

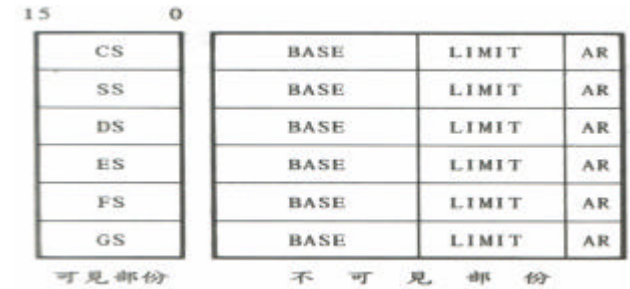
System Initialization

Jassie Tsai 2002 NCTU



Outline

- ✍ Prehistoric Age: The BIOS
- ✍ Ancient Age: The Boot Loader
- ✍ Middle Ages: The `setup()` Function
- ✍ Renaissance: The `startup_32()` Functions
- ✍ Industrial Revolution: The `start_kernel()` Function
- ✍ Modern Age: The first process – `init()` Kernel Thread
- ✍ Related Resources



Prehistoric Age: The BIOS

- ✍ After System power is up, some registers of the processor are set to fixed values, and the code found at physical address **0xffffffff**(*) is executed. In general x86 machine, this address is mapped by the BIOS.
- ✍ BIOS runs under Real Mode and performs the following four operations:
 - Make Power-On Self-Test
 - Initializes the hardware devices such as PCI devices, video cards...
 - Searches for an operating system to boot.
 - As soon as a valid devices is found, copies the contents of its first sector into RAM, starting from physical address **0x07c00**, then jumps into that address and executes the code just loaded.

Ancient Age: The Boot Loader

- ✍ The boot sector used to boot Linux kernel could be either:
 - Linux boot-sector(arch/i386/boot/bootsect.S)
 - LILO or other boot-loader
- ✍ Linux boot-sector is always placed at the first 512 bytes of the kernel image file(the size of sector). Thus, it is very easy to produce a bootable floppy containing the Linux kernel.
- ✍ The floppy can be created by copying the kernel image starting from the first sector of the disk. When the BIOS loads the first sector of the floppy disk, it actually copies the code of the boot loader.

Ancient Age: The Boot Loader

- ✍ The Linux boot-sector performs the following operations:
- Move itself from **0x07c00** to **0x90000** (9th 64KB)
 - Sets up the Real Mode stack, from address 9000:3FF4. The stack will grow toward lower addresses.
 - Set up the disk parameter table, used by the BIOS to handle the floppy device driver.
 - Invokes a BIOS procedure to display a “Loading” message.
 - Invokes a BIOS procedure to load the **setup()** code of kernel image from the floppy disk and puts it in RAM starting from address **0x90200** (9th 64KB + **512B**).
 - Invokes a BIOS procedure to load the rest of the kernel images from the floppy disk and puts the image in RAM starting from either **0x10000** (1th 64KB) or **0x100000** (2th MB).
 - Jumps to the **setup()** code.

setup() is always placed in kernel image just after boot.S.

Load High

Load Low

Ancient Age: The Boot Loader

```
#define DEF_INITSEG 0X9000
#define DEF_SYSSEG 0X1000
#define DEF_SETUPSEG 0x9020
#define DEF_SYSSIZE 0x7f00
```

arch/i386/boot/bootsect.S

SETUPSECTS = 4

BOOTSEG = 0x07C0

INITSEG = DEF_INITSEG

SETUPSEG = DEF_SETUPSEG

SYSSEG = DEF_SYSSEG

SYSSIZE = DEF_SYSSIZE

...

```
movw $BOOTSEG, %ax
```

```
movw %ax, %ds
```

```
movw $INITSEG, %ax
```

```
movw %ax, %es
```

```
movw $256, %cx
```

```
subw %si, %si
```

```
subw %di, %di
```

```
cld
```

```
rep movsw
```

```
ljmp $INITSEG, $go
```

go:

```
movw $0x4000-12, %di
```

```
movw %ax, %ds
```

```
movw %ax, %ss
```

```
movw %di, %sp
```

/ default nr of setup-sectors */*

/ original address of boot-sector */*

/ we move boot here - out of the way */*

/ setup starts here */*

/ system loaded at 0x10000 (65536) */*

/ system size: # of 16-byte clicks */*

%ds = BOOTSEG

%ax = %es = INITSEG

%ax and %es already contain INITSEG

put stack at INITSEG:0x4000-12.

Ancient Age: The Boot Loader

```
#define DEF_INITSEG 0X9000
#define DEF_SYSSEG 0X1000
#define DEF_SETUPSEG 0x9020
#define DEF_SYSSIZE 0x7f00
```

arch/i386/boot/bootsect.S

```
SETUPSECTS = 4
```

```
BOOTSEG = 0x07C0
```

```
INITSEG = DEF_INITSEG
```

```
SETUPSEG = DEF_SETUPSEG
```

```
SYSSEG = DEF_SYSSEG
```

```
SYSSIZE = DEF_SYSSIZE
```

```
...
```

```
movw $BOOTSEG, %ax
```

```
movw %ax, %ds
```

```
movw $INITSEG, %ax
```

```
movw %ax, %es
```

```
movw $256, %cx
```

```
subw %si, %si
```

```
subw %di, %di
```

```
cld
```

```
rep movsw
```

```
ljmp $INITSEG, $go
```

```
go:
```

```
movw $0x4000-12, %di
```

```
movw %ax, %ds
```

```
movw %ax, %ss
```

```
movw %di, %sp
```

```
/* default nr of setup-sectors */
```

```
/* original address of boot-sector */
```

```
/* we move boot here - out of the way */
```

```
/* setup starts here */
```

```
/* system loaded at 0x10000 (65536) */
```

```
/* system size: # of 16-byte clicks */
```

```
# %ds = BOOTSEG
```

```
# %ax = %es = INITSEG
```

Move the boot-sector code from address 0x07c00 to 0x90000.

```
DS:SI -> 07C0 : 0000
```

```
ES:DI -> 0900 : 0000
```

```
# %ax and %es already contain INITSEG
```

```
# put stack at INITSEG:0x4000-12.
```

Ancient Age: The Boot Loader

```
#define DEF_INITSEG 0X9000
#define DEF_SYSSEG 0X1000
#define DEF_SETUPSEG 0x9020
#define DEF_SYSSIZE 0x7f00
```

arch/i386/boot/bootsect.S

```
SETUPSECTS = 4
BOOTSEG = 0x07C0
INITSEG = DEF_INITSEG
SETUPSEG = DEF_SETUPSEG
SYSSEG = DEF_SYSSEG
SYSSIZE = DEF_SYSSIZE
...
movw    $BOOTSEG, %ax
movw    %ax, %ds          # %ds = BOOTSEG
movw    $INITSEG, %ax
movw    %ax, %es          # %ax = %es = INITSEG
movw    $256, %cx
subw    %si, %si
subw    %di, %di
cld
rep movsw
ljmp   $INITSEG, $go
go:
movw    $0x4000-12, %di
movw    %ax, %ds          # %ax and %es already contain INITSEG
movw    %ax, %ss
movw    %di, %sp          # put stack at INITSEG:0x4000-12.
```

Jump to the new "go".

Ancient Age: The Boot Loader

```
#define DEF_INITSEG 0X9000
#define DEF_SYSSEG 0X1000
#define DEF_SETUPSEG 0x9020
#define DEF_SYSSIZE 0x7f00
```

arch/i386/boot/bootsect.S

```
SETUPSECTS = 4
BOOTSEG = 0x07C0
INITSEG = DEF_INITSEG
SETUPSEG = DEF_SETUPSEG
SYSSEG = DEF_SYSSEG
SYSSIZE = DEF_SYSSIZE

...
/* default nr of setup-sectors */
/* original address of boot-sector */
/* we move boot here - out of the way */
/* setup starts here */
/* system loaded at 0x10000 (65536) */
/* system size: # of 16-byte clicks */
```

```
...
movw $BOOTSEG, %ax
movw %ax, %ds          # %ds = BOOTSEG
movw $INITSEG, %ax
movw %ax, %es         # %ax = %es = INITSEG
movw $256, %cx
subw %si, %si
subw %di, %di
cld
rep movsw
ljmp $INITSEG, $go
```

Prepare a stack to use. 0x4000 is an arbitrary value \geq length of boot sector + length of setup + room for stack. 12 is disk parm size.

```
go:
movw $0x4000-12, %di
movw %ax, %ds          # %ax and %es already contain INITSEG
movw %ax, %ss
movw %di, %sp         # put stack at INITSEG:0x4000-12.
```

After doing this, SS:SP will be 9000:3FF4. (*)

