Linux 0.11 Source Code Listing
by Linus Torvalds

Courtesy from http://oldlinux.org/
Source: http://oldlinux.org/bochs/linux-0.11-devel-040329.zip

Complied by Marconi Jiang marconi.jiang@gmail.com

Directory tree

```
/linux-0.11/
-rw-rw-rw-@  1 apple  staff  2887 Dec  6  1991 Makefile
drwxrwxrwx@  5 apple  staff   170 Dec  6  1991 boot
drwxrwxrwx@ 20 apple  staff   680 Dec  8  1991 fs
drwxrwxrwx@ 19 apple  staff   646 Dec  5  2015 include
|- drwxrwxrwx@  6 apple  staff   204 Sep 17  1991 asm
|- drwxrwxrwx@ 12 apple  staff   408 Nov  2  1991 linux
|- drwxrwxrwx@  7 apple  staff   238 Sep 17  1991 sys
drwxrwxrwx@  3 apple  staff   102 Dec  6  1991 init
drwxrwxrwx@ 19 apple  staff   646 Oct 15 18:45 kernel
|- drwxrwxrwx@ 8 apple  staff   272 Dec  8  1991 blk_drv
|- drwxrwxrwx@ 9 apple  staff   306 Dec  9  1991 chr_drv
|- drwxrwxrwx@ 4 apple  staff   136 Dec  8  1991 math
drwxrwxrwx@ 15 apple  staff   510 Dec  8  1991 lib
drwxrwxrwx@  5 apple  staff   170 Dec  8  1991 mm
drwxrwxrwx@  3 apple  staff   102 Dec  4  1991 tools
```

Directory tree & files

```
/linux-0.11/boot
|-rw-rw-rw-  1 apple  staff  5052 Dec  6  1991 bootsect.s
|-rw-rw-rw-@ 1 apple  staff  5938 Nov 18  1991 head.s
|-rw-rw-rw-@ 1 apple  staff  5364 Dec  6  1991 setup.s

/linux-0.11/fs
|-rw-rw-rw-@ 1 apple  staff   5053 Dec  2  1991 Makefile
|-rw-rw-rw-@ 1 apple  staff   4042 Nov 27  1991 bitmap.c
|-rw-rw-rw-@ 1 apple  staff   1422 Nov  1  1991 block_dev.c
|-rw-rw-rw-@ 1 apple  staff   9072 Dec  7  1991 buffer.c
|-rw-rw-rw-@ 1 apple  staff   2103 Nov 19  1991 char_dev.c
|-rw-rw-rw-@ 1 apple  staff   9134 Dec  2  1991 exec.c
|-rw-rw-rw-@ 1 apple  staff   1455 Oct  2  1991 fcntl.c
|-rw-rw-rw-@ 1 apple  staff   1852 Dec  2  1991 file_dev.c
|-rw-rw-rw-@ 1 apple  staff    122 Oct  2  1991 file_table.c
|-rw-rw-rw-@ 1 apple  staff   6933 Dec  7  1991 inode.c
|-rw-rw-rw-@ 1 apple  staff    977 Nov 19  1991 ioctl.c
|-rw-rw-rw-@ 1 apple  staff  16562 Nov 26  1991 namei.c
|-rw-rw-rw-@ 1 apple  staff   4340 Nov 26  1991 open.c
|-rw-rw-rw-@ 1 apple  staff   2385 Oct 19  1991 pipe.c
|-rw-rw-rw-@ 1 apple  staff   2802 Nov 25  1991 read_write.c
|-rw-rw-rw-@ 1 apple  staff   1175 Oct  2  1991 stat.c
|-rw-rw-rw-@ 1 apple  staff   5628 Dec  7  1991 super.c
|-rw-rw-rw-@ 1 apple  staff   1148 Oct  2  1991 truncate.c

/linux-0.11/include
|-rw-rw-rw-@  1 apple  staff  6047 Sep 17  1991 a.out.h
|-rw-rw-rw-@  1 apple  staff   321 Sep 17  1991 const.h
|-rw-rw-rw-@  1 apple  staff  1049 Nov  8  1991 ctype.h
|-rw-rw-rw-@  1 apple  staff  1268 Sep 17  1991 errno.h
|-rw-rw-rw-@  1 apple  staff  1374 Sep 17  1991 fcntl.h
|-rw-rw-rw-@  1 apple  staff  1762 Sep 23  1991 signal.h
|-rwxrwxrwx@  1 apple  staff   780 Sep 17  1991 stdarg.h
|-rw-rw-rw-@  1 apple  staff   286 Sep 17  1991 stddef.h
|-rw-rw-rw-@  1 apple  staff  7881 Sep 17  1991 string.h
|-rw-rw-rw-@  1 apple  staff  5325 Nov 26  1991 termios.h
|-rw-rw-rw-@  1 apple  staff   734 Sep 17  1991 time.h
|-rw-rw-rw-@  1 apple  staff  6410 Nov 26  1991 unistd.h
|-rw-rw-rw-@  1 apple  staff   225 Sep 17  1991 utime.h
 /linux-0.11/include/asm
  |-rw-rw-rw-@ 1 apple  staff   477 Aug  7  1991 io.h
  |-rw-rw-rw-@ 1 apple  staff   507 Jun 16  1991 memory.h
```

```
  |-rw-rw-rw-@ 1 apple  staff  1366 Nov 26  1991 segment.h
  |-rw-rw-rw-@ 1 apple  staff  1711 Sep 17  1991 system.h
 /linux-0.11/include/linux
  |-rw-rw-rw-@ 1 apple  staff  1289 Dec  9  1991 config.h
  |-rw-rw-rw-@ 1 apple  staff  2466 Nov  2  1991 fdreg.h
  |-rw-rw-rw-@ 1 apple  staff  5474 Dec  2  1991 fs.h
  |-rw-rw-rw-@ 1 apple  staff  1968 Oct 13  1991 hdreg.h
  |-rw-rw-rw-@ 1 apple  staff   304 Jun 20  1991 head.h
  |-rw-rw-rw-@ 1 apple  staff   734 Dec  2  1991 kernel.h
  |-rw-rw-rw-@ 1 apple  staff   219 Jul 30  1991 mm.h
  |-rw-rw-rw-@ 1 apple  staff  5838 Nov 20  1991 sched.h
  |-rw-rw-rw-@ 1 apple  staff  2588 Nov 26  1991 sys.h
  |-rw-rw-rw-@ 1 apple  staff  2173 Sep 21  1991 tty.h
 /linux-0.11/include/sys
  |-rw-rw-rw-@ 1 apple  staff  1304 Sep 17  1991 stat.h
  |-rw-rw-rw-@ 1 apple  staff   200 Sep 17  1991 times.h
  |-rw-rw-rw-@ 1 apple  staff   805 Sep 17  1991 types.h
  |-rw-rw-rw-@ 1 apple  staff   234 Sep 17  1991 utsname.h
  |-rw-rw-rw-@ 1 apple  staff   560 Sep 17  1991 wait.h

/linux-0.11/init
|-rw-rw-rw-@ 1 apple  staff  5221 Dec  8  1991 main.c

/linux-0.11/kernel
|-rw-rw-rw-@ 1 apple  staff  3309 Dec  2  1991 Makefile
|-rw-rw-rw-@ 1 apple  staff  2335 Nov 18  1991 asm.s
|-rw-rw-rw-@ 1 apple  staff  4175 Dec  7  1991 exit.c
|-rw-rw-rw-@ 1 apple  staff  3693 Nov 25  1991 fork.c
|-rw-rw-rw-@ 1 apple  staff  1461 Oct  2  1991 mktime.c
|-rw-rw-rw-@ 1 apple  staff   448 Oct 17  1991 panic.c
|-rw-rw-rw-@ 1 apple  staff   734 Oct  2  1991 printk.c
|-rw-rw-rw-@ 1 apple  staff  8242 Dec  5  1991 sched.c
|-rw-rw-rw-@ 1 apple  staff  2651 Dec  7  1991 signal.c
|-rw-rw-rw-@ 1 apple  staff  3706 Nov 26  1991 sys.c
|-rw-rw-rw-@ 1 apple  staff  5265 Dec  4  1991 system_call.s
|-rw-rw-rw-@ 1 apple  staff  4951 Oct 31  1991 traps.c
|-rw-rw-rw-@ 1 apple  staff  4800 Oct  2  1991 vsprintf.c
 /linux-0.11/kernel/blk_drv
  |-rw-rw-rw-@ 1 apple  staff   1951 Dec  6  1991 Makefile
  |-rw-rw-rw-@ 1 apple  staff   3464 Dec  6  1991 blk.h
  |-rw-rw-rw-@ 1 apple  staff  11429 Dec  7  1991 floppy.c
  |-rw-rw-rw-@ 1 apple  staff   7807 Dec  6  1991 hd.c
  |-rw-rw-rw-@ 1 apple  staff   3539 Dec  4  1991 ll_rw_blk.c
  |-rw-rw-rw-@ 1 apple  staff   2740 Dec  6  1991 ramdisk.c
 /linux-0.11/kernel/chr_drv
  |-rw-rw-rw-@ 1 apple  staff   2443 Dec  2  1991 Makefile
  |-rw-rw-rw-@ 1 apple  staff  14568 Nov 24  1991 console.c
  |-rw-rw-rw-@ 1 apple  staff  12780 Dec  4  1991 keyboard.S
  |-rw-rw-rw-@ 1 apple  staff   2718 Oct  2  1991 rs_io.s
  |-rw-rw-rw-@ 1 apple  staff   1406 Nov 18  1991 serial.c
  |-rw-rw-rw-@ 1 apple  staff   7634 Dec  9  1991 tty_io.c
  |-rw-rw-rw-@ 1 apple  staff   4979 Nov 26  1991 tty_ioctl.c
 /linux-0.11/kernel/math
  |-rw-rw-rw-@ 1 apple  staff   936 Nov 18  1991 Makefile
  |-rw-rw-rw-@ 1 apple  staff  1023 Nov 23  1991 math_emulate.c

/linux-0.11/lib
|-rw-rw-rw-@ 1 apple  staff  2602 Dec  2  1991 Makefile
|-rw-rw-rw-@ 1 apple  staff   198 Oct  2  1991 _exit.c
|-rw-rw-rw-@ 1 apple  staff   131 Oct  2  1991 close.c
|-rw-rw-rw-@ 1 apple  staff  1202 Oct  2  1991 ctype.c
|-rw-rw-rw-@ 1 apple  staff   127 Oct  2  1991 dup.c
|-rw-rw-rw-@ 1 apple  staff    73 Oct  2  1991 errno.c
|-rw-rw-rw-@ 1 apple  staff   170 Oct  2  1991 execve.c
|-rw-rw-rw-@ 1 apple  staff  7469 Dec  2  1991 malloc.c
|-rw-rw-rw-@ 1 apple  staff   389 Oct  2  1991 open.c
|-rw-rw-rw-@ 1 apple  staff   128 Oct  2  1991 setsid.c
|-rw-rw-rw-@ 1 apple  staff   177 Oct  2  1991 string.c
|-rw-rw-rw-@ 1 apple  staff   253 Oct  2  1991 wait.c
```

```
|-rw-rw-rw-@ 1 apple  staff   160 Oct  2  1991 write.c


/linux-0.11/mm
|-rw-rw-rw-@ 1 apple  staff    813 Dec  2  1991 Makefile
|-rw-rw-rw-@ 1 apple  staff  11223 Dec  3  1991 memory.c
|-rw-rw-rw-@ 1 apple  staff    508 Oct  2  1991 page.s


/linux-0.11/tools
|-rw-rw-rw-@ 1 apple  staff  4525 Dec  4  1991 build.c
```

```
#
# if you want the ram-disk device, define this to be the
# size in blocks.
#
RAMDISK = #-DRAMDISK=512

AS86    =as86 -0 -a
LD86    =ld86 -0

AS      =gas
LD      =gld
LDFLAGS =-s -x -M
CC      =gcc $(RAMDISK)
CFLAGS  =-Wall -O -fstrength-reduce -fomit-frame-pointer \
-fcombine-regs -mstring-insns
CPP     =cpp -nostdinc -Iinclude

#
# ROOT_DEV specifies the default root-device when making the image.
# This can be either FLOPPY, /dev/xxxx or empty, in which case the
# default of /dev/hd6 is used by 'build'.
#
ROOT_DEV=/dev/hd6

ARCHIVES=kernel/kernel.o mm/mm.o fs/fs.o
DRIVERS =kernel/blk_drv/blk_drv.a kernel/chr_drv/chr_drv.a
MATH    =kernel/math/math.a
LIBS    =lib/lib.a

.c.s:
        $(CC) $(CFLAGS) \
        -nostdinc -Iinclude -S -o $*.s $<
.s.o:
        $(AS) -c -o $*.o $<
.c.o:
        $(CC) $(CFLAGS) \
        -nostdinc -Iinclude -c -o $*.o $<

all:    Image

Image: boot/bootsect boot/setup tools/system tools/build
        tools/build boot/bootsect boot/setup tools/system $(ROOT_DEV) > Image
        sync

disk: Image
        dd bs=8192 if=Image of=/dev/PS0

tools/build: tools/build.c
        $(CC) $(CFLAGS) \
        -o tools/build tools/build.c

boot/head.o: boot/head.s

tools/system:   boot/head.o init/main.o \
                $(ARCHIVES) $(DRIVERS) $(MATH) $(LIBS)
        $(LD) $(LDFLAGS) boot/head.o init/main.o \
        $(ARCHIVES) \
        $(DRIVERS) \
        $(MATH) \
        $(LIBS) \
        -o tools/system > System.map

kernel/math/math.a:
        (cd kernel/math; make)

kernel/blk_drv/blk_drv.a:
        (cd kernel/blk_drv; make)

kernel/chr_drv/chr_drv.a:
```

```
        (cd kernel/chr_drv; make)

kernel/kernel.o:
        (cd kernel; make)

mm/mm.o:
        (cd mm; make)

fs/fs.o:
        (cd fs; make)

lib/lib.a:
        (cd lib; make)

boot/setup: boot/setup.s
        $(AS86) -o boot/setup.o boot/setup.s
        $(LD86) -s -o boot/setup boot/setup.o

boot/bootsect:  boot/bootsect.s
        $(AS86) -o boot/bootsect.o boot/bootsect.s
        $(LD86) -s -o boot/bootsect boot/bootsect.o

tmp.s:  boot/bootsect.s tools/system
        (echo -n "SYSSIZE = (";ls -l tools/system | grep system \
                | cut -c25-31 | tr '\012' ' '; echo "+ 15 ) / 16") > tmp.s
        cat boot/bootsect.s >> tmp.s

clean:
        rm -f Image System.map tmp_make core boot/bootsect boot/setup
        rm -f init/*.o tools/system tools/build boot/*.o
        (cd mm;make clean)
        (cd fs;make clean)
        (cd kernel;make clean)
        (cd lib;make clean)

backup: clean
        (cd .. ; tar cf - linux | compress - > backup.Z)
        sync

dep:
        sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
        (for i in init/*.c;do echo -n "init/";$(CPP) -M $$i;done) >> tmp_make
        cp tmp_make Makefile
        (cd fs; make dep)
        (cd kernel; make dep)
        (cd mm; make dep)

### Dependencies:
init/main.o : init/main.c include/unistd.h include/sys/stat.h \
  include/sys/types.h include/sys/times.h include/sys/utsname.h \
  include/utime.h include/time.h include/linux/tty.h include/termios.h \
  include/linux/sched.h include/linux/head.h include/linux/fs.h \
  include/linux/mm.h include/signal.h include/asm/system.h include/asm/io.h \
  include/stddef.h include/stdarg.h include/fcntl.h
```

```
!
! SYS_SIZE is the number of clicks (16 bytes) to be loaded.
! 0x3000 is 0x30000 bytes = 196kB, more than enough for current
! versions of linux
!
SYSSIZE = 0x3000
!
!       bootsect.s              (C) 1991 Linus Torvalds
!
! bootsect.s is loaded at 0x7c00 by the bios-startup routines, and moves
! iself out of the way to address 0x90000, and jumps there.
!
! It then loads 'setup' directly after itself (0x90200), and the system
! at 0x10000, using BIOS interrupts.
!
! NOTE! currently system is at most 8*65536 bytes long. This should be no
! problem, even in the future. I want to keep it simple. This 512 kB
! kernel size should be enough, especially as this doesn't contain the
! buffer cache as in minix
!
! The loader has been made as simple as possible, and continuos
! read errors will result in a unbreakable loop. Reboot by hand. It
! loads pretty fast by getting whole sectors at a time whenever possible.

.globl begtext, begdata, begbss, endtext, enddata, endbss
.text
begtext:
.data
begdata:
.bss
begbss:
.text

SETUPLEN = 4                            ! nr of setup-sectors
BOOTSEG  = 0x07c0                       ! original address of boot-sector
INITSEG  = 0x9000                       ! we move boot here - out of the way
SETUPSEG = 0x9020                       ! setup starts here
SYSSEG   = 0x1000                       ! system loaded at 0x10000 (65536).
ENDSEG   = SYSSEG + SYSSIZE             ! where to stop loading

! ROOT_DEV:     0x000 - same type of floppy as boot.
!               0x301 - first partition on first drive etc
ROOT_DEV = 0x306

entry start
start:
        mov     ax,#BOOTSEG
        mov     ds,ax
        mov     ax,#INITSEG
        mov     es,ax
        mov     cx,#256
        sub     si,si
        sub     di,di
        rep
        movw
        jmpi    go,INITSEG
go:     mov     ax,cs
        mov     ds,ax
        mov     es,ax
! put stack at 0x9ff00.
        mov     ss,ax
        mov     sp,#0xFF00              ! arbitrary value >>512

! load the setup-sectors directly after the bootblock.
! Note that 'es' is already set up.

load_setup:
        mov     dx,#0x0000              ! drive 0, head 0
        mov     cx,#0x0002              ! sector 2, track 0
```

```
        mov     bx,#0x0200              ! address = 512, in INITSEG
        mov     ax,#0x0200+SETUPLEN     ! service 2, nr of sectors
        int     0x13                    ! read it
        jnc     ok_load_setup           ! ok - continue
        mov     dx,#0x0000
        mov     ax,#0x0000              ! reset the diskette
        int     0x13
        j       load_setup

ok_load_setup:

! Get disk drive parameters, specifically nr of sectors/track

        mov     dl,#0x00
        mov     ax,#0x0800              ! AH=8 is get drive parameters
        int     0x13
        mov     ch,#0x00
        seg cs
        mov     sectors,cx
        mov     ax,#INITSEG
        mov     es,ax

! Print some inane message

        mov     ah,#0x03                ! read cursor pos
        xor     bh,bh
        int     0x10

        mov     cx,#24
        mov     bx,#0x0007              ! page 0, attribute 7 (normal)
        mov     bp,#msg1
        mov     ax,#0x1301              ! write string, move cursor
        int     0x10

! ok, we've written the message, now
! we want to load the system (at 0x10000)

        mov     ax,#SYSSEG
        mov     es,ax           ! segment of 0x010000
        call    read_it
        call    kill_motor

! After that we check which root-device to use. If the device is
! defined (!= 0), nothing is done and the given device is used.
! Otherwise, either /dev/PS0 (2,28) or /dev/at0 (2,8), depending
! on the number of sectors that the BIOS reports currently.

        seg cs
        mov     ax,root_dev
        cmp     ax,#0
        jne     root_defined
        seg cs
        mov     bx,sectors
        mov     ax,#0x0208              ! /dev/ps0 - 1.2Mb
        cmp     bx,#15
        je      root_defined
        mov     ax,#0x021c              ! /dev/PS0 - 1.44Mb
        cmp     bx,#18
        je      root_defined
undef_root:
        jmp undef_root
root_defined:
        seg cs
        mov     root_dev,ax

! after that (everyting loaded), we jump to
! the setup-routine loaded directly after
! the bootblock:
```

```
        jmpi    0,SETUPSEG

! This routine loads the system at address 0x10000, making sure
! no 64kB boundaries are crossed. We try to load it as fast as
! possible, loading whole tracks whenever we can.
!
! in:   es - starting address segment (normally 0x1000)
!
sread:  .word 1+SETUPLEN        ! sectors read of current track
head:   .word 0                 ! current head
track:  .word 0                 ! current track

read_it:
        mov ax,es
        test ax,#0x0fff
die:    jne die                 ! es must be at 64kB boundary
        xor bx,bx               ! bx is starting address within segment
rp_read:
        mov ax,es
        cmp ax,#ENDSEG          ! have we loaded all yet?
        jb ok1_read
        ret
ok1_read:
        seg cs
        mov ax,sectors
        sub ax,sread
        mov cx,ax
        shl cx,#9
        add cx,bx
        jnc ok2_read
        je ok2_read
        xor ax,ax
        sub ax,bx
        shr ax,#9
ok2_read:
        call read_track
        mov cx,ax
        add ax,sread
        seg cs
        cmp ax,sectors
        jne ok3_read
        mov ax,#1
        sub ax,head
        jne ok4_read
        inc track
ok4_read:
        mov head,ax
        xor ax,ax
ok3_read:
        mov sread,ax
        shl cx,#9
        add bx,cx
        jnc rp_read
        mov ax,es
        add ax,#0x1000
        mov es,ax
        xor bx,bx
        jmp rp_read

read_track:
        push ax
        push bx
        push cx
        push dx
        mov dx,track
        mov cx,sread
        inc cx
        mov ch,dl
        mov dx,head
```

```
        mov dh,dl
        mov dl,#0
        and dx,#0x0100
        mov ah,#2
        int 0x13
        jc bad_rt
        pop dx
        pop cx
        pop bx
        pop ax
        ret
bad_rt: mov ax,#0
        mov dx,#0
        int 0x13
        pop dx
        pop cx
        pop bx
        pop ax
        jmp read_track

/*
 * This procedure turns off the floppy drive motor, so
 * that we enter the kernel in a known state, and
 * don't have to worry about it later.
 */
kill_motor:
        push dx
        mov dx,#0x3f2
        mov al,#0
        outb
        pop dx
        ret

sectors:
        .word 0

msg1:
        .byte 13,10
        .ascii "Loading system ..."
        .byte 13,10,13,10

.org 508
root_dev:
        .word ROOT_DEV
boot_flag:
        .word 0xAA55

.text
endtext:
.data
enddata:
.bss
endbss:
```

```
/*
 *  linux/boot/head.s
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 *  head.s contains the 32-bit startup code.
 *
 * NOTE!!! Startup happens at absolute address 0x00000000, which is also where
 * the page directory will exist. The startup code will be overwritten by
 * the page directory.
 */
.text
.globl _idt,_gdt,_pg_dir,_tmp_floppy_area
_pg_dir:
startup_32:
        movl $0x10,%eax
        mov %ax,%ds
        mov %ax,%es
        mov %ax,%fs
        mov %ax,%gs
        lss _stack_start,%esp
        call setup_idt
        call setup_gdt
        movl $0x10,%eax         # reload all the segment registers
        mov %ax,%ds             # after changing gdt. CS was already
        mov %ax,%es             # reloaded in 'setup_gdt'
        mov %ax,%fs
        mov %ax,%gs
        lss _stack_start,%esp
        xorl %eax,%eax
1:      incl %eax               # check that A20 really IS enabled
        movl %eax,0x000000      # loop forever if it isn't
        cmpl %eax,0x100000
        je 1b
/*
 * NOTE! 486 should set bit 16, to check for write-protect in supervisor
 * mode. Then it would be unnecessary with the "verify_area()"-calls.
 * 486 users probably want to set the NE (#5) bit also, so as to use
 * int 16 for math errors.
 */
        movl %cr0,%eax          # check math chip
        andl $0x80000011,%eax   # Save PG,PE,ET
/* "orl $0x10020,%eax" here for 486 might be good */
        orl $2,%eax             # set MP
        movl %eax,%cr0
        call check_x87
        jmp after_page_tables

/*
 * We depend on ET to be correct. This checks for 287/387.
 */
check_x87:
        fninit
        fstsw %ax
        cmpb $0,%al
        je 1f                   /* no coprocessor: have to set bits */
        movl %cr0,%eax
        xorl $6,%eax            /* reset MP, set EM */
        movl %eax,%cr0
        ret
.align 2
1:      .byte 0xDB,0xE4         /* fsetpm for 287, ignored by 387 */
        ret

/*
 *  setup_idt
 *
```

```
 *  sets up a idt with 256 entries pointing to
 *  ignore_int, interrupt gates. It then loads
 *  idt. Everything that wants to install itself
 *  in the idt-table may do so themselves. Interrupts
 *  are enabled elsewhere, when we can be relatively
 *  sure everything is ok. This routine will be over-
 *  written by the page tables.
 */
setup_idt:
        lea ignore_int,%edx
        movl $0x00080000,%eax
        movw %dx,%ax            /* selector = 0x0008 = cs */
        movw $0x8E00,%dx        /* interrupt gate - dpl=0, present */

        lea _idt,%edi
        mov $256,%ecx
rp_sidt:
        movl %eax,(%edi)
        movl %edx,4(%edi)
        addl $8,%edi
        dec %ecx
        jne rp_sidt
        lidt idt_descr
        ret

/*
 *  setup_gdt
 *
 *  This routines sets up a new gdt and loads it.
 *  Only two entries are currently built, the same
 *  ones that were built in init.s. The routine
 *  is VERY complicated at two whole lines, so this
 *  rather long comment is certainly needed :-).
 *  This routine will beoverwritten by the page tables.
 */
setup_gdt:
        lgdt gdt_descr
        ret

/*
 * I put the kernel page tables right after the page directory,
 * using 4 of them to span 16 Mb of physical memory. People with
 * more than 16MB will have to expand this.
 */
.org 0x1000
pg0:

.org 0x2000
pg1:

.org 0x3000
pg2:

.org 0x4000
pg3:

.org 0x5000
/*
 * tmp_floppy_area is used by the floppy-driver when DMA cannot
 * reach to a buffer-block. It needs to be aligned, so that it isn't
 * on a 64kB border.
 */
_tmp_floppy_area:
        .fill 1024,1,0

after_page_tables:
        pushl $0                # These are the parameters to main :-)
        pushl $0
        pushl $0
```

```
        pushl $L6               # return address for main, if it decides to.
        pushl $_main
        jmp setup_paging
L6:
        jmp L6                  # main should never return here, but
                                # just in case, we know what happens.

/* This is the default interrupt "handler" :-) */
int_msg:
        .asciz "Unknown interrupt\n\r"
.align 2
ignore_int:
        pushl %eax
        pushl %ecx
        pushl %edx
        push %ds
        push %es
        push %fs
        movl $0x10,%eax
        mov %ax,%ds
        mov %ax,%es
        mov %ax,%fs
        pushl $int_msg
        call _printk
        popl %eax
        pop %fs
        pop %es
        pop %ds
        popl %edx
        popl %ecx
        popl %eax
        iret


/*
 * Setup_paging
 *
 * This routine sets up paging by setting the page bit
 * in cr0. The page tables are set up, identity-mapping
 * the first 16MB. The pager assumes that no illegal
 * addresses are produced (ie >4Mb on a 4Mb machine).
 *
 * NOTE! Although all physical memory should be identity
 * mapped by this routine, only the kernel page functions
 * use the >1Mb addresses directly. All "normal" functions
 * use just the lower 1Mb, or the local data space, which
 * will be mapped to some other place - mm keeps track of
 * that.
 *
 * For those with more memory than 16 Mb - tough luck. I've
 * not got it, why should you :-) The source is here. Change
 * it. (Seriously - it shouldn't be too difficult. Mostly
 * change some constants etc. I left it at 16Mb, as my machine
 * even cannot be extended past that (ok, but it was cheap :-)
 * I've tried to show which constants to change by having
 * some kind of marker at them (search for "16Mb"), but I
 * won't guarantee that's all :-( )
 */
.align 2
setup_paging:
        movl $1024*5,%ecx               /* 5 pages - pg_dir+4 page tables */
        xorl %eax,%eax
        xorl %edi,%edi                  /* pg_dir is at 0x000 */
        cld;rep;stosl
        movl $pg0+7,_pg_dir             /* set present bit/user r/w */
        movl $pg1+7,_pg_dir+4           /*  --------- " " --------- */
        movl $pg2+7,_pg_dir+8           /*  --------- " " --------- */
        movl $pg3+7,_pg_dir+12          /*  --------- " " --------- */
        movl $pg3+4092,%edi
```

```
        movl $0xfff007,%eax             /*  16Mb - 4096 + 7 (r/w user,p) */
        std
1:      stosl                   /* fill pages backwards - more efficient :-) */
        subl $0x1000,%eax
        jge 1b
        xorl %eax,%eax          /* pg_dir is at 0x0000 */
        movl %eax,%cr3          /* cr3 - page directory start */
        movl %cr0,%eax
        orl $0x80000000,%eax
        movl %eax,%cr0          /* set paging (PG) bit */
        ret                     /* this also flushes prefetch-queue */

.align 2
.word 0
idt_descr:
        .word 256*8-1           # idt contains 256 entries
        .long _idt
.align 2
.word 0
gdt_descr:
        .word 256*8-1           # so does gdt (not that that's any
        .long _gdt              # magic number, but it works for me :^)

        .align 3
_idt:   .fill 256,8,0           # idt is uninitialized

_gdt:   .quad 0x0000000000000000        /* NULL descriptor */
        .quad 0x00c09a0000000fff        /* 16Mb */
        .quad 0x00c0920000000fff        /* 16Mb */
        .quad 0x0000000000000000        /* TEMPORARY - don't use */
        .fill 252,8,0                   /* space for LDT's and TSS's etc */
```

```
!
!       setup.s         (C) 1991 Linus Torvalds
!
! setup.s is responsible for getting the system data from the BIOS,
! and putting them into the appropriate places in system memory.
! both setup.s and system has been loaded by the bootblock.
!
! This code asks the bios for memory/disk/other parameters, and
! puts them in a "safe" place: 0x90000-0x901FF, ie where the
! boot-block used to be. It is then up to the protected mode
! system to read them from there before the area is overwritten
! for buffer-blocks.
!

! NOTE! These had better be the same as in bootsect.s!

INITSEG  = 0x9000       ! we move boot here - out of the way
SYSSEG   = 0x1000       ! system loaded at 0x10000 (65536).
SETUPSEG = 0x9020       ! this is the current segment

.globl begtext, begdata, begbss, endtext, enddata, endbss
.text
begtext:
.data
begdata:
.bss
begbss:
.text

entry start
start:

! ok, the read went well so we get current cursor position and save it for
! posterity.

        mov     ax,#INITSEG     ! this is done in bootsect already, but...
        mov     ds,ax
        mov     ah,#0x03        ! read cursor pos
        xor     bh,bh
        int     0x10            ! save it in known place, con_init fetches
        mov     [0],dx          ! it from 0x90000.

! Get memory size (extended mem, kB)

        mov     ah,#0x88
        int     0x15
        mov     [2],ax

! Get video-card data:

        mov     ah,#0x0f
        int     0x10
        mov     [4],bx          ! bh = display page
        mov     [6],ax          ! al = video mode, ah = window width

! check for EGA/VGA and some config parameters

        mov     ah,#0x12
        mov     bl,#0x10
        int     0x10
        mov     [8],ax
        mov     [10],bx
        mov     [12],cx

! Get hd0 data

        mov     ax,#0x0000
        mov     ds,ax
        lds     si,[4*0x41]
```

```
        mov     ax,#INITSEG
        mov     es,ax
        mov     di,#0x0080
        mov     cx,#0x10
        rep
        movsb

! Get hd1 data

        mov     ax,#0x0000
        mov     ds,ax
        lds     si,[4*0x46]
        mov     ax,#INITSEG
        mov     es,ax
        mov     di,#0x0090
        mov     cx,#0x10
        rep
        movsb

! Check that there IS a hd1 :-)

        mov     ax,#0x01500
        mov     dl,#0x81
        int     0x13
        jc      no_disk1
        cmp     ah,#3
        je      is_disk1
no_disk1:
        mov     ax,#INITSEG
        mov     es,ax
        mov     di,#0x0090
        mov     cx,#0x10
        mov     ax,#0x00
        rep
        stosb
is_disk1:

! now we want to move to protected mode ...

        cli                     ! no interrupts allowed !

! first we move the system to it's rightful place

        mov     ax,#0x0000
        cld                     ! 'direction'=0, movs moves forward
do_move:
        mov     es,ax           ! destination segment
        add     ax,#0x1000
        cmp     ax,#0x9000
        jz      end_move
        mov     ds,ax           ! source segment
        sub     di,di
        sub     si,si
        mov     cx,#0x8000
        rep
        movsw
        jmp     do_move

! then we load the segment descriptors

end_move:
        mov     ax,#SETUPSEG    ! right, forgot this at first. didn't work :-)
        mov     ds,ax
        lidt    idt_48          ! load idt with 0,0
        lgdt    gdt_48          ! load gdt with whatever appropriate

! that was painless, now we enable A20

        call    empty_8042
```

```
        mov     al,#0xD1                ! command write
        out     #0x64,al
        call    empty_8042
        mov     al,#0xDF                ! A20 on
        out     #0x60,al
        call    empty_8042

! well, that went ok, I hope. Now we have to reprogram the interrupts :-(
! we put them right after the intel-reserved hardware interrupts, at
! int 0x20-0x2F. There they won't mess up anything. Sadly IBM really
! messed this up with the original PC, and they haven't been able to
! rectify it afterwards. Thus the bios puts interrupts at 0x08-0x0f,
! which is used for the internal hardware interrupts as well. We just
! have to reprogram the 8259's, and it isn't fun.

        mov     al,#0x11                ! initialization sequence
        out     #0x20,al                ! send it to 8259A-1
        .word   0x00eb,0x00eb           ! jmp $+2, jmp $+2
        out     #0xA0,al                ! and to 8259A-2
        .word   0x00eb,0x00eb
        mov     al,#0x20                ! start of hardware int's (0x20)
        out     #0x21,al
        .word   0x00eb,0x00eb
        mov     al,#0x28                ! start of hardware int's 2 (0x28)
        out     #0xA1,al
        .word   0x00eb,0x00eb
        mov     al,#0x04                ! 8259-1 is master
        out     #0x21,al
        .word   0x00eb,0x00eb
        mov     al,#0x02                ! 8259-2 is slave
        out     #0xA1,al
        .word   0x00eb,0x00eb
        mov     al,#0x01                ! 8086 mode for both
        out     #0x21,al
        .word   0x00eb,0x00eb
        out     #0xA1,al
        .word   0x00eb,0x00eb
        mov     al,#0xFF                ! mask off all interrupts for now
        out     #0x21,al
        .word   0x00eb,0x00eb
        out     #0xA1,al

! well, that certainly wasn't fun :-(. Hopefully it works, and we don't
! need no steenking BIOS anyway (except for the initial loading :-).
! The BIOS-routine wants lots of unnecessary data, and it's less
! "interesting" anyway. This is how REAL programmers do it.
!
! Well, now's the time to actually move into protected mode. To make
! things as simple as possible, we do no register set-up or anything,
! we let the gnu-compiled 32-bit programs do that. We just jump to
! absolute address 0x00000, in 32-bit protected mode.

        mov     ax,#0x0001      ! protected mode (PE) bit
        lmsw    ax              ! This is it!
        jmpi    0,8             ! jmp offset 0 of segment 8 (cs)

! This routine checks that the keyboard command queue is empty
! No timeout is used - if this hangs there is something wrong with
! the machine, and we probably couldn't proceed anyway.
empty_8042:
        .word   0x00eb,0x00eb
        in      al,#0x64        ! 8042 status port
        test    al,#2           ! is input buffer full?
        jnz     empty_8042      ! yes - loop
        ret

gdt:
        .word   0,0,0,0         ! dummy
```

```
        .word   0x07FF          ! 8Mb - limit=2047 (2048*4096=8Mb)
        .word   0x0000          ! base address=0
        .word   0x9A00          ! code read/exec
        .word   0x00C0          ! granularity=4096, 386

        .word   0x07FF          ! 8Mb - limit=2047 (2048*4096=8Mb)
        .word   0x0000          ! base address=0
        .word   0x9200          ! data read/write
        .word   0x00C0          ! granularity=4096, 386

idt_48:
        .word   0                       ! idt limit=0
        .word   0,0                     ! idt base=0L

gdt_48:
        .word   0x800           ! gdt limit=2048, 256 GDT entries
        .word   512+gdt,0x9     ! gdt base = 0X9xxxx

.text
endtext:
.data
enddata:
.bss
endbss:
```

```
AR      =gar
AS      =gas
CC      =gcc
LD      =gld
CFLAGS  =-Wall -O -fstrength-reduce -fcombine-regs -fomit-frame-pointer \
        -mstring-insns -nostdinc -I../include
CPP     =gcc -E -nostdinc -I../include

.c.s:
        $(CC) $(CFLAGS) \
        -S -o $*.s $<
.c.o:
        $(CC) $(CFLAGS) \
        -c -o $*.o $<
.s.o:
        $(AS) -o $*.o $<

OBJS=   open.o read_write.o inode.o file_table.o buffer.o super.o \
        block_dev.o char_dev.o file_dev.o stat.o exec.o pipe.o namei.o \
        bitmap.o fcntl.o ioctl.o truncate.o

fs.o: $(OBJS)
        $(LD) -r -o fs.o $(OBJS)

clean:
        rm -f core *.o *.a tmp_make
        for i in *.c;do rm -f `basename $$i .c`.s;done

dep:
        sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
        (for i in *.c;do $(CPP) -M $$i;done) >> tmp_make
        cp tmp_make Makefile

### Dependencies:
bitmap.o : bitmap.c ../include/string.h ../include/linux/sched.h \
  ../include/linux/head.h ../include/linux/fs.h ../include/sys/types.h \
  ../include/linux/mm.h ../include/signal.h ../include/linux/kernel.h
block_dev.o : block_dev.c ../include/errno.h ../include/linux/sched.h \
  ../include/linux/head.h ../include/linux/fs.h ../include/sys/types.h \
  ../include/linux/mm.h ../include/signal.h ../include/linux/kernel.h \
  ../include/asm/segment.h ../include/asm/system.h
buffer.o : buffer.c ../include/stdarg.h ../include/linux/config.h \
  ../include/linux/sched.h ../include/linux/head.h ../include/linux/fs.h \
  ../include/sys/types.h ../include/linux/mm.h ../include/signal.h \
  ../include/linux/kernel.h ../include/asm/system.h ../include/asm/io.h
char_dev.o : char_dev.c ../include/errno.h ../include/sys/types.h \
  ../include/linux/sched.h ../include/linux/head.h ../include/linux/fs.h \
  ../include/linux/mm.h ../include/signal.h ../include/linux/kernel.h \
  ../include/asm/segment.h ../include/asm/io.h
exec.o : exec.c ../include/errno.h ../include/string.h \
  ../include/sys/stat.h ../include/sys/types.h ../include/a.out.h \
  ../include/linux/fs.h ../include/linux/sched.h ../include/linux/head.h \
  ../include/linux/mm.h ../include/signal.h ../include/linux/kernel.h \
  ../include/asm/segment.h
fcntl.o : fcntl.c ../include/string.h ../include/errno.h \
  ../include/linux/sched.h ../include/linux/head.h ../include/linux/fs.h \
  ../include/sys/types.h ../include/linux/mm.h ../include/signal.h \
  ../include/linux/kernel.h ../include/asm/segment.h ../include/fcntl.h \
  ../include/sys/stat.h
file_dev.o : file_dev.c ../include/errno.h ../include/fcntl.h \
  ../include/sys/types.h ../include/linux/sched.h ../include/linux/head.h \
  ../include/linux/fs.h ../include/linux/mm.h ../include/signal.h \
  ../include/linux/kernel.h ../include/asm/segment.h
file_table.o : file_table.c ../include/linux/fs.h ../include/sys/types.h
inode.o : inode.c ../include/string.h ../include/sys/stat.h \
  ../include/sys/types.h ../include/linux/sched.h ../include/linux/head.h \
  ../include/linux/fs.h ../include/linux/mm.h ../include/signal.h \
  ../include/linux/kernel.h ../include/asm/system.h
ioctl.o : ioctl.c ../include/string.h ../include/errno.h \
```

```
  ../include/sys/stat.h ../include/sys/types.h ../include/linux/sched.h \
  ../include/linux/head.h ../include/linux/fs.h ../include/linux/mm.h \
  ../include/signal.h
namei.o : namei.c ../include/linux/sched.h ../include/linux/head.h \
  ../include/linux/fs.h ../include/sys/types.h ../include/linux/mm.h \
  ../include/signal.h ../include/linux/kernel.h ../include/asm/segment.h \
  ../include/string.h ../include/fcntl.h ../include/errno.h \
  ../include/const.h ../include/sys/stat.h
open.o : open.c ../include/string.h ../include/errno.h ../include/fcntl.h \
  ../include/sys/types.h ../include/utime.h ../include/sys/stat.h \
  ../include/linux/sched.h ../include/linux/head.h ../include/linux/fs.h \
  ../include/linux/mm.h ../include/signal.h ../include/linux/tty.h \
  ../include/termios.h ../include/linux/kernel.h ../include/asm/segment.h
pipe.o : pipe.c ../include/signal.h ../include/sys/types.h \
  ../include/linux/sched.h ../include/linux/head.h ../include/linux/fs.h \
  ../include/linux/mm.h ../include/asm/segment.h
read_write.o : read_write.c ../include/sys/stat.h ../include/sys/types.h \
  ../include/errno.h ../include/linux/kernel.h ../include/linux/sched.h \
  ../include/linux/head.h ../include/linux/fs.h ../include/linux/mm.h \
  ../include/signal.h ../include/asm/segment.h
stat.o : stat.c ../include/errno.h ../include/sys/stat.h \
  ../include/sys/types.h ../include/linux/fs.h ../include/linux/sched.h \
  ../include/linux/head.h ../include/linux/mm.h ../include/signal.h \
  ../include/linux/kernel.h ../include/asm/segment.h
super.o : super.c ../include/linux/config.h ../include/linux/sched.h \
  ../include/linux/head.h ../include/linux/fs.h ../include/sys/types.h \
  ../include/linux/mm.h ../include/signal.h ../include/linux/kernel.h \
  ../include/asm/system.h ../include/errno.h ../include/sys/stat.h
truncate.o : truncate.c ../include/linux/sched.h ../include/linux/head.h \
  ../include/linux/fs.h ../include/sys/types.h ../include/linux/mm.h \
  ../include/signal.h ../include/sys/stat.h
```

```
/*
 *  linux/fs/bitmap.c
 *
 *  (C) 1991  Linus Torvalds
 */

/* bitmap.c contains the code that handles the inode and block bitmaps */
#include <string.h>

#include <linux/sched.h>
#include <linux/kernel.h>

#define clear_block(addr) \
__asm__("cld\n\t" \
        "rep\n\t" \
        "stosl" \
        ::"a" (0),"c" (BLOCK_SIZE/4),"D" ((long) (addr)):"cx","di")

#define set_bit(nr,addr) ({\
register int res __asm__("ax"); \
__asm__ __volatile__("btsl %2,%3\n\tsetb %%al": \
"=a" (res):"0" (0),"r" (nr),"m" (*(addr))); \
res;})

#define clear_bit(nr,addr) ({\
register int res __asm__("ax"); \
__asm__ __volatile__("btrl %2,%3\n\tsetnb %%al": \
"=a" (res):"0" (0),"r" (nr),"m" (*(addr))); \
res;})

#define find_first_zero(addr) ({ \
int __res; \
__asm__("cld\n" \
        "1:\tlodsl\n\t" \
        "notl %%eax\n\t" \
        "bsfl %%eax,%%edx\n\t" \
        "je 2f\n\t" \
        "addl %%edx,%%ecx\n\t" \
        "jmp 3f\n" \
        "2:\taddl $32,%%ecx\n\t" \
        "cmpl $8192,%%ecx\n\t" \
        "jl 1b\n" \
        "3:" \
        :"=c" (__res):"c" (0),"S" (addr):"ax","dx","si"); \
__res;})

void free_block(int dev, int block)
{
        struct super_block * sb;
        struct buffer_head * bh;

        if (!(sb = get_super(dev)))
                panic("trying to free block on nonexistent device");
        if (block < sb->s_firstdatazone || block >= sb->s_nzones)
                panic("trying to free block not in datazone");
        bh = get_hash_table(dev,block);
        if (bh) {
                if (bh->b_count != 1) {
                        printk("trying to free block (%04x:%d), count=%d\n",
                                dev,block,bh->b_count);
                        return;
                }
                bh->b_dirt=0;
                bh->b_uptodate=0;
                brelse(bh);
        }
        block -= sb->s_firstdatazone - 1 ;
        if (clear_bit(block&8191,sb->s_zmap[block/8192]->b_data)) {
                printk("block (%04x:%d) ",dev,block+sb->s_firstdatazone-1);
```

```
                panic("free_block: bit already cleared");
        }
        sb->s_zmap[block/8192]->b_dirt = 1;
}

int new_block(int dev)
{
        struct buffer_head * bh;
        struct super_block * sb;
        int i,j;

        if (!(sb = get_super(dev)))
                panic("trying to get new block from nonexistant device");
        j = 8192;
        for (i=0 ; i<8 ; i++)
                if (bh=sb->s_zmap[i])
                        if ((j=find_first_zero(bh->b_data))<8192)
                                break;
        if (i>=8 || !bh || j>=8192)
                return 0;
        if (set_bit(j,bh->b_data))
                panic("new_block: bit already set");
        bh->b_dirt = 1;
        j += i*8192 + sb->s_firstdatazone-1;
        if (j >= sb->s_nzones)
                return 0;
        if (!(bh=getblk(dev,j)))
                panic("new_block: cannot get block");
        if (bh->b_count != 1)
                panic("new block: count is != 1");
        clear_block(bh->b_data);
        bh->b_uptodate = 1;
        bh->b_dirt = 1;
        brelse(bh);
        return j;
}

void free_inode(struct m_inode * inode)
{
        struct super_block * sb;
        struct buffer_head * bh;

        if (!inode)
                return;
        if (!inode->i_dev) {
                memset(inode,0,sizeof(*inode));
                return;
        }
        if (inode->i_count>1) {
                printk("trying to free inode with count=%d\n",inode->i_count);
                panic("free_inode");
        }
        if (inode->i_nlinks)
                panic("trying to free inode with links");
        if (!(sb = get_super(inode->i_dev)))
                panic("trying to free inode on nonexistent device");
        if (inode->i_num < 1 || inode->i_num > sb->s_ninodes)
                panic("trying to free inode 0 or nonexistant inode");
        if (!(bh=sb->s_imap[inode->i_num>>13]))
                panic("nonexistent imap in superblock");
        if (clear_bit(inode->i_num&8191,bh->b_data))
                printk("free_inode: bit already cleared.\n\r");
        bh->b_dirt = 1;
        memset(inode,0,sizeof(*inode));
}

struct m_inode * new_inode(int dev)
{
        struct m_inode * inode;
```

```
        struct super_block * sb;
        struct buffer_head * bh;
        int i,j;

        if (!(inode=get_empty_inode()))
                return NULL;
        if (!(sb = get_super(dev)))
                panic("new_inode with unknown device");
        j = 8192;
        for (i=0 ; i<8 ; i++)
                if (bh=sb->s_imap[i])
                        if ((j=find_first_zero(bh->b_data))<8192)
                                break;
        if (!bh || j >= 8192 || j+i*8192 > sb->s_ninodes) {
                iput(inode);
                return NULL;
        }
        if (set_bit(j,bh->b_data))
                panic("new_inode: bit already set");
        bh->b_dirt = 1;
        inode->i_count=1;
        inode->i_nlinks=1;
        inode->i_dev=dev;
        inode->i_uid=current->euid;
        inode->i_gid=current->egid;
        inode->i_dirt=1;
        inode->i_num = j + i*8192;
        inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
        return inode;
}
```

```
/*
 *  linux/fs/block_dev.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <errno.h>

#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/segment.h>
#include <asm/system.h>

int block_write(int dev, long * pos, char * buf, int count)
{
        int block = *pos >> BLOCK_SIZE_BITS;
        int offset = *pos & (BLOCK_SIZE-1);
        int chars;
        int written = 0;
        struct buffer_head * bh;
        register char * p;

        while (count>0) {
                chars = BLOCK_SIZE - offset;
                if (chars > count)
                        chars=count;
                if (chars == BLOCK_SIZE)
                        bh = getblk(dev,block);
                else
                        bh = breada(dev,block,block+1,block+2,-1);
                block++;
                if (!bh)
                        return written?written:-EIO;
                p = offset + bh->b_data;
                offset = 0;
                *pos += chars;
                written += chars;
                count -= chars;
                while (chars-->0)
                        *(p++) = get_fs_byte(buf++);
                bh->b_dirt = 1;
                brelse(bh);
        }
        return written;
}

int block_read(int dev, unsigned long * pos, char * buf, int count)
{
        int block = *pos >> BLOCK_SIZE_BITS;
        int offset = *pos & (BLOCK_SIZE-1);
        int chars;
        int read = 0;
        struct buffer_head * bh;
        register char * p;

        while (count>0) {
                chars = BLOCK_SIZE-offset;
                if (chars > count)
                        chars = count;
                if (!(bh = breada(dev,block,block+1,block+2,-1)))
                        return read?read:-EIO;
                block++;
                p = offset + bh->b_data;
                offset = 0;
                *pos += chars;
                read += chars;
                count -= chars;
                while (chars-->0)
                        put_fs_byte(*(p++),buf++);
```

```
                brelse(bh);
        }
        return read;
}
```

```
/*
 *  linux/fs/buffer.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 *  'buffer.c' implements the buffer-cache functions. Race-conditions have
 * been avoided by NEVER letting a interrupt change a buffer (except for the
 * data, of course), but instead letting the caller do it. NOTE! As interrupts
 * can wake up a caller, some cli-sti sequences are needed to check for
 * sleep-on-calls. These should be extremely quick, though (I hope).
 */

/*
 * NOTE! There is one discordant note here: checking floppies for
 * disk change. This is where it fits best, I think, as it should
 * invalidate changed floppy-disk-caches.
 */

#include <stdarg.h>

#include <linux/config.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/system.h>
#include <asm/io.h>

extern int end;
struct buffer_head * start_buffer = (struct buffer_head *) &end;
struct buffer_head * hash_table[NR_HASH];
static struct buffer_head * free_list;
static struct task_struct * buffer_wait = NULL;
int NR_BUFFERS = 0;

static inline void wait_on_buffer(struct buffer_head * bh)
{
        cli();
        while (bh->b_lock)
                sleep_on(&bh->b_wait);
        sti();
}

int sys_sync(void)
{
        int i;
        struct buffer_head * bh;

        sync_inodes();          /* write out inodes into buffers */
        bh = start_buffer;
        for (i=0 ; i<NR_BUFFERS ; i++,bh++) {
                wait_on_buffer(bh);
                if (bh->b_dirt)
                        ll_rw_block(WRITE,bh);
        }
        return 0;
}

int sync_dev(int dev)
{
        int i;
        struct buffer_head * bh;

        bh = start_buffer;
        for (i=0 ; i<NR_BUFFERS ; i++,bh++) {
                if (bh->b_dev != dev)
                        continue;
                wait_on_buffer(bh);
                if (bh->b_dev == dev && bh->b_dirt)
```

```
                        ll_rw_block(WRITE,bh);
        }
        sync_inodes();
        bh = start_buffer;
        for (i=0 ; i<NR_BUFFERS ; i++,bh++) {
                if (bh->b_dev != dev)
                        continue;
                wait_on_buffer(bh);
                if (bh->b_dev == dev && bh->b_dirt)
                        ll_rw_block(WRITE,bh);
        }
        return 0;
}

void inline invalidate_buffers(int dev)
{
        int i;
        struct buffer_head * bh;

        bh = start_buffer;
        for (i=0 ; i<NR_BUFFERS ; i++,bh++) {
                if (bh->b_dev != dev)
                        continue;
                wait_on_buffer(bh);
                if (bh->b_dev == dev)
                        bh->b_uptodate = bh->b_dirt = 0;
        }
}

/*
 * This routine checks whether a floppy has been changed, and
 * invalidates all buffer-cache-entries in that case. This
 * is a relatively slow routine, so we have to try to minimize using
 * it. Thus it is called only upon a 'mount' or 'open'. This
 * is the best way of combining speed and utility, I think.
 * People changing diskettes in the middle of an operation deserve
 * to loose :-)
 *
 * NOTE! Although currently this is only for floppies, the idea is
 * that any additional removable block-device will use this routine,
 * and that mount/open needn't know that floppies/whatever are
 * special.
