

CATernel

SMP Interactive uKernel

By

Saad Talaat
&
Menna Essa

Table of Contents

-Perface.....	2
1. Prototype One.....	3
1.Boot Loader.....	5-9
2.Setting up the environment	10-11
3.Drivers.....	11-22
4.Memory Management.....	23-32
5.Interrupts and System calls.....	33-37
6.Process Management.....	38-39
2. Prototype Two.....	40
1.Design Model.....	41-46
2.Supporting SMP.....	47-52
3.Synchronization.....	52-54
4.Time Management.....	55
5. Virtual File System.....	56
3.Appendix A – Problems faced.....	60
4.Code documentation.	

Preface:

CATernel project aims to develop a non-portable kernel (as a start) that uses Unix interfaces and APIs. CATernel was initiated at the start as a collaborative project in technical community called [Computer Assistance Team] ;It has been initiated (and still) by two students (Saad Talaat Saad , Menna Sherif Essa) on late 2011.Goal is to develop a limited monolithic kernel which supports IA32 architecture for educational and learning purposes.

Project is planned to support a single architecture as a start and also pass through various types of kernel models. The goal of that progressive development is to keep the kernel operating in all cases. And making it usable at any phase of development ; However, Porting the kernel would be carried out at least after we reach the monolithic model. But our main architecture is the IA32 , The Progressive model of the CATernel project can be considered as a prototype design model since in this case the Exokernel model will be a prototype to final goal.

Iterative prototyping is the used software model. Every project sprint contains the components of the project plan. And since we're using the [research & code] way in developing the kernel, iterative prototyping is the convenient software model to use.

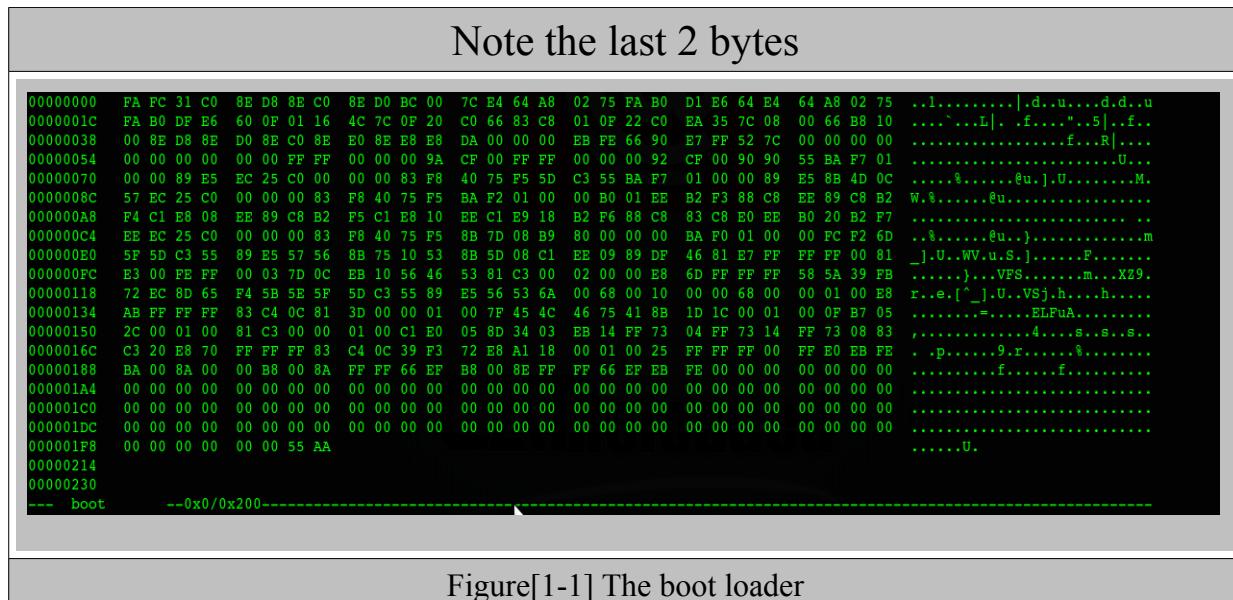
[Prototype One]

1.The Boot Loader:

1.1 Introduction:

booting is the initial set of operations that a computer system performs when electrical power is switched on ; **A boot loader** is a computer program that loads the main operating system or runtime environment for the computer after completion of self-tests.

When a computer is powered the BIOS comes in control and initializes all data. then it looks for a valid boot loader through in the order of the boot device order. a bootable sector is known by the last 2 bytes in the sector, they must be 0xAA55 (boot signature). That Image has the boot loader of our CATernel.



When the BIOS finds a bootable image it loads the first 512 bytes into address 0:07C00 then jumps to it. Then the boot sector comes into control. It starts execution in the **real mode**. Real mode is characterized by a 20-bit segmented memory address space (giving exactly 1 MiB[1] of addressable memory) and unlimited direct software access to all memory, I/O addresses and peripheral hardware. Real mode provides no support for memory protection, multitasking, or code

privilege levels.

```
.global start
start:
.code16 #since we are in real mode
cli #disable interrupts
cld #clear the direction flag
xorw %ax,%ax #clear the ax register
movw %ax,%ds #clear the data segment register
movw %ax,%es #clear the extra segment register
movw %ax,%ss #clear the stack segment register
movw $start,%sp #set the stack pointer to the bootsector stack
```

Listing 1-1 : Boot sector starts at real mode

1.2 Enabling Protected Mode:

Next it switches to **protected mode** which allows system software to use features such as virtual memory, paging and safe multi tasking.

1.2.1 Enabling A20 Gate:

We start by enabling the A20 Gate [2] for more addressing ; On enabling the A20 gate. first we check if input buffer is full by checking if bit 1 is set on the 0x64 port. then output 0xD1 which makes the next byte passed through the 0x60 port written to IBM AT 804x port. in the next procedure A20.2, you check if the input buffer is full or not , then write 0xDF to the 0x60 port which is written to the IBM AT 804x port which finally enables A20 gate

```
A20.1:
inb $0x64,%al
testb $0x2,%al
jnz A20.1
movb $0xd1,%al
outb %al,$0x64
A20.2:
inb $0x64,%al
testb $0x2,%al
```

```
jnz A20.2
movb $0xdf,%al
outb %al,$0x60
```

Listing 1-2 : Enabling A20 Gate

1.2.2 Setting up GDT:

Next step is to enable segmentation , For that you need to setup a GDT[3] -Global Descriptor Table- , For the reason that in Protected mode you can't refer to segments simply by doing this :

```
jmp 0000:7c00h
jmp 0002:0020h
```

You cannot directly access segments. That's why a GDT is used , In the GDT is a table where all the segments are defined . yet they are not stored as values but as descriptors. Descriptors has full information about a segment. A Descriptor is 64 bit long.

31	16	15	0	
Base 0:15		Limit 0:15		
63	56	55 52	51 48 47 40 39 32	
Base 24:31	Flags	Limit 16:19	Access Byte	Base 16:23

Access byte is

byte 0 = Accessed bit set to 1 by CPU when segment is accessed. we will set it to 0

byte 1 = read/write permissions

byte 2 = Direction bit we will set that to 0 for growing up segments and 1 for growing down segments and conforming bit

byte 3 = Executable bit 1 if code segment 0 if data segment

byte 4 = always 1

byte 5,6 = Privilege since we are a kernel we will set that to 0

byte 7 = Present bit one for anything

Figure 1-2 : The GDT Descriptor

As in Figure 1-2 , from the first bit till the bit 15, this is the place of the first 16 bits of the segment limit address. and from bit 16 to bit 31 it's the place of the first 16 bits of the segment base address. from bit 32 to 39 more 6 bits of the base address are placed. and then we go into the access bit which is demonstrated above, and the rest of the limit address of the segment. and Flags which is usually set to 0x1100. Finally followed by the rest of the base address. Indexing these descriptors is made by adding 0x8 for every descriptor. So first descriptor's index is 0x0, 2nd descriptor index is 0x8, 3rd descriptor index is 0x10.

```
gdt_table:  
.word gdt-gdt_end-1 #gdt table size....mostly 0x17  
.long gdt #gdt address  
gdt:  
.long 0,0 #Null segment  
.byte 0xff,0xff,0,0,0,0x9A,0xCF,0 #Code segment  
.byte 0xff,0xff,0,0,0,0x92,0xCF,0 #Data segment
```

Listing 1-3 : Setting up GDT Table

After setting up the GDT and loading the gdt address in the gdt register we set the protected mode in cr0 flag. Doing a far jump usingljmp and using the code segment value (0x8) as a segment and next procedure as an offset modified the CS value to the one in our GDT table.

```

switch_mode:

lgdt gdt_table # Load the global descriptor register
mov %cr0,%eax # Load the control register 0 into eax
orl $1,%eax # set the protected mode flag
mov %eax,%cr0 # reset the control register 0
ljmp $CODE_SEG,$protseg #make a far jump to modify the Code
segment

```

Listing 1-4: loading gdt table and switching to protected mode.

After doing this we set the value of data,stack,extra,f,g segments to the one in the GDT. Now we are fully working on protected mode and ready to load the kernel.

```

protseg:

.code32 #since we are working on protected mode
movw $DATA_SEG,%ax #move the data segment value to ax
movw %ax,%ds #set the data segment to data segment value at the
gdt table
movw %ax,%ss # same
movw %ax,%es # same
movw %ax,%fs # same
movw %ax,%gs # same
call cmain # call our kernel loader

```

Listing 1-5 : Setting Registers to the ones in the GDT

1.3 Loading the kernel Image:

Note That we're compiling the kernel as an ELF image , The *cmain* function responsible for loading the kernel is in arch/x86/boot/main.c. What the function is that it reads the ELF file from the Image , loads it into memory at virtual address 0x10000 then jumps to the entry point of the kernel executable which referred to through the kernel's ELF headers (ELFHDR->entry) .

To do so it uses two functions , *readsect* and *readseg* in order to read the executable from disk , the two functions are also found in main.c file.

2. Establishing environment.

After kernel is fetched from boot loader, environment is reset and memory thunks and segments are remapped. Several components are set before running the kernel:-

- 1- Kernel stack.
- 2- Segmentation and GDT.
- 3- Page tables and directories.

2.1 Kernel Stack

Since execution stream is now in the kernel(0x100000) and not the boot sector(0x7c00). stack must be reset in order to carry on healthy execution we must reset. Kernel stack is paged size memory thunk which we set to global to be able to refer to once we set up paging.

```
.p2align PAGELG      # Will pad allocation to 0x1000
byte
    .global      kernel_stack
kernel_stack:
    .space      KERNEL_STACK
    .global      kernel_stack_end
kernel_stack_end:
```

Listing 2.1 - Kernel stack thunk.

2.2 Segmentation and GDT

while initializing the kernel it is needed to keep in mind the Virtual memory addressing. Thus, Global descriptor table and segmentation is reset to generate new virtual addresses, new kernel segments are 0x10000000 long. Kernel virtual base address is 0xF0000000 which puts the kernel code base address to 0xF0100000. At this point it is important to notice that there's only two segments selectors set.

```
gdt:  
    .long 0,0  
    SEGMENT(0xffffffff,-KERNEL_ADDR, SEGACS_RW|SEGACS_X) #  
code seg  
    SEGMENT(0xffffffff,-KERNEL_ADDR, SEGACS_RW) # data seg  
gdt_end:
```

Listing 2.2 - Kernel Initial Global descriptor table.

CATernel adopts a memory map that is not like any *nix system. Kernel is mapped to high memory addresses like windows. However we are not planning to be a UNIX like operating system.

2.3 Page tables and directories

Before setting up kernel the virtual page tables and page directories are set which will be needed for paging later. We shall touch this point later on a separate chapter.

3.Drivers :

3.1: CMOS/RTC :

CMOS Complementary metal–oxide–semiconductor , including the RTC -Real time clock- is responsible for saving 50 or 114 bytes of setup information for the BIOS , it includes a battery that keeps the clock active.

CMOS is accessed through I/O ports 0x70 and 0x71 , the ports specifications are as follows : note that indexing is little endian [RTL]

```
a)
0070 w      CMOS RAM index register port (ISA, EISA)
            bit 7      = 1  NMI disabled
            = 0  NMI enabled
            bit 6-0     CMOS RAM index (64 bytes, sometimes 128 bytes)

            any write to 0070 should be followed by an action to 0071
            or the RTC wil be left in an unknown state.

b)
0071 r/w    CMOS RAM data port (ISA, EISA)
            RTC registers:
            00      current second in BCD
            01      alarm second   in BCD
            02      current minute in BCD
            03      alarm minute   in BCD
            04      current hour   in BCD
            05      alarm hour     in BCD
            06      day of week   in BCD
            07      day of month  in BCD
            08      month in BCD
            09      year in BCD (00-99)
            0A      status register A
                    bit 7 = 1  update in progress
                    bit 6-4 divider that identifies the time-based
                        frequency
                    bit 3-0 rate selection output  frequency and int. rate
            0B      status register B
                    bit 7 = 0  run
                    = 1  halt
                    bit 6 = 1  enable periodic interrupt
                    bit 5 = 1  enable alarm interrupt
                    bit 4 = 1  enable update-ended interrupt
                    bit 3 = 1  enable square wave interrupt
                    bit 2 = 1  calendar is in binary format
```

```

        = 0  calendar is in BCD format
bit 1 = 1  24-hour mode
        = 0  12-hour mode
bit 0 = 1  enable daylight savings time. only in USA.
        useless in Europe. Some DOS versions clear
        this bit when you use the DAT/TIME command.

.
.
```

Listing 3.1 CMOS Ports , a) I/O Port 0x70 , b) I/O Port 0x71 [1]

CMOS values are accessed one byte at a time so we refer to each byte as a CMOS Register, The first 14 CMOS registers access and control the Real-Time Clock. In port 0x70 -CMOS RAM index register port- as it's name suggests saves the CMOS RAM index which is a Nonvolatile BIOS memory refers to a small memory on PC motherboards that is used to store BIOS settings , Also the last bit in the register indicates whether the NMI -non maskable interrupts-[2] are enabled (0) or disabled (1).

So port 0x70 is used to select the CMOS Register to read , so if you want to read register 0A which holds Status A register you simply do this :
`outb(0xA, 0x70);`

For making things readable , we define all the indexes and constants in cmos.h headers

```

/* CMOS Registers */
#define CMOS_INDEXPORT      0x70
#define CMOS_DATAPORT       0x71
/* RTC Registers */
#define RTC_SECONDS          0x0
#define RTC_ALRMSECOND       0x1
#define RTC_MINUTES          0x2
.
.
```

Listing 3.2 cmos.h

From here we can introduce the first CMOS function ,*cmos_get_reg*; which will take the register the needs to be read as an input , offset the index register and

read it. to explain the procedure in more details , if you see the code in listing [3-3] you'll see that we first read Status A register if bit 7 is 1 , that is the register = "10000000" = 0x80 , means that the CMOS is being updated and you can't use it now so the function busy waits and keeps reading the register until it's free . Then it offsets the index register to the desired register to read held in the parameter "value" , finally reads it from the Data port(0x71) and return it.

```
uint32_t cmos_get_reg(uint8_t value){  
    uint32_t val;  
    uint8_t update;  
    //check status  
    while(update == 0x80){  
        outb(RTC_STATUS_A,CMOS_INDEXPORT);  
        update = inb(CMOS_DATAPORT);  
    }  
    cli();  
    //get the value  
    outb(value,CMOS_INDEXPORT);  
    val = inb(CMOS_DATAPORT);  
    return val;  
}
```

Listing 3.3 cmos.c , reading cmos registers

Next function is *cmos_get_time* which is similar to *cmos_get_reg* which deals only with time -register 0:9- , The function is to be deprecated.

Finally , *cmos_set_power_stat* is responsible for supporting status B register which includes the power options and status It follows the same sequence as the previous functions but sets the register to the values mentioned in status B described in listing [3-4] .

```
/*those 3 sets the respective bit to zero so we mask with AND*/  
if(stat==STAT_RUN || stat==STAT_CAL_BCD ||  
stat==STAT_CAL_HR12)  
    New_Stat &= stat;  
else  
    New_Stat |= stat;
```

Listing 3.4 setting status B power options.

The rest of the registers are not yet used hence not yet supported.

[1]: The full detailed prots description can be found at
<http://bochs.sourceforge.net/techspec/PORTS.LST>

[2]: Refer to interrupts chapter for more details.

3.2 Video

Almost all new kernels and operating systems access and use video mode using the VESA/VBE interface to interface with the attached video card. however in CATernel we choosed to support legacy and old devices before supporting newer versions, therefore we choosed to use CGA(Color Graphic Adapter) to support video and console. There's few differences between CGA and EGA and VGA.

3.2.1 CGA(Color Graphic Adapter)

As mentioned earlier CGA is an old graphic adapter which we choosed to support first in CATernel. It supports two modes, Text mode and Graphic mode. for now we are only using the Text mode.

3.2.1.1 Text Mode

CGA has two text modes with a fixed character size.

40x25 Mode : Each character is 8x8 dots size and has up to 16 colors with resolution of 320x200.

80x25 Mode : It has the same character size and same color count, but with 640x200 resolution.

on CATernel we will support the CGA in text mode(80x25).the memory storage is two bytes of video RAM used for each character. 1st byte is the character code and the 2nd is the attribute. a screen might be 2000 byte or 4000 byte ($40*25*2$) , ($80*25*2$). and CGA's video RAM is 16Kb. and of course all what we can output is ASCII.

bit 0 = Blue foreground
bit 1 = Green foreground
bit 2 = Red foreground
bit 3 = Bright foreground
bit 4 = Blue background
bit 5 = Green background
bit 6 = Red background
bit 7 = Bright background (blink characters)

Listing 3.5 Character color attributes

To control screen cursor and lines two registers are used, Index and Data registers at address 0x3D4 and 0x3D5 respectively. If you took a look at video.c code you will find setters and getters for position. so position desired to read is supplied to index register which is 0xF for the first byte in the position and 0xE for the second byte.

and since we are working on a 80*25 then we wont need more than 4 bytes. for getting the value i read from the data register after specifying the index I want to read and I *inb* the value coming from the data port. for example:

suppose the cursor position is at 0x5a0, so what you will first get is the first byte of the position which is 0xa0. and as you might have noticed we do no operations on that. but on the second position you get 0x05. the operations is for mixing the first byte and second byte so they would make 0x5a0.

also you may notice that CGA_BUFF_OFFSET. which is the offset of the CGA video RAM in memory.

cga_putc:

what i do here is that i put the character i want to type on the screen to the CGA video RAM. and since we are working on 80*25 resolution bytes after the offset 0xB87D0 wont be written to screen yet they will be written to video RAM. this issue can be handled using memory trick like...move all binaries from 0xB8080 to 0xB87D0 80 byte backward which is the row size in 80*25 resolution. and then move the cursor position 80 place backward.

cga_putstr:

this function passes a pointer to an array of characters which are passed in a loop character at a time till we reach the null terminator character.

CGA Ports:

0x3D4	-	index register
0x3D5	-	data register
0x3D6	-	same as 0x3D4
0x3D7	-	same as 0x3D5
0x3D8	-	CGA mode control register -> bit 5 - blink register

```
| -> bit 4 - 640*200 High-Resolution register
| -> bit 3 - video register (if cleared the screen wont output)
| -> bit 2 - Monochrome
| -> bit 1 - text mode
| -> bit 0 - Resolution

0x3D9      -   Palette Register / Color control register
| -> bit5 - chooses color set
| -> bit4 - if set the characters show in intense
| -> bit3 - intense border in 40*25 and intense background in
            300*200 and intense foreground in 640*200
| -> bit2 - red borders in 40*25, red background in 300*200,
            red foreground in 640*200
| -> bit1 - green border in 40*25, red background in 300*200,
            red foreground in 640*200
| -> bit0 - blute border in 40*25, red background in 300*200,
            red foreground in 640*200

0x3DA      -   Status register
| -> bit3 - if set then in vertical retrace.
| -> bit2 - light pin switch off
| -> bit1 - positive edge from light pen has set trigger
| -> bit0 - 0 do not use memory.

0x3DB      -   clear light pen trigger
0x3DC      -   set    light pen trigger
```

Listing 3.6 CGA Ports

Console dependency:

The early console depends on the video driver and keyboard. On the console there's several wrapper functions for the video driver like *console_putc*, *putchr* and *console_clear*.

3.3 PS/2 Keyboard

Unlike video driver, CATernel's keyboard driver is similar to every kernel's keyboard support. However, since Keyboard is a serial device It uses almost a unified interface like any other PS/2 device. CATernel keyboard driver supports only one function which is a keyboard interrupt handler, Of course at this point a keyboard interrupt does not occur since CATernel busy wait on any I/O device.

Like any other device Keyboard commands are passed through I/O ports. In CATernel we barely make use of the keyboard controller, the only two operations are made through the status and data port. First operation is to check the keyboard data register, second one is to read that character from the data port.

```
0064 r      KB controller read status (ISA, EISA)
    bit 7 = 1 parity error on transmission from keyboard
    bit 6 = 1 receive timeout
    bit 5 = 1 transmit timeout
    bit 4 = 0 keyboard inhibit
    bit 3 = 1 data in input register is command
            0 data in input register is data
    bit 2      system flag status.
    bit 1 = 1 input buffer full
    bit 0 = 1 output buffer full
```

Listing 3.7 Keyboard Status port

There's three different scan code sets, CATernel uses the first scan code set. A scan code determine what key is pressed and three keyboard maps are provided, the first is a character map on normal case, second is a character map of keyboard on shift case and third is a character map of keyboard once a toggle button is on.

A keyboard interrupt handler reads the scan code and starts determining what key was pressed and then it is returned to interrupt issuer.

Console dependency

Console uses keyboard controller as an input device, a wrapper function called *console_getc* issues a keyboard interrupt and an index that char to a screen position(but char is not printed).

3.4 Intel 8259 PIC(Programmable Interrupt Controller)

As mentioned before on CATernel we tend to support legacy and old devices first, therefore we choosed to support Intel 8259 PIC before APIC(Advanced PIC) and IOAPIC are supported. Interrupts are the only way to manage execusion over x86 machines since Intel is an interrupt driven ISA. on old machines when only real mode was used PIC was the controller for interrupts and interrupts were handled by what is called vector store in an Interrupt vector table. An interrupt vector table is similar to the modern interrupt descriptor table except that IVT has the vectors already stored on the BIOS. PIC consists of two chip (Master/Slave) each has 8 interrupts which makes the total interrupts 16. an PIC interrupt is called IRQ(interrupt request) since an interrupt can be blocked when a higher priority interrupt handler is currently executing.

```

INT# 00 > F000:FF53 (0x000ffff53) DIVIDE ERROR ; dummy iret
INT# 01 > F000:FF53 (0x000ffff53) SINGLE STEP ; dummy iret
INT# 02 > F000:FF53 (0x000ffff53) NON-MASKABLE INTERRUPT ; dummy iret
INT# 03 > F000:FF53 (0x000ffff53) BREAKPOINT ; dummy iret
INT# 04 > F000:FF53 (0x000ffff53) INTO DETECTED OVERFLOW ; dummy iret
INT# 05 > F000:FF53 (0x000ffff53) BOUND RANGE EXCEED ; dummy iret
INT# 06 > F000:FF53 (0x000ffff53) INVALID OPCODE ; dummy iret
INT# 07 > F000:FF53 (0x000ffff53) PROCESSOR EXTENSION NOT AVAILABLE ; dummy iret
INT# 08 > F000:FEA5 (0x000fffea5) IRQ0 - SYSTEM TIMER
INT# 09 > F000:E987 (0x000fe987) IRQ1 - KEYBOARD DATA READY
INT# 0a > F000:FF53 (0x000ffff53) IRQ2 - LPT2 ; dummy iret
INT# 0b > F000:FF53 (0x000ffff53) IRQ3 - COM2 ; dummy iret
INT# 0c > F000:FF53 (0x000ffff53) IRQ4 - COM1 ; dummy iret
INT# 0d > F000:FF53 (0x000ffff53) IRQ5 - FIXED DISK ; dummy iret
INT# 0e > F000:EF57 (0x000fef57) IRQ6 - DISKETTE CONTROLLER
INT# 0f > F000:FF53 (0x000ffff53) IRQ7 - PARALLEL PRINTER ; dummy iret
INT# 10 > C000:014A (0x000c014a) VIDEO
INT# 11 > F000:F84D (0x000ff84d) GET EQUIPMENT LIST
INT# 12 > F000:F841 (0x000ff841) GET MEMORY SIZE
INT# 13 > F000:E3FE (0x000fe3fe) DISK
INT# 14 > F000:E739 (0x000fe739) SERIAL
INT# 15 > F000:F859 (0x000ff859) SYSTEM
INT# 16 > F000:E82E (0x000fe82e) KEYBOARD
INT# 17 > F000:EFD2 (0x000fefd2) PRINTER
INT# 18 > F000:B023 (0x000fb023) CASSETTE BASIC

```

Figure 3.1 IVT listed by Bochs

3.4.1 Master and Slave PIC

To handle all the 16 we must be able to index the right interrupt to the right PIC and when an EOI is issued we should be able to determine which PIC should handle the EOI. Master PIC and slave PIC have their own ports, Master has 0x20/0x21 and Slave has 0xA0/0xA1. the functionality is similar except for the interrupt types they handle.

- 1- Intel i8253 PIT
- 2- Keyboard
- 3- Video Interrupt
- 4- Serial port 2
- 5- Serial port 1
- 6- Fixed Disk
- 7- Floppy disk
- 8- Parallel printer
- 9- Real time clock (RTC)
- 10- Cascade Redirect
- 13- Mouse interrupt
- 14- Coprocessor exception
- 15- Primary Hard disk
- 16- Secondary Hard disk

Listing 3.8 PIC Interrupt requests (IRQs)

PIC I/O is a little different than former devices we dealt with in CATerel. PICs adopt a terminology called ICW(Initialization command word) and OCW(Operation command word). Intel 8259 manual defines ICW that It is used before any normal operation. as for OCW it can be executed at any point after initialization.

0020 w	PIC initialization command word ICW1
	bit 7-5 = 0 only used in 80/85 mode
	bit 4 = 1 ICW1 is being issued
	bit 3 = 0 edge triggered mode
	= 1 level triggered mode
	bit 2 = 0 successive interrupt vectors use 8 bytes
	= 1 successive interrupt vectors use 4 bytes
	bit 1 = 0 cascade mode
	= 1 single mode, no ICW3 needed

bit 0 = 0	no ICW4 needed
= 1	ICW4 needed

Listing 3.9 Port 0x20 flags for ICW1

3.4.2 PIC in protected mode

In x86 protected mode interrupts are only handled by the IDT and the IVT is omitted. such a case will put us in a problem whenever an IRQ is issued since it will conflict with Intel default 0~32 exceptions, therefore we won't be able to distinguish an exception from an IRQ. Luckily, Offsetting the IRQ indexes is a functionality can be performed through PIC ICWs. This is done through ICW2 in particular.

3.4.2.1 Initializing PICs

PIC initialization is done once the kernel have reached protected mode main-flow execution. It enables IRQs to be handled using IDT after offsetting them so CATernel would be able to use PIC in protected mode. this is done on four steps

First Step, ICW1 is passed a value with Flags ICW4 needed and ICW1 issued flags. Second Step, ICW2 is passed a value with the base offset desired to IRQs, and Since we do this step for both PICs the slave PIC base offset is passed as master PIC offset plus 8. Third Step, ICW3 is passed a value that holds one shifted by the interrupt pin of the slave PIC. and slave PIC ICW3 takes slave PIC index. Fourth Step, ICW4 takes a value with 8088/8086 mode flag set. However other flags are also active (0/1).

3.4.3 Masked Interrupts

Masked interrupts is another terminology an i8259 adopts, since it supports enabling and disabling interrupts through setting masks holds flags of desired interrupts and undesired. an interrupt could be disabled by setting its

corresponding flag. Interrupt masks are held in a PIC register called IMR(Interrupt Mask Register). each PIC has its own IMR since each PIC has its own type of interrupts.

```
0021 r/w  PIC master interrupt mask register
          OCW1:
            bit 7 = 0  enable parallel printer interrupt
            bit 6 = 0  enable diskette interrupt
            bit 5 = 0  enable fixed disk interrupt
            bit 4 = 0  enable serial port 1 interrupt
            bit 3 = 0  enable serial port 2 interrupt
            bit 2 = 0  enable video interrupt
            bit 1 = 0  enable keyboard, mouse, RTC interrupt
            bit 0 = 0  enable timer interrupt
```

Listing 3.10 IMR in Master PIC

Such a functionality gives the ability to disable the whole PIC by setting all flags on both master and slave PICs.

3.4.4 EOI End Of Interrupt

EOI must be used by an IRQ handler since it notifies the PIC that issued an interrupt that the interrupt handler has finished its execution, so the PIC should insert its blocked interrupt (if exists) to processor.

```
0020 w  OCW2:
          bit 7-5 = 000 rotate in auto EOI mode (clear)
                    = 001  nonspecific EOI
                    = 010  no operation
                    = 011  specific EOI
                    = 100  rotate in auto EOI mode (set)
                    = 101  rotate on nonspecific EOI command
                    = 110  set priority command
                    = 111  rotate on specific EOI command
          bit 4      = 0 reserved
          bit 3      = 0 reserved
          bit 2-0   interrupt request which the command applies
```

Listing 3.11 EOI using OCW2

3.5 Intel 8254 PIT(Programmable Interrupt Timer)

Due to the problem with RTC periodic interrupts [Appendix A] we had to support another device for time slicing execusion. The next modern device is PIT but still PIT is a legacy device. however, PIT was easier to program than RTC.

PIT has three channels. First channel is for counter divisor, second channel is for RAM refresh counter and third one is for issuing a beep on cassette or speaker on an interval. For the kernel clock we shall only use the first one which is the divisor of the frequency of the PIT to issue on interrupt on a subsequent interval. in CATernel we set this to issue 20 interrupt per second and handle those interrupts by scheduler.

4. Memory Management

On kernel level, Memory management and allocation is a very crucial and critical part that composes an efficient performance and protection. Dealing with memory on kernel level has to be very careful and clever since kernel doesn't have the user space luxuries like memory allocation errors.

Unlike Linux, BSD and Spartan kernels, CATernel doesn't yet contain any *Zone terminology* although there's different thunks of memory CATernel doesn't make use of the whole memory. Only two thunks of memory are used, Base and extended memory. and allocation on kernel level is done by manually allocating a memory unit which here is Paging.

4.1 Paging

A page is the smallest unit of memory in CATernel. Although a page is considerably huge comparing to processor smallest unit of memory which is a byte the Intel MMU deals with pages as the basic unit of virtual memory. IA32 has different types of paging methods called "Paging Modes", First is 32-bit paging which addresses 32-bit physical addresses to 32-bit virtual addresses, Second is PAE Paging which is used to translate 52-bit Physical addresses to 32-bit Virtual addresses and the third mode is called IA32e Paging which is used to translate same size of former physical addresses to 48-bit virtual addresses.

Since we're applying a minimal implementation for paging in CATernel we are only making use of 32-bit paging which has two types of data structures Page tables and page directories and two modes each has a different page size.

4.1.1 32-bit Paging structures

Two types of Data structures exist to index a page in 32-bit mode. It can be considered a two dimensional page array with the higher level is the page directory. Page directory contains page tables addresses and several flags, each member of page directory is called PDE(Page Directory Entry). Page table contains the addresses of physical pages and several flags, each member of page table is called PTE(Page Table Entry).

```
a 32-bit KByte Paging PDE would be like this.  
[0,11] Page Table Permissions and Ignored bits.  
[12,31] Page Table address.
```

```
a 32-bit KByte Paging PTE would be like this.  
[0,11] Page permissions  
[12,31] Page physical address
```

the permissions of the PDE is almost the same of the permissions of the PTE. PDE permissions:

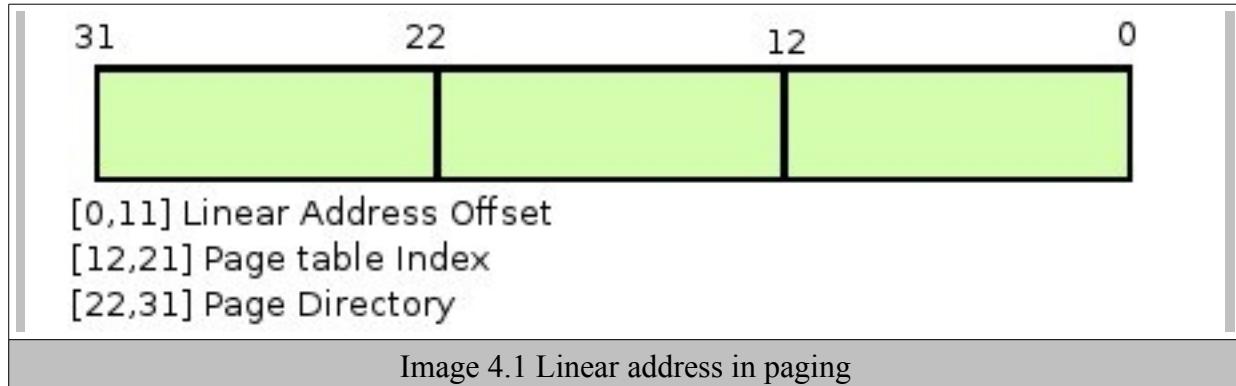
- [0] First bit must be always 1 which marks page as (Present), if it's not set the entry is ignored.
- [1] R/W permissions, If bit is cleared no write operations are allowed
- [2] U/S, it indicates whether the page/page table belongs to user or supervisor (ring 0,3). if it's cleared, it means it belongs to supervisor which forbids access from CPL =3
- [3] WT, Write through flag it indicates the memory type used to access this page either
 - write back caching or write through caching.
- [4] CD, it indicates if this page is cachable or not, if set it's not cachable.
- [5] A, Accessed flag refers to whether a software accessed this page or not.
- [6] D, Dirty flag is set if a software did a write operation to this page.
- [7] PS, Page size flags if set it means we're using 4MByte paging, if not it's 4KB paging
- [8] G, if set it means that the directory translation is Global. we shall refer to it later.

Listing 4.1 Paging structures entries

4.1.2 32-bit Paging Modes

First is 4Kbyte page 32-bit Paging, It uses two data structures to index a page (Page Directory - Page Table), Second type is 4Mbyte page 32-bit paging, which uses only one data structure to index a page (Page Directory). In CATernel the first mode is used since it will enable smaller pages hence, smaller basic memory units. A smaller memory unit has some advantages and disadvantages. having a 4Kbyte page as a virtual memory smallest unit will provide less fragmentation. on

the other hand, a bigger memory size will be used to store paging data structures.



A linear address in 4Kbyte 32-bit paging mode contains three fields, Linear address offset, Page table entry index and page directory entry index. in 4KByte Paging, and since we refer to 4Kbyte sized page which is 2^{13} it means we can offset with FFF into the page. from 0xFFFFF000 to 0xFFFFFFFF for example. the other [12,21] bits indicate the index of the page in the page table.

4.1.3 Initializing paging

Allocating/Clearing Page Directory

we use the boot time allocation scheme to allocate 4096 bytes of memory right after the kernel LOAD segment and clear it. and to provide access to page table for both user and supervisor to access the page directory by making it recursively reference itself when a virtual page table address is used. in our case, VIRTPGT, USERVIRTPGT. which lie in 0xEFC00000 , 0xEF400000 respectively. So for those page numbers/linear address to refer to page directory itself we map it to itself by this line.

```
pdr [ PGDIRX(VIRTPGT) ]=KA2PA(VIRTPGT) | PAGE_PRESENT | PAGE_WRITABLE;
```

and the index of this entry is 3BF. it looks in bochs like this.

```
<bochs:5> x/10x 0xf010befc
[bochs]:
0xf010befc <bogus+      0>: 0x0010b003    0x03ffd027    0x03ffc007    0x03ffb007
0xf010bf0c <bogus+     16>: 0x03ffa007    0x03ff9007    0x03ff8007    0x03ff7007
0xf010bf1c <bogus+    32>: 0x03ff6007    0x03ff5007
<bochs:6> █
```

Image 4.2 recursive page directory indexing

Pages data structure

What first comes in your mind if you need to detect whether there's a free page or not is to scan the page directory and table and detect free pages and search whether the page you want to map lies between those free pages or not. this would create a MASSIVE overhead. But a better way to do this is to create a linked lists, of Pages structures or whatever it might be called, it's not actually page structures but it's a (struct Page list) this is a simple backward linkedlist entry with a pointer to previous element and a value field, in our case this field is called ref which indicates how many pointers or procs refer to that page of course if it's allocated. if this ref field is 0 it makes this page free to use. and this is how it looks in memory.

```
<bochs:19> x/16x 0xf010d000
[bochs]:
0xf010d000 <bogus+      0>: 0x00000000    0xf010d00c    0x00000001    0x00000000
0xf010d010 <bogus+     16>: 0xf010d018    0x00000000    0xf010d00c    0xf010d024
0xf010d020 <bogus+    32>: 0x00000000    0xf010d018    0xf010d030    0x00000000
0xf010d030 <bogus+    48>: 0xf010d024    0xf010d03c    0x00000000    0xf010d030
<bochs:20> █
```

Image 4.3 pages list

Initializing structures.

after setting up the environment, we start to initialize page directories and tables. and since we need to be still operating after paging activation we need to put entries for both Kernel code and stack so after paging is active the same addresses would be translated to same physical position.

in steps,

we map the whole memory into pages and start filling out the free pages list. this can be done by a loop. but there's a memory we need to mark used that has the ACPI system calls and Memory mapped I/O [Section 6.1] plus the kernel code/stack segments are also in use, so those we need to mark used as well. after filling the free pages list. memory mapping procedure should be supported to map physical segments to virtual addresses in runtime. This is the `map_segment_page` function. to provide such a function we need other functionalities, Like the ability to find and create a page table at a specific position, and insert or remove a page. and allocate a page. a simple page allocation and freeing functions is to simply remove and add a page member to the free pages global list.

to insert and remove pages you need to be able to locate and create new Page dir Entries that are dependent on the virtual address. for this `x86_pgdir_find` function is defined, it takes the virtual address which's PDE is desired to be found or created. the function first checks if this entry already exist, if yes a PDE is returned. (it is referred in the code as PTE since it references the table, however it's called PDE in intel manuals) if not it is created if desired.

to remove or insert a page directory entry to a page directory other two functions were defined, removal function uses a lookup function to determine the existence of PDE, then it executes a detach function that determine whether there's processes that are still using this page or not, if not page is freed. then the pte is set to NULL or 0 and TLB is updated. to insert a page, the function first checks if there's a PDE/PTE referring to this page or the creatability of PDE/PTE referring to this page, if the page already is referred. if yes it's freed and reallocated. but we won't be using these functions ATM.

At this point mapping a segment of memory to virtual memory is trivial, for each page of the segment you insert a PDE and and a PTE that refer to this VA and an opposing PA, this makes the PTE random.

4.1.4 Triggering paging

to activate paging we need to first map the pages array so we can still access it via the same virtual address, also we need to map the kernel stack and the kernel code. a kernel code mapping for instance looks like this:-

```
map_segment_page(pgd, KERNEL_ADDR, 0x10000000, 0, PAGE_PRESENT | PAGE_WRITABLE);
```

this maps virtual address 0xF0000000 to 0xFFFFFFFF to the pages from 0 to 0xFFFFFFFF. let's do this manually we're not treating the 0xF0000000 anymore as segment base, but as a page number. which's PDE Index = 3C0 which means the entry's offset from pgdir base is $3C0 * 4 = F00$, let's check that in the bochs debugger.

```
<bochs:24> x 0xf010bf00
[bochs]:
0xf010bf00 <bogus+    0>: 0x03ffd027
<bochs:25>
```

Image 4.4 PDE in memory

It is noticeable that the PDE is marked Accessed since it's already executing. now let's read the Page table, the 0x03ffdXXX refer to the physical address of page table, and since it's in the $0 \sim 0x10000000$ kernel address space we'll just add a FXXXXXXX to it.

```
0x03ff0000 <bogus>          0>.px_dbg_read_linear: physical address not available for linear 0x03ff0000
<bochs:27> x/10x 0xf3ffd000
[bochs]:
0xf3ffd000 <bogus+      0>:    0x00000003      0x00001003      0x00002003      0x00003003
0xf3ffd010 <bogus+      16>:   0x00004003      0x00005003      0x00006003      0x00007003
0xf3ffd020 <bogus+     32>:   0x00008003      0x00009003
<bochs:28> █
```

as you see pages are sequentially ordered in page table as PTEs.

eventually, we load page directory address to cr3 and trigger paging on CR0. after we set the first PDE as the Kernel codes PE. since after paging the 1st PDE is loaded. then we reset segment table to full 4GB memory since we're able to convert 4GB of linear addresses to physical addresses. and a far jump is done to the same code it preserve the execution of the code after removing the page directory[0]. and for the CS to get updated.

4.1.3 Allocation

CATernel uses a weak memory allocation schemes, First is used on boot time which allocates memory heaps after kernel code section. Second is used by paging manager which allocates one page at a time. However, that's a subject to be looked in later, Zoning and slab allocators might be used.

4.2 Segmentation

In CATernel we use segmentation effectively on pure segmentation. and we only use the Global descriptor table. We use segmentation in re-mapping the kernel physical address into a virtual address, and maintain the 32-bit addresses. A global descriptor table is used for the OS level/ring 0 to locate its Code/Data/TSS segments. other tasks use the Local descriptor Table [LDT] which differs from a task to another. Like *Nix and WinXP we only use Paging to have protection and addresses virtualization. once paging is activated, we set GDT selectors base address to 0 with MAX memory limit. since we don't need longer addresses like protected mode. But, It's intended to have full power of Intel memory management like in OS/2.

5. Interrupts and system calls

5.1 Interrupts:

According to Intel Software developer manual Vol 3 [Chp.6 Interrupts and Exception Handling] an Interrupt can be defined as follows:-

Interrupts occur in random times during execution, and they are invoked by hardware. Hardware uses interrupts to handle events that are external to the processor such as request to service a device. software can generate an interrupt also by instruction INT <interrupt_number> also according to varying sources interrupt numbers break into the following.

-IRQ [Interrupt ReQuest]:

Interrupt requests are from 0 to 16

-Interrupts:

Interrupts are from 0~31, 8~16, 70h~78h

-PIC/Keyboard Ports

Programmable Interrupt controller and keyboard ports.

an Exception can be defined as follows:-

Exception occurs when a processor detects an error during trying to execute an instruction like devision by 0. the intel processor detects many error conditions including protection violation as page faults.

0	Programmable Interrupt Timer Interrupt
1	Keyboard Interrupt
2	Cascade (used internally by the two PICs. never raised)
3	COM2 (if enabled)
4	COM1 (if enabled)
5	LPT2 (if enabled)
6	Floppy Disk
7	LPT1 / Unreliable " spurious " interrupt (usually)
8	CMOS real-time clock (if enabled)
9	Free for peripherals / legacy SCSI / NIC
10	Free for peripherals / SCSI / NIC
11	Free for peripherals / SCSI / NIC
12	PS2 Mouse
13	FPU / Coprocessor / Inter-processor
14	Primary ATA Hard Disk
15	Secondary ATA Hard Disk
Listing [5-1] Standard ISA IRQs	

Int	Description
0-31	Protected Mode Exceptions (Reserved by Intel)
8-15	Default mapping of IRQ0-7 by the BIOS at bootstrap
70h-78h	Default mapping of IRQ8-15 by the BIOS at bootstrap
Listing [5-2] Default PC Interrupt Vector Assignment	

5.1.1 Dummy interrupts

For the sake of prototyping, There was a dummy exception and interrupts handler created under a generic name for both as Interrupt. However this still stands till

now, but few modifications was made which will be mentioned later on the document.

an interrupt occurrence causes an interrupt handler to be executed from the protected mode Interrupt Descriptor Table. the indexed function that matches interrupt number will be executed. To do that we must first initialize the IDT by filling the first 64 interrupt by a semi interrupt handler and filling the rest 196 interrupt with a dummy iret. Once an interrupt occurs the execution is altered to kernel space and starts executing the interrupt handler. Before an interrupt handler executes by default there's a stack frame storing the previous cpu state (the interrupt issuer) to be able to return to it, however this is not enough for us. In CATernel we define a soft cpu state structure holding all general purpose and segment registers and address space of the current environment.

Since there's no interrupts (not exceptions) that are yet handled through CATernel except for RTC periodic interrupts and system calls, any other interrupt once issued from user it goes through the interrupt mapping function and returns to user if no handler exists. yet there's illegal interrupts to be issued by user like the RTC which is used in scheduling.

5.1.2 Back end

-The gatedesc structure :

Our initial gate descriptor structure is more like the one in HelenOS, although I find it to be almost useless, since we won't be really using the args, reserved. And it's time consuming executing an assignment statement for every member of type,dpl and present bit. such a structure is implemented in both Linux/Minix as a one type_dpl_present field. We shall save this for later.

-CPU state frames :

to provide an informative and effective switching between caller and interrupt vector, cpu state frame holds info about variants of caller environment. such a frame in Minix for instance holds(vector, error code, eip, cs, eflags, esp, ss) in Linux all registers exist which is the same as HelenOS, although order is different.

5.1.3 Page faults

To this point we have a minimal user space, and poor scheduling mechanism and no signaling since we busy wait on resources. This simplifies the page fault handling for us in CATernel. Page fault handler checks if a page fault came from kernel mode, If so kernel is paniced. If it came from user space by user trying to access kernel space, the exception issuer proc is killed. If a page fault is done by user by jumping to wrong address (by instruction fetch) issuer proc is killed. If the issuer proc did issue a page fault by exceeding the stack, the proc stack is increased if it didn't reach max size for a user space stack.

5.2 System calls

A system call is a request for service that a program makes of the kernel. The service is generally something that only the kernel has the privilege to do, such as doing I/O. users do not need to concern themselves with system calls as they will be all done virtually in a library .

for a system call to happen a number of steps are followed:

|user code|->|Intermediate Library|->|Kernel code|

First , The program calls the function in the user library , this function is responsible for indexing and passing the arguments of the kernel function , then it issues and SYSCALL(0x30) interrupt

Before going into code there's an important table to mention , that is the *sys_call_table*

The table contains function pointers to the kernel level system calls handlers.

```
fnptr_t sys_call_table[] = { sys_exec ,
                            sys_fork ,
                            sys_printf
                            ....
                           } ;
```

Listing[5-3] System call table.

The first thing the call sets is the index for the correct function pointer in that table , indexes are defined in
include/sys.h

```
asm("movl %0,%%eax" :: "a"(S_PRINTF));    ;set call index
asm("movl %0,%%ebx"::: "a"(1));           ;for the function
asm("movl %0,%%ecx": :"=g"(str));        ;function argument
asm("int %0" : :"a"(0x30));              ;SYSCALL interrupt
```

Listing[5-4] printf.c prototype

after the function issues the interrupt ,and the interrupt handler finds that it's a SYSCALL interrupt , it'll call map_syscall function
and pass the current cpu_stat structre to it , the function will index the sys_call_table and call the function with the arguments in ecx register
and return the return value (error code) of the call.

```
s_errno= (sys_call_table [cpu_state->eax])((char *)cpu_state->ecx);
if(s_errno < 0 ) {return -s_errno;} //error code
else {return 0;}
```

Listing[5-5] mapping system calls

6. Process Management:

6.1 Loading Process:

Currently we're only support ELF formats , *elf_load_to_proc* reads the binary from disk and populates the proc structure with the process information and sets the entry point.

```
typedef struct proc {
    gpr_regs_t    gpr_regs;
    seg_regs_t    seg_regs;
    reg_t          eip;
    uint32_t       cs;
    reg_t          eflags;
    reg_t          esp;
    uint32_t       ss;
    uint32_t       proc_id;
    uint32_t       proc_status;
    pde_t          *page_directory;
    uint32_t       cr3;
    uint32_t       preempted;
    uint32_t       dequeqed;
    LIST_ENTRY(proc) link;
    LIFO_ENTRY(proc) q_link;
} proc_t;
```

Listing [6-1] Process structure

The function will remind you of kernel fetching, based on the binary offset it will seek it and read the disk's block into an *elfhdr* structure , which will then iterate through the headers to copy the entire file into memory , update the page table and the CR3 register as an initialization to the user environment.

6.2 Scheduling

Scheduling is the process of organizing the context switching between processes in order to achieve multi-tasking. In prototype1 we're using a simple LIFO Time sharing Round robin (Last In First Out) scheduling algorithm.

For a quick recap , LIFO refers to the way items stored in a data structure are processed. The last data to be added to the structure will be the first data to be removed. LIFO mechanisms include data structures such as stackss. A LIFO structure can be illustrated with the example of a crowded elevator. When the elevator reaches its destination, the last people to get on are typically the first to get off , the same thing applies to processes , the last process added to the queue is the first process to be taken out of the queue so that another one would take it's place.

another important concet in scheduling is context switching , a "context" is a virtual address space, the executable contained in it, its data etc.

A "context switch" occurs for a variety of reasons - because a kernel function has been called, the application has been preempted, or because it had yielded its time slice.

A context switch involves storing the old state and retrieving the new state. The actual information stored and retrieved may include EIP, the general registers, the segment registers, CR3 (and the paging structures), FPU/MMX registers, SSE registers and other things. Because a context switch can involve changing a large amount data it can be the one most costly operation in an operating system.

since Resource management is not yet implemented we currently have 2 queues , a ready queue and a running queue after a quantum of time passes .

the schedule function checks the running and ready queues , last process in the running queue will be poped and replaced with the last process in the ready queue , which is to be scheduled then a context switch to this process occurs with the *switch_address_space* function.

```

schedule(void)
{
    uint32_t idx= 0;
    proc_t *proc, *pproc;
    if(!LIFO_EMPTY(&running_procs))
    {
        pproc = LIFO_POP(&running_procs, q_link);
        printk(" [*] Proc running: %d\n", pproc->proc_id);
    }
    else
        printk(" [*] No running procs\n");

    if(!FIFO_EMPTY(&ready_procs))
    {
        proc = FIFO_POP(&ready_procs);
        printk(" [*] Ready proc: %d\n", proc->proc_id);
    }
    else
    {
        printk(" [*] No ready procs found\n");
        proc = pproc;
    }

    if(!LIFO_EMPTY(&running_procs))
        FIFO_PUSH(&ready_procs, pproc);

    LIFO_PUSH(&running_procs, proc ,q_link);

    printk(" [*] Scheduling to process: %d\n", proc->proc_id);

    switch_address_space(proc);
}

```

Listing[6-2] proc.c

switch_address_space fools the CPU in order to switch to another address space , it needs to reset the stack top to point the the new process structure , set the cpu

state to the process' value the issues an iret.

```
void
switch_address_space(proc_t *proc_to_run){
    proc_to_run->seg_regs.fs = 0x23;
    proc_to_run->seg_regs.es = 0x23;
    proc_to_run->seg_regs.gs = 0x23;
    proc_to_run->seg_regs.ds = 0x23;
    write_cr3(proc_to_run->cr3);
    asm volatile("movl %0,%%esp": : "g" (proc_to_run) : "memory");
    asm volatile("popal");
    asm volatile("popl %gs\ncpyl %fs\ncpyl %es");
    asm volatile("popl %ds");
    asm volatile("iret\n5:\n");

    while(1);
}
```

Listing[6-3] Switching to process address space.

[Prototype Two]

1. Design Model

1.1 Monolithic vs μKernel

Lately in CATernel first prototype has been put to and end since we reached a crossroads point. We had make up our mind to the overall design of the kernel. We had several Options: 1- Monolithic kernel, 2- ExoKernel, 3- μKernel.

Of course the main dilemma would be between Monolithic and μKernel since most Exokernels made are for research purposes and not [rel life] daily kernels that an Operating system might use. However, a minimal Exokernel version of CATernel might be supported anyway since It won't be hard to implement.

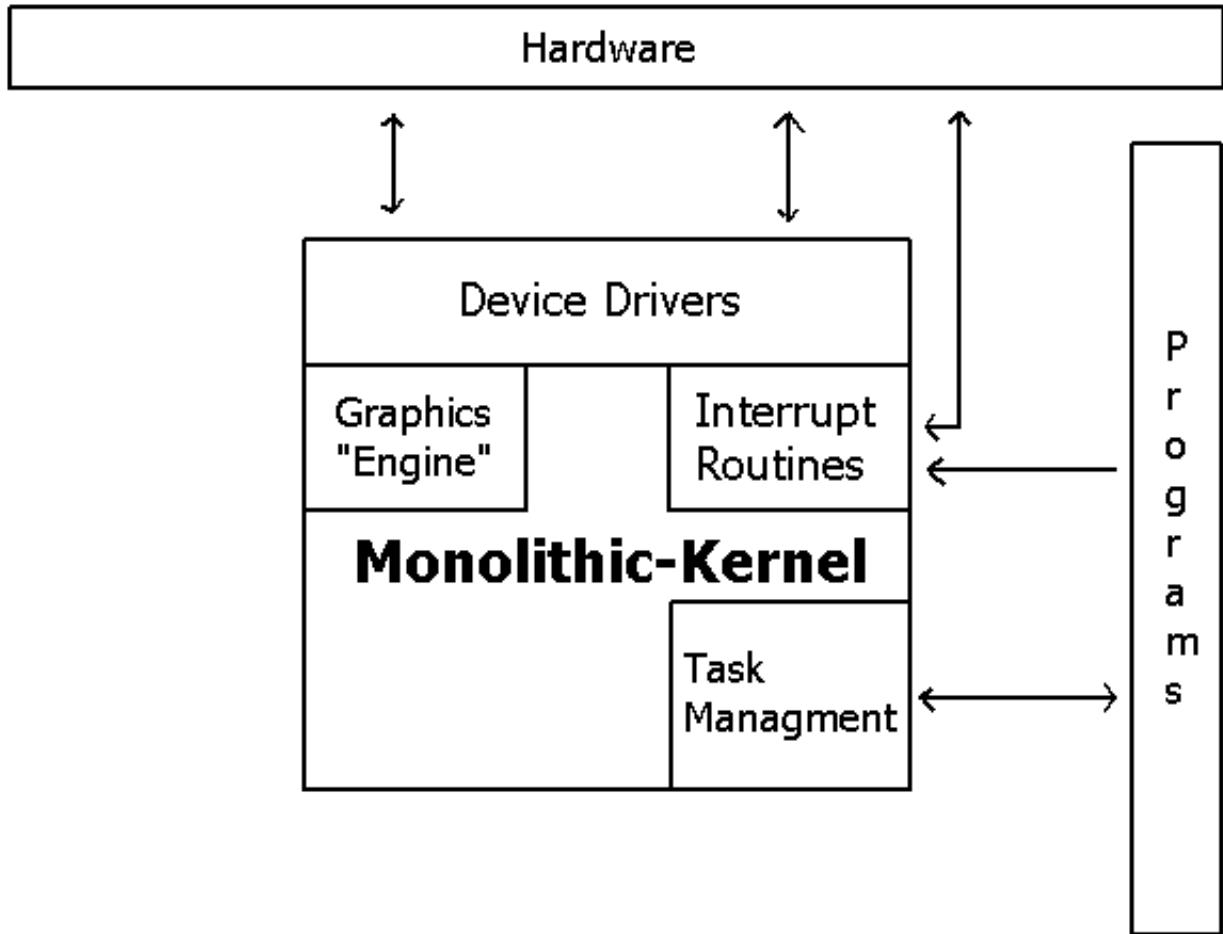
Here, we only stand between the Monolithic design and Micro design. those two designs in particular has been a huge dilemma for years, It has began with the famous Tanenbaum - Torvalds debate. But, Let's pause here for now and start listing the main differences between the Micro and Monolithic designs.

1.1.1 Monolithic Kernel design

Normally a kernel consists of several entities mostly called "services", Like Process Management, File system management, Memory Management. A monolithic design is simple and old, but however effective. In a monolithic kernel all the kernel services are contained within one program. an example to that is the old UNIX of course, Linux and freeBSD. most of the time this (containing all services in one program) results in a huge code base which will result into hard debugging and bug tracking. and since kernel components only connect with each others in execution flow It creates a huge dependencies for kernel components that are hard to track too, at this point I recall the name Andrew Tanenbaum called the monolithic kernel ("The Big Mess"). However, Monolithic kernels have advantages that might distinguish it from a microkernel, such as:

- 1- It takes less code to write a monolithic kernel
- 2- monolithic kernels have relatively fewer bugs, since only way to context between services is using execution flow and no external means are provided.
- 3- They are also relatively fast since it is only one executable entity.

In abstract form this is how monolithic kernel looks like:



Since all services of a monolithic kernel lays into the kernel executable, system calls are used to achieve privileged operations from user mode. System calls in monolithic kernels are much more than those in a Micro kernel since all privelged services lays into kernel space. and system calls are basically interrupts, hence a code executing under a monolithic kernel will issue more interrupts than the same code running under an Exokernel. However, Interrupts can hardly be an overhead since the scheduler can issues interrupts -sometimes- hundreds of times per second on normal Interactive operating systems.

Monolithic kernels have some disadvantages too, some of them crucial and some

tolerable:

- 1- As mentioned before, monolithic kernel contains huge relations and dependencies. Thus, a failure of a part of the kernel will result in a major failure in the whole kernel. Thus less reliability.
- 2- Also monolithic kernels are hard to port, since -almost- the whole kernel needs to be re-written to support a different architecture.

1.1.2 µKernels (microKernels)

are more modular than monolithic kernels, since most of the kernel services are moved to user space. Like, Networking and File systems management. And (as the name states) the real kernel is very minimal containing the Memory management and process management services for instance. The services that are runnable in user space are called servers (will show why later) and they use the real kernel to get access to hardware. a typical microKernel provides an abstraction for hardware which will make servers interactions easier and will lessen the porting pain. Surprisingly, µKernels result into more number of system calls than monolithic kernels, since all services will have to access kernel space to perform privileged operations since most of the services rely on devices.

Unlike monolithic kernels, µKernels come up with a service that monolithic kernel might not even need (not realistic kernels of course) which is IPC(Inter process communication). Inter process communication allows running processes to communicate and send and receive data. That is why kernel entities in user space are called servers, because user programs follow the client server model. they communicate with the kernel service to perform an operation and the server does that for them. µKernels however have some advantages such as:

- 1- µKernels are reliable, that if a service crashed it can be fixed or altered. even if it cannot be fixed the kernel would still be running.
- 2- Testing is easy on µKernels since loadable software doesn't require you to reload the whole kernel.
- 3- Easier to maintain.
- 4- µKernels can be easier to adapt for working under a distributed system.

µKernels disadvantages:

- 1- Very complicated process management.
- 2- IPC bugs can bring the whole system down
- 3- Since more µKernels are more abstract, execution time of software is relatively large.

1.1.3 Modern examples:-

Monolithic kernels years ago were limited to UNIX and BSD and some other minimal kernels. But, Linux came out of nowhere and proved the effectiveness of monolithic kernels after being considered outdated. So, the only examples I can give about modern monolithic kernels are Linux and FreeBSD.

By then µKernels were the hub of interest and most operating systems researchers and hobbyists starts considering the µ design for their kernels, Thus most modern kernels are micro. Such as, MINIX , GNU Mach/Hurd, SPARTAN. Also Windows NT can be considered micro, but It is called hybrid since it is not fully micro.

The Tanenbaum-Torvalds debate:

On the debate, Tanenbaum explicitly expressed that Linux was outdated due to the old monolithic design it uses. And he admitted that microkernels can be slower than monolithic kernels, but with some optimization they could just as fast as monolithic kernels. Also a monolithic design limits and makes portability to different machines harder (as stated above).

To that Linus did agree that micro is a better design than monolithic and portability on monolithic kernels might have issues. and stated that Linux is more functional and stable than minix.

several famous researchers joined the conversation such as:

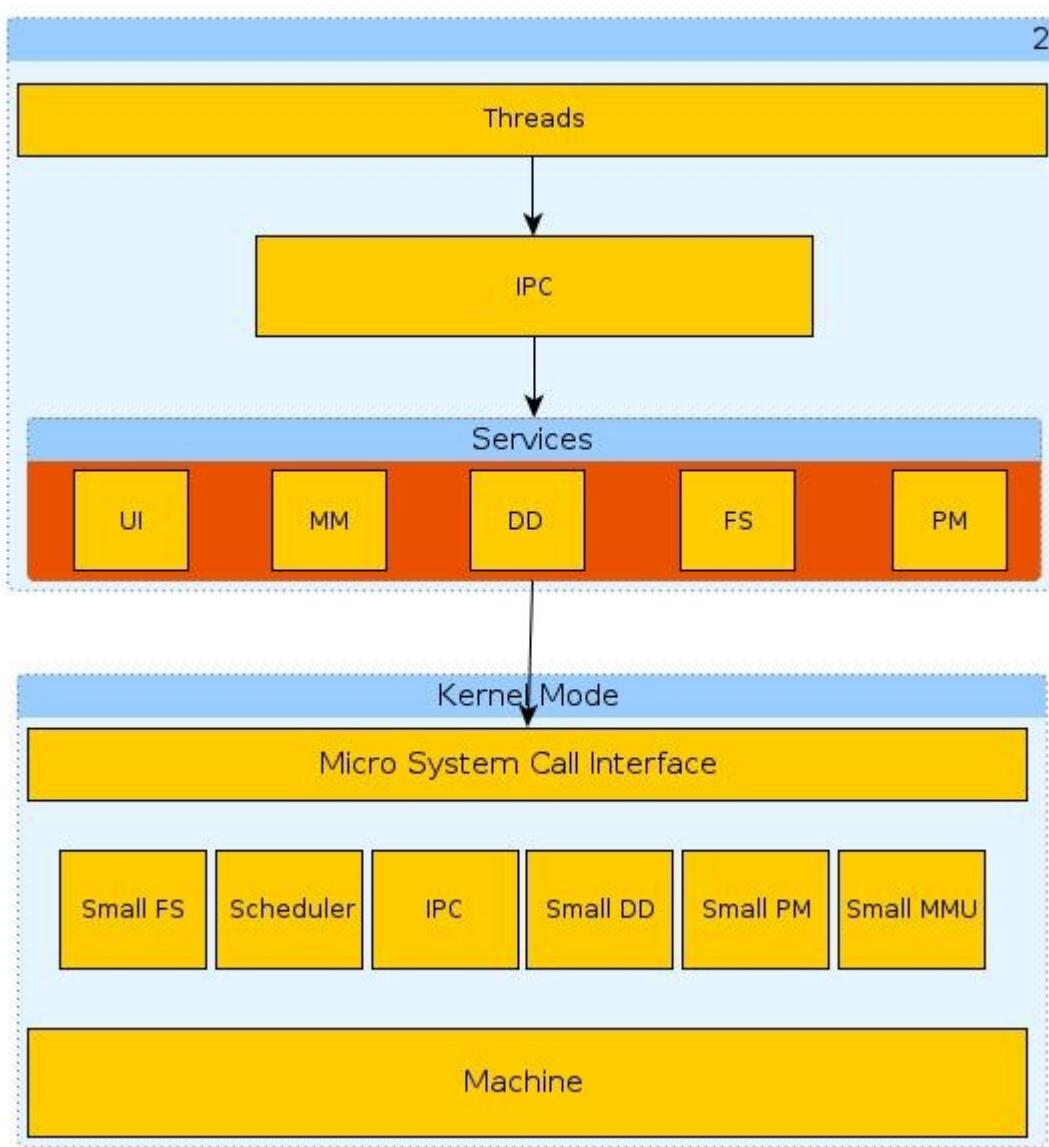
Ken Thompson

Micheal Kufman

<http://oreilly.com/catalog/opensources/book/appa.html>

However, For CATernel We've favored the Micro design, Although monolithic is easier to implement. but for future issues micro kernels are easier to maintain and debug. This kernel is all about researching after all, No real work involved.

1.1.4 CATernel design model



We've established this design as our generic primary design with small resource managers at the kernel level which are enhanced by more detailed and specific manager on user level to achieve micro design of CATernel. With Small file system manager, Scheduler, IPC, Small device drivers manager, small process manager and small memory management unit on kernel mode which are interfaced by few system calls. With the more specific and dedicated managers established as services on user mode. Threads can only communicate with those servers using IPC unlike monolithic convention which uses system calls.

2. Supporting SMP

Since the overall kernel design is determined, we need to work more on basic services such as multitasking and memory management. Symmetric Multi-Processor Management, at the moment we are only able to multi-task over one processor. Of course, we will be using Intel facilities to be able to parallelize over multiple processes. There's challenges faced when it comes to parallelizing execution over a centralized Data Bus and memory, Such as:

- 1- Memory Coherency and (making sure that only one processor get acces to a specific memory address at a time).
- 2- Cache consistency, Since some caches are shared (e.g. L2/L3) corrupted data should be avoided by a processor after being written by another
- 3- Distribute interrupt handling amongst processors.
- 4- Avoiding randomized memory accesses.

2.1 Memory Coherency

Memory coherency can be simply put as follows. It is preventing a cpu from accessing a memory thunk aligned to (8/16/32/64) that is being used by another cpu(former values differ on different architecture). once a value is locked the data bus is automatically locked, so locking here is basically spin-locks. ia32 supports memory locking using atomic operations via several ways :-

2.2 Automated locking

Locking a memory address is automatically carried out on several situations

- 1- executing xchg instruction that reference memory
- 2- Updating segment descriptor
- 3- Updating page directory
- 4- Acknowledging interrupts

the last 3 cases do not really interest us, but the first instruction can be used to make a software locking to memory address. such a locking mechanism force following memory operations to be atomic. An example for that is the classic unix and very early Linux. a very basic mutual exclusion was implemented by merely exchanging the to be locked variable with a new value. Such examples are obsolete and old. With Multiprocessors architectures showing up another problem arises.

2.3 Memory ordering

For the sake of performance optimization, more modern processes adopted a new memory ordering(order of loads and stores issued) models. Instead of strong ordering which executes loads and stores with the order they are in the executable code, Modern processors use different memory models where loads and stores don't need to necessarily done in order they're in the executable.

Due to that fact, busy waiting on a lock might go wrong. since loop is the first in the executable then modifying the lock comes second we suppose we're working on strong ordering modeled memory access. However, this can go badly wrong by obtaining the lock before executing the loop for instance, Maybe even skipping obtaining lock and starting executing critical region. Thus, cpu needs to be instructed to execute a thunk of code's memory loads and stores in the order they're in the executable. And luckily this is what 'mfence' instruction does. Memory ordering is a strict issue since it all gets messy with processors evolution. 486 and Pentium for instance had a less strict memory ordering model than strong ordering but still most of the time it would use strong ordering. P6 family however almost dropped the strong ordering model which Intel name "write ordered with store buffer forwarding". a premature implementation for the spinlocks would look like this at this point

lock:

```
popl %eax  
movl (%eax), %ebx  
test %ebx,%ebx  
jnz lock  
incl %ebx  
xchgl %ebx,(%eax)  
mfence  
ret
```

unlock:

```
popl %eax  
movl $0, %ebx  
xchgl %ebx,(%eax)  
mfence  
ret
```

Listing 2.1: premature spinlock

At this point Intel manuals carry on from memory instructions ordering to the relation between memory ordering and string operations and its atomic property under multiple processors and interrupts, But this is no use to us at the moment since address spaces forbid interrupts to use user stack. Moreover, generic instructions serializing is a necessity. An example mentioned in Intel manual vol.3 shows how critical instruction serialization could be crucial at some points, This example says: suppose we're switching to protected mode under a multiprocessor environment. we need all real mode instructions to be executed before protected mode is triggered. This is where serializing instructions comes in which makes sure that all instructions before it are executed before the serializing instruction is executed (e.g. CPUID)

Multi-Processor Initialization

Before mentioning the way Multiprocessing is initialized over an intel machine, we need to support and point out cpuid instruction and collecting information about the machine since we'd be having different initialization models for every machine.

Anyway, MP initialization is done by a MP protocol called Multiprocessor Specification Version. It supports controlled booting of multiple processors, It allows all IA-32 processors to be booted in the same manner and it can boot a system without defined boot processor. This MP protocol defines two types of processors which are bootstrap processor and application processors. after power one processor is selected as bootstrap processor and performs the booting up routines. Other processors are set to application processors.

Bootstrap processor stars by setting up the global data structures and initializes other application processors, following that is the Operating system code execution. when machine is powered up, Application processors wait for an IPI signal from the bootstrap processor and then performs APIC initialization procedure and put itself in halt state.

Before initializing processors, cpu information should be collected in order to perform a right initialization sequence. such information collection should be carried out after memory initialization on the BSP since values like global cpu use

memory component to allocate space for itself. however we can still store it in absolute address. and we might need that in parallelizing kernel boot up.

CPU information is found in the floating point structure which is usually stored in one of Four places:

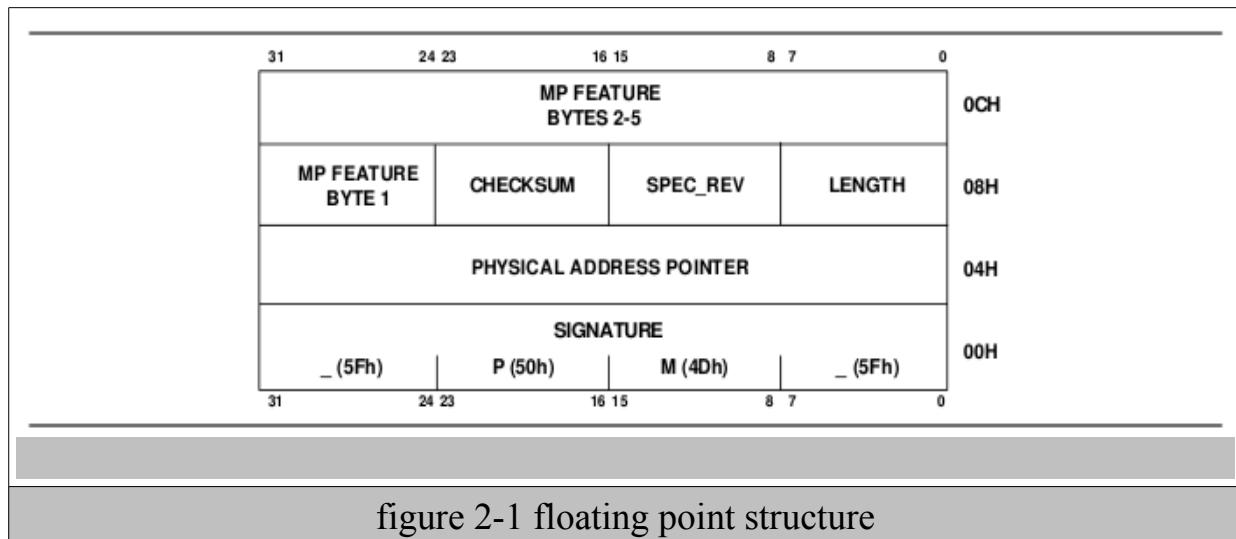
- (1) in the first kilobyte of the extended BIOS data area (EBDA)
- (2) the last kilobyte of base memory,
- (3) the top of physical memory
- (4) the BIOS read-only memory space between 0xe0000 and 0xfffff.

we'll need search these areas for the "_MP_" signature which denotes the start of the floating pointer structure.

Absence of this structure indicates that the system is not MP compliant ; if not the system will continue with UP setup

```
uint32_t length[2] = { 1024, 64 * 1024 };
addr[0] = (uint8_t *) PA2KA(ebda ? (uint32_t)ebda : 639 * 1024);
for (i = 0; i < 2; i++) {
    for (j = 0; j < length[i]; j += 16) {
        if ((*((uint32_t *) &addr[i][j])) ==
            FS_SIGNATURE) && (fps_check(&addr[i][j]) == 0)) {
            fs = (fpstruct_t *) &addr[i][j];
            goto fs_found;
        }
    }
}
```

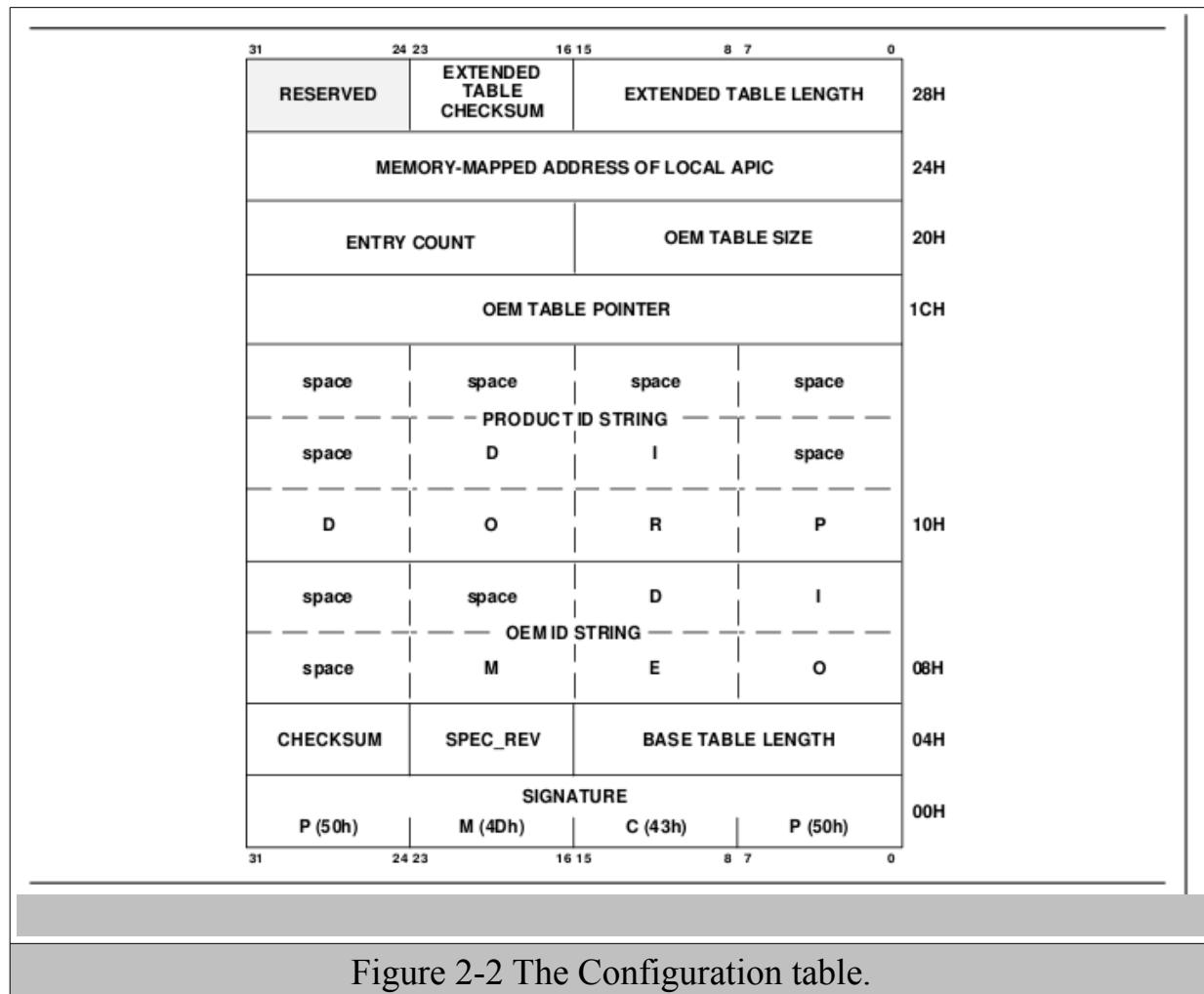
Lisiting 2.2: arch/x86/mp/smp.c - find_set_fps



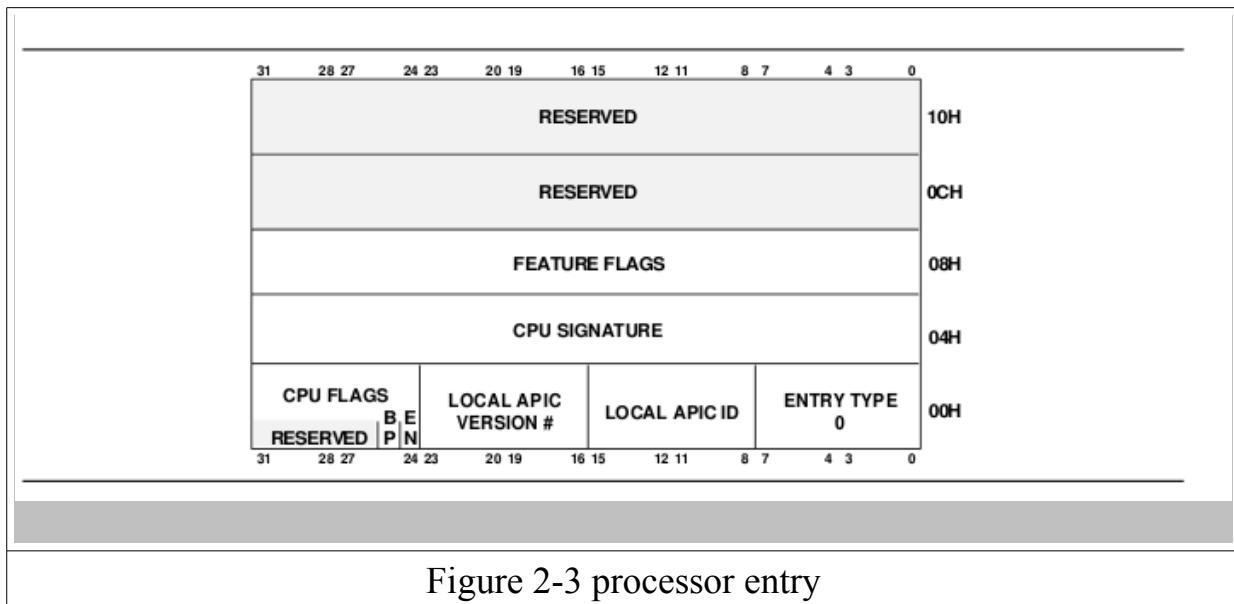
Next step is to read the Configuration table's address form the floating point structure which will include all cpu's information

```
"ct=(ct_hdr *)PA2KA((uint32_t)fs->config_addr);"
```

The configuration table consists of a header , base table and an extended table.



with help of the CT header -arch/x86/mp/smp.c - ct_entries() - , you'll be able to locate the processor entries and read them
most of the fields will come in handy later , refer to smp.h to see the full structure of different base entries .



now that we have all needed cpu info , we can finally signal them to wake up by sending an INIT IPI through the LAPIC in 0xfee00000

we're sending the IPI based on the processor's APIC ID , ofcourse it's a bad idea to send an INIT IPI or SIPI to boot strap since it's already up
so we make sure we check the bootstrap flag in the processor's entry we parsed earlier.

```
void apic_init_ipi(uint8_t lapicid)
{
    icr_t icr;
    icr.lo = lapic[ICR_LOW];
    icr.hi = lapic[ICR_HIGH];
    icr.delivery_mode = ICR_INIT;
    icr.dest_mode = PHYSICAL;
    icr.level = LEVEL_ASSERT;
    icr.trigger_mode = TRIGMOD_LEVEL;
    icr.shorthand = NO_SH;
    icr.vector = 0;
    icr.dest = lapicid;
    lapic[ICR_LOW] = icr.hi;
    lapic[ICR_HIGH] = icr.lo;
}
```

Listing 2.3: arch/x86/mp/apic.c - apic_init_ipi

Intel's specification shows that we need to wait for 20 us till an IPI is successfully delivered , then wait for 1000 us for a processor to wake up In order to send the Startup IPI which is responsible for making the AP jump to custom bootup code to set it up , 200 us are needed to deliver this IPI ;

After making sure the INIT-IPI was delivered , The Application processors will wake up and wait for a S-IPI (Startup IPI) which will include the pointer to the boot up stub needed for the processors initialization , where SMPBOOT_START is the Physical address of the boot up stub shifter by 12 (e.g 000AA000 == AA) and the processor will reverse this operation to reach the stub.

```
icr.lo = lapic[ICR_LOW];
icr.vector = (SMPBOOT_START) >> 12 ;//trampoline
icr.delivery_mode = ICR_SIPI;
icr.dest_mode = PHYSICAL;
icr.level = LEVEL_ASSERT;
icr.shorthand = NO_SH;
icr.trigger_mode = TRIGMOD_LEVEL;
icr.dest=ALL_X_SELF;
lapic[ICR_LOW] = icr.lo;
    //Wait 200 us
delay(200);
apic_read_errors();
```

Listing 2.4 arch/x86/mp/apic.c - apic_s_ipi

To Sum things up , Here's A diagram for the initialization sequence.

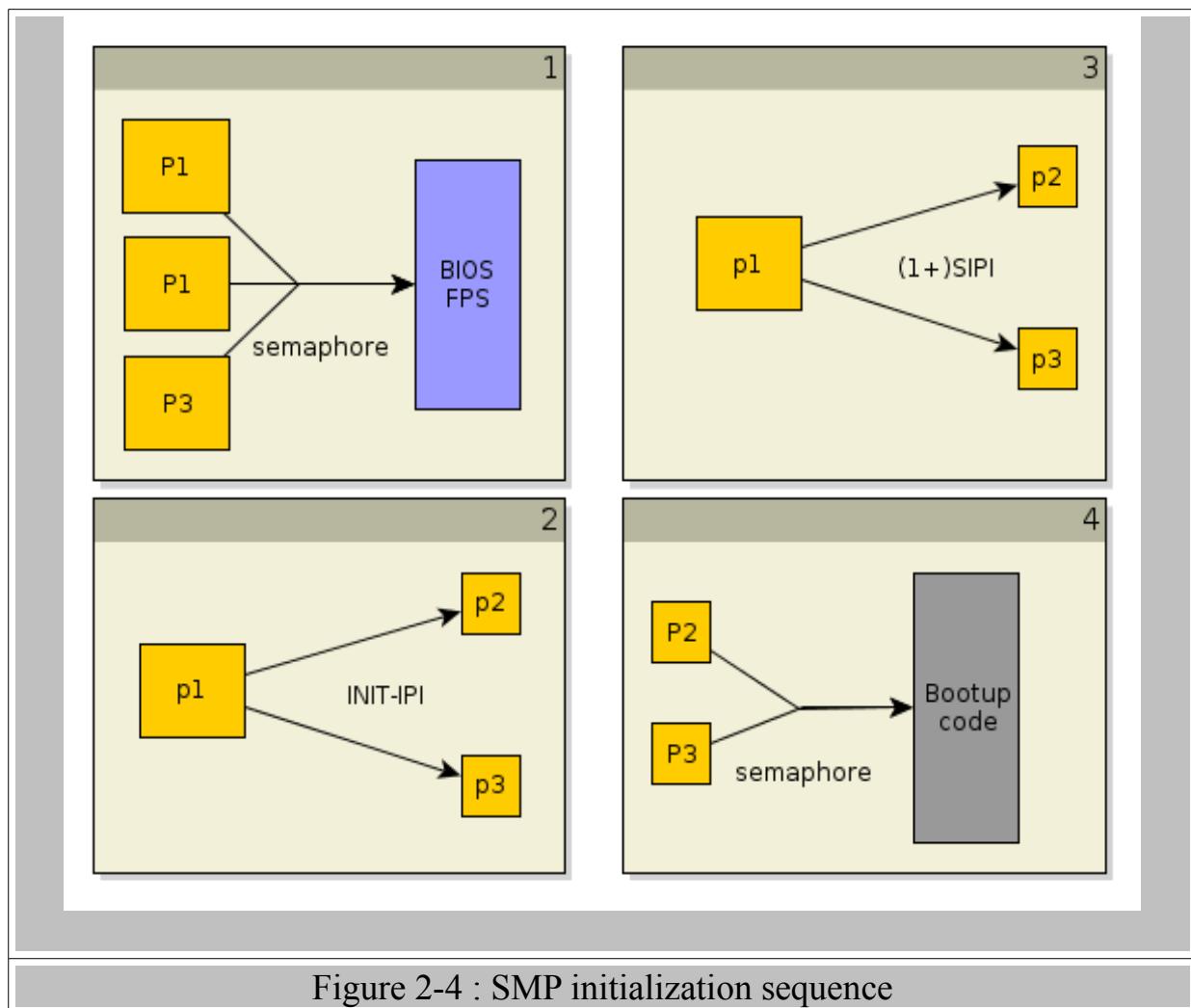


Figure 2-4 : SMP initialization sequence

3. Synchronization

After the decision to change the CATernel design into microkernel, It became inevitable to implement early synchronization primitives and IPC scheme to convert kernel components into services.