Ancient Age: The Boot Loader

Patch the disk parameter table for the first disk to allow multi-sector reads

```
...
...
load_setup:
    xorb    %ah, %ah          # reset FDC
    xorb    %dl, %dl
    int     $0x13

    xorw    %dx, %dx          # drive 0, head 0
    movb    $0x02, %cl        # sector 2, track 0
    movw    $0x0200, %bx      # address = 512, in INITSEG
    movb    $0x02, %ah        # service 2, "read sectors"
    movb    setup_sects, %al  # (assume all on head 0, track 0)
    int     $0x13            # read it
    jnc     ok_load_setup     # ok -continue

    pushw   %ax
    call    print_nl
    movw    $sp, %bp
    call    print_hex
    popw    %ax
    jmp     load_setup

ok_load_setup:
    ...
    ljmp    $SETUPSEG, $0
```

Ancient Age: The Boot Loader

```
...
...
...
Load setup() into 0x90200
load_setup:
    xorb    %ah, %ah          # reset FDC
    xorb    %dl, %dl
    int     $0x13

    xorw    %dx, %dx          # drive 0, head 0
    movb    $0x02, %cl        # sector 2, track 0
    movw    $0x0200, %bx      # address = 512, in INITSEG
    movb    $0x02, %ah        # service 2, "read sectors"
    movb    setup_sects, %al  # (assume all on head 0, track 0)
    int     $0x13            # read it
    jnc     ok_load_setup     # ok -continue

    pushw   %ax
    call    print_nl
    movw    $sp, %bp
    call    print_hex
    popw    %ax
    jmp     load_setup

ok_load_setup:
    ...
    ljmp    $SETUPSEG, $0
```

Ancient Age: The Boot Loader

```
...
...
load_setup:
    xorb    %ah, %ah          # reset FDC
    xorb    %dl, %dl
    int     $0x13

    xorw    %dx, %dx          # drive 0, head 0
    movb    $0x02, %cl        # sector 2, track 0
    movw    $0x0200, %bx      # address = 512, in INITSEG
    movb    $0x02, %ah        # service 2, "read sectors"
    movb    setup_sects, %al  # (assume all on head 0, track 0)
    int     $0x13             # read it
    jnc     ok_load_setup     # ok -continue

    pushw   %ax
    call    print_nl
    movw    $sp, %bp
    call    print_hex
    popw    %ax
    jmp     load_setup

ok_load_setup:
    ...
    ljmp    $SETUPSEG, $0
```

INT 0x13, AH = 0, reset FDC
DL <- Disk id (0 ~ 3)
This BIOS call make floppy stable.

Ancient Age: The Boot Loader

```
...
...
load_setup:
    xorb    %ah, %ah          # reset FDC
    xorb    %dl, %dl
    int     $0x13

    xorw    %dx, %dx          # drive 0, head 0
    movb    $0x02, %cl        # sector 2, track 0
    movw    $0x0200, %bx      # address = 512, in INITSEG
    movb    $0x02, %ah        # service 2, "read sectors"
    movb    setup_sects, %al  # (assume all on head 0, track 0)
    int     $0x13            # read it
    jnc     ok_load_setup     # ok -continue

    pushw   %ax
    call    print_nl
    movw    $sp, %bp
    call    print_hex
    popw    %ax
    jmp     load_setup

ok_load_setup:
    ...
    ljmp    $SETUPSEG, $0
```

INT 0x13, AH = 2, read specified sector

DL <- Disk id (0 ~ 3)

DH <- Head id (0 ~ 1)

CH <- Track number (0 ~ 39)

CL <- Beginning sector number (1 ~ 9)

AL <- Sectors to read

ES:BX <- Buffer

If succeed then CF <- 0

Else CF <- 1

Ancient Age: The Boot Loader

```
...
...
load_setup:
    xorb    %ah, %ah          # reset FDC
    xorb    %dl, %dl
    int     $0x13

    xorw    %dx, %dx          # drive 0, head 0
    movb    $0x02, %cl        # sector 2, track 0
    movw    $0x0200, %bx      # address = 512, in INITSEG
    movb    $0x02, %ah        # service 2, "read sectors"
    movb    setup_sects, %al  # (assume all on head 0, track 0)
    int     $0x13            # read it
    jnc     ok_load_setup     # ok -continue

    pushw   %ax
    call    print_nl
    movw    $sp, %bp
    call    print_hex
    popw    %ax
    jmp     load_setup

ok_load_setup:
    ...
    ljmp    $SETUPSEG, $0
```

If loading failed for some reason, we dump error code and retry in an endless loop.

Ancient Age: The Boot Loader

```
...
...
load_setup:
    xorb    %ah, %ah           # reset FDC
    xorb    %dl, %dl
    int     $0x13

    xorw    %dx, %dx           # drive 0, head 0
    movb    $0x02, %cl         # sector 2, track 0
    movw    $0x0200, %bx       # address = 512, in INITSEG
    movb    $0x02, %ah         # service 2, "read sectors"
    movb    setup_sects, %al   # (assume all on head 0, track 0)
    int     $0x13             # read it
    jnc     ok_load_setup     # ok -continue

    pushw   %ax
    call    print_nl
    movw    $sp, %bp
    call    print_hex
    popw    %ax
    jmp     load_setup

ok_load_setup:
    ...
    ljmp    $SETUPSEG, $0
```

Load compressed kernel
image into 0x10000

Ancient Age: The Boot Loader

```
...
...
load_setup:
    xorb    %ah, %ah          # reset FDC
    xorb    %dl, %dl
    int     $0x13

    xorw    %dx, %dx          # drive 0, head 0
    movb    $0x02, %cl        # sector 2, track 0
    movw    $0x0200, %bx      # address = 512, in INITSEG
    movb    $0x02, %ah        # service 2, "read sectors"
    movb    setup_sects, %al  # (assume all on head 0, track 0)
    int     $0x13            # read it
    jnc     ok_load_setup     # ok -continue

    pushw   %ax
    call    print_nl
    movw    $sp, %bp
    call    print_hex
    popw    %ax
    jmp     load_setup

ok_load_setup:
    ...
    ljmp    $SETUPSEG, $0     Jump to setup()
```

Ancient Age: The Boot Loader

- ✍ Using LILO as boot loader is another choice. There are several advantages in using a specialized boot loader over a bare bones Linux boot-sector:
 - Ability to choose between multiple Linux kernels or even multiple OSes.
 - Ability to pass kernel command line parameters
 - Ability to load much larger kernels (**Load High**)
- ✍ LILO performs essentially the same operations as the boot loader integrated into the kernel image. It copies the integrated boot loader of the kernel image to address **0x90000**, the `setup()` code to address **0x90200**, and the rest of the kernel image to address **0x10000** or **0x100000**. Then it jumps to the `setup()` code.

Middle Ages: The `setup()` Function

- ✍ Although the BIOS already initialized most hardware devices, Linux does not rely on it but reinitializes the devices in its own manner to enhance portability and robustness.
- ✍ The `setup()` function initializes the hardware devices in the computer and setup the environment for the execution of the kernel program.
- ✍ `setup()` essentially performs the following operations:
 - Try to find out the amount of RAM available in the system
 - Sets the keyboard repeat delay and rate.
 - Initializes the video adapter card.

Middle Ages: The setup() Function

- Checks for an IBM Micro Channel bus(MCA)
- Checks for a PS/2 pointing device
- Checks for Advanced Power Management (APM) BIOS support.
- If the kernel image was loaded low in RAM (0x10000), move it to physical address 0x01000 (1024B, Just after Interrupt Vector Table). Conversely, if the kernel image was loaded high in RAM, does nothing.
- Sets up a provisional Interrupt Descriptor Table (IDT) and a provisional Global Descriptor Table (GDT)
- Reset the floating point unit (FPU), if any.

Middle Ages: The setup() Function

- Reprograms the Programmable Interrupt Controller (PIC) and maps the 16 hardware interrupts (IRQ lines) to the range of vectors from 32 to 47.
- Switch the CPU from Real Mode to Protected Mode by setting the PE bit in the cr0 status register.
- Jumps to the startup_32() assembly language function.

Middle Ages: The setup() Function

arch/i386/boot/setup.S

...

start:

jmp trampoline

...

trampoline:

...

Get memory size (extended mem, kB)

xorl %eax, %eax

movl %eax, (0x1e0)

#ifndef STANDARD_MEMORY_BIOS_CALL

...

#endif

movb \$0x88, %ah

int \$0x15

movw %ax, (2)

...

Set the keyboard repeat rate to the max

movw \$0x0305, %ax

xorw %bx, %bx

int \$0x16

...

