
 98:                 q->tail->next = cb;
 99:                 q->tail = cb;
100:         }
101:         return cb;
102: }
103:
104: static void delete_cblock_from_head(struct clist *q)
105: {
106:         struct cblock *tmp;
107:
108:         if(!q->head) {
109:                 return;
110:         }
111:
112:         tmp = q->head;
113:         if(q->head == q->tail) {
114:                 q->head = q->tail = NULL;
115:         } else {
116:                 q->head = q->head->next;
117:                 q->head->prev = NULL;
118:         }
119:
120:         q->count -= tmp->end_off - tmp->start_off;
121:         q->cb_num--;
122:         put_free_cblock(tmp);
123: }
124:
125: static void delete_cblock_from_tail(struct clist *q)
126: {
127:         struct cblock *tmp;
128:
129:         if(!q->tail) {
130:                 return;
131:         }
132:
133:         tmp = q->tail;
134:         if(q->head == q->tail) {
```

drivers/char/tty_queue.c


```
135:                 q->head = q->tail = NULL;
136:         } else {
137:                 q->tail = q->tail->prev;
138:                 q->tail->next = NULL;
139:         }
140:
141:         q->count -= tmp->end_off - tmp->start_off;
142:         q->cb_num--;
143:         put_free_cblock(tmp);
144: }
145:
146: int tty_queue_putchar(struct tty *tty, struct clist *q, unsigned char ch)
147: {
148:         unsigned long int flags;
149:         struct cblock *cb;
150:         int errno;
151:
152:         SAVE_FLAGS(flags); CLI();
153:
154:         cb = q->tail;
155:         if(!cb) {
156:                 cb = insert_cblock_in_tail(q);
157:                 if(!cb) {
158:                         RESTORE_FLAGS(flags);
159:                         return -EAGAIN;
160:                 }
161:         }
162:
163:         if(cb->end_off < CBSIZE) {
164:                 cb->data[cb->end_off] = ch;
165:                 cb->end_off++;
166:                 q->count++;
167:                 errno = 0;
168:         } else if(insert_cblock_in_tail(q)) {
169:                 tty_queue_putchar(tty, q, ch);
170:                 errno = 0;
171:         } else {
172:                 errno = -EAGAIN;
173:         }
174:
175:         RESTORE_FLAGS(flags);
176:         return errno;
177: }
178:
179: int tty_queue_unputchar(struct clist *q)
180: {
181:         unsigned long int flags;
182:         struct cblock *cb;
183:         unsigned char ch;
184:
185:         SAVE_FLAGS(flags); CLI();
186:
187:         ch = 0;
188:         cb = q->tail;
189:         if(cb) {
190:                 if(cb->end_off > cb->start_off) {
191:                         ch = cb->data[cb->end_off - 1];
192:                         cb->end_off--;
193:                         q->count--;
194:                 }
195:                 if(cb->end_off - cb->start_off == 0) {
196:                         delete_cblock_from_tail(q);
197:                 }
198:         }
199:
200:         RESTORE_FLAGS(flags);
201:         return ch;
```

drivers/char/tty_queue.c


```
202: }
203:
204: unsigned char tty_queue_getchar(struct clist *q)
205: {
206:         unsigned long int flags;
207:         struct cblock *cb;
208:         unsigned char ch;
209:
210:         SAVE_FLAGS(flags); CLI();
211:
212:         ch = 0;
213:         cb = q->head;
214:         if(cb) {
215:                 if(cb->start_off < cb->end_off) {
216:                         ch = cb->data[cb->start_off];
217:                         cb->start_off++;
218:                         q->count--;
219:                 }
220:                 if(cb->end_off - cb->start_off == 0) {
221:                         delete_cblock_from_head(q);
222:                 }
223:         }
224:
225:         RESTORE_FLAGS(flags);
226:         return ch;
227: }
228:
229: void tty_queue_flush(struct clist *q)
230: {
231:         unsigned long int flags;
232:
233:         SAVE_FLAGS(flags); CLI();
234:
235:         while(q->head != NULL) {
236:                 delete_cblock_from_head(q);
237:         }
238:
239:         RESTORE_FLAGS(flags);
240: }
241:
242: void tty_queue_init(struct tty *tty)
243: {
244:         int n;
245:         struct cblock *cb;
246:
247:         memset_b(cblock_pool, NULL, sizeof(cblock_pool));
248:
249:         /* cblock free list initialization */
250:         cblock_pool_head = NULL;
251:         n = CB_POOL_SIZE;
252:         while(n--) {
253:                 cb = &cblock_pool[n];
254:                 put_free_cblock(cb);
255:         }
256:         tty->read_q.head = tty->read_q.tail = NULL;
257:         tty->cooked_q.head = tty->cooked_q.tail = NULL;
258:         tty->write_q.head = tty->write_q.tail = NULL;
259: }
```

drivers/char/vt.c


```
    1: /*
    2:  * fiwix/drivers/char/vt.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/console.h>
   10: #include <fiwix/keyboard.h>
   11: #include <fiwix/tty.h>
   12: #include <fiwix/vt.h>
   13: #include <fiwix/kd.h>
   14: #include <fiwix/errno.h>
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/string.h>
   17:
   18: int vt_ioctl(struct tty *tty, int cmd, unsigned long int arg)
   19: {
   20:         struct vconsole *vc;
   21:         int n, errno;
   22:
   23:         /* only virtual consoles support the following ioctl commands */
   24:         if(MAJOR(tty->dev) != VCONSOLES_MAJOR) {
   25:                 return -ENXIO;
   26:         }
   27:
   28:         vc = (struct vconsole *)tty->driver_data;
   29:
   30:         switch(cmd) {
   31:                 case KDGETLED:
   32:                         if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(unsigned char)))) {
   33:                                 return errno;
   34:                         }
   35:                         memset_b((void *)arg, vc->led_status, sizeof(char));
   36:                         break;
   37:
   38:                 case KDSETLED:
   39:                         if(arg > 7) {
   40:                                 return -EINVAL;
   41:                         }
   42:                         vc->led_status = arg;
   43:                         set_leds(vc->led_status);
   44:                         break;
   45:
   46:                 case KDGKBTYPE:
   47:                         if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(unsigned char)))) {
   48:                                 return errno;
   49:                         }
   50:                         memset_b((void *)arg, KB_101, sizeof(char));
   51:                         break;
   52:
   53:                 case KDSETMODE:
   54:                         if(arg != KD_TEXT && arg != KD_GRAPHICS) {
   55:                                 return -EINVAL;
   56:                         }
   57:                         if(vc->vc_mode != arg) {
   58:                                 vc->vc_mode = arg;
   59:                                 if(arg == KD_GRAPHICS) {
   60:                                         blank_screen(vc);
   61:                                 } else {
   62:                                         unblank_screen(vc);
   63:                                 }
   64:                         }
   65:                         break;
```

drivers/char/vt.c


```
   66:
   67:                  case KDGETMODE:
   68:                          if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(unsigned char)))) {
   69:                                  return errno;
   70:                          }
   71:                          memset_b((void *)arg, vc->vc_mode, sizeof(char));
   72:                          break;
   73:
   74:                  case KDSKBENT:
   75:                  {
   76:                          struct kbentry *k = (struct kbentry *)arg;
   77:                          if((errno = check_user_area(VERIFY_WRITE, (void *)k, siz
eof(struct kbentry)))) {
   78:                                  return errno;
   79:                          }
   80:                          if(k->kb_table < NR_MODIFIERS) {
   81:                                  if(k->kb_index < NR_SCODES) {
   82:                                          keymap[(k->kb_index * NR_MODIFIERS) + k-
>kb_table] = k->kb_value;
   83:                                  } else {
   84:                                          return -EINVAL;
   85:                                  }
   86:                          } else {
   87:                                  printk("%s(): kb_table value 'd' not supported.
\n", __FUNCTION__, k->kb_table);
   88:                                  return -EINVAL;
   89:                          }
   90:                  }
   91:                          break;
   92:
   93:                  case VT_OPENQRY:
   94:                  {
   95:                          int *val = (int *)arg;
   96:                          if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(unsigned int)))) {
   97:                                  return errno;
   98:                          }
   99:                          for(n = 1; n < NR_VCONSOLES + 1; n++) {
  100:                                  tty = get_tty(MKDEV(VCONSOLES_MAJOR, n));
  101:                                  if(!tty->count) {
  102:                                          break;
  103:                                  }
  104:                          }
  105:                          *val = (n < NR_VCONSOLES + 1 ? n : -1);
  106:                  }
  107:                          break;
  108:
  109:                  case VT_GETMODE:
  110:                  {
  111:                          struct vt_mode *vt_mode = (struct vt_mode *)arg;
  112:                          if((errno = check_user_area(VERIFY_WRITE, (void *)vt_mod
e, sizeof(struct vt_mode)))) {
  113:                                  return errno;
  114:                          }
  115:                          memcpy_b(vt_mode, &vc->vt_mode, sizeof(struct vt_mode));
  116:                  }
  117:                          break;
  118:
  119:                  case VT_SETMODE:
  120:                  {
  121:                          struct vt_mode *vt_mode = (struct vt_mode *)arg;
  122:                          if((errno = check_user_area(VERIFY_READ, (void *)vt_mode
, sizeof(struct vt_mode)))) {
  123:                                  return errno;
  124:                          }
  125:                          if(vt_mode->mode != VT_AUTO && vt_mode->mode != VT_PROCE
```

drivers/char/vt.c


```
SS) {
  126:                                  return -EINVAL;
  127:                          }
  128:                          memcpy_b(&vc->vt_mode, vt_mode, sizeof(struct vt_mode));
  129:                          vc->vt_mode.frsig = 0;  /* ignored */
  130:                          tty->pid = current->pid;
  131:                          vc->switchto_tty = 0;
  132:                  }
  133:                          break;
  134:
  135:                  case VT_GETSTATE:
  136:                  {
  137:                          struct vt_stat *vt_stat = (struct vt_stat *)arg;
  138:                          if((errno = check_user_area(VERIFY_WRITE, (void *)vt_sta
t, sizeof(struct vt_stat)))) {
  139:                                  return errno;
  140:                          }
  141:                          vt_stat->v_active = current_cons;
  142:                          vt_stat->v_state = 1;   /* /dev/tty0 is always opened */
  143:                          for(n = 1; n < NR_VCONSOLES + 1; n++) {
  144:                                  tty = get_tty(MKDEV(VCONSOLES_MAJOR, n));
  145:                                  if(tty->count) {
  146:                                          vt_stat->v_state |= (1 << n);
  147:                                  }
  148:                          }
  149:                  }
  150:                          break;
  151:
  152:                  case VT_RELDISP:
  153:                          if(vc->vt_mode.mode != VT_PROCESS) {
  154:                                  return -EINVAL;
  155:                          }
  156:                          if(vc->switchto_tty < 0) {
  157:                                  if(arg != VT_ACKACQ) {
  158:                                          return -EINVAL;
  159:                                  }
  160:                          } else {
  161:                                  if(arg) {
  162:                                          int switchto_tty;
  163:                                          switchto_tty = vc->switchto_tty;
  164:                                          vc->switchto_tty = -1;
  165:                                          vconsole_select_final(switchto_tty);
  166:                                  } else {
  167:                                          vc->switchto_tty = -1;
  168:                                  }
  169:                          }
  170:                          break;
  171:
  172:                  case VT_ACTIVATE:
  173:                          if(current_cons == MINOR(tty->dev) || IS_SUPERUSER) {
  174:                                  if(!arg || arg > NR_VCONSOLES) {
  175:                                          return -ENXIO;
  176:                                  }
  177:                                  vconsole_select(--arg);
  178:                          } else {
  179:                                  return -EPERM;
  180:                          }
  181:                          break;
  182:
  183:                  case VT_WAITACTIVE:
  184:                          if(current_cons == MINOR(tty->dev)) {
  185:                                  break;
  186:                          }
  187:                          if(!arg || arg > NR_VCONSOLES) {
  188:                                  return -ENXIO;
  189:                          }
  190:                          printk("ACTIVATING another tty!! (cmd = 0x%x)\n", cmd);
```

drivers/char/vt.c


```
191:                            break;
192:
193:                    default:
194:                            return -EINVAL;
195:            }
196:            return 0;
197: }
```

fs/buffer.c


```
    1: /*
    2:  * fiwix/fs/buffer.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: /*
    9:  * buffer.c implements a cache with a free list as a doubly circular linked
   10:  * list and a chained hash table with doubly linked lists.
   11:  *
   12:  * hash table
   13:  * +--------+  +--------------+  +--------------+  +--------------+
   14:  * | index  |  |prev|data|next|  |prev|data|next|  |prev|data|next|
   15:  * |   0   --> | /  |    | --->  <--- |    | --->  <--- |    |  / |
   16:  * +--------+  +--------------+  +--------------+  +--------------+
   17:  * +--------+  +--------------+  +--------------+  +--------------+
   18:  * | index  |  |prev|data|next|  |prev|data|next|  |prev|data|next|
   19:  * |   1   --> | /  |    | --->  <--- |    | --->  <--- |    |  / |
   20:  * +--------+  +--------------+  +--------------+  +--------------+
   21:  *              (buffer)          (buffer)          (buffer)
   22:  *    ...
   23:  */
   24:
   25: #include <fiwix/asm.h>
   26: #include <fiwix/kernel.h>
   27: #include <fiwix/sleep.h>
   28: #include <fiwix/sched.h>
   29: #include <fiwix/buffer.h>
   30: #include <fiwix/devices.h>
   31: #include <fiwix/fs.h>
   32: #include <fiwix/mm.h>
   33: #include <fiwix/errno.h>
   34: #include <fiwix/stdio.h>
   35: #include <fiwix/string.h>
   36: #include <fiwix/stat.h>
   37:
   38: #define BUFFER_HASH(dev, block) (((__dev_t)(dev) ^ (__blk_t)(block)) % (NR_BUF_H
ASH))
   39: #define NR_BUFFERS      buffer_table_size / sizeof(struct buffer)
   40: #define NR_BUF_HASH     buffer_hash_table_size / sizeof(unsigned int)
   41:
   42: struct buffer *buffer_table;            /* buffer pool */
   43: struct buffer *buffer_head;             /* buffer pool head */
   44: struct buffer **buffer_hash_table;
   45:
   46: static struct resource sync_resource = { NULL, NULL };
   47:
   48: static void insert_to_hash(struct buffer *buf)
   49: {
   50:         struct buffer **h;
   51:         int i;
   52:
   53:         i = BUFFER_HASH(buf->dev, buf->block);
   54:         h = &buffer_hash_table[i];
   55:
   56:         if(!*h) {
   57:                 *h = buf;
   58:                 (*h)->prev_hash = (*h)->next_hash = NULL;
   59:         } else {
   60:                 buf->prev_hash = NULL;
   61:                 buf->next_hash = *h;
   62:                 (*h)->prev_hash = buf;
   63:                 *h = buf;
   64:         }
   65: }
   66:
```

fs/buffer.c


```
 67: static void remove_from_hash(struct buffer *buf)
 68: {
 69:         struct buffer **h;
 70:         int i;
 71:
 72:         i = BUFFER_HASH(buf->dev, buf->block);
 73:         h = &buffer_hash_table[i];
 74:
 75:         while(*h) {
 76:                 if(*h == buf) {
 77:                         if((*h)->next_hash) {
 78:                                 (*h)->next_hash->prev_hash = (*h)->prev_hash;
 79:                         }
 80:                         if((*h)->prev_hash) {
 81:                                 (*h)->prev_hash->next_hash = (*h)->next_hash;
 82:                         }
 83:                         if(h == &buffer_hash_table[i]) {
 84:                                 *h = (*h)->next_hash;
 85:                         }
 86:                         break;
 87:                 }
 88:                 h = &(*h)->next_hash;
 89:         }
 90: }
 91:
 92: static void remove_from_free_list(struct buffer *buf)
 93: {
 94:         buf->prev_free->next_free = buf->next_free;
 95:         buf->next_free->prev_free = buf->prev_free;
 96:         if(buf == buffer_head) {
 97:                 buffer_head = buf->next_free;
 98:         }
 99: }
100:
101: static void buffer_wait(struct buffer *buf)
102: {
103:         unsigned long int flags;
104:
105:         for(;;) {
106:                 SAVE_FLAGS(flags); CLI();
107:                 if(buf->locked) {
108:                         RESTORE_FLAGS(flags);
109:                         sleep(&buffer_wait, PROC_UNINTERRUPTIBLE);
110:                 } else {
111:                         break;
112:                 }
113:         }
114:         buf->locked = 1;
115:         RESTORE_FLAGS(flags);
116: }
117:
118: static struct buffer * get_free_buffer(void)
119: {
120:         unsigned long int flags;
121:         struct buffer *buf;
122:
123:         /* no more buffers on free list */
124:         if(buffer_head == buffer_head->next_free) {
125:                 return NULL;
126:         }
127:
128:         for(;;) {
129:                 SAVE_FLAGS(flags); CLI();
130:                 buf = buffer_head;
131:                 if(buf->locked) {
132:                         RESTORE_FLAGS(flags);
133:                         sleep(&buffer_wait, PROC_UNINTERRUPTIBLE);
```

fs/buffer.c


```
  134:                 } else {
  135:                         break;
  136:                 }
  137:         }
  138:
  139:         buf = buffer_head;
  140:         remove_from_free_list(buf);
  141:         buf->locked = 1;
  142:
  143:         RESTORE_FLAGS(flags);
  144:         return buf;
  145: }
  146:
  147: static void sync_one_buffer(struct buffer *buf)
  148: {
  149:         struct device *d;
  150:         int errno;
  151:
  152:         if(!(d = get_device(BLK_DEV, MAJOR(buf->dev)))) {
  153:                 printk("WARNING: %s(): block device %d,%d not registered!\n", __
FUNCTION__, MAJOR(buf->dev), MINOR(buf->dev));
  154:                 return;
  155:         }
  156:
  157:         if(d->fsop && d->fsop->write_block) {
  158:                 errno = d->fsop->write_block(buf->dev, buf->block, buf->data, bu
f->size);
  159:                 if(errno < 0) {
  160:                         if(errno == -EROFS) {
  161:                                 printk("WARNING: %s(): write protection on devic
e %d,%d.\n", __FUNCTION__, MAJOR(buf->dev), MINOR(buf->dev), buf->block);
  162:                         } else {
  163:                                 printk("WARNING: %s(): I/O error on device %d,%d
.\n", __FUNCTION__, MAJOR(buf->dev), MINOR(buf->dev), buf->block);
  164:                         }
  165:                         return;
  166:                 }
  167:                 buf->dirty = 0;
  168:         } else {
  169:                 printk("WARNING: %s(): device %d,%d does not have the write_bloc
k() method!\n", __FUNCTION__, MAJOR(buf->dev), MINOR(buf->dev));
  170:         }
  171: }
  172:
  173: static struct buffer * search_buffer_hash(__dev_t dev, __blk_t block, int size)
  174: {
  175:         struct buffer *buf;
  176:         int i;
  177:
  178:         i = BUFFER_HASH(dev, block);
  179:         buf = buffer_hash_table[i];
  180:
  181:         while(buf) {
  182:                 if(buf->dev == dev && buf->block == block && buf->size == size)
{
  183:                         return buf;
  184:                 }
  185:                 buf = buf->next_hash;
  186:         }
  187:
  188:         return NULL;
  189: }
  190:
  191: static struct buffer * getblk(__dev_t dev, __blk_t block, int size)
  192: {
  193:         unsigned long int flags;
  194:         struct buffer *buf;
```

fs/buffer.c


```
  195:
  196:         for(;;) {
  197:                 if((buf = search_buffer_hash(dev, block, size))) {
  198:                         SAVE_FLAGS(flags); CLI();
  199:                         if(buf->locked) {
  200:                                 RESTORE_FLAGS(flags);
  201:                                 sleep(&buffer_wait, PROC_UNINTERRUPTIBLE);
  202:                                 continue;
  203:                         }
  204:                         buf->locked = 1;
  205:                         remove_from_free_list(buf);
  206:                         RESTORE_FLAGS(flags);
  207:                         return buf;
  208:                 }
  209:
  210:                 if(!(buf = get_free_buffer())) {
  211:                         printk("WARNING: %s(): no more buffers on free list!\n",
 __FUNCTION__);
  212:                         sleep(&get_free_buffer, PROC_UNINTERRUPTIBLE);
  213:                         continue;
  214:                 }
  215:
  216:                 if(buf->dirty) {
  217:                         sync_one_buffer(buf);
  218:                 } else {
  219:                         if(!buf->data) {
  220:                                 if(!(buf->data = (char *)kmalloc())) {
  221:                                         brelse(buf);
  222:                                         printk("%s(): returning NULL\n", __FUNCT
ION__);
  223:                                         return NULL;
  224:                                 }
  225:                                 kstat.buffers += (PAGE_SIZE / 1024);
  226:                         }
  227:                 }
  228:
  229:                 SAVE_FLAGS(flags); CLI();
  230:                 remove_from_hash(buf);  /* remove it from old hash */
  231:                 buf->dev = dev;
  232:                 buf->block = block;
  233:                 buf->size = size;
  234:                 insert_to_hash(buf);
  235:                 buf->valid = 0;
  236:                 RESTORE_FLAGS(flags);
  237:                 return buf;
  238:         }
  239: }
  240:
  241: struct buffer * get_dirty_buffer(__dev_t dev, __blk_t block, int size)
  242: {
  243:         unsigned long int flags;
  244:         struct buffer *buf;
  245:
  246:         for(;;) {
  247:                 if((buf = search_buffer_hash(dev, block, size))) {
  248:                         if(buf->dirty) {
  249:                                 SAVE_FLAGS(flags); CLI();
  250:                                 if(buf->locked) {
  251:                                         RESTORE_FLAGS(flags);
  252:                                         sleep(&buffer_wait, PROC_UNINTERRUPTIBLE
);
  253:                                         continue;
  254:                                 }
  255:                                 buf->locked = 1;
  256:                                 remove_from_free_list(buf);
  257:                                 RESTORE_FLAGS(flags);
  258:                                 break;
```

fs/buffer.c Page 5/7

```
  259:                         }
  260:                 }
  261:                 buf = NULL;
  262:                 break;
  263:         }
  264:
  265:         return buf;
  266: }
  267:
  268: struct buffer * bread(__dev_t dev, __blk_t block, int size)
  269: {
  270:         struct buffer *buf;
  271:         struct device *d;
  272:
  273:         if(!(d = get_device(BLK_DEV, MAJOR(dev)))) {
  274:                 printk("WARNING: %s(): device major %d not found!\n", __FUNCTION
__, MAJOR(dev));
  275:                 return NULL;
  276:         }
  277:
  278:         if((buf = getblk(dev, block, size))) {
  279:                 if(!buf->valid) {
  280:                         if(d->fsop && d->fsop->read_block) {
  281:                                 if(d->fsop->read_block(dev, block, buf->data, si
ze) >= 0) {
  282:                                         buf->valid = 1;
  283:                                 }
  284:                         }
  285:                 }
  286:                 if(buf->valid) {
  287:                         return buf;
  288:                 }
  289:                 brelse(buf);
  290:         }
  291:
  292:         printk("WARNING: %s(): returning NULL!\n", __FUNCTION__);
  293:         return NULL;
  294: }
  295:
  296: void bwrite(struct buffer *buf)
  297: {
  298:         buf->dirty = 1;
  299:         buf->valid = 1;
  300:         brelse(buf);
  301: }
  302:
  303: void brelse(struct buffer *buf)
  304: {
  305:         unsigned long int flags;
  306:
  307:         SAVE_FLAGS(flags); CLI();
  308:
  309:         if(!buffer_head) {
  310:                 buf->prev_free = buf->next_free = buf;
  311:                 buffer_head = buf;
  312:         } else {
  313:                 buf->next_free = buffer_head;
  314:                 buf->prev_free = buffer_head->prev_free;
  315:                 buffer_head->prev_free->next_free = buf;
  316:                 buffer_head->prev_free = buf;
  317:
  318:                 /* if not valid place the buffer at the head of the free list */
  319:                 if(!buf->valid) {
  320:                         buffer_head = buf;
  321:                 }
  322:         }
  323:         buf->locked = 0;
```

fs/buffer.c

```
324:
325:         RESTORE_FLAGS(flags);
326:
327:         wakeup(&get_free_buffer);
328:         wakeup(&buffer_wait);
329: }
330:
331: void sync_buffers(__dev_t dev)
332: {
333:         struct buffer *buf;
334:         int n;
335:
336:         buf = &buffer_table[0];
337:
338:         lock_resource(&sync_resource);
339:         for(n = 0; n < NR_BUFFERS; n++) {
340:                 if(buf->dirty) {
341:                         if(!dev || buf->dev == dev) {
342:                                 buffer_wait(buf);
343:                                 sync_one_buffer(buf);
344:                                 buf->locked = 0;
345:                                 wakeup(&buffer_wait);
346:                         }
347:                 }
348:                 buf++;
349:         }
350:         unlock_resource(&sync_resource);
351:         return;
352: }
353:
354: void invalidate_buffers(__dev_t dev)
355: {
356:         unsigned long int flags;
357:         unsigned int n;
358:         struct buffer *buf;
359:
360:         buf = &buffer_table[0];
361:         SAVE_FLAGS(flags); CLI();
362:
363:         for(n = 0; n < NR_BUFFERS; n++) {
364:                 if(!buf->locked && buf->dev == dev) {
365:                         buffer_wait(buf);
366:                         remove_from_hash(buf);
367:                         buf->valid = 0;
368:                         buf->locked = 0;
369:                         wakeup(&buffer_wait);
370:                 }
371:                 buf++;
372:         }
373:
374:         RESTORE_FLAGS(flags);
375:         /* FIXME: invalidate_pages(dev); */
376: }
377:
378: /*
379:  * When kernel runs out of pages, kswapd is awaken and it calls this function
380:  * which goes throught the buffer free list, freeing up to NR_BUF_RECLAIM
381:  * buffers.
382:  */
383: int reclaim_buffers(void)
384: {
385:         struct buffer *buf, *first;
386:         int reclaimed;
387:
388:         reclaimed = 0;
389:         first = NULL;
390:
```

fs/buffer.c


```
  391:         for(;;) {
  392:                 if(!(buf = get_free_buffer())) {
  393:                         printk("WARNING: %s(): no more buffers on free list!\n",
 __FUNCTION__);
  394:                         sleep(&get_free_buffer, PROC_UNINTERRUPTIBLE);
  395:                         continue;
  396:                 }
  397:
  398:                 remove_from_hash(buf);
  399:                 if(buf->dirty) {
  400:                         sync_one_buffer(buf);
  401:                 }
  402:
  403:                 /* this ensures the buffer will go to the tail */
  404:                 buf->valid = 1;
  405:
  406:                 if(first) {
  407:                         if(first == buf) {
  408:                                 brelse(buf);
  409:                                 break;
  410:                         }
  411:                 } else {
  412:                         first = buf;
  413:                 }
  414:                 if(buf->data) {
  415:                         kfree((unsigned int)buf->data);
  416:                         buf->data = NULL;
  417:                         kstat.buffers -= (PAGE_SIZE / 1024);
  418:                         reclaimed++;
  419:                         if(reclaimed == NR_BUF_RECLAIM) {
  420:                                 brelse(buf);
  421:                                 break;
  422:                         }
  423:                 }
  424:                 brelse(buf);
  425:                 do_sched();
  426:         }
  427:
  428:         wakeup(&buffer_wait);
  429:         return reclaimed;
  430: }
  431:
  432: void buffer_init(void)
  433: {
  434:         struct buffer *buf;
  435:         unsigned int n;
  436:
  437:         memset_b(buffer_table, NULL, buffer_table_size);
  438:         memset_b(buffer_hash_table, NULL, buffer_hash_table_size);
  439:         for(n = 0; n < NR_BUFFERS; n++) {
  440:                 buf = &buffer_table[n];
  441:                 brelse(buf);
  442:         }
  443: }
```

fs/devices.c


```
/*
 * fiwix/fs/devices.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/buffer.h>
#include <fiwix/devices.h>
#include <fiwix/fs.h>
#include <fiwix/mm.h>
#include <fiwix/process.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct device chr_device_table[NR_CHRDEV];
struct device blk_device_table[NR_BLKDEV];

struct fs_operations def_chr_fsop = {
        0,
        0,

        chr_dev_open,
        NULL,                   /* close */
        NULL,                   /* read */
        NULL,                   /* write */
        NULL,                   /* ioctl */
        NULL,                   /* lseek */
        NULL,                   /* readdir */
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        NULL,                   /* bmap */
        NULL,                   /* lockup */
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        NULL,                   /* truncate */
        NULL,                   /* create */
        NULL,                   /* rename */

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* stats */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

struct fs_operations def_blk_fsop = {
        0,
        0,

        blk_dev_open,
```

fs/devices.c


```
   68:         blk_dev_close,
   69:         blk_dev_read,
   70:         blk_dev_write,
   71:         blk_dev_ioctl,
   72:         blk_dev_lseek,
   73:         NULL,                   /* readdir */
   74:         NULL,                   /* mmap */
   75:         NULL,                   /* select */
   76:
   77:         NULL,                   /* readlink */
   78:         NULL,                   /* followlink */
   79:         NULL,                   /* bmap */
   80:         NULL,                   /* lockup */
   81:         NULL,                   /* rmdir */
   82:         NULL,                   /* link */
   83:         NULL,                   /* unlink */
   84:         NULL,                   /* symlink */
   85:         NULL,                   /* mkdir */
   86:         NULL,                   /* mknod */
   87:         NULL,                   /* truncate */
   88:         NULL,                   /* create */
   89:         NULL,                   /* rename */
   90:
   91:         NULL,                   /* read_block */
   92:         NULL,                   /* write_block */
   93:
   94:         NULL,                   /* read_inode */
   95:         NULL,                   /* write_inode */
   96:         NULL,                   /* ialloc */
   97:         NULL,                   /* ifree */
   98:         NULL,                   /* stats */
   99:         NULL,                   /* read_superblock */
  100:         NULL,                   /* remount_fs */
  101:         NULL,                   /* write_superblock */
  102:         NULL                    /* release_superblock */
  103: };
  104:
  105: int register_device(int type, struct device *new_d)
  106: {
  107:         struct device *d;
  108:
  109:         switch(type) {
  110:                 case CHR_DEV:
  111:                         if(new_d->major >= NR_CHRDEV) {
  112:                                 printk("%s(): character device major %d is great
er than NR_CHRDEV (%d).\n", __FUNCTION__, new_d->major, NR_CHRDEV);
  113:                                 return 1;
  114:                         }
  115:                         d = chr_device_table;
  116:                         break;
  117:                 case BLK_DEV:
  118:                         if(new_d->major >= NR_BLKDEV) {
  119:                                 printk("%s(): block device major %d is greater t
han NR_BLKDEV (%d).\n", __FUNCTION__, new_d->major, NR_BLKDEV);
  120:                                 return 1;
  121:                         }
  122:                         d = blk_device_table;
  123:                         break;
  124:                 default:
  125:                         printk("WARNING: %s(): invalid device type %d.\n", __FUN
CTION__, type);
  126:                         return 1;
  127:                         break;
  128:         }
  129:
  130:         if(d[new_d->major].major) {
  131:                 printk("%s(): device '%s' with major %d is already registered.\n
```

fs/devices.c


```
", __FUNCTION__, new_d->name, new_d->major);
  132:                  return 1;
  133:          }
  134:          memcpy_b(d + new_d->major, new_d, sizeof(struct device));
  135:          return 0;
  136: }
  137:
  138: struct device * get_device(int type, unsigned char major)
  139: {
  140:          char *name;
  141:          struct device *d;
  142:
  143:          switch(type) {
  144:                  case CHR_DEV:
  145:                          if(major >= NR_CHRDEV) {
  146:                                  printk("%s(): character device major %d is great
er than NR_CHRDEV (%d).\n", __FUNCTION__, major, NR_CHRDEV);
  147:                                  return NULL;
  148:                          }
  149:                          d = chr_device_table;
  150:                          name = "character";
  151:                          break;
  152:                  case BLK_DEV:
  153:                          if(major >= NR_BLKDEV) {
  154:                                  printk("%s(): block device major %d is greater t
han NR_BLKDEV (%d).\n", __FUNCTION__, major, NR_BLKDEV);
  155:                                  return NULL;
  156:                          }
  157:                          d = blk_device_table;
  158:                          name = "block";
  159:                          break;
  160:                  default:
  161:                          printk("WARNING: %s(): invalid device type %d.\n", __FUN
CTION__, type);
  162:                          return NULL;
  163:          }
  164:
  165:          if(d[major].major) {
  166:                  return &d[major];
  167:          }
  168:
  169:          printk("WARNING: %s(): no %s device found with major %d.\n", __FUNCTION_
_, name, major);
  170:          return NULL;
  171: }
  172:
  173: int chr_dev_open(struct inode *i, struct fd *fd_table)
  174: {
  175:          struct device *d;
  176:
  177:          if((d = get_device(CHR_DEV, MAJOR(i->rdev)))) {
  178:                  i->fsop = d->fsop;
  179:                  if(i->fsop && i->fsop->open) {
  180:                          return i->fsop->open(i, fd_table);
  181:                  }
  182:          }
  183:
  184:          return -EINVAL;
  185: }
  186:
  187: int blk_dev_open(struct inode *i, struct fd *fd_table)
  188: {
  189:          struct device *d;
  190:
  191:          if((d = get_device(BLK_DEV, MAJOR(i->rdev)))) {
  192:                  if(d->fsop && d->fsop->open) {
  193:                          return d->fsop->open(i, fd_table);
```

fs/devices.c


```
  194:                 }
  195:         }
  196:
  197:         return -EINVAL;
  198: }
  199:
  200: int blk_dev_close(struct inode *i, struct fd *fd_table)
  201: {
  202:         struct device *d;
  203:
  204:         if((d = get_device(BLK_DEV, MAJOR(i->rdev)))) {
  205:                 if(d->fsop && d->fsop->close) {
  206:                         return d->fsop->close(i, fd_table);
  207:                 }
  208:         }
  209:
  210:         printk("WARNING: %s(): block device %d,%d does not have the close() meth
od.\n", __FUNCTION__, MAJOR(i->rdev), MINOR(i->rdev));
  211:         return -EINVAL;
  212: }
  213:
  214: int blk_dev_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t co
unt)
  215: {
  216:         __blk_t block;
  217:         __off_t total_read;
  218:         unsigned long long int device_size;
  219:         int blksize;
  220:         unsigned int boffset, bytes;
  221:         struct buffer *buf;
  222:         struct device *d;
  223:
  224:         if(!(d = get_device(BLK_DEV, MAJOR(i->rdev)))) {
  225:                 return -EINVAL;
  226:         }
  227:
  228:         blksize = d->blksize ? d->blksize : BLKSIZE_1K;
  229:         total_read = 0;
  230:         if(!d->device_data) {
  231:                 printk("%s(): don't know the size of the block device %d,%d.\n",
 __FUNCTION__, MAJOR(i->rdev), MINOR(i->rdev));
  232:                 return -EIO;
  233:         }
  234:
  235:         device_size = ((unsigned int *)d->device_data)[MINOR(i->rdev)];
  236:         device_size *= 1024LLU;
  237:
  238:         count = (fd_table->offset + count > device_size) ? device_size - fd_tabl
e->offset : count;
  239:         if(!count || fd_table->offset > device_size) {
  240:                 return 0;
  241:         }
  242:         while(count) {
  243:                 boffset = fd_table->offset % blksize;
  244:                 block = (fd_table->offset / blksize);
  245:                 if(!(buf = bread(i->rdev, block, blksize))) {
  246:                         return -EIO;
  247:                 }
  248:                 bytes = blksize - boffset;
  249:                 bytes = MIN(bytes, count);
  250:                 memcpy_b(buffer + total_read, buf->data + boffset, bytes);
  251:                 total_read += bytes;
  252:                 count -= bytes;
  253:                 boffset += bytes;
  254:                 boffset %= blksize;
  255:                 fd_table->offset += bytes;
  256:                 brelse(buf);
```

fs/devices.c


```
  257:          }
  258:          return total_read;
  259: }
  260:
  261: int blk_dev_write(struct inode *i, struct fd *fd_table, const char *buffer, __si
ze_t count)
  262: {
  263:          __blk_t block;
  264:          __off_t total_written;
  265:          unsigned long long int device_size;
  266:          int blksize;
  267:          unsigned int boffset, bytes;
  268:          struct buffer *buf;
  269:          struct device *d;
  270:
  271:          if(!(d = get_device(BLK_DEV, MAJOR(i->rdev)))) {
  272:                  return -EINVAL;
  273:          }
  274:
  275:          blksize = d->blksize ? d->blksize : BLKSIZE_1K;
  276:          total_written = 0;
  277:          if(!d->device_data) {
  278:                  printk("%s(): don't know the size of the block device %d,%d.\n",
 __FUNCTION__, MAJOR(i->rdev), MINOR(i->rdev));
  279:                  return -EIO;
  280:          }
  281:
  282:          device_size = ((unsigned int *)d->device_data)[MINOR(i->rdev)];
  283:          device_size *= 1024LLU;
  284:
  285:          count = (fd_table->offset + count > device_size) ? device_size - fd_tabl
e->offset : count;
  286:          if(!count || fd_table->offset > device_size) {
  287:                  printk("%s(): I/O error on device %d,%d, offset %u.\n", __FUNCTI
ON__, MAJOR(i->rdev), MINOR(i->rdev), fd_table->offset);
  288:                  return -EIO;
  289:          }
  290:          while(count) {
  291:                  boffset = fd_table->offset % blksize;
  292:                  block = (fd_table->offset / blksize);
  293:                  if(!(buf = bread(i->rdev, block, blksize))) {
  294:                          return -EIO;
  295:                  }
  296:                  bytes = blksize - boffset;
  297:                  bytes = MIN(bytes, count);
  298:                  memcpy_b(buf->data + boffset, buffer + total_written, bytes);
  299:                  total_written += bytes;
  300:                  count -= bytes;
  301:                  boffset += bytes;
  302:                  boffset %= blksize;
  303:                  fd_table->offset += bytes;
  304:                  bwrite(buf);
  305:          }
  306:          return total_written;
  307: }
  308:
  309: int blk_dev_ioctl(struct inode *i, int cmd, unsigned long int arg)
  310: {
  311:          struct device *d;
  312:
  313:          if((d = get_device(BLK_DEV, MAJOR(i->rdev)))) {
  314:                  if(d->fsop && d->fsop->ioctl) {
  315:                          return d->fsop->ioctl(i, cmd, arg);
  316:                  }
  317:          }
  318:
  319:          printk("WARNING: %s(): block device %d,%d does not have the ioctl() meth
```

fs/devices.c


```
od.\n", __FUNCTION__, MAJOR(i->rdev), MINOR(i->rdev));
  320:          return -EINVAL;
  321: }
  322:
  323: int blk_dev_lseek(struct inode *i, __off_t offset)
  324: {
  325:          struct device *d;
  326:
  327:          if((d = get_device(BLK_DEV, MAJOR(i->rdev)))) {
  328:                  if(d->fsop && d->fsop->lseek) {
  329:                          return d->fsop->lseek(i, offset);
  330:                  }
  331:          }
  332:
  333:          return offset;
  334: }
  335:
  336: void dev_init(void)
  337: {
  338:          memset_b(chr_device_table, NULL, sizeof(chr_device_table));
  339:          memset_b(blk_device_table, NULL, sizeof(blk_device_table));
  340: }
```

fs/elf.c


```
    1: /*
    2:  * fiwix/fs/elf.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/asm.h>
   10: #include <fiwix/types.h>
   11: #include <fiwix/buffer.h>
   12: #include <fiwix/fs.h>
   13: #include <fiwix/i386elf.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/mman.h>
   16: #include <fiwix/fs.h>
   17: #include <fiwix/fcntl.h>
   18: #include <fiwix/process.h>
   19: #include <fiwix/errno.h>
   20: #include <fiwix/stdio.h>
   21: #include <fiwix/string.h>
   22:
   23: #define AT_ITEMS        12      /* ELF Auxiliary Vectors */
   24:
   25: static int check_elf(struct elf32_hdr *elf32_h)
   26: {
   27:         if(elf32_h->e_ident[EI_MAG0] != ELFMAG0 ||
   28:                 elf32_h->e_ident[EI_MAG1] != ELFMAG1 ||
   29:                 elf32_h->e_ident[EI_MAG2] != ELFMAG2 ||
   30:                 elf32_h->e_ident[EI_MAG3] != ELFMAG3 ||
   31:                 (elf32_h->e_type != ET_EXEC && elf32_h->e_type != ET_DYN) ||
   32:                 elf32_h->e_machine != EM_386) {
   33:                 return -EINVAL;
   34:         }
   35:         return 0;
   36: }
   37:
   38: /*
   39:  * Setup the initial process stack (System V ABI for i386)
   40:  * ------------------------------------------------------------------------
   41:  * 0xBFFFFFFF
   42:  *      +---------------+       \
   43:  *      | envp[] str    |       |
   44:  *      +---------------+       |
   45:  *      | argv[] str    |       |
   46:  *      +---------------+       |
   47:  *      | NULL          |       |
   48:  *      +---------------+       |
   49:  *      | ELF Aux.Vect. |       |
   50:  *      +---------------+       |
   51:  *      | NULL          |       | elf_create_stack() setups this section
   52:  *      +---------------+       |
   53:  *      | envp[] ptr    |       |
   54:  *      +---------------+       |
   55:  *      | NULL          |       |
   56:  *      +---------------+       |
   57:  *      | argv[] ptr    |       |
   58:  *      +---------------+       |
   59:  *      | argc          |       |
   60:  *      +---------------+       /
   61:  *      | stack pointer | grows toward lower addresses
   62:  *      +---------------+ ||
   63:  *      |...............| \/
   64:  *      |...............|
   65:  *      |...............|
   66:  *      |...............| /\
   67:  *      +---------------+ ||
```

fs/elf.c

```
   68:  *      |  brk (heap)   | grows toward higher addresses
   69:  *      +---------------+
   70:  *      | .bss section  |
   71:  *      +---------------+
   72:  *      | .data section |
   73:  *      +---------------+
   74:  *      | .text section |
   75:  *      +---------------+
   76:  * 0x08048000
   77:  */
   78: static void elf_create_stack(struct binargs *barg, unsigned int *sp, unsigned in
t str_ptr, int at_base, struct elf32_hdr *elf32_h, unsigned int phdr_addr)
   79: {
   80:         unsigned int n, addr;
   81:         char *str;
   82:
   83:         /* copy strings */
   84:         for(n = 0; n < ARG_MAX; n++) {
   85:                 if(barg->page[n]) {
   86:                         addr = KERNEL_BASE_ADDR - ((ARG_MAX - n) * PAGE_SIZE);
   87:                         memcpy_b((void *)addr, (void *)barg->page[n], PAGE_SIZE)
;
   88:                 }
   89:         }
   90:
   91: #ifdef __DEBUG__
   92:         printk("sp = 0x%08x\n", sp);
   93: #endif /*__DEBUG__ */
   94:
   95:         /* copy the value of 'argc' into the stack */
   96:         memcpy_l((void *)sp, &barg->argc, 1);
   97: #ifdef __DEBUG__
   98:         printk("at 0x%08x -> argc\n", sp);
   99: #endif /*__DEBUG__ */
  100:         sp++;
  101:
  102:         /* copy as many pointers to strings as 'argc' */
  103:         current->argv = (char **)sp;
  104:         for(n = 0; n < barg->argc; n++) {
  105:                 memcpy_l((void *)sp, &str_ptr, 1);
  106:                 str = (char *)str_ptr;
  107: #ifdef __DEBUG__
  108:                 printk("at 0x%08x -> str_ptr(%d) = 0x%08x (+ %d)\n", sp, n, str_
ptr, strlen(str) + 1);
  109: #endif /*__DEBUG__ */
  110:                 sp++;
  111:                 str_ptr += strlen(str) + 1;
  112:         }
  113:
  114:         /* the last element of 'argv[]' must be a NULL-pointer */
  115:         memset_l((void *)sp, NULL, 1);
  116: #ifdef __DEBUG__
  117:         printk("at 0x%08x -> -------------- = 0x%08x\n", sp, 0);
  118: #endif /*__DEBUG__ */
  119:         sp++;
  120:
  121:         /* copy as many pointers to strings as 'envc' */
  122:         current->envp = (char **)sp;
  123:         for(n = 0; n < barg->envc; n++) {
  124:                 memcpy_l((void *)sp, &str_ptr, 1);
  125:                 str = (char *)str_ptr;
  126: #ifdef __DEBUG__
  127:                 printk("at 0x%08x -> str_ptr(%d) = 0x%08x (+ %d)\n", sp, n, str_
ptr, strlen(str) + 1);
  128: #endif /*__DEBUG__ */
  129:                 sp++;
  130:                 str_ptr += strlen(str) + 1;
```

fs/elf.c Page 3/10

```
131:         }
132:
133:         /* the last element of 'envp[]' must be a NULL-pointer */
134:         memset_l((void *)sp, NULL, 1);
135: #ifdef __DEBUG__
136:         printk("at 0x%08x -> -------------- = 0x%08x\n", sp, 0);
137: #endif /*__DEBUG__ */
138:         sp++;
139:
140:
141:         /* copy the Auxiliar Table Items (dlinfo_items) */
142:         if(at_base) {
143:                 memset_l((void *)sp, AT_PHDR, 1);
144: #ifdef __DEBUG__
145:                 printk("at 0x%08x -> AT_PHDR = %d", sp, *sp);
146: #endif /*__DEBUG__ */
147:                 sp++;
148:
149:                 memcpy_l((void *)sp, &phdr_addr, 1);
150: #ifdef __DEBUG__
151:                 printk("\t\tAT_PHDR = 0x%08x\n", *sp);
152: #endif /*__DEBUG__ */
153:                 sp++;
154:
155:                 memset_l((void *)sp, AT_PHENT, 1);
156: #ifdef __DEBUG__
157:                 printk("at 0x%08x -> AT_PHENT = %d", sp, *sp);
158: #endif /*__DEBUG__ */
159:                 sp++;
160:
161:                 memset_l((void *)sp, sizeof(struct elf32_phdr), 1);
162: #ifdef __DEBUG__
163:                 printk("\t\tAT_PHENT = %d\n", *sp);
164: #endif /*__DEBUG__ */
165:                 sp++;
166:
167:                 memset_l((void *)sp, AT_PHNUM, 1);
168: #ifdef __DEBUG__
169:                 printk("at 0x%08x -> AT_PHNUM = %d", sp, *sp);
170: #endif /*__DEBUG__ */
171:                 sp++;
172:
173:                 memset_l((void *)sp, 0, 1);
174:                 memcpy_w((void *)sp, &elf32_h->e_phnum, 1);
175: #ifdef __DEBUG__
176:                 printk("\t\tAT_PHNUM = %d\n", *sp);
177: #endif /*__DEBUG__ */
178:                 sp++;
179:
180:                 memset_l((void *)sp, AT_PAGESZ, 1);
181: #ifdef __DEBUG__
182:                 printk("at 0x%08x -> AT_PGSIZE = %d", sp, *sp);
183: #endif /*__DEBUG__ */
184:                 sp++;
185:
186:                 memset_l((void *)sp, PAGE_SIZE, 1);
187: #ifdef __DEBUG__
188:                 printk("\t\tAT_PGSIZE = %d\n", *sp);
189: #endif /*__DEBUG__ */
190:                 sp++;
191:
192:                 memset_l((void *)sp, AT_BASE, 1);
193: #ifdef __DEBUG__
194:                 printk("at 0x%08x -> AT_BASE = %d", sp, *sp);
195: #endif /*__DEBUG__ */
196:                 sp++;
197:
```

fs/elf.c

```
198:                 memset_l((void *)sp, at_base, 1);
199: #ifdef __DEBUG__
200:                 printk("\t\tAT_BASE = 0x%08x\n", sp);
201: #endif /*__DEBUG__ */
202:                 sp++;
203:
204:                 memset_l((void *)sp, AT_FLAGS, 1);
205: #ifdef __DEBUG__
206:                 printk("at 0x%08x -> AT_FLAGS = %d", sp, *sp);
207: #endif /*__DEBUG__ */
208:                 sp++;
209:
210:                 memset_l((void *)sp, NULL, 1);
211: #ifdef __DEBUG__
212:                 printk("\t\tAT_FLAGS = %d\n", *sp);
213: #endif /*__DEBUG__ */
214:                 sp++;
215:
216:                 memset_l((void *)sp, AT_ENTRY, 1);
217: #ifdef __DEBUG__
218:                 printk("at 0x%08x -> AT_ENTRY = %d ", sp, *sp);
219: #endif /*__DEBUG__ */
220:                 sp++;
221:
222:                 memcpy_l((void *)sp, &elf32_h->e_entry, 1);
223: #ifdef __DEBUG__
224:                 printk("\t\tAT_ENTRY = 0x%08x\n", *sp);
225: #endif /*__DEBUG__ */
226:                 sp++;
227:
228:                 memset_l((void *)sp, AT_UID, 1);
229: #ifdef __DEBUG__
230:                 printk("at 0x%08x -> AT_UID = %d", sp, *sp);
231: #endif /*__DEBUG__ */
232:                 sp++;
233:
234:                 memcpy_l((void *)sp, &current->uid, 1);
235: #ifdef __DEBUG__
236:                 printk("\t\tAT_UID = %d\n", *sp);
237: #endif /*__DEBUG__ */
238:                 sp++;
239:
240:                 memset_l((void *)sp, AT_EUID, 1);
241: #ifdef __DEBUG__
242:                 printk("at 0x%08x -> AT_EUID = %d", sp, *sp);
243: #endif /*__DEBUG__ */
244:                 sp++;
245:
246:                 memcpy_l((void *)sp, &current->euid, 1);
247: #ifdef __DEBUG__
248:                 printk("\t\tAT_EUID = %d\n", *sp);
249: #endif /*__DEBUG__ */
250:                 sp++;
251:
252:                 memset_l((void *)sp, AT_GID, 1);
253: #ifdef __DEBUG__
254:                 printk("at 0x%08x -> AT_GID = %d", sp, *sp);
255: #endif /*__DEBUG__ */
256:                 sp++;
257:
258:                 memcpy_l((void *)sp, &current->gid, 1);
259: #ifdef __DEBUG__
260:                 printk("\t\tAT_GID = %d\n", *sp);
261: #endif /*__DEBUG__ */
262:                 sp++;
263:
264:                 memset_l((void *)sp, AT_EGID, 1);
```

fs/elf.c

```
265: #ifdef __DEBUG__
266:                 printk("at 0x%08x -> AT_EGID = %d", sp, *sp);
267: #endif /*__DEBUG__ */
268:                 sp++;
269:
270:                 memcpy_l((void *)sp, &current->egid, 1);
271: #ifdef __DEBUG__
272:                 printk("\t\tAT_EGID = %d\n", *sp);
273: #endif /*__DEBUG__ */
274:                 sp++;
275:         }
276:
277:         memset_l((void *)sp, AT_NULL, 1);
278: #ifdef __DEBUG__
279:         printk("at 0x%08x -> AT_NULL = %d", sp, *sp);
280: #endif /*__DEBUG__ */
281:         sp++;
282:
283:         memset_l((void *)sp, NULL, 1);
284: #ifdef __DEBUG__
285:         printk("\t\tAT_NULL = %d\n", *sp);
286: #endif /*__DEBUG__ */
287:         sp++;
288:
289: #ifdef __DEBUG__
290:         for(n = 0; n < barg->argc; n++) {
291:                 printk("at 0x%08x -> argv[%d] = '%s'\n", current->argv[n], n, cu
rrent->argv[n]);
292:         }
293:         for(n = 0; n < barg->envc; n++) {
294:                 printk("at 0x%08x -> envp[%d] = '%s'\n", current->envp[n], n, cu
rrent->envp[n]);
295:         }
296: #endif /*__DEBUG__ */
297: }
298:
299: static int elf_load_interpreter(struct inode *ii)
300: {
301:         int n, errno;
302:         struct buffer *buf;
303:         struct elf32_hdr *elf32_h;
304:         struct elf32_phdr *elf32_ph, *last_ptload;
305:         __blk_t block;
306:         unsigned int start, end, length;
307:         unsigned int prot;
308:         char *data;
309:         char type;
310:
311:         if((block = bmap(ii, 0, FOR_READING)) < 0) {
312:                 return block;
313:         }
314:         if(!(buf = bread(ii->dev, block, ii->sb->s_blocksize))) {
315:                 return -EIO;
316:         }
317:
318:         /*
319:          * The contents of the buffer is copied and then freed immediately to
320:          * make sure that it won't conflict while zeroing the BSS fractional
321:          * page, in case that the same block is requested during the page fault.
322:          */
323:         if(!(data = (void *)kmalloc())) {
324:                 brelse(buf);
325:                 return -ENOMEM;
326:         }
327:         memcpy_b(data, buf->data, ii->sb->s_blocksize);
328:         brelse(buf);
329:
```

fs/elf.c

```
  330:          elf32_h = (struct elf32_hdr *)data;
  331:          if(check_elf(elf32_h)) {
  332:                  kfree((unsigned int)data);
  333:                  return -ELIBBAD;
  334:          }
  335:
  336:          last_ptload = NULL;
  337:          for(n = 0; n < elf32_h->e_phnum; n++) {
  338:                  elf32_ph = (struct elf32_phdr *)(data + elf32_h->e_phoff + (size
of(struct elf32_phdr) * n));
  339:                  if(elf32_ph->p_type == PT_LOAD) {
  340: #ifdef __DEBUG__
  341:                          printk("p_offset = 0x%08x\n", elf32_ph->p_offset);
  342:                          printk("p_vaddr  = 0x%08x\n", elf32_ph->p_vaddr);
  343:                          printk("p_paddr  = 0x%08x\n", elf32_ph->p_paddr);
  344:                          printk("p_filesz = 0x%08x\n", elf32_ph->p_filesz);
  345:                          printk("p_memsz  = 0x%08x\n\n", elf32_ph->p_memsz);
  346: #endif /*__DEBUG__ */
  347:                          start = (elf32_ph->p_vaddr & PAGE_MASK) + MMAP_START;
  348:                          length = (elf32_ph->p_vaddr & ~PAGE_MASK) + elf32_ph->p_
filesz;
  349:                          type = P_DATA;
  350:                          prot = 0;
  351:                          if(elf32_ph->p_flags & PF_R) {
  352:                                  prot = PROT_READ;
  353:                          }
  354:                          if(elf32_ph->p_flags & PF_W) {
  355:                                  prot |= PROT_WRITE;
  356:                          }
  357:                          if(elf32_ph->p_flags & PF_X) {
  358:                                  prot |= PROT_EXEC;
  359:                                  type = P_TEXT;
  360:                          }
  361:                          errno = do_mmap(ii, start, length, prot, MAP_PRIVATE | M
AP_FIXED, elf32_ph->p_offset & PAGE_MASK, type, O_RDONLY);
  362:                          if(errno < 0 && errno > -PAGE_SIZE) {
  363:                                  kfree((unsigned int)data);
  364:                                  send_sig(current, SIGSEGV);
  365:                                  return -ENOEXEC;
  366:                          }
  367:                          last_ptload = elf32_ph;
  368:                  }
  369:          }
  370:
  371:          if(!last_ptload) {
  372:                  printk("WARNING: 'last_ptload' is NULL!\n");
  373:          }
  374:          elf32_ph = last_ptload;
  375:
  376:          /* zero-fill the fractional page of the DATA section */
  377:          end = PAGE_ALIGN(elf32_ph->p_vaddr + elf32_ph->p_filesz) + MMAP_START;
  378:          start = (elf32_ph->p_vaddr + elf32_ph->p_filesz) + MMAP_START;
  379:          length = end - start;
  380:
  381:          /* this will generate a page fault which will load the page in */
  382:          memset_b((void *)start, NULL, length);
  383:
  384:          /* setup the BSS section */
  385:          start = (elf32_ph->p_vaddr + elf32_ph->p_filesz) + MMAP_START;
  386:          start = PAGE_ALIGN(start);
  387:          end = (elf32_ph->p_vaddr + elf32_ph->p_memsz) + MMAP_START;
  388:          end = PAGE_ALIGN(end);
  389:          length = end - start;
  390:          errno = do_mmap(NULL, start, length, PROT_READ | PROT_WRITE, MAP_PRIVATE
 | MAP_FIXED, 0, P_BSS, 0);
  391:          if(errno < 0 && errno > -PAGE_SIZE) {
  392:                  kfree((unsigned int)data);
```

fs/elf.c

```
  393:                 send_sig(current, SIGSEGV);
  394:                 return -ENOEXEC;
  395:         }
  396:         kfree((unsigned int)data);
  397:         return elf32_h->e_entry + MMAP_START;
  398: }
  399:
  400: int elf_load(struct inode *i, struct binargs *barg, struct sigcontext *sc, char
*data)
  401: {
  402:         int n, errno;
  403:         struct elf32_hdr *elf32_h;
  404:         struct elf32_phdr *elf32_ph, *last_ptload;
  405:         struct inode *ii;
  406:         unsigned int start, end, length;
  407:         unsigned int prot;
  408:         char *interpreter;
  409:         int at_base, phdr_addr;
  410:         char type;
  411:         unsigned int ae_ptr_len, ae_str_len;
  412:         unsigned int sp, str;
  413:
  414:         elf32_h = (struct elf32_hdr *)data;
  415:         if(check_elf(elf32_h)) {
  416:                 if(current->pid == INIT) {
  417:                         PANIC("%s has an unrecognized binary format.\n", INIT_PR
OGRAM);
  418:                 }
  419:                 return -ENOEXEC;
  420:         }
  421:
  422:         /* check if an interpreter is required */
  423:         interpreter = NULL;
  424:         ii = NULL;
  425:         phdr_addr = at_base = 0;
  426:         for(n = 0; n < elf32_h->e_phnum; n++) {
  427:                 elf32_ph = (struct elf32_phdr *)(data + elf32_h->e_phoff + (size
of(struct elf32_phdr) * n));
  428:                 if(elf32_ph->p_type == PT_INTERP) {
  429:                         at_base = MMAP_START;
  430:                         interpreter = data + elf32_ph->p_offset;
  431:                         if(namei(interpreter, &ii, NULL, FOLLOW_LINKS)) {
  432:                                 printk("%s(): can't find interpreter '%s'.\n", _
_FUNCTION__, interpreter);
  433:                                 send_sig(current, SIGSEGV);
  434:                                 return -ELIBACC;
  435:                         }
  436: #ifdef __DEBUG__
  437:                         printk("p_offset = 0x%08x\n", elf32_ph->p_offset);
  438:                         printk("p_vaddr  = 0x%08x\n", elf32_ph->p_vaddr);
  439:                         printk("p_paddr  = 0x%08x\n", elf32_ph->p_paddr);
  440:                         printk("p_filesz = 0x%08x\n", elf32_ph->p_filesz);
  441:                         printk("p_memsz  = 0x%08x\n", elf32_ph->p_memsz);
  442:                         printk("using interpreter '%s'\n", interpreter);
  443: #endif /*__DEBUG__ */
  444:                 }
  445:         }
  446:
  447:         /*
  448:          * calculate the final size of 'ae_ptr_len' based on:
  449:          *  - argc = 4 bytes (unsigned int)
  450:          *  - barg.argc = (num. of pointers to strings + 1 NULL) x 4 bytes (unsi
gned int)
  451:          *  - barg.envc = (num. of pointers to strings + 1 NULL) x 4 bytes (unsi
gned int)
  452:          */
  453:         ae_ptr_len = (1 + (barg->argc + 1) + (barg->envc + 1)) * sizeof(unsigned
```

fs/elf.c


```
 int);
  454:          ae_str_len = barg->argv_len + barg->envp_len;
  455:          if(ae_ptr_len + ae_str_len > (ARG_MAX * PAGE_SIZE)) {
  456:                  printk("WARNING: %s(): argument list (%d) exceeds ARG_MAX (%d)!\
n", __FUNCTION__, ae_ptr_len + ae_str_len, ARG_MAX * PAGE_SIZE);
  457:                  return -E2BIG;
  458:          }
  459:
  460: #ifdef __DEBUG__
  461:          printk("argc=%d (argv_len=%d) envc=%d (envp_len=%d)  ae_ptr_len=%d ae_st
r_len=%d\n", barg->argc, barg->argv_len, barg->envc, barg->envp_len, ae_ptr_len, ae_str
_len);
  462: #endif /*__DEBUG__ */
  463:
  464:
  465:          /* point of no return */
  466:
  467:          release_binary();
  468:          current->rss = 0;
  469:
  470:          current->entry_address = elf32_h->e_entry;
  471:          if(interpreter) {
  472:                  errno = elf_load_interpreter(ii);
  473:                  if(errno < 0) {
  474:                          printk("%s(): unable to load the interpreter '%s'.\n", _
_FUNCTION__, interpreter);
  475:                          iput(ii);
  476:                          send_sig(current, SIGKILL);
  477:                          return errno;
  478:                  }
  479:                  current->entry_address = errno;
  480:                  iput(ii);
  481:          }
  482:
  483:          elf32_ph = last_ptload = NULL;
  484:          for(n = 0; n < elf32_h->e_phnum; n++) {
  485:                  elf32_ph = (struct elf32_phdr *)(data + elf32_h->e_phoff + (size
of(struct elf32_phdr) * n));
  486:                  if(elf32_ph->p_type == PT_PHDR) {
  487:                          phdr_addr = elf32_ph->p_vaddr;
  488:                  }
  489:                  if(elf32_ph->p_type == PT_LOAD) {
  490:                          start = elf32_ph->p_vaddr & PAGE_MASK;
  491:                          length = (elf32_ph->p_vaddr & ~PAGE_MASK) + elf32_ph->p_
filesz;
  492:                          type = P_DATA;
  493:                          prot = 0;
  494:                          if(elf32_ph->p_flags & PF_R) {
  495:                                  prot = PROT_READ;
  496:                          }
  497:                          if(elf32_ph->p_flags & PF_W) {
  498:                                  prot |= PROT_WRITE;
  499:                          }
  500:                          if(elf32_ph->p_flags & PF_X) {
  501:                                  prot |= PROT_EXEC;
  502:                                  type = P_TEXT;
  503:                          }
  504:                          errno = do_mmap(i, start, length, prot, MAP_PRIVATE | MA
P_FIXED, elf32_ph->p_offset & PAGE_MASK, type, O_RDONLY);
  505:                          if(errno < 0 && errno > -PAGE_SIZE) {
  506:                                  send_sig(current, SIGSEGV);
  507:                                  return -ENOEXEC;
  508:                          }
  509:                          last_ptload = elf32_ph;
  510:                  }
  511:          }
  512:
```

fs/elf.c

```
  513:          elf32_ph = last_ptload;
  514:
  515:          /* zero-fill the fractional page of the DATA section */
  516:          end = PAGE_ALIGN(elf32_ph->p_vaddr + elf32_ph->p_filesz);
  517:          start = elf32_ph->p_vaddr + elf32_ph->p_filesz;
  518:          length = end - start;
  519:
  520:          /* this will generate a page fault which will load the page in */
  521:          memset_b((void *)start, NULL, length);
  522:
  523:          /* setup the BSS section */
  524:          start = elf32_ph->p_vaddr + elf32_ph->p_filesz;
  525:          start = PAGE_ALIGN(start);
  526:          end = elf32_ph->p_vaddr + elf32_ph->p_memsz;
  527:          end = PAGE_ALIGN(end);
  528:          length = end - start;
  529:          errno = do_mmap(NULL, start, length, PROT_READ | PROT_WRITE, MAP_PRIVATE
 | MAP_FIXED, 0, P_BSS, 0);
  530:          if(errno < 0 && errno > -PAGE_SIZE) {
  531:                  send_sig(current, SIGSEGV);
  532:                  return -ENOEXEC;
  533:          }
  534:          current->brk_lower = start;
  535:
  536:          /* setup the HEAP section */
  537:          start = elf32_ph->p_vaddr + elf32_ph->p_memsz;
  538:          start = PAGE_ALIGN(start);
  539:          length = PAGE_SIZE;
  540:          errno = do_mmap(NULL, start, length, PROT_READ | PROT_WRITE, MAP_PRIVATE
 | MAP_FIXED, 0, P_HEAP, 0);
  541:          if(errno < 0 && errno > -PAGE_SIZE) {
  542:                  send_sig(current, SIGSEGV);
  543:                  return -ENOEXEC;
  544:          }
  545:          current->brk = start;
  546:
  547:          /* setup the STACK section */
  548:          sp = KERNEL_BASE_ADDR - 4;      /* formerly 0xBFFFFFFC */
  549:          sp -= ae_str_len;
  550:          str = sp;       /* this is the address of the first string (argv[0]) */
  551:          sp &= ~3;
  552:          sp -= at_base ? (AT_ITEMS * 2) * sizeof(unsigned int) : 2 * sizeof(unsig
ned int);
  553:          sp -= ae_ptr_len;
  554:          length = KERNEL_BASE_ADDR - (sp & PAGE_MASK);
  555:          errno = do_mmap(NULL, sp & PAGE_MASK, length, PROT_READ | PROT_WRITE | P
ROT_EXEC, MAP_PRIVATE | MAP_FIXED, 0, P_STACK, 0);
  556:          if(errno < 0 && errno > -PAGE_SIZE) {
  557:                  send_sig(current, SIGSEGV);
  558:                  return -ENOEXEC;
  559:          }
  560:
  561:          elf_create_stack(barg, (unsigned int *)sp, str, at_base, elf32_h, phdr_a
ddr);
  562:
  563:          /* set %esp to point to 'argc' */
  564:          sc->oldesp = sp;
  565:          sc->eflags = 0x202;     /* FIXME: linux 2.2 = 0x292 */
  566:          sc->eip = current->entry_address;
  567:          sc->err = 0;
  568:          sc->eax = 0;
  569:          sc->ecx = 0;
  570:          sc->edx = 0;
  571:          sc->ebx = 0;
  572:          sc->ebp = 0;
  573:          sc->esi = 0;
  574:          sc->edi = 0;
```

fs/elf.c


```
575:         return 0;
576: }
```

fs/fd.c


```
 1: /*
 2:  * fiwix/fs/fd.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/errno.h>
 9: #include <fiwix/types.h>
10: #include <fiwix/fs.h>
11: #include <fiwix/sleep.h>
12: #include <fiwix/stdio.h>
13: #include <fiwix/string.h>
14:
15: struct fd *fd_table;
16:
17: static struct resource fd_resource = { NULL, NULL };
18:
19: int get_new_fd(struct inode *i)
20: {
21:         unsigned int n;
22:
23:         lock_resource(&fd_resource);
24:
25:         for(n = 1; n < NR_OPENS; n++) {
26:                 if(fd_table[n].count == 0) {
27:                         memset_b(&fd_table[n], NULL, sizeof(struct fd));
28:                         fd_table[n].inode = i;
29:                         fd_table[n].count = 1;
30:                         unlock_resource(&fd_resource);
31:                         return n;
32:                 }
33:         }
34:
35:         unlock_resource(&fd_resource);
36:
37:         return -ENFILE;
38: }
39:
40: void release_fd(unsigned int fd)
41: {
42:         lock_resource(&fd_resource);
43:         fd_table[fd].count = 0;
44:         unlock_resource(&fd_resource);
45: }
46:
47: void fd_init(void)
48: {
49:         memset_b(fd_table, NULL, fd_table_size);
50: }
```

fs/filesystems.c


```
    1: /*
    2:  * fiwix/fs/filesystems.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/errno.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/filesystems.h>
   12: #include <fiwix/fs_pipe.h>
   13: #include <fiwix/fs_iso9660.h>
   14: #include <fiwix/fs_proc.h>
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/string.h>
   17:
   18: int register_filesystem(const char *name, struct fs_operations *fsop)
   19: {
   20:         int n;
   21:         __dev_t dev;
   22:
   23:         for(n = 0; n < NR_FILESYSTEMS; n++) {
   24:                 if(filesystems_table[n].name) {
   25:                         if(strcmp(filesystems_table[n].name, name) == 0) {
   26:                                 printk("WARNING: %s(): filesystem '%s' already r
egistered!\n", __FUNCTION__, name);
   27:                                 return 1;
   28:                         }
   29:                 }
   30:                 if(!filesystems_table[n].name) {
   31:                         filesystems_table[n].name = name;
   32:                         filesystems_table[n].fsop = fsop;
   33:                         if((fsop->flags & FSOP_KERN_MOUNT)) {
   34:                                 dev = fsop->fsdev;
   35:                                 return kern_mount(dev, &filesystems_table[n]);
   36:                         }
   37:                         return 0;
   38:                 }
   39:         }
   40:         printk("WARNING: %s(): filesystems table is full!\n", __FUNCTION__);
   41:         return 1;
   42: }
   43:
   44: struct filesystems * get_filesystem(const char *name)
   45: {
   46:         int n;
   47:
   48:         if(!name) {
   49:                 return NULL;
   50:         }
   51:         for(n = 0; n < NR_FILESYSTEMS; n++) {
   52:                 if(!filesystems_table[n].name) {
   53:                         continue;
   54:                 }
   55:                 if(strcmp(filesystems_table[n].name, name) == 0) {
   56:                         return &filesystems_table[n];
   57:                 }
   58:         }
   59:         return NULL;
   60: }
   61:
   62: void fs_init(void)
   63: {
   64:         memset_b(filesystems_table, NULL, sizeof(filesystems_table));
   65:
   66:         if(minix_init()) {
```

fs/filesystems.c


```
   67:                 printk("%s(): unable to register 'minix' filesystem.\n", __FUNCT
ION__);
   68:         }
   69:         if(ext2_init()) {
   70:                 printk("%s(): unable to register 'ext2' filesystem.\n", __FUNCTI
ON__);
   71:         }
   72:         if(pipefs_init()) {
   73:                 printk("%s(): unable to register 'pipefs' filesystem.\n", __FUNC
TION__);
   74:         }
   75:         if(iso9660_init()) {
   76:                 printk("%s(): unable to register 'iso9660' filesystem.\n", __FUN
CTION__);
   77:         }
   78:         if(procfs_init()) {
   79:                 printk("%s(): unable to register 'procfs' filesystem.\n", __FUNC
TION__);
   80:         }
   81: }
```

fs/inode.c


```
    1: /*
    2:  * fiwix/fs/inode.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: /*
    9:  * inode.c implements a cache with a free list as a doubly circular linked
   10:  * list and a chained hash table with doubly linked lists.
   11:  *
   12:  * hash table
   13:  * +--------+  +--------------+  +--------------+  +--------------+
   14:  * | index  |  |prev|data|next|  |prev|data|next|  |prev|data|next|
   15:  * |   0   --> | /  |    | --->  <--- |    | --->  <--- |    |  / |
   16:  * +--------+  +--------------+  +--------------+  +--------------+
   17:  * +--------+  +--------------+  +--------------+  +--------------+
   18:  * | index  |  |prev|data|next|  |prev|data|next|  |prev|data|next|
   19:  * |   1   --> | /  |    | --->  <--- |    | --->  <--- |    |  / |
   20:  * +--------+  +--------------+  +--------------+  +--------------+
   21:  *              (inode)           (inode)           (inode)
   22:  *    ...
   23:  */
   24:
   25: #include <fiwix/asm.h>
   26: #include <fiwix/sleep.h>
   27: #include <fiwix/sched.h>
   28: #include <fiwix/fs.h>
   29: #include <fiwix/filesystems.h>
   30: #include <fiwix/stat.h>
   31: #include <fiwix/errno.h>
   32: #include <fiwix/mm.h>
   33: #include <fiwix/stdio.h>
   34: #include <fiwix/string.h>
   35:
   36: #define INODE_HASH(dev, inode)  (((__dev_t)(dev) ^ (__ino_t)(inode)) % (NR_INO_H
ASH))
   37: #define NR_INODES       (inode_table_size / sizeof(struct inode))
   38: #define NR_INO_HASH     (inode_hash_table_size / sizeof(unsigned int))
   39:
   40: struct inode *inode_table;              /* inode pool */
   41: struct inode *inode_head;               /* inode pool head */
   42: struct inode **inode_hash_table;
   43:
   44: int inodes_on_free_list = 0;
   45: static struct resource sync_resource = { NULL, NULL };
   46:
   47: static void insert_to_hash(struct inode *i)
   48: {
   49:         struct inode **h;
   50:         int n;
   51:
   52:         n = INODE_HASH(i->dev, i->inode);
   53:         h = &inode_hash_table[n];
   54:
   55:         if(!*h) {
   56:                 *h = i;
   57:                 (*h)->prev_hash = (*h)->next_hash = NULL;
   58:         } else {
   59:                 i->prev_hash = NULL;
   60:                 i->next_hash = *h;
   61:                 (*h)->prev_hash = i;
   62:                 *h = i;
   63:         }
   64: }
   65:
   66: static void remove_from_hash(struct inode *i)
```

fs/inode.c


```
 67: {
 68:         struct inode **h;
 69:         int n;
 70:
 71:         n = INODE_HASH(i->dev, i->inode);
 72:         h = &inode_hash_table[n];
 73:
 74:         while(*h) {
 75:                 if(*h == i) {
 76:                         if((*h)->next_hash) {
 77:                                 (*h)->next_hash->prev_hash = (*h)->prev_hash;
 78:                         }
 79:                         if((*h)->prev_hash) {
 80:                                 (*h)->prev_hash->next_hash = (*h)->next_hash;
 81:                         }
 82:                         if(h == &inode_hash_table[n]) {
 83:                                 *h = (*h)->next_hash;
 84:                         }
 85:                         break;
 86:                 }
 87:                 h = &(*h)->next_hash;
 88:         }
 89: }
 90:
 91: static void remove_from_free_list(struct inode *i)
 92: {
 93:         i->prev_free->next_free = i->next_free;
 94:         i->next_free->prev_free = i->prev_free;
 95:         inodes_on_free_list--;
 96:         if(i == inode_head) {
 97:                 inode_head = i->next_free;
 98:         }
 99: }
100:
101: static void inode_wait(struct inode *i)
102: {
103:         unsigned long int flags;
104:
105:         for(;;) {
106:                 SAVE_FLAGS(flags); CLI();
107:                 if(i->locked) {
108:                         RESTORE_FLAGS(flags);
109:                         sleep(i, PROC_UNINTERRUPTIBLE);
110:                 } else {
111:                         break;
112:                 }
113:         }
114:         RESTORE_FLAGS(flags);
115: }
116:
117: static struct inode * get_free_inode(void)
118: {
119:         struct inode *i;
120:
121:         /* no more inodes on free list */
122:         if(inode_head == inode_head->next_free) {
123:                 return NULL;
124:         }
125:
126:         i = inode_head;
127:         inode_head->next_free->prev_free = inode_head->prev_free;
128:         inode_head->prev_free->next_free = inode_head->next_free;
129:         inode_head = inode_head->next_free;
130:
131:         return i;
132: }
133:
```

fs/inode.c


```
134: static int read_inode(struct inode *i)
135: {
136:         int errno;
137:
138:         inode_lock(i);
139:         if(i->sb && i->sb->fsop && i->sb->fsop->read_inode) {
140:                 errno = i->sb->fsop->read_inode(i);
141:                 inode_unlock(i);
142:                 return errno;
143:         }
144:         inode_unlock(i);
145:         return -EINVAL;
146: }
147:
148: static int write_inode(struct inode *i)
149: {
150:         int errno;
151:
152:         errno = 1;
153:
154:         inode_lock(i);
155:         if(i->sb && i->sb->fsop && i->sb->fsop->write_inode) {
156:                 errno = i->sb->fsop->write_inode(i);
157:         } else {
158:                 /* i.e. PIPE_DEV inodes can't be flushed on disk */
159:                 i->dirty = 0;
160:                 errno = 0;
161:         }
162:         inode_unlock(i);
163:
164:         return errno;
165: }
166:
167: static struct inode * search_inode_hash(__dev_t dev, __ino_t inode)
168: {
169:         struct inode *i;
170:         int n;
171:
172:         n = INODE_HASH(dev, inode);
173:         i = inode_hash_table[n];
174:
175:         while(i) {
176:                 if(i->dev == dev && i->inode == inode) {
177:                         return i;
178:                 }
179:                 i = i->next_hash;
180:         }
181:
182:         return NULL;
183: }
184:
185: static struct inode * get_blank_inode(void)
186: {
187:         unsigned long int flags;
188:         struct inode *i;
189:
190:         SAVE_FLAGS(flags); CLI();
191:
192:         if((i = get_free_inode())) {
193:                 remove_from_free_list(i);
194:                 remove_from_hash(i);
195:                 i->i_mode = 0;
196:                 i->i_uid = 0;
197:                 i->i_size = 0;
198:                 i->i_atime = 0;
199:                 i->i_ctime = 0;
200:                 i->i_mtime = 0;
```

fs/inode.c


```
  201:                 i->i_gid = 0;
  202:                 i->i_nlink = 0;
  203:                 i->i_blocks = 0;
  204:                 i->i_flags = 0;
  205:                 i->locked = 0;
  206:                 i->dirty = 0;
  207:                 i->mount_point = NULL;
  208:                 i->dev = 0;
  209:                 i->inode = 0;
  210:                 i->count = 0;
  211:                 i->rdev = 0;
  212:                 i->fsop = NULL;
  213:                 i->sb = NULL;
  214:                 memset_b(&i->u, NULL, sizeof(i->u));
  215:         }
  216:         RESTORE_FLAGS(flags);
  217:         return i;
  218: }
  219:
  220: void inode_lock(struct inode *i)
  221: {
  222:         unsigned long int flags;
  223:
  224:         for(;;) {
  225:                 SAVE_FLAGS(flags); CLI();
  226:                 if(i->locked) {
  227:                         RESTORE_FLAGS(flags);
  228:                         sleep(i, PROC_UNINTERRUPTIBLE);
  229:                 } else {
  230:                         break;
  231:                 }
  232:         }
  233:         i->locked = 1;
  234:         RESTORE_FLAGS(flags);
  235: }
  236:
  237: void inode_unlock(struct inode *i)
  238: {
  239:         unsigned long int flags;
  240:
  241:         SAVE_FLAGS(flags); CLI();
  242:         i->locked = 0;
  243:         wakeup(i);
  244:         RESTORE_FLAGS(flags);
  245: }
  246:
  247: struct inode * ialloc(struct superblock *sb, int mode)
  248: {
  249:         int errno;
  250:         struct inode *i;
  251:
  252:         if((i = get_blank_inode())) {
  253:                 i->sb = sb;
  254:                 i->rdev = sb->dev;
  255:                 if(i->sb && i->sb->fsop && i->sb->fsop->ialloc) {
  256:                         errno = i->sb->fsop->ialloc(i, mode);
  257:                 } else {
  258:                         printk("WARNING: this filesystem does not have the iallo
c() method!\n");
  259:                         i->count = 1;
  260:                         i->sb = NULL;
  261:                         iput(i);
  262:                         return NULL;
  263:                 }
  264:                 if(errno) {
  265:                         i->count = 1;
  266:                         i->sb = NULL;
```

fs/inode.c


```
  267:                          iput(i);
  268:                          return NULL;
  269:                  }
  270:                  i->dev = sb->dev;
  271:                  insert_to_hash(i);
  272:                  return i;
  273:          }
  274:          printk("WARNING: %s(): no more inodes on free list!\n", __FUNCTION__);
  275:          return NULL;
  276: }
  277:
  278: struct inode * iget(struct superblock *sb, __ino_t inode)
  279: {
  280:          unsigned long int flags;
  281:          struct inode *i;
  282:
  283:          if(!inode) {
  284:                  return NULL;
  285:          }
  286:
  287:          for(;;) {
  288:                  if((i = search_inode_hash(sb->dev, inode))) {
  289:                          inode_wait(i);
  290:                          SAVE_FLAGS(flags); CLI();
  291:
  292:                          /* update superblock pointer from mount_table */
  293:                          i->sb = sb;
  294:
  295:                          if(i->mount_point) {
  296:                                  i = i->mount_point;
  297:                          }
  298:                          /* FIXME: i->locked = 1; ? */
  299:                          if(++i->count == 1) {
  300:                                  remove_from_free_list(i);
  301:                          }
  302:                          RESTORE_FLAGS(flags);
  303:                          return i;
  304:                  }
  305:
  306:                  if(!(i = get_blank_inode())) {
  307:                          printk("WARNING: %s(): no more inodes on free list! (%d)
.\n", __FUNCTION__, inodes_on_free_list);
  308:                          return NULL;
  309:                  }
  310:
  311:                  SAVE_FLAGS(flags); CLI();
  312:                  i->dev = i->rdev = sb->dev;
  313:                  i->inode = inode;
  314:                  i->sb = sb;
  315:                  i->count = 1;
  316:                  RESTORE_FLAGS(flags);
  317:                  if(read_inode(i)) {
  318:                          iput(i);
  319:                          return NULL;
  320:                  }
  321:                  insert_to_hash(i);
  322:                  /* FIXME: i->locked = 1; ? */
  323:                  return i;
  324:          }
  325: }
  326:
  327: int bmap(struct inode *i, __off_t offset, int mode)
  328: {
  329:          if(i->fsop && i->fsop->bmap) {
  330:                  return i->fsop->bmap(i, offset, mode);
  331:          }
  332:          return -EPERM;
```

fs/inode.c


```
  333: }
  334:
  335: int check_fs_busy(__dev_t dev, struct inode *root)
  336: {
  337:         struct inode *i;
  338:         unsigned int n;
  339:
  340:         i = &inode_table[0];
  341:         for(n = 0; n < NR_INODES; n++, i = &inode_table[n]) {
  342:                 if(i->dev == dev && i->count) {
  343:                         if(i == root && i->count == 1) {
  344:                                 continue;
  345:                         }
  346:                         /* FIXME: to be removed */
  347:                         printk("WARNING: root %d with count %d (on dev %d,%d)\n"
, root->inode, root->count, MAJOR(i->dev), MINOR(i->dev));
  348:                         printk("WARNING: inode %d with count %d (on dev %d,%d)\n
", i->inode, i->count, MAJOR(i->dev), MINOR(i->dev));
  349:                         return 1;
  350:                 }
  351:         }
  352:         return 0;
  353: }
  354:
  355: void iput(struct inode *i)
  356: {
  357:         unsigned long int flags;
  358:
  359:         /* this solves the problem with rmdir('/') and iput(dir) which is NULL *
/
  360:         if(!i) {
  361:                 return;
  362:         }
  363:
  364:         if(!i->count) {
  365:                 printk("WARNING: %s(): trying to free an already freed inode (%d
)!\n", __FUNCTION__, i->inode);
  366:                 return;
  367:         }
  368:
  369:         SAVE_FLAGS(flags); CLI();
  370:
  371:         if(--i->count == 0) {
  372:                 if(!i->i_nlink) {
  373:                         if(i->sb && i->sb->fsop && i->sb->fsop->ifree) {
  374:                                 inode_lock(i);
  375:                                 i->sb->fsop->ifree(i);
  376:                                 remove_from_hash(i);
  377:                                 inode_unlock(i);
  378:                         }
  379:                 }
  380:                 if(i->dirty) {
  381:                         if(write_inode(i)) {
  382:                                 printk("WARNING: %s(): can't write inode %d (%d,
%d), will remain as dirty.\n", __FUNCTION__, i->inode, MAJOR(i->dev), MINOR(i->dev));
  383:                                 RESTORE_FLAGS(flags);
  384:                                 return;
  385:                         }
  386:                 }
  387:                 if(!inode_head) {
  388:                         i->prev_free = i->next_free = i;
  389:                         inode_head = i;
  390:                 } else {
  391:                         i->next_free = inode_head;
  392:                         i->prev_free = inode_head->prev_free;
  393:                         inode_head->prev_free->next_free = i;
  394:                         inode_head->prev_free = i;
```

fs/inode.c


```
  395:                 }
  396:                 inodes_on_free_list++;
  397:         }
  398:
  399:         RESTORE_FLAGS(flags);
  400: }
  401:
  402: void sync_inodes(__dev_t dev)
  403: {
  404:         struct inode *i;
  405:         int n;
  406:
  407:         i = &inode_table[0];
  408:
  409:         lock_resource(&sync_resource);
  410:         for(n = 0; n < NR_INODES; n++) {
  411:                 if(i->dirty) {
  412:                         if(!dev || i->dev == dev) {
  413:                                 inode_wait(i);
  414:                                 if(write_inode(i)) {
  415:                                         printk("WARNING: %s(): can't write inode
 %d (%d,%d), will remain as dirty.\n", __FUNCTION__, i->inode, MAJOR(i->dev), MINOR(i->
dev));
  416:                                 }
  417:                         }
  418:                 }
  419:                 i++;
  420:         }
  421:         unlock_resource(&sync_resource);
  422:         return;
  423: }
  424:
  425: void invalidate_inodes(__dev_t dev)
  426: {
  427:         unsigned long int flags;
  428:         unsigned int n;
  429:         struct inode *i;
  430:
  431:         i = &inode_table[0];
  432:         SAVE_FLAGS(flags); CLI();
  433:
  434:         for(n = 0; n < NR_INODES; n++) {
  435:                 if(i->dev == dev) {
  436:                         inode_wait(i);
  437:                         remove_from_hash(i);
  438:                         i->locked = 0;
  439:                         wakeup(&inode_wait);
  440:                 }
  441:                 i++;
  442:         }
  443:
  444:         RESTORE_FLAGS(flags);
  445: }
  446:
  447: void inode_init(void)
  448: {
  449:         struct inode *i;
  450:         unsigned int n;
  451:
  452:         memset_b(inode_table, NULL, inode_table_size);
  453:         memset_b(inode_hash_table, NULL, inode_hash_table_size);
  454:         for(n = 0; n < NR_INODES; n++) {
  455:                 i = &inode_table[n];
  456:                 i->count = 1;
  457:                 iput(i);
  458:         }
  459: }
```

fs/locks.c


```
    1: /*
    2:  * fiwix/fs/locks.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/errno.h>
    9: #include <fiwix/types.h>
   10: #include <fiwix/locks.h>
   11: #include <fiwix/fs.h>
   12: #include <fiwix/sleep.h>
   13: #include <fiwix/sched.h>
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/string.h>
   16:
   17: static struct resource flock_resource = { NULL, NULL };
   18:
   19: static struct flock_file * get_new_flock(struct inode *i)
   20: {
   21:         int n;
   22:         struct flock_file *ff;
   23:
   24:         lock_resource(&flock_resource);
   25:
   26:         for(n = 0; n < NR_FLOCKS; n++) {
   27:                 ff = &flock_file_table[n];
   28:                 if(!ff->inode) {
   29:                         ff->inode = i;  /* mark it as busy */
   30:                         unlock_resource(&flock_resource);
   31:                         return ff;
   32:                 }
   33:         }
   34:
   35:         printk("WARNING: %s(): no more free slots in flock file table.\n");
   36:         unlock_resource(&flock_resource);
   37:         return NULL;
   38: }
   39:
   40: static void release_flock(struct flock_file *ff)
   41: {
   42:         memset_b(ff, 0, sizeof(struct flock_file));
   43: }
   44:
   45: static struct flock_file * get_flock_file(struct inode *i, int op, struct proc *
p)
   46: {
   47:         int n;
   48:         struct flock_file *ff;
   49:
   50:         lock_resource(&flock_resource);
   51:
   52:         ff = NULL;
   53:         for(n = 0; n < NR_FLOCKS; n++) {
   54:                 ff = &flock_file_table[n];
   55:                 if(ff->inode != i) {
   56:                         continue;
   57:                 }
   58:                 if(p && p != ff->proc) {
   59:                         continue;
   60:                 }
   61:                 break;
   62:         }
   63:         unlock_resource(&flock_resource);
   64:         return ff;
   65: }
   66:
```

fs/locks.c


```
 67: int posix_lock(int ufd, int cmd, struct flock *fl)
 68: {
 69:         int n;
 70:         struct flock_file *ff;
 71:         struct inode *i;
 72:         unsigned char type;
 73:
 74:         lock_resource(&flock_resource);
 75:         i = fd_table[current->fd[ufd]].inode;
 76:         for(n = 0; n < NR_FLOCKS; n++) {
 77:                 ff = &flock_file_table[n];
 78:                 if(ff->inode != i) {
 79:                         continue;
 80:                 }
 81:                 break;
 82:         }
 83:         unlock_resource(&flock_resource);
 84:         if(cmd == F_GETLK) {
 85:                 if(ff->inode == i) {
 86:                         fl->l_type = ff->type & LOCK_SH ? F_RDLCK : F_WRLCK;
 87:                         fl->l_whence = SEEK_SET;
 88:                         fl->l_start = 0;
 89:                         fl->l_len = 0;
 90:                         fl->l_pid = ff->proc->pid;
 91:                 } else {
 92:                         fl->l_type = F_UNLCK;
 93:                 }
 94:         }
 95:
 96:         switch(fl->l_type) {
 97:                 case F_RDLCK:
 98:                         type = LOCK_SH;
 99:                         break;
100:                 case F_WRLCK:
101:                         type = LOCK_EX;
102:                         break;
103:                 case F_UNLCK:
104:                         type = LOCK_UN;
105:                         break;
106:                 default:
107:                         return -EINVAL;
108:         }
109:         if(cmd == F_SETLK) {
110:                 return flock_inode(i, type);
111:         }
112:         if(cmd == F_SETLKW) {
113:                 return flock_inode(i, type | LOCK_NB);
114:         }
115:         return 0;
116: }
117:
118: void flock_release_inode(struct inode *i)
119: {
120:         int n;
121:         struct flock_file *ff;
122:
123:         lock_resource(&flock_resource);
124:         for(n = 0; n < NR_FLOCKS; n++) {
125:                 ff = &flock_file_table[n];
126:                 if(ff->inode != i) {
127:                         continue;
128:                 }
129:                 if(ff->proc != current) {
130:                         continue;
131:                 }
132:                 wakeup(ff);
133:                 release_flock(ff);
```

fs/locks.c

```
134:         }
135:         unlock_resource(&flock_resource);
136: }
137:
138: int flock_inode(struct inode *i, int op)
139: {
140:         int n;
141:         struct flock_file *ff, *new;
142:
143:         if(op & LOCK_UN) {
144:                 if((ff = get_flock_file(i, op, current))) {
145:                         wakeup(ff);
146:                         release_flock(ff);
147:                 }
148:                 return 0;
149:         }
150:
151: loop:
152:         lock_resource(&flock_resource);
153:         new = NULL;
154:         for(n = 0; n < NR_FLOCKS; n++) {
155:                 ff = &flock_file_table[n];
156:                 if(ff->inode != i) {
157:                         continue;
158:                 }
159:                 if(op & LOCK_SH) {
160:                         if(ff->type & LOCK_EX) {
161:                                 if(ff->proc == current) {
162:                                         new = ff;
163:                                         wakeup(ff);
164:                                         break;
165:                                 }
166:                                 unlock_resource(&flock_resource);
167:                                 if(op & LOCK_NB) {
168:                                         return -EWOULDBLOCK;
169:                                 }
170:                                 if(sleep(ff, PROC_INTERRUPTIBLE)) {
171:                                         return -EINTR;
172:                                 }
173:                                 goto loop;
174:                         }
175:                 }
176:                 if(op & LOCK_EX) {
177:                         if(ff->proc == current) {
178:                                 new = ff;
179:                                 continue;
180:                         }
181:                         unlock_resource(&flock_resource);
182:                         if(op & LOCK_NB) {
183:                                 return -EWOULDBLOCK;
184:                         }
185:                         if(sleep(ff, PROC_INTERRUPTIBLE)) {
186:                                 return -EINTR;
187:                         }
188:                         goto loop;
189:                 }
190:         }
191:         unlock_resource(&flock_resource);
192:
193:         if(!new) {
194:                 if(!(new = get_new_flock(i))) {
195:                         return -ENOLCK;
196:                 }
197:         }
198:         new->inode = i;
199:         new->type = op;
200:         new->proc = current;
```

fs/locks.c


```
201:
202:         return 0;
203: }
204:
205: void flock_init(void)
206: {
207:         memset_b(flock_file_table, NULL, sizeof(flock_file_table));
208: }
```

fs/Makefile


```
  1: # fiwix/fs/Makefile
  2: #
  3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
  4: # Distributed under the terms of the Fiwix License.
  5: #
  6:
  7: .S.o:
  8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
  9: .c.o:
 10:         $(CC) $(CFLAGS) -c -o $@ $<
 11:
 12: FSDIRS = minix ext2 pipefs iso9660 procfs
 13: FILESYSTEMS = minix/minix.o ext2/ext2.o pipefs/pipefs.o iso9660/iso9660.o \
 14:         procfs/procfs.o
 15: OBJS = filesystems.o devices.o buffer.o fd.o locks.o super.o inode.o \
 16:         namei.o elf.o script.o
 17:
 18: fs:     $(OBJS)
 19:         @for n in $(FSDIRS) ; do (cd $$n ; $(MAKE)) ; done
 20:         $(LD) $(LDFLAGS) -r $(FILESYSTEMS) $(OBJS) -o fs.o
 21:
 22: clean:
 23:         @for n in $(FSDIRS) ; do (cd $$n ; $(MAKE) clean) ; done
 24:         rm -f *.o
 25:
```

fs/namei.c


```
    1: /*
    2:  * fiwix/fs/namei.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/sleep.h>
   10: #include <fiwix/sched.h>
   11: #include <fiwix/fs.h>
   12: #include <fiwix/filesystems.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/mman.h>
   16: #include <fiwix/errno.h>
   17: #include <fiwix/stdio.h>
   18: #include <fiwix/string.h>
   19:
   20: static int namei_lookup(char *name, struct inode *dir, struct inode **i_res)
   21: {
   22:         if(dir->fsop && dir->fsop->lookup) {
   23:                 return dir->fsop->lookup(name, dir, i_res);
   24:         }
   25:         return -EACCES;
   26: }
   27:
   28: static int do_namei(char *path, struct inode *dir, struct inode **i_res, struct
inode **d_res, int follow_links)
   29: {
   30:         char *name, *ptr_name;
   31:         struct inode *i;
   32:         struct superblock *sb;
   33:
   34:         int errno;
   35:
   36:         *i_res = dir;
   37:         for(;;) {
   38:                 while(*path == '/') {
   39:                         path++;
   40:                 }
   41:                 if(*path == NULL) {
   42:                         return 0;
   43:                 }
   44:
   45:                 /* extracts the next component of the path */
   46:                 if(!(name = (char *)kmalloc())) {
   47:                         return -ENOMEM;
   48:                 }
   49:                 ptr_name = name;
   50:                 while(*path != NULL && *path != '/') {
   51:                         if(ptr_name > name + NAME_MAX - 1) {
   52:                                 break;
   53:                         }
   54:                         *ptr_name++ = *path++;
   55:                 }
   56:                 *ptr_name = NULL;
   57:
   58:                 /*
   59:                  * If the inode is the root of a file system, then return the
   60:                  * inode on which the file system was mounted.
   61:                  */
   62:                 if(name[0] == '.' && name[1] == '.' && name[2] == NULL) {
   63:                         if(dir == dir->sb->root) {
   64:                                 sb = dir->sb;
   65:                                 iput(dir);
   66:                                 dir = sb->dir;
```

fs/namei.c