The Basic synchronization primitive is considered the spinlock mentioned in the "Supporting SMP" section. This synchronization primitive is used in active synchronization in kernel only. other type of synchronization primitives is passive synchronization, this is the most used type which includes race conditions.

In the first prototype a minimal structure of the process descriptor was implemented which was sufficient for the first prototype phase. However, moving into multiprocessing environment and to be able to provide a concurrent back-end even with a uniprocessor plus the nature of the kernel design obligated us to extend process structure. The old process descriptor structure contained :-

- 1- Process context
- 2- Process Id
- 3- Process Address space
- 4- Scheduling states
- 5- Pointers Proc List and running Procs LIFO

At this point also only three process states were supported

- 1- Process ready to run [RUNNABLE]
- 2- Process descriptor is undefined [EMPTY]
- 3- Process is not ready [NON_RUNNABLE]

Of course with such a structure processes are not able to support events, and doesn't even support them at all.

3.1 Supporting race conditions

A concurrent environment is unable to work in a consistent and proper way without race conditions of course. for a process/service to be able to process locking/unlocking events of a synchronization primitive it must either busy wait or block. However busy waiting is time wasting like in spinlocks and it probably would lead into deadlocks with prioritized processes design.

Blocking a process is simply done by two components, Scheduler and a waiting

queue. Process simply waits on race conditions locks or IO events with either a state of those states (wait interruptible – wait uninterruptible).

However There's difference between Synchronization process communication, Inter-service/kernel communication and Standard Inter-process Communication.

3.2 Synchronization Communication

Race conditions are somehow an IPC mechanism from semaphores to message queues. However there's different implementations for such synchronization mechanisms and specially using waiting queues. for example Linux kernel makes no use of waiting queues but only timers, Mach makes use of a single waiting queue and SPARTAN makes use of a multiple waiting queues with also timers same as linux only renamed to timeouts.

At this point the [Timers] implementation seems convenient. Possible ways to implement a timer are by the RTC or APIC timer, of course with having a nearly SMP environment apply an RTC timer seems inconvenient. APIC timers however also has one-shot/periodic/tsc timers. a one-shot timer might be convient also. Another way implementing timers is by setting a timer relative to Scheduler timer interrupts with the timer interrupt as a one basic unit of timer.

A pre-mature implementation for a synchronization based on timers and waiting queues is to update timers on each tick with extending implementation of timers via waiting queues and same for synchronization primitives.

3.3 Optimal Implementation

When it came to implementation various designs and schemes came up to my mind (personally) not to mention other implementations adopted by other kernels.

Semaphores

implementing the semaphore's P&V with Dijkstra's algorithm without using spinlocks isn't practical. for example to use a semaphore which only contains the integer value of the semaphore and ups and downs which directly puts an executing entity into a generic waiting process list makes it hard to determine which procs/threads are waiting for this specific resource/primitive and wake them up. a hack would be to make a list of pointers or indexes of these procs into the generic waiting list in the semaphore itself, but this would cause contention over the generic waiting list, another memory unfriendly implementation is to

make a waiting list per resource.

Within a Multiprocessor environment old irq_enable/disable won't work to keep the an up/down operation atomic. thus a spinlock implementation is a must. of course disabling all processors irqs is nonsense.

3.4 Case Studies

GNU Mach

GNU Mach implements wait queues implicitly into a thread structure, with only one type of waiting [THREAD_WAIT]. The thread state is changed into wait so the scheduler won't pick it the next time it is invoked. However such implementation is minimal compared to the Linux implementation.

Minix

Minix implementation to IPC is rather complicated, actually very complicated. it doesn't make use of any processes blocking or suspension and ipc is done via a separate server not as a global entity. This implementation is similar to the System V semaphores implementation. This implementation is for generic IPC purposes and Svr5 Standardization

Linux - POSIX

The way Linux implements locking and IPC waiting is by spinlocking using a unified locking way called ipc_lock. so the lock is being polled on, scheduler however makes use of this waiting into blocking a thread implicitly using the Linux waiting queues in the form of semaphore queues. However This implementation at ipc/sem.c is a System V semaphore which we don't really need at the moment. since it's for IPC Purposes not synchronization.

another implementation of the semaphore is used in kernel/semaphore explicitly makes use of waiting and task state.

SPARTAN

the SPARTAN waiting queues are very similar if not identical to the linux waiting queues, however its semaphores makes use of waiting queue explicitly. Which makes the SPARTAN semaphores and ipc more effective and fast than any other.

4. Time Management

1-Delay function

The delay function was implemented to make the cpu waste cycles for a few microseconds without being interrupted , this approach came in handy in booting up the Application processors "refer to Supporting SMP"

The delay function is hardware based , It uses PIT (i8254_calibrate_delay_loop) to calibrate the cpu to figure out how many loops are executed in 1us, which is what we will refer to as the "delay_loop_const"

The delay loop is pointless , it's only a waste of cpu cycles

```
{
    asm volatile ("movl %0 ,%%ecx\n\t"
                 "0:lahf\n\t"
                 "dec %%ecx\n\t"
                 "jnz 0b" :: "a"(t));
}
```

Listing 4.1: asm_delay_loop

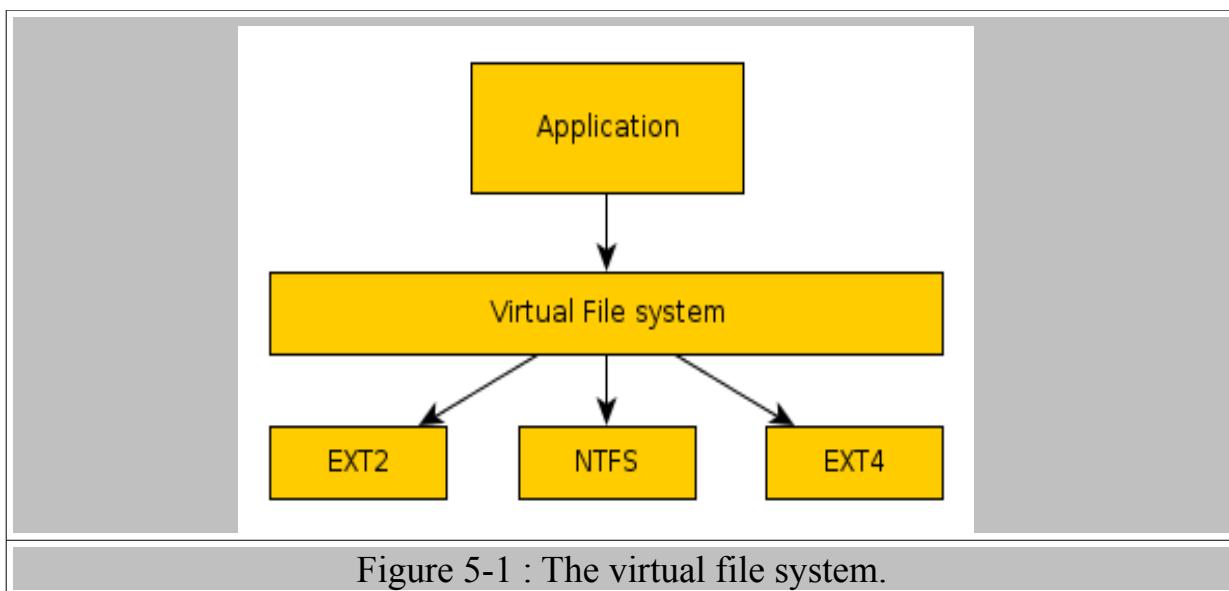
so now we can simply make the cpu wait for X microseconds simply by multiplying the time we need and the delay_loop_const .

```
void delay(uint32_t microsec)
{
    cli();
    asm_delay_loop(microsec * delay_loop_const);
    sti();
}
```

Listing 4.2: delay()

5. Virtual File System

A virtual file system is not a disk or a network file system , It's rather an abstraction that the operating system provides in order to unify different file system calls , For example rather than calling ext2_open you can use the virtual file system read call instead thus allowing any calls to be file system independent



A simple way to accomplish this is to have the node structure store specific file system function pointers which are then called by the kernel ,currently we're supporting :

Open : Called when a node is opened .

Close : Called when the node is closed.

Read : Called on read.

Write :Called on write.

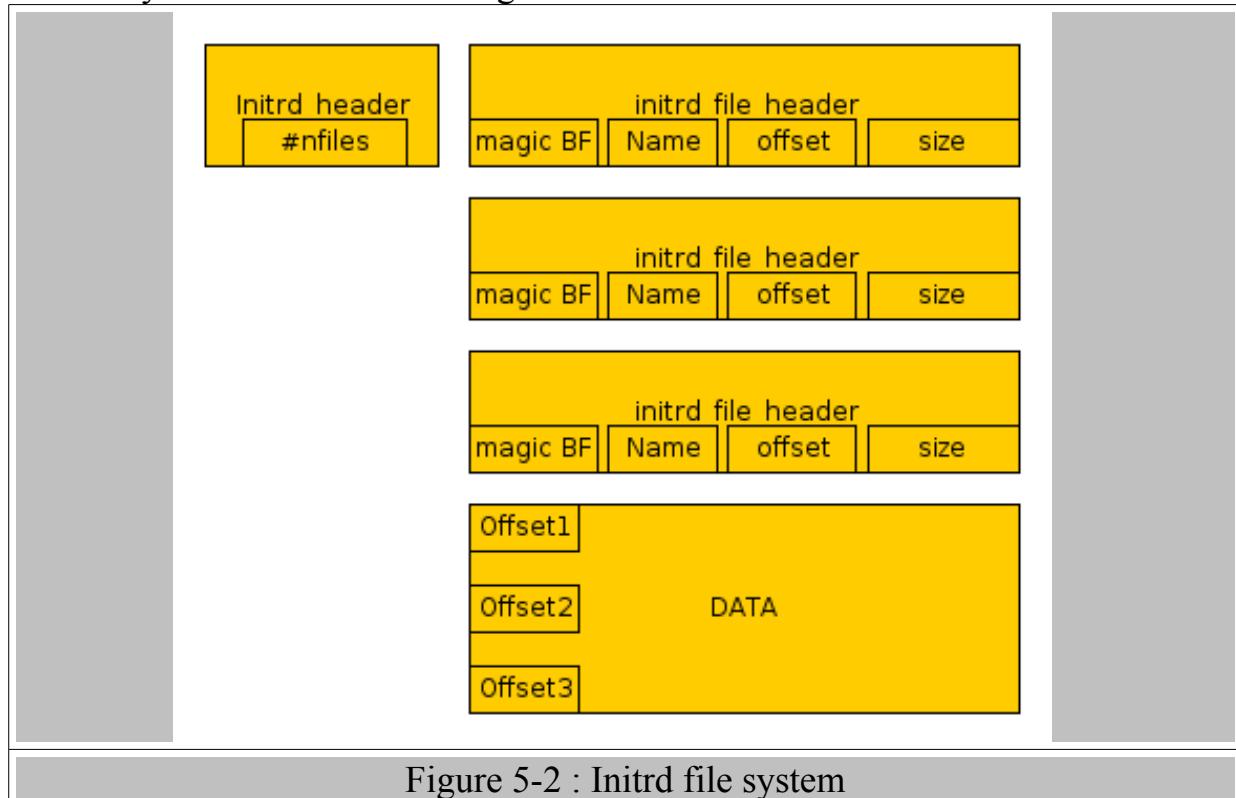
Readdir : If the current node is a directory, it reads its contents and return the next directory entry inside it.

Finddir : We also need a way of finding a child node in a directory.

5.1 Initrd

Initrd , standing for initial RAM disk is a small file system that gets loaded into memory when the kernel boots up .

Many modern operating system kernels, such as Mach , do not by themselves contain enough mechanism to get the system fully operational: they require the presence of additional software modules at boot time in order to access devices, mount file systems, etc. While these additional modules could be embedded in the main OS image along with the kernel itself, and the resulting image be split apart manually by the operating system when it receives control, it is often more flexible, more space-efficient, and more convenient to the operating system and user if the boot loader can load these additional modules independently in the first place. Our version of initrd file system is the simplest it could be as we're using a flat file system with the following format:



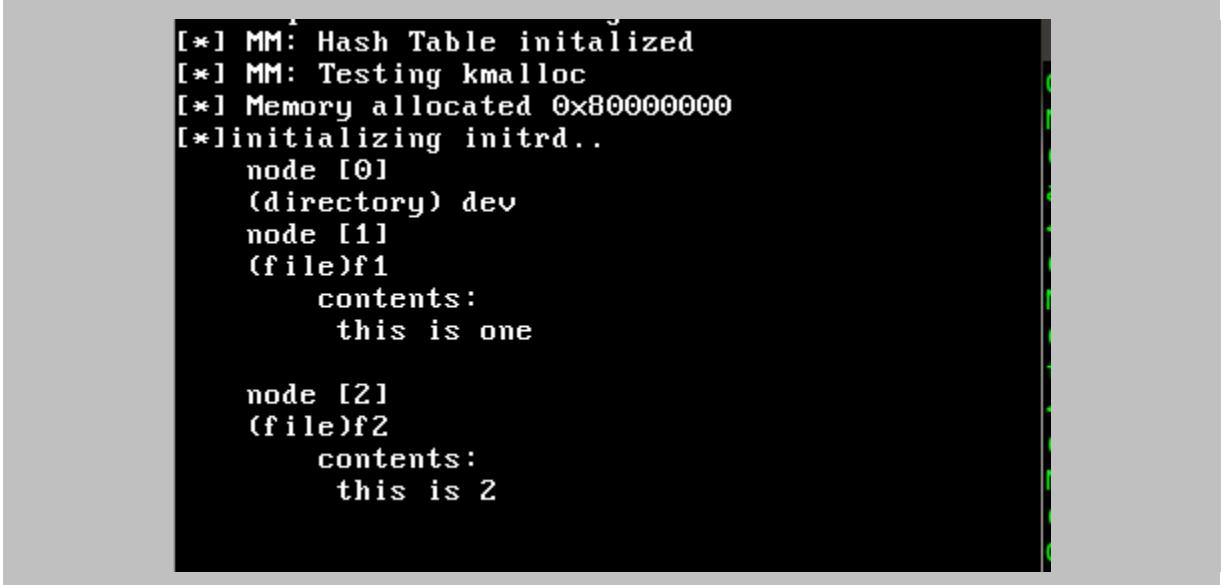
to make things easier we've included a tool that creates the initrd image in the tools/ directory. The initrd module Implements the VFS standards to match our small file system , we'll only need Read , Finddir and Readdir.

Grub loads the initrd image right after the kernel in memory ; GRUB communicates the location of this file to us via the multiboot information structure defined in multiboot.h (information about the structure fount in vfs/multiboot.info).

for more information multi boot specification refer to

<http://www.gnu.org/software/grub/manual/multiboot/multiboot.html>

After loading the modules into memory via grub , It's now just a matter of applying files systems operations in order to read the contents



The screenshot shows a terminal window with a black background and white text. The text is a log of kernel initialization steps:

```
[*] MM: Hash Table initialized
[*] MM: Testing kmalloc
[*] Memory allocated 0x80000000
[*]initializing initrd..
    node [0]
    (directory) dev
    node [1]
    (file)f1
        contents:
        this is one

    node [2]
    (file)f2
        contents:
        this is 2
```

Figure 5-3 : Initrd.

After having an operational Initrd , Next step would be loading file system modules and reading proc0 from disk to move on to user space.

Appendix A – Problems faced :

On User environment initialization:-

Mainly we initialize a user environment at this point to test how effective interrupts are when issued from a user space. the way we did a user space is to simply put another elf image on the disk shifted by several sectors from kernel and load it into memory and jump to it just like we did on boot sector. a far jump with user code segment. But we faced several problems.

1-We forgot to put a writable permission on the page to be able to load data from disk to memory. But such a behavior is not acceptable. since a code segment shouldn't be writable.

2- We forgot to provide a Ring 3 (SEG|3) Or to a user segment

3- We did depend on a statically compiled/linked elf as user environmet.

Goals are:

1- to be able to load code into memory without writable permission, of course this can be done by disabling writing to page after loading the code, but this should be the role of LDT later.

2- to be able to switch to user mode. which can be done only by "Fooling the x86 cpu"

3- Provide a data structure with processes operating.

On scheduling

to schedule we need a kernel clock to issue interrupts. scheduling can be based on two criteria (time slicing, cycle slicing). however we chose to support time slicing scheduling first by using RTC(which is very naive but we chose to support legacy first). and we faced a problem that the RTC interrupt doesn't occur on protected mode. unlike the Intel 8253 PIT interrupt which does happen on

protected mode. However, Some RTC timer code may not work on some real machines. The observed problem is a timer tick happened about once every second. I'm not sure why this is, and am trying to find a solution. This Makes RTC is not a serious solution for real life cases. However, on Bochs emulator an RTC interrupt occur from the slave PIC, But sadly Slave PIC doesn't support auto EOI like the master PIC therefore a handler has to issue an EOI once interrupt is handled which we did. But there was never a second interrupt from Slave PIC.

APIC Initialization

- 1- Due to the fact that APIC registers are at physical address 0xfe000000 we had to identity map the APIC register 4KB page, so we had to modify the kernel address space test to exclude this page from test.
- 2- Bochs APIC0 error, During dumping information of APIC specifically LAVR register, bochs had an error with APIC in the code. however I don't know yet if it is solvable. so this to be edited soon.
- 3- Different DFR Values, While dumping the values of DFR bochs it showed 0xffffffff while on qemu 0xf0000000 this might be due to the virtualization of the real APIC on hosting for the qemu. yet to be investigated too.

Context Switching & Scheduling

A proper way to make a process switch in a healthy manner from Kernel address space to its address space is to Sharing its PEB which holds information about it like page directory and cpu status. a previous approach was to share Whole process table as readable for user processes address spaces. However, this approach was poor and insecure.

Despite that a process switches to ring 0 privilege during scheduling process the scheduling structures would be still invisible to it and would cause a page fault exception. to provide a full kernel mode an address space is being changed to kernel's to prevent any either GP or PF exceptions.

Initial ramdisk & GRUB

an initrd was an entity that we had to provide before any microkernel exclusive structures to loosen and ease the process of developing services and driver/kmods. Grub by default loads any module after its last loaded module which was in initrd case the kernel itself. This contradicted with the fact that we use a boot time

allocator to allocate boot time heaps to hold early structures like Page tables and process tables. we did an easy approach in which we had to statically allocate two pages for initrd before allocating any early heaps.

CATernel

CATernel Code Documentation

2011-2013

Contents

1	Module Index	1
1.1	Modules	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Module Documentation	9
4.1	Interrupts-and-Syscalls	9
4.1.1	Function Documentation	10
4.1.1.1	idt_init	10
4.1.1.2	interrupt_init	10
4.1.1.3	interrupt_printinfo	10
4.1.1.4	map_exception	10
4.1.1.5	map_syscall	11
4.1.1.6	page_fault_handler	11
4.1.1.7	register_exception	11
4.1.1.8	sys_printf	11
4.1.1.9	trap	12
4.1.2	Variable Documentation	12
4.1.2.1	idt	12
4.1.2.2	int_generic	12
4.1.2.3	n_calls	12
4.1.2.4	s_errno	12

4.1.2.5	sys_call_table	12
4.1.2.6	x86_exception_names	12
4.2	Memory-Management	14
4.2.1	Function Documentation	15
4.2.1.1	allocate	15
4.2.1.2	init_tss	16
4.2.1.3	map_segment_page	16
4.2.1.4	memory_printinfo	16
4.2.1.5	scan_memory	16
4.2.1.6	x86_page_alloc	16
4.2.1.7	x86_page_detach	17
4.2.1.8	x86_page_free	17
4.2.1.9	x86_page_init	17
4.2.1.10	x86_page_insert	17
4.2.1.11	x86_page_lookup	18
4.2.1.12	x86_page_remove	18
4.2.1.13	x86_paging_init	18
4.2.1.14	x86_pgdir_find	19
4.2.1.15	x86_read_mem_size	19
4.2.1.16	x86_setup_memory	19
4.2.1.17	x86_test_pgdir	20
4.2.2	Variable Documentation	20
4.2.2.1	alloc_lock	20
4.2.2.2	catgdt	20
4.2.2.3	ext_base	20
4.2.2.4	free_pages	21
4.2.2.5	gdtdesc	21
4.2.2.6	global_cr3	21
4.2.2.7	global_pgdir	21
4.2.2.8	idtdesc	21
4.2.2.9	max_addr	21
4.2.2.10	mem_base	21
4.2.2.11	next_free	22
4.2.2.12	next_free	22

4.2.2.13	page_count	22
4.2.2.14	pages	22
4.2.2.15	pages	22
4.2.2.16	proc_table	22
4.2.2.17	tss	22
4.3	Multiprocessors	23
4.3.1	Function Documentation	23
4.3.1.1	apic_init_ipi	23
4.3.1.2	apic_read_errors	23
4.3.1.3	apic_s_ipi	24
4.3.1.4	lalic_init	24
4.3.1.5	msr_lalic_enable	24
4.3.1.6	soft_lalic_enable	25
4.3.2	Variable Documentation	25
4.3.2.1	lalic	25
4.3.2.2	vector_addr	25
4.4	MultiProcessors	26
4.4.1	Function Documentation	27
4.4.1.1	ap_init	27
4.4.1.2	asm_delay_loop	27
4.4.1.3	asm_fake_loop	27
4.4.1.4	cpu_is_bootstrap	28
4.4.1.5	cpu_is_enabled	28
4.4.1.6	ct_check	28
4.4.1.7	ct_entries	29
4.4.1.8	ct_read_hdr	29
4.4.1.9	delay	29
4.4.1.10	find_set_fps	30
4.4.1.11	fps_check	30
4.4.1.12	fsp_print	30
4.4.1.13	map_AP_Startup	30
4.4.1.14	print_proc_entry	31
4.4.2	Variable Documentation	31
4.4.2.1	bus_entries	31

4.4.2.2	bus_entry_cnt	31
4.4.2.3	ct	31
4.4.2.4	fs	31
4.4.2.5	io_apic_cnt	31
4.4.2.6	io_apic_entries	31
4.4.2.7	io_apic_entry_cnt	32
4.4.2.8	io_intr_entries	32
4.4.2.9	io_intr_entry_cnt	32
4.4.2.10	loc_intr_entries	32
4.4.2.11	loc_intr_entry_cnt	32
4.4.2.12	processor_entries	32
4.4.2.13	processors_count	32
4.4.2.14	trampoline	32
4.4.2.15	trampoline_end	32
4.5	CPU-Initialization	33
4.5.1	Function Documentation	33
4.5.1.1	processor_identify	33
4.5.1.2	processor_printinfo	33
4.5.2	Variable Documentation	33
4.5.2.1	cpu	33
4.5.2.2	cpu_vendors	33
4.6	Drivers	34
4.6.1	Define Documentation	36
4.6.1.1	ALT	36
4.6.1.2	CAPSLOCK	36
4.6.1.3	CL	36
4.6.1.4	CTL	36
4.6.1.5	ESCODE	36
4.6.1.6	LOOPS	37
4.6.1.7	MAGIC_NUMBER	37
4.6.1.8	NUL	37
4.6.1.9	NUMLOCK	37
4.6.1.10	SCROLLLOCK	37
4.6.1.11	SHIFT	37

4.6.2	Function Documentation	37
4.6.2.1	cga_clear	37
4.6.2.2	cga_get_pos	37
4.6.2.3	cga_init	37
4.6.2.4	cga_putc	38
4.6.2.5	cga_putstr	38
4.6.2.6	cga_set_attr	38
4.6.2.7	cga_set_pos	38
4.6.2.8	clock_enable_rtc	39
4.6.2.9	cmos_get_reg	39
4.6.2.10	cmos_set_power_stat	39
4.6.2.11	cmos_set_reg	39
4.6.2.12	i8254_calibrate_delay_loop	40
4.6.2.13	i8254_init	40
4.6.2.14	i8259_disable	40
4.6.2.15	i8259_eoi	40
4.6.2.16	i8259_init	41
4.6.2.17	i8259_mask_irq	41
4.6.2.18	i8259_read_irr	41
4.6.2.19	i8259_read_isr	41
4.6.2.20	i8259_unmask_irq	41
4.6.2.21	kbc_data	42
4.6.2.22	kbc_interrupt	42
4.6.3	Variable Documentation	42
4.6.3.1	cga_attr	42
4.6.3.2	charcode	42
4.6.3.3	ctlmap	42
4.6.3.4	delay_loop_const	43
4.6.3.5	normalmap	43
4.6.3.6	shiftcode	43
4.6.3.7	shiftmap	44
4.6.3.8	togglecode	44
4.7	Main	45
4.7.1	Define Documentation	47

4.7.1.1	BUFSIZE	47
4.7.1.2	hex2ascii	47
4.7.2	Function Documentation	47
4.7.2.1	_panic_	47
4.7.2.2	bootup	47
4.7.2.3	console_clear	47
4.7.2.4	console_getc	47
4.7.2.5	console_init	48
4.7.2.6	console_interrupt	48
4.7.2.7	console_putc	48
4.7.2.8	getchar	48
4.7.2.9	getint	48
4.7.2.10	getuint	49
4.7.2.11	initrd	49
4.7.2.12	initrd_test	49
4.7.2.13	ksprintkn	49
4.7.2.14	kvprintk	50
4.7.2.15	memcpy	50
4.7.2.16	memset	50
4.7.2.17	play	51
4.7.2.18	printk	51
4.7.2.19	putch	51
4.7.2.20	putchr	51
4.7.2.21	readline	52
4.7.2.22	strcmp	52
4.7.2.23	strnlen	52
4.7.2.24	time_print	52
4.7.2.25	vprintf	52
4.7.2.26	work_it_out	53
4.7.3	Variable Documentation	53
4.7.3.1	buf	53
4.7.3.2	error_panic	53
4.8	Category Here--	54
4.8.1	Function Documentation	54

4.8.1.1	initrd_mem	54
4.9	Debug	55
4.9.1	Function Documentation	55
4.9.1.1	kcommand_match	55
4.9.1.2	kcommand_register	55
4.9.1.3	kconsole	55
4.9.1.4	kconsole_help	55
4.9.1.5	kconsole_init	55
4.9.1.6	kcpu_info	55
4.9.1.7	kscheduler_info	56
4.9.2	Variable Documentation	56
4.9.2.1	kconsole_commands	56
4.10	Management	57
4.10.1	Function Documentation	57
4.10.1.1	kfree	57
4.10.1.2	kmalloc	57
4.10.1.3	kmalloc_test	58
4.10.1.4	mm_destroy	58
4.10.1.5	mm_hash	58
4.10.1.6	mm_init	59
4.10.1.7	mm_insert	59
4.10.2	Variable Documentation	60
4.10.2.1	cur_bucket	60
4.10.2.2	mm_table	60
4.10.2.3	mm_va_base	60
4.11	Optimization	61
4.11.1	Function Documentation	61
4.11.1.1	main	61
4.11.1.2	merge	61
4.11.1.3	mergesort	61
4.11.1.4	partition	61
4.11.1.5	quick_sort	62
4.11.1.6	quicksort	62
4.11.1.7	swap	62

4.12 Process-Management	63
4.12.1 Detailed Description	64
4.12.2 Function Documentation	64
4.12.2.1 create_proc	64
4.12.2.2 elf_load_to_proc	64
4.12.2.3 FIFO_HEAD	64
4.12.2.4 init_proc	64
4.12.2.5 init_proc0	65
4.12.2.6 init_proc_table	65
4.12.2.7 proc_alloc_mem	65
4.12.2.8 proc_printinfo	65
4.12.2.9 proc_ready	65
4.12.2.10 sched_init	66
4.12.2.11 schedule	66
4.12.2.12 switch_address_space	66
4.12.2.13 test_fifo	66
4.12.2.14 test_lifo	67
4.12.3 Variable Documentation	67
4.12.3.1 empty_procs	67
4.12.3.2 global_pgdir	67
4.12.3.3 proc_table	67
4.12.3.4 running_procs	67
4.13 Data-Structures	68
4.13.1 Function Documentation	68
4.13.1.1 early_htable_init	68
4.13.1.2 htable_destroy	68
4.13.1.3 htable_get	68
4.13.1.4 htable_init	69
4.13.1.5 htable_insert	69
4.13.1.6 htable_remove	69
4.14 Synchronization	70
4.14.1 Typedef Documentation	71
4.14.1.1 waiting_t	71
4.14.2 Function Documentation	71

4.14.2.1	LIST_HEAD	71
4.14.2.2	mutex_init	71
4.14.2.3	mutex_lock	71
4.14.2.4	mutex_unlock	72
4.14.2.5	semaphore_down	72
4.14.2.6	semaphore_init	72
4.14.2.7	semaphore_up	73
4.14.2.8	wait_init	73
4.14.2.9	wait_sleep	73
4.14.2.10	wait_update	74
4.14.2.11	wait_wakeup	74
4.15	Time	75
4.15.1	Function Documentation	75
4.15.1.1	time_handler	75
4.16	VFS	76
4.16.1	Define Documentation	78
4.16.1.1	FS_BLOCKDEVICE	78
4.16.1.2	FS_CHARDEVICE	78
4.16.1.3	FS_DIRECTORY	78
4.16.1.4	FS_FILE	78
4.16.1.5	FS_MOUNTPOINT	78
4.16.1.6	FS_PIPE	78
4.16.1.7	FS_SYMLINK	78
4.16.2	Typedef Documentation	78
4.16.2.1	close_type_t	78
4.16.2.2	finddir_type_t	78
4.16.2.3	open_type_t	78
4.16.2.4	read_type_t	79
4.16.2.5	readdir_type_t	79
4.16.2.6	vfs_node_t	79
4.16.2.7	write_type_t	79
4.16.3	Function Documentation	79
4.16.3.1	close_fs	79
4.16.3.2	finddir_fs	79

4.16.3.3 initialise_initrd	80
4.16.3.4 initrd_fiaddir	80
4.16.3.5 initrd_read	80
4.16.3.6 initrd_readdir	81
4.16.3.7 open_fs	81
4.16.3.8 read_fs	81
4.16.3.9 readdir_fs	82
4.16.3.10 write_fs	82
4.16.4 Variable Documentation	82
4.16.4.1 dirent	82
4.16.4.2 file_headers	83
4.16.4.3 fs_root	83
4.16.4.4 initrd_dev	83
4.16.4.5 initrd_header	83
4.16.4.6 initrd_root	83
4.16.4.7 nroot_nodes	83
4.16.4.8 root_nodes	83
4.17 X86-boot	84
4.17.1 Define Documentation	85
4.17.1.1 ELF_MAGIC	85
4.17.1.2 ELFHDR	86
4.17.1.3 M_ABIVERSION	86
4.17.1.4 M_CLASS32	86
4.17.1.5 M_CLASS64	86
4.17.1.6 M_CLASS_OFF	86
4.17.1.7 M_CLASSNONE	86
4.17.1.8 M_CLASSNUM	86
4.17.1.9 M_DATA2BE	86
4.17.1.10 M_DATA2LE	86
4.17.1.11 M_DATA_OFF	86
4.17.1.12 M_DATANONE	87
4.17.1.13 M_DATANUM	87
4.17.1.14 M_ELF_PADDING	87
4.17.1.15 M_MACHINE_I386	87

4.17.1.16 M_OSABI	87
4.17.1.17 M_OSABI_HPUX	87
4.17.1.18 M_OSABI_SYSV	87
4.17.1.19 M_VERSION	87
4.17.1.20 MAGIC_LEN	87
4.17.1.21 P_PROGHDR_E	87
4.17.1.22 P_PROGHDR_R	88
4.17.1.23 P_PROGHDR_W	88
4.17.1.24 SECTOR	88
4.17.1.25 T_TYPE_CORE	88
4.17.1.26 T_TYPE_DYN	88
4.17.1.27 T_TYPE_EXEC	88
4.17.1.28 T_TYPE_HIPROC	88
4.17.1.29 T_TYPE_LOPROC	88
4.17.1.30 T_TYPE_NONE	88
4.17.1.31 T_TYPE_REL	88
4.17.1.32 V_VERSION_CURRENT	89
4.17.1.33 V_VERSION_NONE	89
4.17.1.34 V_VERSION_NUM	89
4.17.2 Typedef Documentation	89
4.17.2.1 elfhdr	89
4.17.2.2 prohdr	89
4.17.2.3 secthdr	89
4.17.3 Function Documentation	89
4.17.3.1 cmain	89
4.17.3.2 inb	89
4.17.3.3 inl	89
4.17.3.4 insb	90
4.17.3.5 insl	90
4.17.3.6 insw	90
4.17.3.7 inw	90
4.17.3.8 outb	90
4.17.3.9 outl	90
4.17.3.10 outw	90

4.17.3.11	readsect	90
4.17.3.12	readseg	91
4.17.3.13	waitdisk	91
5	Data Structure Documentation	93
5.1	<u>__attribute Struct Reference</u>	93
5.1.1	Detailed Description	94
5.1.2	Field Documentation	94
5.1.2.1	aout_sym	94
5.1.2.2	apm_table	94
5.1.2.3	boot_device	94
5.1.2.4	boot_loader_name	94
5.1.2.5	cmdline	94
5.1.2.6	config_table	94
5.1.2.7	drives_addr	94
5.1.2.8	drives_length	94
5.1.2.9	elf_sec	94
5.1.2.10	flags	95
5.1.2.11	mem_lower	95
5.1.2.12	mem_upper	95
5.1.2.13	mmap_addr	95
5.1.2.14	mmap_length	95
5.1.2.15	mods_addr	95
5.1.2.16	mods_count	95
5.1.2.17	u	95
5.1.2.18	vbe_control_info	95
5.1.2.19	vbe_interface_len	95
5.1.2.20	vbe_interface_off	95
5.1.2.21	vbe_interface_seg	96
5.1.2.22	vbe_mode	96
5.1.2.23	vbe_mode_info	96
5.2	<u>__attribute__ Struct Reference</u>	96
5.2.1	Detailed Description	100
5.2.2	Field Documentation	100

5.2.2.1	"@1	100
5.2.2.2	"@3	100
5.2.2.3	_rsrvd1	100
5.2.2.4	_rsrvd2	100
5.2.2.5	_rsrvd3	100
5.2.2.6	_rsrvd4	100
5.2.2.7	_rsrvd5	100
5.2.2.8	_rsrvd6	100
5.2.2.9	_rsrvd7	101
5.2.2.10	_rsrvd8	101
5.2.2.11	_rsrvd9	101
5.2.2.12	_rsrvda	101
5.2.2.13	_rsrvdb	101
5.2.2.14	access	101
5.2.2.15	accessed	101
5.2.2.16	accessible	101
5.2.2.17	addr	101
5.2.2.18	addr	101
5.2.2.19	address	102
5.2.2.20	args	102
5.2.2.21	available	102
5.2.2.22	base_0_15	102
5.2.2.23	base_16_23	102
5.2.2.24	base_24_31	102
5.2.2.25	base_table	102
5.2.2.26	base_table_len	102
5.2.2.27	bss_end_addr	102
5.2.2.28	bus_id	102
5.2.2.29	bus_type	103
5.2.2.30	cache_disable	103
5.2.2.31	checksum	103
5.2.2.32	checksum	103
5.2.2.33	cmdline	103
5.2.2.34	config_addr	103

5.2.2.35	cpu_flags	103
5.2.2.36	cpu_signature	103
5.2.2.37	cr3	103
5.2.2.38	cs	103
5.2.2.39	delivery_mode	104
5.2.2.40	delivery_stat	104
5.2.2.41	depth	104
5.2.2.42	dest	104
5.2.2.43	dest_mode	104
5.2.2.44	dirty	104
5.2.2.45	dpl	104
5.2.2.46	ds	104
5.2.2.47	dst_io_apic_id	104
5.2.2.48	dst_io_apic_pin	104
5.2.2.49	dst_loc_apic_id	105
5.2.2.50	dst_loc_apic_pin	105
5.2.2.51	eax	105
5.2.2.52	ebp	105
5.2.2.53	ebx	105
5.2.2.54	ecx	105
5.2.2.55	edi	105
5.2.2.56	edx	105
5.2.2.57	eflags	105
5.2.2.58	eip	105
5.2.2.59	entry_addr	106
5.2.2.60	entry_count	106
5.2.2.61	entry_type	106
5.2.2.62	es	106
5.2.2.63	esi	106
5.2.2.64	esp	106
5.2.2.65	esp0	106
5.2.2.66	esp1	106
5.2.2.67	esp2	106
5.2.2.68	extnd_table_checksum	106

5.2.2.69 extnd_table_len	107
5.2.2.70 feature1	107
5.2.2.71 feature2	107
5.2.2.72 feature_flags	107
5.2.2.73 flags	107
5.2.2.74 fs	107
5.2.2.75 global	107
5.2.2.76 granularity	107
5.2.2.77 gs	107
5.2.2.78 header_addr	107
5.2.2.79 height	108
5.2.2.80 hi	108
5.2.2.81 ignore_this	108
5.2.2.82 int	108
5.2.2.83 intrrupt_type	108
5.2.2.84 io_apic_addr	108
5.2.2.85 io_apic_flags	108
5.2.2.86 io_apic_id	108
5.2.2.87 io_apic_ver	108
5.2.2.88 iomap_base	108
5.2.2.89 lapic_addr	109
5.2.2.90 lapic_id	109
5.2.2.91 lapic_version	109
5.2.2.92 ldt	109
5.2.2.93 len	109
5.2.2.94 len	109
5.2.2.95 level	109
5.2.2.96 limit_0_15	109
5.2.2.97 limit_16_19	109
5.2.2.98 lo	109
5.2.2.99 load_addr	110
5.2.2.100 load_end_addr	110
5.2.2.101 magic	110
5.2.2.102 mod_end	110

5.2.2.103 mod_start	110
5.2.2.104 mode_type	110
5.2.2.105 moduleEnd	110
5.2.2.106 moduleStart	110
5.2.2.107 num	110
5.2.2.108 oem_id	110
5.2.2.109 oem_table_pointer	111
5.2.2.110 oem_table_size	111
5.2.2.111 offset_0_15	111
5.2.2.112 offset_16_31	111
5.2.2.113 p	111
5.2.2.114 pad	111
5.2.2.115 pad	111
5.2.2.116 pat	111
5.2.2.117 po_el	111
5.2.2.118 prelink	111
5.2.2.119 present	112
5.2.2.120 product_id	112
5.2.2.121 reserved	112
5.2.2.122 reserved	112
5.2.2.123 reserved	112
5.2.2.124 s	112
5.2.2.125 segment_s	112
5.2.2.126 shndx	112
5.2.2.127 shorthand	112
5.2.2.128 Signature	112
5.2.2.129 size	113
5.2.2.130 spec_ver	113
5.2.2.131 src_bus_id	113
5.2.2.132 src_bus_irq	113
5.2.2.133 ss	113
5.2.2.134 ss0	113
5.2.2.135 ss1	113
5.2.2.136 ss2	113

5.2.2.137	string	113
5.2.2.138	strsize	113
5.2.2.139	tabsize	114
5.2.2.140	trace	114
5.2.2.141	trigger_mode	114
5.2.2.142	type	114
5.2.2.143	type	114
5.2.2.144	type	114
5.2.2.145	unused	114
5.2.2.146	vector	114
5.2.2.147	version	114
5.2.2.148	width	114
5.2.2.149	writable	115
5.2.2.150	write_through	115
5.3	btree_node Struct Reference	115
5.3.1	Detailed Description	115
5.3.2	Field Documentation	116
5.3.2.1	key	116
5.3.2.2	left	116
5.3.2.3	parent	116
5.3.2.4	right	116
5.3.2.5	val	116
5.4	btree_t Struct Reference	116
5.4.1	Detailed Description	117
5.4.2	Member Function Documentation	117
5.4.2.1	LIST_ENTRY	117
5.4.3	Field Documentation	117
5.4.3.1	root	117
5.4.3.2	size	118
5.5	cpu_cache Struct Reference	118
5.5.1	Detailed Description	118
5.5.2	Member Function Documentation	118
5.5.2.1	__attribute__	118
5.5.2.2	__attribute__	118

5.5.3	Field Documentation	118
5.5.3.1	__pad0__	119
5.5.3.2	assoc	119
5.5.3.3	eax	119
5.5.3.4	inclusive	119
5.5.3.5	indexing	119
5.5.3.6	line_size	119
5.5.3.7	phy_line	119
5.5.3.8	sets	119
5.5.3.9	write_back	119
5.6	cpu_state_t Struct Reference	120
5.6.1	Detailed Description	120
5.6.2	Field Documentation	120
5.6.2.1	cs	120
5.6.2.2	ds	120
5.6.2.3	eax	121
5.6.2.4	ebp_frame	121
5.6.2.5	ebx	121
5.6.2.6	ecx	121
5.6.2.7	edi	121
5.6.2.8	edx	121
5.6.2.9	eflags	121
5.6.2.10	eip	121
5.6.2.11	eip_frame	121
5.6.2.12	error_code	121
5.6.2.13	es	122
5.6.2.14	esi	122
5.6.2.15	esp	122
5.6.2.16	fs	122
5.6.2.17	gs	122
5.6.2.18	o_ebp	122
5.6.2.19	o_esp	122
5.6.2.20	ss	122
5.7	cpu_t Struct Reference	122

5.7.1	Detailed Description	123
5.7.2	Field Documentation	123
5.7.2.1	cache	123
5.7.2.2	vendor	123
5.7.2.3	version	123
5.8	cpu_version Struct Reference	124
5.8.1	Detailed Description	125
5.8.2	Member Function Documentation	125
5.8.2.1	<u>attribute</u>	125
5.8.2.2	<u>attribute</u>	125
5.8.3	Field Documentation	125
5.8.3.1	<u>pad0</u>	125
5.8.3.2	<u>pad1</u>	125
5.8.3.3	<u>pad2</u>	126
5.8.3.4	<u>pad3</u>	126
5.8.3.5	<u>pad4</u>	126
5.8.3.6	acpi	126
5.8.3.7	apic	126
5.8.3.8	brand	126
5.8.3.9	cache	126
5.8.3.10	cmov	126
5.8.3.11	dca	126
5.8.3.12	de	126
5.8.3.13	est	127
5.8.3.14	exfamily	127
5.8.3.15	exmodel	127
5.8.3.16	family	127
5.8.3.17	fms	127
5.8.3.18	fpu	127
5.8.3.19	fxsr	127
5.8.3.20	generic	127
5.8.3.21	htt	127
5.8.3.22	id	127
5.8.3.23	max_id	127

5.8.3.24	mca	127
5.8.3.25	mmx	128
5.8.3.26	model	128
5.8.3.27	pae	128
5.8.3.28	pat	128
5.8.3.29	pbe	128
5.8.3.30	pcid	128
5.8.3.31	pge	128
5.8.3.32	pse	128
5.8.3.33	sep	128
5.8.3.34	smx	128
5.8.3.35	ss	129
5.8.3.36	sse	129
5.8.3.37	sse2	129
5.8.3.38	sse3	129
5.8.3.39	sse4_1	129
5.8.3.40	sse4_2	129
5.8.3.41	sss	129
5.8.3.42	ssse3	129
5.8.3.43	stepping	129
5.8.3.44	tm	129
5.8.3.45	tm2	130
5.8.3.46	type	130
5.8.3.47	vme	130
5.8.3.48	vmx	130
5.8.3.49	x2apic	130
5.9	cpuid_regs Struct Reference	130
5.9.1	Detailed Description	130
5.9.2	Field Documentation	130
5.9.2.1	eax	130
5.9.2.2	ebx	131
5.9.2.3	ecx	131
5.9.2.4	edx	131
5.10	cpuid_t Struct Reference	131

5.10.1	Detailed Description	131
5.10.2	Field Documentation	131
5.10.2.1	eax	131
5.10.2.2	ebx	131
5.10.2.3	ecx	132
5.10.2.4	edx	132
5.11	dirent_t Struct Reference	132
5.11.1	Detailed Description	132
5.11.2	Field Documentation	132
5.11.2.1	inode_n	132
5.11.2.2	name	132
5.12	element Struct Reference	132
5.12.1	Detailed Description	133
5.13	elfhdr Struct Reference	133
5.13.1	Detailed Description	133
5.13.2	Field Documentation	133
5.13.2.1	ehsize	133
5.13.2.2	entry	134
5.13.2.3	flags	134
5.13.2.4	machine	134
5.13.2.5	magic	134
5.13.2.6	magic	134
5.13.2.7	magic2	134
5.13.2.8	phrnum	134
5.13.2.9	phroff	134
5.13.2.10	phrsize	135
5.13.2.11	shrnum	135
5.13.2.12	shroff	135
5.13.2.13	shrszie	135
5.13.2.14	shrsttbl	135
5.13.2.15	type	135
5.13.2.16	version	135
5.14	ext2_dir_entry Struct Reference	136
5.14.1	Detailed Description	136

5.14.2 Field Documentation	136
5.14.2.1 inode	136
5.14.2.2 name	136
5.14.2.3 name_len	136
5.14.2.4 rec_len	136
5.15 ext2_group_desc Struct Reference	136
5.15.1 Detailed Description	137
5.15.2 Field Documentation	137
5.15.2.1 bg_block_bitmap	137
5.15.2.2 bg_free_blocks_count	137
5.15.2.3 bg_free_inodes_count	137
5.15.2.4 bg_inode_bitmap	137
5.15.2.5 bg_inode_table	137
5.15.2.6 bg_pad	137
5.15.2.7 bg_reserved	138
5.15.2.8 bg_used_dirs_count	138
5.16 ext2_inode Struct Reference	138
5.16.1 Detailed Description	139
5.16.2 Field Documentation	139
5.16.2.1 h_i_author	139
5.16.2.2 h_i_frag	139
5.16.2.3 h_i_fsize	139
5.16.2.4 h_i_gid_high	139
5.16.2.5 h_i_mode_high	139
5.16.2.6 h_i_uid_high	140
5.16.2.7 hurd2	140
5.16.2.8 i_atime	140
5.16.2.9 i_block	140
5.16.2.10 i_blocks	140
5.16.2.11 i_ctime	140
5.16.2.12 i_dir_acl	140
5.16.2.13 i_dtime	140
5.16.2.14 i_faddr	140
5.16.2.15 i_file_acl	140

5.16.2.16 <i>i_flags</i>	140
5.16.2.17 <i>i_gid</i>	141
5.16.2.18 <i>i_links_count</i>	141
5.16.2.19 <i>i_mode</i>	141
5.16.2.20 <i>i_mtime</i>	141
5.16.2.21 <i>i_reserved1</i>	141
5.16.2.22 <i>i_size</i>	141
5.16.2.23 <i>i_uid</i>	141
5.16.2.24 <i>i_version</i>	141
5.16.2.25 <i>l_i_frag</i>	141
5.16.2.26 <i>l_i_fsize</i>	141
5.16.2.27 <i>l_i_gid_high</i>	142
5.16.2.28 <i>l_i_reserved1</i>	142
5.16.2.29 <i>l_i_reserved2</i>	142
5.16.2.30 <i>l_i_uid_high</i>	142
5.16.2.31 <i>linux</i>	142
5.16.2.32 <i>m_i_frag</i>	142
5.16.2.33 <i>m_i_fsize</i>	142
5.16.2.34 <i>m_i_reserved2</i>	142
5.16.2.35 <i>m_pad1</i>	142
5.16.2.36 <i>masix2</i>	142
5.16.2.37 <i>osd2</i>	142
5.17 <i>ext2_super_block</i> Struct Reference	143
5.17.1 Detailed Description	144
5.17.2 Field Documentation	144
5.17.2.1 <i>s_algorithm_usage_bitmap</i>	144
5.17.2.2 <i>s_block_group_nr</i>	144
5.17.2.3 <i>s_blocks_count</i>	144
5.17.2.4 <i>s_blocks_per_group</i>	144
5.17.2.5 <i>s_checkinterval</i>	144
5.17.2.6 <i>s_errors</i>	144
5.17.2.7 <i>s_feature_compat</i>	144
5.17.2.8 <i>s_feature_incompat</i>	144
5.17.2.9 <i>s_feature_ro_compat</i>	145

5.17.2.10 s_first_data_block	145
5.17.2.11 s_first_ino	145
5.17.2.12 s frags_per_group	145
5.17.2.13 s_free_blocks_count	145
5.17.2.14 s_free_inodes_count	145
5.17.2.15 s_gid	145
5.17.2.16 s_inode_size	145
5.17.2.17 s_inodes_count	145
5.17.2.18 s_inodes_per_group	145
5.17.2.19 s_journal_dev	146
5.17.2.20 s_journal_inum	146
5.17.2.21 s_journal_uuid	146
5.17.2.22 s_last_mounted	146
5.17.2.23 s_last_orphan	146
5.17.2.24 s_lastcheck	146
5.17.2.25 s_log_block_size	146
5.17.2.26 s_log_frag_size	146
5.17.2.27 s_magic	146
5.17.2.28 s_maj_pad	146
5.17.2.29 s_max_mnt_count	147
5.17.2.30 s_min_pad	147
5.17.2.31 s_mnt_count	147
5.17.2.32 s_mtime	147
5.17.2.33 s_os_id	147
5.17.2.34 s_padding1	147
5.17.2.35 s_prealloc_blocks	147
5.17.2.36 s_prealloc_dir_blocks	147
5.17.2.37 s_r_blocks_count	147
5.17.2.38 s_reserved	147
5.17.2.39 s_state	148
5.17.2.40 s_uid	148
5.17.2.41 s_uuid	148
5.17.2.42 s_volume_name	148
5.17.2.43 s_wtime	148

5.18 Gdtdesc Struct Reference	148
5.18.1 Detailed Description	148
5.18.2 Field Documentation	148
5.18.2.1 base	148
5.18.2.2 size	149
5.19 gpr_regs_t Struct Reference	149
5.19.1 Detailed Description	149
5.19.2 Field Documentation	149
5.19.2.1 eax	149
5.19.2.2 ebp	149
5.19.2.3 ebx	150
5.19.2.4 ecx	150
5.19.2.5 edi	150
5.19.2.6 edx	150
5.19.2.7 esi	150
5.19.2.8 esp	150
5.20 htable_node_t Struct Reference	150
5.20.1 Detailed Description	150
5.20.2 Field Documentation	151
5.20.2.1 key	151
5.20.2.2 value	151
5.21 htable_t Struct Reference	151
5.21.1 Detailed Description	151
5.21.2 Field Documentation	151
5.21.2.1 bucket_length	151
5.21.2.2 bucket_size	152
5.21.2.3 buckets	152
5.21.2.4 destroy	152
5.21.2.5 hash	152
5.21.2.6 size	152
5.21.2.7 table	152
5.22 initrd_file_header_t Struct Reference	152
5.22.1 Detailed Description	152
5.22.2 Field Documentation	153

5.22.2.1	magic	153
5.22.2.2	name	153
5.22.2.3	offset	153
5.22.2.4	size	153
5.23	initrd_header_t Struct Reference	153
5.23.1	Detailed Description	153
5.23.2	Field Documentation	153
5.23.2.1	nfiles	153
5.24	kcommand Struct Reference	154
5.24.1	Detailed Description	154
5.24.2	Field Documentation	154
5.24.2.1	hint	154
5.24.2.2	name	154
5.24.2.3	operation	154
5.25	Page Struct Reference	154
5.25.1	Detailed Description	155
5.25.2	Field Documentation	155
5.25.2.1	link	155
5.25.2.2	ref	155
5.26	proc Struct Reference	155
5.26.1	Detailed Description	156
5.26.2	Member Function Documentation	156
5.26.2.1	LIFO_ENTRY	156
5.26.2.2	LIST_ENTRY	156
5.26.2.3	LIST_ENTRY	156
5.26.3	Field Documentation	156
5.26.3.1	cr3	156
5.26.3.2	cs	156
5.26.3.3	dequeued	156
5.26.3.4	eflags	157
5.26.3.5	eip	157
5.26.3.6	esp	157
5.26.3.7	gpr_regs	157
5.26.3.8	id	157

5.26.3.9 <code>page_directory</code>	157
5.26.3.10 <code>preempted</code>	157
5.26.3.11 <code>seg_regs</code>	157
5.26.3.12 <code>ss</code>	157
5.26.3.13 <code>status</code>	157
5.26.3.14 <code>timer</code>	158
5.27 <code>proghdr</code> Struct Reference	158
5.27.1 Detailed Description	158
5.27.2 Field Documentation	158
5.27.2.1 <code>align</code>	158
5.27.2.2 <code>filesz</code>	158
5.27.2.3 <code>flags</code>	158
5.27.2.4 <code>memsz</code>	159
5.27.2.5 <code>offset</code>	159
5.27.2.6 <code>paddr</code>	159
5.27.2.7 <code>type</code>	159
5.27.2.8 <code>vaddr</code>	159
5.28 <code>sechdr</code> Struct Reference	159
5.28.1 Detailed Description	160
5.28.2 Field Documentation	160
5.28.2.1 <code>addr</code>	160
5.28.2.2 <code>addralign</code>	160
5.28.2.3 <code>entsize</code>	160
5.28.2.4 <code>flags</code>	160
5.28.2.5 <code>info</code>	160
5.28.2.6 <code>link</code>	160
5.28.2.7 <code>name</code>	160
5.28.2.8 <code>offset</code>	160
5.28.2.9 <code>size</code>	160
5.28.2.10 <code>type</code>	161
5.29 <code>seg_regs_t</code> Struct Reference	161
5.29.1 Detailed Description	161
5.29.2 Field Documentation	161
5.29.2.1 <code>ds</code>	161

5.29.2.2	es	161
5.29.2.3	fs	161
5.29.2.4	gs	162
5.30	Segdesc Struct Reference	162
5.30.1	Detailed Description	162
5.30.2	Field Documentation	162
5.30.2.1	base	162
5.30.2.2	base_0	162
5.30.2.3	base_1	162
5.30.2.4	flags	162
5.30.2.5	limit_0	163
5.30.2.6	limit_1	163
5.30.2.7	permission	163
5.31	semaphore_t Struct Reference	163
5.31.1	Detailed Description	163
5.31.2	Field Documentation	163
5.31.2.1	count	163
5.31.2.2	wait_list	163
5.32	spinlock Struct Reference	164
5.32.1	Detailed Description	164
5.32.2	Field Documentation	164
5.32.2.1	lock	164
5.33	spinlock_t Struct Reference	164
5.33.1	Detailed Description	164
5.33.2	Field Documentation	164
5.33.2.1	lock	164
5.34	vfs_node Struct Reference	165
5.34.1	Detailed Description	165
5.34.2	Field Documentation	166
5.34.2.1	close	166
5.34.2.2	finddir	166
5.34.2.3	flags	166
5.34.2.4	gid	166
5.34.2.5	impl	166

5.34.2.6 inode	166
5.34.2.7 mask	166
5.34.2.8 name	166
5.34.2.9 open	166
5.34.2.10 ptr	166
5.34.2.11 read	167
5.34.2.12 readdir	167
5.34.2.13 size	167
5.34.2.14 uid	167
5.34.2.15 write	167
5.35 waiting_proc Struct Reference	167
5.35.1 Detailed Description	168
5.35.2 Member Function Documentation	168
5.35.2.1 LIST_ENTRY	168
5.35.3 Field Documentation	168
5.35.3.1 proc	168
6 File Documentation	171
6.1 arch/x86/boot/main.c File Reference	171
6.1.1 Detailed Description	172
6.2 include/arch/x86/asm.h File Reference	172
6.2.1 Function Documentation	173
6.2.1.1 arch_spinlock_lock	173
6.2.1.2 arch_spinlock_unlock	173
6.3 include/arch/x86/bios/bios.h File Reference	174
6.3.1 Function Documentation	174
6.3.1.1 bios_init	174
6.3.2 Variable Documentation	175
6.3.2.1 ebda	175
6.4 include/arch/x86/cpu_state.h File Reference	175
6.4.1 Detailed Description	176
6.4.2 Define Documentation	176
6.4.2.1 SEGMENT_TSS	176
6.4.2.2 TASK_BUSY	177

6.4.2.3	TASK_INACTIVE	177
6.4.2.4	TSS_AVL	177
6.4.2.5	TSS_DPL_KERNEL	177
6.4.2.6	TSS_DPL_USER	177
6.4.2.7	TSS_GRANULARITY	177
6.4.2.8	TSS_IOMAP_SIZE	177
6.4.2.9	TSS_PRESENT	177
6.4.2.10	TSS_SIZE	177
6.5	include/arch/x86/elf.h File Reference	178
6.5.1	Define Documentation	179
6.5.1.1	ELF_MAGIC	179
6.5.1.2	M_ABIVERSION	179
6.5.1.3	M_CLASS32	179
6.5.1.4	M_CLASS64	179
6.5.1.5	M_CLASS_OFF	179
6.5.1.6	M_CLASSNONE	179
6.5.1.7	M_CLASSNUM	180
6.5.1.8	M_DATA2BE	180
6.5.1.9	M_DATA2LE	180
6.5.1.10	M_DATA_OFF	180
6.5.1.11	M_DATANONE	180
6.5.1.12	M_DATANUM	180
6.5.1.13	M_ELF_PADDING	180
6.5.1.14	M_MACHINE_I386	180
6.5.1.15	M_OSABI	180
6.5.1.16	M_OSABI_HPUX	180
6.5.1.17	M_OSABI_SYSV	181
6.5.1.18	M_VERSION	181
6.5.1.19	MAGIC_LEN	181
6.5.1.20	P_PROGHDR_E	181
6.5.1.21	P_PROGHDR_R	181
6.5.1.22	P_PROGHDR_W	181
6.5.1.23	PRG_LOAD	181
6.5.1.24	T_TYPE_CORE	181

6.5.1.25	T_TYPE_DYN	181
6.5.1.26	T_TYPE_EXEC	181
6.5.1.27	T_TYPE_HIPROC	182
6.5.1.28	T_TYPE_LOPROC	182
6.5.1.29	T_TYPE_NONE	182
6.5.1.30	T_TYPE_REL	182
6.5.1.31	V_VERSION_CURRENT	182
6.5.1.32	V_VERSION_NONE	182
6.5.1.33	V_VERSION_NUM	182
6.5.2	Typedef Documentation	182
6.5.2.1	elfhdr	182
6.5.2.2	prohdr	182
6.5.2.3	sechdr	182
6.6	arch/x86/include/elf.h File Reference	182
6.7	include/arch/x86/interrupt.h File Reference	184
6.7.1	Detailed Description	186
6.7.2	Define Documentation	186
6.7.2.1	_CATERNEL_X86_INTERRUPT_H_	186
6.7.2.2	GATE_FILL	186
6.7.2.3	GATE_INT16	186
6.7.2.4	GATE_INT32	186
6.7.2.5	GATE_OFFSET	186
6.7.2.6	GATE_TASK	187
6.7.2.7	GATE_TRAP16	187
6.7.2.8	GATE_TRAP32	187
6.7.2.9	GP	187
6.7.2.10	IDT_ENTRIES	187
6.7.2.11	PF	187
6.7.2.12	PF_FROM_KERNEL	187
6.7.2.13	PF_FROM_USER	187
6.7.2.14	PF_IFETCH	187
6.7.2.15	PF_NOT_PRESENT	188
6.7.2.16	PF_ON_READ	188
6.7.2.17	PF_ON_WRITE	188

6.7.2.18	PF_RSRVD	188
6.7.2.19	PF_VIOLATION	188
6.7.2.20	SCHED	188
6.7.2.21	SYSCALL	188
6.7.2.22	VECTOR_INDEX	188
6.7.3	Typedef Documentation	188
6.7.3.1	Idtdesc	188
6.8	include/arch/x86/mm/page.h File Reference	188
6.8.1	Detailed Description	192
6.8.2	Define Documentation	192
6.8.2.1	KA2PA	192
6.8.2.2	PA2KA	192
6.8.2.3	PAGE_ACSESSED	192
6.8.2.4	PAGE_CDISABLE	193
6.8.2.5	PAGE_DIRTY	193
6.8.2.6	PAGE_DIRTY	193
6.8.2.7	PAGE_GLOBAL	193
6.8.2.8	PAGE_PAGESZ	193
6.8.2.9	PAGE_PRESENT	193
6.8.2.10	PAGE_USER	193
6.8.2.11	PAGE_WRITABLE	193
6.8.2.12	PAGE_WTHROUGH	193
6.8.2.13	PGDIR_SHIFT	193
6.8.2.14	PGDIRX	194
6.8.2.15	PGOFF	194
6.8.2.16	PGPN	194
6.8.2.17	PGSHIFT	194
6.8.2.18	PGTBL_SHIFT	194
6.8.2.19	PGTBLX	194
6.8.2.20	PTD_ADDR	194
6.8.2.21	PTSHIFT	194
6.8.2.22	VIRT_PGDIRX	194
6.8.2.23	VIRT_PGPN	194
6.8.2.24	VIRT_PGTBLX	195

6.8.3	Typedef Documentation	195
6.8.3.1	pde_t	195
6.8.4	Function Documentation	195
6.8.4.1	LIST_ENTRY	195
6.8.4.2	LIST_HEAD	195
6.8.4.3	pagetopa	195
6.8.4.4	pagetoppn	195
6.8.4.5	pagetova	195
6.8.4.6	patopage	195
6.8.4.7	va2pa	195
6.8.4.8	x86_test_page_alloc	196
6.8.4.9	x86_test_paging	196
6.8.5	Variable Documentation	196
6.8.5.1	gdt	196
6.8.5.2	kernel_stack	196
6.8.5.3	kernel_stack_end	196
6.8.5.4	virtpgd	196
6.8.5.5	virttgt	196
6.9	include/arch/x86/mm/segdesc.h File Reference	196
6.9.1	Detailed Description	197
6.9.2	Define Documentation	197
6.9.2.1	SEG_KERNCODE	197
6.9.2.2	SEG_KERNDATA	197
6.9.2.3	SEG_TSS	197
6.9.2.4	SEG_USERCODE	197
6.9.2.5	SEG_USERDATA	197
6.10	include/arch/x86/mp/apic.h File Reference	197
6.10.1	Detailed Description	200
6.10.2	Define Documentation	200
6.10.2.1	ALL_W_SELF	200
6.10.2.2	ALL_X_SELF	200
6.10.2.3	APR	200
6.10.2.4	CCT	201
6.10.2.5	DCT	201

6.10.2.6 DFR	201
6.10.2.7 EOI	201
6.10.2.8 ERR	201
6.10.2.9 IA32_APIC_BASE	201
6.10.2.10 IA32_APIC_BASE_ADDR	201
6.10.2.11 IA32_APIC_BASE_BSP	201
6.10.2.12 IA32_APIC_BASE_ENABLE	201
6.10.2.13 ICR_FIXED	201
6.10.2.14 ICR_HIGH	202
6.10.2.15 ICR_IDLE	202
6.10.2.16 ICR_INIT	202
6.10.2.17 ICR_LO_PIR	202
6.10.2.18 ICR_LOW	202
6.10.2.19 ICR_NMI	202
6.10.2.20 ICR_PENDING	202
6.10.2.21 ICR_SIPI	202
6.10.2.22 ICR_SMI	202
6.10.2.23 ICT	202
6.10.2.24 is_82489DX_apic	203
6.10.2.25 LAID	203
6.10.2.26 LAVR	203
6.10.2.27 LAVR_Mask	203
6.10.2.28 LDR	203
6.10.2.29 LEVEL_ASSERT	203
6.10.2.30 LEVEL_DEASSERT	203
6.10.2.31 LOGICAL	203
6.10.2.32 LVT_CMCI	203
6.10.2.33 LVT_ERR	203
6.10.2.34 LVT_LINT0	204
6.10.2.35 LVT_LINT1	204
6.10.2.36 LVT_PCR	204
6.10.2.37 LVT_TIMER	204
6.10.2.38 LVT_TMR	204
6.10.2.39 NO_SH	204

6.10.2.40 PHYSICAL	204
6.10.2.41 PPR	204
6.10.2.42 RRD	204
6.10.2.43 SELF	204
6.10.2.44 SVR	205
6.10.2.45 TPR	205
6.10.2.46 TRIGMOD_EDGE	205
6.10.2.47 TRIGMOD_LEVEL	205
6.10.3 Variable Documentation	205
6.10.3.1 delivery_mode	205
6.10.3.2 delivery_stat	205
6.10.3.3 dest	205
6.10.3.4 dest_mode	205
6.10.3.5 int	205
6.10.3.6 level	205
6.10.3.7 shorthand	206
6.10.3.8 trigger_mode	206
6.10.3.9 vector	206
6.11 include/arch/x86/mp/delay.h File Reference	206
6.11.1 Detailed Description	207
6.12 include/arch/x86/mp/smp.h File Reference	207
6.12.1 Detailed Description	209
6.12.2 Define Documentation	209
6.12.2.1 BUS	209
6.12.2.2 CT_SIGNATURE	209
6.12.2.3 EXTINT	209
6.12.2.4 FS_SIGNATURE	209
6.12.2.5 INT	209
6.12.2.6 INTEL486DX	209
6.12.2.7 INTEL486SX	209
6.12.2.8 INVALID	210
6.12.2.9 IO_APIC	210
6.12.2.10 IO_INTRPT_ASS	210
6.12.2.11 LOC_INTRPT_ASS	210

6.12.2.12 NCPUS	210
6.12.2.13 NMI	210
6.12.2.14 PROCESSOR	210
6.12.2.15 SMI	210
6.12.2.16 SMPBOOT_START	210
6.12.2.17 TRAMPOLINE_START	210
6.12.3 Variable Documentation	211
6.12.3.1 __attribute__	211
6.13 include/arch/x86/processor.h File Reference	211
6.13.1 Detailed Description	214
6.13.2 Define Documentation	215
6.13.2.1 FLAG_AC	215
6.13.2.2 FLAG_AF	215
6.13.2.3 FLAG_CF	215
6.13.2.4 FLAG_DF	215
6.13.2.5 FLAG_ID	215
6.13.2.6 FLAG_IF	215
6.13.2.7 FLAG_IOPL	215
6.13.2.8 FLAG_NT	215
6.13.2.9 FLAG_OF	215
6.13.2.10 FLAG_PF	215
6.13.2.11 FLAG_RF	216
6.13.2.12 FLAG_SF	216
6.13.2.13 FLAG_TF	216
6.13.2.14 FLAG_VIF	216
6.13.2.15 FLAG_VIP	216
6.13.2.16 FLAG_VM	216
6.13.2.17 FLAG_ZF	216
6.13.2.18 LFENCE	216
6.13.2.19 MFENCE	216
6.13.2.20 SFENCE	216
6.13.2.21 VENDOR_AMD	217
6.13.2.22 VENDOR_INTEL	217
6.13.2.23 X86_CR0_AM	217

6.13.2.24 X86_CR0_CD	217
6.13.2.25 X86_CR0_EM	217
6.13.2.26 X86_CR0_ET	217
6.13.2.27 X86_CR0_MP	217
6.13.2.28 X86_CR0_NE	217
6.13.2.29 X86_CR0_NW	217
6.13.2.30 X86_CR0_PE	217
6.13.2.31 X86_CR0_PG	218
6.13.2.32 X86_CR0_TS	218
6.13.2.33 X86_CR0_WP	218
6.13.2.34 X86_CR3_PCD	218
6.13.2.35 X86_CR3_PWT	218
6.13.2.36 X86_CR4_DE	218
6.13.2.37 X86_CR4_MCE	218
6.13.2.38 X86_CR4_OSFXSR	218
6.13.2.39 X86_CR4_OSMXMMEXCPT	218
6.13.2.40 X86_CR4_OSXSAVE	218
6.13.2.41 X86_CR4_PAE	219
6.13.2.42 X86_CR4_PCE	219
6.13.2.43 X86_CR4_PGE	219
6.13.2.44 X86_CR4_PSE	219
6.13.2.45 X86_CR4_PVI	219
6.13.2.46 X86_CR4_TSD	219
6.13.2.47 X86_CR4_VME	219
6.13.2.48 X86_CR4_VMXE	219
6.13.2.49 X86_CR8_TPR	219
6.13.3 Function Documentation	219
6.13.3.1 __attribute__	219
6.13.4 Variable Documentation	219
6.13.4.1 __attribute__	220
6.13.4.2 __pad0__	220
6.13.4.3 __pad1__	220
6.13.4.4 __pad2__	220
6.13.4.5 __pad3__	220

6.13.4.6 __pad4__	220
6.13.4.7 acpi	220
6.13.4.8 apic	220
6.13.4.9 assoc	220
6.13.4.10 brand	220
6.13.4.11 cache	220
6.13.4.12 cmov	221
6.13.4.13 dca	221
6.13.4.14 de	221
6.13.4.15 est	221
6.13.4.16 exfamily	221
6.13.4.17 exmodel	221
6.13.4.18 family	221
6.13.4.19 fms	221
6.13.4.20 fpu	221
6.13.4.21 fxsr	221
6.13.4.22 generic	221
6.13.4.23 htt	221
6.13.4.24 id	222
6.13.4.25 inclusive	222
6.13.4.26 indexing	222
6.13.4.27 line_size	222
6.13.4.28 max_id	222
6.13.4.29 mca	222
6.13.4.30 mmx	222
6.13.4.31 model	222
6.13.4.32 pae	222
6.13.4.33 pat	222
6.13.4.34 pbe	223
6.13.4.35 pcid	223
6.13.4.36 pge	223
6.13.4.37 phy_line	223
6.13.4.38 pse	223
6.13.4.39 sep	223

6.13.4.40 smx	223
6.13.4.41 ss	223
6.13.4.42 sse	223
6.13.4.43 sse2	223
6.13.4.44 sse3	224
6.13.4.45 sse4_1	224
6.13.4.46 sse4_2	224
6.13.4.47 sss	224
6.13.4.48 ssse3	224
6.13.4.49 stepping	224
6.13.4.50 tm	224
6.13.4.51 tm2	224
6.13.4.52 type	224
6.13.4.53 vme	224
6.13.4.54 vmx	225
6.13.4.55 write_back	225
6.13.4.56 x2apic	225
6.14 include/arch/x86/spinlock.h File Reference	225
6.14.1 Function Documentation	226
6.14.1.1 acquire_lock	226
6.14.2 Variable Documentation	226
6.14.2.1 __attribute__	226
6.15 include/synchronization/spinlock.h File Reference	226
6.15.1 Define Documentation	227
6.15.1.1 spinlock_lock	227
6.15.1.2 spinlock_unlock	227
6.15.2 Typedef Documentation	227
6.15.2.1 spinlock_t	227
6.16 include/arch/x86/vectors.h File Reference	228
6.16.1 Variable Documentation	230
6.16.1.1 vector_0	230
6.16.1.2 vector_1	230
6.16.1.3 vector_10	230
6.16.1.4 vector_11	230

6.16.1.5 vector_12	230
6.16.1.6 vector_13	230
6.16.1.7 vector_14	230
6.16.1.8 vector_15	230
6.16.1.9 vector_16	230
6.16.1.10 vector_17	230
6.16.1.11 vector_18	230
6.16.1.12 vector_19	230
6.16.1.13 vector_2	230
6.16.1.14 vector_20	230
6.16.1.15 vector_21	230
6.16.1.16 vector_22	230
6.16.1.17 vector_23	230
6.16.1.18 vector_24	230
6.16.1.19 vector_25	230
6.16.1.20 vector_26	230
6.16.1.21 vector_27	230
6.16.1.22 vector_28	230
6.16.1.23 vector_29	230
6.16.1.24 vector_3	231
6.16.1.25 vector_30	231
6.16.1.26 vector_31	231
6.16.1.27 vector_32	231
6.16.1.28 vector_33	231
6.16.1.29 vector_34	231
6.16.1.30 vector_35	231
6.16.1.31 vector_36	231
6.16.1.32 vector_37	231
6.16.1.33 vector_38	231
6.16.1.34 vector_39	231
6.16.1.35 vector_4	231
6.16.1.36 vector_40	231
6.16.1.37 vector_41	231
6.16.1.38 vector_42	231

6.16.1.39 vector_43	231
6.16.1.40 vector_44	231
6.16.1.41 vector_45	231
6.16.1.42 vector_46	231
6.16.1.43 vector_47	231
6.16.1.44 vector_48	231
6.16.1.45 vector_49	231
6.16.1.46 vector_5	231
6.16.1.47 vector_50	231
6.16.1.48 vector_51	232
6.16.1.49 vector_52	232
6.16.1.50 vector_53	232
6.16.1.51 vector_54	232
6.16.1.52 vector_55	232
6.16.1.53 vector_56	232
6.16.1.54 vector_57	232
6.16.1.55 vector_58	232
6.16.1.56 vector_59	232
6.16.1.57 vector_6	232
6.16.1.58 vector_60	232
6.16.1.59 vector_61	232
6.16.1.60 vector_62	232
6.16.1.61 vector_63	232
6.16.1.62 vector_7	232
6.16.1.63 vector_8	232
6.16.1.64 vector_9	232
6.17 include/arch/x86/x86.h File Reference	233
6.17.1 Function Documentation	234
6.17.1.1 cli	234
6.17.1.2 inb	234
6.17.1.3 inl	234
6.17.1.4 insb	234
6.17.1.5 insl	234
6.17.1.6 insw	235

6.17.1.7	invlpg	235
6.17.1.8	inw	235
6.17.1.9	outb	235
6.17.1.10	outl	235
6.17.1.11	outw	235
6.17.1.12	read_cr0	235
6.17.1.13	read_cr1	235
6.17.1.14	read_cr2	235
6.17.1.15	read_cr3	235
6.17.1.16	read_cr4	236
6.17.1.17	read_esp	236
6.17.1.18	read_msr	236
6.17.1.19	sti	236
6.17.1.20	write_cr0	236
6.17.1.21	write_cr1	236
6.17.1.22	write_cr2	236
6.17.1.23	write_cr3	236
6.17.1.24	write_cr4	237
6.17.1.25	write_msr	237
6.17.1.26	write_tr	237
6.18	arch/x86/include/x86.h File Reference	237
6.19	arch/x86/x86.h File Reference	238
6.19.1	Function Documentation	239
6.19.1.1	cli	239
6.19.1.2	cpuid	239
6.19.1.3	inb	239
6.19.1.4	inl	239
6.19.1.5	insb	239
6.19.1.6	insl	239
6.19.1.7	insw	239
6.19.1.8	invlpg	239
6.19.1.9	inw	239
6.19.1.10	outb	240
6.19.1.11	outl	240

6.19.1.12	outw	240
6.19.1.13	read_cr0	240
6.19.1.14	read_cr1	240
6.19.1.15	read_cr2	240
6.19.1.16	read_cr3	240
6.19.1.17	read_cr4	240
6.19.1.18	sti	240
6.19.1.19	write_cr0	240
6.19.1.20	write_cr1	241
6.19.1.21	write_cr2	241
6.19.1.22	write_cr3	241
6.19.1.23	write_cr4	241
6.20	include/cli.h File Reference	241
6.20.1	Define Documentation	242
6.20.1.1	MAXBUFSIZE	242
6.20.2	Variable Documentation	242
6.20.2.1	buf	242
6.20.2.2	cons	242
6.20.2.3	rpos	242
6.20.2.4	wpos	242
6.21	include/cmos.h File Reference	242
6.21.1	Detailed Description	244
6.21.2	Define Documentation	244
6.21.2.1	CMOS_DATAPORT	244
6.21.2.2	CMOS_DRIVE_C	244
6.21.2.3	CMOS_DRIVE_D	244
6.21.2.4	CMOS_EQUIP_BYTE	244
6.21.2.5	CMOS_EXTMEM_LSB	244
6.21.2.6	CMOS_EXTMEM_MSB	245
6.21.2.7	CMOS_FIXED_DISK	245
6.21.2.8	CMOS_INDEXPORT	245
6.21.2.9	CMOS_STATUS_10	245
6.21.2.10	CMOS_STATUS_C	245
6.21.2.11	CMOS_STATUS_D	245

6.21.2.12 CMOS_STATUS_E	245
6.21.2.13 CMOS_STATUS_F	245
6.21.2.14 CMOS_SYSBASE_LSB	245
6.21.2.15 CMOS_SYSBASE_MSB	245
6.21.2.16 RTC_ALRMHOUR	246
6.21.2.17 RTC_ALRMMINUTE	246
6.21.2.18 RTC_ALARMSECOND	246
6.21.2.19 RTC_DAY_MONTH	246
6.21.2.20 RTC_DAY_WEEK	246
6.21.2.21 RTC_HOUR	246
6.21.2.22 RTC_MINUTES	246
6.21.2.23 RTC_MONTH	246
6.21.2.24 RTC_SECONDS	246
6.21.2.25 RTC_STATUS_A	246
6.21.2.26 RTC_STATUS_B	247
6.21.2.27 RTC_YEAR	247
6.21.2.28 STAT_ALRM_INTR	247
6.21.2.29 STAT_CAL_BCD	247
6.21.2.30 STAT_CAL_BIN	247
6.21.2.31 STAT_CAL_HR12	247
6.21.2.32 STAT_CAL_HR24	247
6.21.2.33 STAT_DAY_LGHT	247
6.21.2.34 STAT_HALT	247
6.21.2.35 STAT_PER_INTR	247
6.21.2.36 STAT_RUN	248
6.21.2.37 STAT_SQRWV_INTR	248
6.21.2.38 STAT_UPDT_INTR	248
6.21.3 Function Documentation	248
6.21.3.1 cmos_get_seconds	248
6.22 include/cpuid.h File Reference	248
6.22.1 Define Documentation	249
6.22.1.1 CPUID_CACHE	249
6.22.1.2 CPUID_INFO	249
6.22.1.3 CPUID_INTEL_EBX_0	249

6.22.1.4	CPUID_INTEL_ECX_0	249
6.22.1.5	CPUID_INTEL_EDX_0	249
6.22.1.6	CPUID_P4HT_0	249
6.22.1.7	CPUID_VERSION	249
6.22.2	Function Documentation	249
6.22.2.1	cpuid	249
6.23	include/device.h File Reference	250
6.24	include/drivers/i8254.h File Reference	250
6.24.1	Detailed Description	251
6.24.2	Define Documentation	251
6.24.2.1	LATCH_ALL	251
6.24.2.2	LATCH_LSB	251
6.24.2.3	LATCH_MSB	251
6.24.2.4	MODE_HARDSRB	252
6.24.2.5	MODE_ONESHOT	252
6.24.2.6	MODE_RATEGEN	252
6.24.2.7	MODE_SOFTSRB	252
6.24.2.8	MODE_WAVEGEN	252
6.24.2.9	PIT_CHANNEL0	252
6.24.2.10	PIT_CHANNEL1	252
6.24.2.11	PIT_CHANNEL2	252
6.24.2.12	PIT_MODE	252
6.24.2.13	PIT_TICKS	252
6.24.2.14	SELECT_CNT0	253
6.24.2.15	SELECT_CNT1	253
6.24.2.16	SELECT_CNT2	253
6.25	include/drivers/i8259.h File Reference	253
6.25.1	Detailed Description	255
6.25.2	Define Documentation	256
6.25.2.1	EXC_BASE	256
6.25.2.2	EXC_SIZE	256
6.25.2.3	ICW1_4BTVECTORS	256
6.25.2.4	ICW1_ICW4NEEDED	256
6.25.2.5	ICW1_ISSUED	256

6.25.2.6 ICW1_LEVELMODE	256
6.25.2.7 ICW1_SINGLEMODE	256
6.25.2.8 ICW4_86_88	256
6.25.2.9 ICW4_AUTOEOI	256
6.25.2.10 ICW4_FULLNESTED	256
6.25.2.11 ICW4_MASTER_BUF	257
6.25.2.12 ICW4_NONBUFFER	257
6.25.2.13 ICW4_SLAVE_BUF	257
6.25.2.14 INTR_EMPTY_BASE	257
6.25.2.15 INTR_END	257
6.25.2.16 IRQ_BASE	257
6.25.2.17 IRQ_END	257
6.25.2.18 IRQ_FLPY	257
6.25.2.19 IRQ_FPU	257
6.25.2.20 IRQ_KBD	257
6.25.2.21 IRQ_LPT2	258
6.25.2.22 IRQ_MOUSE	258
6.25.2.23 IRQ_PIT	258
6.25.2.24 IRQ_PRIHD	258
6.25.2.25 IRQ_RTC	258
6.25.2.26 IRQ_SECHD	258
6.25.2.27 IRQ_SER1	258
6.25.2.28 IRQ_SER2	258
6.25.2.29 IRQ_SIZE	258
6.25.2.30 IRQ_SPUR	258
6.25.2.31 IRQ_VID	259
6.25.2.32 OCW1_FDISK	259
6.25.2.33 OCW1_FLPPY	259
6.25.2.34 OCW1_FPU	259
6.25.2.35 OCW1_INTRS_OFF	259
6.25.2.36 OCW1_KYBRD	259
6.25.2.37 OCW1_MOUSE	259
6.25.2.38 OCW1_PIT	259
6.25.2.39 OCW1_PPRNT	259

6.25.2.40 OCW1_PRIHD	259
6.25.2.41 OCW1_RTC	260
6.25.2.42 OCW1_SECHD	260
6.25.2.43 OCW1_SERI1	260
6.25.2.44 OCW1_SERI2	260
6.25.2.45 OCW1_VIDEO	260
6.25.2.46 OCW3_READIRR	260
6.25.2.47 OCW3_READISR	260
6.25.2.48 PIC_EOI	260
6.25.2.49 PIC_MASTER_ICW	260
6.25.2.50 PIC_MASTER_OCW	260
6.25.2.51 PIC_SLAVE_0	261
6.25.2.52 PIC_SLAVE_1	261
6.25.2.53 PIC_SLAVE_2	261
6.25.2.54 PIC_SLAVE_3	261
6.25.2.55 PIC_SLAVE_4	261
6.25.2.56 PIC_SLAVE_5	261
6.25.2.57 PIC_SLAVE_6	261
6.25.2.58 PIC_SLAVE_7	261
6.25.2.59 PIC_SLAVE_ICW	261
6.25.2.60 PIC_SLAVE_OCW	261
6.25.2.61 SLAVE_ON_0	262
6.25.2.62 SLAVE_ON_1	262
6.25.2.63 SLAVE_ON_2	262
6.25.2.64 SLAVE_ON_3	262
6.25.2.65 SLAVE_ON_4	262
6.25.2.66 SLAVE_ON_5	262
6.25.2.67 SLAVE_ON_6	262
6.25.2.68 SLAVE_ON_7	262
6.26 include/fs/ext2fs.h File Reference	263
6.26.1 Define Documentation	265
6.26.1.1 EXT2_64FZ	265
6.26.1.2 EXT2_ACL_DATA_INO	265
6.26.1.3 EXT2_ACL_IDX_INO	265