The entrance of setup()

Middle Ages: The setup() Function

arch/i386/boot/setup.S

```
...
start:
    jmp     trampoline
...
trampoline:
...
# Get memory size (extended mem, kB)
    xorl   %eax, %eax
    movl   %eax, (0x1e0)
#ifdef STANDARD_MEMORY_BIOS_CALL
...
#endif
    movb   $0x88, %ah
    int    $0x15
    movw   %ax, (2)
...
# Set the keyboard repeat rate to the max
    movw   $0x0305, %ax
    xorw   %bx, %bx
    int    $0x16
...
```

Middle Ages: The setup() Function

arch/i386/boot/setup.S

```
...
start:
    jmp    trampoline
...
trampoline:
...
# Get memory size (extended mem, kB)
    xorl   %eax, %eax
    movl   %eax, (0x1e0)
#ifndef STANDARD_MEMORY_BIOS_CALL
...
#endif
    movb   $0x88, %ah
    int    $0x15
    movw   %ax, (2)
...
# Set the keyboard repeat rate to the max
    movw   $0x0305, %ax
    xorw   %bx, %bx
    int    $0x16
...
```

Middle Ages: The setup() Function

```
idt_48:
.word    0          # idt limit = 0
.word    0, 0       # idt base = 0L
```

set up gdt and idt

```
    lidt    idt_48
    xorl    %eax, %eax
    movw    %ds, %ax
    shll    $4, %eax
    addl    $gdt, %eax
    movl    %eax, (gdt_48+2)
    lgdt    gdt_48
    ...
    movw    $1, %ax
    lmsw    %ax
    jmp     flush_instr

flush_instr:
    xorw    %bx, %bx
    xorl    %esi, %esi
    movw    %cs, %si
    subw    $DELTA_INITSEG, %si
    shll    $4, %esi
    .byte   0x66, 0xea

code32:
    .long   0x1000
    .word   __KERNEL_CS
```

load idt with 0,0
Compute gdt_base
(Convert %ds:gdt to a linear ptr)

load gdt with whatever is
protected mode (PE) bit
This is it!

Flag to indicate a boot
Pointer to real-mode code

Convert to 32-bit pointer
prefix + jmp-i-opcode

will be set to 0x100000

Middle Age

on

set up gdt and idt

```
    lidt    idt_48
    xorl    %eax, %eax
    movw   %ds, %ax
    shll   $4, %eax
    addl   $gdt, %eax
    movl   %eax, (gdt_48+2)
    lgdt   gdt_48
    ...
    movw   $1, %ax
    lmsw   %ax
    jmp    flush_instr
flush_instr:
    xorw   %bx, %bx
    xorl   %esi, %esi
    movw   %cs, %si
    subw   $DELTA_INITSEG, %si
    shll   $4, %esi
    .byte  0x66, 0xea
code32:
    .long  0x1000
    .word  __KERNEL_CS
```

```
gdt:
.word    0, 0, 0, 0    # dummy
.word    0, 0, 0, 0    # unused

.word    0xFFFF       # 4Gb - (0x100000*0x1000 = 4Gb)
.word    0             # base address = 0
.word    0x9A00        # code read/exec
.word    0x00CF        # granularity = 4096, 386
                    # (+5th nibble of limit)
.word    0xFFFF       # 4Gb - (0x100000*0x1000 = 4Gb)
.word    0             # base address = 0
.word    0x9200        # data read/write
.word    0x00CF        # granularity = 4096, 386
                    # (+5th nibble of limit)
```

load idt with 0,0

Compute gdt_base

(Convert %ds:gdt to a linear ptr)

load gdt with whatever is

protected mode (PE) bit

This is it!

```
gdt_48:
.word    0x8000        # gdt limit=2048,
                    # 256 GDT entries
.word    0, 0         # gdt base (filled in later)
```


Convert to 32-bit pointer

prefix + jmp-i-opcode

will be set to 0x100000

Middle Ages: The setup() Function

```
# set up gdt and idt
    lidt    idt_48                                # load idt with 0,0
    xorl    %eax, %eax                            # Compute gdt_base
    movw    %ds, %ax                              # (Convert %ds:gdt to a linear ptr)
    shll    $4, %eax
    addl    $gdt, %eax
    movl    %eax, (gdt_48+2)
    lgdt    gdt_48                                # load gdt with whatever is
    ...
    movw    $1, %ax                               # protected mode (PE) bit
    lmsw    %ax                                   # This is it!
    jmp     flush_instr
flush_instr:
    xorw    %bx, %bx                              # Flag to indicate a boot
    xorl    %esi, %esi                             # Pointer to real-mode code
    movw    %cs, %si
    subw    $DELTA_INITSEG, %si
    shll    $4, %esi                              # Convert to 32-bit pointer
    .byte   0x66, 0xea                            # prefix + jmp-i-opcode
code32:
    .long   0x1000                                # will be set to 0x100000
    .word   __KERNEL_CS
```



Middle Ages: The setup() Function

```
# set up gdt and idt
    lidt    idt_48
    xorl    %eax, %eax
    movw   %ds, %ax
    shll   $4, %eax
    addl   $gdt, %eax
    movl   %eax, (gdt_48+2)
    lgdt   gdt_48
    ...
    movw   $1, %ax
    lmsw   %ax
    jmp    flush_instr
flush_instr:
    xorw   %bx, %bx
    xorl   %esi, %esi
    movw   %cs, %si
    subw   $DELTA_INITSEG, %si
    shll   $4, %esi
    .byte  0x66, 0xea
code32:
    .long  0x1000
    .word  __KERNEL_CS
```

load idt with 0,0
Compute gdt_base
(Convert %ds:gdt to a linear ptr)

load gdt with whatever is
protected mode (PE) bit
This is it!

Flag to indicate a boot
Pointer to real-mode code

Convert to 32-bit pointer
prefix + jmp-i-opcode

will be set to 0x100000

Not necessary.

Middle Ages: The setup() Function

```
# set up gdt and idt
    lidt    idt_48                # load idt with 0,0
    xorl   %eax, %eax            # Compute gdt_base
    movw   %ds, %ax             # (Convert %ds:gdt to a linear ptr)
    shll   $4, %eax
    addl   $gdt, %eax
    movl   %eax, (gdt_48+2)
    lgdt   gdt_48                # load gdt with whatever is
    ...
    movw   $1, %ax              # protected mode (PE) bit
    lmsw   %ax                  # This is it!
    jmp    flush_instr

flush_instr:
    xorw   %bx, %bx             # Flag to indicate a boot
    xorl   %esi, %esi           # Flag to indicate a boot
    movw   %cs, %si
    subw   $DELTA_INITSEG, %esi
    shll   $4, %esi             # Convert to 32-bit pointer
    .byte  0x66, 0xea          # prefix + jmpil-opcode
code32:
    .long  0x1000              # will be set to 0x100000
    .word  __KERNEL_CS
```

Jump to 0x100000 and set CS to be KERNEL_CS.
Prefix is needed because ...

de

Renaissance: The startup_32 Functions

- ✍ There are two different startup_32() functions; The one we refer to here is coded in the arch/i386/boot/compressed/head.S. After setup() terminates, the function has been moved either to physical address 0x100000 or to physical address 0x1000, depending on whether the kernel image was loaded high or low in RAM.
- ✍ The function performs the following operations:
 - Initializes the segmentation registers and a provisional stack.
 - Fills the area of un-initialized data of the kernel identified by the _edata and _end symbols with zeros.
 - Invokes the decompress_kernel() to decompress the kernel image. If the kernel image was loaded low, the decompressed kernel is placed at physical address 0x100000. Otherwise, if the kernel image was loaded high, the decompressed kernel is placed in a temporary buffer and is then moved into its final position, which starts at physical address 0x100000.
 - Jumps to physical address 0x100000 (The second startup_32).

Renaissance: The startup_32 Functions

arch/i386/boot/compressed/head.S

startup_32:

`cld`

`cli`

`movl $(__KERNEL_DS), %eax`

`movl %eax, %ds`

`movl %eax, %es`

`movl %eax, %fs`

`movl %eax, %gs`

Initialize various segment register.

`lss SYMBOL_NAME(stack_start), %esp`

`...`

`/*`

`* 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`

Renaissance: The startup_32 Functions

arch/i386/boot/compressed/head.S

startup_32:

```
cld
cli
movl $(__KERNEL_DS), %eax
movl %eax, %ds
movl %eax, %es
movl %eax, %fs
movl %eax, %gs
```

Initialize a provisional stack.

```
lss SYMBOL_NAME(stack_start), %esp
```

```
...
/*
```

```
* Initialize eflags. Some BIOS's leave bits like NT set. This
* confuse the debugger if this code is traced.
* XXX - best to initialize before switching to protected mode.
*/
```

```
pushl $0
popfl
```

```
#define STACK_SIZE (4096)
```

```
long user_stack [STACK_SIZE];
```

```
struct {
```

```
    long * a;
```

```
    short b;
```

```
} stack_start = { & user_stack [STACK_SIZE] , __KERNEL_DS };
```

Renaissance: The startup_32 Functions

arch/i386/boot/compressed/head.S

startup_32:

 cld

 cli

 movl \$(__KERNEL_DS), %eax

 movl %eax, %ds

 movl %eax, %es

 movl %eax, %fs

 movl %eax, %gs

 lss SYMBOL_NAME(stack_start), %esp

 ...