```
    67:                                     dir->count++;
    68:                             }
    69:                     }
    70:
    71:                     if((errno = check_permission(TO_EXEC, dir))) {
    72:                             break;
    73:                     }
    74:
    75:                     if((errno = namei_lookup(name, dir, &i))) {
    76:                             break;
    77:                     }
    78:
    79:                     kfree((unsigned int)name);
    80:                     if(*path == '/') {
    81:                             if(!S_ISDIR(i->i_mode) && !S_ISLNK(i->i_mode)) {
    82:                                     iput(dir);
    83:                                     iput(i);
    84:                                     return -ENOTDIR;
    85:                             }
    86:                             if(S_ISLNK(i->i_mode)) {
    87:                                     if(i->fsop && i->fsop->followlink) {
    88:                                             if((errno = i->fsop->followlink(dir, i,
&i))) {
    89:                                                     iput(dir);
    90:                                                     return errno;
    91:                                             }
    92:                                     }
    93:                             }
    94:                     } else {
    95:                             if((i->fsop && i->fsop->followlink) && follow_links) {
    96:                                     if((errno = i->fsop->followlink(dir, i, &i))) {
    97:                                             iput(dir);
    98:                                             return errno;
    99:                                     }
   100:                             }
   101:                     }
   102:
   103:                     if(d_res) {
   104:                             if(*d_res) {
   105:                                     iput(*d_res);
   106:                             }
   107:                             *d_res = dir;
   108:                     } else {
   109:                             iput(dir);
   110:                     }
   111:                     dir = i;
   112:                     *i_res = i;
   113:             }
   114:
   115:             kfree((unsigned int)name);
   116:             if(d_res) {
   117:                     if(*d_res) {
   118:                             iput(*d_res);
   119:                     }
   120:                     /*
   121:                      * If that was the last component of the path,
   122:                      * then return the directory.
   123:                      */
   124:                     if(*path == NULL) {
   125:                             *d_res = dir;
   126:                             dir->count++;
   127:                     } else {
   128:                             /* that's an non-existent directory */
   129:                             *d_res = NULL;
   130:                             errno = -ENOTDIR;
   131:                     }
   132:                     iput(dir);
```

fs/namei.c


```
  133:                  *i_res = NULL;
  134:          } else {
  135:                  iput(dir);
  136:          }
  137:
  138:          return errno;
  139: }
  140:
  141: int parse_namei(char *path, struct inode *base_dir, struct inode **i_res, struct
 inode **d_res, int follow_links)
  142: {
  143:          struct inode *dir;
  144:          int errno;
  145:
  146:          if(!path) {
  147:                  return -EFAULT;
  148:          }
  149:          if(*path == NULL) {
  150:                  return -ENOENT;
  151:          }
  152:
  153:          if(!(dir = base_dir)) {
  154:                  dir = current->pwd;
  155:          }
  156:
  157:          /* it is definitely an absolute path */
  158:          if(path[0] == '/') {
  159:                  dir = current->root;
  160:          }
  161:          dir->count++;
  162:          errno = do_namei(path, dir, i_res, d_res, follow_links);
  163:          return errno;
  164: }
  165:
  166: /*
  167:  * namei() returns:
  168:  * i_res -> the inode of the last component of the path, or NULL.
  169:  * d_res -> the inode of the directory where i_res resides, or NULL.
  170:  */
  171: int namei(char *path, struct inode **i_res, struct inode **d_res, int follow_lin
ks)
  172: {
  173:          *i_res = NULL;
  174:          if(d_res) {
  175:                  *d_res = NULL;
  176:          }
  177:          return parse_namei(path, NULL, i_res, d_res, follow_links);
  178: }
```

fs/script.c


```
/*
 * fiwix/fs/script.c
 *
 * Copyright 2019, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/limits.h>
#include <fiwix/process.h>
#include <fiwix/errno.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

int script_load(char *interpreter, char *args, char *data)
{
        char *p;
        int n, noargs;

        /* has shebang? */
        if(data[0] != '#' && data[1] != '!') {
                return -ENOEXEC;
        }

        /* discard possible blanks before the interpreter name */
        for(n = 2; n < NAME_MAX; n++) {
                if(data[n] != ' ' && data[n] != '\t') {
                        break;
                }
        }

        /* get the interpreter name */
        p = interpreter;
        noargs = 0;
        while(n < NAME_MAX) {
                if(data[n] == '\n' || data[n] == NULL) {
                        noargs = 1;
                        break;
                }
                if(data[n] == ' ' || data[n] == '\t') {
                        break;
                }
                *p = data[n];
                n++;
                p++;
        }

        if(!interpreter) {
                return -ENOEXEC;
        }


        /* get the interpreter arguments */
        if(!noargs) {
                p = args;
                /* discard possible blanks before the arguments */
                while(n < NAME_MAX) {
                        if(data[n] != ' ' && data[n] != '\t') {
                                break;
                        }
                        n++;
                }
                while(n < NAME_MAX) {
                        if(data[n] == '\n' || data[n] == NULL) {
                                break;
                        }
                        *p = data[n];
                        n++;
```

fs/script.c


```
68:                             p++;
69:                     }
70:             }
71:
72:             return 0;
73: }
```

fs/super.c


```
    1: /*
    2:  * fiwix/fs/super.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/kernel.h>
   10: #include <fiwix/types.h>
   11: #include <fiwix/errno.h>
   12: #include <fiwix/fs.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/filesystems.h>
   15: #include <fiwix/sleep.h>
   16: #include <fiwix/sched.h>
   17: #include <fiwix/stdio.h>
   18: #include <fiwix/string.h>
   19:
   20: struct mount *mount_table;
   21: static struct resource sync_resource = { NULL, NULL };
   22:
   23: void superblock_lock(struct superblock *sb)
   24: {
   25:         unsigned long int flags;
   26:
   27:         for(;;) {
   28:                 SAVE_FLAGS(flags); CLI();
   29:                 if(sb->locked) {
   30:                         sb->wanted = 1;
   31:                         RESTORE_FLAGS(flags);
   32:                         sleep(&superblock_lock, PROC_UNINTERRUPTIBLE);
   33:                 } else {
   34:                         break;
   35:                 }
   36:         }
   37:         sb->locked = 1;
   38:         RESTORE_FLAGS(flags);
   39: }
   40:
   41: void superblock_unlock(struct superblock *sb)
   42: {
   43:         unsigned long int flags;
   44:
   45:         SAVE_FLAGS(flags); CLI();
   46:         sb->locked = 0;
   47:         if(sb->wanted) {
   48:                 sb->wanted = 0;
   49:                 wakeup(&superblock_lock);
   50:         }
   51:         RESTORE_FLAGS(flags);
   52: }
   53:
   54: struct mount * get_free_mount_point(__dev_t dev)
   55: {
   56:         unsigned long int flags;
   57:         int n;
   58:
   59:         if(!dev) {
   60:                 printk("%s(): invalid device %d,%d.\n", __FUNCTION__, MAJOR(dev)
, MINOR(dev));
   61:                 return NULL;
   62:         }
   63:
   64:         for(n = 0; n < NR_MOUNT_POINTS; n++) {
   65:                 if(mount_table[n].dev == dev) {
   66:                         printk("%s(): device %d,%d already mounted.\n", __FUNCTI
```

fs/super.c


```
ON__, MAJOR(dev), MINOR(dev));
   67:                          return NULL;
   68:                  }
   69:          }
   70:
   71:          SAVE_FLAGS(flags); CLI();
   72:          for(n = 0; n < NR_MOUNT_POINTS; n++) {
   73:                  if(!mount_table[n].used) {
   74:                          /* 'dev' is saved here now for get_superblock() (which
   75:                           * in turn is called by read_inode(), which in turn is
   76:                           * called by iget(), which in turn is called by
   77:                           * read_superblock) to be able to find the device.
   78:                           */
   79:                          mount_table[n].dev = dev;
   80:                          mount_table[n].used = 1;
   81:                          RESTORE_FLAGS(flags);
   82:                          return &mount_table[n];
   83:                  }
   84:          }
   85:          RESTORE_FLAGS(flags);
   86:
   87:          printk("WARNING: %s(): mount-point table is full.\n", __FUNCTION__);
   88:          return NULL;
   89: }
   90:
   91: void release_mount_point(struct mount *mt)
   92: {
   93:          memset_b(mt, NULL, sizeof(struct mount));
   94: }
   95:
   96: struct mount * get_mount_point(struct inode *i_target)
   97: {
   98:          int n;
   99:
  100:          for(n = 0; n < NR_MOUNT_POINTS; n++) {
  101:                  if(mount_table[n].used) {
  102:                          if(S_ISDIR(i_target->i_mode)) {
  103:                                  if(mount_table[n].sb.root == i_target) {
  104:                                          return &mount_table[n];
  105:                                  }
  106:                          }
  107:                          if(S_ISBLK(i_target->i_mode)) {
  108:                                  if(mount_table[n].dev == i_target->rdev) {
  109:                                          return &mount_table[n];
  110:                                  }
  111:                          }
  112:                  }
  113:          }
  114:          return NULL;
  115: }
  116:
  117: struct superblock * get_superblock(__dev_t dev)
  118: {
  119:          int n;
  120:
  121:          for(n = 0; n < NR_MOUNT_POINTS; n++) {
  122:                  if(mount_table[n].used && mount_table[n].dev == dev) {
  123:                          return &mount_table[n].sb;
  124:                  }
  125:          }
  126:          return NULL;
  127: }
  128:
  129: void sync_superblocks(__dev_t dev)
  130: {
  131:          struct superblock *sb;
  132:          int n, errno;
```

fs/super.c

```
  133:
  134:          lock_resource(&sync_resource);
  135:          for(n = 0; n < NR_MOUNT_POINTS; n++) {
  136:                  if(mount_table[n].used && (!dev || mount_table[n].dev == dev)) {
  137:                          sb = &mount_table[n].sb;
  138:                          if(sb->dirty && !(sb->flags & MS_RDONLY)) {
  139:                                  if(sb->fsop && sb->fsop->write_superblock) {
  140:                                          errno = sb->fsop->write_superblock(sb);
  141:                                          if(errno) {
  142:                                                  printk("WARNING: %s(): I/O error
 on device %d,%d while syncing superblock.\n", __FUNCTION__, MAJOR(sb->dev), MINOR(sb->
dev));
  143:                                          }
  144:                                  }
  145:                          }
  146:                  }
  147:          }
  148:          unlock_resource(&sync_resource);
  149: }
  150:
  151: /* pseudo-filesystems are only mountable by the kernel */
  152: int kern_mount(__dev_t dev, struct filesystems *fs)
  153: {
  154:          struct mount *mt;
  155:
  156:          if(!(mt = get_free_mount_point(dev))) {
  157:                  return -EBUSY;
  158:          }
  159:
  160:          if(fs->fsop->read_superblock(dev, &mt->sb)) {
  161:                  release_mount_point(mt);
  162:                  return -EINVAL;
  163:          }
  164:
  165:          mt->dev = dev;
  166:          strcpy(mt->devname, "none");
  167:          strcpy(mt->dirname, "none");
  168:          mt->sb.dir = NULL;
  169:          mt->fs = fs;
  170:          fs->mt = mt;
  171:          return 0;
  172: }
  173:
  174: int mount_root(void)
  175: {
  176:          struct filesystems *fs;
  177:          struct mount *mt;
  178:
  179:          /* FIXME: before trying to mount the filesystem, we should first
  180:           * check if '_rootdev' is a device successfully registered.
  181:           */
  182:
  183:          if(!(fs = get_filesystem(_rootfstype))) {
  184:                  printk("WARNING: %s(): '%s' is not a registered filesystem. Defa
ulting to 'minix'.\n", __FUNCTION__, _rootfstype);
  185:                  if(!(fs = get_filesystem("minix"))) {
  186:                          PANIC("minix filesystem is not registered!\n");
  187:                  }
  188:          }
  189:
  190:          if(!(mt = get_free_mount_point(_rootdev))) {
  191:                  PANIC("unable to get a free mount point.\n");
  192:          }
  193:
  194:          mt->sb.flags = MS_RDONLY;
  195:          if(fs->fsop && fs->fsop->read_superblock) {
  196:                  if(fs->fsop->read_superblock(_rootdev, &mt->sb)) {
```

fs/super.c


```
  197:                         PANIC("unable to mount root filesystem on %s.\n", _rootd
evname);
  198:                 }
  199:         }
  200:
  201:         strcpy(mt->devname, "/dev/root");
  202:         strcpy(mt->dirname, "/");
  203:         mt->dev = _rootdev;
  204:         mt->sb.root->mount_point = mt->sb.root;
  205:         mt->sb.root->count++;
  206:         mt->sb.dir = mt->sb.root;
  207:         mt->sb.dir->count++;
  208:         mt->fs = fs;
  209:
  210:         current->root = mt->sb.root;
  211:         current->root->count++;
  212:         current->pwd = mt->sb.root;
  213:         current->pwd->count++;
  214:         iput(mt->sb.root);
  215:
  216:         printk("mounted root device (%s filesystem) in readonly mode.\n", fs->na
me);
  217:         return 0;
  218: }
  219:
  220: void mount_init(void)
  221: {
  222:         memset_b(mount_table, NULL, mount_table_size);
  223: }
```

fs/ext2/bitmaps.c


```
    1: /*
    2:  * fiwix/fs/ext2/bitmaps.c
    3:  *
    4:  * Copyright 2019, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/types.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/filesystems.h>
   12: #include <fiwix/fs_ext2.h>
   13: #include <fiwix/buffer.h>
   14: #include <fiwix/errno.h>
   15: #include <fiwix/stat.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: static int find_first_zero(struct superblock *sb, __blk_t block)
   20: {
   21:         unsigned char c;
   22:         int blksize;
   23:         int n, n2;
   24:         struct buffer *buf;
   25:
   26:         blksize = sb->s_blocksize;
   27:
   28:         if(!(buf = bread(sb->dev, block, blksize))) {
   29:                 return -EIO;
   30:         }
   31:         for(n = 0; n < blksize; n++) {
   32:                 c = (unsigned char)buf->data[n];
   33:                 for(n2 = 0; n2 < 8; n2++) {
   34:                         if(!(c & (1 << n2))) {
   35:                                 brelse(buf);
   36:                                 return n2 + (n * 8) + 1;
   37:                         }
   38:                 }
   39:         }
   40:         brelse(buf);
   41:         return 0;
   42: }
   43:
   44: static int change_bit(int mode, struct superblock *sb, __blk_t block, int item)
   45: {
   46:         int byte, bit, mask;
   47:         struct buffer *buf;
   48:
   49:         block += item / (sb->s_blocksize * 8);
   50:         byte = (item % (sb->s_blocksize * 8)) / 8;
   51:         bit = (item % (sb->s_blocksize * 8)) % 8;
   52:         mask = 1 << bit;
   53:
   54:         if(!(buf = bread(sb->dev, block, sb->s_blocksize))) {
   55:                 return -EIO;
   56:         }
   57:
   58:         if(mode == CLEAR_BIT) {
   59:                 if(!(buf->data[byte] & mask)) {
   60:                         brelse(buf);
   61:                         return 1;
   62:                 }
   63:                 buf->data[byte] &= ~mask;
   64:         }
   65:         if(mode == SET_BIT) {
   66:                 if((buf->data[byte] & mask)) {
   67:                         brelse(buf);
```

fs/ext2/bitmaps.c


```
   68:                          return 1;
   69:                  }
   70:                  buf->data[byte] |= mask;
   71:          }
   72:
   73:          bwrite(buf);
   74:          return 0;
   75: }
   76:
   77: /*
   78:  * Unlike of what Ext2 specifies/suggests, this inode allocation does NOT
   79:  * try to assign inodes in the same block group of the directory in which
   80:  * they will be created.
   81:  */
   82: int ext2_ialloc(struct inode *i, int mode)
   83: {
   84:          __ino_t inode;
   85:          __blk_t block;
   86:          struct superblock *sb;
   87:          struct ext2_group_desc *gd;
   88:          struct buffer *buf;
   89:          int bg, d, errno;
   90:
   91:          sb = i->sb;
   92:          superblock_lock(sb);
   93:
   94:          block = SUPERBLOCK + sb->u.ext2.sb.s_first_data_block;
   95:          inode = 0;
   96:          buf = NULL;
   97:
   98:          /* read through all group descriptors to find the first unallocated inod
e */
   99:          for(bg = 0, d = 0; bg < sb->u.ext2.block_groups; bg++, d++) {
  100:                  if(!(bg % (sb->s_blocksize / sizeof(struct ext2_group_desc)))) {
  101:                          if(buf) {
  102:                                  brelse(buf);
  103:                                  block++;
  104:                                  d = 0;
  105:                          }
  106:                          if(!(buf = bread(sb->dev, block, sb->s_blocksize))) {
  107:                                  superblock_unlock(sb);
  108:                                  return -EIO;
  109:                          }
  110:                  }
  111:                  gd = (struct ext2_group_desc *)(buf->data + (d * sizeof(struct e
xt2_group_desc)));
  112:                  if(gd->bg_free_inodes_count) {
  113:                          if((inode = find_first_zero(sb, gd->bg_inode_bitmap))) {
  114:                                  break;
  115:                          }
  116:                  }
  117:          }
  118:          if(!inode) {
  119:                  brelse(buf);
  120:                  superblock_unlock(sb);
  121:                  return -ENOSPC;
  122:          }
  123:
  124:          errno = change_bit(SET_BIT, sb, gd->bg_inode_bitmap, inode - 1);
  125:          if(errno) {
  126:                  if(errno < 0) {
  127:                          printk("WARNING: %s(): unable to set inode %d.\n", __FUN
CTION__, inode);
  128:                          brelse(buf);
  129:                          superblock_unlock(sb);
  130:                          return errno;
  131:                  } else {
```

fs/ext2/bitmaps.c


```
  132:                          printk("WARNING: %s(): inode %d is already marked as use
d!\n", __FUNCTION__, inode);
  133:                  }
  134:          }
  135:
  136:          inode += bg * EXT2_INODES_PER_GROUP(sb);
  137:          gd->bg_free_inodes_count--;
  138:          sb->u.ext2.sb.s_free_inodes_count--;
  139:          if(S_ISDIR(mode)) {
  140:                  gd->bg_used_dirs_count++;
  141:          }
  142:          bwrite(buf);
  143:
  144:          i->inode = inode;
  145:          i->i_atime = CURRENT_TIME;
  146:          i->i_mtime = CURRENT_TIME;
  147:          i->i_ctime = CURRENT_TIME;
  148:
  149:          superblock_unlock(sb);
  150:          return 0;
  151: }
  152:
  153: void ext2_ifree(struct inode *i)
  154: {
  155:          struct ext2_group_desc *gd;
  156:          struct buffer *buf;
  157:          struct superblock *sb;
  158:          __blk_t b, bg;
  159:          int errno;
  160:
  161:          if(!i->inode || i->inode > i->sb->u.ext2.sb.s_inodes_count) {
  162:                  printk("WARNING: %s(): invalid inode %d!\n", __FUNCTION__, i->in
ode);
  163:                  return;
  164:          }
  165:
  166:          if(i->i_blocks) {
  167:                  ext2_truncate(i, 0);
  168:          }
  169:
  170:          sb = i->sb;
  171:          superblock_lock(sb);
  172:
  173:          b = SUPERBLOCK + sb->u.ext2.sb.s_first_data_block;
  174:          bg = (i->inode - 1) / EXT2_INODES_PER_GROUP(sb);
  175:          if(!(buf = bread(sb->dev, b + (bg / EXT2_DESC_PER_BLOCK(sb)), sb->s_bloc
ksize))) {
  176:                  superblock_unlock(sb);
  177:                  return;
  178:          }
  179:          gd = (struct ext2_group_desc *)(buf->data + ((bg % EXT2_DESC_PER_BLOCK(s
b)) * sizeof(struct ext2_group_desc)));
  180:          errno = change_bit(CLEAR_BIT, sb, gd->bg_inode_bitmap, (i->inode - 1) %
EXT2_INODES_PER_GROUP(sb));
  181:
  182:          if(errno) {
  183:                  if(errno < 0) {
  184:                          printk("WARNING: %s(): unable to clear inode %d.\n", __F
UNCTION__, i->inode);
  185:                          brelse(buf);
  186:                          superblock_unlock(sb);
  187:                          return;
  188:                  } else {
  189:                          printk("WARNING: %s(): inode %d is already marked as fre
e!\n", __FUNCTION__, i->inode);
  190:                  }
  191:          }
```

fs/ext2/bitmaps.c


```
  192:
  193:          gd->bg_free_inodes_count++;
  194:          sb->u.ext2.sb.s_free_inodes_count++;
  195:          if(S_ISDIR(i->i_mode)) {
  196:                  gd->bg_used_dirs_count--;
  197:          }
  198:          bwrite(buf);
  199:
  200:          i->i_size = 0;
  201:          i->i_mtime = CURRENT_TIME;
  202:          i->i_ctime = CURRENT_TIME;
  203:          i->dirty = 1;
  204:
  205:          superblock_unlock(sb);
  206:          return;
  207: }
  208:
  209: int ext2_balloc(struct superblock *sb)
  210: {
  211:          __blk_t b, block;
  212:          struct ext2_group_desc *gd;
  213:          struct buffer *buf;
  214:          int bg, d, errno;
  215:
  216:          superblock_lock(sb);
  217:
  218:          b = SUPERBLOCK + sb->u.ext2.sb.s_first_data_block;
  219:          block = 0;
  220:          buf = NULL;
  221:
  222:          /* read through all group descriptors to find the first unallocated bloc
k */
  223:          for(bg = 0, d = 0; bg < sb->u.ext2.block_groups; bg++, d++) {
  224:                  if(!(bg % (sb->s_blocksize / sizeof(struct ext2_group_desc)))) {
  225:                          if(buf) {
  226:                                  brelse(buf);
  227:                                  b++;
  228:                                  d = 0;
  229:                          }
  230:                          if(!(buf = bread(sb->dev, b, sb->s_blocksize))) {
  231:                                  superblock_unlock(sb);
  232:                                  return -EIO;
  233:                          }
  234:                  }
  235:                  gd = (struct ext2_group_desc *)(buf->data + (d * sizeof(struct e
xt2_group_desc)));
  236:                  if(gd->bg_free_blocks_count) {
  237:                          if((block = find_first_zero(sb, gd->bg_block_bitmap))) {
  238:                                  break;
  239:                          }
  240:                  }
  241:          }
  242:          if(!block) {
  243:                  brelse(buf);
  244:                  superblock_unlock(sb);
  245:                  return -ENOSPC;
  246:          }
  247:
  248:          errno = change_bit(SET_BIT, sb, gd->bg_block_bitmap, block - 1);
  249:          if(errno) {
  250:                  if(errno < 0) {
  251:                          printk("WARNING: %s(): unable to set block %d.\n", __FUN
CTION__, block);
  252:                          brelse(buf);
  253:                          superblock_unlock(sb);
  254:                          return errno;
  255:                  } else {
```

fs/ext2/bitmaps.c


```
  256:                          printk("WARNING: %s(): block %d is already marked as use
d!\n", __FUNCTION__, block);
  257:                  }
  258:          }
  259:
  260:          block += bg * EXT2_BLOCKS_PER_GROUP(sb);
  261:          gd->bg_free_blocks_count--;
  262:          sb->u.ext2.sb.s_free_blocks_count--;
  263:          bwrite(buf);
  264:
  265:          superblock_unlock(sb);
  266:          return block;
  267: }
  268:
  269: void ext2_bfree(struct superblock *sb, int block)
  270: {
  271:          struct ext2_group_desc *gd;
  272:          struct buffer *buf;
  273:          __blk_t b, bg;
  274:          int errno;
  275:
  276:          if(!block || block > sb->u.ext2.sb.s_blocks_count) {
  277:                  printk("WARNING: %s(): invalid block %d!\n", __FUNCTION__, block
);
  278:                  return;
  279:          }
  280:
  281:          superblock_lock(sb);
  282:
  283:          b = SUPERBLOCK + sb->u.ext2.sb.s_first_data_block;
  284:          bg = (block - 1) / EXT2_BLOCKS_PER_GROUP(sb);
  285:          if(!(buf = bread(sb->dev, b + (bg / EXT2_DESC_PER_BLOCK(sb)), sb->s_bloc
ksize))) {
  286:                  superblock_unlock(sb);
  287:                  return;
  288:          }
  289:          gd = (struct ext2_group_desc *)(buf->data + ((bg % EXT2_DESC_PER_BLOCK(s
b)) * sizeof(struct ext2_group_desc)));
  290:          errno = change_bit(CLEAR_BIT, sb, gd->bg_block_bitmap, (block - 1) % EXT
2_BLOCKS_PER_GROUP(sb));
  291:
  292:          if(errno) {
  293:                  if(errno < 0) {
  294:                          printk("WARNING: %s(): unable to free block %d.\n", __FU
NCTION__, block);
  295:                          brelse(buf);
  296:                          superblock_unlock(sb);
  297:                          return;
  298:                  } else {
  299:                          printk("WARNING: %s(): block %d is already marked as fre
e!\n", __FUNCTION__, block);
  300:                  }
  301:          }
  302:
  303:          gd->bg_free_blocks_count++;
  304:          sb->u.ext2.sb.s_free_blocks_count++;
  305:          bwrite(buf);
  306:
  307:          superblock_unlock(sb);
  308:          return;
  309: }
```

fs/ext2/dir.c


```
/*
 * fiwix/fs/ext2/dir.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/buffer.h>
#include <fiwix/fs.h>
#include <fiwix/filesystems.h>
#include <fiwix/stat.h>
#include <fiwix/dirent.h>
#include <fiwix/mm.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct fs_operations ext2_dir_fsop = {
        0,
        0,

        ext2_dir_open,
        ext2_dir_close,
        ext2_dir_read,
        ext2_dir_write,
        NULL,                   /* ioctl */
        NULL,                   /* lseek */
        ext2_dir_readdir,
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        ext2_bmap,
        ext2_lookup,
        ext2_rmdir,
        ext2_link,
        ext2_unlink,
        ext2_symlink,
        ext2_mkdir,
        ext2_mknod,
        NULL,                   /* truncate */
        ext2_create,
        ext2_rename,

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statfs */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

int ext2_dir_open(struct inode *i, struct fd *fd_table)
{
        fd_table->offset = 0;
        return 0;
}

int ext2_dir_close(struct inode *i, struct fd *fd_table)
```

fs/ext2/dir.c


```
   68: {
   69:         return 0;
   70: }
   71:
   72: int ext2_dir_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t c
ount)
   73: {
   74:         return -EISDIR;
   75: }
   76:
   77: int ext2_dir_write(struct inode *i, struct fd *fd_table, const char *buffer, __s
ize_t count)
   78: {
   79:         return -EBADF;
   80: }
   81:
   82: int ext2_dir_readdir(struct inode *i, struct fd *fd_table, struct dirent *dirent
, unsigned int count)
   83: {
   84:         __blk_t block;
   85:         unsigned int doffset, offset;
   86:         unsigned int size, dirent_len;
   87:         struct ext2_dir_entry_2 *d;
   88:         int base_dirent_len;
   89:         int blksize;
   90:         struct buffer *buf;
   91:
   92:         if(!(S_ISDIR(i->i_mode))) {
   93:                 return -EBADF;
   94:         }
   95:
   96:         blksize = i->sb->s_blocksize;
   97:         if(fd_table->offset > i->i_size) {
   98:                 fd_table->offset = i->i_size;
   99:         }
  100:
  101:         base_dirent_len = sizeof(dirent->d_ino) + sizeof(dirent->d_off) + sizeof
(dirent->d_reclen);
  102:         doffset = offset = size = 0;
  103:
  104:         while(doffset < count) {
  105:                 if((block = bmap(i, fd_table->offset, FOR_READING)) < 0) {
  106:                         return block;
  107:                 }
  108:                 if(block) {
  109:                         if(!(buf = bread(i->dev, block, blksize))) {
  110:                                 return -EIO;
  111:                         }
  112:
  113:                         doffset = fd_table->offset;
  114:                         offset = fd_table->offset % blksize;
  115:                         while(doffset < i->i_size && offset < blksize) {
  116:                                 d = (struct ext2_dir_entry_2 *)(buf->data + offs
et);
  117:                                 if(d->inode) {
  118:                                         dirent_len = (base_dirent_len + (d->name
_len + 1)) + 3;
  119:                                         dirent_len &= ~3;       /* round up */
  120:                                         dirent->d_ino = d->inode;
  121:                                         if((size + dirent_len) < count) {
  122:                                                 dirent->d_off = doffset;
  123:                                                 dirent->d_reclen = dirent_len;
  124:                                                 memcpy_b(dirent->d_name, d->name
, d->name_len);
  125:                                                 dirent->d_name[d->name_len] = NU
LL;
  126:                                                 dirent = (struct dirent *)((char
```

fs/ext2/dir.c


```
 *)dirent + dirent_len);
  127:                                                  size += dirent_len;
  128:                                          } else {
  129:                                                  break;
  130:                                          }
  131:                                  }
  132:                                  doffset += d->rec_len;
  133:                                  offset += d->rec_len;
  134:                                  if(!d->rec_len) {
  135:                                          break;
  136:                                  }
  137:                          }
  138:                          brelse(buf);
  139:                  }
  140:                  fd_table->offset &= ~(blksize - 1);
  141:                  doffset = fd_table->offset;
  142:                  fd_table->offset += offset;
  143:                  doffset += blksize;
  144:          }
  145:
  146:          return size;
  147: }
```

fs/ext2/file.c


```
/*
 * fiwix/fs/ext2/file.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/kernel.h>
#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/buffer.h>
#include <fiwix/fs.h>
#include <fiwix/filesystems.h>
#include <fiwix/mm.h>
#include <fiwix/mman.h>
#include <fiwix/fcntl.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct fs_operations ext2_file_fsop = {
        0,
        0,

        ext2_file_open,
        ext2_file_close,
        file_read,
        ext2_file_write,
        NULL,                   /* ioctl */
        ext2_file_lseek,
        NULL,                   /* readdir */
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        ext2_bmap,
        NULL,                   /* lookup */
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        ext2_truncate,
        NULL,                   /* create */
        NULL,                   /* rename */

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statfs */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

int ext2_file_open(struct inode *i, struct fd *fd_table)
{
        if(fd_table->flags & O_APPEND) {
                fd_table->offset = i->i_size;
        } else {
                fd_table->offset = 0;
```

fs/ext2/file.c


```
   68:         }
   69:         if(fd_table->flags & O_TRUNC) {
   70:                 i->i_size = 0;
   71:                 ext2_truncate(i, 0);
   72:         }
   73:         return 0;
   74: }
   75:
   76: int ext2_file_close(struct inode *i, struct fd *fd_table)
   77: {
   78:         return 0;
   79: }
   80:
   81: int ext2_file_write(struct inode *i, struct fd *fd_table, const char *buffer, __
size_t count)
   82: {
   83:         __blk_t block;
   84:         __off_t total_written;
   85:         unsigned int boffset, bytes;
   86:         int blksize;
   87:         struct buffer *buf;
   88:
   89:         inode_lock(i);
   90:
   91:         blksize = i->sb->s_blocksize;
   92:         total_written = 0;
   93:
   94:         if(fd_table->flags & O_APPEND) {
   95:                 fd_table->offset = i->i_size;
   96:         }
   97:
   98:         while(total_written < count) {
   99:                 boffset = fd_table->offset % blksize;
  100:                 if((block = bmap(i, fd_table->offset, FOR_WRITING)) < 0) {
  101:                         inode_unlock(i);
  102:                         return block;
  103:                 }
  104:                 bytes = blksize - boffset;
  105:                 bytes = MIN(bytes, (count - total_written));
  106:                 if(!(buf = bread(i->dev, block, blksize))) {
  107:                         inode_unlock(i);
  108:                         return -EIO;
  109:                 }
  110:                 memcpy_b(buf->data + boffset, buffer + total_written, bytes);
  111:                 update_page_cache(i, fd_table->offset, buffer + total_written, b
ytes);
  112:                 bwrite(buf);
  113:                 total_written += bytes;
  114:                 boffset += bytes;
  115:                 boffset %= blksize;
  116:                 fd_table->offset += bytes;
  117:         }
  118:
  119:         if(fd_table->offset > i->i_size) {
  120:                 i->i_size = fd_table->offset;
  121:         }
  122:         i->i_ctime = CURRENT_TIME;
  123:         i->i_mtime = CURRENT_TIME;
  124:         i->dirty = 1;
  125:
  126:         inode_unlock(i);
  127:         return total_written;
  128: }
  129:
  130: int ext2_file_lseek(struct inode *i, __off_t offset)
  131: {
  132:         return offset;
```

fs/ext2/file.c


```
133: }
```

fs/ext2/inode.c


```
    1: /*
    2:  * fiwix/fs/ext2/inode.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/filesystems.h>
   11: #include <fiwix/fs_ext2.h>
   12: #include <fiwix/fs_pipe.h>
   13: #include <fiwix/statfs.h>
   14: #include <fiwix/sleep.h>
   15: #include <fiwix/stat.h>
   16: #include <fiwix/sched.h>
   17: #include <fiwix/buffer.h>
   18: #include <fiwix/mm.h>
   19: #include <fiwix/process.h>
   20: #include <fiwix/errno.h>
   21: #include <fiwix/stdio.h>
   22: #include <fiwix/string.h>
   23:
   24: #define BLOCKS_PER_IND_BLOCK(sb)        (EXT2_BLOCK_SIZE(sb) / sizeof(unsigned i
nt))
   25: #define BLOCKS_PER_DIND_BLOCK(sb)       (BLOCKS_PER_IND_BLOCK(sb) * BLOCKS_PER_I
ND_BLOCK(sb))
   26: #define BLOCKS_PER_TIND_BLOCK(sb)       (BLOCKS_PER_IND_BLOCK(sb) * BLOCKS_PER_I
ND_BLOCK(sb) * BLOCKS_PER_IND_BLOCK(sb))
   27:
   28: #define EXT2_INODES_PER_BLOCK(sb)       (EXT2_BLOCK_SIZE(sb) / sizeof(struct ext
2_inode))
   29:
   30: static void free_indblock(struct inode *i, int block, int offset)
   31: {
   32:         int n;
   33:         struct buffer *buf;
   34:         __blk_t *indblock;
   35:
   36:         if(!(buf = bread(i->dev, block, i->sb->s_blocksize))) {
   37:                 printk("WARNING: %s(): error reading block %d.\n", __FUNCTION__,
 block);
   38:                 return;
   39:         }
   40:         indblock = (__blk_t *)buf->data;
   41:         for(n = offset; n < BLOCKS_PER_IND_BLOCK(i->sb); n++) {
   42:                 if(indblock[n]) {
   43:                         ext2_bfree(i->sb, indblock[n]);
   44:                         indblock[n] = 0;
   45:                         i->i_blocks -= i->sb->s_blocksize / 512;
   46:                 }
   47:         }
   48:         bwrite(buf);
   49: }
   50:
   51: static int get_group_desc(struct superblock *sb, __blk_t block_group, struct ext
2_group_desc *gd)
   52: {
   53:         __blk_t group_desc_block;
   54:         int group_desc;
   55:         struct buffer *buf;
   56:
   57:         group_desc_block = block_group / EXT2_DESC_PER_BLOCK(sb);
   58:         group_desc = block_group % EXT2_DESC_PER_BLOCK(sb);
   59:         if(!(buf = bread(sb->dev, SUPERBLOCK + sb->u.ext2.sb.s_first_data_block
+ group_desc_block, sb->s_blocksize))) {
   60:                 return -EIO;
```

fs/ext2/inode.c


```
   61:         }
   62:         memcpy_b(gd, (void *)(buf->data + (group_desc * sizeof(struct ext2_group
_desc))), sizeof(struct ext2_group_desc));
   63:         brelse(buf);
   64:         return 0;
   65: }
   66:
   67: int ext2_read_inode(struct inode *i)
   68: {
   69:         __blk_t block_group, block;
   70:         unsigned int offset;
   71:         struct superblock *sb;
   72:         struct ext2_inode *ii;
   73:         struct ext2_group_desc gd;
   74:         struct buffer *buf;
   75:
   76:         if(!(sb = get_superblock(i->dev))) {
   77:                 printk("WARNING: %s(): get_superblock() has returned NULL.\n");
   78:                 return -EINVAL;
   79:         }
   80:         block_group = ((i->inode - 1) / EXT2_INODES_PER_GROUP(sb));
   81:         if(get_group_desc(sb, block_group, &gd)) {
   82:                 return -EIO;
   83:         }
   84:         block = (((i->inode - 1) % EXT2_INODES_PER_GROUP(sb)) / EXT2_INODES_PER_
BLOCK(sb));
   85:
   86:         if(!(buf = bread(i->dev, gd.bg_inode_table + block, i->sb->s_blocksize))
) {
   87:                 return -EIO;
   88:         }
   89:         offset = ((((i->inode - 1) % EXT2_INODES_PER_GROUP(sb)) % EXT2_INODES_PE
R_BLOCK(sb)) * sizeof(struct ext2_inode));
   90:
   91:         ii = (struct ext2_inode *)(buf->data + offset);
   92:         memcpy_b(&i->u.ext2.i_data, ii->i_block, sizeof(ii->i_block));
   93:
   94:         i->i_mode = ii->i_mode;
   95:         i->i_uid = ii->i_uid;
   96:         i->i_size = ii->i_size;
   97:         i->i_atime = ii->i_atime;
   98:         i->i_ctime = ii->i_ctime;
   99:         i->i_mtime = ii->i_mtime;
  100:         i->i_gid = ii->i_gid;
  101:         i->i_nlink = ii->i_links_count;
  102:         i->i_blocks = ii->i_blocks;
  103:         i->i_flags = ii->i_flags;
  104:         i->count = 1;
  105:         switch(i->i_mode & S_IFMT) {
  106:                 case S_IFCHR:
  107:                         i->fsop = &def_chr_fsop;
  108:                         i->rdev = ii->i_block[0];
  109:                         break;
  110:                 case S_IFBLK:
  111:                         i->fsop = &def_blk_fsop;
  112:                         i->rdev = ii->i_block[0];
  113:                         break;
  114:                 case S_IFIFO:
  115:                         i->fsop = &pipefs_fsop;
  116:                         /* it's a union so we need to clear pipefs_i */
  117:                         memset_b(&i->u.pipefs, NULL, sizeof(struct pipefs_inode)
);
  118:                         break;
  119:                 case S_IFDIR:
  120:                         i->fsop = &ext2_dir_fsop;
  121:                         break;
  122:                 case S_IFREG:
```

fs/ext2/inode.c


```
  123:                          i->fsop = &ext2_file_fsop;
  124:                          break;
  125:                  case S_IFLNK:
  126:                          i->fsop = &ext2_symlink_fsop;
  127:                          break;
  128:                  case S_IFSOCK:
  129:                          i->fsop = NULL;
  130:                          break;
  131:                  default:
  132:                          printk("WARNING: %s(): invalid inode (%d) mode %08o.\n",
 __FUNCTION__, i->inode, i->i_mode);
  133:                          brelse(buf);
  134:                          return -ENOENT;
  135:          }
  136:          brelse(buf);
  137:          return 0;
  138: }
  139:
  140: int ext2_write_inode(struct inode *i)
  141: {
  142:          __blk_t block_group, block;
  143:          short int offset;
  144:          struct superblock *sb;
  145:          struct ext2_inode *ii;
  146:          struct ext2_group_desc gd;
  147:          struct buffer *buf;
  148:
  149:          if(!(sb = get_superblock(i->dev))) {
  150:                  printk("WARNING: %s(): get_superblock() has returned NULL.\n");
  151:                  return -EINVAL;
  152:          }
  153:          block_group = ((i->inode - 1) / EXT2_INODES_PER_GROUP(sb));
  154:          if(get_group_desc(sb, block_group, &gd)) {
  155:                  return -EIO;
  156:          }
  157:          block = (((i->inode - 1) % EXT2_INODES_PER_GROUP(sb)) / EXT2_INODES_PER_
BLOCK(sb));
  158:
  159:          if(!(buf = bread(i->dev, gd.bg_inode_table + block, i->sb->s_blocksize))
) {
  160:                  return -EIO;
  161:          }
  162:          offset = ((((i->inode - 1) % EXT2_INODES_PER_GROUP(sb)) % EXT2_INODES_PE
R_BLOCK(sb)) * sizeof(struct ext2_inode));
  163:          ii = (struct ext2_inode *)(buf->data + offset);
  164:          memset_b(ii, 0, sizeof(struct ext2_inode));
  165:
  166:          ii->i_mode = i->i_mode;
  167:          ii->i_uid = i->i_uid;
  168:          ii->i_size = i->i_size;
  169:          ii->i_atime = i->i_atime;
  170:          ii->i_ctime = i->i_ctime;
  171:          ii->i_mtime = i->i_mtime;
  172:          ii->i_dtime = i->u.ext2.i_dtime;
  173:          ii->i_gid = i->i_gid;
  174:          ii->i_links_count = i->i_nlink;
  175:          ii->i_blocks = i->i_blocks;
  176:          ii->i_flags = i->i_flags;
  177:          if(S_ISCHR(i->i_mode) || S_ISBLK(i->i_mode)) {
  178:                  ii->i_block[0] = i->rdev;
  179:          } else {
  180:                  memcpy_b(ii->i_block, &i->u.ext2.i_data, sizeof(i->u.ext2.i_data
));
  181:          }
  182:          i->dirty = 0;
  183:          bwrite(buf);
  184:          return 0;
```

fs/ext2/inode.c


```
  185: }
  186:
  187: int ext2_bmap(struct inode *i, __off_t offset, int mode)
  188: {
  189:         unsigned char level;
  190:         __blk_t *indblock, *dindblock, *tindblock;
  191:         __blk_t block, iblock, dblock, tblock, newblock;
  192:         int blksize;
  193:         struct buffer *buf, *buf2, *buf3, *buf4;
  194:
  195:         blksize = i->sb->s_blocksize;
  196:         block = offset / blksize;
  197:         level = 0;
  198:         buf3 = NULL;    /* makes GCC happy */
  199:
  200:         if(block < EXT2_NDIR_BLOCKS) {
  201:                 level = EXT2_NDIR_BLOCKS - 1;
  202:         } else {
  203:                 if(block < (BLOCKS_PER_IND_BLOCK(i->sb) + EXT2_NDIR_BLOCKS)) {
  204:                         level = EXT2_IND_BLOCK;
  205:                 } else if(block < ((BLOCKS_PER_IND_BLOCK(i->sb) * BLOCKS_PER_IND
_BLOCK(i->sb)) + BLOCKS_PER_IND_BLOCK(i->sb) + EXT2_NDIR_BLOCKS)) {
  206:                         level = EXT2_DIND_BLOCK;
  207:                 } else {
  208:                         level = EXT2_TIND_BLOCK;
  209:                 }
  210:                 block -= EXT2_NDIR_BLOCKS;
  211:         }
  212:
  213:         if(level < EXT2_NDIR_BLOCKS) {
  214:                 if(!i->u.ext2.i_data[block] && mode == FOR_WRITING) {
  215:                         if((newblock = ext2_balloc(i->sb)) < 0) {
  216:                                 return -ENOSPC;
  217:                         }
  218:                         /* initialize the new block */
  219:                         if(!(buf = bread(i->dev, newblock, blksize))) {
  220:                                 ext2_bfree(i->sb, newblock);
  221:                                 return -EIO;
  222:                         }
  223:                         memset_b(buf->data, 0, blksize);
  224:                         bwrite(buf);
  225:                         i->u.ext2.i_data[block] = newblock;
  226:                         i->i_blocks += blksize / 512;
  227:                 }
  228:                 return i->u.ext2.i_data[block];
  229:         }
  230:
  231:         if(!i->u.ext2.i_data[level]) {
  232:                 if(mode == FOR_WRITING) {
  233:                         if((newblock = ext2_balloc(i->sb)) < 0) {
  234:                                 return -ENOSPC;
  235:                         }
  236:                         /* initialize the new block */
  237:                         if(!(buf = bread(i->dev, newblock, blksize))) {
  238:                                 ext2_bfree(i->sb, newblock);
  239:                                 return -EIO;
  240:                         }
  241:                         memset_b(buf->data, 0, blksize);
  242:                         bwrite(buf);
  243:                         i->u.ext2.i_data[level] = newblock;
  244:                         i->i_blocks += blksize / 512;
  245:                 } else {
  246:                         return 0;
  247:                 }
  248:         }
  249:         if(!(buf = bread(i->dev, i->u.ext2.i_data[level], blksize))) {
  250:                 return -EIO;
```

fs/ext2/inode.c


```
  251:         }
  252:         indblock = (__blk_t *)buf->data;
  253:         dblock = block - BLOCKS_PER_IND_BLOCK(i->sb);
  254:         tblock = block - (BLOCKS_PER_IND_BLOCK(i->sb) * BLOCKS_PER_IND_BLOCK(i->
sb)) - BLOCKS_PER_IND_BLOCK(i->sb);
  255:
  256:         if(level == EXT2_DIND_BLOCK) {
  257:                 block = dblock / BLOCKS_PER_IND_BLOCK(i->sb);
  258:         }
  259:         if(level == EXT2_TIND_BLOCK) {
  260:                 block = tblock / (BLOCKS_PER_IND_BLOCK(i->sb) * BLOCKS_PER_IND_B
LOCK(i->sb));
  261:         }
  262:
  263:         if(!indblock[block]) {
  264:                 if(mode == FOR_WRITING) {
  265:                         if((newblock = ext2_balloc(i->sb)) < 0) {
  266:                                 brelse(buf);
  267:                                 return -ENOSPC;
  268:                         }
  269:                         /* initialize the new block */
  270:                         if(!(buf2 = bread(i->dev, newblock, blksize))) {
  271:                                 ext2_bfree(i->sb, newblock);
  272:                                 brelse(buf);
  273:                                 return -EIO;
  274:                         }
  275:                         memset_b(buf2->data, 0, blksize);
  276:                         bwrite(buf2);
  277:                         indblock[block] = newblock;
  278:                         i->i_blocks += blksize / 512;
  279:                         if(level == EXT2_IND_BLOCK) {
  280:                                 bwrite(buf);
  281:                                 return newblock;
  282:                         }
  283:                         buf->dirty = 1;
  284:                         buf->valid = 1;
  285:                 } else {
  286:                         brelse(buf);
  287:                         return 0;
  288:                 }
  289:         }
  290:         if(level == EXT2_IND_BLOCK) {
  291:                 newblock = indblock[block];
  292:                 brelse(buf);
  293:                 return newblock;
  294:         }
  295:
  296:         if(level == EXT2_TIND_BLOCK) {
  297:                 if(!(buf3 = bread(i->dev, indblock[block], blksize))) {
  298:                         printk("%s(): returning -EIO\n", __FUNCTION__);
  299:                         brelse(buf);
  300:                         return -EIO;
  301:                 }
  302:                 tindblock = (__blk_t *)buf3->data;
  303:                 block = tindblock[tblock / BLOCKS_PER_IND_BLOCK(i->sb)];
  304:                 if(!block) {
  305:                         if(mode == FOR_WRITING) {
  306:                                 if((newblock = ext2_balloc(i->sb)) < 0) {
  307:                                         brelse(buf);
  308:                                         brelse(buf3);
  309:                                         return -ENOSPC;
  310:                                 }
  311:                                 /* initialize the new block */
  312:                                 if(!(buf4 = bread(i->dev, newblock, blksize))) {
  313:                                         ext2_bfree(i->sb, newblock);
  314:                                         brelse(buf);
  315:                                         brelse(buf3);
```

fs/ext2/inode.c


```
  316:                                            return -EIO;
  317:                                    }
  318:                                    memset_b(buf4->data, 0, blksize);
  319:                                    bwrite(buf4);
  320:                                    tindblock[tblock / BLOCKS_PER_IND_BLOCK(i->sb)]
= newblock;
  321:                                    i->i_blocks += blksize / 512;
  322:                                    buf3->dirty = 1;
  323:                                    buf3->valid = 1;
  324:                                    block = newblock;
  325:                            } else {
  326:                                    brelse(buf);
  327:                                    brelse(buf3);
  328:                                    return 0;
  329:                            }
  330:                    }
  331:                    dblock = tblock;
  332:                    iblock = tblock / BLOCKS_PER_IND_BLOCK(i->sb);
  333:                    if(!(buf2 = bread(i->dev, block, blksize))) {
  334:                            printk("%s(): returning -EIO\n", __FUNCTION__);
  335:                            brelse(buf);
  336:                            brelse(buf3);
  337:                            return -EIO;
  338:                    }
  339:            } else {
  340:                    iblock = block;
  341:                    if(!(buf2 = bread(i->dev, indblock[iblock], blksize))) {
  342:                            printk("%s(): returning -EIO\n", __FUNCTION__);
  343:                            brelse(buf);
  344:                            return -EIO;
  345:                    }
  346:            }
  347:
  348:            dindblock = (__blk_t *)buf2->data;
  349:            block = dindblock[dblock - (iblock * BLOCKS_PER_IND_BLOCK(i->sb))];
  350:            if(!block && mode == FOR_WRITING) {
  351:                    if((newblock = ext2_balloc(i->sb)) < 0) {
  352:                            brelse(buf);
  353:                            if(level == EXT2_TIND_BLOCK) {
  354:                                    brelse(buf3);
  355:                            }
  356:                            brelse(buf2);
  357:                            return -ENOSPC;
  358:                    }
  359:                    /* initialize the new block */
  360:                    if(!(buf4 = bread(i->dev, newblock, blksize))) {
  361:                            ext2_bfree(i->sb, newblock);
  362:                            brelse(buf);
  363:                            if(level == EXT2_TIND_BLOCK) {
  364:                                    brelse(buf3);
  365:                            }
  366:                            brelse(buf2);
  367:                            return -EIO;
  368:                    }
  369:                    memset_b(buf4->data, 0, blksize);
  370:                    bwrite(buf4);
  371:                    dindblock[dblock - (iblock * BLOCKS_PER_IND_BLOCK(i->sb))] = new
block;
  372:                    i->i_blocks += blksize / 512;
  373:                    buf2->dirty = 1;
  374:                    buf2->valid = 1;
  375:                    block = newblock;
  376:            }
  377:            brelse(buf);
  378:            if(level == EXT2_TIND_BLOCK) {
  379:                    brelse(buf3);
  380:            }
```

fs/ext2/inode.c


```
  381:         brelse(buf2);
  382:         return block;
  383: }
  384:
  385: int ext2_truncate(struct inode *i, __off_t length)
  386: {
  387:         __blk_t block, dblock, *indblock;
  388:         struct buffer *buf;
  389:         int blksize, n;
  390:
  391:         blksize = i->sb->s_blocksize;
  392:         block = length / blksize;
  393:
  394:         if(!S_ISDIR(i->i_mode) && !S_ISREG(i->i_mode) && !S_ISLNK(i->i_mode)) {
  395:                 return -EINVAL;
  396:         }
  397:
  398:         if(block < EXT2_NDIR_BLOCKS) {
  399:                 for(n = block; n < EXT2_NDIR_BLOCKS; n++) {
  400:                         if(i->u.ext2.i_data[n]) {
  401:                                 ext2_bfree(i->sb, i->u.ext2.i_data[n]);
  402:                                 i->u.ext2.i_data[n] = 0;
  403:                                 i->i_blocks -= blksize / 512;
  404:                         }
  405:                 }
  406:                 block = 0;
  407:         }
  408:
  409:         if(!block || block < (BLOCKS_PER_IND_BLOCK(i->sb) + EXT2_NDIR_BLOCKS)) {
  410:                 if(block) {
  411:                         block -= EXT2_NDIR_BLOCKS;
  412:                 }
  413:                 if(i->u.ext2.i_data[EXT2_IND_BLOCK]) {
  414:                         free_indblock(i, i->u.ext2.i_data[EXT2_IND_BLOCK], block
);
  415:                         if(!block) {
  416:                                 ext2_bfree(i->sb, i->u.ext2.i_data[EXT2_IND_BLOC
K]);
  417:                                 i->u.ext2.i_data[EXT2_IND_BLOCK] = 0;
  418:                                 i->i_blocks -= blksize / 512;
  419:                         }
  420:                 }
  421:                 block = 0;
  422:         }
  423:
  424:         if(block) {
  425:                 block -= EXT2_NDIR_BLOCKS;
  426:                 block -= BLOCKS_PER_IND_BLOCK(i->sb);
  427:         }
  428:         if(i->u.ext2.i_data[EXT2_DIND_BLOCK]) {
  429:                 if(!(buf = bread(i->dev, i->u.ext2.i_data[EXT2_DIND_BLOCK], blks
ize))) {
  430:                         printk("%s(): error reading block %d.\n", __FUNCTION__,
i->u.ext2.i_data[EXT2_DIND_BLOCK]);
  431:                         return -EIO;
  432:                 }
  433:                 indblock = (__blk_t *)buf->data;
  434:                 dblock = block % BLOCKS_PER_IND_BLOCK(i->sb);
  435:                 for(n = block / BLOCKS_PER_IND_BLOCK(i->sb); n < BLOCKS_PER_IND_
BLOCK(i->sb); n++) {
  436:                         if(indblock[n]) {
  437:                                 free_indblock(i, indblock[n], dblock);
  438:                                 if(!dblock) {
  439:                                         ext2_bfree(i->sb, indblock[n]);
  440:                                         i->i_blocks -= blksize / 512;
  441:                                 }
  442:                         }
```

fs/ext2/inode.c


```
443:                            dblock = 0;
444:                    }
445:                    bwrite(buf);
446:                    if(!block) {
447:                            ext2_bfree(i->sb, i->u.ext2.i_data[EXT2_DIND_BLOCK]);
448:                            i->u.ext2.i_data[EXT2_DIND_BLOCK] = 0;
449:                            i->i_blocks -= blksize / 512;
450:                    }
451:            }
452:
453:            i->i_mtime = CURRENT_TIME;
454:            i->i_ctime = CURRENT_TIME;
455:            i->i_size = length;
456:            i->dirty = 1;
457:
458:            return 0;
459: }
```

fs/ext2/Makefile


```
 1: # fiwix/fs/ext2/Makefile
 2: #
 3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
 4: # Distributed under the terms of the Fiwix License.
 5: #
 6:
 7: .S.o:
 8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
 9: .c.o:
10:         $(CC) $(CFLAGS) -c -o $@ $<
11:
12: OBJS = inode.o super.o namei.o symlink.o dir.o file.o bitmaps.o
13:
14: ext2:   $(OBJS)
15:         $(LD) $(LDFLAGS) -r $(OBJS) -o ext2.o
16:
17: clean:
18:         rm -f *.o
19:
```

fs/ext2/namei.c


```
    1: /*
    2:  * fiwix/fs/ext2/namei.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/types.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/filesystems.h>
   12: #include <fiwix/fs_ext2.h>
   13: #include <fiwix/buffer.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/errno.h>
   16: #include <fiwix/stat.h>
   17: #include <fiwix/stdio.h>
   18: #include <fiwix/string.h>
   19:
   20: /* finds the entry 'name' with (optionally) inode 'i' in the directory 'dir' */
   21: static struct buffer * find_dir_entry(struct inode *dir, struct inode *i, struct
 ext2_dir_entry_2 **d_res, char *name)
   22: {
   23:         __blk_t block;
   24:         unsigned int blksize;
   25:         unsigned int offset, doffset;
   26:         struct buffer *buf;
   27:         int basesize, rlen, nlen;
   28:
   29:         basesize = sizeof((*d_res)->inode) + sizeof((*d_res)->rec_len) + sizeof(
(*d_res)->name_len) + sizeof((*d_res)->file_type);
   30:         blksize = dir->sb->s_blocksize;
   31:         offset = 0;
   32:
   33:         /* nlen is the length of the new entry, used when searching for the firs
t usable entry */
   34:         nlen = basesize + strlen(name) + 3;
   35:         nlen &= ~3;
   36:
   37:         while(offset < dir->i_size) {
   38:                 if((block = bmap(dir, offset, FOR_READING)) < 0) {
   39:                         break;
   40:                 }
   41:                 if(block) {
   42:                         if(!(buf = bread(dir->dev, block, blksize))) {
   43:                                 break;
   44:                         }
   45:                         doffset = 0;
   46:                         do {
   47:                                 *d_res = (struct ext2_dir_entry_2 *)(buf->data +
 doffset);
   48:                                 if(!i) {
   49:                                         /* calculates the real length of the cur
rent entry */
   50:                                         rlen = basesize + strlen((*d_res)->name)
 + 3;
   51:                                         rlen &= ~3;
   52:                                         /* returns the first entry where *name c
an fit in */
   53:                                         if(!(*d_res)->inode) {
   54:                                                 if(nlen <= (*d_res)->rec_len) {
   55:                                                         return buf;
   56:                                                 }
   57:                                         } else {
   58:                                                 if(rlen + nlen <= (*d_res)->rec_
len) {
   59:                                                         int nrec_len;
```

fs/ext2/namei.c


```
   60:
   61:                                                     nrec_len = (*d_res)->rec
_len - rlen;
   62:                                                     (*d_res)->rec_len = rlen
;
   63:                                                     doffset += (*d_res)->rec
_len;
   64:                                                     *d_res = (struct ext2_di
r_entry_2 *)(buf->data + doffset);
   65:                                                     (*d_res)->rec_len = nrec
_len;
   66:                                                     return buf;
   67:                                             }
   68:                                     }
   69:                                     doffset += (*d_res)->rec_len;
   70:                             } else {
   71:                                     if((*d_res)->inode == i->inode) {
   72:                                     /* returns the first matching in
ode */
   73:                                     if(!name) {
   74:                                             return buf;
   75:                                     }
   76:                                     /* returns the matching inode an
d name */
   77:                                     if((*d_res)->name_len == strlen(
name)) {
   78:                                             if(!strncmp((*d_res)->na
me, name, (*d_res)->name_len)) {
   79:                                                     return buf;
   80:                                             }
   81:                                     }
   82:                                     }
   83:                                     doffset += (*d_res)->rec_len;
   84:                             }
   85:                     } while(doffset < blksize);
   86:                     brelse(buf);
   87:                     offset += blksize;
   88:             } else {
   89:                     break;
   90:             }
   91:     }
   92:
   93:     *d_res = NULL;
   94:     return NULL;
   95: }
   96:
   97: static struct buffer * add_dir_entry(struct inode *dir, struct ext2_dir_entry_2
**d_res, char *name)
   98: {
   99:     __blk_t block;
  100:     struct buffer *buf;
  101:
  102:     if(!(buf = find_dir_entry(dir, NULL, d_res, name))) {
  103:             if((block = bmap(dir, dir->i_size, FOR_WRITING)) < 0) {
  104:                     return NULL;
  105:             }
  106:             if(!(buf = bread(dir->dev, block, dir->sb->s_blocksize))) {
  107:                     return NULL;
  108:             }
  109:             *d_res = (struct ext2_dir_entry_2 *)buf->data;
  110:             dir->i_size += dir->sb->s_blocksize;
  111:             (*d_res)->rec_len = dir->sb->s_blocksize;
  112:     }
  113:
  114:     return buf;
  115: }
  116:
```

fs/ext2/namei.c


```
  117: static int is_dir_empty(struct inode *dir)
  118: {
  119:         __blk_t block;
  120:         unsigned int blksize;
  121:         unsigned int offset, doffset;
  122:         struct buffer *buf;
  123:         struct ext2_dir_entry_2 *d;
  124:
  125:         blksize = dir->sb->s_blocksize;
  126:         offset = 0;
  127:
  128:         while(offset < dir->i_size) {
  129:                 if((block = bmap(dir, offset, FOR_READING)) < 0) {
  130:                         break;
  131:                 }
  132:                 if(block) {
  133:                         if(!(buf = bread(dir->dev, block, blksize))) {
  134:                                 break;
  135:                         }
  136:                         doffset = 0;
  137:                         do {
  138:                                 if(doffset + offset >= dir->i_size) {
  139:                                         break;
  140:                                 }
  141:                                 d = (struct ext2_dir_entry_2 *)(buf->data + doff
set);
  142:                                 doffset += d->rec_len;
  143:                                 if(d->inode && d->name_len == 1 && d->name[0] ==
 '.') {
  144:                                         continue;
  145:                                 }
  146:                                 if(d->inode && d->name_len == 2 && d->name[0] ==
 '.' && d->name[1] == '.') {
  147:                                         continue;
  148:                                 }
  149:                                 if(d->inode) {
  150:                                         brelse(buf);
  151:                                         return 0;
  152:                                 }
  153:                         } while(doffset < blksize);
  154:                         brelse(buf);
  155:                         offset += blksize;
  156:                 } else {
  157:                         break;
  158:                 }
  159:         }
  160:
  161:         return 1;
  162: }
  163:
  164: static int is_prefix(struct inode *dir_new, struct inode *i_old)
  165: {
  166:         __ino_t inode;
  167:         int errno;
  168:
  169:         errno = 0;
  170:         for(;;) {
  171:                 if(dir_new == i_old) {
  172:                         errno = 1;
  173:                         break;
  174:                 }
  175:                 inode = dir_new->inode;
  176:                 if(ext2_lookup("..", dir_new, &dir_new)) {
  177:                         break;
  178:                 }
  179:                 iput(dir_new);  /* lookup eats 1 dir_new */
  180:                 if(dir_new->inode == inode) {
```

fs/ext2/namei.c


```
  181:                          break;
  182:                  }
  183:          }
  184:          return errno;
  185: }
  186:
  187: int ext2_lookup(const char *name, struct inode *dir, struct inode **i_res)
  188: {
  189:          __blk_t block;
  190:          unsigned int blksize;
  191:          unsigned int offset, doffset;
  192:          struct buffer *buf;
  193:          struct ext2_dir_entry_2 *d;
  194:          __ino_t inode;
  195:
  196:          blksize = dir->sb->s_blocksize;
  197:          inode = offset = 0;
  198:          dir->count++;
  199:
  200:          while(offset < dir->i_size && !inode) {
  201:                  if((block = bmap(dir, offset, FOR_READING)) < 0) {
  202:                          return block;
  203:                  }
  204:                  if(block) {
  205:                          if(!(buf = bread(dir->dev, block, blksize))) {
  206:                                  return -EIO;
  207:                          }
  208:                          doffset = 0;
  209:                          do {
  210:                                  d = (struct ext2_dir_entry_2 *)(buf->data + doff
set);
  211:                                  if(d->inode) {
  212:                                          if(d->name_len == strlen(name)) {
  213:                                                  if(strncmp(d->name, name, d->nam
e_len) == 0) {
  214:                                                          inode = d->inode;
  215:                                                  }
  216:                                          }
  217:                                  }
  218:                                  doffset += d->rec_len;
  219:                          } while((doffset < blksize) && (!inode));
  220:
  221:                          brelse(buf);
  222:                          offset += blksize;
  223:                          if(inode) {
  224:                                  /*
  225:                                   * This prevents a deadlock in iget() when
  226:                                   * trying to lock '.' when 'dir' is the same
  227:                                   * directory (ls -lai <dir>).
  228:                                   */
  229:                                  if(inode == dir->inode) {
  230:                                          *i_res = dir;
  231:                                          return 0;
  232:                                  }
  233:
  234:                                  if(!(*i_res = iget(dir->sb, inode))) {
  235:                                          return -EACCES;
  236:                                  }
  237:                                  iput(dir);
  238:                                  return 0;
  239:                          }
  240:                  } else {
  241:                          break;
  242:                  }
  243:          }
  244:          iput(dir);
  245:          return -ENOENT;
```

fs/ext2/namei.c


```
246: }
247:
248: int ext2_rmdir(struct inode *dir, struct inode *i)
249: {
250:         struct buffer *buf;
251:         struct ext2_dir_entry_2 *d;
252:
253:         inode_lock(i);
254:
255:         if(!is_dir_empty(i)) {
256:                 inode_unlock(i);
257:                 return -ENOTEMPTY;
258:         }
259:
260:         inode_lock(dir);
261:
262:         if(!(buf = find_dir_entry(dir, i, &d, NULL))) {
263:                 inode_unlock(i);
264:                 inode_unlock(dir);
265:                 return -ENOENT;
266:         }
267:
268:         d->inode = 0;
269:         i->i_nlink = 0;
270:         dir->i_nlink--;
271:
272:         i->i_ctime = CURRENT_TIME;
273:         i->u.ext2.i_dtime = CURRENT_TIME;
274:         dir->i_mtime = CURRENT_TIME;
275:         dir->i_ctime = CURRENT_TIME;
276:
277:         i->dirty = 1;
278:         dir->dirty = 1;
279:
280:         bwrite(buf);
281:
282:         inode_unlock(i);
283:         inode_unlock(dir);
284:         return 0;
285: }
286:
287: int ext2_link(struct inode *i_old, struct inode *dir_new, char *name)
288: {
289:         struct buffer *buf;
290:         struct ext2_dir_entry_2 *d;
291:         char c;
292:         int n;
293:
294:         inode_lock(i_old);
295:         inode_lock(dir_new);
296:
297:         if(!(buf = add_dir_entry(dir_new, &d, name))) {
298:                 inode_unlock(i_old);
299:                 inode_unlock(dir_new);
300:                 return -ENOSPC;
301:         }
302:
303:         d->inode = i_old->inode;
304:         d->name_len = strlen(name);
305:         /* strcpy() can't be used here because it places a trailing NULL */
306:         for(n = 0; n < NAME_MAX; n++) {
307:                 if((c = name[n])) {
308:                         d->name[n] = c;
309:                         continue;
310:                 }
311:                 break;
312:         }
```

fs/ext2/namei.c


```
313:         d->file_type = 0;       /* not used */
314:
315:         i_old->i_nlink++;
316:         i_old->i_ctime = CURRENT_TIME;
317:         dir_new->i_mtime = CURRENT_TIME;
318:         dir_new->i_ctime = CURRENT_TIME;
319:
320:         i_old->dirty = 1;
321:         dir_new->dirty = 1;
322:
323:         bwrite(buf);
324:
325:         inode_unlock(i_old);
326:         inode_unlock(dir_new);
327:         return 0;
328: }
329:
330: int ext2_unlink(struct inode *dir, struct inode *i, char *name)
331: {
332:         struct buffer *buf;
333:         struct ext2_dir_entry_2 *d;
334:
335:         inode_lock(dir);
336:         inode_lock(i);
337:
338:         if(!(buf = find_dir_entry(dir, i, &d, name))) {
339:                 inode_unlock(dir);
340:                 inode_unlock(i);
341:                 return -ENOENT;
342:         }
343:
344:         /*
345:          * FIXME: in order to avoid low performance when traversing large
346:          * directories plenty of blank entries, it would be interesting
347:          * to merge every removed entry with the previous entry.
348:          */
349:         d->inode = 0;
350:         if(!--i->i_nlink) {
351:                 i->u.ext2.i_dtime = CURRENT_TIME;
352:         }
353:
354:         i->i_ctime = CURRENT_TIME;
355:         dir->i_mtime = CURRENT_TIME;
356:         dir->i_ctime = CURRENT_TIME;
357:
358:         i->dirty = 1;
359:         dir->dirty = 1;
360:
361:         bwrite(buf);
362:
363:         inode_unlock(dir);
364:         inode_unlock(i);
365:         return 0;
366: }
367:
368: int ext2_symlink(struct inode *dir, char *name, char *oldname)
369: {
370:         struct buffer *buf, *buf2;
371:         struct inode *i;
372:         struct ext2_dir_entry_2 *d;
373:         __blk_t block;
374:         char c, *data;
375:         int n;
376:
377:         inode_lock(dir);
378:
379:         if(!(i = ialloc(dir->sb, S_IFLNK))) {
```

fs/ext2/namei.c


```
380:                    inode_unlock(dir);
381:                    return -ENOSPC;
382:            }
383:
384:            if(!(buf = add_dir_entry(dir, &d, name))) {
385:                    iput(i);
386:                    inode_unlock(dir);
387:                    return -ENOSPC;
388:            }
389:
390:            i->i_mode = S_IFLNK | (S_IRWXU | S_IRWXG | S_IRWXO);
391:            i->i_uid = current->euid;
392:            i->i_gid = current->egid;
393:            i->dev = dir->dev;
394:            i->count = 1;
395:            i->fsop = &ext2_symlink_fsop;
396:
397:            if(strlen(oldname) >= EXT2_N_BLOCKS * sizeof(__u32)) {
398:                    /* this will be a slow symlink */
399:                    if((block = ext2_balloc(dir->sb)) < 0) {
400:                            iput(i);
401:                            brelse(buf);
402:                            inode_unlock(dir);
403:                            return block;
404:                    }
405:                    if(!(buf2 = bread(dir->dev, block, dir->sb->s_blocksize))) {
406:                            iput(i);
407:                            brelse(buf);
408:                            ext2_bfree(dir->sb, block);
409:                            inode_unlock(dir);
410:                            return -EIO;
411:                    }
412:                    i->u.ext2.i_data[0] = block;
413:                    for(n = 0; n < NAME_MAX; n++) {
414:                            if((c = oldname[n])) {
415:                                    buf2->data[n] = c;
416:                                    continue;
417:                            }
418:                            break;
419:                    }
420:                    buf2->data[n] = 0;
421:                    i->i_blocks = dir->sb->s_blocksize / 512;
422:                    bwrite(buf2);
423:            } else {
424:                    /* this will be a fast symlink */
425:                    data = (char *)i->u.ext2.i_data;
426:                    for(n = 0; n < NAME_MAX; n++) {
427:                            if((c = oldname[n])) {
428:                                    data[n] = c;
429:                                    continue;
430:                            }
431:                            break;
432:                    }
433:                    data[n] = 0;
434:            }
435:
436:            i->i_size = n;
437:            i->dirty = 1;
438:            i->i_nlink = 1;
439:            d->inode = i->inode;
440:            d->name_len = strlen(name);
441:            /* strcpy() can't be used here because it places a trailing NULL */
442:            for(n = 0; n < NAME_MAX; n++) {
443:                    if((c = name[n])) {
444:                            d->name[n] = c;
445:                            continue;
446:                    }
```

fs/ext2/namei.c


```
447:                    break;
448:            }
449:            d->file_type = 0;       /* EXT2_FT_SYMLINK not used */
450:
451:            dir->i_mtime = CURRENT_TIME;
452:            dir->i_ctime = CURRENT_TIME;
453:            dir->dirty = 1;
454:
455:            bwrite(buf);
456:            iput(i);
457:            inode_unlock(dir);
458:            return 0;
459: }
460:
461: int ext2_mkdir(struct inode *dir, char *name, __mode_t mode)
462: {
463:            struct buffer *buf, *buf2;
464:            struct inode *i;
465:            struct ext2_dir_entry_2 *d, *d2;
466:            __blk_t block;
467:            char c;
468:            int n;
469:
470:            inode_lock(dir);
471:
472:            if(!(i = ialloc(dir->sb, S_IFDIR))) {
473:                    inode_unlock(dir);
474:                    return -ENOSPC;
475:            }
476:
477:            i->i_mode = ((mode & (S_IRWXU | S_IRWXG | S_IRWXO)) & ~current->umask);
478:            i->i_mode |= S_IFDIR;
479:            i->i_uid = current->euid;
480:            i->i_gid = current->egid;
481:            i->dev = dir->dev;
482:            i->count = 1;
483:            i->fsop = &ext2_dir_fsop;
484:
485:            if((block = bmap(i, 0, FOR_WRITING)) < 0) {
486:                    iput(i);
487:                    inode_unlock(dir);
488:                    return block;
489:            }
490:
491:            if(!(buf2 = bread(i->dev, block, dir->sb->s_blocksize))) {
492:                    ext2_bfree(dir->sb, block);
493:                    iput(i);
494:                    inode_unlock(dir);
495:                    return -EIO;
496:            }
497:
498:            if(!(buf = add_dir_entry(dir, &d, name))) {
499:                    ext2_bfree(dir->sb, block);
500:                    iput(i);
501:                    brelse(buf2);
502:                    inode_unlock(dir);
503:                    return -ENOSPC;
504:            }
505:
506:            d->inode = i->inode;
507:            d->name_len = strlen(name);
508:            /* strcpy() can't be used here because it places a trailing NULL */
509:            for(n = 0; n < NAME_MAX; n++) {
510:                    if((c = name[n])) {
511:                            if(c != '/') {
512:                                    d->name[n] = c;
513:                                    continue;
```

fs/ext2/namei.c

```
514:                            }
515:                    }
516:                    break;
517:            }
518:            d->file_type = 0;       /* EXT2_FT_DIR not used */
519:
520:            d2 = (struct ext2_dir_entry_2 *)buf2->data;
521:            d2->inode = i->inode;
522:            d2->name[0] = '.';
523:            d2->name[1] = 0;
524:            d2->name_len = 1;
525:            d2->rec_len = 12;
526:            d2->file_type = 0;      /* EXT2_FT_DIR not used */
527:            i->i_nlink = 1;
528:            d2 = (struct ext2_dir_entry_2 *)(buf2->data + 12);
529:            d2->inode = dir->inode;
530:            d2->name[0] = '.';
531:            d2->name[1] = '.';
532:            d2->name[2] = 0;
533:            d2->name_len = 2;
534:            d2->rec_len = i->sb->s_blocksize - 12;
535:            d2->file_type = 0;      /* EXT2_FT_DIR not used */
536:            i->i_nlink++;
537:            i->i_size = i->sb->s_blocksize;
538:            i->i_blocks = dir->sb->s_blocksize / 512;
539:            i->dirty = 1;
540:
541:            dir->i_mtime = CURRENT_TIME;
542:            dir->i_ctime = CURRENT_TIME;
543:            dir->i_nlink++;
544:            dir->dirty = 1;
545:
546:            bwrite(buf);
547:            bwrite(buf2);
548:            iput(i);
549:            inode_unlock(dir);
550:            return 0;
551: }
552:
553: int ext2_mknod(struct inode *dir, char *name, __mode_t mode, __dev_t dev)
554: {
555:            struct buffer *buf;
556:            struct inode *i;
557:            struct ext2_dir_entry_2 *d;
558:            char c;
559:            int n;
560:
561:            inode_lock(dir);
562:
563:            if(!(i = ialloc(dir->sb, mode & S_IFMT))) {
564:                    inode_unlock(dir);
565:                    return -ENOSPC;
566:            }
567:
568:            if(!(buf = add_dir_entry(dir, &d, name))) {
569:                    i->i_nlink = 0;
570:                    iput(i);
571:                    inode_unlock(dir);
572:                    return -ENOSPC;
573:            }
574:
575:            d->inode = i->inode;
576:            d->name_len = strlen(name);
577:            /* strcpy() can't be used here because it places a trailing NULL */
578:            for(n = 0; n < NAME_MAX; n++) {
579:                    if((c = name[n])) {
580:                            d->name[n] = c;
```

fs/ext2/namei.c


```
  581:                          continue;
  582:                  }
  583:                  break;
  584:          }
  585:
  586:          i->i_mode = (mode & ~current->umask) & ~S_IFMT;
  587:          i->i_uid = current->euid;
  588:          i->i_gid = current->egid;
  589:          i->i_nlink = 1;
  590:          i->dev = dir->dev;
  591:          i->count = 1;
  592:          i->dirty = 1;
  593:
  594:          switch(mode & S_IFMT) {
  595:                  case S_IFCHR:
  596:                          i->fsop = &def_chr_fsop;
  597:                          i->rdev = dev;
  598:                          i->i_mode |= S_IFCHR;
  599:                          d->file_type = 0;       /* EXT2_FT_CHRDEV not used */
  600:                          break;
  601:                  case S_IFBLK:
  602:                          i->fsop = &def_blk_fsop;
  603:                          i->rdev = dev;
  604:                          i->i_mode |= S_IFBLK;
  605:                          d->file_type = 0;       /* EXT2_FT_BLKDEV not used */
  606:                          break;
  607:                  case S_IFIFO:
  608:                          i->fsop = &pipefs_fsop;
  609:                          i->i_mode |= S_IFIFO;
  610:                          /* it's a union so we need to clear pipefs_i */
  611:                          memset_b(&i->u.pipefs, NULL, sizeof(struct pipefs_inode)
);
  612:                          d->file_type = 0;       /* EXT2_FT_FIFO not used */
  613:                          break;
  614:          }
  615:
  616:          dir->i_mtime = CURRENT_TIME;
  617:          dir->i_ctime = CURRENT_TIME;
  618:          dir->dirty = 1;
  619:
  620:          bwrite(buf);
  621:          iput(i);
  622:          inode_unlock(dir);
  623:          return 0;
  624: }
  625:
  626: int ext2_create(struct inode *dir, char *name, __mode_t mode, struct inode **i_r
es)
  627: {
  628:          struct buffer *buf;
  629:          struct inode *i;
  630:          struct ext2_dir_entry_2 *d;
  631:          char c;
  632:          int n;
  633:
  634:          if(IS_RDONLY_FS(dir)) {
  635:                  return -EROFS;
  636:          }
  637:
  638:          inode_lock(dir);
  639:
  640:          if(!(i = ialloc(dir->sb, S_IFREG))) {
  641:                  inode_unlock(dir);
  642:                  return -ENOSPC;
  643:          }
  644:
  645:          if(!(buf = add_dir_entry(dir, &d, name))) {
```

fs/ext2/namei.c

```
  646:                  i->i_nlink = 0;
  647:                  iput(i);
  648:                  inode_unlock(dir);
  649:                  return -ENOSPC;
  650:          }
  651:
  652:          d->inode = i->inode;
  653:          d->name_len = strlen(name);
  654:          /* strcpy() can't be used here because it places a trailing NULL */
  655:          for(n = 0; n < NAME_MAX; n++) {
  656:                  if((c = name[n])) {
  657:                          d->name[n] = c;
  658:                          continue;
  659:                  }
  660:                  break;
  661:          }
  662:          d->file_type = 0;       /* EXT2_FT_REG_FILE not used */
  663:
  664:          i->i_mode = (mode & ~current->umask) & ~S_IFMT;
  665:          i->i_mode |= S_IFREG;
  666:          i->i_uid = current->euid;
  667:          i->i_gid = current->egid;
  668:          i->i_nlink = 1;
  669:          i->i_blocks = 0;
  670:          i->dev = dir->dev;
  671:          i->fsop = &ext2_file_fsop;
  672:          i->count = 1;
  673:          i->dirty = 1;
  674:
  675:          i->u.ext2.i_dtime = 0;
  676:
  677:          dir->i_mtime = CURRENT_TIME;
  678:          dir->i_ctime = CURRENT_TIME;
  679:          dir->dirty = 1;
  680:
  681:          *i_res = i;
  682:          bwrite(buf);
  683:          inode_unlock(dir);
  684:          return 0;
  685: }
  686:
  687: int ext2_rename(struct inode *i_old, struct inode *dir_old, struct inode *i_new,
struct inode *dir_new, char *oldpath, char *newpath)
  688: {
  689:          struct buffer *buf_old, *buf_new;
  690:          struct ext2_dir_entry_2 *d_old, *d_new;
  691:          char c;
  692:          int n, errno;
  693:
  694:          errno = 0;
  695:          buf_new = NULL;
  696:
  697:          if(is_prefix(dir_new, i_old)) {
  698:                  return -EINVAL;
  699:          }
  700:
  701:          inode_lock(i_old);
  702:          inode_lock(dir_old);
  703:          if(dir_old != dir_new) {
  704:                  inode_lock(dir_new);
  705:          }
  706:
  707:          if(!(buf_old = find_dir_entry(dir_old, i_old, &d_old, oldpath))) {
  708:                  errno = -ENOENT;
  709:                  goto end;
  710:          }
  711:          if(dir_old == dir_new) {
```

fs/ext2/namei.c


```
 712:                   buf_old->locked = 0;
 713:           }
 714:
 715:           if(i_new) {
 716:                   if(S_ISDIR(i_old->i_mode)) {
 717:                           if(!is_dir_empty(i_new)) {
 718:                                   brelse(buf_old);
 719:                                   errno = -ENOTEMPTY;
 720:                                   goto end;
 721:                           }
 722:                   }
 723:                   if(!(buf_new = find_dir_entry(dir_new, i_new, &d_new, newpath)))
{
 724:                           brelse(buf_old);
 725:                           errno = -ENOENT;
 726:                           goto end;
 727:                   }
 728:           } else {
 729:                   if(!(buf_new = add_dir_entry(dir_new, &d_new, newpath))) {
 730:                           brelse(buf_old);
 731:                           errno = -ENOSPC;
 732:                           goto end;
 733:                   }
 734:           }
 735:           if(i_new) {
 736:                   i_new->i_nlink--;
 737:           } else {
 738:                   i_new = i_old;
 739:                   d_new->name_len = strlen(newpath);
 740:                   /* strcpy() can't be used here because it places a trailing NULL
*/
 741:                   for(n = 0; n < NAME_MAX; n++) {
 742:                           if((c = newpath[n])) {
 743:                                   d_new->name[n] = c;
 744:                                   continue;
 745:                           }
 746:                           break;
 747:                   }
 748:           }
 749:
 750:           d_old->inode = 0;
 751:           d_new->inode = i_old->inode;
 752:           dir_new->i_mtime = CURRENT_TIME;
 753:           dir_new->i_ctime = CURRENT_TIME;
 754:           i_new->dirty = 1;
 755:           dir_new->dirty = 1;
 756:
 757:           dir_old->i_mtime = CURRENT_TIME;
 758:           dir_old->i_ctime = CURRENT_TIME;
 759:           i_old->dirty = 1;
 760:           dir_old->dirty = 1;
 761:
 762:           bwrite(buf_old);
 763:           if(buf_new) {
 764:                   bwrite(buf_new);
 765:           }
 766:
 767: end:
 768:           inode_unlock(i_old);
 769:           inode_unlock(dir_old);
 770:           inode_unlock(dir_new);
 771:           return errno;
 772: }
```

fs/ext2/super.c


```
    1: /*
    2:  * fiwix/fs/ext2/super.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/types.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/fs.h>
   12: #include <fiwix/filesystems.h>
   13: #include <fiwix/fs_ext2.h>
   14: #include <fiwix/buffer.h>
   15: #include <fiwix/sched.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: struct fs_operations ext2_fsop = {
   20:         FSOP_REQUIRES_DEV,
   21:         NULL,
   22:
   23:         NULL,                   /* open */
   24:         NULL,                   /* close */
   25:         NULL,                   /* read */
   26:         NULL,                   /* write */
   27:         NULL,                   /* ioctl */
   28:         NULL,                   /* lseek */
   29:         NULL,                   /* readdir */
   30:         NULL,                   /* mmap */
   31:         NULL,                   /* select */
   32:
   33:         NULL,                   /* readlink */
   34:         NULL,                   /* followlink */
   35:         NULL,                   /* bmap */
   36:         NULL,                   /* lookup */
   37:         NULL,                   /* rmdir */
   38:         NULL,                   /* link */
   39:         NULL,                   /* unlink */
   40:         NULL,                   /* symlink */
   41:         NULL,                   /* mkdir */
   42:         NULL,                   /* mknod */
   43:         NULL,                   /* truncate */
   44:         NULL,                   /* create */
   45:         NULL,                   /* rename */
   46:
   47:         NULL,                   /* read_block */
   48:         NULL,                   /* write_block */
   49:
   50:         ext2_read_inode,
   51:         ext2_write_inode,
   52:         ext2_ialloc,
   53:         ext2_ifree,
   54:         ext2_statfs,
   55:         ext2_read_superblock,
   56:         ext2_remount_fs,
   57:         ext2_write_superblock,
   58:         ext2_release_superblock
   59: };
   60:
   61: static void check_superblock(struct ext2_super_block *sb)
   62: {
   63:         if(!(sb->s_state & EXT2_VALID_FS)) {
   64:                 printk("WARNING: filesystem unchecked, fsck recommended.\n");
   65:         } else if((sb->s_state & EXT2_ERROR_FS)) {
   66:                 printk("WARNING: filesystem contains errors, fsck recommended.\n
");
```

fs/ext2/super.c


```
   67:         } else if(sb->s_max_mnt_count >= 0 && sb->s_mnt_count >= (unsigned short
 int)sb->s_max_mnt_count) {
   68:                 printk("WARNING: maximal mount count reached, fsck recommended.\
n");
   69:         } else if(sb->s_checkinterval && (sb->s_lastcheck + sb->s_checkinterval
<= CURRENT_TIME)) {
   70:                 printk("WARNING: checktime reached, fsck recommended.\n");
   71:         }
   72: }
   73:
   74: void ext2_statfs(struct superblock *sb, struct statfs *statfsbuf)
   75: {
   76:         statfsbuf->f_type = EXT2_SUPER_MAGIC;
   77:         statfsbuf->f_bsize = sb->s_blocksize;
   78:         statfsbuf->f_blocks = sb->u.ext2.sb.s_blocks_count;
   79:         statfsbuf->f_bfree = sb->u.ext2.sb.s_free_blocks_count;
   80:         if(statfsbuf->f_bfree >= sb->u.ext2.sb.s_r_blocks_count) {
   81:                 statfsbuf->f_bavail = statfsbuf->f_bfree - sb->u.ext2.sb.s_r_blo
cks_count;
   82:         } else {
   83:                 statfsbuf->f_bavail = 0;
   84:         }
   85:         statfsbuf->f_files = sb->u.ext2.sb.s_inodes_count;
   86:         statfsbuf->f_ffree = sb->u.ext2.sb.s_free_inodes_count;
   87:         /* statfsbuf->f_fsid = ? */
   88:         statfsbuf->f_namelen = EXT2_NAME_LEN;
   89: }
   90:
   91: int ext2_read_superblock(__dev_t dev, struct superblock *sb)
   92: {
   93:         struct buffer *buf;
   94:         struct ext2_super_block *ext2sb;
   95:
   96:         superblock_lock(sb);
   97:         if(!(buf = bread(dev, SUPERBLOCK, BLKSIZE_1K))) {
   98:                 printk("WARNING: %s(): I/O error on device %d,%d.\n", __FUNCTION
__, MAJOR(dev), MINOR(dev));
   99:                 superblock_unlock(sb);
  100:                 return -EIO;
  101:         }
  102:
  103:         ext2sb = (struct ext2_super_block *)buf->data;
  104:         if(ext2sb->s_magic != EXT2_SUPER_MAGIC) {
  105:                 printk("WARNING: %s(): invalid filesystem type or bad superblock
 on device %d,%d.\n", __FUNCTION__, MAJOR(dev), MINOR(dev));
  106:                 superblock_unlock(sb);
  107:                 brelse(buf);
  108:                 return -EINVAL;
  109:         }
  110:
  111:         if(ext2sb->s_minor_rev_level || ext2sb->s_rev_level) {
  112:                 printk("WARNING: %s(): unsupported ext2 filesystem revision.\n",
 __FUNCTION__);
  113:                 printk("Only revision 0 (original without features) is supported
.\n");
  114:                 superblock_unlock(sb);
  115:                 brelse(buf);
  116:                 return -EINVAL;
  117:         }
  118:
  119:         sb->dev = dev;
  120:         sb->fsop = &ext2_fsop;
  121:         sb->s_blocksize = EXT2_MIN_BLOCK_SIZE << ext2sb->s_log_block_size;
  122:         memcpy_b(&sb->u.ext2.sb, ext2sb, sizeof(struct ext2_super_block));
  123:         EXT2_DESC_PER_BLOCK(sb) = sb->s_blocksize / sizeof(struct ext2_group_des
c);
  124:         sb->u.ext2.block_groups = 1 + (ext2sb->s_blocks_count - 1) / ext2sb->s_b
```

fs/ext2/super.c


```
locks_per_group;
  125:
  126:          if(!(sb->root = iget(sb, EXT2_ROOT_INO))) {
  127:                  printk("WARNING: %s(): unable to get root inode.\n", __FUNCTION_
_);
  128:                  superblock_unlock(sb);
  129:                  brelse(buf);
  130:                  return -EINVAL;
  131:          }
  132:
  133:          check_superblock(ext2sb);
  134:          if(!(sb->flags & MS_RDONLY)) {
  135:                  sb->u.ext2.sb.s_state &= ~EXT2_VALID_FS;
  136:                  sb->u.ext2.sb.s_mnt_count++;
  137:                  sb->u.ext2.sb.s_mtime = CURRENT_TIME;
  138:                  memcpy_b(buf->data, &sb->u.ext2.sb, sizeof(struct ext2_super_blo
ck));
  139:                  bwrite(buf);
  140:          } else {
  141:                  brelse(buf);
  142:          }
  143:          superblock_unlock(sb);
  144:          return 0;
  145: }
  146:
  147: int ext2_remount_fs(struct superblock *sb, int flags)
  148: {
  149:          struct buffer *buf;
  150:          struct ext2_super_block *ext2sb;
  151:
  152:          if((flags & MS_RDONLY) == (sb->flags & MS_RDONLY)) {
  153:                  return 0;
  154:          }
  155:
  156:          superblock_lock(sb);
  157:          if(!(buf = bread(sb->dev, SUPERBLOCK, BLKSIZE_1K))) {
  158:                  superblock_unlock(sb);
  159:                  return -EIO;
  160:          }
  161:          ext2sb = (struct ext2_super_block *)buf->data;
  162:
  163:          if(flags & MS_RDONLY) {
  164:                  /* switching from RW to RO */
  165:                  sb->u.ext2.sb.s_state |= EXT2_VALID_FS;
  166:                  ext2sb->s_state |= EXT2_VALID_FS;
  167:          } else {
  168:                  /* switching from RO to RW */
  169:                  check_superblock(ext2sb);
  170:                  sb->u.ext2.sb.s_state &= ~EXT2_VALID_FS;
  171:                  sb->u.ext2.sb.s_mnt_count++;
  172:                  sb->u.ext2.sb.s_mtime = CURRENT_TIME;
  173:                  ext2sb->s_state &= ~EXT2_VALID_FS;
  174:          }
  175:
  176:          sb->dirty = 1;
  177:          superblock_unlock(sb);
  178:          bwrite(buf);
  179:          return 0;
  180: }
  181:
  182: int ext2_write_superblock(struct superblock *sb)
  183: {
  184:          struct buffer *buf;
  185:
  186:          superblock_lock(sb);
  187:          if(!(buf = bread(sb->dev, SUPERBLOCK, BLKSIZE_1K))) {
  188:                  superblock_unlock(sb);
```

fs/ext2/super.c


```
189:                    return -EIO;
190:            }
191:
192:            memcpy_b(buf->data, &sb->u.ext2.sb, sizeof(struct ext2_super_block));
193:            sb->dirty = 0;
194:            superblock_unlock(sb);
195:            bwrite(buf);
196:            return 0;
197: }
198:
199: void ext2_release_superblock(struct superblock *sb)
200: {
201:            if(sb->flags & MS_RDONLY) {
202:                    return;
203:            }
204:
205:            superblock_lock(sb);
206:
207:            sb->u.ext2.sb.s_state |= EXT2_VALID_FS;
208:            sb->dirty = 1;
209:
210:            superblock_unlock(sb);
211: }
212:
213: int ext2_init(void)
214: {
215:            return register_filesystem("ext2", &ext2_fsop);
216: }
```

fs/ext2/symlink.c


```
    1: /*
    2:  * fiwix/fs/ext2/symlink.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/errno.h>
   10: #include <fiwix/buffer.h>
   11: #include <fiwix/fs.h>
   12: #include <fiwix/filesystems.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/stdio.h>
   16: #include <fiwix/string.h>
   17:
   18: struct fs_operations ext2_symlink_fsop = {
   19:         0,
   20:         0,
   21:
   22:         NULL,                   /* open */
   23:         NULL,                   /* close */
   24:         NULL,                   /* read */
   25:         NULL,                   /* write */
   26:         NULL,                   /* ioctl */
   27:         NULL,                   /* lseek */
   28:         NULL,                   /* readdir */
   29:         NULL,                   /* mmap */
   30:         NULL,                   /* select */
   31:
   32:         ext2_readlink,
   33:         ext2_followlink,
   34:         NULL,                   /* bmap */
   35:         NULL,                   /* lookup */
   36:         NULL,                   /* rmdir */
   37:         NULL,                   /* link */
   38:         NULL,                   /* unlink */
   39:         NULL,                   /* symlink */
   40:         NULL,                   /* mkdir */
   41:         NULL,                   /* mknod */
   42:         NULL,                   /* truncate */
   43:         NULL,                   /* create */
   44:         NULL,                   /* rename */
   45:
   46:         NULL,                   /* read_block */
   47:         NULL,                   /* write_block */
   48:
   49:         NULL,                   /* read_inode */
   50:         NULL,                   /* write_inode */
   51:         NULL,                   /* ialloc */
   52:         NULL,                   /* ifree */
   53:         NULL,                   /* statfs */
   54:         NULL,                   /* read_superblock */
   55:         NULL,                   /* remount_fs */
   56:         NULL,                   /* write_superblock */
   57:         NULL                    /* release_superblock */
   58: };
   59:
   60: int ext2_readlink(struct inode *i, char *buffer, __size_t count)
   61: {
   62:         __u32 blksize;
   63:         struct buffer *buf;
   64:
   65:         if(!S_ISLNK(i->i_mode)) {
   66:                 printk("%s(): Oops, inode '%d' is not a symlink (!?).\n", __FUNC
TION__, i->inode);
```

fs/ext2/symlink.c


```
   67:                     return 0;
   68:             }
   69:
   70:             inode_lock(i);
   71:             blksize = i->sb->s_blocksize;
   72:             count = MIN(count, i->i_size);
   73:             if(!count) {
   74:                     inode_unlock(i);
   75:                     return 0;
   76:             }
   77:             count = MIN(count, blksize);
   78:             if(i->i_blocks) {       /* slow symlink */
   79:                     if(!(buf = bread(i->dev, i->u.ext2.i_data[0], blksize))) {
   80:                             inode_unlock(i);
   81:                             return -EIO;
   82:                     }
   83:                     memcpy_b(buffer, buf->data, count);
   84:                     brelse(buf);
   85:             } else {        /* fast symlink */
   86:                     memcpy_b(buffer, (char *)i->u.ext2.i_data, count);
   87:             }
   88:             buffer[count] = NULL;
   89:             inode_unlock(i);
   90:             return count;
   91: }
   92:
   93: int ext2_followlink(struct inode *dir, struct inode *i, struct inode **i_res)
   94: {
   95:             struct buffer *buf;
   96:             char *name;
   97:             __ino_t errno;
   98:
   99:             if(!i) {
  100:                     return -ENOENT;
  101:             }
  102:
  103:             if(!S_ISLNK(i->i_mode)) {
  104:                     printk("%s(): Oops, inode '%d' is not a symlink (!?).\n", __FUNC
TION__, i->inode);
  105:                     return 0;
  106:             }
  107:
  108:             if(current->loopcnt > MAX_SYMLINKS) {
  109:                     printk("%s(): too many nested symbolic links!\n", __FUNCTION__);
  110:                     return -ELOOP;
  111:             }
  112:
  113:             inode_lock(i);
  114:             if(i->i_blocks) {       /* slow symlink */
  115:                     if(!(buf = bread(i->dev, i->u.ext2.i_data[0], i->sb->s_blocksize
))) {
  116:                             inode_unlock(i);
  117:                             return -EIO;
  118:                     }
  119:                     name = buf->data;
  120:             } else {        /* fast symlink */
  121:                     buf = NULL;
  122:                     name = (char *)i->u.ext2.i_data;
  123:             }
  124:             inode_unlock(i);
  125:
  126:             current->loopcnt++;
  127:             iput(i);
  128:             if(buf) {
  129:                     brelse(buf);
  130:             }
  131:             errno = parse_namei(name, dir, i_res, NULL, FOLLOW_LINKS);
```

fs/ext2/symlink.c


```
132:         current->loopcnt--;
133:         return errno;
134: }
```

fs/iso9660/dir.c