6.26.1.4 EXT2_AFS_INO	265
6.26.1.5 EXT2_BAD_INO	265
6.26.1.6 EXT2_BOOT_LOADER_INO	265
6.26.1.7 EXT2_CLEAN	266
6.26.1.8 EXT2_COMPRESSION	266
6.26.1.9 EXT2_DIND_BLOCK	266
6.26.1.10 EXT2_DIR_BIN_TREE	266
6.26.1.11 EXT2_DIR_HASHI	266
6.26.1.12 EXT2_DIR_TYPE_F	266
6.26.1.13 EXT2_ERR0R	266
6.26.1.14 EXT2_FIRST_INO	266
6.26.1.15 EXT2_FS_JOR_DEV	266
6.26.1.16 EXT2_FS_REPLAY_JOR	266
6.26.1.17 EXT2_FS_RESIZE	267
6.26.1.18 EXT2_IGNORE	267
6.26.1.19 EXT2_IND_BLOCK	267
6.26.1.20 EXT2_INO_EXTNDATTR	267
6.26.1.21 EXT2_INOSZ_V0	267
6.26.1.22 EXT2_JRNL	267
6.26.1.23 EXT2_KPANIC	267
6.26.1.24 EXT2_LINK_MAX	267
6.26.1.25 EXT2_N_BLOCKS	267
6.26.1.26 EXT2_NAME_LEN	267
6.26.1.27 EXT2_NDIR_BLOCKS	268
6.26.1.28 EXT2_OS_FBSD	268
6.26.1.29 EXT2_OS_HURD	268
6.26.1.30 EXT2_OS_LINUX	268
6.26.1.31 EXT2_OS_LITES	268
6.26.1.32 EXT2_OS_MASIX	268
6.26.1.33 EXT2_PRE_02B_MAGIC	268
6.26.1.34 EXT2_PRELOC	268
6.26.1.35 EXT2_REMOUNT_RO	268
6.26.1.36 EXT2_ROOT_INO	268
6.26.1.37 EXT2_ROOT_INODE	269

6.26.1.38 EXT2_SPARSE	269
6.26.1.39 EXT2_SUPER_MAGIC	269
6.26.1.40 EXT2_TIND_BLOCK	269
6.26.1.41 EXT2_UNDEL_DIR_INO	269
6.26.1.42 I_AFS_DIR	269
6.26.1.43 I_APPEND_ONLY	269
6.26.1.44 I_BLOCK_DEVICE	269
6.26.1.45 I_CHAR_DEVICE	269
6.26.1.46 I_DIR_BLOCK_DEV	269
6.26.1.47 I_DIR_CHAR_DEV	270
6.26.1.48 I_DIR_DIRECTORY	270
6.26.1.49 I_DIR_FIFO	270
6.26.1.50 I_DIR_REGULAR_F	270
6.26.1.51 I_DIR SOCK	270
6.26.1.52 I_DIR_SYM_LINK	270
6.26.1.53 I_DIR_UNKOWN	270
6.26.1.54 I_DIRECTORY	270
6.26.1.55 I_FIFO	270
6.26.1.56 I_FILE_COMPRESSION	270
6.26.1.57 I_GEXEC_P	271
6.26.1.58 I_GREAD_P	271
6.26.1.59 I_GWRITE_P	271
6.26.1.60 I_HASH_INDEXED_DIR	271
6.26.1.61 I_IMMUTABLE_FILE	271
6.26.1.62 I_JOURNAL_FILE_DATA	271
6.26.1.63 I_KEEP_CPY	271
6.26.1.64 I_NO_DUMP	271
6.26.1.65 I_NO_LAT	271
6.26.1.66 I_OEXEC_P	271
6.26.1.67 I_OREAD_P	272
6.26.1.68 I_OWRITE_P	272
6.26.1.69 I_REG_FILE	272
6.26.1.70 I_SEC_DEL	272
6.26.1.71 I_SET_GID	272

6.26.1.72 I_SET_UID	272
6.26.1.73 I_STICKY	272
6.26.1.74 I_SYM_LINK	272
6.26.1.75 I_SYNC_UPDATES	272
6.26.1.76 I_UEXEC_P	272
6.26.1.77 I_UNIX SOCK	273
6.26.1.78 I_UREAD_P	273
6.26.1.79 I_UWRITE_P	273
6.27 include/init.h File Reference	274
6.27.1 Define Documentation	275
6.27.1.1 ELFHDR	275
6.27.1.2 SECTOR	275
6.27.2 Function Documentation	275
6.27.2.1 Init_userspace	275
6.28 include/initrd_mem.h File Reference	276
6.29 include/kbc.h File Reference	277
6.29.1 Define Documentation	279
6.29.1.1 KBC_AUXWRITE	279
6.29.1.2 KBC_COMMAND	279
6.29.1.3 KBC_DATAIN	279
6.29.1.4 KBC_DATAPORT	279
6.29.1.5 KBC_DEFAULT	279
6.29.1.6 KBC_DIGDUMP	280
6.29.1.7 KBC_DISA20	280
6.29.1.8 KBC_DISABLE	280
6.29.1.9 KBC_DISKBD	280
6.29.1.10 KBC_DISMOUSE	280
6.29.1.11 KBC_ECHODIG	280
6.29.1.12 KBC_ENABLE	280
6.29.1.13 KBC_ENBA20	280
6.29.1.14 KBC_ENBKBD	280
6.29.1.15 KBC_ENMOUSE	280
6.29.1.16 KBC_FULLBUF	281
6.29.1.17 KBC_INDICATOR	281

6.29.1.18 KBC_INPREAD	281
6.29.1.19 KBC_INTRTEST	281
6.29.1.20 KBC_OUTAUX	281
6.29.1.21 KBC_OUTKBD	281
6.29.1.22 KBC_OUTREAD	281
6.29.1.23 KBC_OUTWRTE	281
6.29.1.24 KBC_PARITY	281
6.29.1.25 KBC_PULSE0	282
6.29.1.26 KBC_PULSE1	282
6.29.1.27 KBC_PULSE2	282
6.29.1.28 KBC_PULSE3	282
6.29.1.29 KBC_READID	282
6.29.1.30 KBC_READLOW	282
6.29.1.31 KBC_READRAM	282
6.29.1.32 KBC_READVER	282
6.29.1.33 KBC_REBOOT	282
6.29.1.34 KBC resend	282
6.29.1.35 KBC_RESET	283
6.29.1.36 KBC_RTIMEOUT	283
6.29.1.37 KBC_SECLOCK	283
6.29.1.38 KBC_SELFTEST	283
6.29.1.39 KBC_SETSCAN	283
6.29.1.40 KBC_STATUSPORT	283
6.29.1.41 KBC_TSTMOUSE	283
6.29.1.42 KBC_TTIMEOUT	283
6.29.1.43 KBC_TYPEMATIC	283
6.29.1.44 KBC_WITERAM	283
6.29.1.45 KEY_DEL	284
6.29.1.46 KEY_DN	284
6.29.1.47 KEY_END	284
6.29.1.48 KEY_HOME	284
6.29.1.49 KEY_INS	284
6.29.1.50 KEY_LF	284
6.29.1.51 KEY_PGDN	284

6.29.1.52 KEY_PGUP	284
6.29.1.53 KEY_RT	284
6.29.1.54 KEY_UP	284
6.30 include/kconsole.h File Reference	285
6.30.1 Detailed Description	286
6.30.2 Define Documentation	286
6.30.2.1 KCONSOLE_MAX_CMDS	286
6.30.3 Typedef Documentation	287
6.30.3.1 kcommand_t	287
6.31 include/memvals.h File Reference	287
6.31.1 Define Documentation	288
6.31.1.1 EXTMEM	288
6.31.1.2 GB	288
6.31.1.3 IOPHYMEM	288
6.31.1.4 KB	288
6.31.1.5 KERNEL_ADDR	288
6.31.1.6 KERNEL_STACK	289
6.31.1.7 KERNEL_STACK_TOP	289
6.31.1.8 MB	289
6.31.1.9 PA	289
6.31.1.10 PAGECNT	289
6.31.1.11 PAGELG	289
6.31.1.12 PAGESZ	289
6.31.1.13 PAGETSZ	289
6.31.1.14 PROC_LIST	289
6.31.1.15 SEG_NULL	290
6.31.1.16 SEGACS_D	290
6.31.1.17 SEGACS_RW	290
6.31.1.18 SEGACS_USR	290
6.31.1.19 SEGACS_X	290
6.31.1.20 SEGMENT	290
6.31.1.21 USEREND	290
6.31.1.22 USERPAGES	291
6.31.1.23 USERSPACESTART	291

6.31.1.24 USERSTACK_TOP	291
6.31.1.25 USERSTART	291
6.31.1.26 USERVIRTPGT	291
6.31.1.27 VIRTPGT	291
6.31.2 Function Documentation	291
6.31.2.1 __attribute__	291
6.31.3 Variable Documentation	291
6.31.3.1 base	291
6.31.3.2 size	291
6.32 include/mm/mm.h File Reference	291
6.32.1 Detailed Description	293
6.32.2 Define Documentation	293
6.32.2.1 _CATERNEL_MM_MM_H_	293
6.33 include/multiboot.h File Reference	293
6.33.1 Detailed Description	295
6.33.2 Define Documentation	296
6.33.2.1 MULTIBOOT_AOUTF_FLAG_KLUDGE	296
6.33.2.2 MULTIBOOT_BOOTLOADER_MAGIC	296
6.33.2.3 MULTIBOOT_FLAG_AOUT	296
6.33.2.4 MULTIBOOT_FLAG_APM	296
6.33.2.5 MULTIBOOT_FLAG_CMDLINE	296
6.33.2.6 MULTIBOOT_FLAG_CONFIG	296
6.33.2.7 MULTIBOOT_FLAG_DEVICE	296
6.33.2.8 MULTIBOOT_FLAG_ELF	296
6.33.2.9 MULTIBOOT_FLAG_LOADER	296
6.33.2.10 MULTIBOOT_FLAG_MEM	296
6.33.2.11 MULTIBOOT_FLAG_MMAP	297
6.33.2.12 MULTIBOOT_FLAG_MODS	297
6.33.2.13 MULTIBOOT_FLAG_VBE	297
6.33.2.14 MULTIBOOT_HEADER_MAGIC	297
6.33.2.15 MULTIBOOT_INFO_ALIGN	297
6.33.2.16 MULTIBOOT_MEMORY_AVAILABLE	297
6.33.2.17 MULTIBOOT_MEMORY_RESERVED	297
6.33.2.18 MULTIBOOT_MOD_ALIGN	297

6.33.2.19 MULTIBOOT_SEARCH	297
6.33.2.20 MULTIBOOT_UNSUPPORTED	297
6.33.3 Variable Documentation	298
6.33.3.1 __attribute__	298
6.34 include/proc/proc.h File Reference	298
6.34.1 Detailed Description	300
6.34.2 Define Documentation	300
6.34.2.1 BLOCKED	300
6.34.2.2 MAX_PROC_NAME	300
6.34.2.3 MAX_PROCS	300
6.34.2.4 NON_RUNNABLE	300
6.34.2.5 PROC_EMPTY	301
6.34.2.6 PROC_TABLE_SIZE	301
6.34.2.7 RUNNABLE	301
6.34.3 Typedef Documentation	301
6.34.3.1 proc_t	301
6.34.4 Function Documentation	301
6.34.4.1 LIFO_HEAD	301
6.34.4.2 LIST_HEAD	301
6.34.4.3 proc_setup	301
6.34.4.4 proc_setup_mem	301
6.35 include/rdisk.h File Reference	302
6.35.1 Define Documentation	303
6.35.1.1 ELF_MAGIC2	303
6.35.1.2 SECTOR	303
6.36 include/stdarg.h File Reference	303
6.36.1 Define Documentation	304
6.36.1.1 __va_size	304
6.36.1.2 va_arg	304
6.36.1.3 va_end	304
6.36.1.4 va_start	304
6.36.2 Typedef Documentation	304
6.36.2.1 va_list	304
6.37 include/stdio.h File Reference	304

6.37.1	Detailed Description	306
6.38	include/string.h File Reference	306
6.38.1	Detailed Description	307
6.38.2	Function Documentation	307
6.38.2.1	memcmp	307
6.38.2.2	memmove	307
6.38.2.3	strcat	307
6.38.2.4	strchr	307
6.38.2.5	strcpy	307
6.38.2.6	strlcat	307
6.38.2.7	strlcpy	307
6.38.2.8	strlen	307
6.38.2.9	strncat	308
6.38.2.10	strnchr	308
6.38.2.11	strncmp	308
6.38.2.12	strncpy	308
6.38.2.13	strrchr	308
6.39	include/structs/btree.h File Reference	308
6.39.1	Detailed Description	309
6.39.2	Typedef Documentation	309
6.39.2.1	btree_node_t	309
6.39.3	Function Documentation	309
6.39.3.1	btree_get	309
6.39.3.2	btree_init	309
6.39.3.3	btree_insert	309
6.39.3.4	btree_remove	309
6.39.3.5	LIST_HEAD	309
6.40	include/structs/htable.h File Reference	309
6.40.1	Detailed Description	311
6.40.2	Function Documentation	311
6.40.2.1	table_lookup	311
6.41	include/structs/linkedlist.h File Reference	312
6.41.1	Define Documentation	312
6.41.1.1	LIST_EMPTY	312

6.41.1.2 LIST_ENTRY	312
6.41.1.3 LIST_FIRST	313
6.41.1.4 LIST_FOREACH	313
6.41.1.5 LIST_HEAD	313
6.41.1.6 LIST_HEAD_INIT	313
6.41.1.7 LIST_INIT	313
6.41.1.8 LIST_INSERT_AFTER	313
6.41.1.9 LIST_INSERT_BEFORE	314
6.41.1.10 LIST_INSERT_HEAD	314
6.41.1.11 LIST_NEXT	314
6.41.1.12 LIST_REMOVE	314
6.42 include/structs/list.h File Reference	315
6.43 include/structs/queue.h File Reference	316
6.43.1 Define Documentation	317
6.43.1.1 FIFO_EMPTY	317
6.43.1.2 FIFO_ENTRY	317
6.43.1.3 FIFO_FIRST	317
6.43.1.4 FIFO_HEAD	318
6.43.1.5 FIFO_INIT	318
6.43.1.6 FIFO_POP	318
6.43.1.7 FIFO_PUSH	318
6.43.1.8 LIFO_EMPTY	318
6.43.1.9 LIFO_ENTRY	318
6.43.1.10 LIFO_FIRST	318
6.43.1.11 LIFO_HEAD	319
6.43.1.12 LIFO_HEAD_INIT	319
6.43.1.13 LIFO_INIT	319
6.43.1.14 LIFO_POP	319
6.43.1.15 LIFO_PUSH	319
6.44 include/synchronization/semaphore.h File Reference	319
6.44.1 Detailed Description	321
6.45 include/synchronization/wait_queue.h File Reference	322
6.46 include/sys.h File Reference	324
6.46.1 Define Documentation	324

6.46.1.1	S_PRINTF	324
6.47	include/syscalls/syscalls.h File Reference	325
6.48	include/test.h File Reference	326
6.48.1	Define Documentation	327
6.48.1.1	assert	327
6.48.1.2	panic	327
6.49	include/time/time.h File Reference	327
6.49.1	Detailed Description	327
6.50	include/types.h File Reference	328
6.50.1	Define Documentation	328
6.50.1.1	NULL	328
6.50.1.2	ROUND_DOWN	329
6.50.1.3	ROUND_UP	329
6.50.2	Typedef Documentation	329
6.50.2.1	fnptr_t	329
6.50.2.2	int16_t	329
6.50.2.3	int32_t	329
6.50.2.4	int64_t	329
6.50.2.5	int8_t	329
6.50.2.6	intmax_t	329
6.50.2.7	paddr_t	330
6.50.2.8	reg_t	330
6.50.2.9	size_t	330
6.50.2.10	uint16_t	330
6.50.2.11	uint32_t	330
6.50.2.12	uint64_t	330
6.50.2.13	uint8_t	330
6.50.2.14	uintmax_t	330
6.50.2.15	uintptr_t	330
6.50.2.16	vaddr_t	330
6.51	arch/x86/include/types.h File Reference	331
6.51.1	Typedef Documentation	331
6.51.1.1	int16_t	331
6.51.1.2	int32_t	331

6.51.1.3	int64_t	331
6.51.1.4	int8_t	331
6.51.1.5	uint16_t	331
6.51.1.6	uint32_t	331
6.51.1.7	uint64_t	332
6.51.1.8	uint8_t	332
6.52	include/video.h File Reference	332
6.52.1	Define Documentation	333
6.52.1.1	BACKGROUND_BLINK	333
6.52.1.2	BACKGROUND_BLUE	333
6.52.1.3	BACKGROUND_GRAY	333
6.52.1.4	BACKGROUND_GREEN	333
6.52.1.5	BACKGROUND_RED	333
6.52.1.6	BACKGROUND_WHITE	334
6.52.1.7	CGA_BUFF_OFF	334
6.52.1.8	CGA_COLS	334
6.52.1.9	CGA_DATA1	334
6.52.1.10	CGA_INDEX1	334
6.52.1.11	CGA_ROWS	334
6.52.1.12	CGA_SIZE	334
6.52.1.13	COLOR_BLUE	334
6.52.1.14	COLOR_DARK_GRAY	334
6.52.1.15	COLOR_GRAY	334
6.52.1.16	COLOR_GREEN	335
6.52.1.17	COLOR_RED	335
6.52.1.18	COLOR_WHITE	335
6.52.2	Variable Documentation	335
6.52.2.1	char_buff	335
6.52.2.2	cursor_position	335
6.53	kernel/arch/x86/bios/bios.c File Reference	336
6.53.1	Define Documentation	336
6.53.1.1	BIOS_EBDA_PTR	336
6.53.2	Function Documentation	337
6.53.2.1	bios_init	337

6.53.3 Variable Documentation	337
6.53.3.1 ebda	337
6.54 kernel/arch/x86/interrupt.c File Reference	337
6.54.1 Detailed Description	338
6.55 kernel/arch/x86/mm/init_mem.c File Reference	338
6.55.1 Detailed Description	340
6.56 kernel/arch/x86/mm/page.c File Reference	340
6.56.1 Detailed Description	341
6.57 kernel/arch/x86/mp/apic.c File Reference	342
6.57.1 Detailed Description	342
6.58 kernel/arch/x86/mp/delay.c File Reference	343
6.58.1 Detailed Description	344
6.59 kernel/arch/x86/mp/smp.c File Reference	344
6.59.1 Detailed Description	345
6.60 kernel/arch/x86/processor.c File Reference	346
6.60.1 Detailed Description	346
6.61 kernel/drivers/clock.c File Reference	347
6.61.1 Detailed Description	347
6.62 kernel/drivers/cmos.c File Reference	347
6.62.1 Detailed Description	348
6.63 kernel/drivers/i8254.c File Reference	348
6.63.1 Detailed Description	349
6.64 kernel/drivers/i8259.c File Reference	349
6.64.1 Detailed Description	350
6.65 kernel/drivers/kbc.c File Reference	351
6.65.1 Detailed Description	352
6.66 kernel/drivers/video.c File Reference	352
6.66.1 Detailed Description	353
6.67 kernel/generic/cli.c File Reference	353
6.67.1 Detailed Description	354
6.68 kernel/generic/cpuid.c File Reference	355
6.68.1 Function Documentation	355
6.68.1.1 cpuid_get_eax	355
6.68.1.2 cpuid_get_ebx	355

6.68.1.3	cpuid_get_ecx	355
6.68.1.4	cpuid_get_edx	355
6.68.1.5	cpuid_print	356
6.68.1.6	make_cpuid	356
6.68.1.7	set_eax	356
6.69	kernel/generic/init.c File Reference	356
6.69.1	Function Documentation	357
6.69.1.1	Init_userspace	357
6.69.2	Variable Documentation	358
6.69.2.1	processes	358
6.70	kernel/generic/initrd_mem.c File Reference	358
6.71	kernel/generic/kconsole.c File Reference	359
6.72	kernel/generic/play.c File Reference	359
6.73	kernel/generic/printf.c File Reference	360
6.73.1	Detailed Description	361
6.74	kernel/generic/rdisk.c File Reference	362
6.74.1	Function Documentation	363
6.74.1.1	readsect	363
6.74.1.2	readseg	363
6.74.1.3	waitdisk	363
6.75	kernel/generic/readline.c File Reference	364
6.75.1	Detailed Description	364
6.76	kernel/generic/string.c File Reference	365
6.76.1	Detailed Description	366
6.77	kernel/generic/work_it_out.c File Reference	366
6.77.1	Detailed Description	367
6.78	kernel/mm/mm.c File Reference	368
6.78.1	Detailed Description	369
6.79	kernel/perf/sort/mergesort.c File Reference	369
6.79.1	Detailed Description	369
6.80	kernel/perf/sort/quicksort.c File Reference	370
6.81	kernel/proc/load_elf.c File Reference	370
6.81.1	Detailed Description	371
6.82	kernel/proc/proc.c File Reference	372

6.82.1	Detailed Description	373
6.83	kernel/structs/htable.c File Reference	373
6.83.1	Detailed Description	375
6.84	kernel/synchronization/mutex.c File Reference	376
6.84.1	Detailed Description	377
6.85	kernel/synchronization/semaphore.c File Reference	378
6.85.1	Detailed Description	379
6.86	kernel/synchronization/spinlock.c File Reference	380
6.87	kernel/synchronization/wait_queue.c File Reference	380
6.87.1	Detailed Description	382
6.88	kernel/syscalls/s_print.c File Reference	382
6.88.1	Detailed Description	383
6.89	kernel/syscalls/syscalls.c File Reference	383
6.89.1	Detailed Description	384
6.90	kernel/time/periodic.c File Reference	384
6.90.1	Detailed Description	384
6.91	kernel/tmp/init_elf.c File Reference	385
6.91.1	Function Documentation	385
6.91.1.1	main	385
6.92	kernel/vfs/initrd.c File Reference	385
6.92.1	Detailed Description	386
6.93	kernel/vfs/initrd.h File Reference	386
6.93.1	Detailed Description	388
6.94	kernel/vfs/vfs.c File Reference	388
6.94.1	Detailed Description	389
6.95	kernel/vfs/vfs.h File Reference	389
6.95.1	Detailed Description	391

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Interrupts-and-Syscalls	9
Memory-Management	14
Multiprocessors	23
MultiProcessors	26
CPU-Initialization	33
Drivers	34
Main	45
Category Here--	54
Debug	55
Management	57
Optimization	61
Process-Management	63
Data-Structures	68
Synchronization	70
Time	75
VFS	76
X86-boot	84

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

<code>__attribute</code>	93
<code>__attribute</code>	
Task State segment	96
<code>btree_node</code>	115
<code>btree_t</code>	116
<code>cpu_cache</code>	118
<code>cpu_state_t</code>	
Interrupt/Trap cpu state	120
<code>cpu_t</code>	122
<code>cpu_version</code>	124
<code>cpuid_regs</code>	130
<code>cpuid_t</code>	131
<code>direntry_t</code>	132
<code>element</code>	132
<code>elfhdr</code>	
Elf binary header values	133
<code>ext2_dir_entry</code>	136
<code>ext2_group_desc</code>	136
<code>ext2_inode</code>	138
<code>ext2_super_block</code>	143
<code>Gdtdesc</code>	148
<code>gpr_regs_t</code>	
General purpose registers	149
<code>htable_node_t</code>	150
<code>htable_t</code>	
A hash table structure	151
<code>initrd_file_header_t</code>	152
<code>initrd_header_t</code>	153
<code>kcommand</code>	154

Page	
A page list entry, used to hold info about pages	154
proc	155
proghdr	158
sechdr	159
seg_regs_t	161
Segdesc	162
semaphore_t	163
spinlock	164
spinlock_t	164
vfs_node	165
waiting_proc	167

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

arch/x86/ x86.h	238
arch/x86/boot/ main.c	
Elf binary structures and constants, for parsing binaries	171
arch/x86/include/ elf.h	182
arch/x86/include/ types.h	331
arch/x86/include/ x86.h	237
include/ cli.h	241
include/ cmos.h	
Defines constants for RTC http://catreloaded.net	242
include/ cpuid.h	248
include/ device.h	250
include/ init.h	274
include/ initrd_mem.h	276
include/ kbc.h	277
include/ kconsole.h	
Kernel console for debugging ease	285
include/ memvals.h	287
include/ multiboot.h	
Multiboot specifications	293
include/ rdisk.h	302
include/ stdarg.h	303
include/ stdio.h	
Standard Input Output operations	304
include/ string.h	
A header file for string and memory operations	306
include/ sys.h	324
include/ test.h	326
include/ types.h	328
include/ video.h	332

include/arch/x86/asm.h	172
include/arch/x86/cpu_state.h	
Structures and constants used in storing and determining cpu state	175
include/arch/x86/elf.h	178
include/arch/x86/interrupt.h	
Interrupt structures and constants	184
include/arch/x86/processor.h	
Contains constants that are used by Intel x86	211
include/arch/x86/spinlock.h	225
include/arch/x86/vectors.h	228
include/arch/x86/x86.h	233
include/arch/x86/bios/bios.h	174
include/arch/x86/mm/page.h	
Intel Paging	188
include/arch/x86/mm/segdesc.h	
Segment offset constants	196
include/arch/x86/mp/apic.h	
Support for xAPIC for MPs	197
include/arch/x86/mp/delay.h	
Simple delay functions headers	206
include/arch/x86/mp/smp.h	
Multiprocessor support header	207
include/drivers/i8254.h	
Intel 8254 PIT	250
include/drivers/i8259.h	
Intel 8259 PIC header file	253
include/fs/ext2fs.h	263
include/mm/mm.h	
Memory management	291
include/proc/proc.h	
Process manager header file	298
include/structs/btree.h	
The B-Trees data structure	308
include/structs/htable.h	
Chained Hash Table	309
include/structs/linkedlist.h	312
include/structs/list.h	315
include/structs/queue.h	316
include/synchronization/semaphore.h	
Semaphore structures and function headers	319
include/synchronization/spinlock.h	226
include/synchronization/wait_queue.h	322
include/syscalls/syscalls.h	325
include/time/time.h	
Include all time info	327
kernel/arch/x86/interrupt.c	
Interrupt and fault handling	337
kernel/arch/x86/processor.c	
Identify and initialize cpu structures	346
kernel/arch/x86/bios/bios.c	336

kernel/arch/x86/mm/ init_mem.c	
Virtual Memory initialization	338
kernel/arch/x86/mm/ page.c	
Intel Paging	340
kernel/arch/x86/mp/ apic.c	
Supporting xAPIC for MPs	342
kernel/arch/x86/mp/ delay.c	
Simple delay functions for synchronization	343
kernel/arch/x86/mp/ smp.c	
SMP Support functions	344
kernel/drivers/ clock.c	
Clock Low level controller. sets periodic interrupts and power state .	347
kernel/drivers/ cmos.c	
CMOS Driver	347
kernel/drivers/ i8254.c	
Intel 8253/8254 PIT(Programmable Interrupt Timer) Controller . . .	348
kernel/drivers/ i8259.c	
Intel 8259 PIC(Programmable Interrupt Controller) Low Level Driver	349
kernel/drivers/ kbc.c	
Keyboard driver	351
kernel/drivers/ video.c	
Color Graphics Adapter (CGA) driver	352
kernel/generic/ cli.c	
CLI I/O facilities	353
kernel/generic/ cpuid.c	
.	355
kernel/generic/ init.c	
.	356
kernel/generic/ initrd_mem.c	
.	358
kernel/generic/ kconsole.c	
.	359
kernel/generic/ play.c	
.	359
kernel/generic/ printf.c	
A printf family like functions	360
kernel/generic/ rdisk.c	
.	362
kernel/generic/ readline.c	
Readline operation	364
kernel/generic/ string.c	
String and memory operations, originally from linux	365
kernel/generic/ work_it_out.c	
Kernel device facilities initialization and set up start	366
kernel/mm/ mm.c	
.	368
kernel/perf/sort/ mergesort.c	
Merge Sort implementation	369
kernel/perf/sort/ quicksort.c	
.	370
kernel/proc/ load_elf.c	
Elf binary loader	370
kernel/proc/ proc.c	
Process Manager	372
kernel/structs/ htable.c	
Chained hash table	373
kernel/synchronization/ mutex.c	
.	376
kernel/synchronization/ semaphore.c	
.	378

kernel/synchronization/ spinlock.c	380
kernel/synchronization/ wait_queue.c	380
kernel/syscalls/ s_print.c	
The actual printf procedure -to be implemented-	382
kernel/syscalls/ syscalls.c	
Maps calls into their respective handlers	383
kernel/time/ periodic.c	
Periodic interrupt handler	384
kernel/tmp/ init_elf.c	385
kernel/vfs/ initrd.c	
Initrd	385
kernel/vfs/ initrd.h	
Initrd file system headers	386
kernel/vfs/ vfs.c	
Virtual file system standards , In order to abstract different file system calls	388
kernel/vfs/ vfs.h	
Standardized file system structures and constants	389

Chapter 4

Module Documentation

4.1 Interrupts-and-Syscalls

Files

- file `interrupt.c`
Interrupt and fault handling.
- file `s_print.c`
The actual printf procedure -to be implemented- .
- file `syscalls.c`
maps calls into their respective handlers.

Interrupts Initialization

- gatedesc `idt` [64]
Declare IDT with 255 width interrupts.
- `char * x86_exception_names []`
Fault names.
- `int_generic`
- `void trap (void)`
- `void interrupt_printinfo (void)`
- `void idt_init (void)`
Initializing Interrupt Descriptor table.
- `void register_exception (uint32_t index, char *name, uint16_t present, void(handler)(void))`
register an interrupt handling procedure
- `void interrupt_init (void)`
- `void map_exception (uint32_t int_index, cpu_state_t *cpu_state)`
maps an exception/interrupt to it's handler
- `uint32_t page_fault_handler (cpu_state_t *cpu_state)`
Page Fault Handler.

Print procedure .

- `int32_t sys_printf (char *str)`

The actual print procedure requested by the printf system call.

System calls handler .

- `fnptr_t sys_call_table [] = { sys_printf }`

Declare system calls table with pointers to calls handlers.

- `int32_t n_calls = sizeof(sys_call_table)/sizeof(int32_t)`

keeps the size of the table for boundary checking.

- `int32_t s_errno = 0`

Keeps error codes to return to the caller.

- `int32_t map_syscall (cpu_state_t *cpu_state)`

maps the system call by offseting the sys_call_table structure using the offset in eax register.

4.1.1 Function Documentation

4.1.1.1 void idt_init(void)

Initializing Interrupt Descriptor table.

the idt init fills the idt table with generic interrupt vector which actually does nothing. later on interrupt initialization we'll set specific interrupt numbers to specific vectors [vector_X] is independent proc, that does rely on interrupt number, which is a drawback indeed but it's fixable later on.

Definition at line 82 of file interrupt.c.

4.1.1.2 void interrupt_init(void)

Definition at line 191 of file interrupt.c.

4.1.1.3 void interrupt_printinfo(void)

Definition at line 59 of file interrupt.c.

4.1.1.4 void map_exception(uint32_t int_index, cpu_state_t *cpu_state)

maps an exception/interrupt to it's handler

This function is called by the vector_x procedure and used to map faults to handler function like page fault handler and system call handler and zero division handler..etc.

Definition at line 204 of file interrupt.c.

4.1.1.5 `int32_t map_syscall(cpu_state_t *cpu_state)`

maps the system call by offseting the `sys_call_table` structure using the offset in eax register.

Parameters

<code>cpu_state</code>	: pointer to <code>cpu_state_t</code> structure
------------------------	---

Returns

exit status / error code.

Definition at line 38 of file syscalls.c.

4.1.1.6 `uint32_t page_fault_handler(cpu_state_t *cpu_state)`

[Page](#) Fault Handler.

[Page](#) fault handler

Check if proc is accessing kernel space and not kernel mode.

If from kernel, The kernel panics. if from user space, stack is extended if needed or proc is killed.

Definition at line 262 of file interrupt.c.

4.1.1.7 `void register_exception(uint32_t index, char * name, uint16_t present, void(handler)(void))`

register an interrupt handling procedure

Parameters

<code>index</code>	of the interrupt OR interrupt offset in IDT
<code>*name</code>	the name to be attached to interrupt.
<code>either</code>	interrupt is present or not.
<code>interrupt</code>	handler.

Definition at line 185 of file interrupt.c.

4.1.1.8 `int32_t sys_printf(char * str)`

The actual print procedure requested by the printf system call.

Parameters

<code>str</code>	: format string to print
------------------	--------------------------

Returns

exit status.

Definition at line 21 of file s_print.c.

4.1.1.9 void trap (void)

Definition at line 54 of file interrupt.c.

4.1.2 Variable Documentation**4.1.2.1 gatedesc idt**

Declare IDT with 255 width interrupts.

Definition at line 24 of file interrupt.c.

4.1.2.2 int_generic**4.1.2.3 int32_t n_calls = sizeof(sys_call_table)/sizeof(int32_t)**

keeps the size of the table for boundary checking.

Definition at line 27 of file syscalls.c.

4.1.2.4 int32_t s_errno = 0

Keeps error codes to return to the caller.

Definition at line 31 of file syscalls.c.

4.1.2.5 fnptr_t sys_call_table[] = { sys_printf }

Declare system calls table with pointers to calls handlers.

Definition at line 23 of file syscalls.c.

4.1.2.6 char* x86_exception_names[]**Initial value:**

```
{  
    "Divide Error #DE",  
    "Debug",  
    "NMI",  
    "Breakpoint #BP",  
    "Overflow #OV",  
    "Bound Range Exceeded #BR",
```

```
"Undefined Opcode",
"Device not available",
"Double fault",
"Coprocessor segment overrun",
"Invalid TSS",
"Segment not present",
"Stack Segment Fault",
"General Protection",
"Page Fault",
"\0",
"x87 FPU error",
"Alignment Mask",
"Machine Check",
"SIMD exception"
}
```

Fault names.

Definition at line 29 of file interrupt.c.

4.2 Memory-Management

Files

- file `init_mem.c`
Virtual Memory initialization.
- file `page.c`
Intel Paging.

Virtual Memory Initialization.

Memory size is read and segmentation is reset. and kernel page directory is initialized and mapped. also other segments are initialized and mapped.

- `uint32_t max_addr`
- static `uint32_t mem_base`
- static `uint32_t ext_base`
- `char * next_free = 0`
- static `uint32_t alloc_lock = 0`
- struct `Page * pages`
- `proc_t * proc_table`
- struct `Segdesc catgdt []`
New global descriptor table.
- struct `Gdtdesc gdtdesc`
- `Idtdesc idtdesc`
Interrupt descriptor table descriptor.
- `tss_t tss`
- static `uint32_t x86_read_mem_size (int x)`
Reading memory size using the CMOS RAM/RTC device.
- void `memory_printinfo (void)`
Reads memory size and determine the pages count needed to map it.
- void `scan_memory (void)`
- `void * allocate (uint32_t n, uint32_t align)`
a boot time allocation function, allocate heaps.
- void `init_tss (void)`
sets up the Task state segment and task register.
- void `x86_setup_memory (void)`
main memory initialization function.

Paging.

Programming the hardware support, Initially 32-bit paging is used. and Segment register refer to the whole memory since we'll be operating on protected paging memory mode. and Kernel is mapped to Virtual address 0xF0000000. Paging is set recursively by making cr3 refer to the paging directory.

- `uint32_t page_count`
- `pde_t * global_pkdir`
- `uint32_t global_cr3`
- `char * next_free`
- `struct Page * pages`
- `static struct PageList free_pages`
- `void x86_paging_init (void)`
 - Initiate global pages.*
- `void x86_page_init (struct Page *page)`
 - Initiate a page entry.*
- `int x86_page_alloc (struct Page **page_byref)`
 - Allocate a page page is removed from the free pages list and initiated.*
- `void x86_page_free (struct Page *page)`
 - given page is freed.*
- `void x86_page_detach (struct Page *page)`
 - when [un]refering a page, ref member is decremented.*
- `pte_t * x86_pgdir_find (pde_t *pgdir, const void *va, int allocate)`
 - find a page, if doesn't exist allocate it*
- `struct Page * x86_page_lookup (pde_t *pgdir, void *va, pte_t **pte)`
 - looks up a page refers to vaddr.*
- `void x86_page_remove (pde_t *pgdir, void *va)`
 - removes a page that refers to a given Virtual address from a page directory.*
- `int x86_page_insert (pde_t *pgdir, struct Page *page, void *va, uint32_t perm)`
 - inserts a page that refers to a given va, into a given page directory.*
- `void map_segment_page (pde_t *pgdir, vaddr_t linear, size_t size, paddr_t physical, int perm)`
 - maps a segment to a virtual address on a specific page directory.*
- `void x86_test_pgdir (void)`

4.2.1 Function Documentation

4.2.1.1 `void* allocate (uint32_t n, uint32_t align)`

a boot time allocation function, allocate heaps.

kernel code has an 'end' symbol inserted into its code, it's externed and allocation starts from the end of kernel code.

Definition at line 150 of file init_mem.c.

4.2.1.2 void init_tss (void)

sets up the Task state segment and task register.

sets the TSS according to the base environment to carry out successful interrupt handling from user space.

Definition at line 177 of file init_mem.c.

4.2.1.3 void map_segment_page (pde_t * pgdir, vaddr_t linear, size_t size, paddr_t physical, int perm)

maps a segment to a virtual address on a specific page directory.

Parameters

<i>pde_t*</i>	page directory to use.
<i>vaddr_t</i>	virtual address to use.
<i>size_t</i>	segment size.
<i>paddr_t</i>	physical address of segment start.
<i>int</i>	page permissions.

Definition at line 303 of file page.c.

4.2.1.4 void memory_printinfo (void)

Reads memory size and determine the pages count needed to map it.

Definition at line 107 of file init_mem.c.

4.2.1.5 void scan_memory (void)

Definition at line 121 of file init_mem.c.

4.2.1.6 int x86_page_alloc (struct Page ** page_byref)

Allocate a page page is removed from the free pages list and initiated.

Parameters

<i>struct</i>	Page** reference to page pointer.
---------------	-----------------------------------

Returns

0 if success, -1 if fails

Definition at line 99 of file page.c.

4.2.1.7 void x86_page_detach (struct Page * page)

when [un]refering a page, ref member is decremented.

Parameters

<i>struct</i>	Page* page to detach from calling execusion stream.
---------------	---

Definition at line 135 of file page.c.

4.2.1.8 void x86_page_free (struct Page * page)

given page is freed.

Parameters

<i>struct</i>	Page*, the page to free
---------------	-------------------------

A page is freed by inserting it into the free pages list.

Definition at line 122 of file page.c.

4.2.1.9 void x86_page_init (struct Page * page)

Initiate a page entry.

Parameters

<i>struct</i>	Page*, reference to page that to be initialized.
---------------	--

Page memory is set to Zeros.

Definition at line 86 of file page.c.

4.2.1.10 int x86_page_insert (pde_t * pgdir, struct Page * page, void * va, uint32_t perm)

inserts a page that refers to a given va, into a given page directory.

Parameters

<i>pde_t*</i>	page directory to insert page into.
<i>struct</i>	Page*, the page to insert.
<i>void*</i>	Virtual address to use.
<i>uint32_t</i>	permissions on a page.

Returns

0 if success, -1 if fail.

a previously set page is inserted into a page directory and refer to a given virtual address no matter what physical address is, and permission flags are set on a page.

Definition at line 248 of file page.c.

**4.2.1.11 struct Page* x86_page_lookup (pde_t * pgdir, void * va, pte_t ** pte)
[read]**

looks up a page refers to vaddr.

Parameters

<i>pde_t</i>	page directory to search.
<i>void*</i>	virtual address of desired page.
<i>pte_t**</i>	page table reference to set.

Returns

the found page.

the functions looks up for a page into a page directory and return back to values in two different manner.

it returns the desired page address if found. and the page table address is set to a given paramter.

Definition at line 197 of file page.c.

4.2.1.12 void x86_page_remove (pde_t * pgdir, void * va)

removes a page that refers to a given Virtual address from a page directory.

Parameters

<i>pde_t*</i>	page directory to remove page from.
<i>void*</i>	virtual address the page refers to.

a virtual address is looked up in a page directory and detached. then the TLB (Translate Lookaside Buffer) is updated to avoid misbehaviour or page faults.

Definition at line 221 of file page.c.

4.2.1.13 void x86_paging_init (void)

Initiate global pages.

1-initialize the global free pages, since they all free at the start. whatever that means.

- 2- Loop around the page list and map it to the free pages.
 3- Remove I/O Hub Mapping. we need to remove the io hole from pages
 Definition at line 44 of file page.c.

4.2.1.14 pte_t* x86_pgdir_find(pde_t * pgdir, const void * va, int allocate)

find a page, if doesn't exist allocate it

Parameters

<i>pde_t</i>	Page directory to search into.
<i>const</i>	void* virtual address to allocate pages for.
<i>int</i>	allocation flag, if set allocate a page.

Returns

pte_t* page table containing the created or existing page.

the function searches for the page table containing the page that refers to a given virtual address, if it doesn't exist and allocate flag is unset, NULL is returned. If it doesn't exist and allocate flag is set, page is created and its parent page table is returned.

Definition at line 160 of file page.c.

4.2.1.15 static uint32_t x86_read_mem_size(int x) [static]

Reading memory size using the CMOS RAM/RTC device.

Parameters

<i>int</i>	the memory range to read, either base memory or extended.
------------	---

Returns

memory size.

Definition at line 99 of file init_mem.c.

4.2.1.16 void x86_setup_memory(void)

main memory initialization function.

this function carries out paging initialization and Interrupt handling initialization. and maps kernel important segments into page directory before refetching execusion after activating CR0.PG. At this point paging is on

Definition at line 200 of file init_mem.c.

4.2.1.17 void x86_test_pgd (void)

test the global page directory

Definition at line 322 of file page.c.

4.2.2 Variable Documentation

4.2.2.1 uint32_t alloc_lock = 0 [static]

Definition at line 39 of file init_mem.c.

4.2.2.2 struct Segdesc catgdt[]

Initial value:

```
{
    SEG_NULL,
    [1] = SEGMENT(0xffffffff, 0, SEGACS_RW | SEGACS_X),
    [2] = SEGMENT(0xffffffff, 0, SEGACS_RW),
    [3] = SEGMENT(0xffffffff, 0x000000, SEGACS_RW | SEGACS_USR | SEGACS_X),
    [4] = SEGMENT(0xffffffff, 0x0, SEGACS_RW | SEGACS_USR),
    [5] = SEG_NULL
}
```

New global descriptor table.

Since paging is on, Discard segment registers, and refer to full memory, with different DPL values. Segments are.

1- Kernel code segment.

2- Kernel Data segment.

3- User code segment.

4- User Data segment.

5- TSS empty segment.

Definition at line 58 of file init_mem.c.

4.2.2.3 uint32_t ext_base [static]

Definition at line 37 of file init_mem.c.

4.2.2.4 struct PageList free_pages [static]

Definition at line 31 of file page.c.

4.2.2.5 struct Gdtdesc gdtdesc

Initial value:

```
{  
    sizeof(catgdt)-1,  
    (unsigned long) catgdt  
}
```

Definition at line 76 of file init_mem.c.

4.2.2.6 uint32_t global_cr3

Definition at line 26 of file page.c.

4.2.2.7 pde_t* global_pkdir

Definition at line 25 of file page.c.

4.2.2.8 Idtdesc idtdesc

Initial value:

```
{  
    (256*sizeof(gatedesc))-1,  
    (unsigned long) idt  
}
```

Interrupt descriptor table descriptor.

Definition at line 85 of file init_mem.c.

4.2.2.9 uint32_t max_addr

Definition at line 36 of file init_mem.c.

4.2.2.10 uint32_t mem_base [static]

Definition at line 37 of file init_mem.c.

4.2.2.11 char* next_free

Definition at line 38 of file init_mem.c.

4.2.2.12 char* next_free = 0

Definition at line 38 of file init_mem.c.

4.2.2.13 uint32_t page_count

This Macro returns paddr of page containing the given virtual address

```
#define VA2PA(pgdir, va)\ ({ \ pte_t *pte; \ pgdir = &pgdir[PGDIRX(va)]; \ if( !(*pgdir & PAGE_PRESENT)) \ ( (paddr_t)~0x0); \ pte = ( (pte_t *) PA2KA(pgdir->address)) \ if(!(*pte[PGTBLX(va)] & PAGE_PRESENT)) \ ( (paddr_t)~0x0); \ else \ pte[PGTBL- X(va)]->address; \ })
```

Definition at line 24 of file page.c.

4.2.2.14 struct Page* pages

Definition at line 29 of file page.c.

4.2.2.15 struct Page* pages

Definition at line 29 of file page.c.

4.2.2.16 proc_t* proc_table

Definition at line 27 of file proc.c.

4.2.2.17 tss_t tss

Definition at line 91 of file init_mem.c.

4.3 Multiprocessors

Files

- file [apic.c](#)
supporting xAPIC for MPs

APIC

- `uint32_t vector_addr = 0x00022000`
- `uint32_t * lapic = (uint32_t *)(0xfee00000)`
- void [lapic_init](#) (void)
Soft Initiates the LAPIC.
- void [soft_lapic_enable](#) (void)
Soft To enable the Local APIC to receive interrupts you also have to set bit 8 in the Spurious Interrupt Vector Register.
- void [msr_lapic_enable](#) (void)
APIC setting IA32_APIC_BASE Model Specific Register (MSR)
- void [apic_s_ipi](#) (icr_t icr, `uint8_t lapicid`)
Sending Startup IPI to processor speicified with lapicid.
- void [apic_init_ipi](#) (`uint8_t lapicid`)
Sending INIT IPI to processor speicified with lapicid.
- void [apic_read_errors](#) (void)
Reads error field in lapic.

4.3.1 Function Documentation

4.3.1.1 void [apic_init_ipi](#) (`uint8_t lapicid`)

Sending INIT IPI to processor speicified with lapicid.

Parameters

<code>lapicid</code>	the processors lapic id
----------------------	-------------------------

Returns

none

Definition at line 160 of file apic.c.

4.3.1.2 void [apic_read_errors](#) (`void`)

Reads error field in lapic.

Parameters

<i>none</i>

Returns

none

Definition at line 188 of file apic.c.

4.3.1.3 void apic_s_ipi (icr_t *icr*, uint8_t *lapicid*)

Sending Startup IPI to processor speicified with lapicid.

Parameters

<i>icr</i>	Interrupt control register structure to set the options for the interrupt
<i>lapicid</i>	the processors lapic id

Returns

none

Definition at line 117 of file apic.c.

4.3.1.4 void lapic_init (void)

Soft Initiates the LAPIC.

Parameters

<i>none</i>

Returns

none

Definition at line 29 of file apic.c.

4.3.1.5 void msr_lapic_enable (void)

APIC setting IA32_APIC_BASE Model Specific Register (MSR)

Parameters

<i>none</i>

Returns

none

Definition at line 75 of file apic.c.

4.3.1.6 void soft_lapic_enable(void)

Soft To enable the Local APIC to receive interrupts you also have to set bit 8 in the Spurious Interrupt Vector Register.

Enabling LAPIC

Parameters

<i>none</i>

Returns

none

Definition at line 45 of file apic.c.

4.3.2 Variable Documentation**4.3.2.1 uint32_t* lapic = (uint32_t *)0xfee00000**

Definition at line 21 of file apic.c.

4.3.2.2 uint32_t vector_addr = 0x00022000

Definition at line 20 of file apic.c.

4.4 MultiProcessors

Files

- file `delay.c`
Simple delay functions for synchronization.
- file `smp.c`
SMP Support functions.

Delay

- void `asm_delay_loop (uint32_t t)`
useless loop to use while delaying
- void `asm_fake_loop (uint32_t t)`
useless loop to calibrate cpu
- void `delay (uint32_t microsec)`
uninterruptable delay function

Delay

- const char `trampoline []`
- const char `trampoline_end []`
- `fpstruct_t * fs`
floating point structure
- `ct_hdr * ct`
configuration table.
- static `ct_proc_entry * processor_entries`
- `ct_bus_entry * bus_entries`
- `ct_io_apic_entry * io_apic_entries`
- `ct_io_intr_entry * io_intr_entries`
- `ct_loc_intr_entry * loc_intr_entries`
- `uint32_t processors_count`
- `uint32_t io_apic_cnt`
- `uint32_t bus_entry_cnt`
- `uint32_t io_apic_entry_cnt`
- `uint32_t io_intr_entry_cnt`
- `uint32_t loc_intr_entry_cnt`
- `uint8_t fps_check (uint8_t *base)`
checks if floating point structure is valid , the sum of all elements must be 0
- `uint8_t ct_check (void)`
checks if configuration is valid , the sum of all elements must be 0
- void `fsp_print (fpstruct_t *fs)`
parses the floating point structure
- void `ct_read_hdr (void)`

- `parses the configuration table structure`
- void `print_proc_entry` (`ct_proc_entry *processor_entry`)
 - parses the cpu entry*
- void `ct_entries` (`void`)
 - parses the config table and sets the entries pointers*
- void `find_set_fps` (`void`)
 - Searches and Sets the global Floating pointer structure. 1. search first 1K of EBDA
2. if EBDA is undefined, search last 1K of base memory 3. search 64K starting at 0xf0000.*
- `uint8_t cpu_is_bootstrap` (`ct_proc_entry *processor_entry`)
 - checks if cpu is bootstrap*
- `uint8_t cpu_is_enabled` (`ct_proc_entry *processor_entry`)
 - checks if cpu is enabled*
- void `map_AP_Startup` (`void`)
 - Initializes application processors.*
- void `ap_init` (`void`)

4.4.1 Function Documentation

4.4.1.1 void `ap_init` (`void`)

Definition at line 291 of file smp.c.

4.4.1.2 void `asm_delay_loop` (`uint32_t t`)

useless loop to use while delaying

Parameters

<code>t</code>	number of loops
----------------	-----------------

Returns

none

Definition at line 19 of file delay.c.

4.4.1.3 void `asm_fake_loop` (`uint32_t t`)

useless loop to calibrate cpu

Parameters

<code>number</code>	of loops
---------------------	----------

Returns

none

Definition at line 32 of file delay.c.

4.4.1.4 uint8_t cpu_is_bootstrap (ct_proc_entry * processor_entry)

checks if cpu is bootstrap

Parameters

<i>processor</i> - <i>_entry</i>	pointer to processor structure in config table
-------------------------------------	--

Returns

1 if bootstrap 0 if not

Definition at line 261 of file smp.c.

4.4.1.5 uint8_t cpu_is_enabled (ct_proc_entry * processor_entry)

checks if cpu is enabled

Parameters

<i>processor</i> - <i>_entry</i>	pointer to processor structure in config table
-------------------------------------	--

Returns

1 if enabled 0 if not

Definition at line 273 of file smp.c.

4.4.1.6 uint8_t ct_check (void)

checks if configuration is valid , the sum of all elements must be 0

Parameters

<i>base</i>	base of table
-------------	---------------

Returns

0 if true 1 if false

Definition at line 67 of file smp.c.

4.4.1.7 void ct_entries (void)

parses the config table and sets the entries pointers

Parameters

<i>none</i>

Returns

none

Definition at line 147 of file smp.c.

4.4.1.8 void ct_read_hdr (void)

parses the configuration table structure

Parameters

<i>none</i>

Returns

none

Definition at line 110 of file smp.c.

4.4.1.9 void delay (uint32_t microsec)

uninterruptable delay function

Parameters

<i>microsec-</i> <i>onds</i>	to wait
---------------------------------	---------

Returns

none

Definition at line 47 of file delay.c.

4.4.1.10 void find_set_fps(void)

Searches and Sets the global Floating pointer structure. 1. search first 1K of EBDA 2. if EBDA is undefined, search last 1K of base memory 3. search 64K starting at 0xf0000.

Definition at line 231 of file smp.c.

4.4.1.11 uint8_t fps_check(uint8_t *base)

checks if floating point structure is valid , the sum of all elements must be 0

Parameters

<i>base</i>	base of table
-------------	---------------

Returns

0 if true 1 if false

Definition at line 50 of file smp.c.

4.4.1.12 void fsp_print(fpstruct_t *fs)

parses the floating point structure

Parameters

<i>fs</i>	pointer to structure
-----------	----------------------

Returns

none

Definition at line 96 of file smp.c.

4.4.1.13 void map_AP_Startup(void)

Initializes application processors.

Parameters

<i>void</i>

Returns

void

Definition at line 286 of file smp.c.

4.4.1.14 void print_proc_entry(ct_proc_entry * processor_entry)

parses the cpu entry

Parameters

<i>processor_entry</i>	pointer to ct_proc_entry structure
------------------------	------------------------------------

Returns

none

Definition at line 132 of file smp.c.

4.4.2 Variable Documentation**4.4.2.1 ct_bus_entry* bus_entries**

Definition at line 32 of file smp.c.

4.4.2.2 uint32_t bus_entry_cnt

Definition at line 40 of file smp.c.

4.4.2.3 ct_hdr* ct

configuration table.

Definition at line 28 of file smp.c.

4.4.2.4 fpstruct_t* fs

floating point structure

Definition at line 24 of file smp.c.

4.4.2.5 uint32_t io_apic_cnt

Definition at line 39 of file smp.c.

4.4.2.6 ct_io_apic_entry* io_apic_entries

Definition at line 33 of file smp.c.

4.4.2.7 uint32_t io_apic_entry_cnt

Definition at line 41 of file smp.c.

4.4.2.8 ct_io_intr_entry* io_intr_entries

Definition at line 34 of file smp.c.

4.4.2.9 uint32_t io_intr_entry_cnt

Definition at line 42 of file smp.c.

4.4.2.10 ct_loc_intr_entry* loc_intr_entries

Definition at line 35 of file smp.c.

4.4.2.11 uint32_t loc_intr_entry_cnt

Definition at line 43 of file smp.c.

4.4.2.12 ct_proc_entry* processor_entries [static]

Definition at line 31 of file smp.c.

4.4.2.13 uint32_t processors_count

Definition at line 38 of file smp.c.

4.4.2.14 const char trampoline[]**4.4.2.15 const char trampoline_end[]**

4.5 CPU-Initialization

Files

- file [processor.c](#)
Identify and initialize cpu structures.

CPU-Identification

- `char * cpu_vendors [3]`
- `cpu_t * cpu`
- `void processor_printinfo ()`
- `void processor_identify (void)`

4.5.1 Function Documentation

4.5.1.1 `void processor_identify (void)`

Definition at line 23 of file processor.c.

4.5.1.2 `void processor_printinfo ()`

Definition at line 47 of file processor.c.

4.5.2 Variable Documentation

4.5.2.1 `cpu_t* cpu`

Definition at line 19 of file processor.c.

4.5.2.2 `char* cpu_vendors[3]`

Initial value:

```
{  
    "Undefined\0",  
    "Intel\0",  
    "AMD\0"  
}
```

Definition at line 13 of file processor.c.

4.6 Drivers

Files

- file [clock.c](#)
Clock Low level controller. sets periodic interrupts and power state.
- file [cmos.c](#)
CMOS Driver.
- file [i8254.c](#)
Intel 8253/8254 PIT(Programmable Interrupt Timer) Controller.
- file [i8259.c](#)
Intel 8259 PIC(Programmable Interrupt Controller) Low Level Driver.
- file [kbc.c](#)
Keyboard driver.
- file [video.c](#)
Color Graphics Adapter (CGA) driver.

RTC Driver.

- void [clock_enable_rtc](#) (void)

CMOS Driver.

supports status A and B Power options , Refer to boch's Ports.lst for details.

- [uint8_t cmos_set_power_stat \(uint8_t stat\)](#)
supports CMOS Status B power options.
- [uint32_t cmos_get_reg \(uint8_t value\)](#)
Gets RTC registers values.
- [uint32_t cmos_set_reg \(uint8_t index, uint8_t value\)](#)
set RTC registers values.

Intel 8254 PIT

- [uint32_t delay_loop_const](#)
- void [i8254_init](#) (void)
Initiate the PIT to interrupt every time slice.
- void [i8254_calibrate_delay_loop](#) (void)
calibrates the number of loops per 1 ms
- #define [MAGIC_NUMBER](#) 1194
- #define [LOOPS](#) 150000

i8259-PIC Driver.

- void `i8259_init` (void)

Initializing the PIC to work under Protected mode.
- void `i8259_disable` (void)

Disables the PIC IRQs.
- void `i8259_mask_irq` (`uint16_t` irq)

Disables the specified interrupt.
- void `i8259_unmask_irq` (`uint16_t` irq)

Enables the specified interrupt.
- `uint16_t i8259_read_isr` (void)

get the in service interrupt.
- `uint16_t i8259_read_irr` (void)

reads the IRR register which holds raised interrupt priority
- void `i8259_eoi` (`uint8_t` irq)

End of interrupt instruction, used for ISRs.

PS/2 Keyboard Driver

This file contains keyboard mapping and input handling functions.

- static `uint8_t shiftcode` [256]

codes to shift character values.
- static `uint8_t togglecode` [256]

button codes that toggle button values
- static `uint8_t normalmap` [256]

the normal keyboard map.
- static `uint8_t shiftmap` [256]

the keyboard map once shiftcode is pressed.
- static `uint8_t ctlmap` [256]

the keyboard map once control is pressed.
- static `uint8_t * charcode` [4]

keycode definitions
- `int kbc_data` (void)

keyboard input controller driver
- void `kbc_interrupt` (void)

calls console interrupt which is used in character read.
- #define `ESCODE` (`1<<6`)

`ESCAPE` key code
- #define `CTL` (`1<<1`)

`CONTROL` key code
- #define `SHIFT` (`1<<0`)

`SHIFT` key code
- #define `ALT` (`1<<2`)

`ALT` key code
- #define `CAPSLOCK` (`1<<3`)

`CAPSLOCK` key code
- #define `NUMLOCK` (`1<<4`)

`NUMLOCK` key code
- #define `SCROLLLOCK` (`1<<5`)

`SCROLLLOCK` key code
- #define `NUL` 0

`NUL` key code
- #define `CL(x)` (`((x)-'@')`)

`CL` key code

CGA Driver

An early initialization for the console can be useful in debugging and logging operations. here screen is initialized and get ready for I/O operations.

- static `cga_attr`
holds the colors attribute of the written character
- `uint16_t cga_get_pos (void)`
gets the cursor position on the screen.
- `void cga_set_attr (uint16_t c)`
sets global color attribute to a given value.
- `void cga_clear (void)`
clears the screen.
- `void cga_set_pos (uint16_t pos)`
sets the position to a given value on the screen map
- `void cga_init (void)`
initiates CGA.
- `void cga_putc (int c)`
writes a character to screen using cga.
- `void cga_putstr (char *c)`
uses cga_putc to write a string

4.6.1 Define Documentation

4.6.1.1 #define ALT (1<<2)

Definition at line 23 of file kbc.c.

4.6.1.2 #define CAPSLOCK (1<<3)

Definition at line 24 of file kbc.c.

4.6.1.3 #define CL(x) ((x)'@')

Definition at line 97 of file kbc.c.

4.6.1.4 #define CTL (1<<1)

Definition at line 21 of file kbc.c.

4.6.1.5 #define ESCODE (1<<6)

Definition at line 20 of file kbc.c.

4.6.1.6 #define LOOPS 150000

Definition at line 17 of file i8254.c.

4.6.1.7 #define MAGIC_NUMBER 1194

Definition at line 16 of file i8254.c.

4.6.1.8 #define NUL 0

Definition at line 49 of file kbc.c.

4.6.1.9 #define NUMLOCK (1<<4)

Definition at line 25 of file kbc.c.

4.6.1.10 #define SCROLLLOCK (1<<5)

Definition at line 26 of file kbc.c.

4.6.1.11 #define SHIFT (1<<0)

Definition at line 22 of file kbc.c.

4.6.2 Function Documentation

4.6.2.1 void cga_clear(void)

clears the screen.

Definition at line 41 of file video.c.

4.6.2.2 uint16_t cga_get_pos(void)

gets the cursor position on the screen.

The get position function gets the cursor position by reading it from the data register, by indexing the cga index register.

Definition at line 28 of file video.c.

4.6.2.3 void cga_init(void)

initiates CGA.

This function sets global variables to its initial values. sets **start** to address of the C-GA buffer. and sets current character buffer to the start value. and initializes cga color attributes to none.

Definition at line 58 of file video.c.

4.6.2.4 void cga_putc (int c)

writes a character to screen using cga.

Parameters

<i>int</i>	c the character to be written.
------------	--------------------------------

The CGA putc is relatively big function and similar somehow to the printing function, here we take only one character at a time and print it to screen and attribute is set to every printed char. then character type is being checked and every character has its own operation, for example:

Backspace: cursor position is decremented by one.

New-Line: a whole line character count is added to the cursor position

After a character is written, screen checks if it needs to be shifted up.

Definition at line 69 of file video.c.

4.6.2.5 void cga_putstr (char * c)

uses cga_putc to write a string

Definition at line 118 of file video.c.

4.6.2.6 void cga_set_attr (uint16_t c)

sets global color attribute to a given value.

Parameters

<i>uint16_t</i>	c the color attribute
-----------------	-----------------------

Definition at line 37 of file video.c.

4.6.2.7 void cga_set_pos (uint16_t pos)

sets the position to a given value on the screen map

Parameters

<i>uint32_t</i>	pos the position to be set.
-----------------	-----------------------------

Function behaviour is similar to cga_get_pos

Definition at line 50 of file video.c.

4.6.2.8 void clock_enable_rtc (void)

Definition at line 15 of file clock.c.

4.6.2.9 uint32_t cmos_get_reg (uint8_t value)

Gets RTC registers values.

Parameters

<i>uint8_t</i>	value , the register to be read.
----------------	----------------------------------

Returns

uint32_t val , the value of the register.

Definition at line 58 of file cmos.c.

4.6.2.10 cmos_set_power_stat (uint8_t stat)

supports CMOS Status B power options.

Parameters

<i>uint8_t</i>	stat , the value of the option to be set.
----------------	---

Returns

uint8_t New_Stat , the CMOS new power status.

Definition at line 26 of file cmos.c.

4.6.2.11 uint32_t cmos_set_reg (uint8_t index, uint8_t value)

set RTC registers values.

Parameters

<i>uint8_t</i>	value , the value to set register to.
<i>uint8_t</i>	index, index of the register.

Returns

`uint32_t val` , the value of the register.

Definition at line 83 of file cmos.c.

4.6.2.12 void i8254_calibrate_delay_loop (void)

calibrates the number of loops per 1 ms

Parameters

<code>none</code>

Returns

`none`

Definition at line 43 of file i8254.c.

4.6.2.13 void i8254_init (void)

Initiate the PIT to interrupt every time slice.

PIT is programmed to interrupt 20 times per second which is relatively big time interval to schedule on

Definition at line 28 of file i8254.c.

4.6.2.14 void i8259_disable (void)

Disables the PIC IRQs.

This function is mostly used before setting up APIC to usage.

Definition at line 77 of file i8259.c.

4.6.2.15 void i8259_eoi (uint8_t irq)

End of interrupt instruction, used for ISRs.

Parameters

<code>uint8_t</code>	irq, the Interrupt index that has ended.
----------------------	--

In interrupt service routines an [end of interrupt] must be issued after fulfilling the service routine. In order to avoid issuing an eoi for an interrupt raised on slave PIC by an ISR of an interrupt in service on master PIC, an eoi is issued for a single PIC at a time.

Definition at line 169 of file i8259.c.

4.6.2.16 void i8259_init(void)

Initializing the PIC to work under Protected mode.

Since PIC was designed to work under real mode, it worked under the IVT which's IRQ indexes conflict with the default Exceptions in IDT defined by Intel, so we need to offset the default IRQs with 0x20 slot or whatever it requires to avoid conflict with the Intel default Exceptions.

PIC Initialization is done by issuing four Initialization command words. ICW1 = 0x11 both ICW4 Needed and ICW1 ISSUED Flags are set. ICW2 = the IRQ Base offset. ICW3 = The PIC Slave attachment situation ICW4 = Additional info. 00010001 Flags. ICW1 ISSUED - CASCADE MODE - EDGE TRIGGERED MODE - ICW4 NEEDED

00000001 Flags. 8086/8088 MODE - NORMAL EOI - NON BUFFERED MODE - NO SPECIAL FULL NESTED MODE

Output flags to PICs

Definition at line 32 of file i8259.c.

4.6.2.17 void i8259_mask_irq(uint16_t irq)

Definition at line 91 of file i8259.c.

4.6.2.18 uint16_t i8259_read_irr(void)

reads the IRR register which holds raised interrupt priority

Returns

returns the Interrupt request register.

Definition at line 150 of file i8259.c.

4.6.2.19 uint16_t i8259_read_isr(void)

get the in service interrupt.

Returns

returns the In service interrupt mask

Definition at line 136 of file i8259.c.

4.6.2.20 void i8259_unmask_irq(uint16_t irq)

Definition at line 113 of file i8259.c.

4.6.2.21 int kbc_data(void)

keyboard input controller driver

Returns

character read from screen

The task of this function is to read the keyboard input, It makes sure that there's data in the input register, then reads the data port. and maps input data to a character depending on the code generated.

Definition at line 127 of file kbc.c.

4.6.2.22 void kbc_interrupt(void)

calls console interrupt which is used in character read.

Definition at line 161 of file kbc.c.

4.6.3 Variable Documentation**4.6.3.1 cga_attr [static]**

holds the colors attribute of the written character

Definition at line 26 of file video.c.

4.6.3.2 uint8_t* charcode[4] [static]**Initial value:**

```
{
    normalmap,
    shiftmap,
    ctlmap,
    ctlmap
}
```

Definition at line 121 of file kbc.c.

4.6.3.3 uint8_t ctlmap[256] [static]**Initial value:**

```
{
    NUL,      NUL,      NUL,      NUL,      NUL,      NUL,      NUL,      NUL,
    NUL,      NUL,      NUL,      NUL,      NUL,      NUL,      NUL,      NUL,
    CL('Q'), CL('W'), CL('E'), CL('R'), CL('T'), CL('Y'), CL('U'), CL('I'),
    CL('O'), CL('P'), NUL,      NUL,      '\r',      NUL,      CL('A'), CL('S'),
```

```

CL('D'), CL('F'), CL('G'), CL('H'), CL('J'), CL('K'), CL('L'), NUL,
NUL, NUL, NUL, CL('\\'), CL('Z'), CL('X'), CL('C'), CL('V'),
CL('B'), CL('N'), CL('M'), NUL, NUL, CL('/'), NUL, NUL,
[0x97] KEY_HOME, [0x9C] '\n',
[0xB5] CL('/'), [0xC8] KEY_UP,
[0xC9] KEY_PGUP, [0xCB] KEY_LF,
[0xCD] KEY_RT, [0xCF] KEY_END,
[0xD0] KEY_DN, [0xD1] KEY_PGDN,
[0xD2] KEY_INS, [0xD3] KEY_DEL

}

```

the keyboard map once control is pressed.

Definition at line 102 of file kbc.c.

4.6.3.4 uint32_t delay_loop_const

Definition at line 19 of file i8254.c.

4.6.3.5 uint8_t normalmap[256] [static]

Initial value:

```

{
    NUL, 0x1B, '1', '2', '3', '4', '5', '6',
    '7', '8', '9', '0', '-', '=', '\b', '\t',
    'q', 'w', 'e', 'r', 't', 'y', 'u', 'i',
    'o', 'p', '[', ']', '\n', NUL, 'a', 's',
    'd', 'f', 'g', 'h', 'j', 'k', 'l', ';',
    '\\', ' ', NUL, '\\', 'z', 'x', 'c',
    'v', 'b', 'n', 'm', '/', '!', '!', NUL, '*',
    NUL, ' ', NUL, NUL, NUL, NUL, NUL, NUL,
    NUL, NUL, NUL, NUL, NUL, '7', '8', '9',
    '-', '4', '5', '6', '+', '1', '2', '3', '0',
    '.', NUL, NUL, NUL, NUL,
    [0x97] KEY_HOME, [0x9C] '\n',
    [0xB5] '/', [0xC8] KEY_UP,
    [0xC9] KEY_PGUP, [0xCB] KEY_LF,
    [0xCD] KEY_RT, [0xCF] KEY_END,
    [0xD0] KEY_DN, [0xD1] KEY_PGDN,
    [0xD2] KEY_INS, [0xD3] KEY_DEL
}

```

the normal keyboard map.

Definition at line 54 of file kbc.c.

4.6.3.6 uint8_t shiftcode[256] [static]

Initial value:

```
{
    [0x1D] CTL,
    [0x2A] SHIFT,
    [0x36] SHIFT,
    [0x38] ALT,
    [0x9D] CTL,
    [0xB8] ALT
}
```

codes to shift character values.

Definition at line 30 of file kbc.c.

4.6.3.7 uint8_t shiftmap[256] [static]

Initial value:

```
{
    NUL, 0x1B, '!','@','#','$','%','^',
    '&','*','(',')','-','+',`\b','\t',
    'Q','W','E','R','T','Y','U','I','O',
    'P','{','}','\n',NUL,'A','S','D','F',
    'G','H','J','K','L',':','"','~',
    NUL,'|','Z','X','C','V','B','N','M',
    '<','>','?',NUL,'*',NUL,' ',NUL,NUL,
    NUL,NUL,NUL,NUL,NUL,NUL,NUL,NUL,
    NUL,NUL,'7','8','9','-',',','4','5','6',
    '+','1','2','3','0','.', NUL,NUL,NUL,NUL,
    [0x97] KEY_HOME,           [0x9C] '\n',
    [0xB5] '/',               [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,          [0xCB] KEY_LF,
    [0xCD] KEY_RT,             [0xCF] KEY_END,
    [0xD0] KEY_DN,             [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,            [0xD3] KEY_DEL
}
```

the keyboard map once shiftcode is pressed.

Definition at line 78 of file kbc.c.

4.6.3.8 uint8_t togglecode[256] [static]

Initial value:

```
{
    [0x3A] CAPSLOCK,
    [0x45] NUMLOCK,
    [0x46] SCROLLLOCK
}
```

button codes that toggle button values

Definition at line 43 of file kbc.c.

4.7 Main

Files

- file [cli.c](#)
CLI I/O facilities.
- file [printf.c](#)
a printf family like functions
- file [readline.c](#)
readline operation.
- file [string.c](#)
String and memory operations, originally from linux.
- file [work_it_out.c](#)
kernel device facilities initialization and set up start.

Defines

- #define [BUFSIZE](#) 1024

Functions

- void [play](#) (void)
this function holds initializations and setting up facilities
- void [_panic](#) (const char *file, [int](#) nline, const char *fmt,...)
- void [time_print](#) (void)
a completely useless function that prints current time.
- [uint8_t](#) [initrd](#) ([multiboot_info_t](#) *mboot_ptr, [uint32_t](#) *initrd_location_ptr, [uint32_t](#) *initrd_end_ptr)
- void [initrd_test](#) ([uint32_t](#) initrd_location)
- void [bootup](#) ([uint32_t](#) mboot_ptr)
- char * [readline](#) (const char *towrite)
reads a string from keyboard into screen.

Variables

- static const char * [error_panic](#) = NULL
- static char [buf](#) [[BUFSIZE](#)]

Console

A generic wrapping for the display I/O.

- void `console_init` (void)
initializes console.
- void `console_interrupt` (int(*intr)(void))
*reads a char from a input device interrupt. int (*intr)(void) interrupt handler.*
- void `console_clear` (void)
- int `console_getc` (void)
reads a character from an input device [keyboard].
- void `console_putc` (int c)
writes a character to screen wraps cga_putc function
- void `putchr` (int c)
writes a character to screen.
- int `getchar` (void)
recives a character from keyboard and write it to screen and buffer.

Printing

- static void `putch` (int c, int *count)
prints a character at a time.
- void `ksprintkn` (void(*func)(int, int *), int *count, uintmax_t num, int base, int width, int padc)
Prints numbers.
- int `getint` (va_list *ap, int lflag)
gets the number of the va_list on a printk.
- int `getuint` (va_list *ap, int lflag)
same as getint except for the signed nature of getint.
- int `kvprintf` (const char *format, void(*func)(int, int *), int *count, va_list ap)
A BSD like printing function.
- `vprintf` (const char *format, va_list ap)
passes arugments to kvprintf after tokenizing them
- int `printf` (const char *format,...)
An ideal printing function.
- #define `hex2ascii`(x) ("0123456789ABCDEF"[x])

Strings and Memory.

Basic copying and sizing and setting operations.

- uint32_t `strlen` (const char *str, uint32_t count)
the function loops over the string character values till '\0' is reached or reached max value.

- `void * memcpy (void *dst, const void *src, uint32_t count)`
copies bytes from a memory address to another memory address
- `void * memset (void *ptr, int c, uint32_t count)`
sets a number of bytes starting from a given pointer to a given character.
- `int strcmp (const char *s1, const char *s2)`

Mainflow.

After kernel is loaded into IP, it fetches into `_start` function which makes a soft reboot by the bios, and reloads a new global descriptor table and resets segments. then just before it jumps to the `work_it_out` function it sets stack up and jumps to earlier mentioned function.

- `void work_it_out (uint32_t mboot_ptr)`

4.7.1 Define Documentation

4.7.1.1 #define BUFSIZE 1024

Definition at line 18 of file readline.c.

4.7.1.2 #define hex2ascii(x) ("0123456789ABCDEF"[x])

Definition at line 14 of file printf.c.

4.7.2 Function Documentation

4.7.2.1 void _panic_(const char * file, int nline, const char * fmt, ...)

Definition at line 45 of file play.c.

4.7.2.2 void bootup(uint32_t mboot_ptr)

Definition at line 132 of file play.c.

4.7.2.3 void console_clear(void)

Definition at line 40 of file cli.c.

4.7.2.4 int console_getc(void)

reads a character from an input device [keyboard].

Returns

int c Character read.

It issues a keyboard interrupt which issues a console interrupt, and wait for a character to be read.

Definition at line 44 of file cli.c.

4.7.2.5 void console_init(void)

initializes console.

Initializes the display device and sets cursor positions to zero.

Definition at line 20 of file cli.c.

4.7.2.6 void console_interrupt(int(*)(void) intr)

reads a char from a input device interrupt. int (*intr)(void) interrupt handler.

This function takes the input device interrupt handler as a paramter and reads a character from it into the console buffer.

Definition at line 28 of file cli.c.

4.7.2.7 void console_putc(int c)

writes a character to screen wraps cga_putc function

int c character to print.

Definition at line 57 of file cli.c.

4.7.2.8 int getchar(void)

recives a character from keyboard and write it to screen and buffer.

Returns

int char read from keyboard.

Definition at line 65 of file cli.c.

4.7.2.9 int getint(va_list * ap, int lflag)

gets the number of the va_list on a printk.

Parameters

<i>va_list*</i>	the list of parameters passed by printk
<i>int</i>	the long flag.

Returns

the passed number.

this function is issued everytime a number flag in the format string is found to fetch a number from the va_list passed.

Definition at line 49 of file printf.c.

4.7.2.10 int getuint (va_list * ap, int lflag)

same as getint except for the signed nature of getint.

Parameters

<i>va_list*</i>	the list of parameters passed by printk
<i>int</i>	the long flag.

Returns

the passed number.

Definition at line 66 of file printf.c.

4.7.2.11 uint8_t initrd (multiboot_info_t * mboot_ptr, uint32_t * initrd_location_ptr, uint32_t * initrd_end_ptr)

Definition at line 80 of file play.c.

4.7.2.12 void initrd_test (uint32_t initrd_location)

Definition at line 98 of file play.c.

4.7.2.13 void ksprintkn (void(*)(int, int *) func, int * count, uintmax_t num, int base, int width, int padc)

Prints numbers.

Parameters

<i>void</i>	(*)(int, int*) a char printing function.
<i>int*</i>	reference of the count variable, to store new count in.
<i>uintmax_t</i>	the number to print
<i>int</i>	base of the number.
<i>int</i>	digit width to print according to.
<i>int</i>	the padding character to print to pad a number.

this function converts numbers into their character representation the function recursively calls itself to print lower digits first.

Definition at line 28 of file printf.c.

4.7.2.14 int kvprintf (const char * *format*, void(*)(int, int *) *func*, int * *count*, va_list *ap*)

A BSD like printing function.

Parameters

<i>const</i>	char* format string.
<i>void</i>	(*)(int,int*) a char printing function.
<i>int*</i>	reference of count variable, to store count in.
<i>va_list</i>	the list of paramters passed by printf->vprintf

Returns

count of chars written

Definition at line 78 of file printf.c.

4.7.2.15 void * memcpy (void * *dst*, const void * *src*, uint32_t *count*)

copies bytes from a memory address to another memory address

Parameters

<i>void*</i>	the destination pointer.
<i>const</i>	void* the source pointer.
<i>uint32_t</i>	the number of bytes to copy.

Returns

the destination pointer.

Definition at line 22 of file string.c.

4.7.2.16 memset (void * *ptr*, int *c*, uint32_t *count*)

sets a number of bytes starting from a given pointer to a given character.

Parameters

<i>void*</i>	the pointer to use.
<i>int</i>	character to set.
<i>uint32_t</i>	the number of bytes to set.

Returns

the pointer after being set.

Definition at line 34 of file string.c.

4.7.2.17 void play(void)

this function holds initializations and setting up facilities

Definition at line 175 of file play.c.

4.7.2.18 int printk(const char * format, ...)

An ideal printing function.

Parameters

<i>const</i>	char* format string, an ideal format string.
<i>stdarg</i>	argument list, variable referenced.

Returns

int number of chars written.

A printk function is simply a wrapper to the vprintf function. It reads the parameter list and pass it to vprintf.

Definition at line 202 of file printf.c.

4.7.2.19 static void putch(int c, int * count) [static]

prints a character at a time.

Parameters

<i>int</i>	character to write.
<i>int*</i>	reference of count variable to increment per print.

Definition at line 23 of file printf.c.

4.7.2.20 void putchr(int c)

writes a character to screen.

Parameters

<i>int</i>	char to be written to display buffer.
------------	---------------------------------------

Definition at line 61 of file cli.c.

4.7.2.21 `char * readline (const char * twrite)`

reads a string from keyboard into screen.

Parameters

<code>const</code>	char * character to write
--------------------	---------------------------

Returns

the string read.

It uses the put and get char functions from the cli code. then keep on receiving characters until [ENTER] is pressed.

Definition at line 22 of file readline.c.

4.7.2.22 `int strcmp (const char * s1, const char * s2)`

Definition at line 43 of file string.c.

4.7.2.23 `uint32_t strnlen (const char * str, uint32_t count)`

the function loops over the string character values till '\0' is reached or reached max value.

Parameters

<code>const</code>	char* string.
<code>uint32_t</code>	the maximum size desired.

Returns

the number of chars in a string.

Definition at line 14 of file string.c.

4.7.2.24 `void time_print (void)`

a completely useless function that prints current time.

Definition at line 63 of file play.c.

4.7.2.25 `int vprintk (const char * format, va_list ap)`

passes arguments to kvprintk after tokenizing them

Parameters

<i>const</i>	char* format string.
<i>va_list</i>	paramter list.

Returns

int number of chars written.

Definition at line 195 of file printf.c.

4.7.2.26 void work_it_out (uint32_t mboot_ptr)

Definition at line 35 of file work_it_out.c.

4.7.3 Variable Documentation**4.7.3.1 char buf[BUFLEN] [static]**

Definition at line 19 of file readline.c.

4.7.3.2 const char* error_panic = NULL [static]

Definition at line 40 of file play.c.

4.8 Category Here--

Collaboration diagram for Category Here--:



Files

- file [initrd.c](#)
Initrd.

Initrd

- void * [initrd_mem\(\)](#)

4.8.1 Function Documentation

4.8.1.1 void* [initrd_mem\(\)](#)

Definition at line 15 of file `initrd_mem.c`.

4.9 Debug

Files

- file [kconsole.h](#)
Kernel console for debugging ease.

KConsole

- [kcommand_t * kconsole_commands](#)
- [void kconsole_init \(void\)](#)
- [void kconsole \(void\)](#)
- [void kcommand_register \(kcommand_t *cmd\)](#)
- [kcommand_t * kcommand_match \(const char *buf\)](#)
- [void kscheduler_info \(void\)](#)
- [void kcpu_info \(void\)](#)
- [void kconsole_help \(void\)](#)

4.9.1 Function Documentation

4.9.1.1 `kcommand_t* kcommand_match (const char * buf)`

Definition at line 74 of file kconsole.c.

4.9.1.2 `void kcommand_register (kcommand_t * cmd)`

Definition at line 56 of file kconsole.c.

4.9.1.3 `void kconsole (void)`

Definition at line 36 of file kconsole.c.

4.9.1.4 `void kconsole_help (void)`

Definition at line 101 of file kconsole.c.

4.9.1.5 `void kconsole_init (void)`

Definition at line 16 of file kconsole.c.

4.9.1.6 `void kcpu_info (void)`

Definition at line 96 of file kconsole.c.

4.9.1.7 **void kscheduler_info(void)**

KCOMMANDS

Definition at line 91 of file kconsole.c.

4.9.2 Variable Documentation

4.9.2.1 **kcommand_t* kconsole_commands**

Definition at line 13 of file kconsole.c.

4.10 Management

Files

- file [mm.c](#)

Memory allocation/deallocation

Memory Management

Memory allocation is done by initiating an early hash table in which every page is stored as a bucket, within every bucket the size of memory allocated on that page sequentially. e.g. [Bucket:1]->[heap1:4]->[heap2:32]->[heap3:1024]->... in the first page 3 heaps are allocated 1st with size 4 bytes 2nd with size 32 byte and third with size 1Kb.

- `htable_t * mm_table`
- `uint32_t mm_va_base`
- `uint32_t cur_bucket`
- `void kmalloc_test (void)`
- `uint32_t mm_hash (uint32_t key)`
hash function for allocated memory entries in heaps hashtable
- `uint32_t mm_destroy (uint32_t destroy)`
destroy function to free unwanted entries
- `void mm_init (void)`
memory heap manager initialization function
- `static uint32_t mm_insert (uint32_t va, uint32_t size)`
- `void * kmalloc (uint32_t size)`
- `void kfree (void *va)`

4.10.1 Function Documentation

4.10.1.1 void `kfree (void * va)`

Parameters

<code>void</code>	<code>*va removes entry corresponding to heap.</code>
-------------------	---

Returns

nothing.

Definition at line 153 of file mm.c.

4.10.1.2 `void* kmalloc (uint32_t size)`

Parameters

<i>uint32_t</i>	size, the size of desired heap.
-----------------	---------------------------------

Returns

fails Null, success returns heap base address.

Loops per each page and checks if it is already full. If full use next page for next bucket.
else return base address.

Definition at line 132 of file mm.c.

4.10.1.3 void kmalloc_test (void)**Parameters**

<i>nothing</i>

Returns

nothing

tests abilities to remove and allocate heaps in healthy manner.

Definition at line 166 of file mm.c.

4.10.1.4 uint32_t mm_destroy (uint32_t destroy)

destroy function to free unwanted entries

Parameters

<i>uint32_t</i>	destroy is the entry or key to be destroyed
-----------------	---

Returns

uint32_t returns 0 (indicating given entry is free)

a free entry is identified by the value 0, to indicate that an entry is free value is simply left however a key is freed

Definition at line 59 of file mm.c.

