```
/*
 *  linux/boot/head.S
 *
 *  Copyright (C) 1991, 1992  Linus Torvalds
 */

/*
 *  head.S contains the 32-bit startup code.
 */

.text
.globl _idt,_gdt,
.globl _swapper_pg_dir,_pg0
.globl _empty_bad_page
.globl _empty_bad_page_table
.globl _empty_zero_page
.globl _tmp_floppy_area,_floppy_track_buffer

#include <linux/tasks.h>
#include <linux/segment.h>

#define CL_MAGIC_ADDR 0x90020
#define CL_MAGIC      0xA33F
#define CL_BASE_ADDR  0x90000
#define CL_OFFSET     0x90022

/*
 * swapper_pg_dir is the main page directory, address 0x00001000 (or at
 * address 0x00101000 for a compressed boot).
 */
startup_32:
        cld
        movl $(KERNEL_DS),%eax
        mov %ax,%ds
        mov %ax,%es
        mov %ax,%fs
        mov %ax,%gs
        lss _stack_start,%esp
/*
 * Clear BSS first so that there are no surprises...
 */
        xorl %eax,%eax
        movl $__edata,%edi
        movl $__end,%ecx
        subl %edi,%ecx
        cld
        rep
        stosb
/*
 * start system 32-bit setup. We need to re-do some of the things done
 * in 16-bit mode for the "real" operations.
 */
        call setup_idt
        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
/*
 * Initialize eflags.  Some BIOS's leave bits like NT set.  This would
 * confuse the debugger if this code is traced.
 * XXX - best to initialize before switching to protected mode.
 */
        pushl $0
        popfl
/*
 * Copy bootup parameters out of the way. First 2kB of
 * _empty_zero_page is for boot parameters, second 2kB
 * is for the command line.
 */
        movl $0x90000,%esi
        movl $_empty_zero_page,%edi
        movl $512,%ecx
        cld
        rep
        movsl
        xorl %eax,%eax
        movl $512,%ecx
        rep
        stosl
```

```
	cmpw $(CL_MAGIC),CL_MAGIC_ADDR
	jne 1f
	movl $_empty_zero_page+2048,%edi
	movzwl CL_OFFSET,%esi
	addl $(CL_BASE_ADDR),%esi
	movl $2048,%ecx
	rep
	movsb
1:
/* check if it is 486 or 386. */
/*
 * XXX - this does a lot of unnecessary setup.  Alignment checks don't
 * apply at our cpl of 0 and the stack ought to be aligned already, and
 * we don't need to preserve eflags.
 */
	movl %esp,%edi		# save stack pointer
	andl $0xfffffffc,%esp	# align stack to avoid AC fault
	movl $3,_x86
	pushfl			# push EFLAGS
	popl %eax		# get EFLAGS
	movl %eax,%ecx		# save original EFLAGS
	xorl $0x40000,%eax	# flip AC bit in EFLAGS
	pushl %eax		# copy to EFLAGS
	popfl			# set EFLAGS
	pushfl			# get new EFLAGS
	popl %eax		# put it in eax
	xorl %ecx,%eax		# change in flags
	andl $0x40000,%eax	# check if AC bit changed
	je is386
	movl $4,_x86
	movl %ecx,%eax
	xorl $0x200000,%eax	# check ID flag
	pushl %eax
	popfl			# if we are on a straight 486DX, SX, or
	pushfl			# 487SX we can't change it
	popl %eax
	xorl %ecx,%eax
	andl $0x200000,%eax
	je is486
isnew:	pushl %ecx		# restore original EFLAGS
	popfl
	movl $1, %eax		# Use the CPUID instruction to
	.byte 0x0f, 0xa2	# check the processor type
	andl $0xf00, %eax	# Set _x86 with the family
	shrl $8, %eax		# returned.
	movl %eax, _x86
	movl %edi,%esp		# restore esp
	movl %cr0,%eax		# 486+
	andl $0x80000011,%eax	# Save PG,PE,ET
	orl $0x50022,%eax	# set AM, WP, NE and MP
	jmp 2f
is486:	pushl %ecx		# restore original EFLAGS
	popfl
	movl %edi,%esp		# restore esp
	movl %cr0,%eax		# 486
	andl $0x80000011,%eax	# Save PG,PE,ET
	orl $0x50022,%eax	# set AM, WP, NE and MP
	jmp 2f
is386:	pushl %ecx		# restore original EFLAGS
	popfl
	movl %edi,%esp		# restore esp
	movl %cr0,%eax		# 386
	andl $0x80000011,%eax	# Save PG,PE,ET
	orl $2,%eax		# set MP
2:	movl %eax,%cr0
	call check_x87
	call setup_paging
	lgdt gdt_descr
	lidt idt_descr
	ljmp $(KERNEL_CS),$1f
1:	movl $(KERNEL_DS),%eax# reload all the segment registers
	mov %ax,%ds		# after changing gdt.
	mov %ax,%es
	mov %ax,%fs
	mov %ax,%gs
	lss _stack_start,%esp
	xorl %eax,%eax
	lldt %ax
	pushl %eax		# These are the parameters to main :-)
	pushl %eax
```

```
        pushl %eax
        cld                     # gcc2 wants the direction flag cleared at all times
        call _start_kernel
L6:
        jmp L6                  # main should never return here, but
                                # just in case, we know what happens.

/*
 * We depend on ET to be correct. This checks for 287/387.