```
/*
 * fiwix/fs/iso9660/dir.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/buffer.h>
#include <fiwix/fs.h>
#include <fiwix/filesystems.h>
#include <fiwix/stat.h>
#include <fiwix/dirent.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct fs_operations iso9660_dir_fsop = {
        0,
        0,

        iso9660_dir_open,
        iso9660_dir_close,
        iso9660_dir_read,
        NULL,                   /* write */
        NULL,                   /* ioctl */
        NULL,                   /* lseek */
        iso9660_dir_readdir,
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        iso9660_bmap,
        iso9660_lookup,
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        NULL,                   /* truncate */
        NULL,                   /* create */
        NULL,                   /* rename */

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statsfs */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

int iso9660_dir_open(struct inode *i, struct fd *fd_table)
{
        fd_table->offset = 0;
        return 0;
}

int iso9660_dir_close(struct inode *i, struct fd *fd_table)
{
```

fs/iso9660/dir.c


```
   68:         return 0;
   69: }
   70:
   71: int iso9660_dir_read(struct inode *i, struct fd *fd_table, char *buffer, __size_
t count)
   72: {
   73:         return -EISDIR;
   74: }
   75:
   76: int iso9660_dir_readdir(struct inode *i, struct fd *fd_table, struct dirent *dir
ent, unsigned int count)
   77: {
   78:         __blk_t block;
   79:         unsigned int doffset, offset;
   80:         unsigned int size, dirent_len;
   81:         struct iso9660_directory_record *d;
   82:         int base_dirent_len;
   83:         int blksize;
   84:         struct buffer *buf;
   85:         int nm_len;
   86:         char nm_name[NAME_MAX + 1];
   87:
   88:         if(!(S_ISDIR(i->i_mode))) {
   89:                 return -EBADF;
   90:         }
   91:
   92:         blksize = i->sb->s_blocksize;
   93:         if(fd_table->offset > i->i_size) {
   94:                 fd_table->offset = i->i_size;
   95:         }
   96:
   97:         base_dirent_len = sizeof(dirent->d_ino) + sizeof(dirent->d_off) + sizeof
(dirent->d_reclen);
   98:         doffset = offset = size = 0;
   99:
  100:         while(doffset < count) {
  101:                 if((block = bmap(i, fd_table->offset, FOR_READING)) < 0) {
  102:                         return block;
  103:                 }
  104:                 if(block) {
  105:                         if(!(buf = bread(i->dev, block, blksize))) {
  106:                                 return -EIO;
  107:                         }
  108:
  109:                         doffset = fd_table->offset;
  110:                         offset = fd_table->offset % blksize;
  111:
  112:                         while(doffset < i->i_size && offset < blksize) {
  113:                                 d = (struct iso9660_directory_record *)(buf->dat
a + offset);
  114:                                 if(isonum_711(d->length)) {
  115:                                         dirent_len = (base_dirent_len + (isonum_
711(d->name_len) + 1)) + 3;
  116:                                         dirent_len &= ~3;       /* round up */
  117:                                         if((size + dirent_len) < count) {
  118:                                                 dirent->d_ino = (block << ISO966
0_INODE_BITS) + (doffset & ISO9660_INODE_MASK);
  119:                                                 dirent->d_off = doffset;
  120:                                                 dirent->d_reclen = dirent_len;
  121:                                                 if(isonum_711(d->name_len) == 1
&& d->name[0] == 0) {
  122:                                                         dirent->d_name[0] = '.';
  123:                                                         dirent->d_name[1] = NULL
;
  124:                                                 } else if(isonum_711(d->name_len
) == 1 && d->name[0] == 1) {
  125:                                                         dirent->d_name[0] = '.';
```

fs/iso9660/dir.c


```
  126:                                                    dirent->d_name[1] = '.';
  127:                                                    dirent->d_name[2] = NULL
;
  128:                                                    dirent_len = 16;
  129:                                                    dirent->d_reclen = 16;
  130:                                                    if(i->u.iso9660.i_parent
) {
  131:                                                            dirent->d_ino =
i->u.iso9660.i_parent->inode;
  132:                                                    } else {
  133:                                                            dirent->d_ino =
i->inode;
  134:                                                    }
  135:                                            } else {
  136:                                                    nm_len = 0;
  137:                                                    if(i->sb->u.iso9660.rrip
) {
  138:                                                            nm_len = get_rri
p_filename(d, i, nm_name);
  139:                                                    }
  140:                                                    if(nm_len) {
  141:                                                            memcpy_b(dirent-
>d_name, nm_name, nm_len);
  142:                                                            dirent->d_name[n
m_len] = NULL;
  143:                                                            dirent->d_reclen
 = (base_dirent_len + nm_len + 1) + 3;
  144:                                                            dirent->d_reclen
 &= ~3; /* round up */
  145:                                                            dirent_len = dir
ent->d_reclen;
  146:                                                    } else {
  147:                                                            memcpy_b(dirent-
>d_name, d->name, isonum_711(d->name_len));
  148:                                                            dirent->d_name[i
sonum_711(d->name_len)] = NULL;
  149:                                                    }
  150:                                            }
  151:                                            if(!((char)d->flags[0] & ISO9660
_FILE_ISDIR)) {
  152:                                                    iso9660_cleanfilename(di
rent->d_name, isonum_711(d->name_len));
  153:                                            }
  154:                                            dirent = (struct dirent *)((char
 *)dirent + dirent_len);
  155:                                            size += dirent_len;
  156:                                    } else {
  157:                                            break;
  158:                                    }
  159:                                    doffset += isonum_711(d->length);
  160:                                    offset += isonum_711(d->length);
  161:                            } else {
  162:                                    break;
  163:                            }
  164:                    }
  165:                    brelse(buf);
  166:            }
  167:            fd_table->offset &= ~(blksize - 1);
  168:            doffset = fd_table->offset;
  169:            doffset += blksize;
  170:            fd_table->offset += blksize;
  171:    }
  172:    return size;
  173: }
```

fs/iso9660/file.c


```
/*
 * fiwix/fs/iso9660/file.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/buffer.h>
#include <fiwix/fs.h>
#include <fiwix/filesystems.h>
#include <fiwix/mm.h>
#include <fiwix/mman.h>
#include <fiwix/fcntl.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct fs_operations iso9660_file_fsop = {
        0,
        0,

        iso9660_file_open,
        iso9660_file_close,
        file_read,
        NULL,                   /* write */
        NULL,                   /* ioctl */
        iso9660_file_lseek,
        NULL,                   /* readdir */
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        iso9660_bmap,
        NULL,                   /* lookup */
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        NULL,                   /* truncate */
        NULL,                   /* create */
        NULL,                   /* rename */

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statfs */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

int iso9660_file_open(struct inode *i, struct fd *fd_table)
{
        if(fd_table->flags & (O_WRONLY | O_RDWR | O_TRUNC | O_APPEND)) {
                return -ENOENT;
        }
        fd_table->offset = 0;
        return 0;
```

fs/iso9660/file.c


```
68: }
69:
70: int iso9660_file_close(struct inode *i, struct fd *fd_table)
71: {
72:         return 0;
73: }
74:
75: int iso9660_file_lseek(struct inode *i, __off_t offset)
76: {
77:         return offset;
78: }
```

fs/iso9660/inode.c


```
    1: /*
    2:  * fiwix/fs/iso9660/inode.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/types.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/fs.h>
   12: #include <fiwix/filesystems.h>
   13: #include <fiwix/fs_iso9660.h>
   14: #include <fiwix/fs_pipe.h>
   15: #include <fiwix/buffer.h>
   16: #include <fiwix/stat.h>
   17: #include <fiwix/mm.h>
   18: #include <fiwix/sched.h>
   19: #include <fiwix/stdio.h>
   20: #include <fiwix/string.h>
   21:
   22: static int read_pathtable(struct inode *i)
   23: {
   24:         int n, offset, pt_len, pt_blk;
   25:         struct iso9660_sb_info *sbi;
   26:         struct iso9660_pathtable_record *ptr;
   27:         struct buffer *buf;
   28:
   29:         sbi = (struct iso9660_sb_info *)&i->sb->u.iso9660;
   30:         pt_len = isonum_733(sbi->sb->path_table_size);
   31:         pt_blk = isonum_731(sbi->sb->type_l_path_table);
   32:
   33:         if(pt_len > PAGE_SIZE) {
   34:                 printk("WARNING: %s(): path table record size (%d) > 4096, not s
upported yet.\n", __FUNCTION__, pt_len);
   35:                 return -EINVAL;
   36:         }
   37:
   38:         if(!(sbi->pathtable_raw = (void *)kmalloc())) {
   39:                 return -ENOMEM;
   40:         }
   41:         offset = 0;
   42:         while(offset < pt_len) {
   43:                 if(!(buf = bread(i->dev, pt_blk++, BLKSIZE_2K))) {
   44:                         kfree((unsigned int)sbi->pathtable_raw);
   45:                         return -EIO;
   46:                 }
   47:                 memcpy_b(sbi->pathtable_raw + offset, (void *)buf->data, MIN(pt_
len - offset, BLKSIZE_2K));
   48:                 offset += MIN(pt_len - offset, BLKSIZE_2K);
   49:                 brelse(buf);
   50:         }
   51:
   52:         /* allocate and count the number of records in the Path Table */
   53:         offset = n = 0;
   54:         if(!(sbi->pathtable = (struct iso9660_pathtable_record **)kmalloc())) {
   55:                 kfree((unsigned int)sbi->pathtable_raw);
   56:                 return -ENOMEM;
   57:         }
   58:         sbi->pathtable[n] = NULL;
   59:         while(offset < pt_len) {
   60:                 ptr = (struct iso9660_pathtable_record *)(sbi->pathtable_raw + o
ffset);
   61:                 sbi->pathtable[++n] = ptr;
   62:                 offset += sizeof(struct iso9660_pathtable_record) + isonum_711(p
tr->length) + (isonum_711(ptr->length) & 1);
   63:         }
```

fs/iso9660/inode.c


```
   64:         sbi->paths = n;
   65:
   66:         return 0;
   67: }
   68:
   69: static int get_parent_dir_size(struct superblock *sb, __blk_t extent)
   70: {
   71:         int n;
   72:         struct iso9660_pathtable_record *ptr;
   73:         __blk_t parent;
   74:
   75:         for(n = 0; n < sb->u.iso9660.paths; n++) {
   76:                 ptr = (struct iso9660_pathtable_record *)sb->u.iso9660.pathtable
[n];
   77:                 if(isonum_731(ptr->extent) == extent) {
   78:
   79:                         parent = isonum_723(ptr->parent);
   80:                         ptr = (struct iso9660_pathtable_record *)sb->u.iso9660.p
athtable[parent];
   81:                         parent = isonum_731(ptr->extent);
   82:                         return parent;
   83:                 }
   84:         }
   85:         printk("WARNING: %s(): unable to locate extent '%d' in path table.\n", _
_FUNCTION__, extent);
   86:         return 0;
   87: }
   88:
   89: int iso9660_read_inode(struct inode *i)
   90: {
   91:         int errno;
   92:         __u32 blksize;
   93:         struct superblock *sb;
   94:         struct iso9660_directory_record *d;
   95:         struct buffer *buf;
   96:         __blk_t dblock;
   97:         __off_t doffset;
   98:
   99:         sb = (struct superblock *)i->sb;
  100:         if(!sb->u.iso9660.pathtable) {
  101:                 if((errno = read_pathtable(i))) {
  102:                         return errno;
  103:                 }
  104:         }
  105:
  106:         dblock = (i->inode & ~ISO9660_INODE_MASK) >> ISO9660_INODE_BITS;
  107:         doffset = i->inode & ISO9660_INODE_MASK;
  108:         blksize = i->sb->s_blocksize;
  109:
  110:         /* FIXME: it only looks in one directory block */
  111:         if(!(buf = bread(i->dev, dblock, blksize))) {
  112:                 return -EIO;
  113:         }
  114:
  115:         if(doffset >= blksize) {
  116:                 printk("WARNING: %s(): inode %d (dblock=%d, doffset=%d) not foun
d in directory entry.\n", __FUNCTION__, i->inode, dblock, doffset);
  117:                 brelse(buf);
  118:                 return -EIO;
  119:         }
  120:         d = (struct iso9660_directory_record *)(buf->data + doffset);
  121:
  122:         i->i_mode = S_IFREG;
  123:         if((char)d->flags[0] & ISO9660_FILE_ISDIR) {
  124:                 i->i_mode = S_IFDIR;
  125:         }
  126:         if(!((char)d->flags[0] & ISO9660_FILE_HASOWNER)) {
```

fs/iso9660/inode.c


```
  127:                 i->i_mode |= S_IRUSR | S_IXUSR | S_IRGRP | S_IXGRP | S_IROTH | S
_IXOTH;
  128:         }
  129:         i->i_uid = 0;
  130:         i->i_size = isonum_733(d->size);
  131:         i->i_atime = isodate(d->date);
  132:         i->i_ctime = isodate(d->date);
  133:         i->i_mtime = isodate(d->date);
  134:         i->i_gid = 0;
  135:         i->i_nlink = 1;
  136:         i->i_blocks = 0;        /* FIXME */
  137:         i->i_flags = 0;         /* FIXME */
  138:         i->locked = 1;
  139:         i->dirty = 0;
  140:         i->mount_point = NULL;
  141:         i->count = 1;
  142:
  143:         i->u.iso9660.i_extent = isonum_733(d->extent);
  144:         check_rrip_inode(d, i);
  145:         brelse(buf);
  146:
  147:         switch(i->i_mode & S_IFMT) {
  148:                 case S_IFCHR:
  149:                         i->fsop = &def_chr_fsop;
  150:                         break;
  151:                 case S_IFBLK:
  152:                         i->fsop = &def_blk_fsop;
  153:                         break;
  154:                 case S_IFIFO:
  155:                         i->fsop = &pipefs_fsop;
  156:                         /* it's a union so we need to clear pipefs_inode */
  157:                         memset_b(&i->u.pipefs, NULL, sizeof(struct pipefs_inode)
);
  158:                         break;
  159:                 case S_IFDIR:
  160:                         i->fsop = &iso9660_dir_fsop;
  161:                         i->i_nlink++;
  162:                         break;
  163:                 case S_IFREG:
  164:                         i->fsop = &iso9660_file_fsop;
  165:                         break;
  166:                 case S_IFLNK:
  167:                         i->fsop = &iso9660_symlink_fsop;
  168:                         break;
  169:                 case S_IFSOCK:
  170:                         i->fsop = NULL;
  171:                         break;
  172:                 default:
  173:                         PANIC("invalid inode (%d) mode %08o.\n", i->inode, i->i_
mode);
  174:         }
  175:         return 0;
  176: }
  177:
  178: int iso9660_bmap(struct inode *i, __off_t offset, int mode)
  179: {
  180:         __blk_t block;
  181:
  182:         block = i->u.iso9660.i_extent + (offset / i->sb->s_blocksize);
  183:         return block;
  184: }
```

fs/iso9660/Makefile


```
 1: # fiwix/fs/iso9660/Makefile
 2: #
 3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
 4: # Distributed under the terms of the Fiwix License.
 5: #
 6:
 7: .S.o:
 8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
 9: .c.o:
10:         $(CC) $(CFLAGS) -c -o $@ $<
11:
12: OBJS = inode.o super.o namei.o dir.o file.o rrip.o symlink.o
13:
14: iso9660:$(OBJS)
15:         $(LD) $(LDFLAGS) -r $(OBJS) -o iso9660.o
16:
17: clean:
18:         rm -f *.o
19:
```

fs/iso9660/namei.c


```
    1: /*
    2:  * fiwix/fs/iso9660/namei.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/filesystems.h>
   11: #include <fiwix/fs_iso9660.h>
   12: #include <fiwix/buffer.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/errno.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: int iso9660_lookup(const char *name, struct inode *dir, struct inode **i_res)
   20: {
   21:         __blk_t dblock;
   22:         __u32 blksize;
   23:         int len, dnlen;
   24:         unsigned int offset, doffset;
   25:         struct buffer *buf;
   26:         struct iso9660_directory_record *d;
   27:         __ino_t inode;
   28:         int nm_len;
   29:         char *nm_name;
   30:
   31:         blksize = dir->sb->s_blocksize;
   32:         inode = offset = 0;
   33:         len = strlen(name);
   34:         dir->count++;
   35:
   36:         while(offset < dir->i_size && !inode) {
   37:                 if((dblock = bmap(dir, offset, FOR_READING)) < 0) {
   38:                         return dblock;
   39:                 }
   40:                 if(dblock) {
   41:                         if(!(buf = bread(dir->dev, dblock, blksize))) {
   42:                                 return -EIO;
   43:                         }
   44:                         doffset = 0;
   45:                         do {
   46:                                 d = (struct iso9660_directory_record *)(buf->dat
a + doffset);
   47:                                 if(isonum_711(d->length) == 0) {
   48:                                         break;
   49:                                 }
   50:                                 if(len == 1) {
   51:                                         if(name[0] == '.' && name[1] == NULL) {
   52:                                                 if(isonum_711(d->name_len) == 1
&& d->name[0] == 0) {
   53:                                                         inode = dir->inode;
   54:                                                 }
   55:                                         }
   56:                                 }
   57:                                 if(len == 2) {
   58:                                         if(name[0] == '.' && name[1] == '.' && n
ame[2] == NULL) {
   59:                                                 if(isonum_711(d->name_len) == 1
&& d->name[0] == 1) {
   60:                                                         inode = dir->u.iso9660.i
_parent->inode;
   61:                                                 }
   62:                                         }
```

fs/iso9660/namei.c


```
   63:                                  }
   64:                                  if(!(nm_name = (char *)kmalloc())) {
   65:                                          return -ENOMEM;
   66:                                  }
   67:                                  nm_len = 0;
   68:                                  if(dir->sb->u.iso9660.rrip) {
   69:                                          nm_len = get_rrip_filename(d, dir, nm_na
me);
   70:                                  }
   71:                                  if(nm_len) {
   72:                                          dnlen = nm_len;
   73:                                  } else {
   74:                                          dnlen = isonum_711(d->name_len);
   75:                                          if(!((char)d->flags[0] & ISO9660_FILE_IS
DIR)) {
   76:                                                  iso9660_cleanfilename(d->name, d
nlen);
   77:                                                  dnlen = strlen(d->name);
   78:                                          }
   79:                                  }
   80:                                  if(len == dnlen) {
   81:                                          if(nm_len) {
   82:                                                  if(strncmp(nm_name, name, dnlen)
 == 0) {
   83:                                                          inode = (dblock << ISO96
60_INODE_BITS) + (doffset & ISO9660_INODE_MASK);
   84:                                                  }
   85:                                          } else {
   86:                                                  if(strncmp(d->name, name, dnlen)
 == 0) {
   87:                                                          inode = (dblock << ISO96
60_INODE_BITS) + (doffset & ISO9660_INODE_MASK);
   88:                                                  }
   89:                                          }
   90:                                  }
   91:                                  kfree((unsigned int)nm_name);
   92:                                  doffset += isonum_711(d->length);
   93:                          } while((doffset < blksize) && (!inode));
   94:                          brelse(buf);
   95:                          offset += blksize;
   96:                          if(inode) {
   97:                                  /*
   98:                                   * This prevents a deadlock in iget() when
   99:                                   * trying to lock '.' when 'dir' is the same
  100:                                   * directory (ls -lai <tmp>).
  101:                                   */
  102:                                  if(inode == dir->inode) {
  103:                                          *i_res = dir;
  104:                                          return 0;
  105:                                  }
  106:
  107:                                  if(!(*i_res = iget(dir->sb, inode))) {
  108:                                          return -EACCES;
  109:                                  }
  110:                                  if(S_ISDIR((*i_res)->i_mode)) {
  111:                                          if(!(*i_res)->u.iso9660.i_parent) {
  112:                                                  (*i_res)->u.iso9660.i_parent = d
ir;
  113:                                          }
  114:                                  }
  115:                                  iput(dir);
  116:                                  return 0;
  117:                          }
  118:                  }
  119:          }
  120:          iput(dir);
  121:          return -ENOENT;
```

fs/iso9660/namei.c


```
122: }
```

fs/iso9660/rrip.c


```
    1: /*
    2:  * fiwix/fs/iso9660/rrip.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/errno.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/stat.h>
   12: #include <fiwix/filesystems.h>
   13: #include <fiwix/buffer.h>
   14: #include <fiwix/fs_iso9660.h>
   15: #include <fiwix/mm.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: void check_rrip_inode(struct iso9660_directory_record *d, struct inode *i)
   20: {
   21:         unsigned int total_len;
   22:         unsigned int len;
   23:         unsigned int sig;
   24:         int n, nm_len, rootflag;
   25:         struct susp_rrip *rrip;
   26:         unsigned int dev_h, dev_l;
   27:         unsigned int ce_block, ce_offset, ce_size;
   28:         struct buffer *buf;
   29:         unsigned char *sue;
   30:         int sl_len;
   31:         struct rrip_sl_component *slc;
   32:
   33:         ce_block = ce_offset = ce_size = 0;
   34:         buf = NULL;
   35:         total_len = isonum_711(d->length);
   36:         len = isonum_711(d->name_len);
   37:         if(!(len % 2)) {
   38:                 len++;
   39:         }
   40:         sue = (unsigned char *)d->name;
   41:         nm_len = 0;
   42:
   43: loop:
   44:         if(ce_block && ce_size) {
   45:                 /* FIXME: it only looks in one directory block */
   46:                 if(!(buf = bread(i->dev, ce_block, i->sb->s_blocksize))) {
   47:                         return;
   48:                 }
   49:                 sue = (unsigned char *)buf->data + ce_offset;
   50:                 total_len = ce_size;
   51:                 len = 0;
   52:         }
   53:
   54:         while(len < total_len) {
   55:                 rrip = (struct susp_rrip *)(sue + len);
   56:                 if(rrip->len == 0) {
   57:                         break;
   58:                 }
   59:                 sig = GET_SIG(rrip->signature[0], rrip->signature[1]);
   60:                 switch(sig) {
   61:                         case GET_SIG('S', 'P'):
   62:                                 if(rrip->u.sp.magic[0] != SP_MAGIC1 || rrip->u.s
p.magic[1] != SP_MAGIC2) {
   63:                                         if(ce_block) {
   64:                                                 brelse(buf);
   65:                                         }
   66:                                         return;
```

fs/iso9660/rrip.c


```
   67:                                     }
   68:                                     break;
   69:                             case GET_SIG('C', 'E'):
   70:                                     if(ce_block) {
   71:                                             brelse(buf);
   72:                                     }
   73:                                     ce_block = isonum_733(rrip->u.ce.block);
   74:                                     ce_offset = isonum_733(rrip->u.ce.offset);
   75:                                     ce_size = isonum_733(rrip->u.ce.size);
   76:                                     goto loop;
   77:                                     break;
   78:                             case GET_SIG('E', 'R'):
   79:                                     i->sb->u.iso9660.rrip = 1;
   80:                                     printk("ISO 9660 Extensions: ");
   81:                                     for(n = 0; n < rrip->u.er.len_id; n++) {
   82:                                             printk("%c", rrip->u.er.data[n]);
   83:                                     }
   84:                                     printk("\n");
   85:                                     break;
   86:                             case GET_SIG('P', 'X'):
   87:                                     i->i_mode  = isonum_733(rrip->u.px.mode);
   88:                                     i->i_nlink = isonum_733(rrip->u.px.nlink);
   89:                                     i->i_uid   = isonum_733(rrip->u.px.uid);
   90:                                     i->i_gid   = isonum_733(rrip->u.px.gid);
   91:                                     break;
   92:                             case GET_SIG('P', 'N'):
   93:                                     if(S_ISBLK(i->i_mode) || S_ISCHR(i->i_mode)) {
   94:                                             dev_h = isonum_733(rrip->u.pn.dev_h);
   95:                                             dev_l = isonum_733(rrip->u.pn.dev_l);
   96:                                             i->rdev = MKDEV(dev_h, dev_l);
   97:                                     }
   98:                                     break;
   99:                             case GET_SIG('S', 'L'):
  100:                                     sl_len = rootflag = 0;
  101:                                     slc = (struct rrip_sl_component *)&rrip->u.sl.ar
ea;
  102:                                     while(sl_len < (rrip->len - 5)) {
  103:                                             if(sl_len && !rootflag) {
  104:                                                     nm_len++;
  105:                                             }
  106:                                             rootflag = 0;
  107:                                             switch(slc->flags & 0xE) {
  108:                                                     case 0:
  109:                                                             nm_len += slc->len;
  110:                                                             break;
  111:                                                     case SL_CURRENT:
  112:                                                             nm_len += 1;
  113:                                                             break;
  114:                                                     case SL_PARENT:
  115:                                                             nm_len += 2;
  116:                                                             break;
  117:                                                     case SL_ROOT:
  118:                                                             nm_len += 1;
  119:                                                             rootflag = 1;
  120:                                                             break;
  121:                                                     default:
  122:                                                             printk("WARNING: %s(): u
nsupported RRIP SL flags %d.\n", __FUNCTION__, slc->flags & 0xE);
  123:                                             }
  124:                                             slc = (struct rrip_sl_component *)(((cha
r *)slc) + slc->len + sizeof(struct rrip_sl_component));
  125:                                             sl_len += slc->len + sizeof(struct rrip_
sl_component);
  126:                                     }
  127:                                     i->i_size = nm_len;
  128:                                     break;
  129:                             case GET_SIG('T', 'F'):
```

fs/iso9660/rrip.c


```
  130:                                    n = 0;
  131:                                    if(rrip->u.tf.flags & TF_CREATION) {
  132:                                            i->i_ctime = isodate(rrip->u.tf.times[n+
+].time);
  133:                                    }
  134:                                    if(rrip->u.tf.flags & TF_MODIFY) {
  135:                                            i->i_mtime = isodate(rrip->u.tf.times[n+
+].time);
  136:                                    }
  137:                                    if(rrip->u.tf.flags & TF_ACCESS) {
  138:                                            i->i_atime = isodate(rrip->u.tf.times[n+
+].time);
  139:                                    }
  140:                                    if(rrip->u.tf.flags & TF_ATTRIBUTES) {
  141:                                            i->i_ctime = isodate(rrip->u.tf.times[n+
+].time);
  142:                                    }
  143:                                    break;
  144:                            }
  145:                            len += rrip->len;
  146:                    }
  147:                    if(ce_block) {
  148:                            brelse(buf);
  149:                    }
  150: }
  151:
  152: int get_rrip_filename(struct iso9660_directory_record *d, struct inode *i, char
*name)
  153: {
  154:                    unsigned int total_len;
  155:                    unsigned int len;
  156:                    unsigned int sig;
  157:                    int nm_len;
  158:                    struct susp_rrip *rrip;
  159:                    unsigned int ce_block, ce_offset, ce_size;
  160:                    struct buffer *buf;
  161:                    unsigned char *sue;
  162:
  163:                    ce_block = ce_offset = ce_size = 0;
  164:                    buf = NULL;
  165:                    total_len = isonum_711(d->length);
  166:                    len = isonum_711(d->name_len);
  167:                    if(!(len % 2)) {
  168:                            len++;
  169:                    }
  170:                    sue = (unsigned char *)d->name;
  171:                    nm_len = 0;
  172:
  173: loop:
  174:                    if(ce_block && ce_size) {
  175:                            /* FIXME: it only looks in one directory block */
  176:                            if(!(buf = bread(i->dev, ce_block, i->sb->s_blocksize))) {
  177:                                    return 0;
  178:                            }
  179:                            sue = (unsigned char *)buf->data + ce_offset;
  180:                            total_len = ce_size;
  181:                            len = 0;
  182:                    }
  183:
  184:                    while(len < total_len) {
  185:                            rrip = (struct susp_rrip *)(sue + len);
  186:                            if(rrip->len == 0) {
  187:                                    break;
  188:                            }
  189:                            sig = GET_SIG(rrip->signature[0], rrip->signature[1]);
  190:                            switch(sig) {
  191:                                    case GET_SIG('S', 'P'):
```

fs/iso9660/rrip.c


```
  192:                                    if(rrip->u.sp.magic[0] != SP_MAGIC1 || rrip->u.s
p.magic[1] != SP_MAGIC2) {
  193:                                            if(ce_block) {
  194:                                                    brelse(buf);
  195:                                            }
  196:                                            return 0;
  197:                                    }
  198:                                    break;
  199:                            case GET_SIG('C', 'E'):
  200:                                    if(ce_block) {
  201:                                            brelse(buf);
  202:                                    }
  203:                                    ce_block = isonum_733(rrip->u.ce.block);
  204:                                    ce_offset = isonum_733(rrip->u.ce.offset);
  205:                                    ce_size = isonum_733(rrip->u.ce.size);
  206:                                    goto loop;
  207:                            case GET_SIG('N', 'M'):
  208:                                    if(rrip->u.nm.flags) { /* FIXME: & ~(NM_CONTINUE
 | NM_CURRENT | NM_PARENT)) { */
  209:                                            printk("WARNING: %s(): unsupported NM fl
ag settings (%d).\n", __FUNCTION__, rrip->u.nm.flags);
  210:                                            if(ce_block) {
  211:                                                    brelse(buf);
  212:                                            }
  213:                                            return 0;
  214:                                    }
  215:                                    nm_len = rrip->len - 5;
  216:                                    memcpy_b(name, rrip->u.nm.name, nm_len);
  217:                                    name[nm_len] = NULL;
  218:                                    break;
  219:                    }
  220:                    len += rrip->len;
  221:            }
  222:            if(ce_block) {
  223:                    brelse(buf);
  224:            }
  225:            return nm_len;
  226: }
  227:
  228: int get_rrip_symlink(struct inode *i, char *name)
  229: {
  230:            unsigned int total_len;
  231:            unsigned int len;
  232:            unsigned int sig;
  233:            int nm_len;
  234:            struct susp_rrip *rrip;
  235:            unsigned int ce_block, ce_offset, ce_size;
  236:            struct buffer *buf;
  237:            struct buffer *buf2;
  238:            unsigned char *sue;
  239:            struct iso9660_directory_record *d;
  240:            __blk_t dblock;
  241:            __off_t doffset;
  242:            int sl_len, rootflag;
  243:            struct rrip_sl_component *slc;
  244:
  245:            dblock = (i->inode & ~ISO9660_INODE_MASK) >> ISO9660_INODE_BITS;
  246:            doffset = i->inode & ISO9660_INODE_MASK;
  247:            /* FIXME: it only looks in one directory block */
  248:            if(!(buf = bread(i->dev, dblock, i->sb->s_blocksize))) {
  249:                    return -EIO;
  250:            }
  251:            d = (struct iso9660_directory_record *)(buf->data + doffset);
  252:
  253:            ce_block = ce_offset = ce_size = 0;
  254:            buf2 = NULL;
  255:            total_len = isonum_711(d->length);
```

fs/iso9660/rrip.c


```
  256:          len = isonum_711(d->name_len);
  257:          if(!(len % 2)) {
  258:                  len++;
  259:          }
  260:          sue = (unsigned char *)d->name;
  261:          nm_len = 0;
  262:
  263: loop:
  264:          if(ce_block && ce_size) {
  265:                  /* FIXME: it only looks in one directory block */
  266:                  if(!(buf2 = bread(i->dev, ce_block, i->sb->s_blocksize))) {
  267:                          return 0;
  268:                  }
  269:                  sue = (unsigned char *)buf2->data + ce_offset;
  270:                  total_len = ce_size;
  271:                  len = 0;
  272:          }
  273:
  274:          while(len < total_len) {
  275:                  rrip = (struct susp_rrip *)(sue + len);
  276:                  if(rrip->len == 0) {
  277:                          break;
  278:                  }
  279:                  sig = GET_SIG(rrip->signature[0], rrip->signature[1]);
  280:                  switch(sig) {
  281:                          case GET_SIG('S', 'P'):
  282:                                  if(rrip->u.sp.magic[0] != SP_MAGIC1 || rrip->u.s
p.magic[1] != SP_MAGIC2) {
  283:                                          if(ce_block) {
  284:                                                  brelse(buf2);
  285:                                          }
  286:                                          return 0;
  287:                                  }
  288:                                  break;
  289:                          case GET_SIG('C', 'E'):
  290:                                  if(ce_block) {
  291:                                          brelse(buf2);
  292:                                  }
  293:                                  ce_block = isonum_733(rrip->u.ce.block);
  294:                                  ce_offset = isonum_733(rrip->u.ce.offset);
  295:                                  ce_size = isonum_733(rrip->u.ce.size);
  296:                                  goto loop;
  297:                          case GET_SIG('S', 'L'):
  298:                                  sl_len = rootflag = 0;
  299:                                  slc = (struct rrip_sl_component *)&rrip->u.sl.ar
ea;
  300:                                  while(sl_len < (rrip->len - 5)) {
  301:                                          if(sl_len && !rootflag) {
  302:                                                  strcat(name, "/");
  303:                                                  nm_len++;
  304:                                          }
  305:                                          rootflag = 0;
  306:                                          switch(slc->flags & 0xE) {
  307:                                                  case 0:
  308:                                                          nm_len += slc->len;
  309:                                                          strncat(name, slc->name,
 slc->len);
  310:                                                          break;
  311:                                                  case SL_CURRENT:
  312:                                                          nm_len += 1;
  313:                                                          strcat(name, ".");
  314:                                                          break;
  315:                                                  case SL_PARENT:
  316:                                                          nm_len += 2;
  317:                                                          strcat(name, "..");
  318:                                                          break;
  319:                                                  case SL_ROOT:
```

fs/iso9660/rrip.c


```
  320:                                                                    nm_len += 1;
  321:                                                                    rootflag = 1;
  322:                                                                    strcat(name, "/");
  323:                                                                    break;
  324:                                                            default:
  325:                                                                    printk("WARNING: %s(): u
nsupported RRIP SL flags %d.\n", __FUNCTION__, slc->flags & 0xE);
  326:                                                            }
  327:                                                            slc = (struct rrip_sl_component *)(((cha
r *)slc) + slc->len + sizeof(struct rrip_sl_component));
  328:                                                            sl_len += slc->len + sizeof(struct rrip_
sl_component);
  329:                                    }
  330:                                    name[nm_len] = NULL;
  331:                                    break;
  332:                    }
  333:                    len += rrip->len;
  334:            }
  335:            if(ce_block) {
  336:                    brelse(buf2);
  337:            }
  338:            brelse(buf);
  339:            return nm_len;
  340: }
```

fs/iso9660/super.c


```
    1: /*
    2:  * fiwix/fs/iso9660/super.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/errno.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/filesystems.h>
   12: #include <fiwix/fs_iso9660.h>
   13: #include <fiwix/buffer.h>
   14: #include <fiwix/time.h>
   15: #include <fiwix/sched.h>
   16: #include <fiwix/mm.h>
   17: #include <fiwix/stdio.h>
   18: #include <fiwix/string.h>
   19:
   20: struct fs_operations iso9660_fsop = {
   21:         FSOP_REQUIRES_DEV,
   22:         NULL,
   23:
   24:         NULL,                   /* open */
   25:         NULL,                   /* close */
   26:         NULL,                   /* read */
   27:         NULL,                   /* write */
   28:         NULL,                   /* ioctl */
   29:         NULL,                   /* lseek */
   30:         NULL,                   /* readdir */
   31:         NULL,                   /* mmap */
   32:         NULL,                   /* select */
   33:
   34:         NULL,                   /* readlink */
   35:         NULL,                   /* followlink */
   36:         NULL,                   /* bmap */
   37:         NULL,                   /* lookup */
   38:         NULL,                   /* rmdir */
   39:         NULL,                   /* link */
   40:         NULL,                   /* unlink */
   41:         NULL,                   /* symlink */
   42:         NULL,                   /* mkdir */
   43:         NULL,                   /* mknod */
   44:         NULL,                   /* truncate */
   45:         NULL,                   /* create */
   46:         NULL,                   /* rename */
   47:
   48:         NULL,                   /* read_block */
   49:         NULL,                   /* write_block */
   50:
   51:         iso9660_read_inode,
   52:         NULL,                   /* write_inode */
   53:         NULL,                   /* ialloc */
   54:         NULL,                   /* ifree */
   55:         iso9660_statfs,
   56:         iso9660_read_superblock,
   57:         NULL,                   /* remount_fs */
   58:         NULL,                   /* write_superblock */
   59:         iso9660_release_superblock
   60: };
   61:
   62: int isonum_711(char *str)
   63: {
   64:         unsigned char *le;
   65:
   66:         le = (unsigned char *)str;
   67:         return le[0];
```

fs/iso9660/super.c


```
 68: }
 69:
 70: /* return a 16bit little-endian number */
 71: int isonum_723(char *str)
 72: {
 73:         unsigned char *le;
 74:
 75:         le = (unsigned char *)str;
 76:         return le[0] | (le[1] << 8);
 77: }
 78:
 79: /* return a 32bit little-endian number */
 80: int isonum_731(char *str)
 81: {
 82:         unsigned char *le;
 83:
 84:         le = (unsigned char *)str;
 85:         return le[0] | (le[1] << 8) | (le[2] << 16) | (le[3] << 24);
 86: }
 87:
 88: /* return a 32bit little-endian number */
 89: int isonum_733(char *p)
 90: {
 91:         return isonum_731(p);
 92: }
 93:
 94: /* return a date and time format */
 95: unsigned long int isodate(char *p)
 96: {
 97:         struct mt mt;
 98:
 99:         if(!p[0]) {
100:                 return 0;
101:         }
102:
103:         mt.mt_sec = p[5];
104:         mt.mt_min = p[4];
105:         mt.mt_hour = p[3];
106:         mt.mt_day = p[2];
107:         mt.mt_month = p[1];
108:         mt.mt_year = p[0];
109:         mt.mt_year += 1900;
110:         mt.mt_min += p[6] * 15;
111:
112:         return mktime(&mt);
113: }
114:
115: /* return a clean filename */
116: int iso9660_cleanfilename(char *filename, int len)
117: {
118:         int n;
119:         char *p;
120:
121:         p = filename;
122:         if(len > 2) {
123:                 for(n = 0; n < len; n++) {
124:                         if((len - n) == 2) {
125:                                 if(p[n] == ';' && p[n + 1] == '1') {
126:                                         filename[n] = NULL;
127:                                         if(p[n - 1] == '.') {
128:                                                 filename[n - 1] = NULL;
129:                                         }
130:                                         return 0;
131:                                 }
132:                         }
133:                 }
134:         }
```

fs/iso9660/super.c


```
  135:          return 1;
  136: }
  137:
  138: void iso9660_statfs(struct superblock *sb, struct statfs *statfsbuf)
  139: {
  140:          statfsbuf->f_type = ISO9660_SUPER_MAGIC;
  141:          statfsbuf->f_bsize = sb->s_blocksize;
  142:          statfsbuf->f_blocks = isonum_733(sb->u.iso9660.sb->volume_space_size);
  143:          statfsbuf->f_bfree = 0;
  144:          statfsbuf->f_bavail = 0;
  145:          statfsbuf->f_files = 0;         /* FIXME */
  146:          statfsbuf->f_ffree = 0;
  147:          /* statfsbuf->f_fsid = ? */
  148:          statfsbuf->f_namelen = NAME_MAX;
  149: }
  150:
  151: int iso9660_read_superblock(__dev_t dev, struct superblock *sb)
  152: {
  153:          struct buffer *buf;
  154:          struct iso9660_super_block *iso9660sb;
  155:          struct iso9660_super_block *pvd;
  156:          struct iso9660_directory_record *dr;
  157:          __ino_t root_inode;
  158:          int n;
  159:
  160:          superblock_lock(sb);
  161:          pvd = NULL;
  162:
  163:          for(n = 0; n < ISO9660_MAX_VD; n++) {
  164:                  if(!(buf = bread(dev, ISO9660_SUPERBLOCK + n, BLKSIZE_2K))) {
  165:                          superblock_unlock(sb);
  166:                          return -EIO;
  167:                  }
  168:
  169:                  iso9660sb = (struct iso9660_super_block *)buf->data;
  170:                  if(strncmp(iso9660sb->id, ISO9660_STANDARD_ID, sizeof(iso9660sb-
>id)) || (isonum_711(iso9660sb->type) == ISO9660_VD_END)) {
  171:                          break;
  172:                  }
  173:                  if(isonum_711(iso9660sb->type) == ISO9660_VD_PRIMARY) {
  174:                          pvd = (struct iso9660_super_block *)buf->data;
  175:                          break;
  176:                  }
  177:                  brelse(buf);
  178:          }
  179:          if(!pvd) {
  180:                  printk("WARNING: %s(): invalid filesystem type or bad superblock
 on device %d,%d.\n", __FUNCTION__, MAJOR(dev), MINOR(dev));
  181:                  superblock_unlock(sb);
  182:                  brelse(buf);
  183:                  return -EINVAL;
  184:          }
  185:
  186:          dr = (struct iso9660_directory_record *)pvd->root_directory_record;
  187:          root_inode = isonum_711(dr->extent);
  188:
  189:          sb->dev = dev;
  190:          sb->fsop = &iso9660_fsop;
  191:          sb->flags = MS_RDONLY;
  192:          sb->s_blocksize = isonum_723(pvd->logical_block_size);
  193:          sb->u.iso9660.rrip = 0;
  194:          if(!(sb->u.iso9660.sb = (void *)kmalloc())) {
  195:                  superblock_unlock(sb);
  196:                  brelse(buf);
  197:                  return -ENOMEM;
  198:          }
  199:          memcpy_b(sb->u.iso9660.sb, pvd, sizeof(struct iso9660_super_block));
```

fs/iso9660/super.c


```
  200:          brelse(buf);
  201:
  202:          root_inode = (root_inode << ISO9660_INODE_BITS) + (0 & ISO9660_INODE_MAS
K);
  203:          if(!(sb->root = iget(sb, root_inode))) {
  204:                  printk("WARNING: %s(): unable to get root inode.\n", __FUNCTION_
_);
  205:                  superblock_unlock(sb);
  206:                  return -EINVAL;
  207:          }
  208:          sb->u.iso9660.s_root_inode = root_inode;
  209:
  210:          superblock_unlock(sb);
  211:          return 0;
  212: }
  213:
  214: void iso9660_release_superblock(struct superblock *sb)
  215: {
  216:          kfree((unsigned int)sb->u.iso9660.sb);
  217:          kfree((unsigned int)sb->u.iso9660.pathtable);
  218:          kfree((unsigned int)sb->u.iso9660.pathtable_raw);
  219: }
  220:
  221: int iso9660_init(void)
  222: {
  223:          return register_filesystem("iso9660", &iso9660_fsop);
  224: }
```

fs/iso9660/symlink.c


```
/*
 * fiwix/fs/iso9660/symlink.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/buffer.h>
#include <fiwix/fs.h>
#include <fiwix/filesystems.h>
#include <fiwix/fs_iso9660.h>
#include <fiwix/stat.h>
#include <fiwix/mm.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct fs_operations iso9660_symlink_fsop = {
        0,
        0,

        NULL,                   /* open */
        NULL,                   /* close */
        NULL,                   /* read */
        NULL,                   /* write */
        NULL,                   /* ioctl */
        NULL,                   /* lseek */
        NULL,                   /* readdir */
        NULL,                   /* mmap */
        NULL,                   /* select */

        iso9660_readlink,
        iso9660_followlink,
        NULL,                   /* bmap */
        NULL,                   /* lookup */
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        NULL,                   /* truncate */
        NULL,                   /* create */
        NULL,                   /* rename */

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statfs */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

int iso9660_readlink(struct inode *i, char *buffer, __size_t count)
{
        __off_t size_read;
        char *name;

        if(!(name = (char *)kmalloc())) {
                return -ENOMEM;
```

fs/iso9660/symlink.c


```
   68:         }
   69:
   70:         inode_lock(i);
   71:         name[0] = NULL;
   72:         if((size_read = get_rrip_symlink(i, name))) {
   73:                 size_read = MIN(size_read, count);
   74:                 memcpy_b(buffer, name, size_read);
   75:         }
   76:         kfree((unsigned int)name);
   77:         inode_unlock(i);
   78:         return size_read;
   79: }
   80:
   81: int iso9660_followlink(struct inode *dir, struct inode *i, struct inode **i_res)
   82: {
   83:         char *name;
   84:         __off_t size_read;
   85:         __ino_t errno;
   86:
   87:         if(!i) {
   88:                 return -ENOENT;
   89:         }
   90:         if(!S_ISLNK(i->i_mode)) {
   91:                 printk("%s(): Oops, inode '%d' is not a symlink (!?).\n", __FUNC
TION__, i->inode);
   92:                 return 0;
   93:         }
   94:
   95:         if(!(name = (char *)kmalloc())) {
   96:                 return -ENOMEM;
   97:         }
   98:
   99:         name[0] = NULL;
  100:         if((size_read = get_rrip_symlink(i, name))) {
  101:                 iput(i);
  102:                 if((errno = parse_namei(name, dir, i_res, NULL, FOLLOW_LINKS)))
{
  103:                         kfree((unsigned int)name);
  104:                         return errno;
  105:                 }
  106:         }
  107:         kfree((unsigned int)name);
  108:         return 0;
  109: }
```

fs/minix/bitmaps.c


```
    1: /*
    2:  * fiwix/fs/minix/bitmaps.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/filesystems.h>
   11: #include <fiwix/fs_minix.h>
   12: #include <fiwix/buffer.h>
   13: #include <fiwix/errno.h>
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/string.h>
   16:
   17: #define COUNT           1
   18: #define FIRST_ZERO      2
   19:
   20: static int count_bits(struct superblock *sb, __blk_t offset, int num, int blocks
, int mode)
   21: {
   22:         unsigned char c;
   23:         int blksize;
   24:         int n, n2, last, bits, count, mapb;
   25:         struct buffer *buf;
   26:
   27:         count = mapb = 0;
   28:         blksize = sb->s_blocksize;
   29:
   30:         while(offset < blocks) {
   31:                 if(!(buf = bread(sb->dev, offset, blksize))) {
   32:                         return -EIO;
   33:                 }
   34:                 last = (num / 8) > blksize ? blksize : (num / 8);
   35:                 for(n = 0; n < last; n++) {
   36:                         c = (unsigned char)buf->data[n];
   37:                         bits = n < last ? 8 : num & 8;
   38:                         for(n2 = 0; n2 < bits; n2++) {
   39:                                 if(c & (1 << n2)) {
   40:                                         if(mode == COUNT) {
   41:                                                 count++;
   42:                                         }
   43:                                 } else {
   44:                                         if(mode == FIRST_ZERO) {
   45:                                                 brelse(buf);
   46:                                                 return n2 + ((n * 8) + (mapb * b
lksize * 8));
   47:                                         }
   48:                                 }
   49:                         }
   50:                 }
   51:                 offset++;
   52:                 mapb++;
   53:                 num -= (blksize * 8);
   54:                 brelse(buf);
   55:         }
   56:         return count;
   57: }
   58:
   59: int minix_change_bit(int mode, struct superblock *sb, int map, int item)
   60: {
   61:         int byte, bit, mask;
   62:         struct buffer *buf;
   63:
   64:         map += item / (sb->s_blocksize * 8);
   65:         byte = (item % (sb->s_blocksize * 8)) / 8;
```

fs/minix/bitmaps.c


```
   66:         bit = (item % (sb->s_blocksize * 8)) % 8;
   67:         mask = 1 << bit;
   68:
   69:         if(!(buf = bread(sb->dev, map, sb->s_blocksize))) {
   70:                 return -EIO;
   71:         }
   72:
   73:         if(mode == CLEAR_BIT) {
   74:                 if(!(buf->data[byte] & mask)) {
   75:                         brelse(buf);
   76:                         return 1;
   77:                 }
   78:                 buf->data[byte] &= ~mask;
   79:         }
   80:         if(mode == SET_BIT) {
   81:                 if((buf->data[byte] & mask)) {
   82:                         brelse(buf);
   83:                         return 1;
   84:                 }
   85:                 buf->data[byte] |= mask;
   86:         }
   87:
   88:         bwrite(buf);
   89:         return 0;
   90: }
   91:
   92: int minix_balloc(struct superblock *sb)
   93: {
   94:         int map, block, errno;
   95:
   96:         superblock_lock(sb);
   97:
   98:         map = 1 + SUPERBLOCK + sb->u.minix.sb.s_imap_blocks;
   99:
  100:         if(!(block = minix_find_first_zero(sb, map, sb->u.minix.nzones, map + sb
->u.minix.sb.s_zmap_blocks))) {
  101:                 superblock_unlock(sb);
  102:                 return -ENOSPC;
  103:         }
  104:
  105:         errno = minix_change_bit(SET_BIT, sb, map, block);
  106:         block += sb->u.minix.sb.s_firstdatazone - 1;
  107:
  108:         if(errno) {
  109:                 if(errno < 0) {
  110:                         printk("WARNING: %s(): unable to set block %d.\n", __FUN
CTION__, block);
  111:                         superblock_unlock(sb);
  112:                         return errno;
  113:                 } else {
  114:                         printk("WARNING: %s(): block %d is already marked as use
d!\n", __FUNCTION__, block);
  115:                 }
  116:         }
  117:
  118:         superblock_unlock(sb);
  119:         return block;
  120: }
  121:
  122: void minix_bfree(struct superblock *sb, int block)
  123: {
  124:         int map, errno;
  125:
  126:         if(block < sb->u.minix.sb.s_firstdatazone || block > sb->u.minix.nzones)
 {
  127:                 printk("WARNING: %s(): block %d is not in datazone.\n", __FUNCTI
ON__, block);
```

fs/minix/bitmaps.c


```
  128:                    return;
  129:            }
  130:
  131:            superblock_lock(sb);
  132:
  133:            map = 1 + SUPERBLOCK + sb->u.minix.sb.s_imap_blocks;
  134:            block -= sb->u.minix.sb.s_firstdatazone - 1;
  135:            errno = minix_change_bit(CLEAR_BIT, sb, map, block);
  136:
  137:            if(errno) {
  138:                    if(errno < 0) {
  139:                            printk("WARNING: %s(): unable to free block %d.\n", __FU
NCTION__, block);
  140:                    } else {
  141:                            printk("WARNING: %s(): block %d is already marked as fre
e!\n", __FUNCTION__, block);
  142:                    }
  143:            }
  144:
  145:            superblock_unlock(sb);
  146:            return;
  147: }
  148:
  149: int minix_count_free_inodes(struct superblock *sb)
  150: {
  151:            __blk_t offset;
  152:
  153:            offset = 1 + SUPERBLOCK;
  154:            return count_bits(sb, offset, sb->u.minix.sb.s_ninodes, offset + sb->u.m
inix.sb.s_imap_blocks, COUNT);
  155: }
  156:
  157: int minix_count_free_blocks(struct superblock *sb)
  158: {
  159:            __blk_t offset;
  160:
  161:            offset = 1 + SUPERBLOCK + sb->u.minix.sb.s_imap_blocks;
  162:            return count_bits(sb, offset, sb->u.minix.nzones, offset + sb->u.minix.s
b.s_zmap_blocks, COUNT);
  163: }
  164:
  165: int minix_find_first_zero(struct superblock *sb, __blk_t offset, int num, int bl
ocks)
  166: {
  167:            return count_bits(sb, offset, num, blocks, FIRST_ZERO);
  168: }
```

fs/minix/dir.c


```
/*
 * fiwix/fs/minix/dir.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/buffer.h>
#include <fiwix/fs.h>
#include <fiwix/filesystems.h>
#include <fiwix/stat.h>
#include <fiwix/dirent.h>
#include <fiwix/mm.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct fs_operations minix_dir_fsop = {
        0,
        0,

        minix_dir_open,
        minix_dir_close,
        minix_dir_read,
        minix_dir_write,
        NULL,                   /* ioctl */
        NULL,                   /* lseek */
        minix_dir_readdir,
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        minix_bmap,
        minix_lookup,
        minix_rmdir,
        minix_link,
        minix_unlink,
        minix_symlink,
        minix_mkdir,
        minix_mknod,
        NULL,                   /* truncate */
        minix_create,
        minix_rename,

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statfs */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

int minix_dir_open(struct inode *i, struct fd *fd_table)
{
        fd_table->offset = 0;
        return 0;
}

int minix_dir_close(struct inode *i, struct fd *fd_table)
```

fs/minix/dir.c


```
   68: {
   69:         return 0;
   70: }
   71:
   72: int minix_dir_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t
count)
   73: {
   74:         return -EISDIR;
   75: }
   76:
   77: int minix_dir_write(struct inode *i, struct fd *fd_table, const char *buffer, __
size_t count)
   78: {
   79:         return -EBADF;
   80: }
   81:
   82: int minix_dir_readdir(struct inode *i, struct fd *fd_table, struct dirent *diren
t, unsigned int count)
   83: {
   84:         __blk_t block;
   85:         unsigned int doffset, offset;
   86:         unsigned int size, dirent_len;
   87:         struct minix_dir_entry *d;
   88:         int base_dirent_len;
   89:         int blksize;
   90:         struct buffer *buf;
   91:
   92:         if(!(S_ISDIR(i->i_mode))) {
   93:                 return -EBADF;
   94:         }
   95:
   96:         blksize = i->sb->s_blocksize;
   97:         if(fd_table->offset > i->i_size) {
   98:                 fd_table->offset = i->i_size;
   99:         }
  100:
  101:         base_dirent_len = sizeof(dirent->d_ino) + sizeof(dirent->d_off) + sizeof
(dirent->d_reclen);
  102:         doffset = offset = size = 0;
  103:
  104:         while(doffset < count) {
  105:                 if((block = bmap(i, fd_table->offset, FOR_READING)) < 0) {
  106:                         return block;
  107:                 }
  108:                 if(block) {
  109:                         if(!(buf = bread(i->dev, block, blksize))) {
  110:                                 return -EIO;
  111:                         }
  112:
  113:                         doffset = fd_table->offset;
  114:                         offset = fd_table->offset % blksize;
  115:                         while(doffset < i->i_size && offset < blksize) {
  116:                                 d = (struct minix_dir_entry *)(buf->data + offse
t);
  117:                                 if(d->inode) {
  118:                                         dirent_len = (base_dirent_len + (strlen(
d->name) + 1)) + 3;
  119:                                         dirent_len &= ~3;       /* round up */
  120:                                         dirent->d_ino = d->inode;
  121:                                         if((size + dirent_len) < count) {
  122:                                                 dirent->d_off = doffset;
  123:                                                 dirent->d_reclen = dirent_len;
  124:                                                 memcpy_b(dirent->d_name, d->name
, strlen(d->name));
  125:                                                 dirent->d_name[strlen(d->name)]
= NULL;
  126:                                                 dirent = (struct dirent *)((char
```

fs/minix/dir.c


```
 *)dirent + dirent_len);
  127:                                                  size += dirent_len;
  128:                                          } else {
  129:                                                  break;
  130:                                          }
  131:                                  }
  132:                                  doffset += i->sb->u.minix.dirsize;
  133:                                  offset += i->sb->u.minix.dirsize;
  134:                          }
  135:                          brelse(buf);
  136:                  }
  137:                  fd_table->offset &= ~(blksize - 1);
  138:                  doffset = fd_table->offset;
  139:                  fd_table->offset += offset;
  140:                  doffset += blksize;
  141:          }
  142:
  143:          return size;
  144: }
```

fs/minix/file.c


```
    1: /*
    2:  * fiwix/fs/minix/file.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/types.h>
   10: #include <fiwix/errno.h>
   11: #include <fiwix/buffer.h>
   12: #include <fiwix/fs.h>
   13: #include <fiwix/filesystems.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/mman.h>
   16: #include <fiwix/fcntl.h>
   17: #include <fiwix/stdio.h>
   18: #include <fiwix/string.h>
   19:
   20: struct fs_operations minix_file_fsop = {
   21:         0,
   22:         0,
   23:
   24:         minix_file_open,
   25:         minix_file_close,
   26:         file_read,
   27:         minix_file_write,
   28:         NULL,                   /* ioctl */
   29:         minix_file_lseek,
   30:         NULL,                   /* readdir */
   31:         NULL,                   /* mmap */
   32:         NULL,                   /* select */
   33:
   34:         NULL,                   /* readlink */
   35:         NULL,                   /* followlink */
   36:         minix_bmap,
   37:         NULL,                   /* lookup */
   38:         NULL,                   /* rmdir */
   39:         NULL,                   /* link */
   40:         NULL,                   /* unlink */
   41:         NULL,                   /* symlink */
   42:         NULL,                   /* mkdir */
   43:         NULL,                   /* mknod */
   44:         minix_truncate,
   45:         NULL,                   /* create */
   46:         NULL,                   /* rename */
   47:
   48:         NULL,                   /* read_block */
   49:         NULL,                   /* write_block */
   50:
   51:         NULL,                   /* read_inode */
   52:         NULL,                   /* write_inode */
   53:         NULL,                   /* ialloc */
   54:         NULL,                   /* ifree */
   55:         NULL,                   /* statfs */
   56:         NULL,                   /* read_superblock */
   57:         NULL,                   /* remount_fs */
   58:         NULL,                   /* write_superblock */
   59:         NULL                    /* release_superblock */
   60: };
   61:
   62: int minix_file_open(struct inode *i, struct fd *fd_table)
   63: {
   64:         if(fd_table->flags & O_APPEND) {
   65:                 fd_table->offset = i->i_size;
   66:         } else {
   67:                 fd_table->offset = 0;
```

fs/minix/file.c

```
   68:         }
   69:         if(fd_table->flags & O_TRUNC) {
   70:                 i->i_size = 0;
   71:                 minix_truncate(i, 0);
   72:         }
   73:         return 0;
   74: }
   75:
   76: int minix_file_close(struct inode *i, struct fd *fd_table)
   77: {
   78:         return 0;
   79: }
   80:
   81: int minix_file_write(struct inode *i, struct fd *fd_table, const char *buffer, _
_size_t count)
   82: {
   83:         __blk_t block;
   84:         __off_t total_written;
   85:         unsigned int boffset, bytes;
   86:         int blksize;
   87:         struct buffer *buf;
   88:
   89:         inode_lock(i);
   90:
   91:         blksize = i->sb->s_blocksize;
   92:         total_written = 0;
   93:
   94:         if(fd_table->flags & O_APPEND) {
   95:                 fd_table->offset = i->i_size;
   96:         }
   97:
   98:         while(total_written < count) {
   99:                 boffset = fd_table->offset % blksize;
  100:                 if((block = bmap(i, fd_table->offset, FOR_WRITING)) < 0) {
  101:                         inode_unlock(i);
  102:                         return block;
  103:                 }
  104:                 bytes = blksize - boffset;
  105:                 bytes = MIN(bytes, (count - total_written));
  106:                 if(!(buf = bread(i->dev, block, blksize))) {
  107:                         inode_unlock(i);
  108:                         return -EIO;
  109:                 }
  110:                 memcpy_b(buf->data + boffset, buffer + total_written, bytes);
  111:                 update_page_cache(i, fd_table->offset, buffer + total_written, b
ytes);
  112:                 bwrite(buf);
  113:                 total_written += bytes;
  114:                 boffset += bytes;
  115:                 boffset %= blksize;
  116:                 fd_table->offset += bytes;
  117:         }
  118:
  119:         if(fd_table->offset > i->i_size) {
  120:                 i->i_size = fd_table->offset;
  121:         }
  122:         i->i_ctime = CURRENT_TIME;
  123:         i->i_mtime = CURRENT_TIME;
  124:         i->dirty = 1;
  125:
  126:         inode_unlock(i);
  127:         return total_written;
  128: }
  129:
  130: int minix_file_lseek(struct inode *i, __off_t offset)
  131: {
  132:         return offset;
```

fs/minix/file.c


```
133: }
```

fs/minix/inode.c


```
 1: /*
 2:  * fiwix/fs/minix/inode.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/kernel.h>
 9: #include <fiwix/fs.h>
10: #include <fiwix/filesystems.h>
11: #include <fiwix/fs_minix.h>
12: #include <fiwix/fs_pipe.h>
13: #include <fiwix/statfs.h>
14: #include <fiwix/sleep.h>
15: #include <fiwix/stat.h>
16: #include <fiwix/sched.h>
17: #include <fiwix/buffer.h>
18: #include <fiwix/process.h>
19: #include <fiwix/errno.h>
20: #include <fiwix/stdio.h>
21: #include <fiwix/string.h>
22:
23: int minix_read_inode(struct inode *i)
24: {
25:         if(i->sb->u.minix.version == 1) {
26:                 return v1_minix_read_inode(i);
27:         }
28:
29:         return v2_minix_read_inode(i);
30: }
31:
32: int minix_write_inode(struct inode *i)
33: {
34:         if(i->sb->u.minix.version == 1) {
35:                 return v1_minix_write_inode(i);
36:         }
37:
38:         return v2_minix_write_inode(i);
39: }
40:
41: int minix_ialloc(struct inode *i, int mode)
42: {
43:         if(i->sb->u.minix.version == 1) {
44:                 return v1_minix_ialloc(i, mode);
45:         }
46:
47:         return v2_minix_ialloc(i, mode);
48: }
49:
50: void minix_ifree(struct inode *i)
51: {
52:         if(i->sb->u.minix.version == 1) {
53:                 return v1_minix_ifree(i);
54:         }
55:
56:         return v2_minix_ifree(i);
57: }
58:
59: int minix_bmap(struct inode *i, __off_t offset, int mode)
60: {
61:         if(i->sb->u.minix.version == 1) {
62:                 return v1_minix_bmap(i, offset, mode);
63:         }
64:
65:         return v2_minix_bmap(i, offset, mode);
66: }
67:
```

fs/minix/inode.c


```
68: int minix_truncate(struct inode *i, __off_t length)
69: {
70:         if(i->sb->u.minix.version == 1) {
71:                 return v1_minix_truncate(i, length);
72:         }
73:
74:         return v2_minix_truncate(i, length);
75: }
```

fs/minix/Makefile


```
    1: # fiwix/fs/minix/Makefile
    2: #
    3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
    4: # Distributed under the terms of the Fiwix License.
    5: #
    6:
    7: .S.o:
    8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
    9: .c.o:
   10:         $(CC) $(CFLAGS) -c -o $@ $<
   11:
   12: OBJS = super.o bitmaps.o inode.o namei.o symlink.o dir.o file.o v1_inode.o v2_in
ode.o
   13:
   14: minix:  $(OBJS)
   15:         $(LD) $(LDFLAGS) -r $(OBJS) -o minix.o
   16:
   17: clean:
   18:         rm -f *.o
   19:
```

fs/minix/namei.c


```
    1: /*
    2:  * fiwix/fs/minix/namei.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/types.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/filesystems.h>
   12: #include <fiwix/fs_minix.h>
   13: #include <fiwix/buffer.h>
   14: #include <fiwix/errno.h>
   15: #include <fiwix/stat.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: static int is_dir_empty(struct inode *dir)
   20: {
   21:         __blk_t block;
   22:         unsigned int blksize;
   23:         unsigned int offset, doffset;
   24:         struct buffer *buf;
   25:         struct minix_dir_entry *d;
   26:
   27:         blksize = dir->sb->s_blocksize;
   28:         doffset = dir->sb->u.minix.dirsize * 2; /* accept only "." and ".." */
   29:         offset = 0;
   30:
   31:         while(offset < dir->i_size) {
   32:                 if((block = bmap(dir, offset, FOR_READING)) < 0) {
   33:                         break;
   34:                 }
   35:                 if(block) {
   36:                         if(!(buf = bread(dir->dev, block, blksize))) {
   37:                                 break;
   38:                         }
   39:                         do {
   40:                                 if(doffset + offset >= dir->i_size) {
   41:                                         break;
   42:                                 }
   43:                                 d = (struct minix_dir_entry *)(buf->data + doffs
et);
   44:                                 if(d->inode) {
   45:                                         brelse(buf);
   46:                                         return 0;
   47:                                 }
   48:                                 doffset += dir->sb->u.minix.dirsize;
   49:                         } while(doffset < blksize);
   50:                         brelse(buf);
   51:                         offset += blksize;
   52:                         doffset = 0;
   53:                 } else {
   54:                         break;
   55:                 }
   56:         }
   57:
   58:         return 1;
   59: }
   60:
   61: /* finds the entry 'name' with inode 'i' in the directory 'dir' */
   62: static struct buffer * find_dir_entry(struct inode *dir, struct inode *i, struct
 minix_dir_entry **d_res, char *name)
   63: {
   64:         __blk_t block;
   65:         unsigned int blksize;
```

fs/minix/namei.c


```
   66:         unsigned int offset, doffset;
   67:         struct buffer *buf;
   68:
   69:         blksize = dir->sb->s_blocksize;
   70:         offset = 0;
   71:
   72:         while(offset < dir->i_size) {
   73:                 if((block = bmap(dir, offset, FOR_READING)) < 0) {
   74:                         break;
   75:                 }
   76:                 if(block) {
   77:                         if(!(buf = bread(dir->dev, block, blksize))) {
   78:                                 break;
   79:                         }
   80:                         doffset = 0;
   81:                         do {
   82:                                 *d_res = (struct minix_dir_entry *)(buf->data +
doffset);
   83:                                 if(!i) {
   84:                                         /* returns the first empty entry */
   85:                                         if(!(*d_res)->inode || (doffset + offset
 >= dir->i_size)) {
   86:                                                 /* the directory grows by direct
ory entry size */
   87:                                                 if(doffset + offset >= dir->i_si
ze) {
   88:                                                         dir->i_size += dir->sb->
u.minix.dirsize;
   89:                                                 }
   90:                                                 return buf;
   91:                                         }
   92:                                 } else {
   93:                                         if((*d_res)->inode == i->inode) {
   94:                                                 /* returns the first matching in
ode */
   95:                                                 if(!name) {
   96:                                                         return buf;
   97:                                                 }
   98:                                                 /* returns the matching inode an
d name */
   99:                                                 if(!strcmp((*d_res)->name, name)
) {
  100:                                                         return buf;
  101:                                                 }
  102:                                         }
  103:                                 }
  104:                                 doffset += dir->sb->u.minix.dirsize;
  105:                         } while(doffset < blksize);
  106:                         brelse(buf);
  107:                         offset += blksize;
  108:                 } else {
  109:                         break;
  110:                 }
  111:         }
  112:
  113:         *d_res = NULL;
  114:         return NULL;
  115: }
  116:
  117: static struct buffer * add_dir_entry(struct inode *dir, struct minix_dir_entry *
*d_res)
  118: {
  119:         __blk_t block;
  120:         struct buffer *buf;
  121:
  122:         if(!(buf = find_dir_entry(dir, NULL, d_res, NULL))) {
  123:                 if((block = bmap(dir, dir->i_size, FOR_WRITING)) < 0) {
```

fs/minix/namei.c


```
  124:                          return NULL;
  125:                  }
  126:                  if(!(buf = bread(dir->dev, block, dir->sb->s_blocksize))) {
  127:                          return NULL;
  128:                  }
  129:                  *d_res = (struct minix_dir_entry *)buf->data;
  130:                  dir->i_size += dir->sb->u.minix.dirsize;
  131:          }
  132:
  133:          return buf;
  134: }
  135:
  136: static int is_prefix(struct inode *dir_new, struct inode *i_old)
  137: {
  138:          __ino_t inode;
  139:          int errno;
  140:
  141:          errno = 0;
  142:          for(;;) {
  143:                  if(dir_new == i_old) {
  144:                          errno = 1;
  145:                          break;
  146:                  }
  147:                  inode = dir_new->inode;
  148:                  if(minix_lookup("..", dir_new, &dir_new)) {
  149:                          break;
  150:                  }
  151:                  iput(dir_new);  /* lookup eats 1 dir_new */
  152:                  if(dir_new->inode == inode) {
  153:                          break;
  154:                  }
  155:          }
  156:          return errno;
  157: }
  158:
  159: int minix_lookup(const char *name, struct inode *dir, struct inode **i_res)
  160: {
  161:          __blk_t block;
  162:          unsigned int blksize;
  163:          unsigned int offset, doffset;
  164:          struct buffer *buf;
  165:          struct minix_dir_entry *d;
  166:          __ino_t inode;
  167:
  168:          blksize = dir->sb->s_blocksize;
  169:          inode = offset = 0;
  170:          dir->count++;
  171:
  172:          while(offset < dir->i_size && !inode) {
  173:                  if((block = bmap(dir, offset, FOR_READING)) < 0) {
  174:                          iput(dir);
  175:                          return block;
  176:                  }
  177:                  if(block) {
  178:                          if(!(buf = bread(dir->dev, block, blksize))) {
  179:                                  iput(dir);
  180:                                  return -EIO;
  181:                          }
  182:                          doffset = 0;
  183:                          do {
  184:                                  d = (struct minix_dir_entry *)(buf->data + doffs
et);
  185:                                  if(d->inode) {
  186:                                          if(strlen(d->name) == strlen(name)) {
  187:                                                  if(!(strcmp(d->name, name))) {
  188:                                                          inode = d->inode;
  189:                                                  }
```

fs/minix/namei.c


```
190:                                            }
191:                                    }
192:                                    doffset += dir->sb->u.minix.dirsize;
193:                            } while((doffset < blksize) && (!inode));
194:
195:                            brelse(buf);
196:                            if(inode) {
197:                                    if(!(*i_res = iget(dir->sb, inode))) {
198:                                            iput(dir);
199:                                            return -EACCES;
200:                                    }
201:                                    iput(dir);
202:                                    return 0;
203:                            }
204:                            offset += blksize;
205:                    } else {
206:                            break;
207:                    }
208:            }
209:            iput(dir);
210:            return -ENOENT;
211: }
212:
213: int minix_rmdir(struct inode *dir, struct inode *i)
214: {
215:            struct buffer *buf;
216:            struct minix_dir_entry *d;
217:
218:            inode_lock(i);
219:
220:            if(!is_dir_empty(i)) {
221:                    inode_unlock(i);
222:                    return -ENOTEMPTY;
223:            }
224:
225:            inode_lock(dir);
226:
227:            if(!(buf = find_dir_entry(dir, i, &d, NULL))) {
228:                    inode_unlock(i);
229:                    inode_unlock(dir);
230:                    return -ENOENT;
231:            }
232:
233:            d->inode = 0;
234:            i->i_nlink = 0;
235:            dir->i_nlink--;
236:
237:            i->i_ctime = CURRENT_TIME;
238:            dir->i_mtime = CURRENT_TIME;
239:            dir->i_ctime = CURRENT_TIME;
240:
241:            i->dirty = 1;
242:            dir->dirty = 1;
243:
244:            bwrite(buf);
245:
246:            inode_unlock(i);
247:            inode_unlock(dir);
248:            return 0;
249: }
250:
251: int minix_link(struct inode *i_old, struct inode *dir_new, char *name)
252: {
253:            struct buffer *buf;
254:            struct minix_dir_entry *d;
255:            int n;
256:
```

fs/minix/namei.c


```
257:         inode_lock(i_old);
258:         inode_lock(dir_new);
259:
260:         if(!(buf = add_dir_entry(dir_new, &d))) {
261:                 inode_unlock(i_old);
262:                 inode_unlock(dir_new);
263:                 return -ENOSPC;
264:         }
265:
266:         d->inode = i_old->inode;
267:         for(n = 0; n < i_old->sb->u.minix.namelen; n++) {
268:                 d->name[n] = name[n];
269:                 if(!name[n]) {
270:                         break;
271:                 }
272:         }
273:         for(; n < i_old->sb->u.minix.namelen; n++) {
274:                 d->name[n] = 0;
275:         }
276:
277:         i_old->i_nlink++;
278:         i_old->i_ctime = CURRENT_TIME;
279:         dir_new->i_mtime = CURRENT_TIME;
280:         dir_new->i_ctime = CURRENT_TIME;
281:
282:         i_old->dirty = 1;
283:         dir_new->dirty = 1;
284:
285:         bwrite(buf);
286:
287:         inode_unlock(i_old);
288:         inode_unlock(dir_new);
289:         return 0;
290: }
291:
292: int minix_unlink(struct inode *dir, struct inode *i, char *name)
293: {
294:         struct buffer *buf;
295:         struct minix_dir_entry *d;
296:
297:         inode_lock(dir);
298:         inode_lock(i);
299:
300:         if(!(buf = find_dir_entry(dir, i, &d, name))) {
301:                 inode_unlock(dir);
302:                 inode_unlock(i);
303:                 return -ENOENT;
304:         }
305:
306:         d->inode = 0;
307:         i->i_nlink--;
308:
309:         i->i_ctime = CURRENT_TIME;
310:         dir->i_mtime = CURRENT_TIME;
311:         dir->i_ctime = CURRENT_TIME;
312:
313:         i->dirty = 1;
314:         dir->dirty = 1;
315:
316:         bwrite(buf);
317:
318:         inode_unlock(dir);
319:         inode_unlock(i);
320:         return 0;
321: }
322:
323: int minix_symlink(struct inode *dir, char *name, char *oldname)
```

fs/minix/namei.c


```
324: {
325:         struct buffer *buf, *buf_new;
326:         struct inode *i;
327:         struct minix_dir_entry *d;
328:         unsigned int blksize;
329:         int n, block;
330:         char c;
331:
332:         inode_lock(dir);
333:
334:         if(!(i = ialloc(dir->sb, S_IFLNK))) {
335:                 inode_unlock(dir);
336:                 return -ENOSPC;
337:         }
338:
339:         i->i_mode = S_IFLNK | (S_IRWXU | S_IRWXG | S_IRWXO);
340:         i->i_uid = current->euid;
341:         i->i_gid = current->egid;
342:         i->i_nlink = 1;
343:         i->dev = dir->dev;
344:         i->count = 1;
345:         i->fsop = &minix_symlink_fsop;
346:         i->dirty = 1;
347:
348:         block = minix_balloc(dir->sb);
349:         if(block < 0) {
350:                 i->i_nlink = 0;
351:                 iput(i);
352:                 inode_unlock(dir);
353:                 return -ENOSPC;
354:         }
355:
356:         if(i->sb->u.minix.version == 1) {
357:                 i->u.minix.u.i1_zone[0] = block;
358:         } else {
359:                 i->u.minix.u.i2_zone[0] = block;
360:         }
361:         blksize = dir->sb->s_blocksize;
362:         if(!(buf_new = bread(dir->dev, block, blksize))) {
363:                 minix_bfree(dir->sb, block);
364:                 i->i_nlink = 0;
365:                 iput(i);
366:                 inode_unlock(dir);
367:                 return -EIO;
368:         }
369:
370:         if(!(buf = add_dir_entry(dir, &d))) {
371:                 minix_bfree(dir->sb, block);
372:                 i->i_nlink = 0;
373:                 iput(i);
374:                 inode_unlock(dir);
375:                 return -ENOSPC;
376:         }
377:
378:         d->inode = i->inode;
379:         for(n = 0; n < i->sb->u.minix.namelen; n++) {
380:                 d->name[n] = name[n];
381:                 if(!name[n]) {
382:                         break;
383:                 }
384:         }
385:         for(; n < i->sb->u.minix.namelen; n++) {
386:                 d->name[n] = 0;
387:         }
388:
389:         for(n = 0; n < NAME_MAX; n++) {
390:                 if((c = oldname[n])) {
```

fs/minix/namei.c


```
391:                            buf_new->data[n] = c;
392:                            continue;
393:                    }
394:                    break;
395:            }
396:            buf_new->data[n] = 0;
397:            i->i_size = n;
398:
399:            dir->i_mtime = CURRENT_TIME;
400:            dir->i_ctime = CURRENT_TIME;
401:            dir->dirty = 1;
402:
403:            bwrite(buf);
404:            bwrite(buf_new);
405:            iput(i);
406:            inode_unlock(dir);
407:            return 0;
408: }
409:
410: int minix_mkdir(struct inode *dir, char *name, __mode_t mode)
411: {
412:            struct buffer *buf, *buf_new;
413:            struct inode *i;
414:            struct minix_dir_entry *d, *d_new;
415:            unsigned int blksize;
416:            int n, block;
417:
418:            if(strlen(name) > dir->sb->u.minix.namelen) {
419:                    return -ENAMETOOLONG;
420:            }
421:
422:            inode_lock(dir);
423:
424:            if(!(i = ialloc(dir->sb, S_IFDIR))) {
425:                    inode_unlock(dir);
426:                    return -ENOSPC;
427:            }
428:
429:            i->i_mode = ((mode & (S_IRWXU | S_IRWXG | S_IRWXO)) & ~current->umask);
430:            i->i_mode |= S_IFDIR;
431:            i->i_uid = current->euid;
432:            i->i_gid = current->egid;
433:            i->i_nlink = 1;
434:            i->dev = dir->dev;
435:            i->count = 1;
436:            i->fsop = &minix_dir_fsop;
437:            i->dirty = 1;
438:
439:            if((block = bmap(i, 0, FOR_WRITING)) < 0) {
440:                    i->i_nlink = 0;
441:                    iput(i);
442:                    inode_unlock(dir);
443:                    return -ENOSPC;
444:            }
445:
446:            blksize = dir->sb->s_blocksize;
447:            if(!(buf_new = bread(i->dev, block, blksize))) {
448:                    minix_bfree(dir->sb, block);
449:                    i->i_nlink = 0;
450:                    iput(i);
451:                    inode_unlock(dir);
452:                    return -EIO;
453:            }
454:
455:            if(!(buf = add_dir_entry(dir, &d))) {
456:                    minix_bfree(dir->sb, block);
457:                    i->i_nlink = 0;
```

fs/minix/namei.c


```
458:                    iput(i);
459:                    inode_unlock(dir);
460:                    return -ENOSPC;
461:            }
462:
463:            d->inode = i->inode;
464:            for(n = 0; n < i->sb->u.minix.namelen; n++) {
465:                    d->name[n] = name[n];
466:                    if(!name[n] || name[n] == '/') {
467:                            break;
468:                    }
469:            }
470:            for(; n < i->sb->u.minix.namelen; n++) {
471:                    d->name[n] = 0;
472:            }
473:
474:            d_new = (struct minix_dir_entry *)buf_new->data;
475:            d_new->inode = i->inode;
476:            d_new->name[0] = '.';
477:            d_new->name[1] = 0;
478:            i->i_size += i->sb->u.minix.dirsize;
479:            i->i_nlink++;
480:            d_new = (struct minix_dir_entry *)(buf_new->data + i->sb->u.minix.dirsiz
e);
481:            d_new->inode = dir->inode;
482:            d_new->name[0] = '.';
483:            d_new->name[1] = '.';
484:            d_new->name[2] = 0;
485:            i->i_size += i->sb->u.minix.dirsize;
486:
487:            dir->i_mtime = CURRENT_TIME;
488:            dir->i_ctime = CURRENT_TIME;
489:            dir->i_nlink++;
490:            dir->dirty = 1;
491:
492:            bwrite(buf);
493:            bwrite(buf_new);
494:            iput(i);
495:            inode_unlock(dir);
496:            return 0;
497: }
498:
499: int minix_mknod(struct inode *dir, char *name, __mode_t mode, __dev_t dev)
500: {
501:            struct buffer *buf;
502:            struct inode *i;
503:            struct minix_dir_entry *d;
504:            int n;
505:
506:            inode_lock(dir);
507:
508:            if(!(i = ialloc(dir->sb, mode & S_IFMT))) {
509:                    inode_unlock(dir);
510:                    return -ENOSPC;
511:            }
512:
513:            if(!(buf = add_dir_entry(dir, &d))) {
514:                    i->i_nlink = 0;
515:                    iput(i);
516:                    inode_unlock(dir);
517:                    return -ENOSPC;
518:            }
519:
520:            d->inode = i->inode;
521:            for(n = 0; n < i->sb->u.minix.namelen; n++) {
522:                    d->name[n] = name[n];
523:                    if(!name[n]) {
```

fs/minix/namei.c

```
  524:                          break;
  525:                  }
  526:          }
  527:          for(; n < i->sb->u.minix.namelen; n++) {
  528:                  d->name[n] = 0;
  529:          }
  530:
  531:          i->i_mode = (mode & ~current->umask) & ~S_IFMT;
  532:          i->i_uid = current->euid;
  533:          i->i_gid = current->egid;
  534:          i->i_nlink = 1;
  535:          i->dev = dir->dev;
  536:          i->count = 1;
  537:          i->dirty = 1;
  538:
  539:          switch(mode & S_IFMT) {
  540:                  case S_IFCHR:
  541:                          i->fsop = &def_chr_fsop;
  542:                          i->rdev = dev;
  543:                          i->i_mode |= S_IFCHR;
  544:                          break;
  545:                  case S_IFBLK:
  546:                          i->fsop = &def_blk_fsop;
  547:                          i->rdev = dev;
  548:                          i->i_mode |= S_IFBLK;
  549:                          break;
  550:                  case S_IFIFO:
  551:                          i->fsop = &pipefs_fsop;
  552:                          i->i_mode |= S_IFIFO;
  553:                          /* it's a union so we need to clear pipefs_i */
  554:                          memset_b(&i->u.pipefs, NULL, sizeof(struct pipefs_inode)
);
  555:                          break;
  556:          }
  557:
  558:          dir->i_mtime = CURRENT_TIME;
  559:          dir->i_ctime = CURRENT_TIME;
  560:          dir->dirty = 1;
  561:
  562:          bwrite(buf);
  563:          iput(i);
  564:          inode_unlock(dir);
  565:          return 0;
  566: }
  567:
  568: int minix_create(struct inode *dir, char *name, __mode_t mode, struct inode **i_
res)
  569: {
  570:          struct buffer *buf;
  571:          struct inode *i;
  572:          struct minix_dir_entry *d;
  573:          int n;
  574:
  575:          if(IS_RDONLY_FS(dir)) {
  576:                  return -EROFS;
  577:          }
  578:
  579:          inode_lock(dir);
  580:
  581:          if(!(i = ialloc(dir->sb, S_IFREG))) {
  582:                  inode_unlock(dir);
  583:                  return -ENOSPC;
  584:          }
  585:
  586:          if(!(buf = add_dir_entry(dir, &d))) {
  587:                  i->i_nlink = 0;
  588:                  iput(i);
```


fs/minix/namei.c


```
  589:                  inode_unlock(dir);
  590:                  return -ENOSPC;
  591:          }
  592:
  593:          d->inode = i->inode;
  594:          for(n = 0; n < i->sb->u.minix.namelen; n++) {
  595:                  d->name[n] = name[n];
  596:                  if(!name[n]) {
  597:                          break;
  598:                  }
  599:          }
  600:          for(; n < i->sb->u.minix.namelen; n++) {
  601:                  d->name[n] = 0;
  602:          }
  603:
  604:          i->i_mode = (mode & ~current->umask) & ~S_IFMT;
  605:          i->i_mode |= S_IFREG;
  606:          i->i_uid = current->euid;
  607:          i->i_gid = current->egid;
  608:          i->i_nlink = 1;
  609:          i->dev = dir->dev;
  610:          i->fsop = &minix_file_fsop;
  611:          i->count = 1;
  612:          i->dirty = 1;
  613:
  614:          dir->i_mtime = CURRENT_TIME;
  615:          dir->i_ctime = CURRENT_TIME;
  616:          dir->dirty = 1;
  617:
  618:          *i_res = i;
  619:          bwrite(buf);
  620:          inode_unlock(dir);
  621:          return 0;
  622: }
  623:
  624: int minix_rename(struct inode *i_old, struct inode *dir_old, struct inode *i_new
, struct inode *dir_new, char *oldpath, char *newpath)
  625: {
  626:          struct buffer *buf_old, *buf_new;
  627:          struct minix_dir_entry *d_old, *d_new;
  628:          int errno;
  629:
  630:          errno = 0;
  631:          buf_new = NULL;
  632:
  633:          if(is_prefix(dir_new, i_old)) {
  634:                  return -EINVAL;
  635:          }
  636:
  637:          inode_lock(i_old);
  638:          inode_lock(dir_old);
  639:          if(dir_old != dir_new) {
  640:                  inode_lock(dir_new);
  641:          }
  642:
  643:          if(!(buf_old = find_dir_entry(dir_old, i_old, &d_old, oldpath))) {
  644:                  errno = -ENOENT;
  645:                  goto end;
  646:          }
  647:          if(dir_old == dir_new) {
  648:                  buf_old->locked = 0;
  649:          }
  650:
  651:          if(i_new) {
  652:                  if(S_ISDIR(i_old->i_mode)) {
  653:                          if(!is_dir_empty(i_new)) {
  654:                                  brelse(buf_old);
```

fs/minix/namei.c


```
655:                                    errno = -ENOTEMPTY;
656:                                    goto end;
657:                            }
658:                    }
659:                    if(!(buf_new = find_dir_entry(dir_new, i_new, &d_new, newpath)))
 {
660:                            brelse(buf_old);
661:                            errno = -ENOENT;
662:                            goto end;
663:                    }
664:            } else {
665:                    if(!(buf_new = add_dir_entry(dir_new, &d_new))) {
666:                            brelse(buf_old);
667:                            errno = -ENOSPC;
668:                            goto end;
669:                    }
670:            }
671:            if(i_new) {
672:                    i_new->i_nlink--;
673:            } else {
674:                    i_new = i_old;
675:                    strcpy(d_new->name, newpath);
676:            }
677:
678:            d_old->inode = 0;
679:            d_new->inode = i_old->inode;
680:            dir_new->i_mtime = CURRENT_TIME;
681:            dir_new->i_ctime = CURRENT_TIME;
682:            i_new->dirty = 1;
683:            dir_new->dirty = 1;
684:
685:            dir_old->i_mtime = CURRENT_TIME;
686:            dir_old->i_ctime = CURRENT_TIME;
687:            i_old->dirty = 1;
688:            dir_old->dirty = 1;
689:
690:            bwrite(buf_old);
691:            if(buf_new) {
692:                    bwrite(buf_new);
693:            }
694:
695: end:
696:            inode_unlock(i_old);
697:            inode_unlock(dir_old);
698:            inode_unlock(dir_new);
699:            return errno;
700: }
```

fs/minix/super.c


```
/*
 * fiwix/fs/minix/super.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/kernel.h>
#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/fs.h>
#include <fiwix/filesystems.h>
#include <fiwix/fs_minix.h>
#include <fiwix/buffer.h>
#include <fiwix/sched.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct fs_operations minix_fsop = {
        FSOP_REQUIRES_DEV,
        NULL,

        NULL,                   /* open */
        NULL,                   /* close */
        NULL,                   /* read */
        NULL,                   /* write */
        NULL,                   /* ioctl */
        NULL,                   /* lseek */
        NULL,                   /* readdir */
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        NULL,                   /* bmap */
        NULL,                   /* lookup */
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        NULL,                   /* truncate */
        NULL,                   /* create */
        NULL,                   /* rename */

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        minix_read_inode,
        minix_write_inode,
        minix_ialloc,
        minix_ifree,
        minix_statfs,
        minix_read_superblock,
        minix_remount_fs,
        minix_write_superblock,
        minix_release_superblock
};

static void check_superblock(struct minix_super_block *sb)
{
        if(!(sb->s_state & MINIX_VALID_FS)) {
                printk("WARNING: filesystem not checked, fsck recommended.\n");
        }
        if(sb->s_state & MINIX_ERROR_FS) {
                printk("WARNING: filesystem contains errors, fsck recommended.\n
```

fs/minix/super.c


```
");
   68:         }
   69: }
   70:
   71: void minix_statfs(struct superblock *sb, struct statfs *statfsbuf)
   72: {
   73:         statfsbuf->f_type = sb->u.minix.sb.s_magic;
   74:         statfsbuf->f_bsize = sb->s_blocksize;
   75:         statfsbuf->f_blocks = sb->u.minix.nzones << sb->u.minix.sb.s_log_zone_si
ze;
   76:         statfsbuf->f_bfree = sb->u.minix.nzones - minix_count_free_blocks(sb);
   77:         statfsbuf->f_bavail = statfsbuf->f_bfree;
   78:
   79:         statfsbuf->f_files = sb->u.minix.sb.s_ninodes;
   80:         statfsbuf->f_ffree = sb->u.minix.sb.s_ninodes - minix_count_free_inodes(
sb);
   81:         /* statfsbuf->f_fsid = ? */
   82:         statfsbuf->f_namelen = sb->u.minix.namelen;
   83: }
   84:
   85: int minix_read_superblock(__dev_t dev, struct superblock *sb)
   86: {
   87:         struct buffer *buf;
   88:         int maps;
   89:
   90:         superblock_lock(sb);
   91:         if(!(buf = bread(dev, SUPERBLOCK, BLKSIZE_1K))) {
   92:                 printk("WARNING: %s(): I/O error on device %d,%d.\n", __FUNCTION
__, MAJOR(dev), MINOR(dev));
   93:                 superblock_unlock(sb);
   94:                 return -EIO;
   95:         }
   96:         memcpy_b(&sb->u.minix.sb, buf->data, sizeof(struct minix_super_block));
   97:
   98:         switch(sb->u.minix.sb.s_magic) {
   99:                 case MINIX_SUPER_MAGIC:
  100:                         sb->u.minix.namelen = 14;
  101:                         sb->u.minix.dirsize = sizeof(__u16) + sb->u.minix.namele
n;
  102:                         sb->u.minix.version = 1;
  103:                         sb->u.minix.nzones = sb->u.minix.sb.s_nzones;
  104:                         printk("minix v1 (14 char names) filesystem detected on
device %d,%d.\n", MAJOR(dev), MINOR(dev));
  105:                         break;
  106:                 case MINIX_SUPER_MAGIC2:
  107:                         sb->u.minix.namelen = 30;
  108:                         sb->u.minix.dirsize = sizeof(__u16) + sb->u.minix.namele
n;
  109:                         sb->u.minix.version = 1;
  110:                         sb->u.minix.nzones = sb->u.minix.sb.s_nzones;
  111:                         printk("minix v1 (30 char names) filesystem detected on
device %d,%d.\n", MAJOR(dev), MINOR(dev));
  112:                         break;
  113:                 case MINIX2_SUPER_MAGIC:
  114:                         sb->u.minix.namelen = 14;
  115:                         sb->u.minix.dirsize = sizeof(__u16) + sb->u.minix.namele
n;
  116:                         sb->u.minix.version = 2;
  117:                         sb->u.minix.nzones = sb->u.minix.sb.s_zones;
  118:                         printk("minix v2 (14 char names) filesystem detected on
device %d,%d.\n", MAJOR(dev), MINOR(dev));
  119:                         break;
  120:                 case MINIX2_SUPER_MAGIC2:
  121:                         sb->u.minix.namelen = 30;
  122:                         sb->u.minix.dirsize = sizeof(__u16) + sb->u.minix.namele
n;
  123:                         sb->u.minix.version = 2;
```

fs/minix/super.c


```
  124:                          sb->u.minix.nzones = sb->u.minix.sb.s_zones;
  125:                          printk("minix v2 (30 char names) filesystem detected on
device %d,%d.\n", MAJOR(dev), MINOR(dev));
  126:                          break;
  127:                  default:
  128:                          printk("ERROR: %s(): invalid filesystem type or bad supe
rblock on device %d,%d.\n", __FUNCTION__, MAJOR(dev), MINOR(dev));
  129:                          superblock_unlock(sb);
  130:                          brelse(buf);
  131:                          return -EINVAL;
  132:          }
  133:
  134:          sb->dev = dev;
  135:          sb->fsop = &minix_fsop;
  136:          sb->s_blocksize = BLKSIZE_1K << sb->u.minix.sb.s_log_zone_size;
  137:
  138:          if(sb->s_blocksize != BLKSIZE_1K) {
  139:                  printk("ERROR: %s(): block sizes > %d not supported in this file
system.\n", __FUNCTION__, BLKSIZE_1K);
  140:                  superblock_unlock(sb);
  141:                  brelse(buf);
  142:                  return -EINVAL;
  143:          }
  144:
  145:          /*
  146:          printk("s_ninodes       = %d\n", sb->u.minix.sb.s_ninodes);
  147:          printk("s_nzones        = %d (nzones = %d)\n", sb->u.minix.sb.s_nzones,
sb->u.minix.nzones);
  148:          printk("s_imap_blocks   = %d\n", sb->u.minix.sb.s_imap_blocks);
  149:          printk("s_zmap_blocks   = %d\n", sb->u.minix.sb.s_zmap_blocks);
  150:          printk("s_firstdatazone = %d\n", sb->u.minix.sb.s_firstdatazone);
  151:          printk("s_log_zone_size = %d\n", sb->u.minix.sb.s_log_zone_size);
  152:          printk("s_max_size      = %d\n", sb->u.minix.sb.s_max_size);
  153:          printk("s_magic         = %x\n", sb->u.minix.sb.s_magic);
  154:          printk("s_state         = %d\n", sb->u.minix.sb.s_state);
  155:          printk("s_zones         = %d\n", sb->u.minix.sb.s_zones);
  156:          */
  157:
  158:          /* Minix fs size is limited to: # of bitmaps * 8192 * 1024 */
  159:          if(sb->u.minix.version == 1) {
  160:                  maps = V1_MAX_BITMAP_BLOCKS;    /* 64MB limit */
  161:          }
  162:          if(sb->u.minix.version == 2) {
  163:                  maps = V2_MAX_BITMAP_BLOCKS;    /* 1GB limit */
  164:          }
  165:
  166:          if(sb->u.minix.sb.s_imap_blocks > maps) {
  167:                  printk("ERROR: %s(): number of imap blocks (%d) is greater than
%d!\n", __FUNCTION__, sb->u.minix.sb.s_imap_blocks, maps);
  168:                  superblock_unlock(sb);
  169:                  brelse(buf);
  170:                  return -EINVAL;
  171:          }
  172:          if(sb->u.minix.sb.s_zmap_blocks > maps) {
  173:                  printk("ERROR: %s(): number of zmap blocks (%d) is greater than
%d!\n", __FUNCTION__, sb->u.minix.sb.s_zmap_blocks, maps);
  174:                  superblock_unlock(sb);
  175:                  brelse(buf);
  176:                  return -EINVAL;
  177:          }
  178:
  179:          superblock_unlock(sb);
  180:
  181:          if(!(sb->root = iget(sb, MINIX_ROOT_INO))) {
  182:                  printk("ERROR: %s(): unable to get root inode.\n", __FUNCTION__)
;
  183:                  brelse(buf);
```

fs/minix/super.c


```
  184:                    return -EINVAL;
  185:            }
  186:
  187:            check_superblock(&sb->u.minix.sb);
  188:
  189:            if(!(sb->flags & MS_RDONLY)) {
  190:                    sb->u.minix.sb.s_state &= ~MINIX_VALID_FS;
  191:                    memcpy_b(buf->data, &sb->u.minix.sb, sizeof(struct minix_super_b
lock));
  192:                    bwrite(buf);
  193:            } else {
  194:                    brelse(buf);
  195:            }
  196:
  197:            return 0;
  198: }
  199:
  200: int minix_remount_fs(struct superblock *sb, int flags)
  201: {
  202:            struct buffer *buf;
  203:            struct minix_super_block *ms;
  204:
  205:            if((flags & MS_RDONLY) == (sb->flags & MS_RDONLY)) {
  206:                    return 0;
  207:            }
  208:
  209:            superblock_lock(sb);
  210:            if(!(buf = bread(sb->dev, SUPERBLOCK, BLKSIZE_1K))) {
  211:                    superblock_unlock(sb);
  212:                    return -EIO;
  213:            }
  214:            ms = (struct minix_super_block *)buf->data;
  215:
  216:            if(flags & MS_RDONLY) {
  217:                    /* switching from RW to RO */
  218:                    sb->u.minix.sb.s_state |= MINIX_VALID_FS;
  219:                    ms->s_state |= MINIX_VALID_FS;
  220:            } else {
  221:                    /* switching from RO to RW */
  222:                    check_superblock(ms);
  223:                    sb->u.minix.sb.s_state &= ~MINIX_VALID_FS;
  224:                    ms->s_state &= ~MINIX_VALID_FS;
  225:            }
  226:
  227:            sb->dirty = 1;
  228:            superblock_unlock(sb);
  229:            bwrite(buf);
  230:            return 0;
  231: }
  232:
  233: int minix_write_superblock(struct superblock *sb)
  234: {
  235:            struct buffer *buf;
  236:
  237:            superblock_lock(sb);
  238:            if(!(buf = bread(sb->dev, SUPERBLOCK, BLKSIZE_1K))) {
  239:                    superblock_unlock(sb);
  240:                    return -EIO;
  241:            }
  242:
  243:            memcpy_b(buf->data, &sb->u.minix.sb, sizeof(struct minix_super_block));
  244:            sb->dirty = 0;
  245:            superblock_unlock(sb);
  246:            bwrite(buf);
  247:            return 0;
  248: }
  249:
```

fs/minix/super.c