4.10.1.5 uint32_t mm_hash (uint32_t key)

hash function for allocated memory entries in heaps hashtable

Parameters

<i>uint32_t</i>	key is the virtual address of table entry
-----------------	---

Returns

uint32_t the returned hashed index of given key

since heaps are ordered and traced by a hashtable a hashing function to determine index of a given address into table is mandatory. such a function returns the index of page into page tables structures since in a chained hashtable heaps that belong to same page exist in the same linkedlist

Definition at line 46 of file mm.c.

4.10.1.6 void mm_init(void)

memory heap manager intialization function

Parameters

<i>nothing</i>

Returns

nothing

For a heap manager to be initialized a boot time heap is allocated to hold heaps entries. Early allocated heap is mapped to kernel address space and virtual address base of heaps is set. Tests are exectued eventually.

Definition at line 76 of file mm.c.

4.10.1.7 static uint32_t mm_insert(uint32_t va, uint32_t size) [static]**Parameters**

<i>uint32_t</i>	va, the desired heap virtual base address
<i>uint32_t</i>	size, the heap size to be allocated

Returns

fail -1, else return inserted base virtual address

on first use of a bucket a page is allocated then iteration is done until free address is reached then insertion to hash table is committed. On success it returns base address.

Definition at line 98 of file mm.c.

4.10.2 Variable Documentation

4.10.2.1 `uint32_t cur_bucket`

Definition at line 28 of file mm.c.

4.10.2.2 `htable_t* mm_table`

Here's the hash entry values 1- key = page number 2- val = size

Definition at line 26 of file mm.c.

4.10.2.3 `uint32_t mm_va_base`

Definition at line 27 of file mm.c.

4.11 Optimization

Files

- file [mergesort.c](#)
Merge Sort implementation.
- file [mergesort.c](#)
Merge Sort implementation.

Sorting(Merge)

- void [mergesort \(uint32_t array\[\], uint32_t temp\[\], uint32_t left, uint32_t right\)](#)
- void [merge \(uint32_t array\[\], uint32_t temp\[\], uint32_t left, uint32_t center, uint32_t right\)](#)

Sorting(Merge)

- static void [swap \(int array\[\], int i, int j\)](#)
- static int [partition \(int array\[\], int low, int high\)](#)
- static void [quicksort \(int array\[\], int low, int high\)](#)
- void [quicksort \(int array\[\], int size\)](#)
- int [main \(void\)](#)

4.11.1 Function Documentation

4.11.1.1 int main (void)

Definition at line 62 of file quicksort.c.

4.11.1.2 void merge (uint32_t array[], uint32_t temp[], uint32_t left, uint32_t center, uint32_t right)

Definition at line 23 of file mergesort.c.

4.11.1.3 void mergesort (uint32_t array[], uint32_t temp[], uint32_t left, uint32_t right)

Definition at line 12 of file mergesort.c.

4.11.1.4 static int partition (int array[], int low, int high) [static]

Definition at line 23 of file quicksort.c.

4.11.1.5 `static void quick_sort(int array[], int low, int high) [static]`

Definition at line 47 of file quicksort.c.

4.11.1.6 `void quicksort(int array[], int size)`

Definition at line 55 of file quicksort.c.

4.11.1.7 `static void swap(int array[], int i, int j) [static]`

Definition at line 15 of file quicksort.c.

4.12 Process-Management

Files

- file `load_elf.c`
Elf binary loader.
- file `proc.c`
Process Manager.

Binary Loader.

- `uint32_t elf_load_to_proc (proc_t *proc, uint32_t offset)`
loads a binary into a proc from an offset into image.

Process Manager.

- `pde_t * global_pgdir`
- struct Proc_List `empty_procs`
- `proc_t * proc_table`
- struct Proc_Lifo `running_procs`
- `FIFO_HEAD (ready_procs, proc, 256)`
- void `proc_printinfo (void)`
Initialize the proc array.
- void `init_proc_table (void)`
- `uint32_t create_proc (proc_t **proc_s)`
Initialize and create a proc information block.
- void `proc_ready (proc_t *proc)`
- `uint32_t proc_alloc_mem (proc_t *proc, void *va, uint32_t len)`
allocates memory thunk for a proc.
- `uint32_t init_proc0 ()`
loads and initiated the proc0(first proc) from image.
- void `init_proc (void)`
initiates the process table and tests queues and initiates proc0
- void `test_lifo (void)`
Tests the LIFO queues data structure functionality.
- void `test_fifo (void)`
Tests the FIFO queue data structure.
- void `switch_address_space (proc_t *proc_to_run)`
switches between the kernel and a given proc
- void `sched_init (void)`
- void `schedule (void)`
main scheduling function

4.12.1 Detailed Description

odefine FIFO_EXTERN(nme, member)\ extern struct fifo name;

4.12.2 Function Documentation

4.12.2.1 uint32_t create_proc (proc_t ** proc_s)

Initialize and create a proc information block.

Parameters

<i>proc_t</i>	proc pointer reference to store created proc in.
---------------	--

Returns

0 if success

a proc address space is initialized and segment descriptors are set, and the Stack is also set.

Definition at line 103 of file proc.c.

4.12.2.2 uint32_t elf_load_to_proc (proc_t * proc, uint32_t offset)

loads a binary into a proc from an offset into image.

Parameters

<i>proc_t*</i>	proc to load binary into
<i>uint32_t</i>	offset of the binary into image

Returns

0 if success

The binary loader loads a binary image into the 0xA000000 VA where all procs elf are loaded.

Program headers are loaded into proc page directory and eip is set.

Definition at line 28 of file load_elf.c.

4.12.2.3 FIFO_HEAD (ready_procs , proc , 256)

4.12.2.4 void init_proc (void)

initiates the process table and tests queues and initiates proc0

Definition at line 206 of file proc.c.

4.12.2.5 uint32_t init_proc0()

loads and initiated the proc0(first proc) from image.

Definition at line 176 of file proc.c.

4.12.2.6 void init_proc_table(void)

Parameters

<i>nothing</i>

Returns

nothing

Commits process holding structures (LIST/LIFO/FIFO) initializes PEBs and address spaces.

Definition at line 57 of file proc.c.

4.12.2.7 uint32_t proc_alloc_mem(proc_t *proc, void *va, uint32_t len)

allocates memory thunk for a proc.

Parameters

<i>proc_t</i>	proc reference to allocate memory for
<i>void*</i>	the base address of memory needed to be allocated.
<i>uint32_t</i>	the size of segment.

Returns

0 if success

Definition at line 154 of file proc.c.

4.12.2.8 void proc_printinfo(void)

Initialize the proc array.

Initializes all the procs data structures, Proc table, empty proc list and the scheduling process queues. All proc entries are set to 0's. and empty proc list is fill. and proc binary loader is mapped.

Definition at line 39 of file proc.c.

4.12.2.9 void proc_ready(proc_t *proc)

Parameters

<i>proc_t*</i>	proc, process to use
----------------	----------------------

Returns

nothing

this small process is used by scheduler to set a process as ready and push it into ready procs to be scheduled into ready_procs FIFO

Definition at line 140 of file proc.c.

4.12.2.10 void sched_init(void)

Definition at line 317 of file proc.c.

4.12.2.11 void schedule(void)

main scheduling function

Parameters

<i>nothing</i>

Returns

nothing

a very basic RR scheduling function which detach running procs and push them into ready procs then fetch them. UNREACHABLE

Definition at line 333 of file proc.c.

4.12.2.12 void switch_address_space(proc_t * proc_to_run)

switches between the kernel and a given proc

switching to a proc is made from a calling proc which is the kernel to a given proc address space. switching is done by fooling the x86 into believing that it is coming back from an interrupt by setting the hardware interrupt stack frame and issuing an iret. switching can also be made by SYSENTER/SYSEXIT.

Definition at line 269 of file proc.c.

4.12.2.13 void test_fifo(void)

Tests the FIFO queue data structure.

Definition at line 241 of file proc.c.

4.12.2.14 void test_lifo (void)

Tests the LIFO queues data structure functionality.

Definition at line 218 of file proc.c.

4.12.3 Variable Documentation**4.12.3.1 struct Proc_List empty_procs**

Definition at line 25 of file proc.c.

4.12.3.2 pde_t* global_pgdir

Definition at line 25 of file page.c.

4.12.3.3 proc_t* proc_table

Definition at line 27 of file proc.c.

4.12.3.4 struct Proc_Lifo running_procs

Definition at line 28 of file proc.c.

4.13 Data-Structures

Files

- file [htable.c](#)

Chained hash table.

Functions

- `uint32_t early_htable_init (htable_t *table, uint32_t buckets_count, uint32_t bucket_length, uint32_t(*hash)(uint32_t), uint32_t(*destroy)(uint32_t))`
Initializes new hash table.
- `void htable_init (htable_t *table, uint32_t buckets_count, uint32_t(*hash)(uint32_t), uint32_t(*destroy)(uint32_t))`
Initializes new hash table.
- `void htable_destroy (htable_t *table)`
Deletes hash table
- `uint32_t htable_insert (htable_t *table, uint32_t key, uint32_t value)`
Inserts value into the hash table.
- `uint32_t htable_remove (htable_t *table, uint32_t key)`
Deletes value from the hash table.
- `uint32_t htable_get (htable_t *table, uint32_t key)`
hash table look up function with O(n)

4.13.1 Function Documentation

4.13.1.1 `uint32_t early_htable_init (htable_t * table, uint32_t buckets_count, uint32_t bucket_length, uint32_t(*)(uint32_t) hash, uint32_t(*)(uint32_t) destroy)`

Definition at line 23 of file htable.c.

4.13.1.2 `htable_destroy (htable_t * table)`

Deletes hash table

Parameters

<code>htable_t*</code>	the reference to hash table
------------------------	-----------------------------

Destroys hash tables buckets using the internal destroy function if not it just frees the memory the HTable is using Note: Early hash table SHOULD NOT be destroyed.

Definition at line 94 of file htable.c.

4.13.1.3 `htable_get (htable_t * table, uint32_t key)`

hash table look up function with O(n)

Parameters

<i>htable_t*</i>	reference to Hash table
<i>uint32_t</i>	key to return its value or -1 if not found

Definition at line 169 of file htable.c.

4.13.1.4 `void htable_init (htable_t * table, uint32_t buckets_count,
 uint32_t(*)(uint32_t) hash, uint32_t(*)(uint32_t) destroy)`

Initializes new hash table.

Parameters

<i>htable_t*</i>	a non NULL hash-table pointer
<i>uint32_t</i>	the number of buckets to be initialized
<i>uint32_t(*)(uint32_t)</i>	hashing function
<i>uint32_t(*)(uint32_t)</i>	table destroy function

FIXME

allocate space for table inside the htable once kmalloc is made.

Definition at line 64 of file htable.c.

4.13.1.5 `uint32_t htable_insert (htable_t * table, uint32_t key, uint32_t value)`

Definition at line 120 of file htable.c.

4.13.1.6 `htable_remove (htable_t * table, uint32_t key)`

Inserts value into the hash table.

Parameters

<i>htable_t*</i>	reference to hash table
<i>uint32_t</i>	the value to be stored.

Definition at line 146 of file htable.c.

4.14 Synchronization

Data Structures

- struct `semaphore_t`
- struct `waiting_proc`

Files

- file `mutex.c`
- file `semaphore.c`
- file `wait_queue.c`
- file `semaphore.h`

Semaphore structures and function headers.

Typedefs

- typedef struct `waiting_proc` `waiting_t`

Functions

- `LIST_HEAD` (`Wait_list`, `waiting_proc`)

Mutex

Mutex synchronization primitives

- void `mutex_init (semaphore_t *s)`
initialization to mutex structs and values.
- void `mutex_lock (semaphore_t *s, proc_t *proc)`
blocks a process over a mutex
- void `mutex_unlock (semaphore_t *s)`
sets binary resource to free

Semaphores

Semaphore synchronization primitives

- void `semaphore_init (semaphore_t *s, uint32_t num)`
Initialization of a semaphore primitive function.
- void `semaphore_down (semaphore_t *s, proc_t *proc)`
does down operation over semaphore primitive
- void `semaphore_up (semaphore_t *s)`
Up function for a semaphore.

Wait queues

Wait queues

- void `wait_init` (struct Proc_List *waiting_procs)
Waiting procs linked list initialization function.
- void `wait_sleep` (struct Proc_List *list, `proc_t` *proc, `uint32_t` ticks)
function that blocks a process
- void `wait_update` (void)
blocked process updating function
- void `wait_wakeup` (struct Proc_List *list)
wakes up a process once timeout or resource free

4.14.1 Typedef Documentation

4.14.1.1 `typedef struct waiting_proc waiting_t`

4.14.2 Function Documentation

4.14.2.1 `LIST_HEAD(Wait_list , waiting_proc)`

4.14.2.2 `void mutex_init(semaphore_t * s)`

initialization to mutex structs and values.

Parameters

<code>s</code>	the base semahpore of mutex to initalize.
----------------	---

Returns

nothing

A mutex is a binary semaphore thus, counting variable is set to one and a list of waiting processes is initialized.

Definition at line 20 of file mutex.c.

4.14.2.3 `void mutex_lock(semaphore_t * s, proc_t * proc)`

blocks a process over a mutex

Parameters

<code>s</code>	the semaphore that belongs to mutex
<code>proc</code>	the process to block

Returns

nothing

a binary clone to the semaphore down function.

Definition at line 34 of file mutex.c.

4.14.2.4 void mutex_unlock (semaphore_t * s)

sets binary resource to free

Parameters

<i>s</i>	mutex base semaphore
----------	----------------------

Returns

nothing

a clone of semaphore up function

Definition at line 49 of file mutex.c.

4.14.2.5 void semaphore_down (semaphore_t * s, proc_t * proc)

does down operation over semaphore primitive

Parameters

<i>s</i>	the semaphore to do down for.
<i>proc</i>	the process to block

quire semaphore If a resource is available aquire it else wait with max time till waken up on resource release

Definition at line 35 of file semaphore.c.

4.14.2.6 void semaphore_init (semaphore_t * s, uint32_t num)

Initialization of a semaphore primitive function.

Parameters

<i>s</i>	the semaphore to initialize
<i>num</i>	the number of resources (counting/binary)

Returns

nothing

Initialization is simply done by setting semaphore count to given number and initialize the list of waiting procs

Definition at line 22 of file semaphore.c.

4.14.2.7 void semaphore_up (semaphore_t * s)

Up function for a semaphore.

Parameters

<i>s</i>	the semaphore to free resource for
----------	------------------------------------

Returns

nothing

Once a resource is free the semaphore counting variable is incremented if there's no waiting processes.. else a waiting process is brought out to take its place. release semaphore

Definition at line 58 of file semaphore.c.

4.14.2.8 void wait_init (struct Proc_List * waiting_procs)

Waiting procs linked list initialization function.

Parameters

<i>waiting_procs</i> , <i>list</i>	to hold waiting procs pointer
---------------------------------------	-------------------------------

Returns

nothing

Definition at line 20 of file wait_queue.c.

4.14.2.9 void wait_sleep (struct Proc_List * list, proc_t * proc, uint32_t ticks)

function that blocks a process

Parameters

<i>list</i> , <i>the</i>	sleeping processes list
<i>proc</i> , <i>the</i>	process to put to sleep
<i>ticks</i> , <i>number</i>	of ticks before wake-up

Returns

void

Generally, a sleep is done on an I/O or race conditions. Here, a sleep is done either timed out or forever..relatively. and proc status is changed into blocked, inserted into waiting list and scheduler is set to reschedule procs.

Definition at line 38 of file wait_queue.c.

4.14.2.10 void wait_update(void)

blocked process updating function

Parameters

<i>nothing</i>

Returns

nothing

For each timer tick/interrupt a wait_update is made to wake up timed out procs and update other waiting procs timers. __NOT_USED__

Definition at line 66 of file wait_queue.c.

4.14.2.11 void wait_wakeup(struct Proc_List * list)

wakes up a process once timeout or resource free

Parameters

<i>list, the</i>	blocked processes list
------------------	------------------------

Returns

nothing

Basically, This function serves procs blocked due to a lock or semaphore..etc the first proc is marked as ready and pushed into ready procs

Definition at line 93 of file wait_queue.c.

4.15 Time

Files

- file [periodic.c](#)
Periodic interrupt handler.

Periodic_Timer

- void [time_handler \(void\)](#)

4.15.1 Function Documentation

4.15.1.1 void [time_handler \(void \)](#)

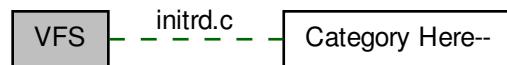
TODO

- Wake up sleeping procs if timed out
- update timers
- Execute scheduler

Definition at line 13 of file [periodic.c](#).

4.16 VFS

Collaboration diagram for VFS:



Data Structures

- struct `initrd_header_t`
- struct `initrd_file_header_t`
- struct `direntry_t`
- struct `vfs_node`

Files

- file `initrd.c`
Initrd.
- file `initrd.h`
Initrd file system headers.
- file `vfs.c`
Virtual file system standards , In order to abstract different file system calls.
- file `vfs.h`
Standardized file system structures and constants.

Initrd

- static `initrd_header_t * initrd_header = NULL`
- static `initrd_file_header_t * file_headers = NULL`
- static `vfs_node_t * initrd_root = NULL`
- static `vfs_node_t * initrd_dev = NULL`
- static `vfs_node_t * root_nodes = NULL`
- `int nroot_nodes`
- `direntry_t dirent`
- `uint32_t initrd_read (vfs_node_t *node, uint32_t offset, uint32_t size, uint8_t *buffer)`

Read operation implementation , reads the offset from the node structure , Reads data of size specified by the call ; after checking that the file is legal.

- static `direntry_t * initrd_readdir (vfs_node_t *node, uint32_t index)`

Directory traversal , reads next directory ; still immature.
- static `vfs_node_t * initrd_faddir (vfs_node_t *node, char *name)`

Directory traversal , searches for a direcotry , very Immature.
- `vfs_node_t * initialise_initrd (uint32_t location)`

Initializes file systems structures (root directory , /dev direcotry and file nodes) to be able to preform files system operations.

Initrd

- `vfs_node_t * fs_root = 0`
- `uint32_t read_fs (vfs_node_t *node, uint32_t offset, uint32_t size, uint8_t *buffer)`

Calls the node's read operation , saved as read in the node structure.
- `uint32_t write_fs (vfs_node_t *node, uint32_t offset, uint32_t size, uint8_t *buffer)`

Calls the node's write operation , saved as write in the node structure.
- `uint8_t open_fs (vfs_node_t *node, uint8_t read, uint8_t write)`

Calls the node's open operation , saved as write in the node structure.
- `uint8_t close_fs (vfs_node_t *node)`

Calls the node's open operation , saved as write in the node structure.
- `direntry_t * readdir_fs (vfs_node_t *node, uint32_t index)`

Calls the node's read directory operation , saved as readdir in the node structure.
- `vfs_node_t * faddir_fs (vfs_node_t *node, char *name)`

Calls the node's find directory operation , save as faddir in the node strucutre.

Initrd

- `typedef uint32_t(* read_type_t)(struct vfs_node *, uint32_t, uint32_t, uint8_t *)`
- `typedef uint32_t(* write_type_t)(struct vfs_node *, uint32_t, uint32_t, uint8_t *)`
- `typedef uint8_t(* open_type_t)(struct vfs_node *, uint32_t, uint32_t)`
- `typedef uint8_t(* close_type_t)(struct vfs_node *)`
- `typedef direntry_t *(* readdir_type_t)(struct vfs_node *, uint32_t)`
- `typedef struct vfs_node *(* faddir_type_t)(struct vfs_node *, char *name)`
- `typedef struct vfs_node vfs_node_t`
- `#define FS_FILE 0x01`
- `#define FS_DIRECTORY 0x02`
- `#define FS_CHARDEVICE 0x03`
- `#define FS_BLOCKDEVICE 0x04`
- `#define FS_PIPE 0x05`
- `#define FS_SYMLINK 0x06`
- `#define FS_MOUNTPOINT 0x08`

4.16.1 Define Documentation

4.16.1.1 `#define FS_BLOCKDEVICE 0x04`

Definition at line 18 of file vfs.h.

4.16.1.2 `#define FS_CHARDEVICE 0x03`

Definition at line 17 of file vfs.h.

4.16.1.3 `#define FS_DIRECTORY 0x02`

Definition at line 16 of file vfs.h.

4.16.1.4 `#define FS_FILE 0x01`

Definition at line 15 of file vfs.h.

4.16.1.5 `#define FS_MOUNTPOINT 0x08`

Definition at line 21 of file vfs.h.

4.16.1.6 `#define FS_PIPE 0x05`

Definition at line 19 of file vfs.h.

4.16.1.7 `#define FS_SYMLINK 0x06`

Definition at line 20 of file vfs.h.

4.16.2 Typedef Documentation

4.16.2.1 `typedef uint8_t(* close_type_t)(struct vfs_node *)`

Definition at line 28 of file vfs.h.

4.16.2.2 `typedef struct vfs_node*(* finddir_type_t)(struct vfs_node *, char *name)`

Definition at line 35 of file vfs.h.

4.16.2.3 `typedef uint8_t(* open_type_t)(struct vfs_node *, uint32_t, uint32_t)`

Definition at line 27 of file vfs.h.

```
4.16.2.4  typedef uint32_t(* read_type_t)(struct vfs_node *, uint32_t, uint32_t, uint8_t
*)
```

Definition at line 25 of file vfs.h.

```
4.16.2.5  typedef direntry_t*(* readdir_type_t)(struct vfs_node *, uint32_t)
```

Definition at line 34 of file vfs.h.

```
4.16.2.6  typedef struct vfs_node vfs_node_t
```

```
4.16.2.7  typedef uint32_t(* write_type_t)(struct vfs_node *, uint32_t, uint32_t, uint8_t
*)
```

Definition at line 26 of file vfs.h.

4.16.3 Function Documentation

```
4.16.3.1  uint8_t close_fs( vfs_node_t * node )
```

Calls the node's open operation , saved as write in the node structure.

Parameters

<i>vfs_node_t</i>	*node the file's node to open.
-------------------	--------------------------------

Returns

uint32_t : 0 if the node does not support open operation else The result of the close operation.

Definition at line 74 of file vfs.c.

```
4.16.3.2  vfs_node_t* finddir_fs( vfs_node_t * node, char * name )
```

Calls the node's find directory operation , save as finddir in the node strucutre.

Parameters

<i>vfs_node_t</i>	*node head to start from.
<i>uint32_t</i>	char * name name of the directory to search for.

Returns

uint32_t : pointer to the node if succeeded , 0 if failed.

Definition at line 106 of file vfs.c.

4.16.3.3 `vfs_node_t* initialise_initrd(uint32_t location)`

Initializes file systems structures (root directory , /dev direcotry and file nodes) to be able to preform files system operations.

Parameters

<code>uint32_t</code>	location : pointer to the start of the initrd image.
-----------------------	--

Returns

`vfs_node_t*` initrd filesystem root node.

Definition at line 94 of file initrd.c.

4.16.3.4 `static vfs_node_t* initrd_finddir(vfs_node_t * node, char * name) [static]`

Directory traversal , searches for a direcotry , very Immature.

Parameters

<code>vfs_node_t</code>	*node head to start from.
<code>uint32_t</code>	char * name name of the directory to search for.

Returns

`uint32_t` : pointer to directory entry if succeeded , 0 if failed.

Definition at line 78 of file initrd.c.

4.16.3.5 `uint32_t initrd_read(vfs_node_t * node, uint32_t offset, uint32_t size, uint8_t * buffer)`

Read operation implementation , reads the offset from the node structure , Reads data of size specified by the call ; after checking that the file is legal.

Parameters

<code>vfs_node_t</code>	*node the file's node to be read.
<code>uint32_t</code>	offset where to start reading (0 for beggining of the file).
<code>uint32_t</code>	size size of data to read.
<code>uint8_t</code>	*buffer buffer to be read into.

Returns

`uint32_t` : 0 if failed , size of data read if succeeded .

Definition at line 32 of file initrd.c.

**4.16.3.6 static dirent_t* initrd_readdir (vfs_node_t * node, uint32_t index)
[static]**

Directory traversal , reads next directory ; still immature.

Parameters

<code>vfs_node_t</code>	*node the directory node to be read.
<code>uint32_t</code>	directory index.

Returns

`uint32_t` : pointer to directory entry if succeeded , 0 if failed.

Definition at line 54 of file initrd.c.

4.16.3.7 uint8_t open_fs (vfs_node_t * node, uint8_t read, uint8_t write)

Calls the node's open operation , saved as write in the node structure.

Parameters

<code>vfs_node_t</code>	*node the file's node to open.
<code>uint8_t</code>	read read flag.
<code>uint8_t</code>	write write flag

Returns

`uint32_t` : 0 if the node does not support open operation else The result of the open operation.

Definition at line 60 of file vfs.c.

4.16.3.8 uint32_t read_fs (vfs_node_t * node, uint32_t offset, uint32_t size, uint8_t * buffer)

Calls the node's read operation , saved as read in the node structure.

Parameters

<code>vfs_node_t</code>	*node the file's node to be read.
<code>uint32_t</code>	offset where to start reading (0 for beginning of the file).
<code>uint32_t</code>	size size of data to read.
<code>uint8_t</code>	*buffer buffer to be read into.

Returns

`uint32_t` : 0 if the node does not support read operation else The result of the read operation.

Definition at line 27 of file vfs.c.

4.16.3.9 `direntry_t*` `readdir_fs` (`vfs_node_t *node`, `uint32_t index`)

Calls the node's read directory operation , saved as readdir in the node structure.

Parameters

<code>vfs_node_t</code>	*node the directory node to be read.
<code>uint32_t</code>	directory index.

Returns

`direntry_t *` : 0 if the node does not support open operation else The directory entry pointer returned by the readdir operation

Definition at line 89 of file vfs.c.

4.16.3.10 `uint32_t` `write_fs` (`vfs_node_t *node`, `uint32_t offset`, `uint32_t size`, `uint8_t *buffer`)

Calls the node's write operation , saved as write in the node structure.

Parameters

<code>vfs_node_t</code>	*node the file's node to be written into.
<code>uint32_t</code>	offset where to start write (0 for beginning of the file).
<code>uint32_t</code>	size size of data to write.
<code>uint8_t</code>	*buffer buffer containing data to write.

Returns

`uint32_t` : 0 if the node does not support open operation else The result of the write operation.

Definition at line 45 of file vfs.c.

4.16.4 Variable Documentation**4.16.4.1 `direntry_t` `dirent`**

Definition at line 20 of file initrd.c.

4.16.4.2 **initrd_file_header_t* file_headers = NULL [static]**

Definition at line 15 of file initrd.c.

4.16.4.3 **vfs_node_t* fs_root = 0**

Definition at line 13 of file vfs.c.

4.16.4.4 **vfs_node_t* initrd_dev = NULL [static]**

Definition at line 17 of file initrd.c.

4.16.4.5 **initrd_header_t* initrd_header = NULL [static]**

Definition at line 14 of file initrd.c.

4.16.4.6 **vfs_node_t* initrd_root = NULL [static]**

Definition at line 16 of file initrd.c.

4.16.4.7 **int nroot_nodes**

Definition at line 19 of file initrd.c.

4.16.4.8 **vfs_node_t* root_nodes = NULL [static]**

Definition at line 18 of file initrd.c.

4.17 X86-boot

Data Structures

- struct `elfhdr`
Elf binary header values.
- struct `proghdr`
An ELF program header structure.
- struct `sechdr`
An ELF section header structure.

Files

- file `main.c`
Elf binary structures and constants, for parsing binaries.

Functions

- static __inline void `outb` (`uint8_t` data, `int` port)
- static __inline `uint8_t` `inb` (`int` port)
- static void `outw` (`uint16_t` data, `int` port)
- static `uint16_t` `inw` (`int` port)
- static __inline void `outl` (`uint32_t` data, `int` port)
- static `uint32_t` `inl` (`int` port)
- static void `insb` (`void *`addr, `int` cnt, `int` port)
- static void `insl` (`void *`addr, `int` cnt, `int` port)
- static void `insw` (`void *`addr, `int` cnt, `int` port)

Bootloader-KernelLoading

- void `readsect` (`void *`, `uint32_t`)
reads disk at a given offset
- void `readseg` (`uint32_t`, `uint32_t`, `uint32_t`)
loads data from disk into memory with data size to read.
- void `cmain` (`void`)
reads and validates kernel image and loads it and fetch it into IP
- void `waitdisk` (`void`)
hangs till disk operation is done.
- #define `SECTOR` 512
equals the sector size in bytes
- #define `ELFHDR` ((struct `elfhdr` *) 0x10000)
The kernel physical address to load image into.

Elf-Member-Constants

- #define `ELF_MAGIC` 0x464C457F
- #define `MAGIC_LEN` 16
- #define `M_CLASS_OFF` 4
- #define `M_CLASSNONE` 0
- #define `M_CLASS32` 1
- #define `M_CLASS64` 2
- #define `M_CLASSNUM` 3
- #define `M_DATA_OFF` 5
- #define `M_DATANONE` 0
- #define `M_DATA2LE` 1
- #define `M_DATA2BE` 2
- #define `M_DATANUM` 3
- #define `M_VERSION` 6
- #define `M_OSABI` 7
- #define `M_OSABI_SYSV` 0
- #define `M_OSABI_HPUX` 1
- #define `M_ABIVERSION` 8
- #define `M_ELF_PADDING` 9
- #define `T_TYPE_NONE` 0
- #define `T_TYPE_REL` 1
- #define `T_TYPE_EXEC` 2
- #define `T_TYPE_DYN` 3
- #define `T_TYPE_CORE` 4
- #define `T_TYPE_LOPROC` 0xff00
- #define `T_TYPE_HIPROC` 0xffff
- #define `M_MACHINE_I386` 3
- #define `V_VERSION_NONE` 0
- #define `V_VERSION_CURRENT` 1
- #define `V_VERSION_NUM` 2
- #define `P_PROGHDR_R` 0x4
- #define `P_PROGHDR_W` 0x2
- #define `P_PROGHDR_E` 0x1

Elf-structures

- typedef struct `elfhdr` `elfhdr`
- typedef struct `proghdr` `prohdr`
- typedef struct `sechdr` `secthdr`

4.17.1 Define Documentation

4.17.1.1 #define `ELF_MAGIC` 0x464C457F

Definition at line 21 of file elf.h.

4.17.1.2 `#define ELFHDR ((struct elfhdr *) 0x10000)`

The kernel physical address to load image into.

Definition at line 37 of file main.c.

4.17.1.3 `#define M_ABIVERSION 8`

Definition at line 42 of file elf.h.

4.17.1.4 `#define M_CLASS32 1`

Definition at line 26 of file elf.h.

4.17.1.5 `#define M_CLASS64 2`

Definition at line 27 of file elf.h.

4.17.1.6 `#define M_CLASS_OFF 4`

Definition at line 24 of file elf.h.

4.17.1.7 `#define M_CLASSNONE 0`

Definition at line 25 of file elf.h.

4.17.1.8 `#define M_CLASSNUM 3`

Definition at line 28 of file elf.h.

4.17.1.9 `#define M_DATA2BE 2`

Definition at line 33 of file elf.h.

4.17.1.10 `#define M_DATA2LE 1`

Definition at line 32 of file elf.h.

4.17.1.11 `#define M_DATA_OFF 5`

Definition at line 30 of file elf.h.

4.17.1.12 #define M_DATANONE 0

Definition at line 31 of file elf.h.

4.17.1.13 #define M_DATANUM 3

Definition at line 34 of file elf.h.

4.17.1.14 #define M_ELF_PADDING 9

Definition at line 43 of file elf.h.

4.17.1.15 #define M_MACHINE_I386 3

Definition at line 59 of file elf.h.

4.17.1.16 #define M_OSABI 7

Definition at line 38 of file elf.h.

4.17.1.17 #define M_OSABI_HPUX 1

Definition at line 40 of file elf.h.

4.17.1.18 #define M_OSABI_SYSV 0

Definition at line 39 of file elf.h.

4.17.1.19 #define M_VERSION 6

Definition at line 36 of file elf.h.

4.17.1.20 #define MAGIC_LEN 16

Definition at line 22 of file elf.h.

4.17.1.21 #define P_PROGHDR_E 0x1

Definition at line 73 of file elf.h.

4.17.1.22 #define P_PROGHDR_R 0x4

Definition at line 71 of file elf.h.

4.17.1.23 #define P_PROGHDR_W 0x2

Definition at line 72 of file elf.h.

4.17.1.24 #define SECTOR 512

equals the sector size in bytes

Definition at line 32 of file main.c.

4.17.1.25 #define T_TYPE_CORE 4

Definition at line 52 of file elf.h.

4.17.1.26 #define T_TYPE_DYN 3

Definition at line 51 of file elf.h.

4.17.1.27 #define T_TYPE_EXEC 2

Definition at line 50 of file elf.h.

4.17.1.28 #define T_TYPE_HIPROC 0xffff

Definition at line 54 of file elf.h.

4.17.1.29 #define T_TYPE_LOPROC 0xff00

Definition at line 53 of file elf.h.

4.17.1.30 #define T_TYPE_NONE 0

Definition at line 48 of file elf.h.

4.17.1.31 #define T_TYPE_REL 1

Definition at line 49 of file elf.h.

4.17.1.32 `#define V_VERSION_CURRENT 1`

Definition at line 65 of file elf.h.

4.17.1.33 `#define V_VERSION_NONE 0`

Definition at line 64 of file elf.h.

4.17.1.34 `#define V_VERSION_NUM 2`

Definition at line 66 of file elf.h.

4.17.2 Typedef Documentation

4.17.2.1 `typedef struct elfhdr elfhdr`

4.17.2.2 `typedef struct proghdr prohdr`

4.17.2.3 `typedef struct sechdr secthdr`

4.17.3 Function Documentation

4.17.3.1 `void cmain(void)`

reads and validates kernel image and loads it and fetch it into IP.

Returns

nothing.

this function is called by the boot.S after setting up segmentation and operating in protected mode. it loads the kernel image and validate that it's an ELF image, then loads the program headers into memory. and jump to kernel image, if ever a kernel return it will spin or hang. and such thing is merely impossible. initialize magic to the first four characters of ELF MAGIC signature

Definition at line 74 of file main.c.

4.17.3.2 `static __inline uint8_t inb(int port) [static]`

Definition at line 21 of file x86.h.

4.17.3.3 `static uint32_t inl(int port) [inline, static]`

Definition at line 43 of file x86.h.

4.17.3.4 static void insb (void * *addr*, int *cnt*, int *port*) [inline, static]

Definition at line 49 of file x86.h.

4.17.3.5 static void insl (void * *addr*, int *cnt*, int *port*) [inline, static]

Definition at line 55 of file x86.h.

4.17.3.6 static void insw (void * *addr*, int *cnt*, int *port*) [inline, static]

Definition at line 61 of file x86.h.

4.17.3.7 static uint16_t inw (int *port*) [inline, static]

Definition at line 32 of file x86.h.

4.17.3.8 static __inline void outb (uint8_t *data*, int *port*) [static]

Definition at line 17 of file x86.h.

4.17.3.9 static __inline void outl (uint32_t *data*, int *port*) [static]

Definition at line 38 of file x86.h.

4.17.3.10 static void outw (uint16_t *data*, int *port*) [inline, static]

Definition at line 27 of file x86.h.

4.17.3.11 void readsect (void * *dst*, uint32_t *offset*)

reads disk at a given offset

Parameters

void*	va, is the address to load read data into
uint32_t	offset, offset in image

Returns

nothing.

Definition at line 36 of file rdisk.c.

4.17.3.12 void readseg (uint32_t va, uint32_t count, uint32_t offset)

loads data from disk into memory with data size to read.

Parameters

<i>uint32_t</i>	va, virtual address to load data into
<i>uint32_t</i>	count, the number of sectors to load into memory.
<i>uint32_t</i>	offset, the offset of data into disk.

Returns

nothing.

It could be called as a wrapper function to readsect, since readseg uses readsect int its operations.

Definition at line 11 of file rdisk.c.

4.17.3.13 void waitdisk (void)

hangs till disk operation is done.

Returns

nothing.

this function just reads the disk port status register holds the value of 0x40 which means disk is not executing a command and ready.

Definition at line 110 of file main.c.

Chapter 5

Data Structure Documentation

5.1 __attribute Struct Reference

```
#include <multiboot.h>
```

Data Fields

- `uint32_t flags`
- `uint32_t mem_lower`
- `uint32_t mem_upper`
- `uint32_t boot_device`
- `uint32_t cmdline`
- `uint32_t mods_count`
- `uint32_t mods_addr`
- `union {`
 - `multiboot_aout_symbol_table_t aout_sym`
 - `multiboot_elf_section_header_t elf_sec``} u`
- `uint32_t mmap_length`
- `uint32_t mmap_addr`
- `uint32_t drives_length`
- `uint32_t drives_addr`
- `uint32_t config_table`
- `uint32_t boot_loader_name`
- `uint32_t apm_table`
- `uint32_t vbe_control_info`
- `uint32_t vbe_mode_info`
- `uint16_t vbe_mode`
- `uint16_t vbe_interface_seg`
- `uint16_t vbe_interface_off`
- `uint16_t vbe_interface_len`

5.1.1 Detailed Description

Definition at line 82 of file multiboot.h.

5.1.2 Field Documentation

5.1.2.1 `multiboot_aout_symbol_table_t __attribute::aout_sym`

Definition at line 98 of file multiboot.h.

5.1.2.2 `uint32_t __attribute::apm_table`

Definition at line 112 of file multiboot.h.

5.1.2.3 `uint32_t __attribute::boot_device`

Definition at line 90 of file multiboot.h.

5.1.2.4 `uint32_t __attribute::boot_loader_name`

Definition at line 110 of file multiboot.h.

5.1.2.5 `uint32_t __attribute::cmdline`

Definition at line 92 of file multiboot.h.

5.1.2.6 `uint32_t __attribute::config_table`

Definition at line 108 of file multiboot.h.

5.1.2.7 `uint32_t __attribute::drives_addr`

Definition at line 106 of file multiboot.h.

5.1.2.8 `uint32_t __attribute::drives_length`

Definition at line 105 of file multiboot.h.

5.1.2.9 `multiboot_elf_section_header_t __attribute::elf_sec`

Definition at line 99 of file multiboot.h.

5.1.2.10 `uint32_t __attribute::flags`

Definition at line 85 of file multiboot.h.

5.1.2.11 `uint32_t __attribute::mem_lower`

Definition at line 87 of file multiboot.h.

5.1.2.12 `uint32_t __attribute::mem_upper`

Definition at line 88 of file multiboot.h.

5.1.2.13 `uint32_t __attribute::mmap_addr`

Definition at line 103 of file multiboot.h.

5.1.2.14 `uint32_t __attribute::mmap_length`

Definition at line 102 of file multiboot.h.

5.1.2.15 `uint32_t __attribute::mods_addr`

Definition at line 95 of file multiboot.h.

5.1.2.16 `uint32_t __attribute::mods_count`

Definition at line 94 of file multiboot.h.

5.1.2.17 `union { ... } __attribute::u`

5.1.2.18 `uint32_t __attribute::vbe_control_info`

Definition at line 114 of file multiboot.h.

5.1.2.19 `uint16_t __attribute::vbe_interface_len`

Definition at line 119 of file multiboot.h.

5.1.2.20 `uint16_t __attribute::vbe_interface_off`

Definition at line 118 of file multiboot.h.

5.1.2.21 uint16_t __attribute__::vbe_interface_seg

Definition at line 117 of file multiboot.h.

5.1.2.22 uint16_t __attribute__::vbe_mode

Definition at line 116 of file multiboot.h.

5.1.2.23 uint32_t __attribute__::vbe_mode_info

Definition at line 115 of file multiboot.h.

The documentation for this struct was generated from the following file:

- include/multiboot.h

5.2 __attribute__ Struct Reference

Task State segment.

```
#include <cpu_state.h>
```

Data Fields

- `uint16_t prelink`
- `uint16_t _rsrvd1`
- `reg_t esp0`
- `uint16_t ss0`
- `uint16_t _rsrvd2`
- `reg_t esp1`
- `uint16_t ss1`
- `uint16_t _rsrvd3`
- `reg_t esp2`
- `uint16_t ss2`
- `uint16_t _rsrvd4`
- `uint32_t cr3`
- `reg_t eip`
- `reg_t eflags`
- `reg_t eax`
- `reg_t ecx`
- `reg_t edx`
- `reg_t ebx`
- `reg_t esp`
- `reg_t ebp`
- `reg_t esi`

- `reg_t edi`
- `uint16_t es`
- `uint16_t _rsrvd5`
- `uint16_t cs`
- `uint16_t _rsrvd6`
- `uint16_t ss`
- `uint16_t _rsrvd7`
- `uint16_t ds`
- `uint16_t _rsrvd8`
- `uint16_t fs`
- `uint16_t _rsrvd9`
- `uint16_t gs`
- `uint16_t _rsrvda`
- `uint16_t ldt`
- `uint16_t _rsrvdb`
- `uint16_t trace`
- `uint16_t iomap_base`
- `uint16_t limit_0_15`
- `uint16_t base_0_15`
- `uint8_t base_16_23`
- `uint8_t access`
- `unsigned limit_16_19:4`
- `unsigned available:1`
- `unsigned unused:2`
- `unsigned granularity:1`
- `uint8_t base_24_31`
- `uint16_t offset_0_15`
- `uint16_t segment_s`
- `unsigned args:5`
- `unsigned reserved:3`
- `unsigned type:4`
- `unsigned s:1`
- `unsigned dpl:2`
- `unsigned p:1`
- `uint16_t offset_16_31`
- `unsigned present:1`
- `unsigned writable:1`
- `unsigned accessible:1`
- `unsigned write_through:1`
- `unsigned cache_disable:1`
- `unsigned accessed:1`
- `unsigned dirty:1`
- `unsigned pat:1`
- `unsigned global:1`
- `unsigned ignore_this:3`
- `unsigned address:20`

- union {
 - uint32_t lo}
- union {
 - uint32_t hi}
- uint32_t Signature
- uintptr_t config_addr
- uint8_t len
- uint8_t version
- uint8_t checksum
- uint8_t feature1
- uint8_t feature2
- uint16_t base_table_len
- uint8_t spec_ver
- uint8_t oem_id [8]
- uint8_t product_id [12]
- uintptr_t oem_table_pointer
- uint16_t oem_table_size
- uint16_t entry_count
- uint32_t lapic_addr
- uint16_t extnd_table_len
- uint8_t extnd_table_checksum
- uint8_t reserved
- uint8_t base_table [0]
- uint8_t entry_type
- uint8_t lapic_id
- uint8_t lapic_version
- uint8_t cpu_flags
- uint8_t cpu_signature [4]
- uint32_t feature_flags
- uint8_t bus_id
- uint8_t bus_type [6]
- uint8_t io_apic_id
- uint8_t io_apic_ver
- uint8_t io_apic_flags
- uint8_t io_apic_addr
- uint8_t type
- uint8_t intrrupt_type
- uint8_t po_el
- uint8_t pad
- uint8_t src_bus_id
- uint8_t src_bus_irq
- uint8_t dst_io_apic_id
- uint8_t dst_io_apic_pin

- `uint8_t dst_loc_apic_id`
- `uint8_t dst_loc_apic_pin`
- `uint32_t magic`
- `uint32_t flags`
- `uint32_t checksum`
- `uint32_t header_addr`
- `uint32_t load_addr`
- `uint32_t load_end_addr`
- `uint32_t bss_end_addr`
- `uint32_t entry_addr`
- `uint32_t mode_type`
- `uint32_t width`
- `uint32_t height`
- `uint32_t depth`
- `uint32_t num`
- `uint32_t size`
- `uint32_t addr`
- `uint32_t shndx`
- `uint32_t tabsize`
- `uint32_t strsize`
- `uint32_t reserved`
- `uint32_t moduleStart`
- `uint32_t moduleEnd`
- `char string [8]`
- `uint64_t addr`
- `uint64_t len`
- `uint32_t type`
- `uint32_t mod_start`
- `uint32_t mod_end`
- `uint32_t cmdline`
- `uint32_t pad`
- `uint8_t vector`
- `unsigned int delivery_mode: 3`
- `unsigned int dest_mode: 1`
- `unsigned int delivery_stat: 1`
- `unsigned int: 1`
- `unsigned int level: 1`
- `unsigned int trigger_mode: 1`
- `unsigned int shorthand: 2`
- `uint8_t dest`

5.2.1 Detailed Description

Task State segment.

Task state segment descriptor.

The Task state segment entry fields, merely used into the kernel.

TSS is only used to support interrupts from user space. other than that soft task switching is used.

TSS descriptor is somehow a child of a gate descriptor and similar to segment selectors.

Gate descriptors as defined by Intel Manuals.

Definition at line 109 of file cpu_state.h.

5.2.2 Field Documentation

5.2.2.1 `union { ... }`

5.2.2.2 `union { ... }`

5.2.2.3 `uint16_t __attribute__::__rsrvd1`

Definition at line 110 of file cpu_state.h.

5.2.2.4 `uint16_t __attribute__::__rsrvd2`

Definition at line 112 of file cpu_state.h.

5.2.2.5 `uint16_t __attribute__::__rsrvd3`

Definition at line 114 of file cpu_state.h.

5.2.2.6 `uint16_t __attribute__::__rsrvd4`

Definition at line 116 of file cpu_state.h.

5.2.2.7 `uint16_t __attribute__::__rsrvd5`

Definition at line 128 of file cpu_state.h.

5.2.2.8 `uint16_t __attribute__::__rsrvd6`

Definition at line 129 of file cpu_state.h.

5.2.2.9 uint16_t __attribute__::__rsrvd7

Definition at line 130 of file cpu_state.h.

5.2.2.10 uint16_t __attribute__::__rsrvd8

Definition at line 131 of file cpu_state.h.

5.2.2.11 uint16_t __attribute__::__rsrvd9

Definition at line 132 of file cpu_state.h.

5.2.2.12 uint16_t __attribute__::__rsrvda

Definition at line 133 of file cpu_state.h.

5.2.2.13 uint16_t __attribute__::__rsrvdb

Definition at line 134 of file cpu_state.h.

5.2.2.14 uint8_t __attribute__::__access

Definition at line 150 of file cpu_state.h.

5.2.2.15 unsigned __attribute__::__accessed

Definition at line 42 of file page.h.

5.2.2.16 unsigned __attribute__::__accessible

Definition at line 39 of file page.h.

5.2.2.17 uint32_t __attribute__::__addr

Definition at line 62 of file multiboot.h.

5.2.2.18 uint64_t __attribute__::__addr

Definition at line 125 of file multiboot.h.

5.2.2.19 `unsigned __attribute__::address`

Definition at line 47 of file page.h.

5.2.2.20 `unsigned __attribute__::args`

Definition at line 29 of file interrupt.h.

5.2.2.21 `unsigned __attribute__::available`

Definition at line 152 of file cpu_state.h.

5.2.2.22 `uint16_t __attribute__::base_0_15`

Definition at line 148 of file cpu_state.h.

5.2.2.23 `uint8_t __attribute__::base_16_23`

Definition at line 149 of file cpu_state.h.

5.2.2.24 `uint8_t __attribute__::base_24_31`

Definition at line 155 of file cpu_state.h.

5.2.2.25 `uint8_t __attribute__::base_table[0]`

Definition at line 65 of file smp.h.

5.2.2.26 `uint16_t __attribute__::base_table_len`

Definition at line 52 of file smp.h.

5.2.2.27 `uint32_t __attribute__::bss_end_addr`

Definition at line 47 of file multiboot.h.

5.2.2.28 `uint8_t __attribute__::bus_id`

Definition at line 83 of file smp.h.

5.2.2.29 `uint8_t __attribute__::bus_type[6]`

Definition at line 84 of file smp.h.

5.2.2.30 `unsigned __attribute__::cache_disable`

Definition at line 41 of file page.h.

5.2.2.31 `uint32_t __attribute__::checksum`

Definition at line 42 of file multiboot.h.

5.2.2.32 `uint8_t __attribute__::checksum`

Definition at line 43 of file smp.h.

5.2.2.33 `uint32_t __attribute__::cmdline`

Definition at line 138 of file multiboot.h.

5.2.2.34 `uintptr_t __attribute__::config_addr`

Definition at line 40 of file smp.h.

5.2.2.35 `uint8_t __attribute__::cpu_flags`

Definition at line 73 of file smp.h.

5.2.2.36 `uint8_t __attribute__::cpu_signature[4]`

Definition at line 77 of file smp.h.

5.2.2.37 `uint32_t __attribute__::cr3`

Definition at line 117 of file cpu_state.h.

5.2.2.38 `uint16_t __attribute__::cs`

Definition at line 129 of file cpu_state.h.

5.2.2.39 `unsigned int __attribute__::delivery_mode`

Definition at line 81 of file apic.h.

5.2.2.40 `unsigned int __attribute__::delivery_stat`

Definition at line 83 of file apic.h.

5.2.2.41 `uint32_t __attribute__::depth`

Definition at line 54 of file multiboot.h.

5.2.2.42 `uint8_t __attribute__::dest`

Definition at line 96 of file apic.h.

5.2.2.43 `unsigned int __attribute__::dest_mode`

Definition at line 82 of file apic.h.

5.2.2.44 `unsigned __attribute__::dirty`

Definition at line 43 of file page.h.

5.2.2.45 `unsigned __attribute__::dpl`

Definition at line 41 of file interrupt.h.

5.2.2.46 `uint16_t __attribute__::ds`

Definition at line 131 of file cpu_state.h.

5.2.2.47 `uint8_t __attribute__::dst_io_apic_id`

Definition at line 103 of file smp.h.

5.2.2.48 `uint8_t __attribute__::dst_io_apic_pin`

Definition at line 104 of file smp.h.

5.2.2.49 `uint8_t __attribute__::dst_loc_apic_id`

Definition at line 114 of file smp.h.

5.2.2.50 `uint8_t __attribute__::dst_loc_apic_pin`

Definition at line 115 of file smp.h.

5.2.2.51 `reg_t __attribute__::eax`

Definition at line 120 of file cpu_state.h.

5.2.2.52 `reg_t __attribute__::ebp`

Definition at line 125 of file cpu_state.h.

5.2.2.53 `reg_t __attribute__::ebx`

Definition at line 123 of file cpu_state.h.

5.2.2.54 `reg_t __attribute__::ecx`

Definition at line 121 of file cpu_state.h.

5.2.2.55 `reg_t __attribute__::edi`

Definition at line 127 of file cpu_state.h.

5.2.2.56 `reg_t __attribute__::edx`

Definition at line 122 of file cpu_state.h.

5.2.2.57 `reg_t __attribute__::eflags`

Definition at line 119 of file cpu_state.h.

5.2.2.58 `reg_t __attribute__::eip`

Definition at line 118 of file cpu_state.h.

5.2.2.59 `uint32_t __attribute__::entry_addr`

Definition at line 48 of file multiboot.h.

5.2.2.60 `uint16_t __attribute__::entry_count`

Definition at line 59 of file smp.h.

5.2.2.61 `uint8_t __attribute__::entry_type`

Definition at line 70 of file smp.h.

5.2.2.62 `uint16_t __attribute__::es`

Definition at line 128 of file cpu_state.h.

5.2.2.63 `reg_t __attribute__::esi`

Definition at line 126 of file cpu_state.h.

5.2.2.64 `reg_t __attribute__::esp`

Definition at line 124 of file cpu_state.h.

5.2.2.65 `reg_t __attribute__::esp0`

Definition at line 111 of file cpu_state.h.

5.2.2.66 `reg_t __attribute__::esp1`

Definition at line 113 of file cpu_state.h.

5.2.2.67 `reg_t __attribute__::esp2`

Definition at line 115 of file cpu_state.h.

5.2.2.68 `uint8_t __attribute__::extnd_table_checksum`

Definition at line 62 of file smp.h.

5.2.2.69 `uint16_t __attribute__::extnd_table_len`

Definition at line 61 of file smp.h.

5.2.2.70 `uint8_t __attribute__::feature1`

Definition at line 44 of file smp.h.

5.2.2.71 `uint8_t __attribute__::feature2`

Definition at line 45 of file smp.h.

5.2.2.72 `uint32_t __attribute__::feature_flags`

Definition at line 78 of file smp.h.

5.2.2.73 `uint32_t __attribute__::flags`

Definition at line 40 of file multiboot.h.

5.2.2.74 `uint16_t __attribute__::fs`

Definition at line 132 of file cpu_state.h.

5.2.2.75 `unsigned __attribute__::global`

Definition at line 45 of file page.h.

5.2.2.76 `unsigned __attribute__::granularity`

Definition at line 154 of file cpu_state.h.

5.2.2.77 `uint16_t __attribute__::gs`

Definition at line 133 of file cpu_state.h.

5.2.2.78 `uint32_t __attribute__::header_addr`

Definition at line 44 of file multiboot.h.

5.2.2.79 `uint32_t __attribute__::height`

Definition at line 53 of file multiboot.h.

5.2.2.80 `uint32_t __attribute__::hi`

Definition at line 93 of file apic.h.

5.2.2.81 `unsigned __attribute__::ignore_this`

Definition at line 46 of file page.h.

5.2.2.82 `unsigned __attribute__::int`

Definition at line 84 of file apic.h.

5.2.2.83 `uint8_t __attribute__::interrupt_type`

Definition at line 98 of file smp.h.

5.2.2.84 `uint8_t __attribute__::io_apic_addr`

Definition at line 93 of file smp.h.

5.2.2.85 `uint8_t __attribute__::io_apic_flags`

Definition at line 92 of file smp.h.

5.2.2.86 `uint8_t __attribute__::io_apic_id`

Definition at line 90 of file smp.h.

5.2.2.87 `uint8_t __attribute__::io_apic_ver`

Definition at line 91 of file smp.h.

5.2.2.88 `uint16_t __attribute__::iomap_base`

Definition at line 136 of file cpu_state.h.

5.2.2.89 `uint32_t __attribute__::lapic_addr`

Definition at line 60 of file smp.h.

5.2.2.90 `uint8_t __attribute__::lapic_id`

Definition at line 71 of file smp.h.

5.2.2.91 `uint8_t __attribute__::lapic_version`

Definition at line 72 of file smp.h.

5.2.2.92 `uint16_t __attribute__::ldt`

Definition at line 134 of file cpu_state.h.

5.2.2.93 `uint8_t __attribute__::len`

Definition at line 41 of file smp.h.

5.2.2.94 `uint64_t __attribute__::len`

Definition at line 126 of file multiboot.h.

5.2.2.95 `unsigned int __attribute__::level`

Definition at line 85 of file apic.h.

5.2.2.96 `uint16_t __attribute__::limit_0_15`

Definition at line 147 of file cpu_state.h.

5.2.2.97 `unsigned __attribute__::limit_16_19`

Definition at line 151 of file cpu_state.h.

5.2.2.98 `uint32_t __attribute__::lo`

Definition at line 78 of file apic.h.

5.2.2.99 `uint32_t __attribute__::load_addr`

Definition at line 45 of file multiboot.h.

5.2.2.100 `uint32_t __attribute__::load_end_addr`

Definition at line 46 of file multiboot.h.

5.2.2.101 `uint32_t __attribute__::magic`

Definition at line 39 of file multiboot.h.

5.2.2.102 `uint32_t __attribute__::mod_end`

Definition at line 136 of file multiboot.h.

5.2.2.103 `uint32_t __attribute__::mod_start`

Definition at line 135 of file multiboot.h.

5.2.2.104 `uint32_t __attribute__::mode_type`

Definition at line 51 of file multiboot.h.

5.2.2.105 `uint32_t __attribute__::moduleEnd`

Definition at line 78 of file multiboot.h.

5.2.2.106 `uint32_t __attribute__::moduleStart`

Definition at line 77 of file multiboot.h.

5.2.2.107 `uint32_t __attribute__::num`

Definition at line 60 of file multiboot.h.

5.2.2.108 `uint8_t __attribute__::oem_id[8]`

Definition at line 55 of file smp.h.

5.2.2.109 `uintptr_t __attribute__::oem_table_pointer`

Definition at line 57 of file smp.h.

5.2.2.110 `uint16_t __attribute__::oem_table_size`

Definition at line 58 of file smp.h.

5.2.2.111 `uint16_t __attribute__::offset_0_15`

Definition at line 22 of file interrupt.h.

5.2.2.112 `uint16_t __attribute__::offset_16_31`

Definition at line 43 of file interrupt.h.

5.2.2.113 `unsigned __attribute__::p`

Definition at line 42 of file interrupt.h.

5.2.2.114 `uint8_t __attribute__::pad`

Definition at line 100 of file smp.h.

5.2.2.115 `uint32_t __attribute__::pad`

Definition at line 140 of file multiboot.h.

5.2.2.116 `unsigned __attribute__::pat`

Definition at line 44 of file page.h.

5.2.2.117 `uint8_t __attribute__::po_el`

Definition at line 99 of file smp.h.

5.2.2.118 `uint16_t __attribute__::prelink`

Definition at line 110 of file cpu_state.h.

5.2.2.119 `unsigned __attribute__::present`

Definition at line 37 of file page.h.

5.2.2.120 `uint8_t __attribute__::product_id[12]`

Definition at line 56 of file smp.h.

5.2.2.121 `unsigned __attribute__::reserved`

Definition at line 30 of file interrupt.h.

5.2.2.122 `uint8_t __attribute__::reserved`

Definition at line 64 of file smp.h.

5.2.2.123 `uint32_t __attribute__::reserved`

Definition at line 71 of file multiboot.h.

5.2.2.124 `unsigned __attribute__::s`

Definition at line 40 of file interrupt.h.

5.2.2.125 `uint16_t __attribute__::segment_s`

Definition at line 23 of file interrupt.h.

5.2.2.126 `uint32_t __attribute__::shndx`

Definition at line 63 of file multiboot.h.

5.2.2.127 `unsigned int __attribute__::shorthand`

Definition at line 88 of file apic.h.

5.2.2.128 `uint32_t __attribute__::Signature`

Definition at line 39 of file smp.h.

5.2.2.129 `uint32_t __attribute__::size`

Definition at line 61 of file multiboot.h.

5.2.2.130 `uint8_t __attribute__::spec_ver`

Definition at line 53 of file smp.h.

5.2.2.131 `uint8_t __attribute__::src_bus_id`

Definition at line 101 of file smp.h.

5.2.2.132 `uint8_t __attribute__::src_bus_irq`

Definition at line 102 of file smp.h.

5.2.2.133 `uint16_t __attribute__::ss`

Definition at line 130 of file cpu_state.h.

5.2.2.134 `uint16_t __attribute__::ss0`

Definition at line 112 of file cpu_state.h.

5.2.2.135 `uint16_t __attribute__::ss1`

Definition at line 114 of file cpu_state.h.

5.2.2.136 `uint16_t __attribute__::ss2`

Definition at line 116 of file cpu_state.h.

5.2.2.137 `char __attribute__::string[8]`

Definition at line 79 of file multiboot.h.

5.2.2.138 `uint32_t __attribute__::strsize`

Definition at line 69 of file multiboot.h.

5.2.2.139 `uint32_t __attribute__::tabsize`

Definition at line 68 of file multiboot.h.

5.2.2.140 `uint16_t __attribute__::trace`

Definition at line 135 of file cpu_state.h.

5.2.2.141 `unsigned int __attribute__::trigger_mode`

Definition at line 86 of file apic.h.

5.2.2.142 `uint8_t __attribute__::type`

Definition at line 39 of file interrupt.h.

5.2.2.143 `uint8_t __attribute__::type`

Definition at line 97 of file smp.h.

5.2.2.144 `uint32_t __attribute__::type`

Definition at line 129 of file multiboot.h.

5.2.2.145 `unsigned __attribute__::unused`

Definition at line 153 of file cpu_state.h.

5.2.2.146 `uint8_t __attribute__::vector`

Definition at line 80 of file apic.h.

5.2.2.147 `uint8_t __attribute__::version`

Definition at line 42 of file smp.h.

5.2.2.148 `uint32_t __attribute__::width`

Definition at line 52 of file multiboot.h.

5.2.2.149 unsigned __attribute__::writable

Definition at line 38 of file page.h.

5.2.2.150 unsigned __attribute__::write_through

Definition at line 40 of file page.h.

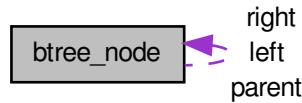
The documentation for this struct was generated from the following files:

- include/arch/x86/cpu_state.h
- include/arch/x86/interrupt.h
- include/arch/x86/mm/page.h
- include/arch/x86/mp/apic.h
- include/arch/x86/mp/smp.h
- include/multiboot.h

5.3 btree_node Struct Reference

```
#include <btree.h>
```

Collaboration diagram for btree_node:



Data Fields

- struct `btree_node` * parent
- struct `btree_node` * left
- struct `btree_node` * right
- `uint32_t` key
- `uint32_t` val

5.3.1 Detailed Description

Definition at line 9 of file btree.h.

5.3.2 Field Documentation

5.3.2.1 `uint32_t btree_node::key`

Definition at line 13 of file btree.h.

5.3.2.2 `struct btree_node* btree_node::left`

Definition at line 11 of file btree.h.

5.3.2.3 `struct btree_node* btree_node::parent`

Definition at line 10 of file btree.h.

5.3.2.4 `struct btree_node* btree_node::right`

Definition at line 12 of file btree.h.

5.3.2.5 `uint32_t btree_node::val`

Definition at line 14 of file btree.h.

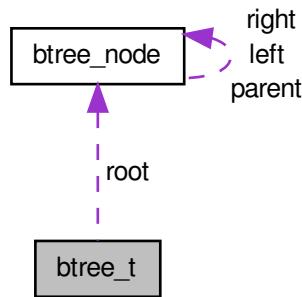
The documentation for this struct was generated from the following file:

- include/structs/btree.h

5.4 `btree_t` Struct Reference

```
#include <btree.h>
```

Collaboration diagram for btree_t:



Public Member Functions

- [LIST_ENTRY \(btree_t\) link](#)

Data Fields

- `btree_node_t * root`
- `uint32_t size`

5.4.1 Detailed Description

Definition at line 17 of file btree.h.

5.4.2 Member Function Documentation

5.4.2.1 btree_t::LIST_ENTRY (btree_t)

5.4.3 Field Documentation

5.4.3.1 btree_node_t* btree_t::root

Definition at line 18 of file btree.h.

5.4.3.2 `uint32_t btree_t::size`

Definition at line 19 of file btree.h.

The documentation for this struct was generated from the following file:

- include/structs/btree.h

5.5 `cpu_cache` Struct Reference

```
#include <processor.h>
```

Public Member Functions

- struct {
 unsigned `line_size`:12
 unsigned `phy_line`:10
 unsigned `assoc`:10
} `__attribute__((packed)) lpw`
- struct {
 unsigned `write_back`:1
 unsigned `inclusive`:1
 unsigned `indexing`:1
 unsigned `__pad0__`:29
} `__attribute__((packed)) wcc`

Data Fields

- `uint32_t eax`
- `uint32_t sets`

5.5.1 Detailed Description

Definition at line 135 of file processor.h.

5.5.2 Member Function Documentation

5.5.2.1 `struct cpu_cache::@14 cpu_cache::__attribute__((packed))`

5.5.2.2 `struct cpu_cache::@15 cpu_cache::__attribute__((packed))`

5.5.3 Field Documentation

5.5.3.1 unsigned cpu_cache::__pad0__

Definition at line 147 of file processor.h.

5.5.3.2 unsigned cpu_cache::assoc

Definition at line 140 of file processor.h.

5.5.3.3 uint32_t cpu_cache::eax

Definition at line 136 of file processor.h.

5.5.3.4 unsigned cpu_cache::inclusive

Definition at line 145 of file processor.h.

5.5.3.5 unsigned cpu_cache::indexing

Definition at line 146 of file processor.h.

5.5.3.6 unsigned cpu_cache::line_size

Definition at line 138 of file processor.h.

5.5.3.7 unsigned cpu_cache::phy_line

Definition at line 139 of file processor.h.

5.5.3.8 uint32_t cpu_cache::sets

Definition at line 142 of file processor.h.

5.5.3.9 unsigned cpu_cache::write_back

Definition at line 144 of file processor.h.

The documentation for this struct was generated from the following file:

- include/arch/x86/processor.h

5.6 cpu_state_t Struct Reference

Interaupt/Trap cpu state.

```
#include <cpu_state.h>
```

Data Fields

- `reg_t ebp_frame`
- `reg_t eip_frame`
- `reg_t gs`
- `reg_t fs`
- `reg_t es`
- `reg_t ds`
- `reg_t edi`
- `reg_t esi`
- `reg_t o_ebp`
- `reg_t o_esp`
- `reg_t ebx`
- `reg_t edx`
- `reg_t ecx`
- `reg_t eax`
- `uint32_t error_code`
- `reg_t eip`
- `reg_t cs`
- `reg_t eflags`
- `reg_t esp`
- `reg_t ss`

5.6.1 Detailed Description

Interaupt/Trap cpu state.

the `cpu_state` structure extends the default intel hardware interrupt/trap frame stored into stack. More detailed than the hardware frame.

Definition at line 25 of file `cpu_state.h`.

5.6.2 Field Documentation

5.6.2.1 `reg_t cpu_state_t::cs`

Definition at line 61 of file `cpu_state.h`.

5.6.2.2 `reg_t cpu_state_t::ds`

Definition at line 38 of file `cpu_state.h`.

5.6.2.3 `reg_t cpu_state_t::eax`

Definition at line 51 of file `cpu_state.h`.

5.6.2.4 `reg_t cpu_state_t::ebp_frame`

Definition at line 30 of file `cpu_state.h`.

5.6.2.5 `reg_t cpu_state_t::ebx`

Definition at line 48 of file `cpu_state.h`.

5.6.2.6 `reg_t cpu_state_t::ecx`

Definition at line 50 of file `cpu_state.h`.

5.6.2.7 `reg_t cpu_state_t::edi`

Definition at line 44 of file `cpu_state.h`.

5.6.2.8 `reg_t cpu_state_t::edx`

Definition at line 49 of file `cpu_state.h`.

5.6.2.9 `reg_t cpu_state_t::eflags`

Definition at line 62 of file `cpu_state.h`.

5.6.2.10 `reg_t cpu_state_t::eip`

Definition at line 60 of file `cpu_state.h`.

5.6.2.11 `reg_t cpu_state_t::eip_frame`

Definition at line 31 of file `cpu_state.h`.

5.6.2.12 `uint32_t cpu_state_t::error_code`

Definition at line 59 of file `cpu_state.h`.

5.6.2.13 reg_t cpu_state_t::es

Definition at line 37 of file `cpu_state.h`.

5.6.2.14 reg_t cpu_state_t::esi

Definition at line 45 of file `cpu_state.h`.

5.6.2.15 reg_t cpu_state_t::esp

Definition at line 63 of file `cpu_state.h`.

5.6.2.16 reg_t cpu_state_t::fs

Definition at line 36 of file `cpu_state.h`.

5.6.2.17 reg_t cpu_state_t::gs

Definition at line 35 of file `cpu_state.h`.

5.6.2.18 reg_t cpu_state_t::o_ebp

Definition at line 46 of file `cpu_state.h`.

5.6.2.19 reg_t cpu_state_t::o_esp

Definition at line 47 of file `cpu_state.h`.

5.6.2.20 reg_t cpu_state_t::ss

Definition at line 64 of file `cpu_state.h`.

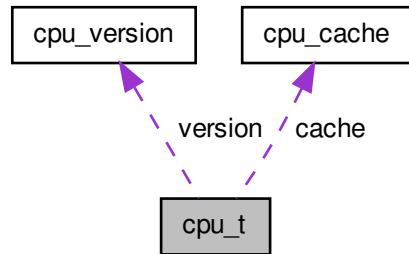
The documentation for this struct was generated from the following file:

- `include/arch/x86/cpu_state.h`

5.7 `cpu_t` Struct Reference

```
#include <processor.h>
```

Collaboration diagram for `cpu_t`:



Data Fields

- `uint32_t vendor`
- `struct cpu_version version`
- `struct cpu_cache cache`

5.7.1 Detailed Description

Definition at line 153 of file processor.h.

5.7.2 Field Documentation

5.7.2.1 `struct cpu_cache cpu_t::cache`

Definition at line 156 of file processor.h.

5.7.2.2 `uint32_t cpu_t::vendor`

Definition at line 154 of file processor.h.

5.7.2.3 `struct cpu_version cpu_t::version`

Definition at line 155 of file processor.h.

The documentation for this struct was generated from the following file:

- include/arch/x86/processor.h

5.8 cpu_version Struct Reference

```
#include <processor.h>
```

Public Member Functions

- struct {
 unsigned sse3:1
 unsigned __pad0__:4
 unsigned vmx:1
 unsigned smx:1
 unsigned est:1
 unsigned tm2:1
 unsigned ssse3:1
 unsigned __pad1__:7
 unsigned pcid:1
 unsigned dca:1
 unsigned sse4_1:1
 unsigned sse4_2:1
 unsigned x2apic:1
 unsigned __pad2__:10
} __attribute__ ((packed)) featured_ecx

- struct {
 unsigned fpu:1
 unsigned vme:1
 unsigned de:1
 unsigned pse:1
 unsigned __pad0__:2
 unsigned pae:1
 unsigned __pad1__:2
 unsigned apic:1
 unsigned __pad2__:1
 unsigned sep:1
 unsigned pge:1
 unsigned mca:1
 unsigned cmov:1
 unsigned pat:1
 unsigned __pad3__:5
 unsigned acpi:1
 unsigned mmx:1
 unsigned fxsr:1
 unsigned sse:1
 unsigned sse2:1
 unsigned ss:1
 unsigned ht:1
 unsigned tm:1
 unsigned __pad4__:1

```
    unsigned pbe:1
} __attribute__ ((packed)) featured_edx
```

Data Fields

- struct {
 unsigned stepping:4
 unsigned model:4
 unsigned family:4
 unsigned type:2
 unsigned ss:2
 unsigned exmodel:4
 unsigned exfamily:8
 unsigned sss:4
} fms
- struct {
 char brand
 char cache
 char max_id
 char id
} generic

5.8.1 Detailed Description

Definition at line 73 of file processor.h.

5.8.2 Member Function Documentation

5.8.2.1 struct `cpu_version`::@8 `cpu_version`::__attribute__ ((packed))

5.8.2.2 struct `cpu_version`::@9 `cpu_version`::__attribute__ ((packed))

5.8.3 Field Documentation

5.8.3.1 unsigned `cpu_version`::__pad0__

Definition at line 92 of file processor.h.

5.8.3.2 unsigned `cpu_version`::__pad1__

Definition at line 98 of file processor.h.

5.8.3.3 `unsigned cpu_version::__pad2__`

Definition at line 104 of file processor.h.

5.8.3.4 `unsigned cpu_version::__pad3__`

Definition at line 121 of file processor.h.

5.8.3.5 `unsigned cpu_version::__pad4__`

Definition at line 130 of file processor.h.

5.8.3.6 `unsigned cpu_version::acpi`

Definition at line 122 of file processor.h.

5.8.3.7 `unsigned cpu_version::apic`

Definition at line 114 of file processor.h.

5.8.3.8 `char cpu_version::brand`

Definition at line 85 of file processor.h.

5.8.3.9 `char cpu_version::cache`

Definition at line 86 of file processor.h.

5.8.3.10 `unsigned cpu_version::cmov`

Definition at line 119 of file processor.h.

5.8.3.11 `unsigned cpu_version::dca`

Definition at line 100 of file processor.h.

5.8.3.12 `unsigned cpu_version::de`

Definition at line 109 of file processor.h.

5.8.3.13 unsigned cpu_version::est

Definition at line 95 of file processor.h.

5.8.3.14 unsigned cpu_version::exfamily

Definition at line 81 of file processor.h.

5.8.3.15 unsigned cpu_version::exmodel

Definition at line 80 of file processor.h.

5.8.3.16 unsigned cpu_version::family

Definition at line 77 of file processor.h.

5.8.3.17 struct { ... } cpu_version::fms**5.8.3.18 unsigned cpu_version::fpu**

Definition at line 107 of file processor.h.

5.8.3.19 unsigned cpu_version::fxsr

Definition at line 124 of file processor.h.

5.8.3.20 struct { ... } cpu_version::generic**5.8.3.21 unsigned cpu_version::htt**

Definition at line 128 of file processor.h.

5.8.3.22 char cpu_version::id

Definition at line 88 of file processor.h.

5.8.3.23 char cpu_version::max_id

Definition at line 87 of file processor.h.

5.8.3.24 unsigned cpu_version::mca

Definition at line 118 of file processor.h.

5.8.3.25 unsigned cpu_version::mmx

Definition at line 123 of file processor.h.

5.8.3.26 unsigned cpu_version::model

Definition at line 76 of file processor.h.

5.8.3.27 unsigned cpu_version::pae

Definition at line 112 of file processor.h.

5.8.3.28 unsigned cpu_version::pat

Definition at line 120 of file processor.h.

5.8.3.29 unsigned cpu_version::pbe

Definition at line 131 of file processor.h.

5.8.3.30 unsigned cpu_version::pcid

Definition at line 99 of file processor.h.

5.8.3.31 unsigned cpu_version::pge

Definition at line 117 of file processor.h.

5.8.3.32 unsigned cpu_version::pse

Definition at line 110 of file processor.h.

5.8.3.33 unsigned cpu_version::sep

Definition at line 116 of file processor.h.

5.8.3.34 unsigned cpu_version::smx

Definition at line 94 of file processor.h.

5.8.3.35 unsigned cpu_version::ss

Definition at line 79 of file processor.h.

5.8.3.36 unsigned cpu_version::sse

Definition at line 125 of file processor.h.

5.8.3.37 unsigned cpu_version::sse2

Definition at line 126 of file processor.h.

5.8.3.38 unsigned cpu_version::sse3

Definition at line 91 of file processor.h.

5.8.3.39 unsigned cpu_version::sse4_1

Definition at line 101 of file processor.h.

5.8.3.40 unsigned cpu_version::sse4_2

Definition at line 102 of file processor.h.

5.8.3.41 unsigned cpu_version::sss

Definition at line 82 of file processor.h.

5.8.3.42 unsigned cpu_version::ssse3

Definition at line 97 of file processor.h.

5.8.3.43 unsigned cpu_version::stepping

Definition at line 75 of file processor.h.

5.8.3.44 unsigned cpu_version::tm

Definition at line 129 of file processor.h.

5.8.3.45 unsigned cpu_version::tm2

Definition at line 96 of file processor.h.

5.8.3.46 unsigned cpu_version::type

Definition at line 78 of file processor.h.

5.8.3.47 unsigned cpu_version::vme

Definition at line 108 of file processor.h.

5.8.3.48 unsigned cpu_version::vmx

Definition at line 93 of file processor.h.

5.8.3.49 unsigned cpu_version::x2apic

Definition at line 103 of file processor.h.

The documentation for this struct was generated from the following file:

- include/arch/x86/processor.h

5.9 cpuid_regs Struct Reference

Data Fields

- `uint32_t eax`
- `uint32_t ebx`
- `uint32_t ecx`
- `uint32_t edx`

5.9.1 Detailed Description

Definition at line 5 of file cpuid.c.

5.9.2 Field Documentation

5.9.2.1 `uint32_t cpuid_regs::eax`

Definition at line 6 of file cpuid.c.

5.9.2.2 uint32_t cpuid_regs::ebx

Definition at line 7 of file cpuid.c.

5.9.2.3 uint32_t cpuid_regs::ecx

Definition at line 8 of file cpuid.c.

5.9.2.4 uint32_t cpuid_regs::edx

Definition at line 9 of file cpuid.c.

The documentation for this struct was generated from the following file:

- kernel/generic/[cpuid.c](#)

5.10 cpuid_t Struct Reference

```
#include <cpuid.h>
```

Data Fields

- [uint32_t eax](#)
- [uint32_t ebx](#)
- [uint32_t ecx](#)
- [uint32_t edx](#)

5.10.1 Detailed Description

Definition at line 18 of file cpuid.h.

5.10.2 Field Documentation

5.10.2.1 uint32_t cpuid_t::eax

Definition at line 20 of file cpuid.h.

5.10.2.2 uint32_t cpuid_t::ebx

Definition at line 21 of file cpuid.h.

5.10.2.3 uint32_t cpuid_t::ecx

Definition at line 22 of file cpuid.h.

5.10.2.4 uint32_t cpuid_t::edx

Definition at line 23 of file cpuid.h.

The documentation for this struct was generated from the following file:

- include/cpuid.h

5.11 direntry_t Struct Reference

```
#include <vfs.h>
```

Data Fields

- char name [128]
- uint32_t inode_n

5.11.1 Detailed Description

Definition at line 29 of file vfs.h.

5.11.2 Field Documentation

5.11.2.1 uint32_t direntry_t::inode_n

Definition at line 31 of file vfs.h.

5.11.2.2 char direntry_t::name[128]

Definition at line 30 of file vfs.h.

The documentation for this struct was generated from the following file:

- kernel/vfs/vfs.h

5.12 element Struct Reference

```
#include <list.h>
```

5.12.1 Detailed Description

Definition at line 4 of file list.h.

The documentation for this struct was generated from the following file:

- include/structs/list.h

5.13 elfhdr Struct Reference

Elf binary header values.

```
#include <elf.h>
```

Data Fields

- uint32_t magic
- uint8_t magic2 [MAGIC_LEN-4]
- uint16_t type
- uint16_t machine
- uint32_t version
- uint32_t entry
- uint32_t phroff
- uint32_t shroff
- uint32_t flags
- uint16_t ehsiz
- uint16_t phrsiz
- uint16_t phrnum
- uint16_t shrsize
- uint16_t shrnum
- uint16_t shrstrtbl
- uint8_t magic [MAGIC_LEN]

5.13.1 Detailed Description

Elf binary header values.

Definition at line 49 of file elf.h.

5.13.2 Field Documentation

5.13.2.1 uint16_t elfhdr::ehsiz

Elf header size

Definition at line 59 of file elf.h.

5.13.2.2 uint32_t elfhdr::entry

Address of entry point

Definition at line 55 of file elf.h.

5.13.2.3 uint32_t elfhdr::flags

Flags

Definition at line 58 of file elf.h.

5.13.2.4 uint16_t elfhdr::machine

Machine type

Definition at line 53 of file elf.h.

5.13.2.5 uint32_t elfhdr::magic

Definition at line 50 of file elf.h.

5.13.2.6 uint8_t elfhdr::magic[MAGIC_LEN]

the elf magic number

Definition at line 90 of file elf.h.

5.13.2.7 uint8_t elfhdr::magic2[MAGIC_LEN-4]

Definition at line 51 of file elf.h.

5.13.2.8 uint16_t elfhdr::phrnum

Number of program headers

Definition at line 61 of file elf.h.

5.13.2.9 uint32_t elfhdr::phroff

offset of program headers

Definition at line 56 of file elf.h.

5.13.2.10 uint16_t elfhdr::phrsize

Program header size

Definition at line 60 of file elf.h.

5.13.2.11 uint16_t elfhdr::shrnnum

Number of section headers

Definition at line 63 of file elf.h.

5.13.2.12 uint32_t elfhdr::shroff

offset of Section headers

Definition at line 57 of file elf.h.

5.13.2.13 uint16_t elfhdr::shrsizze

Section header size

Definition at line 62 of file elf.h.

5.13.2.14 uint16_t elfhdr::shrsttbl

Section header string table

Definition at line 64 of file elf.h.

5.13.2.15 uint16_t elfhdr::type

File type (EXEC/RELOC..etc)

Definition at line 52 of file elf.h.

5.13.2.16 uint32_t elfhdr::version

Version

Definition at line 54 of file elf.h.

The documentation for this struct was generated from the following files:

- include/arch/x86/[elf.h](#)
- arch/x86/include/[elf.h](#)

5.14 ext2_dir_entry Struct Reference

```
#include <ext2fs.h>
```

Data Fields

- `uint32_t inode`
- `uint16_t rec_len`
- `uint8_t name_len`
- `uint8_t type int8_t name [EXT2_NAME_LEN]`

5.14.1 Detailed Description

Definition at line 172 of file ext2fs.h.

5.14.2 Field Documentation

5.14.2.1 `uint32_t ext2_dir_entry::inode`

Definition at line 173 of file ext2fs.h.

5.14.2.2 `uint8_t type int8_t ext2_dir_entry::name[EXT2_NAME_LEN]`

Definition at line 180 of file ext2fs.h.

5.14.2.3 `uint8_t ext2_dir_entry::name_len`

Definition at line 175 of file ext2fs.h.

5.14.2.4 `uint16_t ext2_dir_entry::rec_len`

Definition at line 174 of file ext2fs.h.

The documentation for this struct was generated from the following file:

- include/fs/[ext2fs.h](#)

5.15 ext2_group_desc Struct Reference

```
#include <ext2fs.h>
```

Data Fields

- `uint32_t bg_block_bitmap`
- `uint32_t bg_inode_bitmap`
- `uint32_t bg_inode_table`
- `uint16_t bg_free_blocks_count`
- `uint16_t bg_free_inodes_count`
- `uint16_t bg_used_dirs_count`
- `uint16_t bg_pad`
- `uint32_t bg_reserved [3]`

5.15.1 Detailed Description

Definition at line 184 of file ext2fs.h.

5.15.2 Field Documentation

5.15.2.1 `uint32_t ext2_group_desc::bg_block_bitmap`

Definition at line 186 of file ext2fs.h.

5.15.2.2 `uint16_t ext2_group_desc::bg_free_blocks_count`

Definition at line 189 of file ext2fs.h.

5.15.2.3 `uint16_t ext2_group_desc::bg_free_inodes_count`

Definition at line 190 of file ext2fs.h.

5.15.2.4 `uint32_t ext2_group_desc::bg_inode_bitmap`

Definition at line 187 of file ext2fs.h.

5.15.2.5 `uint32_t ext2_group_desc::bg_inode_table`

Definition at line 188 of file ext2fs.h.

5.15.2.6 `uint16_t ext2_group_desc::bg_pad`

Definition at line 192 of file ext2fs.h.

5.15.2.7 uint32_t ext2_group_desc::bg_reserved[3]

Definition at line 193 of file ext2fs.h.

5.15.2.8 uint16_t ext2_group_desc::bg_used_dirs_count

Definition at line 191 of file ext2fs.h.

The documentation for this struct was generated from the following file:

- include/fs/[ext2fs.h](#)

5.16 ext2_inode Struct Reference

```
#include <ext2fs.h>
```

Data Fields

- [uint16_t i_mode](#)
- [uint16_t i_uid](#)
- [uint32_t i_size](#)
- [uint32_t i_atime](#)
- [uint32_t i_ctime](#)
- [uint32_t i_mtime](#)
- [uint32_t i_dtime](#)
- [uint16_t i_gid](#)
- [uint16_t i_links_count](#)
- [uint32_t i_blocks](#)
- [uint32_t i_flags](#)
- [uint32_t i_reserved1](#)
- [uint32_t i_block \[EXT2_N_BLOCKS\]](#)
- [uint32_t i_version](#)
- [uint32_t i_file_acl](#)
- [uint32_t i_dir_acl](#)
- [uint32_t i_faddr](#)
- union {
 - struct {
 - [uint8_t l_i_frag](#)
 - [uint8_t l_i_fsize](#)
 - [uint16_t l_i_reserved1](#)
 - [uint16_t l_i_uid_high](#)
 - [uint16_t l_i_gid_high](#)
 - [uint32_t l_i_reserved2](#)
- } [linux](#)
- struct {

```
    uint8_t h_i_frag  
    uint8_t h_i_fsize  
    uint16_t h_i_mode_high  
    uint16_t h_i_uid_high  
    uint16_t h_i_gid_high  
    uint32_t h_i_author  
} hurd2  
struct {  
    uint8_t m_i_frag  
    uint8_t m_i_fsize  
    uint16_t m_pad1  
    uint32_t m_i_reserved2 [2]  
} masix2  
} osd2
```

5.16.1 Detailed Description

Definition at line 197 of file ext2fs.h.