 */
check_x87:
        movl $0,_hard_math
        clts
        fninit
        fstsw %ax
        cmpb $0,%al
        je 1f
        movl %cr0,%eax          /* no coprocessor: have to set bits */
        xorl $4,%eax            /* set EM */
        movl %eax,%cr0
        ret
.align 2
1:      movl $1,_hard_math
        .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 doesn't actually load
 *  idt - that can be done only after paging has been enabled
 *  and the kernel moved to 0xC0000000. Interrupts
 *  are enabled elsewhere, when we can be relatively
 *  sure everything is ok.
 */
setup_idt:
        lea ignore_int,%edx
        movl $(KERNEL_CS << 16),%eax
        movw %dx,%ax            /* selector = 0x0010 = 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
        ret


/*
 * Setup_paging
 *
 * This routine sets up paging by setting the page bit
 * in cr0. The page tables are set up, identity-mapping
 * the first 4MB.  The rest are initialized later.
 *
 * (ref: added support for up to 32mb, 17Apr92)  -- Rik Faith
 * (ref: update, 25Sept92)  -- croutons@crunchy.uucp
 * (ref: 92.10.11 - Linus Torvalds. Corrected 16M limit - no upper memory limit)
 */
.align 2
setup_paging:
        movl $1024*2,%ecx               /* 2 pages - swapper_pg_dir+1 page table */
        xorl %eax,%eax
        movl $_swapper_pg_dir,%edi      /* swapper_pg_dir is at 0x1000 */
        cld;rep;stosl
/* Identity-map the kernel in low 4MB memory for ease of transition */
        movl $_pg0+7,_swapper_pg_dir            /* set present bit/user r/w */
/* But the real place is at 0xC0000000 */
        movl $_pg0+7,_swapper_pg_dir+3072       /* set present bit/user r/w */
        movl $_pg0+4092,%edi
        movl $0x03ff007,%eax            /*  4Mb - 4096 + 7 (r/w user,p) */
        std
1:      stosl                   /* fill the page backwards - more efficient :-) */
        subl $0x1000,%eax
        jge 1b
```

```
        cld
        movl $_swapper_pg_dir,%eax
        movl %eax,%cr3               /* cr3 - page directory start */
        movl %cr0,%eax
        orl $0x80000000,%eax
        movl %eax,%cr0        /* set paging (PG) bit */
        ret                   /* this also flushes the prefetch-queue */

/*
 * page 0 is made non-existent, so that kernel NULL pointer references get
 * caught. Thus the swapper page directory has been moved to 0x1000
 *
 * XXX Actually, the swapper page directory is at 0x1000 plus 1 megabyte,
 * with the introduction of the compressed boot code.  Theoretically,
 * the original design of overlaying the startup code with the swapper
 * page directory is still possible --- it would reduce the size of the kernel
 * by 2-3k.  This would be a good thing to do at some point.....
 */
.org 0x1000
_swapper_pg_dir:
/*
 * The page tables are initialized to only 4MB here - the final page
 * tables are set up later depending on memory size.
 */
.org 0x2000
_pg0:

.org 0x3000
_empty_bad_page:

.org 0x4000
_empty_bad_page_table:

.org 0x5000
_empty_zero_page:

.org 0x6000
/*
 * 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
/*
 * floppy_track_buffer is used to buffer one track of floppy data: it
 * has to be separate from the tmp_floppy area, as otherwise a single-
 * sector read/write can mess it up. It can contain one full track of
 * data (18*2*512 bytes).
 */
_floppy_track_buffer:
        .fill 512*2*18,1,0

/* This is the default interrupt "handler" :-) */
int_msg:
        .asciz "Unknown interrupt\n"
.align 2
ignore_int:
        cld
        pushl %eax
        pushl %ecx
        pushl %edx
        push %ds
        push %es
        push %fs
        movl $(KERNEL_DS),%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
```

```
/*
 * The interrupt descriptor table has room for 256 idt's
 */
.align 4
.word 0
idt_descr:
        .word 256*8-1         # idt contains 256 entries
        .long 0xc0000000+_idt

.align 4
_idt:
        .fill 256,8,0         # idt is uninitialized

.align 4
.word 0
gdt_descr:
        .word (8+2*NR_TASKS)*8-1
        .long 0xc0000000+_gdt

/*
 * This gdt setup gives the kernel a 1GB address space at virtual
 * address 0xC0000000 - space enough for expansion, I hope.
 */
.align 4
_gdt:
        .quad 0x0000000000000000      /* NULL descriptor */
        .quad 0x0000000000000000      /* not used */
        .quad 0xc0c39a000000ffff      /* 0x10 kernel 1GB code at 0xC0000000 */
        .quad 0xc0c392000000ffff      /* 0x18 kernel 1GB data at 0xC0000000 */
        .quad 0x00cbfa000000ffff      /* 0x23 user   3GB code at 0x00000000 */
        .quad 0x00cbf2000000ffff      /* 0x2b user   3GB data at 0x00000000 */
        .quad 0x0000000000000000      /* not used */
        .quad 0x0000000000000000      /* not used */
        .fill 2*NR_TASKS,8,0          /* space for LDT's and TSS's etc */
```