```
250: void minix_release_superblock(struct superblock *sb)
251: {
252:         if(sb->flags & MS_RDONLY) {
253:                 return;
254:         }
255:
256:         superblock_lock(sb);
257:
258:         sb->u.minix.sb.s_state |= MINIX_VALID_FS;
259:         sb->dirty = 1;
260:
261:         superblock_unlock(sb);
262: }
263:
264: int minix_init(void)
265: {
266:         return register_filesystem("minix", &minix_fsop);
267: }
```

fs/minix/symlink.c


```
    1: /*
    2:  * fiwix/fs/minix/symlink.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/errno.h>
   10: #include <fiwix/buffer.h>
   11: #include <fiwix/fs.h>
   12: #include <fiwix/filesystems.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/stdio.h>
   15: #include <fiwix/string.h>
   16:
   17: struct fs_operations minix_symlink_fsop = {
   18:         0,
   19:         0,
   20:
   21:         NULL,                   /* open */
   22:         NULL,                   /* close */
   23:         NULL,                   /* read */
   24:         NULL,                   /* write */
   25:         NULL,                   /* ioctl */
   26:         NULL,                   /* lseek */
   27:         NULL,                   /* readdir */
   28:         NULL,                   /* mmap */
   29:         NULL,                   /* select */
   30:
   31:         minix_readlink,
   32:         minix_followlink,
   33:         NULL,                   /* bmap */
   34:         NULL,                   /* lookup */
   35:         NULL,                   /* rmdir */
   36:         NULL,                   /* link */
   37:         NULL,                   /* unlink */
   38:         NULL,                   /* symlink */
   39:         NULL,                   /* mkdir */
   40:         NULL,                   /* mknod */
   41:         NULL,                   /* truncate */
   42:         NULL,                   /* create */
   43:         NULL,                   /* rename */
   44:
   45:         NULL,                   /* read_block */
   46:         NULL,                   /* write_block */
   47:
   48:         NULL,                   /* read_inode */
   49:         NULL,                   /* write_inode */
   50:         NULL,                   /* ialloc */
   51:         NULL,                   /* ifree */
   52:         NULL,                   /* statfs */
   53:         NULL,                   /* read_superblock */
   54:         NULL,                   /* remount_fs */
   55:         NULL,                   /* write_superblock */
   56:         NULL                    /* release_superblock */
   57: };
   58:
   59: int minix_readlink(struct inode *i, char *buffer, __size_t count)
   60: {
   61:         __u32 blksize;
   62:         struct buffer *buf;
   63:
   64:         if(!S_ISLNK(i->i_mode)) {
   65:                 printk("%s(): Oops, inode '%d' is not a symlink (!?).\n", __FUNC
TION__, i->inode);
   66:                 return 0;
```

fs/minix/symlink.c


```
   67:         }
   68:
   69:         inode_lock(i);
   70:         blksize = i->sb->s_blocksize;
   71:         count = MIN(count, i->i_size);
   72:         if(!count) {
   73:                 inode_unlock(i);
   74:                 return 0;
   75:         }
   76:         count = MIN(count, blksize);
   77:         if(i->sb->u.minix.version == 1) {
   78:                 if(!(buf = bread(i->dev, i->u.minix.u.i1_zone[0], blksize))) {
   79:                         inode_unlock(i);
   80:                         return -EIO;
   81:                 }
   82:         } else {
   83:                 if(!(buf = bread(i->dev, i->u.minix.u.i2_zone[0], blksize))) {
   84:                         inode_unlock(i);
   85:                         return -EIO;
   86:                 }
   87:         }
   88:         memcpy_b(buffer, buf->data, count);
   89:         brelse(buf);
   90:         buffer[count] = NULL;
   91:         inode_unlock(i);
   92:         return count;
   93: }
   94:
   95: int minix_followlink(struct inode *dir, struct inode *i, struct inode **i_res)
   96: {
   97:         struct buffer *buf;
   98:         char *name;
   99:         __ino_t errno;
  100:
  101:         if(!i) {
  102:                 return -ENOENT;
  103:         }
  104:
  105:         if(!S_ISLNK(i->i_mode)) {
  106:                 printk("%s(): Oops, inode '%d' is not a symlink (!?).\n", __FUNC
TION__, i->inode);
  107:                 return 0;
  108:         }
  109:
  110:         if(current->loopcnt > MAX_SYMLINKS) {
  111:                 printk("%s(): too many nested symbolic links!\n", __FUNCTION__);
  112:                 return -ELOOP;
  113:         }
  114:
  115:         inode_lock(i);
  116:         if(i->sb->u.minix.version == 1) {
  117:                 if(!(buf = bread(i->dev, i->u.minix.u.i1_zone[0], i->sb->s_block
size))) {
  118:                         inode_unlock(i);
  119:                         return -EIO;
  120:                 }
  121:         } else {
  122:                 if(!(buf = bread(i->dev, i->u.minix.u.i2_zone[0], i->sb->s_block
size))) {
  123:                         inode_unlock(i);
  124:                         return -EIO;
  125:                 }
  126:         }
  127:         name = buf->data;
  128:         inode_unlock(i);
  129:
  130:         current->loopcnt++;
```

fs/minix/symlink.c


```
131:         iput(i);
132:         brelse(buf);
133:         errno = parse_namei(name, dir, i_res, NULL, FOLLOW_LINKS);
134:         current->loopcnt--;
135:         return errno;
136: }
```

fs/minix/v1_inode.c


```
    1: /*
    2:  * fiwix/fs/minix/v1_inode.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/filesystems.h>
   11: #include <fiwix/fs_minix.h>
   12: #include <fiwix/fs_pipe.h>
   13: #include <fiwix/statfs.h>
   14: #include <fiwix/sleep.h>
   15: #include <fiwix/stat.h>
   16: #include <fiwix/sched.h>
   17: #include <fiwix/buffer.h>
   18: #include <fiwix/process.h>
   19: #include <fiwix/errno.h>
   20: #include <fiwix/stdio.h>
   21: #include <fiwix/string.h>
   22:
   23: #define BLOCKS_PER_IND_BLOCK(sb)        (sb->s_blocksize / sizeof(__u16))
   24: #define MINIX_INODES_PER_BLOCK(sb)      (sb->s_blocksize / sizeof(struct minix_i
node))
   25:
   26: #define MINIX_NDIR_BLOCKS               7
   27: #define MINIX_IND_BLOCK                 MINIX_NDIR_BLOCKS
   28: #define MINIX_DIND_BLOCK                (MINIX_NDIR_BLOCKS + 1)
   29:
   30: static void free_zone(struct inode *i, int block, int offset)
   31: {
   32:         int n;
   33:         struct buffer *buf;
   34:         __u16 *zone;
   35:
   36:         if(!(buf = bread(i->dev, block, i->sb->s_blocksize))) {
   37:                 printk("WARNING: %s(): error reading block %d.\n", __FUNCTION__,
 block);
   38:                 return;
   39:         }
   40:         zone = (__u16 *)buf->data;
   41:         for(n = offset; n < BLOCKS_PER_IND_BLOCK(i->sb); n++) {
   42:                 if(zone[n]) {
   43:                         minix_bfree(i->sb, zone[n]);
   44:                         zone[n] = 0;
   45:                 }
   46:         }
   47:         bwrite(buf);
   48: }
   49:
   50: int v1_minix_read_inode(struct inode *i)
   51: {
   52:         __ino_t block;
   53:         short int offset;
   54:         struct minix_inode *ii;
   55:         struct buffer *buf;
   56:         int errno;
   57:
   58:         block = 1 + SUPERBLOCK + i->sb->u.minix.sb.s_imap_blocks + i->sb->u.mini
x.sb.s_zmap_blocks + (i->inode - 1) / MINIX_INODES_PER_BLOCK(i->sb);
   59:
   60:         if(!(buf = bread(i->dev, block, i->sb->s_blocksize))) {
   61:                 return -EIO;
   62:         }
   63:         offset = (i->inode - 1) % MINIX_INODES_PER_BLOCK(i->sb);
   64:         ii = ((struct minix_inode *)buf->data) + offset;
```

fs/minix/v1_inode.c


```
   65:
   66:         i->i_mode = ii->i_mode;
   67:         i->i_uid = ii->i_uid;
   68:         i->i_size = ii->i_size;
   69:         i->i_atime = ii->i_time;
   70:         i->i_ctime = ii->i_time;
   71:         i->i_mtime = ii->i_time;
   72:         i->i_gid = ii->i_gid;
   73:         i->i_nlink = ii->i_nlinks;
   74:         memcpy_b(i->u.minix.u.i1_zone, ii->i_zone, sizeof(ii->i_zone));
   75:         i->count = 1;
   76:
   77:         errno = 0;
   78:         switch(i->i_mode & S_IFMT) {
   79:                 case S_IFCHR:
   80:                         i->fsop = &def_chr_fsop;
   81:                         i->rdev = ii->i_zone[0];
   82:                         break;
   83:                 case S_IFBLK:
   84:                         i->fsop = &def_blk_fsop;
   85:                         i->rdev = ii->i_zone[0];
   86:                         break;
   87:                 case S_IFIFO:
   88:                         i->fsop = &pipefs_fsop;
   89:                         /* it's a union so we need to clear pipefs_i */
   90:                         memset_b(&i->u.pipefs, NULL, sizeof(struct pipefs_inode)
);
   91:                         break;
   92:                 case S_IFDIR:
   93:                         i->fsop = &minix_dir_fsop;
   94:                         break;
   95:                 case S_IFREG:
   96:                         i->fsop = &minix_file_fsop;
   97:                         break;
   98:                 case S_IFLNK:
   99:                         i->fsop = &minix_symlink_fsop;
  100:                         break;
  101:                 case S_IFSOCK:
  102:                         i->fsop = NULL;
  103:                         break;
  104:                 default:
  105:                         printk("WARNING: %s(): invalid inode (%d) mode %o.\n", _
_FUNCTION__, i->inode, i->i_mode);
  106:                         errno = -ENOENT;
  107:                         break;
  108:         }
  109:
  110:         brelse(buf);
  111:         return errno;
  112: }
  113:
  114: int v1_minix_write_inode(struct inode *i)
  115: {
  116:         __ino_t block;
  117:         short int offset;
  118:         struct minix_inode *ii;
  119:         struct buffer *buf;
  120:
  121:         block = 1 + SUPERBLOCK + i->sb->u.minix.sb.s_imap_blocks + i->sb->u.mini
x.sb.s_zmap_blocks + (i->inode - 1) / MINIX_INODES_PER_BLOCK(i->sb);
  122:
  123:         if(!(buf = bread(i->dev, block, i->sb->s_blocksize))) {
  124:                 return -EIO;
  125:         }
  126:         offset = (i->inode - 1) % MINIX_INODES_PER_BLOCK(i->sb);
  127:         ii = ((struct minix_inode *)buf->data) + offset;
  128:
```

fs/minix/v1_inode.c


```
  129:          ii->i_mode = i->i_mode;
  130:          ii->i_uid = i->i_uid;
  131:          ii->i_size = i->i_size;
  132:          ii->i_time = i->i_mtime;
  133:          ii->i_gid = i->i_gid;
  134:          ii->i_nlinks = i->i_nlink;
  135:          if(S_ISCHR(i->i_mode) || S_ISBLK(i->i_mode)) {
  136:                  ii->i_zone[0] = i->rdev;
  137:          } else {
  138:                  memcpy_b(ii->i_zone, i->u.minix.u.i1_zone, sizeof(i->u.minix.u.i
1_zone));
  139:          }
  140:          i->dirty = 0;
  141:          bwrite(buf);
  142:          return 0;
  143: }
  144:
  145: int v1_minix_ialloc(struct inode *i, int mode)
  146: {
  147:          __blk_t offset;
  148:          int inode, errno;
  149:          struct superblock *sb;
  150:
  151:          sb = i->sb;
  152:          superblock_lock(sb);
  153:
  154:          offset = 1 + SUPERBLOCK;
  155:
  156:          if(!(inode = minix_find_first_zero(sb, offset, sb->u.minix.sb.s_ninodes,
 offset + sb->u.minix.sb.s_imap_blocks))) {
  157:                  superblock_unlock(sb);
  158:                  return -ENOSPC;
  159:          }
  160:
  161:          errno = minix_change_bit(SET_BIT, sb, offset, inode);
  162:
  163:          if(errno) {
  164:                  if(errno < 0) {
  165:                          printk("WARNING: %s(): unable to set inode %d.\n", __FUN
CTION__, inode);
  166:                          superblock_unlock(sb);
  167:                          return errno;
  168:                  } else {
  169:                          printk("WARNING: %s(): inode %d is already marked as use
d!\n", __FUNCTION__, inode);
  170:                  }
  171:          }
  172:
  173:          i->inode = inode;
  174:          i->i_atime = CURRENT_TIME;
  175:          i->i_mtime = CURRENT_TIME;
  176:          i->i_ctime = CURRENT_TIME;
  177:          superblock_unlock(sb);
  178:          return 0;
  179: }
  180:
  181: void v1_minix_ifree(struct inode *i)
  182: {
  183:          int errno;
  184:          struct superblock *sb;
  185:
  186:          minix_truncate(i, 0);
  187:
  188:          sb = i->sb;
  189:          superblock_lock(sb);
  190:
  191:          errno = minix_change_bit(CLEAR_BIT, i->sb, 1 + SUPERBLOCK, i->inode);
```

fs/minix/v1_inode.c


```
  192:
  193:         if(errno) {
  194:                 if(errno < 0) {
  195:                         printk("WARNING: %s(): unable to clear inode %d.\n", __F
UNCTION__, i->inode);
  196:                 } else {
  197:                         printk("WARNING: %s(): inode %d is already marked as fre
e!\n", __FUNCTION__, i->inode);
  198:                 }
  199:         }
  200:
  201:         i->i_size = 0;
  202:         i->i_mtime = CURRENT_TIME;
  203:         i->i_ctime = CURRENT_TIME;
  204:         i->dirty = 1;
  205:         superblock_unlock(sb);
  206: }
  207:
  208: int v1_minix_bmap(struct inode *i, __off_t offset, int mode)
  209: {
  210:         unsigned char level;
  211:         __u16 *indblock, *dindblock;
  212:         __blk_t block, iblock, dblock, newblock;
  213:         int blksize;
  214:         struct buffer *buf, *buf2, *buf3;
  215:
  216:         blksize = i->sb->s_blocksize;
  217:         block = offset / blksize;
  218:         level = 0;
  219:
  220:         if(block < MINIX_NDIR_BLOCKS) {
  221:                 level = MINIX_NDIR_BLOCKS - 1;
  222:         } else {
  223:                 if(block < (BLOCKS_PER_IND_BLOCK(i->sb) + MINIX_NDIR_BLOCKS)) {
  224:                         level = MINIX_IND_BLOCK;
  225:                 } else {
  226:                         level = MINIX_DIND_BLOCK;
  227:                 }
  228:                 block -= MINIX_NDIR_BLOCKS;
  229:         }
  230:
  231:         if(level < MINIX_NDIR_BLOCKS) {
  232:                 if(!i->u.minix.u.i1_zone[block] && mode == FOR_WRITING) {
  233:                         if((newblock = minix_balloc(i->sb)) < 0) {
  234:                                 return -ENOSPC;
  235:                         }
  236:                         /* initialize the new block */
  237:                         if(!(buf = bread(i->dev, newblock, blksize))) {
  238:                                 minix_bfree(i->sb, newblock);
  239:                                 return -EIO;
  240:                         }
  241:                         memset_b(buf->data, 0, blksize);
  242:                         bwrite(buf);
  243:                         i->u.minix.u.i1_zone[block] = newblock;
  244:                 }
  245:                 return i->u.minix.u.i1_zone[block];
  246:         }
  247:
  248:         if(!i->u.minix.u.i1_zone[level]) {
  249:                 if(mode == FOR_WRITING) {
  250:                         if((newblock = minix_balloc(i->sb)) < 0) {
  251:                                 return -ENOSPC;
  252:                         }
  253:                         /* initialize the new block */
  254:                         if(!(buf = bread(i->dev, newblock, blksize))) {
  255:                                 minix_bfree(i->sb, newblock);
  256:                                 return -EIO;
```

fs/minix/v1_inode.c


```
257:                            }
258:                            memset_b(buf->data, 0, blksize);
259:                            bwrite(buf);
260:                            i->u.minix.u.i1_zone[level] = newblock;
261:                    } else {
262:                            return 0;
263:                    }
264:            }
265:            if(!(buf = bread(i->dev, i->u.minix.u.i1_zone[level], blksize))) {
266:                    return -EIO;
267:            }
268:            indblock = (__u16 *)buf->data;
269:            dblock = block - BLOCKS_PER_IND_BLOCK(i->sb);
270:
271:            if(level == MINIX_DIND_BLOCK) {
272:                    block = dblock / BLOCKS_PER_IND_BLOCK(i->sb);
273:            }
274:
275:            if(!indblock[block]) {
276:                    if(mode == FOR_WRITING) {
277:                            if((newblock = minix_balloc(i->sb)) < 0) {
278:                                    brelse(buf);
279:                                    return -ENOSPC;
280:                            }
281:                            /* initialize the new block */
282:                            if(!(buf2 = bread(i->dev, newblock, blksize))) {
283:                                    minix_bfree(i->sb, newblock);
284:                                    brelse(buf);
285:                                    return -EIO;
286:                            }
287:                            memset_b(buf2->data, 0, blksize);
288:                            bwrite(buf2);
289:                            indblock[block] = newblock;
290:                            if(level == MINIX_IND_BLOCK) {
291:                                    bwrite(buf);
292:                                    return newblock;
293:                            }
294:                            buf->dirty = 1;
295:                            buf->valid = 1;
296:                    } else {
297:                            brelse(buf);
298:                            return 0;
299:                    }
300:            }
301:            if(level == MINIX_IND_BLOCK) {
302:                    newblock = indblock[block];
303:                    brelse(buf);
304:                    return newblock;
305:            }
306:
307:            iblock = block;
308:            if(!(buf2 = bread(i->dev, indblock[iblock], blksize))) {
309:                    printk("%s(): returning -EIO\n", __FUNCTION__);
310:                    brelse(buf);
311:                    return -EIO;
312:            }
313:            dindblock = (__u16 *)buf2->data;
314:            block = dindblock[dblock - (iblock * BLOCKS_PER_IND_BLOCK(i->sb))];
315:            if(!block && mode == FOR_WRITING) {
316:                    if((newblock = minix_balloc(i->sb)) < 0) {
317:                            brelse(buf);
318:                            brelse(buf2);
319:                            return -ENOSPC;
320:                    }
321:                    /* initialize the new block */
322:                    if(!(buf3 = bread(i->dev, newblock, blksize))) {
323:                            minix_bfree(i->sb, newblock);
```

fs/minix/v1_inode.c


```
  324:                          brelse(buf);
  325:                          brelse(buf2);
  326:                          return -EIO;
  327:                  }
  328:                  memset_b(buf3->data, 0, blksize);
  329:                  bwrite(buf3);
  330:                  dindblock[dblock - (iblock * BLOCKS_PER_IND_BLOCK(i->sb))] = new
block;
  331:                  buf2->dirty = 1;
  332:                  buf2->valid = 1;
  333:                  block = newblock;
  334:          }
  335:          brelse(buf);
  336:          brelse(buf2);
  337:          return block;
  338: }
  339:
  340: int v1_minix_truncate(struct inode *i, __off_t length)
  341: {
  342:          int n;
  343:          __blk_t block, dblock;
  344:          __u16 *zone;
  345:          struct buffer *buf;
  346:
  347:          block = length / i->sb->s_blocksize;
  348:
  349:          if(!S_ISDIR(i->i_mode) && !S_ISREG(i->i_mode) && !S_ISLNK(i->i_mode)) {
  350:                  return -EINVAL;
  351:          }
  352:
  353:          if(block < MINIX_NDIR_BLOCKS) {
  354:                  for(n = block; n < MINIX_NDIR_BLOCKS; n++) {
  355:                          if(i->u.minix.u.i1_zone[n]) {
  356:                                  minix_bfree(i->sb, i->u.minix.u.i1_zone[n]);
  357:                                  i->u.minix.u.i1_zone[n] = 0;
  358:                          }
  359:                  }
  360:                  block = 0;
  361:          }
  362:
  363:          if(!block || block < (BLOCKS_PER_IND_BLOCK(i->sb) + MINIX_NDIR_BLOCKS))
{
  364:                  if(block) {
  365:                          block -= MINIX_NDIR_BLOCKS;
  366:                  }
  367:                  if(i->u.minix.u.i1_zone[MINIX_IND_BLOCK]) {
  368:                          free_zone(i, i->u.minix.u.i1_zone[MINIX_IND_BLOCK], bloc
k);
  369:                          if(!block) {
  370:                                  minix_bfree(i->sb, i->u.minix.u.i1_zone[MINIX_IN
D_BLOCK]);
  371:                                  i->u.minix.u.i1_zone[MINIX_IND_BLOCK] = 0;
  372:                          }
  373:                  }
  374:                  block = 0;
  375:          }
  376:
  377:          if(block) {
  378:                  block -= MINIX_NDIR_BLOCKS;
  379:                  block -= BLOCKS_PER_IND_BLOCK(i->sb);
  380:          }
  381:          if(i->u.minix.u.i1_zone[MINIX_DIND_BLOCK]) {
  382:                  if(!(buf = bread(i->dev, i->u.minix.u.i1_zone[MINIX_DIND_BLOCK],
 i->sb->s_blocksize))) {
  383:                          printk("%s(): error reading block %d.\n", __FUNCTION__,
i->u.minix.u.i1_zone[MINIX_DIND_BLOCK]);
  384:                  }
```

fs/minix/v1_inode.c


```
  385:                  zone = (__u16 *)buf->data;
  386:                  dblock = block % BLOCKS_PER_IND_BLOCK(i->sb);
  387:                  for(n = block / BLOCKS_PER_IND_BLOCK(i->sb); n < BLOCKS_PER_IND_
BLOCK(i->sb); n++) {
  388:                          if(zone[n]) {
  389:                                  free_zone(i, zone[n], dblock);
  390:                                  if(!dblock) {
  391:                                          minix_bfree(i->sb, zone[n]);
  392:                                  }
  393:                          }
  394:                          dblock = 0;
  395:                  }
  396:                  bwrite(buf);
  397:                  if(!block) {
  398:                          minix_bfree(i->sb, i->u.minix.u.i1_zone[MINIX_DIND_BLOCK
]);
  399:                          i->u.minix.u.i1_zone[MINIX_DIND_BLOCK] = 0;
  400:                  }
  401:          }
  402:
  403:          i->i_mtime = CURRENT_TIME;
  404:          i->i_ctime = CURRENT_TIME;
  405:          i->i_size = length;
  406:          i->dirty = 1;
  407:
  408:          return 0;
  409: }
```

fs/minix/v2_inode.c


```
    1: /*
    2:  * fiwix/fs/minix/v2_inode.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/filesystems.h>
   11: #include <fiwix/fs_minix.h>
   12: #include <fiwix/fs_pipe.h>
   13: #include <fiwix/statfs.h>
   14: #include <fiwix/sleep.h>
   15: #include <fiwix/stat.h>
   16: #include <fiwix/sched.h>
   17: #include <fiwix/buffer.h>
   18: #include <fiwix/process.h>
   19: #include <fiwix/errno.h>
   20: #include <fiwix/stdio.h>
   21: #include <fiwix/string.h>
   22:
   23: #define BLOCKS_PER_IND_BLOCK(sb)        (sb->s_blocksize / sizeof(__u32))
   24: #define MINIX2_INODES_PER_BLOCK(sb)     (sb->s_blocksize / sizeof(struct minix2_
inode))
   25:
   26: #define MINIX_NDIR_BLOCKS               7
   27: #define MINIX_IND_BLOCK                 MINIX_NDIR_BLOCKS
   28: #define MINIX_DIND_BLOCK                (MINIX_NDIR_BLOCKS + 1)
   29: #define MINIX_TIND_BLOCK                (MINIX_NDIR_BLOCKS + 2)
   30:
   31: static void free_zone(struct inode *i, int block, int offset)
   32: {
   33:         int n;
   34:         struct buffer *buf;
   35:         __u32 *zone;
   36:
   37:         if(!(buf = bread(i->dev, block, i->sb->s_blocksize))) {
   38:                 printk("WARNING: %s(): error reading block %d.\n", __FUNCTION__,
 block);
   39:                 return;
   40:         }
   41:         zone = (__u32 *)buf->data;
   42:         for(n = offset; n < BLOCKS_PER_IND_BLOCK(i->sb); n++) {
   43:                 if(zone[n]) {
   44:                         minix_bfree(i->sb, zone[n]);
   45:                         zone[n] = 0;
   46:                 }
   47:         }
   48:         bwrite(buf);
   49: }
   50:
   51: int v2_minix_read_inode(struct inode *i)
   52: {
   53:         __ino_t block;
   54:         short int offset;
   55:         struct minix2_inode *ii;
   56:         struct buffer *buf;
   57:         int errno;
   58:
   59:         block = 1 + SUPERBLOCK + i->sb->u.minix.sb.s_imap_blocks + i->sb->u.mini
x.sb.s_zmap_blocks + (i->inode - 1) / MINIX2_INODES_PER_BLOCK(i->sb);
   60:
   61:         if(!(buf = bread(i->dev, block, i->sb->s_blocksize))) {
   62:                 return -EIO;
   63:         }
   64:         offset = (i->inode - 1) % MINIX2_INODES_PER_BLOCK(i->sb);
```

fs/minix/v2_inode.c


```
   65:         ii = ((struct minix2_inode *)buf->data) + offset;
   66:
   67:         i->i_mode = ii->i_mode;
   68:         i->i_nlink = ii->i_nlink;
   69:         i->i_uid = ii->i_uid;
   70:         i->i_gid = ii->i_gid;
   71:         i->i_size = ii->i_size;
   72:         i->i_atime = ii->i_atime;
   73:         i->i_mtime = ii->i_mtime;
   74:         i->i_ctime = ii->i_ctime;
   75:         memcpy_b(i->u.minix.u.i2_zone, ii->i_zone, sizeof(ii->i_zone));
   76:         i->count = 1;
   77:
   78:         errno = 0;
   79:         switch(i->i_mode & S_IFMT) {
   80:                 case S_IFCHR:
   81:                         i->fsop = &def_chr_fsop;
   82:                         i->rdev = ii->i_zone[0];
   83:                         break;
   84:                 case S_IFBLK:
   85:                         i->fsop = &def_blk_fsop;
   86:                         i->rdev = ii->i_zone[0];
   87:                         break;
   88:                 case S_IFIFO:
   89:                         i->fsop = &pipefs_fsop;
   90:                         /* it's a union so we need to clear pipefs_i */
   91:                         memset_b(&i->u.pipefs, NULL, sizeof(struct pipefs_inode)
);
   92:                         break;
   93:                 case S_IFDIR:
   94:                         i->fsop = &minix_dir_fsop;
   95:                         break;
   96:                 case S_IFREG:
   97:                         i->fsop = &minix_file_fsop;
   98:                         break;
   99:                 case S_IFLNK:
  100:                         i->fsop = &minix_symlink_fsop;
  101:                         break;
  102:                 case S_IFSOCK:
  103:                         i->fsop = NULL;
  104:                         break;
  105:                 default:
  106:                         printk("WARNING: %s(): invalid inode (%d) mode %o.\n", _
_FUNCTION__, i->inode, i->i_mode);
  107:                         errno = -ENOENT;
  108:                         break;
  109:         }
  110:
  111:         brelse(buf);
  112:         return errno;
  113: }
  114:
  115: int v2_minix_write_inode(struct inode *i)
  116: {
  117:         __ino_t block;
  118:         short int offset;
  119:         struct minix2_inode *ii;
  120:         struct buffer *buf;
  121:
  122:         block = 1 + SUPERBLOCK + i->sb->u.minix.sb.s_imap_blocks + i->sb->u.mini
x.sb.s_zmap_blocks + (i->inode - 1) / MINIX2_INODES_PER_BLOCK(i->sb);
  123:
  124:         if(!(buf = bread(i->dev, block, i->sb->s_blocksize))) {
  125:                 return -EIO;
  126:         }
  127:         offset = (i->inode - 1) % MINIX2_INODES_PER_BLOCK(i->sb);
  128:         ii = ((struct minix2_inode *)buf->data) + offset;
```

fs/minix/v2_inode.c


```
  129:
  130:          ii->i_mode = i->i_mode;
  131:          ii->i_nlink = i->i_nlink;
  132:          ii->i_uid = i->i_uid;
  133:          ii->i_gid = i->i_gid;
  134:          ii->i_size = i->i_size;
  135:          ii->i_atime = i->i_atime;
  136:          ii->i_mtime = i->i_mtime;
  137:          ii->i_ctime = i->i_ctime;
  138:          if(S_ISCHR(i->i_mode) || S_ISBLK(i->i_mode)) {
  139:                  ii->i_zone[0] = i->rdev;
  140:          } else {
  141:                  memcpy_b(ii->i_zone, i->u.minix.u.i2_zone, sizeof(i->u.minix.u.i
2_zone));
  142:          }
  143:          i->dirty = 0;
  144:          bwrite(buf);
  145:          return 0;
  146: }
  147:
  148: int v2_minix_ialloc(struct inode *i, int mode)
  149: {
  150:          __blk_t offset;
  151:          int inode, errno;
  152:          struct superblock *sb;
  153:
  154:          sb = i->sb;
  155:          superblock_lock(sb);
  156:
  157:          offset = 1 + SUPERBLOCK;
  158:
  159:          if(!(inode = minix_find_first_zero(sb, offset, sb->u.minix.sb.s_ninodes,
 offset + sb->u.minix.sb.s_imap_blocks))) {
  160:                  superblock_unlock(sb);
  161:                  return -ENOSPC;
  162:          }
  163:
  164:          errno = minix_change_bit(SET_BIT, sb, offset, inode);
  165:
  166:          if(errno) {
  167:                  if(errno < 0) {
  168:                          printk("WARNING: %s(): unable to set inode %d.\n", __FUN
CTION__, inode);
  169:                          superblock_unlock(sb);
  170:                          return errno;
  171:                  } else {
  172:                          printk("WARNING: %s(): inode %d is already marked as use
d!\n", __FUNCTION__, inode);
  173:                  }
  174:          }
  175:
  176:          i->inode = inode;
  177:          i->i_atime = CURRENT_TIME;
  178:          i->i_mtime = CURRENT_TIME;
  179:          i->i_ctime = CURRENT_TIME;
  180:          superblock_unlock(sb);
  181:          return 0;
  182: }
  183:
  184: void v2_minix_ifree(struct inode *i)
  185: {
  186:          int errno;
  187:          struct superblock *sb;
  188:
  189:          minix_truncate(i, 0);
  190:
  191:          sb = i->sb;
```

fs/minix/v2_inode.c


```
  192:         superblock_lock(sb);
  193:
  194:         errno = minix_change_bit(CLEAR_BIT, i->sb, 1 + SUPERBLOCK, i->inode);
  195:
  196:         if(errno) {
  197:                 if(errno < 0) {
  198:                         printk("WARNING: %s(): unable to clear inode %d.\n", __F
UNCTION__, i->inode);
  199:                 } else {
  200:                         printk("WARNING: %s(): inode %d is already marked as fre
e!\n", __FUNCTION__, i->inode);
  201:                 }
  202:         }
  203:
  204:         i->i_size = 0;
  205:         i->i_mtime = CURRENT_TIME;
  206:         i->i_ctime = CURRENT_TIME;
  207:         i->dirty = 1;
  208:         superblock_unlock(sb);
  209: }
  210:
  211: int v2_minix_bmap(struct inode *i, __off_t offset, int mode)
  212: {
  213:         unsigned char level;
  214:         __u32 *indblock, *dindblock, *tindblock;
  215:         __blk_t block, iblock, dblock, tblock, newblock;
  216:         int blksize;
  217:         struct buffer *buf, *buf2, *buf3, *buf4;
  218:
  219:         blksize = i->sb->s_blocksize;
  220:         block = offset / blksize;
  221:         level = 0;
  222:         buf3 = NULL;    /* makes GCC happy */
  223:
  224:         if(block < MINIX_NDIR_BLOCKS) {
  225:                 level = MINIX_NDIR_BLOCKS - 1;
  226:         } else {
  227:                 if(block < (BLOCKS_PER_IND_BLOCK(i->sb) + MINIX_NDIR_BLOCKS)) {
  228:                         level = MINIX_IND_BLOCK;
  229:                 } else if(block < ((BLOCKS_PER_IND_BLOCK(i->sb) * BLOCKS_PER_IND
_BLOCK(i->sb)) + BLOCKS_PER_IND_BLOCK(i->sb) + MINIX_NDIR_BLOCKS)) {
  230:                         level = MINIX_DIND_BLOCK;
  231:                 } else {
  232:                         level = MINIX_TIND_BLOCK;
  233:                 }
  234:                 block -= MINIX_NDIR_BLOCKS;
  235:         }
  236:
  237:         if(level < MINIX_NDIR_BLOCKS) {
  238:                 if(!i->u.minix.u.i2_zone[block] && mode == FOR_WRITING) {
  239:                         if((newblock = minix_balloc(i->sb)) < 0) {
  240:                                 return -ENOSPC;
  241:                         }
  242:                         /* initialize the new block */
  243:                         if(!(buf = bread(i->dev, newblock, blksize))) {
  244:                                 minix_bfree(i->sb, newblock);
  245:                                 return -EIO;
  246:                         }
  247:                         memset_b(buf->data, 0, blksize);
  248:                         bwrite(buf);
  249:                         i->u.minix.u.i2_zone[block] = newblock;
  250:                 }
  251:                 return i->u.minix.u.i2_zone[block];
  252:         }
  253:
  254:         if(!i->u.minix.u.i2_zone[level]) {
  255:                 if(mode == FOR_WRITING) {
```

fs/minix/v2_inode.c


```
  256:                          if((newblock = minix_balloc(i->sb)) < 0) {
  257:                                  return -ENOSPC;
  258:                          }
  259:                          /* initialize the new block */
  260:                          if(!(buf = bread(i->dev, newblock, blksize))) {
  261:                                  minix_bfree(i->sb, newblock);
  262:                                  return -EIO;
  263:                          }
  264:                          memset_b(buf->data, 0, blksize);
  265:                          bwrite(buf);
  266:                          i->u.minix.u.i2_zone[level] = newblock;
  267:                  } else {
  268:                          return 0;
  269:                  }
  270:          }
  271:          if(!(buf = bread(i->dev, i->u.minix.u.i2_zone[level], blksize))) {
  272:                  return -EIO;
  273:          }
  274:          indblock = (__u32 *)buf->data;
  275:          dblock = block - BLOCKS_PER_IND_BLOCK(i->sb);
  276:          tblock = block - (BLOCKS_PER_IND_BLOCK(i->sb) * BLOCKS_PER_IND_BLOCK(i->
sb)) - BLOCKS_PER_IND_BLOCK(i->sb);
  277:
  278:          if(level == MINIX_DIND_BLOCK) {
  279:                  block = dblock / BLOCKS_PER_IND_BLOCK(i->sb);
  280:          }
  281:          if(level == MINIX_TIND_BLOCK) {
  282:                  block = tblock / (BLOCKS_PER_IND_BLOCK(i->sb) * BLOCKS_PER_IND_B
LOCK(i->sb));
  283:          }
  284:
  285:          if(!indblock[block]) {
  286:                  if(mode == FOR_WRITING) {
  287:                          if((newblock = minix_balloc(i->sb)) < 0) {
  288:                                  brelse(buf);
  289:                                  return -ENOSPC;
  290:                          }
  291:                          /* initialize the new block */
  292:                          if(!(buf2 = bread(i->dev, newblock, blksize))) {
  293:                                  minix_bfree(i->sb, newblock);
  294:                                  brelse(buf);
  295:                                  return -EIO;
  296:                          }
  297:                          memset_b(buf2->data, 0, blksize);
  298:                          bwrite(buf2);
  299:                          indblock[block] = newblock;
  300:                          if(level == MINIX_IND_BLOCK) {
  301:                                  bwrite(buf);
  302:                                  return newblock;
  303:                          }
  304:                          buf->dirty = 1;
  305:                          buf->valid = 1;
  306:                  } else {
  307:                          brelse(buf);
  308:                          return 0;
  309:                  }
  310:          }
  311:          if(level == MINIX_IND_BLOCK) {
  312:                  newblock = indblock[block];
  313:                  brelse(buf);
  314:                  return newblock;
  315:          }
  316:
  317:          if(level == MINIX_TIND_BLOCK) {
  318:                  if(!(buf3 = bread(i->dev, indblock[block], blksize))) {
  319:                          printk("%s(): returning -EIO\n", __FUNCTION__);
  320:                          brelse(buf);
```

fs/minix/v2_inode.c


```
   321:                         return -EIO;
   322:                 }
   323:                 tindblock = (__u32 *)buf3->data;
   324:                 block = tindblock[tblock / BLOCKS_PER_IND_BLOCK(i->sb)];
   325:                 if(!block) {
   326:                         if(mode == FOR_WRITING) {
   327:                                 if((newblock = minix_balloc(i->sb)) < 0) {
   328:                                         brelse(buf);
   329:                                         brelse(buf3);
   330:                                         return -ENOSPC;
   331:                                 }
   332:                                 /* initialize the new block */
   333:                                 if(!(buf4 = bread(i->dev, newblock, blksize))) {
   334:                                         minix_bfree(i->sb, newblock);
   335:                                         brelse(buf);
   336:                                         brelse(buf3);
   337:                                         return -EIO;
   338:                                 }
   339:                                 memset_b(buf4->data, 0, blksize);
   340:                                 bwrite(buf4);
   341:                                 tindblock[tblock / BLOCKS_PER_IND_BLOCK(i->sb)]
= newblock;
   342:                                 buf3->dirty = 1;
   343:                                 buf3->valid = 1;
   344:                                 block = newblock;
   345:                         } else {
   346:                                 brelse(buf);
   347:                                 brelse(buf3);
   348:                                 return 0;
   349:                         }
   350:                 }
   351:                 dblock = tblock;
   352:                 iblock = tblock / BLOCKS_PER_IND_BLOCK(i->sb);
   353:                 if(!(buf2 = bread(i->dev, block, blksize))) {
   354:                         printk("%s(): returning -EIO\n", __FUNCTION__);
   355:                         brelse(buf);
   356:                         brelse(buf3);
   357:                         return -EIO;
   358:                 }
   359:         } else {
   360:                 iblock = block;
   361:                 if(!(buf2 = bread(i->dev, indblock[iblock], blksize))) {
   362:                         printk("%s(): returning -EIO\n", __FUNCTION__);
   363:                         brelse(buf);
   364:                         return -EIO;
   365:                 }
   366:         }
   367:
   368:         dindblock = (__u32 *)buf2->data;
   369:         block = dindblock[dblock - (iblock * BLOCKS_PER_IND_BLOCK(i->sb))];
   370:         if(!block && mode == FOR_WRITING) {
   371:                 if((newblock = minix_balloc(i->sb)) < 0) {
   372:                         brelse(buf);
   373:                         if(level == MINIX_TIND_BLOCK) {
   374:                                 brelse(buf3);
   375:                         }
   376:                         brelse(buf2);
   377:                         return -ENOSPC;
   378:                 }
   379:                 /* initialize the new block */
   380:                 if(!(buf4 = bread(i->dev, newblock, blksize))) {
   381:                         minix_bfree(i->sb, newblock);
   382:                         brelse(buf);
   383:                         if(level == MINIX_TIND_BLOCK) {
   384:                                 brelse(buf3);
   385:                         }
   386:                         brelse(buf2);
```

fs/minix/v2_inode.c


```
  387:                          return -EIO;
  388:                  }
  389:                  memset_b(buf4->data, 0, blksize);
  390:                  bwrite(buf4);
  391:                  dindblock[dblock - (iblock * BLOCKS_PER_IND_BLOCK(i->sb))] = new
block;
  392:                  buf2->dirty = 1;
  393:                  buf2->valid = 1;
  394:                  block = newblock;
  395:          }
  396:          brelse(buf);
  397:          if(level == MINIX_TIND_BLOCK) {
  398:                  brelse(buf3);
  399:          }
  400:          brelse(buf2);
  401:          return block;
  402: }
  403:
  404: int v2_minix_truncate(struct inode *i, __off_t length)
  405: {
  406:          int n;
  407:          __blk_t block, dblock;
  408:          __u32 *zone;
  409:          struct buffer *buf;
  410:
  411:          block = length / i->sb->s_blocksize;
  412:
  413:          if(!S_ISDIR(i->i_mode) && !S_ISREG(i->i_mode) && !S_ISLNK(i->i_mode)) {
  414:                  return -EINVAL;
  415:          }
  416:
  417:          if(block < MINIX_NDIR_BLOCKS) {
  418:                  for(n = block; n < MINIX_NDIR_BLOCKS; n++) {
  419:                          if(i->u.minix.u.i2_zone[n]) {
  420:                                  minix_bfree(i->sb, i->u.minix.u.i2_zone[n]);
  421:                                  i->u.minix.u.i2_zone[n] = 0;
  422:                          }
  423:                  }
  424:                  block = 0;
  425:          }
  426:
  427:          if(!block || block < (BLOCKS_PER_IND_BLOCK(i->sb) + MINIX_NDIR_BLOCKS))
{
  428:                  if(block) {
  429:                          block -= MINIX_NDIR_BLOCKS;
  430:                  }
  431:                  if(i->u.minix.u.i2_zone[MINIX_IND_BLOCK]) {
  432:                          free_zone(i, i->u.minix.u.i2_zone[MINIX_IND_BLOCK], bloc
k);
  433:                          if(!block) {
  434:                                  minix_bfree(i->sb, i->u.minix.u.i2_zone[MINIX_IN
D_BLOCK]);
  435:                                  i->u.minix.u.i2_zone[MINIX_IND_BLOCK] = 0;
  436:                          }
  437:                  }
  438:                  block = 0;
  439:          }
  440:
  441:          if(block) {
  442:                  block -= MINIX_NDIR_BLOCKS;
  443:                  block -= BLOCKS_PER_IND_BLOCK(i->sb);
  444:          }
  445:          if(i->u.minix.u.i2_zone[MINIX_DIND_BLOCK]) {
  446:                  if(!(buf = bread(i->dev, i->u.minix.u.i2_zone[MINIX_DIND_BLOCK],
 i->sb->s_blocksize))) {
  447:                          printk("%s(): error reading block %d.\n", __FUNCTION__,
i->u.minix.u.i2_zone[MINIX_DIND_BLOCK]);
```

fs/minix/v2_inode.c


```
  448:                  }
  449:                  zone = (__u32 *)buf->data;
  450:                  dblock = block % BLOCKS_PER_IND_BLOCK(i->sb);
  451:                  for(n = block / BLOCKS_PER_IND_BLOCK(i->sb); n < BLOCKS_PER_IND_
BLOCK(i->sb); n++) {
  452:                          if(zone[n]) {
  453:                                  free_zone(i, zone[n], dblock);
  454:                                  if(!dblock) {
  455:                                          minix_bfree(i->sb, zone[n]);
  456:                                  }
  457:                          }
  458:                          dblock = 0;
  459:                  }
  460:                  bwrite(buf);
  461:                  if(!block) {
  462:                          minix_bfree(i->sb, i->u.minix.u.i2_zone[MINIX_DIND_BLOCK
]);
  463:                          i->u.minix.u.i2_zone[MINIX_DIND_BLOCK] = 0;
  464:                  }
  465:          }
  466:
  467:          i->i_mtime = CURRENT_TIME;
  468:          i->i_ctime = CURRENT_TIME;
  469:          i->i_size = length;
  470:          i->dirty = 1;
  471:
  472:          return 0;
  473: }
```

fs/pipefs/fifo.c


```
    1: /*
    2:  * fiwix/fs/pipefs/fifo.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/errno.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/filesystems.h>
   12: #include <fiwix/fs_pipe.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/sleep.h>
   16: #include <fiwix/fcntl.h>
   17: #include <fiwix/sched.h>
   18: #include <fiwix/stdio.h>
   19:
   20: int fifo_open(struct inode *i, struct fd *fd_table)
   21: {
   22:         /* first open */
   23:         if(i->count == 1) {
   24:                 if(!(i->u.pipefs.i_data = (void *)kmalloc())) {
   25:                         return -ENOMEM;
   26:                 }
   27:                 i->u.pipefs.i_readoff = 0;
   28:                 i->u.pipefs.i_writeoff = 0;
   29:         }
   30:
   31:         if((fd_table->flags & O_ACCMODE) == O_RDONLY) {
   32:                 i->u.pipefs.i_readers++;
   33:                 wakeup(&pipefs_write);
   34:                 if(!(fd_table->flags & O_NONBLOCK)) {
   35:                         while(!i->u.pipefs.i_writers) {
   36:                                 if(sleep(&pipefs_read, PROC_INTERRUPTIBLE)) {
   37:                                         if(!--i->u.pipefs.i_readers) {
   38:                                                 wakeup(&pipefs_write);
   39:                                         }
   40:                                         return -EINTR;
   41:                                 }
   42:                         }
   43:                 }
   44:         }
   45:
   46:         if((fd_table->flags & O_ACCMODE) == O_WRONLY) {
   47:                 if((fd_table->flags & O_NONBLOCK) && !i->u.pipefs.i_readers) {
   48:                         return -ENXIO;
   49:                 }
   50:
   51:                 i->u.pipefs.i_writers++;
   52:                 wakeup(&pipefs_read);
   53:                 if(!(fd_table->flags & O_NONBLOCK)) {
   54:                         while(!i->u.pipefs.i_readers) {
   55:                                 if(sleep(&pipefs_write, PROC_INTERRUPTIBLE)) {
   56:                                         if(!--i->u.pipefs.i_writers) {
   57:                                                 wakeup(&pipefs_read);
   58:                                         }
   59:                                         return -EINTR;
   60:                                 }
   61:                         }
   62:                 }
   63:         }
   64:
   65:         if((fd_table->flags & O_ACCMODE) == O_RDWR) {
   66:                 i->u.pipefs.i_readers++;
   67:                 i->u.pipefs.i_writers++;
```

fs/pipefs/fifo.c


```
68:                     wakeup(&pipefs_write);
69:                     wakeup(&pipefs_read);
70:             }
71:
72:             return 0;
73: }
```

fs/pipefs/Makefile


```
 1: # fiwix/fs/pipefs/Makefile
 2: #
 3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
 4: # Distributed under the terms of the Fiwix License.
 5: #
 6:
 7: .S.o:
 8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
 9: .c.o:
10:         $(CC) $(CFLAGS) -c -o $@ $<
11:
12: OBJS = super.o fifo.o pipe.o
13:
14: pipefs: $(OBJS)
15:         $(LD) $(LDFLAGS) -r $(OBJS) -o pipefs.o
16:
17: clean:
18:         rm -f *.o
19:
```

fs/pipefs/pipe.c


```
    1: /*
    2:  * fiwix/fs/pipefs/pipe.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/errno.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/filesystems.h>
   12: #include <fiwix/fs_pipe.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/fcntl.h>
   15: #include <fiwix/ioctl.h>
   16: #include <fiwix/sleep.h>
   17: #include <fiwix/sched.h>
   18: #include <fiwix/stdio.h>
   19: #include <fiwix/string.h>
   20:
   21: static struct resource pipe_resource = { NULL, NULL };
   22:
   23: int pipefs_close(struct inode *i, struct fd *fd_table)
   24: {
   25:         if((fd_table->flags & O_ACCMODE) == O_RDONLY) {
   26:                 if(!--i->u.pipefs.i_readers) {
   27:                         wakeup(&pipefs_write);
   28:                 }
   29:         }
   30:         if((fd_table->flags & O_ACCMODE) == O_WRONLY) {
   31:                 if(!--i->u.pipefs.i_writers) {
   32:                         wakeup(&pipefs_read);
   33:                 }
   34:         }
   35:         if((fd_table->flags & O_ACCMODE) == O_RDWR) {
   36:                 if(!--i->u.pipefs.i_readers) {
   37:                         wakeup(&pipefs_write);
   38:                 }
   39:                 if(!--i->u.pipefs.i_writers) {
   40:                         wakeup(&pipefs_read);
   41:                 }
   42:         }
   43:         return 0;
   44: }
   45:
   46: int pipefs_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t cou
nt)
   47: {
   48:         __off_t bytes_read;
   49:         __size_t n, limit;
   50:         char *data;
   51:
   52:         bytes_read = 0;
   53:         data = i->u.pipefs.i_data;
   54:
   55:         while(count) {
   56:                 if(i->u.pipefs.i_writeoff) {
   57:                         if(i->u.pipefs.i_readoff >= i->u.pipefs.i_writeoff) {
   58:                                 limit = PIPE_BUF - i->u.pipefs.i_readoff;
   59:                         } else {
   60:                                 limit = i->u.pipefs.i_writeoff - i->u.pipefs.i_r
eadoff;
   61:                         }
   62:                 } else {
   63:                         limit = PIPE_BUF - i->u.pipefs.i_readoff;
   64:                 }
   65:                 n = MIN(limit, count);
```

fs/pipefs/pipe.c


```
   66:                 if(i->i_size && n) {
   67:                         lock_resource(&pipe_resource);
   68:                         memcpy_b(buffer + bytes_read, data + i->u.pipefs.i_reado
ff, n);
   69:                         bytes_read += n;
   70:                         i->u.pipefs.i_readoff += n;
   71:                         i->i_size -= n;
   72:                         if(i->u.pipefs.i_writeoff >= PIPE_BUF) {
   73:                                 i->u.pipefs.i_writeoff = 0;
   74:                         }
   75:                         unlock_resource(&pipe_resource);
   76:                         wakeup(&pipefs_write);
   77:                         break;
   78:                 } else {
   79:                         if(i->u.pipefs.i_writers) {
   80:                                 if(fd_table->flags & O_NONBLOCK) {
   81:                                         return -EAGAIN;
   82:                                 }
   83:                                 if(sleep(&pipefs_read, PROC_INTERRUPTIBLE)) {
   84:                                         return -EINTR;
   85:                                 }
   86:                         } else {
   87:                                 if(i->i_size) {
   88:                                         if(i->u.pipefs.i_readoff >= PIPE_BUF) {
   89:                                                 i->u.pipefs.i_readoff = 0;
   90:                                                 continue;
   91:                                         }
   92:                                 }
   93:                                 break;
   94:                         }
   95:                 }
   96:         }
   97:         if(!i->i_size) {
   98:                 i->u.pipefs.i_readoff = 0;
   99:                 i->u.pipefs.i_writeoff = 0;
  100:         }
  101:         return bytes_read;
  102: }
  103:
  104: int pipefs_write(struct inode *i, struct fd *fd_table, const char *buffer, __siz
e_t count)
  105: {
  106:         __off_t bytes_written;
  107:         __size_t n;
  108:         char *data;
  109:         int limit;
  110:
  111:         bytes_written = 0;
  112:         data = i->u.pipefs.i_data;
  113:
  114:         while(bytes_written < count) {
  115:                 /* if the read end closes then send signal and return */
  116:                 if(!i->u.pipefs.i_readers) {
  117:                         send_sig(current, SIGPIPE);
  118:                         return -EPIPE;
  119:                 }
  120:
  121:                 if(i->u.pipefs.i_readoff) {
  122:                         if(i->u.pipefs.i_writeoff <= i->u.pipefs.i_readoff) {
  123:                                 limit = i->u.pipefs.i_readoff;
  124:                         } else {
  125:                                 limit = PIPE_BUF;
  126:                         }
  127:                 } else {
  128:                         limit = PIPE_BUF;
  129:                 }
  130:
```

fs/pipefs/pipe.c


```
  131:                 n = MIN((count - bytes_written), (limit - i->u.pipefs.i_writeoff
));
  132:
  133:                 /*
  134:                  * POSIX requires that any write operation involving fewer than
  135:                  * PIPE_BUF bytes must be automatically executed and finished
  136:                  * without being interleaved with write operations of other
  137:                  * processes to the same pipe.
  138:                  */
  139:                 if(n && n <= PIPE_BUF) {
  140:                         lock_resource(&pipe_resource);
  141:                         memcpy_b(data + i->u.pipefs.i_writeoff, buffer + bytes_w
ritten, n);
  142:                         bytes_written += n;
  143:                         i->u.pipefs.i_writeoff += n;
  144:                         i->i_size += n;
  145:                         if(i->u.pipefs.i_readoff >= PIPE_BUF) {
  146:                                 i->u.pipefs.i_readoff = 0;
  147:                         }
  148:                         unlock_resource(&pipe_resource);
  149:                         wakeup(&pipefs_read);
  150:                         continue;
  151:                 }
  152:
  153:                 wakeup(&pipefs_read);
  154:                 if(!(fd_table->flags & O_NONBLOCK)) {
  155:                         if(sleep(&pipefs_write, PROC_INTERRUPTIBLE)) {
  156:                                 return -EINTR;
  157:                         }
  158:                 } else {
  159:                         return -EAGAIN;
  160:                 }
  161:         }
  162:         return bytes_written;
  163: }
  164:
  165: int pipefs_ioctl(struct inode *i, int cmd, unsigned long int arg)
  166: {
  167:         int errno;
  168:
  169:         switch(cmd) {
  170:                 case FIONREAD:
  171:                         if((errno = check_user_area(VERIFY_WRITE, (void *)arg, s
izeof(unsigned int)))) {
  172:                                 return errno;
  173:                         }
  174:                         memcpy_b((void *)arg, &i->i_size, sizeof(unsigned int));
  175:                         break;
  176:                 default:
  177:                         return -EINVAL;
  178:         }
  179:         return 0;
  180: }
  181:
  182: int pipefs_lseek(struct inode *i, __off_t offset)
  183: {
  184:         return -ESPIPE;
  185: }
  186:
  187: int pipefs_select(struct inode *i, int flag)
  188: {
  189:         switch(flag) {
  190:                 case SEL_R:
  191:                         if(i->i_size || !i->u.pipefs.i_writers) {
  192:                                 return 1;
  193:                         }
  194:                         break;
```

fs/pipefs/pipe.c


```
195:                    case SEL_W:
196:                            if(i->i_size < PIPE_BUF || !i->u.pipefs.i_readers) {
197:                                    return 1;
198:                            }
199:                            break;
200:            }
201:            return 0;
202: }
```

fs/pipefs/super.c


```
    1: /*
    2:  * fiwix/fs/pipefs/super.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/errno.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/filesystems.h>
   12: #include <fiwix/fs_pipe.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/sched.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: static unsigned int i_counter;
   20:
   21: struct fs_operations pipefs_fsop = {
   22:         FSOP_KERN_MOUNT,
   23:         PIPE_DEV,
   24:
   25:         fifo_open,
   26:         pipefs_close,
   27:         pipefs_read,
   28:         pipefs_write,
   29:         pipefs_ioctl,
   30:         pipefs_lseek,
   31:         NULL,                   /* readdir */
   32:         NULL,                   /* mmap */
   33:         pipefs_select,
   34:
   35:         NULL,                   /* readlink */
   36:         NULL,                   /* followlink */
   37:         NULL,                   /* bmap */
   38:         NULL,                   /* lookup */
   39:         NULL,                   /* rmdir */
   40:         NULL,                   /* link */
   41:         NULL,                   /* unlink */
   42:         NULL,                   /* symlink */
   43:         NULL,                   /* mkdir */
   44:         NULL,                   /* mknod */
   45:         NULL,                   /* truncate */
   46:         NULL,                   /* create */
   47:         NULL,                   /* rename */
   48:
   49:         NULL,                   /* read_block */
   50:         NULL,                   /* write_block */
   51:
   52:         NULL,                   /* read_inode */
   53:         NULL,                   /* write_inode */
   54:         pipefs_ialloc,
   55:         pipefs_ifree,
   56:         NULL,                   /* statfs */
   57:         pipefs_read_superblock,
   58:         NULL,                   /* remount_fs */
   59:         NULL,                   /* write_superblock */
   60:         NULL                    /* release_superblock */
   61: };
   62:
   63: int pipefs_read_superblock(__dev_t dev, struct superblock *sb)
   64: {
   65:         superblock_lock(sb);
   66:         sb->dev = dev;
   67:         sb->fsop = &pipefs_fsop;
```

fs/pipefs/super.c


```
 68:         sb->s_blocksize = BLKSIZE_1K;
 69:         i_counter = 0;
 70:         superblock_unlock(sb);
 71:         return 0;
 72: }
 73:
 74: int pipefs_ialloc(struct inode *i, int mode)
 75: {
 76:         struct superblock *sb = i->sb;
 77:
 78:         superblock_lock(sb);
 79:         i_counter++;
 80:         superblock_unlock(sb);
 81:
 82:         i->i_mode = S_IFIFO;
 83:         i->dev = i->rdev = sb->dev;
 84:         i->fsop = &pipefs_fsop;
 85:         i->inode = i_counter;
 86:         i->count = 2;
 87:         if(!(i->u.pipefs.i_data = (void *)kmalloc())) {
 88:                 return -ENOMEM;
 89:         }
 90:         i->u.pipefs.i_readoff = 0;
 91:         i->u.pipefs.i_writeoff = 0;
 92:         i->u.pipefs.i_readers = 1;
 93:         i->u.pipefs.i_writers = 1;
 94:         return 0;
 95: }
 96:
 97: void pipefs_ifree(struct inode *i)
 98: {
 99:         if(!i->u.pipefs.i_readers && !i->u.pipefs.i_writers) {
100:                 /*
101:                  * We need to ask before to kfree() because this function is
102:                  * also called to free removed (with sys_unlink) fifo files.
103:                  */
104:                 if(i->u.pipefs.i_data) {
105:                         kfree((unsigned int)i->u.pipefs.i_data);
106:                 }
107:         }
108: }
109:
110: int pipefs_init(void)
111: {
112:         return register_filesystem("pipefs", &pipefs_fsop);
113: }
```

fs/procfs/data.c


```
    1: /*
    2:  * fiwix/fs/procfs/data.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/system.h>
   10: #include <fiwix/types.h>
   11: #include <fiwix/process.h>
   12: #include <fiwix/cmos.h>
   13: #include <fiwix/dma.h>
   14: #include <fiwix/ide.h>
   15: #include <fiwix/fs.h>
   16: #include <fiwix/filesystems.h>
   17: #include <fiwix/devices.h>
   18: #include <fiwix/locks.h>
   19: #include <fiwix/mm.h>
   20: #include <fiwix/mman.h>
   21: #include <fiwix/fs_proc.h>
   22: #include <fiwix/cpu.h>
   23: #include <fiwix/pic.h>
   24: #include <fiwix/sched.h>
   25: #include <fiwix/timer.h>
   26: #include <fiwix/utsname.h>
   27: #include <fiwix/version.h>
   28: #include <fiwix/errno.h>
   29: #include <fiwix/stdio.h>
   30: #include <fiwix/string.h>
   31:
   32: #define FSHIFT16        16
   33: #define FIXED16_1       (1 << FSHIFT16)
   34: #define LOAD_INT(x)     ((x) >> FSHIFT16)
   35: #define LOAD_FRAC(x)    LOAD_INT(((x) & (FIXED16_1 - 1)) * 100)
   36:
   37: static const char *pstate[] = {
   38:         "? (unused!)",
   39:         "R (running)",
   40:         "S (sleeping)",
   41:         "Z (zombie)",
   42:         "T (stopped)",
   43:         "D (idle)",
   44: };
   45:
   46: /*
   47:  * procfs root directory related functions
   48:  * --------------------------------------
   49:  */
   50: int data_proc_self(char *buffer, __pid_t pid)
   51: {
   52:         return sprintk(buffer, "%s", current->pidstr);
   53: }
   54:
   55: int data_proc_cmdline(char *buffer, __pid_t pid)
   56: {
   57:         return sprintk(buffer, "%s\n", cmdline);
   58: }
   59:
   60: int data_proc_cpuinfo(char *buffer, __pid_t pid)
   61: {
   62:         int size;
   63:
   64:         size = sprintk(buffer, "processor       : 0\n");
   65:         size += sprintk(buffer + size, "cpu family      : %d86\n", cpu_table.fam
ily <= 6 ? cpu_table.family : 6);
   66:         if(cpu_table.model >= 0) {
```

fs/procfs/data.c


```
   67:                 size += sprintk(buffer + size, "model           : %d\n", cpu_tab
le.model);
   68:         } else {
   69:                 size += sprintk(buffer + size, "model           : unknown\n");
   70:         }
   71:
   72:         if(cpu_table.vendor_id) {
   73:                 size += sprintk(buffer + size, "vendor_id       : %s\n", cpu_tab
le.vendor_id);
   74:         }
   75:         if(cpu_table.model_name) {
   76:                 size += sprintk(buffer + size, "model name      : %s\n", cpu_tab
le.model_name);
   77:         }
   78:         if(cpu_table.stepping >= 0) {
   79:                 size += sprintk(buffer + size, "stepping        : %d\n", cpu_tab
le.stepping);
   80:         } else {
   81:                 size += sprintk(buffer + size, "stepping        : unknown\n");
   82:         }
   83:
   84:         size += sprintk(buffer + size, "cpu MHz         : ");
   85:         if(cpu_table.hz) {
   86:                 size += sprintk(buffer + size, "%d.%d\n", (cpu_table.hz / 100000
0), ((cpu_table.hz % 1000000) / 100000));
   87:         } else {
   88:                 size += sprintk(buffer + size, "unknown\n");
   89:         }
   90:         if(cpu_table.cache) {
   91:                 size += sprintk(buffer + size, "cache size      : %s\n", cpu_tab
le.cache);
   92:         }
   93:         size += sprintk(buffer + size, "cpuid           : %s\n", cpu_table.has_c
puid ? "yes" : "no");
   94:         size += sprintk(buffer + size, "fpu             : %s\n", cpu_table.has_f
pu ? "yes" : "no");
   95:         size += get_cpu_flags(buffer, size);
   96:         return size;
   97: }
   98:
   99: int data_proc_devices(char *buffer, __pid_t pid)
  100: {
  101:         int n, size;
  102:         struct device *d;
  103:
  104:         d = chr_device_table;
  105:         size = sprintk(buffer, "Character devices:\n");
  106:         for(n = 0; n < NR_CHRDEV; n++, d++) {
  107:                 if(d->major) {
  108:                         size += sprintk(buffer + size, "%3d %s\n", d->major, d->
name);
  109:                 }
  110:         }
  111:
  112:         size += sprintk(buffer + size, "\nBlock devices:\n");
  113:         d = blk_device_table;
  114:         for(n = 0; n < NR_BLKDEV; n++, d++) {
  115:                 if(d->major) {
  116:                         size += sprintk(buffer + size, "%3d %s\n", d->major, d->
name);
  117:                 }
  118:         }
  119:         return size;
  120: }
  121:
  122: int data_proc_dma(char *buffer, __pid_t pid)
  123: {
```

fs/procfs/data.c


```
  124:         int n, size;
  125:
  126:         size = 0;
  127:         for(n = 0; n < DMA_CHANNELS; n++) {
  128:                 if(dma_resources[n]) {
  129:                         size += sprintk(buffer + size, "%2d: %s\n", n, dma_resou
rces[n]);
  130:                 }
  131:         }
  132:         return size;
  133: }
  134:
  135: int data_proc_filesystems(char *buffer, __pid_t pid)
  136: {
  137:         int n, size;
  138:         int nodev;
  139:
  140:         size = 0;
  141:         for(n = 0; n < NR_FILESYSTEMS; n++) {
  142:                 if(filesystems_table[n].name) {
  143:                         nodev = 0;
  144:                         if(filesystems_table[n].fsop->flags != FSOP_REQUIRES_DEV
) {
  145:                                 nodev = 1;
  146:                         }
  147:                         size += sprintk(buffer + size, "%s %s\n", nodev ? "nodev
" : "     ", filesystems_table[n].name);
  148:                 }
  149:         }
  150:         return size;
  151: }
  152:
  153: int data_proc_interrupts(char *buffer, __pid_t pid)
  154: {
  155:         int n, size;
  156:
  157:         size = 0;
  158:         for(n = 0; n < NR_IRQS; n++) {
  159:                 if(irq_table[n].registered) {
  160:                         size += sprintk(buffer + size, "%3d: %9u %s\n", n, irq_t
able[n].ticks, irq_table[n].name);
  161:                 }
  162:         }
  163:         size += sprintk(buffer + size, "SPU: %9u %s\n", kstat.sirqs, "Spurious i
nterrupts");
  164:         return size;
  165: }
  166:
  167: int data_proc_loadavg(char *buffer, __pid_t pid)
  168: {
  169:         int a, b, c;
  170:         int size;
  171:         struct proc *p;
  172:         int nrun = 0;
  173:         int nprocs = 0;
  174:
  175:         a = avenrun[0] << (SI_LOAD_SHIFT - FSHIFT);
  176:         b = avenrun[1] << (SI_LOAD_SHIFT - FSHIFT);
  177:         c = avenrun[2] << (SI_LOAD_SHIFT - FSHIFT);
  178:
  179:         FOR_EACH_PROCESS(p) {
  180:                 if(p->state) {
  181:                         nprocs++;
  182:                         if(p->state == PROC_RUNNING) {
  183:                                 nrun++;
  184:                         }
  185:                 }
```

fs/procfs/data.c


```
  186:          }
  187:
  188:          size = sprintk(buffer, "%d.%02d %d.%02d %d.%02d %d/%d %d\n", LOAD_INT(a)
, LOAD_FRAC(a), LOAD_INT(b), LOAD_FRAC(b), LOAD_INT(c), LOAD_FRAC(c), nrun, nprocs, las
tpid);
  189:          return size;
  190: }
  191:
  192: int data_proc_locks(char *buffer, __pid_t pid)
  193: {
  194:          int n, size;
  195:          struct flock_file *ff;
  196:
  197:          size = 0;
  198:
  199:          for(n = 0; n < NR_FLOCKS; n++) {
  200:                  ff = &flock_file_table[n];
  201:                  if(ff->inode) {
  202:                          size += sprintk(buffer + size, "%d: FLOCK  ADVISORY  %s
", n + 1, ff->type & LOCK_SH ? "READ " : "WRITE");
  203:                          size += sprintk(buffer + size, "%d %x:%d:%d 0 EOF\n", ff
->proc->pid, MAJOR(ff->inode->dev), MINOR(ff->inode->dev), ff->inode->inode);
  204:                  }
  205:          }
  206:
  207:          return size;
  208: }
  209:
  210: int data_proc_meminfo(char *buffer, __pid_t pid)
  211: {
  212:          struct page *pg;
  213:          int n, size;
  214:
  215:          kstat.shared = 0;
  216:          for(n = 0; n < kstat.physical_pages; n ++) {
  217:                  pg = &page_table[n];
  218:                  if(pg->flags & PAGE_RESERVED) {
  219:                          continue;
  220:                  }
  221:                  if(!pg->count) {
  222:                          continue;
  223:                  }
  224:                  kstat.shared += pg->count - 1;
  225:          }
  226:
  227:          size = 0;
  228:          size += sprintk(buffer + size, "        total:    used:    free:  shared
: buffers:  cached:\n");
  229:          size += sprintk(buffer + size, "Mem:  %8u %8u %8u %8u %8u %8u\n", kstat.
total_mem_pages << PAGE_SHIFT, (kstat.total_mem_pages << PAGE_SHIFT) - (kstat.free_page
s << PAGE_SHIFT), kstat.free_pages << PAGE_SHIFT, kstat.shared * 1024, kstat.buffers *
1024, kstat.cached * 1024);
  230:          size += sprintk(buffer + size, "Swap: %8u %8u %8u\n", 0, 0, 0);
  231:          size += sprintk(buffer + size, "MemTotal: %9d kB\n", kstat.total_mem_pag
es << 2);
  232:          size += sprintk(buffer + size, "MemFree:  %9d kB\n", kstat.free_pages <<
 2);
  233:          size += sprintk(buffer + size, "MemShared:%9d kB\n", kstat.shared);
  234:          size += sprintk(buffer + size, "Buffers:  %9d kB\n", kstat.buffers);
  235:          size += sprintk(buffer + size, "Cached:   %9d kB\n", kstat.cached);
  236:          size += sprintk(buffer + size, "SwapTotal:%9d kB\n", 0);
  237:          size += sprintk(buffer + size, "SwapFree: %9d kB\n", 0);
  238:          return size;
  239: }
  240:
  241: int data_proc_mounts(char *buffer, __pid_t pid)
  242: {
```

fs/procfs/data.c


```
  243:         int n, size;
  244:         char *flag;
  245:
  246:         size = 0;
  247:         for(n = 0; n < NR_MOUNT_POINTS; n++) {
  248:                 if(mount_table[n].used) {
  249:                         if(mount_table[n].fs->fsop->flags != FSOP_KERN_MOUNT) {
  250:                                 flag = "rw";
  251:                                 if(mount_table[n].sb.flags & MS_RDONLY) {
  252:                                         flag = "ro";
  253:                                 }
  254:                                 size += sprintk(buffer + size, "%s %s %s %s 0 0\
n", mount_table[n].devname, mount_table[n].dirname, mount_table[n].fs->name, flag);
  255:                         }
  256:                 }
  257:         }
  258:         return size;
  259: }
  260:
  261: int data_proc_partitions(char *buffer, __pid_t pid)
  262: {
  263:         int n, ctrl, drv, size;
  264:         int minor, major;
  265:         unsigned int blocks;
  266:         struct ide *ide;
  267:         struct ide_drv *drive;
  268:
  269:         size = 0;
  270:         size += sprintk(buffer + size, "major minor  #blocks  name\n\n");
  271:
  272:         for(ctrl = 0; ctrl < NR_IDE_CTRLS; ctrl++) {
  273:                 ide = &ide_table[ctrl];
  274:                 for(drv = 0; drv < NR_IDE_DRVS; drv++) {
  275:                         drive = &ide->drive[drv];
  276:                         if(!drive->nr_sects) {
  277:                                 continue;
  278:                         }
  279:                         if(drive->flags & DEVICE_IS_DISK) {
  280:                                 major = (int)drive->major;
  281:                                 minor = (int)drive->minor_shift;
  282:                                 blocks = drive->nr_sects / 2;
  283:                                 size += sprintk(buffer + size, "%4d  %4d  %9d %s
\n", major, 0, blocks, drive->dev_name);
  284:                                 for(n = 0; n < NR_PARTITIONS; n++) {
  285:                                         if(drive->part_table[n].type) {
  286:                                                 blocks = drive->part_table[n].nr
_sects / 2;
  287:                                                 size += sprintk(buffer + size, "
%4d  %4d  %9u %s%d\n", major, (n + 1) << minor, blocks, drive->dev_name, n + 1);
  288:                                         }
  289:                                 }
  290:                         }
  291:                 }
  292:         }
  293:         return size;
  294: }
  295:
  296: int data_proc_rtc(char *buffer, __pid_t pid)
  297: {
  298:         int size;
  299:         short int sec, min, hour;
  300:         short int day, month, year, century;
  301:
  302:         sec = cmos_read_date(CMOS_SEC);
  303:         min = cmos_read_date(CMOS_MIN);
  304:         hour = cmos_read_date(CMOS_HOUR);
  305:         day = cmos_read_date(CMOS_DAY);
```

fs/procfs/data.c


```
  306:          month = cmos_read_date(CMOS_MONTH);
  307:          year = cmos_read_date(CMOS_YEAR);
  308:          century = cmos_read_date(CMOS_CENTURY);
  309:          year += century * 100;
  310:
  311:          size = 0;
  312:          size += sprintk(buffer + size, "rtc_time\t: %02d:%02d:%02d\n", hour, min
, sec);
  313:          size += sprintk(buffer + size, "rtc_date\t: %02d-%02d-%02d\n", year, mon
th, day);
  314:          sec = cmos_read_date(CMOS_ASEC);
  315:          min = cmos_read_date(CMOS_AMIN);
  316:          hour = cmos_read_date(CMOS_AHOUR);
  317:          size += sprintk(buffer + size, "alarm\t\t: %02d:%02d:%02d\n", hour, min,
 sec);
  318:          size += sprintk(buffer + size, "DST_enable\t: %s\n", cmos_read(CMOS_STAT
B) & CMOS_STATB_DSE ? "yes" : "no");
  319:          size += sprintk(buffer + size, "BCD\t\t: %s\n", cmos_read(CMOS_STATB) &
CMOS_STATB_DM ? "no" : "yes");
  320:          size += sprintk(buffer + size, "24hr\t\t: %s\n", cmos_read(CMOS_STATB) &
 CMOS_STATB_24H ? "yes" : "no");
  321:          size += sprintk(buffer + size, "square_wave\t: %s\n", cmos_read(CMOS_STA
TB) & CMOS_STATB_SQWE ? "yes" : "no");
  322:          size += sprintk(buffer + size, "alarm_IRQ\t: %s\n", cmos_read(CMOS_STATB
) & CMOS_STATB_AIE ? "yes" : "no");
  323:          size += sprintk(buffer + size, "update_IRQ\t: %s\n", cmos_read(CMOS_STAT
B) & CMOS_STATB_UIE ? "yes" : "no");
  324:          size += sprintk(buffer + size, "periodic_IRQ\t: %s\n", cmos_read(CMOS_ST
ATB) & CMOS_STATB_PIE ? "yes" : "no");
  325:          size += sprintk(buffer + size, "periodic_freq\t: %s\n", (cmos_read(CMOS_
STATA) & CMOS_STATA_IRQF) == 0x6 ? "1024" : "?");
  326:          size += sprintk(buffer + size, "batt_status\t: %s\n", cmos_read(CMOS_STA
TD) & CMOS_STATD_VRT ? "okay" : "dead");
  327:          return size;
  328: }
  329:
  330: int data_proc_stat(char *buffer, __pid_t pid)
  331: {
  332:          int n, size;
  333:          unsigned int idle;
  334:
  335:          idle = kstat.ticks - (kstat.cpu_user + kstat.cpu_nice + kstat.cpu_system
);
  336:          size = 0;
  337:          size += sprintk(buffer + size, "cpu %d %d %d %d\n", kstat.cpu_user, ksta
t.cpu_nice, kstat.cpu_system, idle);
  338:          size += sprintk(buffer + size, "disk 0 0 0 0\n");
  339:          size += sprintk(buffer + size, "page 0 0\n");
  340:          size += sprintk(buffer + size, "swap 0 0\n");
  341:          size += sprintk(buffer + size, "intr %u", kstat.irqs);
  342:          for(n = 0; n < NR_IRQS; n++) {
  343:                  size += sprintk(buffer + size, " %u", irq_table[n].ticks);
  344:          }
  345:          size += sprintk(buffer + size, "\n");
  346:          size += sprintk(buffer + size, "ctxt %u\n", kstat.ctxt);
  347:          size += sprintk(buffer + size, "btime %d\n", kstat.boot_time);
  348:          size += sprintk(buffer + size, "processes %d\n", kstat.processes);
  349:          return size;
  350: }
  351:
  352: int data_proc_uptime(char *buffer, __pid_t pid)
  353: {
  354:          struct proc *p;
  355:          unsigned long int idle;
  356:
  357:          p = &proc_table[IDLE];
  358:          idle = tv2ticks(&p->usage.ru_utime);
```

fs/procfs/data.c


```
  359:         idle += tv2ticks(&p->usage.ru_stime);
  360:         return sprintk(buffer, "%u.%02u %u.%02u\n", kstat.uptime, kstat.ticks %
HZ, idle / HZ, idle % HZ);
  361: }
  362:
  363: int data_proc_fullversion(char *buffer, __pid_t pid)
  364: {
  365:         return sprintk(buffer, "Fiwix version %s %s\n", UTS_RELEASE, UTS_VERSION
);
  366: }
  367:
  368: int data_proc_domainname(char *buffer, __pid_t pid)
  369: {
  370:         return sprintk(buffer, "%s\n", sys_utsname.domainname);
  371: }
  372:
  373: int data_proc_filemax(char *buffer, __pid_t pid)
  374: {
  375:         return sprintk(buffer, "%d\n", NR_OPENS);
  376: }
  377:
  378: int data_proc_filenr(char *buffer, __pid_t pid)
  379: {
  380:         int n, nr;
  381:
  382:         nr = 0;
  383:         for(n = 1; n < NR_OPENS; n++) {
  384:                 if(fd_table[n].count != 0) {
  385:                         nr++;
  386:                 }
  387:         }
  388:         return sprintk(buffer, "%d\n", nr);
  389: }
  390:
  391: int data_proc_hostname(char *buffer, __pid_t pid)
  392: {
  393:         return sprintk(buffer, "%s\n", sys_utsname.nodename);
  394: }
  395:
  396: int data_proc_inodemax(char *buffer, __pid_t pid)
  397: {
  398:         return sprintk(buffer, "%d\n", inode_table_size / sizeof(struct inode));
  399: }
  400:
  401: int data_proc_inodenr(char *buffer, __pid_t pid)
  402: {
  403:         return sprintk(buffer, "%d\n", (inode_table_size / sizeof(struct inode))
 - inodes_on_free_list);
  404: }
  405:
  406: int data_proc_osrelease(char *buffer, __pid_t pid)
  407: {
  408:         return sprintk(buffer, "%s\n", UTS_RELEASE);
  409: }
  410:
  411: int data_proc_ostype(char *buffer, __pid_t pid)
  412: {
  413:         return sprintk(buffer, "%s\n", UTS_SYSNAME);
  414: }
  415:
  416: int data_proc_version(char *buffer, __pid_t pid)
  417: {
  418:         return sprintk(buffer, "%s\n", UTS_VERSION);
  419: }
  420:
  421:
  422: /*
```

fs/procfs/data.c


```
  423:  * PID directory related functions
  424:  * -------------------------------
  425:  */
  426: int data_proc_pid_cmdline(char *buffer, __pid_t pid)
  427: {
  428:         int n, size;
  429:         char *arg;
  430:         char **argv;
  431:         unsigned int paddr, offset;
  432:         struct proc *p;
  433:
  434:         size = 0;
  435:         if((p = get_proc_by_pid(pid))) {
  436:                 if(p->argv) {
  437:                         offset = (int)p->argv & ~PAGE_MASK;
  438:                         paddr = get_mapped_addr(p, (int)p->argv) & PAGE_MASK;
  439:                         paddr = P2V(paddr);
  440:                         argv = (char **)(paddr + offset);
  441:                         for(n = 0; argv[n]; n++) {
  442:                                 offset = (int)argv[n] & ~PAGE_MASK;
  443:                                 paddr = get_mapped_addr(p, (int)argv[n]) & PAGE_
MASK;
  444:                                 paddr = P2V(paddr);
  445:                                 arg = (char *)(paddr + offset);
  446:                                 size += sprintk(buffer + size, "%s", arg);
  447:                                 buffer[size++] = NULL;
  448:                         }
  449:                 }
  450:         }
  451:         return size;
  452: }
  453:
  454: int data_proc_pid_cwd(char *buffer, __pid_t pid)
  455: {
  456:         int size;
  457:         struct proc *p;
  458:         struct inode *i;
  459:
  460:         size = 0;
  461:         if((p = get_proc_by_pid(pid))) {
  462:
  463:                 /* zombie processes don't have current working directory */
  464:                 if(!p->pwd) {
  465:                         return -ENOENT;
  466:                 }
  467:
  468:                 i = p->pwd;
  469:                 size = sprintk(buffer, "[%02d%02d]:%d", MAJOR(i->rdev), MINOR(i-
>rdev), i->inode);
  470:         }
  471:         return size;
  472: }
  473:
  474: int data_proc_pid_environ(char *buffer, __pid_t pid)
  475: {
  476:         int n, size;
  477:         char *env;
  478:         char **envp;
  479:         unsigned int paddr, offset;
  480:         struct proc *p;
  481:
  482:         size = 0;
  483:         if((p = get_proc_by_pid(pid))) {
  484:                 if(p->envp) {
  485:                         offset = (int)p->envp & ~PAGE_MASK;
  486:                         paddr = get_mapped_addr(p, (int)p->envp) & PAGE_MASK;
  487:                         paddr = P2V(paddr);
```

fs/procfs/data.c


```
  488:                          envp = (char **)(paddr + offset);
  489:                          for(n = 0; envp[n]; n++) {
  490:                                  offset = (int)envp[n] & ~PAGE_MASK;
  491:                                  paddr = get_mapped_addr(p, (int)envp[n]) & PAGE_
MASK;
  492:                                  paddr = P2V(paddr);
  493:                                  env = (char *)(paddr + offset);
  494:                                  size += sprintk(buffer + size, "%s", env);
  495:                                  buffer[size++] = NULL;
  496:                          }
  497:                  }
  498:          }
  499:          return size;
  500: }
  501:
  502: int data_proc_pid_exe(char *buffer, __pid_t pid)
  503: {
  504:          int size;
  505:          struct proc *p;
  506:          struct inode *i;
  507:
  508:          size = 0;
  509:          if((p = get_proc_by_pid(pid))) {
  510:
  511:                  /* kernel and zombie processes are programless */
  512:                  if(!p->vma || !p->vma->inode) {
  513:                          return -ENOENT;
  514:                  }
  515:
  516:                  i = p->vma->inode;
  517:                  size = sprintk(buffer, "[%02d%02d]:%d", MAJOR(i->rdev), MINOR(i-
>rdev), i->inode);
  518:          }
  519:          return size;
  520: }
  521:
  522: int data_proc_pid_maps(char *buffer, __pid_t pid)
  523: {
  524:          unsigned int n;
  525:          int size, len;
  526:          __ino_t inode;
  527:          int major, minor;
  528:          char *section;
  529:          char r, w, x, f;
  530:          struct proc *p;
  531:          struct vma *vma;
  532:
  533:          size = 0;
  534:          if((p = get_proc_by_pid(pid))) {
  535:                  if(!p->vma) {
  536:                          return 0;
  537:                  }
  538:                  vma = p->vma;
  539:                  for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  540:                          r = vma->prot & PROT_READ ? 'r' : '-';
  541:                          w = vma->prot & PROT_WRITE ? 'w' : '-';
  542:                          x = vma->prot & PROT_EXEC ? 'x' : '-';
  543:                          if(vma->flags & MAP_SHARED) {
  544:                                  f = 's';
  545:                          } else if(vma->flags & MAP_PRIVATE) {
  546:                                  f = 'p';
  547:                          } else {
  548:                                  f = '-';
  549:                          }
  550:                          switch(vma->s_type) {
  551:                                  case P_TEXT:    section = "text";
  552:                                                  break;
```

fs/procfs/data.c


```
  553:                                  case P_DATA:    section = "data";
  554:                                                  break;
  555:                                  case P_BSS:     section = "bss";
  556:                                                  break;
  557:                                  case P_HEAP:    section = "heap";
  558:                                                  break;
  559:                                  case P_STACK:   section = "stack";
  560:                                                  break;
  561:                                  case P_MMAP:    section = "mmap";
  562:                                                  break;
  563:                                  default:
  564:                                          section = NULL;
  565:                                          break;
  566:                          }
  567:                          inode = major = minor = 0;
  568:                          if(vma->inode) {
  569:                                  inode = vma->inode->inode;
  570:                                  major = MAJOR(vma->inode->dev);
  571:                                  minor = MINOR(vma->inode->dev);
  572:                          }
  573:                          len = sprintk(buffer + size, "%08x-%08x %c%c%c%c %08x %0
2d:%02d %- 10u [%s]\n", vma->start, vma->end, r, w, x, f, vma->offset, major, minor, in
ode, section);
  574:                          size += len;
  575:                  }
  576:          }
  577:          return size;
  578: }
  579:
  580: int data_proc_pid_mountinfo(char *buffer, __pid_t pid)
  581: {
  582:          int n, size;
  583:          char *flag, *devname;
  584:
  585:          size = 0;
  586:          for(n = 0; n < NR_MOUNT_POINTS; n++) {
  587:                  if(mount_table[n].used) {
  588:                          if(mount_table[n].fs->fsop->flags != FSOP_KERN_MOUNT) {
  589:                                  flag = "rw";
  590:                                  if(mount_table[n].sb.flags & MS_RDONLY) {
  591:                                          flag = "ro";
  592:                                  }
  593:                                  devname = mount_table[n].devname;
  594:                                  if(!strcmp(mount_table[n].devname, "/dev/root"))
 {
  595:                                          devname = _rootdevname;
  596:                                  }
  597:                                  size += sprintk(buffer + size, "%d 0 %d:%d %s %s
 %s - %s %s %s\n", n, MAJOR(mount_table[n].dev), MINOR(mount_table[n].dev), "/", mount_
table[n].dirname, flag, mount_table[n].fs->name, devname, flag);
  598:                          }
  599:                  }
  600:          }
  601:          return size;
  602: }
  603:
  604: int data_proc_pid_root(char *buffer, __pid_t pid)
  605: {
  606:          int size;
  607:          struct proc *p;
  608:          struct inode *i;
  609:
  610:          size = 0;
  611:          if((p = get_proc_by_pid(pid))) {
  612:
  613:                  /* zombie processes don't have root directory */
  614:                  if(!p->root) {
```

fs/procfs/data.c


```
  615:                          return -ENOENT;
  616:                  }
  617:
  618:                  i = p->root;
  619:                  size = sprintk(buffer, "[%02d%02d]:%d", MAJOR(i->rdev), MINOR(i-
>rdev), i->inode);
  620:          }
  621:          return size;
  622: }
  623:
  624: int data_proc_pid_stat(char *buffer, __pid_t pid)
  625: {
  626:          int n, size, vma_start, vma_end;
  627:          unsigned int esp, eip;
  628:          int signum, mask;
  629:          __sigset_t sigignored, sigcaught;
  630:          struct proc *p;
  631:          struct sigcontext *sc;
  632:          struct vma *vma;
  633:          int text, data, stack, mmap;
  634:
  635:          size = vma_start = vma_end = 0;
  636:          text = data = stack = mmap = 0;
  637:          if((p = get_proc_by_pid(pid))) {
  638:                  if(!p->vma) {
  639:                          return 0;
  640:                  }
  641:                  vma_start = p->vma[0].start;
  642:                  vma_end = p->vma[0].end;
  643:
  644:                  vma = p->vma;
  645:                  for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  646:                          switch(vma->s_type) {
  647:                                  case P_TEXT:
  648:                                          text += vma->end - vma->start;
  649:                                          break;
  650:                                  case P_HEAP:
  651:                                          data += vma->end - vma->start;
  652:                                          break;
  653:                                  case P_STACK:
  654:                                          stack += vma->end - vma->start;
  655:                                          break;
  656:                                  case P_MMAP:
  657:                                          mmap += vma->end - vma->start;
  658:                                          break;
  659:                          }
  660:                  }
  661:
  662:                  sigignored = sigcaught = 0;
  663:                  for(signum = 0, mask = 1; signum < NSIG; signum++, mask <<= 1) {
  664:                          if(p->sigaction[signum].sa_handler == SIG_IGN) {
  665:                                  sigignored |= mask;
  666:                          }
  667:                          if(p->sigaction[signum].sa_handler == SIG_DFL) {
  668:                                  sigcaught |= mask;
  669:                          }
  670:                  }
  671:
  672:                  esp = eip = 0;
  673:                  if(p->sp) {
  674:                          sc = (struct sigcontext *)p->sp;
  675:                          esp = sc->oldesp;
  676:                          eip = sc->eip;
  677:                  }
  678:                  size = sprintk(buffer, "%d (%s) %c %d %d %d %d %d %d %d %d %d %d
 %u %u %u %u %d %d %d %d %d %d %u %u %u %u %u %u %u %d %d %u %u %u\n",
  679:                  p->pid,
```

fs/procfs/data.c


```
  680:                 p->argv0,
  681:                 pstate[p->state][0],
  682:                 p->ppid, p->pgid, p->sid,
  683:                 p->ctty ? p->ctty->dev : 0,
  684:                 p->ctty ? p->ctty->pgid : - 1,
  685:                 0,                      /* flags */
  686:                 0, 0, 0, 0,             /* minflt, cminflt, majflt, cmajflt */
  687:                 tv2ticks(&p->usage.ru_utime),
  688:                 tv2ticks(&p->usage.ru_stime),
  689:                 tv2ticks(&p->cusage.ru_utime),
  690:                 tv2ticks(&p->cusage.ru_stime),
  691:                 0,                      /* counter */
  692:                 0,                      /* priority */
  693:                 0,                      /* timeout */
  694:                 0,                      /* itrealvalue */
  695:                 p->start_time,
  696:                 text + data + stack + mmap,
  697:                 p->rss,
  698:                 0x7FFFFFFF,             /* rlim */
  699:                 vma_start,              /* startcode */
  700:                 vma_end,                /* endcode */
  701:                 KERNEL_BASE_ADDR - 1,   /* startstack */
  702:                 esp,                    /* kstkesp */
  703:                 eip,                    /* kstkeip */
  704:                 p->sigpending,
  705:                 p->sigblocked,
  706:                 sigignored,
  707:                 sigcaught,
  708:                 p->sleep_address
  709:                 );
  710:         }
  711:         return size;
  712: }
  713:
  714: int data_proc_pid_statm(char *buffer, __pid_t pid)
  715: {
  716:         int n, size;
  717:         struct proc *p;
  718:         struct vma *vma;
  719:         int text, data, stack, mmap;
  720:
  721:         size = text = data = stack = mmap = 0;
  722:         if((p = get_proc_by_pid(pid))) {
  723:                 if(!p->vma) {
  724:                         return 0;
  725:                 }
  726:                 vma = p->vma;
  727:                 for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  728:                         switch(vma->s_type) {
  729:                                 case P_TEXT:
  730:                                         text += vma->end - vma->start;
  731:                                         break;
  732:                                 case P_HEAP:
  733:                                         data += vma->end - vma->start;
  734:                                         break;
  735:                                 case P_STACK:
  736:                                         stack += vma->end - vma->start;
  737:                                         break;
  738:                                 case P_MMAP:
  739:                                         mmap += vma->end - vma->start;
  740:                                         break;
  741:                         }
  742:                 }
  743:
  744:                 size = sprintk(buffer, "%d", (text + data + stack + mmap) / PAGE
_SIZE);
  745:                 size += sprintk(buffer + size, " %d", p->rss);
```

fs/procfs/data.c


```
  746:                 size += sprintk(buffer + size, " 0");   /* shared mappings */
  747:                 size += sprintk(buffer + size, " %d", text / PAGE_SIZE);
  748:                 size += sprintk(buffer + size, " 0");
  749:                 size += sprintk(buffer + size, " %d", (data + stack) / PAGE_SIZE
);
  750:                 size += sprintk(buffer + size, " 0\n");
  751:         }
  752:         return size;
  753: }
  754:
  755: int data_proc_pid_status(char *buffer, __pid_t pid)
  756: {
  757:         int n, size;
  758:         int signum, mask;
  759:         __sigset_t sigignored, sigcaught;
  760:         struct proc *p;
  761:         struct vma *vma;
  762:         int text, data, stack, mmap;
  763:
  764:         size = text = data = stack = mmap = 0;
  765:         if((p = get_proc_by_pid(pid))) {
  766:                 if(!p->vma) {
  767:                         return 0;
  768:                 }
  769:                 vma = p->vma;
  770:                 for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  771:                         switch(vma->s_type) {
  772:                                 case P_TEXT:
  773:                                         text += vma->end - vma->start;
  774:                                         break;
  775:                                 case P_HEAP:
  776:                                         data += vma->end - vma->start;
  777:                                         break;
  778:                                 case P_STACK:
  779:                                         stack += vma->end - vma->start;
  780:                                         break;
  781:                                 case P_MMAP:
  782:                                         mmap += vma->end - vma->start;
  783:                                         break;
  784:                         }
  785:                 }
  786:
  787:                 size = sprintk(buffer, "Name:\t%s\n", p->argv0);
  788:                 size += sprintk(buffer + size, "State:\t%s\n", pstate[p->state])
;
  789:                 size += sprintk(buffer + size, "Pid:\t%d\n", p->pid);
  790:                 size += sprintk(buffer + size, "PPid:\t%d\n", p->ppid);
  791:                 size += sprintk(buffer + size, "Uid:\t%d\t%d\t%d\t-\n", p->uid,
p->euid, p->suid);
  792:                 size += sprintk(buffer + size, "Gid:\t%d\t%d\t%d\t-\n", p->gid,
p->egid, p->sgid);
  793:                 size += sprintk(buffer + size, "VmSize:\t%8d kB\n", (text + data
 + stack + mmap) / 1024);
  794:                 size += sprintk(buffer + size, "VmLck:\t%8d kB\n", 0);
  795:                 size += sprintk(buffer + size, "VmRSS:\t%8d kB\n", p->rss << 2);
  796:                 size += sprintk(buffer + size, "VmData:\t%8d kB\n", data / 1024)
;
  797:                 size += sprintk(buffer + size, "VmStk:\t%8d kB\n", stack / 1024)
;
  798:                 size += sprintk(buffer + size, "VmExe:\t%8d kB\n", text / 1024);
  799:                 size += sprintk(buffer + size, "VmLib:\t%8d kB\n", 0);
  800:                 size += sprintk(buffer + size, "SigPnd:\t%08x\n", p->sigpending)
;
  801:                 size += sprintk(buffer + size, "SigBlk:\t%08x\n", p->sigblocked)
;
  802:                 sigignored = sigcaught = 0;
  803:                 for(signum = 0, mask = 1; signum < NSIG; signum++, mask <<= 1) {
```

fs/procfs/data.c


```
804:                            if(p->sigaction[signum].sa_handler == SIG_IGN) {
805:                                    sigignored |= mask;
806:                            }
807:                            if(p->sigaction[signum].sa_handler == SIG_DFL) {
808:                                    sigcaught |= mask;
809:                            }
810:                    }
811:                    size += sprintk(buffer + size, "SigIgn:\t%08x\n", sigignored);
812:                    size += sprintk(buffer + size, "SigCgt:\t%08x\n", sigcaught);
813:            }
814:            return size;
815: }
```

fs/procfs/dir.c


```
/*
 * fiwix/fs/procfs/dir.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/fs.h>
#include <fiwix/filesystems.h>
#include <fiwix/fs_proc.h>
#include <fiwix/dirent.h>
#include <fiwix/stat.h>
#include <fiwix/mm.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct fs_operations procfs_dir_fsop = {
        0,
        0,

        procfs_dir_open,
        procfs_dir_close,
        procfs_dir_read,
        NULL,                   /* write */
        NULL,                   /* ioctl */
        NULL,                   /* lseek */
        procfs_dir_readdir,
        NULL,                   /* mmap */
        NULL,                   /* select */