5.16.2 Field Documentation

5.16.2.1 uint32_t ext2_inode::h_i_author

Definition at line 231 of file ext2fs.h.

5.16.2.2 uint8_t ext2_inode::h_i_frag

Definition at line 226 of file ext2fs.h.

5.16.2.3 uint8_t ext2_inode::h_i_fsize

Definition at line 227 of file ext2fs.h.

5.16.2.4 uint16_t ext2_inode::h_i_gid_high

Definition at line 230 of file ext2fs.h.

5.16.2.5 uint16_t ext2_inode::h_i_mode_high

Definition at line 228 of file ext2fs.h.

5.16.2.6 `uint16_t ext2_inode::h_i_uid_high`

Definition at line 229 of file ext2fs.h.

5.16.2.7 `struct { ... } ext2_inode::hurd2`**5.16.2.8 `uint32_t ext2_inode::i_atime`**

Definition at line 201 of file ext2fs.h.

5.16.2.9 `uint32_t ext2_inode::i_block[EXT2_N_BLOCKS]`

Definition at line 210 of file ext2fs.h.

5.16.2.10 `uint32_t ext2_inode::i_blocks`

Definition at line 207 of file ext2fs.h.

5.16.2.11 `uint32_t ext2_inode::i_ctime`

Definition at line 202 of file ext2fs.h.

5.16.2.12 `uint32_t ext2_inode::i_dir_acl`

Definition at line 213 of file ext2fs.h.

5.16.2.13 `uint32_t ext2_inode::i_dtime`

Definition at line 204 of file ext2fs.h.

5.16.2.14 `uint32_t ext2_inode::i_faddr`

Definition at line 214 of file ext2fs.h.

5.16.2.15 `uint32_t ext2_inode::i_file_acl`

Definition at line 212 of file ext2fs.h.

5.16.2.16 `uint32_t ext2_inode::i_flags`

Definition at line 208 of file ext2fs.h.

5.16.2.17 `uint16_t ext2_inode::i_gid`

Definition at line 205 of file ext2fs.h.

5.16.2.18 `uint16_t ext2_inode::i_links_count`

Definition at line 206 of file ext2fs.h.

5.16.2.19 `uint16_t ext2_inode::i_mode`

Definition at line 198 of file ext2fs.h.

5.16.2.20 `uint32_t ext2_inode::i_mtime`

Definition at line 203 of file ext2fs.h.

5.16.2.21 `uint32_t ext2_inode::i_reserved1`

Definition at line 209 of file ext2fs.h.

5.16.2.22 `uint32_t ext2_inode::i_size`

Definition at line 200 of file ext2fs.h.

5.16.2.23 `uint16_t ext2_inode::i_uid`

Definition at line 199 of file ext2fs.h.

5.16.2.24 `uint32_t ext2_inode::i_version`

Definition at line 211 of file ext2fs.h.

5.16.2.25 `uint8_t ext2_inode::l_i_frag`

Definition at line 218 of file ext2fs.h.

5.16.2.26 `uint8_t ext2_inode::l_i_fsize`

Definition at line 219 of file ext2fs.h.

5.16.2.27 `uint16_t ext2_inode::l_i_gid_high`

Definition at line 222 of file ext2fs.h.

5.16.2.28 `uint16_t ext2_inode::l_i_reserved1`

Definition at line 220 of file ext2fs.h.

5.16.2.29 `uint32_t ext2_inode::l_i_reserved2`

Definition at line 223 of file ext2fs.h.

5.16.2.30 `uint16_t ext2_inode::l_i_uid_high`

Definition at line 221 of file ext2fs.h.

5.16.2.31 `struct { ... } ext2_inode::linux`

5.16.2.32 `uint8_t ext2_inode::m_i_frag`

Definition at line 234 of file ext2fs.h.

5.16.2.33 `uint8_t ext2_inode::m_i_fsize`

Definition at line 235 of file ext2fs.h.

5.16.2.34 `uint32_t ext2_inode::m_i_reserved2[2]`

Definition at line 237 of file ext2fs.h.

5.16.2.35 `uint16_t ext2_inode::m_pad1`

Definition at line 236 of file ext2fs.h.

5.16.2.36 `struct { ... } ext2_inode::masix2`

5.16.2.37 `union { ... } ext2_inode::osd2`

OS SPECIFIC VALUES from minix inode.h

The documentation for this struct was generated from the following file:

- include/fs/[ext2fs.h](#)

5.17 ext2_super_block Struct Reference

```
#include <ext2fs.h>
```

Data Fields

- `uint32_t s_inodes_count`
- `uint32_t s_blocks_count`
- `uint32_t s_r_blocks_count`
- `uint32_t s_free_blocks_count`
- `uint32_t s_free_inodes_count`
- `uint32_t s_first_data_block`
- `uint32_t s_log_block_size`
- `uint32_t s_log_frag_size`
- `uint32_t s_blocks_per_group`
- `uint32_t s_frags_per_group`
- `uint32_t s_inodes_per_group`
- `uint32_t s_mtime`
- `uint32_t s_wtime`
- `uint16_t s_mnt_count`
- `uint16_t s_max_mnt_count`
- `uint16_t s_magic`
- `uint16_t s_state`
- `uint16_t s_errors`
- `uint16_t s_min_pad`
- `uint32_t s_lastcheck`
- `uint32_t s_checkinterval`
- `uint32_t s_os_id`
- `uint32_t s_maj_pad`
- `uint16_t s_uid`
- `uint16_t s_gid`
- `uint32_t s_first_ino`
- `uint16_t s_inode_size`
- `uint16_t s_block_group_nr`
- `uint32_t s_feature_compat`
- `uint32_t s_feature_incompat`
- `uint32_t s_feature_ro_compat`
- `uint8_t s_uuid [16]`
- `char s_volume_name [16]`
- `char s_last_mounted [64]`
- `uint32_t s_algorithm_usage_bitmap`
- `uint8_t s_prealloc_blocks`
- `uint8_t s_prealloc_dir_blocks`
- `uint16_t s_padding1`
- `uint8_t s_journal_uuid [16]`
- `uint32_t s_journal_inum`

- `uint32_t s_journal_dev`
- `uint32_t s_last_orphan`
- `uint32_t s_reserved [197]`

5.17.1 Detailed Description

Definition at line 122 of file ext2fs.h.

5.17.2 Field Documentation

5.17.2.1 `uint32_t ext2_super_block::s_algorithm_usage_bitmap`

Definition at line 159 of file ext2fs.h.

5.17.2.2 `uint16_t ext2_super_block::s_block_group_nr`

Definition at line 152 of file ext2fs.h.

5.17.2.3 `uint32_t ext2_super_block::s_blocks_count`

Definition at line 124 of file ext2fs.h.

5.17.2.4 `uint32_t ext2_super_block::s_blocks_per_group`

Definition at line 131 of file ext2fs.h.

5.17.2.5 `uint32_t ext2_super_block::s_checkinterval`

Definition at line 143 of file ext2fs.h.

5.17.2.6 `uint16_t ext2_super_block::s_errors`

Definition at line 140 of file ext2fs.h.

5.17.2.7 `uint32_t ext2_super_block::s_feature_compat`

Definition at line 153 of file ext2fs.h.

5.17.2.8 `uint32_t ext2_super_block::s_feature_incompat`

Definition at line 154 of file ext2fs.h.

5.17.2.9 `uint32_t ext2_super_block::s_feature_ro_compat`

Definition at line 155 of file ext2fs.h.

5.17.2.10 `uint32_t ext2_super_block::s_first_data_block`

Definition at line 128 of file ext2fs.h.

5.17.2.11 `uint32_t ext2_super_block::s_first_ino`

Definition at line 150 of file ext2fs.h.

5.17.2.12 `uint32_t ext2_super_block::s_frags_per_group`

Definition at line 132 of file ext2fs.h.

5.17.2.13 `uint32_t ext2_super_block::s_free_blocks_count`

Definition at line 126 of file ext2fs.h.

5.17.2.14 `uint32_t ext2_super_block::s_free_inodes_count`

Definition at line 127 of file ext2fs.h.

5.17.2.15 `uint16_t ext2_super_block::s_gid`

Definition at line 147 of file ext2fs.h.

5.17.2.16 `uint16_t ext2_super_block::s_inode_size`

Definition at line 151 of file ext2fs.h.

5.17.2.17 `uint32_t ext2_super_block::s_inodes_count`

Definition at line 123 of file ext2fs.h.

5.17.2.18 `uint32_t ext2_super_block::s_inodes_per_group`

Definition at line 133 of file ext2fs.h.

5.17.2.19 `uint32_t ext2_super_block::s_journal_dev`

Definition at line 165 of file ext2fs.h.

5.17.2.20 `uint32_t ext2_super_block::s_journal_inum`

Definition at line 164 of file ext2fs.h.

5.17.2.21 `uint8_t ext2_super_block::s_journal_uuid[16]`

Definition at line 163 of file ext2fs.h.

5.17.2.22 `char ext2_super_block::s_last_mounted[64]`

Definition at line 158 of file ext2fs.h.

5.17.2.23 `uint32_t ext2_super_block::s_last_orphan`

Definition at line 166 of file ext2fs.h.

5.17.2.24 `uint32_t ext2_super_block::s_lastcheck`

Definition at line 142 of file ext2fs.h.

5.17.2.25 `uint32_t ext2_super_block::s_log_block_size`

Definition at line 129 of file ext2fs.h.

5.17.2.26 `uint32_t ext2_super_block::s_log_frag_size`

Definition at line 130 of file ext2fs.h.

5.17.2.27 `uint16_t ext2_super_block::s_magic`

Definition at line 138 of file ext2fs.h.

5.17.2.28 `uint32_t ext2_super_block::s_maj_pad`

Definition at line 145 of file ext2fs.h.

5.17.2.29 `uint16_t ext2_super_block::s_max_mnt_count`

Definition at line 137 of file ext2fs.h.

5.17.2.30 `uint16_t ext2_super_block::s_min_pad`

Definition at line 141 of file ext2fs.h.

5.17.2.31 `uint16_t ext2_super_block::s_mnt_count`

Definition at line 136 of file ext2fs.h.

5.17.2.32 `uint32_t ext2_super_block::s_mtime`

Definition at line 134 of file ext2fs.h.

5.17.2.33 `uint32_t ext2_super_block::s_os_id`

Definition at line 144 of file ext2fs.h.

5.17.2.34 `uint16_t ext2_super_block::s_padding1`

Definition at line 162 of file ext2fs.h.

5.17.2.35 `uint8_t ext2_super_block::s_preadloc_blocks`

Definition at line 160 of file ext2fs.h.

5.17.2.36 `uint8_t ext2_super_block::s_preadloc_dir_blocks`

Definition at line 161 of file ext2fs.h.

5.17.2.37 `uint32_t ext2_super_block::s_r_blocks_count`

Definition at line 125 of file ext2fs.h.

5.17.2.38 `uint32_t ext2_super_block::s_reserved[197]`

Definition at line 167 of file ext2fs.h.

5.17.2.39 uint16_t ext2_super_block::s_state

Definition at line 139 of file ext2fs.h.

5.17.2.40 uint16_t ext2_super_block::s_uid

Definition at line 146 of file ext2fs.h.

5.17.2.41 uint8_t ext2_super_block::s_uuid[16]

Definition at line 156 of file ext2fs.h.

5.17.2.42 char ext2_super_block::s_volume_name[16]

Definition at line 157 of file ext2fs.h.

5.17.2.43 uint32_t ext2_super_block::s_wtime

Definition at line 135 of file ext2fs.h.

The documentation for this struct was generated from the following file:

- include/fs/[ext2fs.h](#)

5.18 Gdtdesc Struct Reference

```
#include <memvals.h>
```

Data Fields

- [uint16_t size](#)
- [uint32_t base](#)

5.18.1 Detailed Description

Definition at line 46 of file memvals.h.

5.18.2 Field Documentation

5.18.2.1 uint32_t Gdtdesc::base

Definition at line 48 of file memvals.h.

5.18.2.2 uint16_t Gdtdesc::size

Definition at line 47 of file memvals.h.

The documentation for this struct was generated from the following file:

- include/memvals.h

5.19 gpr_regs_t Struct Reference

General purpose registers.

```
#include <cpu_state.h>
```

Data Fields

- reg_t edi
- reg_t esi
- reg_t ebp
- reg_t esp
- reg_t ebx
- reg_t edx
- reg_t ecx
- reg_t eax

5.19.1 Detailed Description

General purpose registers.

gpr registers structure contains general purpose register. ordered to be filled by a single PUSHAD/PUSHAL

Definition at line 74 of file cpu_state.h.

5.19.2 Field Documentation

5.19.2.1 reg_t gpr_regs_t::eax

Definition at line 83 of file cpu_state.h.

5.19.2.2 reg_t gpr_regs_t::ebp

Definition at line 78 of file cpu_state.h.

5.19.2.3 reg_t gpr_regs_t::ebx

Definition at line 80 of file cpu_state.h.

5.19.2.4 reg_t gpr_regs_t::ecx

Definition at line 82 of file cpu_state.h.

5.19.2.5 reg_t gpr_regs_t::edi

Definition at line 76 of file cpu_state.h.

5.19.2.6 reg_t gpr_regs_t::edx

Definition at line 81 of file cpu_state.h.

5.19.2.7 reg_t gpr_regs_t::esi

Definition at line 77 of file cpu_state.h.

5.19.2.8 reg_t gpr_regs_t::esp

Definition at line 79 of file cpu_state.h.

The documentation for this struct was generated from the following file:

- include/arch/x86/cpu_state.h

5.20 htable_node_t Struct Reference

```
#include <htable.h>
```

Data Fields

- uint32_t key
- uint32_t value

5.20.1 Detailed Description

Definition at line 24 of file htable.h.

5.20.2 Field Documentation

5.20.2.1 uint32_t htable_node_t::key

Definition at line 25 of file htable.h.

5.20.2.2 uint32_t htable_node_t::value

Definition at line 26 of file htable.h.

The documentation for this struct was generated from the following file:

- include/structs/htable.h

5.21 htable_t Struct Reference

a hash table structure

```
#include <htable.h>
```

Data Fields

- void ** table
- uint32_t buckets
- uint32_t * bucket_size
- uint32_t(* hash)(uint32_t)
- uint32_t(* destroy)(uint32_t)
- uint32_t size
- uint32_t bucket_length

5.21.1 Detailed Description

a hash table structure

Hash table holds a pointer-to-pointer which is considered the actual table, It also holds the bucket_size array which tracks the size of buckets the buckets which represents the number of buckets, a hash function passed on initialization time, same as destroy function. the overall size which is used to determine the overall inserted values count.

Definition at line 29 of file htable.h.

5.21.2 Field Documentation

5.21.2.1 uint32_t htable_t::bucket_length

Definition at line 36 of file htable.h.

5.21.2.2 uint32_t* htable_t::bucket_size

Definition at line 32 of file htable.h.

5.21.2.3 uint32_t htable_t::buckets

Definition at line 31 of file htable.h.

5.21.2.4 uint32_t(* htable_t::destroy)(uint32_t)

Definition at line 34 of file htable.h.

5.21.2.5 uint32_t(* htable_t::hash)(uint32_t)

Definition at line 33 of file htable.h.

5.21.2.6 uint32_t htable_t::size

Definition at line 35 of file htable.h.

5.21.2.7 void htable_t::table**

Definition at line 30 of file htable.h.

The documentation for this struct was generated from the following file:

- include/structs/[htable.h](#)

5.22 initrd_file_header_t Struct Reference

```
#include <initrd.h>
```

Data Fields

- [uint8_t magic](#)
- [char name \[64\]](#)
- [uint32_t offset](#)
- [uint32_t size](#)

5.22.1 Detailed Description

Definition at line 25 of file initrd.h.

5.22.2 Field Documentation

5.22.2.1 uint8_t initrd_file_header_t::magic

Definition at line 27 of file initrd.h.

5.22.2.2 char initrd_file_header_t::name[64]

Definition at line 28 of file initrd.h.

5.22.2.3 uint32_t initrd_file_header_t::offset

Definition at line 29 of file initrd.h.

5.22.2.4 uint32_t initrd_file_header_t::size

Definition at line 30 of file initrd.h.

The documentation for this struct was generated from the following file:

- kernel/vfs/[initrd.h](#)

5.23 initrd_header_t Struct Reference

```
#include <initrd.h>
```

Data Fields

- [uint32_t nfiles](#)

5.23.1 Detailed Description

Definition at line 20 of file initrd.h.

5.23.2 Field Documentation

5.23.2.1 uint32_t initrd_header_t::nfiles

Definition at line 22 of file initrd.h.

The documentation for this struct was generated from the following file:

- kernel/vfs/[initrd.h](#)

5.24 kcommand Struct Reference

```
#include <kconsole.h>
```

Data Fields

- const char * **name**
- const char * **hint**
- void(* **operation**)(void)

5.24.1 Detailed Description

Definition at line 17 of file kconsole.h.

5.24.2 Field Documentation

5.24.2.1 const char* kcommand::hint

Definition at line 19 of file kconsole.h.

5.24.2.2 const char* kcommand::name

Definition at line 18 of file kconsole.h.

5.24.2.3 void(* kcommand::operation)(void)

Definition at line 20 of file kconsole.h.

The documentation for this struct was generated from the following file:

- include/kconsole.h

5.25 Page Struct Reference

a page list entry, used to hold info about pages.

```
#include <page.h>
```

Data Fields

- page_entry_t **link**
- uint16_t **ref**

5.25.1 Detailed Description

a page list entry, used to hold info about pages.

Definition at line 199 of file page.h.

5.25.2 Field Documentation

5.25.2.1 `page_entry_t Page::link`

Definition at line 201 of file page.h.

5.25.2.2 `uint16_t Page::ref`

Definition at line 202 of file page.h.

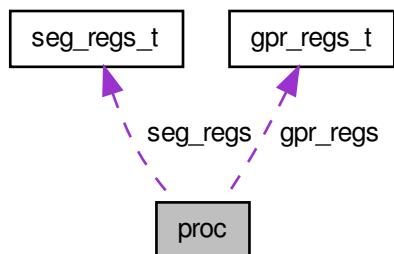
The documentation for this struct was generated from the following file:

- [include/arch/x86/mm/page.h](#)

5.26 proc Struct Reference

```
#include <proc.h>
```

Collaboration diagram for proc:



Public Member Functions

- [LIST_ENTRY \(proc\) link](#)
- [LIFO_ENTRY \(proc\) q_link](#)
- [LIST_ENTRY \(proc\) wait_link](#)

Data Fields

- `gpr_regs_t gpr_regs`
- `seg_regs_t seg_regs`
- `reg_t eip`
- `uint32_t cs`
- `reg_t eflags`
- `reg_t esp`
- `uint32_t ss`
- `uint32_t id`
- `uint32_t status`
- `pde_t * page_directory`
- `uint32_t cr3`
- `uint32_t preempted`
- `uint32_t dequeqed`
- `uint32_t timer`

5.26.1 Detailed Description

Definition at line 23 of file proc.h.

5.26.2 Member Function Documentation

5.26.2.1 `proc::LIFO_ENTRY(proc)`

5.26.2.2 `proc::LIST_ENTRY(proc)`

5.26.2.3 `proc::LIST_ENTRY(proc)`

5.26.3 Field Documentation

5.26.3.1 `uint32_t proc::cr3`

Definition at line 35 of file proc.h.

5.26.3.2 `uint32_t proc::cs`

Definition at line 27 of file proc.h.

5.26.3.3 `uint32_t proc::dequeqed`

Definition at line 37 of file proc.h.

5.26.3.4 reg_t proc::eflags

Definition at line 28 of file proc.h.

5.26.3.5 reg_t proc::eip

Definition at line 26 of file proc.h.

5.26.3.6 reg_t proc::esp

Definition at line 29 of file proc.h.

5.26.3.7 gpr_regs_t proc::gpr_regs

Definition at line 24 of file proc.h.

5.26.3.8 uint32_t proc::id

Definition at line 31 of file proc.h.

5.26.3.9 pde_t* proc::page_directory

Definition at line 34 of file proc.h.

5.26.3.10 uint32_t proc::preempted

Definition at line 36 of file proc.h.

5.26.3.11 seg_regs_t proc::seg_regs

Definition at line 25 of file proc.h.

5.26.3.12 uint32_t proc::ss

Definition at line 30 of file proc.h.

5.26.3.13 uint32_t proc::status

Definition at line 32 of file proc.h.

5.26.3.14 `uint32_t proc::timer`

Definition at line 43 of file proc.h.

The documentation for this struct was generated from the following file:

- `include/proc/proc.h`

5.27 proghdr Struct Reference

An ELF program header structure.

```
#include <elf.h>
```

Data Fields

- `uint32_t type`
- `uint32_t offset`
- `uint32_t vaddr`
- `uint32_t paddr`
- `uint32_t filesz`
- `uint32_t memsz`
- `uint32_t flags`
- `uint32_t align`

5.27.1 Detailed Description

An ELF program header structure.

Definition at line 71 of file elf.h.

5.27.2 Field Documentation

5.27.2.1 `uint32_t proghdr::align`

Definition at line 79 of file elf.h.

5.27.2.2 `uint32_t proghdr::filesz`

Definition at line 76 of file elf.h.

5.27.2.3 `uint32_t proghdr::flags`

Definition at line 78 of file elf.h.

5.27.2.4 uint32_t proghdr::memsz

Definition at line 77 of file elf.h.

5.27.2.5 uint32_t proghdr::offset

Definition at line 73 of file elf.h.

5.27.2.6 uint32_t proghdr::paddr

Definition at line 75 of file elf.h.

5.27.2.7 uint32_t proghdr::type

Definition at line 72 of file elf.h.

5.27.2.8 uint32_t proghdr::vaddr

Definition at line 74 of file elf.h.

The documentation for this struct was generated from the following files:

- include/arch/x86/elf.h
- arch/x86/include/elf.h

5.28 sechdr Struct Reference

An ELF section header structure.

```
#include <elf.h>
```

Data Fields

- uint32_t name
- uint32_t type
- uint32_t flags
- uint32_t addr
- uint32_t offset
- uint32_t size
- uint32_t link
- uint32_t info
- uint32_t addralign
- uint32_t entsize

5.28.1 Detailed Description

An ELF section header structure.

Definition at line 82 of file elf.h.

5.28.2 Field Documentation

5.28.2.1 `uint32_t sechdr::addr`

Definition at line 86 of file elf.h.

5.28.2.2 `uint32_t sechdr::addralign`

Definition at line 91 of file elf.h.

5.28.2.3 `uint32_t sechdr::entsize`

Definition at line 92 of file elf.h.

5.28.2.4 `uint32_t sechdr::flags`

Definition at line 85 of file elf.h.

5.28.2.5 `uint32_t sechdr::info`

Definition at line 90 of file elf.h.

5.28.2.6 `uint32_t sechdr::link`

Definition at line 89 of file elf.h.

5.28.2.7 `uint32_t sechdr::name`

Definition at line 83 of file elf.h.

5.28.2.8 `uint32_t sechdr::offset`

Definition at line 87 of file elf.h.

5.28.2.9 `uint32_t sechdr::size`

Definition at line 88 of file elf.h.

5.28.2.10 uint32_t sechdr::type

Definition at line 84 of file elf.h.

The documentation for this struct was generated from the following files:

- include/arch/x86/elf.h
- arch/x86/include/elf.h

5.29 seg_regs_t Struct Reference

Segment registers.

```
#include <cpu_state.h>
```

Data Fields

- reg_t gs
- reg_t fs
- reg_t es
- reg_t ds

5.29.1 Detailed Description

Segment registers.

the structure contains segment registers except for SP and CS since they are stored by the hardware into the structure.

Definition at line 93 of file cpu_state.h.

5.29.2 Field Documentation

5.29.2.1 reg_t seg_regs_t::ds

Definition at line 97 of file cpu_state.h.

5.29.2.2 reg_t seg_regs_t::es

Definition at line 96 of file cpu_state.h.

5.29.2.3 reg_t seg_regs_t::fs

Definition at line 95 of file cpu_state.h.

5.29.2.4 `reg_t seg_regs_t::gs`

Definition at line 94 of file `cpu_state.h`.

The documentation for this struct was generated from the following file:

- `include/arch/x86/cpu_state.h`

5.30 Segdesc Struct Reference

```
#include <memvals.h>
```

Data Fields

- `unsigned limit_0: 16`
- `unsigned base_0: 16`
- `unsigned base_1: 8`
- `unsigned permission: 8`
- `unsigned limit_1: 4`
- `unsigned flags: 4`
- `unsigned base: 8`

5.30.1 Detailed Description

Definition at line 34 of file `memvals.h`.

5.30.2 Field Documentation

5.30.2.1 `unsigned Segdesc::base`

Definition at line 41 of file `memvals.h`.

5.30.2.2 `unsigned Segdesc::base_0`

Definition at line 36 of file `memvals.h`.

5.30.2.3 `unsigned Segdesc::base_1`

Definition at line 37 of file `memvals.h`.

5.30.2.4 `unsigned Segdesc::flags`

Definition at line 40 of file `memvals.h`.

5.30.2.5 unsigned Segdesc::limit_0

Definition at line 35 of file memvals.h.

5.30.2.6 unsigned Segdesc::limit_1

Definition at line 39 of file memvals.h.

5.30.2.7 unsigned Segdesc::permission

Definition at line 38 of file memvals.h.

The documentation for this struct was generated from the following file:

- include/[memvals.h](#)

5.31 semaphore_t Struct Reference

```
#include <semaphore.h>
```

Data Fields

- [uint32_t count](#)
- struct Proc_List [wait_list](#)

5.31.1 Detailed Description

Definition at line 14 of file semaphore.h.

5.31.2 Field Documentation

5.31.2.1 uint32_t semaphore_t::count

Definition at line 15 of file semaphore.h.

5.31.2.2 struct Proc_List semaphore_t::wait_list

Definition at line 16 of file semaphore.h.

The documentation for this struct was generated from the following file:

- include/synchronization/[semaphore.h](#)

5.32 spinlock Struct Reference

```
#include <spinlock.h>
```

Data Fields

- [uint32_t lock](#)

5.32.1 Detailed Description

Definition at line 4 of file spinlock.h.

5.32.2 Field Documentation

5.32.2.1 [uint32_t spinlock::lock](#)

Definition at line 5 of file spinlock.h.

The documentation for this struct was generated from the following file:

- include/synchronization/[spinlock.h](#)

5.33 spinlock_t Struct Reference

```
#include <spinlock.h>
```

Data Fields

- [atomic_t lock](#)

5.33.1 Detailed Description

Definition at line 11 of file spinlock.h.

5.33.2 Field Documentation

5.33.2.1 [atomic_t spinlock_t::lock](#)

Definition at line 12 of file spinlock.h.

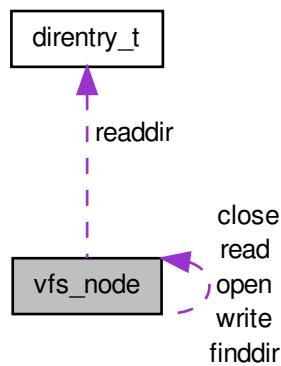
The documentation for this struct was generated from the following file:

- include/arch/x86/[spinlock.h](#)

5.34 vfs_node Struct Reference

```
#include <vfs.h>
```

Collaboration diagram for vfs_node:



Data Fields

- `char name [128]`
- `uint32_t inode`
- `uint32_t mask`
- `uint32_t uid`
- `uint32_t gid`
- `uint32_t flags`
- `uint32_t size`
- `uint32_t impl`
- `read_type_t read`
- `write_type_t write`
- `open_type_t open`
- `close_type_t close`
- `readdir_type_t readdir`
- `finddir_type_t finddir`
- `struct fs_node * ptr`

5.34.1 Detailed Description

Definition at line 37 of file `vfs.h`.

5.34.2 Field Documentation

5.34.2.1 `close_type_t vfs_node::close`

Definition at line 50 of file vfs.h.

5.34.2.2 `finddir_type_t vfs_node::finddir`

Definition at line 52 of file vfs.h.

5.34.2.3 `uint32_t vfs_node::flags`

Definition at line 44 of file vfs.h.

5.34.2.4 `uint32_t vfs_node::gid`

Definition at line 43 of file vfs.h.

5.34.2.5 `uint32_t vfs_node::impl`

Definition at line 46 of file vfs.h.

5.34.2.6 `uint32_t vfs_node::inode`

Definition at line 40 of file vfs.h.

5.34.2.7 `uint32_t vfs_node::mask`

Definition at line 41 of file vfs.h.

5.34.2.8 `char vfs_node::name[128]`

Definition at line 39 of file vfs.h.

5.34.2.9 `open_type_t vfs_node::open`

Definition at line 49 of file vfs.h.

5.34.2.10 `struct fs_node* vfs_node::ptr`

Definition at line 53 of file vfs.h.

5.34.2.11 read_type_t vfs_node::read

Definition at line 47 of file vfs.h.

5.34.2.12 readdir_type_t vfs_node::readdir

Definition at line 51 of file vfs.h.

5.34.2.13 uint32_t vfs_node::size

Definition at line 45 of file vfs.h.

5.34.2.14 uint32_t vfs_node::uid

Definition at line 42 of file vfs.h.

5.34.2.15 write_type_t vfs_node::write

Definition at line 48 of file vfs.h.

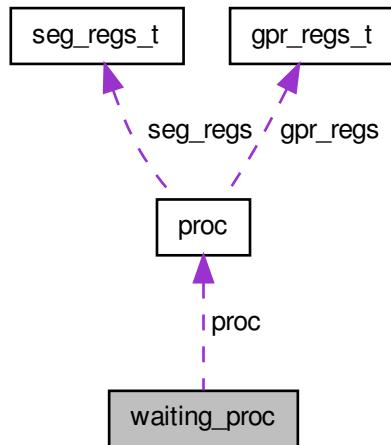
The documentation for this struct was generated from the following file:

- kernel/vfs/vfs.h

5.35 waiting_proc Struct Reference

```
#include <wait_queue.h>
```

Collaboration diagram for waiting_proc:



Public Member Functions

- `LIST_ENTRY (waiting_proc) next`

Data Fields

- `proc_t * proc`

5.35.1 Detailed Description

Definition at line 13 of file wait_queue.h.

5.35.2 Member Function Documentation

5.35.2.1 `waiting_proc::LIST_ENTRY(waiting_proc)`

5.35.3 Field Documentation

5.35.3.1 `proc_t* waiting_proc::proc`

Definition at line 15 of file wait_queue.h.

The documentation for this struct was generated from the following file:

- include/synchronization/[wait_queue.h](#)

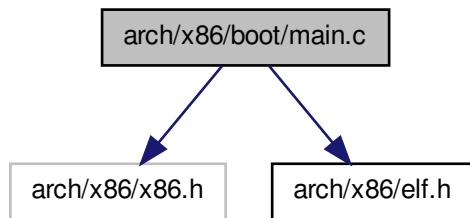
Chapter 6

File Documentation

6.1 arch/x86/boot/main.c File Reference

Elf binary structures and constants, for parsing binaries.

```
#include <arch/x86/x86.h> #include <arch/x86/elf.h> Include  
dependency graph for main.c:
```



Bootloader-KernelLoading

- `#define SECTOR 512`
equals the sector size in bytes
- `#define ELFHDR ((struct elfhdr *) 0x10000)`
The kernel physical address to load image into.
- `void readsect (void *, uint32_t)`
reads disk at a given offset
- `void readseg (uint32_t, uint32_t, uint32_t)`

- loads data from disk into memory with data size to read.*
- void [cmain](#) (void)
reads and validates kernel image and loads it and fetch it into IP.
 - void [waitdisk](#) (void)
hangs till disk operation is done.

6.1.1 Detailed Description

Elf binary structures and constants, for parsing binaries.

Author

Saad Talaat

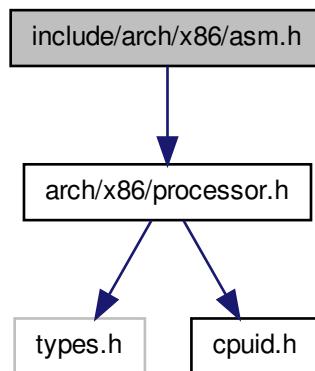
Date

26/10/2011 Loads the kernel from image. includes an elf parsing and loading for loading the kernel from disk, which is done by reading the disk at offset 512 = 1 sector, and loading it into memory using readsect and readseg respectively. after loading the kernel image into memory, the LOAD program header is loaded into cs and jumps to kernel entry point. Here, we use the **Fixed Disk Controller** device which's ports are from 0x1F0 ~ 0x1F7

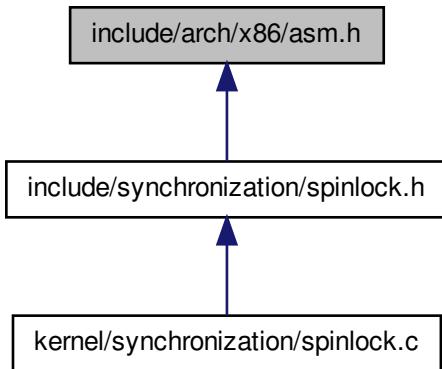
Definition in file [main.c](#).

6.2 include/arch/x86/asm.h File Reference

```
#include <arch/x86/processor.h> Include dependency graph for asm.h:
```



This graph shows which files directly or indirectly include this file:



Functions

- static void `arch_spinlock_lock (uint32_t *lock)`
- static void `arch_spinlock_unlock (uint32_t *lock)`

6.2.1 Function Documentation

6.2.1.1 static void `arch_spinlock_lock (uint32_t * lock)` [inline, static]

TODO 1- Disable scheduling 2- make atomic lock as in doc

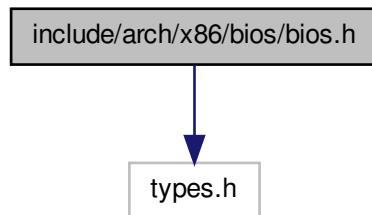
Definition at line 4 of file `asm.h`.

6.2.1.2 static void `arch_spinlock_unlock (uint32_t * lock)` [inline, static]

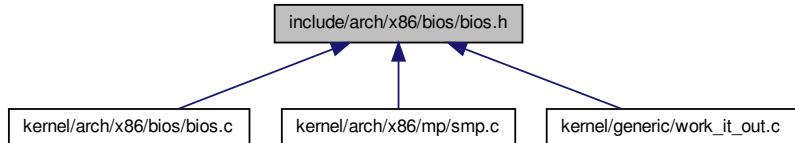
Definition at line 21 of file `asm.h`.

6.3 include/arch/x86/bios/bios.h File Reference

#include <types.h> Include dependency graph for bios.h:



This graph shows which files directly or indirectly include this file:



Functions

- void [bios_init](#) (void)

Variables

- [uintptr_t ebda](#)

6.3.1 Function Documentation

6.3.1.1 void bios_init(void)

Definition at line 5 of file bios.c.

6.3.2 Variable Documentation

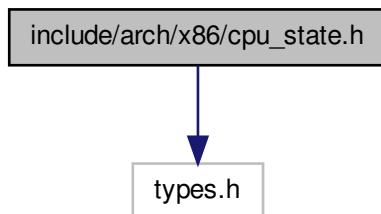
6.3.2.1 uintptr_t ebda

Definition at line 4 of file bios.c.

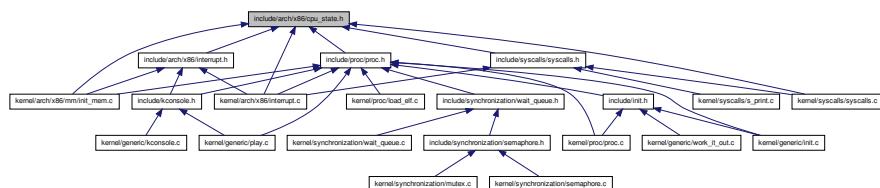
6.4 include/arch/x86/cpu_state.h File Reference

structures and constants used in storing and determining cpu state.

```
#include <types.h> // Include dependency graph for cpu_state.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [cpu_state_t](#)
Interaupt/Trap cpu state.
- struct [gpr_regs_t](#)
General purpose registers.
- struct [seg_regs_t](#)

- *Segment registers.*
- struct [__attribute__](#)
Task State segment.
- struct [__attribute__](#)
Task State segment.

Defines

- #define [TSS_SIZE](#) 104
- #define [TSS_IOMAP_SIZE](#) ((0x8 * 0x400) +1)
- #define [TASK_BUSY](#) 0xB
- #define [TASK_INACTIVE](#) 0x9
- #define [TSS_AVL](#) 1
- #define [TSS_DPL_KERNEL](#) 0
- #define [TSS_DPL_USER](#) 60
- #define [TSS_PRESENT](#) 0x80
- #define [TSS_GRANULARITY](#) 0x8
- #define [SEGMENT_TSS](#)(*limit*, *base*, *type*, *flags*)

6.4.1 Detailed Description

structures and constants used in storing and determining cpu state.

Date

28 Sept, 2012

Author

Saad Talaat

Definition in file [cpu_state.h](#).

6.4.2 Define Documentation

6.4.2.1 #define SEGMENT_TSS(*limit*, *base*, *type*, *flags*)

Value:

```
(struct Segdesc)\n    {\n        \\\n        (((limit)>>12) & 0xffff),           \\\n        ((base) & 0xffff),                   \\\n        (((base) >> 16) & 0xff),           \\\n        (type),\n        ((limit) >> 28),                 \\\n        ((flags)),\n        (((base) >> 24) & 0xff) \\\n    }
```

Definition at line 170 of file [cpu_state.h](#).

6.4.2.2 #define TASK_BUSY 0xB

Definition at line 162 of file cpu_state.h.

6.4.2.3 #define TASK_INACTIVE 0x9

Definition at line 163 of file cpu_state.h.

6.4.2.4 #define TSS_AVL 1

Definition at line 164 of file cpu_state.h.

6.4.2.5 #define TSS_DPL_KERNEL 0

Definition at line 165 of file cpu_state.h.

6.4.2.6 #define TSS_DPL_USER 60

Definition at line 166 of file cpu_state.h.

6.4.2.7 #define TSS_GRANULARITY 0x8

Definition at line 168 of file cpu_state.h.

6.4.2.8 #define TSS_IOMAP_SIZE ((0x8 * 0x400) +1)

Definition at line 161 of file cpu_state.h.

6.4.2.9 #define TSS_PRESENT 0x80

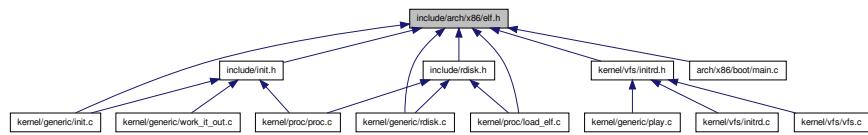
Definition at line 167 of file cpu_state.h.

6.4.2.10 #define TSS_SIZE 104

Definition at line 160 of file cpu_state.h.

6.5 include/arch/x86/elf.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct `elfhdr`
Elf binary header values.
- struct `proghdr`
An ELF program header structure.
- struct `sechdr`
An ELF section header structure.

Defines

- #define `ELF_MAGIC` 0x464C457F
- #define `MAGIC_LEN` 16
- #define `M_CLASS_OFF` 4
- #define `M_CLASSNONE` 0
- #define `M_CLASS32` 1
- #define `M_CLASS64` 2
- #define `M_CLASSNUM` 3
- #define `M_DATA_OFF` 5
- #define `M_DATANONE` 0
- #define `M_DATA2LE` 1
- #define `M_DATA2BE` 2
- #define `M_DATANUM` 3
- #define `M_VERSION` 6
- #define `M_OSABI` 7
- #define `M_OSABI_SYSV` 0
- #define `M_OSABI_HPUX` 1
- #define `M_ABIVERSION` 8
- #define `M_ELF_PADDING` 9
- #define `T_TYPE_NONE` 0
- #define `T_TYPE_REL` 1
- #define `T_TYPE_EXEC` 2
- #define `T_TYPE_DYN` 3

- #define **T_TYPE_CORE** 4
- #define **T_TYPE_LOPROC** 0xff00
- #define **T_TYPE_HIPROC** 0xffff
- #define **M_MACHINE_I386** 3
- #define **V_VERSION_NONE** 0
- #define **V_VERSION_CURRENT** 1
- #define **V_VERSION_NUM** 2
- #define **P_PROGHDR_R** 0x4
- #define **P_PROGHDR_W** 0x2
- #define **P_PROGHDR_E** 0x1
- #define **PRG_LOAD** 1

Typedefs

- typedef struct **elfhdr** elfhdr
- typedef struct **proghdr** prohdr
- typedef struct **sechdr** secthdr

6.5.1 Define Documentation

6.5.1.1 #define **ELF_MAGIC** 0x464C457F

Definition at line 9 of file elf.h.

6.5.1.2 #define **M_ABIVERSION** 8

Definition at line 30 of file elf.h.

6.5.1.3 #define **M_CLASS32** 1

Definition at line 14 of file elf.h.

6.5.1.4 #define **M_CLASS64** 2

Definition at line 15 of file elf.h.

6.5.1.5 #define **M_CLASS_OFF** 4

Definition at line 12 of file elf.h.

6.5.1.6 #define **M_CLASSNONE** 0

Definition at line 13 of file elf.h.

6.5.1.7 #define M_CLASSNUM 3

Definition at line 16 of file elf.h.

6.5.1.8 #define M_DATA2BE 2

Definition at line 21 of file elf.h.

6.5.1.9 #define M_DATA2LE 1

Definition at line 20 of file elf.h.

6.5.1.10 #define M_DATA_OFF 5

Definition at line 18 of file elf.h.

6.5.1.11 #define M_DATANONE 0

Definition at line 19 of file elf.h.

6.5.1.12 #define M_DATANUM 3

Definition at line 22 of file elf.h.

6.5.1.13 #define M_ELF_PADDING 9

Definition at line 31 of file elf.h.

6.5.1.14 #define M_MACHINE_I386 3

Definition at line 43 of file elf.h.

6.5.1.15 #define M_OSABI 7

Definition at line 26 of file elf.h.

6.5.1.16 #define M_OSABI_HPUX 1

Definition at line 28 of file elf.h.

6.5.1.17 #define M_OSABI_SYSV 0

Definition at line 27 of file elf.h.

6.5.1.18 #define M_VERSION 6

Definition at line 24 of file elf.h.

6.5.1.19 #define MAGIC_LEN 16

Definition at line 10 of file elf.h.

6.5.1.20 #define P_PROGHDR_E 0x1

Definition at line 70 of file elf.h.

6.5.1.21 #define P_PROGHDR_R 0x4

Definition at line 68 of file elf.h.

6.5.1.22 #define P_PROGHDR_W 0x2

Definition at line 69 of file elf.h.

6.5.1.23 #define PRG_LOAD 1

Definition at line 81 of file elf.h.

6.5.1.24 #define T_TYPE_CORE 4

Definition at line 38 of file elf.h.

6.5.1.25 #define T_TYPE_DYN 3

Definition at line 37 of file elf.h.

6.5.1.26 #define T_TYPE_EXEC 2

Definition at line 36 of file elf.h.

6.5.1.27 #define T_TYPE_HIPROC 0xffff

Definition at line 40 of file elf.h.

6.5.1.28 #define T_TYPE_LOPROC 0xff00

Definition at line 39 of file elf.h.

6.5.1.29 #define T_TYPE_NONE 0

Definition at line 34 of file elf.h.

6.5.1.30 #define T_TYPE_REL 1

Definition at line 35 of file elf.h.

6.5.1.31 #define V_VERSION_CURRENT 1

Definition at line 47 of file elf.h.

6.5.1.32 #define V_VERSION_NONE 0

Definition at line 46 of file elf.h.

6.5.1.33 #define V_VERSION_NUM 2

Definition at line 48 of file elf.h.

6.5.2 Typedef Documentation

6.5.2.1 typedef struct elfhdr elfhdr

6.5.2.2 typedef struct proghdr prohdr

6.5.2.3 typedef struct sechdr secthdr

6.6 arch/x86/include/elf.h File Reference

Data Structures

- struct [elfhdr](#)

Elf binary header values.

- struct `proghdr`
An ELF program header structure.
- struct `sechdr`
An ELF section header structure.

Defines

Elf-Member-Constants

- #define `ELF_MAGIC` 0x464C457F
- #define `MAGIC_LEN` 16
- #define `M_CLASS_OFF` 4
- #define `M_CLASSNONE` 0
- #define `M_CLASS32` 1
- #define `M_CLASS64` 2
- #define `M_CLASSNUM` 3
- #define `M_DATA_OFF` 5
- #define `M_DATANONE` 0
- #define `M_DATA2LE` 1
- #define `M_DATA2BE` 2
- #define `M_DATANUM` 3
- #define `M_VERSION` 6
- #define `M_OSABI` 7
- #define `M_OSABI_SYSV` 0
- #define `M_OSABI_HPUX` 1
- #define `M_ABIVERSION` 8
- #define `M_ELF_PADDING` 9
- #define `T_TYPE_NONE` 0
- #define `T_TYPE_REL` 1
- #define `T_TYPE_EXEC` 2
- #define `T_TYPE_DYN` 3
- #define `T_TYPE_CORE` 4
- #define `T_TYPE_LOPROC` 0xff00
- #define `T_TYPE_HIPROC` 0xffff
- #define `M_MACHINE_I386` 3
- #define `V_VERSION_NONE` 0
- #define `V_VERSION_CURRENT` 1
- #define `V_VERSION_NUM` 2
- #define `P_PROGHDR_R` 0x4
- #define `P_PROGHDR_W` 0x2
- #define `P_PROGHDR_E` 0x1

Typedefs

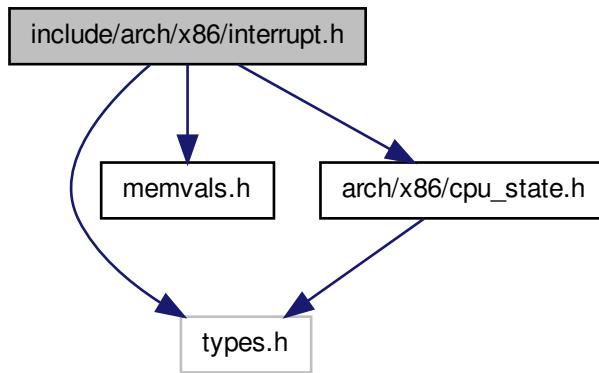
Elf-structures

- typedef struct `elfhdr` `elfhdr`
- typedef struct `proghdr` `prohdr`
- typedef struct `sechdr` `sechdr`

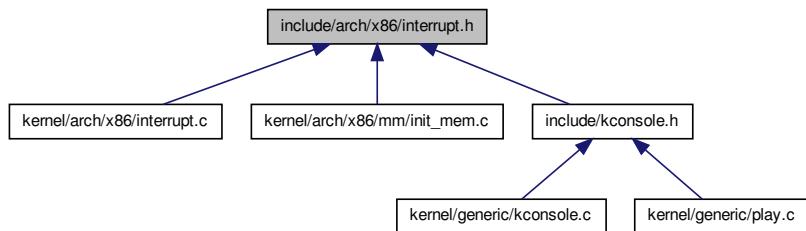
6.7 include/arch/x86/interrupt.h File Reference

Interrupt structures and constants.

```
#include <types.h> #include <memvals.h> #include <arch/x86/cpu-
_state.h> Include dependency graph for interrupt.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [_attribute_](#)

Task State segment.

Defines

- #define _CATERNEL_X86_INTERRUPT_H_
- #define GATE_TASK 0x5
- #define GATE_INT16 0x6
- #define GATE_TRAP16 0x7
- #define GATE_INT32 0xE
- #define GATE_TRAP32 0xF
- #define IDT_ENTRIES 64
- #define GATE_FILL(gate, type, dpl, sel, offset)
- #define VECTOR_INDEX(cnt) vector_##cnt
- #define GATE_OFFSET(gate, vector)
- #define GP 13
- #define PF 14
- #define SCHED 0x20
- #define SYSCALL 0x30
- #define PF_VIOLATION 0x1
- #define PF_NOT_PRESENT ~PF_VIOLATION
- #define PF_ON_WRITE 0x2
- #define PF_ON_READ ~PF_ON_WRITE
- #define PF_FROM_USER 0x4
- #define PF_FROM_KERNEL ~PF_FROM_USER
- #define PF_IFETCH 0x10
- #define PF_RSRVD 0x8

Typedefs

- typedef struct Gdtdesc Idtdesc

Functions

- void idt_init (void)

Initializing Interrupt Descriptor table.
- void map_exception (uint32_t, cpu_state_t *)

maps an exception/interrupt to its handler
- uint32_t page_fault_handler (cpu_state_t *)

Page Fault Handler.
- void interrupt_printinfo (void)

Variables

- gatedesc idt []

Declare IDT with 255 width interrupts.
- char * x86_exception_names []

Fault names.

6.7.1 Detailed Description

Interrupt structures and constants.

Author

Saad Talaat.

Date

27 Sept, 2012

Definition in file [interrupt.h](#).

6.7.2 Define Documentation

6.7.2.1 #define _CATERNEL_X86_INTERRUPT_H_

Definition at line 11 of file interrupt.h.

6.7.2.2 #define GATE_FILL(gate, type, dpl, sel, offset)

Value:

```
\n
(gate)->offset_0_15 = (uint16_t)      ((offset) & 0xffff);      \
(gate)->segment_s = (uint16_t) (sel);          \
(gate)->args      = 0;                  \
(gate)->reserved   = 0;                  \
(gate)->type       = 0xe;                \
(gate)->dpl        = 0;                  \
(gate)->p          = 1;                  \
(gate)->offset_16_31 = (uint16_t) ((offset) >> 16);
```

Definition at line 70 of file interrupt.h.

6.7.2.3 #define GATE_INT16 0x6

Definition at line 59 of file interrupt.h.

6.7.2.4 #define GATE_INT32 0xE

Definition at line 61 of file interrupt.h.

6.7.2.5 #define GATE_OFFSET(gate, vector)

Value:

```
(gate).offset_0_15 = (uint32_t) (((uint32_t) vector) & 0xffff); \
(gate).offset_16_31= (uint32_t) (((uint32_t) vector) >> 16);
```

Definition at line 88 of file interrupt.h.

6.7.2.6 #define GATE_TASK 0x5

Definition at line 58 of file interrupt.h.

6.7.2.7 #define GATE_TRAP16 0x7

Definition at line 60 of file interrupt.h.

6.7.2.8 #define GATE_TRAP32 0xF

Definition at line 62 of file interrupt.h.

6.7.2.9 #define GP 13

Definition at line 93 of file interrupt.h.

6.7.2.10 #define IDT_ENTRIES 64

Definition at line 64 of file interrupt.h.

6.7.2.11 #define PF 14

Definition at line 94 of file interrupt.h.

6.7.2.12 #define PF_FROM_KERNEL ~PF_FROM_USER

Definition at line 103 of file interrupt.h.

6.7.2.13 #define PF_FROM_USER 0x4

Definition at line 102 of file interrupt.h.

6.7.2.14 #define PF_IFETCH 0x10

Definition at line 104 of file interrupt.h.

6.7.2.15 #define PF_NOT_PRESENT ~PF_VIOLATION

Definition at line 99 of file interrupt.h.

6.7.2.16 #define PF_ON_READ ~PF_ON_WRITE

Definition at line 101 of file interrupt.h.

6.7.2.17 #define PF_ON_WRITE 0x2

Definition at line 100 of file interrupt.h.

6.7.2.18 #define PF_RSRVD 0x8

Definition at line 105 of file interrupt.h.

6.7.2.19 #define PF_VIOLATION 0x1

Definition at line 98 of file interrupt.h.

6.7.2.20 #define SCHED 0x20

Definition at line 95 of file interrupt.h.

6.7.2.21 #define SYSCALL 0x30

Definition at line 96 of file interrupt.h.

6.7.2.22 #define VECTOR_INDEX(*cnt*) vector_##cnt

Definition at line 86 of file interrupt.h.

6.7.3 Typedef Documentation

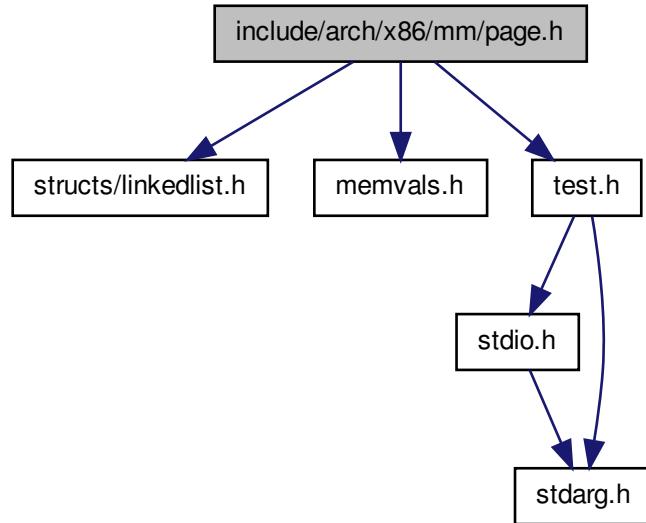
6.7.3.1 **typedef struct Gdtdesc Idtdesc**

Definition at line 49 of file interrupt.h.

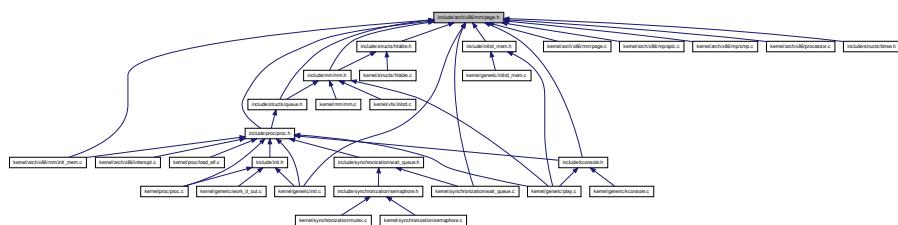
6.8 include/arch/x86/mm/page.h File Reference

Intel Paging.

```
#include <structs/linkedlist.h>      #include <memvals.h> ×
#include <test.h> Include dependency graph for page.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [__attribute__](#)
Task State segment.
- struct [Page](#)
a page list entry, used to hold info about pages.

Defines

- #define PAGE_PRESENT 0x1
- #define PAGE_WRITABLE 0x2
- #define PAGE_USER 0x4
- #define PAGE_WTHROUGH 0x8
- #define PAGE_CDISABLE 0x10
- #define PAGE_ACCESSED 0x20
- #define PAGE_DIRTY 0x40
- #define PAGE_DIRTY 0x40
- #define PAGE_GLOBAL 0x100
- #define PAGE_PAGESZ 0x80
- #define PGTBL_SHIFT 12
- #define PGDIR_SHIFT 22
- #define PGSHIFT 12
- #define PTSIIFT 22
- #define PGDIRX(lin) ((((uint32_t)(lin)) >> PGDIR_SHIFT) & 0x3FF)
- #define VIRT_PGDIRX(lin) PGDIRX((lin))
- #define PGTBLX(lin) ((((uint32_t)(lin)) >> PGTBL_SHIFT) & 0x3FF)
- #define VIRT_PGTBLX(lin) PGTBLX((lin))
- #define PGPN(lin) (((uint32_t)(lin)) >> PGTBL_SHIFT)
- #define VIRT_PGPN(lin) PGPN((lin))
- #define PGOFF(lin) (((lin) & 0xFFF))
- #define PTD_ADDR(lin) (((uint32_t) (lin)) & ~0xFFFF)
- #define PA2KA(pa)
- #define KA2PA(va)

TypeDefs

- typedef uint32_t pde_t

Functions

- static paddr_t va2pa (pde_t *pgdir, void *va)

searches a page directory for a virtual address and returns referring page.
- LIST_HEAD (PageList, Page)
- typedef LIST_ENTRY (Page) page_entry_t
- static uint32_t pagetoppn (struct Page *p)

Get Page number.
- static void * patopage (paddr_t p)

get a page referring to a paddr
- static paddr_t pagetopa (struct Page *p)

get a paddr a page referring to
- static void * pagetova (struct Page *p)
- void x86_paging_init (void)

- `Initiate global pages.`
- `void x86_page_init (struct Page *)`
Initiate a page entry.
- `int x86_page_alloc (struct Page **)`
Allocate a page page is removed from the free pages list and initiated.
- `void x86_page_free (struct Page *)`
given page is freed.
- `void x86_page_detach (struct Page *)`
when [un]refering a page, ref member is decremented.
- `pte_t * x86_pgdir_find (pde_t *, const void *, int)`
find a page, if doesn't exist allocate it
- `struct Page * x86_page_lookup (pde_t *, void *, pte_t **)`
looks up a page refers to vaddr.
- `void x86_page_remove (pde_t *, void *)`
removes a page that refers to a given Virtual address from a page directory.
- `int x86_page_insert (pde_t *, struct Page *, void *, uint32_t)`
inserts a page that refers to a given va, into a given page directory.
- `void map_segment_page (pde_t *, vaddr_t, size_t, paddr_t, int)`
maps a segment to a virtual address on a specific page directory.
- `void x86_test_pgdir (void)`
- `void x86_test.paging (void)`
- `static void x86_test_page_alloc (void)`
- `void * allocate (uint32_t, uint32_t)`
a boot time allocation function, allocate heaps.
- `void memory_printinfo (void)`
Reads memory size and determine the pages count needed to map it.

Variables

- `size_t page_count`
- `volatile pte_t virtptg[]`
- `volatile pde_t virtpgd[]`
- `char kernel_stack[]`
- `char kernel_stack_end[]`
- `struct Page * pages`
- `uint32_t global_cr3`
- `pde_t * global_pgdir`
- `struct Segdesc gdt[]`
- `char * next_free`

6.8.1 Detailed Description

Intel Paging.

Date

Wednesday 19/09/2012

Author

Saad Talaat

Definition in file [page.h](#).

6.8.2 Define Documentation

6.8.2.1 #define KA2PA(va)

Value:

```
({
    vaddr_t m_va = (vaddr_t) (va); \
    if( m_va < KERNEL_ADDR){ \
        panic("KA2PA called with bad address \n"); \
    } \
    m_va = KERNEL_ADDR; \
})
```

Definition at line 100 of file page.h.

6.8.2.2 #define PA2KA(pa)

Value:

```
({
    paddr_t m_pa = ((paddr_t) pa); \
    paddr_t m_ppn = PGPN(m_pa); \
    if( m_ppn >= page_count){ \
        panic("PA2KA called with invalid paddr\n"); \
    } \
    (void *) (m_pa + KERNEL_ADDR); \
})
```

Definition at line 90 of file page.h.

6.8.2.3 #define PAGE_ACSESSED 0x20

Definition at line 57 of file page.h.

6.8.2.4 #define PAGE_CDISABLE 0x10

Definition at line 56 of file page.h.

6.8.2.5 #define PAGE_DIRTY 0x40

Definition at line 61 of file page.h.

6.8.2.6 #define PAGE_DIRTY 0x40

Definition at line 61 of file page.h.

6.8.2.7 #define PAGE_GLOBAL 0x100

Definition at line 62 of file page.h.

6.8.2.8 #define PAGE_PAGESZ 0x80

Definition at line 65 of file page.h.

6.8.2.9 #define PAGE_PRESENT 0x1

Definition at line 52 of file page.h.

6.8.2.10 #define PAGE_USER 0x4

Definition at line 54 of file page.h.

6.8.2.11 #define PAGE_WRITABLE 0x2

Definition at line 53 of file page.h.

6.8.2.12 #define PAGE_WTHROUGH 0x8

Definition at line 55 of file page.h.

6.8.2.13 #define PGDIR_SHIFT 22

Definition at line 69 of file page.h.

6.8.2.14 #define PGDIRX(*lin*) ((((uint32_t)(lin)) >> PGDIR_SHIFT) & 0x3FF)

Definition at line 74 of file page.h.

6.8.2.15 #define PGOFF(*lin*) (((lin) & 0xFF))

Definition at line 83 of file page.h.

6.8.2.16 #define PGPN(*lin*) (((uint32_t)(lin)) >> PGTBL_SHIFT)

Definition at line 80 of file page.h.

6.8.2.17 #define PGSHIFT 12

Definition at line 70 of file page.h.

6.8.2.18 #define PGTBL_SHIFT 12

Definition at line 68 of file page.h.

6.8.2.19 #define PGTBLX(*lin*) ((((uint32_t)(lin)) >> PGTBL_SHIFT) & 0x3FF)

Definition at line 77 of file page.h.

6.8.2.20 #define PTD_ADDR(*lin*) (((uint32_t)(lin)) & ~0xFFFF)

Definition at line 84 of file page.h.

6.8.2.21 #define PTSHIFT 22

Definition at line 71 of file page.h.

6.8.2.22 #define VIRT_PGDIX(*lin*) PGDIRX((lin))

Definition at line 75 of file page.h.

6.8.2.23 #define VIRT_PGPN(*lin*) PGPN((lin))

Definition at line 81 of file page.h.

6.8.2.24 #define VIRT_PGTBLX(*lin*) PGTBLX((*lin*))

Definition at line 78 of file page.h.

6.8.3 Typedef Documentation

6.8.3.1 **typedef uint32_t pde_t**

Definition at line 30 of file page.h.

6.8.4 Function Documentation

6.8.4.1 **typedef LIST_ENTRY(Page)**

6.8.4.2 **LIST_HEAD(PageList , Page)**

6.8.4.3 **static paddr_t pagetopa(struct Page *p) [inline, static]**

get a paddr a page refering to

Definition at line 236 of file page.h.

6.8.4.4 **static uint32_t pagetoppn(struct Page *p) [inline, static]**

Get [Page](#) number.

Definition at line 214 of file page.h.

6.8.4.5 **static void* pagetova(struct Page *p) [inline, static]**

Definition at line 241 of file page.h.

6.8.4.6 **static void* patopage(paddr_t p) [inline, static]**

get a page refering to a paddr

Definition at line 223 of file page.h.

6.8.4.7 **static paddr_t va2pa(pde_t *pgdir, void *va) [inline, static]**

searches a page directory for a virtual address and returns refering page.

Definition at line 131 of file page.h.

6.8.4.8 static void x86_test_page_alloc(void) [static]

6.8.4.9 void x86_test.paging(void)

6.8.5 Variable Documentation

6.8.5.1 struct Segdesc gdt[]

6.8.5.2 char kernel_stack[]

6.8.5.3 char kernel_stack_end[]

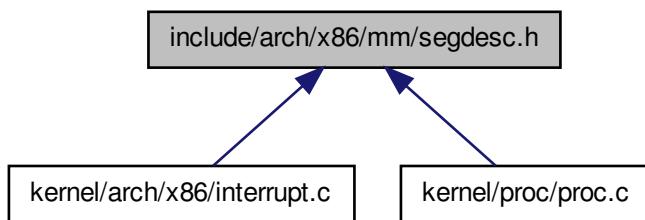
6.8.5.4 volatile pde_t virtpgd[]

6.8.5.5 volatile pte_t virtpgt[]

6.9 include/arch/x86/mm/segdesc.h File Reference

Segment offset constants.

This graph shows which files directly or indirectly include this file:



Defines

- #define SEG_KERNCODE 0x8
- #define SEG_KERNDATA 0x10
- #define SEG_USERCODE 0x18
- #define SEG_USERDATA 0x20
- #define SEG_TSS 0x28

6.9.1 Detailed Description

Segment offset constants.

Date

Tuesday 18/09/2012

Author

Saad Talaat

Definition in file [segdesc.h](#).

6.9.2 Define Documentation

6.9.2.1 #define SEG_KERNCODE 0x8

Definition at line 14 of file segdesc.h.

6.9.2.2 #define SEG_KERNDATA 0x10

Definition at line 15 of file segdesc.h.

6.9.2.3 #define SEG_TSS 0x28

Definition at line 18 of file segdesc.h.

6.9.2.4 #define SEG_USERCODE 0x18

Definition at line 16 of file segdesc.h.

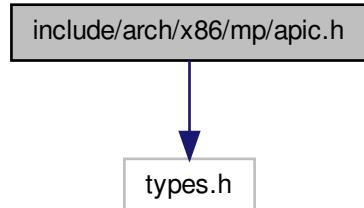
6.9.2.5 #define SEG_USERDATA 0x20

Definition at line 17 of file segdesc.h.

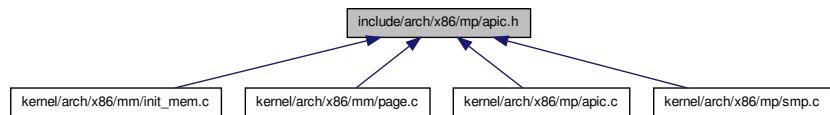
6.10 include/arch/x86/mp/apic.h File Reference

Support for xAPIC for MPs.

#include <types.h> Include dependency graph for apic.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [__attribute__](#)

Task State segment.

Defines

- #define IA32_APIC_BASE 0x1B
- #define IA32_APIC_BASE_BSP 0x8
- #define IA32_APIC_BASE_ENABLE 0xB
- #define IA32_APIC_BASE_ADDR 0xC
- #define LAID 0x8 /* RW - Local APIC ID */
- #define LAVR 0x12 /* R - Local APIC Version */
- #define TPR 0x20 /* RW - Task Priority Register */
- #define APR 0x24 /* R - Arbitration Priority Register */
- #define PPR 0x28 /* R - Processor Priority Register */
- #define EOI 0x2C /* W - EOI Register */
- #define RRD 0x30 /* R - Remote Read Register */

- #define **LDR** 0x34 /* RW - Logical destination register */
- #define **DFR** 0x38 /* RW - Destination format register */
- #define **SVR** 0x3C /* RW - Spurious interrupts */
- #define **ERR** 0xA0 /* R - Error status register */
- #define **ICR_LOW** 0xC0 /* RW - ICR 0~31 */
- #define **ICR_HIGH** 0xC4 /* RW - ICR 32~63 */
- #define **ICT** 0xE0 /* RW - Initial Count for timer */
- #define **CCT** 0xE4 /* R - Current count for timer */
- #define **DCT** 0xF8 /* RW - Divide configuration for timer */
- #define **LVT_CMCI** 0xBC
- #define **LVT_TIMER** 0xC8 /* RW - APIC Timer interrupt register */
- #define **LVT_TMR** 0xCC /* RW - APIC Thermal monitor register */
- #define **LVT_PCR** 0xD0 /* RW - APIC performance counter register */
- #define **LVT_LINT0** 0xD4 /* RW - APIC LINT0 */
- #define **LVT_LINT1** 0xD8 /* RW - APIC LINT1 */
- #define **LVT_ERR** 0xDC /* RW - APIC Error register */
- #define **ICR_FIXED** 0x0
- #define **ICR_LO_PIR** 0X1 /*lowest priority*/
- #define **ICR_SMI** 0x2 /*SMI Interrupt*/
- #define **ICR_NMI** 0x4 /*NMI interrupt*/
- #define **ICR_INIT** 0x5 /*INIT REQ*/
- #define **ICR_SIPI** 0x6 /*Startup*/
- #define **PHYSICAL** 0x0
- #define **LOGICAL** 0x1
- #define **ICR_IDLE** 0x0
- #define **ICR_PENDING** 0x1
- #define **LEVEL_DEASSERT** 0x0
- #define **LEVEL_ASSERT** 0x1
- #define **TRIGMOD_EDGE** 0x0
- #define **TRIGMOD_LEVEL** 0x1
- #define **NO_SH** 0x0 /*no short hand*/
- #define **SELF** 0x1 /*send to self*/
- #define **ALL_W_SELF** 0x2 /*all with self*/
- #define **ALL_X_SELF** 0x3 /*all excluding self*/
- #define **LAVR_Mask** 0xffU
- #define **is_82489DX_apic**(x) (((x) & **LAVR_Mask** & 0xf0U) == 0x0U))

Functions

- void **soft_lapic_enable** (void)

Soft To enable the Local APIC to receive interrupts you also have to set bit 8 in the Spurious Interrupt Vector Register.
- void **msr_lapic_enable** (void)

APIC setting IA32_APIC_BASE Model Specific Register (MSR)
- void **apic_init_ipi** (uint8_t lapicid)

Sending INIT IPI to processor speicified with lapicid.

- void [apic_s_ipi](#) (icr_t icr, uint8_t lapicid)

Sending Startup IPI to processor specified with lapicid.
- void [apic_read_errors](#) (void)

Reads error field in lapic.

Variables

- [uint32_t * lapic](#)
- [uint8_t vector](#)
- [unsigned int delivery_mode](#)
- [unsigned int dest_mode](#)
- [unsigned int delivery_stat](#)
- [unsigned int](#)
- [unsigned int level](#)
- [unsigned int trigger_mode](#)
- [unsigned int shorthand](#)
- [uint8_t dest](#)

6.10.1 Detailed Description

Support for xAPIC for MPs.

Author

Saad Talaat , Menna Essa

Date

18 November, 2012

Definition in file [apic.h](#).

6.10.2 Define Documentation

6.10.2.1 #define ALL_W_SELF 0x2 /*all with self*/

Definition at line 69 of file apic.h.

6.10.2.2 #define ALL_X_SELF 0x3 /*all excluding self*/

Definition at line 70 of file apic.h.

6.10.2.3 #define APR 0x24 /* R - Arbitration Priority Register */

Definition at line 24 of file apic.h.

6.10.2.4 #define CCT 0xE4 /* R - Current count for timer */

Definition at line 35 of file apic.h.

6.10.2.5 #define DCT 0xF8 /* RW - Divide configuration for timer */

Definition at line 36 of file apic.h.

6.10.2.6 #define DFR 0x38 /* RW - Destination format register */

Definition at line 29 of file apic.h.

6.10.2.7 #define EOI 0x2C /* W - EOI Register */

Definition at line 26 of file apic.h.

6.10.2.8 #define ERR 0xA0 /* R - Error status register */

Definition at line 31 of file apic.h.

6.10.2.9 #define IA32_APIC_BASE 0x1B

Definition at line 15 of file apic.h.

6.10.2.10 #define IA32_APIC_BASE_ADDR 0xC

Definition at line 18 of file apic.h.

6.10.2.11 #define IA32_APIC_BASE_BSP 0x8

Definition at line 16 of file apic.h.

6.10.2.12 #define IA32_APIC_BASE_ENABLE 0xB

Definition at line 17 of file apic.h.

6.10.2.13 #define ICR_FIXED 0x0

Definition at line 48 of file apic.h.

6.10.2.14 #define ICR_HIGH 0xC4 /* RW - ICR 32~63 */

Definition at line 33 of file apic.h.

6.10.2.15 #define ICR_IDLE 0x0

Definition at line 58 of file apic.h.

6.10.2.16 #define ICR_INIT 0x5 /*INIT REQ*/

Definition at line 52 of file apic.h.

6.10.2.17 #define ICR_LO_PIR 0X1 /*lowest priority*/

Definition at line 49 of file apic.h.

6.10.2.18 #define ICR_LOW 0xC0 /* RW - ICR 0~31 */

Definition at line 32 of file apic.h.

6.10.2.19 #define ICR_NMI 0x4 /*NMI interrupt*/

Definition at line 51 of file apic.h.

6.10.2.20 #define ICR_PENDING 0x1

Definition at line 59 of file apic.h.

6.10.2.21 #define ICR_SIPI 0x6 /*Startup*/

Definition at line 53 of file apic.h.

6.10.2.22 #define ICR_SMI 0x2 /*SMI Interrupt*/

Definition at line 50 of file apic.h.

6.10.2.23 #define ICT 0xE0 /* RW - Initial Count for timer */

Definition at line 34 of file apic.h.

6.10.2.24 #define **is_82489DX_apic(x)** (((x) & **LAVR_Mask** & 0xf0U) == 0x0U))

Definition at line 73 of file apic.h.

6.10.2.25 #define **LAIID** 0x8 /* RW - Local APIC ID */

Definition at line 21 of file apic.h.

6.10.2.26 #define **LAVR** 0x12 /* R - Local APIC Version */

Definition at line 22 of file apic.h.

6.10.2.27 #define **LAVR_Mask** 0xffU

Definition at line 72 of file apic.h.

6.10.2.28 #define **LDR** 0x34 /* RW - Logical destination register */

Definition at line 28 of file apic.h.

6.10.2.29 #define **LEVEL_ASSERT** 0x1

Definition at line 62 of file apic.h.

6.10.2.30 #define **LEVEL_DEASSERT** 0x0

Levels.

Definition at line 61 of file apic.h.

6.10.2.31 #define **LOGICAL** 0x1

Definition at line 56 of file apic.h.

6.10.2.32 #define **LVT_CMCI** 0xBC

Definition at line 39 of file apic.h.

6.10.2.33 #define **LVT_ERR** 0xDC /* RW - APIC Error register */

Definition at line 45 of file apic.h.

6.10.2.34 #define LVT_LINT0 0xD4 /* RW - APIC LINT0 */

Definition at line 43 of file apic.h.

6.10.2.35 #define LVT_LINT1 0xD8 /* RW - APIC LINT1 */

Definition at line 44 of file apic.h.

6.10.2.36 #define LVT_PCR 0xD0 /* RW - APIC performance counter register */

Definition at line 42 of file apic.h.

6.10.2.37 #define LVT_TIMER 0xC8 /* RW - APIC Timer interrupt register */

Definition at line 40 of file apic.h.

6.10.2.38 #define LVT_TMR 0xCC /* RW - APIC Thermal monitor register */

Definition at line 41 of file apic.h.

6.10.2.39 #define NO_SH 0x0 /*no short hand*/

Definition at line 67 of file apic.h.

6.10.2.40 #define PHYSICAL 0x0

Definition at line 55 of file apic.h.

6.10.2.41 #define PPR 0x28 /* R - Processor Priority Register */

Definition at line 25 of file apic.h.

6.10.2.42 #define RRD 0x30 /* R - Remote Read Register */

Definition at line 27 of file apic.h.

6.10.2.43 #define SELF 0x1 /*send to self*/

Definition at line 68 of file apic.h.

6.10.2.44 #define **SVR** 0x3C /* RW - Spurious interrupts */

Definition at line 30 of file apic.h.

6.10.2.45 #define **TPR** 0x20 /* RW - Task Priority Register */

Definition at line 23 of file apic.h.

6.10.2.46 #define **TRIGMOD_EDGE** 0x0

Trigger Modes.

Definition at line 64 of file apic.h.

6.10.2.47 #define **TRIGMOD_LEVEL** 0x1

Definition at line 65 of file apic.h.

6.10.3 Variable Documentation

6.10.3.1 **unsigned int delivery_mode**

Definition at line 92 of file apic.h.

6.10.3.2 **unsigned int delivery_stat**

Definition at line 94 of file apic.h.

6.10.3.3 **uint8_t dest**

Definition at line 99 of file apic.h.

6.10.3.4 **unsigned int dest_mode**

Definition at line 93 of file apic.h.

6.10.3.5 **unsigned int**

Definition at line 95 of file apic.h.

6.10.3.6 **unsigned int level**

Definition at line 96 of file apic.h.

6.10.3.7 unsigned int shorthand

Definition at line 99 of file apic.h.

6.10.3.8 unsigned int trigger_mode

Definition at line 97 of file apic.h.

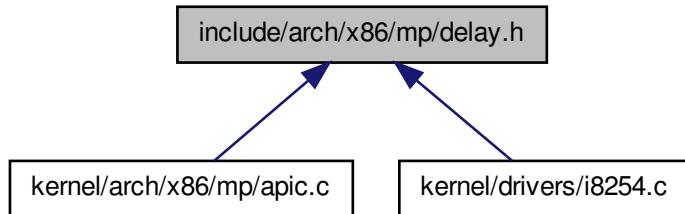
6.10.3.9 uint8_t vector

Definition at line 91 of file apic.h.

6.11 include/arch/x86/mp/delay.h File Reference

Simple delay functions headers.

This graph shows which files directly or indirectly include this file:



Functions

- void [delay \(uint32_t microsec\)](#)
 uninterruptable delay function
- void [asm_delay_loop \(uint32_t t\)](#)
 useless loop to use while delaying
- void [asm_fake_loop \(uint32_t t\)](#)
 useless loop to calibrate cpu

6.11.1 Detailed Description

Simple delay functions headers.

Author

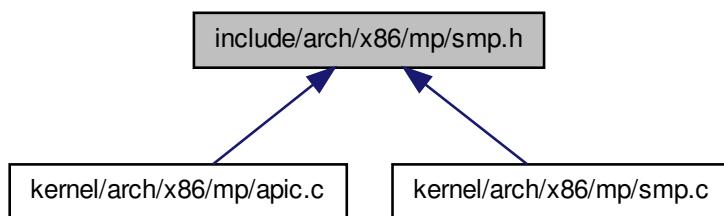
Menna Essa

Definition in file [delay.h](#).

6.12 include/arch/x86/mp/smp.h File Reference

Multiprocessor support header.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct [attribute](#)
Task State segment.

Defines

- #define SMPBOOT_START 0xBD000
- #define TRAMPOLINE_START PA2KA(SMPBOOT_START)
- #define NCPUS 4
- #define FS_SIGNATURE 0x5f504d5f
- #define CT_SIGNATURE 0x504d4350
- #define PROCESSOR 0
- #define BUS 1
- #define IO_APIC 2
- #define IO_INTRPT_ASS 3
- #define LOC_INTRPT_ASS 4
- #define INVALID 000000000000
- #define INTEL486DX 010000000001
- #define INTEL486SX 010000100000
- #define INT 0
- #define NMI 1
- #define SMI 2
- #define EXTINT 3

Functions

- uint8_t fps_check (uint8_t *base)
checks if floating point structure is valid , the sum of all elements must be 0
- uint8_t ct_check (void)
checks if configuration is valid , the sum of all elements must be 0
- void fsp_print (fpstruct_t *fs)
parses the floating point structure
- void ct_read_hdr (void)
parses the configuration table structure
- void ct_entries (void)
parses the config table and sets the entries pointers
- void find_set_fps (void)
*Searches and Sets the global Floating pointer structure. 1. search first 1K of EBDA
2. if EBDA is undefined, search last 1K of base memory 3. search 64K starting at 0xf0000.*

Variables

- typedef __attribute__
• fpstruct_t * fs
floating point structure
- ct_hdr * ct
configuration table.
- uint32_t processors_count

6.12.1 Detailed Description

Multiprocessor support header.

Author

Saad Talaat , Menna Essa

Date

19 November, 2012

Definition in file [smp.h](#).

6.12.2 Define Documentation

6.12.2.1 #define BUS 1

Definition at line 19 of file smp.h.

6.12.2.2 #define CT_SIGNATURE 0x504d4350

Definition at line 16 of file smp.h.

6.12.2.3 #define EXTINT 3

Definition at line 36 of file smp.h.

6.12.2.4 #define FS_SIGNATURE 0x5f504d5f

Definition at line 15 of file smp.h.

6.12.2.5 #define INT 0

Definition at line 33 of file smp.h.

6.12.2.6 #define INTEL486DX 010000000001

Definition at line 25 of file smp.h.

6.12.2.7 #define INTEL486SX 010000100000

Definition at line 26 of file smp.h.

6.12.2.8 #define INVALID 000000000000

Definition at line 24 of file smp.h.

6.12.2.9 #define IO_APIC 2

Definition at line 20 of file smp.h.

6.12.2.10 #define IO_INTRPT_ASS 3

Definition at line 21 of file smp.h.

6.12.2.11 #define LOC_INTRPT_ASS 4

Definition at line 22 of file smp.h.

6.12.2.12 #define NCPUS 4

Definition at line 14 of file smp.h.

6.12.2.13 #define NMI 1

Definition at line 34 of file smp.h.

6.12.2.14 #define PROCESSOR 0

Definition at line 18 of file smp.h.

6.12.2.15 #define SMI 2

Definition at line 35 of file smp.h.

6.12.2.16 #define SMPBOOT_START 0xBD000

Definition at line 12 of file smp.h.

6.12.2.17 #define TRAMPOLINE_START PA2KA(SMPBOOT_START)

Definition at line 13 of file smp.h.

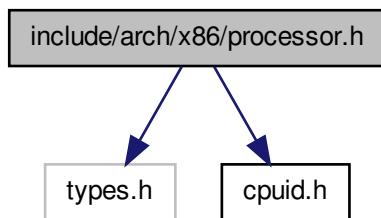
6.12.3 Variable Documentation

6.12.3.1 struct { ... } __attribute__

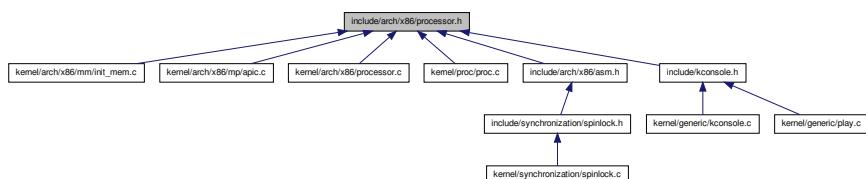
6.13 include/arch/x86/processor.h File Reference

contains constants that are used by Intel x86

```
#include <types.h> #include <cpuid.h> Include dependency graph for
processor.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [cpu_version](#)
- struct [cpu_cache](#)
- struct [cpu_t](#)

Defines

- `#define X86_CR0_PE 0x00000001 /* Protection Enable */`

- #define X86_CR0_MP 0x00000002 /* Monitor Coprocessor */
- #define X86_CR0_EM 0x00000004 /* Emulation */
- #define X86_CR0_TS 0x00000008 /* Task Switched */
- #define X86_CR0_ET 0x00000010 /* Extension type */
- #define X86_CR0_NE 0x00000020 /* Numeric error */
- #define X86_CR0_WP 0x00010000 /* Write Protect */
- #define X86_CR0_AM 0x00040000 /* Alignment Mask */
- #define X86_CR0_NW 0x20000000 /* Not writable through */
- #define X86_CR0_CD 0x40000000 /* Cache disabled */
- #define X86_CR0_PG 0x80000000 /* Paging enable */
- #define X86_CR3_PWT 0x00000008 /* Page Write Through */
- #define X86_CR3_PCD 0x00000010 /* Page Cache Disabled */
- #define X86_CR4_VME 0x00000001 /* Enable VM86 Extension */
- #define X86_CR4_PVI 0x00000002 /* Virtual interrupts flag enable */
- #define X86_CR4_TSD 0x00000004 /* Disable time stamp at ipl 3 */
- #define X86_CR4_DE 0x00000008 /* Enable Debugging extensions */
- #define X86_CR4_PSE 0x00000010 /* Enable Page Size extension */
- #define X86_CR4_PAE 0x00000020 /* Enable Physical address extension */
- #define X86_CR4_MCE 0x00000040 /* Machine Check enable */
- #define X86_CR4_PGE 0x00000080 /* Enable Global Pages */
- #define X86_CR4_PCE 0x00000100 /* Enable Performance Counters at ipl 3*/
- #define X86_CR4_OSFXSR 0x00000200 /* Enable Fast FPU save and restore */
- #define X86_CR4_OSXMMEXCPT 0x00000400 /* Enable unmasked SSE exceptions */
- #define X86_CR4_VMXE 0x00002000 /* Enable VMX virtualization */
- #define X86_CR4_OSXSAVE 0x00004000 /* Enable Xsave and xrestore */
- #define X86_CR8_TPR 0x0000000F /* Task Priority Register */
- #define FLAG_CF 0x1
- #define FLAG_PF 0x4
- #define FLAG_AF 0x10
- #define FLAG_ZF 0x40
- #define FLAG_SF 0x80
- #define FLAG_TF 0x100
- #define FLAG_IF 0x200
- #define FLAG_DF 0x400
- #define FLAG_OF 0x800
- #define FLAG_IOPL 0x1000
- #define FLAG_NT 0x2000
- #define FLAG_RF 0x4000
- #define FLAG_VM 0x8000
- #define FLAG_AC 0x10000
- #define FLAG_VIF 0x20000
- #define FLAG_VIP 0x40000
- #define FLAG_ID 0x80000
- #define VENDOR_INTEL 0x1

- #define **VENDOR_AMD** 0x2
- #define **MFENCE** asm volatile("mfence");
- #define **LFENCE** asm volatile("lfence");
- #define **SFENCE** asm volatile("sfence");

Functions

- struct **cpu_version __attribute__ ((packed))**

Variables

- unsigned **sse3**
- unsigned **_pad0_**
- unsigned **vmx**
- unsigned **smx**
- unsigned **est**
- unsigned **tm2**
- unsigned **ssse3**
- unsigned **_pad1_**
- unsigned **pcid**
- unsigned **dca**
- unsigned **sse4_1**
- unsigned **sse4_2**
- unsigned **x2apic**
- unsigned **_pad2_**
- unsigned **fpu**
- unsigned **vme**
- unsigned **de**
- unsigned **pse**
- unsigned **pae**
- unsigned **apic**
- unsigned **sep**
- unsigned **pge**
- unsigned **mca**
- unsigned **cmov**
- unsigned **pat**
- unsigned **_pad3_**
- unsigned **acpi**
- unsigned **mmx**
- unsigned **fxsr**
- unsigned **sse**
- unsigned **sse2**
- unsigned **ss**
- unsigned **htt**
- unsigned **tm**

- unsigned `__pad4__`
- unsigned `pbe`
- struct {
 - unsigned `stepping`:4
 - unsigned `model`:4
 - unsigned `family`:4
 - unsigned `type`:2
 - unsigned `ss`:2
 - unsigned `exmodel`:4
 - unsigned `exfamily`:8
 - unsigned `sss`:4
} `fms`
- struct {
 - char `brand`
 - char `cache`
 - char `max_id`
 - char `id`
} `generic`
- unsigned `line_size`
- unsigned `phy_line`
- unsigned `assoc`
- unsigned `write_back`
- unsigned `inclusive`
- unsigned `indexing`
- struct `cpu_cache __attribute__`
- char * `cpu_vendors` []
- `cpu_t * cpu`

6.13.1 Detailed Description

contains constants that are used by Intel x86

Author

Saad Talaat

Date

Thursday 24/11/2011

The header contains Control registers constants and EFLAGS masks.