 */
void check_disk_change(int dev)
{
        int i;

        if (MAJOR(dev) != 2)
                return;
        if (!floppy_change(dev & 0x03))
                return;
        for (i=0 ; i<NR_SUPER ; i++)
                if (super_block[i].s_dev == dev)
                        put_super(super_block[i].s_dev);
        invalidate_inodes(dev);
        invalidate_buffers(dev);
}

#define _hashfn(dev,block) (((unsigned)(dev^block))%NR_HASH)
#define hash(dev,block) hash_table[_hashfn(dev,block)]

static inline void remove_from_queues(struct buffer_head * bh)
{
/* remove from hash-queue */
        if (bh->b_next)
                bh->b_next->b_prev = bh->b_prev;
        if (bh->b_prev)
                bh->b_prev->b_next = bh->b_next;
        if (hash(bh->b_dev,bh->b_blocknr) == bh)
```

```
                hash(bh->b_dev,bh->b_blocknr) = bh->b_next;
/* remove from free list */
        if (!(bh->b_prev_free) || !(bh->b_next_free))
                panic("Free block list corrupted");
        bh->b_prev_free->b_next_free = bh->b_next_free;
        bh->b_next_free->b_prev_free = bh->b_prev_free;
        if (free_list == bh)
                free_list = bh->b_next_free;
}

static inline void insert_into_queues(struct buffer_head * bh)
{
/* put at end of free list */
        bh->b_next_free = free_list;
        bh->b_prev_free = free_list->b_prev_free;
        free_list->b_prev_free->b_next_free = bh;
        free_list->b_prev_free = bh;
/* put the buffer in new hash-queue if it has a device */
        bh->b_prev = NULL;
        bh->b_next = NULL;
        if (!bh->b_dev)
                return;
        bh->b_next = hash(bh->b_dev,bh->b_blocknr);
        hash(bh->b_dev,bh->b_blocknr) = bh;
        bh->b_next->b_prev = bh;
}

static struct buffer_head * find_buffer(int dev, int block)
{
        struct buffer_head * tmp;

        for (tmp = hash(dev,block) ; tmp != NULL ; tmp = tmp->b_next)
                if (tmp->b_dev==dev && tmp->b_blocknr==block)
                        return tmp;
        return NULL;
}

/*
 * Why like this, I hear you say... The reason is race-conditions.
 * As we don't lock buffers (unless we are readint them, that is),
 * something might happen to it while we sleep (ie a read-error
 * will force it bad). This shouldn't really happen currently, but
 * the code is ready.
 */
struct buffer_head * get_hash_table(int dev, int block)
{
        struct buffer_head * bh;

        for (;;) {
                if (!(bh=find_buffer(dev,block)))
                        return NULL;
                bh->b_count++;
                wait_on_buffer(bh);
                if (bh->b_dev == dev && bh->b_blocknr == block)
                        return bh;
                bh->b_count--;
        }
}

/*
 * Ok, this is getblk, and it isn't very clear, again to hinder
 * race-conditions. Most of the code is seldom used, (ie repeating),
 * so it should be much more efficient than it looks.
 *
 * The algoritm is changed: hopefully better, and an elusive bug removed.
 */
#define BADNESS(bh) (((bh)->b_dirt<<1)+(bh)->b_lock)
struct buffer_head * getblk(int dev,int block)
{
```

```
        struct buffer_head * tmp, * bh;

repeat:
        if (bh = get_hash_table(dev,block))
                return bh;
        tmp = free_list;
        do {
                if (tmp->b_count)
                        continue;
                if (!bh || BADNESS(tmp)<BADNESS(bh)) {
                        bh = tmp;
                        if (!BADNESS(tmp))
                                break;
                }
/* and repeat until we find something good */
        } while ((tmp = tmp->b_next_free) != free_list);
        if (!bh) {
                sleep_on(&buffer_wait);
                goto repeat;
        }
        wait_on_buffer(bh);
        if (bh->b_count)
                goto repeat;
        while (bh->b_dirt) {
                sync_dev(bh->b_dev);
                wait_on_buffer(bh);
                if (bh->b_count)
                        goto repeat;
        }
/* NOTE!! While we slept waiting for this block, somebody else might */
/* already have added "this" block to the cache. check it */
        if (find_buffer(dev,block))
                goto repeat;
/* OK, FINALLY we know that this buffer is the only one of it's kind, */
/* and that it's unused (b_count=0), unlocked (b_lock=0), and clean */
        bh->b_count=1;
        bh->b_dirt=0;
        bh->b_uptodate=0;
        remove_from_queues(bh);
        bh->b_dev=dev;
        bh->b_blocknr=block;
        insert_into_queues(bh);
        return bh;
}

void brelse(struct buffer_head * buf)
{
        if (!buf)
                return;
        wait_on_buffer(buf);
        if (!(buf->b_count--))
                panic("Trying to free free buffer");
        wake_up(&buffer_wait);
}

/*
 * bread() reads a specified block and returns the buffer that contains
 * it. It returns NULL if the block was unreadable.
 */
struct buffer_head * bread(int dev,int block)
{
        struct buffer_head * bh;

        if (!(bh=getblk(dev,block)))
                panic("bread: getblk returned NULL\n");
        if (bh->b_uptodate)
                return bh;
        ll_rw_block(READ,bh);
        wait_on_buffer(bh);
```

```
        if (bh->b_uptodate)
                return bh;
        brelse(bh);
        return NULL;
}

#define COPYBLK(from,to) \
__asm__("cld\n\t" \
        "rep\n\t" \
        "movsl\n\t" \
        ::"c" (BLOCK_SIZE/4),"S" (from),"D" (to) \
        :"cx","di","si")

/*
 * bread_page reads four buffers into memory at the desired address. It's
 * a function of its own, as there is some speed to be got by reading them
 * all at the same time, not waiting for one to be read, and then another
 * etc.
 */
void bread_page(unsigned long address,int dev,int b[4])
{
        struct buffer_head * bh[4];
        int i;

        for (i=0 ; i<4 ; i++)
                if (b[i]) {
                        if (bh[i] = getblk(dev,b[i]))
                                if (!bh[i]->b_uptodate)
                                        ll_rw_block(READ,bh[i]);
                } else
                        bh[i] = NULL;
        for (i=0 ; i<4 ; i++,address += BLOCK_SIZE)
                if (bh[i]) {
                        wait_on_buffer(bh[i]);
                        if (bh[i]->b_uptodate)
                                COPYBLK((unsigned long) bh[i]->b_data,address);
                        brelse(bh[i]);
                }
}

/*
 * Ok, breada can be used as bread, but additionally to mark other
 * blocks for reading as well. End the argument list with a negative
 * number.
 */
struct buffer_head * breada(int dev,int first, ...)
{
        va_list args;
        struct buffer_head * bh, *tmp;

        va_start(args,first);
        if (!(bh=getblk(dev,first)))
                panic("bread: getblk returned NULL\n");
        if (!bh->b_uptodate)
                ll_rw_block(READ,bh);
        while ((first=va_arg(args,int))>=0) {
                tmp=getblk(dev,first);
                if (tmp) {
                        if (!tmp->b_uptodate)
                                ll_rw_block(READA,bh);
                        tmp->b_count--;
                }
        }
        va_end(args);
        wait_on_buffer(bh);
        if (bh->b_uptodate)
                return bh;
        brelse(bh);
        return (NULL);
```

```
}

void buffer_init(long buffer_end)
{
        struct buffer_head * h = start_buffer;
        void * b;
        int i;

        if (buffer_end == 1<<20)
                b = (void *) (640*1024);
        else
                b = (void *) buffer_end;
        while ( (b -= BLOCK_SIZE) >= ((void *) (h+1)) ) {
                h->b_dev = 0;
                h->b_dirt = 0;
                h->b_count = 0;
                h->b_lock = 0;
                h->b_uptodate = 0;
                h->b_wait = NULL;
                h->b_next = NULL;
                h->b_prev = NULL;
                h->b_data = (char *) b;
                h->b_prev_free = h-1;
                h->b_next_free = h+1;
                h++;
                NR_BUFFERS++;
                if (b == (void *) 0x100000)
                        b = (void *) 0xA0000;
        }
        h--;
        free_list = start_buffer;
        free_list->b_prev_free = h;
        h->b_next_free = free_list;
        for (i=0;i<NR_HASH;i++)
                hash_table[i]=NULL;
}
```

```
/*
 *  linux/fs/char_dev.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <errno.h>
#include <sys/types.h>

#include <linux/sched.h>
#include <linux/kernel.h>

#include <asm/segment.h>
#include <asm/io.h>

extern int tty_read(unsigned minor,char * buf,int count);
extern int tty_write(unsigned minor,char * buf,int count);

typedef (*crw_ptr)(int rw,unsigned minor,char * buf,int count,off_t * pos);

static int rw_ttyx(int rw,unsigned minor,char * buf,int count,off_t * pos)
{
        return ((rw==READ)?tty_read(minor,buf,count):
                tty_write(minor,buf,count));
}

static int rw_tty(int rw,unsigned minor,char * buf,int count, off_t * pos)
{
        if (current->tty<0)
                return -EPERM;
        return rw_ttyx(rw,current->tty,buf,count,pos);
}

static int rw_ram(int rw,char * buf, int count, off_t *pos)
{
        return -EIO;
}

static int rw_mem(int rw,char * buf, int count, off_t * pos)
{
        return -EIO;
}

static int rw_kmem(int rw,char * buf, int count, off_t * pos)
{
        return -EIO;
}

static int rw_port(int rw,char * buf, int count, off_t * pos)
{
        int i=*pos;

        while (count-->0 && i<65536) {
                if (rw==READ)
                        put_fs_byte(inb(i),buf++);
                else
                        outb(get_fs_byte(buf++),i);
                i++;
        }
        i -= *pos;
        *pos += i;
        return i;
}

static int rw_memory(int rw, unsigned minor, char * buf, int count, off_t * pos)
{
        switch(minor) {
                case 0:
                        return rw_ram(rw,buf,count,pos);
```

```
                case 1:
                        return rw_mem(rw,buf,count,pos);
                case 2:
                        return rw_kmem(rw,buf,count,pos);
                case 3:
                        return (rw==READ)?0:count;      /* rw_null */
                case 4:
                        return rw_port(rw,buf,count,pos);
                default:
                        return -EIO;
        }
}

#define NRDEVS ((sizeof (crw_table))/(sizeof (crw_ptr)))

static crw_ptr crw_table[]={
        NULL,           /* nodev */
        rw_memory,      /* /dev/mem etc */
        NULL,           /* /dev/fd */
        NULL,           /* /dev/hd */
        rw_ttyx,        /* /dev/ttyx */
        rw_tty,         /* /dev/tty */
        NULL,           /* /dev/lp */
        NULL};          /* unnamed pipes */

int rw_char(int rw,int dev, char * buf, int count, off_t * pos)
{
        crw_ptr call_addr;

        if (MAJOR(dev)>=NRDEVS)
                return -ENODEV;
        if (!(call_addr=crw_table[MAJOR(dev)]))
                return -ENODEV;
        return call_addr(rw,MINOR(dev),buf,count,pos);
}
```

```
/*
 *  linux/fs/exec.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * #!-checking implemented by tytso.
 */

/*
 * Demand-loading implemented 01.12.91 - no need to read anything but
 * the header into memory. The inode of the executable is put into
 * "current->executable", and page faults do the actual loading. Clean.
 *
 * Once more I can proudly say that linux stood up to being changed: it
 * was less than 2 hours work to get demand-loading completely implemented.
 */

#include <errno.h>
#include <string.h>
#include <sys/stat.h>
#include <a.out.h>

#include <linux/fs.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <linux/mm.h>
#include <asm/segment.h>

extern int sys_exit(int exit_code);
extern int sys_close(int fd);

/*
 * MAX_ARG_PAGES defines the number of pages allocated for arguments
 * and envelope for the new program. 32 should suffice, this gives
 * a maximum env+arg of 128kB !
 */
#define MAX_ARG_PAGES 32

/*
 * create_tables() parses the env- and arg-strings in new user
 * memory and creates the pointer tables from them, and puts their
 * addresses on the "stack", returning the new stack pointer value.
 */
static unsigned long * create_tables(char * p,int argc,int envc)
{
        unsigned long *argv,*envp;
        unsigned long * sp;

        sp = (unsigned long *) (0xfffffffc & (unsigned long) p);
        sp -= envc+1;
        envp = sp;
        sp -= argc+1;
        argv = sp;
        put_fs_long((unsigned long)envp,--sp);
        put_fs_long((unsigned long)argv,--sp);
        put_fs_long((unsigned long)argc,--sp);
        while (argc-->0) {
                put_fs_long((unsigned long) p,argv++);
                while (get_fs_byte(p++)) /* nothing */ ;
        }
        put_fs_long(0,argv);
        while (envc-->0) {
                put_fs_long((unsigned long) p,envp++);
                while (get_fs_byte(p++)) /* nothing */ ;
        }
        put_fs_long(0,envp);
        return sp;
```

```
}

/*
 * count() counts the number of arguments/envelopes
 */
static int count(char ** argv)
{
        int i=0;
        char ** tmp;

        if (tmp = argv)
                while (get_fs_long((unsigned long *) (tmp++)))
                        i++;

        return i;
}

/*
 * 'copy_string()' copies argument/envelope strings from user
 * memory to free pages in kernel mem. These are in a format ready
 * to be put directly into the top of new user memory.
 *
 * Modified by TYT, 11/24/91 to add the from_kmem argument, which specifies
 * whether the string and the string array are from user or kernel segments:
 *
 * from_kmem     argv *        argv **
 *    0          user space    user space
 *    1          kernel space  user space
 *    2          kernel space  kernel space
 *
 * We do this by playing games with the fs segment register.  Since it
 * it is expensive to load a segment register, we try to avoid calling
 * set_fs() unless we absolutely have to.
 */
static unsigned long copy_strings(int argc,char ** argv,unsigned long *page,
                unsigned long p, int from_kmem)
{
        char *tmp, *pag;
        int len, offset = 0;
        unsigned long old_fs, new_fs;

        if (!p)
                return 0;       /* bullet-proofing */
        new_fs = get_ds();
        old_fs = get_fs();
        if (from_kmem==2)
                set_fs(new_fs);
        while (argc-- > 0) {
                if (from_kmem == 1)
                        set_fs(new_fs);
                if (!(tmp = (char *)get_fs_long(((unsigned long *)argv)+argc)))
                        panic("argc is wrong");
                if (from_kmem == 1)
                        set_fs(old_fs);
                len=0;          /* remember zero-padding */
                do {
                        len++;
                } while (get_fs_byte(tmp++));
                if (p-len < 0) {        /* this shouldn't happen - 128kB */
                        set_fs(old_fs);
                        return 0;
                }
                while (len) {
                        --p; --tmp; --len;
                        if (--offset < 0) {
                                offset = p % PAGE_SIZE;
                                if (from_kmem==2)
                                        set_fs(old_fs);
                                if (!(pag = (char *) page[p/PAGE_SIZE]) &&
```

```
                                !(pag = (char *) page[p/PAGE_SIZE] =
                                  (unsigned long *) get_free_page()))
                                    return 0;
                            if (from_kmem==2)
                                    set_fs(new_fs);

                        }
                        *(pag + offset) = get_fs_byte(tmp);
                }
        }
        if (from_kmem==2)
                set_fs(old_fs);
        return p;
}

static unsigned long change_ldt(unsigned long text_size,unsigned long * page)
{
        unsigned long code_limit,data_limit,code_base,data_base;
        int i;

        code_limit = text_size+PAGE_SIZE -1;
        code_limit &= 0xFFFFF000;
        data_limit = 0x4000000;
        code_base = get_base(current->ldt[1]);
        data_base = code_base;
        set_base(current->ldt[1],code_base);
        set_limit(current->ldt[1],code_limit);
        set_base(current->ldt[2],data_base);
        set_limit(current->ldt[2],data_limit);
/* make sure fs points to the NEW data segment */
        __asm__("pushl $0x17\n\tpop %%fs"::);
        data_base += data_limit;
        for (i=MAX_ARG_PAGES-1 ; i>=0 ; i--) {
                data_base -= PAGE_SIZE;
                if (page[i])
                        put_page(page[i],data_base);
        }
        return data_limit;
}

/*
 * 'do_execve()' executes a new program.
 */
int do_execve(unsigned long * eip,long tmp,char * filename,
        char ** argv, char ** envp)
{
        struct m_inode * inode;
        struct buffer_head * bh;
        struct exec ex;
        unsigned long page[MAX_ARG_PAGES];
        int i,argc,envc;
        int e_uid, e_gid;
        int retval;
        int sh_bang = 0;
        unsigned long p=PAGE_SIZE*MAX_ARG_PAGES-4;

        if ((0xffff & eip[1]) != 0x000f)
                panic("execve called from supervisor mode");
        for (i=0 ; i<MAX_ARG_PAGES ; i++)       /* clear page-table */
                page[i]=0;
        if (!(inode=namei(filename)))           /* get executables inode */
                return -ENOENT;
        argc = count(argv);
        envc = count(envp);

restart_interp:
        if (!S_ISREG(inode->i_mode)) {  /* must be regular file */
                retval = -EACCES;
                goto exec_error2;
```

```
	}
	i = inode->i_mode;
	e_uid = (i & S_ISUID) ? inode->i_uid : current->euid;
	e_gid = (i & S_ISGID) ? inode->i_gid : current->egid;
	if (current->euid == inode->i_uid)
		i >>= 6;
	else if (current->egid == inode->i_gid)
		i >>= 3;
	if (!(i & 1) &&
	    !((inode->i_mode & 0111) && suser())) {
		retval = -ENOEXEC;
		goto exec_error2;
	}
	if (!(bh = bread(inode->i_dev,inode->i_zone[0]))) {
		retval = -EACCES;
		goto exec_error2;
	}
	ex = *((struct exec *) bh->b_data);	/* read exec-header */
	if ((bh->b_data[0] == '#') && (bh->b_data[1] == '!') && (!sh_bang)) {
		/*
		 * This section does the #! interpretation.
		 * Sorta complicated, but hopefully it will work.  -TYT
		 */

		char buf[1023], *cp, *interp, *i_name, *i_arg;
		unsigned long old_fs;

		strncpy(buf, bh->b_data+2, 1022);
		brelse(bh);
		iput(inode);
		buf[1022] = '\0';
		if (cp = strchr(buf, '\n')) {
			*cp = '\0';
			for (cp = buf; (*cp == ' ') || (*cp == '\t'); cp++);
		}
		if (!cp || *cp == '\0') {
			retval = -ENOEXEC; /* No interpreter name found */
			goto exec_error1;
		}
		interp = i_name = cp;
		i_arg = 0;
		for ( ; *cp && (*cp != ' ') && (*cp != '\t'); cp++) {
			if (*cp == '/')
				i_name = cp+1;
		}
		if (*cp) {
			*cp++ = '\0';
			i_arg = cp;
		}
		/*
		 * OK, we've parsed out the interpreter name and
		 * (optional) argument.
		 */
		if (sh_bang++ == 0) {
			p = copy_strings(envc, envp, page, p, 0);
			p = copy_strings(--argc, argv+1, page, p, 0);
		}
		/*
		 * Splice in (1) the interpreter's name for argv[0]
		 *           (2) (optional) argument to interpreter
		 *           (3) filename of shell script
		 *
		 * This is done in reverse order, because of how the
		 * user environment and arguments are stored.
		 */
		p = copy_strings(1, &filename, page, p, 1);
		argc++;
		if (i_arg) {
			p = copy_strings(1, &i_arg, page, p, 2);
```

```
                        argc++;
                }
                p = copy_strings(1, &i_name, page, p, 2);
                argc++;
                if (!p) {
                        retval = -ENOMEM;
                        goto exec_error1;
                }
                /*
                 * OK, now restart the process with the interpreter's inode.
                 */
                old_fs = get_fs();
                set_fs(get_ds());
                if (!(inode=namei(interp))) { /* get executables inode */
                        set_fs(old_fs);
                        retval = -ENOENT;
                        goto exec_error1;
                }
                set_fs(old_fs);
                goto restart_interp;
        }
        brelse(bh);
        if (N_MAGIC(ex) != ZMAGIC || ex.a_trsize || ex.a_drsize ||
                ex.a_text+ex.a_data+ex.a_bss>0x3000000 ||
                inode->i_size < ex.a_text+ex.a_data+ex.a_syms+N_TXTOFF(ex)) {
                retval = -ENOEXEC;
                goto exec_error2;
        }
        if (N_TXTOFF(ex) != BLOCK_SIZE) {
                printk("%s: N_TXTOFF != BLOCK_SIZE. See a.out.h.", filename);
                retval = -ENOEXEC;
                goto exec_error2;
        }
        if (!sh_bang) {
                p = copy_strings(envc,envp,page,p,0);
                p = copy_strings(argc,argv,page,p,0);
                if (!p) {
                        retval = -ENOMEM;
                        goto exec_error2;
                }
        }
/* OK, This is the point of no return */
        if (current->executable)
                iput(current->executable);
        current->executable = inode;
        for (i=0 ; i<32 ; i++)
                current->sigaction[i].sa_handler = NULL;
        for (i=0 ; i<NR_OPEN ; i++)
                if ((current->close_on_exec>>i)&1)
                        sys_close(i);
        current->close_on_exec = 0;
        free_page_tables(get_base(current->ldt[1]),get_limit(0x0f));
        free_page_tables(get_base(current->ldt[2]),get_limit(0x17));
        if (last_task_used_math == current)
                last_task_used_math = NULL;
        current->used_math = 0;
        p += change_ldt(ex.a_text,page)-MAX_ARG_PAGES*PAGE_SIZE;
        p = (unsigned long) create_tables((char *)p,argc,envc);
        current->brk = ex.a_bss +
                (current->end_data = ex.a_data +
                (current->end_code = ex.a_text));
        current->start_stack = p & 0xfffff000;
        current->euid = e_uid;
        current->egid = e_gid;
        i = ex.a_text+ex.a_data;
        while (i&0xfff)
                put_fs_byte(0,(char *) (i++));
        eip[0] = ex.a_entry;            /* eip, magic happens :-) */
        eip[3] = p;                     /* stack pointer */
```

```
        return 0;
exec_error2:
        iput(inode);
exec_error1:
        for (i=0 ; i<MAX_ARG_PAGES ; i++)
                free_page(page[i]);
        return(retval);
}
```

```
/*
 *  linux/fs/fcntl.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <string.h>
#include <errno.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/segment.h>

#include <fcntl.h>
#include <sys/stat.h>

extern int sys_close(int fd);

static int dupfd(unsigned int fd, unsigned int arg)
{
        if (fd >= NR_OPEN || !current->filp[fd])
                return -EBADF;
        if (arg >= NR_OPEN)
                return -EINVAL;
        while (arg < NR_OPEN)
                if (current->filp[arg])
                        arg++;
                else
                        break;
        if (arg >= NR_OPEN)
                return -EMFILE;
        current->close_on_exec &= ~(1<<arg);
        (current->filp[arg] = current->filp[fd])->f_count++;
        return arg;
}

int sys_dup2(unsigned int oldfd, unsigned int newfd)
{
        sys_close(newfd);
        return dupfd(oldfd,newfd);
}

int sys_dup(unsigned int fildes)
{
        return dupfd(fildes,0);
}

int sys_fcntl(unsigned int fd, unsigned int cmd, unsigned long arg)
{
        struct file * filp;

        if (fd >= NR_OPEN || !(filp = current->filp[fd]))
                return -EBADF;
        switch (cmd) {
                case F_DUPFD:
                        return dupfd(fd,arg);
                case F_GETFD:
                        return (current->close_on_exec>>fd)&1;
                case F_SETFD:
                        if (arg&1)
                                current->close_on_exec |= (1<<fd);
                        else
                                current->close_on_exec &= ~(1<<fd);
                        return 0;
                case F_GETFL:
                        return filp->f_flags;
                case F_SETFL:
                        filp->f_flags &= ~(O_APPEND | O_NONBLOCK);
                        filp->f_flags |= arg & (O_APPEND | O_NONBLOCK);
                        return 0;
```

```
                case F_GETLK:   case F_SETLK:   case F_SETLKW:
                        return -1;
                default:
                        return -1;
        }
}
```

```
/*
 *  linux/fs/file_dev.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <errno.h>
#include <fcntl.h>

#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/segment.h>

#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))

int file_read(struct m_inode * inode, struct file * filp, char * buf, int count)
{
        int left,chars,nr;
        struct buffer_head * bh;

        if ((left=count)<=0)
                return 0;
        while (left) {
                if (nr = bmap(inode,(filp->f_pos)/BLOCK_SIZE)) {
                        if (!(bh=bread(inode->i_dev,nr)))
                                break;
                } else
                        bh = NULL;
                nr = filp->f_pos % BLOCK_SIZE;
                chars = MIN( BLOCK_SIZE-nr , left );
                filp->f_pos += chars;
                left -= chars;
                if (bh) {
                        char * p = nr + bh->b_data;
                        while (chars-->0)
                                put_fs_byte(*(p++),buf++);
                        brelse(bh);
                } else {
                        while (chars-->0)
                                put_fs_byte(0,buf++);
                }
        }
        inode->i_atime = CURRENT_TIME;
        return (count-left)?(count-left):-ERROR;
}

int file_write(struct m_inode * inode, struct file * filp, char * buf, int count)
{
        off_t pos;
        int block,c;
        struct buffer_head * bh;
        char * p;
        int i=0;

/*
 * ok, append may not work when many processes are writing at the same time
 * but so what. That way leads to madness anyway.
 */
        if (filp->f_flags & O_APPEND)
                pos = inode->i_size;
        else
                pos = filp->f_pos;
        while (i<count) {
                if (!(block = create_block(inode,pos/BLOCK_SIZE)))
                        break;
                if (!(bh=bread(inode->i_dev,block)))
                        break;
                c = pos % BLOCK_SIZE;
```

```
                p = c + bh->b_data;
                bh->b_dirt = 1;
                c = BLOCK_SIZE-c;
                if (c > count-i) c = count-i;
                pos += c;
                if (pos > inode->i_size) {
                        inode->i_size = pos;
                        inode->i_dirt = 1;
                }
                i += c;
                while (c-->0)
                        *(p++) = get_fs_byte(buf++);
                brelse(bh);
        }
        inode->i_mtime = CURRENT_TIME;
        if (!(filp->f_flags & O_APPEND)) {
                filp->f_pos = pos;
                inode->i_ctime = CURRENT_TIME;
        }
        return (i?i:-1);
}
```

```
/*
 *  linux/fs/file_table.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <linux/fs.h>

struct file file_table[NR_FILE];
```

```
/*
 *  linux/fs/inode.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <string.h>
#include <sys/stat.h>

#include <linux/sched.h>
#include <linux/kernel.h>
#include <linux/mm.h>
#include <asm/system.h>

struct m_inode inode_table[NR_INODE]={{0,},};

static void read_inode(struct m_inode * inode);
static void write_inode(struct m_inode * inode);

static inline void wait_on_inode(struct m_inode * inode)
{
        cli();
        while (inode->i_lock)
                sleep_on(&inode->i_wait);
        sti();
}

static inline void lock_inode(struct m_inode * inode)
{
        cli();
        while (inode->i_lock)
                sleep_on(&inode->i_wait);
        inode->i_lock=1;
        sti();
}

static inline void unlock_inode(struct m_inode * inode)
{
        inode->i_lock=0;
        wake_up(&inode->i_wait);
}

void invalidate_inodes(int dev)
{
        int i;
        struct m_inode * inode;

        inode = 0+inode_table;
        for(i=0 ; i<NR_INODE ; i++,inode++) {
                wait_on_inode(inode);
                if (inode->i_dev == dev) {
                        if (inode->i_count)
                                printk("inode in use on removed disk\n\r");
                        inode->i_dev = inode->i_dirt = 0;
                }
        }
}

void sync_inodes(void)
{
        int i;
        struct m_inode * inode;

        inode = 0+inode_table;
        for(i=0 ; i<NR_INODE ; i++,inode++) {
                wait_on_inode(inode);
                if (inode->i_dirt && !inode->i_pipe)
                        write_inode(inode);
        }
```

```
}

static int _bmap(struct m_inode * inode,int block,int create)
{
        struct buffer_head * bh;
        int i;

        if (block<0)
                panic("_bmap: block<0");
        if (block >= 7+512+512*512)
                panic("_bmap: block>big");
        if (block<7) {
                if (create && !inode->i_zone[block])
                        if (inode->i_zone[block]=new_block(inode->i_dev)) {
                                inode->i_ctime=CURRENT_TIME;
                                inode->i_dirt=1;
                        }
                return inode->i_zone[block];
        }
        block -= 7;
        if (block<512) {
                if (create && !inode->i_zone[7])
                        if (inode->i_zone[7]=new_block(inode->i_dev)) {
                                inode->i_dirt=1;
                                inode->i_ctime=CURRENT_TIME;
                        }
                if (!inode->i_zone[7])
                        return 0;
                if (!(bh = bread(inode->i_dev,inode->i_zone[7])))
                        return 0;
                i = ((unsigned short *) (bh->b_data))[block];
                if (create && !i)
                        if (i=new_block(inode->i_dev)) {
                                ((unsigned short *) (bh->b_data))[block]=i;
                                bh->b_dirt=1;
                        }
                brelse(bh);
                return i;
        }
        block -= 512;
        if (create && !inode->i_zone[8])
                if (inode->i_zone[8]=new_block(inode->i_dev)) {
                        inode->i_dirt=1;
                        inode->i_ctime=CURRENT_TIME;
                }
        if (!inode->i_zone[8])
                return 0;
        if (!(bh=bread(inode->i_dev,inode->i_zone[8])))
                return 0;
        i = ((unsigned short *)bh->b_data)[block>>9];
        if (create && !i)
                if (i=new_block(inode->i_dev)) {
                        ((unsigned short *) (bh->b_data))[block>>9]=i;
                        bh->b_dirt=1;
                }
        brelse(bh);
        if (!i)
                return 0;
        if (!(bh=bread(inode->i_dev,i)))
                return 0;
        i = ((unsigned short *)bh->b_data)[block&511];
        if (create && !i)
                if (i=new_block(inode->i_dev)) {
                        ((unsigned short *) (bh->b_data))[block&511]=i;
                        bh->b_dirt=1;
                }
        brelse(bh);
        return i;
}
```

```
int bmap(struct m_inode * inode,int block)
{
        return _bmap(inode,block,0);
}

int create_block(struct m_inode * inode, int block)
{
        return _bmap(inode,block,1);
}

void iput(struct m_inode * inode)
{
        if (!inode)
                return;
        wait_on_inode(inode);
        if (!inode->i_count)
                panic("iput: trying to free free inode");
        if (inode->i_pipe) {
                wake_up(&inode->i_wait);
                if (--inode->i_count)
                        return;
                free_page(inode->i_size);
                inode->i_count=0;
                inode->i_dirt=0;
                inode->i_pipe=0;
                return;
        }
        if (!inode->i_dev) {
                inode->i_count--;
                return;
        }
        if (S_ISBLK(inode->i_mode)) {
                sync_dev(inode->i_zone[0]);
                wait_on_inode(inode);
        }
repeat:
        if (inode->i_count>1) {
                inode->i_count--;
                return;
        }
        if (!inode->i_nlinks) {
                truncate(inode);
                free_inode(inode);
                return;
        }
        if (inode->i_dirt) {
                write_inode(inode);     /* we can sleep - so do again */
                wait_on_inode(inode);
                goto repeat;
        }
        inode->i_count--;
        return;
}

struct m_inode * get_empty_inode(void)
{
        struct m_inode * inode;
        static struct m_inode * last_inode = inode_table;
        int i;

        do {
                inode = NULL;
                for (i = NR_INODE; i ; i--) {
                        if (++last_inode >= inode_table + NR_INODE)
                                last_inode = inode_table;
                        if (!last_inode->i_count) {
                                inode = last_inode;
                                if (!inode->i_dirt && !inode->i_lock)
```

```
                                        break;
                        }
                }
                if (!inode) {
                        for (i=0 ; i<NR_INODE ; i++)
                                printk("%04x: %6d\t",inode_table[i].i_dev,
                                        inode_table[i].i_num);
                        panic("No free inodes in mem");
                }
                wait_on_inode(inode);
                while (inode->i_dirt) {
                        write_inode(inode);
                        wait_on_inode(inode);
                }
        } while (inode->i_count);
        memset(inode,0,sizeof(*inode));
        inode->i_count = 1;
        return inode;
}

struct m_inode * get_pipe_inode(void)
{
        struct m_inode * inode;

        if (!(inode = get_empty_inode()))
                return NULL;
        if (!(inode->i_size=get_free_page())) {
                inode->i_count = 0;
                return NULL;
        }
        inode->i_count = 2;     /* sum of readers/writers */
        PIPE_HEAD(*inode) = PIPE_TAIL(*inode) = 0;
        inode->i_pipe = 1;
        return inode;
}

struct m_inode * iget(int dev,int nr)
{
        struct m_inode * inode, * empty;

        if (!dev)
                panic("iget with dev==0");
        empty = get_empty_inode();
        inode = inode_table;
        while (inode < NR_INODE+inode_table) {
                if (inode->i_dev != dev || inode->i_num != nr) {
                        inode++;
                        continue;
                }
                wait_on_inode(inode);
                if (inode->i_dev != dev || inode->i_num != nr) {
                        inode = inode_table;
                        continue;
                }
                inode->i_count++;
                if (inode->i_mount) {
                        int i;

                        for (i = 0 ; i<NR_SUPER ; i++)
                                if (super_block[i].s_imount==inode)
                                        break;
                        if (i >= NR_SUPER) {
                                printk("Mounted inode hasn't got sb\n");
                                if (empty)
                                        iput(empty);
                                return inode;
                        }
                        iput(inode);
                        dev = super_block[i].s_dev;
```

```
                        nr = ROOT_INO;
                        inode = inode_table;
                        continue;
                }
                if (empty)
                        iput(empty);
                return inode;
        }
        if (!empty)
                return (NULL);
        inode=empty;
        inode->i_dev = dev;
        inode->i_num = nr;
        read_inode(inode);
        return inode;
}

static void read_inode(struct m_inode * inode)
{
        struct super_block * sb;
        struct buffer_head * bh;
        int block;

        lock_inode(inode);
        if (!(sb=get_super(inode->i_dev)))
                panic("trying to read inode without dev");
        block = 2 + sb->s_imap_blocks + sb->s_zmap_blocks +
                (inode->i_num-1)/INODES_PER_BLOCK;
        if (!(bh=bread(inode->i_dev,block)))
                panic("unable to read i-node block");
        *(struct d_inode *)inode =
                ((struct d_inode *)bh->b_data)
                        [(inode->i_num-1)%INODES_PER_BLOCK];
        brelse(bh);
        unlock_inode(inode);
}

static void write_inode(struct m_inode * inode)
{
        struct super_block * sb;
        struct buffer_head * bh;
        int block;

        lock_inode(inode);
        if (!inode->i_dirt || !inode->i_dev) {
                unlock_inode(inode);
                return;
        }
        if (!(sb=get_super(inode->i_dev)))
                panic("trying to write inode without device");
        block = 2 + sb->s_imap_blocks + sb->s_zmap_blocks +
                (inode->i_num-1)/INODES_PER_BLOCK;
        if (!(bh=bread(inode->i_dev,block)))
                panic("unable to read i-node block");
        ((struct d_inode *)bh->b_data)
                [(inode->i_num-1)%INODES_PER_BLOCK] =
                        *(struct d_inode *)inode;
        bh->b_dirt=1;
        inode->i_dirt=0;
        brelse(bh);
        unlock_inode(inode);
}
```

```
/*
 *  linux/fs/ioctl.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <string.h>
#include <errno.h>
#include <sys/stat.h>

#include <linux/sched.h>

extern int tty_ioctl(int dev, int cmd, int arg);

typedef int (*ioctl_ptr)(int dev,int cmd,int arg);

#define NRDEVS ((sizeof (ioctl_table))/(sizeof (ioctl_ptr)))

static ioctl_ptr ioctl_table[]={
        NULL,           /* nodev */
        NULL,           /* /dev/mem */
        NULL,           /* /dev/fd */
        NULL,           /* /dev/hd */
        tty_ioctl,      /* /dev/ttyx */
        tty_ioctl,      /* /dev/tty */
        NULL,           /* /dev/lp */
        NULL};          /* named pipes */


int sys_ioctl(unsigned int fd, unsigned int cmd, unsigned long arg)
{
        struct file * filp;
        int dev,mode;

        if (fd >= NR_OPEN || !(filp = current->filp[fd]))
                return -EBADF;
        mode=filp->f_inode->i_mode;
        if (!S_ISCHR(mode) && !S_ISBLK(mode))
                return -EINVAL;
        dev = filp->f_inode->i_zone[0];
        if (MAJOR(dev) >= NRDEVS)
                return -ENODEV;
        if (!ioctl_table[MAJOR(dev)])
                return -ENOTTY;
        return ioctl_table[MAJOR(dev)](dev,cmd,arg);
}
```

```
/*
 *  linux/fs/namei.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * Some corrections by tytso.
 */

#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/segment.h>

#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <const.h>
#include <sys/stat.h>

#define ACC_MODE(x) ("\004\002\006\377"[(x)&O_ACCMODE])

/*
 * comment out this line if you want names > NAME_LEN chars to be
 * truncated. Else they will be disallowed.
 */
/* #define NO_TRUNCATE */

#define MAY_EXEC 1
#define MAY_WRITE 2
#define MAY_READ 4

/*
 *      permission()
 *
 * is used to check for read/write/execute permissions on a file.
 * I don't know if we should look at just the euid or both euid and
 * uid, but that should be easily changed.
 */
static int permission(struct m_inode * inode,int mask)
{
        int mode = inode->i_mode;

/* special case: not even root can read/write a deleted file */
        if (inode->i_dev && !inode->i_nlinks)
                return 0;
        else if (current->euid==inode->i_uid)
                mode >>= 6;
        else if (current->egid==inode->i_gid)
                mode >>= 3;
        if (((mode & mask & 0007) == mask) || suser())
                return 1;
        return 0;
}

/*
 * ok, we cannot use strncmp, as the name is not in our data space.
 * Thus we'll have to use match. No big problem. Match also makes
 * some sanity tests.
 *
 * NOTE! unlike strncmp, match returns 1 for success, 0 for failure.
 */
static int match(int len,const char * name,struct dir_entry * de)
{
        register int same __asm__("ax");

        if (!de || !de->inode || len > NAME_LEN)
                return 0;
        if (len < NAME_LEN && de->name[len])
```

```
                return 0;
        __asm__("cld\n\t"
                "fs ; repe ; cmpsb\n\t"
                "setz %%al"
                :"=a" (same)
                :"0" (0),"S" ((long) name),"D" ((long) de->name),"c" (len)
                :"cx","di","si");
        return same;
}

/*
 *      find_entry()
 *
 * finds an entry in the specified directory with the wanted name. It
 * returns the cache buffer in which the entry was found, and the entry
 * itself (as a parameter - res_dir). It does NOT read the inode of the
 * entry - you'll have to do that yourself if you want to.
 *
 * This also takes care of the few special cases due to '..'-traversal
 * over a pseudo-root and a mount point.
 */
static struct buffer_head * find_entry(struct m_inode ** dir,
        const char * name, int namelen, struct dir_entry ** res_dir)
{
        int entries;
        int block,i;
        struct buffer_head * bh;
        struct dir_entry * de;
        struct super_block * sb;

#ifdef NO_TRUNCATE
        if (namelen > NAME_LEN)
                return NULL;
#else
        if (namelen > NAME_LEN)
                namelen = NAME_LEN;
#endif
        entries = (*dir)->i_size / (sizeof (struct dir_entry));
        *res_dir = NULL;
        if (!namelen)
                return NULL;
/* check for '..', as we might have to do some "magic" for it */
        if (namelen==2 && get_fs_byte(name)=='.' && get_fs_byte(name+1)=='.') {
/* '..' in a pseudo-root results in a faked '.' (just change namelen) */
                if ((*dir) == current->root)
                        namelen=1;
                else if ((*dir)->i_num == ROOT_INO) {
/* '..' over a mount-point results in 'dir' being exchanged for the mounted
   directory-inode. NOTE! We set mounted, so that we can iput the new dir */
                        sb=get_super((*dir)->i_dev);
                        if (sb->s_imount) {
                                iput(*dir);
                                (*dir)=sb->s_imount;
                                (*dir)->i_count++;
                        }
                }
        }
        if (!(block = (*dir)->i_zone[0]))
                return NULL;
        if (!(bh = bread((*dir)->i_dev,block)))
                return NULL;
        i = 0;
        de = (struct dir_entry *) bh->b_data;
        while (i < entries) {
                if ((char *)de >= BLOCK_SIZE+bh->b_data) {
                        brelse(bh);
                        bh = NULL;
                        if (!(block = bmap(*dir,i/DIR_ENTRIES_PER_BLOCK)) ||
                            !(bh = bread((*dir)->i_dev,block))) {
```

```
                                i += DIR_ENTRIES_PER_BLOCK;
                                continue;
                        }
                        de = (struct dir_entry *) bh->b_data;
                }
                if (match(namelen,name,de)) {
                        *res_dir = de;
                        return bh;
                }
                de++;
                i++;
        }
        brelse(bh);
        return NULL;
}

/*
 *      add_entry()
 *
 * adds a file entry to the specified directory, using the same
 * semantics as find_entry(). It returns NULL if it failed.
 *
 * NOTE!! The inode part of 'de' is left at 0 - which means you
 * may not sleep between calling this and putting something into
 * the entry, as someone else might have used it while you slept.
 */
static struct buffer_head * add_entry(struct m_inode * dir,
        const char * name, int namelen, struct dir_entry ** res_dir)
{
        int block,i;
        struct buffer_head * bh;
        struct dir_entry * de;

        *res_dir = NULL;
#ifdef NO_TRUNCATE
        if (namelen > NAME_LEN)
                return NULL;
#else
        if (namelen > NAME_LEN)
                namelen = NAME_LEN;
#endif
        if (!namelen)
                return NULL;
        if (!(block = dir->i_zone[0]))
                return NULL;
        if (!(bh = bread(dir->i_dev,block)))
                return NULL;
        i = 0;
        de = (struct dir_entry *) bh->b_data;
        while (1) {
                if ((char *)de >= BLOCK_SIZE+bh->b_data) {
                        brelse(bh);
                        bh = NULL;
                        block = create_block(dir,i/DIR_ENTRIES_PER_BLOCK);
                        if (!block)
                                return NULL;
                        if (!(bh = bread(dir->i_dev,block))) {
                                i += DIR_ENTRIES_PER_BLOCK;
                                continue;
                        }
                        de = (struct dir_entry *) bh->b_data;
                }
                if (i*sizeof(struct dir_entry) >= dir->i_size) {
                        de->inode=0;
                        dir->i_size = (i+1)*sizeof(struct dir_entry);
                        dir->i_dirt = 1;
                        dir->i_ctime = CURRENT_TIME;
                }
                if (!de->inode) {
```

```
                        dir->i_mtime = CURRENT_TIME;
                        for (i=0; i < NAME_LEN ; i++)
                                de->name[i]=(i<namelen)?get_fs_byte(name+i):0;
                        bh->b_dirt = 1;
                        *res_dir = de;
                        return bh;
                }
                de++;
                i++;
        }
        brelse(bh);
        return NULL;
}

/*
 *      get_dir()
 *
 * Getdir traverses the pathname until it hits the topmost directory.
 * It returns NULL on failure.
 */
static struct m_inode * get_dir(const char * pathname)
{
        char c;
        const char * thisname;
        struct m_inode * inode;
        struct buffer_head * bh;
        int namelen,inr,idev;
        struct dir_entry * de;

        if (!current->root || !current->root->i_count)
                panic("No root inode");
        if (!current->pwd || !current->pwd->i_count)
                panic("No cwd inode");
        if ((c=get_fs_byte(pathname))=='/') {
                inode = current->root;
                pathname++;
        } else if (c)
                inode = current->pwd;
        else
                return NULL;    /* empty name is bad */
        inode->i_count++;
        while (1) {
                thisname = pathname;
                if (!S_ISDIR(inode->i_mode) || !permission(inode,MAY_EXEC)) {
                        iput(inode);
                        return NULL;
                }
                for(namelen=0;(c=get_fs_byte(pathname++))&&(c!='/');namelen++)
                        /* nothing */ ;
                if (!c)
                        return inode;
                if (!(bh = find_entry(&inode,thisname,namelen,&de))) {
                        iput(inode);
                        return NULL;
                }
                inr = de->inode;
                idev = inode->i_dev;
                brelse(bh);
                iput(inode);
                if (!(inode = iget(idev,inr)))
                        return NULL;
        }
}

/*
 *      dir_namei()
 *
 * dir_namei() returns the inode of the directory of the
 * specified name, and the name within that directory.
```

```
 */
static struct m_inode * dir_namei(const char * pathname,
        int * namelen, const char ** name)
{
        char c;
        const char * basename;
        struct m_inode * dir;

        if (!(dir = get_dir(pathname)))
                return NULL;
        basename = pathname;
        while (c=get_fs_byte(pathname++))
                if (c=='/')
                        basename=pathname;
        *namelen = pathname-basename-1;
        *name = basename;
        return dir;
}

/*
 *      namei()
 *
 * is used by most simple commands to get the inode of a specified name.
 * Open, link etc use their own routines, but this is enough for things
 * like 'chmod' etc.
 */
struct m_inode * namei(const char * pathname)
{
        const char * basename;
        int inr,dev,namelen;
        struct m_inode * dir;
        struct buffer_head * bh;
        struct dir_entry * de;

        if (!(dir = dir_namei(pathname,&namelen,&basename)))
                return NULL;
        if (!namelen)                   /* special case: '/usr/' etc */
                return dir;
        bh = find_entry(&dir,basename,namelen,&de);
        if (!bh) {
                iput(dir);
                return NULL;
        }
        inr = de->inode;
        dev = dir->i_dev;
        brelse(bh);
        iput(dir);
        dir=iget(dev,inr);
        if (dir) {
                dir->i_atime=CURRENT_TIME;
                dir->i_dirt=1;
        }
        return dir;
}

/*
 *      open_namei()
 *
 * namei for open - this is in fact almost the whole open-routine.
 */
int open_namei(const char * pathname, int flag, int mode,
        struct m_inode ** res_inode)
{
        const char * basename;
        int inr,dev,namelen;
        struct m_inode * dir, *inode;
        struct buffer_head * bh;
        struct dir_entry * de;
```

```
        if ((flag & O_TRUNC) && !(flag & O_ACCMODE))
                flag |= O_WRONLY;
        mode &= 0777 & ~current->umask;
        mode |= I_REGULAR;
        if (!(dir = dir_namei(pathname,&namelen,&basename)))
                return -ENOENT;
        if (!namelen) {                 /* special case: '/usr/' etc */
                if (!(flag & (O_ACCMODE|O_CREAT|O_TRUNC))) {
                        *res_inode=dir;
                        return 0;
                }
                iput(dir);
                return -EISDIR;
        }
        bh = find_entry(&dir,basename,namelen,&de);
        if (!bh) {
                if (!(flag & O_CREAT)) {
                        iput(dir);
                        return -ENOENT;
                }
                if (!permission(dir,MAY_WRITE)) {
                        iput(dir);
                        return -EACCES;
                }
                inode = new_inode(dir->i_dev);
                if (!inode) {
                        iput(dir);
                        return -ENOSPC;
                }
                inode->i_uid = current->euid;
                inode->i_mode = mode;
                inode->i_dirt = 1;
                bh = add_entry(dir,basename,namelen,&de);
                if (!bh) {
                        inode->i_nlinks--;
                        iput(inode);
                        iput(dir);
                        return -ENOSPC;
                }
                de->inode = inode->i_num;
                bh->b_dirt = 1;
                brelse(bh);
                iput(dir);
                *res_inode = inode;
                return 0;
        }
        inr = de->inode;
        dev = dir->i_dev;
        brelse(bh);
        iput(dir);
        if (flag & O_EXCL)
                return -EEXIST;
        if (!(inode=iget(dev,inr)))
                return -EACCES;
        if ((S_ISDIR(inode->i_mode) && (flag & O_ACCMODE)) ||
            !permission(inode,ACC_MODE(flag))) {
                iput(inode);
                return -EPERM;
        }
        inode->i_atime = CURRENT_TIME;
        if (flag & O_TRUNC)
                truncate(inode);
        *res_inode = inode;
        return 0;
}

int sys_mknod(const char * filename, int mode, int dev)
{
        const char * basename;
```

```
        int namelen;
        struct m_inode * dir, * inode;
        struct buffer_head * bh;
        struct dir_entry * de;

        if (!suser())
                return -EPERM;
        if (!(dir = dir_namei(filename,&namelen,&basename)))
                return -ENOENT;
        if (!namelen) {
                iput(dir);
                return -ENOENT;
        }
        if (!permission(dir,MAY_WRITE)) {
                iput(dir);
                return -EPERM;
        }
        bh = find_entry(&dir,basename,namelen,&de);
        if (bh) {
                brelse(bh);
                iput(dir);
                return -EEXIST;
        }
        inode = new_inode(dir->i_dev);
        if (!inode) {
                iput(dir);
                return -ENOSPC;
        }
        inode->i_mode = mode;
        if (S_ISBLK(mode) || S_ISCHR(mode))
                inode->i_zone[0] = dev;
        inode->i_mtime = inode->i_atime = CURRENT_TIME;
        inode->i_dirt = 1;
        bh = add_entry(dir,basename,namelen,&de);
        if (!bh) {
                iput(dir);
                inode->i_nlinks=0;
                iput(inode);
                return -ENOSPC;
        }
        de->inode = inode->i_num;
        bh->b_dirt = 1;
        iput(dir);
        iput(inode);
        brelse(bh);
        return 0;
}

int sys_mkdir(const char * pathname, int mode)
{
        const char * basename;
        int namelen;
        struct m_inode * dir, * inode;
        struct buffer_head * bh, *dir_block;
        struct dir_entry * de;

        if (!suser())
                return -EPERM;
        if (!(dir = dir_namei(pathname,&namelen,&basename)))
                return -ENOENT;
        if (!namelen) {
                iput(dir);
                return -ENOENT;
        }
        if (!permission(dir,MAY_WRITE)) {
                iput(dir);
                return -EPERM;
        }
        bh = find_entry(&dir,basename,namelen,&de);
```

```
        if (bh) {
                brelse(bh);
                iput(dir);
                return -EEXIST;
        }
        inode = new_inode(dir->i_dev);
        if (!inode) {
                iput(dir);
                return -ENOSPC;
        }
        inode->i_size = 32;
        inode->i_dirt = 1;
        inode->i_mtime = inode->i_atime = CURRENT_TIME;
        if (!(inode->i_zone[0]=new_block(inode->i_dev))) {
                iput(dir);
                inode->i_nlinks--;
                iput(inode);
                return -ENOSPC;
        }
        inode->i_dirt = 1;
        if (!(dir_block=bread(inode->i_dev,inode->i_zone[0]))) {
                iput(dir);
                free_block(inode->i_dev,inode->i_zone[0]);
                inode->i_nlinks--;
                iput(inode);
                return -ERROR;
        }
        de = (struct dir_entry *) dir_block->b_data;
        de->inode=inode->i_num;
        strcpy(de->name,".");
        de++;
        de->inode = dir->i_num;
        strcpy(de->name,"..");
        inode->i_nlinks = 2;
        dir_block->b_dirt = 1;
        brelse(dir_block);
        inode->i_mode = I_DIRECTORY | (mode & 0777 & ~current->umask);
        inode->i_dirt = 1;
        bh = add_entry(dir,basename,namelen,&de);
        if (!bh) {
                iput(dir);
                free_block(inode->i_dev,inode->i_zone[0]);
                inode->i_nlinks=0;
                iput(inode);
                return -ENOSPC;
        }
        de->inode = inode->i_num;
        bh->b_dirt = 1;
        dir->i_nlinks++;
        dir->i_dirt = 1;
        iput(dir);
        iput(inode);
        brelse(bh);
        return 0;
}

/*
 * routine to check that the specified directory is empty (for rmdir)
 */
static int empty_dir(struct m_inode * inode)
{
        int nr,block;
        int len;
        struct buffer_head * bh;
        struct dir_entry * de;

        len = inode->i_size / sizeof (struct dir_entry);
        if (len<2 || !inode->i_zone[0] ||
            !(bh=bread(inode->i_dev,inode->i_zone[0]))) {
```

```
                printk("warning - bad directory on dev %04x\n",inode->i_dev);
                return 0;
        }
        de = (struct dir_entry *) bh->b_data;
        if (de[0].inode != inode->i_num || !de[1].inode ||
            strcmp(".",de[0].name) || strcmp("..",de[1].name)) {
                printk("warning - bad directory on dev %04x\n",inode->i_dev);
                return 0;
        }
        nr = 2;
        de += 2;
        while (nr<len) {
                if ((void *) de >= (void *) (bh->b_data+BLOCK_SIZE)) {
                        brelse(bh);
                        block=bmap(inode,nr/DIR_ENTRIES_PER_BLOCK);
                        if (!block) {
                                nr += DIR_ENTRIES_PER_BLOCK;
                                continue;
                        }
                        if (!(bh=bread(inode->i_dev,block)))
                                return 0;
                        de = (struct dir_entry *) bh->b_data;
                }
                if (de->inode) {
                        brelse(bh);
                        return 0;
                }
                de++;
                nr++;
        }
        brelse(bh);
        return 1;
}

int sys_rmdir(const char * name)
{
        const char * basename;
        int namelen;
        struct m_inode * dir, * inode;
        struct buffer_head * bh;
        struct dir_entry * de;

        if (!suser())
                return -EPERM;
        if (!(dir = dir_namei(name,&namelen,&basename)))
                return -ENOENT;
        if (!namelen) {
                iput(dir);
                return -ENOENT;
        }
        if (!permission(dir,MAY_WRITE)) {
                iput(dir);
                return -EPERM;
        }
        bh = find_entry(&dir,basename,namelen,&de);
        if (!bh) {
                iput(dir);
                return -ENOENT;
        }
        if (!(inode = iget(dir->i_dev, de->inode))) {
                iput(dir);
                brelse(bh);
                return -EPERM;
        }
        if ((dir->i_mode & S_ISVTX) && current->euid &&
            inode->i_uid != current->euid) {
                iput(dir);
                iput(inode);
                brelse(bh);
```

```
                return -EPERM;
        }
        if (inode->i_dev != dir->i_dev || inode->i_count>1) {
                iput(dir);
                iput(inode);
                brelse(bh);
                return -EPERM;
        }
        if (inode == dir) {     /* we may not delete ".", but "../dir" is ok */
                iput(inode);
                iput(dir);
                brelse(bh);
                return -EPERM;
        }
        if (!S_ISDIR(inode->i_mode)) {
                iput(inode);
                iput(dir);
                brelse(bh);
                return -ENOTDIR;
        }
        if (!empty_dir(inode)) {
                iput(inode);
                iput(dir);
                brelse(bh);
                return -ENOTEMPTY;
        }
        if (inode->i_nlinks != 2)
                printk("empty directory has nlink!=2 (%d)",inode->i_nlinks);
        de->inode = 0;
        bh->b_dirt = 1;
        brelse(bh);
        inode->i_nlinks=0;
        inode->i_dirt=1;
        dir->i_nlinks--;
        dir->i_ctime = dir->i_mtime = CURRENT_TIME;
        dir->i_dirt=1;
        iput(dir);
        iput(inode);
        return 0;
}

int sys_unlink(const char * name)
{
        const char * basename;
        int namelen;
        struct m_inode * dir, * inode;
        struct buffer_head * bh;
        struct dir_entry * de;

        if (!(dir = dir_namei(name,&namelen,&basename)))
                return -ENOENT;
        if (!namelen) {
                iput(dir);
                return -ENOENT;
        }
        if (!permission(dir,MAY_WRITE)) {
                iput(dir);
                return -EPERM;
        }
        bh = find_entry(&dir,basename,namelen,&de);
        if (!bh) {
                iput(dir);
                return -ENOENT;
        }
        if (!(inode = iget(dir->i_dev, de->inode))) {
                iput(dir);
                brelse(bh);
                return -ENOENT;
        }
```

```
        if ((dir->i_mode & S_ISVTX) && !suser() &&
            current->euid != inode->i_uid &&
            current->euid != dir->i_uid) {
                iput(dir);
                iput(inode);
                brelse(bh);
                return -EPERM;
        }
        if (S_ISDIR(inode->i_mode)) {
                iput(inode);
                iput(dir);
                brelse(bh);
                return -EPERM;
        }
        if (!inode->i_nlinks) {
                printk("Deleting nonexistent file (%04x:%d), %d\n",
                        inode->i_dev,inode->i_num,inode->i_nlinks);
                inode->i_nlinks=1;
        }
        de->inode = 0;
        bh->b_dirt = 1;
        brelse(bh);
        inode->i_nlinks--;
        inode->i_dirt = 1;
        inode->i_ctime = CURRENT_TIME;
        iput(inode);
        iput(dir);
        return 0;
}

int sys_link(const char * oldname, const char * newname)
{
        struct dir_entry * de;
        struct m_inode * oldinode, * dir;
        struct buffer_head * bh;
        const char * basename;
        int namelen;

        oldinode=namei(oldname);
        if (!oldinode)
                return -ENOENT;
        if (S_ISDIR(oldinode->i_mode)) {
                iput(oldinode);
                return -EPERM;
        }
        dir = dir_namei(newname,&namelen,&basename);
        if (!dir) {
                iput(oldinode);
                return -EACCES;
        }
        if (!namelen) {
                iput(oldinode);
                iput(dir);
                return -EPERM;
        }
        if (dir->i_dev != oldinode->i_dev) {
                iput(dir);
                iput(oldinode);
                return -EXDEV;
        }
        if (!permission(dir,MAY_WRITE)) {
                iput(dir);
                iput(oldinode);
                return -EACCES;
        }
        bh = find_entry(&dir,basename,namelen,&de);
        if (bh) {
                brelse(bh);
                iput(dir);
```

```
                iput(oldinode);
                return -EEXIST;
        }
        bh = add_entry(dir,basename,namelen,&de);
        if (!bh) {
                iput(dir);
                iput(oldinode);
                return -ENOSPC;
        }
        de->inode = oldinode->i_num;
        bh->b_dirt = 1;
        brelse(bh);
        iput(dir);
        oldinode->i_nlinks++;
        oldinode->i_ctime = CURRENT_TIME;
        oldinode->i_dirt = 1;
        iput(oldinode);
        return 0;
}
```

```
/*
 *  linux/fs/open.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/types.h>
#include <utime.h>
#include <sys/stat.h>

#include <linux/sched.h>
#include <linux/tty.h>
#include <linux/kernel.h>
#include <asm/segment.h>

int sys_ustat(int dev, struct ustat * ubuf)
{
        return -ENOSYS;
}

int sys_utime(char * filename, struct utimbuf * times)
{
        struct m_inode * inode;
        long actime,modtime;

        if (!(inode=namei(filename)))
                return -ENOENT;
        if (times) {
                actime = get_fs_long((unsigned long *) &times->actime);
                modtime = get_fs_long((unsigned long *) &times->modtime);
        } else
                actime = modtime = CURRENT_TIME;
        inode->i_atime = actime;
        inode->i_mtime = modtime;
        inode->i_dirt = 1;
        iput(inode);
        return 0;
}

/*
 * XXX should we use the real or effective uid?  BSD uses the real uid,
 * so as to make this call useful to setuid programs.
 */
int sys_access(const char * filename,int mode)
{
        struct m_inode * inode;
        int res, i_mode;

        mode &= 0007;
        if (!(inode=namei(filename)))
                return -EACCES;
        i_mode = res = inode->i_mode & 0777;
        iput(inode);
        if (current->uid == inode->i_uid)
                res >>= 6;
        else if (current->gid == inode->i_gid)
                res >>= 6;
        if ((res & 0007 & mode) == mode)
                return 0;
        /*
         * XXX we are doing this test last because we really should be
         * swapping the effective with the real user id (temporarily),
         * and then calling suser() routine.  If we do call the
         * suser() routine, it needs to be called last.
         */
        if ((!current->uid) &&
```

```
            (!(mode & 1) || (i_mode & 0111)))
                return 0;
        return -EACCES;
}

int sys_chdir(const char * filename)
{
        struct m_inode * inode;

        if (!(inode = namei(filename)))
                return -ENOENT;
        if (!S_ISDIR(inode->i_mode)) {
                iput(inode);
                return -ENOTDIR;
        }
        iput(current->pwd);
        current->pwd = inode;
        return (0);
}

int sys_chroot(const char * filename)
{
        struct m_inode * inode;

        if (!(inode=namei(filename)))
                return -ENOENT;
        if (!S_ISDIR(inode->i_mode)) {
                iput(inode);
                return -ENOTDIR;
        }
        iput(current->root);
        current->root = inode;
        return (0);
}

int sys_chmod(const char * filename,int mode)
{
        struct m_inode * inode;

        if (!(inode=namei(filename)))
                return -ENOENT;
        if ((current->euid != inode->i_uid) && !suser()) {
                iput(inode);
                return -EACCES;
        }
        inode->i_mode = (mode & 07777) | (inode->i_mode & ~07777);
        inode->i_dirt = 1;
        iput(inode);
        return 0;
}

int sys_chown(const char * filename,int uid,int gid)
{
        struct m_inode * inode;

        if (!(inode=namei(filename)))
                return -ENOENT;
        if (!suser()) {
                iput(inode);
                return -EACCES;
        }
        inode->i_uid=uid;
        inode->i_gid=gid;
        inode->i_dirt=1;
        iput(inode);
        return 0;
}

int sys_open(const char * filename,int flag,int mode)
```

```
{
        struct m_inode * inode;
        struct file * f;
        int i,fd;

        mode &= 0777 & ~current->umask;
        for(fd=0 ; fd<NR_OPEN ; fd++)
                if (!current->filp[fd])
                        break;
        if (fd>=NR_OPEN)
                return -EINVAL;
        current->close_on_exec &= ~(1<<fd);
        f=0+file_table;
        for (i=0 ; i<NR_FILE ; i++,f++)
                if (!f->f_count) break;
        if (i>=NR_FILE)
                return -EINVAL;
        (current->filp[fd]=f)->f_count++;
        if ((i=open_namei(filename,flag,mode,&inode))<0) {
                current->filp[fd]=NULL;
                f->f_count=0;
                return i;
        }
/* ttys are somewhat special (ttyxx major==4, tty major==5) */
        if (S_ISCHR(inode->i_mode))
                if (MAJOR(inode->i_zone[0])==4) {
                        if (current->leader && current->tty<0) {
                                current->tty = MINOR(inode->i_zone[0]);
                                tty_table[current->tty].pgrp = current->pgrp;
                        }
                } else if (MAJOR(inode->i_zone[0])==5)
                        if (current->tty<0) {
                                iput(inode);
                                current->filp[fd]=NULL;
                                f->f_count=0;
                                return -EPERM;
                        }
/* Likewise with block-devices: check for floppy_change */
        if (S_ISBLK(inode->i_mode))
                check_disk_change(inode->i_zone[0]);
        f->f_mode = inode->i_mode;
        f->f_flags = flag;
        f->f_count = 1;
        f->f_inode = inode;
        f->f_pos = 0;
        return (fd);
}

int sys_creat(const char * pathname, int mode)
{
        return sys_open(pathname, O_CREAT | O_TRUNC, mode);
}

int sys_close(unsigned int fd)
{
        struct file * filp;

        if (fd >= NR_OPEN)
                return -EINVAL;
        current->close_on_exec &= ~(1<<fd);
        if (!(filp = current->filp[fd]))
                return -EINVAL;
        current->filp[fd] = NULL;
        if (filp->f_count == 0)
                panic("Close: file count is 0");
        if (--filp->f_count)
                return (0);
        iput(filp->f_inode);
```

```
        return (0);
}
```

```
/*
 *  linux/fs/pipe.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <signal.h>

#include <linux/sched.h>
#include <linux/mm.h>   /* for get_free_page */
#include <asm/segment.h>

int read_pipe(struct m_inode * inode, char * buf, int count)
{
        int chars, size, read = 0;

        while (count>0) {
                while (!(size=PIPE_SIZE(*inode))) {
                        wake_up(&inode->i_wait);
                        if (inode->i_count != 2) /* are there any writers? */
                                return read;
                        sleep_on(&inode->i_wait);
                }
                chars = PAGE_SIZE-PIPE_TAIL(*inode);
                if (chars > count)
                        chars = count;
                if (chars > size)
                        chars = size;
                count -= chars;
                read += chars;
                size = PIPE_TAIL(*inode);
                PIPE_TAIL(*inode) += chars;
                PIPE_TAIL(*inode) &= (PAGE_SIZE-1);
                while (chars-->0)
                        put_fs_byte(((char *)inode->i_size)[size++],buf++);
        }
        wake_up(&inode->i_wait);
        return read;
}

int write_pipe(struct m_inode * inode, char * buf, int count)
{
        int chars, size, written = 0;

        while (count>0) {
                while (!(size=(PAGE_SIZE-1)-PIPE_SIZE(*inode))) {
                        wake_up(&inode->i_wait);
                        if (inode->i_count != 2) { /* no readers */
                                current->signal |= (1<<(SIGPIPE-1));
                                return written?written:-1;
                        }
                        sleep_on(&inode->i_wait);
                }
                chars = PAGE_SIZE-PIPE_HEAD(*inode);
                if (chars > count)
                        chars = count;
                if (chars > size)
                        chars = size;
                count -= chars;
                written += chars;
                size = PIPE_HEAD(*inode);
                PIPE_HEAD(*inode) += chars;
                PIPE_HEAD(*inode) &= (PAGE_SIZE-1);
                while (chars-->0)
                        ((char *)inode->i_size)[size++]=get_fs_byte(buf++);
        }
        wake_up(&inode->i_wait);
        return written;
}
```

```
int sys_pipe(unsigned long * fildes)
{
        struct m_inode * inode;
        struct file * f[2];
        int fd[2];
        int i,j;

        j=0;
        for(i=0;j<2 && i<NR_FILE;i++)
                if (!file_table[i].f_count)
                        (f[j++]=i+file_table)->f_count++;
        if (j==1)
                f[0]->f_count=0;
        if (j<2)
                return -1;
        j=0;
        for(i=0;j<2 && i<NR_OPEN;i++)
                if (!current->filp[i]) {
                        current->filp[ fd[j]=i ] = f[j];
                        j++;
                }
        if (j==1)
                current->filp[fd[0]]=NULL;
        if (j<2) {
                f[0]->f_count=f[1]->f_count=0;
                return -1;
        }
        if (!(inode=get_pipe_inode())) {
                current->filp[fd[0]] =
                        current->filp[fd[1]] = NULL;
                f[0]->f_count = f[1]->f_count = 0;
                return -1;
        }
        f[0]->f_inode = f[1]->f_inode = inode;
        f[0]->f_pos = f[1]->f_pos = 0;
        f[0]->f_mode = 1;               /* read */
        f[1]->f_mode = 2;               /* write */
        put_fs_long(fd[0],0+fildes);
        put_fs_long(fd[1],1+fildes);
        return 0;
}
```

```
/*
 *  linux/fs/read_write.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <sys/stat.h>
#include <errno.h>
#include <sys/types.h>

#include <linux/kernel.h>
#include <linux/sched.h>
#include <asm/segment.h>

extern int rw_char(int rw,int dev, char * buf, int count, off_t * pos);
extern int read_pipe(struct m_inode * inode, char * buf, int count);
extern int write_pipe(struct m_inode * inode, char * buf, int count);
extern int block_read(int dev, off_t * pos, char * buf, int count);
extern int block_write(int dev, off_t * pos, char * buf, int count);
extern int file_read(struct m_inode * inode, struct file * filp,
                char * buf, int count);
extern int file_write(struct m_inode * inode, struct file * filp,
                char * buf, int count);

int sys_lseek(unsigned int fd,off_t offset, int origin)
{
        struct file * file;
        int tmp;

        if (fd >= NR_OPEN || !(file=current->filp[fd]) || !(file->f_inode)
           || !IS_SEEKABLE(MAJOR(file->f_inode->i_dev)))
                return -EBADF;
        if (file->f_inode->i_pipe)
                return -ESPIPE;
        switch (origin) {
                case 0:
                        if (offset<0) return -EINVAL;
                        file->f_pos=offset;
                        break;
                case 1:
                        if (file->f_pos+offset<0) return -EINVAL;
                        file->f_pos += offset;
                        break;
                case 2:
                        if ((tmp=file->f_inode->i_size+offset) < 0)
                                return -EINVAL;
                        file->f_pos = tmp;
                        break;
                default:
                        return -EINVAL;
        }
        return file->f_pos;
}

int sys_read(unsigned int fd,char * buf,int count)
{
        struct file * file;
        struct m_inode * inode;

        if (fd>=NR_OPEN || count<0 || !(file=current->filp[fd]))
                return -EINVAL;
        if (!count)
                return 0;
        verify_area(buf,count);
        inode = file->f_inode;
        if (inode->i_pipe)
                return (file->f_mode&1)?read_pipe(inode,buf,count):-EIO;
        if (S_ISCHR(inode->i_mode))
                return rw_char(READ,inode->i_zone[0],buf,count,&file->f_pos);
```

```
        if (S_ISBLK(inode->i_mode))
                return block_read(inode->i_zone[0],&file->f_pos,buf,count);
        if (S_ISDIR(inode->i_mode) || S_ISREG(inode->i_mode)) {
                if (count+file->f_pos > inode->i_size)
                        count = inode->i_size - file->f_pos;
                if (count<=0)
                        return 0;
                return file_read(inode,file,buf,count);
        }
        printk("(Read)inode->i_mode=%06o\n\r",inode->i_mode);
        return -EINVAL;
}

int sys_write(unsigned int fd,char * buf,int count)
{
        struct file * file;
        struct m_inode * inode;

        if (fd>=NR_OPEN || count <0 || !(file=current->filp[fd]))
                return -EINVAL;
        if (!count)
                return 0;
        inode=file->f_inode;
        if (inode->i_pipe)
                return (file->f_mode&2)?write_pipe(inode,buf,count):-EIO;
        if (S_ISCHR(inode->i_mode))
                return rw_char(WRITE,inode->i_zone[0],buf,count,&file->f_pos);
        if (S_ISBLK(inode->i_mode))
                return block_write(inode->i_zone[0],&file->f_pos,buf,count);
        if (S_ISREG(inode->i_mode))
                return file_write(inode,file,buf,count);
        printk("(Write)inode->i_mode=%06o\n\r",inode->i_mode);
        return -EINVAL;
}
```

```
/*
 *  linux/fs/stat.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <errno.h>
#include <sys/stat.h>

#include <linux/fs.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/segment.h>

static void cp_stat(struct m_inode * inode, struct stat * statbuf)
{
        struct stat tmp;
        int i;

        verify_area(statbuf,sizeof (* statbuf));
        tmp.st_dev = inode->i_dev;
        tmp.st_ino = inode->i_num;
        tmp.st_mode = inode->i_mode;
        tmp.st_nlink = inode->i_nlinks;
        tmp.st_uid = inode->i_uid;
        tmp.st_gid = inode->i_gid;
        tmp.st_rdev = inode->i_zone[0];
        tmp.st_size = inode->i_size;
        tmp.st_atime = inode->i_atime;
        tmp.st_mtime = inode->i_mtime;
        tmp.st_ctime = inode->i_ctime;
        for (i=0 ; i<sizeof (tmp) ; i++)
                put_fs_byte(((char *) &tmp)[i],&((char *) statbuf)[i]);
}

int sys_stat(char * filename, struct stat * statbuf)
{
        struct m_inode * inode;

        if (!(inode=namei(filename)))
                return -ENOENT;
        cp_stat(inode,statbuf);
        iput(inode);
        return 0;
}

int sys_fstat(unsigned int fd, struct stat * statbuf)
{
        struct file * f;
        struct m_inode * inode;

        if (fd >= NR_OPEN || !(f=current->filp[fd]) || !(inode=f->f_inode))
                return -EBADF;
        cp_stat(inode,statbuf);
        return 0;
}
```

```
/*
 *  linux/fs/super.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * super.c contains code to handle the super-block tables.
 */
#include <linux/config.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/system.h>

#include <errno.h>
#include <sys/stat.h>

int sync_dev(int dev);
void wait_for_keypress(void);

/* set_bit uses setb, as gas doesn't recognize setc */
#define set_bit(bitnr,addr) ({ \
register int __res __asm__("ax"); \
__asm__("bt %2,%3;setb %%al":"=a" (__res):"a" (0),"r" (bitnr),"m" (*(addr))); \
__res; })

struct super_block super_block[NR_SUPER];
/* this is initialized in init/main.c */
int ROOT_DEV = 0;

static void lock_super(struct super_block * sb)
{
        cli();
        while (sb->s_lock)
                sleep_on(&(sb->s_wait));
        sb->s_lock = 1;
        sti();
}

static void free_super(struct super_block * sb)
{
        cli();
        sb->s_lock = 0;
        wake_up(&(sb->s_wait));
        sti();
}

static void wait_on_super(struct super_block * sb)
{
        cli();
        while (sb->s_lock)
                sleep_on(&(sb->s_wait));
        sti();
}

struct super_block * get_super(int dev)
{
        struct super_block * s;

        if (!dev)
                return NULL;
        s = 0+super_block;
        while (s < NR_SUPER+super_block)
                if (s->s_dev == dev) {
                        wait_on_super(s);
                        if (s->s_dev == dev)
                                return s;
                        s = 0+super_block;
                } else
```

```
                        s++;
        return NULL;
}

void put_super(int dev)
{
        struct super_block * sb;
        struct m_inode * inode;
        int i;

        if (dev == ROOT_DEV) {
                printk("root diskette changed: prepare for armageddon\n\r");
                return;
        }
        if (!(sb = get_super(dev)))
                return;
        if (sb->s_imount) {
                printk("Mounted disk changed - tssk, tssk\n\r");
                return;
        }
        lock_super(sb);
        sb->s_dev = 0;
        for(i=0;i<I_MAP_SLOTS;i++)
                brelse(sb->s_imap[i]);
        for(i=0;i<Z_MAP_SLOTS;i++)
                brelse(sb->s_zmap[i]);
        free_super(sb);
        return;
}

static struct super_block * read_super(int dev)
{
        struct super_block * s;
        struct buffer_head * bh;
        int i,block;

        if (!dev)
                return NULL;
        check_disk_change(dev);
        if (s = get_super(dev))
                return s;
        for (s = 0+super_block ;; s++) {
                if (s >= NR_SUPER+super_block)
                        return NULL;
                if (!s->s_dev)
                        break;
        }
        s->s_dev = dev;
        s->s_isup = NULL;
        s->s_imount = NULL;
        s->s_time = 0;
        s->s_rd_only = 0;
        s->s_dirt = 0;
        lock_super(s);
        if (!(bh = bread(dev,1))) {
                s->s_dev=0;
                free_super(s);
                return NULL;
        }
        *((struct d_super_block *) s) =
                *((struct d_super_block *) bh->b_data);
        brelse(bh);
        if (s->s_magic != SUPER_MAGIC) {
                s->s_dev = 0;
                free_super(s);
                return NULL;
        }
        for (i=0;i<I_MAP_SLOTS;i++)
                s->s_imap[i] = NULL;
```

```
        for (i=0;i<Z_MAP_SLOTS;i++)
                s->s_zmap[i] = NULL;
        block=2;
        for (i=0 ; i < s->s_imap_blocks ; i++)
                if (s->s_imap[i]=bread(dev,block))
                        block++;
                else
                        break;
        for (i=0 ; i < s->s_zmap_blocks ; i++)
                if (s->s_zmap[i]=bread(dev,block))
                        block++;
                else
                        break;
        if (block != 2+s->s_imap_blocks+s->s_zmap_blocks) {
                for(i=0;i<I_MAP_SLOTS;i++)
                        brelse(s->s_imap[i]);
                for(i=0;i<Z_MAP_SLOTS;i++)
                        brelse(s->s_zmap[i]);
                s->s_dev=0;
                free_super(s);
                return NULL;
        }
        s->s_imap[0]->b_data[0] |= 1;
        s->s_zmap[0]->b_data[0] |= 1;
        free_super(s);
        return s;
}

int sys_umount(char * dev_name)
{
        struct m_inode * inode;
        struct super_block * sb;
        int dev;

        if (!(inode=namei(dev_name)))
                return -ENOENT;
        dev = inode->i_zone[0];
        if (!S_ISBLK(inode->i_mode)) {
                iput(inode);
                return -ENOTBLK;
        }
        iput(inode);
        if (dev==ROOT_DEV)
                return -EBUSY;
        if (!(sb=get_super(dev)) || !(sb->s_imount))
                return -ENOENT;
        if (!sb->s_imount->i_mount)
                printk("Mounted inode has i_mount=0\n");
        for (inode=inode_table+0 ; inode<inode_table+NR_INODE ; inode++)
                if (inode->i_dev==dev && inode->i_count)
                                return -EBUSY;
        sb->s_imount->i_mount=0;
        iput(sb->s_imount);
        sb->s_imount = NULL;
        iput(sb->s_isup);
        sb->s_isup = NULL;
        put_super(dev);
        sync_dev(dev);
        return 0;
}

int sys_mount(char * dev_name, char * dir_name, int rw_flag)
{
        struct m_inode * dev_i, * dir_i;
        struct super_block * sb;
        int dev;

        if (!(dev_i=namei(dev_name)))
                return -ENOENT;
```

```
        dev = dev_i->i_zone[0];
        if (!S_ISBLK(dev_i->i_mode)) {
                iput(dev_i);
                return -EPERM;
        }
        iput(dev_i);
        if (!(dir_i=namei(dir_name)))
                return -ENOENT;
        if (dir_i->i_count != 1 || dir_i->i_num == ROOT_INO) {
                iput(dir_i);
                return -EBUSY;
        }
        if (!S_ISDIR(dir_i->i_mode)) {
                iput(dir_i);
                return -EPERM;
        }
        if (!(sb=read_super(dev))) {
                iput(dir_i);
                return -EBUSY;
        }
        if (sb->s_imount) {
                iput(dir_i);
                return -EBUSY;
        }
        if (dir_i->i_mount) {
                iput(dir_i);
                return -EPERM;
        }
        sb->s_imount=dir_i;
        dir_i->i_mount=1;
        dir_i->i_dirt=1;                /* NOTE! we don't iput(dir_i) */
        return 0;                       /* we do that in umount */
}

void mount_root(void)
{
        int i,free;
        struct super_block * p;
        struct m_inode * mi;

        if (32 != sizeof (struct d_inode))
                panic("bad i-node size");
        for(i=0;i<NR_FILE;i++)
                file_table[i].f_count=0;
        if (MAJOR(ROOT_DEV) == 2) {
                printk("Insert root floppy and press ENTER");
                wait_for_keypress();
        }
        for(p = &super_block[0] ; p < &super_block[NR_SUPER] ; p++) {
                p->s_dev = 0;
                p->s_lock = 0;
                p->s_wait = NULL;
        }
        if (!(p=read_super(ROOT_DEV)))
                panic("Unable to mount root");
        if (!(mi=iget(ROOT_DEV,ROOT_INO)))
                panic("Unable to read root i-node");
        mi->i_count += 3 ;      /* NOTE! it is logically used 4 times, not 1 */
        p->s_isup = p->s_imount = mi;
        current->pwd = mi;
        current->root = mi;
        free=0;
        i=p->s_nzones;
        while (-- i >= 0)
                if (!set_bit(i&8191,p->s_zmap[i>>13]->b_data))
                        free++;
        printk("%d/%d free blocks\n\r",free,p->s_nzones);
        free=0;
        i=p->s_ninodes+1;
```

```
        while (-- i >= 0)
                if (!set_bit(i&8191,p->s_imap[i>>13]->b_data))
                        free++;
        printk("%d/%d free inodes\n\r",free,p->s_ninodes);
}
```

```
/*
 *  linux/fs/truncate.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <linux/sched.h>

#include <sys/stat.h>

static void free_ind(int dev,int block)
{
        struct buffer_head * bh;
        unsigned short * p;
        int i;

        if (!block)
                return;
        if (bh=bread(dev,block)) {
                p = (unsigned short *) bh->b_data;
                for (i=0;i<512;i++,p++)
                        if (*p)
                                free_block(dev,*p);
                brelse(bh);
        }
        free_block(dev,block);
}

static void free_dind(int dev,int block)
{
        struct buffer_head * bh;
        unsigned short * p;
        int i;

        if (!block)
                return;
        if (bh=bread(dev,block)) {
                p = (unsigned short *) bh->b_data;
                for (i=0;i<512;i++,p++)
                        if (*p)
                                free_ind(dev,*p);
                brelse(bh);
        }
        free_block(dev,block);
}

void truncate(struct m_inode * inode)
{
        int i;

        if (!(S_ISREG(inode->i_mode) || S_ISDIR(inode->i_mode)))
                return;
        for (i=0;i<7;i++)
                if (inode->i_zone[i]) {
                        free_block(inode->i_dev,inode->i_zone[i]);
                        inode->i_zone[i]=0;
                }
        free_ind(inode->i_dev,inode->i_zone[7]);
        free_dind(inode->i_dev,inode->i_zone[8]);
        inode->i_zone[7] = inode->i_zone[8] = 0;
        inode->i_size = 0;
        inode->i_dirt = 1;
        inode->i_mtime = inode->i_ctime = CURRENT_TIME;
}
```

```
#ifndef _A_OUT_H
#define _A_OUT_H

#define __GNU_EXEC_MACROS__

struct exec {
  unsigned long a_magic;        /* Use macros N_MAGIC, etc for access */
  unsigned a_text;              /* length of text, in bytes */
  unsigned a_data;              /* length of data, in bytes */
  unsigned a_bss;               /* length of uninitialized data area for file, in bytes
*/
  unsigned a_syms;              /* length of symbol table data in file, in bytes */
  unsigned a_entry;             /* start address */
  unsigned a_trsize;            /* length of relocation info for text, in bytes */
  unsigned a_drsize;            /* length of relocation info for data, in bytes */
};

#ifndef N_MAGIC
#define N_MAGIC(exec) ((exec).a_magic)
#endif

#ifndef OMAGIC
/* Code indicating object file or impure executable.  */
#define OMAGIC 0407
/* Code indicating pure executable.  */
#define NMAGIC 0410
/* Code indicating demand-paged executable.  */
#define ZMAGIC 0413
#endif /* not OMAGIC */

#ifndef N_BADMAG
#define N_BADMAG(x)                                     \
 (N_MAGIC(x) != OMAGIC && N_MAGIC(x) != NMAGIC          \
  && N_MAGIC(x) != ZMAGIC)
#endif

#define _N_BADMAG(x)                                    \
 (N_MAGIC(x) != OMAGIC && N_MAGIC(x) != NMAGIC          \
  && N_MAGIC(x) != ZMAGIC)

#define _N_HDROFF(x) (SEGMENT_SIZE - sizeof (struct exec))

#ifndef N_TXTOFF
#define N_TXTOFF(x) \
 (N_MAGIC(x) == ZMAGIC ? _N_HDROFF((x)) + sizeof (struct exec) : sizeof (struct exec))
#endif

#ifndef N_DATOFF
#define N_DATOFF(x) (N_TXTOFF(x) + (x).a_text)
#endif

#ifndef N_TRELOFF
#define N_TRELOFF(x) (N_DATOFF(x) + (x).a_data)
#endif

#ifndef N_DRELOFF
#define N_DRELOFF(x) (N_TRELOFF(x) + (x).a_trsize)
#endif

#ifndef N_SYMOFF
#define N_SYMOFF(x) (N_DRELOFF(x) + (x).a_drsize)
#endif

#ifndef N_STROFF
#define N_STROFF(x) (N_SYMOFF(x) + (x).a_syms)
#endif

/* Address of text segment in memory after it is loaded.  */
#ifndef N_TXTADDR
```

```
#define N_TXTADDR(x) 0
#endif

/* Address of data segment in memory after it is loaded.
   Note that it is up to you to define SEGMENT_SIZE
   on machines not listed here.  */
#if defined(vax) || defined(hp300) || defined(pyr)
#define SEGMENT_SIZE PAGE_SIZE
#endif
#ifdef  hp300
#define PAGE_SIZE       4096
#endif
#ifdef  sony
#define SEGMENT_SIZE    0x2000
#endif  /* Sony.  */
#ifdef is68k
#define SEGMENT_SIZE 0x20000
#endif
#if defined(m68k) && defined(PORTAR)
#define PAGE_SIZE 0x400
#define SEGMENT_SIZE PAGE_SIZE
#endif

#define PAGE_SIZE 4096
#define SEGMENT_SIZE 1024

#define _N_SEGMENT_ROUND(x) (((x) + SEGMENT_SIZE - 1) & ~(SEGMENT_SIZE - 1))

#define _N_TXTENDADDR(x) (N_TXTADDR(x)+(x).a_text)

#ifndef N_DATADDR
#define N_DATADDR(x) \
    (N_MAGIC(x)==OMAGIC? (_N_TXTENDADDR(x)) \
     : (_N_SEGMENT_ROUND (_N_TXTENDADDR(x))))
#endif

/* Address of bss segment in memory after it is loaded.  */
#ifndef N_BSSADDR
#define N_BSSADDR(x) (N_DATADDR(x) + (x).a_data)
#endif

#ifndef N_NLIST_DECLARED
struct nlist {
  union {
    char *n_name;
    struct nlist *n_next;
    long n_strx;
  } n_un;
  unsigned char n_type;
  char n_other;
  short n_desc;
  unsigned long n_value;
};
#endif

#ifndef N_UNDF
#define N_UNDF 0
#endif
#ifndef N_ABS
#define N_ABS 2
#endif
#ifndef N_TEXT
#define N_TEXT 4
#endif
#ifndef N_DATA
#define N_DATA 6
#endif
#ifndef N_BSS
#define N_BSS 8
```

```
#endif
#ifndef N_COMM
#define N_COMM 18
#endif
#ifndef N_FN
#define N_FN 15
#endif

#ifndef N_EXT
#define N_EXT 1
#endif
#ifndef N_TYPE
#define N_TYPE 036
#endif
#ifndef N_STAB
#define N_STAB 0340
#endif

/* The following type indicates the definition of a symbol as being
   an indirect reference to another symbol.  The other symbol
   appears as an undefined reference, immediately following this symbol.

   Indirection is asymmetrical.  The other symbol's value will be used
   to satisfy requests for the indirect symbol, but not vice versa.
   If the other symbol does not have a definition, libraries will
   be searched to find a definition.  */
#define N_INDR 0xa

/* The following symbols refer to set elements.
   All the N_SET[ATDB] symbols with the same name form one set.
   Space is allocated for the set in the text section, and each set
   element's value is stored into one word of the space.
   The first word of the space is the length of the set (number of elements).

   The address of the set is made into an N_SETV symbol
   whose name is the same as the name of the set.
   This symbol acts like a N_DATA global symbol
   in that it can satisfy undefined external references.  */

/* These appear as input to LD, in a .o file.  */
#define N_SETA  0x14            /* Absolute set element symbol */
#define N_SETT  0x16            /* Text set element symbol */
#define N_SETD  0x18            /* Data set element symbol */
#define N_SETB  0x1A            /* Bss set element symbol */

/* This is output from LD.  */
#define N_SETV  0x1C            /* Pointer to set vector in data area.  */

#ifndef N_RELOCATION_INFO_DECLARED

/* This structure describes a single relocation to be performed.
   The text-relocation section of the file is a vector of these structures,
   all of which apply to the text section.
   Likewise, the data-relocation section applies to the data section.  */

struct relocation_info
{
  /* Address (within segment) to be relocated.  */
  int r_address;
  /* The meaning of r_symbolnum depends on r_extern.  */
  unsigned int r_symbolnum:24;
  /* Nonzero means value is a pc-relative offset
     and it should be relocated for changes in its own address
     as well as for changes in the symbol or section specified.  */
  unsigned int r_pcrel:1;
  /* Length (as exponent of 2) of the field to be relocated.
     Thus, a value of 2 indicates 1<<2 bytes.  */
  unsigned int r_length:2;
  /* 1 => relocate with value of symbol.
```

```
           r_symbolnum is the index of the symbol
           in file's the symbol table.
      0 => relocate with the address of a segment.
           r_symbolnum is N_TEXT, N_DATA, N_BSS or N_ABS
           (the N_EXT bit may be set also, but signifies nothing).  */
  unsigned int r_extern:1;
  /* Four bits that aren't used, but when writing an object file
     it is desirable to clear them.  */
  unsigned int r_pad:4;
};
#endif /* no N_RELOCATION_INFO_DECLARED.  */


#endif /* __A_OUT_GNU_H__ */
```

```
#ifndef _CONST_H
#define _CONST_H

#define BUFFER_END 0x200000

#define I_TYPE          0170000
#define I_DIRECTORY     0040000
#define I_REGULAR       0100000
#define I_BLOCK_SPECIAL 0060000
#define I_CHAR_SPECIAL  0020000
#define I_NAMED_PIPE    0010000
#define I_SET_UID_BIT   0004000
#define I_SET_GID_BIT   0002000

#endif
```

```
#ifndef _CTYPE_H
#define _CTYPE_H

#define _U      0x01    /* upper */
#define _L      0x02    /* lower */
#define _D      0x04    /* digit */
#define _C      0x08    /* cntrl */
#define _P      0x10    /* punct */
#define _S      0x20    /* white space (space/lf/tab) */
#define _X      0x40    /* hex digit */
#define _SP     0x80    /* hard space (0x20) */

extern unsigned char _ctype[];
extern char _ctmp;

#define isalnum(c) ((_ctype+1)[c]&(_U|_L|_D))
#define isalpha(c) ((_ctype+1)[c]&(_U|_L))
#define iscntrl(c) ((_ctype+1)[c]&(_C))
#define isdigit(c) ((_ctype+1)[c]&(_D))
#define isgraph(c) ((_ctype+1)[c]&(_P|_U|_L|_D))
#define islower(c) ((_ctype+1)[c]&(_L))
#define isprint(c) ((_ctype+1)[c]&(_P|_U|_L|_D|_SP))
#define ispunct(c) ((_ctype+1)[c]&(_P))
#define isspace(c) ((_ctype+1)[c]&(_S))
#define isupper(c) ((_ctype+1)[c]&(_U))
#define isxdigit(c) ((_ctype+1)[c]&(_D|_X))

#define isascii(c) (((unsigned) c)<=0x7f)
#define toascii(c) (((unsigned) c)&0x7f)

#define tolower(c) (_ctmp=c,isupper(_ctmp)?_ctmp-('A'-'a'):_ctmp)
#define toupper(c) (_ctmp=c,islower(_ctmp)?_ctmp-('a'-'A'):_ctmp)

#endif
```

```
#ifndef _ERRNO_H
#define _ERRNO_H

/*
 * ok, as I hadn't got any other source of information about
 * possible error numbers, I was forced to use the same numbers
 * as minix.
 * Hopefully these are posix or something. I wouldn't know (and posix
 * isn't telling me - they want $$$ for their f***ing standard).
 *
 * We don't use the _SIGN cludge of minix, so kernel returns must
 * see to the sign by themselves.
 *
 * NOTE! Remember to change strerror() if you change this file!
 */

extern int errno;

#define ERROR           99
#define EPERM            1
#define ENOENT           2
#define ESRCH            3
#define EINTR            4
#define EIO              5
#define ENXIO            6
#define E2BIG            7
#define ENOEXEC          8
#define EBADF            9
#define ECHILD          10
#define EAGAIN          11
#define ENOMEM          12
#define EACCES          13
#define EFAULT          14
#define ENOTBLK         15
#define EBUSY           16
#define EEXIST          17
#define EXDEV           18
#define ENODEV          19
#define ENOTDIR         20
#define EISDIR          21
#define EINVAL          22
#define ENFILE          23
#define EMFILE          24
#define ENOTTY          25
#define ETXTBSY         26
#define EFBIG           27
#define ENOSPC          28
#define ESPIPE          29
#define EROFS           30
#define EMLINK          31
#define EPIPE           32
#define EDOM            33
#define ERANGE          34
#define EDEADLK         35
#define ENAMETOOLONG    36
#define ENOLCK          37
#define ENOSYS          38
#define ENOTEMPTY       39

#endif
```

```
#ifndef _FCNTL_H
#define _FCNTL_H

#include <sys/types.h>

/* open/fcntl - NOCTTY, NDELAY isn't implemented yet */
#define O_ACCMODE       00003
#define O_RDONLY           00
#define O_WRONLY           01
#define O_RDWR             02
#define O_CREAT         00100   /* not fcntl */
#define O_EXCL          00200   /* not fcntl */
#define O_NOCTTY        00400   /* not fcntl */
#define O_TRUNC         01000   /* not fcntl */
#define O_APPEND        02000
#define O_NONBLOCK      04000   /* not fcntl */
#define O_NDELAY        O_NONBLOCK

/* Defines for fcntl-commands. Note that currently
 * locking isn't supported, and other things aren't really
 * tested.
 */
#define F_DUPFD         0       /* dup */
#define F_GETFD         1       /* get f_flags */
#define F_SETFD         2       /* set f_flags */
#define F_GETFL         3       /* more flags (cloexec) */
#define F_SETFL         4
#define F_GETLK         5       /* not implemented */
#define F_SETLK         6
#define F_SETLKW        7

/* for F_[GET|SET]FL */
#define FD_CLOEXEC      1       /* actually anything with low bit set goes */

/* Ok, these are locking features, and aren't implemented at any
 * level. POSIX wants them.
 */
#define F_RDLCK         0
#define F_WRLCK         1
#define F_UNLCK         2

/* Once again - not implemented, but ... */
struct flock {
        short l_type;
        short l_whence;
        off_t l_start;
        off_t l_len;
        pid_t l_pid;
};

extern int creat(const char * filename,mode_t mode);
extern int fcntl(int fildes,int cmd, ...);
extern int open(const char * filename, int flags, ...);

#endif
```

```
#ifndef _SIGNAL_H
#define _SIGNAL_H

#include <sys/types.h>

typedef int sig_atomic_t;
typedef unsigned int sigset_t;          /* 32 bits */

#define _NSIG             32
#define NSIG            _NSIG

#define SIGHUP           1
#define SIGINT           2
#define SIGQUIT          3
#define SIGILL           4
#define SIGTRAP          5
#define SIGABRT          6
#define SIGIOT           6
#define SIGUNUSED        7
#define SIGFPE           8
#define SIGKILL          9
#define SIGUSR1         10
#define SIGSEGV         11
#define SIGUSR2         12
#define SIGPIPE         13
#define SIGALRM         14
#define SIGTERM         15
#define SIGSTKFLT       16
#define SIGCHLD         17
#define SIGCONT         18
#define SIGSTOP         19
#define SIGTSTP         20
#define SIGTTIN         21
#define SIGTTOU         22

/* Ok, I haven't implemented sigactions, but trying to keep headers POSIX */
#define SA_NOCLDSTOP    1
#define SA_NOMASK       0x40000000
#define SA_ONESHOT      0x80000000

#define SIG_BLOCK          0    /* for blocking signals */
#define SIG_UNBLOCK        1    /* for unblocking signals */
#define SIG_SETMASK        2    /* for setting the signal mask */

#define SIG_DFL         ((void (*)(int))0)      /* default signal handling */
#define SIG_IGN         ((void (*)(int))1)      /* ignore signal */

struct sigaction {
        void (*sa_handler)(int);
        sigset_t sa_mask;
        int sa_flags;
        void (*sa_restorer)(void);
};

void (*signal(int _sig, void (*_func)(int)))(int);
int raise(int sig);
int kill(pid_t pid, int sig);
int sigaddset(sigset_t *mask, int signo);
int sigdelset(sigset_t *mask, int signo);
int sigemptyset(sigset_t *mask);
int sigfillset(sigset_t *mask);
int sigismember(sigset_t *mask, int signo); /* 1 - is, 0 - not, -1 error */
int sigpending(sigset_t *set);
int sigprocmask(int how, sigset_t *set, sigset_t *oldset);
int sigsuspend(sigset_t *sigmask);
int sigaction(int sig, struct sigaction *act, struct sigaction *oldact);

#endif /* _SIGNAL_H */
```

```
#ifndef _STDARG_H
#define _STDARG_H

typedef char *va_list;

/* Amount of space required in an argument list for an arg of type TYPE.
   TYPE may alternatively be an expression whose type is used.  */

#define __va_rounded_size(TYPE)  \
  (((sizeof (TYPE) + sizeof (int) - 1) / sizeof (int)) * sizeof (int))

#ifndef __sparc__
#define va_start(AP, LASTARG)                                           \
 (AP = ((char *) &(LASTARG) + __va_rounded_size (LASTARG)))
#else
#define va_start(AP, LASTARG)                                           \
 (__builtin_saveregs (),                                                \
  AP = ((char *) &(LASTARG) + __va_rounded_size (LASTARG)))
#endif

void va_end (va_list);          /* Defined in gnulib */
#define va_end(AP)

#define va_arg(AP, TYPE)                                                \
 (AP += __va_rounded_size (TYPE),                                       \
  *((TYPE *) (AP - __va_rounded_size (TYPE))))

#endif /* _STDARG_H */
```

```
#ifndef _STDDEF_H
#define _STDDEF_H

#ifndef _PTRDIFF_T
#define _PTRDIFF_T
typedef long ptrdiff_t;
#endif

#ifndef _SIZE_T
#define _SIZE_T
typedef unsigned long size_t;
#endif

#undef NULL
#define NULL ((void *)0)

#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *)0)->MEMBER)

#endif
```

```
#ifndef _STRING_H_
#define _STRING_H_

#ifndef NULL
#define NULL ((void *) 0)
#endif

#ifndef _SIZE_T
#define _SIZE_T
typedef unsigned int size_t;
#endif

extern char * strerror(int errno);

/*
 * This string-include defines all string functions as inline
 * functions. Use gcc. It also assumes ds=es=data space, this should be
 * normal. Most of the string-functions are rather heavily hand-optimized,
 * see especially strtok,strstr,str[c]spn. They should work, but are not
 * very easy to understand. Everything is done entirely within the register
 * set, making the functions fast and clean. String instructions have been
 * used through-out, making for "slightly" unclear code :-)
 *
 *              (C) 1991 Linus Torvalds
 */

extern inline char * strcpy(char * dest,const char *src)
{
__asm__("cld\n"
        "1:\tlodsb\n\t"
        "stosb\n\t"
        "testb %%al,%%al\n\t"
        "jne 1b"
        ::"S" (src),"D" (dest):"si","di","ax");
return dest;
}

extern inline char * strncpy(char * dest,const char *src,int count)
{
__asm__("cld\n"
        "1:\tdecl %2\n\t"
        "js 2f\n\t"
        "lodsb\n\t"
        "stosb\n\t"
        "testb %%al,%%al\n\t"
        "jne 1b\n\t"
        "rep\n\t"
        "stosb\n"
        "2:"
        ::"S" (src),"D" (dest),"c" (count):"si","di","ax","cx");
return dest;
}

extern inline char * strcat(char * dest,const char * src)
{
__asm__("cld\n\t"
        "repne\n\t"
        "scasb\n\t"
        "decl %1\n"
        "1:\tlodsb\n\t"
        "stosb\n\t"
        "testb %%al,%%al\n\t"
        "jne 1b"
        ::"S" (src),"D" (dest),"a" (0),"c" (0xffffffff):"si","di","ax","cx");
return dest;
}

extern inline char * strncat(char * dest,const char * src,int count)
{
```

```
__asm__("cld\n\t"
        "repne\n\t"
        "scasb\n\t"
        "decl %1\n\t"
        "movl %4,%3\n"
        "1:\tdecl %3\n\t"
        "js 2f\n\t"
        "lodsb\n\t"
        "stosb\n\t"
        "testb %%al,%%al\n\t"
        "jne 1b\n"
        "2:\txorl %2,%2\n\t"
        "stosb"
        ::"S" (src),"D" (dest),"a" (0),"c" (0xffffffff),"g" (count)
        :"si","di","ax","cx");
return dest;
}

extern inline int strcmp(const char * cs,const char * ct)
{
register int __res __asm__("ax");
__asm__("cld\n"
        "1:\tlodsb\n\t"
        "scasb\n\t"
        "jne 2f\n\t"
        "testb %%al,%%al\n\t"
        "jne 1b\n\t"
        "xorl %%eax,%%eax\n\t"
        "jmp 3f\n"
        "2:\tmovl $1,%%eax\n\t"
        "jl 3f\n\t"
        "negl %%eax\n"
        "3:"
        :"=a" (__res):"D" (cs),"S" (ct):"si","di");
return __res;
}

extern inline int strncmp(const char * cs,const char * ct,int count)
{
register int __res __asm__("ax");
__asm__("cld\n"
        "1:\tdecl %3\n\t"
        "js 2f\n\t"
        "lodsb\n\t"
        "scasb\n\t"
        "jne 3f\n\t"
        "testb %%al,%%al\n\t"
        "jne 1b\n"
        "2:\txorl %%eax,%%eax\n\t"
        "jmp 4f\n"
        "3:\tmovl $1,%%eax\n\t"
        "jl 4f\n\t"
        "negl %%eax\n"
        "4:"
        :"=a" (__res):"D" (cs),"S" (ct),"c" (count):"si","di","cx");
return __res;
}

extern inline char * strchr(const char * s,char c)
{
register char * __res __asm__("ax");
__asm__("cld\n\t"
        "movb %%al,%%ah\n"
        "1:\tlodsb\n\t"
        "cmpb %%ah,%%al\n\t"
        "je 2f\n\t"
        "testb %%al,%%al\n\t"
        "jne 1b\n\t"
        "movl $1,%1\n"
```

```
        "2:\tmovl %1,%0\n\t"
        "decl %0"
        :"=a" (__res):"S" (s),"0" (c):"si");
return __res;
}

extern inline char * strrchr(const char * s,char c)
{
register char * __res __asm__("dx");
__asm__("cld\n\t"
        "movb %%al,%%ah\n"
        "1:\tlodsb\n\t"
        "cmpb %%ah,%%al\n\t"
        "jne 2f\n\t"
        "movl %%esi,%0\n\t"
        "decl %0\n"
        "2:\ttestb %%al,%%al\n\t"
        "jne 1b"
        :"=d" (__res):"0" (0),"S" (s),"a" (c):"ax","si");
return __res;
}

extern inline int strspn(const char * cs, const char * ct)
{
register char * __res __asm__("si");
__asm__("cld\n\t"
        "movl %4,%%edi\n\t"
        "repne\n\t"
        "scasb\n\t"
        "notl %%ecx\n\t"
        "decl %%ecx\n\t"
        "movl %%ecx,%%edx\n"
        "1:\tlodsb\n\t"
        "testb %%al,%%al\n\t"
        "je 2f\n\t"
        "movl %4,%%edi\n\t"
        "movl %%edx,%%ecx\n\t"
        "repne\n\t"
        "scasb\n\t"
        "je 1b\n"
        "2:\tdecl %0"
        :"=S" (__res):"a" (0),"c" (0xffffffff),"0" (cs),"g" (ct)
        :"ax","cx","dx","di");
return __res-cs;
}

extern inline int strcspn(const char * cs, const char * ct)
{
register char * __res __asm__("si");
__asm__("cld\n\t"
        "movl %4,%%edi\n\t"
        "repne\n\t"
        "scasb\n\t"
        "notl %%ecx\n\t"
        "decl %%ecx\n\t"
        "movl %%ecx,%%edx\n"
        "1:\tlodsb\n\t"
        "testb %%al,%%al\n\t"
        "je 2f\n\t"
        "movl %4,%%edi\n\t"
        "movl %%edx,%%ecx\n\t"
        "repne\n\t"
        "scasb\n\t"
        "jne 1b\n"
        "2:\tdecl %0"
        :"=S" (__res):"a" (0),"c" (0xffffffff),"0" (cs),"g" (ct)
        :"ax","cx","dx","di");
return __res-cs;
}
```

```
extern inline char * strpbrk(const char * cs,const char * ct)
{
register char * __res __asm__("si");
__asm__("cld\n\t"
        "movl %4,%%edi\n\t"
        "repne\n\t"
        "scasb\n\t"
        "notl %%ecx\n\t"
        "decl %%ecx\n\t"
        "movl %%ecx,%%edx\n"
        "1:\tlodsb\n\t"
        "testb %%al,%%al\n\t"
        "je 2f\n\t"
        "movl %4,%%edi\n\t"
        "movl %%edx,%%ecx\n\t"
        "repne\n\t"
        "scasb\n\t"
        "jne 1b\n\t"
        "decl %0\n\t"
        "jmp 3f\n"
        "2:\txorl %0,%0\n"
        "3:"
        :"=S" (__res):"a" (0),"c" (0xffffffff),"0" (cs),"g" (ct)
        :"ax","cx","dx","di");
return __res;
}

extern inline char * strstr(const char * cs,const char * ct)
{
register char * __res __asm__("ax");
__asm__("cld\n\t" \
        "movl %4,%%edi\n\t"
        "repne\n\t"
        "scasb\n\t"
        "notl %%ecx\n\t"
        "decl %%ecx\n\t"        /* NOTE! This also sets Z if searchstring='' */
        "movl %%ecx,%%edx\n"
        "1:\tmovl %4,%%edi\n\t"
        "movl %%esi,%%eax\n\t"
        "movl %%edx,%%ecx\n\t"
        "repe\n\t"
        "cmpsb\n\t"
        "je 2f\n\t"             /* also works for empty string, see above */
        "xchgl %%eax,%%esi\n\t"
        "incl %%esi\n\t"
        "cmpb $0,-1(%%eax)\n\t"
        "jne 1b\n\t"
        "xorl %%eax,%%eax\n\t"
        "2:"
        :"=a" (__res):"0" (0),"c" (0xffffffff),"S" (cs),"g" (ct)
        :"cx","dx","di","si");
return __res;
}

extern inline int strlen(const char * s)
{
register int __res __asm__("cx");
__asm__("cld\n\t"
        "repne\n\t"
        "scasb\n\t"
        "notl %0\n\t"
        "decl %0"
        :"=c" (__res):"D" (s),"a" (0),"0" (0xffffffff):"di");
return __res;
}

extern char * ___strtok;
```

```
extern inline char * strtok(char * s,const char * ct)
{
register char * __res __asm__("si");
__asm__("testl %1,%1\n\t"
        "jne 1f\n\t"
        "testl %0,%0\n\t"
        "je 8f\n\t"
        "movl %0,%1\n"
        "1:\txorl %0,%0\n\t"
        "movl $-1,%%ecx\n\t"
        "xorl %%eax,%%eax\n\t"
        "cld\n\t"
        "movl %4,%%edi\n\t"
        "repne\n\t"
        "scasb\n\t"
        "notl %%ecx\n\t"
        "decl %%ecx\n\t"
        "je 7f\n\t"                     /* empty delimeter-string */
        "movl %%ecx,%%edx\n"
        "2:\tlodsb\n\t"
        "testb %%al,%%al\n\t"
        "je 7f\n\t"
        "movl %4,%%edi\n\t"
        "movl %%edx,%%ecx\n\t"
        "repne\n\t"
        "scasb\n\t"
        "je 2b\n\t"
        "decl %1\n\t"
        "cmpb $0,(%1)\n\t"
        "je 7f\n\t"
        "movl %1,%0\n"
        "3:\tlodsb\n\t"
        "testb %%al,%%al\n\t"
        "je 5f\n\t"
        "movl %4,%%edi\n\t"
        "movl %%edx,%%ecx\n\t"
        "repne\n\t"
        "scasb\n\t"
        "jne 3b\n\t"
        "decl %1\n\t"
        "cmpb $0,(%1)\n\t"
        "je 5f\n\t"
        "movb $0,(%1)\n\t"
        "incl %1\n\t"
        "jmp 6f\n"
        "5:\txorl %1,%1\n"
        "6:\tcmpb $0,(%0)\n\t"
        "jne 7f\n\t"
        "xorl %0,%0\n"
        "7:\ttestl %0,%0\n\t"
        "jne 8f\n\t"
        "movl %0,%1\n"
        "8:"
        :"=b" (__res),"=S" (___strtok)
        :"0" (___strtok),"1" (s),"g" (ct)
        :"ax","cx","dx","di");
return __res;
}

extern inline void * memcpy(void * dest,const void * src, int n)
{
__asm__("cld\n\t"
        "rep\n\t"
        "movsb"
        ::"c" (n),"S" (src),"D" (dest)
        :"cx","si","di");
return dest;
}
```

```
extern inline void * memmove(void * dest,const void * src, int n)
{
if (dest<src)
__asm__("cld\n\t"
        "rep\n\t"
        "movsb"
        ::"c" (n),"S" (src),"D" (dest)
        :"cx","si","di");
else
__asm__("std\n\t"
        "rep\n\t"
        "movsb"
        ::"c" (n),"S" (src+n-1),"D" (dest+n-1)
        :"cx","si","di");
return dest;
}

extern inline int memcmp(const void * cs,const void * ct,int count)
{
register int __res __asm__("ax");
__asm__("cld\n\t"
        "repe\n\t"
        "cmpsb\n\t"
        "je 1f\n\t"
        "movl $1,%%eax\n\t"
        "jl 1f\n\t"
        "negl %%eax\n"
        "1:"
        :"=a" (__res):"0" (0),"D" (cs),"S" (ct),"c" (count)
        :"si","di","cx");
return __res;
}

extern inline void * memchr(const void * cs,char c,int count)
{
register void * __res __asm__("di");
if (!count)
        return NULL;
__asm__("cld\n\t"
        "repne\n\t"
        "scasb\n\t"
        "je 1f\n\t"
        "movl $1,%0\n"
        "1:\tdecl %0"
        :"=D" (__res):"a" (c),"D" (cs),"c" (count)
        :"cx");
return __res;
}

extern inline void * memset(void * s,char c,int count)
{
__asm__("cld\n\t"
        "rep\n\t"
        "stosb"
        ::"a" (c),"D" (s),"c" (count)
        :"cx","di");
return s;
}

#endif
```

```
#ifndef _TERMIOS_H
#define _TERMIOS_H

#define TTY_BUF_SIZE 1024

/* 0x54 is just a magic number to make these relatively uniqe ('T') */

#define TCGETS          0x5401
#define TCSETS          0x5402
#define TCSETSW         0x5403
#define TCSETSF         0x5404
#define TCGETA          0x5405
#define TCSETA          0x5406
#define TCSETAW         0x5407
#define TCSETAF         0x5408
#define TCSBRK          0x5409
#define TCXONC          0x540A
#define TCFLSH          0x540B
#define TIOCEXCL        0x540C
#define TIOCNXCL        0x540D
#define TIOCSCTTY       0x540E
#define TIOCGPGRP       0x540F
#define TIOCSPGRP       0x5410
#define TIOCOUTQ        0x5411
#define TIOCSTI         0x5412
#define TIOCGWINSZ      0x5413
#define TIOCSWINSZ      0x5414
#define TIOCMGET        0x5415
#define TIOCMBIS        0x5416
#define TIOCMBIC        0x5417
#define TIOCMSET        0x5418
#define TIOCGSOFTCAR    0x5419
#define TIOCSSOFTCAR    0x541A
#define TIOCINQ         0x541B

struct winsize {
        unsigned short ws_row;
        unsigned short ws_col;
        unsigned short ws_xpixel;
        unsigned short ws_ypixel;
};

#define NCC 8
struct termio {
        unsigned short c_iflag;         /* input mode flags */
        unsigned short c_oflag;         /* output mode flags */
        unsigned short c_cflag;         /* control mode flags */
        unsigned short c_lflag;         /* local mode flags */
        unsigned char c_line;           /* line discipline */
        unsigned char c_cc[NCC];        /* control characters */
};

#define NCCS 17
struct termios {
        unsigned long c_iflag;          /* input mode flags */
        unsigned long c_oflag;          /* output mode flags */
        unsigned long c_cflag;          /* control mode flags */
        unsigned long c_lflag;          /* local mode flags */
        unsigned char c_line;           /* line discipline */
        unsigned char c_cc[NCCS];       /* control characters */
};

/* c_cc characters */
#define VINTR 0
#define VQUIT 1
#define VERASE 2
#define VKILL 3
#define VEOF 4
#define VTIME 5
```

```
#define VMIN 6
#define VSWTC 7
#define VSTART 8
#define VSTOP 9
#define VSUSP 10
#define VEOL 11
#define VREPRINT 12
#define VDISCARD 13
#define VWERASE 14
#define VLNEXT 15
#define VEOL2 16

/* c_iflag bits */
#define IGNBRK  0000001
#define BRKINT  0000002
#define IGNPAR  0000004
#define PARMRK  0000010
#define INPCK   0000020
#define ISTRIP  0000040
#define INLCR   0000100
#define IGNCR   0000200
#define ICRNL   0000400
#define IUCLC   0001000
#define IXON    0002000
#define IXANY   0004000
#define IXOFF   0010000
#define IMAXBEL 0020000

/* c_oflag bits */
#define OPOST   0000001
#define OLCUC   0000002
#define ONLCR   0000004
#define OCRNL   0000010
#define ONOCR   0000020
#define ONLRET  0000040
#define OFILL   0000100
#define OFDEL   0000200
#define NLDLY   0000400
#define   NL0   0000000
#define   NL1   0000400
#define CRDLY   0003000
#define   CR0   0000000
#define   CR1   0001000
#define   CR2   0002000
#define   CR3   0003000
#define TABDLY  0014000
#define   TAB0  0000000
#define   TAB1  0004000
#define   TAB2  0010000
#define   TAB3  0014000
#define   XTABS 0014000
#define BSDLY   0020000
#define   BS0   0000000
#define   BS1   0020000
#define VTDLY   0040000
#define   VT0   0000000
#define   VT1   0040000
#define FFDLY   0040000
#define   FF0   0000000
#define   FF1   0040000

/* c_cflag bit meaning */
#define CBAUD   0000017
#define  B0     0000000         /* hang up */
#define  B50    0000001
#define  B75    0000002
#define  B110   0000003
#define  B134   0000004
#define  B150   0000005
```

```
#define  B200   0000006
#define  B300   0000007
#define  B600   0000010
#define  B1200  0000011
#define  B1800  0000012
#define  B2400  0000013
#define  B4800  0000014
#define  B9600  0000015
#define  B19200 0000016
#define  B38400 0000017
#define EXTA B19200
#define EXTB B38400
#define CSIZE   0000060
#define   CS5   0000000
#define   CS6   0000020
#define   CS7   0000040
#define   CS8   0000060
#define CSTOPB  0000100
#define CREAD   0000200
#define CPARENB 0000400
#define CPARODD 0001000
#define HUPCL   0002000
#define CLOCAL  0004000
#define CIBAUD  03600000               /* input baud rate (not used) */
#define CRTSCTS 020000000000           /* flow control */

#define PARENB CPARENB
#define PARODD CPARODD

/* c_lflag bits */
#define ISIG    0000001
#define ICANON  0000002
#define XCASE   0000004
#define ECHO    0000010
#define ECHOE   0000020
#define ECHOK   0000040
#define ECHONL  0000100
#define NOFLSH  0000200
#define TOSTOP  0000400
#define ECHOCTL 0001000
#define ECHOPRT 0002000
#define ECHOKE  0004000
#define FLUSHO  0010000
#define PENDIN  0040000
#define IEXTEN  0100000

/* modem lines */
#define TIOCM_LE        0x001
#define TIOCM_DTR       0x002
#define TIOCM_RTS       0x004
#define TIOCM_ST        0x008
#define TIOCM_SR        0x010
#define TIOCM_CTS       0x020
#define TIOCM_CAR       0x040
#define TIOCM_RNG       0x080
#define TIOCM_DSR       0x100
#define TIOCM_CD        TIOCM_CAR
#define TIOCM_RI        TIOCM_RNG

/* tcflow() and TCXONC use these */
#define TCOOFF          0
#define TCOON           1
#define TCIOFF          2
#define TCION           3

/* tcflush() and TCFLSH use these */
#define TCIFLUSH        0
#define TCOFLUSH        1
#define TCIOFLUSH       2
```

```
/* tcsetattr uses these */
#define TCSANOW         0
#define TCSADRAIN       1
#define TCSAFLUSH       2

typedef int speed_t;

extern speed_t cfgetispeed(struct termios *termios_p);
extern speed_t cfgetospeed(struct termios *termios_p);
extern int cfsetispeed(struct termios *termios_p, speed_t speed);
extern int cfsetospeed(struct termios *termios_p, speed_t speed);
extern int tcdrain(int fildes);
extern int tcflow(int fildes, int action);
extern int tcflush(int fildes, int queue_selector);
extern int tcgetattr(int fildes, struct termios *termios_p);
extern int tcsendbreak(int fildes, int duration);
extern int tcsetattr(int fildes, int optional_actions,
        struct termios *termios_p);

#endif
```

```
#ifndef _TIME_H
#define _TIME_H

#ifndef _TIME_T
#define _TIME_T
typedef long time_t;
#endif

#ifndef _SIZE_T
#define _SIZE_T
typedef unsigned int size_t;
#endif

#define CLOCKS_PER_SEC 100

typedef long clock_t;

struct tm {
        int tm_sec;
        int tm_min;
        int tm_hour;
        int tm_mday;
        int tm_mon;
        int tm_year;
        int tm_wday;
        int tm_yday;
        int tm_isdst;
};

clock_t clock(void);
time_t time(time_t * tp);
double difftime(time_t time2, time_t time1);
time_t mktime(struct tm * tp);

char * asctime(const struct tm * tp);
char * ctime(const time_t * tp);
struct tm * gmtime(const time_t *tp);
struct tm *localtime(const time_t * tp);
size_t strftime(char * s, size_t smax, const char * fmt, const struct tm * tp);
void tzset(void);

#endif
```

```
#ifndef _UNISTD_H
#define _UNISTD_H

/* ok, this may be a joke, but I'm working on it */
#define _POSIX_VERSION 198808L

#define _POSIX_CHOWN_RESTRICTED /* only root can do a chown (I think..) */
#define _POSIX_NO_TRUNC         /* no pathname truncation (but see in kernel) */
#define _POSIX_VDISABLE '\0'    /* character to disable things like ^C */
/*#define _POSIX_SAVED_IDS */   /* we'll get to this yet */
/*#define _POSIX_JOB_CONTROL */ /* we aren't there quite yet. Soon hopefully */

#define STDIN_FILENO    0
#define STDOUT_FILENO   1
#define STDERR_FILENO   2

#ifndef NULL
#define NULL    ((void *)0)
#endif

/* access */
#define F_OK    0
#define X_OK    1
#define W_OK    2
#define R_OK    4

/* lseek */
#define SEEK_SET        0
#define SEEK_CUR        1
#define SEEK_END        2

/* _SC stands for System Configuration. We don't use them much */
#define _SC_ARG_MAX             1
#define _SC_CHILD_MAX           2
#define _SC_CLOCKS_PER_SEC      3
#define _SC_NGROUPS_MAX         4
#define _SC_OPEN_MAX            5
#define _SC_JOB_CONTROL         6
#define _SC_SAVED_IDS           7
#define _SC_VERSION             8

/* more (possibly) configurable things - now pathnames */
#define _PC_LINK_MAX            1
#define _PC_MAX_CANON           2
#define _PC_MAX_INPUT           3
#define _PC_NAME_MAX            4
#define _PC_PATH_MAX            5
#define _PC_PIPE_BUF            6
#define _PC_NO_TRUNC            7
#define _PC_VDISABLE            8
#define _PC_CHOWN_RESTRICTED    9

#include <sys/stat.h>
#include <sys/times.h>
#include <sys/utsname.h>
#include <utime.h>

#ifdef __LIBRARY__

#define __NR_setup      0       /* used only by init, to get system going */
#define __NR_exit       1
#define __NR_fork       2
#define __NR_read       3
#define __NR_write      4
#define __NR_open       5
#define __NR_close      6
#define __NR_waitpid    7
#define __NR_creat      8
#define __NR_link       9
```

```
#define __NR_unlink     10
#define __NR_execve     11
#define __NR_chdir      12
#define __NR_time       13
#define __NR_mknod      14
#define __NR_chmod      15
#define __NR_chown      16
#define __NR_break      17
#define __NR_stat       18
#define __NR_lseek      19
#define __NR_getpid     20
#define __NR_mount      21
#define __NR_umount     22
#define __NR_setuid     23
#define __NR_getuid     24
#define __NR_stime      25
#define __NR_ptrace     26
#define __NR_alarm      27
#define __NR_fstat      28
#define __NR_pause      29
#define __NR_utime      30
#define __NR_stty       31
#define __NR_gtty       32
#define __NR_access     33
#define __NR_nice       34
#define __NR_ftime      35
#define __NR_sync       36
#define __NR_kill       37
#define __NR_rename     38
#define __NR_mkdir      39
#define __NR_rmdir      40
#define __NR_dup        41
#define __NR_pipe       42
#define __NR_times      43
#define __NR_prof       44
#define __NR_brk        45
#define __NR_setgid     46
#define __NR_getgid     47
#define __NR_signal     48
#define __NR_geteuid    49
#define __NR_getegid    50
#define __NR_acct       51
#define __NR_phys       52
#define __NR_lock       53
#define __NR_ioctl      54
#define __NR_fcntl      55
#define __NR_mpx        56
#define __NR_setpgid    57
#define __NR_ulimit     58
#define __NR_uname      59
#define __NR_umask      60
#define __NR_chroot     61
#define __NR_ustat      62
#define __NR_dup2       63
#define __NR_getppid    64
#define __NR_getpgrp    65
#define __NR_setsid     66
#define __NR_sigaction  67
#define __NR_sgetmask   68
#define __NR_ssetmask   69
#define __NR_setreuid   70
#define __NR_setregid   71

#define _syscall0(type,name) \
type name(void) \
{ \
long __res; \
__asm__ volatile ("int $0x80" \
        : "=a" (__res) \
```

```
        : "0" (__NR_##name)); \
if (__res >= 0) \
        return (type) __res; \
errno = -__res; \
return -1; \
}

#define _syscall1(type,name,atype,a) \
type name(atype a) \
{ \
long __res; \
__asm__ volatile ("int $0x80" \
        : "=a" (__res) \
        : "0" (__NR_##name),"b" ((long)(a))); \
if (__res >= 0) \
        return (type) __res; \
errno = -__res; \
return -1; \
}

#define _syscall2(type,name,atype,a,btype,b) \
type name(atype a,btype b) \
{ \
long __res; \
__asm__ volatile ("int $0x80" \
        : "=a" (__res) \
        : "0" (__NR_##name),"b" ((long)(a)),"c" ((long)(b))); \
if (__res >= 0) \
        return (type) __res; \
errno = -__res; \
return -1; \
}

#define _syscall3(type,name,atype,a,btype,b,ctype,c) \
type name(atype a,btype b,ctype c) \
{ \
long __res; \
__asm__ volatile ("int $0x80" \
        : "=a" (__res) \
        : "0" (__NR_##name),"b" ((long)(a)),"c" ((long)(b)),"d" ((long)(c))); \
if (__res>=0) \
        return (type) __res; \
errno=-__res; \
return -1; \
}

#endif /* __LIBRARY__ */

extern int errno;

int access(const char * filename, mode_t mode);
int acct(const char * filename);
int alarm(int sec);
int brk(void * end_data_segment);
void * sbrk(ptrdiff_t increment);
int chdir(const char * filename);
int chmod(const char * filename, mode_t mode);
int chown(const char * filename, uid_t owner, gid_t group);
int chroot(const char * filename);
int close(int fildes);
int creat(const char * filename, mode_t mode);
int dup(int fildes);
int execve(const char * filename, char ** argv, char ** envp);
int execv(const char * pathname, char ** argv);
int execvp(const char * file, char ** argv);
int execl(const char * pathname, char * arg0, ...);
int execlp(const char * file, char * arg0, ...);
int execle(const char * pathname, char * arg0, ...);
volatile void exit(int status);
```

```
volatile void _exit(int status);
int fcntl(int fildes, int cmd, ...);
int fork(void);
int getpid(void);
int getuid(void);
int geteuid(void);
int getgid(void);
int getegid(void);
int ioctl(int fildes, int cmd, ...);
int kill(pid_t pid, int signal);
int link(const char * filename1, const char * filename2);
int lseek(int fildes, off_t offset, int origin);
int mknod(const char * filename, mode_t mode, dev_t dev);
int mount(const char * specialfile, const char * dir, int rwflag);
int nice(int val);
int open(const char * filename, int flag, ...);
int pause(void);
int pipe(int * fildes);
int read(int fildes, char * buf, off_t count);
int setpgrp(void);
int setpgid(pid_t pid,pid_t pgid);
int setuid(uid_t uid);
int setgid(gid_t gid);
void (*signal(int sig, void (*fn)(int)))(int);
int stat(const char * filename, struct stat * stat_buf);
int fstat(int fildes, struct stat * stat_buf);
int stime(time_t * tptr);
int sync(void);
time_t time(time_t * tloc);
time_t times(struct tms * tbuf);
int ulimit(int cmd, long limit);
mode_t umask(mode_t mask);
int umount(const char * specialfile);
int uname(struct utsname * name);
int unlink(const char * filename);
int ustat(dev_t dev, struct ustat * ubuf);
int utime(const char * filename, struct utimbuf * times);
pid_t waitpid(pid_t pid,int * wait_stat,int options);
pid_t wait(int * wait_stat);
int write(int fildes, const char * buf, off_t count);
int dup2(int oldfd, int newfd);
int getppid(void);
pid_t getpgrp(void);
pid_t setsid(void);

#endif
```

```
#ifndef _UTIME_H
#define _UTIME_H

#include <sys/types.h>  /* I know - shouldn't do this, but .. */

struct utimbuf {
        time_t actime;
        time_t modtime;
};

extern int utime(const char *filename, struct utimbuf *times);

#endif
```

```
#define outb(value,port) \
__asm__ ("outb %%al,%%dx"::"a" (value),"d" (port))


#define inb(port) ({ \
unsigned char _v; \
__asm__ volatile ("inb %%dx,%%al":"=a" (_v):"d" (port)); \
_v; \
})

#define outb_p(value,port) \
__asm__ ("outb %%al,%%dx\n" \
                "\tjmp 1f\n" \
                "1:\tjmp 1f\n" \
                "1:"::"a" (value),"d" (port))

#define inb_p(port) ({ \
unsigned char _v; \
__asm__ volatile ("inb %%dx,%%al\n" \
        "\tjmp 1f\n" \
        "1:\tjmp 1f\n" \
        "1:":"=a" (_v):"d" (port)); \
_v; \
})
```

```
/*
 *  NOTE!!! memcpy(dest,src,n) assumes ds=es=normal data segment. This
 *  goes for all kernel functions (ds=es=kernel space, fs=local data,
 *  gs=null), as well as for all well-behaving user programs (ds=es=
 *  user data space). This is NOT a bug, as any user program that changes
 *  es deserves to die if it isn't careful.
 */
#define memcpy(dest,src,n) ({ \
void * _res = dest; \
__asm__ ("cld;rep;movsb" \
        ::"D" ((long)(_res)),"S" ((long)(src)),"c" ((long) (n)) \
        :"di","si","cx"); \
_res; \
})
```

```
extern inline unsigned char get_fs_byte(const char * addr)
{
        unsigned register char _v;

        __asm__ ("movb %%fs:%1,%0":"=r" (_v):"m" (*addr));
        return _v;
}

extern inline unsigned short get_fs_word(const unsigned short *addr)
{
        unsigned short _v;

        __asm__ ("movw %%fs:%1,%0":"=r" (_v):"m" (*addr));
        return _v;
}

extern inline unsigned long get_fs_long(const unsigned long *addr)
{
        unsigned long _v;

        __asm__ ("movl %%fs:%1,%0":"=r" (_v):"m" (*addr)); \
        return _v;
}

extern inline void put_fs_byte(char val,char *addr)
{
__asm__ ("movb %0,%%fs:%1"::"r" (val),"m" (*addr));
}

extern inline void put_fs_word(short val,short * addr)
{
__asm__ ("movw %0,%%fs:%1"::"r" (val),"m" (*addr));
}

extern inline void put_fs_long(unsigned long val,unsigned long * addr)
{
__asm__ ("movl %0,%%fs:%1"::"r" (val),"m" (*addr));
}

/*
 * Someone who knows GNU asm better than I should double check the followig.
 * It seems to work, but I don't know if I'm doing something subtly wrong.
 * --- TYT, 11/24/91
 * [ nothing wrong here, Linus ]
 */

extern inline unsigned long get_fs()
{
        unsigned short _v;
        __asm__("mov %%fs,%%ax":"=a" (_v):);
        return _v;
}

extern inline unsigned long get_ds()
{
        unsigned short _v;
        __asm__("mov %%ds,%%ax":"=a" (_v):);
        return _v;
}

extern inline void set_fs(unsigned long val)
{
        __asm__("mov %0,%%fs"::"a" ((unsigned short) val));
}
```

```
#define move_to_user_mode() \
__asm__ ("movl %%esp,%%eax\n\t" \
        "pushl $0x17\n\t" \
        "pushl %%eax\n\t" \
        "pushfl\n\t" \
        "pushl $0x0f\n\t" \
        "pushl $1f\n\t" \
        "iret\n" \
        "1:\tmovl $0x17,%%eax\n\t" \
        "movw %%ax,%%ds\n\t" \
        "movw %%ax,%%es\n\t" \
        "movw %%ax,%%fs\n\t" \
        "movw %%ax,%%gs" \
        :::"ax")

#define sti() __asm__ ("sti"::)
#define cli() __asm__ ("cli"::)
#define nop() __asm__ ("nop"::)

#define iret() __asm__ ("iret"::)

#define _set_gate(gate_addr,type,dpl,addr) \
__asm__ ("movw %%dx,%%ax\n\t" \
        "movw %0,%%dx\n\t" \
        "movl %%eax,%1\n\t" \
        "movl %%edx,%2" \
        : \
        : "i" ((short) (0x8000+(dpl<<13)+(type<<8))), \
        "o" (*((char *) (gate_addr))), \
        "o" (*(4+(char *) (gate_addr))), \
        "d" ((char *) (addr)),"a" (0x00080000))

#define set_intr_gate(n,addr) \
        _set_gate(&idt[n],14,0,addr)

#define set_trap_gate(n,addr) \
        _set_gate(&idt[n],15,0,addr)

#define set_system_gate(n,addr) \
        _set_gate(&idt[n],15,3,addr)

#define _set_seg_desc(gate_addr,type,dpl,base,limit) {\
        *(gate_addr) = ((base) & 0xff000000) | \
                (((base) & 0x00ff0000)>>16) | \
                ((limit) & 0xf0000) | \
                ((dpl)<<13) | \
                (0x00408000) | \
                ((type)<<8); \
        *((gate_addr)+1) = (((base) & 0x0000ffff)<<16) | \
                ((limit) & 0x0ffff); }

#define _set_tssldt_desc(n,addr,type) \
__asm__ ("movw $104,%1\n\t" \
        "movw %%ax,%2\n\t" \
        "rorl $16,%%eax\n\t" \
        "movb %%al,%3\n\t" \
        "movb $" type ",%4\n\t" \
        "movb $0x00,%5\n\t" \
        "movb %%ah,%6\n\t" \
        "rorl $16,%%eax" \
        ::"a" (addr), "m" (*(n)), "m" (*(n+2)), "m" (*(n+4)), \
         "m" (*(n+5)), "m" (*(n+6)), "m" (*(n+7)) \
        )

#define set_tss_desc(n,addr) _set_tssldt_desc(((char *) (n)),addr,"0x89")
#define set_ldt_desc(n,addr) _set_tssldt_desc(((char *) (n)),addr,"0x82")
```

```
#ifndef _CONFIG_H
#define _CONFIG_H

/*
 * The root-device is no longer hard-coded. You can change the default
 * root-device by changing the line ROOT_DEV = XXX in boot/bootsect.s
 */

/*
 * define your keyboard here -
 * KBD_FINNISH for Finnish keyboards
 * KBD_US for US-type
 * KBD_GR for German keyboards
 * KBD_FR for Frech keyboard
 */
/*#define KBD_US */
/*#define KBD_GR */
/*#define KBD_FR */
#define KBD_FINNISH

/*
 * Normally, Linux can get the drive parameters from the BIOS at
 * startup, but if this for some unfathomable reason fails, you'd
 * be left stranded. For this case, you can define HD_TYPE, which
 * contains all necessary info on your harddisk.
 *
 * The HD_TYPE macro should look like this:
 *
 * #define HD_TYPE { head, sect, cyl, wpcom, lzone, ctl}
 *
 * In case of two harddisks, the info should be sepatated by
 * commas:
 *
 * #define HD_TYPE { h,s,c,wpcom,lz,ctl },{ h,s,c,wpcom,lz,ctl }
 */
/*
 This is an example, two drives, first is type 2, second is type 3:

#define HD_TYPE { 4,17,615,300,615,8 }, { 6,17,615,300,615,0 }

 NOTE: ctl is 0 for all drives with heads<=8, and ctl=8 for drives
 with more than 8 heads.

 If you want the BIOS to tell what kind of drive you have, just
 leave HD_TYPE undefined. This is the normal thing to do.
*/

#endif
```

```
/*
 * This file contains some defines for the floppy disk controller.
 * Various sources. Mostly "IBM Microcomputers: A Programmers
 * Handbook", Sanches and Canton.
 */
#ifndef _FDREG_H
#define _FDREG_H

extern int ticks_to_floppy_on(unsigned int nr);
extern void floppy_on(unsigned int nr);
extern void floppy_off(unsigned int nr);
extern void floppy_select(unsigned int nr);
extern void floppy_deselect(unsigned int nr);

/* Fd controller regs. S&C, about page 340 */
#define FD_STATUS       0x3f4
#define FD_DATA         0x3f5
#define FD_DOR          0x3f2           /* Digital Output Register */
#define FD_DIR          0x3f7           /* Digital Input Register (read) */
#define FD_DCR          0x3f7           /* Diskette Control Register (write)*/

/* Bits of main status register */
#define STATUS_BUSYMASK 0x0F            /* drive busy mask */
#define STATUS_BUSY     0x10            /* FDC busy */
#define STATUS_DMA      0x20            /* 0- DMA mode */
#define STATUS_DIR      0x40            /* 0- cpu->fdc */
#define STATUS_READY    0x80            /* Data reg ready */

/* Bits of FD_ST0 */
#define ST0_DS          0x03            /* drive select mask */
#define ST0_HA          0x04            /* Head (Address) */
#define ST0_NR          0x08            /* Not Ready */
#define ST0_ECE         0x10            /* Equipment chech error */
#define ST0_SE          0x20            /* Seek end */
#define ST0_INTR        0xC0            /* Interrupt code mask */

/* Bits of FD_ST1 */
#define ST1_MAM         0x01            /* Missing Address Mark */
#define ST1_WP          0x02            /* Write Protect */
#define ST1_ND          0x04            /* No Data - unreadable */
#define ST1_OR          0x10            /* OverRun */
#define ST1_CRC         0x20            /* CRC error in data or addr */
#define ST1_EOC         0x80            /* End Of Cylinder */

/* Bits of FD_ST2 */
#define ST2_MAM         0x01            /* Missing Addess Mark (again) */
#define ST2_BC          0x02            /* Bad Cylinder */
#define ST2_SNS         0x04            /* Scan Not Satisfied */
#define ST2_SEH         0x08            /* Scan Equal Hit */
#define ST2_WC          0x10            /* Wrong Cylinder */
#define ST2_CRC         0x20            /* CRC error in data field */
#define ST2_CM          0x40            /* Control Mark = deleted */

/* Bits of FD_ST3 */
#define ST3_HA          0x04            /* Head (Address) */
#define ST3_TZ          0x10            /* Track Zero signal (1=track 0) */
#define ST3_WP          0x40            /* Write Protect */

/* Values for FD_COMMAND */
#define FD_RECALIBRATE  0x07            /* move to track 0 */
#define FD_SEEK         0x0F            /* seek track */
#define FD_READ         0xE6            /* read with MT, MFM, SKip deleted */
#define FD_WRITE        0xC5            /* write with MT, MFM */
#define FD_SENSEI       0x08            /* Sense Interrupt Status */
#define FD_SPECIFY      0x03            /* specify HUT etc */

/* DMA commands */
#define DMA_READ        0x46
#define DMA_WRITE       0x4A
```

```
#endif
```

```
/*
 * This file has definitions for some important file table
 * structures etc.
 */

#ifndef _FS_H
#define _FS_H

#include <sys/types.h>

/* devices are as follows: (same as minix, so we can use the minix
 * file system. These are major numbers.)
 *
 * 0 - unused (nodev)
 * 1 - /dev/mem
 * 2 - /dev/fd
 * 3 - /dev/hd
 * 4 - /dev/ttyx
 * 5 - /dev/tty
 * 6 - /dev/lp
 * 7 - unnamed pipes
 */

#define IS_SEEKABLE(x) ((x)>=1 && (x)<=3)

#define READ 0
#define WRITE 1
#define READA 2         /* read-ahead - don't pause */
#define WRITEA 3        /* "write-ahead" - silly, but somewhat useful */

void buffer_init(long buffer_end);

#define MAJOR(a) (((unsigned)(a))>>8)
#define MINOR(a) ((a)&0xff)

#define NAME_LEN 14
#define ROOT_INO 1

#define I_MAP_SLOTS 8
#define Z_MAP_SLOTS 8
#define SUPER_MAGIC 0x137F

#define NR_OPEN 20
#define NR_INODE 32
#define NR_FILE 64
#define NR_SUPER 8
#define NR_HASH 307
#define NR_BUFFERS nr_buffers
#define BLOCK_SIZE 1024
#define BLOCK_SIZE_BITS 10
#ifndef NULL
#define NULL ((void *) 0)
#endif

#define INODES_PER_BLOCK ((BLOCK_SIZE)/(sizeof (struct d_inode)))
#define DIR_ENTRIES_PER_BLOCK ((BLOCK_SIZE)/(sizeof (struct dir_entry)))

#define PIPE_HEAD(inode) ((inode).i_zone[0])
#define PIPE_TAIL(inode) ((inode).i_zone[1])
#define PIPE_SIZE(inode) ((PIPE_HEAD(inode)-PIPE_TAIL(inode))&(PAGE_SIZE-1))
#define PIPE_EMPTY(inode) (PIPE_HEAD(inode)==PIPE_TAIL(inode))
#define PIPE_FULL(inode) (PIPE_SIZE(inode)==(PAGE_SIZE-1))
#define INC_PIPE(head) \
__asm__("incl %0\n\tandl $4095,%0"::"m" (head))

typedef char buffer_block[BLOCK_SIZE];

struct buffer_head {
        char * b_data;                  /* pointer to data block (1024 bytes) */
```

```
        unsigned long b_blocknr;        /* block number */
        unsigned short b_dev;           /* device (0 = free) */
        unsigned char b_uptodate;
        unsigned char b_dirt;           /* 0-clean,1-dirty */
        unsigned char b_count;          /* users using this block */
        unsigned char b_lock;           /* 0 - ok, 1 -locked */
        struct task_struct * b_wait;
        struct buffer_head * b_prev;
        struct buffer_head * b_next;
        struct buffer_head * b_prev_free;
        struct buffer_head * b_next_free;
};

struct d_inode {
        unsigned short i_mode;
        unsigned short i_uid;
        unsigned long i_size;
        unsigned long i_time;
        unsigned char i_gid;
        unsigned char i_nlinks;
        unsigned short i_zone[9];
};

struct m_inode {
        unsigned short i_mode;
        unsigned short i_uid;
        unsigned long i_size;
        unsigned long i_mtime;
        unsigned char i_gid;
        unsigned char i_nlinks;
        unsigned short i_zone[9];
/* these are in memory also */
        struct task_struct * i_wait;
        unsigned long i_atime;
        unsigned long i_ctime;
        unsigned short i_dev;
        unsigned short i_num;
        unsigned short i_count;
        unsigned char i_lock;
        unsigned char i_dirt;
        unsigned char i_pipe;
        unsigned char i_mount;
        unsigned char i_seek;
        unsigned char i_update;
};

struct file {
        unsigned short f_mode;
        unsigned short f_flags;
        unsigned short f_count;
        struct m_inode * f_inode;
        off_t f_pos;
};

struct super_block {
        unsigned short s_ninodes;
        unsigned short s_nzones;
        unsigned short s_imap_blocks;
        unsigned short s_zmap_blocks;
        unsigned short s_firstdatazone;
        unsigned short s_log_zone_size;
        unsigned long s_max_size;
        unsigned short s_magic;
/* These are only in memory */
        struct buffer_head * s_imap[8];
        struct buffer_head * s_zmap[8];
        unsigned short s_dev;
        struct m_inode * s_isup;
        struct m_inode * s_imount;
```

```
        unsigned long s_time;
        struct task_struct * s_wait;
        unsigned char s_lock;
        unsigned char s_rd_only;
        unsigned char s_dirt;
};

struct d_super_block {
        unsigned short s_ninodes;
        unsigned short s_nzones;
        unsigned short s_imap_blocks;
        unsigned short s_zmap_blocks;
        unsigned short s_firstdatazone;
        unsigned short s_log_zone_size;
        unsigned long s_max_size;
        unsigned short s_magic;
};

struct dir_entry {
        unsigned short inode;
        char name[NAME_LEN];
};

extern struct m_inode inode_table[NR_INODE];
extern struct file file_table[NR_FILE];
extern struct super_block super_block[NR_SUPER];
extern struct buffer_head * start_buffer;
extern int nr_buffers;

extern void check_disk_change(int dev);
extern int floppy_change(unsigned int nr);
extern int ticks_to_floppy_on(unsigned int dev);
extern void floppy_on(unsigned int dev);
extern void floppy_off(unsigned int dev);
extern void truncate(struct m_inode * inode);
extern void sync_inodes(void);
extern void wait_on(struct m_inode * inode);
extern int bmap(struct m_inode * inode,int block);
extern int create_block(struct m_inode * inode,int block);
extern struct m_inode * namei(const char * pathname);
extern int open_namei(const char * pathname, int flag, int mode,
        struct m_inode ** res_inode);
extern void iput(struct m_inode * inode);
extern struct m_inode * iget(int dev,int nr);
extern struct m_inode * get_empty_inode(void);
extern struct m_inode * get_pipe_inode(void);
extern struct buffer_head * get_hash_table(int dev, int block);
extern struct buffer_head * getblk(int dev, int block);
extern void ll_rw_block(int rw, struct buffer_head * bh);
extern void brelse(struct buffer_head * buf);
extern struct buffer_head * bread(int dev,int block);
extern void bread_page(unsigned long addr,int dev,int b[4]);
extern struct buffer_head * breada(int dev,int block,...);
extern int new_block(int dev);
extern void free_block(int dev, int block);
extern struct m_inode * new_inode(int dev);
extern void free_inode(struct m_inode * inode);
extern int sync_dev(int dev);
extern struct super_block * get_super(int dev);
extern int ROOT_DEV;

extern void mount_root(void);

#endif
```

```
/*
 * This file contains some defines for the AT-hd-controller.
 * Various sources. Check out some definitions (see comments with
 * a ques).
 */
#ifndef _HDREG_H
#define _HDREG_H

/* Hd controller regs. Ref: IBM AT Bios-listing */
#define HD_DATA         0x1f0   /* _CTL when writing */
#define HD_ERROR        0x1f1   /* see err-bits */
#define HD_NSECTOR      0x1f2   /* nr of sectors to read/write */
#define HD_SECTOR       0x1f3   /* starting sector */
#define HD_LCYL         0x1f4   /* starting cylinder */
#define HD_HCYL         0x1f5   /* high byte of starting cyl */
#define HD_CURRENT      0x1f6   /* 101dhhhh , d=drive, hhhh=head */
#define HD_STATUS       0x1f7   /* see status-bits */
#define HD_PRECOMP HD_ERROR     /* same io address, read=error, write=precomp */
#define HD_COMMAND HD_STATUS    /* same io address, read=status, write=cmd */

#define HD_CMD          0x3f6

/* Bits of HD_STATUS */
#define ERR_STAT        0x01
#define INDEX_STAT      0x02
#define ECC_STAT        0x04    /* Corrected error */
#define DRQ_STAT        0x08
#define SEEK_STAT       0x10
#define WRERR_STAT      0x20
#define READY_STAT      0x40
#define BUSY_STAT       0x80

/* Values for HD_COMMAND */
#define WIN_RESTORE             0x10
#define WIN_READ                0x20
#define WIN_WRITE               0x30
#define WIN_VERIFY              0x40
#define WIN_FORMAT              0x50
#define WIN_INIT                0x60
#define WIN_SEEK                0x70
#define WIN_DIAGNOSE            0x90
#define WIN_SPECIFY             0x91

/* Bits for HD_ERROR */
#define MARK_ERR        0x01    /* Bad address mark ? */
#define TRK0_ERR        0x02    /* couldn't find track 0 */
#define ABRT_ERR        0x04    /* ? */
#define ID_ERR          0x10    /* ? */
#define ECC_ERR         0x40    /* ? */
#define BBD_ERR         0x80    /* ? */

struct partition {
        unsigned char boot_ind;         /* 0x80 - active (unused) */
        unsigned char head;             /* ? */
        unsigned char sector;           /* ? */
        unsigned char cyl;              /* ? */
        unsigned char sys_ind;          /* ? */
        unsigned char end_head;         /* ? */
        unsigned char end_sector;       /* ? */
        unsigned char end_cyl;          /* ? */
        unsigned int start_sect;        /* starting sector counting from 0 */
        unsigned int nr_sects;          /* nr of sectors in partition */
};

#endif
```

```
#ifndef _HEAD_H
#define _HEAD_H

typedef struct desc_struct {
        unsigned long a,b;
} desc_table[256];

extern unsigned long pg_dir[1024];
extern desc_table idt,gdt;

#define GDT_NUL 0
#define GDT_CODE 1
#define GDT_DATA 2
#define GDT_TMP 3

#define LDT_NUL 0
#define LDT_CODE 1
#define LDT_DATA 2

#endif
```

```
/*
 * 'kernel.h' contains some often-used function prototypes etc
 */
void verify_area(void * addr,int count);
volatile void panic(const char * str);
int printf(const char * fmt, ...);
int printk(const char * fmt, ...);
int tty_write(unsigned ch,char * buf,int count);
void * malloc(unsigned int size);
void free_s(void * obj, int size);

#define free(x) free_s((x), 0)

/*
 * This is defined as a macro, but at some point this might become a
 * real subroutine that sets a flag if it returns true (to do
 * BSD-style accounting where the process is flagged if it uses root
 * privs).  The implication of this is that you should do normal
 * permissions checks first, and check suser() last.
 */
#define suser() (current->euid == 0)
```

```
#ifndef _MM_H
#define _MM_H

#define PAGE_SIZE 4096

extern unsigned long get_free_page(void);
extern unsigned long put_page(unsigned long page,unsigned long address);
extern void free_page(unsigned long addr);

#endif
```

```
#ifndef _SCHED_H
#define _SCHED_H

#define NR_TASKS 64
#define HZ 100

#define FIRST_TASK task[0]
#define LAST_TASK task[NR_TASKS-1]

#include <linux/head.h>
#include <linux/fs.h>
#include <linux/mm.h>
#include <signal.h>

#if (NR_OPEN > 32)
#error "Currently the close-on-exec-flags are in one word, max 32 files/proc"
#endif

#define TASK_RUNNING            0
#define TASK_INTERRUPTIBLE      1
#define TASK_UNINTERRUPTIBLE    2
#define TASK_ZOMBIE             3
#define TASK_STOPPED            4

#ifndef NULL
#define NULL ((void *) 0)
#endif

extern int copy_page_tables(unsigned long from, unsigned long to, long size);
extern int free_page_tables(unsigned long from, unsigned long size);

extern void sched_init(void);
extern void schedule(void);
extern void trap_init(void);
extern void panic(const char * str);
extern int tty_write(unsigned minor,char * buf,int count);

typedef int (*fn_ptr)();

struct i387_struct {
        long    cwd;
        long    swd;
        long    twd;
        long    fip;
        long    fcs;
        long    foo;
        long    fos;
        long    st_space[20];   /* 8*10 bytes for each FP-reg = 80 bytes */
};

struct tss_struct {
        long    back_link;      /* 16 high bits zero */
        long    esp0;
        long    ss0;            /* 16 high bits zero */
        long    esp1;
        long    ss1;            /* 16 high bits zero */
        long    esp2;
        long    ss2;            /* 16 high bits zero */
        long    cr3;
        long    eip;
        long    eflags;
        long    eax,ecx,edx,ebx;
        long    esp;
        long    ebp;
        long    esi;
        long    edi;
        long    es;             /* 16 high bits zero */
        long    cs;             /* 16 high bits zero */
        long    ss;             /* 16 high bits zero */
```

```
        long    ds;             /* 16 high bits zero */
        long    fs;             /* 16 high bits zero */
        long    gs;             /* 16 high bits zero */
        long    ldt;            /* 16 high bits zero */
        long    trace_bitmap;   /* bits: trace 0, bitmap 16-31 */
        struct i387_struct i387;
};

struct task_struct {
/* these are hardcoded - don't touch */
        long state;     /* -1 unrunnable, 0 runnable, >0 stopped */
        long counter;
        long priority;
        long signal;
        struct sigaction sigaction[32];
        long blocked;   /* bitmap of masked signals */
/* various fields */
        int exit_code;
        unsigned long start_code,end_code,end_data,brk,start_stack;
        long pid,father,pgrp,session,leader;
        unsigned short uid,euid,suid;
        unsigned short gid,egid,sgid;
        long alarm;
        long utime,stime,cutime,cstime,start_time;
        unsigned short used_math;
/* file system info */
        int tty;                /* -1 if no tty, so it must be signed */
        unsigned short umask;
        struct m_inode * pwd;
        struct m_inode * root;
        struct m_inode * executable;
        unsigned long close_on_exec;
        struct file * filp[NR_OPEN];
/* ldt for this task 0 - zero 1 - cs 2 - ds&ss */
        struct desc_struct ldt[3];
/* tss for this task */
        struct tss_struct tss;
};

/*
 *  INIT_TASK is used to set up the first task table, touch at
 * your own risk!. Base=0, limit=0x9ffff (=640kB)
 */
#define INIT_TASK \
/* state etc */ { 0,15,15, \
/* signals */   0,{{},},0, \
/* ec,brk... */ 0,0,0,0,0,0, \
/* pid etc.. */ 0,-1,0,0,0, \
/* uid etc */   0,0,0,0,0,0, \
/* alarm */     0,0,0,0,0,0, \
/* math */      0, \
/* fs info */   -1,0022,NULL,NULL,NULL,0, \
/* filp */      {NULL,}, \
        { \
                {0,0}, \
/* ldt */       {0x9f,0xc0fa00}, \
                {0x9f,0xc0f200}, \
        }, \
/*tss*/ {0,PAGE_SIZE+(long)&init_task,0x10,0,0,0,0,(long)&pg_dir,\
         0,0,0,0,0,0,0,0, \
         0,0,0x17,0x17,0x17,0x17,0x17,0x17, \
         _LDT(0),0x80000000, \
                {} \
        }, \
}

extern struct task_struct *task[NR_TASKS];
extern struct task_struct *last_task_used_math;
extern struct task_struct *current;
```

```
extern long volatile jiffies;
extern long startup_time;

#define CURRENT_TIME (startup_time+jiffies/HZ)

extern void add_timer(long jiffies, void (*fn)(void));
extern void sleep_on(struct task_struct ** p);
extern void interruptible_sleep_on(struct task_struct ** p);
extern void wake_up(struct task_struct ** p);

/*
 * Entry into gdt where to find first TSS. 0-nul, 1-cs, 2-ds, 3-syscall
 * 4-TSS0, 5-LDT0, 6-TSS1 etc ...
 */
#define FIRST_TSS_ENTRY 4
#define FIRST_LDT_ENTRY (FIRST_TSS_ENTRY+1)
#define _TSS(n) ((((unsigned long) n)<<4)+(FIRST_TSS_ENTRY<<3))
#define _LDT(n) ((((unsigned long) n)<<4)+(FIRST_LDT_ENTRY<<3))
#define ltr(n) __asm__("ltr %%ax"::"a" (_TSS(n)))
#define lldt(n) __asm__("lldt %%ax"::"a" (_LDT(n)))
#define str(n) \
__asm__("str %%ax\n\t" \
        "subl %2,%%eax\n\t" \
        "shrl $4,%%eax" \
        :"=a" (n) \
        :"a" (0),"i" (FIRST_TSS_ENTRY<<3))
/*
 *      switch_to(n) should switch tasks to task nr n, first
 * checking that n isn't the current task, in which case it does nothing.
 * This also clears the TS-flag if the task we switched to has used
 * tha math co-processor latest.
 */
#define switch_to(n) {\
struct {long a,b;} __tmp; \
__asm__("cmpl %%ecx,_current\n\t" \
        "je 1f\n\t" \
        "movw %%dx,%1\n\t" \
        "xchgl %%ecx,_current\n\t" \
        "ljmp %0\n\t" \
        "cmpl %%ecx,_last_task_used_math\n\t" \
        "jne 1f\n\t" \
        "clts\n" \
        "1:" \
        ::"m" (*&__tmp.a),"m" (*&__tmp.b), \
        "d" (_TSS(n)),"c" ((long) task[n])); \
}

#define PAGE_ALIGN(n) (((n)+0xfff)&0xfffff000)

#define _set_base(addr,base) \
__asm__("movw %%dx,%0\n\t" \
        "rorl $16,%%edx\n\t" \
        "movb %%dl,%1\n\t" \
        "movb %%dh,%2" \
        ::"m" (*((addr)+2)), \
          "m" (*((addr)+4)), \
          "m" (*((addr)+7)), \
          "d" (base) \
        :"dx")

#define _set_limit(addr,limit) \
__asm__("movw %%dx,%0\n\t" \
        "rorl $16,%%edx\n\t" \
        "movb %1,%%dh\n\t" \
        "andb $0xf0,%%dh\n\t" \
        "orb %%dh,%%dl\n\t" \
        "movb %%dl,%1" \
        ::"m" (*(addr)), \
          "m" (*((addr)+6)), \
```

```
          "d" (limit) \
        :"dx")

#define set_base(ldt,base) _set_base( ((char *)&(ldt)) , base )
#define set_limit(ldt,limit) _set_limit( ((char *)&(ldt)) , (limit-1)>>12 )

#define _get_base(addr) ({\
unsigned long __base; \
__asm__("movb %3,%%dh\n\t" \
        "movb %2,%%dl\n\t" \
        "shll $16,%%edx\n\t" \
        "movw %1,%%dx" \
        :"=d" (__base) \
        :"m" (*((addr)+2)), \
         "m" (*((addr)+4)), \
         "m" (*((addr)+7))); \
__base;})

#define get_base(ldt) _get_base( ((char *)&(ldt)) )

#define get_limit(segment) ({ \
unsigned long __limit; \
__asm__("lsll %1,%0\n\tincl %0":"=r" (__limit):"r" (segment)); \
__limit;})

#endif
```

```
extern int sys_setup();
extern int sys_exit();
extern int sys_fork();
extern int sys_read();
extern int sys_write();
extern int sys_open();
extern int sys_close();
extern int sys_waitpid();
extern int sys_creat();
extern int sys_link();
extern int sys_unlink();
extern int sys_execve();
extern int sys_chdir();
extern int sys_time();
extern int sys_mknod();
extern int sys_chmod();
extern int sys_chown();
extern int sys_break();
extern int sys_stat();
extern int sys_lseek();
extern int sys_getpid();
extern int sys_mount();
extern int sys_umount();
extern int sys_setuid();
extern int sys_getuid();
extern int sys_stime();
extern int sys_ptrace();
extern int sys_alarm();
extern int sys_fstat();
extern int sys_pause();
extern int sys_utime();
extern int sys_stty();
extern int sys_gtty();
extern int sys_access();
extern int sys_nice();
extern int sys_ftime();
extern int sys_sync();
extern int sys_kill();
extern int sys_rename();
extern int sys_mkdir();
extern int sys_rmdir();
extern int sys_dup();
extern int sys_pipe();
extern int sys_times();
extern int sys_prof();
extern int sys_brk();
extern int sys_setgid();
extern int sys_getgid();
extern int sys_signal();
extern int sys_geteuid();
extern int sys_getegid();
extern int sys_acct();
extern int sys_phys();
extern int sys_lock();
extern int sys_ioctl();
extern int sys_fcntl();
extern int sys_mpx();
extern int sys_setpgid();
extern int sys_ulimit();
extern int sys_uname();
extern int sys_umask();
extern int sys_chroot();
extern int sys_ustat();
extern int sys_dup2();
extern int sys_getppid();
extern int sys_getpgrp();
extern int sys_setsid();
extern int sys_sigaction();
extern int sys_sgetmask();
```

```
extern int sys_ssetmask();
extern int sys_setreuid();
extern int sys_setregid();

fn_ptr sys_call_table[] = { sys_setup, sys_exit, sys_fork, sys_read,
sys_write, sys_open, sys_close, sys_waitpid, sys_creat, sys_link,
sys_unlink, sys_execve, sys_chdir, sys_time, sys_mknod, sys_chmod,
sys_chown, sys_break, sys_stat, sys_lseek, sys_getpid, sys_mount,
sys_umount, sys_setuid, sys_getuid, sys_stime, sys_ptrace, sys_alarm,
sys_fstat, sys_pause, sys_utime, sys_stty, sys_gtty, sys_access,
sys_nice, sys_ftime, sys_sync, sys_kill, sys_rename, sys_mkdir,
sys_rmdir, sys_dup, sys_pipe, sys_times, sys_prof, sys_brk, sys_setgid,
sys_getgid, sys_signal, sys_geteuid, sys_getegid, sys_acct, sys_phys,
sys_lock, sys_ioctl, sys_fcntl, sys_mpx, sys_setpgid, sys_ulimit,
sys_uname, sys_umask, sys_chroot, sys_ustat, sys_dup2, sys_getppid,
sys_getpgrp, sys_setsid, sys_sigaction, sys_sgetmask, sys_ssetmask,
sys_setreuid,sys_setregid };
```

```
/*
 * 'tty.h' defines some structures used by tty_io.c and some defines.
 *
 * NOTE! Don't touch this without checking that nothing in rs_io.s or
 * con_io.s breaks. Some constants are hardwired into the system (mainly
 * offsets into 'tty_queue'
 */

#ifndef _TTY_H
#define _TTY_H

#include <termios.h>

#define TTY_BUF_SIZE 1024

struct tty_queue {
        unsigned long data;
        unsigned long head;
        unsigned long tail;
        struct task_struct * proc_list;
        char buf[TTY_BUF_SIZE];
};

#define INC(a) ((a) = ((a)+1) & (TTY_BUF_SIZE-1))
#define DEC(a) ((a) = ((a)-1) & (TTY_BUF_SIZE-1))
#define EMPTY(a) ((a).head == (a).tail)
#define LEFT(a) (((a).tail-(a).head-1)&(TTY_BUF_SIZE-1))
#define LAST(a) ((a).buf[(TTY_BUF_SIZE-1)&((a).head-1)])
#define FULL(a) (!LEFT(a))
#define CHARS(a) (((a).head-(a).tail)&(TTY_BUF_SIZE-1))
#define GETCH(queue,c) \
(void)({c=(queue).buf[(queue).tail];INC((queue).tail);})
#define PUTCH(c,queue) \
(void)({(queue).buf[(queue).head]=(c);INC((queue).head);})

#define INTR_CHAR(tty) ((tty)->termios.c_cc[VINTR])
#define QUIT_CHAR(tty) ((tty)->termios.c_cc[VQUIT])
#define ERASE_CHAR(tty) ((tty)->termios.c_cc[VERASE])
#define KILL_CHAR(tty) ((tty)->termios.c_cc[VKILL])
#define EOF_CHAR(tty) ((tty)->termios.c_cc[VEOF])
#define START_CHAR(tty) ((tty)->termios.c_cc[VSTART])
#define STOP_CHAR(tty) ((tty)->termios.c_cc[VSTOP])
#define SUSPEND_CHAR(tty) ((tty)->termios.c_cc[VSUSP])

struct tty_struct {
        struct termios termios;
        int pgrp;
        int stopped;
        void (*write)(struct tty_struct * tty);
        struct tty_queue read_q;
        struct tty_queue write_q;
        struct tty_queue secondary;
        };

extern struct tty_struct tty_table[];

/*      intr=^C         quit=^|         erase=del       kill=^U
        eof=^D          vtime=\0        vmin=\1         sxtc=\0
        start=^Q        stop=^S         susp=^Z         eol=\0
        reprint=^R      discard=^U      werase=^W       lnext=^V
        eol2=\0
*/
#define INIT_C_CC "\003\034\177\025\004\0\1\0\021\023\032\0\022\017\027\026\0"

void rs_init(void);
void con_init(void);
void tty_init(void);

int tty_read(unsigned c, char * buf, int n);
```

```
int tty_write(unsigned c, char * buf, int n);

void rs_write(struct tty_struct * tty);
void con_write(struct tty_struct * tty);

void copy_to_cooked(struct tty_struct * tty);

#endif
```

```
#ifndef _SYS_STAT_H
#define _SYS_STAT_H

#include <sys/types.h>

struct stat {
        dev_t   st_dev;
        ino_t   st_ino;
        umode_t st_mode;
        nlink_t st_nlink;
        uid_t   st_uid;
        gid_t   st_gid;
        dev_t   st_rdev;
        off_t   st_size;
        time_t  st_atime;
        time_t  st_mtime;
        time_t  st_ctime;
};

#define S_IFMT  00170000
#define S_IFREG  0100000
#define S_IFBLK  0060000
#define S_IFDIR  0040000
#define S_IFCHR  0020000
#define S_IFIFO  0010000
#define S_ISUID  0004000
#define S_ISGID  0002000
#define S_ISVTX  0001000

#define S_ISREG(m)      (((m) & S_IFMT) == S_IFREG)
#define S_ISDIR(m)      (((m) & S_IFMT) == S_IFDIR)
#define S_ISCHR(m)      (((m) & S_IFMT) == S_IFCHR)
#define S_ISBLK(m)      (((m) & S_IFMT) == S_IFBLK)
#define S_ISFIFO(m)     (((m) & S_IFMT) == S_IFIFO)

#define S_IRWXU 00700
#define S_IRUSR 00400
#define S_IWUSR 00200
#define S_IXUSR 00100

#define S_IRWXG 00070
#define S_IRGRP 00040
#define S_IWGRP 00020
#define S_IXGRP 00010

#define S_IRWXO 00007
#define S_IROTH 00004
#define S_IWOTH 00002
#define S_IXOTH 00001

extern int chmod(const char *_path, mode_t mode);
extern int fstat(int fildes, struct stat *stat_buf);
extern int mkdir(const char *_path, mode_t mode);
extern int mkfifo(const char *_path, mode_t mode);
extern int stat(const char *filename, struct stat *stat_buf);
extern mode_t umask(mode_t mask);

#endif
```

```
#ifndef _TIMES_H
#define _TIMES_H

#include <sys/types.h>

struct tms {
        time_t tms_utime;
        time_t tms_stime;
        time_t tms_cutime;
        time_t tms_cstime;
};

extern time_t times(struct tms * tp);

#endif
```

```
#ifndef _SYS_TYPES_H
#define _SYS_TYPES_H

#ifndef _SIZE_T
#define _SIZE_T
typedef unsigned int size_t;
#endif

#ifndef _TIME_T
#define _TIME_T
typedef long time_t;
#endif

#ifndef _PTRDIFF_T
#define _PTRDIFF_T
typedef long ptrdiff_t;
#endif

#ifndef NULL
#define NULL ((void *) 0)
#endif

typedef int pid_t;
typedef unsigned short uid_t;
typedef unsigned char gid_t;
typedef unsigned short dev_t;
typedef unsigned short ino_t;
typedef unsigned short mode_t;
typedef unsigned short umode_t;
typedef unsigned char nlink_t;
typedef int daddr_t;
typedef long off_t;
typedef unsigned char u_char;
typedef unsigned short ushort;

typedef struct { int quot,rem; } div_t;
typedef struct { long quot,rem; } ldiv_t;

struct ustat {
        daddr_t f_tfree;
        ino_t f_tinode;
        char f_fname[6];
        char f_fpack[6];
};

#endif
```

```
#ifndef _SYS_UTSNAME_H
#define _SYS_UTSNAME_H

#include <sys/types.h>

struct utsname {
        char sysname[9];
        char nodename[9];
        char release[9];
        char version[9];
        char machine[9];
};

extern int uname(struct utsname * utsbuf);

#endif
```

```
#ifndef _SYS_WAIT_H
#define _SYS_WAIT_H

#include <sys/types.h>

#define _LOW(v)         ( (v) & 0377)
#define _HIGH(v)        ( ((v) >> 8) & 0377)

/* options for waitpid, WUNTRACED not supported */
#define WNOHANG         1
#define WUNTRACED       2

#define WIFEXITED(s)    (!((s)&0xFF)
#define WIFSTOPPED(s)   (((s)&0xFF)==0x7F)
#define WEXITSTATUS(s)  (((s)>>8)&0xFF)
#define WTERMSIG(s)     ((s)&0x7F)
#define WSTOPSIG(s)     (((s)>>8)&0xFF)
#define WIFSIGNALED(s)  (((unsigned int)(s)-1 & 0xFFFF) < 0xFF)

pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);

#endif
```

```
/*
 *  linux/init/main.c
 *
 *  (C) 1991  Linus Torvalds
 */

#define __LIBRARY__
#include <unistd.h>
#include <time.h>

/*
 * we need this inline - forking from kernel space will result
 * in NO COPY ON WRITE (!!!), until an execve is executed. This
 * is no problem, but for the stack. This is handled by not letting
 * main() use the stack at all after fork(). Thus, no function
 * calls - which means inline code for fork too, as otherwise we
 * would use the stack upon exit from 'fork()'.
 *
 * Actually only pause and fork are needed inline, so that there
 * won't be any messing with the stack from main(), but we define
 * some others too.
 */
static inline _syscall0(int,fork)
static inline _syscall0(int,pause)
static inline _syscall1(int,setup,void *,BIOS)
static inline _syscall0(int,sync)

#include <linux/tty.h>
#include <linux/sched.h>
#include <linux/head.h>
#include <asm/system.h>
#include <asm/io.h>

#include <stddef.h>
#include <stdarg.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>

#include <linux/fs.h>

static char printbuf[1024];

extern int vsprintf();
extern void init(void);
extern void blk_dev_init(void);
extern void chr_dev_init(void);
extern void hd_init(void);
extern void floppy_init(void);
extern void mem_init(long start, long end);
extern long rd_init(long mem_start, int length);
extern long kernel_mktime(struct tm * tm);
extern long startup_time;

/*
 * This is set up by the setup-routine at boot-time
 */
#define EXT_MEM_K (*(unsigned short *)0x90002)
#define DRIVE_INFO (*(struct drive_info *)0x90080)
#define ORIG_ROOT_DEV (*(unsigned short *)0x901FC)

/*
 * Yeah, yeah, it's ugly, but I cannot find how to do this correctly
 * and this seems to work. I anybody has more info on the real-time
 * clock I'd be interested. Most of this was trial and error, and some
 * bios-listing reading. Urghh.
 */

#define CMOS_READ(addr) ({ \
```

```
outb_p(0x80|addr,0x70); \
inb_p(0x71); \
})

#define BCD_TO_BIN(val) ((val)=((val)&15) + ((val)>>4)*10)

static void time_init(void)
{
        struct tm time;

        do {
                time.tm_sec = CMOS_READ(0);
                time.tm_min = CMOS_READ(2);
                time.tm_hour = CMOS_READ(4);
                time.tm_mday = CMOS_READ(7);
                time.tm_mon = CMOS_READ(8);
                time.tm_year = CMOS_READ(9);
        } while (time.tm_sec != CMOS_READ(0));
        BCD_TO_BIN(time.tm_sec);
        BCD_TO_BIN(time.tm_min);
        BCD_TO_BIN(time.tm_hour);
        BCD_TO_BIN(time.tm_mday);
        BCD_TO_BIN(time.tm_mon);
        BCD_TO_BIN(time.tm_year);
        time.tm_mon--;
        startup_time = kernel_mktime(&time);
}

static long memory_end = 0;
static long buffer_memory_end = 0;
static long main_memory_start = 0;

struct drive_info { char dummy[32]; } drive_info;

void main(void)         /* This really IS void, no error here. */
{                       /* The startup routine assumes (well, ...) this */
/*
 * Interrupts are still disabled. Do necessary setups, then
 * enable them
 */
        ROOT_DEV = ORIG_ROOT_DEV;
        drive_info = DRIVE_INFO;
        memory_end = (1<<20) + (EXT_MEM_K<<10);
        memory_end &= 0xfffff000;
        if (memory_end > 16*1024*1024)
                memory_end = 16*1024*1024;
        if (memory_end > 12*1024*1024)
                buffer_memory_end = 4*1024*1024;
        else if (memory_end > 6*1024*1024)
                buffer_memory_end = 2*1024*1024;
        else
                buffer_memory_end = 1*1024*1024;
        main_memory_start = buffer_memory_end;
#ifdef RAMDISK
        main_memory_start += rd_init(main_memory_start, RAMDISK*1024);
#endif
        mem_init(main_memory_start,memory_end);
        trap_init();
        blk_dev_init();
        chr_dev_init();
        tty_init();
        time_init();
        sched_init();
        buffer_init(buffer_memory_end);
        hd_init();
        floppy_init();
        sti();
        move_to_user_mode();
        if (!fork()) {          /* we count on this going ok */
```

```
                init();
        }
/*
 *   NOTE!!   For any other task 'pause()' would mean we have to get a
 * signal to awaken, but task0 is the sole exception (see 'schedule()')
 * as task 0 gets activated at every idle moment (when no other tasks
 * can run). For task0 'pause()' just means we go check if some other
 * task can run, and if not we return here.
 */
        for(;;) pause();
}

static int printf(const char *fmt, ...)
{
        va_list args;
        int i;

        va_start(args, fmt);
        write(1,printbuf,i=vsprintf(printbuf, fmt, args));
        va_end(args);
        return i;
}

static char * argv_rc[] = { "/bin/sh", NULL };
static char * envp_rc[] = { "HOME=/", NULL };

static char * argv[] = { "-/bin/sh",NULL };
static char * envp[] = { "HOME=/usr/root", NULL };

void init(void)
{
        int pid,i;

        setup((void *) &drive_info);
        (void) open("/dev/tty0",O_RDWR,0);
        (void) dup(0);
        (void) dup(0);
        printf("%d buffers = %d bytes buffer space\n\r",NR_BUFFERS,
                NR_BUFFERS*BLOCK_SIZE);
        printf("Free mem: %d bytes\n\r",memory_end-main_memory_start);
        if (!(pid=fork())) {
                close(0);
                if (open("/etc/rc",O_RDONLY,0))
                        _exit(1);
                execve("/bin/sh",argv_rc,envp_rc);
                _exit(2);
        }
        if (pid>0)
                while (pid != wait(&i))
                        /* nothing */;
        while (1) {
                if ((pid=fork())<0) {
                        printf("Fork failed in init\r\n");
                        continue;
                }
                if (!pid) {
                        close(0);close(1);close(2);
                        setsid();
                        (void) open("/dev/tty0",O_RDWR,0);
                        (void) dup(0);
                        (void) dup(0);
                        _exit(execve("/bin/sh",argv,envp));
                }
                while (1)
                        if (pid == wait(&i))
                                break;
                printf("\n\rchild %d died with code %04x\n\r",pid,i);
                sync();
        }
```

```
        _exit(0);       /* NOTE! _exit, not exit() */
}
```

```
#
# Makefile for the FREAX-kernel.
#
# Note! Dependencies are done automagically by 'make dep', which also
# removes any old dependencies. DON'T put your own dependencies here
# unless it's something special (ie not a .c file).
#

AR      =gar
AS      =gas
LD      =gld
LDFLAGS =-s -x
CC      =gcc
CFLAGS  =-Wall -O -fstrength-reduce -fomit-frame-pointer -fcombine-regs \
        -finline-functions -mstring-insns -nostdinc -I../include
CPP     =gcc -E -nostdinc -I../include

.c.s:
        $(CC) $(CFLAGS) \
        -S -o $*.s $<
.s.o:
        $(AS) -c -o $*.o $<
.c.o:
        $(CC) $(CFLAGS) \
        -c -o $*.o $<

OBJS  = sched.o system_call.o traps.o asm.o fork.o \
        panic.o printk.o vsprintf.o sys.o exit.o \
        signal.o mktime.o

kernel.o: $(OBJS)
        $(LD) -r -o kernel.o $(OBJS)
        sync

clean:
        rm -f core *.o *.a tmp_make keyboard.s
        for i in *.c;do rm -f `basename $$i .c`.s;done
        (cd chr_drv; make clean)
        (cd blk_drv; make clean)
        (cd math; make clean)

dep:
        sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
        (for i in *.c;do echo -n `echo $$i | sed 's,\.c,\.s,'`" "; \
                $(CPP) -M $$i;done) >> tmp_make
        cp tmp_make Makefile
        (cd chr_drv; make dep)
        (cd blk_drv; make dep)

### Dependencies:
exit.s exit.o : exit.c ../include/errno.h ../include/signal.h \
  ../include/sys/types.h ../include/sys/wait.h ../include/linux/sched.h \
  ../include/linux/head.h ../include/linux/fs.h ../include/linux/mm.h \
  ../include/linux/kernel.h ../include/linux/tty.h ../include/termios.h \
  ../include/asm/segment.h
fork.s fork.o : fork.c ../include/errno.h ../include/linux/sched.h \
  ../include/linux/head.h ../include/linux/fs.h ../include/sys/types.h \
  ../include/linux/mm.h ../include/signal.h ../include/linux/kernel.h \
  ../include/asm/segment.h ../include/asm/system.h
mktime.s mktime.o : mktime.c ../include/time.h
panic.s panic.o : panic.c ../include/linux/kernel.h ../include/linux/sched.h \
  ../include/linux/head.h ../include/linux/fs.h ../include/sys/types.h \
  ../include/linux/mm.h ../include/signal.h
printk.s printk.o : printk.c ../include/stdarg.h ../include/stddef.h \
  ../include/linux/kernel.h
sched.s sched.o : sched.c ../include/linux/sched.h ../include/linux/head.h \
  ../include/linux/fs.h ../include/sys/types.h ../include/linux/mm.h \
  ../include/signal.h ../include/linux/kernel.h ../include/linux/sys.h \
  ../include/linux/fdreg.h ../include/asm/system.h ../include/asm/io.h \
```

```
    ../include/asm/segment.h
signal.s signal.o : signal.c ../include/linux/sched.h ../include/linux/head.h \
    ../include/linux/fs.h ../include/sys/types.h ../include/linux/mm.h \
    ../include/signal.h ../include/linux/kernel.h ../include/asm/segment.h
sys.s sys.o : sys.c ../include/errno.h ../include/linux/sched.h \
    ../include/linux/head.h ../include/linux/fs.h ../include/sys/types.h \
    ../include/linux/mm.h ../include/signal.h ../include/linux/tty.h \
    ../include/termios.h ../include/linux/kernel.h ../include/asm/segment.h \
    ../include/sys/times.h ../include/sys/utsname.h
traps.s traps.o : traps.c ../include/string.h ../include/linux/head.h \
    ../include/linux/sched.h ../include/linux/fs.h ../include/sys/types.h \
    ../include/linux/mm.h ../include/signal.h ../include/linux/kernel.h \
    ../include/asm/system.h ../include/asm/segment.h ../include/asm/io.h
vsprintf.s vsprintf.o : vsprintf.c ../include/stdarg.h ../include/string.h
```

```
/*
 *  linux/kernel/asm.s
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * asm.s contains the low-level code for most hardware faults.
 * page_exception is handled by the mm, so that isn't here. This
 * file also handles (hopefully) fpu-exceptions due to TS-bit, as
 * the fpu must be properly saved/resored. This hasn't been tested.
 */

.globl _divide_error,_debug,_nmi,_int3,_overflow,_bounds,_invalid_op
.globl _double_fault,_coprocessor_segment_overrun
.globl _invalid_TSS,_segment_not_present,_stack_segment
.globl _general_protection,_coprocessor_error,_irq13,_reserved

_divide_error:
        pushl $_do_divide_error
no_error_code:
        xchgl %eax,(%esp)
        pushl %ebx
        pushl %ecx
        pushl %edx
        pushl %edi
        pushl %esi
        pushl %ebp
        push %ds
        push %es
        push %fs
        pushl $0                # "error code"
        lea 44(%esp),%edx
        pushl %edx
        movl $0x10,%edx
        mov %dx,%ds
        mov %dx,%es
        mov %dx,%fs
        call *%eax
        addl $8,%esp
        pop %fs
        pop %es
        pop %ds
        popl %ebp
        popl %esi
        popl %edi
        popl %edx
        popl %ecx
        popl %ebx
        popl %eax
        iret

_debug:
        pushl $_do_int3         # _do_debug
        jmp no_error_code

_nmi:
        pushl $_do_nmi
        jmp no_error_code

_int3:
        pushl $_do_int3
        jmp no_error_code

_overflow:
        pushl $_do_overflow
        jmp no_error_code

_bounds:
```

```
        pushl $_do_bounds
        jmp no_error_code

_invalid_op:
        pushl $_do_invalid_op
        jmp no_error_code

_coprocessor_segment_overrun:
        pushl $_do_coprocessor_segment_overrun
        jmp no_error_code

_reserved:
        pushl $_do_reserved
        jmp no_error_code

_irq13:
        pushl %eax
        xorb %al,%al
        outb %al,$0xF0
        movb $0x20,%al
        outb %al,$0x20
        jmp 1f
1:      jmp 1f
1:      outb %al,$0xA0
        popl %eax
        jmp _coprocessor_error

_double_fault:
        pushl $_do_double_fault
error_code:
        xchgl %eax,4(%esp)              # error code <-> %eax
        xchgl %ebx,(%esp)               # &function <-> %ebx
        pushl %ecx
        pushl %edx
        pushl %edi
        pushl %esi
        pushl %ebp
        push %ds
        push %es
        push %fs
        pushl %eax                      # error code
        lea 44(%esp),%eax               # offset
        pushl %eax
        movl $0x10,%eax
        mov %ax,%ds
        mov %ax,%es
        mov %ax,%fs
        call *%ebx
        addl $8,%esp
        pop %fs
        pop %es
        pop %ds
        popl %ebp
        popl %esi
        popl %edi
        popl %edx
        popl %ecx
        popl %ebx
        popl %eax
        iret

_invalid_TSS:
        pushl $_do_invalid_TSS
        jmp error_code

_segment_not_present:
        pushl $_do_segment_not_present
        jmp error_code
```

```
_stack_segment:
        pushl $_do_stack_segment
        jmp error_code

_general_protection:
        pushl $_do_general_protection
        jmp error_code
```

```
/*
 *  linux/kernel/exit.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <errno.h>
#include <signal.h>
#include <sys/wait.h>

#include <linux/sched.h>
#include <linux/kernel.h>
#include <linux/tty.h>
#include <asm/segment.h>

int sys_pause(void);
int sys_close(int fd);

void release(struct task_struct * p)
{
        int i;

        if (!p)
                return;
        for (i=1 ; i<NR_TASKS ; i++)
                if (task[i]==p) {
                        task[i]=NULL;
                        free_page((long)p);
                        schedule();
                        return;
                }
        panic("trying to release non-existent task");
}

static inline int send_sig(long sig,struct task_struct * p,int priv)
{
        if (!p || sig<1 || sig>32)
                return -EINVAL;
        if (priv || (current->euid==p->euid) || suser())
                p->signal |= (1<<(sig-1));
        else
                return -EPERM;
        return 0;
}

static void kill_session(void)
{
        struct task_struct **p = NR_TASKS + task;

        while (--p > &FIRST_TASK) {
                if (*p && (*p)->session == current->session)
                        (*p)->signal |= 1<<(SIGHUP-1);
        }
}

/*
 * XXX need to check permissions needed to send signals to process
 * groups, etc. etc.  kill() permissions semantics are tricky!
 */
int sys_kill(int pid,int sig)
{
        struct task_struct **p = NR_TASKS + task;
        int err, retval = 0;

        if (!pid) while (--p > &FIRST_TASK) {
                if (*p && (*p)->pgrp == current->pid)
                        if (err=send_sig(sig,*p,1))
                                retval = err;
        } else if (pid>0) while (--p > &FIRST_TASK) {
```

```
                if (*p && (*p)->pid == pid)
                        if (err=send_sig(sig,*p,0))
                                retval = err;
        } else if (pid == -1) while (--p > &FIRST_TASK)
                if (err = send_sig(sig,*p,0))
                        retval = err;
        else while (--p > &FIRST_TASK)
                if (*p && (*p)->pgrp == -pid)
                        if (err = send_sig(sig,*p,0))
                                retval = err;
        return retval;
}

static void tell_father(int pid)
{
        int i;

        if (pid)
                for (i=0;i<NR_TASKS;i++) {
                        if (!task[i])
                                continue;
                        if (task[i]->pid != pid)
                                continue;
                        task[i]->signal |= (1<<(SIGCHLD-1));
                        return;
                }
/* if we don't find any fathers, we just release ourselves */
/* This is not really OK. Must change it to make father 1 */
        printk("BAD BAD - no father found\n\r");
        release(current);
}

int do_exit(long code)
{
        int i;

        free_page_tables(get_base(current->ldt[1]),get_limit(0x0f));
        free_page_tables(get_base(current->ldt[2]),get_limit(0x17));
        for (i=0 ; i<NR_TASKS ; i++)
                if (task[i] && task[i]->father == current->pid) {
                        task[i]->father = 1;
                        if (task[i]->state == TASK_ZOMBIE)
                                /* assumption task[1] is always init */
                                (void) send_sig(SIGCHLD, task[1], 1);
                }
        for (i=0 ; i<NR_OPEN ; i++)
                if (current->filp[i])
                        sys_close(i);
        iput(current->pwd);
        current->pwd=NULL;
        iput(current->root);
        current->root=NULL;
        iput(current->executable);
        current->executable=NULL;
        if (current->leader && current->tty >= 0)
                tty_table[current->tty].pgrp = 0;
        if (last_task_used_math == current)
                last_task_used_math = NULL;
        if (current->leader)
                kill_session();
        current->state = TASK_ZOMBIE;
        current->exit_code = code;
        tell_father(current->father);
        schedule();
        return (-1);    /* just to suppress warnings */
}

int sys_exit(int error_code)
{
```

```
        return do_exit((error_code&0xff)<<8);
}

int sys_waitpid(pid_t pid,unsigned long * stat_addr, int options)
{
        int flag, code;
        struct task_struct ** p;

        verify_area(stat_addr,4);
repeat:
        flag=0;
        for(p = &LAST_TASK ; p > &FIRST_TASK ; --p) {
                if (!*p || *p == current)
                        continue;
                if ((*p)->father != current->pid)
                        continue;
                if (pid>0) {
                        if ((*p)->pid != pid)
                                continue;
                } else if (!pid) {
                        if ((*p)->pgrp != current->pgrp)
                                continue;
                } else if (pid != -1) {
                        if ((*p)->pgrp != -pid)
                                continue;
                }
                switch ((*p)->state) {
                        case TASK_STOPPED:
                                if (!(options & WUNTRACED))
                                        continue;
                                put_fs_long(0x7f,stat_addr);
                                return (*p)->pid;
                        case TASK_ZOMBIE:
                                current->cutime += (*p)->utime;
                                current->cstime += (*p)->stime;
                                flag = (*p)->pid;
                                code = (*p)->exit_code;
                                release(*p);
                                put_fs_long(code,stat_addr);
                                return flag;
                        default:
                                flag=1;
                                continue;
                }
        }
        if (flag) {
                if (options & WNOHANG)
                        return 0;
                current->state=TASK_INTERRUPTIBLE;
                schedule();
                if (!(current->signal &= ~(1<<(SIGCHLD-1))))
                        goto repeat;
                else
                        return -EINTR;
        }
        return -ECHILD;
}
```

```
/*
 *  linux/kernel/fork.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 *  'fork.c' contains the help-routines for the 'fork' system call
 * (see also system_call.s), and some misc functions ('verify_area').
 * Fork is rather simple, once you get the hang of it, but the memory
 * management can be a bitch. See 'mm/mm.c': 'copy_page_tables()'
 */
#include <errno.h>

#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/segment.h>
#include <asm/system.h>

extern void write_verify(unsigned long address);

long last_pid=0;

void verify_area(void * addr,int size)
{
        unsigned long start;

        start = (unsigned long) addr;
        size += start & 0xfff;
        start &= 0xfffff000;
        start += get_base(current->ldt[2]);
        while (size>0) {
                size -= 4096;
                write_verify(start);
                start += 4096;
        }
}

int copy_mem(int nr,struct task_struct * p)
{
        unsigned long old_data_base,new_data_base,data_limit;
        unsigned long old_code_base,new_code_base,code_limit;

        code_limit=get_limit(0x0f);
        data_limit=get_limit(0x17);
        old_code_base = get_base(current->ldt[1]);
        old_data_base = get_base(current->ldt[2]);
        if (old_data_base != old_code_base)
                panic("We don't support separate I&D");
        if (data_limit < code_limit)
                panic("Bad data_limit");
        new_data_base = new_code_base = nr * 0x4000000;
        p->start_code = new_code_base;
        set_base(p->ldt[1],new_code_base);
        set_base(p->ldt[2],new_data_base);
        if (copy_page_tables(old_data_base,new_data_base,data_limit)) {
                free_page_tables(new_data_base,data_limit);
                return -ENOMEM;
        }
        return 0;
}

/*
 *  Ok, this is the main fork-routine. It copies the system process
 * information (task[nr]) and sets up the necessary registers. It
 * also copies the data segment in it's entirety.
 */
int copy_process(int nr,long ebp,long edi,long esi,long gs,long none,
                long ebx,long ecx,long edx,
```

```
                long fs,long es,long ds,
                long eip,long cs,long eflags,long esp,long ss)
{
        struct task_struct *p;
        int i;
        struct file *f;

        p = (struct task_struct *) get_free_page();
        if (!p)
                return -EAGAIN;
        task[nr] = p;
        *p = *current;  /* NOTE! this doesn't copy the supervisor stack */
        p->state = TASK_UNINTERRUPTIBLE;
        p->pid = last_pid;
        p->father = current->pid;
        p->counter = p->priority;
        p->signal = 0;
        p->alarm = 0;
        p->leader = 0;          /* process leadership doesn't inherit */
        p->utime = p->stime = 0;
        p->cutime = p->cstime = 0;
        p->start_time = jiffies;
        p->tss.back_link = 0;
        p->tss.esp0 = PAGE_SIZE + (long) p;
        p->tss.ss0 = 0x10;
        p->tss.eip = eip;
        p->tss.eflags = eflags;
        p->tss.eax = 0;
        p->tss.ecx = ecx;
        p->tss.edx = edx;
        p->tss.ebx = ebx;
        p->tss.esp = esp;
        p->tss.ebp = ebp;
        p->tss.esi = esi;
        p->tss.edi = edi;
        p->tss.es = es & 0xffff;
        p->tss.cs = cs & 0xffff;
        p->tss.ss = ss & 0xffff;
        p->tss.ds = ds & 0xffff;
        p->tss.fs = fs & 0xffff;
        p->tss.gs = gs & 0xffff;
        p->tss.ldt = _LDT(nr);
        p->tss.trace_bitmap = 0x80000000;
        if (last_task_used_math == current)
                __asm__("clts ; fnsave %0"::"m" (p->tss.i387));
        if (copy_mem(nr,p)) {
                task[nr] = NULL;
                free_page((long) p);
                return -EAGAIN;
        }
        for (i=0; i<NR_OPEN;i++)
                if (f=p->filp[i])
                        f->f_count++;
        if (current->pwd)
                current->pwd->i_count++;
        if (current->root)
                current->root->i_count++;
        if (current->executable)
                current->executable->i_count++;
        set_tss_desc(gdt+(nr<<1)+FIRST_TSS_ENTRY,&(p->tss));
        set_ldt_desc(gdt+(nr<<1)+FIRST_LDT_ENTRY,&(p->ldt));
        p->state = TASK_RUNNING;        /* do this last, just in case */
        return last_pid;
}

int find_empty_process(void)
{
        int i;
```

```
        repeat:
                if ((++last_pid)<0) last_pid=1;
                for(i=0 ; i<NR_TASKS ; i++)
                        if (task[i] && task[i]->pid == last_pid) goto repeat;
        for(i=1 ; i<NR_TASKS ; i++)
                if (!task[i])
                        return i;
        return -EAGAIN;
}
```

```
/*
 *  linux/kernel/mktime.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <time.h>

/*
 * This isn't the library routine, it is only used in the kernel.
 * as such, we don't care about years<1970 etc, but assume everything
 * is ok. Similarly, TZ etc is happily ignored. We just do everything
 * as easily as possible. Let's find something public for the library
 * routines (although I think minix times is public).
 */
/*
 * PS. I hate whoever though up the year 1970 - couldn't they have gotten
 * a leap-year instead? I also hate Gregorius, pope or no. I'm grumpy.
 */
#define MINUTE 60
#define HOUR (60*MINUTE)
#define DAY (24*HOUR)
#define YEAR (365*DAY)

/* interestingly, we assume leap-years */
static int month[12] = {
        0,
        DAY*(31),
        DAY*(31+29),
        DAY*(31+29+31),
        DAY*(31+29+31+30),
        DAY*(31+29+31+30+31),
        DAY*(31+29+31+30+31+30),
        DAY*(31+29+31+30+31+30+31),
        DAY*(31+29+31+30+31+30+31+31),
        DAY*(31+29+31+30+31+30+31+31+30),
        DAY*(31+29+31+30+31+30+31+31+30+31),
        DAY*(31+29+31+30+31+30+31+31+30+31+30)
};

long kernel_mktime(struct tm * tm)
{
        long res;
        int year;

        year = tm->tm_year - 70;
/* magic offsets (y+1) needed to get leapyears right.*/
        res = YEAR*year + DAY*((year+1)/4);
        res += month[tm->tm_mon];
/* and (y+2) here. If it wasn't a leap-year, we have to adjust */
        if (tm->tm_mon>1 && ((year+2)%4))
                res -= DAY;
        res += DAY*(tm->tm_mday-1);
        res += HOUR*tm->tm_hour;
        res += MINUTE*tm->tm_min;
        res += tm->tm_sec;
        return res;
}
```

```
/*
 *  linux/kernel/panic.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * This function is used through-out the kernel (includeinh mm and fs)
 * to indicate a major problem.
 */
#include <linux/kernel.h>
#include <linux/sched.h>

void sys_sync(void);    /* it's really int */

volatile void panic(const char * s)
{
        printk("Kernel panic: %s\n\r",s);
        if (current == task[0])
                printk("In swapper task - not syncing\n\r");
        else
                sys_sync();
        for(;;);
}
```

```
/*
 *  linux/kernel/printk.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * When in kernel-mode, we cannot use printf, as fs is liable to
 * point to 'interesting' things. Make a printf with fs-saving, and
 * all is well.
 */
#include <stdarg.h>
#include <stddef.h>

#include <linux/kernel.h>

static char buf[1024];

extern int vsprintf(char * buf, const char * fmt, va_list args);

int printk(const char *fmt, ...)
{
        va_list args;
        int i;

        va_start(args, fmt);
        i=vsprintf(buf,fmt,args);
        va_end(args);
        __asm__("push %%fs\n\t"
                "push %%ds\n\t"
                "pop %%fs\n\t"
                "pushl %0\n\t"
                "pushl $_buf\n\t"
                "pushl $0\n\t"
                "call _tty_write\n\t"
                "addl $8,%%esp\n\t"
                "popl %0\n\t"
                "pop %%fs"
                ::"r" (i):"ax","cx","dx");
        return i;
}
```

```
/*
 *  linux/kernel/sched.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * 'sched.c' is the main kernel file. It contains scheduling primitives
 * (sleep_on, wakeup, schedule etc) as well as a number of simple system
 * call functions (type getpid(), which just extracts a field from
 * current-task
 */
#include <linux/sched.h>
#include <linux/kernel.h>
#include <linux/sys.h>
#include <linux/fdreg.h>
#include <asm/system.h>
#include <asm/io.h>
#include <asm/segment.h>

#include <signal.h>

#define _S(nr) (1<<((nr)-1))
#define _BLOCKABLE (~(_S(SIGKILL) | _S(SIGSTOP)))

void show_task(int nr,struct task_struct * p)
{
        int i,j = 4096-sizeof(struct task_struct);

        printk("%d: pid=%d, state=%d, ",nr,p->pid,p->state);
        i=0;
        while (i<j && !((char *)(p+1))[i])
                i++;
        printk("%d (of %d) chars free in kernel stack\n\r",i,j);
}

void show_stat(void)
{
        int i;

        for (i=0;i<NR_TASKS;i++)
                if (task[i])
                        show_task(i,task[i]);
}

#define LATCH (1193180/HZ)

extern void mem_use(void);

extern int timer_interrupt(void);
extern int system_call(void);

union task_union {
        struct task_struct task;
        char stack[PAGE_SIZE];
};

static union task_union init_task = {INIT_TASK,};

long volatile jiffies=0;
long startup_time=0;
struct task_struct *current = &(init_task.task);
struct task_struct *last_task_used_math = NULL;

struct task_struct * task[NR_TASKS] = {&(init_task.task), };

long user_stack [ PAGE_SIZE>>2 ] ;

struct {
```

```
        long * a;
        short b;
        } stack_start = { & user_stack [PAGE_SIZE>>2] , 0x10 };
/*
 *  'math_state_restore()' saves the current math information in the
 * old math state array, and gets the new ones from the current task
 */
void math_state_restore()
{
        if (last_task_used_math == current)
                return;
        __asm__("fwait");
        if (last_task_used_math) {
                __asm__("fnsave %0"::"m" (last_task_used_math->tss.i387));
        }
        last_task_used_math=current;
        if (current->used_math) {
                __asm__("frstor %0"::"m" (current->tss.i387));
        } else {
                __asm__("fninit"::);
                current->used_math=1;
        }
}

/*
 *  'schedule()' is the scheduler function. This is GOOD CODE! There
 * probably won't be any reason to change this, as it should work well
 * in all circumstances (ie gives IO-bound processes good response etc).
 * The one thing you might take a look at is the signal-handler code here.
 *
 *   NOTE!!  Task 0 is the 'idle' task, which gets called when no other
 * tasks can run. It can not be killed, and it cannot sleep. The 'state'
 * information in task[0] is never used.
 */
void schedule(void)
{
        int i,next,c;
        struct task_struct ** p;

/* check alarm, wake up any interruptible tasks that have got a signal */

        for(p = &LAST_TASK ; p > &FIRST_TASK ; --p)
                if (*p) {
                        if ((*p)->alarm && (*p)->alarm < jiffies) {
                                        (*p)->signal |= (1<<(SIGALRM-1));
                                        (*p)->alarm = 0;
                                }
                        if (((*p)->signal & ~(_BLOCKABLE & (*p)->blocked)) &&
                        (*p)->state==TASK_INTERRUPTIBLE)
                                (*p)->state=TASK_RUNNING;
                }

/* this is the scheduler proper: */

        while (1) {
                c = -1;
                next = 0;
                i = NR_TASKS;
                p = &task[NR_TASKS];
                while (--i) {
                        if (!*--p)
                                continue;
                        if ((*p)->state == TASK_RUNNING && (*p)->counter > c)
                                c = (*p)->counter, next = i;
                }
                if (c) break;
                for(p = &LAST_TASK ; p > &FIRST_TASK ; --p)
                        if (*p)
                                (*p)->counter = ((*p)->counter >> 1) +
```

```
                                                (*p)->priority;
        }
        switch_to(next);
}

int sys_pause(void)
{
        current->state = TASK_INTERRUPTIBLE;
        schedule();
        return 0;
}

void sleep_on(struct task_struct **p)
{
        struct task_struct *tmp;

        if (!p)
                return;
        if (current == &(init_task.task))
                panic("task[0] trying to sleep");
        tmp = *p;
        *p = current;
        current->state = TASK_UNINTERRUPTIBLE;
        schedule();
        if (tmp)
                tmp->state=0;
}

void interruptible_sleep_on(struct task_struct **p)
{
        struct task_struct *tmp;

        if (!p)
                return;
        if (current == &(init_task.task))
                panic("task[0] trying to sleep");
        tmp=*p;
        *p=current;
repeat: current->state = TASK_INTERRUPTIBLE;
        schedule();
        if (*p && *p != current) {
                (**p).state=0;
                goto repeat;
        }
        *p=NULL;
        if (tmp)
                tmp->state=0;
}

void wake_up(struct task_struct **p)
{
        if (p && *p) {
                (**p).state=0;
                *p=NULL;
        }
}

/*
 * OK, here are some floppy things that shouldn't be in the kernel
 * proper. They are here because the floppy needs a timer, and this
 * was the easiest way of doing it.
 */
static struct task_struct * wait_motor[4] = {NULL,NULL,NULL,NULL};
static int  mon_timer[4]={0,0,0,0};
static int moff_timer[4]={0,0,0,0};
unsigned char current_DOR = 0x0C;

int ticks_to_floppy_on(unsigned int nr)
{
```

```
        extern unsigned char selected;
        unsigned char mask = 0x10 << nr;

        if (nr>3)
                panic("floppy_on: nr>3");
        moff_timer[nr]=10000;           /* 100 s = very big :-) */
        cli();                          /* use floppy_off to turn it off */
        mask |= current_DOR;
        if (!selected) {
                mask &= 0xFC;
                mask |= nr;
        }
        if (mask != current_DOR) {
                outb(mask,FD_DOR);
                if ((mask ^ current_DOR) & 0xf0)
                        mon_timer[nr] = HZ/2;
                else if (mon_timer[nr] < 2)
                        mon_timer[nr] = 2;
                current_DOR = mask;
        }
        sti();
        return mon_timer[nr];
}

void floppy_on(unsigned int nr)
{
        cli();
        while (ticks_to_floppy_on(nr))
                sleep_on(nr+wait_motor);
        sti();
}

void floppy_off(unsigned int nr)
{
        moff_timer[nr]=3*HZ;
}

void do_floppy_timer(void)
{
        int i;
        unsigned char mask = 0x10;

        for (i=0 ; i<4 ; i++,mask <<= 1) {
                if (!(mask & current_DOR))
                        continue;
                if (mon_timer[i]) {
                        if (!--mon_timer[i])
                                wake_up(i+wait_motor);
                } else if (!moff_timer[i]) {
                        current_DOR &= ~mask;
                        outb(current_DOR,FD_DOR);
                } else
                        moff_timer[i]--;
        }
}

#define TIME_REQUESTS 64

static struct timer_list {
        long jiffies;
        void (*fn)();
        struct timer_list * next;
} timer_list[TIME_REQUESTS], * next_timer = NULL;

void add_timer(long jiffies, void (*fn)(void))
{
        struct timer_list * p;

        if (!fn)
```

```
                return;
        cli();
        if (jiffies <= 0)
                (fn)();
        else {
                for (p = timer_list ; p < timer_list + TIME_REQUESTS ; p++)
                        if (!p->fn)
                                break;
                if (p >= timer_list + TIME_REQUESTS)
                        panic("No more time requests free");
                p->fn = fn;
                p->jiffies = jiffies;
                p->next = next_timer;
                next_timer = p;
                while (p->next && p->next->jiffies < p->jiffies) {
                        p->jiffies -= p->next->jiffies;
                        fn = p->fn;
                        p->fn = p->next->fn;
                        p->next->fn = fn;
                        jiffies = p->jiffies;
                        p->jiffies = p->next->jiffies;
                        p->next->jiffies = jiffies;
                        p = p->next;
                }
        }
        sti();
}

void do_timer(long cpl)
{
        extern int beepcount;
        extern void sysbeepstop(void);

        if (beepcount)
                if (!--beepcount)
                        sysbeepstop();

        if (cpl)
                current->utime++;
        else
                current->stime++;

        if (next_timer) {
                next_timer->jiffies--;
                while (next_timer && next_timer->jiffies <= 0) {
                        void (*fn)(void);

                        fn = next_timer->fn;
                        next_timer->fn = NULL;
                        next_timer = next_timer->next;
                        (fn)();
                }
        }
        if (current_DOR & 0xf0)
                do_floppy_timer();
        if ((--current->counter)>0) return;
        current->counter=0;
        if (!cpl) return;
        schedule();
}

int sys_alarm(long seconds)
{
        int old = current->alarm;

        if (old)
                old = (old - jiffies) / HZ;
        current->alarm = (seconds>0)?(jiffies+HZ*seconds):0;
        return (old);
```

```
}

int sys_getpid(void)
{
        return current->pid;
}

int sys_getppid(void)
{
        return current->father;
}

int sys_getuid(void)
{
        return current->uid;
}

int sys_geteuid(void)
{
        return current->euid;
}

int sys_getgid(void)
{
        return current->gid;
}

int sys_getegid(void)
{
        return current->egid;
}

int sys_nice(long increment)
{
        if (current->priority-increment>0)
                current->priority -= increment;
        return 0;
}

void sched_init(void)
{
        int i;
        struct desc_struct * p;

        if (sizeof(struct sigaction) != 16)
                panic("Struct sigaction MUST be 16 bytes");
        set_tss_desc(gdt+FIRST_TSS_ENTRY,&(init_task.task.tss));
        set_ldt_desc(gdt+FIRST_LDT_ENTRY,&(init_task.task.ldt));
        p = gdt+2+FIRST_TSS_ENTRY;
        for(i=1;i<NR_TASKS;i++) {
                task[i] = NULL;
                p->a=p->b=0;
                p++;
                p->a=p->b=0;
                p++;
        }
/* Clear NT, so that we won't have troubles with that later on */
        __asm__("pushfl ; andl $0xffffbfff,(%esp) ; popfl");
        ltr(0);
        lldt(0);
        outb_p(0x36,0x43);              /* binary, mode 3, LSB/MSB, ch 0 */
        outb_p(LATCH & 0xff , 0x40);    /* LSB */
        outb(LATCH >> 8 , 0x40);        /* MSB */
        set_intr_gate(0x20,&timer_interrupt);
        outb(inb_p(0x21)&~0x01,0x21);
        set_system_gate(0x80,&system_call);
}
```

```
/*
 *  linux/kernel/signal.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/segment.h>

#include <signal.h>

volatile void do_exit(int error_code);

int sys_sgetmask()
{
        return current->blocked;
}

int sys_ssetmask(int newmask)
{
        int old=current->blocked;

        current->blocked = newmask & ~(1<<(SIGKILL-1));
        return old;
}

static inline void save_old(char * from,char * to)
{
        int i;

        verify_area(to, sizeof(struct sigaction));
        for (i=0 ; i< sizeof(struct sigaction) ; i++) {
                put_fs_byte(*from,to);
                from++;
                to++;
        }
}

static inline void get_new(char * from,char * to)
{
        int i;

        for (i=0 ; i< sizeof(struct sigaction) ; i++)
                *(to++) = get_fs_byte(from++);
}

int sys_signal(int signum, long handler, long restorer)
{
        struct sigaction tmp;

        if (signum<1 || signum>32 || signum==SIGKILL)
                return -1;
        tmp.sa_handler = (void (*)(int)) handler;
        tmp.sa_mask = 0;
        tmp.sa_flags = SA_ONESHOT | SA_NOMASK;
        tmp.sa_restorer = (void (*)(void)) restorer;
        handler = (long) current->sigaction[signum-1].sa_handler;
        current->sigaction[signum-1] = tmp;
        return handler;
}

int sys_sigaction(int signum, const struct sigaction * action,
        struct sigaction * oldaction)
{
        struct sigaction tmp;

        if (signum<1 || signum>32 || signum==SIGKILL)
                return -1;
```

```
        tmp = current->sigaction[signum-1];
        get_new((char *) action,
                (char *) (signum-1+current->sigaction));
        if (oldaction)
                save_old((char *) &tmp,(char *) oldaction);
        if (current->sigaction[signum-1].sa_flags & SA_NOMASK)
                current->sigaction[signum-1].sa_mask = 0;
        else
                current->sigaction[signum-1].sa_mask |= (1<<(signum-1));
        return 0;
}

void do_signal(long signr,long eax, long ebx, long ecx, long edx,
        long fs, long es, long ds,
        long eip, long cs, long eflags,
        unsigned long * esp, long ss)
{
        unsigned long sa_handler;
        long old_eip=eip;
        struct sigaction * sa = current->sigaction + signr - 1;
        int longs;
        unsigned long * tmp_esp;

        sa_handler = (unsigned long) sa->sa_handler;
        if (sa_handler==1)
                return;
        if (!sa_handler) {
                if (signr==SIGCHLD)
                        return;
                else
                        do_exit(1<<(signr-1));
        }
        if (sa->sa_flags & SA_ONESHOT)
                sa->sa_handler = NULL;
        *(&eip) = sa_handler;
        longs = (sa->sa_flags & SA_NOMASK)?7:8;
        *(&esp) -= longs;
        verify_area(esp,longs*4);
        tmp_esp=esp;
        put_fs_long((long) sa->sa_restorer,tmp_esp++);
        put_fs_long(signr,tmp_esp++);
        if (!(sa->sa_flags & SA_NOMASK))
                put_fs_long(current->blocked,tmp_esp++);
        put_fs_long(eax,tmp_esp++);
        put_fs_long(ecx,tmp_esp++);
        put_fs_long(edx,tmp_esp++);
        put_fs_long(eflags,tmp_esp++);
        put_fs_long(old_eip,tmp_esp++);
        current->blocked |= sa->sa_mask;
}
```

```
/*
 *  linux/kernel/sys.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <errno.h>

#include <linux/sched.h>
#include <linux/tty.h>
#include <linux/kernel.h>
#include <asm/segment.h>
#include <sys/times.h>
#include <sys/utsname.h>

int sys_ftime()
{
        return -ENOSYS;
}

int sys_break()
{
        return -ENOSYS;
}

int sys_ptrace()
{
        return -ENOSYS;
}

int sys_stty()
{
        return -ENOSYS;
}

int sys_gtty()
{
        return -ENOSYS;
}

int sys_rename()
{
        return -ENOSYS;
}

int sys_prof()
{
        return -ENOSYS;
}

int sys_setregid(int rgid, int egid)
{
        if (rgid>0) {
                if ((current->gid == rgid) ||
                    suser())
                        current->gid = rgid;
                else
                        return(-EPERM);
        }
        if (egid>0) {
                if ((current->gid == egid) ||
                    (current->egid == egid) ||
                    (current->sgid == egid) ||
                    suser())
                        current->egid = egid;
                else
                        return(-EPERM);
        }
        return 0;
```

```
}

int sys_setgid(int gid)
{
        return(sys_setregid(gid, gid));
}

int sys_acct()
{
        return -ENOSYS;
}

int sys_phys()
{
        return -ENOSYS;
}

int sys_lock()
{
        return -ENOSYS;
}

int sys_mpx()
{
        return -ENOSYS;
}

int sys_ulimit()
{
        return -ENOSYS;
}

int sys_time(long * tloc)
{
        int i;

        i = CURRENT_TIME;
        if (tloc) {
                verify_area(tloc,4);
                put_fs_long(i,(unsigned long *)tloc);
        }
        return i;
}

/*
 * Unprivileged users may change the real user id to the effective uid
 * or vice versa.
 */
int sys_setreuid(int ruid, int euid)
{
        int old_ruid = current->uid;

        if (ruid>0) {
                if ((current->euid==ruid) ||
                    (old_ruid == ruid) ||
                    suser())
                        current->uid = ruid;
                else
                        return(-EPERM);
        }
        if (euid>0) {
                if ((old_ruid == euid) ||
                    (current->euid == euid) ||
                    suser())
                        current->euid = euid;
                else {
                        current->uid = old_ruid;
                        return(-EPERM);
                }
```

```
        }
        return 0;
}

int sys_setuid(int uid)
{
        return(sys_setreuid(uid, uid));
}

int sys_stime(long * tptr)
{
        if (!suser())
                return -EPERM;
        startup_time = get_fs_long((unsigned long *)tptr) - jiffies/HZ;
        return 0;
}

int sys_times(struct tms * tbuf)
{
        if (tbuf) {
                verify_area(tbuf,sizeof *tbuf);
                put_fs_long(current->utime,(unsigned long *)&tbuf->tms_utime);
                put_fs_long(current->stime,(unsigned long *)&tbuf->tms_stime);
                put_fs_long(current->cutime,(unsigned long *)&tbuf->tms_cutime);
                put_fs_long(current->cstime,(unsigned long *)&tbuf->tms_cstime);
        }
        return jiffies;
}

int sys_brk(unsigned long end_data_seg)
{
        if (end_data_seg >= current->end_code &&
            end_data_seg < current->start_stack - 16384)
                current->brk = end_data_seg;
        return current->brk;
}

/*
 * This needs some heave checking ...
 * I just haven't get the stomach for it. I also don't fully
 * understand sessions/pgrp etc. Let somebody who does explain it.
 */
int sys_setpgid(int pid, int pgid)
{
        int i;

        if (!pid)
                pid = current->pid;
        if (!pgid)
                pgid = current->pid;
        for (i=0 ; i<NR_TASKS ; i++)
                if (task[i] && task[i]->pid==pid) {
                        if (task[i]->leader)
                                return -EPERM;
                        if (task[i]->session != current->session)
                                return -EPERM;
                        task[i]->pgrp = pgid;
                        return 0;
                }
        return -ESRCH;
}

int sys_getpgrp(void)
{
        return current->pgrp;
}

int sys_setsid(void)
{
```

```
        if (current->leader && !suser())
                return -EPERM;
        current->leader = 1;
        current->session = current->pgrp = current->pid;
        current->tty = -1;
        return current->pgrp;
}

int sys_uname(struct utsname * name)
{
        static struct utsname thisname = {
                "linux .0","nodename","release ","version ","machine "
        };
        int i;

        if (!name) return -ERROR;
        verify_area(name,sizeof *name);
        for(i=0;i<sizeof *name;i++)
                put_fs_byte(((char *) &thisname)[i],i+(char *) name);
        return 0;
}

int sys_umask(int mask)
{
        int old = current->umask;

        current->umask = mask & 0777;
        return (old);
}
```

```
/*
 *  linux/kernel/system_call.s
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 *  system_call.s  contains the system-call low-level handling routines.
 * This also contains the timer-interrupt handler, as some of the code is
 * the same. The hd- and flopppy-interrupts are also here.
 *
 * NOTE: This code handles signal-recognition, which happens every time
 * after a timer-interrupt and after each system call. Ordinary interrupts
 * don't handle signal-recognition, as that would clutter them up totally
 * unnecessarily.
 *
 * Stack layout in 'ret_from_system_call':
 *
 *       0(%esp) - %eax
 *       4(%esp) - %ebx
 *       8(%esp) - %ecx
 *       C(%esp) - %edx
 *      10(%esp) - %fs
 *      14(%esp) - %es
 *      18(%esp) - %ds
 *      1C(%esp) - %eip
 *      20(%esp) - %cs
 *      24(%esp) - %eflags
 *      28(%esp) - %oldesp
 *      2C(%esp) - %oldss
 */

SIG_CHLD        = 17

EAX             = 0x00
EBX             = 0x04
ECX             = 0x08
EDX             = 0x0C
FS              = 0x10
ES              = 0x14
DS              = 0x18
EIP             = 0x1C
CS              = 0x20
EFLAGS          = 0x24
OLDESP          = 0x28
OLDSS           = 0x2C

state   = 0             # these are offsets into the task-struct.
counter = 4
priority = 8
signal  = 12
sigaction = 16          # MUST be 16 (=len of sigaction)
blocked = (33*16)

# offsets within sigaction
sa_handler = 0
sa_mask = 4
sa_flags = 8
sa_restorer = 12

nr_system_calls = 72

/*
 * Ok, I get parallel printer interrupts while using the floppy for some
 * strange reason. Urgel. Now I just ignore them.
 */
.globl _system_call,_sys_fork,_timer_interrupt,_sys_execve
.globl _hd_interrupt,_floppy_interrupt,_parallel_interrupt
.globl _device_not_available, _coprocessor_error
```

```
.align 2
bad_sys_call:
        movl $-1,%eax
        iret
.align 2
reschedule:
        pushl $ret_from_sys_call
        jmp _schedule
.align 2
_system_call:
        cmpl $nr_system_calls-1,%eax
        ja bad_sys_call
        push %ds
        push %es
        push %fs
        pushl %edx
        pushl %ecx              # push %ebx,%ecx,%edx as parameters
        pushl %ebx              # to the system call
        movl $0x10,%edx         # set up ds,es to kernel space
        mov %dx,%ds
        mov %dx,%es
        movl $0x17,%edx         # fs points to local data space
        mov %dx,%fs
        call _sys_call_table(,%eax,4)
        pushl %eax
        movl _current,%eax
        cmpl $0,state(%eax)             # state
        jne reschedule
        cmpl $0,counter(%eax)           # counter
        je reschedule
ret_from_sys_call:
        movl _current,%eax              # task[0] cannot have signals
        cmpl _task,%eax
        je 3f
        cmpw $0x0f,CS(%esp)             # was old code segment supervisor ?
        jne 3f
        cmpw $0x17,OLDSS(%esp)          # was stack segment = 0x17 ?
        jne 3f
        movl signal(%eax),%ebx
        movl blocked(%eax),%ecx
        notl %ecx
        andl %ebx,%ecx
        bsfl %ecx,%ecx
        je 3f
        btrl %ecx,%ebx
        movl %ebx,signal(%eax)
        incl %ecx
        pushl %ecx
        call _do_signal
        popl %eax
3:      popl %eax
        popl %ebx
        popl %ecx
        popl %edx
        pop %fs
        pop %es
        pop %ds
        iret

.align 2
_coprocessor_error:
        push %ds
        push %es
        push %fs
        pushl %edx
        pushl %ecx
        pushl %ebx
        pushl %eax
```

```
        movl $0x10,%eax
        mov %ax,%ds
        mov %ax,%es
        movl $0x17,%eax
        mov %ax,%fs
        pushl $ret_from_sys_call
        jmp _math_error

.align 2
_device_not_available:
        push %ds
        push %es
        push %fs
        pushl %edx
        pushl %ecx
        pushl %ebx
        pushl %eax
        movl $0x10,%eax
        mov %ax,%ds
        mov %ax,%es
        movl $0x17,%eax
        mov %ax,%fs
        pushl $ret_from_sys_call
        clts                            # clear TS so that we can use math
        movl %cr0,%eax
        testl $0x4,%eax                 # EM (math emulation bit)
        je _math_state_restore
        pushl %ebp
        pushl %esi
        pushl %edi
        call _math_emulate
        popl %edi
        popl %esi
        popl %ebp
        ret

.align 2
_timer_interrupt:
        push %ds                # save ds,es and put kernel data space
        push %es                # into them. %fs is used by _system_call
        push %fs
        pushl %edx              # we save %eax,%ecx,%edx as gcc doesn't
        pushl %ecx              # save those across function calls. %ebx
        pushl %ebx              # is saved as we use that in ret_sys_call
        pushl %eax
        movl $0x10,%eax
        mov %ax,%ds
        mov %ax,%es
        movl $0x17,%eax
        mov %ax,%fs
        incl _jiffies
        movb $0x20,%al          # EOI to interrupt controller #1
        outb %al,$0x20
        movl CS(%esp),%eax
        andl $3,%eax            # %eax is CPL (0 or 3, 0=supervisor)
        pushl %eax
        call _do_timer          # 'do_timer(long CPL)' does everything from
        addl $4,%esp            # task switching to accounting ...
        jmp ret_from_sys_call

.align 2
_sys_execve:
        lea EIP(%esp),%eax
        pushl %eax
        call _do_execve
        addl $4,%esp
        ret

.align 2
```

```
_sys_fork:
        call _find_empty_process
        testl %eax,%eax
        js 1f
        push %gs
        pushl %esi
        pushl %edi
        pushl %ebp
        pushl %eax
        call _copy_process
        addl $20,%esp
1:      ret

_hd_interrupt:
        pushl %eax
        pushl %ecx
        pushl %edx
        push %ds
        push %es
        push %fs
        movl $0x10,%eax
        mov %ax,%ds
        mov %ax,%es
        movl $0x17,%eax
        mov %ax,%fs
        movb $0x20,%al
        outb %al,$0xA0          # EOI to interrupt controller #1
        jmp 1f                  # give port chance to breathe
1:      jmp 1f
1:      xorl %edx,%edx
        xchgl _do_hd,%edx
        testl %edx,%edx
        jne 1f
        movl $_unexpected_hd_interrupt,%edx
1:      outb %al,$0x20
        call *%edx              # "interesting" way of handling intr.
        pop %fs
        pop %es
        pop %ds
        popl %edx
        popl %ecx
        popl %eax
        iret

_floppy_interrupt:
        pushl %eax
        pushl %ecx
        pushl %edx
        push %ds
        push %es
        push %fs
        movl $0x10,%eax
        mov %ax,%ds
        mov %ax,%es
        movl $0x17,%eax
        mov %ax,%fs
        movb $0x20,%al
        outb %al,$0x20          # EOI to interrupt controller #1
        xorl %eax,%eax
        xchgl _do_floppy,%eax
        testl %eax,%eax
        jne 1f
        movl $_unexpected_floppy_interrupt,%eax
1:      call *%eax              # "interesting" way of handling intr.
        pop %fs
        pop %es
        pop %ds
        popl %edx
        popl %ecx
```

```
        popl %eax
        iret

_parallel_interrupt:
        pushl %eax
        movb $0x20,%al
        outb %al,$0x20
        popl %eax
        iret
```

```
/*
 *  linux/kernel/traps.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * 'Traps.c' handles hardware traps and faults after we have saved some
 * state in 'asm.s'. Currently mostly a debugging-aid, will be extended
 * to mainly kill the offending process (probably by giving it a signal,
 * but possibly by killing it outright if necessary).
 */
#include <string.h>

#include <linux/head.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/system.h>
#include <asm/segment.h>
#include <asm/io.h>

#define get_seg_byte(seg,addr) ({ \
register char __res; \
__asm__("push %%fs;mov %%ax,%%fs;movb %%fs:%2,%%al;pop %%fs" \
        :"=a" (__res):"0" (seg),"m" (*(addr))); \
__res;})

#define get_seg_long(seg,addr) ({ \
register unsigned long __res; \
__asm__("push %%fs;mov %%ax,%%fs;movl %%fs:%2,%%eax;pop %%fs" \
        :"=a" (__res):"0" (seg),"m" (*(addr))); \
__res;})

#define _fs() ({ \
register unsigned short __res; \
__asm__("mov %%fs,%%ax":"=a" (__res):); \
__res;})

int do_exit(long code);

void page_exception(void);

void divide_error(void);
void debug(void);
void nmi(void);
void int3(void);
void overflow(void);
void bounds(void);
void invalid_op(void);
void device_not_available(void);
void double_fault(void);
void coprocessor_segment_overrun(void);
void invalid_TSS(void);
void segment_not_present(void);
void stack_segment(void);
void general_protection(void);
void page_fault(void);
void coprocessor_error(void);
void reserved(void);
void parallel_interrupt(void);
void irq13(void);

static void die(char * str,long esp_ptr,long nr)
{
        long * esp = (long *) esp_ptr;
        int i;

        printk("%s: %04x\n\r",str,nr&0xffff);
        printk("EIP:\t%04x:%p\nEFLAGS:\t%p\nESP:\t%04x:%p\n",
```

```
                esp[1],esp[0],esp[2],esp[4],esp[3]);
        printk("fs: %04x\n",_fs());
        printk("base: %p, limit: %p\n",get_base(current->ldt[1]),get_limit(0x17));
        if (esp[4] == 0x17) {
                printk("Stack: ");
                for (i=0;i<4;i++)
                        printk("%p ",get_seg_long(0x17,i+(long *)esp[3]));
                printk("\n");
        }
        str(i);
        printk("Pid: %d, process nr: %d\n\r",current->pid,0xffff & i);
        for(i=0;i<10;i++)
                printk("%02x ",0xff & get_seg_byte(esp[1],(i+(char *)esp[0])));
        printk("\n\r");
        do_exit(11);            /* play segment exception */
}

void do_double_fault(long esp, long error_code)
{
        die("double fault",esp,error_code);
}

void do_general_protection(long esp, long error_code)
{
        die("general protection",esp,error_code);
}

void do_divide_error(long esp, long error_code)
{
        die("divide error",esp,error_code);
}

void do_int3(long * esp, long error_code,
                long fs,long es,long ds,
                long ebp,long esi,long edi,
                long edx,long ecx,long ebx,long eax)
{
        int tr;

        __asm__("str %%ax":"=a" (tr):"0" (0));
        printk("eax\t\tebx\t\tecx\t\tedx\n\r%8x\t%8x\t%8x\t%8x\n\r",
                eax,ebx,ecx,edx);
        printk("esi\t\tedi\t\tebp\t\tesp\n\r%8x\t%8x\t%8x\t%8x\n\r",
                esi,edi,ebp,(long) esp);
        printk("\n\rds\tes\tfs\ttr\n\r%4x\t%4x\t%4x\t%4x\n\r",
                ds,es,fs,tr);
        printk("EIP: %8x   CS: %4x  EFLAGS: %8x\n\r",esp[0],esp[1],esp[2]);
}

void do_nmi(long esp, long error_code)
{
        die("nmi",esp,error_code);
}

void do_debug(long esp, long error_code)
{
        die("debug",esp,error_code);
}

void do_overflow(long esp, long error_code)
{
        die("overflow",esp,error_code);
}

void do_bounds(long esp, long error_code)
{
        die("bounds",esp,error_code);
}
```

```
void do_invalid_op(long esp, long error_code)
{
        die("invalid operand",esp,error_code);
}

void do_device_not_available(long esp, long error_code)
{
        die("device not available",esp,error_code);
}

void do_coprocessor_segment_overrun(long esp, long error_code)
{
        die("coprocessor segment overrun",esp,error_code);
}

void do_invalid_TSS(long esp,long error_code)
{
        die("invalid TSS",esp,error_code);
}

void do_segment_not_present(long esp,long error_code)
{
        die("segment not present",esp,error_code);
}

void do_stack_segment(long esp,long error_code)
{
        die("stack segment",esp,error_code);
}

void do_coprocessor_error(long esp, long error_code)
{
        if (last_task_used_math != current)
                return;
        die("coprocessor error",esp,error_code);
}

void do_reserved(long esp, long error_code)
{
        die("reserved (15,17-47) error",esp,error_code);
}

void trap_init(void)
{
        int i;

        set_trap_gate(0,&divide_error);
        set_trap_gate(1,&debug);
        set_trap_gate(2,&nmi);
        set_system_gate(3,&int3);       /* int3-5 can be called from all */
        set_system_gate(4,&overflow);
        set_system_gate(5,&bounds);
        set_trap_gate(6,&invalid_op);
        set_trap_gate(7,&device_not_available);
        set_trap_gate(8,&double_fault);
        set_trap_gate(9,&coprocessor_segment_overrun);
        set_trap_gate(10,&invalid_TSS);
        set_trap_gate(11,&segment_not_present);
        set_trap_gate(12,&stack_segment);
        set_trap_gate(13,&general_protection);
        set_trap_gate(14,&page_fault);
        set_trap_gate(15,&reserved);
        set_trap_gate(16,&coprocessor_error);
        for (i=17;i<48;i++)
                set_trap_gate(i,&reserved);
        set_trap_gate(45,&irq13);
        outb_p(inb_p(0x21)&0xfb,0x21);
        outb(inb_p(0xA1)&0xdf,0xA1);
```

```
        set_trap_gate(39,&parallel_interrupt);
}
```

```
/*
 *  linux/kernel/vsprintf.c
 *
 *  (C) 1991  Linus Torvalds
 */

/* vsprintf.c -- Lars Wirzenius & Linus Torvalds. */
/*
 * Wirzenius wrote this portably, Torvalds fucked it up :-)
 */

#include <stdarg.h>
#include <string.h>

/* we use this so that we can do without the ctype library */
#define is_digit(c)     ((c) >= '0' && (c) <= '9')

static int skip_atoi(const char **s)
{
        int i=0;

        while (is_digit(**s))
                i = i*10 + *((*s)++) - '0';
        return i;
}

#define ZEROPAD 1               /* pad with zero */
#define SIGN    2               /* unsigned/signed long */
#define PLUS    4               /* show plus */
#define SPACE   8               /* space if plus */
#define LEFT    16              /* left justified */
#define SPECIAL 32              /* 0x */
#define SMALL   64              /* use 'abcdef' instead of 'ABCDEF' */

#define do_div(n,base) ({ \
int __res; \
__asm__("divl %4":"=a" (n),"=d" (__res):"0" (n),"1" (0),"r" (base)); \
__res; })

static char * number(char * str, int num, int base, int size, int precision
        ,int type)
{
        char c,sign,tmp[36];
        const char *digits="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        int i;

        if (type&SMALL) digits="0123456789abcdefghijklmnopqrstuvwxyz";
        if (type&LEFT) type &= ~ZEROPAD;
        if (base<2 || base>36)
                return 0;
        c = (type & ZEROPAD) ? '0' : ' ' ;
        if (type&SIGN && num<0) {
                sign='-';
                num = -num;
        } else
                sign=(type&PLUS) ? '+' : ((type&SPACE) ? ' ' : 0);
        if (sign) size--;
        if (type&SPECIAL)
                if (base==16) size -= 2;
                else if (base==8) size--;
        i=0;
        if (num==0)
                tmp[i++]='0';
        else while (num!=0)
                tmp[i++]=digits[do_div(num,base)];
        if (i>precision) precision=i;
        size -= precision;
        if (!(type&(ZEROPAD+LEFT)))
                while(size-->0)
```

```
                        *str++ = ' ';
        if (sign)
                *str++ = sign;
        if (type&SPECIAL)
                if (base==8)
                        *str++ = '0';
                else if (base==16) {
                        *str++ = '0';
                        *str++ = digits[33];
                }
        if (!(type&LEFT))
                while(size-->0)
                        *str++ = c;
        while(i<precision--)
                *str++ = '0';
        while(i-->0)
                *str++ = tmp[i];
        while(size-->0)
                *str++ = ' ';
        return str;
}

int vsprintf(char *buf, const char *fmt, va_list args)
{
        int len;
        int i;
        char * str;
        char *s;
        int *ip;

        int flags;              /* flags to number() */

        int field_width;        /* width of output field */
        int precision;          /* min. # of digits for integers; max
                                   number of chars for from string */
        int qualifier;          /* 'h', 'l', or 'L' for integer fields */

        for (str=buf ; *fmt ; ++fmt) {
                if (*fmt != '%') {
                        *str++ = *fmt;
                        continue;
                }

                /* process flags */
                flags = 0;
                repeat:
                        ++fmt;          /* this also skips first '%' */
                        switch (*fmt) {
                                case '-': flags |= LEFT; goto repeat;
                                case '+': flags |= PLUS; goto repeat;
                                case ' ': flags |= SPACE; goto repeat;
                                case '#': flags |= SPECIAL; goto repeat;
                                case '0': flags |= ZEROPAD; goto repeat;
                                }

                /* get field width */
                field_width = -1;
                if (is_digit(*fmt))
                        field_width = skip_atoi(&fmt);
                else if (*fmt == '*') {
                        /* it's the next argument */
                        field_width = va_arg(args, int);
                        if (field_width < 0) {
                                field_width = -field_width;
                                flags |= LEFT;
                        }
                }

                /* get the precision */
```

```
precision = -1;
if (*fmt == '.') {
        ++fmt;
        if (is_digit(*fmt))
                precision = skip_atoi(&fmt);
        else if (*fmt == '*') {
                /* it's the next argument */
                precision = va_arg(args, int);
        }
        if (precision < 0)
                precision = 0;
}

/* get the conversion qualifier */
qualifier = -1;
if (*fmt == 'h' || *fmt == 'l' || *fmt == 'L') {
        qualifier = *fmt;
        ++fmt;
}

switch (*fmt) {
case 'c':
        if (!(flags & LEFT))
                while (--field_width > 0)
                        *str++ = ' ';
        *str++ = (unsigned char) va_arg(args, int);
        while (--field_width > 0)
                *str++ = ' ';
        break;

case 's':
        s = va_arg(args, char *);
        len = strlen(s);
        if (precision < 0)
                precision = len;
        else if (len > precision)
                len = precision;

        if (!(flags & LEFT))
                while (len < field_width--)
                        *str++ = ' ';
        for (i = 0; i < len; ++i)
                *str++ = *s++;
        while (len < field_width--)
                *str++ = ' ';
        break;

case 'o':
        str = number(str, va_arg(args, unsigned long), 8,
                field_width, precision, flags);
        break;

case 'p':
        if (field_width == -1) {
                field_width = 8;
                flags |= ZEROPAD;
        }
        str = number(str,
                (unsigned long) va_arg(args, void *), 16,
                field_width, precision, flags);
        break;

case 'x':
        flags |= SMALL;
case 'X':
        str = number(str, va_arg(args, unsigned long), 16,
                field_width, precision, flags);
        break;
```

```
                case 'd':
                case 'i':
                        flags |= SIGN;
                case 'u':
                        str = number(str, va_arg(args, unsigned long), 10,
                                field_width, precision, flags);
                        break;

                case 'n':
                        ip = va_arg(args, int *);
                        *ip = (str - buf);
                        break;

                default:
                        if (*fmt != '%')
                                *str++ = '%';
                        if (*fmt)
                                *str++ = *fmt;
                        else
                                --fmt;
                        break;
                }
        }
        *str = '\0';
        return str-buf;
}
```

```
#
# Makefile for the FREAX-kernel block device drivers.
#
# Note! Dependencies are done automagically by 'make dep', which also
# removes any old dependencies. DON'T put your own dependencies here
# unless it's something special (ie not a .c file).
#

AR      =gar
AS      =gas
LD      =gld
LDFLAGS =-s -x
CC      =gcc
CFLAGS  =-Wall -O -fstrength-reduce -fomit-frame-pointer -fcombine-regs \
        -finline-functions -mstring-insns -nostdinc -I../../include
CPP     =gcc -E -nostdinc -I../../include

.c.s:
        $(CC) $(CFLAGS) \
        -S -o $*.s $<
.s.o:
        $(AS) -c -o $*.o $<
.c.o:
        $(CC) $(CFLAGS) \
        -c -o $*.o $<

OBJS  = ll_rw_blk.o floppy.o hd.o ramdisk.o

blk_drv.a: $(OBJS)
        $(AR) rcs blk_drv.a $(OBJS)
        sync

clean:
        rm -f core *.o *.a tmp_make
        for i in *.c;do rm -f `basename $$i .c`.s;done

dep:
        sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
        (for i in *.c;do echo -n `echo $$i | sed 's,\.c,\.s,'`" "; \
                $(CPP) -M $$i;done) >> tmp_make
        cp tmp_make Makefile

### Dependencies:
floppy.s floppy.o : floppy.c ../../include/linux/sched.h ../../include/linux/head.h \
  ../../include/linux/fs.h ../../include/sys/types.h ../../include/linux/mm.h \
  ../../include/signal.h ../../include/linux/kernel.h \
  ../../include/linux/fdreg.h ../../include/asm/system.h \
  ../../include/asm/io.h ../../include/asm/segment.h blk.h
hd.s hd.o : hd.c ../../include/linux/config.h ../../include/linux/sched.h \
  ../../include/linux/head.h ../../include/linux/fs.h \
  ../../include/sys/types.h ../../include/linux/mm.h ../../include/signal.h \
  ../../include/linux/kernel.h ../../include/linux/hdreg.h \
  ../../include/asm/system.h ../../include/asm/io.h \
  ../../include/asm/segment.h blk.h
ll_rw_blk.s ll_rw_blk.o : ll_rw_blk.c ../../include/errno.h ../../include/linux/sched.h \
  ../../include/linux/head.h ../../include/linux/fs.h \
  ../../include/sys/types.h ../../include/linux/mm.h ../../include/signal.h \
  ../../include/linux/kernel.h ../../include/asm/system.h blk.h
```

```
#ifndef _BLK_H
#define _BLK_H

#define NR_BLK_DEV	7
/*
 * NR_REQUEST is the number of entries in the request-queue.
 * NOTE that writes may use only the low 2/3 of these: reads
 * take precedence.
 *
 * 32 seems to be a reasonable number: enough to get some benefit
 * from the elevator-mechanism, but not so much as to lock a lot of
 * buffers when they are in the queue. 64 seems to be too many (easily
 * long pauses in reading when heavy writing/syncing is going on)
 */
#define NR_REQUEST	32

/*
 * Ok, this is an expanded form so that we can use the same
 * request for paging requests when that is implemented. In
 * paging, 'bh' is NULL, and 'waiting' is used to wait for
 * read/write completion.
 */
struct request {
	int dev;		/* -1 if no request */
	int cmd;		/* READ or WRITE */
	int errors;
	unsigned long sector;
	unsigned long nr_sectors;
	char * buffer;
	struct task_struct * waiting;
	struct buffer_head * bh;
	struct request * next;
};

/*
 * This is used in the elevator algorithm: Note that
 * reads always go before writes. This is natural: reads
 * are much more time-critical than writes.
 */
#define IN_ORDER(s1,s2) \
((s1)->cmd<(s2)->cmd || (s1)->cmd==(s2)->cmd && \
((s1)->dev < (s2)->dev || ((s1)->dev == (s2)->dev && \
(s1)->sector < (s2)->sector)))

struct blk_dev_struct {
	void (*request_fn)(void);
	struct request * current_request;
};

extern struct blk_dev_struct blk_dev[NR_BLK_DEV];
extern struct request request[NR_REQUEST];
extern struct task_struct * wait_for_request;

#ifdef MAJOR_NR

/*
 * Add entries as needed. Currently the only block devices
 * supported are hard-disks and floppies.
 */

#if (MAJOR_NR == 1)
/* ram disk */
#define DEVICE_NAME "ramdisk"
#define DEVICE_REQUEST do_rd_request
#define DEVICE_NR(device) ((device) & 7)
#define DEVICE_ON(device)
#define DEVICE_OFF(device)

#elif (MAJOR_NR == 2)
```

```
/* floppy */
#define DEVICE_NAME "floppy"
#define DEVICE_INTR do_floppy
#define DEVICE_REQUEST do_fd_request
#define DEVICE_NR(device) ((device) & 3)
#define DEVICE_ON(device) floppy_on(DEVICE_NR(device))
#define DEVICE_OFF(device) floppy_off(DEVICE_NR(device))

#elif (MAJOR_NR == 3)
/* harddisk */
#define DEVICE_NAME "harddisk"
#define DEVICE_INTR do_hd
#define DEVICE_REQUEST do_hd_request
#define DEVICE_NR(device) (MINOR(device)/5)
#define DEVICE_ON(device)
#define DEVICE_OFF(device)

#elif
/* unknown blk device */
#error "unknown blk device"

#endif

#define CURRENT (blk_dev[MAJOR_NR].current_request)
#define CURRENT_DEV DEVICE_NR(CURRENT->dev)

#ifdef DEVICE_INTR
void (*DEVICE_INTR)(void) = NULL;
#endif
static void (DEVICE_REQUEST)(void);

extern inline void unlock_buffer(struct buffer_head * bh)
{
        if (!bh->b_lock)
                printk(DEVICE_NAME ": free buffer being unlocked\n");
        bh->b_lock=0;
        wake_up(&bh->b_wait);
}

extern inline void end_request(int uptodate)
{
        DEVICE_OFF(CURRENT->dev);
        if (CURRENT->bh) {
                CURRENT->bh->b_uptodate = uptodate;
                unlock_buffer(CURRENT->bh);
        }
        if (!uptodate) {
                printk(DEVICE_NAME " I/O error\n\r");
                printk("dev %04x, block %d\n\r",CURRENT->dev,
                        CURRENT->bh->b_blocknr);
        }
        wake_up(&CURRENT->waiting);
        wake_up(&wait_for_request);
        CURRENT->dev = -1;
        CURRENT = CURRENT->next;
}

#define INIT_REQUEST \
repeat: \
        if (!CURRENT) \
                return; \
        if (MAJOR(CURRENT->dev) != MAJOR_NR) \
                panic(DEVICE_NAME ": request list destroyed"); \
        if (CURRENT->bh) { \
                if (!CURRENT->bh->b_lock) \
                        panic(DEVICE_NAME ": block not locked"); \
        }

#endif
```

```
#endif
```

```
/*
 *  linux/kernel/floppy.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * 02.12.91 - Changed to static variables to indicate need for reset
 * and recalibrate. This makes some things easier (output_byte reset
 * checking etc), and means less interrupt jumping in case of errors,
 * so the code is hopefully easier to understand.
 */

/*
 * This file is certainly a mess. I've tried my best to get it working,
 * but I don't like programming floppies, and I have only one anyway.
 * Urgel. I should check for more errors, and do more graceful error
 * recovery. Seems there are problems with several drives. I've tried to
 * correct them. No promises.
 */

/*
 * As with hd.c, all routines within this file can (and will) be called
 * by interrupts, so extreme caution is needed. A hardware interrupt
 * handler may not sleep, or a kernel panic will happen. Thus I cannot
 * call "floppy-on" directly, but have to set a special timer interrupt
 * etc.
 *
 * Also, I'm not certain this works on more than 1 floppy. Bugs may
 * abund.
 */

#include <linux/sched.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/fdreg.h>
#include <asm/system.h>
#include <asm/io.h>
#include <asm/segment.h>

#define MAJOR_NR 2
#include "blk.h"

static int recalibrate = 0;
static int reset = 0;
static int seek = 0;

extern unsigned char current_DOR;

#define immoutb_p(val,port) \
__asm__("outb %0,%1\n\tjmp 1f\n1:\tjmp 1f\n1:"::"a" ((char) (val)),"i" (port))

#define TYPE(x) ((x)>>2)
#define DRIVE(x) ((x)&0x03)
/*
 * Note that MAX_ERRORS=8 doesn't imply that we retry every bad read
 * max 8 times - some types of errors increase the errorcount by 2,
 * so we might actually retry only 5-6 times before giving up.
 */
#define MAX_ERRORS 8

/*
 * globals used by 'result()'
 */
#define MAX_REPLIES 7
static unsigned char reply_buffer[MAX_REPLIES];
#define ST0 (reply_buffer[0])
#define ST1 (reply_buffer[1])
#define ST2 (reply_buffer[2])
```

```
#define ST3 (reply_buffer[3])

/*
 * This struct defines the different floppy types. Unlike minix
 * linux doesn't have a "search for right type"-type, as the code
 * for that is convoluted and weird. I've got enough problems with
 * this driver as it is.
 *
 * The 'stretch' tells if the tracks need to be boubled for some
 * types (ie 360kB diskette in 1.2MB drive etc). Others should
 * be self-explanatory.
 */
static struct floppy_struct {
        unsigned int size, sect, head, track, stretch;
        unsigned char gap,rate,spec1;
} floppy_type[] = {
        {    0, 0,0, 0,0,0x00,0x00,0x00 },      /* no testing */
        {  720, 9,2,40,0,0x2A,0x02,0xDF },      /* 360kB PC diskettes */
        { 2400,15,2,80,0,0x1B,0x00,0xDF },      /* 1.2 MB AT-diskettes */
        {  720, 9,2,40,1,0x2A,0x02,0xDF },      /* 360kB in 720kB drive */
        { 1440, 9,2,80,0,0x2A,0x02,0xDF },      /* 3.5" 720kB diskette */
        {  720, 9,2,40,1,0x23,0x01,0xDF },      /* 360kB in 1.2MB drive */
        { 1440, 9,2,80,0,0x23,0x01,0xDF },      /* 720kB in 1.2MB drive */
        { 2880,18,2,80,0,0x1B,0x00,0xCF },      /* 1.44MB diskette */
};
/*
 * Rate is 0 for 500kb/s, 2 for 300kbps, 1 for 250kbps
 * Spec1 is 0xSH, where S is stepping rate (F=1ms, E=2ms, D=3ms etc),
 * H is head unload time (1=16ms, 2=32ms, etc)
 *
 * Spec2 is (HLD<<1 | ND), where HLD is head load time (1=2ms, 2=4 ms etc)
 * and ND is set means no DMA. Hardcoded to 6 (HLD=6ms, use DMA).
 */

extern void floppy_interrupt(void);
extern char tmp_floppy_area[1024];

/*
 * These are global variables, as that's the easiest way to give
 * information to interrupts. They are the data used for the current
 * request.
 */
static int cur_spec1 = -1;
static int cur_rate = -1;
static struct floppy_struct * floppy = floppy_type;
static unsigned char current_drive = 0;
static unsigned char sector = 0;
static unsigned char head = 0;
static unsigned char track = 0;
static unsigned char seek_track = 0;
static unsigned char current_track = 255;
static unsigned char command = 0;
unsigned char selected = 0;
struct task_struct * wait_on_floppy_select = NULL;

void floppy_deselect(unsigned int nr)
{
        if (nr != (current_DOR & 3))
                printk("floppy_deselect: drive not selected\n\r");
        selected = 0;
        wake_up(&wait_on_floppy_select);
}

/*
 * floppy-change is never called from an interrupt, so we can relax a bit
 * here, sleep etc. Note that floppy-on tries to set current_DOR to point
 * to the desired drive, but it will probably not survive the sleep if
 * several floppies are used at the same time: thus the loop.
 */
```

```
int floppy_change(unsigned int nr)
{
repeat:
        floppy_on(nr);
        while ((current_DOR & 3) != nr && selected)
                interruptible_sleep_on(&wait_on_floppy_select);
        if ((current_DOR & 3) != nr)
                goto repeat;
        if (inb(FD_DIR) & 0x80) {
                floppy_off(nr);
                return 1;
        }
        floppy_off(nr);
        return 0;
}

#define copy_buffer(from,to) \
__asm__("cld ; rep ; movsl" \
        ::"c" (BLOCK_SIZE/4),"S" ((long)(from)),"D" ((long)(to)) \
        :"cx","di","si")

static void setup_DMA(void)
{
        long addr = (long) CURRENT->buffer;

        cli();
        if (addr >= 0x100000) {
                addr = (long) tmp_floppy_area;
                if (command == FD_WRITE)
                        copy_buffer(CURRENT->buffer,tmp_floppy_area);
        }
/* mask DMA 2 */
        immoutb_p(4|2,10);
/* output command byte. I don't know why, but everyone (minix, */
/* sanches & canton) output this twice, first to 12 then to 11 */
        __asm__("outb %%al,$12\n\tjmp 1f\n1:\tjmp 1f\n1:\t"
        "outb %%al,$11\n\tjmp 1f\n1:\tjmp 1f\n1:"::
        "a" ((char) ((command == FD_READ)?DMA_READ:DMA_WRITE)));
/* 8 low bits of addr */
        immoutb_p(addr,4);
        addr >>= 8;
/* bits 8-15 of addr */
        immoutb_p(addr,4);
        addr >>= 8;
/* bits 16-19 of addr */
        immoutb_p(addr,0x81);
/* low 8 bits of count-1 (1024-1=0x3ff) */
        immoutb_p(0xff,5);
/* high 8 bits of count-1 */
        immoutb_p(3,5);
/* activate DMA 2 */
        immoutb_p(0|2,10);
        sti();
}

static void output_byte(char byte)
{
        int counter;
        unsigned char status;

        if (reset)
                return;
        for(counter = 0 ; counter < 10000 ; counter++) {
                status = inb_p(FD_STATUS) & (STATUS_READY | STATUS_DIR);
                if (status == STATUS_READY) {
                        outb(byte,FD_DATA);
                        return;
                }
        }
```

```
        reset = 1;
        printk("Unable to send byte to FDC\n\r");
}

static int result(void)
{
        int i = 0, counter, status;

        if (reset)
                return -1;
        for (counter = 0 ; counter < 10000 ; counter++) {
                status = inb_p(FD_STATUS)&(STATUS_DIR|STATUS_READY|STATUS_BUSY);
                if (status == STATUS_READY)
                        return i;
                if (status == (STATUS_DIR|STATUS_READY|STATUS_BUSY)) {
                        if (i >= MAX_REPLIES)
                                break;
                        reply_buffer[i++] = inb_p(FD_DATA);
                }
        }
        reset = 1;
        printk("Getstatus times out\n\r");
        return -1;
}

static void bad_flp_intr(void)
{
        CURRENT->errors++;
        if (CURRENT->errors > MAX_ERRORS) {
                floppy_deselect(current_drive);
                end_request(0);
        }
        if (CURRENT->errors > MAX_ERRORS/2)
                reset = 1;
        else
                recalibrate = 1;
}

/*
 * Ok, this interrupt is called after a DMA read/write has succeeded,
 * so we check the results, and copy any buffers.
 */
static void rw_interrupt(void)
{
        if (result() != 7 || (ST0 & 0xf8) || (ST1 & 0xbf) || (ST2 & 0x73)) {
                if (ST1 & 0x02) {
                        printk("Drive %d is write protected\n\r",current_drive);
                        floppy_deselect(current_drive);
                        end_request(0);
                } else
                        bad_flp_intr();
                do_fd_request();
                return;
        }
        if (command == FD_READ && (unsigned long)(CURRENT->buffer) >= 0x100000)
                copy_buffer(tmp_floppy_area,CURRENT->buffer);
        floppy_deselect(current_drive);
        end_request(1);
        do_fd_request();
}

inline void setup_rw_floppy(void)
{
        setup_DMA();
        do_floppy = rw_interrupt;
        output_byte(command);
        output_byte(head<<2 | current_drive);
        output_byte(track);
        output_byte(head);
```

```
        output_byte(sector);
        output_byte(2);         /* sector size = 512 */
        output_byte(floppy->sect);
        output_byte(floppy->gap);
        output_byte(0xFF);      /* sector size (0xff when n!=0 ?) */
        if (reset)
                do_fd_request();
}

/*
 * This is the routine called after every seek (or recalibrate) interrupt
 * from the floppy controller. Note that the "unexpected interrupt" routine
 * also does a recalibrate, but doesn't come here.
 */
static void seek_interrupt(void)
{
/* sense drive status */
        output_byte(FD_SENSEI);
        if (result() != 2 || (ST0 & 0xF8) != 0x20 || ST1 != seek_track) {
                bad_flp_intr();
                do_fd_request();
                return;
        }
        current_track = ST1;
        setup_rw_floppy();
}

/*
 * This routine is called when everything should be correctly set up
 * for the transfer (ie floppy motor is on and the correct floppy is
 * selected).
 */
static void transfer(void)
{
        if (cur_spec1 != floppy->spec1) {
                cur_spec1 = floppy->spec1;
                output_byte(FD_SPECIFY);
                output_byte(cur_spec1);         /* hut etc */
                output_byte(6);                 /* Head load time =6ms, DMA */
        }
        if (cur_rate != floppy->rate)
                outb_p(cur_rate = floppy->rate,FD_DCR);
        if (reset) {
                do_fd_request();
                return;
        }
        if (!seek) {
                setup_rw_floppy();
                return;
        }
        do_floppy = seek_interrupt;
        if (seek_track) {
                output_byte(FD_SEEK);
                output_byte(head<<2 | current_drive);
                output_byte(seek_track);
        } else {
                output_byte(FD_RECALIBRATE);
                output_byte(head<<2 | current_drive);
        }
        if (reset)
                do_fd_request();
}

/*
 * Special case - used after a unexpected interrupt (or reset)
 */
static void recal_interrupt(void)
{
        output_byte(FD_SENSEI);
```

```
        if (result()!=2 || (ST0 & 0xE0) == 0x60)
                reset = 1;
        else
                recalibrate = 0;
        do_fd_request();
}

void unexpected_floppy_interrupt(void)
{
        output_byte(FD_SENSEI);
        if (result()!=2 || (ST0 & 0xE0) == 0x60)
                reset = 1;
        else
                recalibrate = 1;
}

static void recalibrate_floppy(void)
{
        recalibrate = 0;
        current_track = 0;
        do_floppy = recal_interrupt;
        output_byte(FD_RECALIBRATE);
        output_byte(head<<2 | current_drive);
        if (reset)
                do_fd_request();
}

static void reset_interrupt(void)
{
        output_byte(FD_SENSEI);
        (void) result();
        output_byte(FD_SPECIFY);
        output_byte(cur_spec1);         /* hut etc */
        output_byte(6);                 /* Head load time =6ms, DMA */
        do_fd_request();
}

/*
 * reset is done by pulling bit 2 of DOR low for a while.
 */
static void reset_floppy(void)
{
        int i;

        reset = 0;
        cur_spec1 = -1;
        cur_rate = -1;
        recalibrate = 1;
        printk("Reset-floppy called\n\r");
        cli();
        do_floppy = reset_interrupt;
        outb_p(current_DOR & ~0x04,FD_DOR);
        for (i=0 ; i<100 ; i++)
                __asm__("nop");
        outb(current_DOR,FD_DOR);
        sti();
}

static void floppy_on_interrupt(void)
{
/* We cannot do a floppy-select, as that might sleep. We just force it */
        selected = 1;
        if (current_drive != (current_DOR & 3)) {
                current_DOR &= 0xFC;
                current_DOR |= current_drive;
                outb(current_DOR,FD_DOR);
                add_timer(2,&transfer);
        } else
                transfer();
```

```
}

void do_fd_request(void)
{
        unsigned int block;

        seek = 0;
        if (reset) {
                reset_floppy();
                return;
        }
        if (recalibrate) {
                recalibrate_floppy();
                return;
        }
        INIT_REQUEST;
        floppy = (MINOR(CURRENT->dev)>>2) + floppy_type;
        if (current_drive != CURRENT_DEV)
                seek = 1;
        current_drive = CURRENT_DEV;
        block = CURRENT->sector;
        if (block+2 > floppy->size) {
                end_request(0);
                goto repeat;
        }
        sector = block % floppy->sect;
        block /= floppy->sect;
        head = block % floppy->head;
        track = block / floppy->head;
        seek_track = track << floppy->stretch;
        if (seek_track != current_track)
                seek = 1;
        sector++;
        if (CURRENT->cmd == READ)
                command = FD_READ;
        else if (CURRENT->cmd == WRITE)
                command = FD_WRITE;
        else
                panic("do_fd_request: unknown command");
        add_timer(ticks_to_floppy_on(current_drive),&floppy_on_interrupt);
}

void floppy_init(void)
{
        blk_dev[MAJOR_NR].request_fn = DEVICE_REQUEST;
        set_trap_gate(0x26,&floppy_interrupt);
        outb(inb_p(0x21)&~0x40,0x21);
}
```

```
/*
 *  linux/kernel/hd.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * This is the low-level hd interrupt support. It traverses the
 * request-list, using interrupts to jump between functions. As
 * all the functions are called within interrupts, we may not
 * sleep. Special care is recommended.
 *
 *  modified by Drew Eckhardt to check nr of hd's from the CMOS.
 */

#include <linux/config.h>
#include <linux/sched.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/hdreg.h>
#include <asm/system.h>
#include <asm/io.h>
#include <asm/segment.h>

#define MAJOR_NR 3
#include "blk.h"

#define CMOS_READ(addr) ({ \
outb_p(0x80|addr,0x70); \
inb_p(0x71); \
})

/* Max read/write errors/sector */
#define MAX_ERRORS      7
#define MAX_HD          2

static void recal_intr(void);

static int recalibrate = 1;
static int reset = 1;

/*
 *  This struct defines the HD's and their types.
 */
struct hd_i_struct {
        int head,sect,cyl,wpcom,lzone,ctl;
        };
#ifdef HD_TYPE
struct hd_i_struct hd_info[] = { HD_TYPE };
#define NR_HD ((sizeof (hd_info))/(sizeof (struct hd_i_struct)))
#else
struct hd_i_struct hd_info[] = { {0,0,0,0,0,0},{0,0,0,0,0,0} };
static int NR_HD = 0;
#endif

static struct hd_struct {
        long start_sect;
        long nr_sects;
} hd[5*MAX_HD]={{0,0},};

#define port_read(port,buf,nr) \
__asm__("cld;rep;insw"::"d" (port),"D" (buf),"c" (nr):"cx","di")

#define port_write(port,buf,nr) \
__asm__("cld;rep;outsw"::"d" (port),"S" (buf),"c" (nr):"cx","si")

extern void hd_interrupt(void);
extern void rd_load(void);
```

```
/* This may be used only once, enforced by 'static int callable' */
int sys_setup(void * BIOS)
{
        static int callable = 1;
        int i,drive;
        unsigned char cmos_disks;
        struct partition *p;
        struct buffer_head * bh;

        if (!callable)
                return -1;
        callable = 0;
#ifndef HD_TYPE
        for (drive=0 ; drive<2 ; drive++) {
                hd_info[drive].cyl = *(unsigned short *) BIOS;
                hd_info[drive].head = *(unsigned char *) (2+BIOS);
                hd_info[drive].wpcom = *(unsigned short *) (5+BIOS);
                hd_info[drive].ctl = *(unsigned char *) (8+BIOS);
                hd_info[drive].lzone = *(unsigned short *) (12+BIOS);
                hd_info[drive].sect = *(unsigned char *) (14+BIOS);
                BIOS += 16;
        }
        if (hd_info[1].cyl)
                NR_HD=2;
        else
                NR_HD=1;
#endif
        for (i=0 ; i<NR_HD ; i++) {
                hd[i*5].start_sect = 0;
                hd[i*5].nr_sects = hd_info[i].head*
                                hd_info[i].sect*hd_info[i].cyl;
        }

        /*
                We querry CMOS about hard disks : it could be that
                we have a SCSI/ESDI/etc controller that is BIOS
                compatable with ST-506, and thus showing up in our
                BIOS table, but not register compatable, and therefore
                not present in CMOS.

                Furthurmore, we will assume that our ST-506 drives
                <if any> are the primary drives in the system, and
                the ones reflected as drive 1 or 2.

                The first drive is stored in the high nibble of CMOS
                byte 0x12, the second in the low nibble.  This will be
                either a 4 bit drive type or 0xf indicating use byte 0x19
                for an 8 bit type, drive 1, 0x1a for drive 2 in CMOS.

                Needless to say, a non-zero value means we have
                an AT controller hard disk for that drive.


        */

        if ((cmos_disks = CMOS_READ(0x12)) & 0xf0)
                if (cmos_disks & 0x0f)
                        NR_HD = 2;
                else
                        NR_HD = 1;
        else
                NR_HD = 0;
        for (i = NR_HD ; i < 2 ; i++) {
                hd[i*5].start_sect = 0;
                hd[i*5].nr_sects = 0;
        }
        for (drive=0 ; drive<NR_HD ; drive++) {
                if (!(bh = bread(0x300 + drive*5,0))) {
                        printk("Unable to read partition table of drive %d\n\r",
```

```
                                drive);
                        panic("");
                }
                if (bh->b_data[510] != 0x55 || (unsigned char)
                    bh->b_data[511] != 0xAA) {
                        printk("Bad partition table on drive %d\n\r",drive);
                        panic("");
                }
                p = 0x1BE + (void *)bh->b_data;
                for (i=1;i<5;i++,p++) {
                        hd[i+5*drive].start_sect = p->start_sect;
                        hd[i+5*drive].nr_sects = p->nr_sects;
                }
                brelse(bh);
        }
        if (NR_HD)
                printk("Partition table%s ok.\n\r",(NR_HD>1)?"s":"");
        rd_load();
        mount_root();
        return (0);
}

static int controller_ready(void)
{
        int retries=10000;

        while (--retries && (inb_p(HD_STATUS)&0xc0)!=0x40);
        return (retries);
}

static int win_result(void)
{
        int i=inb_p(HD_STATUS);

        if ((i & (BUSY_STAT | READY_STAT | WRERR_STAT | SEEK_STAT | ERR_STAT))
                == (READY_STAT | SEEK_STAT))
                return(0); /* ok */
        if (i&1) i=inb(HD_ERROR);
        return (1);
}

static void hd_out(unsigned int drive,unsigned int nsect,unsigned int sect,
                unsigned int head,unsigned int cyl,unsigned int cmd,
                void (*intr_addr)(void))
{
        register int port asm("dx");

        if (drive>1 || head>15)
                panic("Trying to write bad sector");
        if (!controller_ready())
                panic("HD controller not ready");
        do_hd = intr_addr;
        outb_p(hd_info[drive].ctl,HD_CMD);
        port=HD_DATA;
        outb_p(hd_info[drive].wpcom>>2,++port);
        outb_p(nsect,++port);
        outb_p(sect,++port);
        outb_p(cyl,++port);
        outb_p(cyl>>8,++port);
        outb_p(0xA0|(drive<<4)|head,++port);
        outb(cmd,++port);
}

static int drive_busy(void)
{
        unsigned int i;

        for (i = 0; i < 10000; i++)
                if (READY_STAT == (inb_p(HD_STATUS) & (BUSY_STAT|READY_STAT)))
```

```
                        break;
        i = inb(HD_STATUS);
        i &= BUSY_STAT | READY_STAT | SEEK_STAT;
        if (i == READY_STAT | SEEK_STAT)
                return(0);
        printk("HD controller times out\n\r");
        return(1);
}

static void reset_controller(void)
{
        int     i;

        outb(4,HD_CMD);
        for(i = 0; i < 100; i++) nop();
        outb(hd_info[0].ctl & 0x0f ,HD_CMD);
        if (drive_busy())
                printk("HD-controller still busy\n\r");
        if ((i = inb(HD_ERROR)) != 1)
                printk("HD-controller reset failed: %02x\n\r",i);
}

static void reset_hd(int nr)
{
        reset_controller();
        hd_out(nr,hd_info[nr].sect,hd_info[nr].sect,hd_info[nr].head-1,
                hd_info[nr].cyl,WIN_SPECIFY,&recal_intr);
}

void unexpected_hd_interrupt(void)
{
        printk("Unexpected HD interrupt\n\r");
}

static void bad_rw_intr(void)
{
        if (++CURRENT->errors >= MAX_ERRORS)
                end_request(0);
        if (CURRENT->errors > MAX_ERRORS/2)
                reset = 1;
}

static void read_intr(void)
{
        if (win_result()) {
                bad_rw_intr();
                do_hd_request();
                return;
        }
        port_read(HD_DATA,CURRENT->buffer,256);
        CURRENT->errors = 0;
        CURRENT->buffer += 512;
        CURRENT->sector++;
        if (--CURRENT->nr_sectors) {
                do_hd = &read_intr;
                return;
        }
        end_request(1);
        do_hd_request();
}

static void write_intr(void)
{
        if (win_result()) {
                bad_rw_intr();
                do_hd_request();
                return;
        }
        if (--CURRENT->nr_sectors) {
```

```
                CURRENT->sector++;
                CURRENT->buffer += 512;
                do_hd = &write_intr;
                port_write(HD_DATA,CURRENT->buffer,256);
                return;
        }
        end_request(1);
        do_hd_request();
}

static void recal_intr(void)
{
        if (win_result())
                bad_rw_intr();
        do_hd_request();
}

void do_hd_request(void)
{
        int i,r;
        unsigned int block,dev;
        unsigned int sec,head,cyl;
        unsigned int nsect;

        INIT_REQUEST;
        dev = MINOR(CURRENT->dev);
        block = CURRENT->sector;
        if (dev >= 5*NR_HD || block+2 > hd[dev].nr_sects) {
                end_request(0);
                goto repeat;
        }
        block += hd[dev].start_sect;
        dev /= 5;
        __asm__("divl %4":"=a" (block),"=d" (sec):"0" (block),"1" (0),
                "r" (hd_info[dev].sect));
        __asm__("divl %4":"=a" (cyl),"=d" (head):"0" (block),"1" (0),
                "r" (hd_info[dev].head));
        sec++;
        nsect = CURRENT->nr_sectors;
        if (reset) {
                reset = 0;
                recalibrate = 1;
                reset_hd(CURRENT_DEV);
                return;
        }
        if (recalibrate) {
                recalibrate = 0;
                hd_out(dev,hd_info[CURRENT_DEV].sect,0,0,0,
                        WIN_RESTORE,&recal_intr);
                return;
        }
        if (CURRENT->cmd == WRITE) {
                hd_out(dev,nsect,sec,head,cyl,WIN_WRITE,&write_intr);
                for(i=0 ; i<3000 && !(r=inb_p(HD_STATUS)&DRQ_STAT) ; i++)
                        /* nothing */ ;
                if (!r) {
                        bad_rw_intr();
                        goto repeat;
                }
                port_write(HD_DATA,CURRENT->buffer,256);
        } else if (CURRENT->cmd == READ) {
                hd_out(dev,nsect,sec,head,cyl,WIN_READ,&read_intr);
        } else
                panic("unknown hd-command");
}

void hd_init(void)
{
        blk_dev[MAJOR_NR].request_fn = DEVICE_REQUEST;
```

```
        set_intr_gate(0x2E,&hd_interrupt);
        outb_p(inb_p(0x21)&0xfb,0x21);
        outb(inb_p(0xA1)&0xbf,0xA1);
}
```

```
/*
 *  linux/kernel/blk_dev/ll_rw.c
 *
 * (C) 1991 Linus Torvalds
 */

/*
 * This handles all read/write requests to block devices
 */
#include <errno.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/system.h>

#include "blk.h"

/*
 * The request-struct contains all necessary data
 * to load a nr of sectors into memory
 */
struct request request[NR_REQUEST];

/*
 * used to wait on when there are no free requests
 */
struct task_struct * wait_for_request = NULL;

/* blk_dev_struct is:
 *      do_request-address
 *      next-request
 */
struct blk_dev_struct blk_dev[NR_BLK_DEV] = {
        { NULL, NULL },         /* no_dev */
        { NULL, NULL },         /* dev mem */
        { NULL, NULL },         /* dev fd */
        { NULL, NULL },         /* dev hd */
        { NULL, NULL },         /* dev ttyx */
        { NULL, NULL },         /* dev tty */
        { NULL, NULL }          /* dev lp */
};

static inline void lock_buffer(struct buffer_head * bh)
{
        cli();
        while (bh->b_lock)
                sleep_on(&bh->b_wait);
        bh->b_lock=1;
        sti();
}

static inline void unlock_buffer(struct buffer_head * bh)
{
        if (!bh->b_lock)
                printk("ll_rw_block.c: buffer not locked\n\r");
        bh->b_lock = 0;
        wake_up(&bh->b_wait);
}

/*
 * add-request adds a request to the linked list.
 * It disables interrupts so that it can muck with the
 * request-lists in peace.
 */
static void add_request(struct blk_dev_struct * dev, struct request * req)
{
        struct request * tmp;

        req->next = NULL;
        cli();
```

```
        if (req->bh)
                req->bh->b_dirt = 0;
        if (!(tmp = dev->current_request)) {
                dev->current_request = req;
                sti();
                (dev->request_fn)();
                return;
        }
        for ( ; tmp->next ; tmp=tmp->next)
                if ((IN_ORDER(tmp,req) ||
                    !IN_ORDER(tmp,tmp->next)) &&
                    IN_ORDER(req,tmp->next))
                        break;
        req->next=tmp->next;
        tmp->next=req;
        sti();
}

static void make_request(int major,int rw, struct buffer_head * bh)
{
        struct request * req;
        int rw_ahead;

/* WRITEA/READA is special case - it is not really needed, so if the */
/* buffer is locked, we just forget about it, else it's a normal read */
        if (rw_ahead = (rw == READA || rw == WRITEA)) {
                if (bh->b_lock)
                        return;
                if (rw == READA)
                        rw = READ;
                else
                        rw = WRITE;
        }
        if (rw!=READ && rw!=WRITE)
                panic("Bad block dev command, must be R/W/RA/WA");
        lock_buffer(bh);
        if ((rw == WRITE && !bh->b_dirt) || (rw == READ && bh->b_uptodate)) {
                unlock_buffer(bh);
                return;
        }
repeat:
/* we don't allow the write-requests to fill up the queue completely:
 * we want some room for reads: they take precedence. The last third
 * of the requests are only for reads.
 */
        if (rw == READ)
                req = request+NR_REQUEST;
        else
                req = request+((NR_REQUEST*2)/3);
/* find an empty request */
        while (--req >= request)
                if (req->dev<0)
                        break;
/* if none found, sleep on new requests: check for rw_ahead */
        if (req < request) {
                if (rw_ahead) {
                        unlock_buffer(bh);
                        return;
                }
                sleep_on(&wait_for_request);
                goto repeat;
        }
/* fill up the request-info, and add it to the queue */
        req->dev = bh->b_dev;
        req->cmd = rw;
        req->errors=0;
        req->sector = bh->b_blocknr<<1;
        req->nr_sectors = 2;
        req->buffer = bh->b_data;
```

```
        req->waiting = NULL;
        req->bh = bh;
        req->next = NULL;
        add_request(major+blk_dev,req);
}

void ll_rw_block(int rw, struct buffer_head * bh)
{
        unsigned int major;

        if ((major=MAJOR(bh->b_dev)) >= NR_BLK_DEV ||
        !(blk_dev[major].request_fn)) {
                printk("Trying to read nonexistent block-device\n\r");
                return;
        }
        make_request(major,rw,bh);
}

void blk_dev_init(void)
{
        int i;

        for (i=0 ; i<NR_REQUEST ; i++) {
                request[i].dev = -1;
                request[i].next = NULL;
        }
}
```

```
/*
 *  linux/kernel/blk_drv/ramdisk.c
 *
 *  Written by Theodore Ts'o, 12/2/91
 */

#include <string.h>

#include <linux/config.h>
#include <linux/sched.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <asm/system.h>
#include <asm/segment.h>
#include <asm/memory.h>

#define MAJOR_NR 1
#include "blk.h"

char    *rd_start;
int     rd_length = 0;

void do_rd_request(void)
{
        int     len;
        char    *addr;

        INIT_REQUEST;
        addr = rd_start + (CURRENT->sector << 9);
        len = CURRENT->nr_sectors << 9;
        if ((MINOR(CURRENT->dev) != 1) || (addr+len > rd_start+rd_length)) {
                end_request(0);
                goto repeat;
        }
        if (CURRENT-> cmd == WRITE) {
                (void ) memcpy(addr,
                               CURRENT->buffer,
                               len);
        } else if (CURRENT->cmd == READ) {
                (void) memcpy(CURRENT->buffer,
                               addr,
                               len);
        } else
                panic("unknown ramdisk-command");
        end_request(1);
        goto repeat;
}

/*
 * Returns amount of memory which needs to be reserved.
 */
long rd_init(long mem_start, int length)
{
        int     i;
        char    *cp;

        blk_dev[MAJOR_NR].request_fn = DEVICE_REQUEST;
        rd_start = (char *) mem_start;
        rd_length = length;
        cp = rd_start;
        for (i=0; i < length; i++)
                *cp++ = '\0';
        return(length);
}

/*
 * If the root device is the ram disk, try to load it.
 * In order to do this, the root device is originally set to the
 * floppy, and we later change it to be ram disk.
```

```
 */
void rd_load(void)
{
        struct buffer_head *bh;
        struct super_block      s;
        int             block = 256;    /* Start at block 256 */
        int             i = 1;
        int             nblocks;
        char            *cp;            /* Move pointer */

        if (!rd_length)
                return;
        printk("Ram disk: %d bytes, starting at 0x%x\n", rd_length,
                (int) rd_start);
        if (MAJOR(ROOT_DEV) != 2)
                return;
        bh = breada(ROOT_DEV,block+1,block,block+2,-1);
        if (!bh) {
                printk("Disk error while looking for ramdisk!\n");
                return;
        }
        *((struct d_super_block *) &s) = *((struct d_super_block *) bh->b_data);
        brelse(bh);
        if (s.s_magic != SUPER_MAGIC)
                /* No ram disk image present, assume normal floppy boot */
                return;
        nblocks = s.s_nzones << s.s_log_zone_size;
        if (nblocks > (rd_length >> BLOCK_SIZE_BITS)) {
                printk("Ram disk image too big!  (%d blocks, %d avail)\n",
                        nblocks, rd_length >> BLOCK_SIZE_BITS);
                return;
        }
        printk("Loading %d bytes into ram disk... 0000k",
                nblocks << BLOCK_SIZE_BITS);
        cp = rd_start;
        while (nblocks) {
                if (nblocks > 2)
                        bh = breada(ROOT_DEV, block, block+1, block+2, -1);
                else
                        bh = bread(ROOT_DEV, block);
                if (!bh) {
                        printk("I/O error on block %d, aborting load\n",
                                block);
                        return;
                }
                (void) memcpy(cp, bh->b_data, BLOCK_SIZE);
                brelse(bh);
                printk("\010\010\010\010\010%4dk",i);
                cp += BLOCK_SIZE;
                block++;
                nblocks--;
                i++;
        }
        printk("\010\010\010\010\010done \n");
        ROOT_DEV=0x0101;
}
```

```
#
# Makefile for the FREAX-kernel character device drivers.
#
# Note! Dependencies are done automagically by 'make dep', which also
# removes any old dependencies. DON'T put your own dependencies here
# unless it's something special (ie not a .c file).
#

AR      =gar
AS      =gas
LD      =gld
LDFLAGS =-s -x
CC      =gcc
CFLAGS  =-Wall -O -fstrength-reduce -fomit-frame-pointer -fcombine-regs \
        -finline-functions -mstring-insns -nostdinc -I../../include
CPP     =gcc -E -nostdinc -I../../include

.c.s:
        $(CC) $(CFLAGS) \
        -S -o $*.s $<
.s.o:
        $(AS) -c -o $*.o $<
.c.o:
        $(CC) $(CFLAGS) \
        -c -o $*.o $<

OBJS  = tty_io.o console.o keyboard.o serial.o rs_io.o \
        tty_ioctl.o

chr_drv.a: $(OBJS)
        $(AR) rcs chr_drv.a $(OBJS)
        sync

keyboard.s: keyboard.S ../../include/linux/config.h
        $(CPP) -traditional keyboard.S -o keyboard.s

clean:
        rm -f core *.o *.a tmp_make keyboard.s
        for i in *.c;do rm -f `basename $$i .c`.s;done

dep:
        sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
        (for i in *.c;do echo -n `echo $$i | sed 's,\.c,\.s,'`" "; \
                $(CPP) -M $$i;done) >> tmp_make
        cp tmp_make Makefile

### Dependencies:
console.s console.o : console.c ../../include/linux/sched.h \
  ../../include/linux/head.h ../../include/linux/fs.h \
  ../../include/sys/types.h ../../include/linux/mm.h ../../include/signal.h \
  ../../include/linux/tty.h ../../include/termios.h ../../include/asm/io.h \
  ../../include/asm/system.h
serial.s serial.o : serial.c ../../include/linux/tty.h ../../include/termios.h \
  ../../include/linux/sched.h ../../include/linux/head.h \
  ../../include/linux/fs.h ../../include/sys/types.h ../../include/linux/mm.h \
  ../../include/signal.h ../../include/asm/system.h ../../include/asm/io.h
tty_io.s tty_io.o : tty_io.c ../../include/ctype.h ../../include/errno.h \
  ../../include/signal.h ../../include/sys/types.h \
  ../../include/linux/sched.h ../../include/linux/head.h \
  ../../include/linux/fs.h ../../include/linux/mm.h ../../include/linux/tty.h \
  ../../include/termios.h ../../include/asm/segment.h \
  ../../include/asm/system.h
tty_ioctl.s tty_ioctl.o : tty_ioctl.c ../../include/errno.h ../../include/termios.h \
  ../../include/linux/sched.h ../../include/linux/head.h \
  ../../include/linux/fs.h ../../include/sys/types.h ../../include/linux/mm.h \
  ../../include/signal.h ../../include/linux/kernel.h \
  ../../include/linux/tty.h ../../include/asm/io.h \
  ../../include/asm/segment.h ../../include/asm/system.h
```

```
/*
 *  linux/kernel/console.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 *      console.c
 *
 * This module implements the console io functions
 *      'void con_init(void)'
 *      'void con_write(struct tty_queue * queue)'
 * Hopefully this will be a rather complete VT102 implementation.
 *
 * Beeping thanks to John T Kohl.
 */

/*
 *  NOTE!!! We sometimes disable and enable interrupts for a short while
 * (to put a word in video IO), but this will work even for keyboard
 * interrupts. We know interrupts aren't enabled when getting a keyboard
 * interrupt, as we use trap-gates. Hopefully all is well.
 */

/*
 * Code to check for different video-cards mostly by Galen Hunt,
 * <g-hunt@ee.utah.edu>
 */

#include <linux/sched.h>
#include <linux/tty.h>
#include <asm/io.h>
#include <asm/system.h>

/*
 * These are set up by the setup-routine at boot-time:
 */

#define ORIG_X                  (*(unsigned char *)0x90000)
#define ORIG_Y                  (*(unsigned char *)0x90001)
#define ORIG_VIDEO_PAGE         (*(unsigned short *)0x90004)
#define ORIG_VIDEO_MODE         ((*(unsigned short *)0x90006) & 0xff)
#define ORIG_VIDEO_COLS         (((*(unsigned short *)0x90006) & 0xff00) >> 8)
#define ORIG_VIDEO_LINES        (25)
#define ORIG_VIDEO_EGA_AX       (*(unsigned short *)0x90008)
#define ORIG_VIDEO_EGA_BX       (*(unsigned short *)0x9000a)
#define ORIG_VIDEO_EGA_CX       (*(unsigned short *)0x9000c)

#define VIDEO_TYPE_MDA          0x10    /* Monochrome Text Display      */
#define VIDEO_TYPE_CGA          0x11    /* CGA Display                  */
#define VIDEO_TYPE_EGAM         0x20    /* EGA/VGA in Monochrome Mode   */
#define VIDEO_TYPE_EGAC         0x21    /* EGA/VGA in Color Mode        */

#define NPAR 16

extern void keyboard_interrupt(void);

static unsigned char    video_type;             /* Type of display being used   */
static unsigned long    video_num_columns;      /* Number of text columns       */
static unsigned long    video_size_row;         /* Bytes per row                */
static unsigned long    video_num_lines;        /* Number of test lines         */
static unsigned char    video_page;             /* Initial video page           */
static unsigned long    video_mem_start;        /* Start of video RAM           */
static unsigned long    video_mem_end;          /* End of video RAM (sort of)   */
static unsigned short   video_port_reg;         /* Video register select port   */
static unsigned short   video_port_val;         /* Video register value port    */
static unsigned short   video_erase_char;       /* Char+Attrib to erase with    */

static unsigned long    origin;         /* Used for EGA/VGA fast scroll */
```

```
static unsigned long	scr_end;	/* Used for EGA/VGA fast scroll */
static unsigned long	pos;
static unsigned long	x,y;
static unsigned long	top,bottom;
static unsigned long	state=0;
static unsigned long	npar,par[NPAR];
static unsigned long	ques=0;
static unsigned char	attr=0x07;

static void sysbeep(void);

/*
 * this is what the terminal answers to a ESC-Z or csi0c
 * query (= vt100 response).
 */
#define RESPONSE "\033[?1;2c"

/* NOTE! gotoxy thinks x==video_num_columns is ok */
static inline void gotoxy(unsigned int new_x,unsigned int new_y)
{
	if (new_x > video_num_columns || new_y >= video_num_lines)
		return;
	x=new_x;
	y=new_y;
	pos=origin + y*video_size_row + (x<<1);
}

static inline void set_origin(void)
{
	cli();
	outb_p(12, video_port_reg);
	outb_p(0xff&((origin-video_mem_start)>>9), video_port_val);
	outb_p(13, video_port_reg);
	outb_p(0xff&((origin-video_mem_start)>>1), video_port_val);
	sti();
}

static void scrup(void)
{
	if (video_type == VIDEO_TYPE_EGAC || video_type == VIDEO_TYPE_EGAM)
	{
		if (!top && bottom == video_num_lines) {
			origin += video_size_row;
			pos += video_size_row;
			scr_end += video_size_row;
			if (scr_end > video_mem_end) {
				__asm__("cld\n\t"
					"rep\n\t"
					"movsl\n\t"
					"movl _video_num_columns,%1\n\t"
					"rep\n\t"
					"stosw"
					::"a" (video_erase_char),
					"c" ((video_num_lines-1)*video_num_columns>>1),
					"D" (video_mem_start),
					"S" (origin)
					:"cx","di","si");
				scr_end -= origin-video_mem_start;
				pos -= origin-video_mem_start;
				origin = video_mem_start;
			} else {
				__asm__("cld\n\t"
					"rep\n\t"
					"stosw"
					::"a" (video_erase_char),
					"c" (video_num_columns),
					"D" (scr_end-video_size_row)
					:"cx","di");
			}
```

```
                        set_origin();
                } else {
                        __asm__("cld\n\t"
                                "rep\n\t"
                                "movsl\n\t"
                                "movl _video_num_columns,%%ecx\n\t"
                                "rep\n\t"
                                "stosw"
                                ::"a" (video_erase_char),
                                "c" ((bottom-top-1)*video_num_columns>>1),
                                "D" (origin+video_size_row*top),
                                "S" (origin+video_size_row*(top+1))
                                :"cx","di","si");
                }
        }
        else            /* Not EGA/VGA */
        {
                __asm__("cld\n\t"
                        "rep\n\t"
                        "movsl\n\t"
                        "movl _video_num_columns,%%ecx\n\t"
                        "rep\n\t"
                        "stosw"
                        ::"a" (video_erase_char),
                        "c" ((bottom-top-1)*video_num_columns>>1),
                        "D" (origin+video_size_row*top),
                        "S" (origin+video_size_row*(top+1))
                        :"cx","di","si");
        }
}

static void scrdown(void)
{
        if (video_type == VIDEO_TYPE_EGAC || video_type == VIDEO_TYPE_EGAM)
        {
                __asm__("std\n\t"
                        "rep\n\t"
                        "movsl\n\t"
                        "addl $2,%%edi\n\t"     /* %edi has been decremented by 4 */
                        "movl _video_num_columns,%%ecx\n\t"
                        "rep\n\t"
                        "stosw"
                        ::"a" (video_erase_char),
                        "c" ((bottom-top-1)*video_num_columns>>1),
                        "D" (origin+video_size_row*bottom-4),
                        "S" (origin+video_size_row*(bottom-1)-4)
                        :"ax","cx","di","si");
        }
        else            /* Not EGA/VGA */
        {
                __asm__("std\n\t"
                        "rep\n\t"
                        "movsl\n\t"
                        "addl $2,%%edi\n\t"     /* %edi has been decremented by 4 */
                        "movl _video_num_columns,%%ecx\n\t"
                        "rep\n\t"
                        "stosw"
                        ::"a" (video_erase_char),
                        "c" ((bottom-top-1)*video_num_columns>>1),
                        "D" (origin+video_size_row*bottom-4),
                        "S" (origin+video_size_row*(bottom-1)-4)
                        :"ax","cx","di","si");
        }
}

static void lf(void)
{
        if (y+1<bottom) {
                y++;
```

```
                pos += video_size_row;
                return;
        }
        scrup();
}

static void ri(void)
{
        if (y>top) {
                y--;
                pos -= video_size_row;
                return;
        }
        scrdown();
}

static void cr(void)
{
        pos -= x<<1;
        x=0;
}

static void del(void)
{
        if (x) {
                pos -= 2;
                x--;
                *(unsigned short *)pos = video_erase_char;
        }
}

static void csi_J(int par)
{
        long count __asm__("cx");
        long start __asm__("di");

        switch (par) {
                case 0: /* erase from cursor to end of display */
                        count = (scr_end-pos)>>1;
                        start = pos;
                        break;
                case 1: /* erase from start to cursor */
                        count = (pos-origin)>>1;
                        start = origin;
                        break;
                case 2: /* erase whole display */
                        count = video_num_columns * video_num_lines;
                        start = origin;
                        break;
                default:
                        return;
        }
        __asm__("cld\n\t"
                "rep\n\t"
                "stosw\n\t"
                ::"c" (count),
                "D" (start),"a" (video_erase_char)
                :"cx","di");
}

static void csi_K(int par)
{
        long count __asm__("cx");
        long start __asm__("di");

        switch (par) {
                case 0: /* erase from cursor to end of line */
                        if (x>=video_num_columns)
                                return;
```

```
                        count = video_num_columns-x;
                        start = pos;
                        break;
                case 1: /* erase from start of line to cursor */
                        start = pos - (x<<1);
                        count = (x<video_num_columns)?x:video_num_columns;
                        break;
                case 2: /* erase whole line */
                        start = pos - (x<<1);
                        count = video_num_columns;
                        break;
                default:
                        return;
        }
        __asm__("cld\n\t"
                "rep\n\t"
                "stosw\n\t"
                ::"c" (count),
                "D" (start),"a" (video_erase_char)
                :"cx","di");
}

void csi_m(void)
{
        int i;

        for (i=0;i<=npar;i++)
                switch (par[i]) {
                        case 0:attr=0x07;break;
                        case 1:attr=0x0f;break;
                        case 4:attr=0x0f;break;
                        case 7:attr=0x70;break;
                        case 27:attr=0x07;break;
                }
}

static inline void set_cursor(void)
{
        cli();
        outb_p(14, video_port_reg);
        outb_p(0xff&((pos-video_mem_start)>>9), video_port_val);
        outb_p(15, video_port_reg);
        outb_p(0xff&((pos-video_mem_start)>>1), video_port_val);
        sti();
}

static void respond(struct tty_struct * tty)
{
        char * p = RESPONSE;

        cli();
        while (*p) {
                PUTCH(*p,tty->read_q);
                p++;
        }
        sti();
        copy_to_cooked(tty);
}

static void insert_char(void)
{
        int i=x;
        unsigned short tmp, old = video_erase_char;
        unsigned short * p = (unsigned short *) pos;

        while (i++<video_num_columns) {
                tmp=*p;
                *p=old;
                old=tmp;
```

```
                p++;
        }
}

static void insert_line(void)
{
        int oldtop,oldbottom;

        oldtop=top;
        oldbottom=bottom;
        top=y;
        bottom = video_num_lines;
        scrdown();
        top=oldtop;
        bottom=oldbottom;
}

static void delete_char(void)
{
        int i;
        unsigned short * p = (unsigned short *) pos;

        if (x>=video_num_columns)
                return;
        i = x;
        while (++i < video_num_columns) {
                *p = *(p+1);
                p++;
        }
        *p = video_erase_char;
}

static void delete_line(void)
{
        int oldtop,oldbottom;

        oldtop=top;
        oldbottom=bottom;
        top=y;
        bottom = video_num_lines;
        scrup();
        top=oldtop;
        bottom=oldbottom;
}

static void csi_at(unsigned int nr)
{
        if (nr > video_num_columns)
                nr = video_num_columns;
        else if (!nr)
                nr = 1;
        while (nr--)
                insert_char();
}

static void csi_L(unsigned int nr)
{
        if (nr > video_num_lines)
                nr = video_num_lines;
        else if (!nr)
                nr = 1;
        while (nr--)
                insert_line();
}

static void csi_P(unsigned int nr)
{
        if (nr > video_num_columns)
                nr = video_num_columns;
```

```
        else if (!nr)
                nr = 1;
        while (nr--)
                delete_char();
}

static void csi_M(unsigned int nr)
{
        if (nr > video_num_lines)
                nr = video_num_lines;
        else if (!nr)
                nr=1;
        while (nr--)
                delete_line();
}

static int saved_x=0;
static int saved_y=0;

static void save_cur(void)
{
        saved_x=x;
        saved_y=y;
}

static void restore_cur(void)
{
        gotoxy(saved_x, saved_y);
}

void con_write(struct tty_struct * tty)
{
        int nr;
        char c;

        nr = CHARS(tty->write_q);
        while (nr--) {
                GETCH(tty->write_q,c);
                switch(state) {
                        case 0:
                                if (c>31 && c<127) {
                                        if (x>=video_num_columns) {
                                                x -= video_num_columns;
                                                pos -= video_size_row;
                                                lf();
                                        }
                                        __asm__("movb _attr,%%ah\n\t"
                                                "movw %%ax,%1\n\t"
                                                ::"a" (c),"m" (*(short *)pos)
                                                :"ax");
                                        pos += 2;
                                        x++;
                                } else if (c==27)
                                        state=1;
                                else if (c==10 || c==11 || c==12)
                                        lf();
                                else if (c==13)
                                        cr();
                                else if (c==ERASE_CHAR(tty))
                                        del();
                                else if (c==8) {
                                        if (x) {
                                                x--;
                                                pos -= 2;
                                        }
                                } else if (c==9) {
                                        c=8-(x&7);
                                        x += c;
                                        pos += c<<1;
```

```
                        if (x>video_num_columns) {
                                x -= video_num_columns;
                                pos -= video_size_row;
                                lf();
                        }
                        c=9;
                } else if (c==7)
                        sysbeep();
                break;
        case 1:
                state=0;
                if (c=='[')
                        state=2;
                else if (c=='E')
                        gotoxy(0,y+1);
                else if (c=='M')
                        ri();
                else if (c=='D')
                        lf();
                else if (c=='Z')
                        respond(tty);
                else if (x=='7')
                        save_cur();
                else if (x=='8')
                        restore_cur();
                break;
        case 2:
                for(npar=0;npar<NPAR;npar++)
                        par[npar]=0;
                npar=0;
                state=3;
                if (ques=(c=='?'))
                        break;
        case 3:
                if (c==';' && npar<NPAR-1) {
                        npar++;
                        break;
                } else if (c>='0' && c<='9') {
                        par[npar]=10*par[npar]+c-'0';
                        break;
                } else state=4;
        case 4:
                state=0;
                switch(c) {
                        case 'G': case '`':
                                if (par[0]) par[0]--;
                                gotoxy(par[0],y);
                                break;
                        case 'A':
                                if (!par[0]) par[0]++;
                                gotoxy(x,y-par[0]);
                                break;
                        case 'B': case 'e':
                                if (!par[0]) par[0]++;
                                gotoxy(x,y+par[0]);
                                break;
                        case 'C': case 'a':
                                if (!par[0]) par[0]++;
                                gotoxy(x+par[0],y);
                                break;
                        case 'D':
                                if (!par[0]) par[0]++;
                                gotoxy(x-par[0],y);
                                break;
                        case 'E':
                                if (!par[0]) par[0]++;
                                gotoxy(0,y+par[0]);
                                break;
                        case 'F':
```

```
                                        if (!par[0]) par[0]++;
                                        gotoxy(0,y-par[0]);
                                        break;
                                case 'd':
                                        if (par[0]) par[0]--;
                                        gotoxy(x,par[0]);
                                        break;
                                case 'H': case 'f':
                                        if (par[0]) par[0]--;
                                        if (par[1]) par[1]--;
                                        gotoxy(par[1],par[0]);
                                        break;
                                case 'J':
                                        csi_J(par[0]);
                                        break;
                                case 'K':
                                        csi_K(par[0]);
                                        break;
                                case 'L':
                                        csi_L(par[0]);
                                        break;
                                case 'M':
                                        csi_M(par[0]);
                                        break;
                                case 'P':
                                        csi_P(par[0]);
                                        break;
                                case '@':
                                        csi_at(par[0]);
                                        break;
                                case 'm':
                                        csi_m();
                                        break;
                                case 'r':
                                        if (par[0]) par[0]--;
                                        if (!par[1]) par[1] = video_num_lines;
                                        if (par[0] < par[1] &&
                                            par[1] <= video_num_lines) {
                                                top=par[0];
                                                bottom=par[1];
                                        }
                                        break;
                                case 's':
                                        save_cur();
                                        break;
                                case 'u':
                                        restore_cur();
                                        break;
                                }
                        }
                }
        }
        set_cursor();
}

/*
 *  void con_init(void);
 *
 * This routine initalizes console interrupts, and does nothing
 * else. If you want the screen to clear, call tty_write with
 * the appropriate escape-sequece.
 *
 * Reads the information preserved by setup.s to determine the current display
 * type and sets everything accordingly.
 */
void con_init(void)
{
        register unsigned char a;
        char *display_desc = "????";
        char *display_ptr;
```

```
        video_num_columns = ORIG_VIDEO_COLS;
        video_size_row = video_num_columns * 2;
        video_num_lines = ORIG_VIDEO_LINES;
        video_page = ORIG_VIDEO_PAGE;
        video_erase_char = 0x0720;

        if (ORIG_VIDEO_MODE == 7)                       /* Is this a monochrome display?
*/
        {
                video_mem_start = 0xb0000;
                video_port_reg = 0x3b4;
                video_port_val = 0x3b5;
                if ((ORIG_VIDEO_EGA_BX & 0xff) != 0x10)
                {
                        video_type = VIDEO_TYPE_EGAM;
                        video_mem_end = 0xb8000;
                        display_desc = "EGAm";
                }
                else
                {
                        video_type = VIDEO_TYPE_MDA;
                        video_mem_end   = 0xb2000;
                        display_desc = "*MDA";
                }
        }
        else                                                    /* If not, it is
color. */
        {
                video_mem_start = 0xb8000;
                video_port_reg  = 0x3d4;
                video_port_val  = 0x3d5;
                if ((ORIG_VIDEO_EGA_BX & 0xff) != 0x10)
                {
                        video_type = VIDEO_TYPE_EGAC;
                        video_mem_end = 0xbc000;
                        display_desc = "EGAc";
                }
                else
                {
                        video_type = VIDEO_TYPE_CGA;
                        video_mem_end = 0xba000;
                        display_desc = "*CGA";
                }
        }

        /* Let the user known what kind of display driver we are using */

        display_ptr = ((char *)video_mem_start) + video_size_row - 8;
        while (*display_desc)
        {
                *display_ptr++ = *display_desc++;
                display_ptr++;
        }

        /* Initialize the variables used for scrolling (mostly EGA/VGA) */

        origin  = video_mem_start;
        scr_end = video_mem_start + video_num_lines * video_size_row;
        top     = 0;
        bottom  = video_num_lines;

        gotoxy(ORIG_X,ORIG_Y);
        set_trap_gate(0x21,&keyboard_interrupt);
        outb_p(inb_p(0x21)&0xfd,0x21);
        a=inb_p(0x61);
        outb_p(a|0x80,0x61);
        outb(a,0x61);
}
```

```
/* from bsd-net-2: */

void sysbeepstop(void)
{
        /* disable counter 2 */
        outb(inb_p(0x61)&0xFC, 0x61);
}

int beepcount = 0;

static void sysbeep(void)
{
        /* enable counter 2 */
        outb_p(inb_p(0x61)|3, 0x61);
        /* set command for counter 2, 2 byte write */
        outb_p(0xB6, 0x43);
        /* send 0x637 for 750 HZ */
        outb_p(0x37, 0x42);
        outb(0x06, 0x42);
        /* 1/8 second */
        beepcount = HZ/8;
}
```

```
/*
 *  linux/kernel/keyboard.S
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 *      Thanks to Alfred Leung for US keyboard patches
 *              Wolfgang Thiel for German keyboard patches
 *              Marc Corsini for the French keyboard
 */

#include <linux/config.h>

.text
.globl _keyboard_interrupt

/*
 * these are for the keyboard read functions
 */
size    = 1024          /* must be a power of two ! And MUST be the same
                           as in tty_io.c !!!! */
head = 4
tail = 8
proc_list = 12
buf = 16

mode:   .byte 0         /* caps, alt, ctrl and shift mode */
leds:   .byte 2         /* num-lock, caps, scroll-lock mode (nom-lock on) */
e0:     .byte 0

/*
 *  con_int is the real interrupt routine that reads the
 *  keyboard scan-code and converts it into the appropriate
 *  ascii character(s).
 */
_keyboard_interrupt:
        pushl %eax
        pushl %ebx
        pushl %ecx
        pushl %edx
        push %ds
        push %es
        movl $0x10,%eax
        mov %ax,%ds
        mov %ax,%es
        xorl %al,%al            /* %eax is scan code */
        inb $0x60,%al
        cmpb $0xe0,%al
        je set_e0
        cmpb $0xe1,%al
        je set_e1
        call key_table(,%eax,4)
        movb $0,e0
e0_e1:  inb $0x61,%al
        jmp 1f
1:      jmp 1f
1:      orb $0x80,%al
        jmp 1f
1:      jmp 1f
1:      outb %al,$0x61
        jmp 1f
1:      jmp 1f
1:      andb $0x7F,%al
        outb %al,$0x61
        movb $0x20,%al
        outb %al,$0x20
        pushl $0
        call _do_tty_interrupt
```

```
        addl $4,%esp
        pop %es
        pop %ds
        popl %edx
        popl %ecx
        popl %ebx
        popl %eax
        iret
set_e0: movb $1,e0
        jmp e0_e1
set_e1: movb $2,e0
        jmp e0_e1

/*
 * This routine fills the buffer with max 8 bytes, taken from
 * %ebx:%eax. (%edx is high). The bytes are written in the
 * order %al,%ah,%eal,%eah,%bl,%bh ... until %eax is zero.
 */
put_queue:
        pushl %ecx
        pushl %edx
        movl _table_list,%edx           # read-queue for console
        movl head(%edx),%ecx
1:      movb %al,buf(%edx,%ecx)
        incl %ecx
        andl $size-1,%ecx
        cmpl tail(%edx),%ecx            # buffer full - discard everything
        je 3f
        shrdl $8,%ebx,%eax
        je 2f
        shrl $8,%ebx
        jmp 1b
2:      movl %ecx,head(%edx)
        movl proc_list(%edx),%ecx
        testl %ecx,%ecx
        je 3f
        movl $0,(%ecx)
3:      popl %edx
        popl %ecx
        ret

ctrl:   movb $0x04,%al
        jmp 1f
alt:    movb $0x10,%al
1:      cmpb $0,e0
        je 2f
        addb %al,%al
2:      orb %al,mode
        ret
unctrl: movb $0x04,%al
        jmp 1f
unalt:  movb $0x10,%al
1:      cmpb $0,e0
        je 2f
        addb %al,%al
2:      notb %al
        andb %al,mode
        ret

lshift:
        orb $0x01,mode
        ret
unlshift:
        andb $0xfe,mode
        ret
rshift:
        orb $0x02,mode
        ret
unrshift:
```

```
        andb $0xfd,mode
        ret

caps:   testb $0x80,mode
        jne 1f
        xorb $4,leds
        xorb $0x40,mode
        orb $0x80,mode
set_leds:
        call kb_wait
        movb $0xed,%al          /* set leds command */
        outb %al,$0x60
        call kb_wait
        movb leds,%al
        outb %al,$0x60
        ret
uncaps: andb $0x7f,mode
        ret
scroll:
        xorb $1,leds
        jmp set_leds
num:    xorb $2,leds
        jmp set_leds

/*
 *  curosr-key/numeric keypad cursor keys are handled here.
 *  checking for numeric keypad etc.
 */
cursor:
        subb $0x47,%al
        jb 1f
        cmpb $12,%al
        ja 1f
        jne cur2                /* check for ctrl-alt-del */
        testb $0x0c,mode
        je cur2
        testb $0x30,mode
        jne reboot
cur2:   cmpb $0x01,e0           /* e0 forces cursor movement */
        je cur
        testb $0x02,leds        /* not num-lock forces cursor */
        je cur
        testb $0x03,mode        /* shift forces cursor */
        jne cur
        xorl %ebx,%ebx
        movb num_table(%eax),%al
        jmp put_queue
1:      ret

cur:    movb cur_table(%eax),%al
        cmpb $'9,%al
        ja ok_cur
        movb $'~,%ah
ok_cur: shll $16,%eax
        movw $0x5b1b,%ax
        xorl %ebx,%ebx
        jmp put_queue

#if defined(KBD_FR)
num_table:
        .ascii "789 456 1230."
#else
num_table:
        .ascii "789 456 1230,"
#endif
cur_table:
        .ascii "HA5 DGC YB623"

/*
```

```
 * this routine handles function keys
 */
func:
        pushl %eax
        pushl %ecx
        pushl %edx
        call _show_stat
        popl %edx
        popl %ecx
        popl %eax
        subb $0x3B,%al
        jb end_func
        cmpb $9,%al
        jbe ok_func
        subb $18,%al
        cmpb $10,%al
        jb end_func
        cmpb $11,%al
        ja end_func
ok_func:
        cmpl $4,%ecx            /* check that there is enough room */
        jl end_func
        movl func_table(,%eax,4),%eax
        xorl %ebx,%ebx
        jmp put_queue
end_func:
        ret

/*
 * function keys send F1:'esc [ [ A' F2:'esc [ [ B' etc.
 */
func_table:
        .long 0x415b5b1b,0x425b5b1b,0x435b5b1b,0x445b5b1b
        .long 0x455b5b1b,0x465b5b1b,0x475b5b1b,0x485b5b1b
        .long 0x495b5b1b,0x4a5b5b1b,0x4b5b5b1b,0x4c5b5b1b

#if     defined(KBD_FINNISH)
key_map:
        .byte 0,27
        .ascii "1234567890+'"
        .byte 127,9
        .ascii "qwertyuiop}"
        .byte 0,13,0
        .ascii "asdfghjkl|{"
        .byte 0,0
        .ascii "'zxcvbnm,.-"
        .byte 0,'*,0,32         /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte '-,0,0,0,'+       /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '<
        .fill 10,1,0

shift_map:
        .byte 0,27
        .ascii "!\"#$%&/()=?`"
        .byte 127,9
        .ascii "QWERTYUIOP]^"
        .byte 13,0
        .ascii "ASDFGHJKL\\["
        .byte 0,0
        .ascii "*ZXCVBNM;:_"
        .byte 0,'*,0,32         /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte '-,0,0,0,'+       /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '>
        .fill 10,1,0
```

```
alt_map:
        .byte 0,0
        .ascii "\0@\0$\0\0{[]}\\\0"
        .byte 0,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte '~,13,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte 0,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte 0,0,0,0           /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte 0,0,0,0,0         /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '|
        .fill 10,1,0

#elif defined(KBD_US)

key_map:
        .byte 0,27
        .ascii "1234567890-="
        .byte 127,9
        .ascii "qwertyuiop[]"
        .byte 13,0
        .ascii "asdfghjkl;'"
        .byte '`,0
        .ascii "\\zxcvbnm,./"
        .byte 0,'*,0,32         /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte '-,0,0,0,'+       /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '<
        .fill 10,1,0


shift_map:
        .byte 0,27
        .ascii "!@#$%^&*()_+"
        .byte 127,9
        .ascii "QWERTYUIOP{}"
        .byte 13,0
        .ascii "ASDFGHJKL:\""
        .byte '~,0
        .ascii "|ZXCVBNM<>?"
        .byte 0,'*,0,32         /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte '-,0,0,0,'+       /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '>
        .fill 10,1,0

alt_map:
        .byte 0,0
        .ascii "\0@\0$\0\0{[]}\\\0"
        .byte 0,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte '~,13,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte 0,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte 0,0,0,0           /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte 0,0,0,0,0         /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '|
        .fill 10,1,0

#elif defined(KBD_GR)
```

```
key_map:
        .byte 0,27
        .ascii "1234567890\\'"
        .byte 127,9
        .ascii "qwertzuiop@+"
        .byte 13,0
        .ascii "asdfghjkl[]^"
        .byte 0,'#
        .ascii "yxcvbnm,.-"
        .byte 0,'*,0,32         /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte '-,0,0,0,'+       /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '<
        .fill 10,1,0


shift_map:
        .byte 0,27
        .ascii "!\"#$%&/()=?`"
        .byte 127,9
        .ascii "QWERTZUIOP\\*"
        .byte 13,0
        .ascii "ASDFGHJKL{}~"
        .byte 0,''
        .ascii "YXCVBNM;:_"
        .byte 0,'*,0,32         /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte '-,0,0,0,'+       /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '>
        .fill 10,1,0

alt_map:
        .byte 0,0
        .ascii "\0@\0$\0\0{[]}\\\0"
        .byte 0,0
        .byte '@,0,0,0,0,0,0,0,0,0,0
        .byte '~,13,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte 0,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte 0,0,0,0           /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte 0,0,0,0,0         /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '|
        .fill 10,1,0


#elif defined(KBD_FR)

key_map:
        .byte 0,27
        .ascii "&{\"'(-}_/@)="
        .byte 127,9
        .ascii "azertyuiop^$"
        .byte 13,0
        .ascii "qsdfghjklm|"
        .byte '`,0,42           /* coin sup gauche, don't know, [*|mu] */
        .ascii "wxcvbn,;:!"
        .byte 0,'*,0,32         /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte '-,0,0,0,'+       /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '<
        .fill 10,1,0

shift_map:
```

```
        .byte 0,27
        .ascii "1234567890]+"
        .byte 127,9
        .ascii "AZERTYUIOP<>"
        .byte 13,0
        .ascii "QSDFGHJKLM%"
        .byte '~,0,'#
        .ascii "WXCVBN?./\\"
        .byte 0,'*,0,32         /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte '-,0,0,0,'+       /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '>
        .fill 10,1,0

alt_map:
        .byte 0,0
        .ascii "\0~#{[|`\\^@]}"
        .byte 0,0
        .byte '@,0,0,0,0,0,0,0,0,0,0
        .byte '~,13,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte 0,0
        .byte 0,0,0,0,0,0,0,0,0,0,0
        .byte 0,0,0,0           /* 36-39 */
        .fill 16,1,0            /* 3A-49 */
        .byte 0,0,0,0,0         /* 4A-4E */
        .byte 0,0,0,0,0,0,0     /* 4F-55 */
        .byte '|
        .fill 10,1,0

#else
#error "KBD-type not defined"
#endif
/*
 * do_self handles "normal" keys, ie keys that don't change meaning
 * and which have just one character returns.
 */
do_self:
        lea alt_map,%ebx
        testb $0x20,mode                /* alt-gr */
        jne 1f
        lea shift_map,%ebx
        testb $0x03,mode
        jne 1f
        lea key_map,%ebx
1:      movb (%ebx,%eax),%al
        orb %al,%al
        je none
        testb $0x4c,mode                /* ctrl or caps */
        je 2f
        cmpb $'a,%al
        jb 2f
        cmpb $'},%al
        ja 2f
        subb $32,%al
2:      testb $0x0c,mode                /* ctrl */
        je 3f
        cmpb $64,%al
        jb 3f
        cmpb $64+32,%al
        jae 3f
        subb $64,%al
3:      testb $0x10,mode                /* left alt */
        je 4f
        orb $0x80,%al
4:      andl $0xff,%eax
        xorl %ebx,%ebx
        call put_queue
```

```
none:   ret

/*
 * minus has a routine of it's own, as a 'E0h' before
 * the scan code for minus means that the numeric keypad
 * slash was pushed.
 */
minus:  cmpb $1,e0
        jne do_self
        movl $'/,%eax
        xorl %ebx,%ebx
        jmp put_queue

/*
 * This table decides which routine to call when a scan-code has been
 * gotten. Most routines just call do_self, or none, depending if
 * they are make or break.
 */
key_table:
        .long none,do_self,do_self,do_self      /* 00-03 s0 esc 1 2 */
        .long do_self,do_self,do_self,do_self   /* 04-07 3 4 5 6 */
        .long do_self,do_self,do_self,do_self   /* 08-0B 7 8 9 0 */
        .long do_self,do_self,do_self,do_self   /* 0C-0F + ' bs tab */
        .long do_self,do_self,do_self,do_self   /* 10-13 q w e r */
        .long do_self,do_self,do_self,do_self   /* 14-17 t y u i */
        .long do_self,do_self,do_self,do_self   /* 18-1B o p } ^ */
        .long do_self,ctrl,do_self,do_self      /* 1C-1F enter ctrl a s */
        .long do_self,do_self,do_self,do_self   /* 20-23 d f g h */
        .long do_self,do_self,do_self,do_self   /* 24-27 j k l | */
        .long do_self,do_self,lshift,do_self    /* 28-2B { para lshift , */
        .long do_self,do_self,do_self,do_self   /* 2C-2F z x c v */
        .long do_self,do_self,do_self,do_self   /* 30-33 b n m , */
        .long do_self,minus,rshift,do_self      /* 34-37 . - rshift * */
        .long alt,do_self,caps,func             /* 38-3B alt sp caps f1 */
        .long func,func,func,func               /* 3C-3F f2 f3 f4 f5 */
        .long func,func,func,func               /* 40-43 f6 f7 f8 f9 */
        .long func,num,scroll,cursor            /* 44-47 f10 num scr home */
        .long cursor,cursor,do_self,cursor      /* 48-4B up pgup - left */
        .long cursor,cursor,do_self,cursor      /* 4C-4F n5 right + end */
        .long cursor,cursor,cursor,cursor       /* 50-53 dn pgdn ins del */
        .long none,none,do_self,func            /* 54-57 sysreq ? < f11 */
        .long func,none,none,none               /* 58-5B f12 ? ? ? */
        .long none,none,none,none               /* 5C-5F ? ? ? ? */
        .long none,none,none,none               /* 60-63 ? ? ? ? */
        .long none,none,none,none               /* 64-67 ? ? ? ? */
        .long none,none,none,none               /* 68-6B ? ? ? ? */
        .long none,none,none,none               /* 6C-6F ? ? ? ? */
        .long none,none,none,none               /* 70-73 ? ? ? ? */
        .long none,none,none,none               /* 74-77 ? ? ? ? */
        .long none,none,none,none               /* 78-7B ? ? ? ? */
        .long none,none,none,none               /* 7C-7F ? ? ? ? */
        .long none,none,none,none               /* 80-83 ? br br br */
        .long none,none,none,none               /* 84-87 br br br br */
        .long none,none,none,none               /* 88-8B br br br br */
        .long none,none,none,none               /* 8C-8F br br br br */
        .long none,none,none,none               /* 90-93 br br br br */
        .long none,none,none,none               /* 94-97 br br br br */
        .long none,none,none,none               /* 98-9B br br br br */
        .long none,unctrl,none,none             /* 9C-9F br unctrl br br */
        .long none,none,none,none               /* A0-A3 br br br br */
        .long none,none,none,none               /* A4-A7 br br br br */
        .long none,none,unlshift,none           /* A8-AB br br unlshift br */
        .long none,none,none,none               /* AC-AF br br br br */
        .long none,none,none,none               /* B0-B3 br br br br */
        .long none,none,unrshift,none           /* B4-B7 br br unrshift br */
        .long unalt,none,uncaps,none            /* B8-BB unalt br uncaps br */
        .long none,none,none,none               /* BC-BF br br br br */
        .long none,none,none,none               /* C0-C3 br br br br */
        .long none,none,none,none               /* C4-C7 br br br br */
```

```
        .long none,none,none,none               /* C8-CB br br br br */
        .long none,none,none,none               /* CC-CF br br br br */
        .long none,none,none,none               /* D0-D3 br br br br */
        .long none,none,none,none               /* D4-D7 br br br br */
        .long none,none,none,none               /* D8-DB br ? ? ? */
        .long none,none,none,none               /* DC-DF ? ? ? ? */
        .long none,none,none,none               /* E0-E3 e0 e1 ? ? */
        .long none,none,none,none               /* E4-E7 ? ? ? ? */
        .long none,none,none,none               /* E8-EB ? ? ? ? */
        .long none,none,none,none               /* EC-EF ? ? ? ? */
        .long none,none,none,none               /* F0-F3 ? ? ? ? */
        .long none,none,none,none               /* F4-F7 ? ? ? ? */
        .long none,none,none,none               /* F8-FB ? ? ? ? */
        .long none,none,none,none               /* FC-FF ? ? ? ? */

/*
 * kb_wait waits for the keyboard controller buffer to empty.
 * there is no timeout - if the buffer doesn't empty, we hang.
 */
kb_wait:
        pushl %eax
1:      inb $0x64,%al
        testb $0x02,%al
        jne 1b
        popl %eax
        ret
/*
 * This routine reboots the machine by asking the keyboard
 * controller to pulse the reset-line low.
 */
reboot:
        call kb_wait
        movw $0x1234,0x472      /* don't do memory check */
        movb $0xfc,%al          /* pulse reset and A20 low */
        outb %al,$0x64
die:    jmp die
```

```
/*
 *  linux/kernel/rs_io.s
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 *      rs_io.s
 *
 * This module implements the rs232 io interrupts.
 */

.text
.globl _rs1_interrupt,_rs2_interrupt

size    = 1024                          /* must be power of two !
                                           and must match the value
                                           in tty_io.c!!! */

/* these are the offsets into the read/write buffer structures */
rs_addr = 0
head = 4
tail = 8
proc_list = 12
buf = 16

startup = 256           /* chars left in write queue when we restart it */

/*
 * These are the actual interrupt routines. They look where
 * the interrupt is coming from, and take appropriate action.
 */
.align 2
_rs1_interrupt:
        pushl $_table_list+8
        jmp rs_int
.align 2
_rs2_interrupt:
        pushl $_table_list+16
rs_int:
        pushl %edx
        pushl %ecx
        pushl %ebx
        pushl %eax
        push %es
        push %ds                /* as this is an interrupt, we cannot */
        pushl $0x10             /* know that bs is ok. Load it */
        pop %ds
        pushl $0x10
        pop %es
        movl 24(%esp),%edx
        movl (%edx),%edx
        movl rs_addr(%edx),%edx
        addl $2,%edx            /* interrupt ident. reg */
rep_int:
        xorl %eax,%eax
        inb %dx,%al
        testb $1,%al
        jne end
        cmpb $6,%al             /* this shouldn't happen, but ... */
        ja end
        movl 24(%esp),%ecx
        pushl %edx
        subl $2,%edx
        call jmp_table(,%eax,2)         /* NOTE! not *4, bit0 is 0 already */
        popl %edx
        jmp rep_int
end:    movb $0x20,%al
        outb %al,$0x20          /* EOI */
```

```
        pop %ds
        pop %es
        popl %eax
        popl %ebx
        popl %ecx
        popl %edx
        addl $4,%esp            # jump over _table_list entry
        iret

jmp_table:
        .long modem_status,write_char,read_char,line_status

.align 2
modem_status:
        addl $6,%edx            /* clear intr by reading modem status reg */
        inb %dx,%al
        ret

.align 2
line_status:
        addl $5,%edx            /* clear intr by reading line status reg. */
        inb %dx,%al
        ret

.align 2
read_char:
        inb %dx,%al
        movl %ecx,%edx
        subl $_table_list,%edx
        shrl $3,%edx
        movl (%ecx),%ecx                # read-queue
        movl head(%ecx),%ebx
        movb %al,buf(%ecx,%ebx)
        incl %ebx
        andl $size-1,%ebx
        cmpl tail(%ecx),%ebx
        je 1f
        movl %ebx,head(%ecx)
1:      pushl %edx
        call _do_tty_interrupt
        addl $4,%esp
        ret

.align 2
write_char:
        movl 4(%ecx),%ecx               # write-queue
        movl head(%ecx),%ebx
        subl tail(%ecx),%ebx
        andl $size-1,%ebx               # nr chars in queue
        je write_buffer_empty
        cmpl $startup,%ebx
        ja 1f
        movl proc_list(%ecx),%ebx       # wake up sleeping process
        testl %ebx,%ebx                 # is there any?
        je 1f
        movl $0,(%ebx)
1:      movl tail(%ecx),%ebx
        movb buf(%ecx,%ebx),%al
        outb %al,%dx
        incl %ebx
        andl $size-1,%ebx
        movl %ebx,tail(%ecx)
        cmpl head(%ecx),%ebx
        je write_buffer_empty
        ret
.align 2
write_buffer_empty:
        movl proc_list(%ecx),%ebx       # wake up sleeping process
        testl %ebx,%ebx                 # is there any?
```

```
        je 1f
        movl $0,(%ebx)
1:      incl %edx
        inb %dx,%al
        jmp 1f
1:      jmp 1f
1:      andb $0xd,%al           /* disable transmit interrupt */
        outb %al,%dx
        ret
```

```
/*
 *  linux/kernel/serial.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 *      serial.c
 *
 * This module implements the rs232 io functions
 *      void rs_write(struct tty_struct * queue);
 *      void rs_init(void);
 * and all interrupts pertaining to serial IO.
 */

#include <linux/tty.h>
#include <linux/sched.h>
#include <asm/system.h>
#include <asm/io.h>

#define WAKEUP_CHARS (TTY_BUF_SIZE/4)

extern void rs1_interrupt(void);
extern void rs2_interrupt(void);

static void init(int port)
{
        outb_p(0x80,port+3);    /* set DLAB of line control reg */
        outb_p(0x30,port);      /* LS of divisor (48 -> 2400 bps */
        outb_p(0x00,port+1);    /* MS of divisor */
        outb_p(0x03,port+3);    /* reset DLAB */
        outb_p(0x0b,port+4);    /* set DTR,RTS, OUT_2 */
        outb_p(0x0d,port+1);    /* enable all intrs but writes */
        (void)inb(port);        /* read data port to reset things (?) */
}

void rs_init(void)
{
        set_intr_gate(0x24,rs1_interrupt);
        set_intr_gate(0x23,rs2_interrupt);
        init(tty_table[1].read_q.data);
        init(tty_table[2].read_q.data);
        outb(inb_p(0x21)&0xE7,0x21);
}

/*
 * This routine gets called when tty_write has put something into
 * the write_queue. It must check wheter the queue is empty, and
 * set the interrupt register accordingly
 *
 *      void _rs_write(struct tty_struct * tty);
 */
void rs_write(struct tty_struct * tty)
{
        cli();
        if (!EMPTY(tty->write_q))
                outb(inb_p(tty->write_q.data+1)|0x02,tty->write_q.data+1);
        sti();
}
```

```
/*
 *  linux/kernel/tty_io.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * 'tty_io.c' gives an orthogonal feeling to tty's, be they consoles
 * or rs-channels. It also implements echoing, cooked mode etc.
 *
 * Kill-line thanks to John T Kohl.
 */
#include <ctype.h>
#include <errno.h>
#include <signal.h>

#define ALRMMASK (1<<(SIGALRM-1))
#define KILLMASK (1<<(SIGKILL-1))
#define INTMASK (1<<(SIGINT-1))
#define QUITMASK (1<<(SIGQUIT-1))
#define TSTPMASK (1<<(SIGTSTP-1))

#include <linux/sched.h>
#include <linux/tty.h>
#include <asm/segment.h>
#include <asm/system.h>

#define _L_FLAG(tty,f)  ((tty)->termios.c_lflag & f)
#define _I_FLAG(tty,f)  ((tty)->termios.c_iflag & f)
#define _O_FLAG(tty,f)  ((tty)->termios.c_oflag & f)

#define L_CANON(tty)    _L_FLAG((tty),ICANON)
#define L_ISIG(tty)     _L_FLAG((tty),ISIG)
#define L_ECHO(tty)     _L_FLAG((tty),ECHO)
#define L_ECHOE(tty)    _L_FLAG((tty),ECHOE)
#define L_ECHOK(tty)    _L_FLAG((tty),ECHOK)
#define L_ECHOCTL(tty)  _L_FLAG((tty),ECHOCTL)
#define L_ECHOKE(tty)   _L_FLAG((tty),ECHOKE)

#define I_UCLC(tty)     _I_FLAG((tty),IUCLC)
#define I_NLCR(tty)     _I_FLAG((tty),INLCR)
#define I_CRNL(tty)     _I_FLAG((tty),ICRNL)
#define I_NOCR(tty)     _I_FLAG((tty),IGNCR)

#define O_POST(tty)     _O_FLAG((tty),OPOST)
#define O_NLCR(tty)     _O_FLAG((tty),ONLCR)
#define O_CRNL(tty)     _O_FLAG((tty),OCRNL)
#define O_NLRET(tty)    _O_FLAG((tty),ONLRET)
#define O_LCUC(tty)     _O_FLAG((tty),OLCUC)

struct tty_struct tty_table[] = {
        {
                {ICRNL,         /* change incoming CR to NL */
                OPOST|ONLCR,    /* change outgoing NL to CRNL */
                0,
                ISIG | ICANON | ECHO | ECHOCTL | ECHOKE,
                0,              /* console termio */
                INIT_C_CC},
                0,                      /* initial pgrp */
                0,                      /* initial stopped */
                con_write,
                {0,0,0,0,""},           /* console read-queue */
                {0,0,0,0,""},           /* console write-queue */
                {0,0,0,0,""}            /* console secondary queue */
        },{
                {0, /* no translation */
                0,  /* no translation */
                B2400 | CS8,
                0,
```

```
                0,
                INIT_C_CC},
                0,
                0,
                rs_write,
                {0x3f8,0,0,0,""},               /* rs 1 */
                {0x3f8,0,0,0,""},
                {0,0,0,0,""}
        },{
                {0, /* no translation */
                0,  /* no translation */
                B2400 | CS8,
                0,
                0,
                INIT_C_CC},
                0,
                0,
                rs_write,
                {0x2f8,0,0,0,""},               /* rs 2 */
                {0x2f8,0,0,0,""},
                {0,0,0,0,""}
        }
};

/*
 * these are the tables used by the machine code handlers.
 * you can implement pseudo-tty's or something by changing
 * them. Currently not done.
 */
struct tty_queue * table_list[]={
        &tty_table[0].read_q, &tty_table[0].write_q,
        &tty_table[1].read_q, &tty_table[1].write_q,
        &tty_table[2].read_q, &tty_table[2].write_q
        };

void tty_init(void)
{
        rs_init();
        con_init();
}

void tty_intr(struct tty_struct * tty, int mask)
{
        int i;

        if (tty->pgrp <= 0)
                return;
        for (i=0;i<NR_TASKS;i++)
                if (task[i] && task[i]->pgrp==tty->pgrp)
                        task[i]->signal |= mask;
}

static void sleep_if_empty(struct tty_queue * queue)
{
        cli();
        while (!current->signal && EMPTY(*queue))
                interruptible_sleep_on(&queue->proc_list);
        sti();
}

static void sleep_if_full(struct tty_queue * queue)
{
        if (!FULL(*queue))
                return;
        cli();
        while (!current->signal && LEFT(*queue)<128)
                interruptible_sleep_on(&queue->proc_list);
        sti();
}
```

```
void wait_for_keypress(void)
{
        sleep_if_empty(&tty_table[0].secondary);
}

void copy_to_cooked(struct tty_struct * tty)
{
        signed char c;

        while (!EMPTY(tty->read_q) && !FULL(tty->secondary)) {
                GETCH(tty->read_q,c);
                if (c==13)
                        if (I_CRNL(tty))
                                c=10;
                        else if (I_NOCR(tty))
                                continue;
                        else ;
                else if (c==10 && I_NLCR(tty))
                        c=13;
                if (I_UCLC(tty))
                        c=tolower(c);
                if (L_CANON(tty)) {
                        if (c==KILL_CHAR(tty)) {
                                /* deal with killing the input line */
                                while(!(EMPTY(tty->secondary) ||
                                        (c=LAST(tty->secondary))==10 ||
                                        c==EOF_CHAR(tty))) {
                                        if (L_ECHO(tty)) {
                                                if (c<32)
                                                        PUTCH(127,tty->write_q);
                                                PUTCH(127,tty->write_q);
                                                tty->write(tty);
                                        }
                                        DEC(tty->secondary.head);
                                }
                                continue;
                        }
                        if (c==ERASE_CHAR(tty)) {
                                if (EMPTY(tty->secondary) ||
                                   (c=LAST(tty->secondary))==10 ||
                                   c==EOF_CHAR(tty))
                                        continue;
                                if (L_ECHO(tty)) {
                                        if (c<32)
                                                PUTCH(127,tty->write_q);
                                        PUTCH(127,tty->write_q);
                                        tty->write(tty);
                                }
                                DEC(tty->secondary.head);
                                continue;
                        }
                        if (c==STOP_CHAR(tty)) {
                                tty->stopped=1;
                                continue;
                        }
                        if (c==START_CHAR(tty)) {
                                tty->stopped=0;
                                continue;
                        }
                }
                if (L_ISIG(tty)) {
                        if (c==INTR_CHAR(tty)) {
                                tty_intr(tty,INTMASK);
                                continue;
                        }
                        if (c==QUIT_CHAR(tty)) {
                                tty_intr(tty,QUITMASK);
                                continue;
```

```
                        }
                }
                if (c==10 || c==EOF_CHAR(tty))
                        tty->secondary.data++;
                if (L_ECHO(tty)) {
                        if (c==10) {
                                PUTCH(10,tty->write_q);
                                PUTCH(13,tty->write_q);
                        } else if (c<32) {
                                if (L_ECHOCTL(tty)) {
                                        PUTCH('^',tty->write_q);
                                        PUTCH(c+64,tty->write_q);
                                }
                        } else
                                PUTCH(c,tty->write_q);
                        tty->write(tty);
                }
                PUTCH(c,tty->secondary);
        }
        wake_up(&tty->secondary.proc_list);
}

int tty_read(unsigned channel, char * buf, int nr)
{
        struct tty_struct * tty;
        char c, * b=buf;
        int minimum,time,flag=0;
        long oldalarm;

        if (channel>2 || nr<0) return -1;
        tty = &tty_table[channel];
        oldalarm = current->alarm;
        time = 10L*tty->termios.c_cc[VTIME];
        minimum = tty->termios.c_cc[VMIN];
        if (time && !minimum) {
                minimum=1;
                if (flag=(!oldalarm || time+jiffies<oldalarm))
                        current->alarm = time+jiffies;
        }
        if (minimum>nr)
                minimum=nr;
        while (nr>0) {
                if (flag && (current->signal & ALRMMASK)) {
                        current->signal &= ~ALRMMASK;
                        break;
                }
                if (current->signal)
                        break;
                if (EMPTY(tty->secondary) || (L_CANON(tty) &&
                !tty->secondary.data && LEFT(tty->secondary)>20)) {
                        sleep_if_empty(&tty->secondary);
                        continue;
                }
                do {
                        GETCH(tty->secondary,c);
                        if (c==EOF_CHAR(tty) || c==10)
                                tty->secondary.data--;
                        if (c==EOF_CHAR(tty) && L_CANON(tty))
                                return (b-buf);
                        else {
                                put_fs_byte(c,b++);
                                if (!--nr)
                                        break;
                        }
                } while (nr>0 && !EMPTY(tty->secondary));
                if (time && !L_CANON(tty))
                        if (flag=(!oldalarm || time+jiffies<oldalarm))
                                current->alarm = time+jiffies;
                        else
```

```
                                current->alarm = oldalarm;
                if (L_CANON(tty)) {
                        if (b-buf)
                                break;
                } else if (b-buf >= minimum)
                        break;
        }
        current->alarm = oldalarm;
        if (current->signal && !(b-buf))
                return -EINTR;
        return (b-buf);
}

int tty_write(unsigned channel, char * buf, int nr)
{
        static cr_flag=0;
        struct tty_struct * tty;
        char c, *b=buf;

        if (channel>2 || nr<0) return -1;
        tty = channel + tty_table;
        while (nr>0) {
                sleep_if_full(&tty->write_q);
                if (current->signal)
                        break;
                while (nr>0 && !FULL(tty->write_q)) {
                        c=get_fs_byte(b);
                        if (O_POST(tty)) {
                                if (c=='\r' && O_CRNL(tty))
                                        c='\n';
                                else if (c=='\n' && O_NLRET(tty))
                                        c='\r';
                                if (c=='\n' && !cr_flag && O_NLCR(tty)) {
                                        cr_flag = 1;
                                        PUTCH(13,tty->write_q);
                                        continue;
                                }
                                if (O_LCUC(tty))
                                        c=toupper(c);
                        }
                        b++; nr--;
                        cr_flag = 0;
                        PUTCH(c,tty->write_q);
                }
                tty->write(tty);
                if (nr>0)
                        schedule();
        }
        return (b-buf);
}

/*
 * Jeh, sometimes I really like the 386.
 * This routine is called from an interrupt,
 * and there should be absolutely no problem
 * with sleeping even in an interrupt (I hope).
 * Of course, if somebody proves me wrong, I'll
 * hate intel for all time :-). We'll have to
 * be careful and see to reinstating the interrupt
 * chips before calling this, though.
 *
 * I don't think we sleep here under normal circumstances
 * anyway, which is good, as the task sleeping might be
 * totally innocent.
 */
void do_tty_interrupt(int tty)
{
        copy_to_cooked(tty_table+tty);
}
```

```
void chr_dev_init(void)
{
}
```

```
/*
 *  linux/kernel/chr_drv/tty_ioctl.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <errno.h>
#include <termios.h>

#include <linux/sched.h>
#include <linux/kernel.h>
#include <linux/tty.h>

#include <asm/io.h>
#include <asm/segment.h>
#include <asm/system.h>

static unsigned short quotient[] = {
        0, 2304, 1536, 1047, 857,
        768, 576, 384, 192, 96,
        64, 48, 24, 12, 6, 3
};

static void change_speed(struct tty_struct * tty)
{
        unsigned short port,quot;

        if (!(port = tty->read_q.data))
                return;
        quot = quotient[tty->termios.c_cflag & CBAUD];
        cli();
        outb_p(0x80,port+3);            /* set DLAB */
        outb_p(quot & 0xff,port);       /* LS of divisor */
        outb_p(quot >> 8,port+1);       /* MS of divisor */
        outb(0x03,port+3);              /* reset DLAB */
        sti();
}

static void flush(struct tty_queue * queue)
{
        cli();
        queue->head = queue->tail;
        sti();
}

static void wait_until_sent(struct tty_struct * tty)
{
        /* do nothing - not implemented */
}

static void send_break(struct tty_struct * tty)
{
        /* do nothing - not implemented */
}

static int get_termios(struct tty_struct * tty, struct termios * termios)
{
        int i;

        verify_area(termios, sizeof (*termios));
        for (i=0 ; i< (sizeof (*termios)) ; i++)
                put_fs_byte( ((char *)&tty->termios)[i] , i+(char *)termios );
        return 0;
}

static int set_termios(struct tty_struct * tty, struct termios * termios)
{
        int i;
```

```
        for (i=0 ; i< (sizeof (*termios)) ; i++)
                ((char *)&tty->termios)[i]=get_fs_byte(i+(char *)termios);
        change_speed(tty);
        return 0;
}

static int get_termio(struct tty_struct * tty, struct termio * termio)
{
        int i;
        struct termio tmp_termio;

        verify_area(termio, sizeof (*termio));
        tmp_termio.c_iflag = tty->termios.c_iflag;
        tmp_termio.c_oflag = tty->termios.c_oflag;
        tmp_termio.c_cflag = tty->termios.c_cflag;
        tmp_termio.c_lflag = tty->termios.c_lflag;
        tmp_termio.c_line = tty->termios.c_line;
        for(i=0 ; i < NCC ; i++)
                tmp_termio.c_cc[i] = tty->termios.c_cc[i];
        for (i=0 ; i< (sizeof (*termio)) ; i++)
                put_fs_byte( ((char *)&tmp_termio)[i] , i+(char *)termio );
        return 0;
}

/*
 * This only works as the 386 is low-byt-first
 */
static int set_termio(struct tty_struct * tty, struct termio * termio)
{
        int i;
        struct termio tmp_termio;

        for (i=0 ; i< (sizeof (*termio)) ; i++)
                ((char *)&tmp_termio)[i]=get_fs_byte(i+(char *)termio);
        *(unsigned short *)&tty->termios.c_iflag = tmp_termio.c_iflag;
        *(unsigned short *)&tty->termios.c_oflag = tmp_termio.c_oflag;
        *(unsigned short *)&tty->termios.c_cflag = tmp_termio.c_cflag;
        *(unsigned short *)&tty->termios.c_lflag = tmp_termio.c_lflag;
        tty->termios.c_line = tmp_termio.c_line;
        for(i=0 ; i < NCC ; i++)
                tty->termios.c_cc[i] = tmp_termio.c_cc[i];
        change_speed(tty);
        return 0;
}

int tty_ioctl(int dev, int cmd, int arg)
{
        struct tty_struct * tty;
        if (MAJOR(dev) == 5) {
                dev=current->tty;
                if (dev<0)
                        panic("tty_ioctl: dev<0");
        } else
                dev=MINOR(dev);
        tty = dev + tty_table;
        switch (cmd) {
                case TCGETS:
                        return get_termios(tty,(struct termios *) arg);
                case TCSETSF:
                        flush(&tty->read_q); /* fallthrough */
                case TCSETSW:
                        wait_until_sent(tty); /* fallthrough */
                case TCSETS:
                        return set_termios(tty,(struct termios *) arg);
                case TCGETA:
                        return get_termio(tty,(struct termio *) arg);
                case TCSETAF:
                        flush(&tty->read_q); /* fallthrough */
                case TCSETAW:
```

```
                        wait_until_sent(tty); /* fallthrough */
                case TCSETA:
                        return set_termio(tty,(struct termio *) arg);
                case TCSBRK:
                        if (!arg) {
                                wait_until_sent(tty);
                                send_break(tty);
                        }
                        return 0;
                case TCXONC:
                        return -EINVAL; /* not implemented */
                case TCFLSH:
                        if (arg==0)
                                flush(&tty->read_q);
                        else if (arg==1)
                                flush(&tty->write_q);
                        else if (arg==2) {
                                flush(&tty->read_q);
                                flush(&tty->write_q);
                        } else
                                return -EINVAL;
                        return 0;
                case TIOCEXCL:
                        return -EINVAL; /* not implemented */
                case TIOCNXCL:
                        return -EINVAL; /* not implemented */
                case TIOCSCTTY:
                        return -EINVAL; /* set controlling term NI */
                case TIOCGPGRP:
                        verify_area((void *) arg,4);
                        put_fs_long(tty->pgrp,(unsigned long *) arg);
                        return 0;
                case TIOCSPGRP:
                        tty->pgrp=get_fs_long((unsigned long *) arg);
                        return 0;
                case TIOCOUTQ:
                        verify_area((void *) arg,4);
                        put_fs_long(CHARS(tty->write_q),(unsigned long *) arg);
                        return 0;
                case TIOCINQ:
                        verify_area((void *) arg,4);
                        put_fs_long(CHARS(tty->secondary),
                                (unsigned long *) arg);
                        return 0;
                case TIOCSTI:
                        return -EINVAL; /* not implemented */
                case TIOCGWINSZ:
                        return -EINVAL; /* not implemented */
                case TIOCSWINSZ:
                        return -EINVAL; /* not implemented */
                case TIOCMGET:
                        return -EINVAL; /* not implemented */
                case TIOCMBIS:
                        return -EINVAL; /* not implemented */
                case TIOCMBIC:
                        return -EINVAL; /* not implemented */
                case TIOCMSET:
                        return -EINVAL; /* not implemented */
                case TIOCGSOFTCAR:
                        return -EINVAL; /* not implemented */
                case TIOCSSOFTCAR:
                        return -EINVAL; /* not implemented */
                default:
                        return -EINVAL;
        }
}
```

```
#
# Makefile for the FREAX-kernel character device drivers.
#
# Note! Dependencies are done automagically by 'make dep', which also
# removes any old dependencies. DON'T put your own dependencies here
# unless it's something special (ie not a .c file).
#

AR      =gar
AS      =gas
LD      =gld
LDFLAGS =-s -x
CC      =gcc
CFLAGS  =-Wall -O -fstrength-reduce -fomit-frame-pointer -fcombine-regs \
        -finline-functions -mstring-insns -nostdinc -I../../include
CPP     =gcc -E -nostdinc -I../../include

.c.s:
        $(CC) $(CFLAGS) \
        -S -o $*.s $<
.s.o:
        $(AS) -c -o $*.o $<
.c.o:
        $(CC) $(CFLAGS) \
        -c -o $*.o $<

OBJS  = math_emulate.o

math.a: $(OBJS)
        $(AR) rcs math.a $(OBJS)
        sync

clean:
        rm -f core *.o *.a tmp_make
        for i in *.c;do rm -f `basename $$i .c`.s;done

dep:
        sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
        (for i in *.c;do echo -n `echo $$i | sed 's,\.c,\.s,'`" "; \
                $(CPP) -M $$i;done) >> tmp_make
        cp tmp_make Makefile

### Dependencies:
```

```
/*
 * linux/kernel/math/math_emulate.c
 *
 * (C) 1991 Linus Torvalds
 */

/*
 * This directory should contain the math-emulation code.
 * Currently only results in a signal.
 */

#include <signal.h>

#include <linux/sched.h>
#include <linux/kernel.h>
#include <asm/segment.h>

void math_emulate(long edi, long esi, long ebp, long sys_call_ret,
        long eax,long ebx,long ecx,long edx,
        unsigned short fs,unsigned short es,unsigned short ds,
        unsigned long eip,unsigned short cs,unsigned long eflags,
        unsigned short ss, unsigned long esp)
{
        unsigned char first, second;

/* 0x0007 means user code space */
        if (cs != 0x000F) {
                printk("math_emulate: %04x:%08x\n\r",cs,eip);
                panic("Math emulation needed in kernel");
        }
        first = get_fs_byte((char *)((*&eip)++));
        second = get_fs_byte((char *)((*&eip)++));
        printk("%04x:%08x %02x %02x\n\r",cs,eip-2,first,second);
        current->signal |= 1<<(SIGFPE-1);
}

void math_error(void)
{
        __asm__("fnclex");
        if (last_task_used_math)
                last_task_used_math->signal |= 1<<(SIGFPE-1);
}
```

```
#
# Makefile for some libs needed in the kernel.
#
# Note! Dependencies are done automagically by 'make dep', which also
# removes any old dependencies. DON'T put your own dependencies here
# unless it's something special (ie not a .c file).
#

AR      =gar
AS      =gas
LD      =gld
LDFLAGS =-s -x
CC      =gcc
CFLAGS  =-Wall -O -fstrength-reduce -fomit-frame-pointer -fcombine-regs \
        -finline-functions -mstring-insns -nostdinc -I../include
CPP     =gcc -E -nostdinc -I../include

.c.s:
        $(CC) $(CFLAGS) \
        -S -o $*.s $<
.s.o:
        $(AS) -c -o $*.o $<
.c.o:
        $(CC) $(CFLAGS) \
        -c -o $*.o $<

OBJS  = ctype.o _exit.o open.o close.o errno.o write.o dup.o setsid.o \
        execve.o wait.o string.o malloc.o

lib.a: $(OBJS)
        $(AR) rcs lib.a $(OBJS)
        sync

clean:
        rm -f core *.o *.a tmp_make
        for i in *.c;do rm -f `basename $$i .c`.s;done

dep:
        sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
        (for i in *.c;do echo -n `echo $$i | sed 's,\.c,\.s,'`" "; \
                $(CPP) -M $$i;done) >> tmp_make
        cp tmp_make Makefile

### Dependencies:
_exit.s _exit.o : _exit.c ../include/unistd.h ../include/sys/stat.h \
  ../include/sys/types.h ../include/sys/times.h ../include/sys/utsname.h \
  ../include/utime.h
close.s close.o : close.c ../include/unistd.h ../include/sys/stat.h \
  ../include/sys/types.h ../include/sys/times.h ../include/sys/utsname.h \
  ../include/utime.h
ctype.s ctype.o : ctype.c ../include/ctype.h
dup.s dup.o : dup.c ../include/unistd.h ../include/sys/stat.h \
  ../include/sys/types.h ../include/sys/times.h ../include/sys/utsname.h \
  ../include/utime.h
errno.s errno.o : errno.c
execve.s execve.o : execve.c ../include/unistd.h ../include/sys/stat.h \
  ../include/sys/types.h ../include/sys/times.h ../include/sys/utsname.h \
  ../include/utime.h
malloc.s malloc.o : malloc.c ../include/linux/kernel.h ../include/linux/mm.h \
  ../include/asm/system.h
open.s open.o : open.c ../include/unistd.h ../include/sys/stat.h \
  ../include/sys/types.h ../include/sys/times.h ../include/sys/utsname.h \
  ../include/utime.h ../include/stdarg.h
setsid.s setsid.o : setsid.c ../include/unistd.h ../include/sys/stat.h \
  ../include/sys/types.h ../include/sys/times.h ../include/sys/utsname.h \
  ../include/utime.h
string.s string.o : string.c ../include/string.h
wait.s wait.o : wait.c ../include/unistd.h ../include/sys/stat.h \
  ../include/sys/types.h ../include/sys/times.h ../include/sys/utsname.h \
```

```
    ../include/utime.h ../include/sys/wait.h
write.s write.o : write.c ../include/unistd.h ../include/sys/stat.h \
    ../include/sys/types.h ../include/sys/times.h ../include/sys/utsname.h \
    ../include/utime.h
```

```
/*
 *  linux/lib/_exit.c
 *
 *  (C) 1991  Linus Torvalds
 */

#define __LIBRARY__
#include <unistd.h>

volatile void _exit(int exit_code)
{
        __asm__("int $0x80"::"a" (__NR_exit),"b" (exit_code));
}
```

```
/*
 *  linux/lib/close.c
 *
 *  (C) 1991  Linus Torvalds
 */

#define __LIBRARY__
#include <unistd.h>

_syscall1(int,close,int,fd)
```

```
/*
 *  linux/lib/ctype.c
 *
 *  (C) 1991  Linus Torvalds
 */

#include <ctype.h>

char _ctmp;
unsigned char _ctype[] = {0x00,                 /* EOF */
_C,_C,_C,_C,_C,_C,_C,_C,                        /* 0-7 */
_C,_C|_S,_C|_S,_C|_S,_C|_S,_C|_S,_C,_C,         /* 8-15 */
_C,_C,_C,_C,_C,_C,_C,_C,                        /* 16-23 */
_C,_C,_C,_C,_C,_C,_C,_C,                        /* 24-31 */
_S|_SP,_P,_P,_P,_P,_P,_P,_P,                    /* 32-39 */
_P,_P,_P,_P,_P,_P,_P,_P,                        /* 40-47 */
_D,_D,_D,_D,_D,_D,_D,_D,                        /* 48-55 */
_D,_D,_P,_P,_P,_P,_P,_P,                        /* 56-63 */
_P,_U|_X,_U|_X,_U|_X,_U|_X,_U|_X,_U|_X,_U,      /* 64-71 */
_U,_U,_U,_U,_U,_U,_U,_U,                        /* 72-79 */
_U,_U,_U,_U,_U,_U,_U,_U,                        /* 80-87 */
_U,_U,_U,_P,_P,_P,_P,_P,                        /* 88-95 */
_P,_L|_X,_L|_X,_L|_X,_L|_X,_L|_X,_L|_X,_L,      /* 96-103 */
_L,_L,_L,_L,_L,_L,_L,_L,                        /* 104-111 */
_L,_L,_L,_L,_L,_L,_L,_L,                        /* 112-119 */
_L,_L,_L,_P,_P,_P,_P,_C,                        /* 120-127 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,                /* 128-143 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,                /* 144-159 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,                /* 160-175 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,                /* 176-191 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,                /* 192-207 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,                /* 208-223 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,                /* 224-239 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};               /* 240-255 */
```

```
/*
 *  linux/lib/dup.c
 *
 *  (C) 1991  Linus Torvalds
 */

#define __LIBRARY__
#include <unistd.h>

_syscall1(int,dup,int,fd)
```

```
/*
 *  linux/lib/errno.c
 *
 *  (C) 1991  Linus Torvalds
 */

int errno;
```

```
/*
 *  linux/lib/execve.c
 *
 *  (C) 1991  Linus Torvalds
 */

#define __LIBRARY__
#include <unistd.h>

_syscall3(int,execve,const char *,file,char **,argv,char **,envp)
```

```
/*
 * malloc.c --- a general purpose kernel memory allocator for Linux.
 *
 * Written by Theodore Ts'o (tytso@mit.edu), 11/29/91
 *
 * This routine is written to be as fast as possible, so that it
 * can be called from the interrupt level.
 *
 * Limitations: maximum size of memory we can allocate using this routine
 *      is 4k, the size of a page in Linux.
 *
 * The general game plan is that each page (called a bucket) will only hold
 * objects of a given size.  When all of the object on a page are released,
 * the page can be returned to the general free pool.  When malloc() is
 * called, it looks for the smallest bucket size which will fulfill its
 * request, and allocate a piece of memory from that bucket pool.
 *
 * Each bucket has as its control block a bucket descriptor which keeps
 * track of how many objects are in use on that page, and the free list
 * for that page.  Like the buckets themselves, bucket descriptors are
 * stored on pages requested from get_free_page().  However, unlike buckets,
 * pages devoted to bucket descriptor pages are never released back to the
 * system.  Fortunately, a system should probably only need 1 or 2 bucket
 * descriptor pages, since a page can hold 256 bucket descriptors (which
 * corresponds to 1 megabyte worth of bucket pages.)  If the kernel is using
 * that much allocated memory, it's probably doing something wrong.  :-)
 *
 * Note: malloc() and free() both call get_free_page() and free_page()
 *      in sections of code where interrupts are turned off, to allow
 *      malloc() and free() to be safely called from an interrupt routine.
 *      (We will probably need this functionality when networking code,
 *      particularily things like NFS, is added to Linux.)  However, this
 *      presumes that get_free_page() and free_page() are interrupt-level
 *      safe, which they may not be once paging is added.  If this is the
 *      case, we will need to modify malloc() to keep a few unused pages
 *      "pre-allocated" so that it can safely draw upon those pages if
 *      it is called from an interrupt routine.
 *
 *      Another concern is that get_free_page() should not sleep; if it
 *      does, the code is carefully ordered so as to avoid any race
 *      conditions.  The catch is that if malloc() is called re-entrantly,
 *      there is a chance that unecessary pages will be grabbed from the
 *      system.  Except for the pages for the bucket descriptor page, the
 *      extra pages will eventually get released back to the system, though,
 *      so it isn't all that bad.
 */

#include <linux/kernel.h>
#include <linux/mm.h>
#include <asm/system.h>

struct bucket_desc {    /* 16 bytes */
        void                    *page;
        struct bucket_desc      *next;
        void                    *freeptr;
        unsigned short          refcnt;
        unsigned short          bucket_size;
};

struct _bucket_dir {    /* 8 bytes */
        int                     size;
        struct bucket_desc      *chain;
};

/*
 * The following is the where we store a pointer to the first bucket
 * descriptor for a given size.
 *
 * If it turns out that the Linux kernel allocates a lot of objects of a
```

```
 * specific size, then we may want to add that specific size to this list,
 * since that will allow the memory to be allocated more efficiently.
 * However, since an entire page must be dedicated to each specific size
 * on this list, some amount of temperance must be exercised here.
 *
 * Note that this list *must* be kept in order.
 */
struct _bucket_dir bucket_dir[] = {
        { 16,   (struct bucket_desc *) 0},
        { 32,   (struct bucket_desc *) 0},
        { 64,   (struct bucket_desc *) 0},
        { 128,  (struct bucket_desc *) 0},
        { 256,  (struct bucket_desc *) 0},
        { 512,  (struct bucket_desc *) 0},
        { 1024, (struct bucket_desc *) 0},
        { 2048, (struct bucket_desc *) 0},
        { 4096, (struct bucket_desc *) 0},
        { 0,    (struct bucket_desc *) 0}};   /* End of list marker */

/*
 * This contains a linked list of free bucket descriptor blocks
 */
struct bucket_desc *free_bucket_desc = (struct bucket_desc *) 0;

/*
 * This routine initializes a bucket description page.
 */
static inline void init_bucket_desc()
{
        struct bucket_desc *bdesc, *first;
        int     i;

        first = bdesc = (struct bucket_desc *) get_free_page();
        if (!bdesc)
                panic("Out of memory in init_bucket_desc()");
        for (i = PAGE_SIZE/sizeof(struct bucket_desc); i > 1; i--) {
                bdesc->next = bdesc+1;
                bdesc++;
        }
        /*
         * This is done last, to avoid race conditions in case
         * get_free_page() sleeps and this routine gets called again....
         */
        bdesc->next = free_bucket_desc;
        free_bucket_desc = first;
}

void *malloc(unsigned int len)
{
        struct _bucket_dir      *bdir;
        struct bucket_desc      *bdesc;
        void                    *retval;

        /*
         * First we search the bucket_dir to find the right bucket change
         * for this request.
         */
        for (bdir = bucket_dir; bdir->size; bdir++)
                if (bdir->size >= len)
                        break;
        if (!bdir->size) {
                printk("malloc called with impossibly large argument (%d)\n",
                        len);
                panic("malloc: bad arg");
        }
        /*
         * Now we search for a bucket descriptor which has free space
         */
        cli();  /* Avoid race conditions */
```

```
	for (bdesc = bdir->chain; bdesc; bdesc = bdesc->next)
		if (bdesc->freeptr)
			break;
	/*
	 * If we didn't find a bucket with free space, then we'll
	 * allocate a new one.
	 */
	if (!bdesc) {
		char		*cp;
		int		i;

		if (!free_bucket_desc)
			init_bucket_desc();
		bdesc = free_bucket_desc;
		free_bucket_desc = bdesc->next;
		bdesc->refcnt = 0;
		bdesc->bucket_size = bdir->size;
		bdesc->page = bdesc->freeptr = (void *) cp = get_free_page();
		if (!cp)
			panic("Out of memory in kernel malloc()");
		/* Set up the chain of free objects */
		for (i=PAGE_SIZE/bdir->size; i > 1; i--) {
			*((char **) cp) = cp + bdir->size;
			cp += bdir->size;
		}
		*((char **) cp) = 0;
		bdesc->next = bdir->chain; /* OK, link it in! */
		bdir->chain = bdesc;
	}
	retval = (void *) bdesc->freeptr;
	bdesc->freeptr = *((void **) retval);
	bdesc->refcnt++;
	sti();	/* OK, we're safe again */
	return(retval);
}

/*
 * Here is the free routine.  If you know the size of the object that you
 * are freeing, then free_s() will use that information to speed up the
 * search for the bucket descriptor.
 *
 * We will #define a macro so that "free(x)" is becomes "free_s(x, 0)"
 */
void free_s(void *obj, int size)
{
	void		*page;
	struct _bucket_dir	*bdir;
	struct bucket_desc	*bdesc, *prev;

	/* Calculate what page this object lives in */
	page = (void *)  ((unsigned long) obj & 0xfffff000);
	/* Now search the buckets looking for that page */
	for (bdir = bucket_dir; bdir->size; bdir++) {
		prev = 0;
		/* If size is zero then this conditional is always false */
		if (bdir->size < size)
			continue;
		for (bdesc = bdir->chain; bdesc; bdesc = bdesc->next) {
			if (bdesc->page == page)
				goto found;
			prev = bdesc;
		}
	}
	panic("Bad address passed to kernel free_s()");
found:
	cli(); /* To avoid race conditions */
	*((void **)obj) = bdesc->freeptr;
	bdesc->freeptr = obj;
	bdesc->refcnt--;
```

```
        if (bdesc->refcnt == 0) {
                /*
                 * We need to make sure that prev is still accurate.  It
                 * may not be, if someone rudely interrupted us....
                 */
                if ((prev && (prev->next != bdesc)) ||
                    (!prev && (bdir->chain != bdesc)))
                        for (prev = bdir->chain; prev; prev = prev->next)
                                if (prev->next == bdesc)
                                        break;
                if (prev)
                        prev->next = bdesc->next;
                else {
                        if (bdir->chain != bdesc)
                                panic("malloc bucket chains corrupted");
                        bdir->chain = bdesc->next;
                }
                free_page((unsigned long) bdesc->page);
                bdesc->next = free_bucket_desc;
                free_bucket_desc = bdesc;
        }
        sti();
        return;
}
```

```
/*
 *  linux/lib/open.c
 *
 *  (C) 1991  Linus Torvalds
 */

#define __LIBRARY__
#include <unistd.h>
#include <stdarg.h>

int open(const char * filename, int flag, ...)
{
        register int res;
        va_list arg;

        va_start(arg,flag);
        __asm__("int $0x80"
                :"=a" (res)
                :"0" (__NR_open),"b" (filename),"c" (flag),
                "d" (va_arg(arg,int)));
        if (res>=0)
                return res;
        errno = -res;
        return -1;
}
```

```
/*
 *  linux/lib/setsid.c
 *
 *  (C) 1991  Linus Torvalds
 */

#define __LIBRARY__
#include <unistd.h>

_syscall0(pid_t,setsid)
```

```
/*
 *  linux/lib/string.c
 *
 *  (C) 1991  Linus Torvalds
 */

#ifndef __GNUC__
#error I want gcc!
#endif

#define extern
#define inline
#define __LIBRARY__
#include <string.h>
```

```
/*
 *  linux/lib/wait.c
 *
 *  (C) 1991  Linus Torvalds
 */

#define __LIBRARY__
#include <unistd.h>
#include <sys/wait.h>

_syscall3(pid_t,waitpid,pid_t,pid,int *,wait_stat,int,options)

pid_t wait(int * wait_stat)
{
        return waitpid(-1,wait_stat,0);
}
```

```
/*
 *  linux/lib/write.c
 *
 *  (C) 1991  Linus Torvalds
 */

#define __LIBRARY__
#include <unistd.h>

_syscall3(int,write,int,fd,const char *,buf,off_t,count)
```

```
CC      =gcc
CFLAGS  =-O -Wall -fstrength-reduce -fcombine-regs -fomit-frame-pointer \
        -finline-functions -nostdinc -I../include
AS      =gas
AR      =gar
LD      =gld
CPP     =gcc -E -nostdinc -I../include

.c.o:
        $(CC) $(CFLAGS) \
        -c -o $*.o $<
.s.o:
        $(AS) -o $*.o $<
.c.s:
        $(CC) $(CFLAGS) \
        -S -o $*.s $<

OBJS    = memory.o page.o

all: mm.o

mm.o: $(OBJS)
        $(LD) -r -o mm.o $(OBJS)

clean:
        rm -f core *.o *.a tmp_make
        for i in *.c;do rm -f `basename $$i .c`.s;done

dep:
        sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
        (for i in *.c;do $(CPP) -M $$i;done) >> tmp_make
        cp tmp_make Makefile

### Dependencies:
memory.o : memory.c ../include/signal.h ../include/sys/types.h \
  ../include/asm/system.h ../include/linux/sched.h ../include/linux/head.h \
  ../include/linux/fs.h ../include/linux/mm.h ../include/linux/kernel.h
```

```
/*
 *  linux/mm/memory.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * demand-loading started 01.12.91 - seems it is high on the list of
 * things wanted, and it should be easy to implement. - Linus
 */

/*
 * Ok, demand-loading was easy, shared pages a little bit tricker. Shared
 * pages started 02.12.91, seems to work. - Linus.
 *
 * Tested sharing by executing about 30 /bin/sh: under the old kernel it
 * would have taken more than the 6M I have free, but it worked well as
 * far as I could see.
 *
 * Also corrected some "invalidate()"s - I wasn't doing enough of them.
 */

#include <signal.h>

#include <asm/system.h>

#include <linux/sched.h>
#include <linux/head.h>
#include <linux/kernel.h>

volatile void do_exit(long code);

static inline volatile void oom(void)
{
        printk("out of memory\n\r");
        do_exit(SIGSEGV);
}

#define invalidate() \
__asm__("movl %%eax,%%cr3"::"a" (0))

/* these are not to be changed without changing head.s etc */
#define LOW_MEM 0x100000
#define PAGING_MEMORY (15*1024*1024)
#define PAGING_PAGES (PAGING_MEMORY>>12)
#define MAP_NR(addr) (((addr)-LOW_MEM)>>12)
#define USED 100

#define CODE_SPACE(addr) ((((addr)+4095)&~4095) < \
current->start_code + current->end_code)

static long HIGH_MEMORY = 0;

#define copy_page(from,to) \
__asm__("cld ; rep ; movsl"::"S" (from),"D" (to),"c" (1024):"cx","di","si")

static unsigned char mem_map [ PAGING_PAGES ] = {0,};

/*
 * Get physical address of first (actually last :-) free page, and mark it
 * used. If no free pages left, return 0.
 */
unsigned long get_free_page(void)
{
register unsigned long __res asm("ax");

__asm__("std ; repne ; scasb\n\t"
        "jne 1f\n\t"
        "movb $1,1(%%edi)\n\t"
```

```
        "sall $12,%%ecx\n\t"
        "addl %2,%%ecx\n\t"
        "movl %%ecx,%%edx\n\t"
        "movl $1024,%%ecx\n\t"
        "leal 4092(%%edx),%%edi\n\t"
        "rep ; stosl\n\t"
        "movl %%edx,%%eax\n"
        "1:"
        :"=a" (__res)
        :"0" (0),"i" (LOW_MEM),"c" (PAGING_PAGES),
        "D" (mem_map+PAGING_PAGES-1)
        :"di","cx","dx");
return __res;
}

/*
 * Free a page of memory at physical address 'addr'. Used by
 * 'free_page_tables()'
 */
void free_page(unsigned long addr)
{
        if (addr < LOW_MEM) return;
        if (addr >= HIGH_MEMORY)
                panic("trying to free nonexistent page");
        addr -= LOW_MEM;
        addr >>= 12;
        if (mem_map[addr]--) return;
        mem_map[addr]=0;
        panic("trying to free free page");
}

/*
 * This function frees a continuos block of page tables, as needed
 * by 'exit()'. As does copy_page_tables(), this handles only 4Mb blocks.
 */
int free_page_tables(unsigned long from,unsigned long size)
{
        unsigned long *pg_table;
        unsigned long * dir, nr;

        if (from & 0x3fffff)
                panic("free_page_tables called with wrong alignment");
        if (!from)
                panic("Trying to free up swapper memory space");
        size = (size + 0x3fffff) >> 22;
        dir = (unsigned long *) ((from>>20) & 0xffc); /* _pg_dir = 0 */
        for ( ; size-->0 ; dir++) {
                if (!(1 & *dir))
                        continue;
                pg_table = (unsigned long *) (0xfffff000 & *dir);
                for (nr=0 ; nr<1024 ; nr++) {
                        if (1 & *pg_table)
                                free_page(0xfffff000 & *pg_table);
                        *pg_table = 0;
                        pg_table++;
                }
                free_page(0xfffff000 & *dir);
                *dir = 0;
        }
        invalidate();
        return 0;
}

/*
 *  Well, here is one of the most complicated functions in mm. It
 * copies a range of linerar addresses by copying only the pages.
 * Let's hope this is bug-free, 'cause this one I don't want to debug :-)
 *
 * Note! We don't copy just any chunks of memory - addresses have to
```

```
 * be divisible by 4Mb (one page-directory entry), as this makes the
 * function easier. It's used only by fork anyway.
 *
 * NOTE 2!! When from==0 we are copying kernel space for the first
 * fork(). Then we DONT want to copy a full page-directory entry, as
 * that would lead to some serious memory waste - we just copy the
 * first 160 pages - 640kB. Even that is more than we need, but it
 * doesn't take any more memory - we don't copy-on-write in the low
 * 1 Mb-range, so the pages can be shared with the kernel. Thus the
 * special case for nr=xxxx.
 */
int copy_page_tables(unsigned long from,unsigned long to,long size)
{
        unsigned long * from_page_table;
        unsigned long * to_page_table;
        unsigned long this_page;
        unsigned long * from_dir, * to_dir;
        unsigned long nr;

        if ((from&0x3fffff) || (to&0x3fffff))
                panic("copy_page_tables called with wrong alignment");
        from_dir = (unsigned long *) ((from>>20) & 0xffc); /* _pg_dir = 0 */
        to_dir = (unsigned long *) ((to>>20) & 0xffc);
        size = ((unsigned) (size+0x3fffff)) >> 22;
        for( ; size-->0 ; from_dir++,to_dir++) {
                if (1 & *to_dir)
                        panic("copy_page_tables: already exist");
                if (!(1 & *from_dir))
                        continue;
                from_page_table = (unsigned long *) (0xfffff000 & *from_dir);
                if (!(to_page_table = (unsigned long *) get_free_page()))
                        return -1;      /* Out of memory, see freeing */
                *to_dir = ((unsigned long) to_page_table) | 7;
                nr = (from==0)?0xA0:1024;
                for ( ; nr-- > 0 ; from_page_table++,to_page_table++) {
                        this_page = *from_page_table;
                        if (!(1 & this_page))
                                continue;
                        this_page &= ~2;
                        *to_page_table = this_page;
                        if (this_page > LOW_MEM) {
                                *from_page_table = this_page;
                                this_page -= LOW_MEM;
                                this_page >>= 12;
                                mem_map[this_page]++;
                        }
                }
        }
        invalidate();
        return 0;
}

/*
 * This function puts a page in memory at the wanted address.
 * It returns the physical address of the page gotten, 0 if
 * out of memory (either when trying to access page-table or
 * page.)
 */
unsigned long put_page(unsigned long page,unsigned long address)
{
        unsigned long tmp, *page_table;

/* NOTE !!! This uses the fact that _pg_dir=0 */

        if (page < LOW_MEM || page >= HIGH_MEMORY)
                printk("Trying to put page %p at %p\n",page,address);
        if (mem_map[(page-LOW_MEM)>>12] != 1)
                printk("mem_map disagrees with %p at %p\n",page,address);
        page_table = (unsigned long *) ((address>>20) & 0xffc);
```

```
        if ((*page_table)&1)
                page_table = (unsigned long *) (0xfffff000 & *page_table);
        else {
                if (!(tmp=get_free_page()))
                        return 0;
                *page_table = tmp|7;
                page_table = (unsigned long *) tmp;
        }
        page_table[(address>>12) & 0x3ff] = page | 7;
/* no need for invalidate */
        return page;
}

void un_wp_page(unsigned long * table_entry)
{
        unsigned long old_page,new_page;

        old_page = 0xfffff000 & *table_entry;
        if (old_page >= LOW_MEM && mem_map[MAP_NR(old_page)]==1) {
                *table_entry |= 2;
                invalidate();
                return;
        }
        if (!(new_page=get_free_page()))
                oom();
        if (old_page >= LOW_MEM)
                mem_map[MAP_NR(old_page)]--;
        *table_entry = new_page | 7;
        invalidate();
        copy_page(old_page,new_page);
}

/*
 * This routine handles present pages, when users try to write
 * to a shared page. It is done by copying the page to a new address
 * and decrementing the shared-page counter for the old page.
 *
 * If it's in code space we exit with a segment error.
 */
void do_wp_page(unsigned long error_code,unsigned long address)
{
#if 0
/* we cannot do this yet: the estdio library writes to code space */
/* stupid, stupid. I really want the libc.a from GNU */
        if (CODE_SPACE(address))
                do_exit(SIGSEGV);
#endif
        un_wp_page((unsigned long *)
                (((address>>10) & 0xffc) + (0xfffff000 &
                *((unsigned long *) ((address>>20) &0xffc)))));

}

void write_verify(unsigned long address)
{
        unsigned long page;

        if (!( (page = *((unsigned long *) ((address>>20) & 0xffc)) )&1))
                return;
        page &= 0xfffff000;
        page += ((address>>10) & 0xffc);
        if ((3 & *(unsigned long *) page) == 1)  /* non-writeable, present */
                un_wp_page((unsigned long *) page);
        return;
}

void get_empty_page(unsigned long address)
{
        unsigned long tmp;
```

```
        if (!(tmp=get_free_page()) || !put_page(tmp,address)) {
                free_page(tmp);         /* 0 is ok - ignored */
                oom();
        }
}

/*
 * try_to_share() checks the page at address "address" in the task "p",
 * to see if it exists, and if it is clean. If so, share it with the current
 * task.
 *
 * NOTE! This assumes we have checked that p != current, and that they
 * share the same executable.
 */
static int try_to_share(unsigned long address, struct task_struct * p)
{
        unsigned long from;
        unsigned long to;
        unsigned long from_page;
        unsigned long to_page;
        unsigned long phys_addr;

        from_page = to_page = ((address>>20) & 0xffc);
        from_page += ((p->start_code>>20) & 0xffc);
        to_page += ((current->start_code>>20) & 0xffc);
/* is there a page-directory at from? */
        from = *(unsigned long *) from_page;
        if (!(from & 1))
                return 0;
        from &= 0xfffff000;
        from_page = from + ((address>>10) & 0xffc);
        phys_addr = *(unsigned long *) from_page;
/* is the page clean and present? */
        if ((phys_addr & 0x41) != 0x01)
                return 0;
        phys_addr &= 0xfffff000;
        if (phys_addr >= HIGH_MEMORY || phys_addr < LOW_MEM)
                return 0;
        to = *(unsigned long *) to_page;
        if (!(to & 1))
                if (to = get_free_page())
                        *(unsigned long *) to_page = to | 7;
                else
                        oom();
        to &= 0xfffff000;
        to_page = to + ((address>>10) & 0xffc);
        if (1 & *(unsigned long *) to_page)
                panic("try_to_share: to_page already exists");
/* share them: write-protect */
        *(unsigned long *) from_page &= ~2;
        *(unsigned long *) to_page = *(unsigned long *) from_page;
        invalidate();
        phys_addr -= LOW_MEM;
        phys_addr >>= 12;
        mem_map[phys_addr]++;
        return 1;
}

/*
 * share_page() tries to find a process that could share a page with
 * the current one. Address is the address of the wanted page relative
 * to the current data space.
 *
 * We first check if it is at all feasible by checking executable->i_count.
 * It should be >1 if there are other tasks sharing this inode.
 */
static int share_page(unsigned long address)
{
```

```
        struct task_struct ** p;

        if (!current->executable)
                return 0;
        if (current->executable->i_count < 2)
                return 0;
        for (p = &LAST_TASK ; p > &FIRST_TASK ; --p) {
                if (!*p)
                        continue;
                if (current == *p)
                        continue;
                if ((*p)->executable != current->executable)
                        continue;
                if (try_to_share(address,*p))
                        return 1;
        }
        return 0;
}

void do_no_page(unsigned long error_code,unsigned long address)
{
        int nr[4];
        unsigned long tmp;
        unsigned long page;
        int block,i;

        address &= 0xfffff000;
        tmp = address - current->start_code;
        if (!current->executable || tmp >= current->end_data) {
                get_empty_page(address);
                return;
        }
        if (share_page(tmp))
                return;
        if (!(page = get_free_page()))
                oom();
/* remember that 1 block is used for header */
        block = 1 + tmp/BLOCK_SIZE;
        for (i=0 ; i<4 ; block++,i++)
                nr[i] = bmap(current->executable,block);
        bread_page(page,current->executable->i_dev,nr);
        i = tmp + 4096 - current->end_data;
        tmp = page + 4096;
        while (i-- > 0) {
                tmp--;
                *(char *)tmp = 0;
        }
        if (put_page(page,address))
                return;
        free_page(page);
        oom();
}

void mem_init(long start_mem, long end_mem)
{
        int i;

        HIGH_MEMORY = end_mem;
        for (i=0 ; i<PAGING_PAGES ; i++)
                mem_map[i] = USED;
        i = MAP_NR(start_mem);
        end_mem -= start_mem;
        end_mem >>= 12;
        while (end_mem-->0)
                mem_map[i++]=0;
}

void calc_mem(void)
{
```

```
        int i,j,k,free=0;
        long * pg_tbl;

        for(i=0 ; i<PAGING_PAGES ; i++)
                if (!mem_map[i]) free++;
        printk("%d pages free (of %d)\n\r",free,PAGING_PAGES);
        for(i=2 ; i<1024 ; i++) {
                if (1&pg_dir[i]) {
                        pg_tbl=(long *) (0xfffff000 & pg_dir[i]);
                        for(j=k=0 ; j<1024 ; j++)
                                if (pg_tbl[j]&1)
                                        k++;
                        printk("Pg-dir[%d] uses %d pages\n",i,k);
                }
        }
}
```

```
/*
 *  linux/mm/page.s
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * page.s contains the low-level page-exception code.
 * the real work is done in mm.c
 */

.globl _page_fault

_page_fault:
        xchgl %eax,(%esp)
        pushl %ecx
        pushl %edx
        push %ds
        push %es
        push %fs
        movl $0x10,%edx
        mov %dx,%ds
        mov %dx,%es
        mov %dx,%fs
        movl %cr2,%edx
        pushl %edx
        pushl %eax
        testl $1,%eax
        jne 1f
        call _do_no_page
        jmp 2f
1:      call _do_wp_page
2:      addl $8,%esp
        pop %fs
        pop %es
        pop %ds
        popl %edx
        popl %ecx
        popl %eax
        iret
```

```
/*
 *  linux/tools/build.c
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 * This file builds a disk-image from three different files:
 *
 * - bootsect: max 510 bytes of 8086 machine code, loads the rest
 * - setup: max 4 sectors of 8086 machine code, sets up system parm
 * - system: 80386 code for actual system
 *
 * It does some checking that all files are of the correct type, and
 * just writes the result to stdout, removing headers and padding to
 * the right amount. It also writes some system data to stderr.
 */

/*
 * Changes by tytso to allow root device specification
 */

#include <stdio.h>      /* fprintf */
#include <string.h>
#include <stdlib.h>     /* contains exit */
#include <sys/types.h>  /* unistd.h needs this */
#include <sys/stat.h>
#include <linux/fs.h>
#include <unistd.h>     /* contains read/write */
#include <fcntl.h>

#define MINIX_HEADER 32
#define GCC_HEADER 1024

#define SYS_SIZE 0x2000

#define DEFAULT_MAJOR_ROOT 3
#define DEFAULT_MINOR_ROOT 6

/* max nr of sectors of setup: don't change unless you also change
 * bootsect etc */
#define SETUP_SECTS 4

#define STRINGIFY(x) #x

void die(char * str)
{
        fprintf(stderr,"%s\n",str);
        exit(1);
}

void usage(void)
{
        die("Usage: build bootsect setup system [rootdev] [> image]");
}

int main(int argc, char ** argv)
{
        int i,c,id;
        char buf[1024];
        char major_root, minor_root;
        struct stat sb;

        if ((argc != 4) && (argc != 5))
                usage();
        if (argc == 5) {
                if (strcmp(argv[4], "FLOPPY")) {
                        if (stat(argv[4], &sb)) {
                                perror(argv[4]);
```

```
                        die("Couldn't stat root device.");
                }
                major_root = MAJOR(sb.st_rdev);
                minor_root = MINOR(sb.st_rdev);
        } else {
                major_root = 0;
                minor_root = 0;
        }
} else {
        major_root = DEFAULT_MAJOR_ROOT;
        minor_root = DEFAULT_MINOR_ROOT;
}
fprintf(stderr, "Root device is (%d, %d)\n", major_root, minor_root);
if ((major_root != 2) && (major_root != 3) &&
    (major_root != 0)) {
        fprintf(stderr, "Illegal root device (major = %d)\n",
                major_root);
        die("Bad root device --- major #");
}
for (i=0;i<sizeof buf; i++) buf[i]=0;
if ((id=open(argv[1],O_RDONLY,0))<0)
        die("Unable to open 'boot'");
if (read(id,buf,MINIX_HEADER) != MINIX_HEADER)
        die("Unable to read header of 'boot'");
if (((long *) buf)[0]!=0x04100301)
        die("Non-Minix header of 'boot'");
if (((long *) buf)[1]!=MINIX_HEADER)
        die("Non-Minix header of 'boot'");
if (((long *) buf)[3]!=0)
        die("Illegal data segment in 'boot'");
if (((long *) buf)[4]!=0)
        die("Illegal bss in 'boot'");
if (((long *) buf)[5] != 0)
        die("Non-Minix header of 'boot'");
if (((long *) buf)[7] != 0)
        die("Illegal symbol table in 'boot'");
i=read(id,buf,sizeof buf);
fprintf(stderr,"Boot sector %d bytes.\n",i);
if (i != 512)
        die("Boot block must be exactly 512 bytes");
if ((*(unsigned short *)(buf+510)) != 0xAA55)
        die("Boot block hasn't got boot flag (0xAA55)");
buf[508] = (char) minor_root;
buf[509] = (char) major_root;
i=write(1,buf,512);
if (i!=512)
        die("Write call failed");
close (id);

if ((id=open(argv[2],O_RDONLY,0))<0)
        die("Unable to open 'setup'");
if (read(id,buf,MINIX_HEADER) != MINIX_HEADER)
        die("Unable to read header of 'setup'");
if (((long *) buf)[0]!=0x04100301)
        die("Non-Minix header of 'setup'");
if (((long *) buf)[1]!=MINIX_HEADER)
        die("Non-Minix header of 'setup'");
if (((long *) buf)[3]!=0)
        die("Illegal data segment in 'setup'");
if (((long *) buf)[4]!=0)
        die("Illegal bss in 'setup'");
if (((long *) buf)[5] != 0)
        die("Non-Minix header of 'setup'");
if (((long *) buf)[7] != 0)
        die("Illegal symbol table in 'setup'");
for (i=0 ; (c=read(id,buf,sizeof buf))>0 ; i+=c )
        if (write(1,buf,c)!=c)
                die("Write call failed");
close (id);
```

```
        if (i > SETUP_SECTS*512)
                die("Setup exceeds " STRINGIFY(SETUP_SECTS)
                        " sectors - rewrite build/boot/setup");
        fprintf(stderr,"Setup is %d bytes.\n",i);
        for (c=0 ; c<sizeof(buf) ; c++)
                buf[c] = '\0';
        while (i<SETUP_SECTS*512) {
                c = SETUP_SECTS*512-i;
                if (c > sizeof(buf))
                        c = sizeof(buf);
                if (write(1,buf,c) != c)
                        die("Write call failed");
                i += c;
        }

        if ((id=open(argv[3],O_RDONLY,0))<0)
                die("Unable to open 'system'");
        if (read(id,buf,GCC_HEADER) != GCC_HEADER)
                die("Unable to read header of 'system'");
        if (((long *) buf)[5] != 0)
                die("Non-GCC header of 'system'");
        for (i=0 ; (c=read(id,buf,sizeof buf))>0 ; i+=c )
                if (write(1,buf,c)!=c)
                        die("Write call failed");
        close(id);
        fprintf(stderr,"System is %d bytes.\n",i);
        if (i > SYS_SIZE*16)
                die("System is too big");
        return(0);
}
```