        NULL,                   /* readlink */
        NULL,                   /* followlink */
        procfs_bmap,
        procfs_lookup,
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        NULL,                   /* truncate */
        NULL,                   /* create */
        NULL,                   /* rename */

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statfs */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

static int proc_listdir(char *buffer)
{
        int n;
        struct proc *p;
        struct procfs_dir_entry *pd;
        struct procfs_dir_entry d;
        int size;
```

fs/procfs/dir.c


```
   68:
   69:         size = 0;
   70:         pd = (struct procfs_dir_entry *)buffer;
   71:
   72:         FOR_EACH_PROCESS(p) {
   73:                 if(p->state) {
   74:                         d.inode = PROC_PID_INO + (p->pid << 12);
   75:                         d.mode = S_IFDIR | S_IRUSR | S_IXUSR | S_IRGRP | S_IXGRP
 | S_IROTH | S_IXOTH;
   76:                         d.nlink = 1;
   77:                         d.lev = -1;
   78:                         d.name_len = 1;
   79:                         n = p->pid;
   80:                         while(n / 10) {
   81:                                 n /= 10;
   82:                                 d.name_len++;
   83:                         }
   84:                         d.name = p->pidstr;
   85:                         d.data_fn = NULL;
   86:
   87:                         if(size + sizeof(struct procfs_dir_entry) >= PAGE_SIZE)
{
   88:                                 printk("WARNING: kmalloc() is limited to 4096 by
tes.\n");
   89:                                 break;
   90:                         }
   91:
   92:                         size += sizeof(struct procfs_dir_entry);
   93:                         memcpy_b((void *)pd, (void *)&d, sizeof(struct procfs_di
r_entry));
   94:                         pd++;
   95:                 }
   96:         }
   97:         return size;
   98: }
   99:
  100: /*
  101: static int proc_listfd(struct inode *i, char *buffer)
  102: {
  103:         int n;
  104:         struct proc *p;
  105:         struct procfs_dir_entry *pd;
  106:         struct procfs_dir_entry d;
  107:         int size;
  108:
  109:         size = 0;
  110:         pd = (struct procfs_dir_entry *)buffer;
  111:
  112:         p = get_proc_by_pid((i->inode >> 12) & 0xFFFF);
  113:         for(n = 0; n < OPEN_MAX; n++) {
  114:                 if(p->fd[n]) {
  115:                         d.inode = PROC_PID_INO + (p->pid << 12) + n;
  116:                         d.mode = S_IFREG | S_IRWXU;
  117:                         d.nlink = 1;
  118:                         d.lev = -1;
  119:                         d.name_len = sprintk(d.name, "%d", n);
  120:                         d.data_fn = NULL;
  121:
  122:                         if(size + sizeof(struct procfs_dir_entry) >= 4096) {
  123:                                 printk("WARNING: kmalloc() is limited to 4096 by
tes.\n");
  124:                                 break;
  125:                         }
  126:
  127:                         size += sizeof(struct procfs_dir_entry);
  128:                         memcpy_b((void *)pd, (void *)&d, sizeof(struct procfs_di
r_entry));
```

fs/procfs/dir.c


```
  129:                          pd++;
  130:                  }
  131:          }
  132:          memset_b((void *)pd + size, NULL, sizeof(struct procfs_dir_entry));
  133:          return size;
  134: }
  135: */
  136:
  137: int procfs_dir_open(struct inode *i, struct fd *fd_table)
  138: {
  139:          fd_table->offset = 0;
  140:          return 0;
  141: }
  142:
  143: int procfs_dir_close(struct inode *i, struct fd *fd_table)
  144: {
  145:          return 0;
  146: }
  147:
  148: int procfs_dir_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t
 count)
  149: {
  150:          __off_t total_read;
  151:          unsigned int bytes;
  152:          int len, lev;
  153:          char *buf;
  154:
  155:          if(!(buf = (void *)kmalloc())) {
  156:                  return -ENOMEM;
  157:          }
  158:
  159:          /* create the list of directories for each process */
  160:          len = 0;
  161:          if(i->inode == PROC_ROOT_INO) {
  162:                  len = proc_listdir(buf);
  163:          }
  164:
  165:          /* TODO: create the list of fds used for each process
  166:          if((i->inode & 0xF0000FFF) == PROC_PID_FD) {
  167:                  len = proc_listfd(i, buf);
  168:          }
  169:          */
  170:
  171:          /* add the rest of static files in the main directory */
  172:          lev = i->u.procfs.i_lev;
  173:
  174:          /* assigns the size of the level without the last entry (NULL) */
  175:          bytes = sizeof(procfs_array[lev]) - sizeof(struct procfs_dir_entry);
  176:
  177:          if((len + bytes) > (PAGE_SIZE - 1)) {
  178:                  printk("WARNING: %s(): len > 4095 (%d).\n", __FUNCTION__, len);
  179:          }
  180:          memcpy_b(buf + len, (char *)&procfs_array[lev], bytes);
  181:          len += bytes;
  182:          total_read = fd_table->offset = len;
  183:          memcpy_b(buffer, buf, PAGE_SIZE);
  184:          kfree((unsigned int)buf);
  185:          return total_read;
  186: }
  187:
  188: int procfs_dir_readdir(struct inode *i, struct fd *fd_table, struct dirent *dire
nt, unsigned int count)
  189: {
  190:          unsigned int offset, boffset, dirent_offset, doffset;
  191:          int dirent_len;
  192:          unsigned int total_read;
  193:          struct procfs_dir_entry *d;
```

fs/procfs/dir.c


```
  194:          int base_dirent_len;
  195:          char *buffer;
  196:          int lev;
  197:
  198:          if(!i->fsop || !i->fsop->read) {
  199:                  return -EBADF;
  200:          }
  201:          if(!(buffer = (void *)kmalloc())) {
  202:                  return -ENOMEM;
  203:          }
  204:
  205:          lev = i->u.procfs.i_lev;
  206:          base_dirent_len = sizeof(dirent->d_ino) + sizeof(dirent->d_off) + sizeof
(dirent->d_reclen);
  207:
  208:          offset = fd_table->offset;
  209:          boffset = dirent_offset = doffset = 0;
  210:
  211:          boffset = offset % PAGE_SIZE;
  212:
  213:          total_read = i->fsop->read(i, fd_table, buffer, PAGE_SIZE);
  214:          if((count = MIN(total_read, count)) == 0) {
  215:                  kfree((unsigned int)buffer);
  216:                  return dirent_offset;
  217:          }
  218:
  219:          while(boffset < fd_table->offset) {
  220:                  d = (struct procfs_dir_entry *)(buffer + boffset);
  221:                  if(!d->inode) {
  222:                          break;
  223:                  }
  224:                  dirent_len = (base_dirent_len + (d->name_len + 1)) + 3;
  225:                  dirent_len &= ~3;       /* round up */
  226:                  if((doffset + dirent_len) <= count) {
  227:                          boffset += sizeof(struct procfs_dir_entry);
  228:                          offset += sizeof(struct procfs_dir_entry);
  229:                          doffset += dirent_len;
  230:                          dirent->d_ino = d->inode;
  231:                          dirent->d_off = offset;
  232:                          dirent->d_reclen = dirent_len;
  233:                          memcpy_b(dirent->d_name, d->name, d->name_len);
  234:                          dirent->d_name[d->name_len] = NULL;
  235:                          dirent = (struct dirent *)((char *)dirent + dirent_len);
  236:                          dirent_offset += dirent_len;
  237:                  } else {
  238:                          break;
  239:                  }
  240:          }
  241:          fd_table->offset = boffset;
  242:          kfree((unsigned int)buffer);
  243:          return dirent_offset;
  244: }
```

fs/procfs/file.c


```
 1: /*
 2:  * fiwix/fs/procfs/file.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/errno.h>
10: #include <fiwix/fs.h>
11: #include <fiwix/filesystems.h>
12: #include <fiwix/fs_proc.h>
13: #include <fiwix/fcntl.h>
14: #include <fiwix/mm.h>
15: #include <fiwix/stdio.h>
16: #include <fiwix/string.h>
17:
18: struct fs_operations procfs_file_fsop = {
19:         0,
20:         0,
21:
22:         procfs_file_open,
23:         procfs_file_close,
24:         procfs_file_read,
25:         NULL,                   /* write */
26:         NULL,                   /* ioctl */
27:         procfs_file_lseek,
28:         NULL,                   /* readdir */
29:         NULL,                   /* mmap */
30:         NULL,                   /* select */
31:
32:         NULL,                   /* readlink */
33:         NULL,                   /* followlink */
34:         procfs_bmap,
35:         NULL,                   /* lookup */
36:         NULL,                   /* rmdir */
37:         NULL,                   /* link */
38:         NULL,                   /* unlink */
39:         NULL,                   /* symlink */
40:         NULL,                   /* mkdir */
41:         NULL,                   /* mknod */
42:         NULL,                   /* truncate */
43:         NULL,                   /* create */
44:         NULL,                   /* rename */
45:
46:         NULL,                   /* read_block */
47:         NULL,                   /* write_block */
48:
49:         NULL,                   /* read_inode */
50:         NULL,                   /* write_inode */
51:         NULL,                   /* ialloc */
52:         NULL,                   /* ifree */
53:         NULL,                   /* statfs */
54:         NULL,                   /* read_superblock */
55:         NULL,                   /* remount_fs */
56:         NULL,                   /* write_superblock */
57:         NULL                    /* release_superblock */
58: };
59:
60: int procfs_file_open(struct inode *i, struct fd *fd_table)
61: {
62:         if(fd_table->flags & (O_WRONLY | O_RDWR | O_TRUNC | O_APPEND)) {
63:                 return -EINVAL;
64:         }
65:         fd_table->offset = 0;
66:         return 0;
67: }
```

fs/procfs/file.c


```
   68:
   69: int procfs_file_close(struct inode *i, struct fd *fd_table)
   70: {
   71:         return 0;
   72: }
   73:
   74: int procfs_file_read(struct inode *i, struct fd *fd_table, char *buffer, __size_
t count)
   75: {
   76:         __off_t total_read;
   77:         unsigned int boffset, bytes, size;
   78:         int blksize;
   79:         struct procfs_dir_entry *d;
   80:         char *buf;
   81:
   82:         if(!(d = get_procfs_by_inode(i))) {
   83:                 return -EINVAL;
   84:         }
   85:         if(!d->data_fn) {
   86:                 return -EINVAL;
   87:         }
   88:         if(!(buf = (void *)kmalloc())) {
   89:                 return -ENOMEM;
   90:         }
   91:
   92:         size = d->data_fn(buf, (i->inode >> 12) & 0xFFFF);
   93:         blksize = i->sb->s_blocksize;
   94:         if(fd_table->offset > size) {
   95:                 fd_table->offset = size;
   96:         }
   97:
   98:         total_read = 0;
   99:
  100:         for(;;) {
  101:                 count = (fd_table->offset + count > size) ? size - fd_table->off
set : count;
  102:                 if(!count) {
  103:                         break;
  104:                 }
  105:
  106:                 boffset = fd_table->offset % blksize;
  107:                 bytes = blksize - boffset;
  108:                 bytes = MIN(bytes, count);
  109:                 memcpy_b(buffer + total_read, buf + boffset, bytes);
  110:                 total_read += bytes;
  111:                 count -= bytes;
  112:                 boffset += bytes;
  113:                 boffset %= blksize;
  114:                 fd_table->offset += bytes;
  115:         }
  116:
  117:         kfree((unsigned int)buf);
  118:         return total_read;
  119: }
  120:
  121: int procfs_file_lseek(struct inode *i, __off_t offset)
  122: {
  123:         return offset;
  124: }
```

fs/procfs/inode.c


```
    1: /*
    2:  * fiwix/fs/procfs/inode.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/filesystems.h>
   11: #include <fiwix/fs_proc.h>
   12: #include <fiwix/statfs.h>
   13: #include <fiwix/sleep.h>
   14: #include <fiwix/stat.h>
   15: #include <fiwix/sched.h>
   16: #include <fiwix/mm.h>
   17: #include <fiwix/process.h>
   18: #include <fiwix/errno.h>
   19: #include <fiwix/stdio.h>
   20: #include <fiwix/string.h>
   21:
   22: int procfs_read_inode(struct inode *i)
   23: {
   24:         int lev;
   25:         __mode_t mode;
   26:         __nlink_t nlink;
   27:         struct procfs_dir_entry *d;
   28:
   29:         if((i->inode & 0xF0000FFF) == PROC_PID_INO) {   /* dynamic PID dir */
   30:                 mode = S_IFDIR | S_IRUSR | S_IXUSR | S_IRGRP | S_IXGRP | S_IROTH
 | S_IXOTH;
   31:                 nlink = 3;
   32:                 lev = PROC_PID_LEV;
   33:         } else {
   34:                 if(!(d = get_procfs_by_inode(i))) {
   35:                         return NULL;
   36:                 }
   37:                 mode = d->mode;
   38:                 nlink = d->nlink;
   39:                 lev = d->lev;
   40:         }
   41:
   42:         i->i_mode = mode;
   43:         i->i_uid = 0;
   44:         i->i_size = 0;
   45:         i->i_atime = CURRENT_TIME;
   46:         i->i_ctime = CURRENT_TIME;
   47:         i->i_mtime = CURRENT_TIME;
   48:         i->i_gid = 0;
   49:         i->i_nlink = nlink;
   50:         i->i_blocks = 0;
   51:         i->i_flags = 0;
   52:         i->locked = 1;
   53:         i->dirty = 0;
   54:         i->mount_point = NULL;
   55:         i->count = 1;
   56:         i->u.procfs.i_lev = lev;
   57:         switch(i->i_mode & S_IFMT) {
   58:                 case S_IFDIR:
   59:                         i->fsop = &procfs_dir_fsop;
   60:                         break;
   61:                 case S_IFREG:
   62:                         i->fsop = &procfs_file_fsop;
   63:                         break;
   64:                 case S_IFLNK:
   65:                         i->fsop = &procfs_symlink_fsop;
   66:                         break;
```

fs/procfs/inode.c


```
   67:                 default:
   68:                         PANIC("invalid inode (%d) mode %08o.\n", i->inode, i->i_
mode);
   69:         }
   70:         return 0;
   71: }
   72:
   73: int procfs_bmap(struct inode *i, __off_t offset, int mode)
   74: {
   75:         return i->u.procfs.i_lev;
   76: }
   77:
   78: void procfs_statfs(struct superblock *sb, struct statfs *statfsbuf)
   79: {
   80:         statfsbuf->f_type = PROC_SUPER_MAGIC;
   81:         statfsbuf->f_bsize = sb->s_blocksize;
   82:         statfsbuf->f_blocks = 0;
   83:         statfsbuf->f_bfree = 0;
   84:         statfsbuf->f_bavail = 0;
   85:         statfsbuf->f_files = 0;
   86:         statfsbuf->f_ffree = 0;
   87:         /* statfsbuf->f_fsid = ? */
   88:         statfsbuf->f_namelen = NAME_MAX;
   89: }
```

fs/procfs/Makefile


```
 1: # fiwix/fs/procfs/Makefile
 2: #
 3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
 4: # Distributed under the terms of the Fiwix License.
 5: #
 6:
 7: .S.o:
 8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
 9: .c.o:
10:         $(CC) $(CFLAGS) -c -o $@ $<
11:
12: OBJS = super.o inode.o namei.o dir.o file.o symlink.o tree.o data.o
13:
14: procfs: $(OBJS)
15:         $(LD) $(LDFLAGS) -r $(OBJS) -o procfs.o
16:
17: clean:
18:         rm -f *.o
19:
```

fs/procfs/namei.c


```
    1: /*
    2:  * fiwix/fs/procfs/namei.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/fs.h>
   10: #include <fiwix/filesystems.h>
   11: #include <fiwix/fs_proc.h>
   12: #include <fiwix/process.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/errno.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: int procfs_lookup(const char *name, struct inode *dir, struct inode **i_res)
   20: {
   21:         int len, lev;
   22:         __ino_t inode;
   23:         __pid_t pid;
   24:         struct proc *p;
   25:         struct procfs_dir_entry *pdirent;
   26:
   27:         pid = inode = 0;
   28:         len = strlen(name);
   29:         if((dir->inode & 0xF0000000) == PROC_PID_INO) {
   30:                 pid = (dir->inode >> 12) & 0xFFFF;
   31:         }
   32:         dir->count++;
   33:
   34:         lev = bmap(dir, 0, FOR_READING);
   35:         pdirent = procfs_array[lev];
   36:         while(pdirent->inode && !inode) {
   37:                 if(len == pdirent->name_len) {
   38:                         if(!(strcmp(pdirent->name, name))) {
   39:                                 inode = pdirent->inode;
   40:                                 if(pid) {
   41:                                         inode = (PROC_PID_INO + (pid << 12)) + (
inode & 0xFFF);
   42:                                         if(strcmp(".", name) == 0) {
   43:                                                 inode = dir->inode;
   44:                                         }
   45:                                         if(strcmp("..", name) == 0) {
   46:                                                 inode = pdirent->inode;
   47:                                         }
   48:                                 }
   49:                         }
   50:                 }
   51:                 if(inode) {
   52:                         /*
   53:                          * This prevents a deadlock in iget() when
   54:                          * trying to lock '.' when 'dir' is the same
   55:                          * directory (ls -lai <dir>).
   56:                          */
   57:                         if(inode == dir->inode) {
   58:                                 *i_res = dir;
   59:                                 return 0;
   60:                         }
   61:
   62:                         if(!(*i_res = iget(dir->sb, inode))) {
   63:                                 return -EACCES;
   64:                         }
   65:                         iput(dir);
   66:                         return 0;
```

fs/procfs/namei.c


```
67:                     }
68:                     pdirent++;
69:             }
70:
71:             FOR_EACH_PROCESS(p) {
72:                     if(p->state) {
73:                             if(len == strlen(p->pidstr)) {
74:                                     if(!(strcmp(p->pidstr, name))) {
75:                                             inode = PROC_PID_INO + (p->pid << 12);
76:                                     }
77:                             }
78:                             if(inode) {
79:                                     if(!(*i_res = iget(dir->sb, inode))) {
80:                                             return -EACCES;
81:                                     }
82:                                     iput(dir);
83:                                     return 0;
84:                             }
85:                     }
86:             }
87:             iput(dir);
88:             return -ENOENT;
89: }
```

fs/procfs/super.c


```
    1: /*
    2:  * fiwix/fs/procfs/super.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/errno.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/filesystems.h>
   12: #include <fiwix/fs_proc.h>
   13: #include <fiwix/stat.h>
   14: #include <fiwix/mm.h>
   15: #include <fiwix/sched.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: struct fs_operations procfs_fsop = {
   20:         0,
   21:         PROC_DEV,
   22:
   23:         NULL,                   /* open */
   24:         NULL,                   /* close */
   25:         NULL,                   /* read */
   26:         NULL,                   /* write */
   27:         NULL,                   /* ioctl */
   28:         NULL,                   /* lseek */
   29:         NULL,                   /* readdir */
   30:         NULL,                   /* mmap */
   31:         NULL,                   /* select */
   32:
   33:         NULL,                   /* readlink */
   34:         NULL,                   /* followlink */
   35:         NULL,                   /* bmap */
   36:         NULL,                   /* lookup */
   37:         NULL,                   /* rmdir */
   38:         NULL,                   /* link */
   39:         NULL,                   /* unlink */
   40:         NULL,                   /* symlink */
   41:         NULL,                   /* mkdir */
   42:         NULL,                   /* mknod */
   43:         NULL,                   /* truncate */
   44:         NULL,                   /* create */
   45:         NULL,                   /* rename */
   46:
   47:         NULL,                   /* read_block */
   48:         NULL,                   /* write_block */
   49:
   50:         procfs_read_inode,
   51:         NULL,                   /* write_inode */
   52:         NULL,                   /* ialloc */
   53:         NULL,                   /* ifree */
   54:         procfs_statfs,
   55:         procfs_read_superblock,
   56:         NULL,                   /* remount_fs */
   57:         NULL,                   /* write_superblock */
   58:         NULL                    /* release_superblock */
   59: };
   60:
   61: int procfs_read_superblock(__dev_t dev, struct superblock *sb)
   62: {
   63:         superblock_lock(sb);
   64:         sb->dev = dev;
   65:         sb->fsop = &procfs_fsop;
   66:         sb->s_blocksize = PAGE_SIZE;
   67:
```

fs/procfs/super.c


```
   68:         if(!(sb->root = iget(sb, PROC_ROOT_INO))) {
   69:                 printk("WARNING: %s(): unable to get root inode.\n", __FUNCTION_
_);
   70:                 superblock_unlock(sb);
   71:                 return -EINVAL;
   72:         }
   73:         sb->root->u.procfs.i_lev = 0;
   74:
   75:         superblock_unlock(sb);
   76:         return 0;
   77: }
   78:
   79: int procfs_init(void)
   80: {
   81:         return register_filesystem("proc", &procfs_fsop);
   82: }
```

fs/procfs/symlink.c


```
/*
 * fiwix/fs/procfs/symlink.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/types.h>
#include <fiwix/errno.h>
#include <fiwix/buffer.h>
#include <fiwix/fs.h>
#include <fiwix/filesystems.h>
#include <fiwix/fs_proc.h>
#include <fiwix/stat.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>

struct fs_operations procfs_symlink_fsop = {
        0,
        0,

        NULL,                   /* open */
        NULL,                   /* close */
        NULL,                   /* read */
        NULL,                   /* write */
        NULL,                   /* ioctl */
        NULL,                   /* lseek */
        NULL,                   /* readdir */
        NULL,                   /* mmap */
        NULL,                   /* select */

        procfs_readlink,
        procfs_followlink,
        NULL,                   /* bmap */
        NULL,                   /* lookup */
        NULL,                   /* rmdir */
        NULL,                   /* link */
        NULL,                   /* unlink */
        NULL,                   /* symlink */
        NULL,                   /* mkdir */
        NULL,                   /* mknod */
        NULL,                   /* truncate */
        NULL,                   /* create */
        NULL,                   /* rename */

        NULL,                   /* read_block */
        NULL,                   /* write_block */

        NULL,                   /* read_inode */
        NULL,                   /* write_inode */
        NULL,                   /* ialloc */
        NULL,                   /* ifree */
        NULL,                   /* statfs */
        NULL,                   /* read_superblock */
        NULL,                   /* remount_fs */
        NULL,                   /* write_superblock */
        NULL                    /* release_superblock */
};

int procfs_readlink(struct inode *i, char *buffer, __size_t count)
{
        __off_t size_read;
        struct procfs_dir_entry *d;

        if(!(d = get_procfs_by_inode(i))) {
                return -EINVAL;
        }
```

fs/procfs/symlink.c


```
   68:
   69:         if(!d->data_fn) {
   70:                 return -EINVAL;
   71:         }
   72:
   73:         size_read = d->data_fn(buffer, (i->inode >> 12) & 0xFFFF);
   74:         return size_read;
   75: }
   76:
   77: int procfs_followlink(struct inode *dir, struct inode *i, struct inode **i_res)
   78: {
   79:         __ino_t errno;
   80:         __pid_t pid;
   81:         struct proc *p;
   82:
   83:         if(!i) {
   84:                 return -ENOENT;
   85:         }
   86:         if(!(S_ISLNK(i->i_mode))) {
   87:                 printk("%s(): Oops, inode '%d' is not a symlink (!?).\n", __FUNC
TION__, i->inode);
   88:                 return 0;
   89:         }
   90:
   91:         p = NULL;
   92:         if((pid = (i->inode >> 12) & 0xFFFF)) {
   93:                 if(!(p = get_proc_by_pid(pid))) {
   94:                         return -ENOENT;
   95:                 }
   96:         }
   97:
   98:         /* FIXME!
   99:         if(p && p->root) {
  100:                 printk("(pid %d) p->root->inode = %d (count = %d)\n", p->pid, p-
>root->inode, p->root->count);
  101:         }
  102:         */
  103:
  104:         switch(i->inode & 0xF0000FFF) {
  105:                 case PROC_PID_CWD:
  106:                         if(!p->pwd) {
  107:                                 return -ENOENT;
  108:                         }
  109:                         *i_res = p->pwd;
  110:                         p->pwd->count++;
  111:                         iput(i);
  112:                         break;
  113:                 case PROC_PID_EXE:
  114:                         if(!p->vma || !p->vma->inode) {
  115:                                 return -ENOENT;
  116:                         }
  117:                         *i_res = p->vma->inode;
  118:                         p->vma->inode->count++;
  119:                         iput(i);
  120:                         break;
  121:                 case PROC_PID_ROOT:
  122:                         if(!p->root) {
  123:                                 return -ENOENT;
  124:                         }
  125:                         *i_res = p->root;
  126:                         p->root->count++;
  127:                         iput(i);
  128:                         break;
  129:                 default:
  130:                         iput(i);
  131:                         if((errno = parse_namei(current->pidstr, dir, i_res, NUL
L, FOLLOW_LINKS))) {
```

fs/procfs/symlink.c


```
132:                                    return errno;
133:                            }
134:            }
135:            return 0;
136: }
```

fs/procfs/tree.c


```
    1: /*
    2:  * fiwix/fs/procfs/tree.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/types.h>
    9: #include <fiwix/stat.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/fs_proc.h>
   12: #include <fiwix/errno.h>
   13: #include <fiwix/stdio.h>
   14: #include <fiwix/string.h>
   15:
   16: #define DIR     S_IFDIR | S_IRUSR | S_IXUSR | S_IRGRP | S_IXGRP | S_IROTH | \
   17:                 S_IXOTH                                  /* dr-xr-xr-x */
   18: #define REG     S_IFREG | S_IRUSR | S_IRGRP | S_IROTH   /* -r--r--r-- */
   19: #define REGUSR  S_IFREG | S_IRUSR                       /* -r-------- */
   20: #define LNK     S_IFLNK | S_IRWXU | S_IRWXG | S_IRWXO   /* lrwxrwxrwx */
   21: #define LNKPID  S_IFLNK | S_IRWXU                       /* lrwx------ */
   22:
   23: /*
   24:  * WARNING: every time a new entry is added to this array you must also change
   25:  * the PROC_ARRAY_ENTRIES value defined in fs_proc.h.
   26:  */
   27: struct procfs_dir_entry procfs_array[][PROC_ARRAY_ENTRIES + 1] = {
   28:    {    /* [0] / */
   29:         { 1,     DIR,  2, 0, 1,  ".",           NULL },
   30:         { 2,     DIR,  2, 0, 2,  "..",          NULL },
   31:         { 3,     DIR,  3, 3, 3,  "sys",         NULL },
   32:         { 4,     REG,  1, 0, 7,  "cmdline",     data_proc_cmdline },
   33:         { 5,     REG,  1, 0, 7,  "cpuinfo",     data_proc_cpuinfo },
   34:         { 6,     REG,  1, 0, 7,  "devices",     data_proc_devices },
   35:         { 7,     REG,  1, 0, 3,  "dma",         data_proc_dma },
   36:         { 8,     REG,  1, 0, 11, "filesystems", data_proc_filesystems },
   37:         { 9,     REG,  1, 0, 10, "interrupts",  data_proc_interrupts },
   38:         { 10,    REG,  1, 0, 7,  "loadavg",     data_proc_loadavg },
   39:         { 11,    REG,  1, 0, 5,  "locks",       data_proc_locks },
   40:         { 12,    REG,  1, 0, 7,  "meminfo",     data_proc_meminfo },
   41:         { 13,    REG,  1, 0, 6,  "mounts",      data_proc_mounts },
   42:         { 14,    REG,  1, 0, 10, "partitions",  data_proc_partitions },
   43:         { 15,    REG,  1, 0, 3,  "rtc",         data_proc_rtc },
   44:         { 16,    LNK,  1, 0, 4,  "self",        data_proc_self },
   45:         { 17,    REG,  1, 0, 4,  "stat",        data_proc_stat },
   46:         { 18,    REG,  1, 0, 6,  "uptime",      data_proc_uptime },
   47:         { 19,    REG,  1, 0, 7,  "version",     data_proc_fullversion },
   48:         { 0, 0, 0, 0, 0, NULL, NULL }
   49:    },
   50:    {    /* [1] /PID/ */
   51:         { 1000,  DIR,  2, 1, 1,  ".",           NULL },
   52:         { 1,     DIR,  2, 0, 2,  "..",          NULL },
   53: /*      { PROC_PID_FD,      DIR,    2, 2, 2,  "fd",       data_proc_pid_fd },*/
   54:         { PROC_PID_CMDLINE, REG,    1, 1, 7,  "cmdline",  data_proc_pid_cmdline
},
   55:         { PROC_PID_CWD,     LNKPID, 1, 1, 3,  "cwd",      data_proc_pid_cwd },
   56:         { PROC_PID_ENVIRON, REGUSR, 1, 1, 7,  "environ",  data_proc_pid_environ
},
   57:         { PROC_PID_EXE,     LNKPID, 1, 1, 3,  "exe",      data_proc_pid_exe },
   58:         { PROC_PID_MAPS,    REG,    1, 1, 4,  "maps",     data_proc_pid_maps },
   59:         { PROC_PID_MOUNTINFO,REG,   1, 1, 9,  "mountinfo",data_proc_pid_mountinf
o },
   60:         { PROC_PID_ROOT,    LNKPID, 1, 1, 4,  "root",     data_proc_pid_root },
   61:         { PROC_PID_STAT,    REG,    1, 1, 4,  "stat",     data_proc_pid_stat },
   62:         { PROC_PID_STATM,   REG,    1, 1, 5,  "statm",    data_proc_pid_statm },
   63:         { PROC_PID_STATUS,  REG,    1, 1, 6,  "status",   data_proc_pid_status }
,
```

fs/procfs/tree.c


```
64:         { 0, 0, 0, 0, 0, NULL, NULL }
65:     },
66:
67:     {
68:     },
69:
70:     {    /* [3] /sys/ */
71:         { 3,     DIR,  2, 3, 1,  ".",           NULL },
72:         { 1,     DIR,  2, 0, 2,  "..",          NULL },
73:         { 2001,  DIR,  2, 4, 6,  "kernel",      NULL },
74:         { 0, 0, 0, 0, 0, NULL, NULL }
75:     },
76:     {    /* [4] /sys/kernel/ */
77:         { 2001,  DIR,  2, 4, 1,  ".",           NULL },
78:         { 3,     DIR,  2, 3, 2,  "..",          NULL },
79:         { 3001,  REG,  1, 4, 10, "domainname",  data_proc_domainname },
80:         { 3002,  REG,  1, 4, 8,  "file-max",    data_proc_filemax },
81:         { 3003,  REG,  1, 4, 7,  "file-nr",     data_proc_filenr },
82:         { 3004,  REG,  1, 4, 8,  "hostname",    data_proc_hostname },
83:         { 3005,  REG,  1, 4, 9,  "inode-max",   data_proc_inodemax },
84:         { 3006,  REG,  1, 4, 8,  "inode-nr",    data_proc_inodenr },
85:         { 3007,  REG,  1, 4, 9,  "osrelease",   data_proc_osrelease },
86:         { 3008,  REG,  1, 4, 6,  "ostype",      data_proc_ostype },
87:         { 3009,  REG,  1, 4, 7,  "version",     data_proc_version },
88:         { 0, 0, 0, 0, 0, NULL, NULL }
89:     }
90: };
91:
92: struct procfs_dir_entry * get_procfs_by_inode(struct inode *i)
93: {
94:         __ino_t inode;
95:         int n, lev;
96:         struct procfs_dir_entry *d;
97:
98:         inode = i->inode;
99:         for(lev = 0; procfs_array[lev]; lev++) {
100:                 if(lev == PROC_PID_LEV) {       /* PID entries */
101:                         if((i->inode & 0xF0000000) == PROC_PID_INO) {
102:                                 inode = i->inode & 0xF0000FFF;
103:                         }
104:                 }
105:                 d = procfs_array[lev];
106:                 for(n = 0; n < PROC_ARRAY_ENTRIES && d->inode; n++) {
107:                         if(d->inode == inode) {
108:                                 return d;
109:                         }
110:                         d++;
111:                 }
112:         }
113:
114:         return NULL;
115: }
```

mm/alloc.c


```
 1: /*
 2:  * fiwix/mm/alloc.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/mm.h>
 9: #include <fiwix/stdio.h>
10: #include <fiwix/string.h>
11:
12: /*
13:  * The implementation of kernel memory allocation is extremely simple, it works
14:  * with a granularity of PAGE_SIZE (4096 bytes). There is indeed a lot of room
15:  * for improvements here.
16:  */
17: unsigned int kmalloc(void)
18: {
19:         struct page *pg;
20:         unsigned int addr;
21:
22:         if((pg = get_free_page())) {
23:                 addr = pg->page << PAGE_SHIFT;
24:                 return P2V(addr);
25:         }
26:
27:         /* out of memory! */
28:         return 0;
29: }
30:
31: void kfree(unsigned int addr)
32: {
33:         addr = V2P(addr);
34:         release_page(addr >> PAGE_SHIFT);
35: }
```

mm/bios_map.c


```
    1: /*
    2:  * fiwix/mm/bios_map.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/mm.h>
   10: #include <fiwix/bios.h>
   11: #include <fiwix/stdio.h>
   12: #include <fiwix/string.h>
   13:
   14: /* check if an specific address is available in the BIOS memory map */
   15: int addr_in_bios_map(unsigned int addr)
   16: {
   17:         int n;
   18:         struct bios_mem_map *bmm;
   19:
   20:         bmm = &bios_mem_map[0];
   21:         for(n = 0; n < NR_BIOS_MM_ENT; n++, bmm++) {
   22:                 if(bmm->to && bmm->type == BIOS_MEM_AVAIL) {
   23:                         if(addr >= bmm->from && addr < (bmm->to & PAGE_MASK)) {
   24:                                 return 1;
   25:                         }
   26:                 }
   27:         }
   28:         return 0;       /* not in BIOS map or not available (reserved, ...) */
   29: }
   30:
   31: void bios_map_init(memory_map_t *bmmap_addr, unsigned long int bmmap_length)
   32: {
   33:         unsigned int n;
   34:         unsigned int mem_from, mem_to, len_low, len_high;
   35:         memory_map_t *bmmap;
   36:         char *bios_mem_type[] = { NULL, "available" , "reserved",
   37:                                   "ACPI Reclaim", "ACPI NVS", "unusable",
   38:                                   "disabled" };
   39:
   40:         bmmap = bmmap_addr;
   41:         if(bmmap) {
   42:                 n = 0;
   43:
   44:                 while((unsigned int)bmmap < (unsigned int)bmmap_addr + bmmap_len
gth) {
   45:                         mem_from = (unsigned int)bmmap->base_addr_low;
   46:                         len_low = (unsigned int)bmmap->length_low;
   47:                         if((((mem_from >> 16) & 0xFFFF) + ((len_low >> 16) & 0xF
FFF)) > 0xFFFF) {
   48:                                 mem_to = ~0;
   49:                                 len_high = (mem_from >> 16) + (len_low >> 16);
   50:                                 len_high >>= 16;
   51:                         } else {
   52:                                 mem_to = mem_from + len_low;
   53:                                 len_high = 0;
   54:                         }
   55:                         printk("%s    0x%08X%08X-0x%08X%08X %s\n",
   56:                                 n ? "      " : "memory",
   57:                                 bmmap->base_addr_high, bmmap->base_addr_low,
   58:                                 len_high, mem_from + len_low,
   59:                                 bios_mem_type[(int)bmmap->type]);
   60:                         /* only memory addresses below 4GB are accepted */
   61:                         if(!bmmap->base_addr_high) {
   62:                                 if(n < NR_BIOS_MM_ENT && len_low) {
   63:                                         bios_mem_map[n].from = mem_from;
   64:                                         bios_mem_map[n].to = mem_to;
   65:                                         bios_mem_map[n].type = (int)bmmap->type;
```

mm/bios_map.c


```
   66:                                             n++;
   67:                                     }
   68:                             }
   69:                             bmmap = (memory_map_t *)((unsigned int)bmmap + bmmap->si
ze + sizeof(bmmap->size));
   70:                     }
   71:             } else {
   72:                     printk("WARNING: your BIOS has not provided a memory map.\n");
   73:                     bios_mem_map[0].from = 0;
   74:                     bios_mem_map[0].to = _memsize * 1024;
   75:                     bios_mem_map[0].type = BIOS_MEM_AVAIL;
   76:                     bios_mem_map[1].from = 0x00100000;
   77:                     bios_mem_map[1].to = (_extmemsize + 1024) * 1024;
   78:                     bios_mem_map[1].type = BIOS_MEM_AVAIL;
   79:             }
   80:             kstat.physical_pages = (_extmemsize + 1024) >> 2;
   81:
   82:             /*
   83:              * This truncates to 1GB since it's the maximum physical memory
   84:              * currently supported.
   85:              */
   86:             if(kstat.physical_pages > (0x40000000 >> PAGE_SHIFT)) {
   87:                     kstat.physical_pages = (0x40000000 >> PAGE_SHIFT);
   88:                     printk("WARNING: only up to 1GB of physical memory will be used.
\n");
   89:             }
   90: }
```

mm/fault.c


```
    1: /*
    2:  * fiwix/mm/fault.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/sigcontext.h>
   10: #include <fiwix/asm.h>
   11: #include <fiwix/mm.h>
   12: #include <fiwix/process.h>
   13: #include <fiwix/traps.h>
   14: #include <fiwix/sched.h>
   15: #include <fiwix/fs.h>
   16: #include <fiwix/mman.h>
   17: #include <fiwix/errno.h>
   18: #include <fiwix/stdio.h>
   19: #include <fiwix/string.h>
   20:
   21: /* send the SIGSEGV signal to the ofending process */
   22: static void send_sigsegv(struct sigcontext *sc)
   23: {
   24:         dump_registers(14, sc);
   25:         printk("Memory map:\n");
   26:         show_vma_regions(current);
   27:         send_sig(current, SIGSEGV);
   28: }
   29:
   30: static int page_protection_violation(struct vma *vma, unsigned int cr2, struct s
igcontext *sc)
   31: {
   32:         unsigned int *pgdir;
   33:         unsigned int *pgtbl;
   34:         unsigned int page, newpage;
   35:         unsigned int pde, pte;
   36:         struct page *pg;
   37:
   38:         pde = GET_PGDIR(cr2);
   39:         pte = GET_PGTBL(cr2);
   40:         pgdir = (unsigned int *)P2V(current->tss.cr3);
   41:         pgtbl = (unsigned int *)P2V((pgdir[pde] & PAGE_MASK));
   42:         page = (pgtbl[pte] & PAGE_MASK) >> PAGE_SHIFT;
   43:
   44:         pg = &page_table[page];
   45:
   46:         /* Copy On Write feature */
   47:         if(pg->count > 1) {
   48:                 /* a page not marked as COW means it's read-only */
   49:                 if(!(pg->flags & PAGE_COW)) {
   50:                         printk("Oops!, page %d NOT marked for COW.\n", pg->page)
;
   51:                         send_sigsegv(sc);
   52:                         return 0;
   53:                 }
   54:                 if(!(newpage = kmalloc())) {
   55:                         printk("%s(): not enough memory!\n", __FUNCTION__);
   56:                         return 1;
   57:                 }
   58:                 current->rss++;
   59:                 memcpy_b((void *)newpage, (void *)P2V((page << PAGE_SHIFT)), PAG
E_SIZE);
   60:                 pgtbl[pte] = V2P(newpage) | PAGE_PRESENT | PAGE_RW | PAGE_USER;
   61:                 kfree(P2V((page << PAGE_SHIFT)));
   62:                 current->rss--;
   63:                 invalidate_tlb();
   64:                 return 0;
```

mm/fault.c


```
   65:         } else {
   66:                 /* last page of Copy On Write procedure */
   67:                 if(pg->count == 1) {
   68:                         /* a page not marked as COW means it's read-only */
   69:                         if(!(pg->flags & PAGE_COW)) {
   70:                                 printk("Oops!, last page %d NOT marked for COW.\
n", pg->page);
   71:                                 send_sigsegv(sc);
   72:                                 return 0;
   73:                         }
   74:                         pgtbl[pte] = (page << PAGE_SHIFT) | PAGE_PRESENT | PAGE_
RW | PAGE_USER;
   75:                         invalidate_tlb();
   76:                         return 0;
   77:                 }
   78:         }
   79:         printk("WARNING: %s(): page %d with pg->count = 0!\n", __FUNCTION__, pg-
>page);
   80:         return 1;
   81: }
   82:
   83: static int page_not_present(struct vma *vma, unsigned int cr2, struct sigcontext
 *sc)
   84: {
   85:         unsigned int page, file_offset;
   86:         struct page *pg;
   87:
   88:         if(!vma) {
   89:                 if(cr2 >= (sc->oldesp - 32)) {
   90:                         if(!(vma = find_vma_region(KERNEL_BASE_ADDR - 1))) {
   91:                                 printk("WARNING: %s(): process %d doesn't have a
n stack region in vma!\n", __FUNCTION__, current->pid);
   92:                                 send_sigsegv(sc);
   93:                                 return 0;
   94:                         } else {
   95:                                 /* assuming stack will never reach heap */
   96:                                 vma->start = cr2;
   97:                                 vma->start = vma->start & PAGE_MASK;
   98:                         }
   99:                 }
  100:         }
  101:
  102:         /* if still a non-valid vma is found then kill the process! */
  103:         if(!vma || vma->prot == PROT_NONE) {
  104:                 send_sigsegv(sc);
  105:                 return 0;
  106:         }
  107:
  108:         /* fill the page with its corresponding file content */
  109:         if(vma->inode) {
  110:                 file_offset = (cr2 & PAGE_MASK) - vma->start + vma->offset;
  111:                 file_offset &= PAGE_MASK;
  112:                 pg = NULL;
  113:
  114:                 if(!(vma->prot & PROT_WRITE) || vma->flags & MAP_SHARED) {
  115:                         /* check if it's already in cache */
  116:                         if((pg = search_page_hash(vma->inode, file_offset))) {
  117:                                 if(!map_page(current, cr2, (unsigned int)V2P(pg-
>data), vma->prot)) {
  118:                                         printk("%s(): Oops, map_page() returned
0!\n", __FUNCTION__);
  119:                                         return 1;
  120:                                 }
  121:                                 page = (unsigned int)pg->data;
  122:                         }
  123:                 }
  124:                 if(!pg) {
```

mm/fault.c


```
  125:                          if(!(page = map_page(current, cr2, 0, vma->prot))) {
  126:                                  printk("%s(): Oops, map_page() returned 0!\n", _
_FUNCTION__);
  127:                                  return 1;
  128:                          }
  129:                          pg = &page_table[V2P(page) >> PAGE_SHIFT];
  130:                          if(bread_page(pg, vma->inode, file_offset, vma->prot, vm
a->flags)) {
  131:                                  unmap_page(page);
  132:                                  return 1;
  133:                          }
  134:                          current->usage.ru_majflt++;
  135:                  }
  136:          } else {
  137:                  current->usage.ru_minflt++;
  138:                  page = 0;
  139:          }
  140:
  141:          if(vma->flags & ZERO_PAGE) {
  142:                  if(!page) {
  143:                          if(!(page = map_page(current, cr2, 0, vma->prot))) {
  144:                                  printk("%s(): Oops, map_page() returned 0!\n", _
_FUNCTION__);
  145:                                  return 1;
  146:                          }
  147:                  }
  148:                  memset_b((void *)(page & PAGE_MASK), NULL, PAGE_SIZE);
  149:          }
  150:
  151:          return 0;
  152: }
  153:
  154: /*
  155:  * Exception 0xE: Page Fault
  156:  *
  157:  *               +------+------+------+------+------+------+
  158:  *               | user |kernel|  PV  |  PF  | read |write |
  159:  * +-------------+------+------+------+------+------+------+
  160:  * |the page     | U1   |    K1| U1 K1|      | U1 K1|    K1|
  161:  * |has          | U2   |    K2| U2   |    K2|    K2| U2 K2|
  162:  * |a vma region | U3   |      |      | U3   | U3   | U3   |
  163:  * +-------------+------+------+------+------+------+------+
  164:  * |the page     | U1   |    K1| U1 K1|    K1| U1 K1| U1 K1|
  165:  * |doesn't have | U2   |      |      | U2   | U2   | U2   |
  166:  * |a vma region |      |      |      |      |      |      |
  167:  * +-------------+------+------+------+------+------+------+
  168:  *
  169:  * U1 - vma + user + PV + read
  170:  *      (vma page in user-mode, page-violation during read)
  171:  *      U1.1) if flags match                    -> Demand paging
  172:  *      U1.2) if flags don't match              -> SIGSEV
  173:  *
  174:  * U2 - vma + user + PV + write
  175:  *      (vma page in user-mode, page-violation during write)
  176:  *      U2.1) if flags match                    -> Copy-On-Write
  177:  *      U2.2) if flags don't match              -> SIGSEGV
  178:  *
  179:  * U3 - vma + user + PF + (read | write)        -> Demand paging
  180:  *      (vma page in user-mode, page-fault during read or write)
  181:  *
  182:  * K1 - vma + kernel + PV + (read | write)      -> PANIC
  183:  *      (vma page in kernel-mode, page-violation during read or write)
  184:  * K2 - vma + kernel + PF + (read | write)      -> Demand paging (mmap)
  185:  *      (vma page in kernel-mode, page-fault during read or write)
  186:  *
  187:  * ------------------------------------------------------------------------
  188:  *
```

mm/fault.c

```
  189:  * U1 - !vma + user + PV + (read | write)       -> SIGSEGV
  190:  *      (!vma page in user-mode, page-violation during read or write)
  191:  * U2 - !vma + user + PF + (read | write)       -> STACK grows
  192:  *      (!vma page in user-mode, page-fault during read or write)
  193:  *
  194:  * K1 - !vma + kernel + (PV | PF) + (read | write)      -> PANIC
  195:  *      (!vma page in kernel-mode, page-fault or page-violation during read
  196:  *      or write)
  197:  */
  198: void do_page_fault(unsigned int trap, struct sigcontext *sc)
  199: {
  200:         unsigned int cr2;
  201:         struct vma *vma;
  202:
  203:         GET_CR2(cr2);
  204:         if((vma = find_vma_region(cr2))) {
  205:
  206:                 /* in user mode */
  207:                 if(sc->err & PFAULT_U) {
  208:                         if(sc->err & PFAULT_V) {        /* violation */
  209:                                 if(sc->err & PFAULT_W) {
  210:                                         if((page_protection_violation(vma, cr2,
sc))) {
  211:                                                 send_sig(current, SIGKILL);
  212:                                         }
  213:                                         return;
  214:                                 }
  215:                                 send_sigsegv(sc);
  216:                         } else {                        /* page not present */
  217:                                 if((page_not_present(vma, cr2, sc))) {
  218:                                         send_sig(current, SIGKILL);
  219:                                 }
  220:                         }
  221:                         return;
  222:
  223:                 /* in kernel mode */
  224:                 } else {
  225:                         /*
  226:                          * WP bit marks the order: first check if the page is
  227:                          * present, then check for protection violation.
  228:                          */
  229:                         if(!(sc->err & PFAULT_V)) {     /* page not present */
  230:                                 if((page_not_present(vma, cr2, sc))) {
  231:                                         send_sig(current, SIGKILL);
  232:                                         printk("%s(): kernel was unable to read
a page of process '%s' (pid %d).\n", __FUNCTION__, current->argv0, current->pid);
  233:                                 }
  234:                                 return;
  235:                         }
  236:                         if(sc->err & PFAULT_W) {        /* copy-on-write? */
  237:                                 if((page_protection_violation(vma, cr2, sc))) {
  238:                                         send_sig(current, SIGKILL);
  239:                                         printk("%s(): kernel was unable to write
 a page of process '%s' (pid %d).\n", __FUNCTION__, current->argv0, current->pid);
  240:                                 }
  241:                                 return;
  242:                         }
  243:                 }
  244:         } else {
  245:                 /* in user mode */
  246:                 if(sc->err & PFAULT_U) {
  247:                         if(sc->err & PFAULT_V) {        /* violation */
  248:                                 send_sigsegv(sc);
  249:                         } else {                        /* stack? */
  250:                                 if((page_not_present(vma, cr2, sc))) {
  251:                                         send_sig(current, SIGKILL);
  252:                                 }
```

mm/fault.c


```
253:                            }
254:                            return;
255:                    }
256:            }
257:
258:            dump_registers(trap, sc);
259:            show_vma_regions(current);
260:            PANIC("\n");
261: }
```

mm/Makefile


```
 1: # fiwix/mm/Makefile
 2: #
 3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
 4: # Distributed under the terms of the Fiwix License.
 5: #
 6:
 7: .S.o:
 8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
 9: .c.o:
10:         $(CC) $(CFLAGS) -c -o $@ $<
11:
12: OBJS = bios_map.o memory.o page.o alloc.o fault.o mmap.o swapper.o
13:
14: mm:     $(OBJS)
15:         $(LD) $(LDFLAGS) -r $(OBJS) -o mm.o
16:
17: clean:
18:         rm -f *.o
19:
```

mm/memory.c


```
    1: /*
    2:  * fiwix/mm/memory.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/asm.h>
   10: #include <fiwix/mm.h>
   11: #include <fiwix/mman.h>
   12: #include <fiwix/bios.h>
   13: #include <fiwix/ramdisk.h>
   14: #include <fiwix/process.h>
   15: #include <fiwix/buffer.h>
   16: #include <fiwix/fs.h>
   17: #include <fiwix/filesystems.h>
   18: #include <fiwix/stdio.h>
   19: #include <fiwix/string.h>
   20:
   21: #define KERNEL_TEXT_SIZE        ((int)_etext - (KERNEL_BASE_ADDR + KERNEL_ENTRY_
ADDR))
   22: #define KERNEL_DATA_SIZE        ((int)_edata - (int)_etext)
   23: #define KERNEL_BSS_SIZE         ((int)_end - (int)_edata)
   24:
   25: #define PGDIR_4MB_ADDR          0x90000
   26:
   27: unsigned int *kpage_dir;
   28: unsigned int *kpage_table;
   29:
   30: unsigned int _last_data_addr;
   31:
   32: unsigned int proc_table_size = 0;
   33: struct proc *proc_table;
   34:
   35: unsigned int buffer_table_size = 0;
   36: unsigned int buffer_hash_table_size = 0;
   37: struct buffer *buffer_table;
   38: struct buffer **buffer_hash_table;
   39:
   40: unsigned int inode_table_size = 0;
   41: unsigned int inode_hash_table_size = 0;
   42: struct inode *inode_table;
   43: struct inode **inode_hash_table;
   44:
   45: unsigned int fd_table_size = 0;
   46: struct fd *fd_table;
   47:
   48: unsigned int mount_table_size = 0;
   49: struct mount *mount_table;
   50:
   51: struct ramdisk ramdisk_table[RAMDISK_MINORS];
   52:
   53: unsigned int page_table_size = 0;
   54: unsigned int page_hash_table_size = 0;
   55: struct page *page_table;
   56: struct page **page_hash_table;
   57:
   58: static void map_kaddr(unsigned int from, unsigned int to, int flags)
   59: {
   60:         unsigned int n;
   61:         unsigned int *pgtbl;
   62:         unsigned int pde, pte;
   63:
   64:         for(n = from >> PAGE_SHIFT; n < (to >> PAGE_SHIFT); n++) {
   65:                 pde = GET_PGDIR(n << PAGE_SHIFT);
   66:                 pte = GET_PGTBL(n << PAGE_SHIFT);
```

mm/memory.c


```
   67:                 if(!(kpage_dir[pde] & ~PAGE_MASK)) {
   68:                         unsigned int addr;
   69:                         addr = _last_data_addr;
   70:                         _last_data_addr += PAGE_SIZE;
   71:                         kpage_dir[pde] = addr | flags;
   72:                         memset_b((void *)addr, NULL, PAGE_SIZE);
   73:                 }
   74:                 pgtbl = (unsigned int *)(kpage_dir[pde] & PAGE_MASK);
   75:                 pgtbl[pte] = (kpage_table[(pde * 1024) + pte] & PAGE_MASK) | fla
gs;
   76:         }
   77: }
   78:
   79: void bss_init(void)
   80: {
   81:         memset_b((void *)((int)_edata), NULL, KERNEL_BSS_SIZE);
   82: }
   83:
   84: /*
   85:  * This function creates a minimal Page Directory covering only the first 4MB
   86:  * of physical memory. Just enough to boot the kernel.
   87:  * (it returns the address to be used by the CR3 register)
   88:  */
   89: unsigned int setup_minmem(void)
   90: {
   91:         int n;
   92:         unsigned int addr;
   93:         short int pd, mb4;
   94:
   95:         mb4 = 1;        /* 4MB units */
   96:         addr = KERNEL_BASE_ADDR + PGDIR_4MB_ADDR;
   97:
   98:         kpage_dir = (unsigned int *)addr;
   99:         memset_b(kpage_dir, NULL, PAGE_SIZE);
  100:
  101:         addr += PAGE_SIZE;
  102:         kpage_table = (unsigned int *)addr;
  103:         memset_b(kpage_table, NULL, PAGE_SIZE * mb4);
  104:
  105:         for(n = 0; n < (1024 * mb4); n++) {
  106:                 kpage_table[n] = (n << PAGE_SHIFT) | PAGE_PRESENT | PAGE_RW;
  107:                 if(!(n % 1024)) {
  108:                         pd = n / 1024;
  109:                         kpage_dir[pd] = (unsigned int)(addr + (PAGE_SIZE * pd) +
 0x40000000) | PAGE_PRESENT | PAGE_RW;
  110:                         kpage_dir[GET_PGDIR(KERNEL_BASE_ADDR) + pd] = (unsigned
int)(addr + (PAGE_SIZE * pd) + 0x40000000) | PAGE_PRESENT | PAGE_RW;
  111:                 }
  112:         }
  113:         return (unsigned int)kpage_dir + 0x40000000;
  114: }
  115:
  116: /* returns the mapped address of a virtual address */
  117: unsigned int get_mapped_addr(struct proc *p, unsigned int addr)
  118: {
  119:         unsigned int *pgdir, *pgtbl;
  120:         unsigned int pde, pte;
  121:
  122:         pgdir = (unsigned int *)P2V(p->tss.cr3);
  123:         pde = GET_PGDIR(addr);
  124:         pte = GET_PGTBL(addr);
  125:         pgtbl = (unsigned int *)P2V((pgdir[pde] & PAGE_MASK));
  126:         return pgtbl[pte];
  127: }
  128:
  129: int clone_pages(struct proc *child)
  130: {
```

mm/memory.c


```
  131:         unsigned int *src_pgdir, *dst_pgdir;
  132:         unsigned int *src_pgtbl, *dst_pgtbl;
  133:         unsigned int pde, pte;
  134:         unsigned int p_page, c_page;
  135:         unsigned int n, n2, pages;
  136:         struct page *pg;
  137:         struct vma *vma;
  138:
  139:         src_pgdir = (unsigned int *)P2V(current->tss.cr3);
  140:         dst_pgdir = (unsigned int *)P2V(child->tss.cr3);
  141:         vma = current->vma;
  142:
  143:         for(n = 0, pages = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  144:                 for(n2 = vma->start; n2 < vma->end; n2 += PAGE_SIZE) {
  145:                         pde = GET_PGDIR(n2);
  146:                         pte = GET_PGTBL(n2);
  147:                         if(src_pgdir[pde] & PAGE_PRESENT) {
  148:                                 src_pgtbl = (unsigned int *)P2V((src_pgdir[pde]
& PAGE_MASK));
  149:                                 if(!(dst_pgdir[pde] & PAGE_PRESENT)) {
  150:                                         if(!(c_page = kmalloc())) {
  151:                                                 printk("%s(): returning 0!\n", _
_FUNCTION__);
  152:                                                 return 0;
  153:                                         }
  154:                                         current->rss++;
  155:                                         pages++;
  156:                                         dst_pgdir[pde] = V2P(c_page) | PAGE_PRES
ENT | PAGE_RW | PAGE_USER;
  157:                                         memset_b((void *)c_page, NULL, PAGE_SIZE
);
  158:                                 }
  159:                                 dst_pgtbl = (unsigned int *)P2V((dst_pgdir[pde]
& PAGE_MASK));
  160:                                 if(src_pgtbl[pte] & PAGE_PRESENT) {
  161:                                         p_page = src_pgtbl[pte] >> PAGE_SHIFT;
  162:                                         pg = &page_table[p_page];
  163:                                         if(pg->flags & PAGE_RESERVED) {
  164:                                                 continue;
  165:                                         }
  166:                                         src_pgtbl[pte] &= ~PAGE_RW;
  167:                                         /* mark as COW only writable pages */
  168:                                         if(vma->prot & PROT_WRITE) {
  169:                                                 pg->flags |= PAGE_COW;
  170:                                         }
  171:                                         dst_pgtbl[pte] = src_pgtbl[pte];
  172:                                         if(!valid_page((dst_pgtbl[pte] & PAGE_MA
SK) >> PAGE_SHIFT)) {
  173:                                                 PANIC("%s: missing page %d durin
g copy-on-write process.\n", __FUNCTION__, (dst_pgtbl[pte] & PAGE_MASK) >> PAGE_SHIFT);
  174:                                         }
  175:                                         pg = &page_table[(dst_pgtbl[pte] & PAGE_
MASK) >> PAGE_SHIFT];
  176:                                         pg->count++;
  177:                                 }
  178:                         }
  179:                 }
  180:         }
  181:         return pages;
  182: }
  183:
  184: int free_page_tables(struct proc *p)
  185: {
  186:         unsigned int *pgdir;
  187:         unsigned int n, count;
  188:
  189:         pgdir = (unsigned int *)P2V(p->tss.cr3);
```

mm/memory.c


```
  190:         for(n = 0, count = 0; n < PD_ENTRIES; n++) {
  191:                 if((pgdir[n] & (PAGE_PRESENT | PAGE_RW | PAGE_USER)) == (PAGE_PR
ESENT | PAGE_RW | PAGE_USER)) {
  192:                         kfree(P2V(pgdir[n]) & PAGE_MASK);
  193:                         pgdir[n] = NULL;
  194:                         count++;
  195:                 }
  196:         }
  197:         return count;
  198: }
  199:
  200: unsigned int map_page(struct proc *p, unsigned int vaddr, unsigned int addr, uns
igned int prot)
  201: {
  202:         unsigned int *pgdir, *pgtbl;
  203:         unsigned int vpage;
  204:         int pde, pte;
  205:
  206:         pgdir = (unsigned int *)P2V(p->tss.cr3);
  207:         pde = GET_PGDIR(vaddr);
  208:         pte = GET_PGTBL(vaddr);
  209:
  210:         if(!(pgdir[pde] & PAGE_PRESENT)) {      /* allocating page table */
  211:                 if(!(vpage = kmalloc())) {
  212:                         return 0;
  213:                 }
  214:                 p->rss++;
  215:                 pgdir[pde] = V2P(vpage) | PAGE_PRESENT | PAGE_RW | PAGE_USER;
  216:                 memset_b((void *)vpage, NULL, PAGE_SIZE);
  217:         }
  218:         pgtbl = (unsigned int *)P2V((pgdir[pde] & PAGE_MASK));
  219:         if(!(pgtbl[pte] & PAGE_PRESENT)) {      /* allocating page */
  220:                 if(!addr) {
  221:                         if(!(addr = kmalloc())) {
  222:                                 return 0;
  223:                         }
  224:                         addr = V2P(addr);
  225:                         p->rss++;
  226:                 }
  227:         }
  228:         pgtbl[pte] = addr | PAGE_PRESENT | PAGE_USER;
  229:         if(prot & PROT_WRITE) {
  230:                 pgtbl[pte] |= PAGE_RW;
  231:         }
  232:         return P2V(addr);
  233: }
  234:
  235: int unmap_page(unsigned int vaddr)
  236: {
  237:         unsigned int *pgdir, *pgtbl;
  238:         unsigned int addr;
  239:         int pde, pte;
  240:
  241:         pgdir = (unsigned int *)P2V(current->tss.cr3);
  242:         pde = GET_PGDIR(vaddr);
  243:         pte = GET_PGTBL(vaddr);
  244:         if(!(pgdir[pde] & PAGE_PRESENT)) {
  245:                 printk("WARNING: %s(): trying to unmap an unallocated pde '0x%08
x'\n", __FUNCTION__, vaddr);
  246:                 return 1;
  247:         }
  248:
  249:         pgtbl = (unsigned int *)P2V((pgdir[pde] & PAGE_MASK));
  250:         if(!(pgtbl[pte] & PAGE_PRESENT)) {
  251:                 printk("WARNING: %s(): trying to unmap an unallocated page '0x%0
8x'\n", __FUNCTION__, vaddr);
  252:                 return 1;
```

mm/memory.c Page 5/8

```
  253:          }
  254:
  255:          addr = pgtbl[pte] & PAGE_MASK;
  256:          pgtbl[pte] = NULL;
  257:          kfree(P2V(addr));
  258:          current->rss--;
  259:          return 0;
  260: }
  261:
  262: void mem_init(void)
  263: {
  264:          unsigned int pages, sizek;
  265:          unsigned int n;
  266:          unsigned int physical_page_tables;
  267:          unsigned int physical_memory;
  268:
  269:          physical_page_tables = (kstat.physical_pages / 1024) + ((kstat.physical_
pages % 1024) ? 1 : 0);
  270:          physical_memory = (kstat.physical_pages << PAGE_SHIFT); /* in bytes */
  271:
  272:          /* Page Directory will be aligned to the next page */
  273:          _last_data_addr = PAGE_ALIGN(_last_data_addr);
  274:          kpage_dir = (unsigned int *)_last_data_addr;
  275:          memset_b(kpage_dir, NULL, PAGE_SIZE);
  276:          _last_data_addr += PAGE_SIZE;
  277:
  278:          /* Page Tables */
  279:          kpage_table = (unsigned int *)_last_data_addr;
  280:          memset_b(kpage_table, NULL, physical_page_tables * PAGE_SIZE);
  281:          _last_data_addr += physical_page_tables * PAGE_SIZE;
  282:
  283:          /* Page Directory and Page Tables initialization */
  284:          for(n = 0; n < kstat.physical_pages; n++) {
  285:                  kpage_table[n] = (n << PAGE_SHIFT) | PAGE_PRESENT | PAGE_RW;
  286:          }
  287:          for(n = 0; n < physical_page_tables; n++) {
  288:                  kpage_dir[GET_PGDIR(KERNEL_BASE_ADDR) + n] = (unsigned int)&kpag
e_table[n * 1024] | PAGE_PRESENT | PAGE_RW;
  289:          }
  290:
  291:          map_kaddr(0xA0000, KERNEL_ENTRY_ADDR, PAGE_PRESENT | PAGE_RW);
  292:          map_kaddr(KERNEL_ENTRY_ADDR, _last_data_addr, PAGE_PRESENT | PAGE_RW);
  293: /*       printk("_last_data_addr = 0x%08x-0x%08x (kernel)\n", KERNEL_ENTRY_ADDR,
_last_data_addr); */
  294:          activate_kpage_dir();
  295:
  296:          /* since Page Directory is now activated we can use virtual addresses */
  297:          _last_data_addr = P2V(_last_data_addr);
  298:
  299:
  300:          /* reserve memory space for proc_table[NR_PROCS] */
  301:          proc_table_size = PAGE_ALIGN(sizeof(struct proc) * NR_PROCS);
  302:          if(!addr_in_bios_map(V2P(_last_data_addr) + proc_table_size)) {
  303:                  PANIC("Not enough memory for proc_table.\n");
  304:          }
  305: /*       printk("_last_data_addr = 0x%08x-0x%08x (proc_table)\n", _last_data_addr
, _last_data_addr + proc_table_size); */
  306:          proc_table = (struct proc *)_last_data_addr;
  307:          _last_data_addr += proc_table_size;
  308:
  309:
  310:          /* reserve memory space for buffer_table */
  311:          buffer_table_size = (kstat.physical_pages * BUFFER_PERCENTAGE) / 100;
  312:          buffer_table_size *= sizeof(struct buffer);
  313:          pages = buffer_table_size >> PAGE_SHIFT;
  314:          buffer_table_size = !pages ? 4096 : pages << PAGE_SHIFT;
  315: /*       printk("_last_data_addr = 0x%08x-0x%08x (buffer_table)\n", _last_data_ad
```

mm/memory.c Page 6/8

```
dr, _last_data_addr + buffer_table_size); */
  316:          if(!addr_in_bios_map(V2P(_last_data_addr) + buffer_table_size)) {
  317:                  PANIC("Not enough memory for buffer_table.\n");
  318:          }
  319:          buffer_table = (struct buffer *)_last_data_addr;
  320:          _last_data_addr += buffer_table_size;
  321:
  322:
  323:          /* reserve memory space for buffer_hash_table */
  324:          n = (buffer_table_size / sizeof(struct buffer) * BUFFER_HASH_PERCENTAGE)
 / 100;
  325:          n = MAX(n, 10); /* 10 buffer hashes as minimum */
  326:          /* buffer_hash_table is an array of pointers */
  327:          pages = ((n * sizeof(unsigned int)) / PAGE_SIZE) + 1;
  328:          buffer_hash_table_size = pages << PAGE_SHIFT;
  329: /*       printk("_last_data_addr = 0x%08x-0x%08x (buffer_hash_table)\n", _last_da
ta_addr, _last_data_addr + buffer_hash_table_size); */
  330:          if(!addr_in_bios_map(V2P(_last_data_addr) + buffer_hash_table_size)) {
  331:                  PANIC("Not enough memory for buffer_hash_table.\n");
  332:          }
  333:          buffer_hash_table = (struct buffer **)_last_data_addr;
  334:          _last_data_addr += buffer_hash_table_size;
  335:
  336:
  337:          /* reserve memory space for inode_table */
  338:          sizek = physical_memory / 1024; /* this helps to avoid overflow */
  339:          inode_table_size = (sizek * INODE_PERCENTAGE) / 100;
  340:          inode_table_size *= 1024;
  341:          pages = inode_table_size >> PAGE_SHIFT;
  342:          inode_table_size = pages << PAGE_SHIFT;
  343: /*       printk("_last_data_addr = 0x%08x-0x%08x (inode_table)\n", _last_data_add
r, _last_data_addr + inode_table_size); */
  344:          if(!addr_in_bios_map(V2P(_last_data_addr) + inode_table_size)) {
  345:                  PANIC("Not enough memory for inode_table.\n");
  346:          }
  347:          inode_table = (struct inode *)_last_data_addr;
  348:          _last_data_addr += inode_table_size;
  349:
  350:
  351:          /* reserve memory space for inode_hash_table */
  352:          n = ((inode_table_size / sizeof(struct inode)) * INODE_HASH_PERCENTAGE)
/ 100;
  353:          n = MAX(n, 10); /* 10 inodes hashes as minimum */
  354:          /* inode_hash_table is an array of pointers */
  355:          pages = ((n * sizeof(unsigned int)) / PAGE_SIZE) + 1;
  356:          inode_hash_table_size = pages << PAGE_SHIFT;
  357: /*       printk("_last_data_addr = 0x%08x-0x%08x (inode_hash_table)\n", _last_dat
a_addr, _last_data_addr + inode_hash_table_size); */
  358:          if(!addr_in_bios_map(V2P(_last_data_addr) + inode_hash_table_size)) {
  359:                  PANIC("Not enough memory for inode_hash_table.\n");
  360:          }
  361:          inode_hash_table = (struct inode **)_last_data_addr;
  362:          _last_data_addr += inode_hash_table_size;
  363:
  364:
  365:          /* reserve memory space for fd_table[NR_OPENS] */
  366:          fd_table_size = PAGE_ALIGN(sizeof(struct fd) * NR_OPENS);
  367: /*       printk("_last_data_addr = 0x%08x-0x%08x (fd_table)\n", _last_data_addr,
_last_data_addr + fd_table_size); */
  368:          if(!addr_in_bios_map(V2P(_last_data_addr) + fd_table_size)) {
  369:                  PANIC("Not enough memory for fd_table.\n");
  370:          }
  371:          fd_table = (struct fd *)_last_data_addr;
  372:          _last_data_addr += fd_table_size;
  373:
  374:
  375:          /* reserve memory space for mount_table[NR_MOUNT_POINTS] */
```

mm/memory.c


```
  376:          mount_table_size = PAGE_ALIGN(sizeof(struct mount) * NR_MOUNT_POINTS);
  377: /*       printk("_last_data_addr = 0x%08x-0x%08x (mount_table)\n", _last_data_add
r, _last_data_addr + mount_table_size); */
  378:          if(!addr_in_bios_map(V2P(_last_data_addr) + mount_table_size)) {
  379:                  PANIC("Not enough memory for mount_table.\n");
  380:          }
  381:          mount_table = (struct mount *)_last_data_addr;
  382:          _last_data_addr += mount_table_size;
  383:
  384:
  385:          /* reserve memory space for RAMdisk(s) */
  386:          if(!_noramdisk) {
  387:                  if(!_ramdisksize) {
  388:                          _ramdisksize = RAMDISK_SIZE;
  389:                  }
  390:                  if(!addr_in_bios_map(V2P(_last_data_addr) + (_ramdisksize * 1024
))) {
  391:                          printk("WARNING: RAMdisk device disabled (not enough phy
sical memory).\n");
  392:                          _noramdisk = 1;
  393:                  } else {
  394:                          /*
  395:                           * If the 'initrd=' parameter was supplied, then the fir
st
  396:                           * ramdisk was already assigned to the initial ramdisk i
mage.
  397:                           */
  398:                          if(ramdisk_table[0].addr) {
  399:                                  n = 1;
  400:                                  _last_data_addr += _ramdisksize * 1024;
  401:                          } else {
  402:                                  n = 0;
  403:                          }
  404:                          for(; n < RAMDISK_MINORS; n++) {
  405: /*                                printk("_last_data_addr = 0x%08x-0x%08x (/dev/ra
m%d)\n", _last_data_addr, _last_data_addr + (_ramdisksize * 1024), n); */
  406:                                  ramdisk_table[n].addr = (char *)_last_data_addr;
  407:                                  _last_data_addr += _ramdisksize * 1024;
  408:                          }
  409:                  }
  410:          }
  411:
  412:
  413:          /* the last one must be the page_table structure */
  414:          page_hash_table_size = 1 * PAGE_SIZE;   /* only 1 page size */
  415:          if(!addr_in_bios_map(V2P(_last_data_addr) + page_hash_table_size)) {
  416:                  PANIC("Not enough memory for page_hash_table.\n");
  417:          }
  418:          page_hash_table = (struct page **)_last_data_addr;
  419: /*       printk("_last_data_addr = 0x%08x-0x%08x (page_hash_table)\n", _last_data
_addr, _last_data_addr + page_hash_table_size); */
  420:          _last_data_addr += page_hash_table_size;
  421:
  422:          page_table_size = PAGE_ALIGN(kstat.physical_pages * sizeof(struct page))
;
  423:          if(!addr_in_bios_map(V2P(_last_data_addr) + page_table_size)) {
  424:                  PANIC("Not enough memory for page_table.\n");
  425:          }
  426:          page_table = (struct page *)_last_data_addr;
  427: /*       printk("page_table_size = %d\n", page_table_size); */
  428: /*       printk("_last_data_addr = 0x%08x-0x%08x (page_table)\n", _last_data_addr
, _last_data_addr + page_table_size); */
  429:          _last_data_addr += page_table_size;
  430:
  431:          page_init(kstat.physical_pages);
  432: }
  433:
```

mm/memory.c


```
  434: void mem_stats(void)
  435: {
  436:         printk("\n");
  437:         printk("memory: total/available=%dKB/%dKB, kernel=%dKB, reserved=%dKB\n"
, kstat.physical_pages << 2, kstat.total_mem_pages << 2, kstat.kernel_reserved, kstat.p
hysical_reserved);
  438:         printk("kernel: text=%dKB, data=%dKB, bss=%dKB, i/o buffers=%d (%dKB)\n"
, KERNEL_TEXT_SIZE / 1024, KERNEL_DATA_SIZE / 1024, KERNEL_BSS_SIZE / 1024, buffer_tabl
e_size / sizeof(struct buffer), (buffer_table_size + buffer_hash_table_size) / 1024);
  439:         printk("\tinodes=%d (%dKB)\n\n", inode_table_size / sizeof(struct inode)
, (inode_table_size + inode_hash_table_size) / 1024);
  440: }
```

mm/mmap.c


```
    1: /*
    2:  * fiwix/mm/mmap.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/asm.h>
    9: #include <fiwix/mm.h>
   10: #include <fiwix/fs.h>
   11: #include <fiwix/fcntl.h>
   12: #include <fiwix/stat.h>
   13: #include <fiwix/process.h>
   14: #include <fiwix/mman.h>
   15: #include <fiwix/errno.h>
   16: #include <fiwix/stdio.h>
   17: #include <fiwix/string.h>
   18:
   19: void show_vma_regions(struct proc *p)
   20: {
   21:         __ino_t inode;
   22:         int major, minor;
   23:         char *section;
   24:         char r, w, x, f;
   25:         struct vma *vma;
   26:         unsigned int n;
   27:         int count;
   28:
   29:         vma = p->vma;
   30:         printk("num  address range         flag offset     dev   inode      mod
section cnt\n");
   31:         printk("---- -------------------- ---- ---------- ----- ---------- ---
------- ----\n");
   32:         for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
   33:                 r = vma->prot & PROT_READ ? 'r' : '-';
   34:                 w = vma->prot & PROT_WRITE ? 'w' : '-';
   35:                 x = vma->prot & PROT_EXEC ? 'x' : '-';
   36:                 if(vma->flags & MAP_SHARED) {
   37:                         f = 's';
   38:                 } else if(vma->flags & MAP_PRIVATE) {
   39:                         f = 'p';
   40:                 } else {
   41:                         f = '-';
   42:                 }
   43:                 switch(vma->s_type) {
   44:                         case P_TEXT:    section = "text ";
   45:                                         break;
   46:                         case P_DATA:    section = "data ";
   47:                                         break;
   48:                         case P_BSS:     section = "bss  ";
   49:                                         break;
   50:                         case P_HEAP:    section = "heap ";
   51:                                         break;
   52:                         case P_STACK:   section = "stack";
   53:                                         break;
   54:                         case P_MMAP:    section = "mmap ";
   55:                                         break;
   56:                         default:
   57:                                 section = NULL;
   58:                                 break;
   59:                 }
   60:                 inode = major = minor = count = 0;
   61:                 if(vma->inode) {
   62:                         inode = vma->inode->inode;
   63:                         major = MAJOR(vma->inode->dev);
   64:                         minor = MINOR(vma->inode->dev);
   65:                         count = vma->inode->count;
```

mm/mmap.c


```
   66:                 }
   67:                 printk("[%02d] 0x%08x-0x%08x %c%c%c%c 0x%08x %02d:%02d %- 10u <%
d> [%s]  (%d)\n", n, vma->start, vma->end, r, w, x, f, vma->offset, major, minor, inode
, vma->o_mode, section, count);
   68:         }
   69:         if(!n) {
   70:                 printk("[no vma regions]\n");
   71:         }
   72: }
   73:
   74: static struct vma * get_new_vma_region(void)
   75: {
   76:         unsigned int n;
   77:         struct vma *vma;
   78:
   79:         vma = current->vma;
   80:
   81:         for(n = 0; n < VMA_REGIONS; n++, vma++) {
   82:                 if(!vma->start && !vma->end) {
   83:                         return vma;
   84:                 }
   85:         }
   86:         return NULL;
   87: }
   88:
   89: /*
   90:  * This sorts regions (in ascending order), merging equal regions and keeping
   91:  * the unused ones at the end of the array.
   92:  */
   93: static void sort_vma(void)
   94: {
   95:         unsigned int n, n2, needs_sort;
   96:         struct vma *vma, tmp;
   97:
   98:         vma = current->vma;
   99:
  100:         do {
  101:                 needs_sort = 0;
  102:                 for(n = 0, n2 = 1; n2 < VMA_REGIONS; n++, n2++) {
  103:                         if(vma[n].end && vma[n2].start) {
  104:                                 if((vma[n].end == vma[n2].start) &&
  105:                                   (vma[n].prot == vma[n2].prot) &&
  106:                                   (vma[n].flags == vma[n2].flags) &&
  107:                                   (vma[n].offset == vma[n2].offset) &&
  108:                                   (vma[n].s_type == vma[n2].s_type) &&
  109:                                   (vma[n].inode == vma[n2].inode)) {
  110:                                         vma[n].end = vma[n2].end;
  111:                                         memset_b(&vma[n2], NULL, sizeof(struct v
ma));
  112:                                         needs_sort++;
  113:                                 }
  114:                         }
  115:                         if((vma[n2].start && (vma[n].start > vma[n2].start)) ||
(!vma[n].start && vma[n2].start)) {
  116:                                 memcpy_b(&tmp, &vma[n], sizeof(struct vma));
  117:                                 memcpy_b(&vma[n], &vma[n2], sizeof(struct vma));
  118:                                 memcpy_b(&vma[n2], &tmp, sizeof(struct vma));
  119:                                 needs_sort++;
  120:                         }
  121:                 }
  122:         } while(needs_sort);
  123: }
  124:
  125: /*
  126:  * This function removes all redundant entries.
  127:  *
  128:  * for example, if for any reason the map looks like this:
```

mm/mmap.c

```
  129:  * [01] 0x0808e984-0x08092000 rw-p 0x00000000 0
  130:  * [02] 0x0808f000-0x0808ffff rw-p 0x000c0000 4066
  131:  *
  132:  * this function converts it to this:
  133:  * [01] 0x0808e984-0x0808f000 rw-p 0x00000000 0
  134:  * [02] 0x0808f000-0x0808ffff rw-p 0x000c0000 4066
  135:  * [03] 0x08090000-0x08092000 rw-p 0x00000000 0
  136:  */
  137: static int optimize_vma(void)
  138: {
  139:         unsigned int n, needs_sort;
  140:         struct vma *vma, *prev, *new;
  141:
  142:         for(;;) {
  143:                 needs_sort = 0;
  144:                 prev = new = NULL;
  145:                 vma = current->vma;
  146:                 for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  147:                         if(!prev) {
  148:                                 prev = vma;
  149:                                 continue;
  150:                         }
  151:                         if((vma->start < prev->end)) {
  152:                                 if(!(new = get_new_vma_region())) {
  153:                                         printk("WARNING: %s(): unable to get a f
ree vma region.\n", __FUNCTION__);
  154:                                         return -ENOMEM;
  155:                                 }
  156:                                 new->start = vma->end;
  157:                                 new->end = prev->end;
  158:                                 new->prot = prev->prot;
  159:                                 new->flags = prev->flags;
  160:                                 new->offset = prev->offset;
  161:                                 new->s_type = prev->s_type;
  162:                                 new->inode = prev->inode;
  163:                                 new->o_mode = prev->o_mode;
  164:                                 prev->end = vma->start;
  165:                                 needs_sort++;
  166:                                 if(prev->start == prev->end) {
  167:                                         memset_b(prev, NULL, sizeof(struct vma))
;
  168:                                 }
  169:                                 if(new->start == new->end) {
  170:                                         memset_b(new, NULL, sizeof(struct vma));
  171:                                 }
  172:                                 break;
  173:                         }
  174:                         prev = vma;
  175:                 }
  176:                 if(!needs_sort) {
  177:                         break;
  178:                 }
  179:                 sort_vma();
  180:         }
  181:
  182:         return 0;
  183: }
  184:
  185: /* return the first free address that matches with the size of length */
  186: static unsigned int get_unmapped_vma_region(unsigned int length)
  187: {
  188:         unsigned int n, addr;
  189:         struct vma *vma;
  190:
  191:         if(!length) {
  192:                 return 0;
  193:         }
```

mm/mmap.c


```
  194:
  195:         addr = MMAP_START;
  196:         vma = current->vma;
  197:
  198:         for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  199:                 if(vma->start < MMAP_START) {
  200:                         continue;
  201:                 }
  202:                 if(vma->start - addr >= length) {
  203:                         return PAGE_ALIGN(addr);
  204:                 }
  205:                 addr = PAGE_ALIGN(vma->end);
  206:         }
  207:         return 0;
  208: }
  209:
  210: static void free_vma_pages(unsigned int start, __size_t length, struct vma *vma)
  211: {
  212:         unsigned int n, addr;
  213:         unsigned int *pgdir, *pgtbl;
  214:         unsigned int pde, pte, page;
  215:         struct page *pg;
  216:
  217:         pgdir = (unsigned int *)P2V(current->tss.cr3);
  218:         pgtbl = NULL;
  219:
  220:         for(n = 0; n < (length / PAGE_SIZE); n++) {
  221:                 pde = GET_PGDIR(start + (n * PAGE_SIZE));
  222:                 pte = GET_PGTBL(start + (n * PAGE_SIZE));
  223:                 if(pgdir[pde] & PAGE_PRESENT) {
  224:                         pgtbl = (unsigned int *)P2V((pgdir[pde] & PAGE_MASK));
  225:                         if(pgtbl[pte] & PAGE_PRESENT) {
  226:                                 /* make sure to not free reserved pages */
  227:                                 page = pgtbl[pte] >> PAGE_SHIFT;
  228:                                 pg = &page_table[page];
  229:                                 if(pg->flags & PAGE_RESERVED) {
  230:                                         continue;
  231:                                 }
  232:
  233:                                 if(vma->prot & PROT_WRITE && vma->flags & MAP_SH
ARED) {
  234:                                         addr = start - vma->start + vma->offset;
  235:                                         write_page(pg, vma->inode, addr, length)
;
  236:                                 }
  237:
  238:                                 kfree(P2V(pgtbl[pte]) & PAGE_MASK);
  239:                                 current->rss--;
  240:                                 pgtbl[pte] = NULL;
  241:
  242:                                 /* check if a page table can be freed */
  243:                                 for(pte = 0; pte < PT_ENTRIES; pte++) {
  244:                                         if(pgtbl[pte] & PAGE_MASK) {
  245:                                                 break;
  246:                                         }
  247:                                 }
  248:                                 if(pte == PT_ENTRIES) {
  249:                                         kfree((unsigned int)pgtbl & PAGE_MASK);
  250:                                         current->rss--;
  251:                                         pgdir[pde] = NULL;
  252:                                 }
  253:                         }
  254:                 }
  255:         }
  256: }
  257:
  258: static int free_vma_region(struct vma *vma, unsigned int start, __ssize_t length
```

mm/mmap.c Page 5/8

```
)
  259: {
  260:         struct vma *new;
  261:
  262:         if(!(new = get_new_vma_region())) {
  263:                 printk("WARNING: %s(): unable to get a free vma region.\n", __FU
NCTION__);
  264:                 return -ENOMEM;
  265:         }
  266:
  267:         new->start = start + length;
  268:         new->end = vma->end;
  269:         new->prot = vma->prot;
  270:         new->flags = vma->flags;
  271:         new->offset = vma->offset;
  272:         new->s_type = vma->s_type;
  273:         new->inode = vma->inode;
  274:         new->o_mode = vma->o_mode;
  275:
  276:         vma->end = start;
  277:
  278:         if(vma->start == vma->end) {
  279:                 if(vma->inode) {
  280:                         iput(vma->inode);
  281:                 }
  282:                 memset_b(vma, NULL, sizeof(struct vma));
  283:         }
  284:         if(new->start == new->end) {
  285:                 memset_b(new, NULL, sizeof(struct vma));
  286:         }
  287:         return 0;
  288: }
  289:
  290: void release_binary(void)
  291: {
  292:         unsigned int n;
  293:         struct vma *vma;
  294:
  295:         vma = current->vma;
  296:
  297:         for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  298:                 free_vma_pages(vma->start, vma->end - vma->start, vma);
  299:                 free_vma_region(vma, vma->start, vma->end - vma->start);
  300:         }
  301:         sort_vma();
  302:         optimize_vma();
  303:         invalidate_tlb();
  304: }
  305:
  306: struct vma * find_vma_region(unsigned int addr)
  307: {
  308:         unsigned int n;
  309:         struct vma *vma;
  310:
  311:         if(!addr) {
  312:                 return NULL;
  313:         }
  314:
  315:         addr &= PAGE_MASK;
  316:         vma = current->vma;
  317:
  318:         for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  319:                 if((addr >= vma->start) && (addr < vma->end)) {
  320:                         return vma;
  321:                 }
  322:         }
  323:         return NULL;
```

mm/mmap.c


```
  324: }
  325:
  326: int expand_heap(unsigned int new)
  327: {
  328:         unsigned int n;
  329:         struct vma *vma, *heap;
  330:
  331:         vma = current->vma;
  332:         heap = NULL;
  333:
  334:         for(n = 0; n < VMA_REGIONS && vma->start; n++, vma++) {
  335:                 /* make sure the new heap won't overlap the next region */
  336:                 if(heap && new < vma->start) {
  337:                         heap->end = new;
  338:                         return 0;
  339:                 } else {
  340:                         heap = NULL;    /* was a bad candidate */
  341:                 }
  342:                 if(!heap && vma->s_type == P_HEAP) {
  343:                         heap = vma;     /* possible candidate */
  344:                         continue;
  345:                 }
  346:         }
  347:
  348:         /* out of memory! */
  349:         return 1;
  350: }
  351:
  352: int do_mmap(struct inode *i, unsigned int start, unsigned int length, unsigned i
nt prot, unsigned int flags, unsigned int offset, char type, char mode)
  353: {
  354:         struct vma *vma;
  355:         int errno;
  356:
  357:         if(!(length = PAGE_ALIGN(length))) {
  358:                 return start;
  359:         }
  360:
  361:         /* file mapping */
  362:         if(i) {
  363:                 if(!S_ISREG(i->i_mode) && !S_ISCHR(i->i_mode)) {
  364:                         return -ENODEV;
  365:                 }
  366:
  367:                 /*
  368:                  * The file shall have been opened with read permission,
  369:                  * regardless of the protection options specified.
  370:                  * IEEE Std 1003.1, 2004 Edition.
  371:                  */
  372:                 if(mode == O_WRONLY) {
  373:                         return -EACCES;
  374:                 }
  375:                 switch(flags & MAP_TYPE) {
  376:                         case MAP_SHARED:
  377:                                 if(prot & PROT_WRITE) {
  378:                                         if(!(mode & (O_WRONLY | O_RDWR))) {
  379:                                                 return -EACCES;
  380:                                         }
  381:                                 }
  382:                                 break;
  383:                         case MAP_PRIVATE:
  384:                                 break;
  385:                         default:
  386:                                 return -EINVAL;
  387:                 }
  388:                 i->count++;
  389:
```

mm/mmap.c

```
  390:          /* anonymous mapping */
  391:          } else {
  392:                  if((flags & MAP_TYPE) != MAP_PRIVATE) {
  393:                          return -EINVAL;
  394:                  }
  395:
  396:                  /* anonymous objects must be filled with zeros */
  397:                  flags |= ZERO_PAGE;
  398:          }
  399:
  400:          if(flags & MAP_FIXED) {
  401:                  if(start & ~PAGE_MASK) {
  402:                          return -EINVAL;
  403:                  }
  404:          } else {
  405:                  start = get_unmapped_vma_region(length);
  406:                  if(!start) {
  407:                          printk("WARNING: %s(): unable to get an unmapped vma reg
ion.\n", __FUNCTION__);
  408:                          return -ENOMEM;
  409:                  }
  410:          }
  411:
  412:          if(!(vma = get_new_vma_region())) {
  413:                  printk("WARNING: %s(): unable to get a free vma region.\n", __FU
NCTION__);
  414:                  return -ENOMEM;
  415:          }
  416:
  417:          vma->start = start;
  418:          vma->end = start + length;
  419:          vma->prot = prot;
  420:          vma->flags = flags;
  421:          vma->offset = offset;
  422:          vma->s_type = type;
  423:          vma->inode = i;
  424:          vma->o_mode = mode;
  425:
  426:          if(i && i->fsop->mmap) {
  427:                  if((errno = i->fsop->mmap(i, vma))) {
  428:                          int errno2;
  429:
  430:                          if((errno2 = free_vma_region(vma, start, length))) {
  431:                                  return errno2;
  432:                          }
  433:                          sort_vma();
  434:                          if((errno2 = optimize_vma())) {
  435:                                  return errno2;
  436:                          }
  437:                          return errno;
  438:                  }
  439:          }
  440:
  441:          sort_vma();
  442:          if((errno = optimize_vma())) {
  443:                  return errno;
  444:          }
  445:          return start;
  446: }
  447:
  448: int do_munmap(unsigned int addr, __size_t length)
  449: {
  450:          struct vma *vma;
  451:          unsigned int size;
  452:          int errno;
  453:
  454:          if((addr & ~PAGE_MASK) || length < 0) {
```

mm/mmap.c


```
  455:                    return -EINVAL;
  456:            }
  457:
  458:            length = PAGE_ALIGN(length);
  459:
  460:            while(length) {
  461:                    if((vma = find_vma_region(addr))) {
  462:                            if((addr + length) > vma->end) {
  463:                                    size = vma->end - addr;
  464:                            } else {
  465:                                    size = length;
  466:                            }
  467:
  468:                            free_vma_pages(addr, size, vma);
  469:                            invalidate_tlb();
  470:                            if((errno = free_vma_region(vma, addr, size))) {
  471:                                    return errno;
  472:                            }
  473:                            sort_vma();
  474:                            if((errno = optimize_vma())) {
  475:                                    return errno;
  476:                            }
  477:                            length -= size;
  478:                            addr += size;
  479:                    } else {
  480:                            break;
  481:                    }
  482:            }
  483:
  484:            return 0;
  485: }
  486:
  487: int do_mprotect(struct vma *vma, unsigned int addr, __size_t length, int prot)
  488: {
  489:            struct vma *new;
  490:            int errno;
  491:
  492:            if(!(new = get_new_vma_region())) {
  493:                    printk("WARNING: %s(): unable to get a free vma region.\n", __FU
NCTION__);
  494:                    return -ENOMEM;
  495:            }
  496:
  497:            new->start = addr;
  498:            new->end = addr + length;
  499:            new->prot = prot;
  500:            new->flags = vma->flags;
  501:            new->offset = vma->offset;
  502:            new->s_type = vma->s_type;
  503:            new->inode = vma->inode;
  504:            new->o_mode = vma->o_mode;
  505:
  506:            sort_vma();
  507:            if((errno = optimize_vma())) {
  508:                    return errno;
  509:            }
  510:            return 0;
  511: }
```

mm/page.c


```
    1: /*
    2:  * fiwix/mm/page.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: /*
    9:  * page.c implements a cache with a free list as a doubly circular linked
   10:  * list and a chained hash table with doubly linked lists.
   11:  *
   12:  * hash table
   13:  * +--------+  +--------------+  +--------------+  +--------------+
   14:  * | index  |  |prev|data|next|  |prev|data|next|  |prev|data|next|
   15:  * |   0   --> | /  |    | --->  <--- |    | --->  <--- |    |  / |
   16:  * +--------+  +--------------+  +--------------+  +--------------+
   17:  * +--------+  +--------------+  +--------------+  +--------------+
   18:  * | index  |  |prev|data|next|  |prev|data|next|  |prev|data|next|
   19:  * |   1   --> | /  |    | --->  <--- |    | --->  <--- |    |  / |
   20:  * +--------+  +--------------+  +--------------+  +--------------+
   21:  *              (page)            (page)            (page)
   22:  *    ...
   23:  */
   24:
   25: #include <fiwix/asm.h>
   26: #include <fiwix/kernel.h>
   27: #include <fiwix/mm.h>
   28: #include <fiwix/mman.h>
   29: #include <fiwix/bios.h>
   30: #include <fiwix/sleep.h>
   31: #include <fiwix/sched.h>
   32: #include <fiwix/devices.h>
   33: #include <fiwix/buffer.h>
   34: #include <fiwix/errno.h>
   35: #include <fiwix/stdio.h>
   36: #include <fiwix/string.h>
   37:
   38: #define PAGE_HASH(inode, offset)        (((__ino_t)(inode) ^ (__off_t)(offset))
% NR_PAGE_HASH)
   39: #define NR_PAGES        page_table_size / sizeof(struct page)
   40: #define NR_PAGE_HASH    page_hash_table_size / sizeof(unsigned int)
   41:
   42: struct page *page_table;                /* page pool */
   43: struct page *page_head;                 /* page pool head */
   44: struct page **page_hash_table;
   45:
   46: static void insert_to_hash(struct page *pg)
   47: {
   48:         struct page **h;
   49:         int i;
   50:
   51:         i = PAGE_HASH(pg->inode->inode, pg->offset);
   52:         h = &page_hash_table[i];
   53:
   54:         if(!*h) {
   55:                 *h = pg;
   56:                 (*h)->prev_hash = (*h)->next_hash = NULL;
   57:         } else {
   58:                 pg->prev_hash = NULL;
   59:                 pg->next_hash = *h;
   60:                 (*h)->prev_hash = pg;
   61:                 *h = pg;
   62:         }
   63:         kstat.cached += (PAGE_SIZE / 1024);
   64: }
   65:
   66: static void remove_from_hash(struct page *pg)
```

mm/page.c


```
 67: {
 68:         struct page **h;
 69:         int i;
 70:
 71:         if(!pg->inode) {
 72:                 return;
 73:         }
 74:
 75:         i = PAGE_HASH(pg->inode->inode, pg->offset);
 76:         h = &page_hash_table[i];
 77:
 78:         while(*h) {
 79:                 if(*h == pg) {
 80:                         if((*h)->next_hash) {
 81:                                 (*h)->next_hash->prev_hash = (*h)->prev_hash;
 82:                         }
 83:                         if((*h)->prev_hash) {
 84:                                 (*h)->prev_hash->next_hash = (*h)->next_hash;
 85:                         }
 86:                         if(h == &page_hash_table[i]) {
 87:                                 *h = (*h)->next_hash;
 88:                         }
 89:                         kstat.cached -= (PAGE_SIZE / 1024);
 90:                         break;
 91:                 }
 92:                 h = &(*h)->next_hash;
 93:         }
 94: }
 95:
 96: static void remove_from_free_list(struct page *pg)
 97: {
 98:         pg->prev_free->next_free = pg->next_free;
 99:         pg->next_free->prev_free = pg->prev_free;
100:         kstat.free_pages--;
101:         if(pg == page_head) {
102:                 page_head = pg->next_free;
103:         }
104: }
105:
106: void page_lock(struct page *pg)
107: {
108:         unsigned long int flags;
109:
110:         for(;;) {
111:                 SAVE_FLAGS(flags); CLI();
112:                 if(pg->locked) {
113:                         RESTORE_FLAGS(flags);
114:                         sleep(&pg, PROC_UNINTERRUPTIBLE);
115:                 } else {
116:                         break;
117:                 }
118:         }
119:         pg->locked = 1;
120:         RESTORE_FLAGS(flags);
121: }
122:
123: void page_unlock(struct page *pg)
124: {
125:         unsigned long int flags;
126:
127:         SAVE_FLAGS(flags); CLI();
128:         pg->locked = 0;
129:         wakeup(pg);
130:         RESTORE_FLAGS(flags);
131: }
132:
133: struct page * get_free_page(void)
```

mm/page.c Page 3/7