Definition in file [processor.h](#).

6.13.2 Define Documentation

6.13.2.1 `#define FLAG_AC 0x10000`

Definition at line 64 of file processor.h.

6.13.2.2 `#define FLAG_AF 0x10`

Definition at line 53 of file processor.h.

6.13.2.3 `#define FLAG_CF 0x1`

Definition at line 51 of file processor.h.

6.13.2.4 `#define FLAG_DF 0x400`

Definition at line 58 of file processor.h.

6.13.2.5 `#define FLAG_ID 0x80000`

Definition at line 67 of file processor.h.

6.13.2.6 `#define FLAG_IF 0x200`

Definition at line 57 of file processor.h.

6.13.2.7 `#define FLAG_IOPL 0x1000`

Definition at line 60 of file processor.h.

6.13.2.8 `#define FLAG_NT 0x2000`

Definition at line 61 of file processor.h.

6.13.2.9 `#define FLAG_OF 0x800`

Definition at line 59 of file processor.h.

6.13.2.10 `#define FLAG_PF 0x4`

Definition at line 52 of file processor.h.

6.13.2.11 #define FLAG_RF 0x4000

Definition at line 62 of file processor.h.

6.13.2.12 #define FLAG_SF 0x80

Definition at line 55 of file processor.h.

6.13.2.13 #define FLAG_TF 0x100

Definition at line 56 of file processor.h.

6.13.2.14 #define FLAG_VIF 0x20000

Definition at line 65 of file processor.h.

6.13.2.15 #define FLAG_VIP 0x40000

Definition at line 66 of file processor.h.

6.13.2.16 #define FLAG_VM 0x8000

Definition at line 63 of file processor.h.

6.13.2.17 #define FLAG_ZF 0x40

Definition at line 54 of file processor.h.

6.13.2.18 #define LFENCE asm volatile("lfence");

Definition at line 167 of file processor.h.

6.13.2.19 #define MFENCE asm volatile("mfence");

MEMORY

Definition at line 166 of file processor.h.

6.13.2.20 #define SFENCE asm volatile("sfence");

Definition at line 168 of file processor.h.

6.13.2.21 #define VENDOR_AMD 0x2

Definition at line 71 of file processor.h.

6.13.2.22 #define VENDOR_INTEL 0x1

Definition at line 70 of file processor.h.

6.13.2.23 #define X86_CR0_AM 0x00040000 /* Alignment Mask */

Definition at line 22 of file processor.h.

6.13.2.24 #define X86_CR0_CD 0x40000000 /* Cache disabled */

Definition at line 24 of file processor.h.

6.13.2.25 #define X86_CR0_EM 0x00000004 /* Emulation */

Definition at line 17 of file processor.h.

6.13.2.26 #define X86_CR0_ET 0x00000010 /* Extension type */

Definition at line 19 of file processor.h.

6.13.2.27 #define X86_CR0_MP 0x00000002 /* Monitor Coprocessor */

Definition at line 16 of file processor.h.

6.13.2.28 #define X86_CR0_NE 0x00000020 /* Numeric error */

Definition at line 20 of file processor.h.

6.13.2.29 #define X86_CR0_NW 0x20000000 /* Not writable through */

Definition at line 23 of file processor.h.

6.13.2.30 #define X86_CR0_PE 0x00000001 /* Protection Enable */

Definition at line 15 of file processor.h.

6.13.2.31 #define X86_CR0_PG 0x80000000 /* Paging enable */

Definition at line 25 of file processor.h.

6.13.2.32 #define X86_CR0_TS 0x00000008 /* Task Switched */

Definition at line 18 of file processor.h.

6.13.2.33 #define X86_CR0_WP 0x00010000 /* Write Protect */

Definition at line 21 of file processor.h.

6.13.2.34 #define X86_CR3_PCD 0x00000010 /* Page Cache Disabled */

Definition at line 29 of file processor.h.

6.13.2.35 #define X86_CR3_PWT 0x00000008 /* Page Write Through */

Definition at line 28 of file processor.h.

6.13.2.36 #define X86_CR4_DE 0x00000008 /* Enable Debugging extensions */

Definition at line 35 of file processor.h.

6.13.2.37 #define X86_CR4_MCE 0x00000040 /* Machine Check enable */

Definition at line 38 of file processor.h.

6.13.2.38 #define X86_CR4_OSFXSR 0x00000200 /* Enable Fast FPU save and restore */

Definition at line 41 of file processor.h.

6.13.2.39 #define X86_CR4_OSMXMMEXCPT 0x00000400 /* Enable unmasked SSE exceptions */

Definition at line 42 of file processor.h.

6.13.2.40 #define X86_CR4_OSXSAVE 0x00004000 /* Enable Xsave and xrestore */

Definition at line 44 of file processor.h.

6.13.2.41 `#define X86_CR4_PAE 0x00000020 /* Enable Physical address extension */`

Definition at line 37 of file processor.h.

6.13.2.42 `#define X86_CR4_PCE 0x00000100 /* Enable Performance Counters at ipl 3 */`

Definition at line 40 of file processor.h.

6.13.2.43 `#define X86_CR4_PGE 0x00000080 /* Enable Global Pages */`

Definition at line 39 of file processor.h.

6.13.2.44 `#define X86_CR4_PSE 0x00000010 /* Enable Page Size extension */`

Definition at line 36 of file processor.h.

6.13.2.45 `#define X86_CR4_PVI 0x00000002 /* Virtual interrupts flag enable */`

Definition at line 33 of file processor.h.

6.13.2.46 `#define X86_CR4_TSD 0x00000004 /* Disable time stamp at ipl 3 */`

Definition at line 34 of file processor.h.

6.13.2.47 `#define X86_CR4_VME 0x00000001 /* Enable VM86 Extension */`

Definition at line 32 of file processor.h.

6.13.2.48 `#define X86_CR4_VMXE 0x00002000 /* Enable VMX virtualization */`

Definition at line 43 of file processor.h.

6.13.2.49 `#define X86_CR8_TPR 0x0000000F /* Task Priority Register */`

Definition at line 47 of file processor.h.

6.13.3 Function Documentation

6.13.3.1 `struct cpu_version __attribute__ ((packed))`

6.13.4 Variable Documentation

6.13.4.1 **struct cpu_cache __attribute__**

6.13.4.2 **unsigned __pad0__**

Definition at line 107 of file processor.h.

6.13.4.3 **unsigned __pad1__**

Definition at line 113 of file processor.h.

6.13.4.4 **unsigned __pad2__**

Definition at line 119 of file processor.h.

6.13.4.5 **unsigned __pad3__**

Definition at line 147 of file processor.h.

6.13.4.6 **unsigned __pad4__**

Definition at line 156 of file processor.h.

6.13.4.7 **unsigned acpi**

Definition at line 148 of file processor.h.

6.13.4.8 **unsigned apic**

Definition at line 140 of file processor.h.

6.13.4.9 **unsigned assoc**

Definition at line 144 of file processor.h.

6.13.4.10 **char brand**

Definition at line 146 of file processor.h.

6.13.4.11 **char cache**

Definition at line 147 of file processor.h.

6.13.4.12 unsigned cmov

Definition at line 145 of file processor.h.

6.13.4.13 unsigned dca

Definition at line 115 of file processor.h.

6.13.4.14 unsigned de

Definition at line 135 of file processor.h.

6.13.4.15 unsigned est

Definition at line 110 of file processor.h.

6.13.4.16 unsigned exfamily

Definition at line 142 of file processor.h.

6.13.4.17 unsigned exmodel

Definition at line 141 of file processor.h.

6.13.4.18 unsigned family

Definition at line 138 of file processor.h.

6.13.4.19 struct { ... } fms**6.13.4.20 unsigned fpu**

Definition at line 133 of file processor.h.

6.13.4.21 unsigned fxsr

Definition at line 150 of file processor.h.

6.13.4.22 struct { ... } generic**6.13.4.23 unsigned htt**

Definition at line 154 of file processor.h.

6.13.4.24 char id

Definition at line 149 of file processor.h.

6.13.4.25 unsigned inclusive

Definition at line 150 of file processor.h.

6.13.4.26 unsigned indexing

Definition at line 151 of file processor.h.

6.13.4.27 unsigned line_size

Definition at line 142 of file processor.h.

6.13.4.28 char max_id

Definition at line 148 of file processor.h.

6.13.4.29 unsigned mca

Definition at line 144 of file processor.h.

6.13.4.30 unsigned mmx

Definition at line 149 of file processor.h.

6.13.4.31 unsigned model

Definition at line 137 of file processor.h.

6.13.4.32 unsigned pae

Definition at line 138 of file processor.h.

6.13.4.33 unsigned pat

Definition at line 146 of file processor.h.

6.13.4.34 unsigned pbe

Definition at line 157 of file processor.h.

6.13.4.35 unsigned pcid

Definition at line 114 of file processor.h.

6.13.4.36 unsigned pge

Definition at line 143 of file processor.h.

6.13.4.37 unsigned phy_line

Definition at line 143 of file processor.h.

6.13.4.38 unsigned pse

Definition at line 136 of file processor.h.

6.13.4.39 unsigned sep

Definition at line 142 of file processor.h.

6.13.4.40 unsigned smx

Definition at line 109 of file processor.h.

6.13.4.41 unsigned ss

Definition at line 153 of file processor.h.

6.13.4.42 unsigned sse

Definition at line 151 of file processor.h.

6.13.4.43 unsigned sse2

Definition at line 152 of file processor.h.

6.13.4.44 unsigned sse3

Definition at line 106 of file processor.h.

6.13.4.45 unsigned sse4_1

Definition at line 116 of file processor.h.

6.13.4.46 unsigned sse4_2

Definition at line 117 of file processor.h.

6.13.4.47 unsigned sss

Definition at line 143 of file processor.h.

6.13.4.48 unsigned ssse3

Definition at line 112 of file processor.h.

6.13.4.49 unsigned stepping

Definition at line 136 of file processor.h.

6.13.4.50 unsigned tm

Definition at line 155 of file processor.h.

6.13.4.51 unsigned tm2

Definition at line 111 of file processor.h.

6.13.4.52 unsigned type

Definition at line 139 of file processor.h.

6.13.4.53 unsigned vme

Definition at line 134 of file processor.h.

6.13.4.54 unsigned vmx

Definition at line 108 of file processor.h.

6.13.4.55 unsigned write_back

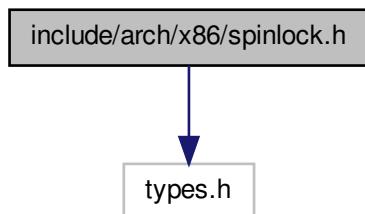
Definition at line 149 of file processor.h.

6.13.4.56 unsigned x2apic

Definition at line 118 of file processor.h.

6.14 include/arch/x86/spinlock.h File Reference

#include <types.h> Include dependency graph for spinlock.h:



Data Structures

- struct [spinlock_t](#)

Functions

- void [acquire_lock \(spinlock_t\)](#)

Variables

- typedef [__attribute__](#)

6.14.1 Function Documentation

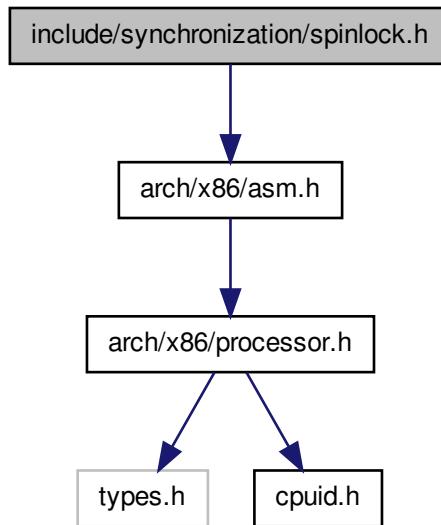
6.14.1.1 `void acquire_lock(spinlock_t)`

6.14.2 Variable Documentation

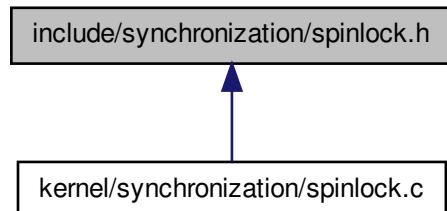
6.14.2.1 `typedef __attribute__`

6.15 include/synchronization/spinlock.h File Reference

#include <arch/x86/asm.h> Include dependency graph for spinlock.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [spinlock](#)

Defines

- #define [spinlock_lock\(lock\)](#) [arch_spinlock_lock\(&\(lock->lock\)\)](#)
- #define [spinlock_unlock\(lock\)](#) [arch_spinlock_unlock\(&\(lock->lock\)\)](#)

Typedefs

- typedef struct [spinlock](#) [spinlock_t](#)

6.15.1 Define Documentation

6.15.1.1 #define [spinlock_lock\(lock \)](#) [arch_spinlock_lock\(&\(lock->lock\)\)](#)

Definition at line 8 of file spinlock.h.

6.15.1.2 #define [spinlock_unlock\(lock \)](#) [arch_spinlock_unlock\(&\(lock->lock\)\)](#)

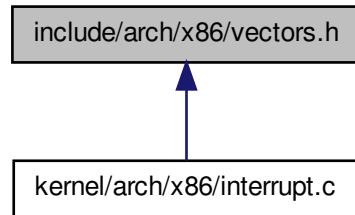
Definition at line 9 of file spinlock.h.

6.15.2 Typedef Documentation

6.15.2.1 typedef struct [spinlock](#) [spinlock_t](#)

6.16 include/arch/x86/vectors.h File Reference

This graph shows which files directly or indirectly include this file:



Variables

- `uint32_t vector_0`
- `uint32_t vector_1`
- `uint32_t vector_2`
- `uint32_t vector_3`
- `uint32_t vector_4`
- `uint32_t vector_5`
- `uint32_t vector_6`
- `uint32_t vector_7`
- `uint32_t vector_8`
- `uint32_t vector_9`
- `uint32_t vector_10`
- `uint32_t vector_11`
- `uint32_t vector_12`
- `uint32_t vector_13`
- `uint32_t vector_14`
- `uint32_t vector_15`
- `uint32_t vector_16`
- `uint32_t vector_17`
- `uint32_t vector_18`
- `uint32_t vector_19`
- `uint32_t vector_20`
- `uint32_t vector_21`
- `uint32_t vector_22`
- `uint32_t vector_23`
- `uint32_t vector_24`

- [uint32_t vector_25](#)
- [uint32_t vector_26](#)
- [uint32_t vector_27](#)
- [uint32_t vector_28](#)
- [uint32_t vector_29](#)
- [uint32_t vector_30](#)
- [uint32_t vector_31](#)
- [uint32_t vector_32](#)
- [uint32_t vector_33](#)
- [uint32_t vector_34](#)
- [uint32_t vector_35](#)
- [uint32_t vector_36](#)
- [uint32_t vector_37](#)
- [uint32_t vector_38](#)
- [uint32_t vector_39](#)
- [uint32_t vector_40](#)
- [uint32_t vector_41](#)
- [uint32_t vector_42](#)
- [uint32_t vector_43](#)
- [uint32_t vector_44](#)
- [uint32_t vector_45](#)
- [uint32_t vector_46](#)
- [uint32_t vector_47](#)
- [uint32_t vector_48](#)
- [uint32_t vector_49](#)
- [uint32_t vector_50](#)
- [uint32_t vector_51](#)
- [uint32_t vector_52](#)
- [uint32_t vector_53](#)
- [uint32_t vector_54](#)
- [uint32_t vector_55](#)
- [uint32_t vector_56](#)
- [uint32_t vector_57](#)
- [uint32_t vector_58](#)
- [uint32_t vector_59](#)
- [uint32_t vector_60](#)
- [uint32_t vector_61](#)
- [uint32_t vector_62](#)
- [uint32_t vector_63](#)

6.16.1 Variable Documentation

6.16.1.1 `uint32_t vector_0`

6.16.1.2 `uint32_t vector_1`

6.16.1.3 `uint32_t vector_10`

6.16.1.4 `uint32_t vector_11`

6.16.1.5 `uint32_t vector_12`

6.16.1.6 `uint32_t vector_13`

6.16.1.7 `uint32_t vector_14`

6.16.1.8 `uint32_t vector_15`

6.16.1.9 `uint32_t vector_16`

6.16.1.10 `uint32_t vector_17`

6.16.1.11 `uint32_t vector_18`

6.16.1.12 `uint32_t vector_19`

6.16.1.13 `uint32_t vector_2`

6.16.1.14 `uint32_t vector_20`

6.16.1.15 `uint32_t vector_21`

6.16.1.16 `uint32_t vector_22`

6.16.1.17 `uint32_t vector_23`

6.16.1.18 `uint32_t vector_24`

6.16.1.19 `uint32_t vector_25`

6.16.1.20 `uint32_t vector_26`

6.16.1.21 `uint32_t vector_27`

6.16.1.22 `uint32_t vector_28`

6.16.1.23 `uint32_t vector_29`

6.16.1.24 `uint32_t vector_3`
6.16.1.25 `uint32_t vector_30`
6.16.1.26 `uint32_t vector_31`
6.16.1.27 `uint32_t vector_32`
6.16.1.28 `uint32_t vector_33`
6.16.1.29 `uint32_t vector_34`
6.16.1.30 `uint32_t vector_35`
6.16.1.31 `uint32_t vector_36`
6.16.1.32 `uint32_t vector_37`
6.16.1.33 `uint32_t vector_38`
6.16.1.34 `uint32_t vector_39`
6.16.1.35 `uint32_t vector_4`
6.16.1.36 `uint32_t vector_40`
6.16.1.37 `uint32_t vector_41`
6.16.1.38 `uint32_t vector_42`
6.16.1.39 `uint32_t vector_43`
6.16.1.40 `uint32_t vector_44`
6.16.1.41 `uint32_t vector_45`
6.16.1.42 `uint32_t vector_46`
6.16.1.43 `uint32_t vector_47`
6.16.1.44 `uint32_t vector_48`
6.16.1.45 `uint32_t vector_49`
6.16.1.46 `uint32_t vector_5`
6.16.1.47 `uint32_t vector_50`

6.16.1.48 `uint32_t vector_51`

6.16.1.49 `uint32_t vector_52`

6.16.1.50 `uint32_t vector_53`

6.16.1.51 `uint32_t vector_54`

6.16.1.52 `uint32_t vector_55`

6.16.1.53 `uint32_t vector_56`

6.16.1.54 `uint32_t vector_57`

6.16.1.55 `uint32_t vector_58`

6.16.1.56 `uint32_t vector_59`

6.16.1.57 `uint32_t vector_6`

6.16.1.58 `uint32_t vector_60`

6.16.1.59 `uint32_t vector_61`

6.16.1.60 `uint32_t vector_62`

6.16.1.61 `uint32_t vector_63`

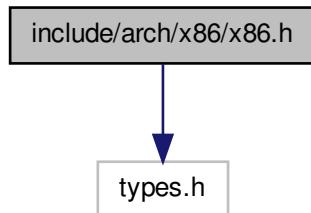
6.16.1.62 `uint32_t vector_7`

6.16.1.63 `uint32_t vector_8`

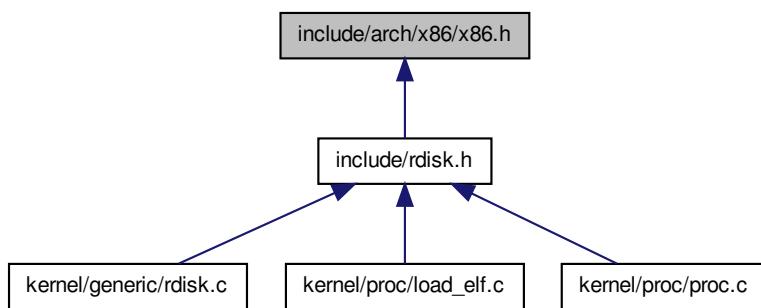
6.16.1.64 `uint32_t vector_9`

6.17 include/arch/x86/x86.h File Reference

#include <types.h> Include dependency graph for x86.h:



This graph shows which files directly or indirectly include this file:



Functions

- static __inline void `outb` (`uint8_t` data, `int` port)
- static __inline `uint8_t inb` (`int` port)
- static void `outw` (`uint16_t` data, `int` port)
- static `uint16_t inw` (`int` port)
- static __inline void `outl` (`uint32_t` data, `int` port)
- static `uint32_t inl` (`int` port)
- static void `insb` (`void *addr`, `int cnt`, `int port`)

- static void `insl` (void *addr, int cnt, int port)
- static void `insw` (void *addr, int cnt, int port)
- static void `cli` (void)
- static void `sti` (void)
- static void `write_cr0` (uint32_t value)
- static uint32_t `read_cr0` (void)
- static void `write_cr1` (uint32_t value)
- static uint32_t `read_cr1` (void)
- static void `write_cr2` (uint32_t value)
- static uint32_t `read_cr2` (void)
- static void `write_cr3` (uint32_t value)
- static uint32_t `read_cr3` (void)
- static void `write_cr4` (uint32_t value)
- static uint32_t `read_cr4` (void)
- static void `invlpg` (void *addr)
- static void `write_tr` (uint16_t selector)
- static uint32_t `read_esp` (void)
- static void `read_msr` (uint32_t ecx, uint32_t *edx, uint32_t *eax)
- static void `write_msr` (uint32_t ecx, uint32_t edx, uint32_t eax)

6.17.1 Function Documentation

6.17.1.1 static void `cli` (void) [inline, static]

Definition at line 60 of file x86.h.

6.17.1.2 static __inline uint8_t `inb` (int port) [static]

Definition at line 14 of file x86.h.

6.17.1.3 static uint32_t `inl` (int port) [inline, static]

Definition at line 36 of file x86.h.

6.17.1.4 static void `insb` (void * addr, int cnt, int port) [inline, static]

Definition at line 42 of file x86.h.

6.17.1.5 static void `insl` (void * addr, int cnt, int port) [inline, static]

Definition at line 48 of file x86.h.

6.17.1.6 static void **insw** (void * *addr*, int *cnt*, int *port*) [inline, static]

Definition at line 54 of file x86.h.

6.17.1.7 static void **invlpg** (void * *addr*) [inline, static]

Definition at line 127 of file x86.h.

6.17.1.8 static uint16_t **inw** (int *port*) [inline, static]

Definition at line 25 of file x86.h.

6.17.1.9 static __inline void **outb** (uint8_t *data*, int *port*) [static]

Definition at line 10 of file x86.h.

6.17.1.10 static __inline void **outl** (uint32_t *data*, int *port*) [static]

Definition at line 31 of file x86.h.

6.17.1.11 static void **outw** (uint16_t *data*, int *port*) [inline, static]

Definition at line 20 of file x86.h.

6.17.1.12 static uint32_t **read_cr0** (void) [inline, static]

Definition at line 72 of file x86.h.

6.17.1.13 static uint32_t **read_cr1** (void) [inline, static]

Definition at line 84 of file x86.h.

6.17.1.14 static uint32_t **read_cr2** (void) [inline, static]

Definition at line 96 of file x86.h.

6.17.1.15 static uint32_t **read_cr3** (void) [inline, static]

Definition at line 108 of file x86.h.

6.17.1.16 **static uint32_t read_cr4(void)** [inline, static]

Definition at line 120 of file x86.h.

6.17.1.17 **static uint32_t read_esp(void)** [inline, static]

Definition at line 138 of file x86.h.

6.17.1.18 **static void read_msr(uint32_t ecx, uint32_t *edx, uint32_t *eax)**
[inline, static]

Parameters

<i>ecx</i>	the MSR index
<i>edx</i>	the high msr value
<i>eax</i>	the low msr value

Returns

none

Definition at line 151 of file x86.h.

6.17.1.19 **static void sti(void)** [inline, static]

Definition at line 63 of file x86.h.

6.17.1.20 **static void write_cr0(uint32_t value)** [inline, static]

Definition at line 67 of file x86.h.

6.17.1.21 **static void write_cr1(uint32_t value)** [inline, static]

Definition at line 79 of file x86.h.

6.17.1.22 **static void write_cr2(uint32_t value)** [inline, static]

Definition at line 91 of file x86.h.

6.17.1.23 **static void write_cr3(uint32_t value)** [inline, static]

Definition at line 103 of file x86.h.

6.17.1.24 static void write_cr4(uint32_t value) [inline, static]

Definition at line 115 of file x86.h.

6.17.1.25 static void write_msr(uint32_t ecx, uint32_t edx, uint32_t eax)
[inline, static]

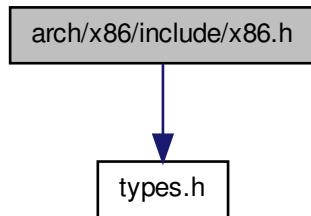
Definition at line 156 of file x86.h.

6.17.1.26 static void write_tr(uint16_t selector) [inline, static]

Definition at line 133 of file x86.h.

6.18 arch/x86/include/x86.h File Reference

#include <types.h> Include dependency graph for x86.h:

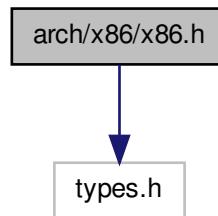


Functions

- static __inline void outb (uint8_t data, int port)
- static __inline uint8_t inb (int port)
- static void outw (uint16_t data, int port)
- static uint16_t inw (int port)
- static __inline void outl (uint32_t data, int port)
- static uint32_t inl (int port)
- static void insb (void *addr, int cnt, int port)
- static void insl (void *addr, int cnt, int port)
- static void insw (void *addr, int cnt, int port)

6.19 arch/x86/x86.h File Reference

#include <types.h> Include dependency graph for x86.h:



This graph shows which files directly or indirectly include this file:



Functions

- static __inline void `outb` (`uint8_t` data, `int` port)
- static __inline `uint8_t` `inb` (`int` port)
- static void `outw` (`uint16_t` data, `int` port)
- static `uint16_t` `inw` (`int` port)
- static __inline void `outl` (`uint32_t` data, `int` port)
- static `uint32_t` `inl` (`int` port)
- static void `insb` (`void *`addr, `int` cnt, `int` port)
- static void `insl` (`void *`addr, `int` cnt, `int` port)
- static void `insw` (`void *`addr, `int` cnt, `int` port)
- static void `cli` (`void`)
- static void `sti` (`void`)
- static void `write_cr0` (`uint32_t` value)
- static `uint32_t` `read_cr0` (`void`)
- static void `write_cr1` (`uint32_t` value)
- static `uint32_t` `read_cr1` (`void`)
- static void `write_cr2` (`uint32_t` value)
- static `uint32_t` `read_cr2` (`void`)
- static void `write_cr3` (`uint32_t` value)
- static `uint32_t` `read_cr3` (`void`)

- static void `write_cr4` (`uint32_t` value)
- static `uint32_t` `read_cr4` (void)
- static void `cpuid` ()
- static void `invlpg` (void *addr)

6.19.1 Function Documentation

6.19.1.1 static void cli (void) [inline, static]

Definition at line 60 of file x86.h.

6.19.1.2 static void cpuid () [inline, static]

Definition at line 126 of file x86.h.

6.19.1.3 static __inline uint8_t inb (int port) [static]

Definition at line 14 of file x86.h.

6.19.1.4 static uint32_t inl (int port) [inline, static]

Definition at line 36 of file x86.h.

6.19.1.5 static void insb (void * addr, int cnt, int port) [inline, static]

Definition at line 42 of file x86.h.

6.19.1.6 static void insl (void * addr, int cnt, int port) [inline, static]

Definition at line 48 of file x86.h.

6.19.1.7 static void insw (void * addr, int cnt, int port) [inline, static]

Definition at line 54 of file x86.h.

6.19.1.8 static void invlpg (void * addr) [inline, static]

Definition at line 131 of file x86.h.

6.19.1.9 static uint16_t inw (int port) [inline, static]

Definition at line 25 of file x86.h.

6.19.1.10 **static __inline void outb(uint8_t data, int port) [static]**

Definition at line 10 of file x86.h.

6.19.1.11 **static __inline void outl(uint32_t data, int port) [static]**

Definition at line 31 of file x86.h.

6.19.1.12 **static void outw(uint16_t data, int port) [inline, static]**

Definition at line 20 of file x86.h.

6.19.1.13 **static uint32_t read_cr0(void) [inline, static]**

Definition at line 72 of file x86.h.

6.19.1.14 **static uint32_t read_cr1(void) [inline, static]**

Definition at line 84 of file x86.h.

6.19.1.15 **static uint32_t read_cr2(void) [inline, static]**

Definition at line 96 of file x86.h.

6.19.1.16 **static uint32_t read_cr3(void) [inline, static]**

Definition at line 108 of file x86.h.

6.19.1.17 **static uint32_t read_cr4(void) [inline, static]**

Definition at line 120 of file x86.h.

6.19.1.18 **static void sti(void) [inline, static]**

Definition at line 63 of file x86.h.

6.19.1.19 **static void write_cr0(uint32_t value) [inline, static]**

Definition at line 67 of file x86.h.

6.19.1.20 `static void write_cr1(uint32_t value) [inline, static]`

Definition at line 79 of file x86.h.

6.19.1.21 `static void write_cr2(uint32_t value) [inline, static]`

Definition at line 91 of file x86.h.

6.19.1.22 `static void write_cr3(uint32_t value) [inline, static]`

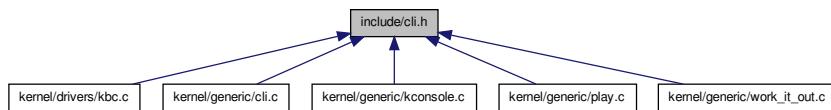
Definition at line 103 of file x86.h.

6.19.1.23 `static void write_cr4(uint32_t value) [inline, static]`

Definition at line 115 of file x86.h.

6.20 include/cli.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define `MAXBUFSIZE` 512

Functions

- void `console_init` (void)
initializes console.
- void `console_clear` (void)
- void `console_interrupt` (`int (*intr)(void)`)
*reads a char from an input device interrupt. `int (*intr)(void)` interrupt handler.*
- `int console_getc` (void)
reads a character from an input device [keyboard].
- void `console_putc` (`int c`)
writes a character to screen wraps `cga_putc` function

Variables

```

• struct {
    uint8_t buf [MAXBUFSIZE]
    uint32_t rpos
    uint32_t wpos
} cons

```

6.20.1 Define Documentation

6.20.1.1 #define MAXBUFSIZE 512

Definition at line 8 of file cli.h.

6.20.2 Variable Documentation

6.20.2.1 uint8_t buf[MAXBUFSIZE]

Definition at line 44 of file cli.h.

6.20.2.2 struct { ... } cons [static]

6.20.2.3 uint32_t rpos

Definition at line 45 of file cli.h.

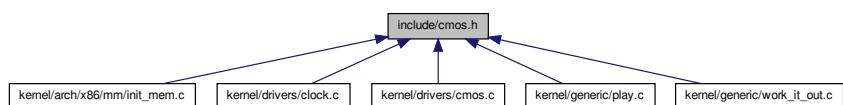
6.20.2.4 uint32_t wpos

Definition at line 46 of file cli.h.

6.21 include/cmos.h File Reference

defines constants for RTC <http://catreloaded.net>

This graph shows which files directly or indirectly include this file:



Defines

- #define CMOS_INDEXPORT 0x70 /* the CMOS/RTC index register port */
- #define CMOS_DATAPORT 0x71 /* the CMOS/RTC data register port */
- #define RTC_SECONDS 0x0
- #define RTC_ALRMSECOND 0x1
- #define RTC_MINUTES 0x2
- #define RTC_ALRMMINUTE 0x3
- #define RTC_HOUR 0x4
- #define RTC_ALRMHOUR 0x5
- #define RTC_DAY_WEEK 0x6
- #define RTC_DAY_MONTH 0x7
- #define RTC_MONTH 0x8
- #define RTC_YEAR 0x9
- #define RTC_STATUS_A 0xA /* the RTC Status register A */
- #define RTC_STATUS_B 0xB /* the RTC Status register B */
- #define STAT_RUN 0x7F /*values to be used in the Masking of the RTC_STATUS_B*/
- #define STAT_HALT 0x80
- #define STAT_PER_INTR 0x40
- #define STAT_ALRM_INTR 0x20
- #define STAT_UPDT_INTR 0x10
- #define STAT_SQRWV_INTR 0x08
- #define STAT_CAL_BIN 0x04
- #define STAT_CAL_BCD 0xFB
- #define STAT_CAL_HR24 0x02
- #define STAT_CAL_HR12 0xFD
- #define STAT_DAY_LGHT 0x01
- #define CMOS_STATUS_C 0xC
- #define CMOS_STATUS_D 0xD
- #define CMOS_STATUS_E 0xE /* RTC diagnostic register E */
- #define CMOS_STATUS_F 0xF /* RTC Shut-down status register F */
- #define CMOS_STATUS_10 0x10 /* RTC DRIVER TYPE for A: and B: register */
- #define CMOS_FIXED_DISK 0x12 /* RTC Fixed disk drive type for drive 0 and drive 1 register */
- #define CMOS_EQUIP_BYTE 0x14 /* RTC Equip byte */
- #define CMOS_SYSBASE_LSB 0x15 /* LSB of System base memory in Kilo byte*/
- #define CMOS_SYSBASE_MSB 0x16 /* MSB of ~~~~~*/
- #define CMOS_EXTMEM_LSB 0x17 /* LSB of total extended memory in Kilo byte */
- #define CMOS_EXTMEM_MSB 0x18 /* MSB of ~~~~~*/
- #define CMOS_DRIVE_C 0x19 /* Drive C extention type */
- #define CMOS_DRIVE_D 0x1A /* Drive D extention type */

Functions

- `uint8_t cmos_set_power_stat (uint8_t stat)`
supports CMOS Status B power options.
- `uint8_t cmos_get_seconds (void)`
- `uint32_t cmos_get_reg (uint8_t value)`
Gets RTC registers values.

6.21.1 Detailed Description

defines constants for RTC <http://catreloaded.net>

Author

Menna Essa , Saad Talaat

Date

29/10/2011

Definition in file `cmos.h`.

6.21.2 Define Documentation

6.21.2.1 `#define CMOS_DATAPORT 0x71 /* the CMOS/RTC data register port */`

Definition at line 13 of file `cmos.h`.

6.21.2.2 `#define CMOS_DRIVE_C 0x19 /* Drive C extention type */`

Definition at line 54 of file `cmos.h`.

6.21.2.3 `#define CMOS_DRIVE_D 0x1A /* Drive D extention type */`

Definition at line 55 of file `cmos.h`.

6.21.2.4 `#define CMOS_EQUIP_BYTE 0x14 /* RTC Equip byte */`

Definition at line 49 of file `cmos.h`.

6.21.2.5 `#define CMOS_EXTMEM_LSB 0x17 /* LSB of total extended memory in Kilo byte */`

Definition at line 52 of file `cmos.h`.

```
6.21.2.6 #define CMOS_EXTMEM_MSB 0x18 /* MSB of  
~~~~~*/
```

Definition at line 53 of file cmos.h.

```
6.21.2.7 #define CMOS_FIXED_DISK 0x12 /* RTC Fixed disk drive type for drive 0 and drive  
1 register */
```

Definition at line 48 of file cmos.h.

```
6.21.2.8 #define CMOS_INDEXPORT 0x70 /* the CMOS/RTC index register port */
```

Definition at line 12 of file cmos.h.

```
6.21.2.9 #define CMOS_STATUS_10 0x10 /* RTC DRIVER TYPE for A: and B: register */
```

Definition at line 47 of file cmos.h.

```
6.21.2.10 #define CMOS_STATUS_C 0xC
```

Definition at line 43 of file cmos.h.

```
6.21.2.11 #define CMOS_STATUS_D 0xD
```

Definition at line 44 of file cmos.h.

```
6.21.2.12 #define CMOS_STATUS_E 0xE /* RTC diagnostic register E */
```

Definition at line 45 of file cmos.h.

```
6.21.2.13 #define CMOS_STATUS_F 0xF /* RTC Shut-down status register F */
```

Definition at line 46 of file cmos.h.

```
6.21.2.14 #define CMOS_SYSBASE_LSB 0x15 /* LSB of System base memory in Kilo byte*/
```

Definition at line 50 of file cmos.h.

```
6.21.2.15 #define CMOS_SYSBASE_MSB 0x16 /* MSB of  
~~~~~*/
```

Definition at line 51 of file cmos.h.

6.21.2.16 #define RTC_ALRMHOUR 0x5

Definition at line 21 of file cmos.h.

6.21.2.17 #define RTC_ALRMMINUTE 0x3

Definition at line 19 of file cmos.h.

6.21.2.18 #define RTC_ALRMSECOND 0x1

Definition at line 17 of file cmos.h.

6.21.2.19 #define RTC_DAY_MONTH 0x7

Definition at line 23 of file cmos.h.

6.21.2.20 #define RTC_DAY_WEEK 0x6

Definition at line 22 of file cmos.h.

6.21.2.21 #define RTC_HOUR 0x4

Definition at line 20 of file cmos.h.

6.21.2.22 #define RTC_MINUTES 0x2

Definition at line 18 of file cmos.h.

6.21.2.23 #define RTC_MONTH 0x8

Definition at line 24 of file cmos.h.

6.21.2.24 #define RTC_SECONDS 0x0

Definition at line 16 of file cmos.h.

6.21.2.25 #define RTC_STATUS_A 0xA /* the RTC Status register A */

Definition at line 26 of file cmos.h.

6.21.2.26 #define RTC_STATUS_B 0xB /* the RTC Status register B */

Definition at line 27 of file cmos.h.

6.21.2.27 #define RTC_YEAR 0x9

Definition at line 25 of file cmos.h.

6.21.2.28 #define STAT_ALRM_INTR 0x20

Definition at line 33 of file cmos.h.

6.21.2.29 #define STAT_CAL_BCD 0xFB

Definition at line 37 of file cmos.h.

6.21.2.30 #define STAT_CAL_BIN 0x04

Definition at line 36 of file cmos.h.

6.21.2.31 #define STAT_CAL_HR12 0xFD

Definition at line 39 of file cmos.h.

6.21.2.32 #define STAT_CAL_HR24 0x02

Definition at line 38 of file cmos.h.

6.21.2.33 #define STAT_DAY_LGHT 0x01

Definition at line 40 of file cmos.h.

6.21.2.34 #define STAT_HALT 0x80

Definition at line 31 of file cmos.h.

6.21.2.35 #define STAT_PER_INTR 0x40

Definition at line 32 of file cmos.h.

6.21.2.36 #define STAT_RUN 0x7F /*values to be used in the Masking of the RTC_STATUS_B*/

Definition at line 30 of file cmos.h.

6.21.2.37 #define STAT_SQRWV_INTR 0x08

Definition at line 35 of file cmos.h.

6.21.2.38 #define STAT UPDT INTR 0x10

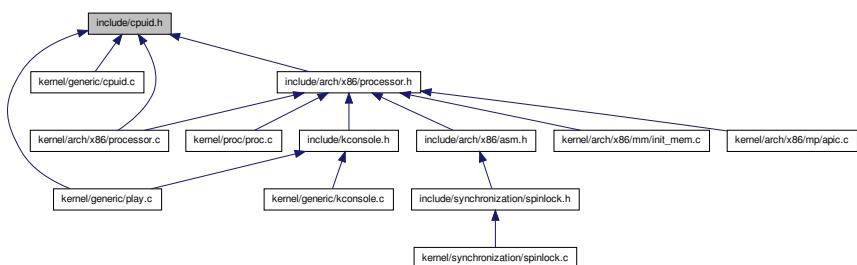
Definition at line 34 of file cmos.h.

6.21.3 Function Documentation

6.21.3.1 uint8_t cmos_get_seconds(void)

6.22 include/cpuid.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct `cpuid_t`

Defines

- #define CPUID_INFO 0x0
 - #define CPUID_VERSION 0x1
 - #define CPUID_CACHE 0x4
 - #define CPUID_INTEL_EBX_0 0x756e6547
 - #define CPUID_INTEL_ECX_0 0x6c65746e

- #define **CPUID_INTEL_EDX_0** 0x49656e69
- #define **CPUID_P4HT_0** 0x49656e69 /* Intel Pentium 4 support Hyperthreading */

Functions

- static void **cpuid** (uint32_t eax_initial, cpuid_t *c)

6.22.1 Define Documentation

6.22.1.1 #define CPUID_CACHE 0x4

Definition at line 7 of file cpuid.h.

6.22.1.2 #define CPUID_INFO 0x0

Definition at line 5 of file cpuid.h.

6.22.1.3 #define CPUID_INTEL_EBX_0 0x756e6547

Definition at line 10 of file cpuid.h.

6.22.1.4 #define CPUID_INTEL_ECX_0 0x6c65746e

Definition at line 11 of file cpuid.h.

6.22.1.5 #define CPUID_INTEL_EDX_0 0x49656e69

Definition at line 12 of file cpuid.h.

6.22.1.6 #define CPUID_P4HT_0 0x49656e69 /* Intel Pentium 4 support Hyperthreading */

Definition at line 13 of file cpuid.h.

6.22.1.7 #define CPUID_VERSION 0x1

Definition at line 6 of file cpuid.h.

6.22.2 Function Documentation

6.22.2.1 static void cpuid(uint32_t eax_initial, cpuid_t * c) [inline, static]

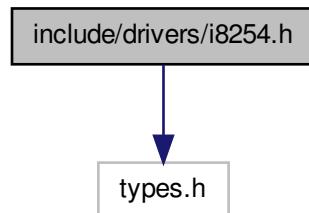
Definition at line 26 of file cpuid.h.

6.23 include/device.h File Reference

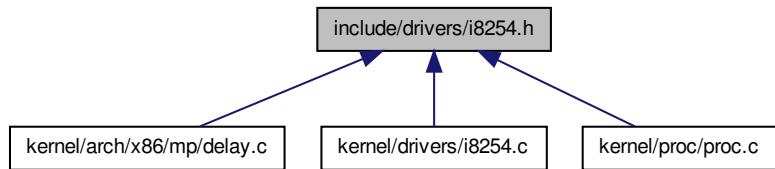
6.24 include/drivers/i8254.h File Reference

Intel 8254 PIT.

```
#include <types.h> Include dependency graph for i8254.h:
```



This graph shows which files directly or indirectly include this file:



Defines

- #define PIT_CHANNEL0 0x40
- #define PIT_CHANNEL1 0x41
- #define PIT_CHANNEL2 0x42
- #define PIT_MODE 0x43
- #define PIT_TICKS 0x1234DF
- #define MODE_ONESHOT 0x2
- #define MODE_RATEGEN 0x4

- #define MODE_WAVEGEN 0x6
- #define MODE_SOFTSRB 0x8
- #define MODE_HARDSRB 0xA
- #define LATCH_LSB 0x10
- #define LATCH_MSB 0x20
- #define LATCH_ALL 0x30
- #define SELECT_CNT0 0x0
- #define SELECT_CNT1 0x40
- #define SELECT_CNT2 0x80

Functions

- void [i8254_init](#) (void)
Initiate the PIT to interrupt every time slice.
- void [i8254_calibrate_delay_loop](#) (void)
calibrates the number of loops per 1 ms

Variables

- [uint32_t delay_loop_const](#)

6.24.1 Detailed Description

Intel 8254 PIT.

Author

Saad Talaat , Menna Essa

Definition in file [i8254.h](#).

6.24.2 Define Documentation

6.24.2.1 #define LATCH_ALL 0x30

Definition at line 27 of file i8254.h.

6.24.2.2 #define LATCH_LSB 0x10

Definition at line 25 of file i8254.h.

6.24.2.3 #define LATCH_MSB 0x20

Definition at line 26 of file i8254.h.

6.24.2.4 #define MODE_HARDSRB 0xA

Definition at line 22 of file i8254.h.

6.24.2.5 #define MODE_ONESHOT 0x2

Definition at line 18 of file i8254.h.

6.24.2.6 #define MODE_RATEGEN 0x4

Definition at line 19 of file i8254.h.

6.24.2.7 #define MODE_SOFTSRB 0x8

Definition at line 21 of file i8254.h.

6.24.2.8 #define MODE_WAVEGEN 0x6

Definition at line 20 of file i8254.h.

6.24.2.9 #define PIT_CHANNEL0 0x40

Definition at line 10 of file i8254.h.

6.24.2.10 #define PIT_CHANNEL1 0x41

Definition at line 11 of file i8254.h.

6.24.2.11 #define PIT_CHANNEL2 0x42

Definition at line 12 of file i8254.h.

6.24.2.12 #define PIT_MODE 0x43

Definition at line 13 of file i8254.h.

6.24.2.13 #define PIT_TICKS 0x1234DF

Definition at line 14 of file i8254.h.

6.24.2.14 #define SELECT_CNT0 0x0

Definition at line 30 of file i8254.h.

6.24.2.15 #define SELECT_CNT1 0x40

Definition at line 31 of file i8254.h.

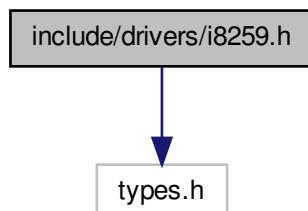
6.24.2.16 #define SELECT_CNT2 0x80

Definition at line 32 of file i8254.h.

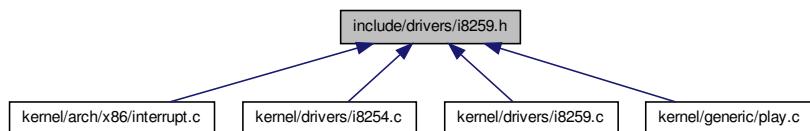
6.25 include/drivers/i8259.h File Reference

Intel 8259 PIC header file.

#include <types.h> Include dependency graph for i8259.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define PIC_MASTER_ICW 0x20
- #define PIC_MASTER_OCW 0x21
- #define PIC_SLAVE_ICW 0xA0
- #define PIC_SLAVE_OCW 0xA1
- #define ICW1_ICW4NEEDED 0x1
- #define ICW1_SINGLEMODE 0x2
- #define ICW1_4BTVECTORS 0x4
- #define ICW1_LEVELMODE 0x8
- #define ICW1_ISSUED 0x10
- #define PIC_SLAVE_0 0x1
- #define PIC_SLAVE_1 0x2
- #define PIC_SLAVE_2 0x4
- #define PIC_SLAVE_3 0x8
- #define PIC_SLAVE_4 0x10
- #define PIC_SLAVE_5 0x20
- #define PIC_SLAVE_6 0x40
- #define PIC_SLAVE_7 0x80
- #define SLAVE_ON_0 0x0
- #define SLAVE_ON_1 0x1
- #define SLAVE_ON_2 0x2
- #define SLAVE_ON_3 0x3
- #define SLAVE_ON_4 0x4
- #define SLAVE_ON_5 0x5
- #define SLAVE_ON_6 0x6
- #define SLAVE_ON_7 0x7
- #define ICW4_86_88 0x1
- #define ICW4_AUTOEOI 0x2
- #define ICW4_NONBUFFER 0x4
- #define ICW4_SLAVE_BUF 0x8
- #define ICW4_MASTER_BUF 0xC
- #define ICW4_FULLNESTED 0x10
- #define EXC_BASE 0x0
- #define EXC_SIZE 0x20
- #define IRQ_BASE (EXC_BASE+EXC_SIZE)
- #define IRQ_SIZE 0x10
- #define IRQ_END (IRQ_BASE+IRQ_SIZE)
- #define INTR_EMPTY_BASE IRQ_END
- #define INTR_END 0xFF
- #define IRQ_PIT 0x0
- #define IRQ_KBD 0x1
- #define IRQ_VID 0x2
- #define IRQ_SER1 0x3
- #define IRQ_SER2 0x4
- #define IRQ_LPT2 0x5
- #define IRQ_FLPY 0x6

- #define `IRQ_SPUR` 0x7
- #define `IRQ_RTC` 0x8
- #define `IRQ_MOUSE` 0xC
- #define `IRQ_FPU` 0xE
- #define `IRQ_PRIHD` 0xF
- #define `IRQ_SECHD` 0x10
- #define `OCW1_PIT` 0x1
- #define `OCW1_RTC` (`OCW1_PIT` << 8)
- #define `OCW1_KYBRD` 0x2
- #define `OCW1_VIDEO` 0x4
- #define `OCW1_SERI2` 0x8
- #define `OCW1_SERI1` 0x10
- #define `OCW1_MOUSE` (`OCW1_SERI1` << 8)
- #define `OCW1_FDISK` 0x20
- #define `OCW1_FPU` (`OCW1_FDISK` << 8)
- #define `OCW1_FLPPY` 0x40
- #define `OCW1_PRIHD` (`OCW1_FLPPY` << 8)
- #define `OCW1_SECHD` 0x80
- #define `OCW1_PPRNT` (`OCW1_SECHD` << 8)
- #define `OCW1_INTRS_OFF` 0xFF
- #define `OCW3_READIRR` 0xA
- #define `OCW3_READISR` 0xB
- #define `PIC_EOI` 0x20

Functions

- void `i8259_init` (void)

Initializing the PIC to work under Protected mode.
- void `i8259_disable` (void)

Disables the PIC IRQs.
- void `i8259_mask_irq` (`uint16_t`)

get the in service interrupt.
- void `i8259_unmask_irq` (`uint16_t`)

reads the IRR register which holds raised interrupt priority
- `uint16_t i8259_read_isr` (void)

End of interrupt instruction, used for ISRs.
- `uint16_t i8259_read_irr` (void)

End of interrupt instruction, used for ISRs.
- void `i8259_eoi` (`uint8_t eoi`)

End of interrupt instruction, used for ISRs.

6.25.1 Detailed Description

Intel 8259 PIC header file.

Author

Saad Talaat

Definition in file `i8259.h`.

6.25.2 Define Documentation

6.25.2.1 #define EXC_BASE 0x0

Definition at line 52 of file i8259.h.

6.25.2.2 #define EXC_SIZE 0x20

Definition at line 53 of file i8259.h.

6.25.2.3 #define ICW1_4BTVECTORS 0x4

Definition at line 21 of file i8259.h.

6.25.2.4 #define ICW1_ICW4NEEDED 0x1

Definition at line 19 of file i8259.h.

6.25.2.5 #define ICW1_ISSUED 0x10

Definition at line 23 of file i8259.h.

6.25.2.6 #define ICW1_LEVELMODE 0x8

Definition at line 22 of file i8259.h.

6.25.2.7 #define ICW1_SINGLEMODE 0x2

Definition at line 20 of file i8259.h.

6.25.2.8 #define ICW4_86_88 0x1

Definition at line 44 of file i8259.h.

6.25.2.9 #define ICW4_AUTOEOI 0x2

Definition at line 45 of file i8259.h.

6.25.2.10 #define ICW4_FULLNESTED 0x10

Definition at line 49 of file i8259.h.

6.25.2.11 #define ICW4_MASTER_BUF 0xC

Definition at line 48 of file i8259.h.

6.25.2.12 #define ICW4_NONBUFFER 0x4

Definition at line 46 of file i8259.h.

6.25.2.13 #define ICW4_SLAVE_BUF 0x8

Definition at line 47 of file i8259.h.

6.25.2.14 #define INTR_EMPTY_BASE IRQ_END

Definition at line 57 of file i8259.h.

6.25.2.15 #define INTR_END 0xFF

Definition at line 58 of file i8259.h.

6.25.2.16 #define IRQ_BASE (EXC_BASE+EXC_SIZE)

Definition at line 54 of file i8259.h.

6.25.2.17 #define IRQ_END (IRQ_BASE+IRQ_SIZE)

Definition at line 56 of file i8259.h.

6.25.2.18 #define IRQ_FLPY 0x6

Definition at line 67 of file i8259.h.

6.25.2.19 #define IRQ_FPU 0xE

Definition at line 71 of file i8259.h.

6.25.2.20 #define IRQ_KBD 0x1

Definition at line 62 of file i8259.h.

6.25.2.21 #define IRQ_LPT2 0x5

Definition at line 66 of file i8259.h.

6.25.2.22 #define IRQ_MOUSE 0xC

Definition at line 70 of file i8259.h.

6.25.2.23 #define IRQ_PIT 0x0

Definition at line 61 of file i8259.h.

6.25.2.24 #define IRQ_PRIHD 0xF

Definition at line 72 of file i8259.h.

6.25.2.25 #define IRQ_RTC 0x8

Definition at line 69 of file i8259.h.

6.25.2.26 #define IRQ_SECHD 0x10

Definition at line 73 of file i8259.h.

6.25.2.27 #define IRQ_SER1 0x3

Definition at line 64 of file i8259.h.

6.25.2.28 #define IRQ_SER2 0x4

Definition at line 65 of file i8259.h.

6.25.2.29 #define IRQ_SIZE 0x10

Definition at line 55 of file i8259.h.

6.25.2.30 #define IRQ_SPUR 0x7

Definition at line 68 of file i8259.h.

6.25.2.31 #define IRQ_VID 0x2

Definition at line 63 of file i8259.h.

6.25.2.32 #define OCW1_FDISK 0x20

Definition at line 84 of file i8259.h.

6.25.2.33 #define OCW1_FLPPY 0x40

Definition at line 86 of file i8259.h.

6.25.2.34 #define OCW1_FPU (OCW1_FDISK << 8)

Definition at line 85 of file i8259.h.

6.25.2.35 #define OCW1_INTRS_OFF 0xFF

Definition at line 90 of file i8259.h.

6.25.2.36 #define OCW1_KYBRD 0x2

Definition at line 79 of file i8259.h.

6.25.2.37 #define OCW1_MOUSE (OCW1_SERI1 << 8)

Definition at line 83 of file i8259.h.

6.25.2.38 #define OCW1_PIT 0x1

Definition at line 77 of file i8259.h.

6.25.2.39 #define OCW1_PPRNT (OCW1_SECHD << 8)

Definition at line 89 of file i8259.h.

6.25.2.40 #define OCW1_PRIHD (OCW1_FLPPY << 8)

Definition at line 87 of file i8259.h.

6.25.2.41 #define OCW1_RTC (OCW1_PIT << 8)

Definition at line 78 of file i8259.h.

6.25.2.42 #define OCW1_SECHD 0x80

Definition at line 88 of file i8259.h.

6.25.2.43 #define OCW1_SERI1 0x10

Definition at line 82 of file i8259.h.

6.25.2.44 #define OCW1_SERI2 0x8

Definition at line 81 of file i8259.h.

6.25.2.45 #define OCW1_VIDEO 0x4

Definition at line 80 of file i8259.h.

6.25.2.46 #define OCW3_READIRR 0xA

Definition at line 93 of file i8259.h.

6.25.2.47 #define OCW3_READISR 0xB

Definition at line 94 of file i8259.h.

6.25.2.48 #define PIC_EOI 0x20

Definition at line 97 of file i8259.h.

6.25.2.49 #define PIC_MASTER_ICW 0x20

Definition at line 13 of file i8259.h.

6.25.2.50 #define PIC_MASTER_OCW 0x21

Definition at line 14 of file i8259.h.

6.25.2.51 #define PIC_SLAVE_0 0x1

Definition at line 25 of file i8259.h.

6.25.2.52 #define PIC_SLAVE_1 0x2

Definition at line 26 of file i8259.h.

6.25.2.53 #define PIC_SLAVE_2 0x4

Definition at line 27 of file i8259.h.

6.25.2.54 #define PIC_SLAVE_3 0x8

Definition at line 28 of file i8259.h.

6.25.2.55 #define PIC_SLAVE_4 0x10

Definition at line 29 of file i8259.h.

6.25.2.56 #define PIC_SLAVE_5 0x20

Definition at line 30 of file i8259.h.

6.25.2.57 #define PIC_SLAVE_6 0x40

Definition at line 31 of file i8259.h.

6.25.2.58 #define PIC_SLAVE_7 0x80

Definition at line 32 of file i8259.h.

6.25.2.59 #define PIC_SLAVE_ICW 0xA0

Definition at line 15 of file i8259.h.

6.25.2.60 #define PIC_SLAVE_OCW 0xA1

Definition at line 16 of file i8259.h.

6.25.2.61 #define SLAVE_ON_0 0x0

Definition at line 34 of file i8259.h.

6.25.2.62 #define SLAVE_ON_1 0x1

Definition at line 35 of file i8259.h.

6.25.2.63 #define SLAVE_ON_2 0x2

Definition at line 36 of file i8259.h.

6.25.2.64 #define SLAVE_ON_3 0x3

Definition at line 37 of file i8259.h.

6.25.2.65 #define SLAVE_ON_4 0x4

Definition at line 38 of file i8259.h.

6.25.2.66 #define SLAVE_ON_5 0x5

Definition at line 39 of file i8259.h.

6.25.2.67 #define SLAVE_ON_6 0x6

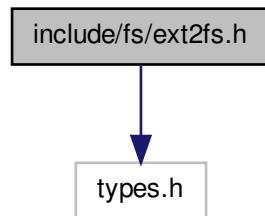
Definition at line 40 of file i8259.h.

6.25.2.68 #define SLAVE_ON_7 0x7

Definition at line 41 of file i8259.h.

6.26 include/fs/ext2fs.h File Reference

#include <types.h> Include dependency graph for ext2fs.h:



Data Structures

- struct `ext2_super_block`
- struct `ext2_dir_entry`
- struct `ext2_group_desc`
- struct `ext2_inode`

Defines

- #define `EXT2_NDIR_BLOCKS` 12
- #define `EXT2_IND_BLOCK` `EXT2_NDIR_BLOCKS`
- #define `EXT2_DIND_BLOCK` (`EXT2_IND_BLOCK` + 1)
- #define `EXT2_TIND_BLOCK` (`EXT2_DIND_BLOCK` + 1)
- #define `EXT2_N_BLOCKS` (`EXT2_TIND_BLOCK` + 1)
- #define `EXT2_INOSZ_V0` 128
- #define `EXT2_NAME_LEN` 255
- #define `EXT2_CLEAN` 1
- #define `EXT2_ERR0R` 2
- #define `EXT2_BAD_INO` 1 /* Bad blocks inode */
- #define `EXT2_ROOT_INO` 2 /* Root inode */
- #define `EXT2_ACL_IDX_INO` 3 /* ACL inode */
- #define `EXT2_ACL_DATA_INO` 4 /* ACL inode */
- #define `EXT2_BOOT_LOADER_INO` 5 /* Boot loader inode */
- #define `EXT2_UNDEL_DIR_INO` 6 /* Undelete directory inode */
- #define `EXT2_FIRST_INO` 11 /* First non reserved inode */
- #define `EXT2_PRE_02B_MAGIC` 0xEF51
- #define `EXT2_SUPER_MAGIC` 0xEF53

- #define EXT2_LINK_MAX 32000
- #define EXT2_IGNORE 1
- #define EXT2_REMOUNT_RO 2
- #define EXT2_KPANIC 3
- #define EXT2_OS_LINUX 0
- #define EXT2_OS_HURD 1
- #define EXT2_OS_MASIX 2
- #define EXT2_OS_FBSD 3
- #define EXT2_OS_LITES 4
- #define EXT2_ROOT_INODE 2 /* Root inode */
- #define I_FIFO 0x1000
- #define I_CHAR_DEVICE 0x2000
- #define I_DIRECTORY 0x4000
- #define I_BLOCK_DEVICE 0x6000
- #define I_REG_FILE 0x8000
- #define I_SYM_LINK 0xA000
- #define I_UNIX SOCK 0xC000
- #define I_OEXEC_P 0X001
- #define I_OWRITE_P 0x002
- #define I_OREAD_P 0x004
- #define I_GEXEC_P 0x008
- #define I_GWRITE_P 0x010
- #define I_GREAD_P 0x020
- #define I_UEXEC_P 0x040
- #define I_UWRITE_P 0x080
- #define I_UREAD_P 0x100
- #define I_STICKY 0x200
- #define I_SET_GID 0x400
- #define I_SET_UID 0x800
- #define I_SEC_DEL 0x00000001 /*Secure deletion*/
- #define I_KEEP_CPY 0x00000002 /*Keep a copy of data when deleted*/
- #define I_FILE_COMPRESSION 0x00000004 /*File compression*/
- #define I_SYNC_UPDATES 0x00000008 /*Synchronous updates*/
- #define I_IMMUTABLE_FILE 0x00000010 /*Immutable file*/
- #define I_APPEND_ONLY 0x00000020 /* Append only */
- #define I_NO_DUMP 0x00000040 /* File is not included in 'dump' command */
- #define I_NO_LAT 0x00000080 /* Last accessed time should not updated */
- #define I_HASH_INDEXED_DIR 0x00010000 /* Hash indexed directory */
- #define I_AFS_DIR 0x00020000 /* AFS directory */
- #define I_JOURNAL_FILE_DATA 0x00040000 /* Journal file data */
- #define I_DIR_UNKOWN 0x0
- #define I_DIR_REGULAR_F 0x1
- #define I_DIR_DIRECTORY 0x2
- #define I_DIR_CHAR_DEV 0x3
- #define I_DIR_BLOCK_DEV 0x4
- #define I_DIR_FIFO 0x5

- #define `I_DIR_SOCK` 0x6
- #define `I_DIR_SYM_LINK` 0x7
- #define `EXT2_PRELOC` 0x0001
- #define `EXT2_AFS_INO` 0x0002
- #define `EXT2_JRNL` 0x0004
- #define `EXT2_INO_EXTNDATTR` 0x0008
- #define `EXT2_FS_RESIZE` 0x0010
- #define `EXT2_DIR_HASHI` 0x0020
- #define `EXT2_COMPRESSION` 0x0001
- #define `EXT2_DIR_TYPE_F` 0x0002
- #define `EXT2_FS_REPLAY_JOR` 0x0004
- #define `EXT2_FS_JOR_DEV` 0x0008
- #define `EXT2_SPARSE` 0x0001
- #define `EXT2_64FZ` 0x0002
- #define `EXT2_DIR_BIN_TREE` 0x0004

6.26.1 Define Documentation

6.26.1.1 #define `EXT2_64FZ` 0x0002

Definition at line 114 of file ext2fs.h.

6.26.1.2 #define `EXT2_ACL_DATA_INO` 4 /* ACL inode */

Definition at line 23 of file ext2fs.h.

6.26.1.3 #define `EXT2_ACL_IDX_INO` 3 /* ACL inode */

Definition at line 22 of file ext2fs.h.

6.26.1.4 #define `EXT2_AFS_INO` 0x0002

Definition at line 102 of file ext2fs.h.

6.26.1.5 #define `EXT2_BAD_INO` 1 /* Bad blocks inode */

Definition at line 20 of file ext2fs.h.

6.26.1.6 #define `EXT2_BOOT_LOADER_INO` 5 /* Boot loader inode */

Definition at line 24 of file ext2fs.h.

6.26.1.7 #define EXT2_CLEAN 1

Definition at line 15 of file ext2fs.h.

6.26.1.8 #define EXT2_COMPRESSION 0x0001

Definition at line 108 of file ext2fs.h.

6.26.1.9 #define EXT2_DIND_BLOCK (EXT2_IND_BLOCK + 1)

Definition at line 9 of file ext2fs.h.

6.26.1.10 #define EXT2_DIR_BIN_TREE 0x0004

Definition at line 115 of file ext2fs.h.

6.26.1.11 #define EXT2_DIR_HASHI 0x0020

Definition at line 106 of file ext2fs.h.

6.26.1.12 #define EXT2_DIR_TYPE_F 0x0002

Definition at line 109 of file ext2fs.h.

6.26.1.13 #define EXT2_ERROR 2

Definition at line 16 of file ext2fs.h.

6.26.1.14 #define EXT2_FIRST_INO 11 /* First non reserved inode */

Definition at line 26 of file ext2fs.h.

6.26.1.15 #define EXT2_FS_JOR_DEV 0x0008

Definition at line 111 of file ext2fs.h.

6.26.1.16 #define EXT2_FS_REPLAY_JOR 0x0004

Definition at line 110 of file ext2fs.h.

6.26.1.17 #define EXT2_FS_RESIZE 0x0010

Definition at line 105 of file ext2fs.h.

6.26.1.18 #define EXT2_IGNORE 1

Definition at line 40 of file ext2fs.h.

6.26.1.19 #define EXT2_IND_BLOCK EXT2_NDIR_BLOCKS

Definition at line 8 of file ext2fs.h.

6.26.1.20 #define EXT2_INO_EXTNDATTR 0x0008

Definition at line 104 of file ext2fs.h.

6.26.1.21 #define EXT2_INOSZ_V0 128

Definition at line 12 of file ext2fs.h.

6.26.1.22 #define EXT2_JRNL 0x0004

Definition at line 103 of file ext2fs.h.

6.26.1.23 #define EXT2_KPANIC 3

Definition at line 42 of file ext2fs.h.

6.26.1.24 #define EXT2_LINK_MAX 32000

Definition at line 37 of file ext2fs.h.

6.26.1.25 #define EXT2_N_BLOCKS (EXT2_TIND_BLOCK + 1)

Definition at line 11 of file ext2fs.h.

6.26.1.26 #define EXT2_NAME_LEN 255

Definition at line 13 of file ext2fs.h.

6.26.1.27 #define EXT2_NDIR_BLOCKS 12

Definition at line 7 of file ext2fs.h.

6.26.1.28 #define EXT2_OS_FBSD 3

Definition at line 47 of file ext2fs.h.

6.26.1.29 #define EXT2_OS_HURD 1

Definition at line 45 of file ext2fs.h.

6.26.1.30 #define EXT2_OS_LINUX 0

Definition at line 44 of file ext2fs.h.

6.26.1.31 #define EXT2_OS_LITES 4

Definition at line 48 of file ext2fs.h.

6.26.1.32 #define EXT2_OS_MASIX 2

Definition at line 46 of file ext2fs.h.

6.26.1.33 #define EXT2_PRE_02B_MAGIC 0xEF51

Definition at line 31 of file ext2fs.h.

6.26.1.34 #define EXT2_PRELOC 0x0001

Features

Definition at line 101 of file ext2fs.h.

6.26.1.35 #define EXT2_REMOUNT_RO 2

Definition at line 41 of file ext2fs.h.

6.26.1.36 #define EXT2_ROOT_INO 2 /* Root inode */

Definition at line 21 of file ext2fs.h.

6.26.1.37 `#define EXT2_ROOT_INODE 2 /* Root inode */`

Definition at line 53 of file ext2fs.h.

6.26.1.38 `#define EXT2_SPARSE 0x0001`

Definition at line 113 of file ext2fs.h.

6.26.1.39 `#define EXT2_SUPER_MAGIC 0xEF53`

Definition at line 32 of file ext2fs.h.

6.26.1.40 `#define EXT2_TIND_BLOCK (EXT2_DIND_BLOCK + 1)`

Definition at line 10 of file ext2fs.h.

6.26.1.41 `#define EXT2_UNDEL_DIR_INO 6 /* Undelete directory inode */`

Definition at line 25 of file ext2fs.h.

6.26.1.42 `#define I_AFS_DIR 0x00020000 /* AFS directory */`

Definition at line 87 of file ext2fs.h.

6.26.1.43 `#define I_APPEND_ONLY 0x00000020 /* Append only */`

Definition at line 83 of file ext2fs.h.

6.26.1.44 `#define I_BLOCK_DEVICE 0x6000`

Definition at line 58 of file ext2fs.h.

6.26.1.45 `#define I_CHAR_DEVICE 0x2000`

Definition at line 56 of file ext2fs.h.

6.26.1.46 `#define I_DIR_BLOCK_DEV 0x4`

Definition at line 95 of file ext2fs.h.

6.26.1.47 #define I_DIR_CHAR_DEV 0x3

Definition at line 94 of file ext2fs.h.

6.26.1.48 #define I_DIR_DIRECTORY 0x2

Definition at line 93 of file ext2fs.h.

6.26.1.49 #define I_DIR_FIFO 0x5

Definition at line 96 of file ext2fs.h.

6.26.1.50 #define I_DIR_REGULAR_F 0x1

Definition at line 92 of file ext2fs.h.

6.26.1.51 #define I_DIR_SOCK 0x6

Definition at line 97 of file ext2fs.h.

6.26.1.52 #define I_DIR_SYM_LINK 0x7

Definition at line 98 of file ext2fs.h.

6.26.1.53 #define I_DIR_UNKOWN 0x0

Directory Entry Types

Definition at line 91 of file ext2fs.h.

6.26.1.54 #define I_DIRECTORY 0x4000

Definition at line 57 of file ext2fs.h.

6.26.1.55 #define I_FIFO 0x1000

Definition at line 55 of file ext2fs.h.

6.26.1.56 #define I_FILE_COMPRESSION 0x00000004 /*File compression*/

Definition at line 80 of file ext2fs.h.

6.26.1.57 #define I_GEXEC_P 0x008

Definition at line 67 of file ext2fs.h.

6.26.1.58 #define I_GREAD_P 0x020

Definition at line 69 of file ext2fs.h.

6.26.1.59 #define I_GWRITE_P 0x010

Definition at line 68 of file ext2fs.h.

6.26.1.60 #define I_HASH_INDEXED_DIR 0x00010000 /* Hash indexed directory */

Definition at line 86 of file ext2fs.h.

6.26.1.61 #define I_IMMUTABLE_FILE 0x00000010 /*Immutable file*/

Definition at line 82 of file ext2fs.h.

6.26.1.62 #define I_JOURNAL_FILE_DATA 0x00040000 /* Journal file data */

Definition at line 88 of file ext2fs.h.

6.26.1.63 #define I_KEEP_CPY 0x00000002 /*Keep a copy of data when deleted*/

Definition at line 79 of file ext2fs.h.

6.26.1.64 #define I_NO_DUMP 0x00000040 /* File is not included in 'dump' command */

Definition at line 84 of file ext2fs.h.

6.26.1.65 #define I_NO_LAT 0x00000080 /* Last accessed time should not updated */

Definition at line 85 of file ext2fs.h.

6.26.1.66 #define I_OEXEC_P 0X001

Definition at line 64 of file ext2fs.h.

6.26.1.67 #define I_OREAD_P 0x004

Definition at line 66 of file ext2fs.h.

6.26.1.68 #define I_OWRITE_P 0x002

Definition at line 65 of file ext2fs.h.

6.26.1.69 #define I_REG_FILE 0x8000

Definition at line 59 of file ext2fs.h.

6.26.1.70 #define I_SEC_DEL 0x00000001 /*Secure deletion*/

Inode flags

Definition at line 78 of file ext2fs.h.

6.26.1.71 #define I_SET_GID 0x400

Definition at line 74 of file ext2fs.h.

6.26.1.72 #define I_SET_UID 0x800

Definition at line 75 of file ext2fs.h.

6.26.1.73 #define I_STICKY 0x200

Definition at line 73 of file ext2fs.h.

6.26.1.74 #define I_SYM_LINK 0xA000

Definition at line 60 of file ext2fs.h.

6.26.1.75 #define I_SYNC_UPDATES 0x00000008 /*Synchronous updates*/

Definition at line 81 of file ext2fs.h.

6.26.1.76 #define I_UEXEC_P 0x040

Definition at line 70 of file ext2fs.h.

6.26.1.77 #define I_UNIX_SOCK 0xC000

Definition at line 61 of file ext2fs.h.

6.26.1.78 #define I_UREAD_P 0x100

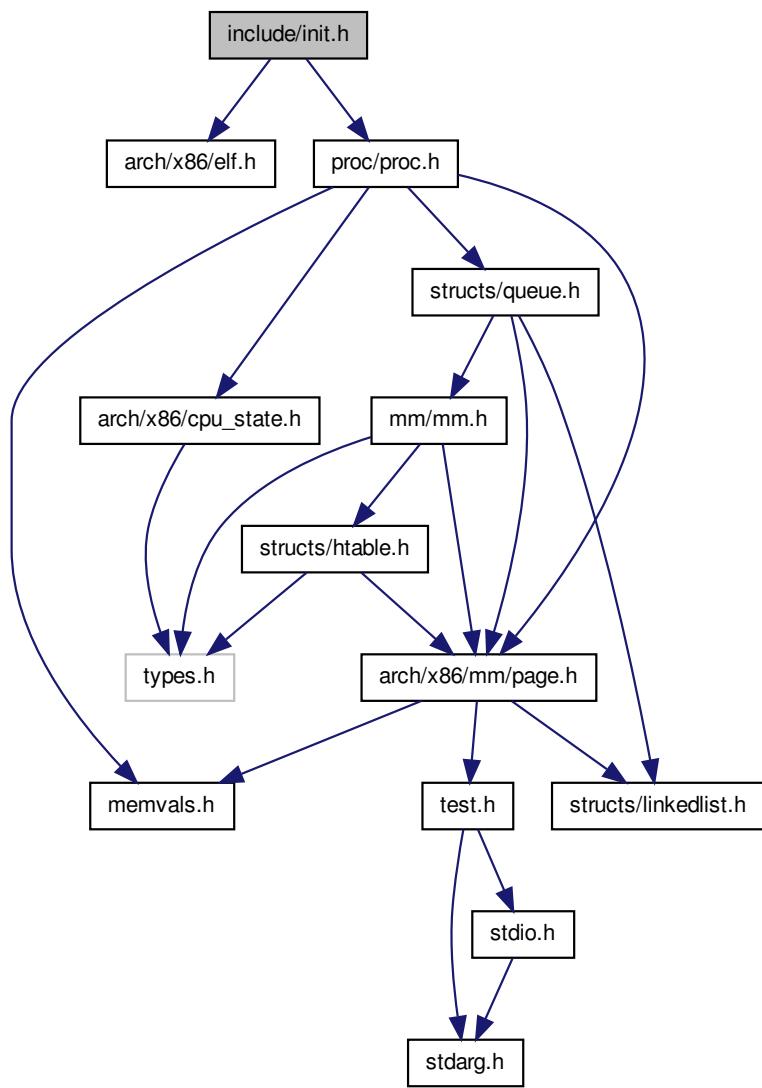
Definition at line 72 of file ext2fs.h.

6.26.1.79 #define I_UWRITE_P 0x080

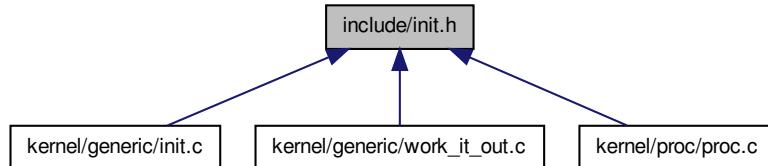
Definition at line 71 of file ext2fs.h.

6.27 include/init.h File Reference

```
#include <arch/x86/elf.h> #include <proc/proc.h> Include de-
pendency graph for init.h:
```



This graph shows which files directly or indirectly include this file:



Defines

- #define **ELFHDR** ((struct **elfhdr** *) 0xA0000000)
- #define **SECTOR** 512

Functions

- void **Init_userspace** (**proc_t** *)

6.27.1 Define Documentation

6.27.1.1 #define ELFHDR ((struct elfhdr *) 0xA0000000)

Definition at line 5 of file init.h.

6.27.1.2 #define SECTOR 512

Definition at line 6 of file init.h.

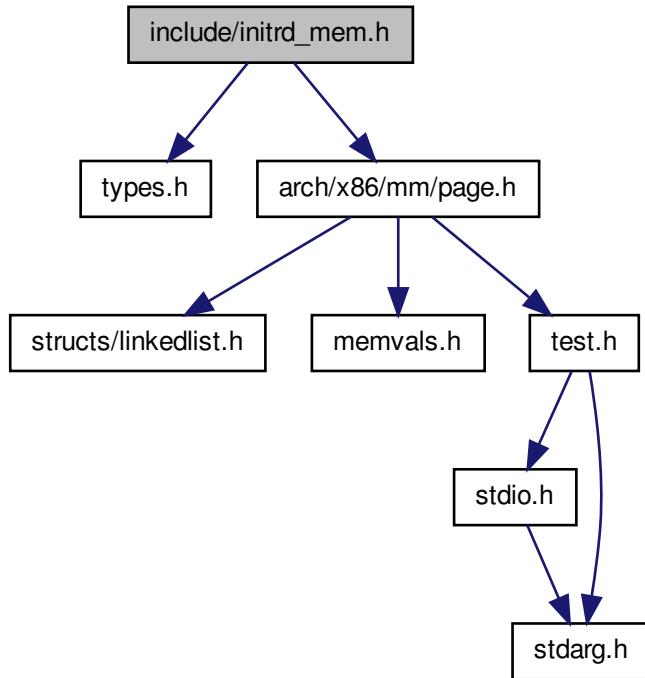
6.27.2 Function Documentation

6.27.2.1 void **Init_userspace** (**proc_t** *)

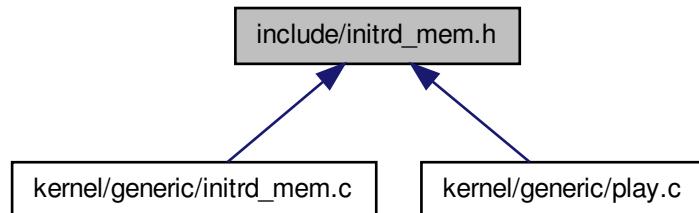
Definition at line 16 of file init.c.

6.28 include/initrd_mem.h File Reference

```
#include <types.h> #include <arch/x86/mm/page.h> Include de-
pendency graph for initrd_mem.h:
```

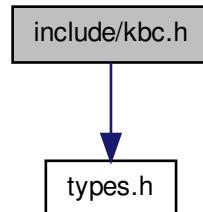


This graph shows which files directly or indirectly include this file:

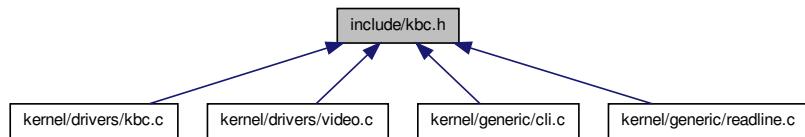


6.29 include/kbc.h File Reference

#include <types.h> Include dependency graph for kbc.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define KBC_STATUSPORT 0x64
- #define KBC_DATAPORT 0x60
- #define KBC_DATAIN 0x01 /* New Data in buffer */
- #define KBC_FULLBUF 0x02 /* Buffer is full */
- #define KBC_REBOOT 0x04 /* soft reboot */
- #define KBC_COMMAND 0x08 /* data in output register is a command */
- #define KBC_SECLOCK 0x10 /* Security lock engaged */
- #define KBC_TTIMEOUT 0x20 /* transmission timeout error */
- #define KBC_RTIMEOUT 0x40 /* receive timeout error */
- #define KBC_PARITY 0x80 /* Parity error */
- #define KBC_READRAM 0x20 /* Read byte 0 from the internal RAM */
- #define KBC_READLOW 0x21 /* Read byte specified in low 5 bits of the command in 804x's internal RAM */
- #define KBC_WITERAM 0x60 /* write the data to the address specified in the 5 lower bits of command */
- #define KBC_DISMOUSE 0xA7 /* disable mouse/auxillary port */
- #define KBC_ENMOUSE 0xA8 /* enable mouse/auxillary port */
- #define KBC_TSTMOUSE 0xA9 /* test mouse/auxillary port */
- #define KBC_SELFTEST 0xAA /* initiate the self-test */
- #define KBC_INTRTEST
- #define KBC_DIGDUMP 0xAC /* dump the content of 804x's RAM, output port, input port, status word */
- #define KBC_DISKBD 0xAD /* disable keyboard */
- #define KBC_ENBKBD 0xAE /* enable keyboard */
- #define KBC_READVER 0xAF /* Read keyboard version */
- #define KBC_INPREAD 0xC0 /* Read input port */
- #define KBC_OUTREAD 0xD0 /* Read Output port */
- #define KBC_OUTWRTE 0xD1 /* write output port ,next byte will be written to 804x */
- #define KBC_OUTKBD 0xD2 /* Echo keyboard output buffer */
- #define KBC_OUTAUX 0xD3 /* Echo pointing device output buffer */
- #define KBC_AUXWRITE 0xD4 /* Write to pointing device output buffer */
- #define KBC_DISA20 0xDD /* Disable the A20 gate */
- #define KBC_ENBA20 0xDF /* Enable the A20 gate */
- #define KBC_PULSE0 0xFE /* Pulse output bit 0 */
- #define KBC_PULSE1 0xFD /* ~~~~~~ 1 */
- #define KBC_PULSE2 0xFB /* ~~~~~~ 2 */
- #define KBC_PULSE3 0xF7 /* ~~~~~~ 3 */
- #define KBC_RESET 0xFF /* RESET the mouse and the keyboard */
- #define KBC_RESEND 0xFE /* Resend the last byte */
- #define KBC_DEFAULT 0xF6 /* Set the keyboard to default parameters */
- #define KBC_DISABLE 0xF5 /* disables key scanning, yet set the default parameters */
- #define KBC_ENABLE 0xF4 /* Enable key scanning */
- #define KBC_TYPEMATIC 0xF3 /* Set typematic rate/delay */

- #define **KBC_READID** 0xF2 /* Read Keyboard ID */
- #define **KBC_SETSCAN** 0xFO /* Set/Get scancodes set */
- #define **KBC_ECHODIG** 0xEE /* request a diagnostic echo from the keyboard */
- #define **KBC_INDICATOR** 0xED /* Set mode indicators */
- #define **KEY_HOME** 0xE0
- #define **KEY_END** 0xE1
- #define **KEY_UP** 0xE2
- #define **KEY_DN** 0xE3
- #define **KEY_LF** 0xE4
- #define **KEY_RT** 0xE5
- #define **KEY_PGUP** 0xE6
- #define **KEY_PGDN** 0xE7
- #define **KEY_INS** 0xE8
- #define **KEY_DEL** 0xE9

Functions

- int **kbc_data** (void)
keyboard input controller driver
- void **kbc_interrupt** (void)
calls console interrupt which is used in character read.