 /*

 * 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

Renaissance: The startup_32 Functions

```
/*
 * Clear BSS
 */
    xorl %eax,%eax
    movl $ SYMBOL_NAME(_edata),%edi
    movl $ SYMBOL_NAME(_end),%ecx
    subl %edi,%ecx
    cld
    rep
    stosb

/*
 * Do the decompression, and jump to the new kernel..
 */
    subl $16,%esp          # place for structure on the stack
    movl %esp,%eax
    pushl %esi             # real mode pointer as second arg
    pushl %eax             # address of structure as first arg
    call SYMBOL_NAME(decompress_kernel)
    orl %eax,%eax
    jnz 3f
    popl %esi              # discard address
    popl %esi              # real mode pointer
    xorl %ebx,%ebx
    ljmp $(__KERNEL_CS), $0x100000
```

Renaissance: The startup_32 Functions

```
/*
 * Clear BSS
 */
    xorl %eax,%eax
    movl $ SYMBOL_NAME(_edata),%edi
    movl $ SYMBOL_NAME(_end),%ecx
    subl %edi,%ecx
    cld
    rep
    stosb

/*
 * Do the decompression, and jump to the new kernel..
 */
    subl $16,%esp          # place for structure on the stack
    movl %esp,%eax
    pushl %esi             # real mode pointer as second arg
    pushl %eax             # address of structure as first arg
    call SYMBOL_NAME(decompress_kernel)
    orl %eax,%eax
    jnz 3f
    popl %esi              # discard address
    popl %esi              # real mode pointer
    xorl %ebx,%ebx
    ljmp $(__KERNEL_CS), $0x100000
```

Renaissance: The startup_32 Functions

```
/*
 * Clear BSS
 */
    xorl %eax,%eax
    movl $ SYMBOL_NAME(_edata),%edi
    movl $ SYMBOL_NAME(_end),%ecx
    subl %edi,%ecx
    cld
    rep
    stosb

/*
 * Do the decompression, and jump to the new kernel..
 */
    subl $16,%esp          # place for structure on the stack
    movl %esp,%eax
    pushl %esi             # real mode pointer as second arg
    pushl %eax             # address of structure as first arg
    call SYMBOL_NAME(decompress_kernel)
    orl %eax,%eax
    jnz 3f
    popl %esi              # discard address
    popl %esi              # real mode pointer
    xorl %ebx,%ebx
    ljmp $(__KERNEL_CS), $0x100000
```

Renaissance: The startup_32 Functions

- ✍ The second `startup_32()` function is coded in the `arch/i386/boot/head.S`. It essentially sets up the execution environment for the first Linux process (process 0). The function performs the following operations:
 - Initialize segment values.
 - Initialize a provisional page tables.
 - Enable paging by setting PG bit in `%cr0`.
 - Zero-clean BSS.
 - Copy the first 2k of boot-up parameters (kernel command-line).
 - Check CPU type.
 - The first CPU calls `start_kernel()`, all others call `initialize_secondary()`.

Renaissance: The startup_32 Functions

[arch/i386/boot/head.S](#)

startup_32:

*/**

** Set segments to known values*

**/*

cld

movl **\$(__KERNEL_DS), %eax**

movl **%eax, %ds**

movl **%eax, %es**

movl **%eax, %fs**

movl **%eax, %gs**

...

*/**

** Initialize page tables*

**/*

movl **\$pg0-__PAGE_OFFSET, %edi** */* initialize page tables */*

movl **\$007, %eax** */* "007" doesn't mean with right to kill, but
PRESENT+RW+USER */*

2: **stosl**

add **\$0x1000, %eax**

cmp **\$empty_zero_page-__PAGE_OFFSET, %edi**

jne **2b**

Renaissance: The startup_32

arch/i386/boot/head.S

startup_32:

```

/*
 * Set segments to known values
 */
    cld
    movl    $(__KERNEL_DS),%eax
    movl    %eax,%ds
    movl    %eax,%es
    movl    %eax,%fs
    movl    %eax,%gs
    ...

```

/*

*** Initialize page tables**

*/

```

    movl    $pg0-__PAGE_OFFSET,%edi /* initialize page tables */
    movl    $007,%eax                /* "007" doesn't mean with right
                                     PRESENT+RW+USER */

```

```

2:    stosl
    add     $0x1000,%eax
    cmp     $empty_zero_page-__PAGE_OFFSET,%edi
    jne     2b

```

.org 0x2000
ENTRY(pg0)

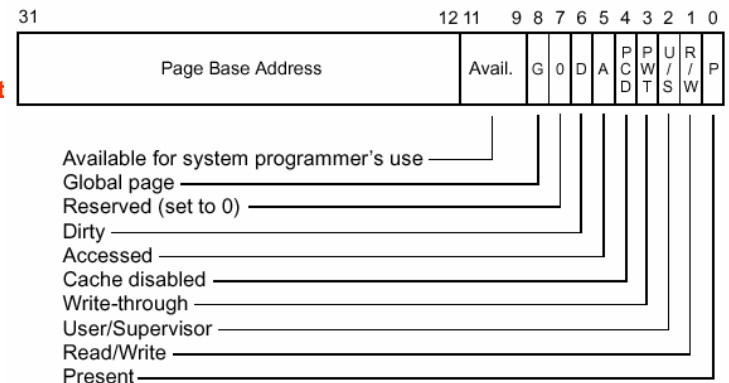
.org 0x3000
ENTRY(pg1)

.org 0x4000
ENTRY(empty_zero_page)

•What we initialized here is a provisional page table which mapping from both the linear addresses 0 through 0x3ffff and the linear addresses PAGE_OFFSET through PAGE_OFFSET + 0x3ffff into the physical address 0 through 0x3ffffff.

•In other words, the kernel during this phase of initialization can address the first 4MB of RAM either using linear address identical to the physical ones or using 4MB worth of linear address starting from PAGE_OFFSET.

Page-Table Entry (4-KByte Page)



Renaissance: The startup_32 Functions

```

/*
 * Enable paging
 */
3:
    movl $swapper_pg_dir-__PAGE_OFFSET,%eax
    movl %eax,%cr3                /* set the page table pointer.. */
    movl %cr0,%eax
    orl $0x80000000,%eax
    movl %eax,%cr0                /* ..and set paging (PG) bit */
    jmp 1f                        /* flush the prefetch-queue */

1:
    movl $1f,%eax
    jmp *%eax                      /* make sure eip is relocated */

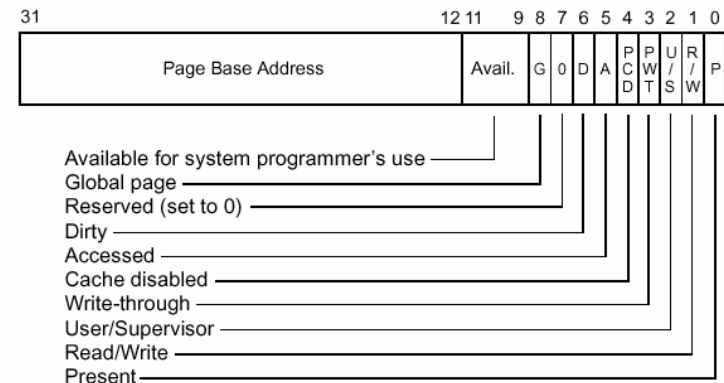
1:
    /* Set up the stack pointer */
    lss stack_start,%esp

/*
 * Clear BSS first so that there are no surprises...
 * No need to cld as DF is already clear from cld above...
 */
    xorl %eax,%eax
    movl $SYMBOL_NAME(__bss_start),%edi
    movl $SYMBOL_NAME(_end),%ecx
    subl %edi,%ecx
    rep
    stosb
    
```

```

ENTRY(swapper_pg_dir)
.long 0x00102007
.long 0x00103007
.fill BOOT_USER_PGD_PTRS-2,4,0
/* default: 766 entries */
.long 0x00102007
.long 0x00103007
/* default: 254 entries */
.fill BOOT_KERNEL_PGD_PTRS-2,4,0
    
```

Page-Table Entry (4-KByte Page)



Renaissance: The startup_32 Functions

```
/*
 * Enable paging
 */
3:
    movl $swapper_pg_dir-__PAGE_OFFSET,%eax
    movl %eax,%cr3                /* set the page table pointer.. */
    movl %cr0,%eax
    orl $0x80000000,%eax
    movl %eax,%cr0                /* ..and set paging (PG) bit */
    jmp 1f                        /* flush the prefetch-queue */

1:
    movl $1f,%eax
    jmp *%eax                      /* make sure eip is relocated */

1:
    /* Set up the stack pointer */
    lss stack_start,%esp

/*
 * Clear BSS first so that there are no surprises...
 * No need to cld as DF is already clear from cld above...
 */
    xorl %eax,%eax
    movl $SYMBOL_NAME(__bss_start),%edi
    movl $SYMBOL_NAME(_end),%ecx
    subl %edi,%ecx
    rep
    stosb
```

Renaissance: The startup_32 Functions

```
/*
 * Enable paging
 */
3:
    movl $swapper_pg_dir-__PAGE_OFFSET,%eax
    movl %eax,%cr3                /* set the page table pointer.. */
    movl %cr0,%eax
    orl $0x80000000,%eax
    movl %eax,%cr0                /* ..and set paging (PG) bit */
    jmp 1f                        /* flush the prefetch-queue */

1:
    movl $1f,%eax
    jmp *%eax                      /* make sure eip is relocated */

1:
    /* Set up the stack pointer */
    lss stack_start,%esp

/*
 * Clear BSS first so that there are no surprises...
 * No need to cld as DF is already clear from cld above...
 */
    xorl %eax,%eax
    movl $ SYMBOL_NAME(__bss_start),%edi
    movl $ SYMBOL_NAME(__end),%ecx
    subl %edi,%ecx
    rep
    stosb
```

This is the final stack of process 0.

```
ENTRY(stack_start)
.long SYMBOL_NAME(init_task_union)+8192
.long __KERNEL_DS
```

Renaissance: The startup_32 Functions

```
/*
 * Enable paging
 */
3:
    movl $swapper_pg_dir-__PAGE_OFFSET,%eax
    movl %eax,%cr3                /* set the page table pointer.. */
    movl %cr0,%eax
    orl $0x80000000,%eax
    movl %eax,%cr0                /* ..and set paging (PG) bit */
    jmp 1f                        /* flush the prefetch-queue */

1:
    movl $1f,%eax
    jmp *%eax                      /* make sure eip is relocated */

1:
    /* Set up the stack pointer */
    lss stack_start,%esp

/*
 * Clear BSS first so that there are no surprises...
 * No need to cld as DF is already clear from cld above...
 */
    xorl %eax,%eax
    movl $SYMBOL_NAME(__bss_start),%edi
    movl $SYMBOL_NAME(_end),%ecx
    subl %edi,%ecx
    rep
    stosb
```

Renaissance: The startup_32 Functions

```
    call    setup_idt

/*
 * 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
    ...
    call    SYMBOL_NAME(start_kernel)
L6:
    jmp L6                                # main should never return here
    ...
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    SYMBOL_NAME(idt_table),%edi
    mov    $256,%ecx
rp_sidt:
    movl   %eax,(%edi)
    movl   %edx,4(%edi)
    addl   $8,%edi
    dec    %ecx
    jne    rp_sidt
    ret
```

Renaiss

op_32 Functions

```

call    se
/*
 * Initialize eflags. S
 * confuse the debu
 * XXX - best to initi
 */
pushl  $0
popfl
...
call   SY
L6:    jmp L6
...
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    SYMBOL_NAME(idt_table),%edi
mov    $256,%ecx

rp_sidt:
movl   %eax,(%edi)
movl   %edx,4(%edi)
addl   $8,%edi
decl   %ecx
jne    rp_sidt
ret

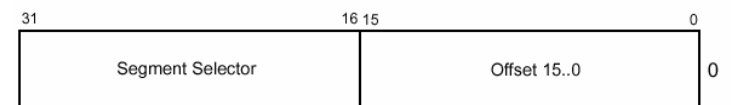
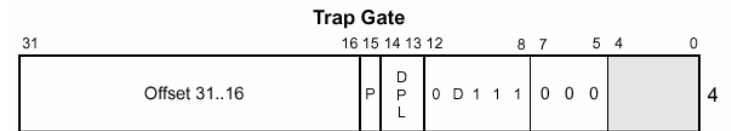
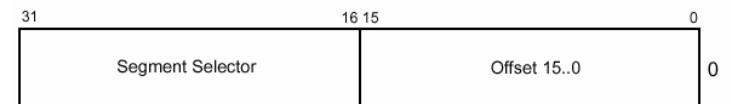
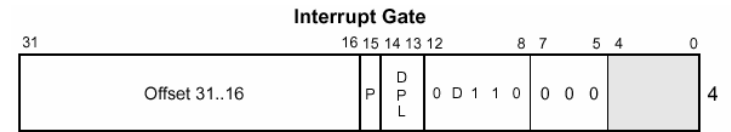
```

```

ignore_int:
cld
pushl %eax
pushl %ecx
pushl %edx
pushl %es
pushl %ds
movl  $(__KERNEL_DS),%eax
movl %eax,%ds
movl %eax,%es
pushl $int_msg
call SYMBOL_NAME(printk)
popl %eax
popl %ds
popl %es
popl %edx
popl %ecx
popl %eax
iret

```

return here



- DPL Descriptor Privilege Level
- Offset Offset to procedure entry point
- P Segment Present flag
- Selector Segment Selector for destination code segment
- D Size of gate: 1 = 32 bits; 0 = 16 bits

Reserved

Renaissance: The startup_32 Functions

```
        call    setup_idt
/*
 * 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
        ...
        call    SYMBOL_NAME(start_kernel)
L6:
        jmp L6                # main should never return here
        ...
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    SYMBOL_NAME(idt_table),%edi
        mov    $256,%ecx
rp_sidt:
        movl   %eax,(%edi)
        movl   %edx,4(%edi)
        addl   $8,%edi
        dec    %ecx
        jne    rp_sidt
        ret
```

Renaissance: The startup_32 Functions

```
    call    setup_idt
/*
 * 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
    ...
    call    SYMBOL_NAME(start_kernel)
L6:
    jmp L6                                # main should never return here
    ...
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    SYMBOL_NAME(idt_table),%edi
    mov    $256,%ecx
rp_sidt:
    movl   %eax,(%edi)
    movl   %edx,4(%edi)
    addl   $8,%edi
    decl   %ecx
    jne    rp_sidt
    ret
```

Renaissance: The startup_32 Functions

```
    call    setup_idt
/*
 * 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
    ...
    call    SYMBOL_NAME(start_kernel)
L6:
    jmp L6                                # main should never return here
    ...
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    SYMBOL_NAME(idt_table),%edi
    mov    $256,%ecx
rp_sidt:
    movl   %eax,(%edi)
    movl   %edx,4(%edi)
    addl   $8,%edi
    dec    %ecx
    jne    rp_sidt
    ret
```

Renaissance: The startup_32 Functions

```
    call    setup_idt
/*
 * 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
    ...
    call    SYMBOL_NAME(start_kernel)
L6:
    jmp L6                                # main should never return here
    ...
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    SYMBOL_NAME(idt_table),%edi
    mov    $256,%ecx
rp_sidt:
    movl   %eax,(%edi)
    movl   %edx,4(%edi)
    addl   $8,%edi
    dec    %ecx
    jne    rp_sidt
    ret
```

Industrial Revolution: The start_kernel() Function

- ✍ The start_kernel() function completes the initialization of the Linux kernel. Nearly every kernel components is initialized by this function. It performs the following operations:
 - Take a global kernel lock (It is needed so that only one CPU goes through initialization).
 - Print Linux kernel “banner”.
 - Perform arch-specific setup(memory layout analysis, copying boot command line again, etc.).
 - Print kernel command line.
 - Parse boot command line options.
 - Initialize various subsystem such as interrupt handling, memory management, file system, IPC, etc...
 - Perform arch-specific “check for bugs”.
 - Call **rest_init()** to create kernel thread **init** (pid = 1) and go into the idle loop. This is an idle thread with pid = 0.

Industrial Revolution: The start_kernel() Function

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

`setup_arch()` performs arch-specific setup, including memory layout analysis, copying boot command line again, call `pagetable_init()` to establish the final kernel page table.

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void __init pagetable_init (void)
{
    unsigned long vaddr, end;
    pgd_t *pgd, *pgd_base;
    int i, j, k;
    pmd_t *pmd;
    pte_t *pte, *pte_base;
    ...
    pgd_base = swapper_pg_dir;
    i = __pgd_offset(PAGE_OFFSET);
    pgd = pgd_base + i;

    for (; i < PTRS_PER_PGD; pgd++, i++) {
        pmd = (pmd_t *)pgd;
        for (j = 0; j < PTRS_PER_PMD; pmd++, j++) {
            pte_base = pte = (pte_t *) alloc_bootmem_low_pages(PAGE_SIZE);
            for (k = 0; k < PTRS_PER_PTE; pte++, k++) {
                vaddr = i*PGDIR_SIZE + j*PMD_SIZE + k*PAGE_SIZE;
                *pte = mk_pte_phys(__pa(vaddr), PAGE_KERNEL);
            }
            set_pmd(pmd, __pmd(_KERNPG_TABLE + __pa(pte_base)));
        }
    }
    ...
}
```

The code listed here are highly simplified by I.

```
static void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void __init pagetable_init (void)
{
    unsigned long vaddr, end;
    pgd_t *pgd, *pgd_base;
    int i, j, k;
    pmd_t *pmd;
    pte_t *pte, *pte_base;
    ...
    pgd_base = swapper_pg_dir;
    i = __pgd_offset(PAGE_OFFSET);
    pgd = pgd_base + i;

    for (; i < PTRS_PER_PGD; pgd++, i++) {
        pmd = (pmd_t *)pgd;
        for (j = 0; j < PTRS_PER_PMD; pmd++, j++) {
            pte_base = pte = (pte_t *) alloc_bootmem_low_pages(PAGE_SIZE);
            for (k = 0; k < PTRS_PER_PTE; pte++, k++) {
                vaddr = i*PGDIR_SIZE + j*PMD_SIZE + k*PAGE_SIZE;
                *pte = mk_pte_phys(__pa(vaddr), PAGE_KERNEL);
            }
            set_pmd(pmd, __pmd(_KERNPG_TABLE + __pa(pte_base)));
        }
    }
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void __init pagetable_init (void)
{
    unsigned long vaddr, end;
    pgd_t *pgd, *pgd_base;
    int i, j, k;
    pmd_t *pmd;
    pte_t *pte, *pte_base;
    ...
    pgd_base = swapper_pg_dir;
    i = __pgd_offset(PAGE_OFFSET);
    pgd = pgd_base + i;

    for (; i < PTRS_PER_PGD; pgd++, i++) {
        pmd = (pmd_t *)pgd;
        for (j = 0; j < PTRS_PER_PMD; pmd++, j++) {
            pte_base = pte = (pte_t *) alloc_bootmem_low_pages(PAGE_SIZE);
            for (k = 0; k < PTRS_PER_PTE; pte++, k++) {
                vaddr = i*PGDIR_SIZE + j*PMD_SIZE + k*PAGE_SIZE;
                *pte = mk_pte_phys(__pa(vaddr), PAGE_KERNEL);
            }
            set_pmd(pmd, __pmd(_KERNPG_TABLE + __pa(pte_base)));
        }
    }
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void __init pagetable_init (void)
{
    unsigned long vaddr, end;
    pgd_t *pgd, *pgd_base;
    int i, j, k;
    pmd_t *pmd;
    pte_t *pte, *pte_base;
    ...
    pgd_base = swapper_pg_dir;
    i = __pgd_offset(PAGE_OFFSET);
    pgd = pgd_base + i;

    for (; i < PTRS_PER_PGD; pgd++, i++) {
        pmd = (pmd_t *)pgd;
        for (j = 0; j < PTRS_PER_PMD; pmd++, j++) {
            pte_base = pte = (pte_t *) alloc_bootmem_low_pages(PAGE_SIZE);
            for (k = 0; k < PTRS_PER_PTE; pte++, k++) {
                vaddr = i*PGDIR_SIZE + j*PMD_SIZE + k*PAGE_SIZE;
                *pte = mk_pte_phys(__pa(vaddr), PAGE_KERNEL);
            }
            set_pmd(pmd, __pmd(_KERNPG_TABLE + __pa(pte_base)));
        }
    }
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void __init pagetable_init (void)
{
    unsigned long vaddr, end;
    pgd_t *pgd, *pgd_base;
    int i, j, k;
    pmd_t *pmd;
    pte_t *pte, *pte_base;
    ...
    pgd_base = swapper_pg_dir;
    i = __pgd_offset(PAGE_OFFSET);
    pgd = pgd_base + i;

    for (; i < PTRS_PER_PGD; pgd++, i++) {
        pmd = (pmd_t *)pgd;
        for (j = 0; j < PTRS_PER_PMD; pmd++, j++) {
            pte_base = pte = (pte_t *) alloc_bootmem_low_pages(PAGE_SIZE);
            for (k = 0; k < PTRS_PER_PTE; pte++, k++) {
                vaddr = i*PGDIR_SIZE + j*PMD_SIZE + k*PAGE_SIZE;
                *pte = mk_pte_phys(__pa(vaddr), PAGE_KERNEL);
            }
            set_pmd(pmd, __pmd(_KERNPG_TABLE + __pa(pte_base)));
        }
    }
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void __init pagetable_init (void)
{
    unsigned long vaddr, end;
    pgd_t *pgd, *pgd_base;
    int i, j, k;
    pmd_t *pmd;
    pte_t *pte, *pte_base;
    ...
    pgd_base = swapper_pg_dir;
    i = __pgd_offset(PAGE_OFFSET);
    pgd = pgd_base + i;

    for (; i < PTRS_PER_PGD; pgd++, i++) {
        pmd = (pmd_t *)pgd;
        for (j = 0; j < PTRS_PER_PMD; pmd++, j++) {
            pte_base = pte = (pte_t *) alloc_bootmem_low_pages(PAGE_SIZE);
            for (k = 0; k < PTRS_PER_PTE; pte++, k++) {
                vaddr = i*PGDIR_SIZE + j*PMD_SIZE + k*PAGE_SIZE;
                *pte = mk_pte_phys(__pa(vaddr), PAGE_KERNEL);
            }
            set_pmd(pmd, __pmd(_KERNPG_TABLE + __pa(pte_base)));
        }
    }
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void __init pagetable_init (void)
{
    unsigned long vaddr, end;
    pgd_t *pgd, *pgd_base;
    int i, j, k;
    pmd_t *pmd;
    pte_t *pte, *pte_base;
    ...
    pgd_base = swapper_pg_dir;
    i = __pgd_offset(PAGE_OFFSET);
    pgd = pgd_base + i;

    for (; i < PTRS_PER_PGD; pgd++, i++) {
        pmd = (pmd_t *)pgd;
        for (j = 0; j < PTRS_PER_PMD; pmd++, j++) {
            pte_base = pte = (pte_t *) alloc_bootmem_low_pages(PAGE_SIZE);
            for (k = 0; k < PTRS_PER_PTE; pte++, k++) {
                vaddr = i*PGDIR_SIZE + j*PMD_SIZE + k*PAGE_SIZE;
                *pte = mk_pte_phys(__pa(vaddr), PAGE_KERNEL);
            }
            set_pmd(pmd, __pmd(_KERNPG_TABLE + __pa(pte_base)));
        }
    }
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void __init pagetable_init (void)
{
    unsigned long vaddr, end;
    pgd_t *pgd, *pgd_base;
    int i, j, k;
    pmd_t *pmd;
    pte_t *pte, *pte_base;
    ...
    pgd_base = swapper_pg_dir;
    i = __pgd_offset(PAGE_OFFSET);
    pgd = pgd_base + i;

    for (; i < PTRS_PER_PGD; pgd++, i++) {
        pmd = (pmd_t *)pgd;
        for (j = 0; j < PTRS_PER_PMD; pmd++, j++) {
            pte_base = pte = (pte_t *) alloc_bootmem_low_pages(PAGE_SIZE);
            for (k = 0; k < PTRS_PER_PTE; pte++, k++) {
                vaddr = i*PGDIR_SIZE + j*PMD_SIZE + k*PAGE_SIZE;
                *pte = mk_pte_phys(__pa(vaddr), PAGE_KERNEL);
            }
            set_pmd(pmd, __pmd(_KERNPG_TABLE + __pa(pte_base)));
        }
    }
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void __init pagetable_init (void)
{
    unsigned long vaddr, end;
    pgd_t *pgd, *pgd_base;
    int i, j, k;
    pmd_t *pmd;
    pte_t *pte, *pte_base;
    ...
    pgd_base = swapper_pg_dir;
    i = __pgd_offset(PAGE_OFFSET);
    pgd = pgd_base + i;

    for (; i < PTRS_PER_PGD; pgd++, i++) {
        pmd = (pmd_t *)pgd;
        for (j = 0; j < PTRS_PER_PMD; pmd++, j++) {
            pte_base = pte = (pte_t *) alloc_bootmem_low_pages(PAGE_SIZE);
            for (k = 0; k < PTRS_PER_PTE; pte++, k++) {
                vaddr = i*PGDIR_SIZE + j*PMD_SIZE + k*PAGE_SIZE;
                *pte = mk_pte_phys(__pa(vaddr), PAGE_KERNEL);
            }
            set_pmd(pmd, __pmd(_KERNPG_TABLE + __pa(pte_base)));
        }
    }
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
```

Parse Kernel command line.

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
```

Industrial Revolution: The start_kernel() Function

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
```

Initializing x86 exception handling.

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
```

Industrial start_kernel

- set_intr_gate(n, addr): Inserts an interrupt gate in the nth IDT entry. The DPL field is set to 0.
- set_system_gate(n, addr): Inserts a trap gate in nth IDT entry. The DPL field is set to 3.
- set_trap_gate(n, addr): Similar to the previous function, except that the DPL field is set to 0.

```
void __init trap_init(void)
{
    ...
    set_trap_gate(0,&divide_error);
    set_trap_gate(1,&debug);
    set_intr_gate(2, &nmi);
    set_system_gate(3,&int3);          /* int3-5 can be called from a
    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_intr_gate(14,&page_fault);
    set_trap_gate(15,&spurious_interrupt_bug);
    set_trap_gate(16,&coprocessor_error);
    set_trap_gate(17,&alignment_check);
    set_trap_gate(18,&machine_check);
    set_trap_gate(19,&simd_coprocessor_error);
    set_system_gate(SYSCALL_VECTOR,&system_call);
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Initializing x86 interrupt handling.

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
void __init init_IRQ(void)
{
    int i;
    ...
    /*
     * Cover the whole vector space, no vector can escape
     * us. (some of these will be overridden and become
     * 'special' SMP interrupts)
     */
    for (i = 0; i < NR_IRQS; i++) {
        int vector = FIRST_EXTERNAL_VECTOR + i;
        if (vector != SYSCALL_VECTOR)
            set_intr_gate(vector, interrupt[i]);
    }

    /*
     * Set the clock to HZ Hz, we already have a valid
     * vector now:
     */
    outb_p(0x34,0x43);                /* binary, mode 2, LSB/MSB, ch 0 */
    outb_p(LATCH & 0xff , 0x40);      /* LSB */
    outb(LATCH >> 8 , 0x40);         /* MSB */
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
void __init init_IRQ(void)
{
    int i;
    ...
    /*
     * Cover the whole vector space, no vector can escape
     * us. (some of these will be overridden and become
     * 'special' SMP interrupts)
     */
    for (i = 0; i < NR_IRQS; i++) {
        int vector = FIRST_EXTERNAL_VECTOR + i;
        if (vector != SYSCALL_VECTOR)
            set_intr_gate(vector, interrupt[i]);
    }

    /*
     * Set the clock to HZ Hz, we already have a valid
     * vector now:
     */
    outb_p(0x34,0x43);          /* binary, mode 2, LSB/MSB, ch 0 */
    outb_p(LATCH & 0xff , 0x40); /* LSB */
    outb(LATCH >> 8 , 0x40);    /* MSB */
    ...
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];
    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: The start_kernel() Function

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

```
asmlinkage void __init start_kernel(void)
{
    char * command_line;
    unsigned long mempages;
    extern char saved_command_line[];

    /*
     * Interrupts are still disabled. Do necessary setups, then
     * enable them
     */
    lock_kernel();
    printk(linux_banner);
    setup_arch(&command_line);
    printk("Kernel command line: %s\n", saved_command_line);
    parse_options(command_line);
    trap_init();
    init_IRQ();
    sched_init();
    softirq_init();
    time_init();
}
```

Industrial Revolution: T start_kernel() Function

```
console_init();
#ifdef CONFIG_MODULES
    init_modules();
#endif
if (prof_shift) {
    unsigned int size;
    /* only text is profiled */
    prof_len = (unsigned long) &_etext - (unsigned long) &_stext;
    prof_len >>= prof_shift;

    size = prof_len * sizeof(unsigned int) + PAGE_SIZE-1;
    prof_buffer = (unsigned int *) alloc_bootmem(size);
}

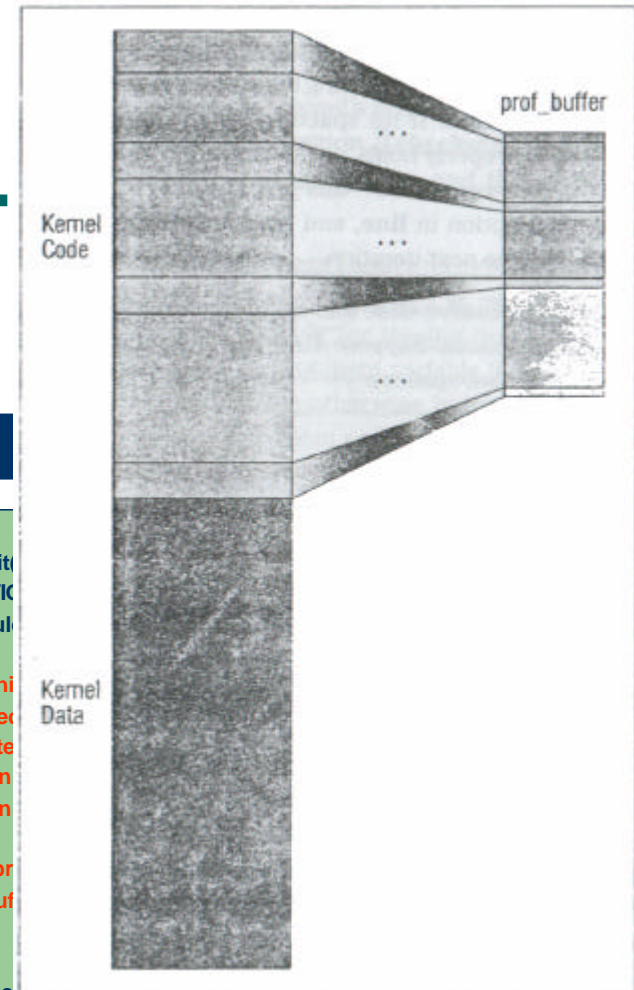
kmem_cache_init();
sti();
calibrate_delay();
#ifdef CONFIG_BLK_DEV_INITRD
if (initrd_start && !initrd_below_start_ok &&
    initrd_start < min_low_pfn << PAGE_SHIFT) {
    printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
        "disabling it.\n",initrd_start,min_low_pfn << PAGE_SHIFT);
    initrd_start = 0;
}
}
```

Initialize profile buffer.

```
console_init
#ifdef CONFIG
    init_modul
#endif
if (prof_shi
    unsigned
    /* only te
    prof_len
    prof_len

    size = pr
    prof_bu
}

kmem_cac
sti();
calibrate_delay();
#ifdef CONFIG_BLK_DEV_INITRD
if (initrd_start && !initrd_below_start_ok &&
    initrd_start < min_low_pfn << PAGE_SHIFT) {
    printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
        "disabling it.\n",initrd_start,min_low_pfn << PAGE_SHIFT);
    initrd_start = 0;
}
}
```



Industrial Revolution: The start_kernel() Function

```
console_init();
#ifdef CONFIG_MODULES
    init_modules();
#endif
if (prof_shift) {
    unsigned int size;
    /* only text is profiled */
    prof_len = (unsigned long) &_amp;_etext - (unsigned long) &_amp;_stext;
    prof_len >>= prof_shift;

    size = prof_len * sizeof(unsigned int) + PAGE_SIZE-1;
    prof_buffer = (unsigned int *) alloc_bootmem(size);
}

kmem_cache_init();
sti();
calibrate_delay();
#ifdef CONFIG_BLK_DEV_INITRD
if (initrd_start && !initrd_below_start_ok &&
    initrd_start < min_low_pfn << PAGE_SHIFT) {
    printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
        "disabling it.\n",initrd_start,min_low_pfn << PAGE_SHIFT);
    initrd_start = 0;
}
}
```

Enable interrupt.
Calculate BogoMIPs.

```
console_init();
#ifdef CONFIG_MODULES
    init_modules();
#endif
if (prof_shift) {
    unsigned int size;
    /* only text is profiled */
    prof_len = (unsigned long) &_amp;_etext - (unsigned long) &_amp;_stext;
    prof_len >>= prof_shift;

    size = prof_len * sizeof(unsigned int) + PAGE_SIZE-1;
    prof_buffer = (unsigned int *) alloc_bootmem(size);
}

kmem_cache_init();
sti();
calibrate_delay();
#ifdef CONFIG_BLK_DEV_INITRD
if (initrd_start && !initrd_below_start_ok &&
    initrd_start < min_low_pfn << PAGE_SHIFT) {
    printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
        "disabling it.\n",initrd_start,min_low_pfn << PAGE_SHIFT);
    initrd_start = 0;
}
}
```

Industrial Revolution: The start_kernel() Function

```
#define LPS_PREC 8

void __init calibrate_delay(void)
{
    unsigned long ticks, loopbit;
    int lps_precision = LPS_PREC;

    loops_per_jiffy = (1<<12);

    printk("Calibrating delay loop... ");
    while (loops_per_jiffy <= 1) {
        /* wait for "start of" clock tick */
        ticks = jiffies;
        while (ticks == jiffies)
            /* nothing */;

        /* Go .. */
        ticks = jiffies;
        __delay(loops_per_jiffy);
        ticks = jiffies - ticks;
        if (ticks)
            break;
    }
}
```

```
console_init();
#ifdef CONFIG_MODULES
    init_modules();
#endif
if (prof_shift) {
    unsigned int size;
    /* only text is profiled */
    prof_len = (unsigned long) &_etext - (unsigned long) &_stext;
    prof_len >>= prof_shift;

    size = prof_len * sizeof(unsigned int) + PAGE_SIZE-1;
    prof_buffer = (unsigned int *) alloc_bootmem(size);
}

kmem_cache_init();
sti();
calibrate_delay();
#ifdef CONFIG_BLK_DEV_INITRD
    if (initrd_start && !initrd_below_start_ok &&
        initrd_start < min_low_pfn << PAGE_SHIFT) {
        printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
            "disabling it.\n", initrd_start, min_low_pfn << PAGE_SHIFT);
        initrd_start = 0;
    }
}
```

Industrial Revolution: The start_kernel() Function

```
/* Do a binary approximation to get loops_per_jiffy set to equal
   (up to lps_precision bits) */
loops_per_jiffy >>= 1;
loopbit = loops_per_jiffy;
while ( lps_precision-- && (loopbit >>= 1) ) {
    loops_per_jiffy |= loopbit;
    ticks = jiffies;
    while (ticks == jiffies);
    ticks = jiffies;
    __delay(loops_per_jiffy);
    if (jiffies != ticks) /* longer than 1 tick */
        loops_per_jiffy &= ~loopbit;
}

/* Round the value and print it */
printk("%lu.%02lu BogoMIPS\n",
        loops_per_jiffy/(500000/HZ),
        (loops_per_jiffy/(5000/HZ)) % 100);
}
```

```
console_init();
#ifdef CONFIG_MODULES
    init_modules();
#endif
if (prof_shift) {
    unsigned int size;
    /* only text is profiled */
    prof_len = (unsigned long) &_etext - (unsigned long) &_stext;
    prof_len >>= prof_shift;

    size = prof_len * sizeof(unsigned int) + PAGE_SIZE-1;
    prof_buffer = (unsigned int *) alloc_bootmem(size);
}

kmem_cache_init();
sti();
calibrate_delay();
#ifdef CONFIG_BLK_DEV_INITRD
if (initrd_start && !initrd_below_start_ok &&
    initrd_start < min_low_pfn << PAGE_SHIFT) {
    printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
           "disabling it.\n",initrd_start,min_low_pfn << PAGE_SHIFT);
    initrd_start = 0;
}
}
```

Industrial Revolution: The start_kernel() Function

```
console_init();
#ifdef CONFIG_MODULES
    init_modules();
#endif
if (prof_shift) {
    unsigned int size;
    /* only text is profiled */
    prof_len = (unsigned long) &_amp;_etext - (unsigned long) &_amp;_stext;
    prof_len >>= prof_shift;

    size = prof_len * sizeof(unsigned int) + PAGE_SIZE-1;
    prof_buffer = (unsigned int *) alloc_bootmem(size);
}

kmem_cache_init();
sti();
calibrate_delay();
#ifdef CONFIG_BLK_DEV_INITRD
if (initrd_start && !initrd_below_start_ok &&
    initrd_start < min_low_pfn << PAGE_SHIFT) {
    printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
        "disabling it.\n",initrd_start,min_low_pfn << PAGE_SHIFT);
    initrd_start = 0;
}
}
```

```
console_init();
#ifdef CONFIG_MODULES
    init_modules();
#endif
if (prof_shift) {
    unsigned int size;
    /* only text is profiled */
    prof_len = (unsigned long) &_amp;_etext - (unsigned long) &_amp;_stext;
    prof_len >>= prof_shift;

    size = prof_len * sizeof(unsigned int) + PAGE_SIZE-1;
    prof_buffer = (unsigned int *) alloc_bootmem(size);
}

kmem_cache_init();
sti();
calibrate_delay();
#ifdef CONFIG_BLK_DEV_INITRD
if (initrd_start && !initrd_below_start_ok &&
    initrd_start < min_low_pfn << PAGE_SHIFT) {
    printk(KERN_CRIT "initrd overwritten (0x%08lx < 0x%08lx) - "
        "disabling it.\n",initrd_start,min_low_pfn << PAGE_SHIFT);
    initrd_start = 0;
}
}
```

Industrial Revolution: The start_kernel() Function

```
mem_init();
kmem_cache_sizes_init();
pgtable_cache_init();
mempages = num_physpages;
fork_init(mempages);
proc_caches_init();
vfs_caches_init(mempages);
buffer_init(mempages);
page_cache_init(mempages);
#if defined(CONFIG_ARCH_S390)
    ccwcache_init();
#endif
    signals_init();
#ifdef CONFIG_PROC_FS
    proc_root_init();
#endif
#if defined(CONFIG_SYSVIPC)
    ipc_init();
#endif
    check_bugs();
    printk("POSIX conformance testing by UNIFIX\n");
    smp_init();
    rest_init();
}
```

```
mem_init();
kmem_cache_sizes_init();
pgtable_cache_init();
mempages = num_physpages;
fork_init(mempages);
proc_caches_init();
vfs_caches_init(mempages);
buffer_init(mempages);
page_cache_init(mempages);
#if defined(CONFIG_ARCH_S390)
    ccwcache_init();
#endif
    signals_init();
#ifdef CONFIG_PROC_FS
    proc_root_init();
#endif
#if defined(CONFIG_SYSVIPC)
    ipc_init();
#endif
    check_bugs();
    printk("POSIX conformance testing by UNIFIX\n");
    smp_init();
    rest_init();
}
```

Industrial Revolution: The start_kernel() Function

```
static void rest_init(void)
{
    kernel_thread(init, NULL, CLONE_FS | CLONE_FILES | CLONE_SIGNAL);
    unlock_kernel();
    current->need_resched = 1;
    cpu_idle();
}
```

```
    mem_init();
    kmem_cache_sizes_init();
    pgtable_cache_init();
    mempages = num_physpages;
    fork_init(mempages);
    proc_caches_init();
    vfs_caches_init(mempages);
    buffer_init(mempages);
    page_cache_init(mempages);
    #if defined(CONFIG_ARCH_S390)
        ccwcache_init();
    #endif
    signals_init();
    #ifdef CONFIG_PROC_FS
        proc_root_init();
    #endif
    #if defined(CONFIG_SYSVIPC)
        ipc_init();
    #endif
    check_bugs();
    printk("POSIX conformance testing by UNIFIX\n");
    smp_init();
    rest_init();
}
```

Modern Age: The first process – init() Kernel Thread

- ✍ init() is the first user process run by the kernel, and it is responsible for firing off all the other processes that are needed to put the system as a whole into a really usable state.
- ✍ The init() does following works:
 - Call do_basic_setup to initialize the buses and spawn some other kernel threads.
 - Call free_initmem to release the functions and data which not needed anymore.
 - Try to find the user provided “init” program and execute it.

Modern Age: The first process – init() Kernel Thread

```
static int init(void * unused)
{
    lock_kernel();
    do_basic_setup();
    prepare_namespace();
    free_initmem();
    unlock_kernel();

    if (open("/dev/console", O_RDWR, 0) < 0)
        printk("Warning: unable to open an initial console.\n");

    (void) dup(0);
    (void) dup(0);

    if (execute_command)
        execve(execute_command,argv_init,envp_init);
    execve("/sbin/init",argv_init,envp_init);
    execve("/etc/init",argv_init,envp_init);
    execve("/bin/init",argv_init,envp_init);
    execve("/bin/sh",argv_init,envp_init);
    panic("No init found. Try passing init= option to kernel.");
}
```

Modern Age: The first process – init() Kernel Thread

```
static int init(void * unused)
{
    lock_kernel();
    do_basic_setup();
    prepare_namespace();
    free_initmem();
    unlock_kernel();

    if (open("/dev/console", O_RDWR, 0) < 0)
        printk("Warning: unable to open an initial console.\n");

    (void) dup(0);
    (void) dup(0);

    if (execute_command)
        execve(execute_command,argv_init,envp_init);
    execve("/sbin/init",argv_init,envp_init);
    execve("/etc/init",argv_init,envp_init);
    execve("/bin/init",argv_init,envp_init);
    execve("/bin/sh",argv_init,envp_init);
    panic("No init found. Try passing init= option to kernel.");
}
```

Modern Age: The first process – init() Kernel Thread

```
static int init(void * unused)
{
    lock_kernel();
    do_basic_setup();
    prepare_namespace();
    free_initmem();
    unlock_kernel();

    if (open("/dev/console", O_RDWR, 0) < 0)
        printk("Warning: unable to open an initial console.\n");

    (void) dup(0);
    (void) dup(0);

    if (execute_command)
        execve(execute_command,argv_init,envp_init);
        execve("/sbin/init",argv_init,envp_init);
        execve("/etc/init",argv_init,envp_init);
        execve("/bin/init",argv_init,envp_init);
        execve("/bin/sh",argv_init,envp_init);
        panic("No init found. Try passing init= option to kernel.");
}
```

Related Resources

- ✍ “Linux Core Kernel Commentary”, CORIOLIS
- ✍ “Understanding the LINUX KERNEL”, O’REILLY
- ✍ “Linux Device Drivers”, O’REILLY
- ✍ Cross-Referencing Linux, <http://lxr.linux.no/>
- ✍ Linux Documentation Project,
<http://www.ibiblio.org/mdw/index.html>