```
  134: {
  135:         unsigned long int flags;
  136:         struct page *pg;
  137:
  138:         /* if no more pages on free list */
  139:         while(page_head == page_head->next_free) {
  140:                 /* reclaim some memory from buffer cache */
  141:                 wakeup(&kswapd);
  142:                 sleep(&get_free_page, PROC_UNINTERRUPTIBLE);
  143:
  144:                 if(page_head == page_head->next_free) {
  145:                         /* definitely out of memory! (no more pages) */
  146:                         printk("%s(): pid %d ran out of memory. OOM killer neede
d!\n", __FUNCTION__, current->pid);
  147:                         return NULL;
  148:                 }
  149:         }
  150:
  151:         SAVE_FLAGS(flags); CLI();
  152:
  153:         pg = page_head;
  154:         remove_from_free_list(pg);
  155:         remove_from_hash(pg);   /* remove it from its old hash */
  156:         pg->count = 1;
  157:         pg->inode = NULL;
  158:         pg->offset = 0;
  159:
  160:         RESTORE_FLAGS(flags);
  161:         return pg;
  162: }
  163:
  164: struct page * search_page_hash(struct inode *inode, __off_t offset)
  165: {
  166:         struct page *pg;
  167:         int i;
  168:
  169:         i = PAGE_HASH(inode->inode, offset);
  170:         pg = page_hash_table[i];
  171:
  172:         while(pg) {
  173:                 if(pg->inode == inode && pg->offset == offset) {
  174:                         if(!pg->count) {
  175:                                 remove_from_free_list(pg);
  176:                         }
  177:                         pg->count++;
  178:                         return pg;
  179:                 }
  180:                 pg = pg->next_hash;
  181:         }
  182:
  183:         return NULL;
  184: }
  185:
  186: void release_page(unsigned int page)
  187: {
  188:         unsigned long int flags;
  189:         struct page *pg;
  190:
  191:         if(!valid_page(page)) {
  192:                 PANIC("Unexpected inconsistency in hash_table. Missing page %d (
0x%x).\n", page, page);
  193:         }
  194:
  195:         pg = &page_table[page];
  196:         if(--pg->count > 0) {
  197:                 return;
  198:         }
```

mm/page.c Page 4/7

```
  199:
  200:          SAVE_FLAGS(flags); CLI();
  201:
  202:          if(!page_head) {
  203:                  pg->prev_free = pg->next_free = pg;
  204:                  page_head = pg;
  205:          } else {
  206:                  pg->next_free = page_head;
  207:                  pg->prev_free = page_head->prev_free;
  208:                  page_head->prev_free->next_free = pg;
  209:                  page_head->prev_free = pg;
  210:          }
  211:
  212:          /* if page is not cached then place it at the head of the free list */
  213:          if(!pg->inode) {
  214:                  page_head = pg;
  215:          }
  216:
  217:          kstat.free_pages++;
  218:
  219:          RESTORE_FLAGS(flags);
  220:          wakeup(&get_free_page);
  221: }
  222:
  223: int valid_page(unsigned int page)
  224: {
  225:          return (page >= 0 && page < NR_PAGES);
  226: }
  227:
  228: void update_page_cache(struct inode *i, __off_t offset, const char *buf, int cou
nt)
  229: {
  230:          __off_t poffset;
  231:          struct page *pg;
  232:          int bytes;
  233:
  234:          poffset = offset % PAGE_SIZE;
  235:          offset &= PAGE_MASK;
  236:          bytes = PAGE_SIZE - poffset;
  237:
  238:          if(count) {
  239:                  bytes = MIN(bytes, count);
  240:                  if((pg = search_page_hash(i, offset))) {
  241:                          page_lock(pg);
  242:                          memcpy_b(pg->data + poffset, buf, bytes);
  243:                          page_unlock(pg);
  244:                          release_page(pg->page);
  245:                  }
  246:          }
  247: }
  248:
  249: int write_page(struct page *pg, struct inode *i, __off_t offset, unsigned int le
ngth)
  250: {
  251:          struct fd fd_table;
  252:          unsigned int size;
  253:          int errno;
  254:
  255:          size = MIN(i->i_size, length);
  256:          fd_table.inode = i;
  257:          fd_table.flags = 0;
  258:          fd_table.count = 0;
  259:          fd_table.offset = offset;
  260:          if(i->fsop && i->fsop->write) {
  261:                  errno = i->fsop->write(i, &fd_table, pg->data, size);
  262:          } else {
  263:                  errno = -EINVAL;
```

mm/page.c Page 5/7

```
  264:         }
  265:
  266:         return errno;
  267: }
  268:
  269: int bread_page(struct page *pg, struct inode *i, __off_t offset, char prot, char
 flags)
  270: {
  271:         __blk_t block;
  272:         __off_t size_read;
  273:         int blksize;
  274:         struct device *d;
  275:         struct buffer *buf;
  276:
  277:         blksize = i->sb->s_blocksize;
  278:         size_read = 0;
  279:
  280:         if(!(d = get_device(BLK_DEV, MAJOR(i->dev)))) {
  281:                 printk("WARNING: %s: device major %d not found!\n", __FUNCTION__
, MAJOR(i->dev));
  282:                 return 1;
  283:         }
  284:         if(!d->fsop || !d->fsop->read_block) {
  285:                 printk("WARNING: %s: device %d,%d does not have the read_block()
 method!\n", __FUNCTION__, MAJOR(i->dev), MINOR(i->dev));
  286:                 return 1;
  287:         }
  288:
  289:         pg->inode = i;
  290:         pg->offset = offset;
  291:         if(!(prot & PROT_WRITE) || flags & MAP_SHARED) {
  292:                 while(size_read < PAGE_SIZE) {
  293:                         if((block = bmap(i, offset, FOR_READING)) < 0) {
  294:                                 return 1;
  295:                         }
  296:                         if(block) {
  297:                                 /* does exist a buffer with recent data? */
  298:                                 if(!(buf = get_dirty_buffer(i->dev, block, blksi
ze))) {
  299:                                         if(d->fsop->read_block(i->dev, block, pg
->data + size_read, blksize) < 0) {
  300:                                                 return 1;
  301:                                         }
  302:                                 } else {
  303:                                         memcpy_b(pg->data + size_read, buf->data
, blksize);
  304:                                         brelse(buf);
  305:                                 }
  306:                         } else {
  307:                                 /* fill the hole with zeros */
  308:                                 memset_b(pg->data + size_read, 0, blksize);
  309:                         }
  310:                         size_read += blksize;
  311:                         offset += blksize;
  312:                 }
  313:                 /* cache all read-only and public (shared) pages */
  314:                 insert_to_hash(pg);
  315:         } else {
  316:                 while(size_read < PAGE_SIZE) {
  317:                         if((block = bmap(i, offset, FOR_READING)) < 0) {
  318:                                 return 1;
  319:                         }
  320:                         if(block) {
  321:                                 /*
  322:                                  * This feeds the buffer cache by reading only
  323:                                  * the writable pages which aren't included in
  324:                                  * the page cache. This will speed up things by
```

mm/page.c Page 6/7

```
  325:                                    * keeping in buffer cache the writable pages
  326:                                    * with its original (disk) content (i.e. pages
  327:                                    * of the data section of an ELF).
  328:                                    */
  329:                                   if(!(buf = bread(i->dev, block, blksize))) {
  330:                                           return 1;
  331:                                   }
  332:                                   memcpy_b(pg->data + size_read, buf->data, blksiz
e);
  333:                                   brelse(buf);
  334:                           }
  335:                           size_read += blksize;
  336:                           offset += blksize;
  337:                   }
  338:                   pg->inode = NULL;
  339:                   pg->offset = 0;
  340:           }
  341:           return 0;
  342: }
  343:
  344: int file_read(struct inode *i, struct fd *fd_table, char *buffer, __size_t count
)
  345: {
  346:           __off_t total_read;
  347:           unsigned int page, poffset, bytes;
  348:           struct page *pg;
  349:
  350:           inode_lock(i);
  351:
  352:           if(fd_table->offset > i->i_size) {
  353:                   fd_table->offset = i->i_size;
  354:           }
  355:
  356:           total_read = 0;
  357:
  358:           for(;;) {
  359:                   count = (fd_table->offset + count > i->i_size) ? i->i_size - fd_
table->offset : count;
  360:                   if(!count) {
  361:                           break;
  362:                   }
  363:
  364:                   poffset = fd_table->offset % PAGE_SIZE;
  365:                   if(!(pg = search_page_hash(i, fd_table->offset & PAGE_MASK))) {
  366:                           if(!(page = kmalloc())) {
  367:                                   inode_unlock(i);
  368:                                   printk("%s(): returning -ENOMEM\n", __FUNCTION__
);
  369:                                   return -ENOMEM;
  370:                           }
  371:                           page = V2P(page);
  372:                           pg = &page_table[page >> PAGE_SHIFT];
  373:                           if(bread_page(pg, i, fd_table->offset & PAGE_MASK, 0, MA
P_SHARED)) {
  374:                                   kfree((unsigned int)pg->data);
  375:                                   inode_unlock(i);
  376:                                   printk("%s(): returning -EIO\n", __FUNCTION__);
  377:                                   return -EIO;
  378:                           }
  379:                   }
  380:
  381:                   page_lock(pg);
  382:                   bytes = PAGE_SIZE - poffset;
  383:                   bytes = MIN(bytes, count);
  384:                   memcpy_b(buffer + total_read, pg->data + poffset, bytes);
  385:                   total_read += bytes;
  386:                   count -= bytes;
```

mm/page.c


```
387:                 poffset += bytes;
388:                 poffset %= PAGE_SIZE;
389:                 fd_table->offset += bytes;
390:                 page_unlock(pg);
391:                 kfree((unsigned int)pg->data);
392:         }
393:
394:         inode_unlock(i);
395:         return total_read;
396: }
397:
398: void page_init(unsigned int pages)
399: {
400:         struct page *pg;
401:         unsigned int n, addr;
402:
403:         memset_b(page_table, NULL, page_table_size);
404:         memset_b(page_hash_table, NULL, page_hash_table_size);
405:
406:         for(n = 0; n < pages; n++) {
407:                 pg = &page_table[n];
408:                 pg->page = n;
409:
410:                 addr = n << PAGE_SHIFT;
411:                 if(addr >= KERNEL_ENTRY_ADDR && addr < V2P(_last_data_addr)) {
412:                         pg->flags = PAGE_RESERVED;
413:                         kstat.kernel_reserved++;
414:                         continue;
415:                 }
416:
417:                 /*
418:                  * Some memory addresses are reserved, like the memory between
419:                  * 0xA0000 and 0xFFFFF and other addresses, mostly used by the
420:                  * VGA graphics adapter and BIOS.
421:                  */
422:                 if(!addr_in_bios_map(addr)) {
423:                         pg->flags = PAGE_RESERVED;
424:                         kstat.physical_reserved++;
425:                         continue;
426:                 }
427:
428:                 pg->data = (char *)P2V(addr);
429:                 if(!page_head) {
430:                         pg->prev_free = pg->next_free = pg;
431:                         page_head = pg;
432:                 } else {
433:                         pg->next_free = page_head;
434:                         pg->prev_free = page_head->prev_free;
435:                         page_head->prev_free->next_free = pg;
436:                         page_head->prev_free = pg;
437:                 }
438:                 kstat.free_pages++;
439:         }
440:         kstat.total_mem_pages = kstat.free_pages;
441:         kstat.kernel_reserved <<= 2;
442:         kstat.physical_reserved <<= 2;
443: }
```

mm/swapper.c


```
    1: /*
    2:  * fiwix/mm/swapper.c
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #include <fiwix/kernel.h>
    9: #include <fiwix/limits.h>
   10: #include <fiwix/process.h>
   11: #include <fiwix/sleep.h>
   12: #include <fiwix/sched.h>
   13: #include <fiwix/tty.h>
   14: #include <fiwix/memdev.h>
   15: #include <fiwix/lp.h>
   16: #include <fiwix/ramdisk.h>
   17: #include <fiwix/floppy.h>
   18: #include <fiwix/ide.h>
   19: #include <fiwix/buffer.h>
   20: #include <fiwix/mm.h>
   21: #include <fiwix/fs.h>
   22: #include <fiwix/locks.h>
   23: #include <fiwix/filesystems.h>
   24: #include <fiwix/stdio.h>
   25:
   26: /* kswapd continues the kernel initialization */
   27: int kswapd(void)
   28: {
   29:         sprintk(current->argv0, "%s", "kswapd");
   30:
   31:         /* char devices */
   32:         memdev_init();
   33:         lp_init();
   34:
   35:         /* block devices */
   36:         ramdisk_init();
   37:         floppy_init();
   38:         ide_init();
   39:
   40:         /* data structures */
   41:         sleep_init();
   42:         buffer_init();
   43:         sched_init();
   44:         mount_init();
   45:         inode_init();
   46:         fd_init();
   47:         flock_init();
   48:
   49:         mem_stats();
   50:         fs_init();
   51:         mount_root();
   52:         init_init();
   53:
   54:         for(;;) {
   55:                 sleep(&kswapd, PROC_UNINTERRUPTIBLE);
   56:                 if(reclaim_buffers()) {
   57:                         continue;
   58:                 }
   59:                 printk("WARNING: %s(): out of memory and swapping is not impleme
nted yet, sorry.\n", __FUNCTION__);
   60:                 wakeup(&get_free_page);
   61:         }
   62: }
```

lib/ctype.c


```
/*
 * fiwix/lib/ctype.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/ctype.h>

unsigned char _ctype[] = {
        0,
        _C,             /* ^@   0x00 (NUL '\0') */
        _C,             /* ^A   0x01 (SOH) */
        _C,             /* ^B   0x02 (STX) */
        _C,             /* ^C   0x03 (ETX) */
        _C,             /* ^D   0x04 (EOT) */
        _C,             /* ^E   0x05 (ENQ) */
        _C,             /* ^F   0x06 (ACK) */
        _C,             /* ^G   0x07 (BEL '\a') */
        _C,             /* ^H   0x08 (BS  '\b') */
        _C | _S,        /* ^I   0x09 (HT  '\t') */
        _C | _S,        /* ^J   0x0A (LF  '\n') */
        _C | _S,        /* ^K   0x0B (VT  '\v') */
        _C | _S,        /* ^L   0x0C (FF  '\f') */
        _C | _S,        /* ^M   0x0D (CR  '\r') */
        _C,             /* ^N   0x0E (SO) */
        _C,             /* ^O   0x0F (SI) */
        _C,             /* ^P   0x10 (DLE) */
        _C,             /* ^Q   0x11 (DC1) */
        _C,             /* ^R   0x12 (DC2) */
        _C,             /* ^S   0x13 (DC3) */
        _C,             /* ^T   0x14 (DC4) */
        _C,             /* ^U   0x15 (NAK) */
        _C,             /* ^V   0x16 (SYN) */
        _C,             /* ^W   0x17 (ETB) */
        _C,             /* ^X   0x18 (CAN) */
        _C,             /* ^Y   0x19 (EM) */
        _C,             /* ^Z   0x1A (SUB) */
        _C,             /* ^[   0x1B (ESC) */
        _C,             /* ^\   0x1C (FS) */
        _C,             /* ^]   0x1D (GS) */
        _C,             /* ^^   0x1E (RS) */
        _C,             /* ^_   0x1F (US) */
        _S,             /* ' '  0x20 */
        _P,             /* '!'  0x21 */
        _P,             /* '"'  0x22 */
        _P,             /* '#'  0x23 */
        _P,             /* '$'  0x24 */
        _P,             /* '%'  0x25 */
        _P,             /* '&'  0x26 */
        _P,             /* '''  0x27 */
        _P,             /* '('  0x28 */
        _P,             /* ')'  0x29 */
        _P,             /* '*'  0x2A */
        _P,             /* '+'  0x2B */
        _P,             /* ','  0x2C */
        _P,             /* '-'  0x2D */
        _P,             /* '.'  0x2E */
        _P,             /* '/'  0x2F */
        _N,             /* '0'  0x30 */
        _N,             /* '1'  0x31 */
        _N,             /* '2'  0x32 */
        _N,             /* '3'  0x33 */
        _N,             /* '4'  0x34 */
        _N,             /* '5'  0x35 */
        _N,             /* '6'  0x36 */
        _N,             /* '7'  0x37 */
```

lib/ctype.c


```
 68:         _N,             /* '8'  0x38 */
 69:         _N,             /* '9'  0x39 */
 70:         _P,             /* ':'  0x3A */
 71:         _P,             /* ';'  0x3B */
 72:         _P,             /* '<'  0x3C */
 73:         _P,             /* '='  0x3D */
 74:         _P,             /* '>'  0x3E */
 75:         _P,             /* '?'  0x3F */
 76:         _P,             /* '@'  0x40 */
 77:         _U | _X,        /* 'A'  0x41 */
 78:         _U | _X,        /* 'B'  0x42 */
 79:         _U | _X,        /* 'C'  0x43 */
 80:         _U | _X,        /* 'D'  0x44 */
 81:         _U | _X,        /* 'E'  0x45 */
 82:         _U | _X,        /* 'F'  0x46 */
 83:         _U,             /* 'G'  0x47 */
 84:         _U,             /* 'H'  0x48 */
 85:         _U,             /* 'I'  0x49 */
 86:         _U,             /* 'J'  0x4A */
 87:         _U,             /* 'K'  0x4B */
 88:         _U,             /* 'L'  0x4C */
 89:         _U,             /* 'M'  0x4D */
 90:         _U,             /* 'N'  0x4E */
 91:         _U,             /* 'O'  0x4F */
 92:         _U,             /* 'P'  0x50 */
 93:         _U,             /* 'Q'  0x51 */
 94:         _U,             /* 'R'  0x52 */
 95:         _U,             /* 'S'  0x53 */
 96:         _U,             /* 'T'  0x54 */
 97:         _U,             /* 'U'  0x55 */
 98:         _U,             /* 'V'  0x56 */
 99:         _U,             /* 'W'  0x57 */
100:         _U,             /* 'X'  0x58 */
101:         _U,             /* 'Y'  0x59 */
102:         _U,             /* 'Z'  0x5A */
103:         _P,             /* '['  0x5B */
104:         _P,             /* '\'  0x5C */
105:         _P,             /* ']'  0x5D */
106:         _P,             /* '^'  0x5E */
107:         _P,             /* '_'  0x5F */
108:         _P,             /* '`'  0x60 */
109:         _L | _X,        /* 'a'  0x61 */
110:         _L | _X,        /* 'b'  0x62 */
111:         _L | _X,        /* 'c'  0x63 */
112:         _L | _X,        /* 'd'  0x64 */
113:         _L | _X,        /* 'e'  0x65 */
114:         _L | _X,        /* 'f'  0x66 */
115:         _L,             /* 'g'  0x67 */
116:         _L,             /* 'h'  0x68 */
117:         _L,             /* 'i'  0x69 */
118:         _L,             /* 'j'  0x6A */
119:         _L,             /* 'k'  0x6B */
120:         _L,             /* 'l'  0x6C */
121:         _L,             /* 'm'  0x6D */
122:         _L,             /* 'n'  0x6E */
123:         _L,             /* 'o'  0x6F */
124:         _L,             /* 'p'  0x70 */
125:         _L,             /* 'q'  0x71 */
126:         _L,             /* 'r'  0x72 */
127:         _L,             /* 's'  0x73 */
128:         _L,             /* 't'  0x74 */
129:         _L,             /* 'u'  0x75 */
130:         _L,             /* 'v'  0x76 */
131:         _L,             /* 'w'  0x77 */
132:         _L,             /* 'x'  0x78 */
133:         _L,             /* 'y'  0x79 */
134:         _L,             /* 'z'  0x7A */
```

lib/ctype.c


```
135:         _P,             /* '{'  0x7B */
136:         _P,             /* '|'  0x7C */
137:         _P,             /* '}'  0x7D */
138:         _P,             /* '~'  0x7E */
139:         _C,             /* DEL  0x7F */
140: };
```

lib/Makefile


```
 1: # fiwix/lib/Makefile
 2: #
 3: # Copyright 2018, Jordi Sanfeliu. All rights reserved.
 4: # Distributed under the terms of the Fiwix License.
 5: #
 6:
 7: .S.o:
 8:         $(CC) -traditional -I$(INCLUDE) -c -o $@ $<
 9: .c.o:
10:         $(CC) $(CFLAGS) -c -o $@ $<
11:
12: OBJS = ctype.o strings.o printk.o
13:
14: lib:    $(OBJS)
15:         $(LD) $(LDFLAGS) -r $(OBJS) -o lib.o
16:
17: clean:
18:         rm -f *.o
19:
```

lib/printk.c


```
/*
 * fiwix/lib/printk.c
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#include <fiwix/kernel.h>
#include <fiwix/tty.h>
#include <fiwix/stdio.h>
#include <fiwix/string.h>
#include <fiwix/stdarg.h>

#define LOG_BUF_LEN     4096
#define MAX_BUF         1024    /* printk() and sprintk() size limit */

static char log_buf[LOG_BUF_LEN];
static unsigned int log_start;

static void puts(char *buffer)
{
        struct tty *tty;
        unsigned short int count;
        char *b;

        /* for special debugging purposes only (X11, SVGAlib, ...)
        {
                struct inode dummy_i;
                memset_b(&dummy_i, 0, sizeof(struct inode));
                dummy_i.dev = dummy_i.rdev = 0x0600;            // /dev/lp0
                lp_write(&dummy_i, NULL, buffer, strlen(buffer));
        }
        */

        tty = get_tty(_syscondev);
        count = strlen(buffer);
        b = buffer;

        while(count--) {
                if(!tty) {
                        if(log_start < LOG_BUF_LEN) {
                                log_buf[log_start++] = *(b++);
                        }
                } else {
                        tty_queue_putchar(tty, &tty->write_q, *(b++));

                        /* kernel messages must be shown immediately */
                        tty->output(tty);
                }
        }
}

/*
 * format identifiers
 * ------------------------------------------------------
 *      %d      decimal conversion
 *      %u      unsigned decimal conversion
 *      %x      hexadecimal conversion (lower case)
 *      %X      hexadecimal conversion (upper case)
 *      %b      binary conversion
 *      %o      octal conversion
 *      %c      character
 *      %s      string
 *
 * flags
 * ------------------------------------------------------
 *      0       result is padded with zeros (e.g.: '%06d')
```

lib/printk.c


```
 *              (maximum value is 32)
 *      blank   result is padded with spaces (e.g.: '% 6d')
 *              (maximum value is 32)
 *      -       the numeric result is left-justified
 *              (default is right-justified)
 */
static void do_printk(char *buffer, const char *format, va_list args)
{
        char sw_neg, in_identifier, n_pad, lf;
        char ch_pad, basecase, c;
        char str[] = {
                NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
                NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
                NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
                NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
                NULL
        };
        char nullstr[7] = { '<', 'N', 'U', 'L', 'L', '>', '\0' };
        char *ptr_s, *p;
        int num, count;
        char simplechar;
        unsigned int unum, digit;

        sw_neg = in_identifier = n_pad = lf = 0;
        count = 0;
        basecase = 'A';
        ch_pad = ' ';
        p = NULL;

        /* assumes buffer has a maximum size of MAX_BUF */
        while((c = *(format++)) && count < MAX_BUF) {
                if((c != '%') && !in_identifier) {
                        *(buffer++) = c;
                        memset_b(str, NULL, 32);
                } else {
                        in_identifier = 1;
                        switch(c = *(format)) {
                                case 'd':
                                        num = va_arg(args, int);
                                        if(num < 0) {
                                                num *= -1;
                                                sw_neg = 1;
                                        }
                                        ptr_s = str;
                                        do {
                                                *(ptr_s++) = '0' + (num % 10);
                                        } while(num /= 10);
                                        if(lf) {
                                                p = ptr_s;
                                        } else {
                                                while(*ptr_s) {
                                                        ptr_s++;
                                                }
                                        }
                                        if(sw_neg) {
                                                sw_neg = 0;
                                                *(ptr_s++) = '-';
                                        }
                                        do {
                                                *(buffer++) = *(--ptr_s);
                                                count++;
                                        } while(ptr_s != str && count < MAX_BUF)
;
                                        if(lf) {
                                                while(*p && count < MAX_BUF) {
                                                        *(buffer++) = *(p++);
                                                        count++;
```

lib/printk.c


```
  134:                                                    }
  135:                                            }
  136:                                            format++;
  137:                                            ch_pad = ' ';
  138:                                            n_pad = 0;
  139:                                            in_identifier = 0;
  140:                                            lf = 0;
  141:                                            break;
  142:
  143:                                    case 'u':
  144:                                            unum = va_arg(args, unsigned int);
  145:                                            ptr_s = str;
  146:                                            do {
  147:                                                    *(ptr_s++) = '0' + (unum % 10);
  148:                                            } while(unum /= 10);
  149:                                            if(lf) {
  150:                                                    p = ptr_s;
  151:                                            } else {
  152:                                                    while(*ptr_s) {
  153:                                                            ptr_s++;
  154:                                                    }
  155:                                            }
  156:                                            do {
  157:                                                    *(buffer++) = *(--ptr_s);
  158:                                                    count++;
  159:                                            } while(ptr_s != str && count < MAX_BUF)
;
  160:                                            if(lf) {
  161:                                                    while(*p && count < MAX_BUF) {
  162:                                                            *(buffer++) = *(p++);
  163:                                                            count++;
  164:                                                    }
  165:                                            }
  166:                                            format++;
  167:                                            ch_pad = ' ';
  168:                                            n_pad = 0;
  169:                                            in_identifier = 0;
  170:                                            lf = 0;
  171:                                            break;
  172:
  173:                                    case 'x':
  174:                                            basecase = 'a';
  175:                                    case 'X':
  176:                                            unum = va_arg(args, unsigned int);
  177:                                            ptr_s = str;
  178:                                            do {
  179:                                                    *(ptr_s++) = (digit = (unum & 0x
0F)) > 9 ? basecase + digit - 10 : '0' + digit;
  180:                                            } while(unum /= 16);
  181:                                            if(lf) {
  182:                                                    p = ptr_s;
  183:                                            } else {
  184:                                                    while(*ptr_s) {
  185:                                                            ptr_s++;
  186:                                                    }
  187:                                            }
  188:                                            do {
  189:                                                    *(buffer++) = *(--ptr_s);
  190:                                                    count++;
  191:                                            } while(ptr_s != str && count < MAX_BUF)
;
  192:                                            if(lf) {
  193:                                                    while(*p && count < MAX_BUF) {
  194:                                                            *(buffer++) = *(p++);
  195:                                                            count++;
  196:                                                    }
  197:                                            }
```

lib/printk.c


```
  198:                                    format++;
  199:                                    ch_pad = ' ';
  200:                                    n_pad = 0;
  201:                                    in_identifier = 0;
  202:                                    lf = 0;
  203:                                    break;
  204:
  205:                            case 'b':
  206:                                    num = va_arg(args, unsigned int);
  207:                                    if(num < 0) {
  208:                                            num *= -1;
  209:                                    }
  210:                                    ptr_s = str;
  211:                                    do {
  212:                                            *(ptr_s++) = '0' + (num % 2);
  213:                                    } while(num /= 2);
  214:                                    if(lf) {
  215:                                            p = ptr_s;
  216:                                    } else {
  217:                                            while(*ptr_s) {
  218:                                                    ptr_s++;
  219:                                            }
  220:                                    }
  221:                                    do {
  222:                                            *(buffer++) = *(--ptr_s);
  223:                                            count++;
  224:                                    } while(ptr_s != str && count < MAX_BUF)
;
  225:                                    if(lf) {
  226:                                            while(*p && count < MAX_BUF) {
  227:                                                    *(buffer++) = *(p++);
  228:                                                    count++;
  229:                                            }
  230:                                    }
  231:                                    format++;
  232:                                    ch_pad = ' ';
  233:                                    n_pad = 0;
  234:                                    in_identifier = 0;
  235:                                    lf = 0;
  236:                                    break;
  237:
  238:                            case 'o':
  239:                                    num = va_arg(args, unsigned int);
  240:                                    if(num < 0) {
  241:                                            num *= -1;
  242:                                    }
  243:                                    ptr_s = str;
  244:                                    do {
  245:                                            *(ptr_s++) = '0' + (num % 8);
  246:                                    } while(num /= 8);
  247:                                    if(lf) {
  248:                                            p = ptr_s;
  249:                                    } else {
  250:                                            while(*ptr_s) {
  251:                                                    ptr_s++;
  252:                                            }
  253:                                    }
  254:                                    do {
  255:                                            *(buffer++) = *(--ptr_s);
  256:                                            count++;
  257:                                    } while(ptr_s != str && count < MAX_BUF)
;
  258:                                    if(lf) {
  259:                                            while(*p && count < MAX_BUF) {
  260:                                                    *(buffer++) = *(p++);
  261:                                                    count++;
  262:                                            }
```

lib/printk.c


```
263:                                            }
264:                                            format++;
265:                                            ch_pad = ' ';
266:                                            n_pad = 0;
267:                                            in_identifier = 0;
268:                                            lf = 0;
269:                                            break;
270:
271:                                    case 'c':
272:                                            simplechar = va_arg(args, int);
273:                                            *(buffer++) = simplechar;
274:                                            format++;
275:                                            in_identifier = 0;
276:                                            lf = 0;
277:                                            break;
278:
279:                                    case 's':
280:                                            num = 0;
281:                                            ptr_s = va_arg(args, char *);
282:                                            if(n_pad) {
283:                                                    num = n_pad - strlen(ptr_s);
284:                                                    if(num < 0) {
285:                                                            num *= -1;
286:                                                    }
287:                                            }
288:                                            /* if it's a NULL then show "<NULL>" */
289:                                            if(ptr_s == NULL) {
290:                                                    ptr_s = (char *)nullstr;
291:                                            }
292:                                            while((c = *(ptr_s++)) && count < MAX_BU
F) {
293:                                                    *(buffer++) = c;
294:                                                    count++;
295:                                            }
296:                                            while(num-- && count < MAX_BUF) {
297:                                                    *(buffer++) = ' ';
298:                                                    count++;
299:                                            }
300:                                            format++;
301:                                            n_pad = 0;
302:                                            in_identifier = 0;
303:                                            lf = 0;
304:                                            break;
305:
306:                                    case ' ':
307:                                            ch_pad = ' ';
308:                                            break;
309:
310:                                    case '0':
311:                                            if(!n_pad) {
312:                                                    ch_pad = '0';
313:                                            }
314:                                    case '1':
315:                                    case '2':
316:                                    case '3':
317:                                    case '4':
318:                                    case '5':
319:                                    case '6':
320:                                    case '7':
321:                                    case '8':
322:                                    case '9':
323:                                            n_pad = !n_pad ? c - '0': ((n_pad * 10)
+ (c - '0'));
324:                                            n_pad = n_pad > 32 ? 32 : n_pad;
325:                                            for(unum = 0; unum < n_pad; unum++) {
326:                                                    str[unum] = ch_pad;
327:                                            }
```

lib/printk.c


```
328:                                            break;
329:
330:                                    case '-':
331:                                            lf = 1;
332:                                            break;
333:                                    case '%':
334:                                            *(buffer++) = c;
335:                                            format++;
336:                                            in_identifier = 0;
337:                                            break;
338:                            }
339:                    }
340:                    count++;
341:            }
342:            *buffer = NULL;
343: }
344:
345: void register_console(void (*fn)(char *, unsigned int))
346: {
347:            (*fn)(log_buf, log_start);
348: }
349:
350: void printk(const char *format, ...)
351: {
352:            va_list args;
353:            char buffer[MAX_BUF];
354:
355:            va_start(args, format);
356:            do_printk(buffer, format, args);
357:            puts(buffer);
358:            va_end(args);
359: }
360:
361: int sprintk(char *buffer, const char *format, ...)
362: {
363:            va_list args;
364:
365:            va_start(args, format);
366:            do_printk(buffer, format, args);
367:            va_end(args);
368:            return strlen(buffer);
369: }
```

lib/strings.c


```
 1: /*
 2:  * fiwix/lib/strings.c
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9: #include <fiwix/tty.h>
10: #include <fiwix/mm.h>
11: #include <fiwix/stdio.h>
12: #include <fiwix/string.h>
13:
14: /* convert from big-endian to little-endian (word swap) */
15: void swap_asc_word(char *str, int len)
16: {
17:         int n, n2;
18:         short int *ptr;
19:         char *buf;
20:
21:         if(!(buf = (void *)kmalloc())) {
22:                 return;
23:         }
24:
25:         ptr = (short int *)str;
26:
27:         for(n = 0, n2 = 0; n < len; n++) {
28:                 buf[n2++] = *ptr >> 8;
29:                 buf[n2++] = *ptr & 0xFF;
30:                 ptr++;
31:         }
32:         for(n = len - 1; n > 0; n--) {
33:                 if(buf[n] == NULL || buf[n] == ' ') {
34:                         buf[n] = NULL;
35:                 } else {
36:                         break;
37:                 }
38:         }
39:         memcpy_b(str, buf, len);
40:         kfree((unsigned int)buf);
41: }
42:
43: int strcmp(const char *str1, const char *str2)
44: {
45:         while(*str1) {
46:                 if(*str1 != *str2) {
47:                         return 1;
48:                 }
49:                 str1++;
50:                 str2++;
51:         }
52:         if(!(*str2)) {
53:                 return 0;
54:         }
55:         return 1;
56: }
57:
58: int strncmp(const char *str1, const char *str2, __ssize_t n)
59: {
60:         while(n > 0) {
61:                 if(*str1 != *str2) {
62:                         return 1;
63:                 }
64:                 str1++;
65:                 str2++;
66:                 n--;
67:         }
```

lib/strings.c


```
 68:         return 0;
 69: }
 70:
 71: char *strcpy(char *dest, const char *src)
 72: {
 73:         if(!dest || !src) {
 74:                 return NULL;
 75:         }
 76:
 77:         while(*src) {
 78:                 *dest = *src;
 79:                 dest++;
 80:                 src++;
 81:         }
 82:         *dest = NULL;           /* NULL-terminated */
 83:         return dest;
 84: }
 85:
 86: void strncpy(char *dest, const char *src, int len)
 87: {
 88:         if(!dest || !src) {
 89:                 return;
 90:         }
 91:
 92:         while((*src) && len) {
 93:                 *dest = *src;
 94:                 dest++;
 95:                 src++;
 96:                 len--;
 97:         }
 98:         *dest = NULL;           /* NULL-terminated */
 99: }
100:
101: char *strcat(char *dest, const char *src)
102: {
103:         char *orig;
104:
105:         orig = dest;
106:         while(*dest) {
107:                 dest++;
108:         }
109:         while(*src) {
110:                 *dest = *src;
111:                 dest++;
112:                 src++;
113:         }
114:         *dest = NULL;
115:         return orig;
116: }
117:
118: char *strncat(char *dest, const char *src, __ssize_t len)
119: {
120:         char *orig;
121:
122:         orig = dest;
123:         while(*dest) {
124:                 dest++;
125:         }
126:         while(*src && len) {
127:                 *dest = *src;
128:                 dest++;
129:                 src++;
130:                 len--;
131:         }
132:         *dest = NULL;
133:         return orig;
134: }
```

lib/strings.c


```
135:
136: int strlen(const char *str)
137: {
138:         int n;
139:
140:         n = 0;
141:         while(str && *str) {
142:                 n++;
143:                 str++;
144:         }
145:         return n;
146: }
147:
148: char * get_basename(const char *path)
149: {
150:         char *basename;
151:         char c;
152:
153:         basename = NULL;
154:
155:         while(path) {
156:                 while(*path == '/') {
157:                         path++;
158:                 }
159:                 if(*path != NULL) {
160:                         basename = (char *)path;
161:                 }
162:                 while((c = *(path++)) && (c != '/'));
163:                 if(!c) {
164:                         break;
165:                 }
166:         }
167:         return basename;
168: }
169:
170: char * remove_trailing_slash(char *path)
171: {
172:         char *p;
173:
174:         p = path + (strlen(path) - 1);
175:         while(p > path && *p == '/') {
176:                 *p = NULL;
177:                 p--;
178:         }
179:         return path;
180: }
181:
182: int is_dir(const char *path)
183: {
184:         while(*(path + 1)) {
185:                 path++;
186:         }
187:         if(*path == '/') {
188:                 return 1;
189:         }
190:         return 0;
191: }
192:
193: int atoi(const char *str)
194: {
195:         int n;
196:
197:         n = 0;
198:         while(IS_SPACE(*str)) {
199:                 str++;
200:         }
201:         while(IS_NUMERIC(*str)) {
```

lib/strings.c


```
202:                 n = (n * 10) + (*str++ - '0');
203:         }
204:         return n;
205: }
206:
207: void memcpy_b(void *dest, const void *src, unsigned int count)
208: {
209:         unsigned char *d;
210:         unsigned char *s;
211:
212:         d = (unsigned char *)dest;
213:         s = (unsigned char *)src;
214:         while(count--) {
215:                 *d = *s;
216:                 d++;
217:                 s++;
218:         }
219: }
220:
221: void memcpy_w(void *dest, const void *src, unsigned int count)
222: {
223:         unsigned short int *d;
224:         unsigned short int *s;
225:
226:         d = (unsigned short int *)dest;
227:         s = (unsigned short int *)src;
228:         while(count--) {
229:                 *d = *s;
230:                 d++;
231:                 s++;
232:         }
233: }
234:
235: void memcpy_l(void *dest, const void *src, unsigned int count)
236: {
237:         unsigned int *d;
238:         unsigned int *s;
239:
240:         d = (unsigned int *)dest;
241:         s = (unsigned int *)src;
242:         while(count--) {
243:                 *d = *s;
244:                 d++;
245:                 s++;
246:         }
247: }
248:
249: void memset_b(void *dest, unsigned char value, unsigned int count)
250: {
251:         unsigned char *d;
252:
253:         d = (unsigned char *)dest;
254:         while(count--) {
255:                 *d = value;
256:                 d++;
257:         }
258: }
259:
260: void memset_w(void *dest, unsigned short int value, unsigned int count)
261: {
262:         unsigned short int *d;
263:
264:         d = (unsigned short int *)dest;
265:         while(count--) {
266:                 *d = value;
267:                 d++;
268:         }
```

lib/strings.c


```
269: }
270:
271: void memset_l(void *dest, unsigned int value, unsigned int count)
272: {
273:         unsigned int *d;
274:
275:         d = (unsigned int *)dest;
276:         while(count--) {
277:                 *d = value;
278:                 d++;
279:         }
280: }
```

include/fiwix/asm.h


```
 1: /*
 2:  * fiwix/include/fiwix/asm.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_ASM_H
 9: #define _FIWIX_ASM_H
10:
11: extern void except0(void);
12: extern void except1(void);
13: extern void except2(void);
14: extern void except3(void);
15: extern void except4(void);
16: extern void except5(void);
17: extern void except6(void);
18: extern void except7(void);
19: extern void except8(void);
20: extern void except9(void);
21: extern void exceptA(void);
22: extern void exceptB(void);
23: extern void exceptC(void);
24: extern void exceptD(void);
25: extern void exceptE(void);
26: extern void exceptF(void);
27: extern void except10(void);
28: extern void except11(void);
29: extern void except12(void);
30: extern void except13(void);
31: extern void except14(void);
32: extern void except15(void);
33: extern void except16(void);
34: extern void except17(void);
35: extern void except18(void);
36: extern void except19(void);
37: extern void except1A(void);
38: extern void except1B(void);
39: extern void except1C(void);
40: extern void except1D(void);
41: extern void except1E(void);
42: extern void except1F(void);
43:
44: extern void irq0(void);
45: extern void irq1(void);
46: extern void irq2(void);
47: extern void irq3(void);
48: extern void irq4(void);
49: extern void irq5(void);
50: extern void irq6(void);
51: extern void irq7(void);
52: extern void irq8(void);
53: extern void irq9(void);
54: extern void irq10(void);
55: extern void irq11(void);
56: extern void irq12(void);
57: extern void irq13(void);
58: extern void irq14(void);
59: extern void irq15(void);
60: extern void unknown_irq(void);
61:
62: extern void switch_to_user_mode(void);
63: extern void sighandler_trampoline(void);
64: extern void end_sighandler_trampoline(void);
65: extern void syscall(void);
66: extern void return_from_syscall(void);
67: extern void do_switch(unsigned int *, unsigned int *, unsigned int, unsigned int
```

include/fiwix/asm.h


```
, unsigned int, unsigned short int);
   68:
   69: int cpuid(void);
   70: int getfpu(void);
   71: int vendor_id(void);
   72: int signature_flags(void);
   73: int brand_str(void);
   74: int tlbinfo(void);
   75:
   76: unsigned char inport_b(unsigned int);
   77: short int inport_w(unsigned int);
   78: void inport_sw(unsigned int, void *, unsigned int);
   79: void outport_b(unsigned int, unsigned char);
   80: void outport_w(unsigned int, unsigned int);
   81: void outport_sw(unsigned int, void *, unsigned int);
   82:
   83: void load_gdt(unsigned int);
   84: void load_idt(unsigned int);
   85: void activate_kpage_dir(void);
   86: void load_tr(unsigned int);
   87: unsigned long long int get_rdtsc(void);
   88: void invalidate_tlb(void);
   89:
   90: #define CLI() __asm__ __volatile__ ("cli":::"memory")
   91: #define STI() __asm__ __volatile__ ("sti":::"memory")
   92: #define NOP() __asm__ __volatile__ ("nop":::"memory")
   93: #define HLT() __asm__ __volatile__ ("hlt":::"memory")
   94:
   95: #define GET_CR2(cr2) __asm__ __volatile__ ("movl %%cr2, %0" : "=r" (cr2));
   96: #define GET_ESP(esp) __asm__ __volatile__ ("movl %%esp, %0" : "=r" (esp));
   97: #define SET_ESP(esp) __asm__ __volatile__ ("movl %0, %%esp" :: "r" (esp));
   98:
   99: #define SAVE_FLAGS(flags)                       \
  100:         __asm__ __volatile__(                   \
  101:                 "pushfl ; popl %0\n\t"          \
  102:                 : "=r" (flags)                  \
  103:                 : /* no input */                \
  104:                 : "memory"                      \
  105:         );
  106:
  107: #define RESTORE_FLAGS(x)                        \
  108:         __asm__ __volatile__(                   \
  109:                 "pushl %0 ; popfl\n\t"          \
  110:                 : /* no output */               \
  111:                 : "r" (flags)                   \
  112:                 : "memory"                      \
  113:         );
  114:
  115: #define USER_SYSCALL(num, arg1, arg2, arg3)     \
  116:         __asm__ __volatile__(                   \
  117:                 "movl   %0, %%eax\n\t"          \
  118:                 "movl   %1, %%ebx\n\t"          \
  119:                 "movl   %2, %%ecx\n\t"          \
  120:                 "movl   %3, %%edx\n\t"          \
  121:                 "int    $0x80\n\t"              \
  122:                 : /* no output */               \
  123:                 : "eax"((unsigned int)num), "ebx"((unsigned int)arg1), "ecx"((un
signed int)arg2), "edx"((unsigned int)arg3)     \
  124:         );
  125:
  126: /*
  127: static inline unsigned long long int get_rdtsc(void)
  128: {
  129:         unsigned int eax, edx;
  130:
  131:         __asm__ __volatile__("rdtsc" : "=a" (eax), "=d" (edx));
  132:         return ((unsigned long long int)eax) | (((unsigned long long int)edx) <<
```

include/fiwix/asm.h


```
32);
 133: }
 134: */
 135:
 136: #endif /* _FIWIX_ASM_H */
```

include/fiwix/bios.h


```
 1: /*
 2:  * fiwix/include/fiwix/bios.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_BIOS_H
 9: #define _FIWIX_BIOS_H
10:
11: #include <fiwix/multiboot.h>
12:
13: #define BIOS_MEM_AVAIL          1       /* BIOS memory available */
14: #define BIOS_MEM_RES            2       /* BIOS memory reserved */
15: #define BIOS_MEM_ACPI_REC       3       /* BIOS memory ACPI reclaim */
16: #define BIOS_MEM_ACPI_NVS       4       /* BIOS memory ACPI NVS */
17: #define NR_BIOS_MM_ENT          25      /* entries in BIOS memory map */
18:
19: struct bios_mem_map {
20:         unsigned long int from;
21:         unsigned long int to;
22:         unsigned long int type;
23: };
24: struct bios_mem_map bios_mem_map[NR_BIOS_MM_ENT];
25:
26: int addr_in_bios_map(unsigned int);
27: void bios_map_init(memory_map_t *, unsigned long int);
28:
29: #endif /* _FIWIX_BIOS_H */
```

include/fiwix/buffer.h


```
 1: /*
 2:  * fiwix/include/fiwix/buffer.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_BUFFER_H
 9: #define _FIWIX_BUFFER_H
10:
11: #include <fiwix/types.h>
12: #include <fiwix/fs.h>
13:
14: struct buffer {
15:         __dev_t dev;                    /* device number */
16:         __blk_t block;                  /* block number */
17:         int size;                       /* block size (in bytes) */
18:         char valid;                     /* 1 = valid */
19:         unsigned char locked;           /* 1 = locked */
20:         unsigned char dirty;            /* 1 = delayed write */
21:         char *data;                     /* block contents */
22:         struct buffer *prev_hash;
23:         struct buffer *next_hash;
24:         struct buffer *prev_free;
25:         struct buffer *next_free;
26: };
27: extern struct buffer *buffer_table;
28: extern struct buffer **buffer_hash_table;
29:
30: /* values to be determined during system startup */
31: extern unsigned int buffer_table_size;          /* size in bytes */
32: extern unsigned int buffer_hash_table_size;     /* size in bytes */
33:
34: struct buffer * get_dirty_buffer(__dev_t, __blk_t, int);
35: struct buffer * bread(__dev_t, __blk_t, int);
36: void bwrite(struct buffer *);
37: void brelse(struct buffer *);
38: void sync_buffers(__dev_t);
39: void invalidate_buffers(__dev_t);
40: int reclaim_buffers(void);
41: void buffer_init(void);
42:
43: #endif /* _FIWIX_BUFFER_H */
```

include/fiwix/cmos.h


```
 1: /*
 2:  * fiwix/include/fiwix/cmos.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_CMOS_H
 9: #define _FIWIX_CMOS_H
10:
11: #define CMOS_INDEX      0x70
12: #define CMOS_DATA       0x71
13:
14: #define CMOS_STATA_IRQF 0x0F    /* periodic interrupt frequency */
15: #define CMOS_STATA_UIP  0x80    /* time update in progress */
16:
17: #define CMOS_STATB_DSE  0x01    /* enable daylight savings */
18: #define CMOS_STATB_24H  0x02    /* 24-hour mode (0=12h, 1=24h) */
19: #define CMOS_STATB_DM   0x04    /* time/date in binary mode (0=BCD, 1=binary) */
20: #define CMOS_STATB_SQWE 0x08    /* enable square wave frequency */
21: #define CMOS_STATB_UIE  0x10    /* enable update-ended interrupt */
22: #define CMOS_STATB_AIE  0x20    /* enable alarm interrupt */
23: #define CMOS_STATB_PIE  0x40    /* enable periodic interrupt */
24: #define CMOS_STATB_SET  0x80    /* abort clock update */
25:
26: #define CMOS_STATD_VRT  0x80    /* valid RAM and time */
27:
28: /* CMOS RAM data registers */
29: #define CMOS_SEC        0x00    /* second */
30: #define CMOS_ASEC       0x01    /* alarm second */
31: #define CMOS_MIN        0x02    /* minute */
32: #define CMOS_AMIN       0x03    /* alarm minute */
33: #define CMOS_HOUR       0x04    /* hour */
34: #define CMOS_AHOUR      0x05    /* alarm hour */
35: #define CMOS_DOW        0x06    /* day of week */
36: #define CMOS_DAY        0x07    /* day */
37: #define CMOS_MONTH      0x08    /* month */
38: #define CMOS_YEAR       0x09    /* last two digits of year */
39: #define CMOS_STATA      0x0A    /* status register A */
40: #define CMOS_STATB      0x0B    /* status register B */
41: #define CMOS_STATC      0x0C    /* status register C */
42: #define CMOS_STATD      0x0D    /* status register D */
43: #define CMOS_DIAG       0x0E    /* diagnostics status */
44: #define CMOS_FDDTYPE    0x10    /* floppy disk drive type */
45: #define CMOS_HDDTYPE    0x12    /* hard disk drive type */
46: #define CMOS_CENTURY    0x32    /* century */
47:
48: /* conversions */
49: #define BCD2BIN(bcd)    (((bcd) >> 4) * 10) + ((bcd) & 0x0F)
50: #define BIN2BCD(bin)    ((bin) % 10) | (((bin) / 10) << 4)
51:
52: int cmos_update_in_progress(void);
53: unsigned char cmos_read_date(unsigned char);
54: void cmos_write_date(unsigned char, unsigned char);
55: unsigned char cmos_read(unsigned char);
56: void cmos_write(unsigned char, unsigned char);
57:
58: #endif /* _FIWIX_CMOS_H */
```

include/fiwix/config.h


```
/*
 * fiwix/include/fiwix/config.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_CONFIG_H
#define _FIWIX_CONFIG_H

/* maximum number of processes */
#define NR_PROCS                64

/* maximum number of callout functions (timer) */
#define NR_CALLOUTS             NR_PROCS

/* maximum number of bottom halves in pool */
#define NR_BH                   NR_PROCS

/* maximum number of mounted filesystems */
#define NR_MOUNT_POINTS         8

/* maximum number of opened files in system */
#define NR_OPENS                1024

/* maximum number of flocks in system */
#define NR_FLOCKS               (NR_PROCS * 5)



/* percentage of memory that buffer cache will borrow from available memory */
#define BUFFER_PERCENTAGE       100

/* percentage of hash buckets relative to the size of the buffer table */
#define BUFFER_HASH_PERCENTAGE  10

/* buffers reclaimed in a single call */
#define NR_BUF_RECLAIM          150


/* percentage of memory assigned to the inode table and hash table */
#define INODE_PERCENTAGE        5
#define INODE_HASH_PERCENTAGE   10

/* percentage of memory assigned to the page hash table */
#define PAGE_HASH_PERCENTAGE    10


/* maximum value for PID */
#define MAX_PID_VALUE           32767

/* number of screens in console' scroll back */
#define SCREENS_LOG             6

/* maximum number of messages on spurious interrupts */
#define MAX_SPU_NOTICES         10

/* maximum number of virtual memory mappings (regions) */
#define VMA_REGIONS             150

#endif /* _FIWIX_CONFIG_H */
```

include/fiwix/console.h


```
/*
 * fiwix/include/fiwix/console.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_CONSOLE_H
#define _FIWIX_CONSOLE_H

#include <fiwix/config.h>
#include <fiwix/termios.h>
#include <fiwix/vt.h>

#define NR_VCONSOLES            12      /* number of virtual consoles */

#define VCONSOLES_MAJOR         4       /* virtual consoles major number */
#define SYSCON_MAJOR            5       /* system console major number */

#define MONO_ADDR               0xB0000L
#define COLOR_ADDR              0xB8000L

#define MONO_6845_ADDR          0x3B4   /* i/o address (+1 for data register) */
#define COLOR_6845_ADDR         0x3D4   /* i/o address (+1 for data register) */

#define ATTR_CONTROLLER         0x3C0   /* attribute controller registrer */
#define ATTR_CONTROLLER_PAS     0x20    /* palette address source */
#define INPUT_STAT1             0x3DA   /* input status #1 register */
#define BLANK_INTERVAL  (600 * HZ)      /* 600 seconds (10 minutes) */

#define CRT_INDEX               0
#define CRT_DATA                1
#define CRT_CURSOR_STR          0xA
#define CRT_CURSOR_END          0xB
#define CRT_START_ADDR_HI       0xC
#define CRT_START_ADDR_LO       0xD
#define CRT_CURSOR_POS_HI       0xE
#define CRT_CURSOR_POS_LO       0xF

#define CURSOR_MASK             0x1F
#define CURSOR_DISABLE          0x20

#define COLOR_NORMAL            0
#define COLOR_BOLD              1
#define COLOR_BOLD_OFF          2
#define COLOR_UNDERLINE         4
#define COLOR_BLINK             5
#define COLOR_REVERSE           7

#define COLOR_BLACK             0x0000
#define COLOR_BLUE              0x0100
#define COLOR_GREEN             0x0200
#define COLOR_CYAN              0x0300
#define COLOR_RED               0x0400
#define COLOR_MAGENTA           0x0500
#define COLOR_BROWN             0x0600
#define COLOR_WHITE             0x0700
#define BG_BLACK                0x0000
#define BG_BLUE                 0x1000
#define BG_GREEN                0x2000
#define BG_CYAN                 0x3000
#define BG_RED                  0x4000
#define BG_MAGENTA              0x5000
#define BG_BROWN                0x6000
#define BG_WHITE                0x7000

#define DEF_MODE                (COLOR_WHITE | BG_BLACK)
```

include/fiwix/console.h


```
 68: #define BLANK_MEM              (DEF_MODE | ' ')
 69:
 70: #define SCREEN_COLS            80
 71: #define SCREEN_LINES           25
 72: #define SCREEN_SIZE            (SCREEN_COLS * SCREEN_LINES * 2)
 73:
 74: #define TAB_SIZE               8
 75: #define BS                     127     /* backspace */
 76:
 77: #define MAX_TAB_COLS           132     /* maximum number of tab stops */
 78:
 79: #define VC_BUF_LINES           (SCREEN_LINES * SCREENS_LOG)
 80: #define VC_BUF_SIZE            (SCREEN_COLS * VC_BUF_LINES * 2)
 81: #define VC_BUF_UP              1
 82: #define VC_BUF_DOWN            2
 83:
 84: unsigned int video_port;
 85: extern short int current_cons;  /* current console (/dev/tty1 ... /dev/tty12) */
 86:
 87: struct vconsole {
 88:         int x;          /* current column */
 89:         int y;          /* current line */
 90:         int top, bottom, columns;
 91:         short int check_x;
 92:         unsigned char led_status;
 93:         unsigned char scrlock, numlock, capslock;
 94:         unsigned char esc, sbracket, semicolon, question;
 95:         int parmv1, parmv2;
 96:         unsigned short int color_attr;
 97:         unsigned char bold, underline, blink, reverse;
 98:         int insert_mode;
 99:         unsigned short int *vidmem;
100:         short int has_focus;
101:         unsigned short int scrbuf[SCREEN_SIZE / 2];
102:         int saved_x;
103:         int saved_y;
104:         char tab_stop[MAX_TAB_COLS];
105:         struct vt_mode vt_mode;
106:         unsigned char vc_mode;
107:         unsigned char blanked;
108:         int switchto_tty;
109:         struct tty *tty;
110: };
111:
112: void vconsole_reset(struct tty *);
113: void vconsole_write(struct tty *);
114: void vconsole_select(int);
115: void vconsole_select_final(int);
116: void vconsole_save(struct vconsole *);
117: void vconsole_restore(struct vconsole *);
118: void vconsole_buffer_scrl(int);
119: void blank_screen(struct vconsole *);
120: void unblank_screen(struct vconsole *);
121: void screen_on(void);
122: void screen_off(unsigned int);
123: void vconsole_start(struct tty *);
124: void vconsole_stop(struct tty *);
125: void vconsole_beep(void);
126: void vconsole_deltab(struct tty *);
127: void console_flush_log_buf(char *, unsigned int);
128: void vconsole_init(void);
129:
130: #endif /* _FIWIX_CONSOLE_H */
```

include/fiwix/const.h


```
 1: /*
 2:  * fiwix/include/fiwix/const.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_CONST_H
 9: #define _FIWIX_CONST_H
10:
11: #define KERNEL_BASE_ADDR        0xC0000000
12: #define KERNEL_ENTRY_ADDR       0x100000
13:
14: #define KERNEL_CS       0x08    /* kernel code segment */
15: #define KERNEL_DS       0x10    /* kernel data segment */
16: #define USER_CS         0x18    /* user code segment */
17: #define USER_DS         0x20    /* user data segment */
18: #define TSS             0x28    /* TSS segment */
19:
20: #endif /* _FIWIX_CONST_H */
```

include/fiwix/cpu.h


```
/*
 * fiwix/include/fiwix/cpu.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_CPU_H
#define _FIWIX_CPU_H

#define CPU_FPU         0x00000001      /* Floating-Point Unit on chip */
#define CPU_VME         0x00000002      /* Virtual 8086 Mode Enhancements */
#define CPU_DE          0x00000004      /* Debugging Extensions */
#define CPU_PSE         0x00000008      /* Page Size Extension */
#define CPU_TSC         0x00000010      /* Time Stamp Counter */
#define CPU_MSR         0x00000020      /* Model Specific Registers */
#define CPU_PAE         0x00000040      /* Physical Address Extension */
#define CPU_MCE         0x00000080      /* Machine Check Exception */
#define CPU_CX8         0x00000100      /* CMPXCHG8B instruction supported */
#define CPU_APIC        0x00000200      /* On-chip APIC hardware supported */
#define CPU_RES10       0x00000400      /* Reserved */
#define CPU_SEP         0x00000800      /* Fast System Call */
#define CPU_MTRR        0x00001000      /* Memory Type Range Registers */
#define CPU_PGE         0x00002000      /* Page Global Enable */
#define CPU_MCA         0x00004000      /* Machine Check Architecture */
#define CPU_CMOV        0x00008000      /* Conditional Move Instruction */
#define CPU_PAT         0x00010000      /* Page Attribute Table */
#define CPU_PSE36       0x00020000      /* 36-bit Page Size Extension */
#define CPU_PSN         0x00040000      /* Processor Serial Number */
#define CPU_CLFSH       0x00080000      /* CLFLUSH instruction supported */
#define CPU_RES20       0x00100000      /* Reserved */
#define CPU_DS          0x00200000      /* Debug Store */
#define CPU_ACPI        0x00400000      /* Thermal Monitor and others */
#define CPU_MMX         0x00800000      /* Intel Architecture MMX Technology */
#define CPU_FXSR        0x01000000      /* Fast Floating Point Save and Rest. */
#define CPU_SSE         0x02000000      /* Streaming SIMD Extensions */
#define CPU_SSE2        0x04000000      /* Streaming SIMD Extensions 2 */
#define CPU_SS          0x08000000      /* Self-Snoop */
#define CPU_HTT         0x10000000      /* Hyper-Threading Technology */
#define CPU_TM          0x20000000      /* Thermal Monitor */
#define CPU_RES30       0x40000000      /* Reserved */
#define CPU_PBE         0x80000000      /* Pending Break Enable */

#define RESERVED_DESC   0x80000000      /* TLB descriptor reserved */

struct cpu {
        char *vendor_id;
        char family;
        char model;
        char *model_name;
        char stepping;
        unsigned long int hz;
        char *cache;
        char has_cpuid;
        char has_fpu;
        int flags;
};
struct cpu cpu_table;

struct cpu_type {
        int cpu;
        char *name[20];
};

int get_cpu_flags(char *, int);
void cpu_init(void);

```

include/fiwix/cpu.h


```
68: #endif /* _FIWIX_CPU_H */
```

include/fiwix/ctype.h


```
 1: /*
 2:  * fiwix/include/fiwix/ctype.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_CTYPE_H
 9: #define _FIWIX_CTYPE_H
10:
11: #define _U      0x01    /* upper case */
12: #define _L      0x02    /* lower case */
13: #define _N      0x04    /* numeral (digit) */
14: #define _S      0x08    /* spacing character */
15: #define _P      0x10    /* punctuation */
16: #define _C      0x20    /* control character */
17: #define _X      0x40    /* hexadecimal */
18: #define _B      0x80    /* blank */
19:
20: extern unsigned char _ctype[];
21:
22: #define ISALPHA(ch)     ((_ctype + 1)[ch] & (_U | _L))
23: #define ISUPPER(ch)     ((_ctype + 1)[ch] & _U)
24: #define ISLOWER(ch)     ((_ctype + 1)[ch] & _L)
25: #define ISDIGIT(ch)     ((_ctype + 1)[ch] & _N)
26: #define ISALNUM(ch)     ((_ctype + 1)[ch] & (_U | _L | _N))
27: #define ISSPACE(ch)     ((_ctype + 1)[ch] & _S)
28: #define ISPUNCT(ch)     ((_ctype + 1)[ch] & _P)
29: #define ISCNTRL(ch)     ((_ctype + 1)[ch] & _C)
30: #define ISXDIGIT(ch)    ((_ctype + 1)[ch] & (_N | _X))
31:
32: #define ISASCII(ch)     ((unsigned) ch <= 0x7F)
33: #define TOASCII(ch)     ((unsigned) ch & 0x7F)
34:
35: #define TOUPPER(ch)     ((ch) & ~32)
36: #define TOLOWER(ch)     ((ch) | 32)
37:
38: #endif /* _FIWIX_CTYPE_H */
```

include/fiwix/devices.h


```
 1: /*
 2:  * fiwix/include/fiwix/devices.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_DEVICES_H
 9: #define _FIWIX_DEVICES_H
10:
11: #include <fiwix/types.h>
12: #include <fiwix/fs.h>
13:
14: #define NR_BLKDEV       128     /* maximum number of block devices */
15: #define NR_CHRDEV       128     /* maximum number of char devices */
16:
17: #define BLK_DEV         1       /* block device */
18: #define CHR_DEV         2       /* character device */
19:
20: #define SET_MINOR(minors, bit)   ((minors[(bit) / 32]) |= (1 << ((bit) % 32)))
21: #define CLEAR_MINOR(minors, bit) ((minors[(bit) / 32]) &= ~(1 << ((bit) % 32)))
22: #define TEST_MINOR(minors, bit)  ((minors[(bit) / 32]) & (1 << ((bit) % 32)))
23:
24: struct device {
25:         char *name;
26:         int irq;
27:         unsigned char major;
28:         unsigned int minors[8];         /* bitmap of 256 bits */
29:         int blksize;
30:         void *device_data;              /* mostly used for minor sizes in KB */
31:         struct fs_operations *fsop;
32: };
33:
34: extern struct device chr_device_table[NR_CHRDEV];
35: extern struct device blk_device_table[NR_BLKDEV];
36:
37: int register_device(int, struct device *);
38: struct device * get_device(int, unsigned char);
39: int chr_dev_open(struct inode *, struct fd *);
40: int blk_dev_open(struct inode *, struct fd *);
41: int blk_dev_close(struct inode *, struct fd *);
42: int blk_dev_read(struct inode *, struct fd *, char *, __size_t);
43: int blk_dev_write(struct inode *, struct fd *, const char *, __size_t);
44: int blk_dev_ioctl(struct inode *, int, unsigned long int);
45: int blk_dev_lseek(struct inode *, __off_t);
46:
47: void dev_init(void);
48:
49: #endif /* _FIWIX_DEVICES_H */
```

include/fiwix/dirent.h


```
 1: /*
 2:  * fiwix/include/fiwix/dirent.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_DIRENT_H
 9: #define _FIWIX_DIRENT_H
10:
11: #include <fiwix/types.h>
12: #include <fiwix/limits.h>
13:
14: struct dirent {
15:         __ino_t d_ino;                  /* inode number */
16:         __off_t d_off;                  /* offset to next dirent */
17:         unsigned short int d_reclen;    /* length of this dirent */
18:         char d_name[NAME_MAX + 1];      /* file name (null-terminated) */
19: };
20:
21: #endif /* _FIWIX_DIRENT_H */
```

include/fiwix/dma.h


```
 1: /*
 2:  * fiwix/include/fiwix/dma.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_DMA_H
 9: #define _FIWIX_DMA_H
10:
11: #define DMA_CHANNELS            8       /* max. number of DMA channels */
12:
13: #define DMA_MASK_CHANNEL        0x04
14: #define DMA_UNMASK_CHANNEL      0x00
15:
16: #define DMA_MODE_VERIFY         0x00
17: #define DMA_MODE_WRITE          0x04    /* read device -> write memory */
18: #define DMA_MODE_READ           0x08    /* read memory -> write device */
19: #define DMA_MODE_AUTOINIT       0x10
20: #define DMA_MODE_ADDRES_DEC     0x20
21: #define DMA_MODE_DEMAND         0x00
22: #define DMA_MODE_SINGLE         0x40
23: #define DMA_MODE_BLOCK          0x80
24: #define DMA_MODE_CASCADE        0xC0
25:
26: char *dma_resources[DMA_CHANNELS];
27:
28: void start_dma(int, void *, unsigned int, int);
29: int dma_register(int, char *);
30: int dma_unregister(int);
31: void dma_init(void);
32:
33: #endif /* _FIWIX_DMA_H */
```

include/fiwix/errno.h


```
  1: #ifndef _FIWIX_ERRNO_H
  2: #define _FIWIX_ERRNO_H
  3:
  4: #define EPERM           1       /* Operation not permitted - Not owner  */
  5: #define ENOENT          2       /* No such file or directory            */
  6: #define ESRCH           3       /* No such process                      */
  7: #define EINTR           4       /* Interrupted system call              */
  8: #define EIO             5       /* I/O error                            */
  9: #define ENXIO           6       /* No such device or address            */
 10: #define E2BIG           7       /* Arg list too long                    */
 11: #define ENOEXEC         8       /* Exec format error                    */
 12: #define EBADF           9       /* Bad file number                      */
 13: #define ECHILD          10      /* No child processes                   */
 14: #define EAGAIN          11      /* Try again - No more processes        */
 15: #define ENOMEM          12      /* Out of memory - No enough space      */
 16: #define EACCES          13      /* Permission denied                    */
 17: #define EFAULT          14      /* Bad address                          */
 18: #define ENOTBLK         15      /* Block device required                */
 19: #define EBUSY           16      /* Device or resource busy              */
 20: #define EEXIST          17      /* File exists                          */
 21: #define EXDEV           18      /* Cross-device link                    */
 22: #define ENODEV          19      /* No such device                       */
 23: #define ENOTDIR         20      /* Not a directory                      */
 24: #define EISDIR          21      /* Is a directory                       */
 25: #define EINVAL          22      /* Invalid argument                     */
 26: #define ENFILE          23      /* File table overflow                  */
 27: #define EMFILE          24      /* Too many open files                  */
 28: #define ENOTTY          25      /* Not a typewriter                     */
 29: #define ETXTBSY         26      /* Text file busy                       */
 30: #define EFBIG           27      /* File too large                       */
 31: #define ENOSPC          28      /* No space left on device              */
 32: #define ESPIPE          29      /* Illegal seek                         */
 33: #define EROFS           30      /* Read-only file system                */
 34: #define EMLINK          31      /* Too many links                       */
 35: #define EPIPE           32      /* Broken pipe                          */
 36: #define EDOM            33      /* Math argument out of domain of func  */
 37: #define ERANGE          34      /* Math result not representable        */
 38: #define EDEADLK         35      /* Resource deadlock would occur        */
 39: #define ENAMETOOLONG    36      /* File name too long                   */
 40: #define ENOLCK          37      /* No record locks available            */
 41: #define ENOSYS          38      /* Function not implemented             */
 42: #define ENOTEMPTY       39      /* Directory not empty                  */
 43: #define ELOOP           40      /* Too many symbolic links encountered  */
 44: #define EWOULDBLOCK     EAGAIN  /* Operation would block                */
 45: #define ENOMSG          42      /* No message of desired type           */
 46: #define EIDRM           43      /* Identifier removed                   */
 47: #define ECHRNG          44      /* Channel number out of range          */
 48: #define EL2NSYNC        45      /* Level 2 not synchronized             */
 49: #define EL3HLT          46      /* Level 3 halted                       */
 50: #define EL3RST          47      /* Level 3 reset                        */
 51: #define ELNRNG          48      /* Link number out of range             */
 52: #define EUNATCH         49      /* Protocol driver not attached         */
 53: #define ENOCSI          50      /* No CSI structure available           */
 54: #define EL2HLT          51      /* Level 2 halted                       */
 55: #define EBADE           52      /* Invalid exchange                     */
 56: #define EBADR           53      /* Invalid request descriptor           */
 57: #define EXFULL          54      /* Exchange full                        */
 58: #define ENOANO          55      /* No anode                             */
 59: #define EBADRQC         56      /* Invalid request code                 */
 60: #define EBADSLT         57      /* Invalid slot                         */
 61:
 62: #define EDEADLOCK       EDEADLK
 63:
 64: #define EBFONT          59      /* Bad font file format                 */
 65: #define ENOSTR          60      /* Device not a stream                  */
 66: #define ENODATA         61      /* No data available                    */
 67: #define ETIME           62      /* Timer expired                        */
```

include/fiwix/errno.h


```
   68: #define ENOSR           63      /* Out of streams resources             */
   69: #define ENONET          64      /* Machine is not on the network        */
   70: #define ENOPKG          65      /* Package not installed                */
   71: #define EREMOTE         66      /* Object is remote                     */
   72: #define ENOLINK         67      /* Link has been severed                */
   73: #define EADV            68      /* Advertise error                      */
   74: #define ESRMNT          69      /* Srmount error                        */
   75: #define ECOMM           70      /* Communication error on send          */
   76: #define EPROTO          71      /* Protocol error                       */
   77: #define EMULTIHOP       72      /* Multihop attempted                   */
   78: #define EDOTDOT         73      /* RFS specific error                   */
   79: #define EBADMSG         74      /* Not a data message                   */
   80: #define EOVERFLOW       75      /* Value too large for defined data type */
   81: #define ENOTUNIQ        76      /* Name not unique on network           */
   82: #define EBADFD          77      /* File descriptor in bad state         */
   83: #define EREMCHG         78      /* Remote address changed               */
   84: #define ELIBACC         79      /* Can not access a needed shared library */
   85: #define ELIBBAD         80      /* Accessing a corrupted shared library */
   86: #define ELIBSCN         81      /* .lib section in a.out corrupted      */
   87: #define ELIBMAX         82      /* Attempting to link in too many shared librari
es */
   88: #define ELIBEXEC        83      /* Cannot exec a shared library directly */
   89: #define EILSEQ          84      /* Illegal byte sequence                */
   90: #define ERESTART        85      /* Interrupted system call should be restarted *
/
   91: #define ESTRPIPE        86      /* Streams pipe error                   */
   92: #define EUSERS          87      /* Too many users                       */
   93: #define ENOTSOCK        88      /* Socket operation on non-socket       */
   94: #define EDESTADDRREQ    89      /* Destination address required         */
   95: #define EMSGSIZE        90      /* Message too long                     */
   96: #define EPROTOTYPE      91      /* Protocol wrong type for socket       */
   97: #define ENOPROTOOPT     92      /* Protocol not available               */
   98: #define EPROTONOSUPPORT 93      /* Protocol not supported               */
   99: #define ESOCKTNOSUPPORT 94      /* Socket type not supported            */
  100: #define EOPNOTSUPP      95      /* Operation not supported on transport endpoint
 */
  101: #define EPFNOSUPPORT    96      /* Protocol family not supported        */
  102: #define EAFNOSUPPORT    97      /* Address family not supported by protocol */
  103: #define EADDRINUSE      98      /* Address already in use               */
  104: #define EADDRNOTAVAIL   99      /* Cannot assign requested address      */
  105: #define ENETDOWN        100     /* Network is down                      */
  106: #define ENETUNREACH     101     /* Network is unreachable               */
  107: #define ENETRESET       102     /* Network dropped connection because of reset *
/
  108: #define ECONNABORTED    103     /* Software caused connection abort     */
  109: #define ECONNRESET      104     /* Connection reset by peer             */
  110: #define ENOBUFS         105     /* No buffer space available            */
  111: #define EISCONN         106     /* Transport endpoint is already connected */
  112: #define ENOTCONN        107     /* Transport endpoint is not connected  */
  113: #define ESHUTDOWN       108     /* Cannot send after transport endpoint shutdown
 */
  114: #define ETOOMANYREFS    109     /* Too many references: cannot splice   */
  115: #define ETIMEDOUT       110     /* Connection timed out                 */
  116: #define ECONNREFUSED    111     /* Connection refused                   */
  117: #define EHOSTDOWN       112     /* Host is down                         */
  118: #define EHOSTUNREACH    113     /* No route to host                     */
  119: #define EALREADY        114     /* Operation already in progress        */
  120: #define EINPROGRESS     115     /* Operation now in progress            */
  121: #define ESTALE          116     /* Stale NFS file handle                */
  122: #define EUCLEAN         117     /* Structure needs cleaning             */
  123: #define ENOTNAM         118     /* Not a XENIX named type file          */
  124: #define ENAVAIL         119     /* No XENIX semaphores available        */
  125: #define EISNAM          120     /* Is a named type file                 */
  126: #define EREMOTEIO       121     /* Remote I/O error                     */
  127: #define EDQUOT          122     /* Quota exceeded                       */
  128:
  129: #define ENOMEDIUM       123     /* No medium found                      */
```

include/fiwix/errno.h


```
130: #define EMEDIUMTYPE     124     /* Wrong medium type                    */
131:
132: #endif  /* _FIWIX_ERRNO_H */
```

include/fiwix/fcntl.h


```
 1: /*
 2:  * fiwix/include/fiwix/fcntl.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_FCNTL_H
 9: #define _FIWIX_FCNTL_H
10:
11: #include <fiwix/types.h>
12:
13: #define O_ACCMODE          0003
14: #define O_RDONLY             00
15: #define O_WRONLY             01
16: #define O_RDWR               02
17:
18: /* for open() only */
19: #define O_CREAT            0100 /* create file if it does not exist */
20: #define O_EXCL             0200 /* exclusive use flag */
21: #define O_NOCTTY           0400 /* do not assign controlling terminal */
22: #define O_TRUNC           01000 /* truncate flag */
23: #define O_NOFOLLOW      0400000 /* do not follow symbolic links */
24:
25: #define O_APPEND          02000
26: #define O_NONBLOCK        04000
27: #define O_NDELAY         O_NONBLOCK
28: #define O_SYNC           010000
29:
30: #define F_DUPFD         0       /* duplicate file descriptor */
31: #define F_GETFD         1       /* get file descriptor flags */
32: #define F_SETFD         2       /* set file descriptor flags */
33: #define F_GETFL         3       /* get status flags and file access modes */
34: #define F_SETFL         4       /* set file status flags */
35: #define F_GETLK         5       /* get record locking information */
36: #define F_SETLK         6       /* set record locking information */
37: #define F_SETLKW        7       /* same as F_SETLK; wait if blocked */
38:
39: /* get/set process or process group ID to receive SIGURG signals */
40: #define F_SETOWN        8       /* for sockets only */
41: #define F_GETOWN        9       /* for sockets only */
42:
43: /* for F_[GET|SET]FL */
44: #define FD_CLOEXEC      1       /* close the file descriptor upon exec() */
45:
46: /* for POSIX fcntl() */
47: #define F_RDLCK         0       /* shared or read lock */
48: #define F_WRLCK         1       /* exclusive or write lock */
49: #define F_UNLCK         2       /* unlock */
50:
51: /* for BSD flock() */
52: #define LOCK_SH         1       /* shared lock */
53: #define LOCK_EX         2       /* exclusive lock */
54: #define LOCK_NB         4       /* or'd with one of the above to prevent
55:                                    blocking */
56: #define LOCK_UN         8       /* unlock */
57:
58: /* IEEE Std 1003.1, 2004 Edition */
59: struct flock {
60:         short int l_type;       /* type of lock: F_RDLCK, F_WRLCK, F_UNLCK */
61:         short int l_whence;     /* flag for 'l_start': SEEK_SET, SEEK_CUR, ...*/
62:         __off_t l_start;        /* relative offset in bytes */
63:         __off_t l_len;          /* size; if 0 then until EOF */
64:         __pid_t l_pid;          /* PID holding the lock; returned in F_GETLK */
65: };
66:
67: #endif /* _FIWIX_FCNTL_H */
```

include/fiwix/filesystems.h


```
    1: /*
    2:  * fiwix/include/fiwix/filesystems.h
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #ifndef _FIWIX_FILESYSTEMS_H
    9: #define _FIWIX_FILESYSTEMS_H
   10:
   11: #include <fiwix/types.h>
   12: #include <fiwix/limits.h>
   13:
   14: #define NR_FILESYSTEMS          5       /* supported filesystems */
   15:
   16: /* value to be determined during system startup */
   17: extern unsigned int mount_table_size;   /* size in bytes */
   18:
   19: struct filesystems {
   20:         const char *name;               /* filesystem name */
   21:         struct fs_operations *fsop;     /* filesystem operations */
   22:         struct mount *mt;               /* mount-table entry (only for nodev) */
   23: };
   24: struct filesystems filesystems_table[NR_FILESYSTEMS];
   25:
   26: struct mount {
   27:         __dev_t dev;                    /* device number */
   28:         char devname[DEVNAME_MAX + 1];  /* device name */
   29:         char dirname[NAME_MAX + 1];     /* mount point directory name */
   30:         unsigned char used;             /* 1=busy, 0=free */
   31:         struct superblock sb;           /* superblock */
   32:         struct filesystems *fs;         /* pointer to filesystem structure */
   33: };
   34: extern struct mount *mount_table;
   35:
   36: int register_filesystem(const char *, struct fs_operations *);
   37: struct filesystems * get_filesystem(const char *);
   38: void fs_init(void);
   39:
   40: struct superblock * get_superblock(__dev_t);
   41: void sync_superblocks(__dev_t);
   42: int kern_mount(__dev_t, struct filesystems *);
   43: int mount_root(void);
   44: void mount_init(void);
   45:
   46:
   47: /* minix prototypes */
   48: int minix_file_open(struct inode *, struct fd *);
   49: int minix_file_close(struct inode *, struct fd *);
   50: int minix_file_write(struct inode *, struct fd *, const char *, __size_t);
   51: int minix_file_lseek(struct inode *, __off_t);
   52: int minix_dir_open(struct inode *, struct fd *);
   53: int minix_dir_close(struct inode *, struct fd *);
   54: int minix_dir_read(struct inode *, struct fd *, char *, __size_t);
   55: int minix_dir_write(struct inode *, struct fd *, const char *, __size_t);
   56: int minix_dir_readdir(struct inode *, struct fd *, struct dirent *, unsigned int
);
   57: int minix_readlink(struct inode *, char *, __size_t);
   58: int minix_followlink(struct inode *, struct inode *, struct inode **);
   59: int minix_bmap(struct inode *, __off_t, int);
   60: int minix_lookup(const char *, struct inode *, struct inode **);
   61: int minix_rmdir(struct inode *, struct inode *);
   62: int minix_link(struct inode *, struct inode *, char *);
   63: int minix_unlink(struct inode *, struct inode *, char *);
   64: int minix_symlink(struct inode *, char *, char *);
   65: int minix_mkdir(struct inode *, char *, __mode_t);
   66: int minix_mknod(struct inode *, char *, __mode_t, __dev_t);
```

include/fiwix/filesystems.h


```
   67: int minix_truncate(struct inode *, __off_t);
   68: int minix_create(struct inode *, char *, __mode_t, struct inode **);
   69: int minix_rename(struct inode *, struct inode *, struct inode *,
 char *, char *);
   70: int minix_read_inode(struct inode *);
   71: int minix_write_inode(struct inode *);
   72: int minix_ialloc(struct inode *, int);
   73: void minix_ifree(struct inode *);
   74: void minix_statfs(struct superblock *, struct statfs *);
   75: int minix_read_superblock(__dev_t, struct superblock *);
   76: int minix_remount_fs(struct superblock *, int);
   77: int minix_write_superblock(struct superblock *);
   78: void minix_release_superblock(struct superblock *);
   79: int minix_init(void);
   80:
   81:
   82: /* ext2 prototypes */
   83: int ext2_file_open(struct inode *, struct fd *);
   84: int ext2_file_close(struct inode *, struct fd *);
   85: int ext2_file_write(struct inode *, struct fd *, const char *, __size_t);
   86: int ext2_file_lseek(struct inode *, __off_t);
   87: int ext2_dir_open(struct inode *, struct fd *);
   88: int ext2_dir_close(struct inode *, struct fd *);
   89: int ext2_dir_read(struct inode *, struct fd *, char *, __size_t);
   90: int ext2_dir_write(struct inode *, struct fd *, const char *, __size_t);
   91: int ext2_dir_readdir(struct inode *, struct fd *, struct dirent *, unsigned int)
;
   92: int ext2_readlink(struct inode *, char *, __size_t);
   93: int ext2_followlink(struct inode *, struct inode *, struct inode **);
   94: int ext2_bmap(struct inode *, __off_t, int);
   95: int ext2_lookup(const char *, struct inode *, struct inode **);
   96: int ext2_rmdir(struct inode *, struct inode *);
   97: int ext2_link(struct inode *, struct inode *, char *);
   98: int ext2_unlink(struct inode *, struct inode *, char *);
   99: int ext2_symlink(struct inode *, char *, char *);
  100: int ext2_mkdir(struct inode *, char *, __mode_t);
  101: int ext2_mknod(struct inode *, char *, __mode_t, __dev_t);
  102: int ext2_truncate(struct inode *, __off_t);
  103: int ext2_create(struct inode *, char *, __mode_t, struct inode **);
  104: int ext2_rename(struct inode *, struct inode *, struct inode *, struct inode *,
char *, char *);
  105: int ext2_read_inode(struct inode *);
  106: int ext2_write_inode(struct inode *);
  107: int ext2_ialloc(struct inode *, int);
  108: void ext2_ifree(struct inode *);
  109: void ext2_statfs(struct superblock *, struct statfs *);
  110: int ext2_read_superblock(__dev_t, struct superblock *);
  111: int ext2_remount_fs(struct superblock *, int);
  112: int ext2_write_superblock(struct superblock *);
  113: void ext2_release_superblock(struct superblock *);
  114: int ext2_init(void);
  115:
  116:
  117: /* pipefs prototypes */
  118: int fifo_open(struct inode *, struct fd *);
  119: int pipefs_close(struct inode *, struct fd *);
  120: int pipefs_read(struct inode *, struct fd *, char *, __size_t);
  121: int pipefs_write(struct inode *, struct fd *, const char *, __size_t);
  122: int pipefs_ioctl(struct inode *, int, unsigned long int);
  123: int pipefs_lseek(struct inode *, __off_t);
  124: int pipefs_select(struct inode *, int);
  125: int pipefs_ialloc(struct inode *, int);
  126: void pipefs_ifree(struct inode *);
  127: int pipefs_read_superblock(__dev_t, struct superblock *);
  128: int pipefs_init(void);
  129:
  130:
```

include/fiwix/filesystems.h


```
  131: /* iso9660 prototypes */
  132: int iso9660_file_open(struct inode *, struct fd *);
  133: int iso9660_file_close(struct inode *, struct fd *);
  134: int iso9660_file_lseek(struct inode *, __off_t);
  135: int iso9660_dir_open(struct inode *, struct fd *);
  136: int iso9660_dir_close(struct inode *, struct fd *);
  137: int iso9660_dir_read(struct inode *, struct fd *, char *, __size_t);
  138: int iso9660_dir_readdir(struct inode *, struct fd *, struct dirent *, unsigned i
nt);
  139: int iso9660_readlink(struct inode *, char *, __size_t);
  140: int iso9660_followlink(struct inode *, struct inode *, struct inode **);
  141: int iso9660_bmap(struct inode *, __off_t, int);
  142: int iso9660_lookup(const char *, struct inode *, struct inode **);
  143: int iso9660_read_inode(struct inode *);
  144: void iso9660_statfs(struct superblock *, struct statfs *);
  145: int iso9660_read_superblock(__dev_t, struct superblock *);
  146: void iso9660_release_superblock(struct superblock *);
  147: int iso9660_init(void);
  148:
  149:
  150: /* procfs prototypes */
  151: int procfs_file_open(struct inode *, struct fd *);
  152: int procfs_file_close(struct inode *, struct fd *);
  153: int procfs_file_read(struct inode *, struct fd *, char *, __size_t);
  154: int procfs_file_lseek(struct inode *, __off_t);
  155: int procfs_dir_open(struct inode *, struct fd *);
  156: int procfs_dir_close(struct inode *, struct fd *);
  157: int procfs_dir_read(struct inode *, struct fd *, char *, __size_t);
  158: int procfs_dir_readdir(struct inode *, struct fd *, struct dirent *, unsigned in
t);
  159: int procfs_readlink(struct inode *, char *, __size_t);
  160: int procfs_followlink(struct inode *, struct inode *, struct inode **);
  161: int procfs_bmap(struct inode *, __off_t, int);
  162: int procfs_lookup(const char *, struct inode *, struct inode **);
  163: int procfs_read_inode(struct inode *);
  164: void procfs_statfs(struct superblock *, struct statfs *);
  165: int procfs_read_superblock(__dev_t, struct superblock *);
  166: int procfs_init(void);
  167:
  168: #endif /* _FIWIX_FILESYSTEMS_H */
```

include/fiwix/floppy.h


```
/*
 * fiwix/include/fiwix/floppy.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_FLOPPY_H
#define _FIWIX_FLOPPY_H

#include <fiwix/fs.h>

#define FLOPPY_IRQ      6
#define FLOPPY_DMA      2       /* DMA channel */

#define FDC_MAJOR       2       /* fdd major number */

#define FDC_SECTSIZE    512     /* sector size (in bytes) */
#define FDC_TR_DEFAULT  0       /* timer reason is IRQ */
#define FDC_TR_MOTOR    1       /* timer reason is motor on */

#define FDC_SRA         0x3F0   /* Status Register A */
#define FDC_SRB         0x3F1   /* Status Register B */
#define FDC_DOR         0x3F2   /* Digital Output Register */
#define FDC_MSR         0x3F4   /* Main Status Register */
#define FDC_DATA        0x3F5   /* command/data register */
#define FDC_DIR         0x3F7   /* Digital Input Register */
#define FDC_CCR         0x3F7   /* Configuration Control Register */

#define FDC_ENABLE      0x04    /* bit #2 FDC enabled (normal op) */
#define FDC_DMA_ENABLE  0x08    /* bit #3 DMA enabled */
#define FDC_DRIVE0      0x10    /* motor on for the first drive, the rest will
                                 * be calculated by left-shifting this value
                                 * with 'current_fdd'.
                                 */

#define FDC_DIO         0x40    /* bit #6 DIO I/O direction */
#define FDC_RQM         0x80    /* bit #7 RQM is ready for I/O */

#define MAX_FDC_RESULTS 7
#define MAX_FDC_ERR     5

#define FDC_RESET       0xFF    /* reset indicador */
#define FDC_READ        0xE6
#define FDC_WRITE       0xC5
#define FDC_VERSION     0x10
#define FDC_FORMAT_TRK  0x4D
#define FDC_RECALIBRATE 0x07
#define FDC_SENSEI      0x08
#define FDC_SPECIFY     0x03
#define FDC_SEEK        0x0F
#define FDC_LOCK        0x14
#define FDC_PARTID      0x18

#define ST0             0x00    /* Status Register 0 */
#define ST1             0x01    /* Status Register 1 */
#define ST2             0x02    /* Status Register 2 */

#define ST0_IC          0xC0    /* bits #7 and #6 interrupt code */
#define ST0_SE          0x20    /* bit #5 successful implied seek */
#define ST0_RECALIBRATE ST0_SE  /* bit #5 successful FDC_RECALIBRATE */
#define ST0_SEEK        ST0_SE  /* bit #5 successful FDC_SEEK */
#define ST0_UC          0x10    /* bit #4 unit needs check (fault) */
#define ST0_NR          0x8     /* bit #3 drive not ready */

#define ST1_NW          0x02    /* bit #1 not writable */

```

include/fiwix/floppy.h


```
 68: #define ST_PCN          0x01    /* present cylinder */
 69: #define ST_CYL          0x03    /* cylinder returned */
 70: #define ST_HEAD         0x04    /* head returned */
 71: #define ST_SECTOR       0x05    /* sector returned */
 72:
 73: /* floppy disk drive type */
 74: struct fddt {
 75:         short int size;         /* number of sectors */
 76:         short int sizekb;       /* size in KB */
 77:         char tracks;            /* number of tracks */
 78:         char spt;               /* number of sectors per track */
 79:         char heads;             /* number of heads */
 80:         char gpl1;              /* GAP in READ/WRITE operations */
 81:         char gpl2;              /* GAP in FORMAT TRACK operations */
 82:         char rate;              /* data rate value */
 83:         char spec;              /* SRT+HUT (StepRate + HeadUnload) Time */
 84:         char hlt;               /* HLT (Head Load Time) */
 85:         char *name;             /* unit name */
 86: };
 87:
 88: void irq_floppy(void);
 89: void fdc_timer(unsigned int);
 90:
 91: int fdc_open(struct inode *, struct fd *);
 92: int fdc_close(struct inode *, struct fd *);
 93: int fdc_read(__dev_t, __blk_t, char *, int);
 94: int fdc_write(__dev_t, __blk_t, char *, int);
 95: int fdc_ioctl(struct inode *, int, unsigned long int);
 96: int fdc_lseek(struct inode *, __off_t);
 97:
 98: void floppy_init(void);
 99:
100: #endif /* _FIWIX_FLOPPY_H */
```

include/fiwix/fs_ext2.h


```
/*
 * fiwix/include/fiwix/fs_ext2.h
 *
 * This file from: Linux 2.0.40
 * Copyright (C) 1992, 1993, 1994, 1995
 * Remy Card (card@masi.ibp.fr)
 * Laboratoire MASI - Institut Blaise Pascal
 * Universite Pierre et Marie Curie (Paris VI)
 * Copyright (C) 1991, 1992  Linus Torvalds
 */

#ifndef _FIWIX_FS_EXT2_H
#define _FIWIX_FS_EXT2_H

#include <fiwix/types.h>

#define EXT2_ROOT_INO            2      /* Root inode */
#define EXT2_SUPER_MAGIC        0xEF53

/*
 * Macro-instructions used to manage several block sizes
 */
#define EXT2_MIN_BLOCK_SIZE             1024
#define EXT2_MAX_BLOCK_SIZE             4096
#define EXT2_MIN_BLOCK_LOG_SIZE           10
# define EXT2_BLOCK_SIZE(s)             ((s)->s_blocksize)
# define EXT2_BLOCK_SIZE_BITS(s)        ((s)->s_blocksize_bits)

/*
 * Structure of a blocks group descriptor
 */
struct ext2_group_desc
{
        __u32   bg_block_bitmap;        /* Blocks bitmap block */
        __u32   bg_inode_bitmap;        /* Inodes bitmap block */
        __u32   bg_inode_table;         /* Inodes table block */
        __u16   bg_free_blocks_count;   /* Free blocks count */
        __u16   bg_free_inodes_count;   /* Free inodes count */
        __u16   bg_used_dirs_count;     /* Directories count */
        __u16   bg_pad;
        __u32   bg_reserved[3];
};

/*
 * Macro-instructions used to manage group descriptors
 */
#define EXT2_BLOCKS_PER_GROUP(s)        ((s)->u.ext2.sb.s_blocks_per_group)
#define EXT2_INODES_PER_GROUP(s)        ((s)->u.ext2.sb.s_inodes_per_group)
# define EXT2_DESC_PER_BLOCK_BITS(s)    ((s)->u.ext2_sb.s_desc_per_block_bits)
#define EXT2_DESC_PER_BLOCK(s)          ((s)->u.ext2.desc_per_block)

/*
 * Constants relative to the data blocks
 */
#define EXT2_NDIR_BLOCKS                12
#define EXT2_IND_BLOCK                  EXT2_NDIR_BLOCKS
#define EXT2_DIND_BLOCK                 (EXT2_IND_BLOCK + 1)
#define EXT2_TIND_BLOCK                 (EXT2_DIND_BLOCK + 1)
#define EXT2_N_BLOCKS                   (EXT2_TIND_BLOCK + 1)

/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        __u16   i_mode;         /* File mode */
        __u16   i_uid;          /* Low 16 bits of Owner Uid */
        __u32   i_size;         /* Size in bytes */
```

include/fiwix/fs_ext2.h


```
68:         __u32   i_atime;        /* Access time */
69:         __u32   i_ctime;        /* Creation time */
70:         __u32   i_mtime;        /* Modification time */
71:         __u32   i_dtime;        /* Deletion Time */
72:         __u16   i_gid;          /* Low 16 bits of Group Id */
73:         __u16   i_links_count;  /* Links count */
74:         __u32   i_blocks;       /* Blocks count */
75:         __u32   i_flags;        /* File flags */
76:         union {
77:                 struct {
78:                         __u32  l_i_reserved1;
79:                 } linux1;
80:                 struct {
81:                         __u32  h_i_translator;
82:                 } hurd1;
83:                 struct {
84:                         __u32  m_i_reserved1;
85:                 } masix1;
86:         } osd1;                         /* OS dependent 1 */
87:         __u32   i_block[EXT2_N_BLOCKS];/* Pointers to blocks */
88:         __u32   i_generation;   /* File version (for NFS) */
89:         __u32   i_file_acl;     /* File ACL */
90:         __u32   i_dir_acl;      /* Directory ACL */
91:         __u32   i_faddr;        /* Fragment address */
92:         union {
93:                 struct {
94:                         __u8    l_i_frag;       /* Fragment number */
95:                         __u8    l_i_fsize;      /* Fragment size */
96:                         __u16   i_pad1;
97:                         __u16   l_i_uid_high;   /* these 2 fields    */
98:                         __u16   l_i_gid_high;   /* were reserved2[0] */
99:                         __u32   l_i_reserved2;
100:                } linux2;
101:                struct {
102:                        __u8    h_i_frag;       /* Fragment number */
103:                        __u8    h_i_fsize;      /* Fragment size */
104:                        __u16   h_i_mode_high;
105:                        __u16   h_i_uid_high;
106:                        __u16   h_i_gid_high;
107:                        __u32   h_i_author;
108:                } hurd2;
109:                struct {
110:                        __u8    m_i_frag;       /* Fragment number */
111:                        __u8    m_i_fsize;      /* Fragment size */
112:                        __u16   m_pad1;
113:                        __u32   m_i_reserved2[2];
114:                } masix2;
115:        } osd2;                         /* OS dependent 2 */
116: };
117:
118: /*
119:  * File system states
120:  */
121: #define EXT2_VALID_FS                   0x0001  /* Unmounted cleanly */
122: #define EXT2_ERROR_FS                   0x0002  /* Errors detected */
123:
124: /*
125:  * Structure of the super block
126:  */
127: struct ext2_super_block {
128:         __u32   s_inodes_count;         /* Inodes count */
129:         __u32   s_blocks_count;         /* Blocks count */
130:         __u32   s_r_blocks_count;       /* Reserved blocks count */
131:         __u32   s_free_blocks_count;    /* Free blocks count */
132:         __u32   s_free_inodes_count;    /* Free inodes count */
133:         __u32   s_first_data_block;     /* First Data Block */
134:         __u32   s_log_block_size;       /* Block size */
```

include/fiwix/fs_ext2.h