6.29.1 Define Documentation

6.29.1.1 #define KBC_AUXWRITE 0xD4 /* Write to pointing device output buffer */

Definition at line 40 of file kbc.h.

6.29.1.2 #define KBC_COMMAND 0x08 /* data in output register is a command */

Definition at line 16 of file kbc.h.

6.29.1.3 #define KBC_DATAIN 0x01 /* New Data in buffer */

Definition at line 13 of file kbc.h.

6.29.1.4 #define KBC_DATAPORT 0x60

Definition at line 11 of file kbc.h.

6.29.1.5 #define KBC_DEFAULT 0xF6 /* Set the keyboard to default parameters */

Definition at line 52 of file kbc.h.

6.29.1.6 #define KBC_DIGDUMP 0xAC /* dump the content of 804x's RAM, output port, input port, status word */

Definition at line 31 of file kbc.h.

6.29.1.7 #define KBC_DISA20 0xDD /* Disable the A20 gate */

Definition at line 41 of file kbc.h.

6.29.1.8 #define KBC_DISABLE 0xF5 /* disables key scanning, yet set the default parameters */

Definition at line 53 of file kbc.h.

6.29.1.9 #define KBC_DISKBD 0xAD /* disable keyboard */

Definition at line 32 of file kbc.h.

6.29.1.10 #define KBC_DISMOUSE 0xA7 /* disable mouse/auxillary port */

Definition at line 26 of file kbc.h.

6.29.1.11 #define KBC_ECHODIG 0xEE /* request a diagnostic echo from the keyboard */

Definition at line 58 of file kbc.h.

6.29.1.12 #define KBC_ENABLE 0xF4 /* Enable key scanning */

Definition at line 54 of file kbc.h.

6.29.1.13 #define KBC_ENBA20 0xDF /* Enable the A20 gate */

Definition at line 42 of file kbc.h.

6.29.1.14 #define KBC_ENBKBD 0xAE /* enable keyboard */

Definition at line 33 of file kbc.h.

6.29.1.15 #define KBC_ENMOUSE 0xA8 /* enable mouse/auxillary port */

Definition at line 27 of file kbc.h.

6.29.1.16 #define KBC_FULLBUF 0x02 /* Buffer is full */

Definition at line 14 of file kbc.h.

6.29.1.17 #define KBC_INDICATOR 0xED /* Set mode indicators */

Definition at line 59 of file kbc.h.

6.29.1.18 #define KBC_INPREAD 0xC0 /* Read input port */

Definition at line 35 of file kbc.h.

6.29.1.19 #define KBC_INTRTEST

Value:

0xAB

Definition at line 30 of file kbc.h.

6.29.1.20 #define KBC_OUTAUX 0xD3 /* Echo pointing device output buffer */

Definition at line 39 of file kbc.h.

6.29.1.21 #define KBC_OUTKBD 0xD2 /* Echo keyboard output buffer */

Definition at line 38 of file kbc.h.

6.29.1.22 #define KBC_OUTREAD 0xD0 /* Read Output port */

Definition at line 36 of file kbc.h.

6.29.1.23 #define KBC_OUTWRTE 0xD1 /* write output port ,next byte will be written to 804x */

Definition at line 37 of file kbc.h.

6.29.1.24 #define KBC_PARITY 0x80 /* Parity error */

Definition at line 20 of file kbc.h.

6.29.1.25 #define KBC_PULSE0 0xFE /* Pulse output bit 0 */

Definition at line 43 of file kbc.h.

6.29.1.26 #define KBC_PULSE1 0xFD /* ~~~~~~1~~~~~ */

Definition at line 44 of file kbc.h.

6.29.1.27 #define KBC_PULSE2 0xFB /* ~~~~~~2~~~~~ */

Definition at line 45 of file kbc.h.

6.29.1.28 #define KBC_PULSE3 0xF7 /* ~~~~~~3~~~~~ */

Definition at line 46 of file kbc.h.

6.29.1.29 #define KBC_READID 0xF2 /* Read Keyboard ID */

Definition at line 56 of file kbc.h.

6.29.1.30 #define KBC_READLOW 0x21 /* Read byte specified in low 5 bits of the command
in 804x's internal RAM */

Definition at line 24 of file kbc.h.

6.29.1.31 #define KBC_READRAM 0x20 /* Read byte 0 from the internal RAM */

Definition at line 23 of file kbc.h.

6.29.1.32 #define KBC_READVER 0xAF /* Read keyboard version */

Definition at line 34 of file kbc.h.

6.29.1.33 #define KBC_REBOOT 0x04 /* soft reboot */

Definition at line 15 of file kbc.h.

6.29.1.34 #define KBC resend 0xFE /* Resend the last byte */

Definition at line 51 of file kbc.h.

6.29.1.35 `#define KBC_RESET 0xFF /* RESET the mouse and the keyboard */`

Definition at line 50 of file kbc.h.

6.29.1.36 `#define KBC_RTIMEOUT 0x40 /* receive timeout error */`

Definition at line 19 of file kbc.h.

6.29.1.37 `#define KBC_SECLOCK 0x10 /* Security lock engaged */`

Definition at line 17 of file kbc.h.

6.29.1.38 `#define KBC_SELFTEST 0xAA /* initiate the self-test */`

Definition at line 29 of file kbc.h.

6.29.1.39 `#define KBC_SETSCAN 0xF0 /* Set/Get scancodes set */`

Definition at line 57 of file kbc.h.

6.29.1.40 `#define KBC_STATUSPORT 0x64`

Definition at line 10 of file kbc.h.

6.29.1.41 `#define KBC_TSTMOUSE 0xA9 /* test mouse/auxillary port */`

Definition at line 28 of file kbc.h.

6.29.1.42 `#define KBC_TTIMEOUT 0x20 /* transmission timeout error */`

Definition at line 18 of file kbc.h.

6.29.1.43 `#define KBC_TYPEMATIC 0xF3 /* Set typematic rate/delay */`

Definition at line 55 of file kbc.h.

6.29.1.44 `#define KBC_WRITERAM 0x60 /* write the data to the address specified in the 5 lower bits of command */`

Definition at line 25 of file kbc.h.

6.29.1.45 #define KEY_DEL 0xE9

Definition at line 72 of file kbc.h.

6.29.1.46 #define KEY_DN 0xE3

Definition at line 66 of file kbc.h.

6.29.1.47 #define KEY_END 0xE1

Definition at line 64 of file kbc.h.

6.29.1.48 #define KEY_HOME 0xE0

Definition at line 63 of file kbc.h.

6.29.1.49 #define KEY_INS 0xE8

Definition at line 71 of file kbc.h.

6.29.1.50 #define KEY_LF 0xE4

Definition at line 67 of file kbc.h.

6.29.1.51 #define KEY_PGDN 0xE7

Definition at line 70 of file kbc.h.

6.29.1.52 #define KEY_PGUP 0xE6

Definition at line 69 of file kbc.h.

6.29.1.53 #define KEY_RT 0xE5

Definition at line 68 of file kbc.h.

6.29.1.54 #define KEY_UP 0xE2

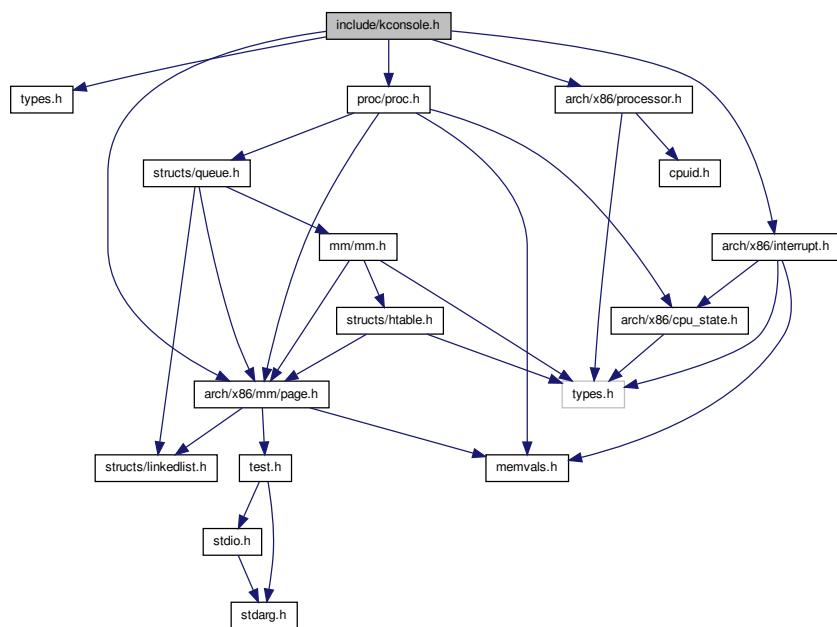
Definition at line 65 of file kbc.h.

6.30 include/kconsole.h File Reference

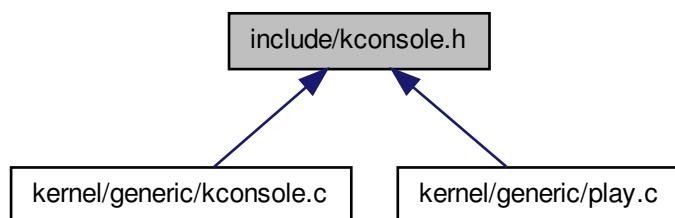
Kernel console for debugging ease.

```
#include <types.h> #include <proc/proc.h> #include <arch/x86/processor.h>
#include <arch/x86/interrupt.h> #include <arch/x86/mm/page.h>

Include dependency graph for kconsole.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `kcommand`

Defines

- `#define KCONSOLE_MAX_CMDS 30`

Typedefs

- `typedef struct kcommand kcommand_t`

Functions

- `void kconsole_init (void)`
- `void kconsole (void)`
- `void kcommand_register (kcommand_t *)`
- `kcommand_t * kcommand_match (const char *)`
- `void kscheduler_info (void)`
- `void kcpu_info (void)`
- `void kconsole_help (void)`

Variables

- `kcommand_t * kconsole_commands`

6.30.1 Detailed Description

Kernel console for debugging ease.

Author

Saad Talaat

Date

19 November, 2012

Definition in file `kconsole.h`.

6.30.2 Define Documentation

6.30.2.1 `#define KCONSOLE_MAX_CMDS 30`

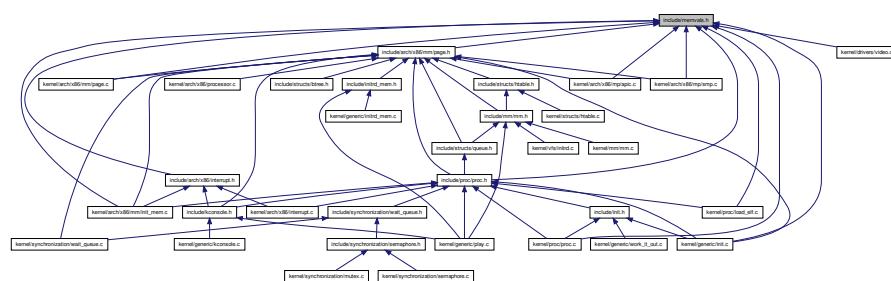
Definition at line 16 of file `kconsole.h`.

6.30.3 Typedef Documentation

6.30.3.1 `typedef struct kcommand kcommand_t`

6.31 include/memvals.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct Segdesc
 - struct Gdtdesc

Defines

- #define SEGACS_RW 0x2 /* Read and write flag Read for code segments and write for data segments */
 - #define SEGACS_X 0x8 /* The Executable flag */
 - #define SEGACS_D
 - #define SEGACS_USR 0x60
 - #define SEG_NULL (struct Segdesc){0,0,0,0,0,0,0}
 - #define SEGMENT(limit, base, access)
 - #define EXTMEM 0x100000
 - #define PAGESZ 0x1000
 - #define PAGELG 0xC
 - #define PAGECNT 0x400
 - #define PAGETSZ (PAGESZ*PAGECNT)
 - #define KERNEL_ADDR 0xF0000000
 - #define KERNEL_STACK PAGESZ*8
 - #define VIRTPGT (KERNEL_ADDR - PAGETSZ)
 - #define KERNEL_STACK_TOP VIRTPGT
 - #define USEREND (VIRTPGT - PAGETSZ)
 - #define USERVIRTPGT (USEREND - PAGETSZ)
 - #define USERPAGES (USERVIRTPGT - PAGETSZ)

- #define PROC_LIST (USERPAGES - PAGETSZ)
- #define USERSTART (PROC_LIST)
- #define USERSTACK_TOP (USERSTART - 2*PAGESZ)
- #define USERSPACESTART 0XEE800000
- #define PA(va)
- #define IOPHYMEM 0x0A0000
- #define KB 0x400
- #define MB KB*KB
- #define GB MB*KB

Functions

- struct Gdtdesc __attribute__ ((packed))

Variables

- uint16_t size
- uint32_t base

6.31.1 Define Documentation

6.31.1.1 #define EXTMEM 0x100000

Definition at line 83 of file memvals.h.

6.31.1.2 #define GB MB*KB

Definition at line 120 of file memvals.h.

6.31.1.3 #define IOPHYMEM 0x0A0000

Paging

Definition at line 114 of file memvals.h.

6.31.1.4 #define KB 0x400

Definition at line 118 of file memvals.h.

6.31.1.5 #define KERNEL_ADDR 0xF0000000

Definition at line 88 of file memvals.h.

6.31.1.6 #define KERNEL_STACK_PAGESZ*8

Definition at line 89 of file memvals.h.

6.31.1.7 #define KERNEL_STACK_TOP_VIRTPGT

Definition at line 91 of file memvals.h.

6.31.1.8 #define MB KB*KB

Definition at line 119 of file memvals.h.

6.31.1.9 #define PA(va)

Value:

```
((\n    uint32_t pa = (uint32_t) va; \n    pa-KERNEL_ADDR; \n))
```

Definition at line 103 of file memvals.h.

6.31.1.10 #define PAGECNT 0x400

Definition at line 86 of file memvals.h.

6.31.1.11 #define PAGELG 0xC

Definition at line 85 of file memvals.h.

6.31.1.12 #define PAGESZ 0x1000

Definition at line 84 of file memvals.h.

6.31.1.13 #define PAGETSZ (PAGESZ*PAGECNT)

Definition at line 87 of file memvals.h.

6.31.1.14 #define PROC_LIST (USERPAGES - PAGETSZ)

Definition at line 96 of file memvals.h.

6.31.1.15 #define SEG_NULL (struct Segdesc){0,0,0,0,0,0}

Definition at line 67 of file memvals.h.

6.31.1.16 #define SEGACS_D

Value:

```
0x4      /* growing down segment for data segments and means that this segments
           can be                                         Executed from the user applications
           "Low Privilege" */
```

Definition at line 54 of file memvals.h.

6.31.1.17 #define SEGACS_RW 0x2 /* Read and write flag Read for code segments and write
 for data segments */

Definition at line 52 of file memvals.h.

6.31.1.18 #define SEGACS_USR 0x60

Definition at line 55 of file memvals.h.

6.31.1.19 #define SEGACS_X 0x8 /* The Executable flag */

Definition at line 53 of file memvals.h.

6.31.1.20 #define SEGMENT(*limit*, *base*, *access*)

Value:

```
(struct Segdesc) \
{\
    ((limit>>12) & 0xFFFF, \
     (base) & 0xFFFF, \
     ((base)>>16) & 0xFF, \
     (access) | 0x90, \
     ((limit) >> 28), \
     0xC, \
     (base)>>24 \
}
```

Definition at line 68 of file memvals.h.

6.31.1.21 #define USEREND (VIRTPGT - PAGETSZ)

Definition at line 93 of file memvals.h.

6.31.1.22 `#define USERPAGES (USERVIRTPGT - PAGETSZ)`

Definition at line 95 of file memvals.h.

6.31.1.23 `#define USERSPACESTART 0XEE800000`

Definition at line 101 of file memvals.h.

6.31.1.24 `#define USERSTACK_TOP (USERSTART - 2*PAGESZ)`

Definition at line 98 of file memvals.h.

6.31.1.25 `#define USERSTART (PROC_LIST)`

Definition at line 97 of file memvals.h.

6.31.1.26 `#define USERVIRTPGT (USEREND - PAGETSZ)`

Definition at line 94 of file memvals.h.

6.31.1.27 `#define VIRTPGT (KERNEL_ADDR - PAGETSZ)`

Definition at line 90 of file memvals.h.

6.31.2 Function Documentation

6.31.2.1 `struct Gdtdesc __attribute__ ((packed))`

6.31.3 Variable Documentation

6.31.3.1 `uint32_t base`

Definition at line 52 of file memvals.h.

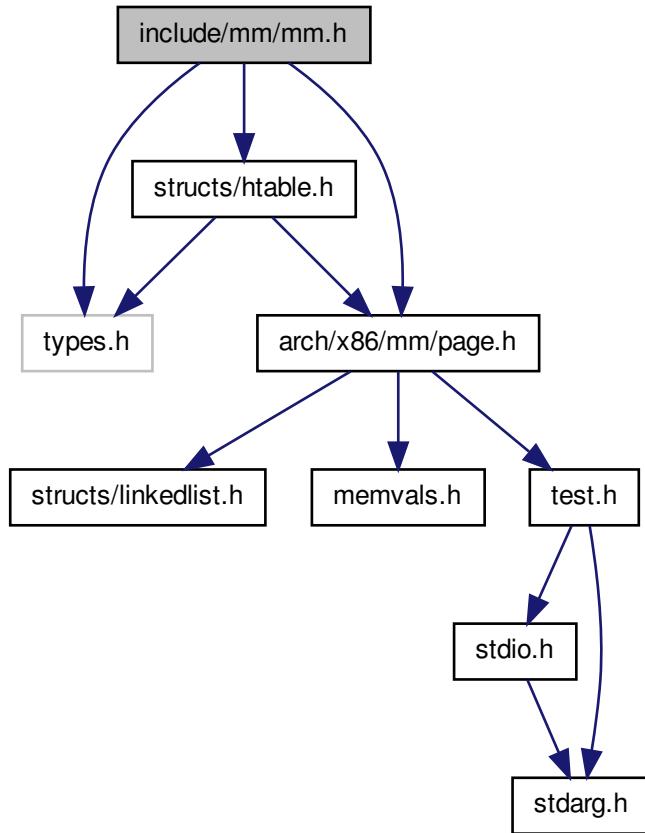
6.31.3.2 `uint16_t size`

Definition at line 51 of file memvals.h.

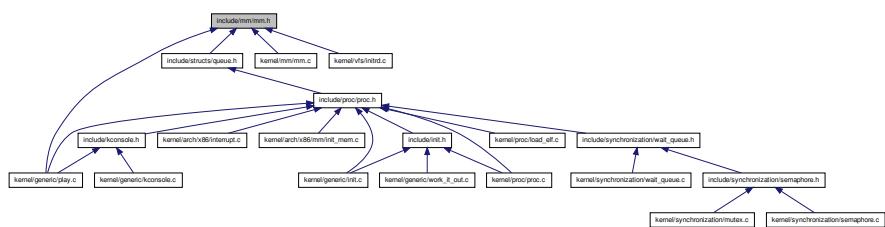
6.32 include/mm/mm.h File Reference

Memory management.

```
#include <types.h> #include <arch/x86/mm/page.h> #include
<structs/htable.h> Include dependency graph for mm.h:
```



This graph shows which files directly or indirectly include this file:



Defines

- `#define _CATERNEL_MM_MM_H_`

Functions

- `void mm_init (void)`
memory heap manager initialization function
- `void * kmalloc (uint32_t)`

6.32.1 Detailed Description

Memory management.

Author

Saad Talaat

Generic Memory management unit is responsible for memory allocation.

Definition in file [mm.h](#).

6.32.2 Define Documentation

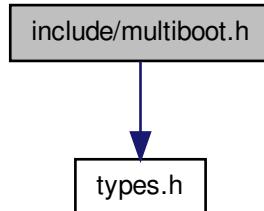
6.32.2.1 `#define _CATERNEL_MM_MM_H_`

Definition at line 11 of file mm.h.

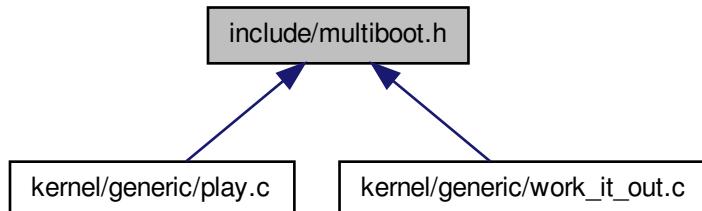
6.33 include/multiboot.h File Reference

Multiboot specifications.

```
#include <types.h> Include dependency graph for multiboot.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [__attribute__](#)
Task State segment.
- struct [__attribute__](#)
- struct [__attribute__](#)

Task State segment.

- struct `__attribute__`

Task State segment.

Defines

- #define `MULTIBOOT_SEARCH` 8192
- #define `MULTIBOOT_HEADER_MAGIC` 0x1BADB002
- #define `MULTIBOOT_BOOTLOADER_MAGIC` 0x2BADB002
- #define `MULTIBOOT_UNSUPPORTED` 0x0000fffc
- #define `MULTIBOOT_AOUTF_FLAG_KLUDGE` 0x00010000
- #define `MULTIBOOT_FLAG_MEM` 0x001
- #define `MULTIBOOT_FLAG_DEVICE` 0x002
- #define `MULTIBOOT_FLAG_CMDLINE` 0x004
- #define `MULTIBOOT_FLAG_MODS` 0x008
- #define `MULTIBOOT_FLAG_AOUT` 0x010
- #define `MULTIBOOT_FLAG_ELF` 0x020
- #define `MULTIBOOT_FLAG_MMAP` 0x040
- #define `MULTIBOOT_FLAG_CONFIG` 0x080
- #define `MULTIBOOT_FLAG_LOADER` 0x100
- #define `MULTIBOOT_FLAG_APM` 0x200
- #define `MULTIBOOT_FLAG_VBE` 0x400
- #define `MULTIBOOT_MOD_ALIGN` 0x00001000
- #define `MULTIBOOT_INFO_ALIGN` 0x00000004
- #define `MULTIBOOT_MEMORY_AVAILABLE` 1
- #define `MULTIBOOT_MEMORY_RESERVED` 2

Variables

- typedef `__attribute__`

6.33.1 Detailed Description

Multiboot specifications.

Author

Menna Essa

Date

19 November, 2012

Definition in file `multiboot.h`.

6.33.2 Define Documentation

6.33.2.1 `#define MULTIBOOT_AOUTF_FLAG_KLUDGE 0x00010000`

Definition at line 17 of file multiboot.h.

6.33.2.2 `#define MULTIBOOT_BOOTLOADER_MAGIC 0x2BADB002`

Definition at line 15 of file multiboot.h.

6.33.2.3 `#define MULTIBOOT_FLAG_AOUT 0x010`

Definition at line 22 of file multiboot.h.

6.33.2.4 `#define MULTIBOOT_FLAG_APM 0x200`

Definition at line 27 of file multiboot.h.

6.33.2.5 `#define MULTIBOOT_FLAG_CMDLINE 0x004`

Definition at line 20 of file multiboot.h.

6.33.2.6 `#define MULTIBOOT_FLAG_CONFIG 0x080`

Definition at line 25 of file multiboot.h.

6.33.2.7 `#define MULTIBOOT_FLAG_DEVICE 0x002`

Definition at line 19 of file multiboot.h.

6.33.2.8 `#define MULTIBOOT_FLAG_ELF 0x020`

Definition at line 23 of file multiboot.h.

6.33.2.9 `#define MULTIBOOT_FLAG_LOADER 0x100`

Definition at line 26 of file multiboot.h.

6.33.2.10 `#define MULTIBOOT_FLAG_MEM 0x001`

Definition at line 18 of file multiboot.h.

6.33.2.11 #define MULTIBOOT_FLAG_MMAP 0x040

Definition at line 24 of file multiboot.h.

6.33.2.12 #define MULTIBOOT_FLAG_MODS 0x008

Definition at line 21 of file multiboot.h.

6.33.2.13 #define MULTIBOOT_FLAG_VBE 0x400

Definition at line 28 of file multiboot.h.

6.33.2.14 #define MULTIBOOT_HEADER_MAGIC 0x1BADB002

Definition at line 14 of file multiboot.h.

6.33.2.15 #define MULTIBOOT_INFO_ALIGN 0x00000004

Definition at line 34 of file multiboot.h.

6.33.2.16 #define MULTIBOOT_MEMORY_AVAILABLE 1

Definition at line 127 of file multiboot.h.

6.33.2.17 #define MULTIBOOT_MEMORY_RESERVED 2

Definition at line 128 of file multiboot.h.

6.33.2.18 #define MULTIBOOT_MOD_ALIGN 0x00001000

Definition at line 32 of file multiboot.h.

6.33.2.19 #define MULTIBOOT_SEARCH 8192

Definition at line 13 of file multiboot.h.

6.33.2.20 #define MULTIBOOT_UNSUPPORTED 0x0000fffc

Definition at line 16 of file multiboot.h.

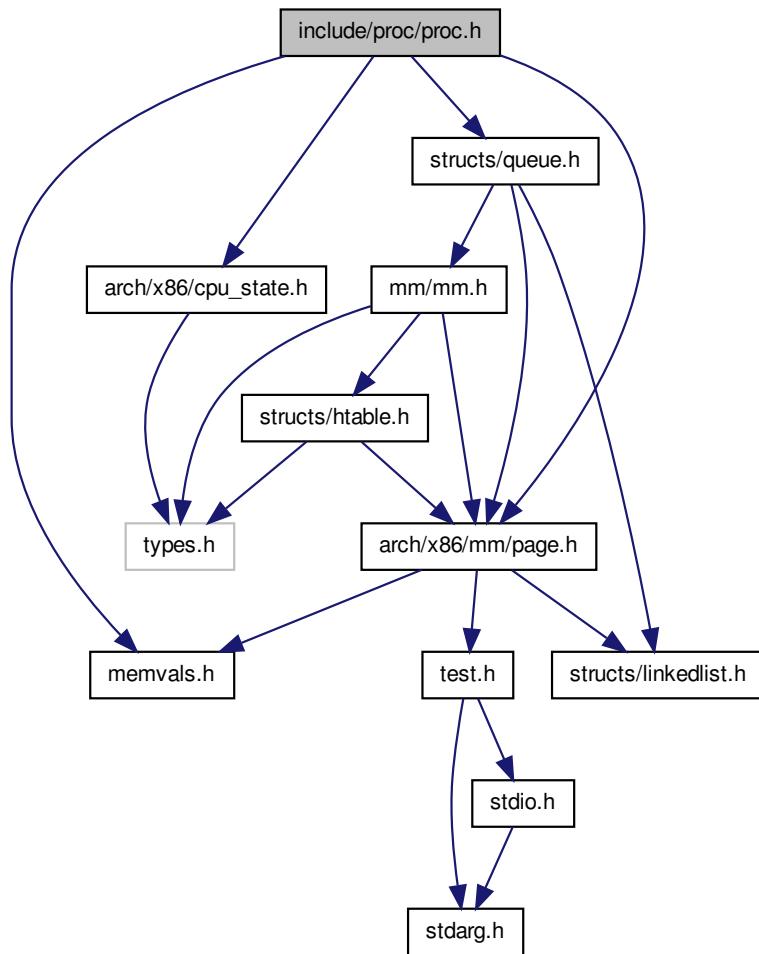
6.33.3 Variable Documentation

6.33.3.1 `typedef __attribute__`

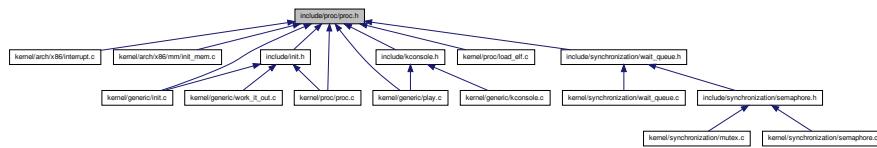
6.34 include/proc/proc.h File Reference

Process manager header file.

```
#include <arch/x86/cpu_state.h>      #include <memvals.h> x
#include <structs/queue.h>    #include <arch/x86/mm/page.h>
h> Include dependency graph for proc.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `proc`

Defines

- #define `MAX_PROCS` 256
- #define `MAX_PROC_NAME` 128
- #define `RUNNABLE` 0
- #define `PROC_EMPTY` 0x1
- #define `NON_RUNNABLE` 0x2
- #define `BLOCKED` 0x3
- #define `PROC_TABLE_SIZE` ROUND_UP(`MAX_PROCS` * sizeof(struct `proc`), - `PAGESZ`)

Typedefs

- typedef struct `proc` `proc_t`

Functions

- `LIST_HEAD` (`Proc_List`, `proc`)
- `LIFO_HEAD` (`Proc_Lifo`, `proc`)
- void `init_proc_table` (void)
- void `switch_address_space` (`proc_t` *)
switches between the kernel and a given proc
- `uint32_t proc_setup` (`proc_t` **)
- `uint32_t proc_setup_mem` (`proc_t` *)
- void `test_lifo` (void)
Tests the LIFO queues data structure functionality.
- void `test_fifo` (void)
Tests the FIFO queue data structure.
- void `sched_init` (void)
- void `schedule` (void)

main scheduling function

- void [proc_printinfo](#) (void)
Initialize the proc array.
- void [proc_ready](#) (proc_t *)

Variables

- [proc_t * proc_table](#)
- struct Proc_Lifo [running_procs](#)

6.34.1 Detailed Description

Process manager header file.

Author

Saad Talaat

Date

4 Oct, 2012

Definition in file [proc.h](#).

6.34.2 Define Documentation

6.34.2.1 #define BLOCKED 0x3

Definition at line 22 of file proc.h.

6.34.2.2 #define MAX_PROC_NAME 128

Definition at line 17 of file proc.h.

6.34.2.3 #define MAX_PROCS 256

Definition at line 16 of file proc.h.

6.34.2.4 #define NON_RUNNABLE 0x2

Definition at line 21 of file proc.h.

6.34.2.5 #define PROC_EMPTY 0x1

Definition at line 20 of file proc.h.

6.34.2.6 #define PROC_TABLE_SIZE ROUND_UP(MAX_PROCS * sizeof(struct proc),
PAGESZ)

Definition at line 47 of file proc.h.

6.34.2.7 #define RUNNABLE 0

Definition at line 19 of file proc.h.

6.34.3 Typedef Documentation

6.34.3.1 typedef struct proc proc_t

6.34.4 Function Documentation

6.34.4.1 LIFO_HEAD(Proc_Lifo , proc)

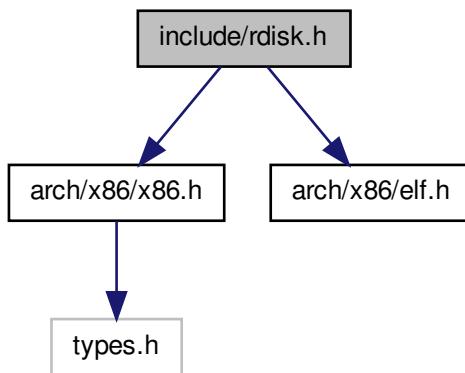
6.34.4.2 LIST_HEAD(Proc_List , proc)

6.34.4.3 uint32_t proc_setup(proc_t**)

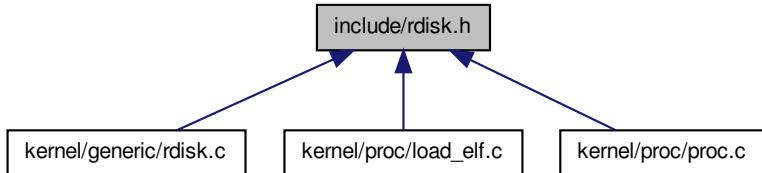
6.34.4.4 uint32_t proc_setup_mem(proc_t*)

6.35 include/rdisk.h File Reference

```
#include <arch/x86/x86.h> #include <arch/x86/elf.h> Include
dependency graph for rdisk.h:
```



This graph shows which files directly or indirectly include this file:



Defines

- #define **SECTOR** 512
- #define **ELF_MAGIC2** 0x8ec031fc

Functions

- void **readsect** (void *, **uint32_t**)

reads disk at a given offset

- void `readseg` (`uint32_t`, `uint32_t`, `uint32_t`)

loads data from disk into memory with data size to read.

- void **waitdisk** (void)

hangs till disk operation is done.

6.35.1 Define Documentation

6.35.1.1 #define ELF_MAGIC2 0x8ec031fc

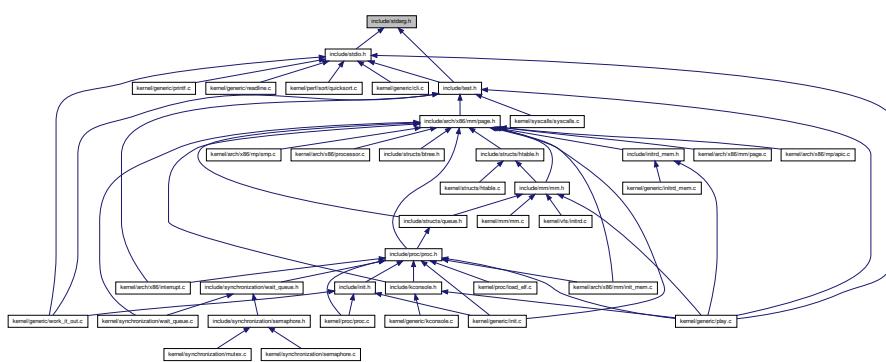
Definition at line 9 of file rdisk.h.

6.35.1.2 #define SECTOR 512

Definition at line 7 of file rdisk.h.

6.36 include/stdarg.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- `#define __va_size(type) (((sizeof(type) + sizeof(int)-1)/sizeof(int))*sizeof(int))`
 - `#define va_start(ap, last) ((ap) = (va_list)&(last) + __va_size(last))`
 - `#define va_arg(ap, type) (*(type *)((ap) += __va_size(type), (ap) - __va_size(type)))`
 - `#define va_end(ap)`

Typedefs

- `typedef char * va_list`

6.36.1 Define Documentation

6.36.1.1 `#define __va_size(type) (((sizeof(type) + sizeof(int)-1)/sizeof(int))*sizeof(int))`

Definition at line 11 of file stdarg.h.

6.36.1.2 `#define va_arg(ap, type) (*(type *)((ap) += __va_size(type), (ap) - __va_size(type)))`

Definition at line 15 of file stdarg.h.

6.36.1.3 `#define va_end(ap)`

Definition at line 17 of file stdarg.h.

6.36.1.4 `#define va_start(ap, last) ((ap) = (va_list)&(last) + __va_size(last))`

Definition at line 13 of file stdarg.h.

6.36.2 Typedef Documentation

6.36.2.1 `typedef char* va_list`

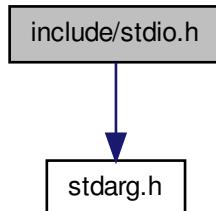
This code is used in both linux and BSD

Definition at line 9 of file stdarg.h.

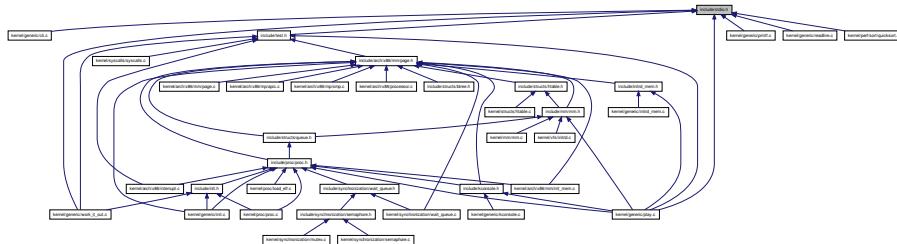
6.37 include/stdio.h File Reference

Standard Input Output operations.

#include <stdarg.h> Include dependency graph for stdio.h:



This graph shows which files directly or indirectly include this file:



Functions

- void [putchr \(int\)](#)
writes a character to screen.
- int [getchar \(void\)](#)
receives a character from keyboard and write it to screen and buffer.
- int [printf \(const char *,...\)](#)
An ideal printing function.
- int [vprintf \(const char *, va_list\)](#)
passes arguments to kvprintf after tokenizing them
- int [kvprintf \(const char *, void\(*func\)\(int, int *\), int *count, va_list ap\)](#)
A BSD like printing function.
- void [ksprintfn \(void\(*func\)\(int, int *\), int *count, uintmax_t num, int base, int width, int padc\)](#)
Prints numbers.
- char * [readline \(const char *\)](#)
reads a string from keyboard into screen.

6.37.1 Detailed Description

Standard Input Output operations.

Author

Saad Talaat

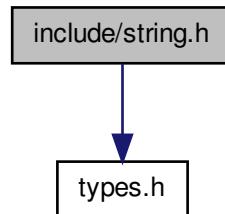
This header includes the Standard I/O operations for The screen and the keyboard, it handles an Interactive session between the user and kernel

Definition in file [stdio.h](#).

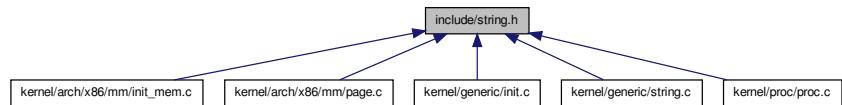
6.38 include/string.h File Reference

a header file for string and memory operations.

#include <types.h> Include dependency graph for string.h:



This graph shows which files directly or indirectly include this file:



Functions

- char * [strcpy](#) (char *dst, char *src)

- `char * strncpy (char *dst, char *src, uint32_t count)`
- `char * strlcpy (char *dst, char *src, uint32_t count)`
- `char * strcat (char *dst, char *src)`
- `char * strncat (char *dst, char *src, uint32_t count)`
- `char * strlcat (char *dst, char *src, uint32_t count)`
- `int strcmp (const char *cmp1, const char *cmp2)`
- `int strncmp (const char *cmp1, const char *cmp2, uint32_t count)`
- `char * strchr (const char *str, int c)`
- `char * strrchr (const char *str, int c)`
- `char * strnchr (const char *str, uint32_t count, int c)`
- `uint32_t strlen (const char *str)`
- `uint32_t strnlen (const char *str, uint32_t count)`

the function loops over the string character values till '\0' is reached or reached max value.
- `void * memset (void *dst, int c, uint32_t count)`

sets a number of bytes starting from a given pointer to a given character.
- `void * memcpy (void *dst, const void *src, uint32_t count)`

copies bytes from a memory address to another memory address
- `void * memmove (void *dst, const void *src, uint32_t count)`
- `void * memcmp (const void *cmp1, const void *cmp2, uint32_t count)`

6.38.1 Detailed Description

a header file for string and memory operations.

Definition in file [string.h](#).

6.38.2 Function Documentation

6.38.2.1 `void* memcmp (const void * cmp1, const void * cmp2, uint32_t count)`

6.38.2.2 `void* memmove (void * dst, const void * src, uint32_t count)`

6.38.2.3 `char* strcat (char * dst, char * src)`

6.38.2.4 `char* strchr (const char * str, int c)`

6.38.2.5 `char* strcpy (char * dst, char * src)`

6.38.2.6 `char* strlcat (char * dst, char * src, uint32_t count)`

6.38.2.7 `char* strlcpy (char * dst, char * src, uint32_t count)`

6.38.2.8 `uint32_t strlen (const char * str)`

6.38.2.9 `char* strncat(char * dst, char * src, uint32_t count)`

6.38.2.10 `char* strnchr(const char * str, uint32_t count, int c)`

6.38.2.11 `int strncmp(const char * cmp1, const char * cmp2, uint32_t count)`

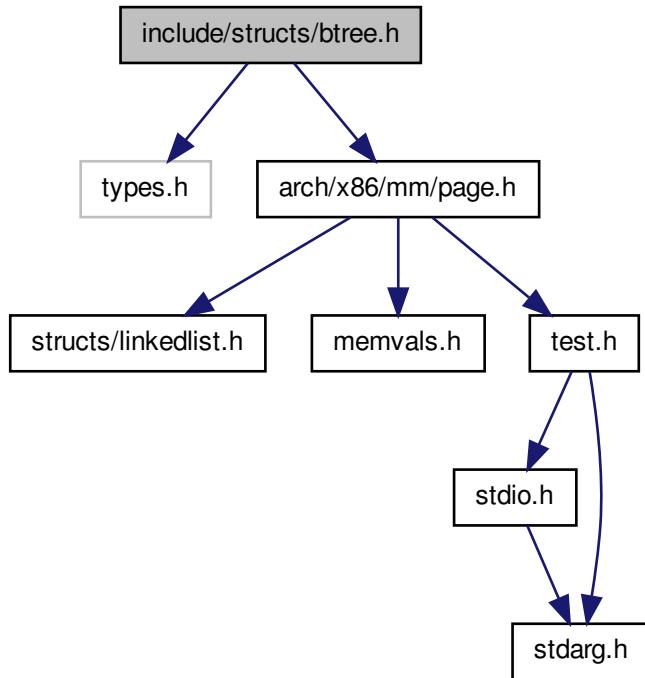
6.38.2.12 `char* strncpy(char * dst, char * src, uint32_t count)`

6.38.2.13 `char* strrchr(const char * str, int c)`

6.39 include/structs/btree.h File Reference

The B-Trees data structure.

#include <types.h> #include <arch/x86/mm/page.h> Include dependency graph for btree.h:



Data Structures

- struct [btree_node](#)
- struct [btree_t](#)

Typedefs

- [typedef struct btree_node btree_node_t](#)

Functions

- [LIST_HEAD \(Btree_List, btree_t\)](#)
- void [btree_init \(btree_t *\)](#)
- void [btree_insert \(btree_t *, uint32_t key, uint32_t value\)](#)
- void [btree_remove \(btree_t *, uint32_t key\)](#)
- void [btree_get \(btree_t *, uint32_t key\)](#)

6.39.1 Detailed Description

The B-Trees data structure.

Author

Saad Talaat

Definition in file [btree.h](#).

6.39.2 Typedef Documentation

6.39.2.1 [typedef struct btree_node btree_node_t](#)

6.39.3 Function Documentation

6.39.3.1 [void btree_get \(btree_t *, uint32_t key \)](#)

6.39.3.2 [void btree_init \(btree_t * \)](#)

6.39.3.3 [void btree_insert \(btree_t *, uint32_t key, uint32_t value \)](#)

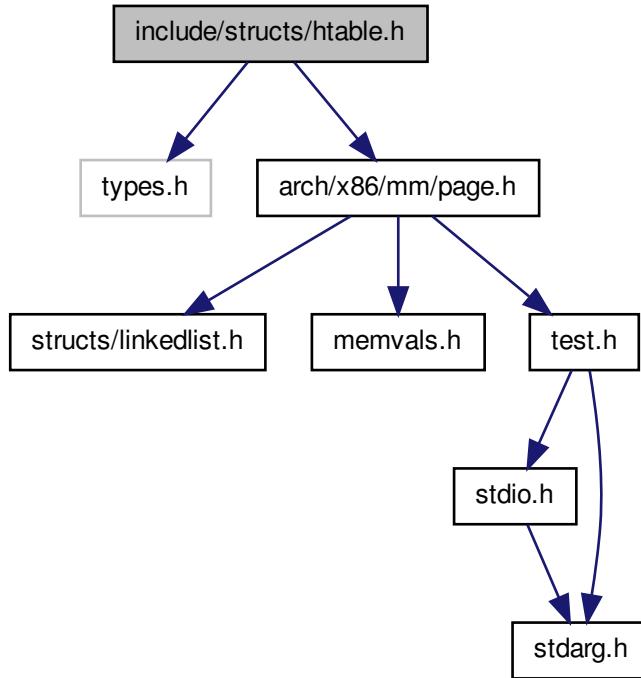
6.39.3.4 [void btree_remove \(btree_t *, uint32_t key \)](#)

6.39.3.5 [LIST_HEAD \(Btree_List , btree_t \)](#)

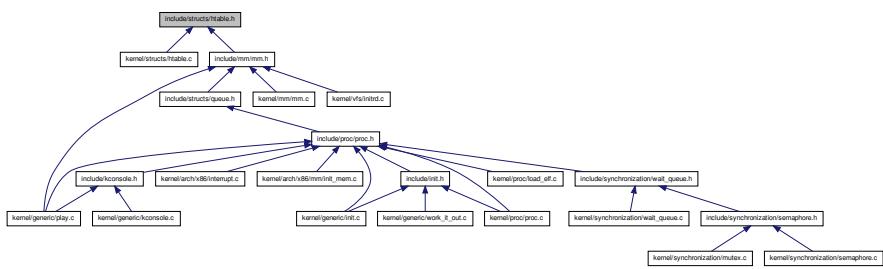
6.40 include/structs/htable.h File Reference

Chained Hash Table.

```
#include <types.h> #include <arch/x86/mm/page.h> Include dependency graph for htable.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [htable_node_t](#)
- struct [htable_t](#)

a hash table structure

Functions

- void [htable_init](#) ([htable_t](#) *, [uint32_t](#), [uint32_t](#)([*hash](#))([uint32_t](#)), [uint32_t](#)([*destroy](#))([uint32_t](#)))
Initializes new hash table.
- void [htable_destroy](#) ([htable_t](#) *)
Deletes hash table
- [uint32_t htable_insert](#) ([htable_t](#) *, [uint32_t](#) key, [uint32_t](#) value)
Inserts value into the hash table.
- [uint32_t htable_remove](#) ([htable_t](#) *, [uint32_t](#) key)
Deletes value from the hash table.
- [uint32_t table_lookup](#) ([htable_t](#) *, [uint32_t](#) value)
- [uint32_t early_htable_init](#) ([htable_t](#) *, [uint32_t](#), [uint32_t](#), [uint32_t](#)([*hash](#))([uint32_t](#)), [uint32_t](#)([*destroy](#))([uint32_t](#)))

6.40.1 Detailed Description

Chained Hash Table.

Author

Saad Talaat

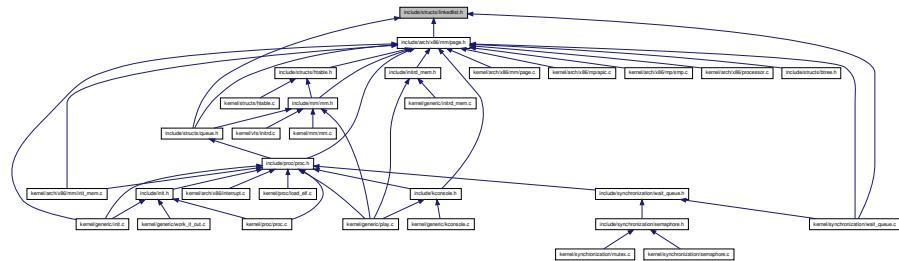
Definition in file [htable.h](#).

6.40.2 Function Documentation

6.40.2.1 [uint32_t table_lookup](#) ([htable_t](#) *, [uint32_t](#) value)

6.41 include/structs/linkedlist.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define LIST_HEAD(Instance, Type)
 - #define LIST_HEAD_INIT(head) {NULL}
 - #define LIST_ENTRY(type)
 - #define LIST_EMPTY(xhead) ((xhead)->first == NULL)
 - #define LIST_FIRST(xhead) ((xhead)->first)
 - #define LIST_NEXT(element, field) ((element)->field.next)
 - #define LIST_FOREACH(var, head, field)
 - #define LIST_INIT(head)
 - #define LIST_INSERT_AFTER(pre, element, field)
 - #define LIST_INSERT_BEFORE(post, element, field)
 - #define LIST_REMOVE(element, field)
 - #define LIST_INSERT_HEAD(head, element, field)

6.41.1 Define Documentation

6.41.1.1 #define LIST_EMPTY(xhead) ((xhead)->first == NULL)

Functions of the list, first,next, insert, empty..etc

Definition at line 37 of file linkedlist.h.

6.41.1.2 #define LIST_ENTRY(type)

Value:

```
struct {\n    struct type *next;\n    struct type **prev;\n}
```

Definition at line 29 of file linkedlist.h.

6.41.1.3 #define LIST_FIRST(*xhead*) ((*xhead*)>first)

Definition at line 39 of file linkedlist.h.

6.41.1.4 #define LIST_FOREACH(*var*, *head*, *field*)**Value:**

```
for ((var) = LIST_FIRST((head)); \
     (var); \
     (var) = LIST_NEXT((var), field))
```

For each loop on teh list

Definition at line 49 of file linkedlist.h.

6.41.1.5 #define LIST_HEAD(*Instance*, *Type*)**Value:**

```
struct Instance{\
    struct Type *first; \
    struct Type *tail; \
}
```

a boot time list is a macro coded list to provide ability to make sequential lists at boot time.

Definition at line 18 of file linkedlist.h.

6.41.1.6 #define LIST_HEAD_INIT(*head*) {NULL}

Definition at line 24 of file linkedlist.h.

6.41.1.7 #define LIST_INIT(*head*)**Value:**

```
do { \
    LIST_FIRST((head)) = NULL; \
} while(0)
```

Definition at line 56 of file linkedlist.h.

6.41.1.8 #define LIST_INSERT_AFTER(*pre*, *element*, *field*)**Value:**

```
do{\
    if((LIST_NEXT((element),field) = (LIST_NEXT((pre),field))) != NULL)\
        LIST_NEXT(pre,field)->field.prev = &(LIST_NEXT((element),field))
    );\
    LIST_NEXT((pre),field)->field.next = (element);\
    (element)->field.prev = &(LIST_NEXT((pre),field));\
} while(0)
```

Definition at line 61 of file linkedlist.h.

6.41.1.9 #define LIST_INSERT_BEFORE(post, element, field)

Value:

```
do{\
    (element)->field.prev = (post)->field.prev; \
    LIST_NEXT((element),field) = (post); \
    *(post)->field.prev = (element); \
    (post)->field.prev = &(LIST_NEXT((element),field)); \
} while(0)
```

Definition at line 70 of file linkedlist.h.

6.41.1.10 #define LIST_INSERT_HEAD(head, element, field)

Value:

```
do {\
    if((LIST_NEXT((element), field) = LIST_FIRST((head))) != NULL)\
        LIST_FIRST((head))->field.prev = &(LIST_NEXT((element),field));
    \
    LIST_FIRST((head)) = (element); \
    (element)->field.prev = &(LIST_FIRST((head))); \
} while(0)
```

Definition at line 86 of file linkedlist.h.

6.41.1.11 #define LIST_NEXT(element, field) ((element)->field.next)

Definition at line 46 of file linkedlist.h.

6.41.1.12 #define LIST_REMOVE(element, field)

Value:

```
do {\
    if(LIST_NEXT((element),field) != NULL)\
        LIST_NEXT((element),field)->field.prev =\
            (element)->field.prev; \
    *(element)->field.prev = LIST_NEXT( (element), field); \
}while(0)
```

Definition at line 78 of file linkedlist.h.

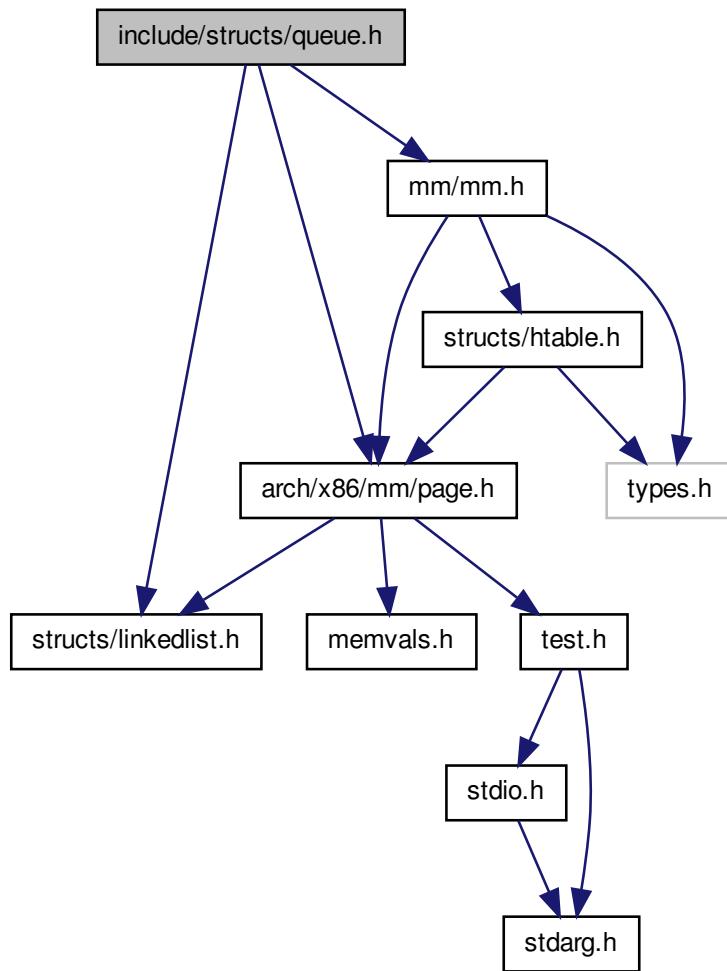
6.42 include/structs/list.h File Reference

Data Structures

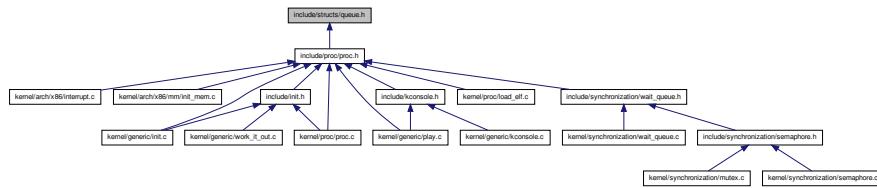
- struct [element](#)

6.43 include/structs/queue.h File Reference

```
#include <structs/linkedlist.h> #include <arch/x86/mm/page.-  
h> #include <mm/mm.h> Include dependency graph for queue.h:
```



This graph shows which files directly or indirectly include this file:



Defines

- #define **LIFO_HEAD**(name, member)
- #define **LIFO_HEAD_INIT**(h) **LIST_INIT**(h)
- #define **LIFO_ENTRY**(entry) **LIST_ENTRY**(entry)
- #define **LIFO_FIRST**(h) ((h)->first)
- #define **LIFO_EMPTY**(h) **LIST_EMPTY**((h))
- #define **LIFO_INIT**(h) **LIFO_FIRST**((h)) = NULL
- #define **LIFO_PUSH**(h, element, field) **LIST_INSERT_HEAD**(h, element, field)
- #define **LIFO_POP**(h, field)
- #define **FIFO_HEAD**(name, member, max)
- #define **FIFO_ENTRY**(entry) **LIST_ENTRY**(entry)
- #define **FIFO_FIRST**(h) ((h)->first)
- #define **FIFO_EMPTY**(h) (((h)->head == (h)->tail))
- #define **FIFO_INIT**(h) **FIFO_FIRST**(h) = kmalloc(sizeof(*(h)->first) * (h)->items)
- #define **FIFO_PUSH**(h, element) **FIFO_FIRST**(h)[(h)->tail = ((h)->tail +1) < (h)->items ? ((h)->tail+1): 0] = *(element)
- #define **FIFO_POP**(h) &**FIFO_FIRST**(h)[(h)->head = ((h)->head +1) < (h)->items ? ((h)->head +1) : 0];

6.43.1 Define Documentation

6.43.1.1 #define FIFO_EMPTY(h) (((h)->head == (h)->tail))

Definition at line 62 of file queue.h.

6.43.1.2 #define FIFO_ENTRY(entry) LIST_ENTRY(entry)

Definition at line 57 of file queue.h.

6.43.1.3 #define FIFO_FIRST(h) ((h)->first)

Definition at line 60 of file queue.h.

6.43.1.4 #define FIFO_HEAD(*name*, *member*, *max*)

Value:

```
struct {
    \
    struct member *first;    \
    size_t items; \
    size_t tail; \
    size_t head; \
} name = \
{ \
    \
    .first = NULL, \
    .items = (max), \
    .tail = 0, \
    .head = 0 \
}
```

FIFO QUEUE

Definition at line 43 of file queue.h.

6.43.1.5 #define FIFO_INIT(*h*) FIFO_FIRST(*h*) = kmalloc(sizeof(*(*h*)->first) * (*h*)->items)

Definition at line 64 of file queue.h.

6.43.1.6 #define FIFO_POP(*h*) &FIFO_FIRST(*h*)[(*h*)->head = ((*h*)->head +1) < (*h*)->items ? ((*h*)->head +1) : 0];

Definition at line 70 of file queue.h.

6.43.1.7 #define FIFO_PUSH(*h*, *element*) FIFO_FIRST(*h*)[(*h*)->tail = ((*h*)->tail +1) < (*h*)->items ? ((*h*)->tail+1): 0] = *(*element*)

Definition at line 67 of file queue.h.

6.43.1.8 #define LIFO_EMPTY(*h*) LIST_EMPTY((*h*))

Definition at line 29 of file queue.h.

6.43.1.9 #define LIFO_ENTRY(*entry*) LIST_ENTRY(*entry*)

Definition at line 24 of file queue.h.

6.43.1.10 #define LIFO_FIRST(*h*) ((*h*)->first)

Definition at line 27 of file queue.h.

6.43.1.11 #define LIFO_HEAD(*name*, *member*)

Value:

```
struct name{           \
    struct member *first;   \
    struct member *tail;   \
}
```

LIFO QUEUE

Definition at line 16 of file queue.h.

6.43.1.12 #define LIFO_HEAD_INIT(*h*) LIST_INIT(*h*)

Definition at line 21 of file queue.h.

6.43.1.13 #define LIFO_INIT(*h*) LIFO_FIRST((*h*)) = NULL

Definition at line 32 of file queue.h.

6.43.1.14 #define LIFO_POP(*h*, *field*)

Value:

```
LIST_FIRST((h)); \
LIST_REMOVE(LIST_FIRST((h)), field)
```

Definition at line 38 of file queue.h.

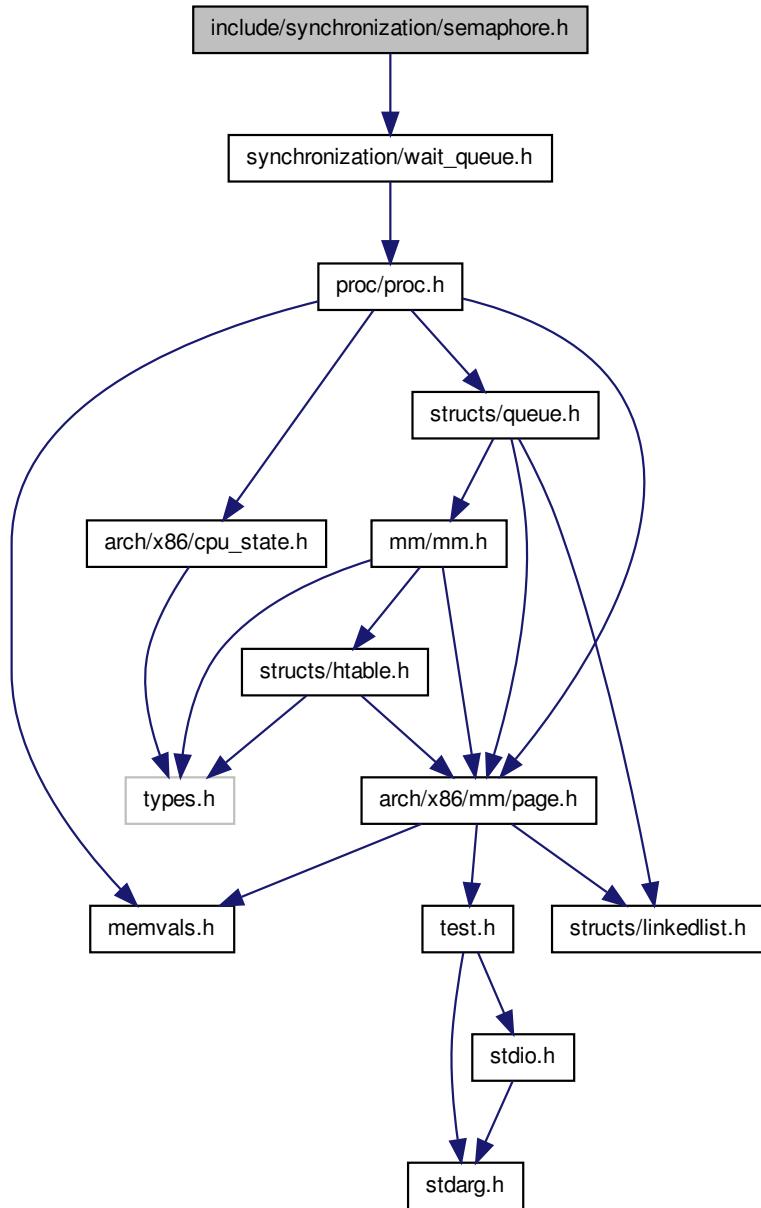
6.43.1.15 #define LIFO_PUSH(*h*, *element*, *field*) LIST_INSERT_HEAD(*h*, *element*, *field*)

Definition at line 35 of file queue.h.

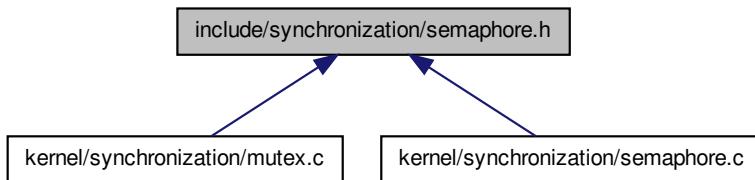
6.44 include/synchronization/semaphore.h File Reference

Semaphore structures and function headers.

```
#include <synchronization/wait_queue.h> Include dependency graph  
for semaphore.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [semaphore_t](#)

Functions

Semaphores

Semaphore synchronization primitives

- void [semaphore_init \(semaphore_t *s, uint32_t num\)](#)
Initialization of a semaphore primitive function.
- void [semaphore_down \(semaphore_t *s, proc_t *proc\)](#)
does down operation over semaphore primitive
- void [semaphore_up \(semaphore_t *s\)](#)
Up function for a semaphore.

6.44.1 Detailed Description

Semaphore structures and function headers.

Author

Saad Talaat

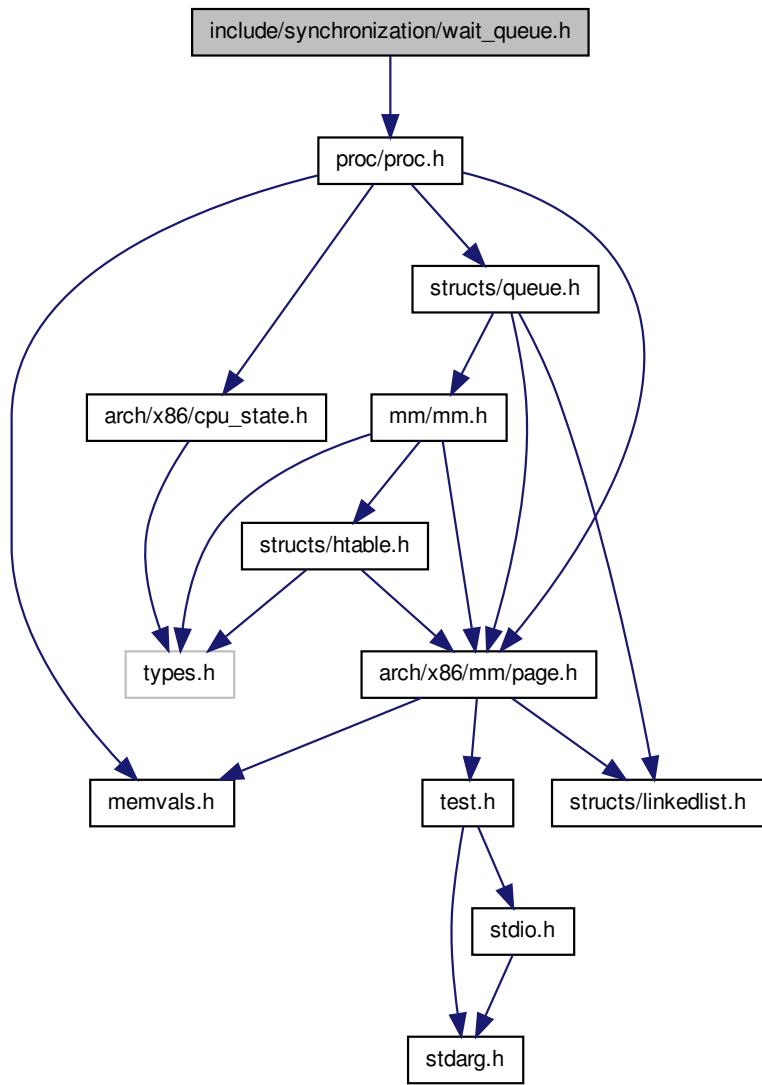
Date

1st December, 2012

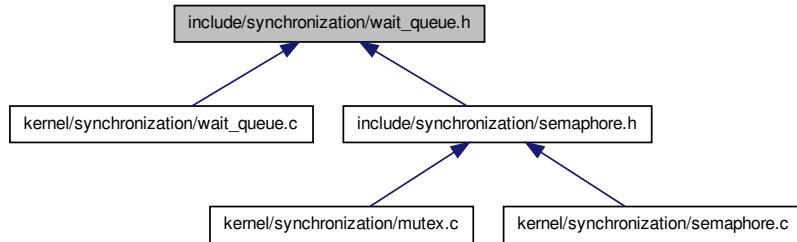
Definition in file [semaphore.h](#).

6.45 include/synchronization/wait_queue.h File Reference

```
#include <proc/proc.h> Include dependency graph for wait_queue.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [waiting_proc](#)

Typedefs

- typedef struct [waiting_proc](#) [waiting_t](#)

Functions

- [LIST_HEAD](#) ([Wait_list](#), [waiting_proc](#))

Wait queues

Wait queues

- void [wait_init](#) (struct Proc_List *[waiting_procs](#))

Waiting procs linked list initialization function.
- void [wait_sleep](#) (struct Proc_List *[list](#), [proc_t](#) *[proc](#), [uint32_t](#) [ticks](#))

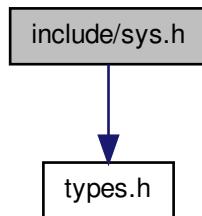
function that blocks a process
- void [wait_update](#) (void)

blocked process updating function
- void [wait_wakeup](#) (struct Proc_List *[list](#))

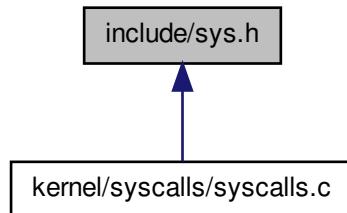
wakes up a process once timeout or resource free

6.46 include/sys.h File Reference

#include <types.h> Include dependency graph for sys.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define [S_PRINTF](#) 0

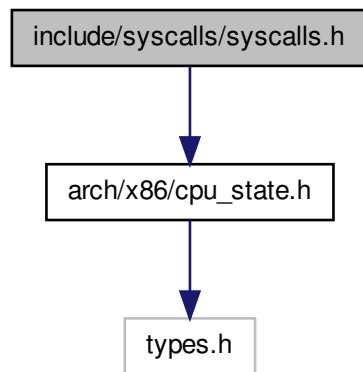
6.46.1 Define Documentation

6.46.1.1 #define S_PRINTF 0

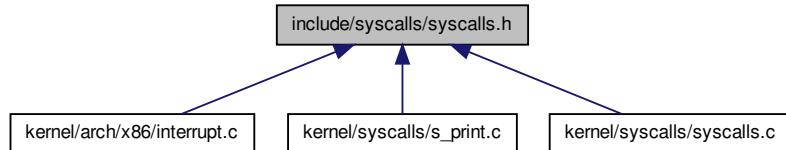
Definition at line 6 of file sys.h.

6.47 include/syscalls/syscalls.h File Reference

```
#include <arch/x86/cpu_state.h>  Include dependency graph for syscalls.h:
```



This graph shows which files directly or indirectly include this file:

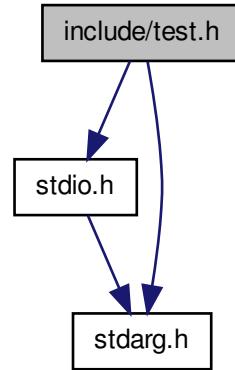


Functions

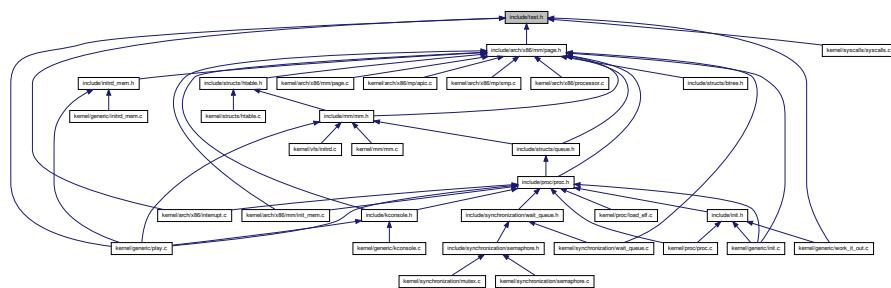
- [int32_t map_syscall \(cpu_state_t *cpu_state\)](#)
maps the system call by offsetting the sys_call_table structure using the offset in eax register.
- [int32_t sys_printf \(char *str\)](#)
The actual print procedure requested by the printf system call.

6.48 include/test.h File Reference

```
#include <stdio.h> #include <stdarg.h> Include dependency graph  
for test.h:
```



This graph shows which files directly or indirectly include this file:



Defines

- `#define panic(...)`
 - `#define assert(x)`

Functions

- void _panic_ (const char *, int, const char *,...) _attribute_((noreturn))

6.48.1 Define Documentation

6.48.1.1 #define assert(x)

Value:

```
if (! (x)) {           \
    panic("Assertion Failed %s\n", #x); \
}
```

Definition at line 14 of file test.h.

6.48.1.2 #define panic(...)

Value:

```
printf(">>>> KERNEL PANIC <<<<<\n"); \
	panic(__FILE__, __LINE__, __VA_ARGS__)
```

Definition at line 10 of file test.h.

6.49 include/time/time.h File Reference

include all time info

Functions

- void [time_handler](#) (void)

6.49.1 Detailed Description

include all time info

Author

Saad Talaat

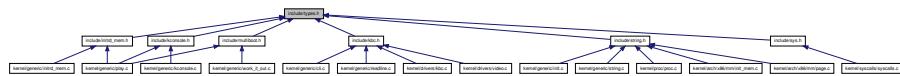
Date

1st December, 2012

Definition in file [time.h](#).

6.50 include/types.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define `NULL` ((void*) 0)
- #define `ROUND_DOWN`(x, y)
- #define `ROUND_UP`(x, y)

Typedefs

- typedef unsigned `uint32_t`
- typedef unsigned short `uint16_t`
- typedef unsigned char `uint8_t`
- typedef unsigned long `uint64_t`
- typedef char `int8_t`
- typedef int `int32_t`
- typedef short `int16_t`
- typedef long long `int64_t`
- typedef unsigned `int size_t`
- typedef `uint64_t uintmax_t`
- typedef `int64_t intmax_t`
- typedef `uint32_t reg_t`
- typedef `uint32_t paddr_t`
- typedef `uint32_t vaddr_t`
- typedef `uint32_t * uintptr_t`
- typedef `int(* fnptr_t)()`

6.50.1 Define Documentation

6.50.1.1 #define NULL ((void*) 0)

Definition at line 8 of file types.h.

6.50.1.2 #define ROUND_DOWN(x, y)**Value:**

```
({          \
    uint32_t z = (uint32_t) (x);      \
    (typeof(x)) (z - z % (y));      \
})
```

Definition at line 28 of file types.h.

6.50.1.3 #define ROUND_UP(x, y)**Value:**

```
({\ \
    uint32_t z = (uint32_t) (y); \
    (typeof(x)) (ROUND_DOWN(x,y) + y); \
})
```

Definition at line 33 of file types.h.

6.50.2 Typedef Documentation**6.50.2.1 typedef int(* fnptr_t)()**

Definition at line 26 of file types.h.

6.50.2.2 typedef short int16_t

Definition at line 15 of file types.h.

6.50.2.3 typedef int int32_t

Definition at line 14 of file types.h.

6.50.2.4 typedef long long int64_t

Definition at line 16 of file types.h.

6.50.2.5 typedef char int8_t

Definition at line 13 of file types.h.

6.50.2.6 typedef int64_t intmax_t

Definition at line 19 of file types.h.

6.50.2.7 `typedef uint32_t paddr_t`

Definition at line 23 of file types.h.

6.50.2.8 `typedef uint32_t reg_t`

Definition at line 20 of file types.h.

6.50.2.9 `typedef unsigned int size_t`

Definition at line 17 of file types.h.

6.50.2.10 `typedef unsigned short uint16_t`

Definition at line 10 of file types.h.

6.50.2.11 `typedef unsigned int uint32_t`

Definition at line 9 of file types.h.

6.50.2.12 `typedef unsigned long uint64_t`

Definition at line 12 of file types.h.

6.50.2.13 `typedef unsigned char uint8_t`

Definition at line 11 of file types.h.

6.50.2.14 `typedef uint64_t uintmax_t`

Definition at line 18 of file types.h.

6.50.2.15 `typedef uint32_t* uintptr_t`

Definition at line 25 of file types.h.

6.50.2.16 `typedef uint32_t vaddr_t`

Definition at line 24 of file types.h.

6.51 arch/x86/include/types.h File Reference

This graph shows which files directly or indirectly include this file:



Typedefs

- `typedef unsigned int uint32_t`
- `typedef unsigned short uint16_t`
- `typedef unsigned char uint8_t`
- `typedef unsigned long long uint64_t`
- `typedef char int8_t`
- `typedef int int32_t`
- `typedef short int16_t`
- `typedef long long int64_t`

6.51.1 Typedef Documentation

6.51.1.1 `typedef short int16_t`

Definition at line 9 of file types.h.

6.51.1.2 `typedef int int32_t`

Definition at line 8 of file types.h.

6.51.1.3 `typedef long long int64_t`

Definition at line 10 of file types.h.

6.51.1.4 `typedef char int8_t`

Definition at line 7 of file types.h.

6.51.1.5 `typedef unsigned short uint16_t`

Definition at line 4 of file types.h.

6.51.1.6 `typedef unsigned int uint32_t`

Definition at line 3 of file types.h.

6.51.1.7 `typedef unsigned long long uint64_t`

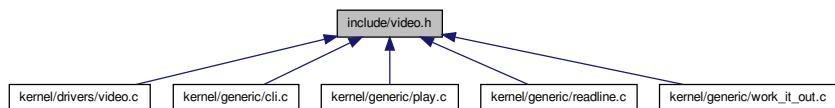
Definition at line 6 of file types.h.

6.51.1.8 `typedef unsigned char uint8_t`

Definition at line 5 of file types.h.

6.52 include/video.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define CGA_INDEX1 0x3D4
- #define CGA_DATA1 0x3D5
- #define CGA_BUFF_OFF 0xB8000
- #define CGA_COLS 80
- #define CGA_ROWS 25
- #define CGA_SIZE (CGA_ROWS * CGA_COLS)
- #define COLOR_DARK_GRAY 0x0700
- #define COLOR_BLUE 0x0100
- #define COLOR_GREEN 0x0200
- #define COLOR_RED 0x0400
- #define COLOR_GRAY 0x0800
- #define COLOR_WHITE 0x0f00
- #define BACKGROUND_GRAY 0x7000
- #define BACKGROUND_BLUE 0x1000
- #define BACKGROUND_GREEN 0x2000
- #define BACKGROUND_RED 0x4000
- #define BACKGROUND_BLINK 0x8000
- #define BACKGROUND_WHITE 0xF000

Functions

- void `cga_init` (void)
initiates CGA.
- `uint16_t cga_get_pos` (void)
gets the cursor position on the screen.
- void `cga_set_pos` (`uint16_t`)
sets the position to a given value on the screen map
- void `cga_putc` (int `c`)
writes a character to screen using cga.
- void `cga_set_attr` (`uint16_t`)
sets global color attribute to a given value.
- void `cga_clear` (void)
clears the screen.
- void `cga_putstr` (char *)
uses cga_putc to write a string

Variables

- `uint16_t cursor_position`
- `uint16_t * char_buff`

6.52.1 Define Documentation

6.52.1.1 #define BACKGROUND_BLINK 0x8000

Definition at line 26 of file video.h.

6.52.1.2 #define BACKGROUND_BLUE 0x1000

Definition at line 23 of file video.h.

6.52.1.3 #define BACKGROUND_GRAY 0x7000

Definition at line 22 of file video.h.

6.52.1.4 #define BACKGROUND_GREEN 0x2000

Definition at line 24 of file video.h.

6.52.1.5 #define BACKGROUND_RED 0x4000

Definition at line 25 of file video.h.

6.52.1.6 #define BACKGROUND_WHITE 0xF000

Definition at line 27 of file video.h.

6.52.1.7 #define CGA_BUFF_OFF 0xB8000

Definition at line 12 of file video.h.

6.52.1.8 #define CGA_COLS 80

Definition at line 13 of file video.h.

6.52.1.9 #define CGA_DATA1 0x3D5

Definition at line 11 of file video.h.

6.52.1.10 #define CGA_INDEX1 0x3D4

Definition at line 10 of file video.h.

6.52.1.11 #define CGA_ROWS 25

Definition at line 14 of file video.h.

6.52.1.12 #define CGA_SIZE (CGA_ROWS * CGA_COLS)

Definition at line 15 of file video.h.

6.52.1.13 #define COLOR_BLUE 0x0100

Definition at line 17 of file video.h.

6.52.1.14 #define COLOR_DARK_GRAY 0x0700

Definition at line 16 of file video.h.

6.52.1.15 #define COLOR_GRAY 0x0800

Definition at line 20 of file video.h.

6.52.1.16 #define COLOR_GREEN 0x0200

Definition at line 18 of file video.h.

6.52.1.17 #define COLOR_RED 0x0400

Definition at line 19 of file video.h.

6.52.1.18 #define COLOR_WHITE 0x0f00

Definition at line 21 of file video.h.

6.52.2 Variable Documentation

6.52.2.1 uint16_t* char_buff

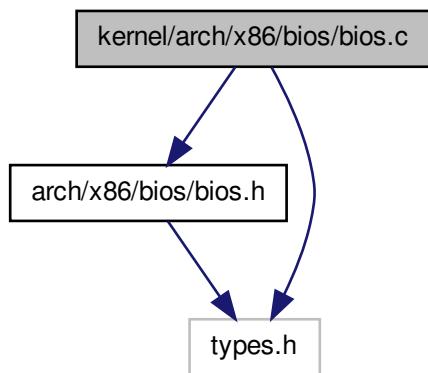
Definition at line 30 of file video.h.

6.52.2.2 uint16_t cursor_position

Definition at line 29 of file video.h.

6.53 kernel/arch/x86/bios/bios.c File Reference

```
#include <arch/x86/bios/bios.h> #include <types.h> Include  
dependency graph for bios.c:
```



Defines

- `#define BIOS_EBDA_PTR 0x40e`

Functions

- `void bios_init (void)`

Variables

- `uintptr_t ebda = 0`

6.53.1 Define Documentation

6.53.1.1 `#define BIOS_EBDA_PTR 0x40e`

Definition at line 3 of file bios.c.

6.53.2 Function Documentation

6.53.2.1 void bios_init(void)

Definition at line 5 of file bios.c.

6.53.3 Variable Documentation

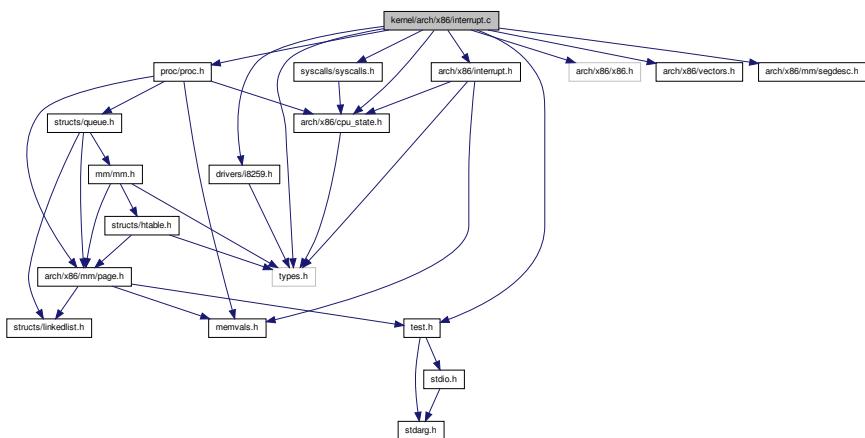
6.53.3.1 uintptr_t ebda = 0

Definition at line 4 of file bios.c.

6.54 kernel/arch/x86/interrupt.c File Reference

Interrupt and fault handling.

```
#include <types.h> #include <test.h> #include <proc/proc.h>
#include <arch/x86/x86.h> #include <arch/x86/interrupt.h>
#include <arch/x86/vectors.h> #include <arch/x86/mm/segdesc.h>
#include <arch/x86/cpu_state.h> #include <syscalls/syscalls.h>
#include <drivers/i8259.h> Include dependency graph for interrupt.c:
```



Interrupts Initialization

- `gatedesc idt [64]`
Declare IDT with 255 width interrupts.
- `char * x86_exception_names []`

Fault names.

- [int_generic](#)
- [void trap \(void\)](#)
- [void interrupt_printinfo \(void\)](#)
- [void idt_init \(void\)](#)

Initializing Interrupt Descriptor table.

- [void register_exception \(uint32_t index, char *name, uint16_t present, void\(handler\)\(void\)\)](#)

register an interrupt handling procedure

- [void interrupt_init \(void\)](#)
 - [void map_exception \(uint32_t int_index, cpu_state_t *cpu_state\)](#)
- maps an exception/interrupt to its handler*
- [uint32_t page_fault_handler \(cpu_state_t *cpu_state\)](#)

Page Fault Handler.

6.54.1 Detailed Description

Interrupt and fault handling.