```
135:         __s32   s_log_frag_size;        /* Fragment size */
136:         __u32   s_blocks_per_group;     /* # Blocks per group */
137:         __u32   s_frags_per_group;      /* # Fragments per group */
138:         __u32   s_inodes_per_group;     /* # Inodes per group */
139:         __u32   s_mtime;                /* Mount time */
140:         __u32   s_wtime;                /* Write time */
141:         __u16   s_mnt_count;            /* Mount count */
142:         __s16   s_max_mnt_count;        /* Maximal mount count */
143:         __u16   s_magic;                /* Magic signature */
144:         __u16   s_state;                /* File system state */
145:         __u16   s_errors;               /* Behaviour when detecting errors */
146:         __u16   s_minor_rev_level;      /* minor revision level */
147:         __u32   s_lastcheck;            /* time of last check */
148:         __u32   s_checkinterval;        /* max. time between checks */
149:         __u32   s_creator_os;           /* OS */
150:         __u32   s_rev_level;            /* Revision level */
151:         __u16   s_def_resuid;           /* Default uid for reserved blocks */
152:         __u16   s_def_resgid;           /* Default gid for reserved blocks */
153:         /*
154:          * These fields are for EXT2_DYNAMIC_REV superblocks only.
155:          *
156:          * Note: the difference between the compatible feature set and
157:          * the incompatible feature set is that if there is a bit set
158:          * in the incompatible feature set that the kernel doesn't
159:          * know about, it should refuse to mount the filesystem.
160:          *
161:          * e2fsck's requirements are more strict; if it doesn't know
162:          * about a feature in either the compatible or incompatible
163:          * feature set, it must abort and not try to meddle with
164:          * things it doesn't understand...
165:          */
166:         __u32   s_first_ino;            /* First non-reserved inode */
167:         __u16   s_inode_size;           /* size of inode structure */
168:         __u16   s_block_group_nr;       /* block group # of this superblock */
169:         __u32   s_feature_compat;       /* compatible feature set */
170:         __u32   s_feature_incompat;     /* incompatible feature set */
171:         __u32   s_feature_ro_compat;    /* readonly-compatible feature set */
172:         __u8    s_uuid[16];             /* 128-bit uuid for volume */
173:         char    s_volume_name[16];      /* volume name */
174:         char    s_last_mounted[64];     /* directory where last mounted */
175:         __u32   s_algorithm_usage_bitmap; /* For compression */
176:         /*
177:          * Performance hints.  Directory preallocation should only
178:          * happen if the EXT2_COMPAT_PREALLOC flag is on.
179:          */
180:         __u8    s_prealloc_blocks;      /* Nr of blocks to try to preallocate*/
181:         __u8    s_prealloc_dir_blocks;  /* Nr to preallocate for dirs */
182:         __u16   s_padding1;
183:         /*
184:          * Journaling support valid if EXT3_FEATURE_COMPAT_HAS_JOURNAL set.
185:          */
186:         __u8    s_journal_uuid[16];     /* uuid of journal superblock */
187:         __u32   s_journal_inum;         /* inode number of journal file */
188:         __u32   s_journal_dev;          /* device number of journal file */
189:         __u32   s_last_orphan;          /* start of list of inodes to delete */
190:         __u32   s_hash_seed[4];         /* HTREE hash seed */
191:         __u8    s_def_hash_version;     /* Default hash version to use */
192:         __u8    s_reserved_char_pad;
193:         __u16   s_reserved_word_pad;
194:         __u32   s_default_mount_opts;
195:         __u32   s_first_meta_bg;        /* First metablock block group */
196:         __u32   s_reserved[190];        /* Padding to the end of the block */
197: };
198:
199: /*
200:  * Structure of a directory entry
201:  */
```

include/fiwix/fs_ext2.h


```
202: #define EXT2_NAME_LEN 255
203:
204: struct ext2_dir_entry {
205:         __u32   inode;                  /* Inode number */
206:         __u16   rec_len;                /* Directory entry length */
207:         __u16   name_len;               /* Name length */
208:         char    name[EXT2_NAME_LEN];    /* File name */
209: };
210:
211: /*
212:  * The new version of the directory entry.  Since EXT2 structures are
213:  * stored in intel byte order, and the name_len field could never be
214:  * bigger than 255 chars, it's safe to reclaim the extra byte for the
215:  * file_type field.
216:  */
217: struct ext2_dir_entry_2 {
218:         __u32   inode;                  /* Inode number */
219:         __u16   rec_len;                /* Directory entry length */
220:         __u8    name_len;               /* Name length */
221:         __u8    file_type;
222:         char    name[EXT2_NAME_LEN];    /* File name */
223: };
224:
225: /*
226:  * Ext2 directory file types.  Only the low 3 bits are used.  The
227:  * other bits are reserved for now.
228:  */
229: #define EXT2_FT_UNKNOWN         0
230: #define EXT2_FT_REG_FILE        1
231: #define EXT2_FT_DIR             2
232: #define EXT2_FT_CHRDEV          3
233: #define EXT2_FT_BLKDEV          4
234: #define EXT2_FT_FIFO            5
235: #define EXT2_FT_SOCK            6
236: #define EXT2_FT_SYMLINK         7
237:
238: /* superblock in memory */
239: struct ext2_sb_info {
240:         unsigned int desc_per_block;
241:         unsigned int block_groups;
242:         struct ext2_super_block sb;
243: };
244:
245: /* inode in memory */
246: struct ext2_i_info {
247:         __u32   i_data[EXT2_N_BLOCKS];  /* Pointers to blocks */
248:         __u32   i_dtime;
249: };
250:
251: #endif  /* _FIWIX_FS_EXT2_H */
```

include/fiwix/fs.h


```
/*
 * fiwix/include/fiwix/fs.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_FS_H
#define _FIWIX_FS_H

#include <fiwix/config.h>
#include <fiwix/types.h>

#define CHECK_UFD(ufd)                                                  \
{                                                                       \
        if((ufd) > OPEN_MAX || current->fd[(ufd)] == 0)  {              \
                return -EBADF;                                          \
        }                                                               \
}

struct fd {
        struct inode *inode;            /* file inode */
        unsigned short int flags;       /* flags */
        unsigned short int count;       /* number of opened instances */
        __off_t offset;                 /* r/w pointer position */
};

#include <fiwix/statfs.h>
#include <fiwix/limits.h>
#include <fiwix/process.h>
#include <fiwix/dirent.h>
#include <fiwix/fs_minix.h>
#include <fiwix/fs_ext2.h>
#include <fiwix/fs_pipe.h>
#include <fiwix/fs_iso9660.h>
#include <fiwix/fs_proc.h>

#define BPS                     512     /* bytes per sector */
#define BLKSIZE_1K              1024    /* 1KB block size */
#define BLKSIZE_2K              2048    /* 2KB block size */
#define SUPERBLOCK              1       /* block 1 is for superblock */

#define MAJOR(dev)              (((__dev_t) (dev)) >> 8)
#define MINOR(dev)              (((__dev_t) (dev)) & 0xFF)
#define MKDEV(major, minor)     (((major) << 8) | (minor))

/* filesystem independent mount-flags */
#define MS_RDONLY               1       /* mount read-only */
#define MS_REMOUNT              32      /* alter flags of a mounted FS */

/* old magic mount flag and mask */
#define MS_MGC_VAL              0xC0ED0000
#define MS_MGC_MSK              0xFFFF0000

#define IS_RDONLY_FS(inode) (((inode)->sb) && ((inode)->sb->flags & MS_RDONLY))

#define FOLLOW_LINKS    1
#define MAX_SYMLINKS    8       /* this prevents infinite loops in symlinks */

#define SEEK_SET        0
#define SEEK_CUR        1
#define SEEK_END        2

#define FOR_READING     0
#define FOR_WRITING     1

#define VERIFY_READ     1
```

include/fiwix/fs.h


```
 68: #define VERIFY_WRITE    2
 69:
 70: #define SEL_R           1
 71: #define SEL_W           2
 72: #define SEL_E           4
 73:
 74: struct inode {
 75:         __mode_t        i_mode;         /* file mode */
 76:         __uid_t         i_uid;          /* owner uid */
 77:         __size_t        i_size;         /* size in bytes */
 78:         __u32           i_atime;        /* access time */
 79:         __u32           i_ctime;        /* creation time */
 80:         __u32           i_mtime;        /* modification time */
 81:         __gid_t         i_gid;          /* group id */
 82:         __nlink_t       i_nlink;        /* links count */
 83:         __blk_t         i_blocks;       /* blocks count */
 84:         __u32           i_flags;        /* file flags */
 85:         unsigned char locked;
 86:         unsigned char dirty;            /* 1 = delayed write */
 87:         struct inode *mount_point;
 88:         __dev_t         dev;
 89:         __ino_t         inode;
 90:         __s16           count;
 91:         __dev_t         rdev;
 92:         struct fs_operations *fsop;
 93:         struct superblock *sb;
 94:         struct inode *prev_hash;
 95:         struct inode *next_hash;
 96:         struct inode *prev_free;
 97:         struct inode *next_free;
 98:         union {
 99:                 struct minix_i_info minix;
100:                 struct ext2_i_info ext2;
101:                 struct pipefs_inode pipefs;
102:                 struct iso9660_inode iso9660;
103:                 struct procfs_inode procfs;
104:         } u;
105: };
106: extern struct inode *inode_table;
107: extern struct inode **inode_hash_table;
108: extern int inodes_on_free_list;
109:
110: /* values to be determined during system startup */
111: extern unsigned int inode_table_size;           /* size in bytes */
112: extern unsigned int inode_hash_table_size;      /* size in bytes */
113: extern unsigned int fd_table_size;              /* size in bytes */
114:
115: extern struct fd *fd_table;
116:
117: struct superblock {
118:         __dev_t dev;
119:         unsigned char locked;
120:         unsigned char wanted;
121:         struct inode *root;             /* root inode of mounted fs */
122:         struct inode *dir;              /* inode on which the fs was mounted */
123:         unsigned int flags;
124:         unsigned char dirty;            /* 1 = delayed write */
125:         struct fs_operations *fsop;
126:         __u32 s_blocksize;
127:         union {
128:                 struct minix_sb_info minix;
129:                 struct ext2_sb_info ext2;
130:                 struct iso9660_sb_info iso9660;
131:         } u;
132: };
133:
134:
```

include/fiwix/fs.h


```
  135: #define FSOP_REQUIRES_DEV       1       /* requires a block device */
  136: #define FSOP_KERN_MOUNT         2       /* mounted by kernel */
  137:
  138: struct fs_operations {
  139:         int flags;
  140:         int fsdev;                      /* internal filesystem (nodev) */
  141:
  142: /* file operations */
  143:         int (*open)(struct inode *, struct fd *);
  144:         int (*close)(struct inode *, struct fd *);
  145:         int (*read)(struct inode *, struct fd *, char *, __size_t);
  146:         int (*write)(struct inode *, struct fd *, const char *, __size_t);
  147:         int (*ioctl)(struct inode *, int, unsigned long int);
  148:         int (*lseek)(struct inode *, __off_t);
  149:         int (*readdir)(struct inode *, struct fd *, struct dirent *, unsigned in
t);
  150:         int (*mmap)(struct inode *, struct vma *);
  151:         int (*select)(struct inode *, int);
  152:
  153: /* inode operations */
  154:         int (*readlink)(struct inode *, char *, __size_t);
  155:         int (*followlink)(struct inode *, struct inode *, struct inode **);
  156:         int (*bmap)(struct inode *, __off_t, int);
  157:         int (*lookup)(const char *, struct inode *, struct inode **);
  158:         int (*rmdir)(struct inode *, struct inode *);
  159:         int (*link)(struct inode *, struct inode *, char *);
  160:         int (*unlink)(struct inode *, struct inode *, char *);
  161:         int (*symlink)(struct inode *, char *, char *);
  162:         int (*mkdir)(struct inode *, char *, __mode_t);
  163:         int (*mknod)(struct inode *, char *, __mode_t, __dev_t);
  164:         int (*truncate)(struct inode *, __off_t);
  165:         int (*create)(struct inode *, char *, __mode_t, struct inode **);
  166:         int (*rename)(struct inode *, struct inode *, struct inode *, struct ino
de *, char *, char *);
  167:
  168: /* block device I/O operations */
  169:         int (*read_block)(__dev_t, __blk_t, char *, int);
  170:         int (*write_block)(__dev_t, __blk_t, char *, int);
  171:
  172: /* superblock operations */
  173:         int (*read_inode)(struct inode *);
  174:         int (*write_inode)(struct inode *);
  175:         int (*ialloc)(struct inode *, int);
  176:         void (*ifree)(struct inode *);
  177:         void (*statfs)(struct superblock *, struct statfs *);
  178:         int (*read_superblock)(__dev_t, struct superblock *);
  179:         int (*remount_fs)(struct superblock *, int);
  180:         int (*write_superblock)(struct superblock *);
  181:         void (*release_superblock)(struct superblock *);
  182: };
  183:
  184: extern struct fs_operations def_chr_fsop;
  185: extern struct fs_operations def_blk_fsop;
  186:
  187: /* fs_minix.h prototypes */
  188: extern struct fs_operations minix_fsop;
  189: extern struct fs_operations minix_file_fsop;
  190: extern struct fs_operations minix_dir_fsop;
  191: extern struct fs_operations minix_symlink_fsop;
  192: extern int minix_count_free_inodes(struct superblock *);
  193: extern int minix_count_free_blocks(struct superblock *);
  194: extern int minix_find_first_zero(struct superblock *, __blk_t, int, int);
  195: extern int minix_change_bit(int, struct superblock *, int, int);
  196: extern int minix_balloc(struct superblock *);
  197: extern void minix_bfree(struct superblock *, int);
  198:
  199: /* fs_ext2.h prototypes */
```

include/fiwix/fs.h


```
  200: extern struct fs_operations ext2_fsop;
  201: extern struct fs_operations ext2_file_fsop;
  202: extern struct fs_operations ext2_dir_fsop;
  203: extern struct fs_operations ext2_symlink_fsop;
  204: extern int ext2_balloc(struct superblock *);
  205: extern void ext2_bfree(struct superblock *, int);
  206:
  207: /* fs_proc.h prototypes */
  208: extern struct fs_operations procfs_fsop;
  209: extern struct fs_operations procfs_file_fsop;
  210: extern struct fs_operations procfs_dir_fsop;
  211: extern struct fs_operations procfs_symlink_fsop;
  212: struct procfs_dir_entry * get_procfs_by_inode(struct inode *);
  213:
  214: /* fs_iso9660.h prototypes */
  215: extern int isonum_711(char *);
  216: extern int isonum_723(char *);
  217: extern int isonum_731(char *);
  218: extern int isonum_733(char *);
  219: extern unsigned long int isodate(char *);
  220: extern int iso9660_cleanfilename(char *, int);
  221: extern struct fs_operations iso9660_fsop;
  222: extern struct fs_operations iso9660_file_fsop;
  223: extern struct fs_operations iso9660_dir_fsop;
  224: extern struct fs_operations iso9660_symlink_fsop;
  225: void check_rrip_inode(struct iso9660_directory_record *, struct inode *);
  226: int get_rrip_filename(struct iso9660_directory_record *, struct inode *, char *)
;
  227: int get_rrip_symlink(struct inode *, char *);
  228:
  229:
  230: /* generic VFS function prototypes */
  231: void inode_lock(struct inode *);
  232: void inode_unlock(struct inode *);
  233: struct inode * ialloc(struct superblock *, int);
  234: struct inode * iget(struct superblock *, __ino_t);
  235: int bmap(struct inode *, __off_t, int);
  236: int check_fs_busy(__dev_t, struct inode *);
  237: void iput(struct inode *);
  238: void sync_inodes(__dev_t);
  239: void invalidate_inodes(__dev_t);
  240: void inode_init(void);
  241:
  242: int parse_namei(char *, struct inode *, struct inode **, struct inode **, int);
  243: int namei(char *, struct inode **, struct inode **, int);
  244:
  245: void superblock_lock(struct superblock *);
  246: void superblock_unlock(struct superblock *);
  247: struct mount * get_free_mount_point(__dev_t);
  248: void release_mount_point(struct mount *);
  249: struct mount * get_mount_point(struct inode *);
  250:
  251: int get_new_fd(struct inode *);
  252: void release_fd(unsigned int);
  253: void fd_init(void);
  254:
  255: void free_name(const char *);
  256: int malloc_name(const char *, char **);
  257: int check_user_permission(struct inode *);
  258: int check_group(struct inode *);
  259: int check_user_area(int, const void *, unsigned int);
  260: int check_permission(int, struct inode *);
  261:
  262: int do_select(int, fd_set *, fd_set *, fd_set *, fd_set *, fd_set *, fd_set *);
  263:
  264: #endif /* _FIWIX_FS_H */
```

include/fiwix/fs_iso9660.h


```
/*
 * fiwix/include/fiwix/fs_iso9660.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_FS_ISO9660_H
#define _FIWIX_FS_ISO9660_H

#include <fiwix/types.h>
#include <fiwix/limits.h>

#define ISO9660_SUPERBLOCK      16      /* ISO9660 superblock is in block 16 */
#define ISO9660_STANDARD_ID     "CD001" /* standard identification */
#define ISO9660_SUPER_MAGIC     0x9660

#define ISO9660_VD_BOOT                 0
#define ISO9660_VD_PRIMARY              1
#define ISO9660_VD_SUPPLEMENTARY        2
#define ISO9660_VD_PARTITION            3
#define ISO9660_VD_END                  255

#define ISODCL(from, to)        ((to - from) + 1)       /* descriptor length */

#define ISO9660_MAX_VD          10      /* maximum number of VD per CDROM */

/* inodes will have their directory block and their offset packed as follows:
 * 7FF7FF
 * \-/\-/
 *  ^  ^
 *  |  +----- offset value                      (11bit entries)
 *  +-------- directory block where to find it  (11bit entries)
 */
#define ISO9660_INODE_BITS      11      /* FIXME: it could be greater (16bit) */
#define ISO9660_INODE_MASK      0x7FF

#define ISO9660_FILE_NOTEXIST   0x01    /* file shouldn't exists for the user */
#define ISO9660_FILE_ISDIR      0x02    /* is a directory */
#define ISO9660_FILE_ISASSOC    0x04    /* associated file */
#define ISO9660_FILE_HASRECFMT  0x08    /* has a record format */
#define ISO9660_FILE_HASOWNER   0x10    /* has owner and group defined */
#define ISO9660_FILE_RESERVED5  0x20    /* reserved */
#define ISO9660_FILE_RESERVED6  0x40    /* reserved */
#define ISO9660_FILE_ISMULTIEXT 0x80    /* has more directory records */

#define SP_MAGIC1               0xBE
#define SP_MAGIC2               0xEF
#define GET_SIG(s1, s2)         ((s1 << 8) | s2)

#define SL_CURRENT              0x02
#define SL_PARENT               0x04
#define SL_ROOT                 0x08

#define TF_CREATION             0x01
#define TF_MODIFY               0x02
#define TF_ACCESS               0x04
#define TF_ATTRIBUTES           0x08
#define TF_BACKUP               0x10
#define TF_EXPIRATION           0x20
#define TF_EFFECTIVE            0x40
#define TF_LONG_FORM            0x80

#define NM_CONTINUE             0
#define NM_CURRENT              1
#define NM_PARENT               2

```

include/fiwix/fs_iso9660.h


```
   68: /* formerly Primary Volume Descriptor */
   69: struct iso9660_super_block {
   70:         char type                       [ISODCL(  1,   1)];     /* 7.1.1 */
   71:         char id                         [ISODCL(  2,   6)];
   72:         char version                    [ISODCL(  7,   7)];     /* 7.1.1 */
   73:         char unused1                    [ISODCL(  8,   8)];
   74:         char system_id                  [ISODCL(  9,  40)];     /* a-chars */
   75:         char volume_id                  [ISODCL( 41,  72)];     /* d-chars */
   76:         char unused2                    [ISODCL( 73,  80)];
   77:         char volume_space_size          [ISODCL( 81,  88)];     /* 7.3.3 */
   78:         char unused3                    [ISODCL( 89, 120)];
   79:         char volume_set_size            [ISODCL(121, 124)];     /* 7.2.3 */
   80:         char volume_sequence_number     [ISODCL(125, 128)];     /* 7.2.3 */
   81:         char logical_block_size         [ISODCL(129, 132)];     /* 7.2.3 */
   82:         char path_table_size            [ISODCL(133, 140)];     /* 7.3.3 */
   83:         char type_l_path_table          [ISODCL(141, 144)];     /* 7.3.1 */
   84:         char opt_type_l_path_table      [ISODCL(145, 148)];     /* 7.3.1 */
   85:         char type_m_path_table          [ISODCL(149, 152)];     /* 7.3.2 */
   86:         char opt_type_m_path_table      [ISODCL(153, 156)];     /* 7.3.2 */
   87:         char root_directory_record      [ISODCL(157, 190)];     /* 9.1 */
   88:         char volume_set_id              [ISODCL(191, 318)];     /* d-chars */
   89:         char publisher_id               [ISODCL(319, 446)];     /* a-chars */
   90:         char preparer_id                [ISODCL(447, 574)];     /* a-chars */
   91:         char application_id             [ISODCL(575, 702)];     /* a-chars */
   92:         char copyright_file_id          [ISODCL(703, 739)];     /* 7.5 d-chars *
/
   93:         char abstract_file_id           [ISODCL(740, 776)];     /* 7.5 d-chars *
/
   94:         char bibliographic_file_id      [ISODCL(777, 813)];     /* 7.5 d-chars *
/
   95:         char creation_date              [ISODCL(814, 830)];     /* 8.4.26.1 */
   96:         char modification_date          [ISODCL(831, 847)];     /* 8.4.26.1 */
   97:         char expiration_date            [ISODCL(848, 864)];     /* 8.4.26.1 */
   98:         char effective_date             [ISODCL(865, 881)];     /* 8.4.26.1 */
   99:         char file_structure_version     [ISODCL(882, 882)];
  100:         char unused4                    [ISODCL(883, 883)];
  101:         char application_data           [ISODCL(884, 1395)];
  102:         char unused5                    [ISODCL(1396, 2048)];
  103: };
  104:
  105: struct iso9660_directory_record
  106: {
  107:         char length                     [ISODCL( 1,  1)];       /* 7.1.1 */
  108:         char ext_attr_length            [ISODCL( 2,  2)];       /* 7.1.1 */
  109:         char extent                     [ISODCL( 3, 10)];       /* 7.3.3 */
  110:         char size                       [ISODCL(11, 18)];       /* 7.3.3 */
  111:         char date                       [ISODCL(19, 25)];       /* 7 by 7.1.1 */
  112:         char flags                      [ISODCL(26, 26)];
  113:         char file_unit_size             [ISODCL(27, 27)];       /* 7.1.1 */
  114:         char interleave                 [ISODCL(28, 28)];       /* 7.1.1 */
  115:         char volume_sequence_number     [ISODCL(29, 32)];       /* 7.2.3 */
  116:         char name_len                   [ISODCL(33, 33)];       /* 7.1.1 */
  117:         char name[0];
  118: };
  119:
  120: struct iso9660_pathtable_record
  121: {
  122:         char length                     [ISODCL( 1,  1)];       /* 7.1.1 */
  123:         char ext_attr_length            [ISODCL( 2,  2)];       /* 7.1.1 */
  124:         char extent                     [ISODCL( 3,  6)];       /* 7.3 */
  125:         char parent                     [ISODCL( 7,  8)];       /* 7.2 */
  126:         char name[0];
  127: };
  128:
  129: struct susp_sp {
  130:         unsigned char magic[2];
  131:         char len_skip;
```

include/fiwix/fs_iso9660.h


```
132: };
133:
134: struct susp_ce {
135:         char block[8];
136:         char offset[8];
137:         char size[8];
138: };
139:
140: struct susp_er {
141:         char len_id;
142:         char len_des;
143:         char len_src;
144:         char ext_ver;
145:         char data[0];
146: };
147:
148: struct rrip_px {
149:         char mode[8];
150:         char nlink[8];
151:         char uid[8];
152:         char gid[8];
153:         char sn[8];
154: };
155:
156: struct rrip_pn {
157:         char dev_h[8];
158:         char dev_l[8];
159: };
160:
161: struct rrip_sl_component {
162:         unsigned char flags;
163:         unsigned char len;
164:         char name[0];
165: };
166:
167: struct rrip_sl {
168:         unsigned char flags;
169:         struct rrip_sl_component area;
170: };
171:
172: struct rrip_nm {
173:         unsigned char flags;
174:         char name[0];
175: };
176:
177: struct rrip_tf_timestamp {
178:         char time[7];           /* assumes LONG_FORM bit always set to zero */
179: };
180:
181: struct rrip_tf {
182:         char flags;
183:         struct rrip_tf_timestamp times[0];
184: };
185:
186: struct susp_rrip {
187:         char signature[2];
188:         unsigned char len;
189:         unsigned char version;
190:         union {
191:                 struct susp_sp sp;
192:                 struct susp_ce ce;
193:                 struct susp_er er;
194:                 struct rrip_px px;
195:                 struct rrip_pn pn;
196:                 struct rrip_sl sl;
197:                 struct rrip_nm nm;
198:                 struct rrip_tf tf;
```

include/fiwix/fs_iso9660.h


```
199:         } u;
200: };
201:
202: struct iso9660_inode {
203:         __blk_t i_extent;
204:         struct inode *i_parent;         /* inode of its parent directory */
205: };
206:
207: struct iso9660_sb_info {
208:         __u32 s_root_inode;
209:         char *pathtable_raw;
210:         struct iso9660_pathtable_record **pathtable;
211:         int paths;
212:         unsigned char rrip;
213:         struct iso9660_super_block *sb;
214: };
215:
216: #endif /* _FIWIX_FS_ISO9660_H */
```

include/fiwix/fs_minix.h


```
/*
 * fiwix/include/fiwix/fs_minix.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_FS_MINIX_H
#define _FIWIX_FS_MINIX_H

#include <fiwix/types.h>
#include <fiwix/limits.h>

#define MINIX_ROOT_INO          1       /* root inode */

#define MINIX_SUPER_MAGIC       0x137F  /* Minix v1, 14 char names */
#define MINIX_SUPER_MAGIC2      0x138F  /* Minix v1, 30 char names */
#define MINIX2_SUPER_MAGIC      0x2468  /* Minix v2, 14 char names */
#define MINIX2_SUPER_MAGIC2     0x2478  /* Minix v2, 30 char names */

#define MINIX_VALID_FS          1       /* clean filesystem */
#define MINIX_ERROR_FS          2       /* needs fsck */

#define CLEAR_BIT               0
#define SET_BIT                 1

#define V1_MAX_BITMAP_BLOCKS    8       /* 64MB filesystem size */
#define V2_MAX_BITMAP_BLOCKS    128     /* 1GB filesystem size */

/*
 * Minix (v1 and v2) file system physical layout:
 *
 *               +-----------------------------------------------
 *               |   size in blocks of BLKSIZE_1K (1024 bytes)  |
 * +-------------+-----------------------------------------------+
 * | block 0     |                                             1 |
 * +-------------+-----------------------------------------------+
 * | superblock  |                                             1 |
 * +-------------+-----------------------------------------------+
 * | inode map   |          number of inodes / (BLKSIZE_1K * 8) |
 * +-------------+-----------------------------------------------+
 * | zone map    |           number of zones / (BLKSIZE_1K * 8) |
 * +-------------+-----------------------------------------------+
 * | inode table | ((32 or 64) * number of inodes) / BLKSIZE_1K |
 * +-------------+-----------------------------------------------+
 * | data zones  |                                           ... |
 * +-------------+-----------------------------------------------+
 *
 * The implementation of this filesystem in Fiwix might have slow disk writes
 * because I don't keep in memory the superblock, nor the blocks of the inode
 * map nor the blocks of the zone map. Keeping them in memory would be a waste
 * of 137KB per each mounted v2 filesystem (1GB of size).
 *
 * - superblock    ->   1KB
 * - inode map     ->   8KB (1KB (8192 bits) x   8 = 65536 inodes)
 * - zone map      -> 128KB (1KB (8192 bits) x 128 = 1048576 1k-blocks)
 *
 */

struct minix_super_block {
        __u16 s_ninodes;                /* number of inodes */
        __u16 s_nzones;                 /* number of data zones */
        __u16 s_imap_blocks;            /* blocks used by inode bitmap */
        __u16 s_zmap_blocks;            /* blocks used by zone bitmap */
        __u16 s_firstdatazone;          /* number of first data zone */
        __u16 s_log_zone_size;          /* 1024 << s_log_zone_size */
        __u32 s_max_size;               /* maximum file size (in bytes) */
```

include/fiwix/fs_minix.h


```
 68:         __u16 s_magic;                  /* magic number */
 69:         __u16 s_state;                  /* filesystem state */
 70:         __u32 s_zones;                  /* number of data zones (for v2 only) */
 71: };
 72:
 73: struct minix_inode {
 74:         __u16 i_mode;
 75:         __u16 i_uid;
 76:         __u32 i_size;
 77:         __u32 i_time;
 78:         __u8  i_gid;
 79:         __u8  i_nlinks;
 80:         __u16 i_zone[9];
 81: };
 82:
 83: struct minix2_inode {
 84:         __u16 i_mode;
 85:         __u16 i_nlink;
 86:         __u16 i_uid;
 87:         __u16 i_gid;
 88:         __u32 i_size;
 89:         __u32 i_atime;
 90:         __u32 i_mtime;
 91:         __u32 i_ctime;
 92:         __u32 i_zone[10];
 93: };
 94:
 95: struct minix_dir_entry {
 96:         __u16 inode;
 97:         char name[0];
 98: };
 99:
100: /* super block in memory */
101: struct minix_sb_info {
102:         unsigned char namelen;
103:         unsigned char dirsize;
104:         unsigned short int version;
105:         unsigned int nzones;
106:         struct minix_super_block sb;
107: };
108:
109: /* inode in memory */
110: struct minix_i_info {
111:         union {
112:                 __u16 i1_zone[9];
113:                 __u32 i2_zone[10];
114:         } u;
115: };
116:
117: int v1_minix_read_inode(struct inode *);
118: int v1_minix_write_inode(struct inode *);
119: int v1_minix_ialloc(struct inode *, int);
120: void v1_minix_ifree(struct inode *);
121: int v1_minix_bmap(struct inode *, __off_t, int);
122: int v1_minix_truncate(struct inode *, __off_t);
123:
124: int v2_minix_read_inode(struct inode *);
125: int v2_minix_write_inode(struct inode *);
126: int v2_minix_ialloc(struct inode *, int);
127: void v2_minix_ifree(struct inode *);
128: int v2_minix_bmap(struct inode *, __off_t, int);
129: int v2_minix_truncate(struct inode *, __off_t);
130:
131: #endif /* _FIWIX_FS_MINIX_H */
```

include/fiwix/fs_pipe.h


```
 1: /*
 2:  * fiwix/include/fiwix/fs_pipe.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_FS_PIPE_H
 9: #define _FIWIX_FS_PIPE_H
10:
11: #define PIPE_DEV        0xFFF0          /* special device number for nodev fs */
12:
13: extern struct fs_operations pipefs_fsop;
14:
15: struct pipefs_inode {
16:         char *i_data;                   /* buffer */
17:         unsigned int i_readoff;         /* offset for reads */
18:         unsigned int i_writeoff;        /* offset for writes */
19:         unsigned int i_readers;         /* number of readers */
20:         unsigned int i_writers;         /* number of writers */
21: };
22:
23: #endif /* _FIWIX_FS_PIPE_H */
```

include/fiwix/fs_proc.h


```
 1: /*
 2:  * fiwix/include/fiwix/fs_proc.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_FS_PROC_H
 9: #define _FIWIX_FS_PROC_H
10:
11: #include <fiwix/types.h>
12:
13: #define PROC_DEV                0xFFF1  /* special device number for nodev fs */
14: #define PROC_ROOT_INO           1       /* root inode */
15: #define PROC_SUPER_MAGIC        0x9FA0  /* same as in Linux */
16:
17: #define PROC_PID_INO            0x40000000      /* base for PID inodes */
18: #define PROC_PID_LEV            1       /* array level for PID */
19:
20: #define PROC_ARRAY_ENTRIES      20
21:
22: enum pid_dir_inodes {
23:         PROC_PID_FD = PROC_PID_INO + 1001,
24:         PROC_PID_CMDLINE,
25:         PROC_PID_CWD,
26:         PROC_PID_ENVIRON,
27:         PROC_PID_EXE,
28:         PROC_PID_MAPS,
29:         PROC_PID_MOUNTINFO,
30:         PROC_PID_ROOT,
31:         PROC_PID_STAT,
32:         PROC_PID_STATM,
33:         PROC_PID_STATUS
34: };
35:
36: struct procfs_inode {
37:         unsigned int i_lev;             /* array level (directory depth) */
38: };
39:
40: struct procfs_dir_entry {
41:         __ino_t inode;
42:         __mode_t mode;
43:         __nlink_t nlink;
44:         int lev;                        /* array level (directory depth) */
45:         unsigned short int name_len;
46:         char *name;
47:         int (*data_fn)(char *, __pid_t);
48: };
49:
50: extern struct procfs_dir_entry procfs_array[][PROC_ARRAY_ENTRIES + 1];
51:
52: int data_proc_cmdline(char *, __pid_t);
53: int data_proc_cpuinfo(char *, __pid_t);
54: int data_proc_devices(char *, __pid_t);
55: int data_proc_dma(char *, __pid_t);
56: int data_proc_filesystems(char *, __pid_t);
57: int data_proc_interrupts(char *, __pid_t);
58: int data_proc_loadavg(char *, __pid_t);
59: int data_proc_locks(char *, __pid_t);
60: int data_proc_meminfo(char *, __pid_t);
61: int data_proc_mounts(char *, __pid_t);
62: int data_proc_partitions(char *, __pid_t);
63: int data_proc_rtc(char *, __pid_t);
64: int data_proc_self(char *, __pid_t);
65: int data_proc_stat(char *, __pid_t);
66: int data_proc_uptime(char *, __pid_t);
67: int data_proc_fullversion(char *, __pid_t);
```

include/fiwix/fs_proc.h


```
68: int data_proc_domainname(char *, __pid_t);
69: int data_proc_filemax(char *, __pid_t);
70: int data_proc_filenr(char *, __pid_t);
71: int data_proc_hostname(char *, __pid_t);
72: int data_proc_inodemax(char *, __pid_t);
73: int data_proc_inodenr(char *, __pid_t);
74: int data_proc_osrelease(char *, __pid_t);
75: int data_proc_ostype(char *, __pid_t);
76: int data_proc_version(char *, __pid_t);
77:
78: /* PID related functions */
79: int data_proc_pid_fd(char *, __pid_t);
80: int data_proc_pid_cmdline(char *, __pid_t);
81: int data_proc_pid_cwd(char *, __pid_t);
82: int data_proc_pid_environ(char *, __pid_t);
83: int data_proc_pid_exe(char *, __pid_t);
84: int data_proc_pid_maps(char *, __pid_t);
85: int data_proc_pid_mountinfo(char *, __pid_t);
86: int data_proc_pid_root(char *, __pid_t);
87: int data_proc_pid_stat(char *, __pid_t);
88: int data_proc_pid_statm(char *, __pid_t);
89: int data_proc_pid_status(char *, __pid_t);
90:
91: #endif /* _FIWIX_FS_PROC_H */
```

include/fiwix/i386elf.h


```
    1: /*
    2:  * fiwix/include/fiwix/i386elf.h
    3:  */
    4:
    5: #ifndef _FIWIX_ELF_H
    6: #define _FIWIX_ELF_H
    7:
    8: typedef unsigned long   Elf32_Addr;
    9: typedef unsigned short  Elf32_Half;
   10: typedef unsigned long   Elf32_Off;
   11: typedef long            Elf32_Sword;
   12: typedef unsigned long   Elf32_Word;
   13:
   14: #define ELFMAG0         0x7f            /* EI_MAG */
   15: #define ELFMAG1         'E'
   16: #define ELFMAG2         'L'
   17: #define ELFMAG3         'F'
   18: #define ELFMAG          "\177ELF"
   19: #define SELFMAG         4
   20:
   21: #define EI_NIDENT       16
   22:
   23: typedef struct elf32_hdr{
   24:   unsigned char e_ident[EI_NIDENT];     /* ELF "magic number" */
   25:   Elf32_Half    e_type;                 /* File type */
   26:   Elf32_Half    e_machine;              /* Target machine */
   27:   Elf32_Word    e_version;              /* File version */
   28:   Elf32_Addr    e_entry;                /* Entry point virtual address */
   29:   Elf32_Off     e_phoff;                /* Program header table file offset */
   30:   Elf32_Off     e_shoff;                /* Section header table file offset */
   31:   Elf32_Word    e_flags;                /* File flags */
   32:   Elf32_Half    e_ehsize;               /* Sizeof Ehdr (ELF header) */
   33:   Elf32_Half    e_phentsize;            /* Sizeof Phdr (Program header) */
   34:   Elf32_Half    e_phnum;                /* Number Phdrs (Program header) */
   35:   Elf32_Half    e_shentsize;            /* Sizeof Shdr (Section header) */
   36:   Elf32_Half    e_shnum;                /* Number Shdrs (Section header) */
   37:   Elf32_Half    e_shstrndx;             /* Shdr string index */
   38: } Elf32_Ehdr;
   39:
   40: #define EI_MAG0         0               /* e_ident[] indexes */
   41: #define EI_MAG1         1
   42: #define EI_MAG2         2
   43: #define EI_MAG3         3
   44: #define EI_CLASS        4
   45: #define EI_DATA         5
   46: #define EI_VERSION      6
   47: #define EI_PAD          7
   48:
   49: #define ELFCLASSNONE    0               /* EI_CLASS */
   50: #define ELFCLASS32      1
   51: #define ELFCLASS64      2
   52: #define ELFCLASSNUM     3
   53:
   54: #define ELFDATANONE     0               /* e_ident[EI_DATA] */
   55: #define ELFDATA2LSB     1
   56: #define ELFDATA2MSB     2
   57: #define ELFDATANUM      3
   58:
   59: /* ELF file types */
   60: #define ET_NONE   0
   61: #define ET_REL    1
   62: #define ET_EXEC   2
   63: #define ET_DYN    3
   64: #define ET_CORE   4
   65: #define ET_LOPROC 5
   66: #define ET_HIPROC 6
   67:
```

include/fiwix/i386elf.h


```
 68: #define EM_386    3
 69:
 70: #define EV_NONE         0               /* e_version, EI_VERSION */
 71: #define EV_CURRENT      1
 72: #define EV_NUM          2
 73:
 74: typedef struct elf32_phdr{
 75:   Elf32_Word    p_type;                 /* Entry type */
 76:   Elf32_Off     p_offset;               /* File offset */
 77:   Elf32_Addr    p_vaddr;                /* Virtual address */
 78:   Elf32_Addr    p_paddr;                /* Physical address */
 79:   Elf32_Word    p_filesz;               /* File size */
 80:   Elf32_Word    p_memsz;                /* Memory size */
 81:   Elf32_Word    p_flags;                /* Entry flags */
 82:   Elf32_Word    p_align;                /* Memory & file alignment */
 83: } Elf32_Phdr;
 84:
 85: /* segment types stored in the image headers */
 86: #define PT_NULL    0
 87: #define PT_LOAD    1
 88: #define PT_DYNAMIC 2
 89: #define PT_INTERP  3
 90: #define PT_NOTE    4
 91: #define PT_SHLIB   5
 92: #define PT_PHDR    6
 93: #define PT_NUM     7
 94: #define PT_LOPROC  0x70000000
 95: #define PT_HIPROC  0x7fffffff
 96:
 97: /* permission types on sections in the program header, p_flags. */
 98: #define PF_R    0x4
 99: #define PF_W    0x2
100: #define PF_X    0x1
101:
102: #define PF_MASKPROC     0xf0000000
103:
104: typedef struct {
105:   Elf32_Word    sh_name;                /* Section name, index in string tbl */
106:   Elf32_Word    sh_type;                /* Type of section */
107:   Elf32_Word    sh_flags;               /* Miscellaneous section attributes */
108:   Elf32_Addr    sh_addr;                /* Section virtual addr at execution */
109:   Elf32_Off     sh_offset;              /* Section file offset */
110:   Elf32_Word    sh_size;                /* Size of section in bytes */
111:   Elf32_Word    sh_link;                /* Index of another section */
112:   Elf32_Word    sh_info;                /* Additional section information */
113:   Elf32_Word    sh_addralign;           /* Section alignment */
114:   Elf32_Word    sh_entsize;             /* Entry size if section holds table */
115: } Elf32_Shdr;
116:
117: /* sh_type */
118: #define SHT_NULL        0
119: #define SHT_PROGBITS    1
120: #define SHT_SYMTAB      2
121: #define SHT_STRTAB      3
122: #define SHT_RELA        4
123: #define SHT_HASH        5
124: #define SHT_DYNAMIC     6
125: #define SHT_NOTE        7
126: #define SHT_NOBITS      8
127: #define SHT_REL         9
128: #define SHT_SHLIB       10
129: #define SHT_DYNSYM      11
130: #define SHT_NUM         12
131:
132: #define SHT_LOPROC      0x70000000
133: #define SHT_HIPROC      0x7fffffff
134: #define SHT_LOUSER      0x80000000
```

include/fiwix/i386elf.h


```
135: #define SHT_HIUSER      0xffffffff
136:
137: /* sh_flags */
138: #define SHF_WRITE       0x1
139: #define SHF_ALLOC       0x2
140: #define SHF_EXECINSTR   0x4
141:
142: /* special section indexes */
143: #define SHN_UNDEF       0
144: #define SHN_LORESERVE   0xff00
145: #define SHN_ABS         0xfff1
146: #define SHN_COMMON      0xfff2
147: #define SHN_HIRESERVE   0xffff
148: #define SHN_LOPROC      0xff00
149: #define SHN_HIPROC      0xff1f
150:
151: typedef struct elf32_sym{
152:   Elf32_Word    st_name;                /* Symbol name, index in string tbl */
153:   Elf32_Addr    st_value;               /* Value of the symbol */
154:   Elf32_Word    st_size;                /* Associated symbol size */
155:   unsigned char st_info;                /* Type and binding attributes */
156:   unsigned char st_other;               /* No defined meaning, 0 */
157:   Elf32_Half    st_shndx;               /* Associated section index */
158: } Elf32_Sym;
159:
160: #define ELF32_ST_BIND(info)     ((info) >> 4)
161: #define ELF32_ST_TYPE(info)     (((unsigned int) info) & 0xf)
162:
163: /* This info is needed when parsing the symbol table */
164: #define STB_LOCAL  0
165: #define STB_GLOBAL 1
166: #define STB_WEAK   2
167: #define STB_NUM    3
168:
169: #define STT_NOTYPE  0
170: #define STT_OBJECT  1
171: #define STT_FUNC    2
172: #define STT_SECTION 3
173: #define STT_FILE    4
174: #define STT_NUM     5
175:
176: typedef struct elf32_rel {
177:   Elf32_Addr    r_offset;       /* Location at which to apply the action */
178:   Elf32_Word    r_info;         /* Index and type of relocation */
179: } Elf32_Rel;
180:
181: typedef struct elf32_rela{
182:   Elf32_Addr    r_offset;       /* Location at which to apply the action */
183:   Elf32_Word    r_info;         /* Index and type of relocation */
184:   Elf32_Sword   r_addend;       /* Constant addend used to compute value */
185: } Elf32_Rela;
186:
187: /* The following are used with relocations */
188: #define ELF32_R_SYM(info)       ((info) >> 8)
189: #define ELF32_R_TYPE(info)      ((info) & 0xff)
190:
191: /* This is the info that is needed to parse the dynamic section of the file */
192: #define DT_NULL         0
193: #define DT_NEEDED       1
194: #define DT_PLTRELSZ     2
195: #define DT_PLTGOT       3
196: #define DT_HASH         4
197: #define DT_STRTAB       5
198: #define DT_SYMTAB       6
199: #define DT_RELA         7
200: #define DT_RELASZ       8
201: #define DT_RELAENT      9
```

include/fiwix/i386elf.h


```
202: #define DT_STRSZ        10
203: #define DT_SYMENT       11
204: #define DT_INIT         12
205: #define DT_FINI         13
206: #define DT_SONAME       14
207: #define DT_RPATH        15
208: #define DT_SYMBOLIC     16
209: #define DT_REL          17
210: #define DT_RELSZ        18
211: #define DT_RELENT       19
212: #define DT_PLTREL       20
213: #define DT_DEBUG        21
214: #define DT_TEXTREL      22
215: #define DT_JMPREL       23
216: #define DT_LOPROC       0x70000000
217: #define DT_HIPROC       0x7fffffff
218:
219: /* Symbolic values for the entries in the auxiliary table
220:    put on the initial stack */
221: #define AT_NULL   0     /* end of vector */
222: #define AT_IGNORE 1     /* entry should be ignored */
223: #define AT_EXECFD 2     /* file descriptor of program */
224: #define AT_PHDR   3     /* program headers for program */
225: #define AT_PHENT  4     /* size of program header entry */
226: #define AT_PHNUM  5     /* number of program headers */
227: #define AT_PAGESZ 6     /* system page size */
228: #define AT_BASE   7     /* base address of interpreter */
229: #define AT_FLAGS  8     /* flags */
230: #define AT_ENTRY  9     /* entry point of program */
231: #define AT_NOTELF 10    /* program is not ELF */
232: #define AT_UID    11    /* real uid */
233: #define AT_EUID   12    /* effective uid */
234: #define AT_GID    13    /* real gid */
235: #define AT_EGID   14    /* effective gid */
236:
237:
238: typedef struct dynamic{
239:   Elf32_Sword d_tag;                    /* entry tabg value */
240:   union{
241:     Elf32_Sword d_val;
242:     Elf32_Addr  d_ptr;
243:   } d_un;
244: } Elf32_Dyn;
245:
246: #define R_386_NONE      0
247: #define R_386_32        1
248: #define R_386_PC32      2
249: #define R_386_GOT32     3
250: #define R_386_PLT32     4
251: #define R_386_COPY      5
252: #define R_386_GLOB_DAT  6
253: #define R_386_JMP_SLOT  7
254: #define R_386_RELATIVE  8
255: #define R_386_GOTOFF    9
256: #define R_386_GOTPC     10
257: #define R_386_NUM       11
258:
259: /* Notes used in ET_CORE */
260: #define NT_PRSTATUS     1
261: #define NT_PRFPREG      2
262: #define NT_PRPSINFO     3
263: #define NT_TASKSTRUCT   4
264:
265: /* Note header in a PT_NOTE section */
266: typedef struct elf32_note {
267:   Elf32_Word    n_namesz;       /* Name size */
268:   Elf32_Word    n_descsz;       /* Content size */
```

include/fiwix/i386elf.h


```
269:   Elf32_Word    n_type;         /* Content type */
270: } Elf32_Nhdr;
271:
272: #define ELF_START_MMAP 0x80000000
273:
274: extern Elf32_Dyn _DYNAMIC [];
275: #define elfhdr          elf32_hdr
276: #define elf_phdr        elf32_phdr
277: #define elf_note        elf32_note
278:
279: #endif /* _FIWIX_ELF_H */
```

include/fiwix/ide_cd.h


```
 1: /*
 2:  * fiwix/include/fiwix/ide_cd.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_IDE_CD_H
 9: #define _FIWIX_IDE_CD_H
10:
11: #include <fiwix/fs.h>
12:
13: #define IDE_CD_SECTSIZE         BLKSIZE_2K      /* sector size (in bytes) */
14:
15: void ide_cd_timer(unsigned int);
16:
17: int ide_cd_open(struct inode *, struct fd *);
18: int ide_cd_close(struct inode *, struct fd *);
19: int ide_cd_read(__dev_t, __blk_t, char *, int);
20: int ide_cd_ioctl(struct inode *, int, unsigned long int);
21:
22: int ide_cd_init(struct ide *, int);
23:
24: #endif /* _FIWIX_IDE_CD_H */
```

include/fiwix/ide.h


```
/*
 * fiwix/include/fiwix/ide.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_IDE_H
#define _FIWIX_IDE_H

#include <fiwix/fs.h>
#include <fiwix/part.h>

#define IDE0_IRQ                14      /* primary controller interrupt */
#define IDE1_IRQ                15      /* secondary controller interrupt */

#define IDE0_MAJOR              3       /* 1st controller major number */
#define IDE1_MAJOR              22      /* 2nd controller major number */
#define IDE_MINORS              4       /* max. minors/partitions per unit */
#define IDE_MASTER_MSF          0       /* IDE master minor shift factor */
#define IDE_SLAVE_MSF           6       /* IDE slave minor shift factor */

#define IDE_PRIMARY             0
#define IDE_SECONDARY           1
#define IDE_MASTER              0
#define IDE_SLAVE               1
#define IDE_ATA                 0
#define IDE_ATAPI               1

#define NR_IDE_CTRLS            2       /* IDE controllers */
#define NR_IDE_DRVS             2       /* max. drives per IDE controller */

/* controller addresses */
#define IDE0_BASE               0x1F0   /* primary controller base addr */
#define IDE0_CTRL               0x3F4   /* primary controller control port */
#define IDE1_BASE               0x170   /* secondary controller base addr */
#define IDE1_CTRL               0x374   /* secondary controller control port */

#define IDE_BASE_LEN            7       /* controller address length */

#define IDE_RDY_RETR_LONG       50000   /* long delay for fast CPUs */
#define IDE_RDY_RETR_SHORT      500     /* short delay for slow CPUs */
#define MAX_IDE_ERR             10      /* number of retries */
#define MAX_CD_ERR              5       /* number of retries in CDROMs */

#define SET_IDE_RDY_RETR(retries)                                       \
        if((cpu_table.hz / 1000000) <= 100) {                           \
                retries = IDE_RDY_RETR_SHORT;                           \
        } else {                                                        \
                retries = IDE_RDY_RETR_LONG;                            \
        }

#define WAIT_FOR_IDE    (1 * HZ)        /* timeout for hard disk */
#define WAIT_FOR_CD     (3 * HZ)        /* timeout for cdrom */

/* controller registers */
#define IDE_DATA                0x0     /* Data Port Register (R/W) */
#define IDE_ERROR               0x1     /* Error Register (R) */
#define IDE_FEATURES            0x1     /* Features Register (W) */
#define IDE_SECCNT              0x2     /* Sector Count Register (R/W) */
#define IDE_SECNUM              0x3     /* Sector Number Register (R/W) */
#define IDE_LCYL                0x4     /* Cylinder Low Register (R/W) */
#define IDE_HCYL                0x5     /* Cylinder High Register (R/W) */
#define IDE_DRVHD               0x6     /* Drive/Head Register (R/W) */
#define IDE_STATUS              0x7     /* Status Register (R) */
#define IDE_COMMAND             0x7     /* Command Register (W) */

```

include/fiwix/ide.h


```
 68: #define IDE_ALT_STATUS          0x2     /* Alternate Register (R) */
 69: #define IDE_DEV_CTRL            0x2     /* Device Control Register (W) */
 70:
 71: /* error register bits */
 72: #define IDE_ERR_AMNF            0x01    /* Address Mark Not Found */
 73: #define IDE_ERR_TK0NF           0x02    /* Track 0 Not Found */
 74: #define IDE_ERR_ABRT            0x04    /* Aborted Command */
 75: #define IDE_ERR_MCR             0x08    /* Media Change Registered */
 76: #define IDE_ERR_IDNF            0x10    /* Sector ID Field Not Found */
 77: #define IDE_ERR_MC              0x20    /* Media Changed */
 78: #define IDE_ERR_UNC             0x40    /* Uncorrectable Data Error */
 79: #define IDE_ERR_BBK             0x80    /* Bad Block */
 80:
 81: /* status register bits */
 82: #define IDE_STAT_ERR            0x01    /* an error ocurred */
 83: #define IDE_STAT_SENS           0x02    /* sense data available */
 84: #define IDE_STAT_CORR           0x04    /* a correctable error ocurred */
 85: #define IDE_STAT_DRQ            0x08    /* device is ready to transfer */
 86: #define IDE_STAT_DSC            0x10    /* device requests service o intr. */
 87: #define IDE_STAT_DWF            0x20    /* drive write fault */
 88: #define IDE_STAT_RDY            0x40    /* drive is ready */
 89: #define IDE_STAT_BSY            0x80    /* drive is busy */
 90:
 91: #define IDE_CHS_MODE            0xA0    /* select CHS mode */
 92: #define IDE_LBA_MODE            0xE0    /* select LBA mode */
 93:
 94: /* alternate & device control register bits */
 95: #define IDE_DEVCTR_DRQ          0x08    /* Data Request */
 96: #define IDE_DEVCTR_NIEN         0x02    /* Disable Interrupt */
 97: #define IDE_DEVCTR_SRST         0x04    /* Software Reset */
 98:
 99: /* ATA commands */
100: #define ATA_READ_PIO            0x20    /* read sector(s) with retries */
101: #define ATA_READ_MULTIPLE_PIO   0xC4    /* read multiple sectors */
102: #define ATA_WRITE_PIO           0x30    /* write sector(s) with retries */
103: #define ATA_WRITE_MULTIPLE_PIO  0xC5    /* write multiple sectors */
104: #define ATA_SET_MULTIPLE_MODE   0xC6
105: #define ATA_PACKET              0xA0
106: #define ATA_IDENTIFY_PACKET     0xA1    /* identify ATAPI device */
107: #define ATA_IDENTIFY            0xEC    /* identify ATA device */
108:
109: /* ATAPI commands */
110: #define ATAPI_TEST_UNIT         0x00
111: #define ATAPI_REQUEST_SENSE     0x03
112: #define ATAPI_START_STOP        0x1B
113: #define ATAPI_MEDIUM_REMOVAL    0x1E
114: #define ATAPI_READ10            0x28
115:
116: #define CD_UNLOCK_MEDIUM        0x00    /* allow medium removal */
117: #define CD_LOCK_MEDIUM          0x01    /* prevent medium removal */
118: #define CD_EJECT                0x02    /* eject the CD if possible */
119: #define CD_LOAD                 0x03    /* load the CD (closes tray) */
120:
121: /* ATAPI CD additional sense code */
122: #define ASC_NOT_READY           0x04
123: #define ASC_NO_MEDIUM           0x3A
124:
125: /* capabilities */
126: #define IDE_SUPPORTS_CFA        0x848A
127: #define IDE_HAS_DMA             0x100
128: #define IDE_HAS_LBA             0x200
129: #define IDE_MIN_LBA             16514064/* sectors limit for using CHS */
130:
131: /* general configuration bits */
132: #define IDE_HAS_UDMA            0x04    /* device supports UDMA */
133: #define IDE_REMOVABLE           0x80    /* removable media device */
134:
```

include/fiwix/ide.h


```
  135: /* ATAPI types */
  136: #define ATAPI_IS_SEQ_ACCESS     0x01    /* sequential-access device */
  137: #define ATAPI_IS_PRINTER        0x02
  138: #define ATAPI_IS_PROCESSOR      0x03
  139: #define ATAPI_IS_WRITE_ONCE     0x04
  140: #define ATAPI_IS_CDROM          0x05
  141: #define ATAPI_IS_SCANNER        0x06
  142:
  143: /* IDE drive flags */
  144: #define DEVICE_IS_ATAPI         0x01
  145: #define DEVICE_IS_CFA           0x02
  146: #define DEVICE_IS_DISK          0x04
  147: #define DEVICE_IS_CDROM         0x08
  148: #define DEVICE_REQUIRES_LBA     0x10
  149: #define DEVICE_HAS_RW_MULTIPLE  0x20
  150:
  151: /* ATA/ATAPI-4 based */
  152: struct ide_drv_ident {
  153:         unsigned short int gen_config;          /* general configuration bits */
  154:         unsigned short int logic_cyls;          /* logical cylinders */
  155:         unsigned short int reserved2;
  156:         unsigned short int logic_heads;         /* logical heads */
  157:         unsigned short int retired4;
  158:         unsigned short int retired5;
  159:         unsigned short int logic_spt;           /* logical sectors/track */
  160:         unsigned short int retired7;
  161:         unsigned short int retired8;
  162:         unsigned short int retired9;
  163:         char serial_number[20];                 /* serial number */
  164:         unsigned short int vendor_spec20;
  165:         unsigned short int buffer_cache;
  166:         unsigned short int vendor_spec22;       /* reserved */
  167:         char firmware_rev[8];                   /* firmware version */
  168:         char model_number[40];                  /* model number */
  169:         unsigned short int rw_multiple;
  170:         unsigned short int reserved48;
  171:         unsigned short int capabilities;        /* capabilities */
  172:         unsigned short int reserved50;
  173:         unsigned short int pio_mode;            /* PIO data transfer mode*/
  174:         unsigned short int dma_mode;
  175:         unsigned short int fields_validity;     /* fields validity */
  176:         unsigned short int cur_log_cyls;        /* current logical cylinders */
  177:         unsigned short int cur_log_heads;       /* current logical heads */
  178:         unsigned short int cur_log_spt;         /* current logical sectors/track
 */
  179:         unsigned short int cur_capacity;        /* current capacity in sectors *
/
  180:         unsigned short int cur_capacity2;       /* 32bit number */
  181:         unsigned short int mult_sect_set;       /* multiple sector settings */
  182:         unsigned short int tot_sectors;         /* sectors (LBA only) */
  183:         unsigned short int tot_sectors2;        /* 32bit number */
  184:         unsigned short int singleword_dma;
  185:         unsigned short int multiword_dma;       /* multiword DMA settings */
  186:         unsigned short int adv_pio_modes;       /* advanced PIO modes */
  187:         unsigned short int min_multiword;       /* min. Multiword DMA transfer *
/
  188:         unsigned short int rec_multiword;       /* recommended Multiword DMS tra
nsfer */
  189:         unsigned short int min_pio_wo_fc;       /* min. PIO w/o flow control */
  190:         unsigned short int min_pio_w_fc;        /* min. PIO with flow control */
  191:         unsigned short int reserved69;
  192:         unsigned short int reserved70;
  193:         unsigned short int reserved71;
  194:         unsigned short int reserved72;
  195:         unsigned short int reserved73;
  196:         unsigned short int reserved74;
  197:         unsigned short int queue_depth;         /* queue depth */
```

include/fiwix/ide.h


```
  198:         unsigned short int reserved76;
  199:         unsigned short int reserved77;
  200:         unsigned short int reserved78;
  201:         unsigned short int reserved79;
  202:         unsigned short int majorver;            /* major version number */
  203:         unsigned short int minorver;            /* minor version number */
  204:         unsigned short int cmdset1;             /* command set supported */
  205:         unsigned short int cmdset2;             /* command set supported */
  206:         unsigned short int cmdsf_ext;           /* command set/feature sup.ext.
*/
  207:         unsigned short int cmdsf_enable1;       /* command s/f enabled */
  208:         unsigned short int cmdsf_enable2;       /* command s/f enabled */
  209:         unsigned short int cmdsf_default;       /* command s/f default */
  210:         unsigned short int ultradma;            /* ultra DMA mode */
  211:         unsigned short int reserved89;
  212:         unsigned short int reserved90;
  213:         unsigned short int curapm;              /* current APM values */
  214:         unsigned short int reserved92_126[35];
  215:         unsigned short int r_status_notif;      /* removable media status notif.
 */
  216:         unsigned short int security_status;     /* security status */
  217:         unsigned short int vendor_spec129_159[31];
  218:         unsigned short int reserved160_255[96];
  219: };
  220:
  221: struct ide_drv {
  222:         int drive;                      /* master or slave */
  223:         char *dev_name;
  224:         unsigned char major;            /* major number */
  225:         unsigned int flags;
  226:         int minor_shift;                /* shift factor to get the real minor */
  227:         int lba_cyls;
  228:         int lba_heads;
  229:         short int lba_factor;
  230:         unsigned int nr_sects;          /* total sectors (LBA) */
  231:         struct fs_operations *fsop;
  232:         struct ide_drv_ident ident;
  233:         struct partition part_table[NR_PARTITIONS];
  234: };
  235:
  236: struct ide {
  237:         int channel;                    /* primary or secondary */
  238:         int base;                       /* base address */
  239:         int ctrl;                       /* control port address */
  240:         short int irq;
  241:         struct ide_drv drive[NR_IDE_DRVS];
  242: };
  243:
  244: extern struct ide ide_table[NR_IDE_CTRLS];
  245:
  246: extern int ide0_need_reset;
  247: extern int ide0_wait_interrupt;
  248: extern int ide0_timeout;
  249: extern int ide1_need_reset;
  250: extern int ide1_wait_interrupt;
  251: extern int ide1_timeout;
  252:
  253: void irq_ide0(void);
  254: void ide0_timer(unsigned int);
  255: void irq_ide1(void);
  256: void ide1_timer(unsigned int);
  257:
  258: void ide_error(struct ide *, int);
  259: void ide_delay(void);
  260: void ide_wait400ns(struct ide *);
  261: int ide_ready(struct ide *);
  262: int ide_drvsel(struct ide *, int, int, unsigned char);
```

include/fiwix/ide.h


```
263: int ide_softreset(struct ide *);
264:
265: struct ide * get_ide_controller(__dev_t);
266: int get_ide_drive(__dev_t);
267:
268: int ide_open(struct inode *, struct fd *);
269: int ide_close(struct inode *, struct fd *);
270: int ide_read(__dev_t, __blk_t, char *, int);
271: int ide_write(__dev_t, __blk_t, char *, int);
272: int ide_ioctl(struct inode *, int, unsigned long int);
273:
274: void ide_init(void);
275:
276: #endif /* _FIWIX_IDE_H */
```

include/fiwix/ide_hd.h


```
 1: /*
 2:  * fiwix/include/fiwix/ide_hd.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_IDE_HD_H
 9: #define _FIWIX_IDE_HD_H
10:
11: #include <fiwix/types.h>
12:
13: #define IDE_HD_SECTSIZE         512     /* sector size (in bytes) */
14:
15: int ide_hd_open(struct inode *, struct fd *);
16: int ide_hd_close(struct inode *, struct fd *);
17: int ide_hd_read(__dev_t, __blk_t, char *, int);
18: int ide_hd_write(__dev_t, __blk_t, char *, int);
19: int ide_hd_ioctl(struct inode *, int, unsigned long int);
20:
21: int ide_hd_init(struct ide *, int);
22:
23: #endif /* _FIWIX_IDE_HD_H */
```

include/fiwix/ioctl.h


```
    1: /*
    2:  * fiwix/include/fiwix/ioctl.h
    3:  */
    4:
    5: #ifndef _FIWIX_IOCTL_H
    6: #define _FIWIX_IOCTL_H
    7:
    8: #define HDIO_GETGEO     0x0301          /* get device geometry */
    9:
   10: #define BLKRRPART       0x125F          /* re-read partition table */
   11: #define BLKGETSIZE      0x1260          /* return device size */
   12: #define BLKFLSBUF       0x1261          /* flush buffer cache */
   13:
   14: /* 0x54 is just a magic number to make these relatively unique ('T') */
   15: #define TCGETS          0x5401
   16: #define TCSETS          0x5402
   17: #define TCSETSW         0x5403
   18: #define TCSETSF         0x5404
   19: #define TCGETA          0x5405
   20: #define TCSETA          0x5406
   21: #define TCSETAW         0x5407
   22: #define TCSETAF         0x5408
   23: #define TCSBRK          0x5409
   24: #define TCXONC          0x540A
   25: #define TCFLSH          0x540B
   26: #define TIOCEXCL        0x540C
   27: #define TIOCNXCL        0x540D
   28: #define TIOCSCTTY       0x540E
   29: #define TIOCGPGRP       0x540F
   30: #define TIOCSPGRP       0x5410
   31: #define TIOCOUTQ        0x5411
   32: #define TIOCSTI         0x5412
   33: #define TIOCGWINSZ      0x5413
   34: #define TIOCSWINSZ      0x5414
   35: #define TIOCMGET        0x5415
   36: #define TIOCMBIS        0x5416
   37: #define TIOCMBIC        0x5417
   38: #define TIOCMSET        0x5418
   39: #define TIOCGSOFTCAR    0x5419
   40: #define TIOCSSOFTCAR    0x541A
   41: #define FIONREAD        0x541B
   42: #define TIOCINQ         FIONREAD
   43: #define TIOCLINUX       0x541C
   44: #define TIOCCONS        0x541D
   45: #define TIOCGSERIAL     0x541E
   46: #define TIOCSSERIAL     0x541F
   47: #define TIOCPKT         0x5420
   48: #define FIONBIO         0x5421
   49: #define TIOCNOTTY       0x5422
   50: #define TIOCSETD        0x5423
   51: #define TIOCGETD        0x5424
   52: #define TCSBRKP         0x5425  /* Needed for POSIX tcsendbreak() */
   53: #define TIOCTTYGSTRUCT  0x5426  /* For debugging only */
   54: #define TIOCSBRK        0x5427  /* BSD compatibility */
   55: #define TIOCCBRK        0x5428  /* BSD compatibility */
   56: #define TIOCGSID        0x5429  /* Return the session ID of FD */
   57: #define TIOCGPTN        _IOR('T',0x30, unsigned int) /* Get Pty Number (of pty-m
ux device) */
   58: #define TIOCSPTLCK      _IOW('T',0x31, int)  /* Lock/unlock Pty */
   59:
   60: #define FIONCLEX        0x5450  /* these numbers need to be adjusted. */
   61: #define FIOCLEX         0x5451
   62: #define FIOASYNC        0x5452
   63: #define TIOCSERCONFIG   0x5453
   64: #define TIOCSERGWILD    0x5454
   65: #define TIOCSERSWILD    0x5455
   66: #define TIOCGLCKTRMIOS  0x5456
```

include/fiwix/ioctl.h


```
67: #define TIOCSLCKTRMIOS  0x5457
68: #define TIOCSERGSTRUCT  0x5458  /* For debugging only */
69: #define TIOCSERGETLSR   0x5459  /* Get line status register */
70: #define TIOCSERGETMULTI 0x545A  /* Get multiport config  */
71: #define TIOCSERSETMULTI 0x545B  /* Set multiport config */
72:
73: #define TIOCMIWAIT      0x545C  /* wait for a change on serial input line(s) */
74: #define TIOCGICOUNT     0x545D  /* read serial port inline interrupt counts */
75: #define TIOCGHAYESESP   0x545E  /* Get Hayes ESP configuration */
76: #define TIOCSHAYESESP   0x545F  /* Set Hayes ESP configuration */
77:
78: /* Used for packet mode */
79: #define TIOCPKT_DATA             0
80: #define TIOCPKT_FLUSHREAD        1
81: #define TIOCPKT_FLUSHWRITE       2
82: #define TIOCPKT_STOP             4
83: #define TIOCPKT_START            8
84: #define TIOCPKT_NOSTOP          16
85: #define TIOCPKT_DOSTOP          32
86:
87: #define TIOCSER_TEMT    0x01    /* Transmitter physically empty */
88:
89: #endif /* _FIWIX_IOCTL_H */
```

include/fiwix/kd.h


```
 1: /*
 2:  * fiwix/include/fiwix/kd.h
 3:  */
 4:
 5: #ifndef _LINUX_KD_H
 6: #define _LINUX_KD_H
 7:
 8: /* Prefix 0x4B is 'K', to avoid collision with termios and vt */
 9:
10: #define GIO_FONT        0x4B60  /* gets font in expanded form */
11: #define PIO_FONT        0x4B61  /* use font in expanded form */
12:
13: #define GIO_FONTX       0x4B6B  /* get font using struct consolefontdesc */
14: #define PIO_FONTX       0x4B6C  /* set font using struct consolefontdesc */
15: struct consolefontdesc {
16:         unsigned short int charcount;   /* characters in font (256 or 512) */
17:         unsigned short int charheight;  /* scan lines per character (1-32) */
18:         char *chardata;                 /* font data in expanded form */
19: };
20:
21: #define PIO_FONTRESET   0x4B6D  /* reset to default font */
22:
23: #define GIO_CMAP        0x4B70  /* gets colour palette on VGA+ */
24: #define PIO_CMAP        0x4B71  /* sets colour palette on VGA+ */
25:
26: #define KIOCSOUND       0x4B2F  /* start sound generation (0 for off) */
27: #define KDMKTONE        0x4B30  /* generate tone */
28:
29: #define KDGETLED        0x4B31  /* return current led state */
30: #define KDSETLED        0x4B32  /* set led state [lights, not flags] */
31: #define LED_SCR         0x01    /* scroll lock led */
32: #define LED_NUM         0x02    /* num lock led */
33: #define LED_CAP         0x04    /* caps lock led */
34:
35: #define KDGKBTYPE       0x4B33  /* get keyboard type */
36: #define KB_84           0x01
37: #define KB_101          0x02    /* this is what we always answer */
38: #define KB_OTHER        0x03
39:
40: #define KDADDIO         0x4B34  /* add i/o port as valid */
41: #define KDDELIO         0x4B35  /* del i/o port as valid */
42: #define KDENABIO        0x4B36  /* enable i/o to video board */
43: #define KDDISABIO       0x4B37  /* disable i/o to video board */
44:
45: #define KDSETMODE       0x4B3A  /* set text/graphics mode */
46: #define KD_TEXT         0x00
47: #define KD_GRAPHICS     0x01
48: #define KD_TEXT0        0x02    /* obsolete */
49: #define KD_TEXT1        0x03    /* obsolete */
50: #define KDGETMODE       0x4B3B  /* get current mode */
51:
52: #define KDMAPDISP       0x4B3C  /* map display into address space */
53: #define KDUNMAPDISP     0x4B3D  /* unmap display from address space */
54:
55: typedef char scrnmap_t;
56: #define E_TABSZ         256
57: #define GIO_SCRNMAP     0x4B40  /* get screen mapping from kernel */
58: #define PIO_SCRNMAP     0x4B41  /* put screen mapping table in kernel */
59: #define GIO_UNISCRNMAP  0x4B69  /* get full Unicode screen mapping */
60: #define PIO_UNISCRNMAP  0x4B6A  /* set full Unicode screen mapping */
61:
62: #define GIO_UNIMAP      0x4B66  /* get unicode-to-font mapping from kernel */
63: struct unipair {
64:         unsigned short int unicode;
65:         unsigned short int fontpos;
66: };
67: struct unimapdesc {
```

include/fiwix/kd.h


```
   68:         unsigned short int entry_ct;
   69:         struct unipair *entries;
   70: };
   71: #define PIO_UNIMAP      0x4B67  /* put unicode-to-font mapping in kernel */
   72: #define PIO_UNIMAPCLR   0x4B68  /* clear table, possibly advise hash algorithm *
/
   73: struct unimapinit {
   74:         unsigned short int advised_hashsize;  /* 0 if no opinion */
   75:         unsigned short int advised_hashstep;  /* 0 if no opinion */
   76:         unsigned short int advised_hashlevel; /* 0 if no opinion */
   77: };
   78:
   79: #define UNI_DIRECT_BASE 0xF000  /* start of Direct Font Region */
   80: #define UNI_DIRECT_MASK 0x01FF  /* Direct Font Region bitmask */
   81:
   82: #define K_RAW           0x00
   83: #define K_XLATE         0x01
   84: #define K_MEDIUMRAW     0x02
   85: #define K_UNICODE       0x03
   86: #define KDGKBMODE       0x4B44  /* gets current keyboard mode */
   87: #define KDSKBMODE       0x4B45  /* sets current keyboard mode */
   88:
   89: #define K_METABIT       0x03
   90: #define K_ESCPREFIX     0x04
   91: #define KDGKBMETA       0x4B62  /* gets meta key handling mode */
   92: #define KDSKBMETA       0x4B63  /* sets meta key handling mode */
   93:
   94: #define K_SCROLLLOCK    0x01
   95: #define K_NUMLOCK       0x02
   96: #define K_CAPSLOCK      0x04
   97: #define KDGKBLED        0x4B64  /* get led flags (not lights) */
   98: #define KDSKBLED        0x4B65  /* set led flags (not lights) */
   99:
  100: struct kbentry {
  101:         unsigned char kb_table;
  102:         unsigned char kb_index;
  103:         unsigned short int kb_value;
  104: };
  105: #define K_NORMTAB       0x00
  106: #define K_SHIFTTAB      0x01
  107: #define K_ALTTAB        0x02
  108: #define K_ALTSHIFTTAB   0x03
  109:
  110: #define KDGKBENT        0x4B46  /* gets one entry in translation table */
  111: #define KDSKBENT        0x4B47  /* sets one entry in translation table */
  112:
  113: struct kbsentry {
  114:         unsigned char kb_func;
  115:         unsigned char kb_string[512];
  116: };
  117: #define KDGKBSENT       0x4B48  /* gets one function key string entry */
  118: #define KDSKBSENT       0x4B49  /* sets one function key string entry */
  119:
  120: struct kbdiacr {
  121:         unsigned char diacr, base, result;
  122: };
  123: struct kbdiacrs {
  124:         unsigned int kb_cnt;    /* number of entries in following array */
  125:         struct kbdiacr kbdiacr[256];    /* MAX_DIACR from keyboard.h */
  126: };
  127: #define KDGKBDIACR      0x4B4A  /* read kernel accent table */
  128: #define KDSKBDIACR      0x4B4B  /* write kernel accent table */
  129:
  130: struct kbkeycode {
  131:         unsigned int scancode, keycode;
  132: };
  133: #define KDGETKEYCODE    0x4B4C  /* read kernel keycode table entry */
```

include/fiwix/kd.h


```
  134: #define KDSETKEYCODE    0x4B4D  /* write kernel keycode table entry */
  135:
  136: #define KDSIGACCEPT     0x4B4E  /* accept kbd generated signals */
  137:
  138: struct kbd_repeat {
  139:         int delay;      /* in msec; <= 0: don't change */
  140:         int rate;       /* in msec; <= 0: don't change */
  141: };
  142:
  143: #define KDKBDREP        0x4B52  /* set keyboard delay/repeat rate;
  144:                                  * actually used values are returned */
  145:
  146: #define KDFONTOP        0x4B72  /* font operations */
  147:
  148: struct console_font_op {
  149:         unsigned int op;        /* operation code KD_FONT_OP_* */
  150:         unsigned int flags;     /* KD_FONT_FLAG_* */
  151:         unsigned int width, height;     /* font size */
  152:         unsigned int charcount;
  153:         unsigned char *data;    /* font data with height fixed to 32 */
  154: };
  155:
  156: #define KD_FONT_OP_SET          0       /* Set font */
  157: #define KD_FONT_OP_GET          1       /* Get font */
  158: #define KD_FONT_OP_SET_DEFAULT  2       /* Set font to default, data points to n
ame / NULL */
  159: #define KD_FONT_OP_COPY         3       /* Copy from another console */
  160:
  161: #define KD_FONT_FLAG_DONT_RECALC        1       /* Don't recalculate hw charcell
 size [compat] */
  162:
  163: /* note: 0x4B00-0x4B4E all have had a value at some time;
  164:    don't reuse for the time being */
  165: /* note: 0x4B60-0x4B6D, 0x4B70-0x4B72 used above */
  166:
  167: #endif /* _LINUX_KD_H */
```

include/fiwix/kernel.h


```
    1: /*
    2:  * fiwix/include/fiwix/kernel.h
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #ifndef _FIWIX_KERNEL_H
    9: #define _FIWIX_KERNEL_H
   10:
   11: #include <fiwix/limits.h>
   12: #include <fiwix/i386elf.h>
   13:
   14: #define PANIC(format, args...)                                          \
   15: {                                                                       \
   16:         printk("\nPANIC: in %s()", __FUNCTION__);                       \
   17:         printk("\n");                                                   \
   18:         printk(format, ## args);                                        \
   19:         stop_kernel();                                                  \
   20: }
   21:
   22: #define CURRENT_TIME    (kstat.system_time)
   23: #define CURRENT_TICKS   (kstat.ticks)
   24: #define INIT_PROGRAM    "/sbin/init"
   25:
   26: extern char *init_argv[];
   27: extern char *init_envp[];
   28: extern char *init_args;
   29:
   30: extern Elf32_Shdr *symtab, *strtab;
   31: extern unsigned int _last_data_addr;
   32:
   33: extern int _memsize;
   34: extern int _extmemsize;
   35: extern int _rootdev;
   36: extern int _noramdisk;
   37: extern int _ramdisksize;
   38: extern char _rootfstype[10];
   39: extern char _rootdevname[DEVNAME_MAX + 1];
   40: extern char _initrd[DEVNAME_MAX + 1];
   41: extern int _syscondev;
   42:
   43: extern int _cputype;
   44: extern int _cpusignature;
   45: extern int _cpuflags;
   46: extern int _brandid;
   47: extern char _vendorid[12];
   48: extern char _brandstr[48];
   49: extern unsigned int _tlbinfo_eax;
   50: extern unsigned int _tlbinfo_ebx;
   51: extern unsigned int _tlbinfo_ecx;
   52: extern unsigned int _tlbinfo_edx;
   53: extern char _etext[], _edata[], _end[];
   54:
   55: extern char cmdline[NAME_MAX + 1];
   56:
   57: struct kernel_stat {
   58:         unsigned int cpu_user;          /* ticks in user-mode */
   59:         unsigned int cpu_nice;          /* ticks in user-mode (with priority) */
   60:         unsigned int cpu_system;        /* ticks in kernel-mode */
   61:         unsigned int irqs;              /* irq counter */
   62:         unsigned int sirqs;             /* spurious irq counter */
   63:         unsigned int ctxt;              /* context switches */
   64:         unsigned int ticks;             /* ticks (1/HZths of sec) since boot */
   65:         unsigned int system_time;       /* current system time (since the Epoch)
 */
   66:         unsigned int boot_time;         /* boot time (since the Epoch) */
```

include/fiwix/kernel.h


```
67:         int tz_minuteswest;             /* minutes west of GMT */
68:         int tz_dsttime;                 /* type of DST correction */
69:         unsigned int uptime;            /* seconds since boot */
70:         unsigned int processes;         /* number of forks since boot */
71:         unsigned int physical_pages;    /* physical memory in pages */
72:         unsigned int kernel_reserved;   /* kernel memory reserved in KB */
73:         unsigned int physical_reserved; /* physical memory reserved in KB */
74:         unsigned int total_mem_pages;   /* total memory in pages */
75:         unsigned int free_pages;        /* pages on free list (available) */
76:         unsigned int buffers;           /* memory used by buffers in KB */
77:         unsigned int cached;            /* memory used to cache file pages */
78:         unsigned int shared;            /* pages with count > 1 */
79:         unsigned long int random_seed;  /* next random seed */
80: };
81: extern struct kernel_stat kstat;
82:
83: unsigned int get_last_boot_addr(unsigned int);
84: void start_kernel(unsigned long, unsigned long, unsigned int);
85: void stop_kernel(void);
86: void init_init(void);
87: void cpu_idle(void);
88:
89: #endif /* _FIWIX_KERNEL_H */
```

include/fiwix/keyboard.h


```
 1: /*
 2:  * fiwix/include/fiwix/keyboard.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #include <fiwix/types.h>
 9:
10: #ifndef _FIWIX_KEYBOARD_H
11: #define _FIWIX_KEYBOARD_H
12:
13: #define KEYBOARD_IRQ    1
14:
15: #define NR_MODIFIERS    16      /* max. number of modifiers per keymap */
16: #define NR_SCODES       128     /* max. number of scancodes */
17: #define NR_DIACR        10
18:
19: #define SCRLBIT         0x01    /* scroll lock led */
20: #define NUMSBIT         0x02    /* num lock led */
21: #define CAPSBIT         0x04    /* caps lock led */
22:
23: #define C(ch)           ((ch) & 0x1F)
24: #define A(ch)           ((ch) | META_KEYS)
25: #define L(ch)           ((ch) | LETTER_KEYS)
26:
27: #define SLASH_NPAD      53
28:
29: #define MOD_BASE        0
30: #define MOD_SHIFT       1
31: #define MOD_ALTGR       2
32: #define MOD_CTRL        3
33: #define MOD_ALT         4
34: #define MOD_SHIFTL      5
35: #define MOD_SHIFTR      6
36: #define MOD_CTRLL       7
37: #define MOD_CTRLR       8
38:
39: #define FN_KEYS         0x100
40: #define SPEC_KEYS       0x200
41: #define PAD_KEYS        0x300
42: #define DEAD_KEYS       0x400
43: #define CONS_KEYS       0x500
44: #define SHIFT_KEYS      0x700
45: #define META_KEYS       0x800
46: #define LETTER_KEYS     0xB00
47:
48: #define CR              (0x01 + SPEC_KEYS)
49: #define SCRL2           (0x02 + SPEC_KEYS)      /* SH_REGS (show registers) */
50: #define SCRL3           (0x03 + SPEC_KEYS)      /* SH_MEM (show memory) */
51: #define SCRL4           (0x04 + SPEC_KEYS)      /* SH_STAT (show status) */
52: #define CAPS            (0x07 + SPEC_KEYS)
53: #define NUMS            (0x08 + SPEC_KEYS)
54: #define SCRL            (0x09 + SPEC_KEYS)
55:
56: #define INS             (0x00 + PAD_KEYS)
57: #define END             (0x01 + PAD_KEYS)
58: #define DOWN            (0x02 + PAD_KEYS)
59: #define PGDN            (0x03 + PAD_KEYS)
60: #define LEFT            (0x04 + PAD_KEYS)
61: #define MID             (0x05 + PAD_KEYS)
62: #define RIGHT           (0x06 + PAD_KEYS)
63: #define HOME            (0x07 + PAD_KEYS)
64: #define UP              (0x08 + PAD_KEYS)
65: #define PGUP            (0x09 + PAD_KEYS)
66: #define PLUS            (0x0A + PAD_KEYS)
67: #define MINUS           (0x0B + PAD_KEYS)
```

include/fiwix/keyboard.h


```
 68: #define ASTSK           (0x0C + PAD_KEYS)
 69: #define SLASH           (0x0D + PAD_KEYS)
 70: #define ENTER           (0x0E + PAD_KEYS)
 71: #define DEL             (0x10 + PAD_KEYS)
 72:
 73: #define GRAVE           (0x00 + DEAD_KEYS)
 74: #define ACUTE           (0x01 + DEAD_KEYS)
 75: #define CIRCM           (0x02 + DEAD_KEYS)
 76: #define DIERE           (0x04 + DEAD_KEYS)
 77:
 78: #define SHIFT           (0x00 + SHIFT_KEYS)
 79: #define ALTGR           (0x01 + SHIFT_KEYS)
 80: #define CTRL            (0x02 + SHIFT_KEYS)
 81: #define ALT             (0x03 + SHIFT_KEYS)
 82:
 83: #define F1              (0x00 + FN_KEYS)
 84: #define F2              (0x01 + FN_KEYS)
 85: #define F3              (0x02 + FN_KEYS)
 86: #define F4              (0x03 + FN_KEYS)
 87: #define F5              (0x04 + FN_KEYS)
 88: #define F6              (0x05 + FN_KEYS)
 89: #define F7              (0x06 + FN_KEYS)
 90: #define F8              (0x07 + FN_KEYS)
 91: #define F9              (0x08 + FN_KEYS)
 92: #define F10             (0x09 + FN_KEYS)
 93: #define F11             (0x0A + FN_KEYS)
 94: #define F12             (0x0B + FN_KEYS)
 95:
 96: #define SF1             (0x0A + FN_KEYS)
 97: #define SF2             (0x0B + FN_KEYS)
 98: #define SF3             (0x0C + FN_KEYS)
 99: #define SF4             (0x0D + FN_KEYS)
100: #define SF5             (0x0E + FN_KEYS)
101: #define SF6             (0x0F + FN_KEYS)
102: #define SF7             (0x10 + FN_KEYS)
103: #define SF8             (0x11 + FN_KEYS)
104: #define SF9             (0x12 + FN_KEYS)
105: #define SF10            (0x13 + FN_KEYS)
106: #define SF11            (0x0A + SHIFT)
107: #define SF12            (0x0B + SHIFT)
108:
109: #define AF1             (0x00 + CONS_KEYS)
110: #define AF2             (0x01 + CONS_KEYS)
111: #define AF3             (0x02 + CONS_KEYS)
112: #define AF4             (0x03 + CONS_KEYS)
113: #define AF5             (0x04 + CONS_KEYS)
114: #define AF6             (0x05 + CONS_KEYS)
115: #define AF7             (0x06 + CONS_KEYS)
116: #define AF8             (0x07 + CONS_KEYS)
117: #define AF9             (0x08 + CONS_KEYS)
118: #define AF10            (0x09 + CONS_KEYS)
119: #define AF11            (0x0A + CONS_KEYS)
120: #define AF12            (0x0B + CONS_KEYS)
121:
122: struct diacritic {
123:         unsigned char letter;
124:         unsigned char code;
125: };
126:
127: extern __key_t keymap[NR_MODIFIERS * NR_SCODES];
128:
129: void set_leds(unsigned char);
130: void irq_keyboard(void);
131: void keyboard_bh(void);
132: void keyboard_init(void);
133:
134: #endif /* _FIWIX_KEYBOARD_H */
```

include/fiwix/kparms.h


```
/*
 * fiwix/include/fiwix/kparms.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_KPARMS_H
#define _FIWIX_KPARMS_H

#define CMDL_ARG_LEN    25      /* max length of cmdline argument */
#define CMDL_NUM_VALUES 30      /* max values of cmdline parameter */

struct kparms {
        char *name;
        char *value[CMDL_NUM_VALUES];
        unsigned int sysval[CMDL_NUM_VALUES];
};

static struct kparms parm_table[] = {
        { "root=",
           { "/dev/ram0", "/dev/fd0", "/dev/fd1",
             "/dev/hda", "/dev/hda1", "/dev/hda2", "/dev/hda3", "/dev/hda4",
             "/dev/hdb", "/dev/hdb1", "/dev/hdb2", "/dev/hdb3", "/dev/hdb4",
             "/dev/hdc", "/dev/hdc1", "/dev/hdc2", "/dev/hdc3", "/dev/hdc4",
             "/dev/hdd", "/dev/hdd1", "/dev/hdd2", "/dev/hdd3", "/dev/hdd4",
           },
           { 0x100, 0x200, 0x201,
             0x300, 0x301, 0x302, 0x303, 0x304,
             0x340, 0x341, 0x342, 0x343, 0x344,
             0x1600, 0x1601, 0x1602, 0x1603, 0x1604,
             0x1640, 0x1641, 0x1642, 0x1643, 0x1644,
           }
        },
        { "noramdisk",
           { NULL },
           { NULL },
        },
        { "ramdisksize=",
           { NULL },
           { NULL },
        },
        { "initrd=",
           { NULL },
           { NULL },
        },
        { "rootfstype=",
           { "minix", "ext2", "iso9660" },
           { 0, 0 }
        },
        { "console=",
           { "/dev/tty1", "/dev/tty2", "/dev/tty3", "/dev/tty4", "/dev/tty5",
             "/dev/tty6", "/dev/tty7", "/dev/tty8", "/dev/tty9", "/dev/tty10",
             "/dev/tty11", "/dev/tty12"
           },
           { 0x401, 0x402, 0x403, 0x404, 0x405,
             0x406, 0x407, 0x408, 0x409, 0x410,
             0x411, 0x412
           }
        },

        { NULL }
};

#endif /* _FIWIX_KPARMS_H */
```

include/fiwix/limits.h


```
 1: /*
 2:  * fiwix/include/fiwix/limits.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_LIMITS_H
 9: #define _FIWIX_LIMITS_H
10:
11: #define DEVNAME_MAX     25      /* device name length in mount table */
12: #define ARG_MAX         32      /* length (in pages) of argv+env in 'execve' */
13: #define CHILD_MAX       64      /* simultaneous processes per real user ID */
14: #define LINK_MAX        255     /* maximum number of links to a file */
15: #define MAX_CANON       255     /* bytes in a terminal canonical input queue */
16: #define MAX_INPUT       255     /* bytes for which space will be available in a
17:                                    terminal input queue */
18: #define NGROUPS_MAX     32      /* simultaneous supplementary group IDs */
19: #define OPEN_MAX        256     /* files one process can have opened at once */
20: #define FD_SETSIZE      OPEN_MAX /* descriptors that a process may examine with
21:                                     'pselect' or 'select' */
22: #define NAME_MAX        255     /* bytes in a filename */
23: #define PATH_MAX        1024    /* bytes in a pathname */
24: #define PIPE_BUF        4096    /* bytes than can be written atomically to a
25:                                    pipe */
26:
27: #endif /* _FIWIX_LIMITS_H */
```

include/fiwix/locks.h


```
 1: /*
 2:  * fiwix/include/fiwix/locks.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_LOCKS_H
 9: #define _FIWIX_LOCKS_H
10:
11: #include <fiwix/config.h>
12: #include <fiwix/fs.h>
13: #include <fiwix/fcntl.h>
14:
15: struct flock_file {
16:         struct inode *inode;    /* file */
17:         unsigned char type;     /* type of lock */
18:         struct proc *proc;      /* owner */
19: };
20:
21: struct flock_file flock_file_table[NR_FLOCKS];
22:
23: int posix_lock(int, int, struct flock *);
24:
25: void flock_release_inode(struct inode *);
26: int flock_inode(struct inode *, int);
27: void flock_init(void);
28:
29: #endif /* _FIWIX_LOCKS_H */
```

include/fiwix/lp.h


```
 1: /*
 2:  * fiwix/include/fiwix/lp.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_LP_H
 9: #define _FIWIX_LP_H
10:
11: #include <fiwix/fs.h>
12:
13: #define LP_MAJOR        6       /* major number for /dev/lp[n] */
14: #define LP_MINORS       1
15:
16: /*#define LP0_ADDR      0x3BC */
17: #define LP0_ADDR        0x378
18: /*#define LP2_ADDR      0x278 */
19:
20: #define LP_STAT_ERR     0x08    /* printer error */
21: #define LP_STAT_SEL     0x10    /* select in */
22: #define LP_STAT_PE      0x20    /* paper empty or no paper */
23: #define LP_STAT_ACK     0x40    /* ack */
24: #define LP_STAT_BUS     0x80    /* printer busy */
25:
26: #define LP_CTRL_STR     0x01    /* strobe */
27: #define LP_CTRL_AUT     0x02    /* auto line feed */
28: #define LP_CTRL_INI     0x04    /* initialize printer (reset) */
29: #define LP_CTRL_SEL     0x08    /* select printer */
30: #define LP_CTRL_IRQ     0x10    /* enable IRQ */
31: #define LP_CTRL_BID     0x20    /* bidireccional (on PS/2 ports) */
32:
33: #define LP_RDY_RETR     100     /* retries before timeout */
34:
35: struct lp {
36:         int data;       /* data port address */
37:         int stat;       /* status port address */
38:         int ctrl;       /* control port address */
39:         char flags;     /* flags */
40: };
41:
42: int lp_open(struct inode *, struct fd *);
43: int lp_close(struct inode *, struct fd *);
44: int lp_write(struct inode *, struct fd *, const char *, __size_t);
45:
46: void lp_init(void);
47:
48: #endif /* _FIWIX_LP_H */
```

include/fiwix/memdev.h


```
 1: /*
 2:  * fiwix/include/fiwix/memdev.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_MEMDEV_H
 9: #define _FIWIX_MEMDEV_H
10:
11: #include <fiwix/fs.h>
12:
13: #define MEMDEV_MAJOR    1       /* major number */
14: #define MEMDEV_MINORS   5       /* number of supported minors */
15:
16: #define MEMDEV_MEM      1       /* minor for /dev/mem */
17: #define MEMDEV_KMEM     2       /* minor for /dev/kmem */
18: #define MEMDEV_NULL     3       /* minor for /dev/null */
19: #define MEMDEV_ZERO     5       /* minor for /dev/zero */
20: #define MEMDEV_RANDOM   8       /* minor for /dev/random */
21: #define MEMDEV_URANDOM  9       /* minor for /dev/urandom */
22:
23: int mem_open(struct inode *, struct fd *);
24: int mem_close(struct inode *, struct fd *);
25: int mem_read(struct inode *, struct fd *, char *, __size_t);
26: int mem_write(struct inode *, struct fd *, const char *, __size_t);
27: int mem_lseek(struct inode *, __off_t);
28:
29: int kmem_open(struct inode *, struct fd *);
30: int kmem_close(struct inode *, struct fd *);
31: int kmem_read(struct inode *, struct fd *, char *, __size_t);
32: int kmem_write(struct inode *, struct fd *, const char *, __size_t);
33: int kmem_lseek(struct inode *, __off_t);
34:
35: int null_open(struct inode *, struct fd *);
36: int null_close(struct inode *, struct fd *);
37: int null_read(struct inode *, struct fd *, char *, __size_t);
38: int null_write(struct inode *, struct fd *, const char *, __size_t);
39: int null_lseek(struct inode *, __off_t);
40:
41: int zero_open(struct inode *, struct fd *);
42: int zero_close(struct inode *, struct fd *);
43: int zero_read(struct inode *, struct fd *, char *, __size_t);
44: int zero_write(struct inode *, struct fd *, const char *, __size_t);
45: int zero_lseek(struct inode *, __off_t);
46:
47: int urandom_open(struct inode *, struct fd *);
48: int urandom_close(struct inode *, struct fd *);
49: int urandom_read(struct inode *, struct fd *, char *, __size_t);
50: int urandom_write(struct inode *, struct fd *, const char *, __size_t);
51: int urandom_lseek(struct inode *, __off_t);
52:
53: int memdev_open(struct inode *, struct fd *);
54: int mem_mmap(struct inode *, struct vma *);
55: void memdev_init(void);
56:
57: #endif /* _FIWIX_MEMDEV_H */
```

include/fiwix/mman.h


```
    1: /*
    2:  * fiwix/include/fiwix/mman.h
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #ifndef _FIWIX_MMAN_H
    9: #define _FIWIX_MMAN_H
   10:
   11: #include <fiwix/fs.h>
   12:
   13: #define PROT_READ       0x1             /* page can be read */
   14: #define PROT_WRITE      0x2             /* page can be written */
   15: #define PROT_EXEC       0x4             /* page can be executed */
   16: #define PROT_NONE       0x0             /* page cannot be accessed */
   17:
   18: #define MAP_SHARED      0x01            /* share changes */
   19: #define MAP_PRIVATE     0x02            /* changes are private */
   20: #define MAP_TYPE        0x0f            /* mask for type of mapping */
   21: #define MAP_FIXED       0x10            /* interpret address exactly */
   22: #define MAP_ANONYMOUS   0x20            /* don't use the file descriptor */
   23:
   24: #define MAP_GROWSDOWN   0x0100          /* stack-like segment */
   25: #define MAP_DENYWRITE   0x0800          /* -ETXTBSY */
   26: #define MAP_EXECUTABLE  0x1000          /* mark it as a executable */
   27: #define MAP_LOCKED      0x2000          /* pages are locked */
   28:
   29: #define ZERO_PAGE       0x80000000      /* this page must be zero-filled */
   30:
   31: #define MS_ASYNC        1               /* sync memory asynchronously */
   32: #define MS_INVALIDATE   2               /* invalidate the caches */
   33: #define MS_SYNC         4               /* synchronous memory sync */
   34:
   35: #define MCL_CURRENT     1               /* lock all current mappings */
   36: #define MCL_FUTURE      2               /* lock all future mappings */
   37:
   38: /* compatibility flags */
   39: #define MAP_ANON        MAP_ANONYMOUS
   40: #define MAP_FILE        0
   41:
   42: struct mmap {
   43:         unsigned int start;
   44:         unsigned int length;
   45:         unsigned int prot;
   46:         unsigned int flags;
   47:         int fd;
   48:         unsigned int offset;
   49: };
   50:
   51: void show_vma_regions(struct proc *);
   52: void release_binary(void);
   53: struct vma * find_vma_region(unsigned int);
   54: int expand_heap(unsigned int);
   55: int do_mmap(struct inode *, unsigned int, unsigned int, unsigned int, unsigned i
nt, unsigned int, char, char);
   56: int do_munmap(unsigned int, __size_t);
   57: int do_mprotect(struct vma *, unsigned int, __size_t, int);
   58:
   59: #endif /* _FIWIX_MMAN_H */
```

include/fiwix/mm.h


```
/*
 * fiwix/include/fiwix/mm.h
 *
 * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 * Distributed under the terms of the Fiwix License.
 */

#ifndef _FIWIX_MEMORY_H
#define _FIWIX_MEMORY_H

#include <fiwix/types.h>
#include <fiwix/const.h>
#include <fiwix/process.h>

#define P2V(addr)               (addr + KERNEL_BASE_ADDR)
#define V2P(addr)               (addr - KERNEL_BASE_ADDR)

#define PAGE_SIZE               4096
#define PAGE_SHIFT              0x0C
#define PAGE_MASK               ~(PAGE_SIZE - 1)        /* 0xFFFFF000 */
#define PAGE_ALIGN(addr)        (((addr) + (PAGE_SIZE - 1)) & PAGE_MASK)
#define PT_ENTRIES              (PAGE_SIZE / sizeof(unsigned int))
#define PD_ENTRIES              (PAGE_SIZE / sizeof(unsigned int))

#define PAGE_PRESENT            0x001   /* Present */
#define PAGE_RW                 0x002   /* Read/Write */
#define PAGE_USER               0x004   /* User */

#define PAGE_RESERVED           0x100   /* kernel, BIOS address, ... */
#define PAGE_COW                0x200   /* marked for Copy-On-Write */

#define PFAULT_V                0x01    /* protection violation */
#define PFAULT_W                0x02    /* during write */
#define PFAULT_U                0x04    /* in user mode */

#define GET_PGDIR(address)      ((unsigned int)((address) >> 22) & 0x3FF)
#define GET_PGTBL(address)      ((unsigned int)((address) >> 12) & 0x3FF)

#define P_TEXT  1       /* text section */
#define P_DATA  2       /* data section */
#define P_BSS   3       /* BSS section */
#define P_HEAP  4       /* heap section (sys_brk()) */
#define P_STACK 5       /* stack section */
#define P_MMAP  6       /* mmap() section */

struct page {
        unsigned int page;      /* page number */
        unsigned int count;     /* usage counter */
        unsigned int flags;
        unsigned char locked;   /* 1 = locked */
        struct inode *inode;
        __off_t offset;         /* file offset */
        char *data;             /* page contents */
        struct page *prev_hash;
        struct page *next_hash;
        struct page *prev_free;
        struct page *next_free;
};

extern struct page *page_table;
extern struct page **page_hash_table;

/* values to be determined during system startup */
extern unsigned int page_table_size;            /* size in bytes */
extern unsigned int page_hash_table_size;       /* size in bytes */

extern unsigned int *kpage_dir;
```

include/fiwix/mm.h


```
 68: extern unsigned int *kpage_table;
 69:
 70: /* alloc.c */
 71: unsigned int kmalloc(void);
 72: void kfree(unsigned int);
 73:
 74: /* page.c */
 75: void page_lock(struct page *);
 76: void page_unlock(struct page *);
 77: struct page * get_free_page(void);
 78: struct page * search_page_hash(struct inode *, __off_t);
 79: void release_page(unsigned int);
 80: int valid_page(unsigned int);
 81: void update_page_cache(struct inode *, __off_t, const char *, int);
 82: int write_page(struct page *, struct inode *, __off_t, unsigned int);
 83: int bread_page(struct page *, struct inode *, __off_t, char, char);
 84: int file_read(struct inode *, struct fd *, char *, __size_t);
 85: void page_init(unsigned int);
 86:
 87: /* memory.c */
 88: void bss_init(void);
 89: unsigned int setup_minmem(void);
 90: unsigned int get_mapped_addr(struct proc *, unsigned int);
 91: int clone_pages(struct proc *);
 92: int free_page_tables(struct proc *);
 93: unsigned int map_page(struct proc *, unsigned int, unsigned int, unsigned int);
 94: int unmap_page(unsigned int);
 95: void mem_init(void);
 96: void mem_stats(void);
 97:
 98: /* swapper.c */
 99: int kswapd(void);
100:
101: #endif /* _FIWIX_MEMORY_H */
```

include/fiwix/multiboot.h


```
/* multiboot.h - the header for Multiboot */
/* Copyright (C) 1999, 2001  Free Software Foundation, Inc.