Author

Saad Talaat

Date

27 Sept, 2012

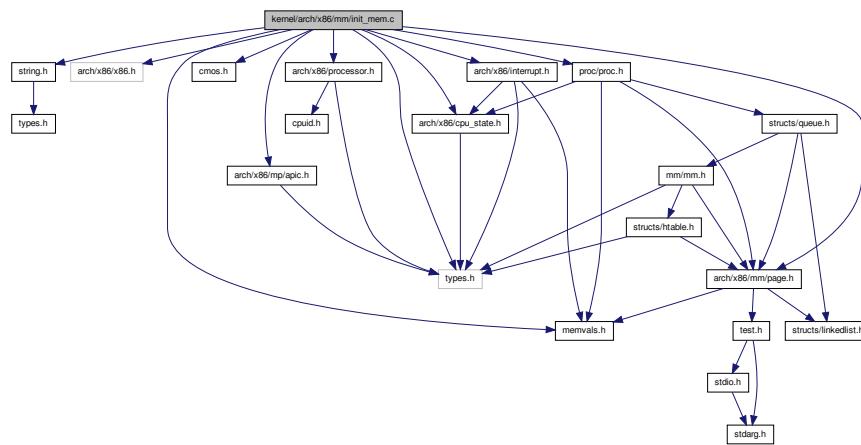
Definition in file [interrupt.c](#).

6.55 kernel/arch/x86/mm/init_mem.c File Reference

Virtual Memory initialization.

```
#include <types.h>    #include <arch/x86/x86.h>    #include
<memvals.h> #include <cmos.h> #include <string.h> #include
<arch/x86/processor.h>    #include <arch/x86/mm/page.h> x
#include <arch/x86/interrupt.h> #include <arch/x86/cpu-
state.h> #include <arch/x86/mp/apic.h> #include <proc/proc.-
```

h> Include dependency graph for init_mem.c:



Virtual Memory Initialization.

Memory size is read and segmentation is reset. and kernel page directory is initialized and mapped. also other segments are initialized and mapped.

- `uint32_t max_addr`
- `static uint32_t mem_base`
- `static uint32_t ext_base`
- `char * next_free = 0`
- `static uint32_t alloc_lock = 0`
- `struct Page * pages`
- `proc_t * proc_table`
- `struct Segdesc catgdt []`
 - New global descriptor table.*
 - `struct Gdtdesc gdtdesc`
 - `Idtdesc idtdesc`
- `tss_t tss`
- `static uint32_t x86_read_mem_size (int x)`
 - Reading memory size using the CMOS RAM/RTC device.*
- `void memory_printinfo (void)`
 - Reads memory size and determine the pages count needed to map it.*
 - `void scan_memory (void)`
 - `void * allocate (uint32_t n, uint32_t align)`
- `void init_tss (void)`
 - a boot time allocation function, allocate heaps.*

sets up the Task state segment and task register.

- void [x86_setup_memory](#) (void)
- main memory initialization function.*

6.55.1 Detailed Description

Virtual Memory initialization.

Author

Saad Talaat

Date

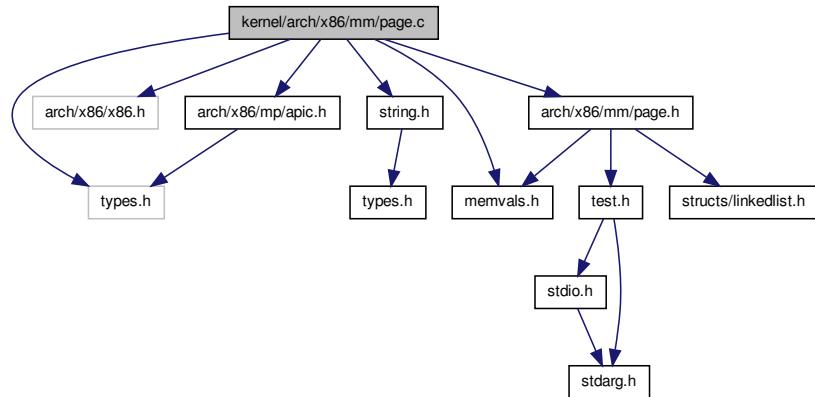
Monday 17/09/2012

Definition in file [init_mem.c](#).

6.56 kernel/arch/x86/mm/page.c File Reference

Intel Paging.

```
#include <types.h>    #include <arch/x86/x86.h>    #include
<memvals.h> #include <string.h> #include <arch/x86/mm/page.-.
h> #include <arch/x86/mp/apic.h> Include dependency graph for page.c:
```



Paging.

Programming the hardware support, Initially 32-bit paging is used. and Segment register refer to the whole memory since we'll be operating on protected paging memory mode. and Kernel is mapped to Virtual address 0xF0000000. Paging is set recursively by making cr3 refer to the paging directory.

- `uint32_t page_count`
- `pde_t * global_pgdir`
- `uint32_t global_cr3`
- `char * next_free`
- `struct Page * pages`
- `static struct PageList free_pages`
- `void x86_paging_init (void)`
 - Initiate global pages.*
- `void x86_page_init (struct Page *page)`
 - Initiate a page entry.*
- `int x86_page_alloc (struct Page **page_byref)`
 - Allocate a page page is removed from the free pages list and initiated.*
- `void x86_page_free (struct Page *page)`
 - given page is freed.*
- `void x86_page_detach (struct Page *page)`
 - when [un]refering a page, ref member is decremented.*
- `pte_t * x86_pgdir_find (pde_t *pgdir, const void *va, int allocate)`
 - find a page, if doesn't exist allocate it*
- `struct Page * x86_page_lookup (pde_t *pgdir, void *va, pte_t **pte)`
 - looks up a page refers to vaddr.*
- `void x86_page_remove (pde_t *pgdir, void *va)`
 - removes a page that refers to a given Virtual address from a page directory.*
- `int x86_page_insert (pde_t *pgdir, struct Page *page, void *va, uint32_t perm)`
 - inserts a page that refers to a given va, into a given page directory.*
- `void map_segment_page (pde_t *pgdir, vaddr_t linear, size_t size, paddr_t physical, int perm)`
 - maps a segment to a virtual address on a specific page directory.*
- `void x86_test_pgdir (void)`

6.56.1 Detailed Description

Intel Paging.

Date

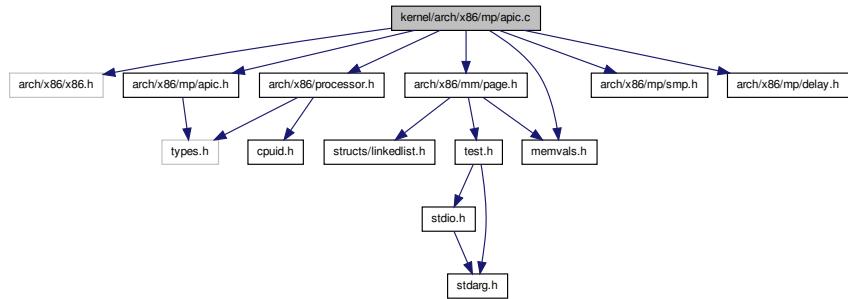
Wednesday 19/09/2012

Definition in file [page.c](#).

6.57 kernel/arch/x86/mp/apic.c File Reference

supporting xAPIC for MPs

```
#include <arch/x86/x86.h> #include <arch/x86/processor.-  
h> #include <arch/x86/mp/apic.h> #include <arch/x86/mm/page.-  
h> #include <memvals.h> #include <arch/x86/mp/smp.h> x  
#include <arch/x86/mp/delay.h> Include dependency graph for apic.c:
```



APIC

- `uint32_t vector_addr = 0x00022000`
- `uint32_t * lapic = (uint32_t *)(0xfee00000)`
- `void lapic_init (void)`

Soft Initiates the LAPIC.
- `void soft_lapic_enable (void)`

Soft To enable the Local APIC to receive interrupts you also have to set bit 8 in the Spurious Interrupt Vector Register.
- `void msr_lapic_enable (void)`

APIC setting IA32_APIC_BASE Model Specific Register (MSR)
- `void apic_s_ipi (icr_t icr, uint8_t lapicid)`

Sending Startup IPI to processor specified with lapicid.
- `void apic_init_ipi (uint8_t lapicid)`

Sending INIT IPI to processor specified with lapicid.
- `void apic_read_errors (void)`

Reads error field in lapic.

6.57.1 Detailed Description

supporting xAPIC for MPs

Author

Saad Talaat , Menna Essa

Date

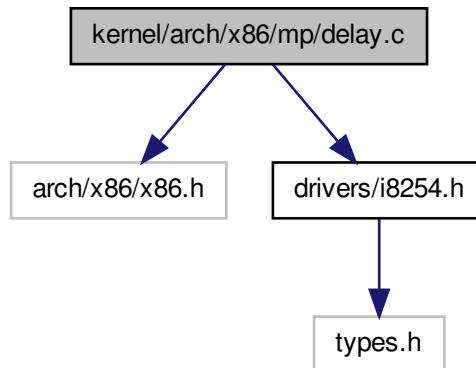
18 November, 2012

Definition in file [apic.c](#).

6.58 kernel/arch/x86/mp/delay.c File Reference

Simple delay functions for synchronization.

```
#include <arch/x86/x86.h>      #include <drivers/i8254.h> x  
Include dependency graph for delay.c:
```



Functions

Delay

- void [asm_delay_loop \(uint32_t t\)](#)
useless loop to use while delaying
- void [asm_fake_loop \(uint32_t t\)](#)
useless loop to calibrate cpu
- void [delay \(uint32_t microsec\)](#)
uninterruptable delay function

6.58.1 Detailed Description

Simple delay functions for synchronization.

Author

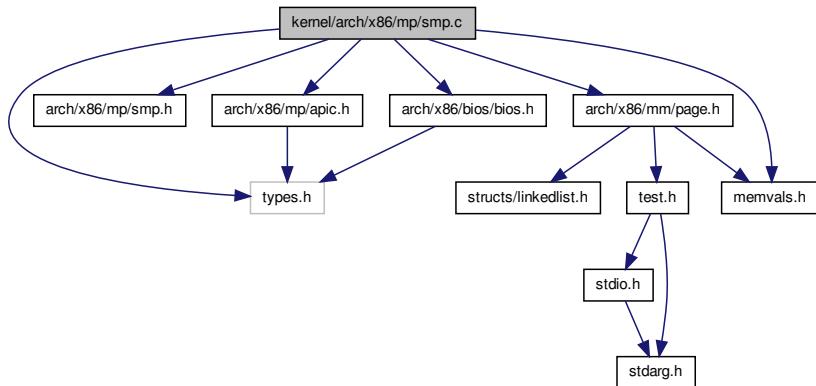
Menna Essa

Definition in file [delay.c](#).

6.59 kernel/arch/x86/mp/smp.c File Reference

SMP Support functions.

```
#include <types.h> #include <arch/x86/mp/smp.h> #include
<arch/x86/mp/apic.h> #include <arch/x86/bios/bios.h> x
#include <arch/x86/mm/page.h> #include <memvals.h> Include
dependency graph for smp.c:
```



Delay

- `const char trampoline []`
- `const char trampoline_end []`
- `fpstruct_t * fs`
floating point structure
- `ct_hdr * ct`
configuration table.
- `static ct_proc_entry * processor_entries`
- `ct_bus_entry * bus_entries`

- `ct_io_apic_entry * io_apic_entries`
- `ct_io_intr_entry * io_intr_entries`
- `ct_loc_intr_entry * loc_intr_entries`
- `uint32_t processors_count`
- `uint32_t io_apic_cnt`
- `uint32_t bus_entry_cnt`
- `uint32_t io_apic_entry_cnt`
- `uint32_t io_intr_entry_cnt`
- `uint32_t loc_intr_entry_cnt`
- `uint8_t fps_check (uint8_t *base)`
checks if floating point structure is valid , the sum of all elements must be 0
- `uint8_t ct_check (void)`
checks if configuration is valid , the sum of all elements must be 0
- `void fsp_print (fpstruct_t *fs)`
parses the floating point structure
- `void ct_read_hdr (void)`
parses the configuration table structure
- `void print_proc_entry (ct_proc_entry *processor_entry)`
parses the cpu entry
- `void ct_entries (void)`
parses the config table and sets the entries pointers
- `void find_set_fps (void)`
*Searches and Sets the global Floating pointer structure. 1. search first 1K of EBDA
2. if EBDA is undefined, search last 1K of base memory 3. search 64K starting at 0xf0000.*
- `uint8_t cpu_is_bootstrap (ct_proc_entry *processor_entry)`
checks if cpu is bootstrap
- `uint8_t cpu_is_enabled (ct_proc_entry *processor_entry)`
checks if cpu is enabled
- `void map_AP_Startup (void)`
Initializes application processors.
- `void ap_init (void)`

6.59.1 Detailed Description

SMP Support functions.

Author

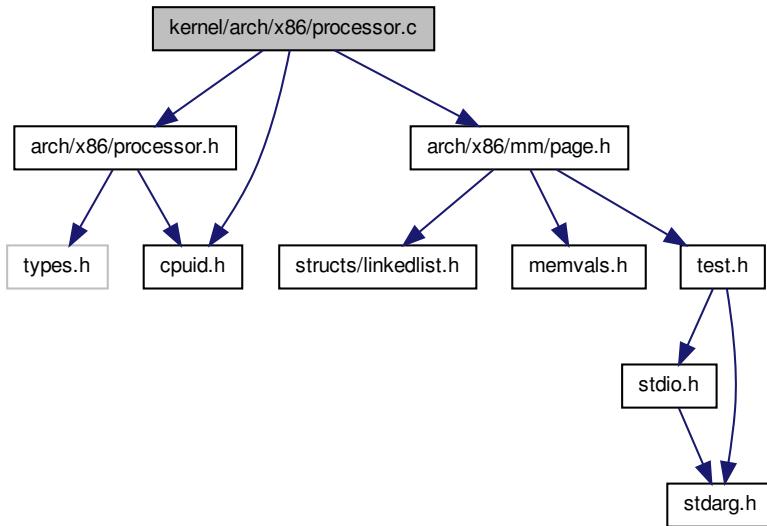
Menna Essa

Definition in file [smp.c](#).

6.60 kernel/arch/x86/processor.c File Reference

Identify and initialize cpu structures.

```
#include <arch/x86/processor.h>      #include <cpuid.h> x
#include <arch/x86/mm/page.h> Include dependency graph for processor.-c:
```



CPU-Identification

- `char * cpu_vendors [3]`
- `cpu_t * cpu`
- `void processor_printinfo ()`
- `void processor_identify (void)`

6.60.1 Detailed Description

Identify and initialize cpu structures.

Author

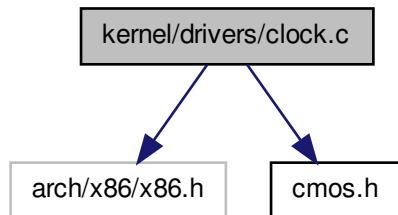
Saad Talaat

Definition in file [processor.c](#).

6.61 kernel/drivers/clock.c File Reference

Clock Low level controller. sets periodic interrupts and power state.

```
#include <arch/x86/x86.h> #include <cmos.h> Include dependency  
graph for clock.c:
```



Functions

RTC Driver.

- void [clock_enable_rtc](#) (void)

6.61.1 Detailed Description

Clock Low level controller. sets periodic interrupts and power state.

Author

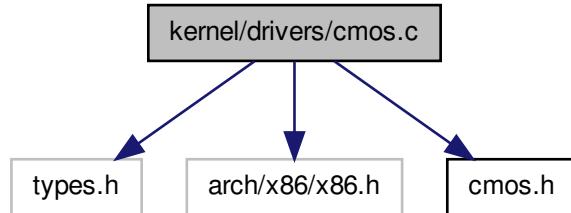
Saad Talaat

Definition in file [clock.c](#).

6.62 kernel/drivers/cmos.c File Reference

CMOS Driver.

```
#include <types.h>    #include <arch/x86/x86.h>    #include
<cmos.h> Include dependency graph for cmos.c:
```



Functions

CMOS Driver.

supports status A and B Power options , Refer to boch's Ports.lst for details.

- `uint8_t cmos_set_power_stat (uint8_t stat)`
supports CMOS Status B power options.
- `uint32_t cmos_get_reg (uint8_t value)`
Gets RTC registers values.
- `uint32_t cmos_set_reg (uint8_t index, uint8_t value)`
set RTC registers values.

6.62.1 Detailed Description

CMOS Driver.

Author

Menna Essa

Date

31/10/2011

Definition in file [cmos.c](#).

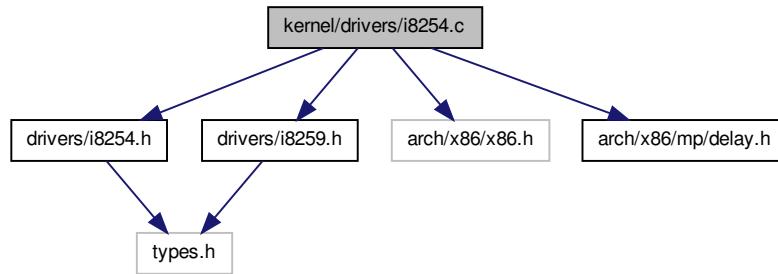
6.63 kernel/drivers/i8254.c File Reference

Intel 8253/8254 PIT(Programmable Interrupt Timer) Controller.

```
#include <drivers/i8254.h>    #include <drivers/i8259.h>
#include <arch/x86/x86.h>    #include <arch/x86/mp/delay.h>


### Include dependency graph for i8254.c:


```



Intel 8254 PIT

- `#define MAGIC_NUMBER 1194`
- `#define LOOPS 150000`
- `uint32_t delay_loop_const`
- `void i8254_init (void)`
Initiate the PIT to interrupt every time slice.
- `void i8254_calibrate_delay_loop (void)`
calibrates the number of loops per 1 ms

6.63.1 Detailed Description

Intel 8253/8254 PIT(Programmable Interrupt Timer) Controller.

Author

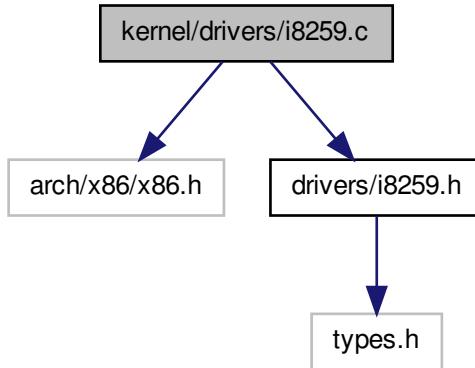
Saad Talaat , Menna Essa

Definition in file [i8254.c](#).

6.64 kernel/drivers/i8259.c File Reference

Intel 8259 PIC(Programmable Interrupt Controller) Low Level Driver.

```
#include <arch/x86/x86.h>      #include <drivers/i8259.h> x
Include dependency graph for i8259.c:
```



Functions

i8259-PIC Driver.

- void `i8259_init` (void)
Initializing the PIC to work under Protected mode.
- void `i8259_disable` (void)
Disables the PIC IRQs.
- void `i8259_mask_irq` (`uint16_t` irq)
get the in service interrupt.
- void `i8259_unmask_irq` (`uint16_t` irq)
reads the IRR register which holds raised interrupt priority
- `uint16_t i8259_read_isr` (void)
End of interrupt instruction, used for ISRs.

6.64.1 Detailed Description

Intel 8259 PIC(Programmable Interrupt Controller) Low Level Driver.

Author

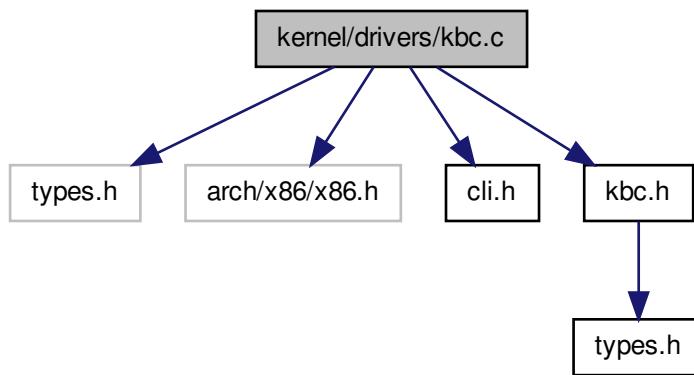
Saad Talaat

Definition in file [i8259.c](#).

6.65 kernel/drivers/kbc.c File Reference

Keyboard driver.

```
#include <types.h>    #include <arch/x86/x86.h>    #include
<cli.h> #include <kbc.h> Include dependency graph for kbc.c:
```



PS/2 Keyboard Driver

This file contains keyboard mapping and input handling functions.

- `#define ESCODE (1<<6)`
- `#define CTL (1<<1)`
- `#define SHIFT (1<<0)`
- `#define ALT (1<<2)`
- `#define CAPSLOCK (1<<3)`
- `#define NUMLOCK (1<<4)`
- `#define SCROLLLOCK (1<<5)`
- `#define NUL 0`
- `#define CL(x) ((x)-'@')`
- static `uint8_t shiftcode [256]`
codes to shift character values.
- static `uint8_t togglecode [256]`
button codes that toggle button values
- static `uint8_t normalmap [256]`
the normal keyboard map.
- static `uint8_t shiftmap [256]`

the keyboard map once shiftcode is pressed.

- static `uint8_t ctlmap [256]`
the keyboard map once control is pressed.
- static `uint8_t * charcode [4]`
- `int kbc_data (void)`
keyboard input controller driver
- void `kbc_interrupt (void)`
calls console interrupt which is used in character read.

6.65.1 Detailed Description

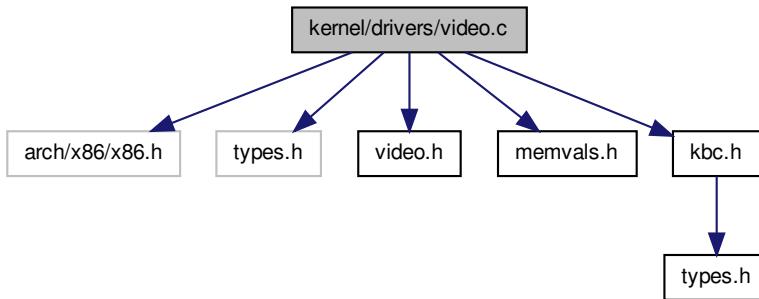
Keyboard driver.

Definition in file [kbc.c](#).

6.66 kernel/drivers/video.c File Reference

Color Graphics Adapter (CGA) driver.

```
#include <arch/x86/x86.h>  #include <types.h>  #include
<video.h> #include <memvals.h> #include <kbc.h> Include de-
pendency graph for video.c:
```



CGA Driver

An early initialization for the console can be useful in debugging and logging operations. here screen is initialized and get ready for I/O operations.

- static `cga_attr`

holds the colors attribute of the written character

- `uint16_t cga_get_pos (void)`
gets the cursor position on the screen.
- `void cga_set_attr (uint16_t c)`
sets global color attribute to a given value.
- `void cga_clear (void)`
clears the screen.
- `void cga_set_pos (uint16_t pos)`
sets the position to a given value on the screen map
- `void cga_init (void)`
initiates CGA.
- `void cga_putc (int c)`
writes a character to screen using cga.
- `void cga_putstr (char *c)`
uses cga_putc to write a string

6.66.1 Detailed Description

Color Graphics Adapter (CGA) driver.

Author

Saad Talaat

Date

Sunday 30/10/2011

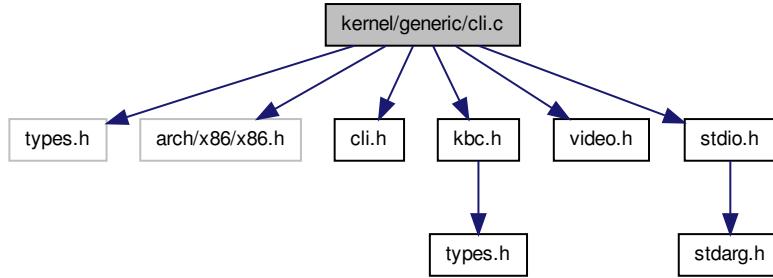
Definition in file [video.c](#).

6.67 kernel/generic/cli.c File Reference

CLI I/O facilities.

```
#include <types.h>    #include <arch/x86/x86.h>    #include
<cli.h>    #include <kbc.h>    #include <video.h>    #include
```

<stdio.h> Include dependency graph for cli.c:



Functions

Console

A generic wrapping for the display I/O.

- void `console_init` (void)
initializes console.
- void `console_interrupt` (int(*intr)(void))
*reads a char from a input device interrupt. int (*intr)(void) interrupt handler.*
- void `console_clear` (void)
- int `console_getc` (void)
reads a character from an input device [keyboard].
- void `console_putc` (int c)
writes a character to screen wraps cga_putc function
- void `putchr` (int c)
writes a character to screen.
- int `getchar` (void)
recives a character from keyboard and write it to screen and buffer.

6.67.1 Detailed Description

CLI I/O facilities.

Author

Saad Talaat

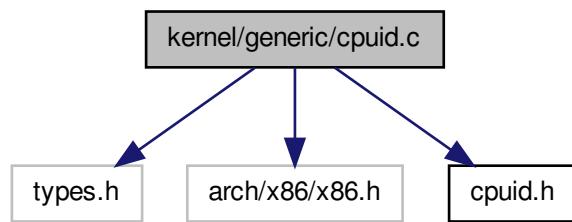
Date

Thursday 03/11/2011

Definition in file [cli.c](#).

6.68 kernel/generic/cpuid.c File Reference

```
#include <types.h>    #include <arch/x86/x86.h>    #include  
<cpuid.h> Include dependency graph for cpuid.c:
```



Data Structures

- struct [cpuid_regs](#)

Functions

- [uint32_t cpuid_get_ebx \(void\)](#)
- [uint32_t cpuid_get_ecx \(void\)](#)
- [uint32_t cpuid_get_edx \(void\)](#)
- [uint32_t cpuid_get_eax \(void\)](#)
- [static void make_cpuid \(uint32_t *eax, uint32_t *ebx, uint32_t *ecx, uint32_t *edx\)](#)
- [uint32_t set_eax \(uint32_t value\)](#)
- [uint32_t cpuid_print \(uint32_t ax_value, uint32_t operation\)](#)

6.68.1 Function Documentation

6.68.1.1 [uint32_t cpuid_get_eax \(void \)](#)

6.68.1.2 [uint32_t cpuid_get_ebx \(void \)](#)

6.68.1.3 [uint32_t cpuid_get_ecx \(void \)](#)

6.68.1.4 [uint32_t cpuid_get_edx \(void \)](#)

6.68.1.5 `uint32_t cpuid_print(uint32_t ax_value, uint32_t operation)`

Definition at line 22 of file cpuid.c.

6.68.1.6 `static void make_cpuid(uint32_t *eax, uint32_t *ebx, uint32_t *ecx, uint32_t *edx) [inline, static]`

Definition at line 45 of file cpuid.c.

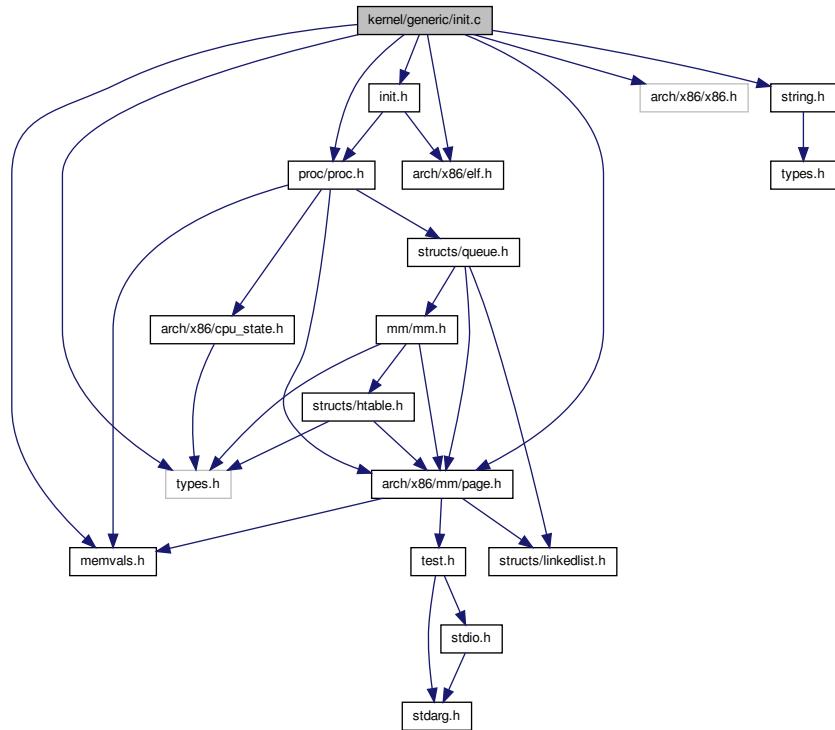
6.68.1.7 `uint32_t set_eax(uint32_t value)`

Definition at line 18 of file cpuid.c.

6.69 kernel/generic/init.c File Reference

```
#include <types.h>    #include <arch/x86/x86.h>    #include
<memvals.h> #include <string.h> #include <arch/x86/mm/page.-  
h>    #include <arch/x86/elf.h>    #include <proc/proc.h> x
```

```
#include <init.h> Include dependency graph for init.c:
```



Functions

- void `Init_userspace (proc_t *proc)`

Variables

- `pde_t * global_pgdir`
- struct `Segdesc catgdt []`
New global descriptor table.
- `proc_t processes []`

6.69.1 Function Documentation

6.69.1.1 void `Init_userspace (proc_t *proc)`

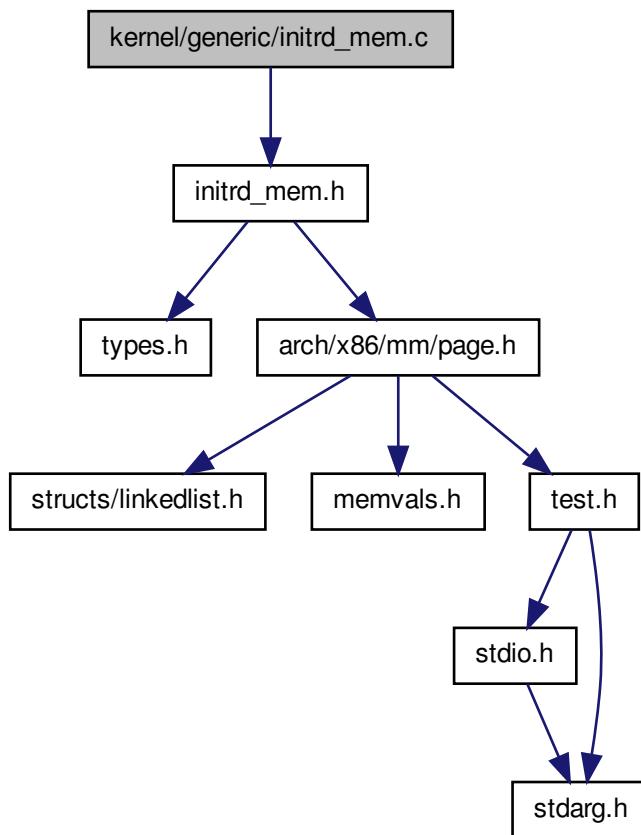
Definition at line 16 of file `init.c`.

6.69.2 Variable Documentation

6.69.2.1 proc_t processes[]

6.70 kernel/generic/initrd_mem.c File Reference

#include <initrd_mem.h> Include dependency graph for initrd_mem.c:



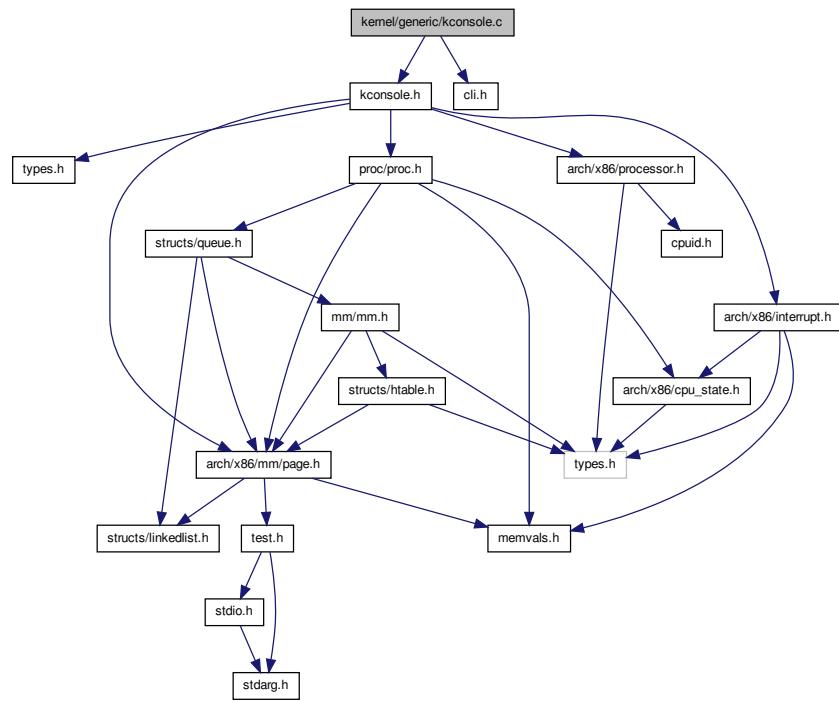
Functions

Initrd

- `void *initrd_mem ()`

6.71 kernel/generic/kconsole.c File Reference

```
#include <kconsole.h> #include <cli.h> Include dependency graph
for kconsole.c:
```



KConsole

- `kcommand_t * kconsole_commands`
- `void kconsole_init (void)`
- `void kconsole (void)`
- `void kcommand_register (kcommand_t *cmd)`
- `kcommand_t * kcommand_match (const char *buf)`
- `void kscheduler_info (void)`
- `void kcpu_info (void)`
- `void kconsole_help (void)`

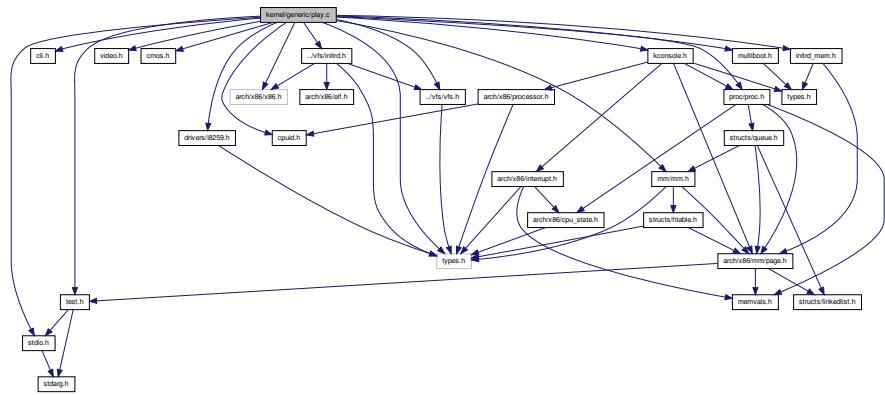
6.72 kernel/generic/play.c File Reference

```
#include <types.h> #include <cli.h> #include <stdio.h>
#include <video.h> #include <cmos.h> #include <arch/x86/x86.-
```

```

h> #include <cpuid.h> #include <proc/proc.h> #include
<test.h> #include <multiboot.h> #include "../vfs/vfs.-
h" #include "../vfs/initrd.h" #include "initrd_mem.h" x
#include <drivers/i8259.h> #include <kconsole.h> #include
<mm/mm.h> Include dependency graph for play.c:

```



Functions

- void `play` (void)

this function holds initializations and setting up facilities
- void `_panic_` (const char *file, int nline, const char *fmt,...)

a completely useless function that prints current time.
- void `time_print` (void)
- `uint8_t initrd` (multiboot_info_t *mboot_ptr, `uint32_t` *initrd_location_ptr, `uint32_t` *initrd_end_ptr)
- void `initrd_test` (`uint32_t` initrd_location)
- void `bootup` (`uint32_t` mboot_ptr)

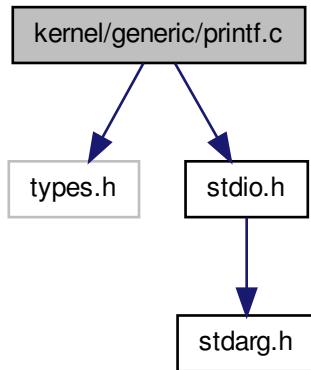
Variables

- static const char * `error_panic` = `NULL`

6.73 kernel/generic/printf.c File Reference

a printf family like functions

```
#include <types.h> #include <stdio.h> Include dependency graph for
printf.c:
```



Printing

- `#define hex2ascii(x) ("0123456789ABCDEF"[x])`
- `static void putch (int c, int *count)`
prints a character at a time.
- `void ksprintkn (void(*func)(int, int *), int *count, uintmax_t num, int base, int width, int padc)`
Prints numbers.
- `int getint (va_list *ap, int lflag)`
gets the number of the va_list on a printk.
- `int getuint (va_list *ap, int lflag)`
same as getint except for the signed nature of getint.
- `int kvprintk (const char *format, void(*func)(int, int *), int *count, va_list ap)`
A BSD like printing function.
- `vprintk (const char *format, va_list ap)`
passes arguments to kvprintk after tokenizing them
- `int printk (const char *format,...)`
An ideal printing function.

6.73.1 Detailed Description

a printf family like functions

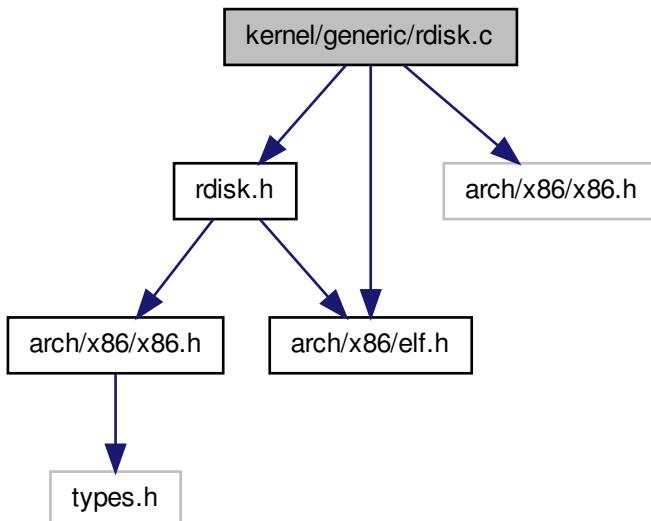
Author

Saad Talaat

Definition in file [printf.c](#).

6.74 kernel/generic/rdisk.c File Reference

```
#include <rdisk.h>    #include <arch/x86/x86.h>    #include
<arch/x86/elf.h> Include dependency graph for rdisk.c:
```



Functions

Bootloader-KernelLoading

- void [readsect](#) (void *, [uint32_t](#))

reads disk at a given offset
- void [readseg](#) ([uint32_t](#), [uint32_t](#), [uint32_t](#))

loads data from disk into memory with data size to read.
- void [waitdisk](#) (void)

hangs till disk operation is done.

6.74.1 Function Documentation

6.74.1.1 void readsect (void * *dst*, uint32_t *offset*)

reads disk at a given offset

Parameters

<i>void*</i>	<i>va</i> , is the address to load read data into
<i>uint32_t</i>	<i>offset</i> , offset in image

Returns

nothing.

Definition at line 36 of file rdisk.c.

6.74.1.2 void readseg (uint32_t *va*, uint32_t *count*, uint32_t *offset*)

loads data from disk into memory with data size to read.

Parameters

<i>uint32_t</i>	<i>va</i> , virtual address to load data into
<i>uint32_t</i>	<i>count</i> , the number of sectors to load into memory.
<i>uint32_t</i>	<i>offset</i> , the offset of data into disk.

Returns

nothing.

It could be called as a wrapper function to readsect, since readseg uses readsect int its operations.

Definition at line 11 of file rdisk.c.

6.74.1.3 void waitdisk (void)

hangs till disk operation is done.

Returns

nothing.

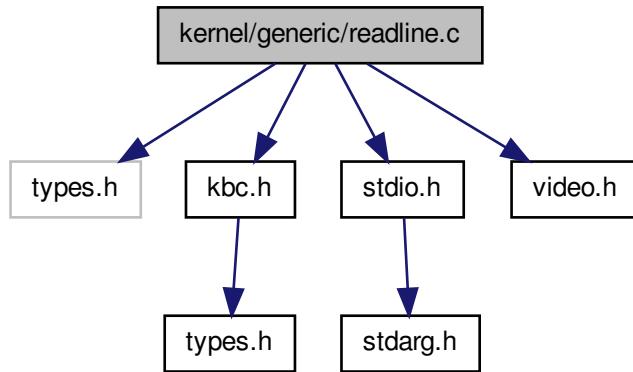
this function just reads the disk port status register holds the value of 0x40 which means disk is not executing a command and ready.

Definition at line 57 of file rdisk.c.

6.75 kernel/generic/readline.c File Reference

readline operation.

```
#include <types.h> #include <kbc.h> #include <stdio.h> x
#include <video.h> Include dependency graph for readline.c:
```



Defines

- `#define BUflen 1024`

Functions

- `char * readline (const char *towrite)`
reads a string from keyboard into screen.

Variables

- `static char buf [BUflen]`

6.75.1 Detailed Description

readline operation.

Date

Thursday 03/11/2011

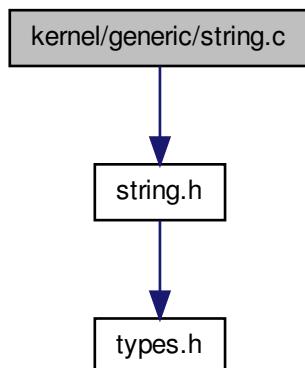
Readline is used in user interaction which is used in late console. can be used in the future as a system call.

Definition in file [readline.c](#).

6.76 kernel/generic/string.c File Reference

String and memory operations, originally from linux.

#include <string.h> Include dependency graph for string.c:



Functions

Strings and Memory.

Basic copying and sizing and setting operations.

- **uint32_t strnlen** (const char *str, **uint32_t** count)
the function loops over the string character values till '\0' is reached or reached max value.
- **void * memcpy** (void *dst, const void *src, **uint32_t** count)
copies bytes from a memory address to another memory address
- **void * memset** (void *ptr, **int** c, **uint32_t** count)
sets a number of bytes starting from a given pointer to a given character.
- **int strcmp** (const char *s1, const char *s2)

6.76.1 Detailed Description

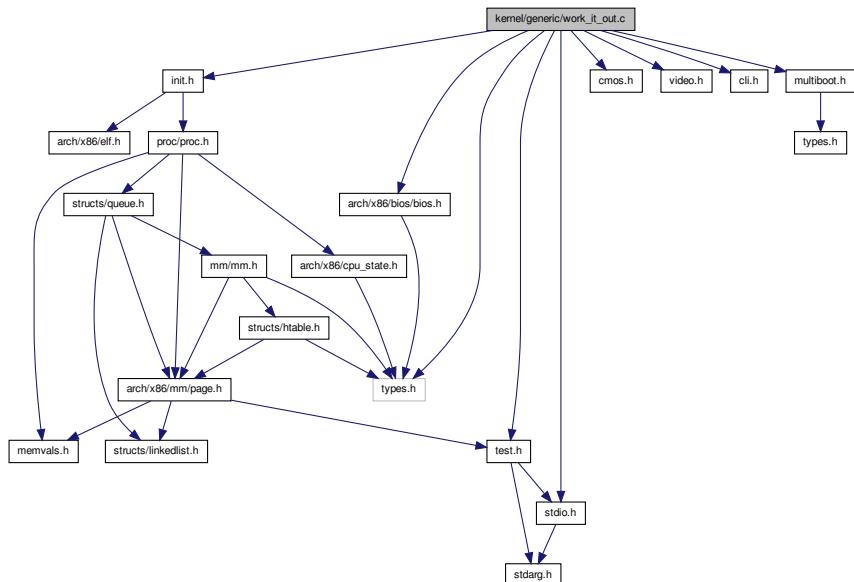
String and memory operations, originally from linux.

Definition in file [string.c](#).

6.77 kernel/generic/work_it_out.c File Reference

kernel device facilities initialization and set up start.

```
#include <types.h> #include <cmos.h> #include <video.h>
#include <cli.h> #include <stdio.h> #include <test.h>
#include <init.h> #include <multiboot.h> #include
<arch/x86/bios/bios.h> Include dependency graph for work_it_out.c:
```



Functions

Mainflow.

After kernel is loaded into IP, it fetches into `_start` function which makes a soft reboot by the bios, and reloads a new global descriptor table and resets segments. then just before it jumps to the `work_it_out` function it sets stack up and jumps to earlier mentioned function.

- void [work_it_out \(uint32_t mboot_ptr\)](#)

6.77.1 Detailed Description

kernel device facilities initialization and set up start.

Author

Saad Talaat

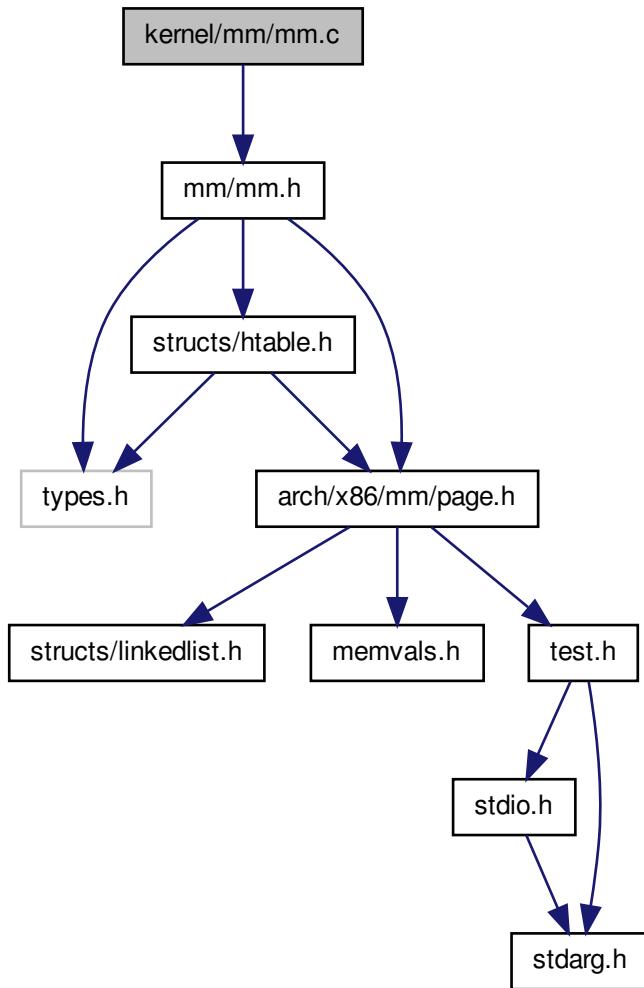
Date

Thursday, 27/10/2011

Definition in file [work_it_out.c](#).

6.78 kernel/mm/mm.c File Reference

#include <mm/mm.h> Include dependency graph for mm.c:



Memory allocation/deallocation

Memory Management

Memory allocation is done by initiating an early hash table in which every page is stored

as a bucket, within every bucket the size of memory allocated on that page sequentially.
e.g. [Bucket:1]->[heap1:4]->[heap2:32]->[heap3:1024]->... in the first page 3 heaps
are allocated 1st with size 4 bytes 2nd with size 32 byte and third with size 1Kb.

- `htable_t * mm_table`
- `uint32_t mm_va_base`
- `uint32_t cur_bucket`
- `void kmalloc_test (void)`
- `uint32_t mm_hash (uint32_t key)`
hash function for allocated memory entries in heaps hashtable
- `uint32_t mm_destroy (uint32_t destroy)`
destroy function to free unwanted entries
- `void mm_init (void)`
memory heap manager initialization function
- `static uint32_t mm_insert (uint32_t va, uint32_t size)`
- `void * kmalloc (uint32_t size)`
- `void kfree (void *va)`

6.78.1 Detailed Description

Author

Saad Talaat

Definition in file [mm.c](#).

6.79 kernel/perf/sort/mergesort.c File Reference

Merge Sort implementation.

Functions

Sorting(Merge)

- `void mergesort (uint32_t array[], uint32_t temp[], uint32_t left, uint32_t right)`
- `void merge (uint32_t array[], uint32_t temp[], uint32_t left, uint32_t center, uint32_t right)`

6.79.1 Detailed Description

Merge Sort implementation.

Author

Saad Talaat

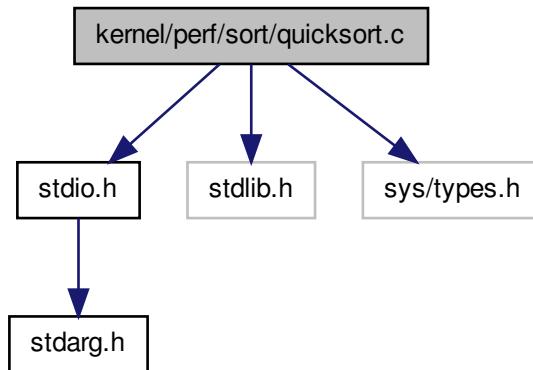
Date

2nd of November, 2012

Definition in file [mergesort.c](#).

6.80 kernel/perf/sort/quicksort.c File Reference

```
#include <stdio.h> #include <stdlib.h> #include <sys/types.h>
Include dependency graph for quicksort.c:
```



Functions

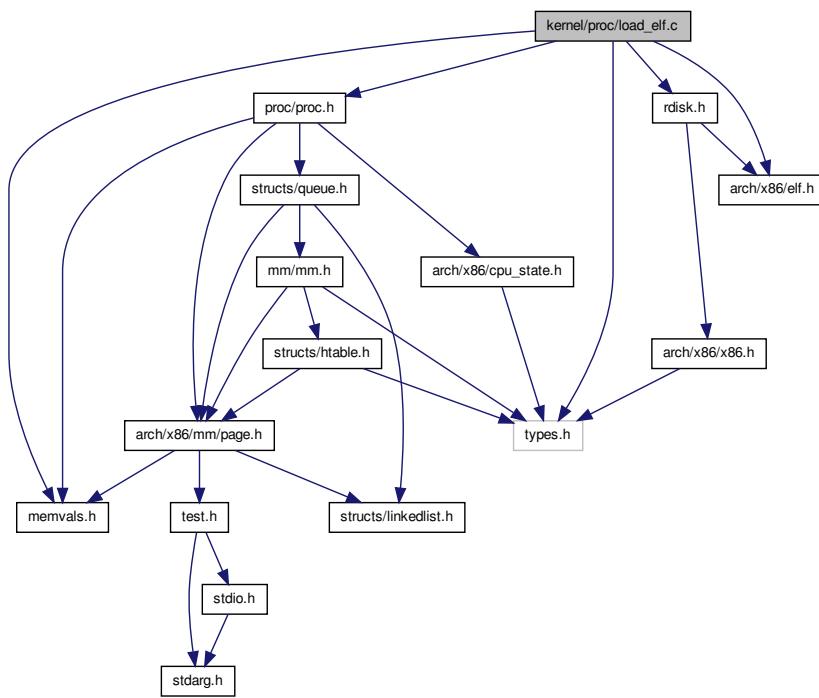
Sorting(Merge)

- static void [swap](#) ([int](#) array[], [int](#) i, [int](#) j)
- static [int](#) [partition](#) ([int](#) array[], [int](#) low, [int](#) high)
- static void [quick_sort](#) ([int](#) array[], [int](#) low, [int](#) high)
- void [quicksort](#) ([int](#) array[], [int](#) size)
- [int main](#) ([void](#))

6.81 kernel/proc/load_elf.c File Reference

Elf binary loader.

```
#include <types.h>    #include <arch/x86/elf.h>    #include
<memvals.h>  #include <proc/proc.h>  #include <rdisk.h>
Include dependency graph for load_elf.c:
```



Functions

Binary Loader.

- `uint32_t elf_load_to_proc (proc_t *proc, uint32_t offset)`
loads a binary into a proc from an offset into image.

6.81.1 Detailed Description

Elf binary loader.

Author

Saad Talaat

Date

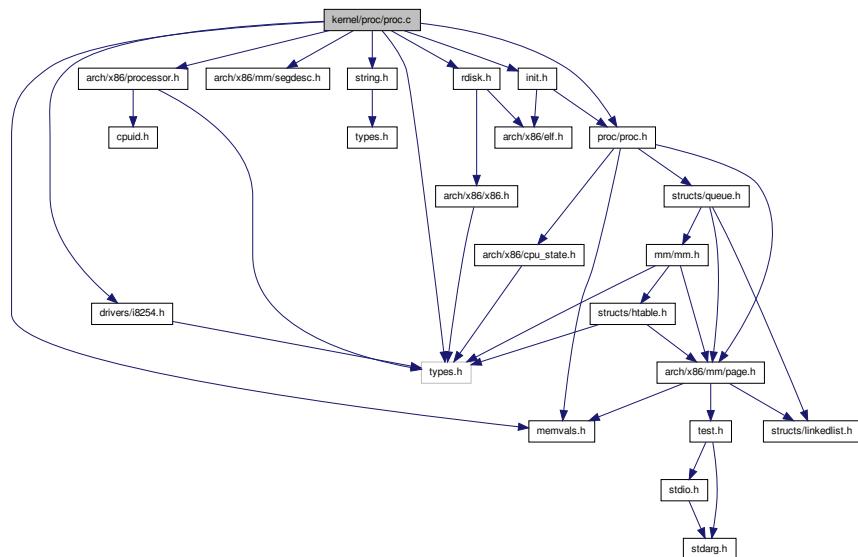
Tuesday 16/10/2012

Definition in file [load_elf.c](#).

6.82 kernel/proc/proc.c File Reference

Process Manager.

```
#include <types.h> #include <memvals.h> #include <proc/proc.-  
h> #include <arch/x86/mm/segdesc.h> #include <arch/x86/processor.-  
h> #include <string.h> #include <init.h> #include <rdisk.-  
h> #include <drivers/i8254.h> Include dependency graph for proc.c:
```



Process Manager.

- `pde_t * global_pgdir`
- struct Proc_List `empty_procs`
- `proc_t * proc_table`
- struct Proc_Lifo `running_procs`
- `FIFO_HEAD (ready_procs, proc, 256)`
- void `proc_printinfo (void)`
Initialize the proc array.
- void `init_proc_table (void)`
- `uint32_t create_proc (proc_t **proc_s)`

Initialize and create a proc information block.

- void [proc_ready](#) (proc_t *proc)

- uint32_t [proc_alloc_mem](#) (proc_t *proc, void *va, uint32_t len)

allocates memory thunk for a proc.

- uint32_t [init_proc0](#) ()

loads and initiated the proc0(first proc) from image.

- void [init_proc](#) (void)

initiates the process table and tests queues and initiates proc0

- void [test_lifo](#) (void)

Tests the LIFO queues data structure functionality.

- void [test_fifo](#) (void)

Tests the FIFO queue data structure.

- void [switch_address_space](#) (proc_t *proc_to_run)

switches between the kernel and a given proc

- void [sched_init](#) (void)

- void [schedule](#) (void)

main scheduling function

6.82.1 Detailed Description

Process Manager.

Author

Saad Talaat

Date

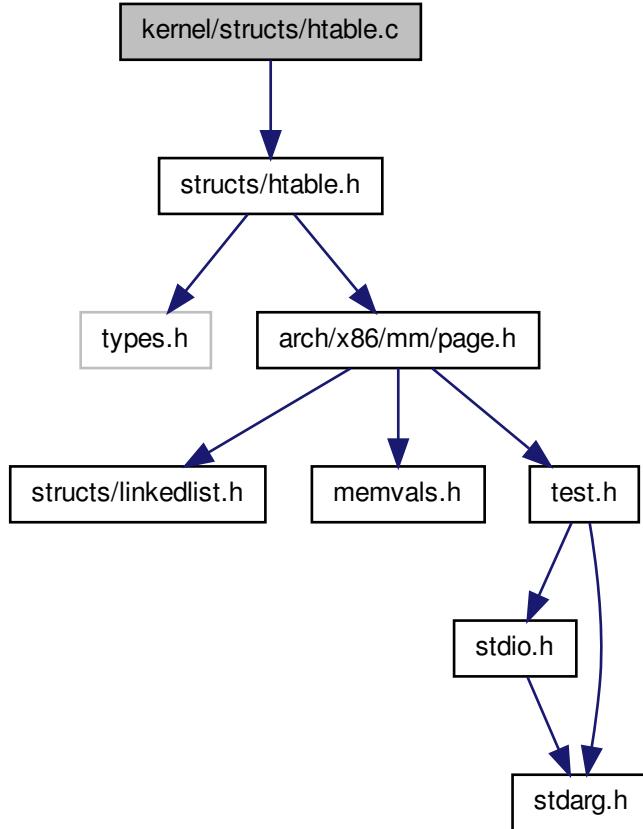
Tuesday 16/10/2012

Definition in file [proc.c](#).

6.83 kernel/structs/htable.c File Reference

Chained hash table.

```
#include <structs/htable.h> Include dependency graph for htable.c:
```



Functions

- `uint32_t early_htable_init (htable_t *table, uint32_t buckets_count, uint32_t bucket_length, uint32_t(*hash)(uint32_t), uint32_t(*destroy)(uint32_t))`
Initializes new hash table.
- `void htable_init (htable_t *table, uint32_t buckets_count, uint32_t(*hash)(uint32_t), uint32_t(*destroy)(uint32_t))`
Initializes new hash table.
- `void htable_destroy (htable_t *table)`
Deletes hash table
- `uint32_t htable_insert (htable_t *table, uint32_t key, uint32_t value)`
- `uint32_t htable_remove (htable_t *table, uint32_t key)`

Inserts value into the hash table.

- `uint32_t htable_get (htable_t *table, uint32_t key)`

hash table look up function with $O(n)$

6.83.1 Detailed Description

Chained hash table.

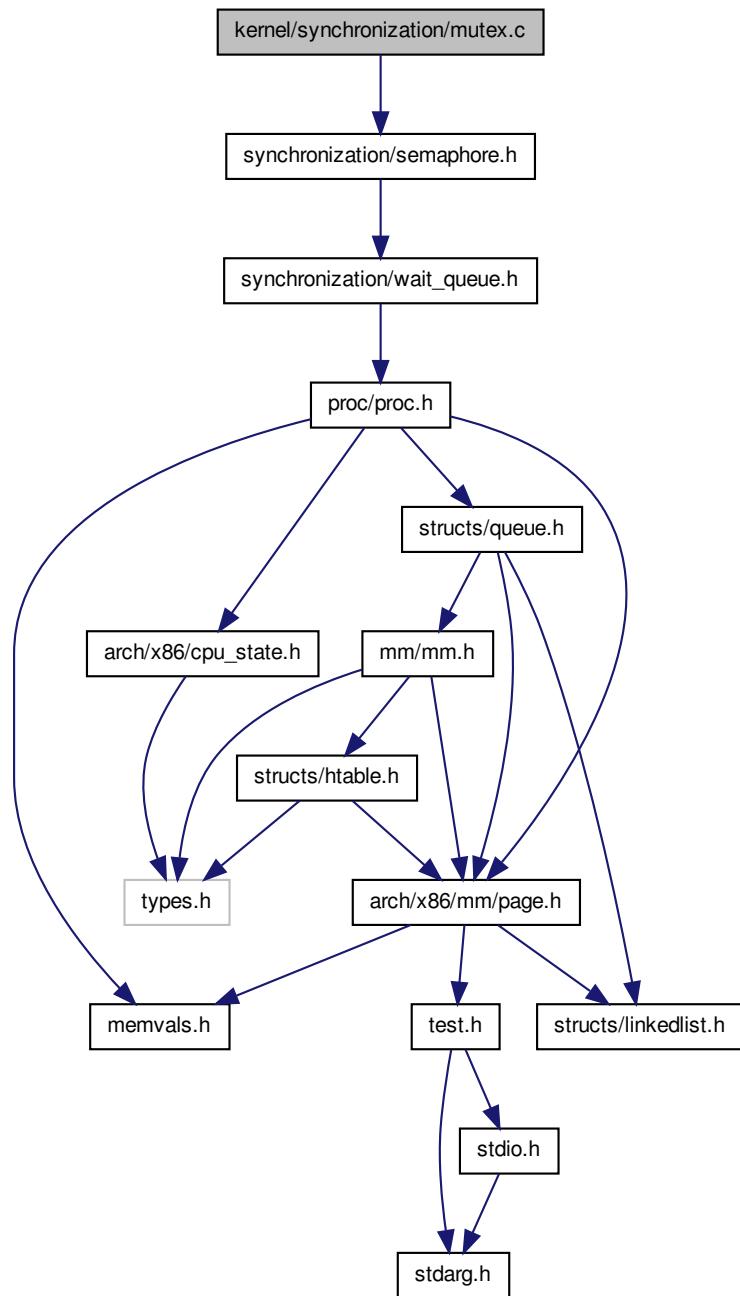
Author

Saad Talaat

Definition in file [htable.c](#).

6.84 kernel/synchronization/mutex.c File Reference

```
#include <synchronization/semaphore.h> Include dependency graph
for mutex.c:
```



Functions

Mutex

Mutex synchronization primitives

- void [mutex_init \(semaphore_t *s\)](#)
initialization to mutex structs and values.
- void [mutex_lock \(semaphore_t *s, proc_t *proc\)](#)
blocks a process over a mutex
- void [mutex_unlock \(semaphore_t *s\)](#)
sets binary resource to free

6.84.1 Detailed Description

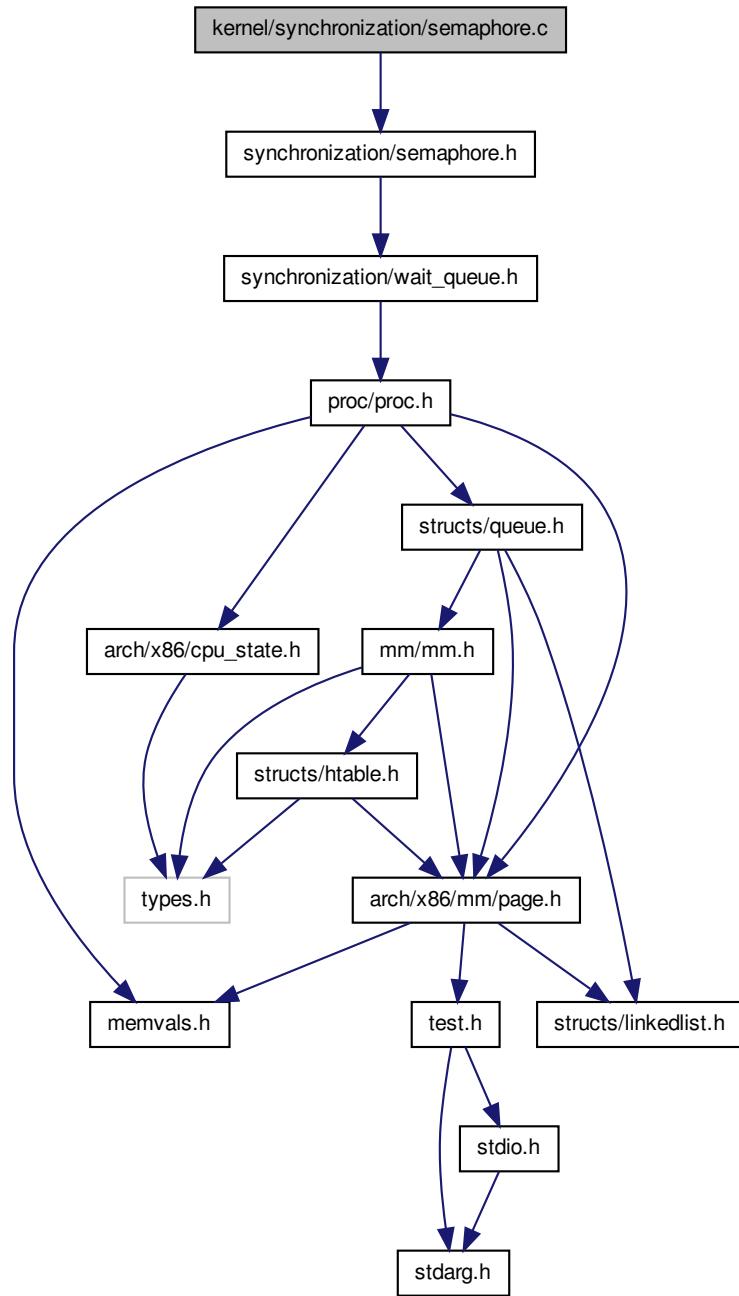
Author

Saad Talaat

Definition in file [mutex.c](#).

6.85 kernel/synchronization/semaphore.c File Reference

```
#include <synchronization/semaphore.h> Include dependency graph
for semaphore.c:
```



Functions

Semaphores

Semaphore synchronization primitives

- void [semaphore_init \(semaphore_t *s, uint32_t num\)](#)

Initialization of a semaphore primitive function.

- void [semaphore_down \(semaphore_t *s, proc_t *proc\)](#)

does down operation over semaphore primitive

- void [semaphore_up \(semaphore_t *s\)](#)

Up function for a semaphore.

6.85.1 Detailed Description

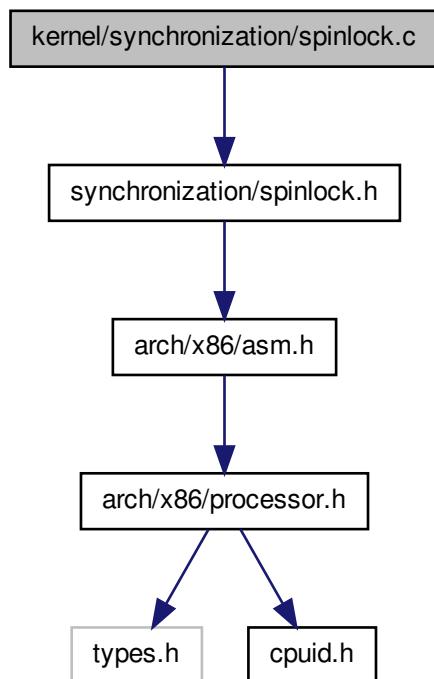
Author

Saad Talaat

Definition in file [semaphore.c](#).

6.86 kernel/synchronization/spinlock.c File Reference

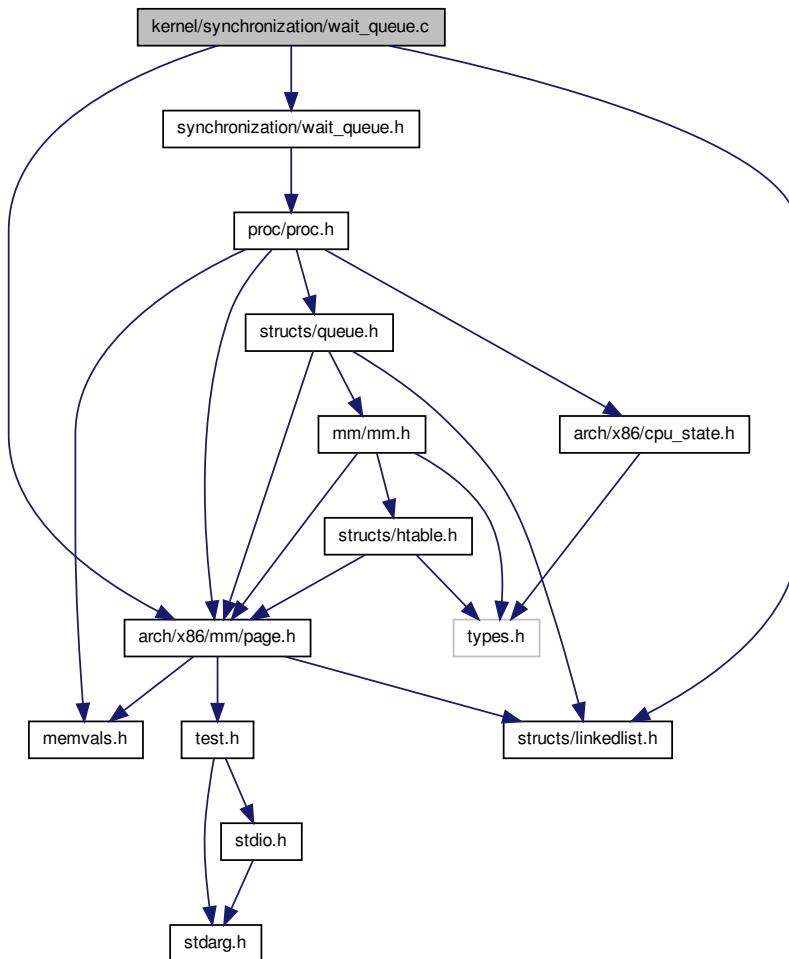
```
#include <synchronization/spinlock.h> Include dependency graph for  
spinlock.c:
```



6.87 kernel/synchronization/wait_queue.c File Reference

```
#include <structs/linkedlist.h> #include <synchronization/wait-  
_queue.h> #include <arch/x86/mm/page.h> Include dependency graph
```

for wait_queue.c:



Functions

Wait queues

Wait queues

- void `wait_init` (struct Proc_List *waiting_procs)
Waiting procs linked list initialization function.
- void `wait_sleep` (struct Proc_List *list, proc_t *proc, uint32_t ticks)
function that blocks a process
- void `wait_update` (void)
blocked process updating function

- void [wait_wakeup](#) (struct Proc_List *list)
wakes up a process once timeout or resource free

6.87.1 Detailed Description

Author

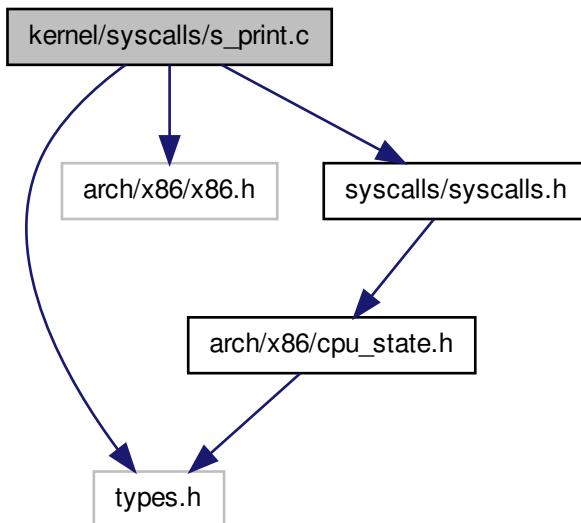
Saad Talaat

Definition in file [wait_queue.c](#).

6.88 kernel/syscalls/s_print.c File Reference

The actual printf procedure -to be implemented- .

```
#include <types.h>    #include <arch/x86/x86.h>    #include
<syscalls/syscalls.h> Include dependency graph for s_print.c:
```



Functions

Print procedure .

- [int32_t sys_printf](#) (char *str)

The actual print procedure requested by the printf system call.

6.88.1 Detailed Description

The actual printf procedure -to be implemented- .

Author

Menna Essa

Date

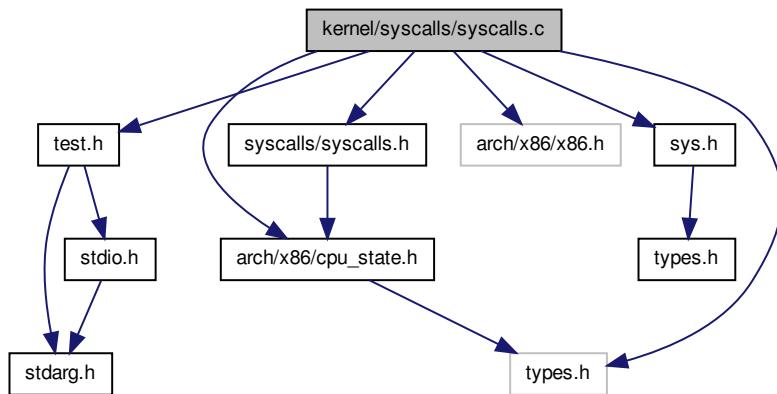
Thursday 25/10/2012

Definition in file [s_print.c](#).

6.89 kernel/syscalls/syscalls.c File Reference

maps calls into their respective handlers.

```
#include <types.h> #include <test.h> #include <sys.h>
#include <arch/x86/x86.h> #include <arch/x86/cpu_state.h> #include <syscalls/syscalls.h> Include dependency graph for syscalls.c:
```



System calls handler .

- `fnptr_t sys_call_table [] = { sys_printf }`

Declare system calls table with pointers to calls handlers.

- `int32_t n_calls` = `sizeof(sys_call_table)/sizeof(int32_t)`
keeps the size of the table for boundary checking.
- `int32_t s_errno` = 0
Keeps error codes to return to the caller.
- `int32_t map_syscall (cpu_state_t *cpu_state)`
maps the system call by offseting the sys_call_table structure using the offset in eax register.

6.89.1 Detailed Description

maps calls into their respective handlers.

Author

Menna Essa

Date

Thursday 25/10/2012

Definition in file [syscalls.c](#).

6.90 kernel/time/periodic.c File Reference

Periodic interrupt handler.

Functions

Periodic_Timer

- `void time_handler (void)`

6.90.1 Detailed Description

Periodic interrupt handler. Wakes up sleeping processes and executes the scheduler according to the scheduling states

Author

Saad Talaat

Definition in file [periodic.c](#).

6.91 kernel/tmp/init_elf.c File Reference

Functions

- `int main (void)`

6.91.1 Function Documentation

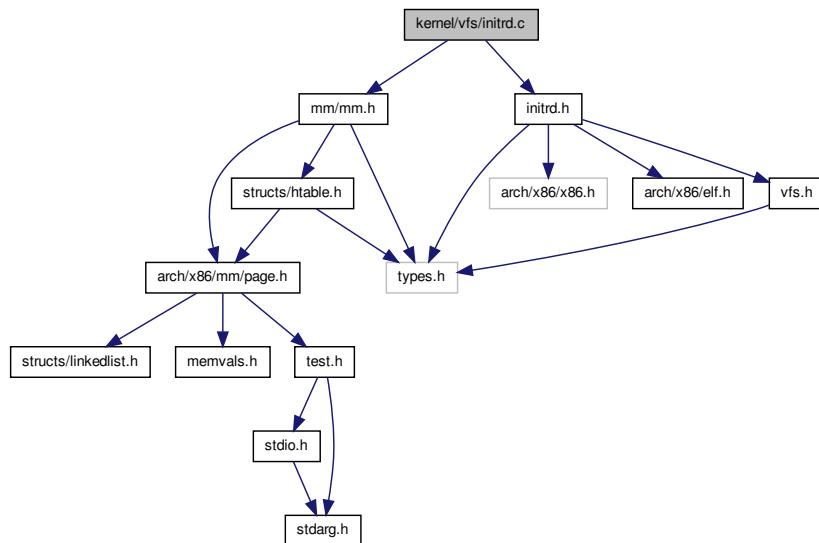
6.91.1.1 `int main (void)`

Definition at line 3 of file init_elf.c.

6.92 kernel/vfs/initrd.c File Reference

Initrd.

```
#include <mm/mm.h> #include "initrd.h" Include dependency graph
for initrd.c:
```



Initrd

- static `initrd_header_t * initrd_header = NULL`
- static `initrd_file_header_t * file_headers = NULL`
- static `vfs_node_t * initrd_root = NULL`

- static `vfs_node_t` * `initrd_dev` = `NULL`
- static `vfs_node_t` * `root_nodes` = `NULL`
- `int nroot_nodes`
- `direntry_t dirent`
- `uint32_t initrd_read (vfs_node_t *node, uint32_t offset, uint32_t size, uint8_t *buffer)`

Read operation implementation , reads the offset from the node structure , Reads data of size specified by the call ; after checking that the file is legal.

- static `direntry_t * initrd_readdir (vfs_node_t *node, uint32_t index)`

Directory traversal , reads next directory ; still immature.

- static `vfs_node_t * initrd_faddir (vfs_node_t *node, char *name)`

Directory traversal , searches for a direcotry , very Immature.

- `vfs_node_t * initialise_initrd (uint32_t location)`

Initializes file systems structures (root directory , /dev direcotry and file nodes) to be able to preform files system operations.

6.92.1 Detailed Description

Initrd. Initrd file system operations specifications and implementation.

Author

Saad Talaat

Date

Saturday 08/06/2013

Author

Menna Essa

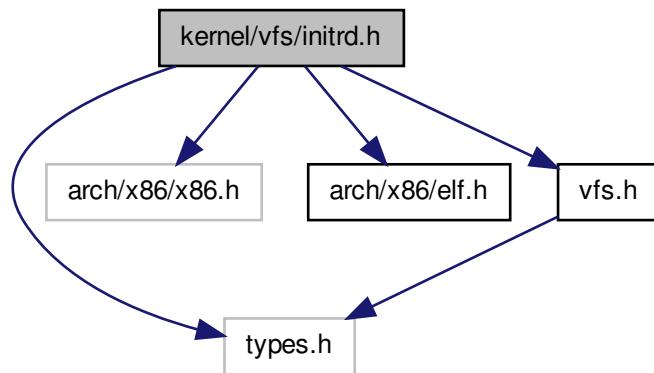
Definition in file [initrd.c](#).

6.93 kernel/vfs/initrd.h File Reference

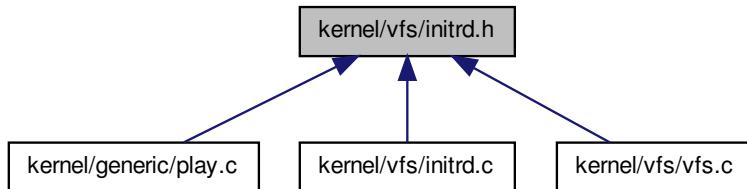
Initrd file system headers.

```
#include <types.h>    #include <arch/x86/x86.h>    #include
<arch/x86/elf.h> #include "vfs.h" Include dependency graph for initrd.-
```

h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `initrd_header_t`
- struct `initrd_file_header_t`

Functions

- `vfs_node_t * initialise_initrd (uint32_t location)`

Initializes file systems structures (root directory , /dev direcotry and file nodes) to be able to preform files system operations.

6.93.1 Detailed Description

Initrd file system headers.

Author

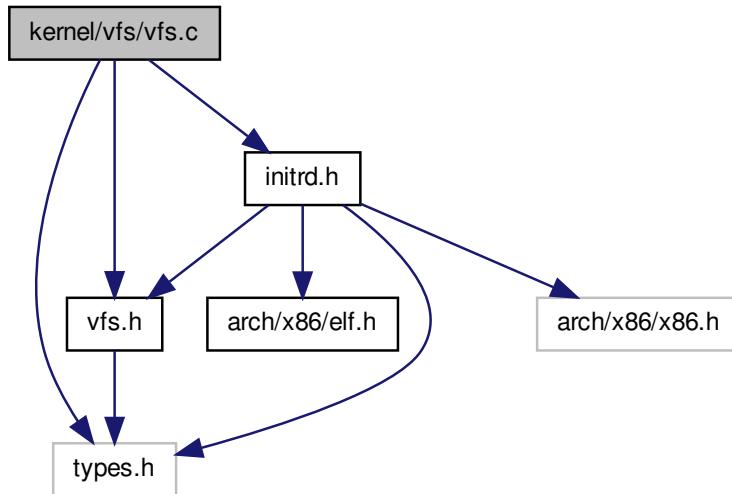
Menna Essa

Definition in file [initrd.h](#).

6.94 kernel/vfs/vfs.c File Reference

Virtual file system standards , In order to abstract different file system calls.

```
#include "vfs.h" #include "initrd.h" #include <types.h> ×
Include dependency graph for vfs.c:
```



Initrd

- `vfs_node_t * fs_root = 0`
- `uint32_t read_fs (vfs_node_t *node, uint32_t offset, uint32_t size, uint8_t *buffer)`

Calls the node's read operation , saved as read in the node structure.

- `uint32_t write_fs (vfs_node_t *node, uint32_t offset, uint32_t size, uint8_t *buffer)`
Calls the node's write operation , saved as write in the node structure.
- `uint8_t open_fs (vfs_node_t *node, uint8_t read, uint8_t write)`
Calls the node's open operation , saved as write in the node structure.
- `uint8_t close_fs (vfs_node_t *node)`
Calls the node's open operation , saved as write in the node structure.
- `direntry_t * readdir_fs (vfs_node_t *node, uint32_t index)`
Calls the node's read directory operation , saved as readdir in the node structure.
- `vfs_node_t * finddir_fs (vfs_node_t *node, char *name)`
Calls the node's find directory operation , save as finddir in the node strucutre.

6.94.1 Detailed Description

Virtual file system standards , In order to abstract different file system calls.

Author

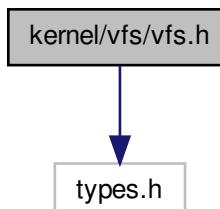
Menna Essa

Definition in file [vfs.c](#).

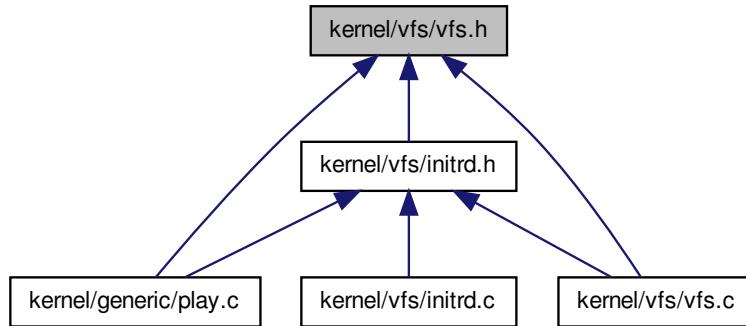
6.95 kernel/vfs/vfs.h File Reference

Standardized file system structures and constants.

#include <types.h> Include dependency graph for vfs.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [direntry_t](#)
- struct [vfs_node](#)

Functions

- `uint32_t read_fs (vfs_node_t *node, uint32_t offset, uint32_t size, uint8_t *buffer)`
Calls the node's read operation , saved as read in the node structure.
- `uint32_t write_fs (vfs_node_t *node, uint32_t offset, uint32_t size, uint8_t *buffer)`
Calls the node's write operation , saved as write in the node structure.
- `uint8_t open_fs (vfs_node_t *node, uint8_t read, uint8_t write)`
Calls the node's open operation , saved as write in the node structure.
- `uint8_t close_fs (vfs_node_t *node)`
Calls the node's open operation , saved as write in the node structure.
- `direntry_t * readdir_fs (vfs_node_t *node, uint32_t index)`
Calls the node's read directory operation , saved as readdir in the node structure.
- `vfs_node_t * finddir_fs (vfs_node_t *node, char *name)`
Calls the node's find directory operation , save as finddir in the node strucutre.

Variables

- `vfs_node_t * fs_root`

Initrd

- #define `FS_FILE` 0x01
- #define `FS_DIRECTORY` 0x02
- #define `FS_CHARDEVICE` 0x03
- #define `FS_BLOCKDEVICE` 0x04
- #define `FS_PIPE` 0x05
- #define `FS_SYMLINK` 0x06
- #define `FS_MOUNTPOINT` 0x08
- typedef `uint32_t`(* `read_type_t`)(`struct vfs_node` *, `uint32_t`, `uint32_t`, `uint8_t` *)
- typedef `uint32_t`(* `write_type_t`)(`struct vfs_node` *, `uint32_t`, `uint32_t`, `uint8_t` *)
- typedef `uint8_t`(* `open_type_t`)(`struct vfs_node` *, `uint32_t`, `uint32_t`)
- typedef `uint8_t`(* `close_type_t`)(`struct vfs_node` *)
- typedef `dirent_t` *(* `readdir_type_t`)(`struct vfs_node` *, `uint32_t`)
- typedef struct `vfs_node` *(* `finddir_type_t`)(`struct vfs_node` *, `char` *name)
- typedef struct `vfs_node` `vfs_node_t`

6.95.1 Detailed Description

Standardized file system structures and constants.

Author

Menna Essa

Definition in file `vfs.h`.