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.  */

/* Macros.  */

#ifndef _FIWIX_MULTIBOOT_H
#define _FIWIX_MULTIBOOT_H

/* The magic number for the Multiboot header.  */
#define MULTIBOOT_HEADER_MAGIC          0x1BADB002

/* The flags for the Multiboot header.  */
#ifdef __ELF__
# define MULTIBOOT_HEADER_FLAGS         0x00000003
#else
# define MULTIBOOT_HEADER_FLAGS         0x00010003
#endif

/* The magic number passed by a Multiboot-compliant boot loader.  */
#define MULTIBOOT_BOOTLOADER_MAGIC      0x2BADB002

/* The size of our stack (16KB).  */
#define STACK_SIZE                      0x4000

/* C symbol format. HAVE_ASM_USCORE is defined by configure.  */
#ifdef HAVE_ASM_USCORE
# define EXT_C(sym)                     _ ## sym
#else
# define EXT_C(sym)                     sym
#endif

#ifndef ASM
/* Do not include here in boot.S.  */

/* Types.  */

/* The Multiboot header.  */
typedef struct multiboot_header
{
  unsigned long magic;
  unsigned long flags;
  unsigned long checksum;
  unsigned long header_addr;
  unsigned long load_addr;
  unsigned long load_end_addr;
  unsigned long bss_end_addr;
  unsigned long entry_addr;
} multiboot_header_t;

/* The symbol table for a.out.  */
typedef struct aout_symbol_table
{
  unsigned long tabsize;
```

include/fiwix/multiboot.h


```
  68:   unsigned long strsize;
  69:   unsigned long addr;
  70:   unsigned long reserved;
  71: } aout_symbol_table_t;
  72:
  73: /* The section header table for ELF.  */
  74: typedef struct elf_section_header_table
  75: {
  76:   unsigned long num;
  77:   unsigned long size;
  78:   unsigned long addr;
  79:   unsigned long shndx;
  80: } elf_section_header_table_t;
  81:
  82: /* The Multiboot information.  */
  83: typedef struct multiboot_info
  84: {
  85:   unsigned long flags;
  86:   unsigned long mem_lower;
  87:   unsigned long mem_upper;
  88:   unsigned long boot_device;
  89:   unsigned long cmdline;
  90:   unsigned long mods_count;
  91:   unsigned long mods_addr;
  92:   union
  93:   {
  94:     aout_symbol_table_t aout_sym;
  95:     elf_section_header_table_t elf_sec;
  96:   } u;
  97:   unsigned long mmap_length;
  98:   unsigned long mmap_addr;
  99: } multiboot_info_t;
 100:
 101: /* The module structure.  */
 102: typedef struct module
 103: {
 104:   unsigned long mod_start;
 105:   unsigned long mod_end;
 106:   unsigned long string;
 107:   unsigned long reserved;
 108: } module_t;
 109:
 110: /* The memory map. Be careful that the offset 0 is base_addr_low
 111:    but no size.  */
 112: typedef struct memory_map
 113: {
 114:   unsigned long size;
 115:   unsigned long base_addr_low;
 116:   unsigned long base_addr_high;
 117:   unsigned long length_low;
 118:   unsigned long length_high;
 119:   unsigned long type;
 120: } memory_map_t;
 121:
 122: #endif /* ! ASM */
 123:
 124: #endif /* _FIWIX_MULTIBOOT_H */
```

include/fiwix/part.h


```
 1: /*
 2:  * fiwix/include/fiwix/part.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_PART_H
 9: #define _FIWIX_PART_H
10:
11: #define PARTITION_BLOCK         0
12: #define NR_PARTITIONS           4       /* partitions in the MBR */
13: #define MBR_CODE_SIZE           446
14: #define ACTIVE_PART             0x80
15:
16: struct hd_geometry {
17:         unsigned char heads;
18:         unsigned char sectors;
19:         unsigned short int cylinders;
20:         unsigned long int start;
21: };
22:
23: struct partition {
24:         unsigned char status;
25:         unsigned char head;
26:         unsigned char sector;
27:         unsigned char cyl;
28:         unsigned char type;
29:         unsigned char endhead;
30:         unsigned char endsector;
31:         unsigned char endcyl;
32:         unsigned int startsect;
33:         unsigned int nr_sects;
34: };
35:
36: int read_msdos_partition(__dev_t, struct partition *);
37:
38: #endif /* _FIWIX_PART_H */
```

include/fiwix/pic.h


```
 1: /*
 2:  * fiwix/include/fiwix/pic.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_PIC_H
 9: #define _FIWIX_PIC_H
10:
11: #include <fiwix/sigcontext.h>
12:
13: #define NR_IRQS         16      /* hardware interrupts */
14: #define PIC_MASTER      0x20    /* I/O base address for master PIC */
15: #define PIC_SLAVE       0xA0    /* I/O base address for slave PIC */
16:
17: #define DATA            0x01    /* offset to data port */
18: #define EOI             0x20    /* End-Of-Interrupt command code */
19:
20: /* Inicialization Command Words */
21: #define ICW1_RESET      0x11    /* ICW1_INIT + ICW1_ICW4 */
22: #define CASCADE_IRQ     0x02
23: #define ICW4_8086EOI    0x01
24:
25: #define PIC_READ_IRR    0x0A    /* OCW3 irq ready */
26: #define PIC_READ_ISR    0x0B    /* OCW3 irq service */
27:
28: /* Operational Command Words */
29: #define OCW1            0xFF    /* mask (disable) all IRQs */
30:
31: struct interrupts {
32:         unsigned int ticks;
33:         char *name;
34:         char registered;
35:         void (*handler)(void *);
36: };
37: struct interrupts irq_table[NR_IRQS];
38:
39: struct bh {
40:         void (*fn)(void);
41:         struct bh *next;
42: };
43:
44: void add_bh(void (*fn)(void));
45: void del_bh(void);
46: void enable_irq(int);
47: void disable_irq(int);
48: int register_irq(int, char *, void *);
49: int unregister_irq(int);
50: void irq_handler(int, struct sigcontext);
51: void do_bh(void);
52: void pic_init(void);
53:
54: #endif /* _FIWIX_PIC_H */
```

include/fiwix/pit.h


```
 1: /*
 2:  * fiwix/include/fiwix/pit.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_PIT_H
 9: #define _FIWIX_PIT_H
10:
11: /* Intel 8253/82c54 Programmable Interval Timer */
12:
13: #define OSCIL           1193182 /* oscillator frequency */
14:
15: #define MODEREG         0x43    /* mode/command register (w) */
16: #define CHANNEL0        0x40    /* channel 0 data port (rw) */
17: #define CHANNEL1        0x41    /* channel 1 data port (rw) */
18: #define CHANNEL2        0x42    /* channel 2 data port (rw) */
19:
20: #define BINARY_CTR      0x00    /* 16bit binary mode counter */
21: #define TERM_COUNT      0x00    /* mode 0 (Terminal Count) */
22: #define RATE_GEN        0x04    /* mode 2 (Rate Generator) */
23: #define SQUARE_WAVE     0x06    /* mode 3 (Square Wave) */
24: #define LSB_MSB         0x30    /* LSB then MSB */
25: #define SEL_CHAN0       0x00    /* select channel 0 */
26: #define SEL_CHAN2       0x80    /* select channel 2 */
27:
28: /*
29:  * PS/2 System Control Port B
30:  * --------------------------------------
31:  * bit 7 -> IRQ=1, 0=reset
32:  * bit 6 -> reserved
33:  * bit 5 -> reserved
34:  * bit 4 -> reserved
35:  * bit 3 -> channel check enable
36:  * bit 2 -> parity check enable
37:  * bit 1 -> speaker data enable
38:  * bit 0 -> timer 2 gate to speaker enable
39:  */
40: #define PS2_SYSCTRL_B   0x61    /* PS/2 system control port B (write) */
41:
42: #define ENABLE_TMR2G    0x01    /* timer 2 gate to speaker enable */
43: #define ENABLE_SDATA    0x02    /* speaker data enable */
44:
45: #define BEEP_FREQ       900     /* 900Hz */
46:
47: void pit_beep_on(void);
48: void pit_beep_off(unsigned int);
49: void pit_init(unsigned short int);
50:
51: #endif /* _FIWIX_PIT_H */
```

include/fiwix/process.h


```
    1: /*
    2:  * fiwix/include/fiwix/process.h
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #ifndef _FIWIX_PROCESS_H
    9: #define _FIWIX_PROCESS_H
   10:
   11: struct vma {
   12:         unsigned int start;
   13:         unsigned int end;
   14:         char prot;              /* PROT_READ, PROT_WRITE, ... */
   15:         unsigned int flags;     /* MAP_SHARED, MAP_PRIVATE, ... */
   16:         unsigned int offset;
   17:         char s_type;            /* section type (P_TEXT, P_DATA, ...) */
   18:         struct inode *inode;    /* file inode */
   19:         char o_mode;            /* open mode (O_RDONLY, O_RDWR, ...) */
   20: };
   21:
   22: #include <fiwix/config.h>
   23: #include <fiwix/types.h>
   24: #include <fiwix/signal.h>
   25: #include <fiwix/limits.h>
   26: #include <fiwix/sigcontext.h>
   27: #include <fiwix/time.h>
   28: #include <fiwix/resource.h>
   29: #include <fiwix/tty.h>
   30:
   31: #define IDLE            0               /* PID of idle */
   32: #define INIT            1               /* PID of /sbin/init */
   33: #define SAFE_SLOTS      2               /* process slots reserved for root */
   34: #define SLOT(p)         ((p) - (&proc_table[0]))
   35:
   36: /* bits in flags */
   37: #define PF_KPROC        0x00000001      /* kernel internal process */
   38: #define PF_PEXEC        0x00000002      /* has performed a sys_execve() */
   39: #define PF_USEREAL      0x00000004      /* use real UID in permission checks */
   40:
   41: #define MMAP_START      0x40000000      /* mmap()s start at 1GB */
   42: #define IS_SUPERUSER    (current->euid == 0)
   43:
   44: #define IO_BITMAP_SIZE  32              /* 32 * 32bit = 1024 = 0x3FF */
   45:                                         /* 2048 * 32bit = 65536 = 0xFFFF */
   46:
   47: #define PG_LEADER(p)    ((p)->pid == (p)->pgid)
   48: #define SESS_LEADER(p)  ((p)->pid == (p)->pgid && (p)->pid == (p)->sid)
   49:
   50: #define FOR_EACH_PROCESS(p)     for(p = &proc_table[INIT]; p; p = p->next)
   51:
   52: /* value to be determined during system startup */
   53: extern unsigned int proc_table_size;    /* size in bytes */
   54:
   55: extern char any_key_to_reboot;
   56: extern int nr_processes;
   57: extern __pid_t lastpid;
   58:
   59: struct binargs {
   60:         unsigned int page[ARG_MAX];
   61:         int argc;
   62:         int argv_len;
   63:         int envc;
   64:         int envp_len;
   65:         int offset;
   66: };
   67:
```

include/fiwix/process.h


```
 68: /* Intel 386 Task Switch State */
 69: struct i386tss {
 70:         unsigned int prev_tss;
 71:         unsigned int esp0;
 72:         unsigned int ss0;
 73:         unsigned int esp1;
 74:         unsigned int ss1;
 75:         unsigned int esp2;
 76:         unsigned int ss2;
 77:         unsigned int cr3;
 78:         unsigned int eip;
 79:         unsigned int eflags;
 80:         unsigned int eax;
 81:         unsigned int ecx;
 82:         unsigned int edx;
 83:         unsigned int ebx;
 84:         unsigned int esp;
 85:         unsigned int ebp;
 86:         unsigned int esi;
 87:         unsigned int edi;
 88:         unsigned int es;
 89:         unsigned int cs;
 90:         unsigned int ss;
 91:         unsigned int ds;
 92:         unsigned int fs;
 93:         unsigned int gs;
 94:         unsigned int ldt;
 95:         unsigned short int debug_trap;
 96:         unsigned short int io_bitmap_addr;
 97: };
 98:
 99: struct proc {
100:         struct i386tss tss;
101:         unsigned int io_bitmap[IO_BITMAP_SIZE + 1];
102:         __pid_t pid;                    /* process ID */
103:         __pid_t ppid;                   /* parent process ID */
104:         __pid_t pgid;                   /* process group ID */
105:         __pid_t sid;                    /* session ID */
106:         int flags;
107:         int groups[NGROUPS_MAX];
108:         int children;                   /* number of children */
109:         struct tty *ctty;               /* controlling terminal */
110:         int state;                      /* process state */
111:         int priority;
112:         int cpu_count;                  /* time of process running */
113:         __time_t start_time;
114:         int exit_code;
115:         void *sleep_address;
116:         unsigned short int uid;         /* real user ID */
117:         unsigned short int gid;         /* real group ID */
118:         unsigned short int euid;        /* effective user ID */
119:         unsigned short int egid;        /* effective group ID */
120:         unsigned short int suid;        /* saved user ID */
121:         unsigned short int sgid;        /* saved group ID */
122:         unsigned short int fd[OPEN_MAX];
123:         unsigned char fd_flags[OPEN_MAX];
124:         struct inode *root;
125:         struct inode *pwd;              /* process working directory */
126:         unsigned int entry_address;
127:         char argv0[NAME_MAX + 1];
128:         char **argv;
129:         char **envp;
130:         char pidstr[5];                 /* pid number converted to string */
131:         struct vma vma[VMA_REGIONS];    /* virtual memory-map addresses */
132:         unsigned int brk_lower;         /* lower limit of the heap section */
133:         unsigned int brk;               /* current limit of the heap */
134:         __sigset_t sigpending;
```

include/fiwix/process.h


```
135:         __sigset_t sigblocked;
136:         __sigset_t sigexecuting;
137:         struct sigaction sigaction[NSIG];
138:         struct sigcontext sc[NSIG];     /* each signal has its own context */
139:         unsigned int sp;                /* current process' stack frame */
140:         struct rusage usage;            /* process resource usage */
141:         struct rusage cusage;           /* children resource usage */
142:         unsigned long int it_real_interval, it_real_value;
143:         unsigned long int it_virt_interval, it_virt_value;
144:         unsigned long int it_prof_interval, it_prof_value;
145:         unsigned long int timeout;
146:         struct rlimit rlim[RLIM_NLIMITS];
147:         unsigned long int rss;
148:         __mode_t umask;
149:         unsigned char loopcnt;          /* nested symlinks counter */
150:         struct proc *prev;
151:         struct proc *next;
152:         struct proc *sleep_prev;
153:         struct proc *sleep_next;
154: };
155:
156: extern struct proc *current;
157: extern struct proc *proc_table;
158:
159: int send_sig(struct proc *, __sigset_t);
160: int kill_pid(__pid_t, __sigset_t);
161: int kill_pgrp(__pid_t, __sigset_t);
162: void add_crusage(struct proc *, struct rusage *);
163: void get_rusage(struct proc *, struct rusage *);
164: void add_rusage(struct proc *);
165: struct proc * get_next_zombie(struct proc *);
166: __pid_t remove_zombie(struct proc *);
167: int is_orphaned_pgrp(__pid_t);
168: struct proc * get_proc_free(void);
169: void release_proc(struct proc *);
170: int get_unused_pid(void);
171: struct proc * get_proc_by_pid(__pid_t);
172:
173: int get_new_user_fd(int);
174: void release_user_fd(int);
175:
176: struct proc * kernel_process(int (*fn)(void));
177: void proc_slot_init(struct proc *);
178: void proc_init(void);
179:
180: int elf_load(struct inode *, struct binargs *, struct sigcontext *, char *);
181: int script_load(char *, char *, char *);
182:
183: #endif /* _FIWIX_PROCESS_H */
```

include/fiwix/ramdisk.h


```
 1: /*
 2:  * fiwix/include/fiwix/ramdisk.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_RAMDISK_H
 9: #define _FIWIX_RAMDISK_H
10:
11: #include <fiwix/fs.h>
12:
13: #define RAMDISK_MAJOR   1       /* ramdisk device major number */
14: #define RAMDISK_MINORS  1       /* number of minors */
15: #define RAMDISK_SIZE    4096    /* default ramdisk size in KBs */
16: #define RAMDISK_MAXSIZE 131072  /* maximum ramdisk size in KBs */
17:
18: struct ramdisk {
19:         char *addr;             /* ramdisk memory address */
20: };
21:
22: extern struct ramdisk ramdisk_table[RAMDISK_MINORS];
23:
24: int ramdisk_open(struct inode *, struct fd *);
25: int ramdisk_close(struct inode *, struct fd *);
26: int ramdisk_read(__dev_t, __blk_t, char *, int);
27: int ramdisk_write(__dev_t, __blk_t, char *, int);
28: int ramdisk_ioctl(struct inode *, int, unsigned long int);
29: int ramdisk_lseek(struct inode *, __off_t);
30:
31: void ramdisk_init(void);
32:
33: #endif /* _FIWIX_RAMDISK_H */
```

include/fiwix/reboot.h


```
 1: /*
 2:  * fiwix/include/fiwix/reboot.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_REBOOT_H
 9: #define _FIWIX_REBOOT_H
10:
11: #define BMAGIC_HARD     0x89ABCDEF
12: #define BMAGIC_SOFT     0
13: #define BMAGIC_REBOOT   0x01234567
14: #define BMAGIC_HALT     0xCDEF0123
15: #define BMAGIC_POWEROFF 0x4321FEDC
16:
17: #define BMAGIC_1        0xFEE1DEAD
18: #define BMAGIC_2        672274793
19:
20: extern char ctrl_alt_del;
21: void reboot(void);
22:
23: #endif /* _FIWIX_REBOOT_H */
```

include/fiwix/resource.h


```
    1: /*
    2:  * fiwix/include/fiwix/resource.h
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #ifndef _FIWIX_RESOURCE_H
    9: #define _FIWIX_RESOURCE_H
   10:
   11: #include <fiwix/time.h>
   12:
   13: #define RLIMIT_INFINITY 0x7FFFFFFF      /* value to indicate "no limit" */
   14: #define RLIM_INFINITY   RLIMIT_INFINITY /* traditional name */
   15:
   16: #define RUSAGE_SELF     0               /* the calling process */
   17: #define RUSAGE_CHILDREN (-1)            /* all of its termin. child processes */
   18:
   19: #define RLIMIT_CPU      0               /* per-process CPU limit (secs) */
   20: #define RLIMIT_FSIZE    1               /* largest file that can be created
   21:                                            (bytes */
   22: #define RLIMIT_DATA     2               /* maximum size of data segment
   23:                                            (bytes) */
   24: #define RLIMIT_STACK    3               /* maximum size of stack segment
   25:                                            (bytes) */
   26: #define RLIMIT_CORE     4               /* largest core file that can be created
   27:                                            (bytes) */
   28: #define RLIMIT_RSS      5               /* largest resident set size (bytes) */
   29: #define RLIMIT_NPROC    6               /* number of processes */
   30: #define RLIMIT_NOFILE   7               /* number of open files */
   31: #define RLIMIT_MEMLOCK  8               /* locked-in-memory address space */
   32: #define RLIMIT_AS       9               /* address space limit */
   33:
   34: #define RLIM_NLIMITS    10
   35:
   36: struct rusage {
   37:         struct timeval ru_utime;        /* total amount of user time used */
   38:         struct timeval ru_stime;        /* total amount of system time used */
   39:         long ru_maxrss;                 /* maximum resident set size (KB) */
   40:         long ru_ixrss;                  /* amount of sharing of text segment
   41:                                            memory with other processes
   42:                                            (KB-secs) */
   43:         long ru_idrss;                  /* amount of data segment memory used
   44:                                            (KB-secs) */
   45:         long ru_isrss;                  /* amount of stack memory used
   46:                                            (KB-secs) */
   47:         long ru_minflt;                 /* number of soft page faults (i.e.
   48:                                            those serviced by reclaiming a page
   49:                                            from the list of pages awaiting
   50:                                            rellocation) */
   51:         long ru_majflt;                 /* number of hard page faults (i.e.
   52:                                            those that required I/O) */
   53:         long ru_nswap;                  /* number of times a process was swapped
   54:                                            out of physical memory */
   55:         long ru_inblock;                /* number of input operations via the
   56:                                            file system. Note this and
   57:                                            'ru_outblock' do not include
   58:                                            operations with the cache */
   59:         long ru_oublock;                /* number of output operations via the
   60:                                            file system */
   61:         long ru_msgsnd;                 /* number of IPC messages sent */
   62:         long ru_msgrcv;                 /* number of IPC messages received */
   63:         long ru_nsignals;               /* number of signals delivered */
   64:         long ru_nvcsw;                  /* number of voluntary context switches,
   65:                                            i.e. because the process gave up the
   66:                                            process before it had to (usually to
   67:                                            wait for some resouce to be
```

include/fiwix/resource.h


```
68:                                             availabe */
69:         long ru_nivcsw;                   /* number of involuntary context
70:                                             switches. i.e. a higher priority
71:                                             process became runnable or the
72:                                             current process used up its time
73:                                             slice */
74: };
75:
76: struct rlimit {
77:         int rlim_cur;                     /* the current (soft) limit */
78:         int rlim_max;                     /* the maximum (hard) limit */
79: };
80:
81: #endif /* _FIWIX_RESOURCE_H */
```

include/fiwix/sched.h


```
 1: /*
 2:  * fiwix/include/fiwix/sched.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_SCHED_H
 9: #define _FIWIX_SCHED_H
10:
11: #include <fiwix/process.h>
12:
13: #define PRIO_PROCESS    0
14: #define PRIO_PGRP       1
15: #define PRIO_USER       2
16:
17: #define PROC_UNUSED     0
18: #define PROC_RUNNING    1
19: #define PROC_SLEEPING   2
20: #define PROC_ZOMBIE     3
21: #define PROC_STOPPED    4
22: #define PROC_IDLE       5
23:
24: #define PROC_INTERRUPTIBLE      1
25: #define PROC_UNINTERRUPTIBLE    2
26:
27: #define DEF_PRIORITY    (20 * HZ / 100) /* 200ms of time slice */
28:
29: extern int need_resched;
30:
31: #define SI_LOAD_SHIFT   16
32:
33: /*
34:  * This was brougth from Linux 2.0.30 (sched.h).
35:  * Copyright Linus Torvalds et al.
36:  */
37: extern unsigned int avenrun[3];         /* Load averages */
38: #define FSHIFT          11              /* nr of bits of precision */
39: #define FIXED_1         (1<<FSHIFT)     /* 1.0 as fixed-point */
40: #define LOAD_FREQ       (5*HZ)          /* 5 sec intervals */
41: #define EXP_1           1884            /* 1/exp(5sec/1min) as fixed-point */
42: #define EXP_5           2014            /* 1/exp(5sec/5min) */
43: #define EXP_15          2037            /* 1/exp(5sec/15min) */
44:
45: #define CALC_LOAD(load,exp,n) \
46:         load *= exp; \
47:         load += n*(FIXED_1-exp); \
48:         load >>= FSHIFT;
49: /* ------------------------------------------------------------------------ */
50:
51:
52: void do_sched(void);
53: void set_tss(struct proc *);
54: void sched_init(void);
55:
56: #endif /* _FIWIX_SCHED_H */
```

include/fiwix/segments.h


```
 1: /*
 2:  * fiwix/include/fiwix/segments.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_SEGMENTS_H
 9: #define _FIWIX_SEGMENTS_H
10:
11: #include <fiwix/types.h>
12:
13: #define NR_GDT_ENTRIES  6       /* entries in GDT descriptor */
14: #define NR_IDT_ENTRIES  256     /* entries in IDT descriptor */
15:
16: /* low flags of Segment Descriptors */
17: #define SD_DATA         0x02    /* DATA Read/Write */
18: #define SD_CODE         0x0A    /* CODE Exec/Read */
19:
20: #define SD_32TSSA       0x09    /* 32-bit TSS (Available) */
21: #define SD_32TSSB       0x0B    /* 32-bit TSS (Busy) */
22: #define SD_32CALLGATE   0x0C    /* 32-bit Call Gate */
23: #define SD_32INTRGATE   0x0E    /* 32-bit Interrupt Gate (0D110) */
24: #define SD_32TRAPGATE   0x0F    /* 32-bit Trap Gate (0D111) */
25:
26: #define SD_CD           0x10    /* 0 = system / 1 = code/data */
27: #define SD_DPL0         0x00    /* priority level 0 */
28: #define SD_DPL1         0x20    /* priority level 1 (unused) */
29: #define SD_DPL2         0x40    /* priority level 2 (unused) */
30: #define SD_DPL3         0x60    /* priority level 3 (user) */
31: #define SD_PRESENT      0x80    /* segment present or valid */
32:
33: /* high flags Segment Descriptors */
34: #define SD_OPSIZE32     0x04    /* 32-bit code and data segments */
35: #define SD_PAGE4KB      0x08    /* page granularity (4KB) */
36:
37: /* low flags of the TSS Descriptors */
38: #define SD_TSSPRESENT   0x89    /* TSS present and not busy flag */
39:
40: #define USR_PL          3       /* User Privilege Level */
41:
42: /* EFLAGS */
43: #define EF_IOPL         12      /* IOPL bit */
44:
45: struct desc_r {
46:         __u16 limit;
47:         __u32 base_addr;
48: } __attribute__((packed));
49:
50: struct seg_desc {
51:         unsigned sd_lolimit : 16;       /* segment limit 0-15 bits */
52:         unsigned sd_lobase  : 24;       /* base address  0-23 bits */
53:         unsigned sd_loflags :  8;       /* flags (P, DPL, S and TYPE) */
54:         unsigned sd_hilimit :  4;       /* segment limit 16-19 bits */
55:         unsigned sd_hiflags :  4;       /* flags (G, DB, 0 and AVL) */
56:         unsigned sd_hibase  :  8;       /* base address 24-31 bits */
57: } __attribute__((packed));
58:
59: struct gate_desc {
60:         unsigned gd_looffset: 16;       /* offset 0-15 bits */
61:         unsigned gd_selector: 16;       /* segment selector */
62:         unsigned gd_flags   : 16;       /* flags (P, DPL, TYPE, 0 and NULL) */
63:         unsigned gd_hioffset: 16;       /* offset 16-31 bits */
64: } __attribute__((packed));
65:
66: void gdt_init(void);
67: void idt_init(void);
```

include/fiwix/segments.h


```
68:
69: #endif /* _FIWIX_SEGMENTS_H */
```

include/fiwix/sigcontext.h


```
 1: /*
 2:  * fiwix/include/fiwix/sigcontext.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_SIGCONTEXT_H
 9: #define _FIWIX_SIGCONTEXT_H
10:
11: struct sigcontext {
12:         unsigned int gs;
13:         unsigned int fs;
14:         unsigned int es;
15:         unsigned int ds;
16:         unsigned int edi;
17:         unsigned int esi;
18:         unsigned int ebp;
19:         unsigned int esp;
20:         int ebx;
21:         int edx;
22:         int ecx;
23:         int eax;
24:         int err;
25:         unsigned int eip;
26:         unsigned int cs;
27:         unsigned int eflags;
28:         unsigned int oldesp;
29:         unsigned int oldss;
30: };
31:
32: #endif /* _FIWIX_SIGCONTEXT_H */
```

include/fiwix/signal.h


```
    1: /*
    2:  * fiwix/include/fiwix/signal.h
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #ifndef _FIWIX_SIGNAL_H
    9: #define _FIWIX_SIGNAL_H
   10:
   11: #define NSIG            32
   12:
   13: #define SIGHUP          1       /* Hangup or Reset */
   14: #define SIGINT          2       /* Interrupt */
   15: #define SIGQUIT         3       /* Quit */
   16: #define SIGILL          4       /* Illegal Instruction */
   17: #define SIGTRAP         5       /* Trace Trap */
   18: #define SIGABRT         6       /* Abort Instruction */
   19: #define SIGIOT          SIGABRT /* I/O Trap Instruction */
   20: #define SIGBUS          7       /* Bus Error */
   21: #define SIGFPE          8       /* Floating Point Exception */
   22: #define SIGKILL         9       /* Kill */
   23: #define SIGUSR1         10      /* User Defined #1 */
   24: #define SIGSEGV         11      /* Segmentation Violation */
   25: #define SIGUSR2         12      /* User Defined #2 */
   26: #define SIGPIPE         13      /* Broken Pipe */
   27: #define SIGALRM         14      /* Alarm Clock */
   28: #define SIGTERM         15      /* Software Termination */
   29: #define SIGSTKFLT       16      /* Stack Fault */
   30: #define SIGCHLD         17      /* Child Termination */
   31: #define SIGCONT         18      /* Continue */
   32: #define SIGSTOP         19      /* Stop */
   33: #define SIGTSTP         20      /* Terminal Stop */
   34: #define SIGTTIN         21      /* Background Read */
   35: #define SIGTTOU         22      /* Background Write */
   36: #define SIGURG          23      /* Urgent Data */
   37: #define SIGXCPU         24      /* CPU eXceeded */
   38: #define SIGXFSZ         25      /* File Size eXceeded */
   39: #define SIGVTALRM       26      /* Virtual Time Alarm */
   40: #define SIGPROF         27      /* Profile Alarm */
   41: #define SIGWINCH        28      /* Window Change */
   42: #define SIGIO           29      /* I/O Asyncronous */
   43: #define SIGPOLL         SIGIO
   44: #define SIGPWR          30      /* Power Fault */
   45: #define SIGUNUSED       31
   46:
   47: typedef unsigned long int __sigset_t;
   48: typedef void (*__sighandler_t)(int);
   49:
   50: struct sigaction {
   51:         __sighandler_t sa_handler;
   52:         __sigset_t sa_mask;
   53:         int sa_flags;
   54:         void (*sa_restorer)(void);
   55: };
   56:
   57: #define SIG_DFL         ((__sighandler_t)  0)
   58: #define SIG_IGN         ((__sighandler_t)  1)
   59: #define SIG_ERR         ((__sighandler_t) -1)
   60:
   61: /* bits in sa_flags */
   62: #define SA_NOCLDSTOP    0x00000001      /* don't send SIGCHLD when children stop
 */
   63: #define SA_NOCLDWAIT    0x00000002      /* don't create zombie on child death */
   64: #define SA_ONSTACK      0x08000000      /* invoke handler on alternate stack */
   65: #define SA_RESTART      0x10000000      /* automatically restart system call */
   66: #define SA_INTERRUPT    0x20000000      /* unused */
```

include/fiwix/signal.h


```
67:
68: /* don't automatically block signal when the handler is executing */
69: #define SA_NODEFER      0x40000000
70: #define SA_NOMASK       SA_NODEFER
71:
72: /* reset signal disposition to SIG_DFL before invoking handler */
73: #define SA_RESETHAND    0x80000000
74: #define SA_ONESHOT      SA_RESETHAND
75:
76: /* bits in the third argument to 'waitpid/wait4' */
77: #define WNOHANG         1       /* don't block waiting */
78: #define WUNTRACED       2       /* report status of stopped children */
79:
80: #define SIG_BLOCK       0       /* for blocking signals */
81: #define SIG_UNBLOCK     1       /* for unblocking signals */
82: #define SIG_SETMASK     2       /* for setting the signal mask */
83:
84: /* SIGKILL and SIGSTOP can't ever be set as blockable signals */
85: #define SIG_BLOCKABLE   (~(1 << (SIGKILL - 1)) | (1 << (SIGSTOP - 1)))
86:
87: #define SIG_MASK(sig)   (~(1 << ((sig) - 1)))
88:
89: int issig(void);
90: void psig(unsigned int);
91:
92: #endif /* _FIWIX_SIGNAL_H */
```

include/fiwix/sleep.h


```
 1: /*
 2:  * fiwix/include/fiwix/sleep.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_SLEEP_H
 9: #define _FIWIX_SLEEP_H
10:
11: #include <fiwix/process.h>
12:
13: #define AREA_TTY_READ   0x00000001
14: #define AREA_CALLOUT    0x00000002
15:
16: struct sleep {
17:         unsigned short int next;
18:         void *address;
19:         struct proc *proc;
20: };
21:
22: struct resource {
23:         char locked;
24:         char wanted;
25: };
26:
27: int sleep(void *, int);
28: void wakeup(void *);
29: void wakeup_proc(struct proc *);
30:
31: void lock_resource(struct resource *);
32: void unlock_resource(struct resource *);
33: int lock_area(unsigned int);
34: int unlock_area(unsigned int);
35:
36: void sleep_init(void);
37:
38: #endif /* _FIWIX_SLEEP_H */
```

include/fiwix/statbuf.h


```
 1: /*
 2:  * fiwix/include/fiwix/statbuf.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_STATBUF_H
 9: #define _FIWIX_STATBUF_H
10:
11: struct old_stat {
12:         __dev_t st_dev;
13:         unsigned short int st_ino;
14:         __mode_t st_mode;
15:         __nlink_t st_nlink;
16:         __uid_t st_uid;
17:         __gid_t st_gid;
18:         __dev_t st_rdev;
19:         unsigned int st_size;
20:         __time_t st_atime;
21:         __time_t st_mtime;
22:         __time_t st_ctime;
23: };
24:
25: struct new_stat {
26:         __dev_t st_dev;
27:         unsigned short int __pad1;
28:         __ino_t st_ino;
29:         __mode_t st_mode;
30:         __nlink_t st_nlink;
31:         __uid_t st_uid;
32:         __gid_t st_gid;
33:         __dev_t st_rdev;
34:         unsigned short int __pad2;
35:         __off_t st_size;
36:         __blk_t st_blksize;
37:         __blk_t st_blocks;
38:         __time_t st_atime;
39:         unsigned int __unused1;
40:         __time_t st_mtime;
41:         unsigned int __unused2;
42:         __time_t st_ctime;
43:         unsigned int __unused3;
44:         unsigned int __unused4;
45:         unsigned int __unused5;
46: };
47:
48: #endif /* _FIWIX_STATBUF_H */
```

include/fiwix/statfs.h


```
 1: /*
 2:  * fiwix/include/fiwix/statfs.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_STATFS_H
 9: #define _FIWIX_STATFS_H
10:
11: typedef struct {
12:         long int val[2];
13: } fsid_t;
14:
15: struct statfs {
16:         long int f_type;
17:         long int f_bsize;
18:         long int f_blocks;
19:         long int f_bfree;
20:         long int f_bavail;
21:         long int f_files;
22:         long int f_ffree;
23:         fsid_t f_fsid;
24:         long int f_namelen;
25:         long int f_spare[6];
26: };
27:
28: #endif /* _FIWIX_STATFS_H */
```

include/fiwix/stat.h


```
#ifndef _FIWIX_STAT_H
#define _FIWIX_STAT_H

#include <fiwix/statbuf.h>

/* Encoding of the file mode.  These are the standard Unix values,
   but POSIX.1 does not specify what values should be used.  */

#define S_IFMT          0170000         /* Type of file mask */

/* File types.  */
#define S_IFIFO         0010000         /* Named pipe (fifo) */
#define S_IFCHR         0020000         /* Character special */
#define S_IFDIR         0040000         /* Directory */
#define S_IFBLK         0060000         /* Block special */
#define S_IFREG         0100000         /* Regular */
#define S_IFLNK         0120000         /* Symbolic link */
#define S_IFSOCK        0140000         /* Socket */

/* Protection bits.  */
#define S_IXUSR         00100           /* USER   --x------ */
#define S_IWUSR         00200           /* USER   -w------- */
#define S_IRUSR         00400           /* USER   r-------- */
#define S_IRWXU         00700           /* USER   rwx------ */

#define S_IXGRP         00010           /* GROUP  -----x--- */
#define S_IWGRP         00020           /* GROUP  ----w---- */
#define S_IRGRP         00040           /* GROUP  ---r----- */
#define S_IRWXG         00070           /* GROUP  ---rwx--- */

#define S_IXOTH         00001           /* OTHERS --------x */
#define S_IWOTH         00002           /* OTHERS -------w- */
#define S_IROTH         00004           /* OTHERS ------r-- */
#define S_IRWXO         00007           /* OTHERS ------rwx */

#define S_ISUID         0004000         /* set user id on execution */
#define S_ISGID         0002000         /* set group id on execution */
#define S_ISVTX         0001000         /* sticky bit */

#define S_IREAD         S_IRUSR         /* Read by owner.  */
#define S_IWRITE        S_IWUSR         /* Write by owner.  */
#define S_IEXEC         S_IXUSR         /* Execute by owner.  */

#define S_ISFIFO(m)     (((m) & S_IFMT) == S_IFIFO)
#define S_ISCHR(m)      (((m) & S_IFMT) == S_IFCHR)
#define S_ISDIR(m)      (((m) & S_IFMT) == S_IFDIR)
#define S_ISBLK(m)      (((m) & S_IFMT) == S_IFBLK)
#define S_ISREG(m)      (((m) & S_IFMT) == S_IFREG)
#define S_ISLNK(m)      (((m) & S_IFMT) == S_IFLNK)
#define S_ISSOCK(m)     (((m) & S_IFMT) == S_IFSOCK)

#define TO_READ         4       /* test for read permission */
#define TO_WRITE        2       /* test for write permission */
#define TO_EXEC         1       /* test for execute permission */

#endif /* _FIWIX_STAT_H */
```

include/fiwix/stdarg.h


```
 1: /*
 2: Copyright (C) 1988 Free Software Foundation
 3:
 4: This file is part of GNU CC.
 5:
 6: GNU CC is distributed in the hope that it will be useful,
 7: but WITHOUT ANY WARRANTY.  No author or distributor
 8: accepts responsibility to anyone for the consequences of using it
 9: or for whether it serves any particular purpose or works at all,
10: unless he says so in writing.  Refer to the GNU CC General Public
11: License for full details.
12:
13: Everyone is granted permission to copy, modify and redistribute
14: GNU CC, but only under the conditions described in the
15: GNU CC General Public License.   A copy of this license is
16: supposed to have been given to you along with GNU CC so you
17: can know your rights and responsibilities.  It should be in a
18: file named COPYING.  Among other things, the copyright notice
19: and this notice must be preserved on all copies.
20: */
21:
22: #ifndef __stdarg_h
23: #define __stdarg_h
24:
25: typedef char *va_list;
26:
27: /* Amount of space required in an argument list for an arg of type TYPE.
28:    TYPE may alternatively be an expression whose type is used.  */
29:
30: #define __va_rounded_size(TYPE)  \
31:   (((sizeof (TYPE) + sizeof (int) - 1) / sizeof (int)) * sizeof (int))
32:
33: #define va_start(AP, LASTARG) \
34:  (AP = ((char *) &(LASTARG) + __va_rounded_size(LASTARG)))
35:
36: extern void va_end (va_list);
37: #define va_end(AP) /* Nothing */
38:
39: #define va_arg(AP, TYPE) (AP += __va_rounded_size (TYPE), \
40:   *((TYPE *) (AP - __va_rounded_size (TYPE))))
41:
42: #endif /* __stdarg_h */
```

include/fiwix/stdio.h


```
 1: /*
 2:  * fiwix/include/fiwix/stdio.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _INCLUDE_STDIO_H
 9: #define _INCLUDE_STDIO_H
10:
11: void register_console(void (*fn)(char *, unsigned int));
12: void printk(const char *, ...);
13: int sprintk(char *, const char *, ...);
14:
15: #endif /* _INCLUDE_STDIO_H */
```

include/fiwix/string.h


```
 1: /*
 2:  * fiwix/include/fiwix/string.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _INCLUDE_STRING_H
 9: #define _INCLUDE_STRING_H
10:
11: #include <fiwix/types.h>
12:
13: #ifndef NULL
14: #define NULL    '\0'    /* ((void *)0) */
15: #endif
16:
17: #define MIN(a,b)        ((a) < (b) ? (a) : (b))
18: #define MAX(a,b)        ((a) > (b) ? (a) : (b))
19:
20: #define IS_NUMERIC(c)   ((c) >= '0' && (c) <= '9')
21: #define IS_SPACE(c)     ((c) == ' ')
22:
23: void swap_asc_word(char *, int);
24: int strcmp(const char *, const char *);
25: int strncmp(const char *, const char *, __ssize_t);
26: char * strcpy(char *, const char *);
27: void strncpy(char *, const char *, int);
28: char * strcat(char *, const char *);
29: char * strncat(char *, const char *, __ssize_t);
30: int strlen(const char *);
31: char * get_basename(const char *);
32: char * remove_trailing_slash(char *);
33: int is_dir(const char *);
34: int atoi(const char *);
35: void memcpy_b(void *, const void *, unsigned int);
36: void memcpy_w(void *, const void *, unsigned int);
37: void memcpy_l(void *, const void *, unsigned int);
38: void memset_b(void *, unsigned char, unsigned int);
39: void memset_w(void *, unsigned short int, unsigned int);
40: void memset_l(void *, unsigned int, unsigned int);
41:
42: #endif /* _INCLUDE_STRING_H */
```

include/fiwix/syscalls.h


```
    1: /*
    2:  * fiwix/include/fiwix/syscalls.h
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #ifndef _FIWIX_SYSCALLS_H
    9: #define _FIWIX_SYSCALLS_H
   10:
   11: #include <fiwix/types.h>
   12: #include <fiwix/system.h>
   13: #include <fiwix/time.h>
   14: #include <fiwix/times.h>
   15: #include <fiwix/timeb.h>
   16: #include <fiwix/utime.h>
   17: #include <fiwix/statbuf.h>
   18: #include <fiwix/ustat.h>
   19: #include <fiwix/signal.h>
   20: #include <fiwix/utsname.h>
   21: #include <fiwix/resource.h>
   22: #include <fiwix/dirent.h>
   23: #include <fiwix/statfs.h>
   24: #include <fiwix/sigcontext.h>
   25: #include <fiwix/mman.h>
   26:
   27: #define NR_SYSCALLS     (sizeof(syscall_table) / sizeof(unsigned int))
   28:
   29: int do_syscall(unsigned int, int, int, int, int, int, struct sigcontext);
   30:
   31: int sys_exit(int);
   32: void do_exit(int);
   33: int sys_fork(int, int, int, int, int, struct sigcontext *);
   34: int sys_read(unsigned int, char *, int);
   35: int sys_write(unsigned int, const char *, int);
   36: int sys_open(const char *, int, __mode_t);
   37: int sys_close(unsigned int);
   38: int sys_waitpid(__pid_t, int *, int);
   39: int sys_creat(const char *, __mode_t);
   40: int sys_link(const char *, const char *);
   41: int sys_unlink(const char *);
   42: int sys_execve(const char *, char **, char **, int, int, struct sigcontext *);
   43: int sys_chdir(const char *);
   44: int sys_time(__time_t *);
   45: int sys_mknod(const char *, __mode_t, __dev_t);
   46: int sys_chmod(const char *, __mode_t);
   47: int sys_chown(const char *, __uid_t, __gid_t);
   48: int sys_stat(const char *, struct old_stat *);
   49: int sys_lseek(unsigned int, __off_t, unsigned int);
   50: int sys_getpid(void);
   51: int sys_mount(const char *, const char *, const char *, unsigned long int, const
 void *);
   52: int sys_umount(const char *);
   53: int sys_setuid(__uid_t);
   54: int sys_getuid(void);
   55: int sys_stime(__time_t *);
   56: int sys_alarm(unsigned int);
   57: int sys_fstat(unsigned int, struct old_stat *);
   58: int sys_pause(void);
   59: int sys_utime(const char *, struct utimbuf *);
   60: int sys_access(const char *, __mode_t);
   61: int sys_ftime(struct timeb *);
   62: void sys_sync(void);
   63: int sys_kill(__pid_t, __sigset_t);
   64: int sys_rename(const char *, const char *);
   65: int sys_mkdir(const char *, __mode_t);
   66: int sys_rmdir(const char *);
```

include/fiwix/syscalls.h


```
 67: int sys_dup(unsigned int);
 68: int sys_pipe(int *);
 69: int sys_times(struct tms *);
 70: int sys_brk(unsigned int);
 71: int sys_setgid(__gid_t);
 72: int sys_getgid(void);
 73: unsigned int sys_signal(__sigset_t, void(*sighandler)(int));
 74: int sys_geteuid(void);
 75: int sys_getegid(void);
 76: int sys_umount2(const char *, int);
 77: int sys_ioctl(unsigned int, int, unsigned long int);
 78: int sys_fcntl(int, int, unsigned long int);
 79: int sys_setpgid(__pid_t, __pid_t);
 80: int sys_olduname(struct oldold_utsname *);
 81: int sys_umask(__mode_t);
 82: int sys_chroot(const char *);
 83: int sys_ustat(__dev_t, struct ustat *);
 84: int sys_dup2(int, int);
 85: int sys_getppid(void);
 86: int sys_getpgrp(void);
 87: int sys_setsid(void);
 88: int sys_sigaction(__sigset_t, const struct sigaction *, struct sigaction *);
 89: int sys_sgetmask(void);
 90: int sys_ssetmask(int);
 91: int sys_setreuid(__uid_t, __uid_t);
 92: int sys_setregid(__gid_t, __gid_t);
 93: int sys_sigsuspend(__sigset_t *);
 94: int sys_sigpending(__sigset_t *);
 95: int sys_sethostname(const char *, int);
 96: int sys_setrlimit(int, const struct rlimit *);
 97: int sys_getrlimit(int, struct rlimit *);
 98: int sys_getrusage(int, struct rusage *);
 99: int sys_gettimeofday(struct timeval *, struct timezone *);
100: int sys_settimeofday(const struct timeval *, const struct timezone *);
101: int sys_getgroups(__ssize_t, __gid_t *);
102: int sys_setgroups(__ssize_t, const __gid_t *);
103: int old_select(unsigned long int *);
104: int sys_symlink(const char *, const char *);
105: int sys_lstat(const char *, struct old_stat *);
106: int sys_readlink(const char *, char *, __size_t);
107: int sys_reboot(int, int, int);
108: int old_mmap(struct mmap *);
109: int sys_munmap(unsigned int, __size_t);
110: int sys_truncate(const char *, __off_t);
111: int sys_ftruncate(int, __off_t);
112: int sys_fchmod(int, __mode_t);
113: int sys_fchown(int, __uid_t, __gid_t);
114: int sys_statfs(const char *, struct statfs *);
115: int sys_fstatfs(unsigned int, struct statfs *);
116: int sys_ioperm(unsigned long int, unsigned long int, int);
117: int sys_socketcall(int, unsigned long int *);
118: int sys_setitimer(int, const struct itimerval *, struct itimerval *);
119: int sys_getitimer(int, struct itimerval *);
120: int sys_newstat(const char *, struct new_stat *);
121: int sys_newlstat(const char *, struct new_stat *);
122: int sys_newfstat(unsigned int, struct new_stat *);
123: int sys_uname(struct old_utsname *);
124: int sys_iopl(int, int, int, int, int, struct sigcontext *);
125: int sys_wait4(__pid_t, int *, int, struct rusage *);
126: int sys_sysinfo(struct sysinfo *);
127: int sys_fsync(int);
128: int sys_sigreturn(unsigned int, int, int, int, int, struct sigcontext *);
129: int sys_setdomainname(const char *, int);
130: int sys_newuname(struct new_utsname *);
131: int sys_mprotect(unsigned int, __size_t, int);
132: int sys_sigprocmask(int, const __sigset_t *, __sigset_t *);
133: int sys_getpgid(__pid_t);
```

include/fiwix/syscalls.h


```
  134: int sys_fchdir(unsigned int);
  135: int sys_personality(unsigned long int);
  136: int sys_setfsuid(__uid_t);
  137: int sys_setfsgid(__gid_t);
  138: int sys_llseek(unsigned int, unsigned long int, unsigned long int, __loff_t *, u
nsigned int);
  139: int sys_getdents(unsigned int, struct dirent *, unsigned int);
  140: int sys_select(int, fd_set *, fd_set *, fd_set *, struct timeval *);
  141: int sys_flock(int, int);
  142: int sys_getsid(__pid_t);
  143: int sys_fdatasync(int);
  144: int sys_nanosleep(const struct timespec *, struct timespec *);
  145: int sys_getcwd(char *, __size_t);
  146:
  147: #endif /* _FIWIX_SYSCALLS_H */
```

include/fiwix/system.h


```
 1: /*
 2:  * fiwix/include/fiwix/system.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_SYSTEM_H
 9: #define _FIWIX_SYSTEM_H
10:
11: #define UTS_SYSNAME     "Fiwix"
12: #define UTS_NODENAME    "(none)"
13: #define UTS_RELEASE     "1.1.0"
14: #define UTS_DOMAINNAME  "(none)"
15:
16: struct sysinfo {
17:         long int uptime;                /* seconds since boot */
18:         unsigned long int loads[3];     /* load average (1, 5 and 15 minutes) */
19:         unsigned long int totalram;     /* total usable main memory size */
20:         unsigned long int freeram;      /* available memory size */
21:         unsigned long int sharedram;    /* amount of shared memory */
22:         unsigned long int bufferram;    /* amount of memory used by buffers */
23:         unsigned long int totalswap;    /* total swap space size */
24:         unsigned long int freeswap;     /* available swap space */
25:         unsigned short int procs;       /* number of current processes */
26:         char _f[22];                    /* pads structure to 64 bytes */
27: };
28:
29: #endif /* _FIWIX_SYSTEM_H */
```

include/fiwix/termbits.h


```
  1: /*
  2:  * fiwix/include/fiwix/termbits.h
  3:  *
  4:  */
  5:
  6: #ifndef _FIWIX_TERMBITS_H
  7: #define _FIWIX_TERMBITS_H
  8:
  9: /* These definitions match those used by the 4.4 BSD kernel.
 10:    If the operating system has termios system calls or ioctls that
 11:    correctly implement the POSIX.1 behavior, there should be a
 12:    system-dependent version of this file that defines 'struct termios',
 13:    'tcflag_t', 'cc_t', 'speed_t' and the 'TC*' constants appropriately.  */
 14:
 15: /* Type of terminal control flag masks.  */
 16: typedef unsigned long int tcflag_t;
 17:
 18: /* Type of control characters.  */
 19: typedef unsigned char cc_t;
 20:
 21: /* Type of baud rate specifiers.  */
 22: typedef long int speed_t;
 23:
 24: #define NCCS 19
 25:
 26: /* Terminal control structure.  */
 27: struct termios {
 28:         tcflag_t c_iflag;       /* Input mode flags */
 29:         tcflag_t c_oflag;       /* Output mode flags */
 30:         tcflag_t c_cflag;       /* Control mode flags */
 31:         tcflag_t c_lflag;       /* Local mode flags */
 32:         cc_t c_line;            /* Line discipline */
 33:         cc_t c_cc[NCCS];        /* Control characters */
 34: };
 35:
 36: /* c_iflag bits */
 37: #define IGNBRK  0000001         /* Ignore break condition */
 38: #define BRKINT  0000002         /* Signal interrupt on break */
 39: #define IGNPAR  0000004         /* Ignore characters with parity errors */
 40: #define PARMRK  0000010         /* Mark parity and framing errors */
 41: #define INPCK   0000020         /* Enable input parity check */
 42: #define ISTRIP  0000040         /* Strip 8th bit off characters */
 43: #define INLCR   0000100         /* Map NL to CR on input */
 44: #define IGNCR   0000200         /* Ignore CR */
 45: #define ICRNL   0000400         /* Map CR to NL on input */
 46: #define IUCLC   0001000         /* Convert to lowercase */
 47: #define IXON    0002000         /* Enable start/stop output control */
 48: #define IXANY   0004000         /* Any character will restart after stop */
 49: #define IXOFF   0010000         /* Enable start/stop input control */
 50: #define IMAXBEL 0020000         /* Ring bell when input queue is full */
 51:
 52: /* c_oflag bits */
 53: #define OPOST   0000001         /* Perform output processing */
 54: #define OLCUC   0000002
 55: #define ONLCR   0000004         /* Map NL to CR-NL on output */
 56: #define OCRNL   0000010
 57: #define ONOCR   0000020
 58: #define ONLRET  0000040
 59: #define OFILL   0000100
 60: #define OFDEL   0000200
 61: #define NLDLY   0000400
 62: #define   NL0   0000000
 63: #define   NL1   0000400
 64: #define CRDLY   0003000
 65: #define   CR0   0000000
 66: #define   CR1   0001000
 67: #define   CR2   0002000
```

include/fiwix/termbits.h


```
 68: #define   CR3   0003000
 69: #define TABDLY  0014000
 70: #define   TAB0  0000000
 71: #define   TAB1  0004000
 72: #define   TAB2  0010000
 73: #define   TAB3  0014000
 74: #define   XTABS 0014000
 75: #define BSDLY   0020000
 76: #define   BS0   0000000
 77: #define   BS1   0020000
 78: #define VTDLY   0040000
 79: #define   VT0   0000000
 80: #define   VT1   0040000
 81: #define FFDLY   0100000
 82: #define   FF0   0000000
 83: #define   FF1   0100000
 84:
 85: /* c_cflag bit meaning */
 86: #define CBAUD   0010017
 87: #define  B0     0000000         /* hang up */
 88: #define  B50    0000001         /* 50 baud */
 89: #define  B75    0000002         /* 75 baud */
 90: #define  B110   0000003         /* 110 baud */
 91: #define  B134   0000004         /* 134 baud */
 92: #define  B150   0000005         /* 150 baud */
 93: #define  B200   0000006         /* 200 baud */
 94: #define  B300   0000007         /* 300 baud */
 95: #define  B600   0000010         /* 600 baud */
 96: #define  B1200  0000011         /* 1200 baud */
 97: #define  B1800  0000012         /* 1800 baud */
 98: #define  B2400  0000013         /* 2400 baud */
 99: #define  B4800  0000014         /* 4800 baud */
100: #define  B9600  0000015         /* 9600 baud */
101: #define  B19200 0000016         /* 19200 baud */
102: #define  B38400 0000017         /* 38400 baud */
103: #define EXTA B19200
104: #define EXTB B38400
105: #define CSIZE   0000060         /* Number of bits per byte (mask) */
106: #define   CS5   0000000         /* 5 bits per byte */
107: #define   CS6   0000020         /* 6 bits per byte */
108: #define   CS7   0000040         /* 7 bits per byte */
109: #define   CS8   0000060         /* 8 bits per byte */
110: #define CSTOPB  0000100         /* Two stop bits instead of one */
111: #define CREAD   0000200         /* Enable receiver */
112: #define PARENB  0000400         /* Parity enable */
113: #define PARODD  0001000         /* Odd parity instead of even */
114: #define HUPCL   0002000         /* Hang up on last close */
115: #define CLOCAL  0004000         /* Ignore modem status lines */
116: #define CBAUDEX 0010000
117: #define    B57600 0010001
118: #define   B115200 0010002
119: #define   B230400 0010003
120: #define   B460800 0010004
121: #define   B500000 0010005
122: #define   B576000 0010006
123: #define   B921600 0010007
124: #define  B1000000 0010010
125: #define  B1152000 0010011
126: #define  B1500000 0010012
127: #define  B2000000 0010013
128: #define  B2500000 0010014
129: #define  B3000000 0010015
130: #define  B3500000 0010016
131: #define  B4000000 0010017
132: #define CIBAUD    002003600000  /* input baud rate (not used) */
133: #define CMSPAR    010000000000          /* mark or space (stick) parity */
134: #define CRTSCTS   020000000000          /* flow control */
```

include/fiwix/termbits.h


```
135:
136: /* c_lflag bits */
137: #define ISIG    0000001         /* Enable signals */
138: #define ICANON  0000002         /* Do erase and kill processing */
139: #define XCASE   0000004
140: #define ECHO    0000010         /* Enable echo */
141: #define ECHOE   0000020         /* Visual erase for ERASE */
142: #define ECHOK   0000040         /* Echo NL after KILL */
143: #define ECHONL  0000100         /* Echo NL even if echo is OFF */
144: #define NOFLSH  0000200         /* Disable flush after interrupt */
145: #define TOSTOP  0000400         /* Send SIGTTOU for background output */
146: #define ECHOCTL 0001000         /* Echo control characters as ^X */
147: #define ECHOPRT 0002000         /* Hardcopy visual erase */
148: #define ECHOKE  0004000         /* Visual erase for KILL */
149: #define FLUSHO  0010000         /* Output being flushed (state) */
150: #define PENDIN  0040000         /* Retype pending input (state) */
151: #define IEXTEN  0100000         /* Enable DISCARD and LNEXT */
152:
153: /* c_cc characters */
154: #define VINTR 0                 /* Interrupt character [ISIG] */
155: #define VQUIT 1                 /* Quit character [ISIG] */
156: #define VERASE 2                /* Erase character [ICANON] */
157: #define VKILL 3                 /* Kill-line character [ICANON] */
158: #define VEOF 4                  /* End-of-file character [ICANON] */
159: #define VTIME 5                 /* Time-out value (1/10 secs) [!ICANON] */
160: #define VMIN 6                  /* Minimum # of bytes read at once [!ICANON] */
161: #define VSWTC 7
162: #define VSTART 8                /* Start (X-ON) character [IXON, IXOFF] */
163: #define VSTOP 9                 /* Stop (X-OFF) character [IXON, IXOFF] */
164: #define VSUSP 10                /* Suspend character [ISIG] */
165: #define VEOL 11                 /* End-of-line character [ICANON] */
166: #define VREPRINT 12             /* Reprint-line character [ICANON] */
167: #define VDISCARD 13             /* Discard character [IEXTEN] */
168: #define VWERASE 14              /* Word-erase character [ICANON] */
169: #define VLNEXT 15               /* Literal-next character [IEXTEN] */
170: #define VEOL2 16                /* Second EOL character [ICANON] */
171:
172: /* Values for the ACTION argument to 'tcflow'.  */
173: #define TCOOFF          0       /* Suspend output */
174: #define TCOON           1       /* Restart suspended output */
175: #define TCIOFF          2       /* Send a STOP character */
176: #define TCION           3       /* Send a START character */
177:
178: /* Values for the QUEUE_SELECTOR argument to 'tcflush'.  */
179: #define TCIFLUSH        0       /* Discard data received but not yet read */
180: #define TCOFLUSH        1       /* Discard data written but not yet sent */
181: #define TCIOFLUSH       2       /* Discard all pending data */
182:
183: /* Values for the OPTIONAL_ACTIONS argument to 'tcsetattr'.  */
184: #define TCSANOW         0       /* Change immediately */
185: #define TCSADRAIN       1       /* Change when pending output is written */
186: #define TCSAFLUSH       2       /* Flush pending input before changing */
187:
188: #endif /* _FIWIX_TERMBITS_H */
```

include/fiwix/termios.h


```
 1: /*
 2:  * fiwix/include/fiwix/termios.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_TERMIOS_H
 9: #define _FIWIX_TERMIOS_H
10:
11: #include <fiwix/termbits.h>
12:
13: struct winsize {
14:         unsigned short int ws_row;
15:         unsigned short int ws_col;
16:         unsigned short int ws_xpixel;
17:         unsigned short int ws_ypixel;
18: };
19:
20: #define NCC     8
21:
22: struct termio {
23:         unsigned short int c_iflag;     /* input mode flags */
24:         unsigned short int c_oflag;     /* output mode flags */
25:         unsigned short int c_cflag;     /* control mode flags */
26:         unsigned short int c_lflag;     /* local mode flags */
27:         unsigned char c_line;           /* line discipline */
28:         unsigned char c_cc[NCC];        /* control characters */
29: };
30:
31: #endif /* _FIWIX_TERMIOS_H */
```

include/fiwix/timeb.h


```
 1: /*
 2:  * fiwix/include/fiwix/timeb.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_TIMEB_H
 9: #define _FIWIX_TIMEB_H
10:
11: struct timeb {
12:         unsigned int time;              /* in seconds since Epoch */
13:         unsigned short int millitm;     /* additional milliseconds */
14:         short int timezone;             /* minutes west of GMT */
15:         short int dstflag;              /* nonzero if DST is used */
16: };
17:
18: #endif /* _FIWIX_TIMEB_H */
```

include/fiwix/time.h


```
 1: /*
 2:  * fiwix/include/fiwix/time.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_TIME_H
 9: #define _FIWIX_TIME_H
10:
11: #define ITIMER_REAL     0
12: #define ITIMER_VIRTUAL  1
13: #define ITIMER_PROF     2
14:
15: struct timespec {
16:         long int tv_sec;        /* seconds since 00:00:00, 1 Jan 1970 UTC */
17:         long int tv_nsec;       /* nanoseconds (1000000000ns = 1sec) */
18: };
19:
20: struct timeval {
21:         long int tv_sec;        /* seconds since 00:00:00, 1 Jan 1970 UTC */
22:         long int tv_usec;       /* microseconds (1000000us = 1sec) */
23: };
24:
25: struct timezone {
26:         int tz_minuteswest;     /* minutes west of GMT */
27:         int tz_dsttime;         /* type of DST correction */
28: };
29:
30: struct itimerval {
31:         struct timeval it_interval;
32:         struct timeval it_value;
33: };
34:
35: struct mt {
36:         int mt_sec;
37:         int mt_min;
38:         int mt_hour;
39:         int mt_day;
40:         int mt_month;
41:         int mt_year;
42: };
43:
44: unsigned long int tv2ticks(const struct timeval *);
45: void ticks2tv(long int, struct timeval *);
46: int setitimer(int, const struct itimerval *, struct itimerval *);
47: unsigned long int mktime(struct mt *);
48:
49: #endif /* _FIWIX_TIME_H */
```

include/fiwix/timer.h


```
 1: /*
 2:  * fiwix/include/fiwix/timer.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_TIMER_H
 9: #define _FIWIX_TIMER_H
10:
11: #include <fiwix/types.h>
12: #include <fiwix/sigcontext.h>
13:
14: #define TIMER_IRQ       0
15: #define HZ              100     /* kernel's Hertz rate (100 = 10ms) */
16: #define TICK            (1000000 / HZ)
17:
18: #define UNIX_EPOCH      1970
19:
20: #define LEAP_YEAR(y)    ((y % 4) == 0 && ((y % 100) != 0 || (y % 400) == 0))
21: #define DAYS_PER_YEAR(y)        ((LEAP_YEAR(y)) ? 366 : 365)
22:
23: #define SECS_PER_MIN    60
24: #define SECS_PER_HOUR   (SECS_PER_MIN * 60)
25: #define SECS_PER_DAY    (SECS_PER_HOUR * 24)
26:
27: #define INFINITE_WAIT   0xFFFFFFFF
28:
29: struct callout {
30:         int expires;
31:         void (*fn)(unsigned int);
32:         unsigned int arg;
33:         struct callout *next;
34: };
35:
36: struct callout_req {
37:         void (*fn)(unsigned int);
38:         unsigned int arg;
39: };
40:
41: void add_callout(struct callout_req *, unsigned int);
42: void del_callout(struct callout_req *);
43: void irq_timer(struct sigcontext *);
44: void timer_bh(void);
45: void callouts_bh(void);
46: void get_system_time(void);
47: void set_system_time(__time_t);
48: void timer_init(void);
49:
50: #endif /* _FIWIX_TIMER_H */
```

include/fiwix/times.h


```
 1: /*
 2:  * fiwix/include/fiwix/times.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_TIMES_H
 9: #define _FIWIX_TIMES_H
10:
11: struct tms {
12:         __clock_t tms_utime;    /* CPU time spent in user-mode */
13:         __clock_t tms_stime;    /* CPU time spent in kernel-mode */
14:         __clock_t tms_cutime;   /* (tms_utime + tms_cutime) of children */
15:         __clock_t tms_cstime;   /* (tms_stime + tms_cstime) of children */
16: };
17:
18: #endif /* _FIWIX_TIMES_H */
```

include/fiwix/traps.h


```
 1: /*
 2:  * fiwix/include/fiwix/traps.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_TRAPS_H
 9: #define _FIWIX_TRAPS_H
10:
11: #include <fiwix/sigcontext.h>
12:
13: #define NR_EXCEPTIONS   32
14:
15: struct traps {
16:         char *name;
17:         void (*handler)(unsigned int, struct sigcontext *);
18:         char errcode;
19: };
20:
21: void do_divide_error(unsigned int, struct sigcontext *);
22: void do_debug(unsigned int, struct sigcontext *);
23: void do_nmi_interrupt(unsigned int, struct sigcontext *);
24: void do_breakpoint(unsigned int, struct sigcontext *);
25: void do_overflow(unsigned int, struct sigcontext *);
26: void do_bound(unsigned int, struct sigcontext *);
27: void do_invalid_opcode(unsigned int, struct sigcontext *);
28: void do_no_math_coprocessor(unsigned int, struct sigcontext *);
29: void do_double_fault(unsigned int, struct sigcontext *);
30: void do_coprocessor_segment_overrun(unsigned int, struct sigcontext *);
31: void do_invalid_tss(unsigned int, struct sigcontext *);
32: void do_segment_not_present(unsigned int, struct sigcontext *);
33: void do_stack_segment_fault(unsigned int, struct sigcontext *);
34: void do_general_protection(unsigned int, struct sigcontext *);
35: void do_page_fault(unsigned int, struct sigcontext *);
36: void do_reserved(unsigned int, struct sigcontext *);
37: void do_floating_point_error(unsigned int, struct sigcontext *);
38: void do_alignment_check(unsigned int, struct sigcontext *);
39: void do_machine_check(unsigned int, struct sigcontext *);
40: void do_simd_fault(unsigned int, struct sigcontext *);
41:
42: void trap_handler(unsigned int, struct sigcontext);
43:
44: int dump_registers(unsigned int, struct sigcontext *);
45:
46: #endif /* _FIWIX_TRAPS_H */
```

include/fiwix/tty.h


```
    1: /*
    2:  * fiwix/include/fiwix/tty.h
    3:  *
    4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
    5:  * Distributed under the terms of the Fiwix License.
    6:  */
    7:
    8: #ifndef _FIWIX_TTY_H
    9: #define _FIWIX_TTY_H
   10:
   11: #include <fiwix/termios.h>
   12: #include <fiwix/fs.h>
   13:
   14: #define NR_TTYS         15      /* maximum number of tty devices */
   15:
   16: #define CBSIZE          32      /* number of characters in cblock */
   17: #define NR_CB_QUEUE     10      /* number of cblocks per queue */
   18: #define CB_POOL_SIZE    128     /* number of cblocks in the central pool */
   19:
   20: #define LAST_CHAR(q)    ((q)->tail ? (q)->tail->data[(q)->tail->end_off - 1] : N
ULL)
   21:
   22: struct clist {
   23:         unsigned short int count;
   24:         unsigned short int cb_num;
   25:         struct cblock *head;
   26:         struct cblock *tail;
   27: };
   28:
   29: struct cblock {
   30:         unsigned short int start_off;
   31:         unsigned short int end_off;
   32:         unsigned char data[CBSIZE];
   33:         struct cblock *prev;
   34:         struct cblock *next;
   35: };
   36:
   37: struct tty {
   38:         __dev_t dev;
   39:         struct clist read_q;
   40:         struct clist cooked_q;
   41:         struct clist write_q;
   42:         unsigned short int count;
   43:         struct termios termios;
   44:         struct winsize winsize;
   45:         __pid_t pid, pgid, sid;
   46:         unsigned char lnext;
   47:         void *driver_data;
   48:         int canon_data;
   49:
   50:         /* formerly tty_operations */
   51:         void (*stop)(struct tty *);
   52:         void (*start)(struct tty *);
   53:         void (*deltab)(struct tty *);
   54:         void (*reset)(struct tty *);
   55:         void (*input)(struct tty *);
   56:         void (*output)(struct tty *);
   57: };
   58: extern struct tty tty_table[];
   59:
   60: int register_tty(__dev_t);
   61: struct tty * get_tty(__dev_t);
   62: void disassociate_ctty(struct tty *);
   63: void termios_reset(struct tty *);
   64: void do_cook(struct tty *);
   65: int tty_putchar(struct tty *, unsigned char);
   66: int tty_open(struct inode *, struct fd *);
```

include/fiwix/tty.h


```
67: int tty_close(struct inode *, struct fd *);
68: int tty_read(struct inode *, struct fd *, char *, __size_t);
69: int tty_write(struct inode *, struct fd *, const char *, __size_t);
70: int tty_ioctl(struct inode *, int cmd, unsigned long int);
71: int tty_lseek(struct inode *, __off_t);
72: int tty_select(struct inode *, int);
73: void tty_init(void);
74:
75: int tty_queue_putchar(struct tty *, struct clist *, unsigned char);
76: int tty_queue_unputchar(struct clist *);
77: unsigned char tty_queue_getchar(struct clist *);
78: void tty_queue_flush(struct clist *);
79: void tty_queue_init(struct tty *);
80:
81: int vt_ioctl(struct tty *, int, unsigned long int);
82:
83: #endif /* _FIWIX_TTY_H */
```

include/fiwix/types.h


```
 1: /*
 2:  * fiwix/include/fiwix/types.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_TYPES_H
 9: #define _FIWIX_TYPES_H
10:
11: typedef __signed__ char __s8;
12: typedef unsigned char __u8;
13: typedef __signed__ short int __s16;
14: typedef unsigned short int __u16;
15: typedef __signed__ int __s32;
16: typedef unsigned int __u32;
17:
18: typedef __u16 __uid_t;
19: typedef __u16 __gid_t;
20: typedef __u32 __ino_t;
21: typedef __u16 __mode_t;
22: typedef __u16 __nlink_t;
23: typedef __u32 __off_t;
24: typedef __s32 __pid_t;
25: typedef __s32 __ssize_t;
26: typedef __u32 __size_t;
27: typedef unsigned long int __clock_t;
28: typedef __u32 __time_t;
29: typedef __u16 __dev_t;
30: typedef __u16 __key_t;
31: typedef __s32 __blk_t;          /* must be signed in order to return -EIO */
32: typedef __s32 __daddr_t;
33: typedef unsigned long long int __loff_t;
34:
35: /* number of descriptors that can fit in an 'fd_set' */
36: /* WARNING: this value must be the same as in the C Library */
37: #define __FD_SETSIZE    64
38:
39: #define __NFDBITS       (sizeof(unsigned long int) * 8)
40: #define __FDELT(d)      ((d) / __NFDBITS)
41: #define __FDMASK(d)     (1 << ((d) % __NFDBITS))
42:
43: /* define the fd_set structure for select() */
44: typedef struct {
45:         unsigned long int fds_bits[__FD_SETSIZE / __NFDBITS];
46: } fd_set;
47:
48: #define __FD_ZERO(set)          (memset_b((void *) (set), 0, sizeof(fd_set)))
49: #define __FD_SET(d, set)        ((set)->fds_bits[__FDELT(d)] |= __FDMASK(d))
50: #define __FD_CLR(d, set)        ((set)->fds_bits[__FDELT(d)] &= ~__FDMASK(d))
51: #define __FD_ISSET(d, set)      ((set)->fds_bits[__FDELT(d)] & __FDMASK(d))
52:
53: #endif /* _FIWIX_TYPES_H */
```

include/fiwix/unistd.h


```
 1: /*
 2:  * fiwix/include/fiwix/unistd.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_UNISTD_H
 9: #define _FIWIX_UNISTD_H
10:
11: /* Linux 2.0.40 ABI (one syscall from Linux 2.2.26) */
12: /* #define SYS_setup */
13: #define SYS_exit                1
14: #define SYS_fork                2
15: #define SYS_read                3
16: #define SYS_write               4
17: #define SYS_open                5
18: #define SYS_close               6
19: #define SYS_waitpid             7
20: #define SYS_creat               8
21: #define SYS_link                9
22: #define SYS_unlink              10
23: #define SYS_execve              11
24: #define SYS_chdir               12
25: #define SYS_time                13
26: #define SYS_mknod               14
27: #define SYS_chmod               15
28: #define SYS_chown               16
29: #define SYS_break               17              /* -ENOSYS */
30: #define SYS_oldstat             18
31: #define SYS_lseek               19
32: #define SYS_getpid              20
33: #define SYS_mount               21
34: #define SYS_umount              22
35: #define SYS_setuid              23
36: #define SYS_getuid              24
37: #define SYS_stime               25
38: /* #define SYS_ptrace */
39: #define SYS_alarm               27
40: #define SYS_oldfstat            28
41: #define SYS_pause               29
42: #define SYS_utime               30
43: #define SYS_stty                31              /* -ENOSYS */
44: #define SYS_gtty                32              /* -ENOSYS */
45: #define SYS_access              33
46: /* #define SYS_nice */
47: #define SYS_ftime               35
48: #define SYS_sync                36
49: #define SYS_kill                37
50: #define SYS_rename              38
51: #define SYS_mkdir               39
52: #define SYS_rmdir               40
53: #define SYS_dup                 41
54: #define SYS_pipe                42
55: #define SYS_times               43
56: #define SYS_prof                44              /* -ENOSYS */
57: #define SYS_brk                 45
58: #define SYS_setgid              46
59: #define SYS_getgid              47
60: #define SYS_signal              48
61: #define SYS_geteuid             49
62: #define SYS_getegid             50
63: /* #define SYS_acct */
64: #define SYS_umount2             52      /* from 2.2.26, it was sys_phys() */
65: #define SYS_lock                53              /* -ENOSYS */
66: #define SYS_ioctl               54
67: #define SYS_fcntl               55
```

include/fiwix/unistd.h


```
 68: #define SYS_mpx                 56              /* -ENOSYS */
 69: #define SYS_setpgid             57
 70: #define SYS_ulimit              58              /* -ENOSYS */
 71: #define SYS_olduname            59
 72: #define SYS_umask               60
 73: #define SYS_chroot              61
 74: #define SYS_ustat               62
 75: #define SYS_dup2                63
 76: #define SYS_getppid             64
 77: #define SYS_getpgrp             65
 78: #define SYS_setsid              66
 79: #define SYS_sigaction           67
 80: #define SYS_sgetmask            68
 81: #define SYS_ssetmask            69
 82: #define SYS_setreuid            70
 83: #define SYS_setregid            71
 84: #define SYS_sigsuspend          72
 85: #define SYS_sigpending          73
 86: #define SYS_sethostname         74
 87: #define SYS_setrlimit           75
 88: #define SYS_getrlimit           76
 89: #define SYS_getrusage           77
 90: #define SYS_gettimeofday        78
 91: #define SYS_settimeofday        79
 92: #define SYS_getgroups           80
 93: #define SYS_setgroups           81
 94: #define SYS_oldselect           82
 95: #define SYS_symlink             83
 96: #define SYS_oldlstat            84
 97: #define SYS_readlink            85
 98: /* #define SYS_uselib */
 99: /* #define SYS_swapon */
100: #define SYS_reboot              88
101: /* #define SYS_oldreaddir */
102: #define SYS_old_mmap            90
103: #define SYS_munmap              91
104: #define SYS_truncate            92
105: #define SYS_ftruncate           93
106: #define SYS_fchmod              94
107: #define SYS_fchown              95
108: /* #define SYS_getpriority */
109: /* #define SYS_setpriority */
110: /* #define SYS_profil */
111: #define SYS_statfs              99
112: #define SYS_fstatfs             100
113: #define SYS_ioperm              101
114: #define SYS_socketcall          102
115: /* #define SYS_syslog */
116: #define SYS_setitimer           104
117: #define SYS_getitimer           105
118: #define SYS_newstat             106
119: #define SYS_newlstat            107
120: #define SYS_newfstat            108
121: #define SYS_uname               109
122: #define SYS_iopl                110
123: /* #define SYS_vhangup */
124: /* #define SYS_idle              112             -ENOSYS */
125: /* #define SYS_vm86old */
126: #define SYS_wait4               114
127: /* #define SYS_swapoff */
128: #define SYS_sysinfo             116
129: /* #define SYS_ipc */
130: #define SYS_fsync               118
131: #define SYS_sigreturn           119
132: /* #define SYS_clone */
133: #define SYS_setdomainname       121
134: #define SYS_newuname            122
```

include/fiwix/unistd.h


```
135: /* #define SYS_modify_ldt */
136: /* #define SYS_adjtimex */
137: #define SYS_mprotect            125
138: #define SYS_sigprocmask         126
139: /* #define SYS_create_module */
140: /* #define SYS_init_module */
141: /* #define SYS_delete_module */
142: /* #define SYS_get_kernel_syms */
143: /* #define SYS_quotactl */
144: #define SYS_getpgid             132
145: #define SYS_fchdir              133
146: /* #define SYS_bdflush */
147: /* #define SYS_sysfs */
148: #define SYS_personality         136
149: /* #define afs_syscall */
150: #define SYS_setfsuid            138
151: #define SYS_setfsgid            139
152: #define SYS_llseek              140
153: #define SYS_getdents            141
154: #define SYS_select              142
155: #define SYS_flock               143
156: /* #define SYS_msync */
157: /* #define SYS_readv */
158: /* #define SYS_writev */
159: #define SYS_getsid              147
160: #define SYS_fdatasync           148
161: /* #define SYS_sysctl */
162: /* #define SYS_mlock */
163: /* #define SYS_munlock */
164: /* #define SYS_mlockall */
165: /* #define SYS_munlockall */
166: /* #define SYS_sched_setparam */
167: /* #define SYS_sched_getparam */
168: /* #define SYS_sched_setscheduler */
169: /* #define SYS_sched_getscheduler */
170: /* #define SYS_sched_yield */
171: /* #define SYS_sched_get_priority_max */
172: /* #define SYS_sched_get_priority_min */
173: /* #define SYS_sched_rr_get_interval */
174: #define SYS_nanosleep           162
175: /* #define SYS_mremap */
176:
177: /* extra system calls from Linux 2.2.26 */
178: /* #define SYS_mremap */
179: /* #define SYS_setresuid */
180: /* #define SYS_getresuid */
181: /* #define SYS_ni_syscall */
182: /* #define SYS_query_module */
183: /* #define SYS_poll */
184: /* #define SYS_nfsservctl */
185: /* #define SYS_setresgid */
186: /* #define SYS_getresgid */
187: /* #define SYS_prctl */
188: /* #define SYS_rt_sigreturn_wrapper */
189: /* #define SYS_rt_sigaction */
190: /* #define SYS_rt_sigprocmask */
191: /* #define SYS_rt_sigpending */
192: /* #define SYS_rt_sigtimedwait */
193: /* #define SYS_rt_sigqueueinfo */
194: /* #define SYS_rt_sigsuspend_wrapper */
195: /* #define SYS_pread */
196: /* #define SYS_pwrite */
197: /* #define SYS_chown */
198: #define SYS_getcwd              183
199: /* #define SYS_capget */
200: /* #define SYS_capset */
201: /* #define SYS_sigaltstack_wrapper */
```

include/fiwix/unistd.h


```
202: /* #define SYS_sendfile */
203: /* #define SYS_ni_syscall */
204: /* #define SYS_ni_syscall */
205: #define SYS_vfork               190
206:
207: #endif /* _FIWIX_UNISTD_H */
```

include/fiwix/ustat.h


```
 1: /*
 2:  * fiwix/include/fiwix/ustat.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_USTAT_H
 9: #define _FIWIX_USTAT_H
10:
11: #include <fiwix/types.h>
12:
13: struct ustat {
14:         __daddr_t f_tfree;      /* total free blocks */
15:         __ino_t f_tinode;       /* number of free inodes */
16:         char f_fname;           /* filesystem name */
17:         char f_fpack;           /* filesystem pack name */
18: };
19:
20: #endif /* _FIWIX_USTAT_H */
```

include/fiwix/utime.h


```
 1: /*
 2:  * fiwix/include/fiwix/utime.h
 3:  *
 4:  * Copyright 2018, Jordi Sanfeliu. All rights reserved.
 5:  * Distributed under the terms of the Fiwix License.
 6:  */
 7:
 8: #ifndef _FIWIX_UTIME_H
 9: #define _FIWIX_UTIME_H
10:
11: #include <fiwix/types.h>
12:
13: struct utimbuf {
14:         __time_t actime;        /* access time */
15:         __time_t modtime;       /* modification time */
16: };
17:
18: #endif /* _FIWIX_UTIME_H */
```

include/fiwix/utsname.h


```
    1: /* Copyright (C) 1991, 1992, 1994, 1996 Free Software Foundation, Inc.
    2:    This file is part of the GNU C Library.
    3:
    4:    The GNU C Library is free software; you can redistribute it and/or
    5:    modify it under the terms of the GNU Library General Public License as
    6:    published by the Free Software Foundation; either version 2 of the
    7:    License, or (at your option) any later version.
    8:
    9:    The GNU C Library is distributed in the hope that it will be useful,
   10:    but WITHOUT ANY WARRANTY; without even the implied warranty of
   11:    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   12:    Library General Public License for more details.
   13:
   14:    You should have received a copy of the GNU Library General Public
   15:    License along with the GNU C Library; see the file COPYING.LIB.  If not,
   16:    write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330,
   17:    Boston, MA 02111-1307, USA.  */
   18:
   19: /*
   20:  *      POSIX Standard: 4.4 System Identification       <fiwix/utsname.h>
   21:  */
   22:
   23: #ifndef _FIWIX_UTSNAME_H
   24: #define _FIWIX_UTSNAME_H
   25:
   26: #define _OLD_UTSNAME_LENGTH     9
   27: #define _UTSNAME_LENGTH         65
   28:
   29: #ifndef _UTSNAME_NODENAME_LENGTH
   30: #define _UTSNAME_NODENAME_LENGTH _UTSNAME_LENGTH
   31: #endif
   32:
   33: /* Very OLD structure describing the system and machine.  */
   34: struct oldold_utsname
   35: {
   36:     char sysname[_OLD_UTSNAME_LENGTH];
   37:     char nodename[_OLD_UTSNAME_LENGTH];
   38:     char release[_OLD_UTSNAME_LENGTH];
   39:     char version[_OLD_UTSNAME_LENGTH];
   40:     char machine[_OLD_UTSNAME_LENGTH];
   41: };
   42:
   43: /* OLD structure describing the system and machine.  */
   44: struct old_utsname
   45: {
   46:     char sysname[_UTSNAME_LENGTH];
   47:     char nodename[_UTSNAME_NODENAME_LENGTH];
   48:     char release[_UTSNAME_LENGTH];
   49:     char version[_UTSNAME_LENGTH];
   50:     char machine[_UTSNAME_LENGTH];
   51: };
   52:
   53: /* NEW structure describing the system and machine.  */
   54: struct new_utsname
   55: {
   56:     /* Name of the implementation of the operating system.  */
   57:     char sysname[_UTSNAME_LENGTH];
   58:
   59:     /* Name of this node on the network.  */
   60:     char nodename[_UTSNAME_NODENAME_LENGTH];
   61:
   62:     /* Current release level of this implementation.  */
   63:     char release[_UTSNAME_LENGTH];
   64:     /* Current version level of this release.  */
   65:     char version[_UTSNAME_LENGTH];
   66:
   67:     /* Name of the hardware type the system is running on.  */
```

include/fiwix/utsname.h


```
68:     char machine[_UTSNAME_LENGTH];
69:     char domainname[_UTSNAME_LENGTH];
70: };
71:
72: extern struct new_utsname sys_utsname;
73: extern char UTS_MACHINE[_UTSNAME_LENGTH];
74:
75: #endif /* _FIWIX_UTSNAME_H */
```

include/fiwix/version.h


```
1: #define UTS_VERSION "Mon Mar 23 19:24:27 CET 2020"
```

include/fiwix/vt.h


```
    1: #ifndef VT_H
    2: #define VT_H
    3:
    4: /* prefix 0x56 is 'V', to avoid collision with termios and kd */
    5:
    6: #define VT_OPENQRY      0x5600  /* find available vt */
    7:
    8: struct vt_mode {
    9:         char mode;              /* vt mode */
   10:         char waitv;             /* if set, hang on writes if not active */
   11:         short int relsig;       /* signal to raise on release req */
   12:         short int acqsig;       /* signal to raise on acquisition */
   13:         short int frsig;        /* unused (set to 0) */
   14: };
   15: #define VT_GETMODE      0x5601  /* get mode of active vt */
   16: #define VT_SETMODE      0x5602  /* set mode of active vt */
   17: #define VT_AUTO         0x00    /* auto vt switching */
   18: #define VT_PROCESS      0x01    /* process controls switching */
   19: #define VT_ACKACQ       0x02    /* acknowledge switch */
   20:
   21: struct vt_stat {
   22:         unsigned short int v_active;    /* active vt */
   23:         unsigned short int v_signal;    /* signal to send */
   24:         unsigned short int v_state;     /* vt bitmask */
   25: };
   26: #define VT_GETSTATE     0x5603  /* get global vt state info */
   27: #define VT_SENDSIG      0x5604  /* signal to send to bitmask of vts */
   28:
   29: #define VT_RELDISP      0x5605  /* release display */
   30:
   31: #define VT_ACTIVATE     0x5606  /* make vt active */
   32: #define VT_WAITACTIVE   0x5607  /* wait for vt active */
   33: #define VT_DISALLOCATE  0x5608  /* free memory associated to vt */
   34:
   35: struct vt_sizes {
   36:         unsigned short int v_rows;      /* number of rows */
   37:         unsigned short int v_cols;      /* number of columns */
   38:         unsigned short int v_scrollsize;/* number of lines of scrollback */
   39: };
   40: #define VT_RESIZE       0x5609          /* set kernel's idea of screensize */
   41:
   42: struct vt_consize {
   43:         unsigned short int v_rows;      /* number of rows */
   44:         unsigned short int v_cols;      /* number of columns */
   45:         unsigned short int v_vlin;      /* number of pixel rows on screen */
   46:         unsigned short int v_clin;      /* number of pixel rows per character */
   47:         unsigned short int v_vcol;      /* number of pixel columns on screen */
   48:         unsigned short int v_ccol;      /* number of pixel columns per character
 */
   49: };
   50: #define VT_RESIZEX      0x560A  /* set kernel's idea of screensize + more */
   51: #define VT_LOCKSWITCH   0x560B  /* disallow vt switching */
   52: #define VT_UNLOCKSWITCH 0x560C  /* allow vt switching */
   53:
   54: #endif /* VT_H */
```

README


```
                        Fiwix kernel release 1.1.0
                        ==========================

Fiwix is an operating system kernel, written by Jordi Sanfeliu from scratch,
based on the UNIX architecture and fully focused on being POSIX compatible.
It is designed and developed mainly as a hobby OS but also for educational
purposes, therefore the kernel code is kept as simple as possible.

It runs on the i386 (x86 32bit) hardware architecture and is compatible with
a good base of existing GNU applications. It offers many UNIX-like features:

 - Mostly written in C language (Assembly only used in the needed parts).
 - GRUB Multiboot Specification v1 compliant.
 - Full 32bit protected mode non-preemptive kernel.
 - For i386 processors and higher.
 - Preemptive multitasking.
 - Protected task environment (independent memory address per process).
 - POSIX-compliant (mostly).
 - Process groups, sessions and job control.
 - Interprocess communication with pipes and signals.
 - BSD file locking mechanism (POSIX restricted to file and advisory only).
 - Virtual memory management up to 4GB (1GB physical only and no swapping yet).
 - Demand paging with Copy-On-Write feature.
 - Linux 2.0 ABI system calls compatibility (mostly).
 - ELF-386 executable format support (statically and dynamically linked).
 - Round Robin based scheduler algorithm (no priorities yet).
 - VFS abstraction layer.
 - EXT2 filesystem support with 1KB, 2KB and 4KB block sizes.
 - Minix v1 and v2 filesystem support.
 - Linux-like PROC filesystem support (read only).
 - PIPE pseudo-filesystem support.
 - ISO9660 filesystem support with Rock Ridge extensions.
 - RAMdisk device support.
 - Initial RAMdisk (initrd) image support.
 - SVGAlib based applications support.
 - Keyboard driver with Linux keymaps support.
 - Parallel port printer driver support.
 - Basic implementation of a Pseudo-Random Number Generator.
 - Floppy disk device driver and DMA management.
 - IDE/ATA ATAPI CD-ROM device driver.
 - IDE/ATA hard disk device driver.

Fiwix is distributed under the terms of the MIT License, see the LICENSE file
for more details.


COMPILING
================================================================================
Before compiling you might want to tweak the kernel configuration by changing
the values in the 'include/fiwix/config.h' file.

The command needed to make a full compilation of the Fiwix kernel is:

make clean ; make

This will create the files 'fiwix' (the kernel itself) and 'System.map.gz' (the
symbol table) in the root directory of the source code tree.

Keep in mind that the kernel doesn't do anything on its own, you need to create
a user-space environment to make use of it. Upon booting, the kernel mounts the
root filesystem and tries to run '/sbin/init' on it, so you need to provide this
program yourself.


TESTING
================================================================================
To create a complete bootable floppy disk you need to download the Fiwix Media
```

README

Setup. Then insert a floppy disk into the drive and type the following:

```
make floppy
```

If you only want to update an existing floppy disk with a newer or modified kernel version, then type the following:

```
make floppy_update
```

If you don’t have a floppy drive but a bootable CD-ROM IDE/ATA drive, you can create your own Fiwix Installation CD-ROM by using your current operating system (i.e., GNU/Linux). To **do** this, you might want to use the scripts and tools that come with the Fiwix Media Setup archive.

The scripts to create such bootable images cannot be executed under Fiwix because they need support **for** the loop device and the ISO9660 creation tools.

INSTALLING

Please keep in mind that this is a kernel in its very early stages and may well have serious bugs and broken features which have not yet been identified or resolved.

Let me repeat that.

Please keep in mind that this is a kernel in its very early stages and may well have serious bugs and broken features which have not yet been identified or resolved.

```
*****************************
*** USE AT YOUR OWN RISK! ***
*****************************
```

You can proceed to install the FiwixOS on a hard disk either booting from the Live CD-ROM or from a floppy. If you chosen the latter, you will also need the Live CD-ROM inserted in order to install the packages that form all the system environment.

The minimal requirements to use Fiwix are as follows:

- Standard IBM PC-AT architecture.
- i386 processor or higher.
- 3MB of RAM memory (128MB recommended).
- IDE/ATAPI CD-ROM or floppy disk (3.5**", 1.44MB).**
- **500MB IDE hard disk (1GB recommended).**

Let the system boot either from a CD-ROM or floppy, and when you are ready just type:

```
install.sh
```

Happy hacking.

--

https://www.fiwix.org

LICENSE


```
MIT License

Copyright (c) 2018 Jordi Sanfeliu

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

```

CODING


```
Fiwix kernel coding standards
-----------------------------------------------------------------------------
It's easier on everyone if all authors working on a shared code base are
consistent in the way they write their programs. Fiwix has the following
conventions in its code:

- Use of snake_case (multi-word names are lower_case_with_underscores) for
  everything except for macro and constant names.

- No space after the name of a function in a call.
  For example, printk("hello") not printk ("hello").

- Optional space after keywords "if", "for", "while", "switch".
  For example, if(x) or if (x).

- Space before braces (except in function declarations).
  For example, if(x) { not if(x){.

- Space between operands.
  For example, for(n = 0; n < 10; n++), not for(n=0;n<10;n++).

- Beginning-of-line indentation via tabs, not spaces.

- Preprocessor macros are always UPPERCASE.

- Pointer types have spaces: (uint16_t *) not (uint16_t*).

- Comments in code are always as in C89 /* ... */.

- Multiline comments start always with a capital letter and the last sentence
  ends with a period.

- Functions that take no arguments are declared as f(void) not f().

- The open parenthesis is always on the same line as the function name.

- There is never a space between the function name and the open parenthesis.

- There is never a space between the parentheses and the parameters.

- The open curly brace is always at the next line of the function declaration.

- Conditionals should always include curly braces, even if there is only a
  single statement within the conditional.


Recommended for reading
-----------------------------------------------------------------------------
- http://skull.piratehaven.org/~bapper/298c/cstyle.html

```

THANKS


```
THANKS!

A huge THANKS to all people who created their own hobby, and not so hobby,
operating systems and made them freely available on Internet.

From those simple "Hello world!\n" kernels to a real self-hosting operating
system, their projects were a invaluable source of information and inspiration
to create the Fiwix kernel.

A special thanks to OSDEV Community for their tutorials, documents, wikis, etc.

Jordi Sanfeliu
http://www.fiwix.org

```

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents