XV6 0.1

Generated by Doxygen 1.9.6

1 ENVIRONMENT PREPARETION	1
1.1 LINUX	1
1.2 BUILDING AND RUNNING XV6	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	9
4.1 backemd	9
4.1.1 Detailed Description	9
4.1.2 Member Data Documentation	9
4.1.2.1 cmd	9
4.1.2.2 type	9
4.2 buf	10
4.2.1 Detailed Description	10
4.2.2 Member Data Documentation	10
4.2.2.1 blockno	10
4.2.2.2 data	10
4.2.2.3 dev	11
4.2.2.4 flags	11
4.2.2.5 lock	11
4.2.2.6 next	11
4.2.2.7 prev	11
4.2.2.8 gnext	12
•	
4.2.2.9 refcnt	12
4.3 cmd	12
4.3.1 Detailed Description	12
4.3.2 Member Data Documentation	12
4.3.2.1 type	12
4.4 context	13
4.4.1 Detailed Description	13
4.4.2 Member Data Documentation	13
4.4.2.1 ebp	13
4.4.2.2 ebx	13
4.4.2.3 edi	13
4.4.2.4 eip	14
4.4.2.5 esi	14
4.5 cpu	14
4.5.1 Detailed Description	14
4.5.2 Member Data Documentation	14

4.5.2.1 apicid	1
4.5.2.2 gdt	1
4.5.2.3 intena	1
4.5.2.4 ncli	1
4.5.2.5 proc	1
4.5.2.6 scheduler	1
4.5.2.7 started	1
4.5.2.8 ts	1
4.6 devsw	1
4.6.1 Detailed Description	1
4.6.2 Member Data Documentation	1
4.6.2.1 read	1
4.6.2.2 write	1
4.7 dinode	1
4.7.1 Detailed Description	1
4.7.2 Member Data Documentation	1
4.7.2.1 addrs	1
4.7.2.2 major	1
4.7.2.3 minor	1
4.7.2.4 nlink	1
4.7.2.5 size	1
4.7.2.6 type	1
4.8 dirent	1
4.8.1 Detailed Description	1
4.8.2 Member Data Documentation	1
4.8.2.1 inum	1
4.8.2.2 name	1
4.9 elfhdr	2
4.9.1 Detailed Description	2
4.9.2 Member Data Documentation	2
4.9.2.1 ehsize	2
4.9.2.2 elf	2
4.9.2.3 entry	2
4.9.2.4 flags	2
4.9.2.5 machine	2
4.9.2.6 magic	2
4.9.2.7 phentsize	2
4.9.2.8 phnum	2
4.9.2.9 phoff	2
4.9.2.10 shentsize	2
4.9.2.11 shnum	2
4.9.2.12 shoff	2

4.9.2.13 shstrndx	22
4.9.2.14 type	22
4.9.2.15 version	23
4.10 execcmd	23
4.10.1 Detailed Description	23
4.10.2 Member Data Documentation	23
4.10.2.1 argv	23
4.10.2.2 eargv	23
4.10.2.3 type	24
4.11 file	24
4.11.1 Detailed Description	24
4.11.2 Member Enumeration Documentation	24
4.11.2.1 anonymous enum	24
4.11.3 Member Data Documentation	25
4.11.3.1 ip	25
4.11.3.2 off	25
4.11.3.3 pipe	25
4.11.3.4 readable	25
4.11.3.5 ref	26
4.11.3.6	26
4.11.3.7 writable	26
4.12 gatedesc	26
4.12.1 Detailed Description	27
4.12.2 Member Data Documentation	27
4.12.2.1 args	27
4.12.2.2 cs	27
4.12.2.3 dpl	27
4.12.2.4 off_15_0	27
4.12.2.5 off_31_16	27
4.12.2.6 p	28
4.12.2.7 rsv1	28
4.12.2.8 s	28
4.12.2.9 type	28
4.13 header	28
4.13.1 Detailed Description	29
4.13.2 Member Data Documentation	29
4.13.2.1 ptr	29
4.13.2.2	29
4.13.2.3 size	29
4.13.2.4 x	29
4.14 inode	30
4.14.1 Detailed Description	30

4.14.2 Member Data Documentation	. 30
4.14.2.1 addrs	. 30
4.14.2.2 dev	. 30
4.14.2.3 inum	. 31
4.14.2.4 lock	. 31
4.14.2.5 major	. 31
4.14.2.6 minor	. 31
4.14.2.7 nlink	. 31
4.14.2.8 ref	. 32
4.14.2.9 size	. 32
4.14.2.10 type	. 32
4.14.2.11 valid	. 32
4.15 ioapic	. 32
4.15.1 Detailed Description	. 33
4.15.2 Member Data Documentation	. 33
4.15.2.1 data	. 33
4.15.2.2 pad	. 33
4.15.2.3 reg	. 33
4.16 kmap	. 33
4.16.1 Detailed Description	. 34
4.16.2 Member Data Documentation	. 34
4.16.2.1 perm	. 34
4.16.2.2 phys_end	. 34
4.16.2.3 phys_start	. 34
4.16.2.4 virt	. 34
4.17 listcmd	. 35
4.17.1 Detailed Description	. 35
4.17.2 Member Data Documentation	. 35
4.17.2.1 left	. 35
4.17.2.2 right	. 35
4.17.2.3 type	. 35
4.18 log	. 36
4.18.1 Detailed Description	. 36
4.18.2 Member Data Documentation	. 36
4.18.2.1 committing	. 36
4.18.2.2 dev	. 36
4.18.2.3 lh	. 36
4.18.2.4 lock	. 37
4.18.2.5 outstanding	. 37
4.18.2.6 size	. 37
4.18.2.7 start	. 37
4.19 logheader	. 37

4.19.1 Detailed Description	 38
4.19.2 Member Data Documentation	 38
4.19.2.1 block	 38
4.19.2.2 n	 38
4.20 mp	 38
4.20.1 Detailed Description	 39
4.20.2 Member Data Documentation	 39
4.20.2.1 checksum	 39
4.20.2.2 imcrp	 39
4.20.2.3 length	 39
4.20.2.4 physaddr	 39
4.20.2.5 reserved	 40
4.20.2.6 signature	 40
4.20.2.7 specrev	 40
4.20.2.8 type	 40
4.21 mpconf	 40
4.21.1 Detailed Description	 41
4.21.2 Member Data Documentation	 41
4.21.2.1 checksum	 41
4.21.2.2 entry	 41
4.21.2.3 lapicaddr	 41
4.21.2.4 length	 41
4.21.2.5 oemlength	 42
4.21.2.6 oemtable	 42
4.21.2.7 product	 42
4.21.2.8 reserved	 42
4.21.2.9 signature	 42
4.21.2.10 version	 42
4.21.2.11 xchecksum	 43
4.21.2.12 xlength	 43
4.22 mpioapic	 43
4.22.1 Detailed Description	 43
4.22.2 Member Data Documentation	 43
4.22.2.1 addr	 43
4.22.2.2 apicno	 44
4.22.2.3 flags	 44
4.22.2.4 type	 44
4.22.2.5 version	 44
4.23 mpproc	 44
4.23.1 Detailed Description	 45
4.23.2 Member Data Documentation	 45
4.23.2.1 apicid	 45

4.23.2.2 feature	 45
4.23.2.3 flags	 45
4.23.2.4 reserved	 45
4.23.2.5 signature	 45
4.23.2.6 type	 46
4.23.2.7 version	 46
4.24 pipe	 46
4.24.1 Detailed Description	 46
4.24.2 Member Data Documentation	 46
4.24.2.1 data	 46
4.24.2.2 lock	 47
4.24.2.3 nread	 47
4.24.2.4 nwrite	 47
4.24.2.5 readopen	 47
4.24.2.6 writeopen	 47
4.25 pipecmd	 48
4.25.1 Detailed Description	 48
4.25.2 Member Data Documentation	 48
4.25.2.1 left	 48
4.25.2.2 right	 48
4.25.2.3 type	 48
4.26 proc	 49
4.26.1 Detailed Description	 49
4.26.2 Member Data Documentation	 49
4.26.2.1 chan	 49
4.26.2.2 context	 49
4.26.2.3 cwd	 50
4.26.2.4 killed	 50
4.26.2.5 kstack	 50
4.26.2.6 name	 50
4.26.2.7 ofile	 50
4.26.2.8 parent	 51
4.26.2.9 pgdir	 51
4.26.2.10 pid	 51
4.26.2.11 priority	 51
4.26.2.12 state	 51
4.26.2.13 sz	 52
4.26.2.14 tf	 52
4.27 proghdr	 52
4.27.1 Detailed Description	 52
4.27.2 Member Data Documentation	 52
4.27.2.1 align	 53

4.27.2.2 filesz	. 53
4.27.2.3 flags	. 53
4.27.2.4 memsz	. 53
4.27.2.5 off	. 53
4.27.2.6 paddr	. 54
4.27.2.7 type	. 54
4.27.2.8 vaddr	. 54
4.28 redircmd	. 54
4.28.1 Detailed Description	. 54
4.28.2 Member Data Documentation	. 54
4.28.2.1 cmd	. 55
4.28.2.2 efile	. 55
4.28.2.3 fd	. 55
4.28.2.4 file	. 55
4.28.2.5 mode	. 55
4.28.2.6 type	. 56
4.29 rtcdate	. 56
4.29.1 Detailed Description	. 56
4.29.2 Member Data Documentation	. 56
4.29.2.1 day	. 56
4.29.2.2 hour	. 57
4.29.2.3 minute	. 57
4.29.2.4 month	. 57
4.29.2.5 second	. 57
4.29.2.6 year	. 57
4.30 run	. 58
4.30.1 Detailed Description	. 58
4.30.2 Member Data Documentation	. 58
4.30.2.1 next	. 58
4.31 segdesc	. 58
4.31.1 Detailed Description	. 59
4.31.2 Member Data Documentation	. 59
4.31.2.1 avl	. 59
4.31.2.2 base_15_0	. 59
4.31.2.3 base_23_16	. 59
4.31.2.4 base_31_24	. 59
4.31.2.5 db	. 59
4.31.2.6 dpl	. 60
4.31.2.7 g	
4.31.2.8 lim_15_0	
4.31.2.9 lim_19_16	. 60
4.31.2.10 p	. 60

4.31.2.11 rsv1	 . 60
4.31.2.12 s	 . 61
4.31.2.13 type	 . 61
4.32 sleeplock	 . 61
4.32.1 Detailed Description	 . 61
4.32.2 Member Data Documentation	 . 61
4.32.2.1 lk	 . 61
4.32.2.2 locked	 . 62
4.32.2.3 name	 . 62
4.32.2.4 pid	 . 62
4.33 spinlock	 . 62
4.33.1 Detailed Description	 . 62
4.33.2 Member Data Documentation	 . 63
4.33.2.1 cpu	 . 63
4.33.2.2 locked	 . 63
4.33.2.3 name	 . 63
4.33.2.4 pcs	 . 63
4.34 stat	 . 63
4.34.1 Detailed Description	 . 64
4.34.2 Member Data Documentation	 . 64
4.34.2.1 dev	 . 64
4.34.2.2 ino	 . 64
4.34.2.3 nlink	 . 64
4.34.2.4 size	 . 65
4.34.2.5 type	 . 65
4.35 superblock	 . 65
4.35.1 Detailed Description	 . 65
4.35.2 Member Data Documentation	 . 65
4.35.2.1 bmapstart	 . 66
4.35.2.2 inodestart	 . 66
4.35.2.3 logstart	 . 66
4.35.2.4 nblocks	 . 66
4.35.2.5 ninodes	 . 66
4.35.2.6 nlog	 . 67
4.35.2.7 size	 . 67
4.36 taskstate	 . 67
4.36.1 Detailed Description	 . 68
4.36.2 Member Data Documentation	 . 68
4.36.2.1 cr3	 . 68
4.36.2.2 cs	 . 68
4.36.2.3 ds	 . 68
4.36.2.4 eax	 . 69

4.36.2.5 ebp	69
4.36.2.6 ebx	69
4.36.2.7 ecx	69
4.36.2.8 edi	69
4.36.2.9 edx	69
4.36.2.10 eflags	70
4.36.2.11 eip	70
4.36.2.12 es	70
4.36.2.13 esi	70
4.36.2.14 esp	70
4.36.2.15 esp0	70
4.36.2.16 esp1	71
4.36.2.17 esp2	71
4.36.2.18 fs	71
4.36.2.19 gs	71
4.36.2.20 iomb	71
4.36.2.21 ldt	71
4.36.2.22 link	72
4.36.2.23 padding1	72
4.36.2.24 padding10	72
4.36.2.25 padding2	72
4.36.2.26 padding3	72
4.36.2.27 padding4	72
4.36.2.28 padding5	73
4.36.2.29 padding6	73
4.36.2.30 padding7	73
4.36.2.31 padding8	73
4.36.2.32 padding9	73
4.36.2.33 ss	73
4.36.2.34 ss0	74
4.36.2.35 ss1	74
4.36.2.36 ss2	74
4.36.2.37 t	74
4.37 trapframe	
4.37.1 Detailed Description	75
4.37.2 Member Data Documentation	75
4.37.2.1 cs	75
4.37.2.2 ds	
4.37.2.3 eax	76
4.37.2.4 ebp	76
4.37.2.5 ebx	
4.37.2.6 ecx	76

4.37.2.7 edi	 76
4.37.2.8 edx	 77
4.37.2.9 eflags	 77
4.37.2.10 eip	 77
4.37.2.11 err	 77
4.37.2.12 es	 77
4.37.2.13 esi	 78
4.37.2.14 esp	 78
4.37.2.15 fs	 78
4.37.2.16 gs	 78
4.37.2.17 oesp	 78
4.37.2.18 padding1	 78
4.37.2.19 padding2	 79
4.37.2.20 padding3	 79
4.37.2.21 padding4	 79
4.37.2.22 padding5	 79
4.37.2.23 padding6	 79
4.37.2.24 ss	 79
4.37.2.25 trapno	 80
4.38 uproc	 80
4.38.1 Detailed Description	 80
4.38.2 Member Data Documentation	 80
4.38.2.1 name	 80
4.38.2.2 pid	 80
4.38.2.3 ppid	 80
	•
5 File Documentation	81
5.1 asm.h File Reference	81
5.1.1 Macro Definition Documentation	81
5.1.1.1 SEG_ASM	81
5.1.1.2 SEG_NULLASM	81
5.1.1.3 STA_R	82
5.1.1.4 STA_W	82
5.1.1.5 STA_X	82
5.2 asm.h	82
5.3 bio.c File Reference	82
5.3.1 Function Documentation	83
5.3.1.1 bget()	83
5.3.1.2 binit()	84
5.3.1.3 bread()	84
5.3.1.4 brelse()	84
5.3.1.5 bwrite()	 85

5.3.2 Variable Documentation	8
5.3.2.1	8
5.3.2.2 buf	8
5.3.2.3 head	8
5.3.2.4 lock	8
5.4 bio.c	8
5.5 bio.d File Reference	8
5.6 bio.d	8
5.7 bootasm.d File Reference	8
5.8 bootasm.d	8
5.9 bootmain.c File Reference	8
5.9.1 Macro Definition Documentation	8
5.9.1.1 SECTSIZE	8
5.9.2 Function Documentation	8
5.9.2.1 bootmain()	8
5.9.2.2 readsect()	8
5.9.2.3 readseg()	9
5.9.2.4 waitdisk()	9
5.10 bootmain.c	9
5.11 bootmain.d File Reference	9
5.12 bootmain.d	9
5.13 buf.h File Reference	9
5.13.1 Macro Definition Documentation	9
5.13.1.1 B_DIRTY	9
5.13.1.2 B_VALID	9
5.14 buf.h	9
5.15 cat.c File Reference	9
5.15.1 Function Documentation	9
5.15.1.1 cat()	9
5.15.1.2 main()	9
5.15.2 Variable Documentation	9
5.15.2.1 buf	9
5.16 cat.c	9
5.17 cat.d File Reference	9
5.18 cat.d	9
5.19 console.c File Reference	9
5.19.1 Macro Definition Documentation	9
5.19.1.1 BACKSPACE	9
5.19.1.2 C	9
5.19.1.3 CRTPORT	9
5.19.1.4 INPUT_BUF	9
5.19.2 Function Documentation	9

	5.19.2.1 cgaputc()
	5.19.2.2 consoleinit()
	5.19.2.3 consoleintr()
	5.19.2.4 consoleread()
	5.19.2.5 consolewrite()
	5.19.2.6 consputc()
	5.19.2.7 cprintf()
	5.19.2.8 panic()
	5.19.2.9 printint()
5.19.3 \	/ariable Documentation
	5.19.3.1 buf
	5.19.3.2
	5.19.3.3 crt
	5.19.3.4 e
	5.19.3.5
	5.19.3.6 lock
	5.19.3.7 locking
	5.19.3.8 panicked
	5.19.3.9 r
	5.19.3.10 w
5.20 console.	c
5.21 console.	d File Reference
5.22 console.	d
	lle Reference
5.24 date.h .	
5.25 defs.h Fi	le Reference
5.25.1 N	Macro Definition Documentation
	5.25.1.1 NELEM
5.25.2 F	Function Documentation
	5.25.2.1 acquire()
	5.25.2.2 acquiresleep()
	5.25.2.3 allocuvm()
	5.25.2.4 argint()
	5.25.2.5 argptr()
	5.25.2.6 argstr()
	5.25.2.7 begin_op()
	5.25.2.8 binit()
	5.25.2.9 bread()
	5.25.2.10 brelse()
	5.25.2.11 bwrite()
	5.25.2.12 chpr()
	5.25.2.13 clearpteu()

5.25.2.14 cmostime()
5.25.2.15 consoleinit()
5.25.2.16 consoleintr()
5.25.2.17 copyout()
5.25.2.18 copyuvm()
5.25.2.19 cprintf()
5.25.2.20 cps()
5.25.2.21 cpuid()
5.25.2.22 deallocuvm()
5.25.2.23 dirlink()
5.25.2.24 dirlookup()
5.25.2.25 end_op()
5.25.2.26 exec()
5.25.2.27 exit()
5.25.2.28 fetchint()
5.25.2.29 fetchstr()
5.25.2.30 filealloc()
5.25.2.31 fileclose()
5.25.2.32 filedup()
5.25.2.33 fileinit()
5.25.2.34 fileread()
5.25.2.35 filestat()
5.25.2.36 filewrite()
5.25.2.37 fork()
5.25.2.38 freevm()
5.25.2.39 getcallerpcs()
5.25.2.40 getprocs()
5.25.2.41 growproc()
5.25.2.42 holding()
5.25.2.43 holdingsleep()
5.25.2.44 ialloc()
5.25.2.45 ideinit()
5.25.2.46 ideintr()
5.25.2.47 iderw()
5.25.2.48 idtinit()
5.25.2.49 idup()
5.25.2.50 iinit()
5.25.2.51 ilock()
5.25.2.52 initlock()
5.25.2.53 initlog()
5.25.2.54 initsleeplock()
5.25.2.55 inituvm()

5.25.2.56 ioapicenable()
5.25.2.57 ioapicinit()
5.25.2.58 iput()
5.25.2.59 iunlock()
5.25.2.60 iunlockput()
5.25.2.61 iupdate()
5.25.2.62 kalloc()
5.25.2.63 kbdintr()
5.25.2.64 kfree()
5.25.2.65 kill()
5.25.2.66 kinit1()
5.25.2.67 kinit2()
5.25.2.68 kvmalloc()
5.25.2.69 lapiceoi()
5.25.2.70 lapicid()
5.25.2.71 lapicinit()
5.25.2.72 lapicstartap()
5.25.2.73 loaduvm()
5.25.2.74 log_write()
5.25.2.75 memcmp()
5.25.2.76 memmove()
5.25.2.77 memset()
5.25.2.78 microdelay()
5.25.2.79 mpinit()
5.25.2.80 mycpu()
5.25.2.81 myproc()
5.25.2.82 namecmp()
5.25.2.83 namei()
5.25.2.84 nameiparent()
5.25.2.85 panic()
5.25.2.86 picenable()
5.25.2.87 picinit()
5.25.2.88 pinit()
5.25.2.89 pipealloc()
5.25.2.90 pipeclose()
5.25.2.91 piperead()
5.25.2.92 pipewrite()
5.25.2.93 popcli()
5.25.2.94 procdump()
5.25.2.95 pushcli()
5.25.2.96 readi()
5.25.2.97 readsb()

5.25.2.98 release()
5.25.2.99 releasesleep()
5.25.2.100 safestrcpy()
5.25.2.101 sched()
5.25.2.102 scheduler()
5.25.2.103 seginit()
5.25.2.104 setproc()
5.25.2.105 setupkvm()
5.25.2.106 sleep()
5.25.2.107 stati()
5.25.2.108 strlen()
5.25.2.109 strncmp()
5.25.2.110 strncpy()
5.25.2.111 switchkvm()
5.25.2.112 switchuvm()
5.25.2.113 swtch()
5.25.2.114 syscall()
5.25.2.115 timerinit()
5.25.2.116 tvinit()
5.25.2.117 uartinit()
5.25.2.118 uartintr()
5.25.2.119 uartputc()
5.25.2.120 userinit()
5.25.2.121 uva2ka()
5.25.2.122 wait()
5.25.2.123 wakeup()
5.25.2.124 writei()
5.25.2.125 yield()
5.25.3 Variable Documentation
5.25.3.1 ioapicid
5.25.3.2 ismp
5.25.3.3 lapic
5.25.3.4 ticks
5.25.3.5 tickslock
5.26 defs.h
5.27 echo.c File Reference
5.27.1 Function Documentation
5.27.1.1 main()
5.28 echo.c
5.29 echo.d File Reference
5.30 echo.d
5.31 elf.h File Reference

5.31.1 Macro Definition Documentation	65
5.31.1.1 ELF_MAGIC	65
5.31.1.2 ELF_PROG_FLAG_EXEC	66
5.31.1.3 ELF_PROG_FLAG_READ	66
5.31.1.4 ELF_PROG_FLAG_WRITE	66
5.31.1.5 ELF_PROG_LOAD	66
5.32 elf.h	66
5.33 entryother.d File Reference	67
5.34 entryother.d	67
5.35 exec.c File Reference	67
5.35.1 Function Documentation	67
5.35.1.1 exec()	68
5.36 exec.c	69
5.37 exec.d File Reference	70
5.38 exec.d	70
5.39 exit.c File Reference	71
5.39.1 Function Documentation	71
5.39.1.1 main()	71
5.40 exit.c	71
5.41 exit.d File Reference	71
5.42 exit.d	71
5.43 fcntl.h File Reference	72
5.43.1 Macro Definition Documentation	72
5.43.1.1 O_CREATE	72
5.43.1.2 O_RDONLY	72
5.43.1.3 O_RDWR	72
5.43.1.4 O_WRONLY	72
5.44 fcntl.h	73
5.45 file.c File Reference	73
5.45.1 Function Documentation	73
5.45.1.1 filealloc()	74
5.45.1.2 fileclose()	74
5.45.1.3 filedup()	74
5.45.1.4 fileinit()	75
5.45.1.5 fileread()	75
5.45.1.6 filestat()	75
5.45.1.7 filewrite()	76
5.45.2 Variable Documentation	76
5.45.2.1 devsw	76
5.45.2.2 file	76
5.45.2.3	77
5.45.2.4 lock	77

5.46 file.c	177
5.47 file.d File Reference	179
5.48 file.d	179
5.49 file.h File Reference	179
5.49.1 Macro Definition Documentation	179
5.49.1.1 CONSOLE	179
5.49.2 Variable Documentation	180
5.49.2.1 devsw	180
5.50 file.h	180
5.51 foo.c File Reference	180
5.51.1 Function Documentation	181
5.51.1.1 main()	181
5.52 foo.c	181
5.53 foo.d File Reference	182
5.54 foo.d	182
5.55 forktest.c File Reference	182
5.55.1 Macro Definition Documentation	182
5.55.1.1 N	182
5.55.2 Function Documentation	183
5.55.2.1 forktest()	183
5.55.2.2 main()	183
5.55.2.3 printf()	184
5.56 forktest.c	184
5.57 forktest.d File Reference	185
5.58 forktest.d	185
5.59 fs.c File Reference	185
5.59.1 Macro Definition Documentation	186
5.59.1.1 min	186
5.59.2 Function Documentation	186
5.59.2.1 balloc()	186
5.59.2.2 bfree()	187
5.59.2.3 bmap()	187
5.59.2.4 bzero()	188
5.59.2.5 dirlink()	188
5.59.2.6 dirlookup()	189
5.59.2.7 ialloc()	189
5.59.2.8 idup()	190
5.59.2.9 iget()	190
5.59.2.10 iinit()	191
5.59.2.11 ilock()	191
5.59.2.12 iput()	192
5.59.2.13 itrunc()	192

5.59.2.14 iunlock()	193
5.59.2.15 iunlockput()	193
5.59.2.16 iupdate()	193
5.59.2.17 namecmp()	194
5.59.2.18 namei()	194
5.59.2.19 nameiparent()	194
5.59.2.20 namex()	195
5.59.2.21 readi()	195
5.59.2.22 readsb()	196
5.59.2.23 skipelem()	196
5.59.2.24 stati()	197
5.59.2.25 writei()	197
5.59.3 Variable Documentation	197
5.59.3.1	198
5.59.3.2 inode	198
5.59.3.3 lock	198
5.59.3.4 sb	198
5.60 fs.c	198
5.61 fs.d File Reference	206
5.62 fs.d	206
5.63 fs.h File Reference	206
5.63.1 Macro Definition Documentation	207
5.63.1.1 BBLOCK	207
5.63.1.2 BPB	207
5.63.1.3 BSIZE	207
5.63.1.4 DIRSIZ	207
5.63.1.5 IBLOCK	207
5.63.1.6 IPB	208
5.63.1.7 MAXFILE	208
5.63.1.8 NDIRECT	208
5.63.1.9 NINDIRECT	208
5.63.1.10 ROOTINO	208
5.64 fs.h	209
5.65 grep.c File Reference	209
5.65.1 Function Documentation	210
5.65.1.1 grep()	210
5.65.1.2 main()	210
5.65.1.3 match()	211
5.65.1.4 matchhere()	211
5.65.1.5 matchstar()	212
5.65.2 Variable Documentation	212
5.65.2.1 buf	212

5.66 grep.c
5.67 grep.d File Reference
5.68 grep.d
5.69 ide.c File Reference
5.69.1 Macro Definition Documentation
5.69.1.1 IDE_BSY
5.69.1.2 IDE_CMD_RDMUL
5.69.1.3 IDE_CMD_READ
5.69.1.4 IDE_CMD_WRITE
5.69.1.5 IDE_CMD_WRMUL
5.69.1.6 IDE_DF
5.69.1.7 IDE_DRDY
5.69.1.8 IDE_ERR
5.69.1.9 SECTOR_SIZE
5.69.2 Function Documentation
5.69.2.1 ideinit()
5.69.2.2 ideintr()
5.69.2.3 iderw()
5.69.2.4 idestart()
5.69.2.5 idewait()
5.69.3 Variable Documentation
5.69.3.1 havedisk1
5.69.3.2 idelock
5.69.3.3 idequeue
5.70 ide.c
5.71 ide.d File Reference
5.72 ide.d
5.73 init.c File Reference
5.73.1 Function Documentation
5.73.1.1 main()
5.73.2 Variable Documentation
5.73.2.1 argv
5.74 init.c
5.75 init.d File Reference
5.76 init.d
5.77 initcode.d File Reference
5.78 initcode.d
5.79 ioapic.c File Reference
5.79.1 Macro Definition Documentation
5.79.1.1 INT_ACTIVELOW
5.79.1.2 INT_DISABLED
5.79.1.3 INT_LEVEL

5.79.1.4 INT_LOGICAL
5.79.1.5 IOAPIC
5.79.1.6 REG_ID
5.79.1.7 REG_TABLE
5.79.1.8 REG_VER
5.79.2 Function Documentation
5.79.2.1 ioapicenable()
5.79.2.2 ioapicinit()
5.79.2.3 ioapicread()
5.79.2.4 ioapicwrite()
5.79.3 Variable Documentation
5.79.3.1 ioapic
5.80 ioapic.c
5.81 ioapic.d File Reference
5.82 ioapic.d
5.83 kalloc.c File Reference
5.83.1 Function Documentation
5.83.1.1 freerange()
5.83.1.2 kalloc()
5.83.1.3 kfree()
5.83.1.4 kinit1()
5.83.1.5 kinit2()
5.83.2 Variable Documentation
5.83.2.1 end
5.83.2.2 freelist
5.83.2.3
5.83.2.4 lock
5.83.2.5 use_lock
5.84 kalloc.c
5.85 kalloc.d File Reference
5.86 kalloc.d
5.87 kbd.c File Reference
5.87.1 Function Documentation
5.87.1.1 kbdgetc()
5.87.1.2 kbdintr()
5.88 kbd.c
5.89 kbd.d File Reference
5.90 kbd.d
5.91 kbd.h File Reference
5.91.1 Macro Definition Documentation
5.91.1.1 ALT
5.91.1.2 C

5.91.1.3 CAPSLOCK	36
5.91.1.4 CTL	36
5.91.1.5 E0ESC	36
5.91.1.6 KBDATAP	36
5.91.1.7 KBS_DIB	36
5.91.1.8 KBSTATP	36
5.91.1.9 KEY_DEL	37
5.91.1.10 KEY_DN	37
5.91.1.11 KEY_END	37
5.91.1.12 KEY_HOME	37
5.91.1.13 KEY_INS	37
5.91.1.14 KEY_LF	37
5.91.1.15 KEY_PGDN	38
5.91.1.16 KEY_PGUP	38
5.91.1.17 KEY_RT	38
5.91.1.18 KEY_UP	38
5.91.1.19 NO	38
5.91.1.20 NUMLOCK	38
5.91.1.21 SCROLLLOCK	39
5.91.1.22 SHIFT	39
5.91.2 Variable Documentation	39
5.91.2.1 ctlmap	39
5.91.2.2 normalmap	Ю
5.91.2.3 shiftcode	Ю
5.91.2.4 shiftmap	10
5.91.2.5 togglecode	ŀ1
5.92 kbd.h	11
5.93 kill.c File Reference	13
5.93.1 Function Documentation	13
5.93.1.1 main()	13
5.94 kill.c	13
5.95 kill.d File Reference	14
5.96 kill.d	14
5.97 lapic.c File Reference	14
5.97.1 Macro Definition Documentation	l 5
5.97.1.1 ASSERT	ŀ5
5.97.1.2 BCAST	ŀ5
5.97.1.3 BUSY	1 6
5.97.1.4 CMOS_PORT	1 6
5.97.1.5 CMOS_RETURN	1 6
5.97.1.6 CMOS_STATA	1 6
5.97.1.7 CMOS_STATB	l 6

	5.97.1.8 CMOS_UIP	246
	5.97.1.9 CONV	247
	5.97.1.10 DAY	247
	5.97.1.11 DEASSERT	247
	5.97.1.12 DELIVS	247
	5.97.1.13 ENABLE	247
	5.97.1.14 EOI	247
	5.97.1.15 ERROR	248
	5.97.1.16 ESR	248
	5.97.1.17 FIXED	248
	5.97.1.18 HOURS	248
	5.97.1.19 ICRHI	248
	5.97.1.20 ICRLO	248
	5.97.1.21 ID	249
	5.97.1.22 INIT	249
	5.97.1.23 LEVEL	249
	5.97.1.24 LINT0	249
	5.97.1.25 LINT1	249
	5.97.1.26 MASKED	249
	5.97.1.27 MINS	250
	5.97.1.28 MONTH	250
	5.97.1.29 PCINT	250
	5.97.1.30 PERIODIC	250
	5.97.1.31 SECS	250
	5.97.1.32 STARTUP	250
	5.97.1.33 SVR	251
	5.97.1.34 TCCR	251
	5.97.1.35 TDCR	251
	5.97.1.36 TICR	251
	5.97.1.37 TIMER	251
	5.97.1.38 TPR	251
	5.97.1.39 VER	252
	5.97.1.40 X1	252
	5.97.1.41 YEAR	252
5.97.2 F	function Documentation	252
	5.97.2.1 cmos_read()	252
	5.97.2.2 cmostime()	253
	5.97.2.3 fill_rtcdate()	253
	5.97.2.4 lapiceoi()	253
	5.97.2.5 lapicid()	254
	5.97.2.6 lapicinit()	254
	5.97.2.7 lapicstartap()	255

5.97.2.8 lapicw()	255
5.97.2.9 microdelay()	255
5.97.3 Variable Documentation	256
5.97.3.1 lapic	256
5.98 lapic.c	256
5.99 lapic.d File Reference	259
5.100 lapic.d	259
5.101 In.c File Reference	259
5.101.1 Function Documentation	259
5.101.1.1 main()	259
5.102 ln.c	259
5.103 In.d File Reference	260
5.104 ln.d	260
5.105 log.c File Reference	260
5.105.1 Function Documentation	260
5.105.1.1 begin_op()	261
5.105.1.2 commit()	261
5.105.1.3 end_op()	261
5.105.1.4 initlog()	262
5.105.1.5 install_trans()	262
5.105.1.6 log_write()	263
5.105.1.7 read_head()	263
5.105.1.8 recover_from_log()	263
5.105.1.9 write_head()	264
5.105.1.10 write_log()	264
5.105.2 Variable Documentation	264
5.105.2.1 log	264
5.106 log.c	265
5.107 log.d File Reference	267
5.108 log.d	267
5.109 ls.c File Reference	268
5.109.1 Function Documentation	268
5.109.1.1 fmtname()	268
5.109.1.2 ls()	268
5.109.1.3 main()	269
5.110 ls.c	269
5.111 ls.d File Reference	
5.112 ls.d	
5.113 main.c File Reference	
5.113.1 Function Documentation	
5.113.1.1attribute()	271
5.113.1.2 mpenter()	272

5.113.1.3 mpmain()
5.113.1.4 startothers()
5.113.2 Variable Documentation
5.113.2.1 entrypgdir
5.114 main.c
5.115 main.d File Reference
5.116 main.d
5.117 memide.c File Reference
5.117.1 Function Documentation
5.117.1.1 ideinit()
5.117.1.2 ideintr()
5.117.1.3 iderw()
5.117.2 Variable Documentation
5.117.2.1 _binary_fs_img_size
5.117.2.2 _binary_fs_img_start
5.117.2.3 disksize
5.117.2.4 memdisk
5.118 memide.c
5.119 memlayout.h File Reference
5.119.1 Macro Definition Documentation
5.119.1.1 DEVSPACE
5.119.1.2 EXTMEM
5.119.1.3 KERNBASE
5.119.1.4 KERNLINK
5.119.1.5 P2V
5.119.1.6 P2V_WO
5.119.1.7 PHYSTOP
5.119.1.8 V2P
5.119.1.9 V2P_WO
5.120 memlayout.h
5.121 mkdir.c File Reference
5.121.1 Function Documentation
5.121.1.1 main()
5.122 mkdir.c
5.123 mkdir.d File Reference
5.124 mkdir.d
5.125 mkfs.c File Reference
5.125.1 Macro Definition Documentation
5.125.1.1 min
5.125.1.2 NINODES
5.125.1.3 stat
5.125.1.4 static_assert

5.125.2 Function Documentation	283
5.125.2.1 balloc()	283
5.125.2.2 ialloc()	283
5.125.2.3 iappend()	284
5.125.2.4 main()	284
5.125.2.5 rinode()	286
5.125.2.6 rsect()	286
5.125.2.7 winode()	286
5.125.2.8 wsect()	287
5.125.2.9 xint()	287
5.125.2.10 xshort()	287
5.125.3 Variable Documentation	287
5.125.3.1 freeblock	288
5.125.3.2 freeinode	288
5.125.3.3 fsfd	288
5.125.3.4 nbitmap	288
5.125.3.5 nblocks	288
5.125.3.6 ninodeblocks	289
5.125.3.7 nlog	289
5.125.3.8 nmeta	289
5.125.3.9 sb	289
5.125.3.10 zeroes	289
5.126 mkfs.c	290
5.127 mmu.h File Reference	293
5.127.1 Macro Definition Documentation	294
5.127.1.1 CR0_PE	294
5.127.1.2 CR0_PG	294
5.127.1.3 CR0_WP	295
5.127.1.4 CR4_PSE	295
5.127.1.5 DPL_USER	295
5.127.1.6 FL_IF	295
5.127.1.7 NPDENTRIES	295
5.127.1.8 NPTENTRIES	295
5.127.1.9 NSEGS	296
5.127.1.10 PDX	296
5.127.1.11 PDXSHIFT	296
5.127.1.12 PGADDR	296
5.127.1.13 PGROUNDDOWN	296
5.127.1.14 PGROUNDUP	296
5.127.1.15 PGSIZE	297
5.127.1.16 PTE_ADDR	97
5.127.1.17 PTE_FLAGS	297

5.127.1.18 PTE_P	97
5.127.1.19 PTE_PS	97
5.127.1.20 PTE_U	97
5.127.1.21 PTE_W	98
5.127.1.22 PTX	98
5.127.1.23 PTXSHIFT	98
5.127.1.24 SEG	98
5.127.1.25 SEG16	98
5.127.1.26 SEG_KCODE	99
5.127.1.27 SEG_KDATA	99
5.127.1.28 SEG_TSS	99
5.127.1.29 SEG_UCODE	99
5.127.1.30 SEG_UDATA	99
5.127.1.31 SETGATE	00
5.127.1.32 STA_R	00
5.127.1.33 STA_W	00
5.127.1.34 STA_X	00
5.127.1.35 STS_IG32	00
5.127.1.36 STS_T32A	01
5.127.1.37 STS_TG32	01
5.127.2 Typedef Documentation	01
5.127.2.1 pte_t	01
5.128 mmu.h	01
5.129 mp.c File Reference	03
5.129.1 Function Documentation	04
5.129.1.1 mpconfig()	04
5.129.1.2 mpinit()	04
5.129.1.3 mpsearch()	05
5.129.1.4 mpsearch1()	05
5.129.1.5 sum()	06
5.129.2 Variable Documentation	06
5.129.2.1 cpus	06
5.129.2.2 ioapicid	06
5.129.2.3 ncpu	07
5.130 mp.c	07
5.131 mp.d File Reference	80
5.132 mp.d	80
5.133 mp.h File Reference	09
5.133.1 Macro Definition Documentation	09
5.133.1.1 MPBOOT	09
5.133.1.2 MPBUS	09
5.133.1.3 MPIOAPIC	09

5.133.1.4 MPIOINTR	310
5.133.1.5 MPLINTR	310
5.133.1.6 MPPROC	310
5.134 mp.h	310
5.135 nice.c File Reference	311
5.135.1 Function Documentation	311
5.135.1.1 main()	311
5.136 nice.c	312
5.137 nice.d File Reference	312
5.138 nice.d	112
5.139 param.h File Reference	312
5.139.1 Macro Definition Documentation	312
5.139.1.1 FSSIZE	13
5.139.1.2 KSTACKSIZE	313
5.139.1.3 LOGSIZE	13
5.139.1.4 MAXARG	313
5.139.1.5 MAXOPBLOCKS	13
5.139.1.6 NBUF	313
5.139.1.7 NCPU	314
5.139.1.8 NDEV	314
5.139.1.9 NFILE	314
5.139.1.10 NINODE	314
5.139.1.11 NOFILE	314
5.139.1.12 NPROC	314
5.139.1.13 ROOTDEV	115
5.140 param.h	15
5.141 picirq.c File Reference	315
5.141.1 Macro Definition Documentation	315
5.141.1.1 IO_PIC1	315
5.141.1.2 IO_PIC2	316
5.141.2 Function Documentation	316
5.141.2.1 picinit()	316
5.142 picirq.c	316
5.143 picirq.d File Reference	316
5.144 picirq.d	16
5.145 pipe.c File Reference	317
5.145.1 Macro Definition Documentation	317
5.145.1.1 PIPESIZE	317
5.145.2 Function Documentation	317
5.145.2.1 pipealloc()	118
5.145.2.2 pipeclose()	318
5.145.2.3 piperead()	319

5.145.2.4 pipewrite()
5.146 pipe.c
5.147 pipe.d File Reference
5.148 pipe.d
5.149 printf.c File Reference
5.149.1 Function Documentation
5.149.1.1 printf()
5.149.1.2 printint()
5.149.1.3 putc()
5.150 printf.c
5.151 printf.d File Reference
5.152 printf.d
5.153 proc.c File Reference
5.153.1 Function Documentation
5.153.1.1 allocproc()
5.153.1.2 chpr()
5.153.1.3 cps()
5.153.1.4 cpuid()
5.153.1.5 exit()
5.153.1.6 fork()
5.153.1.7 forkret()
5.153.1.8 growproc()
5.153.1.9 kill()
5.153.1.10 mycpu()
5.153.1.11 myproc()
5.153.1.12 pinit()
5.153.1.13 procdump()
5.153.1.14 sched()
5.153.1.15 scheduler()
5.153.1.16 sleep()
5.153.1.17 trapret()
5.153.1.18 userinit()
5.153.1.19 wait()
5.153.1.20 wakeup()
5.153.1.21 wakeup1()
5.153.1.22 yield()
5.153.2 Variable Documentation
5.153.2.1 initproc
5.153.2.2 lock
5.153.2.3 nextpid
5.153.2.4 proc
5.153.2.5

5.154 proc.c
5.155 proc.d File Reference
5.156 proc.d
5.157 proc.h File Reference
5.157.1 Enumeration Type Documentation
5.157.1.1 procstate
5.157.2 Variable Documentation
5.157.2.1 cpus
5.157.2.2 ncpu
5.158 proc.h
5.159 ps.c File Reference
5.159.1 Function Documentation
5.159.1.1 main()
5.160 ps.c
5.161 ps.d File Reference
5.162 ps.d
5.163 pstree.c File Reference
5.163.1 Macro Definition Documentation
5.163.1.1 MAXPROC
5.163.2 Function Documentation
5.163.2.1 main()
5.164 pstree.c
5.165 pstree.d File Reference
5.166 pstree.d
5.167 README.md File Reference
5.168 rm.c File Reference
5.168.1 Function Documentation
5.168.1.1 main()
5.169 rm.c
5.170 rm.d File Reference
5.171 rm.d
5.172 sh.c File Reference
5.172.1 Macro Definition Documentation
5.172.1.1 BACK
5.172.1.2 EXEC
5.172.1.3 LIST
5.172.1.4 MAXARGS
5.172.1.5 PIPE
5.172.1.6 REDIR
5.172.2 Function Documentation
5.172.2.1 backcmd()
5.172.2.2 execcmd()

5.172.2.3 fork1()	352
5.172.2.4 getcmd()	353
5.172.2.5 gettoken()	353
5.172.2.6 listcmd()	354
5.172.2.7 main()	354
5.172.2.8 nulterminate()	354
5.172.2.9 panic()	355
5.172.2.10 parseblock()	355
5.172.2.11 parsecmd()	356
5.172.2.12 parseexec()	356
5.172.2.13 parseline()	357
5.172.2.14 parsepipe()	357
5.172.2.15 parseredirs()	357
5.172.2.16 peek()	358
5.172.2.17 pipecmd()	358
5.172.2.18 redircmd()	359
5.172.2.19 runcmd()	359
5.172.3 Variable Documentation	360
5.172.3.1 symbols	360
5.172.3.2 whitespace	360
5.173 sh.c	361
5.174 sh.d File Reference	366
5.175 sh.d	366
5.176 shutdown.c File Reference	366
5.176.1 Function Documentation	367
5.176.1.1 main()	367
5.177 shutdown.c	367
5.178 shutdown.d File Reference	367
5.179 shutdown.d	367
5.180 sleeplock.c File Reference	368
5.180.1 Function Documentation	368
5.180.1.1 acquiresleep()	368
5.180.1.2 holdingsleep()	368
5.180.1.3 initsleeplock()	369
5.180.1.4 releasesleep()	369
5.181 sleeplock.c	369
5.182 sleeplock.d File Reference	370
5.183 sleeplock.d	370
5.184 sleeplock.h File Reference	370
5.185 sleeplock.h	370
5.186 spinlock.c File Reference	370
5.186.1 Function Documentation	371

5.186.1.1 acquire()
5.186.1.2 getcallerpcs()
5.186.1.3 holding()
5.186.1.4 initlock()
5.186.1.5 popcli()
5.186.1.6 pushcli()
5.186.1.7 release()
5.187 spinlock.c
5.188 spinlock.d File Reference
5.189 spinlock.d
5.190 spinlock.h File Reference
5.191 spinlock.h
5.192 stat.h File Reference
5.192.1 Macro Definition Documentation
5.192.1.1 T_DEV
5.192.1.2 T_DIR
5.192.1.3 T_FILE
5.193 stat.h
5.194 stressfs.c File Reference
5.194.1 Function Documentation
5.194.1.1 main()
5.195 stressfs.c
5.196 stressfs.d File Reference
5.197 stressfs.d
5.198 string.c File Reference
5.198.1 Function Documentation
5.198.1.1 memcmp()
5.198.1.2 memcpy()
5.198.1.3 memmove()
5.198.1.4 memset()
5.198.1.5 safestrcpy()
5.198.1.6 strlen()
5.198.1.7 strncmp()
5.198.1.8 strncpy()
5.199 string.c
5.200 string.d File Reference
5.201 string.d
5.202 syscall.c File Reference
5.202.1 Function Documentation
5.202.1.1 argint()
5.202.1.2 argptr()
5,202.1.3 argstr()

5.202.1	.4 fetchint()		 	 	 	 385
5.202.1	.5 fetchstr()		 	 	 	 386
5.202.1	.6 sys_chdir()		 	 	 	 386
5.202.1	.7 sys_chpr()		 	 	 	 386
5.202.1	.8 sys_close()		 	 	 	 387
5.202.1	.9 sys_cps()		 	 	 	 387
5.202.1	.10 sys_dup()		 	 	 	 387
5.202.1	.11 sys_exec()		 	 	 	 388
5.202.1	.12 sys_exit()		 	 	 	 388
5.202.1	.13 sys_fork()		 	 	 	 388
5.202.1	.14 sys_fstat()		 	 	 	 388
5.202.1	.15 sys_getpid()		 	 	 	 389
5.202.1	.16 sys_getprocs()		 	 	 	 389
5.202.1	.17 sys_halt()		 	 	 	 389
5.202.1	.18 sys_kill()		 	 	 	 390
5.202.1	.19 sys_link()		 	 	 	 390
5.202.1	.20 sys_mkdir()		 	 	 	 391
5.202.1	.21 sys_mknod() .		 	 	 	 391
5.202.1	.22 sys_open()		 	 	 	 391
5.202.1	.23 sys_pipe()		 	 	 	 392
5.202.1	.24 sys_read()		 	 	 	 392
5.202.1	.25 sys_sbrk()		 	 	 	 393
5.202.1	.26 sys_sleep()		 	 	 	 393
5.202.1	.27 sys_unlink()		 	 	 	 394
5.202.1	.28 sys_uptime() .		 	 	 	 394
5.202.1	.29 sys_wait()		 	 	 	 395
5.202.1	.30 sys_write()		 	 	 	 395
5.202.1	.31 syscall()		 	 	 	 395
5.202.2 Variabl	e Documentation .		 	 	 	 395
5.202.2	2.1 syscalls		 	 	 	 396
5.203 syscall.c			 	 	 	 396
5.204 syscall.d File F	Reference		 	 	 	 398
5.205 syscall.d			 	 	 	 398
5.206 syscall.h File F	Reference		 	 	 	 398
5.206.1 Macro	Definition Document	tation	 	 	 	 399
5.206.1	.1 SYS_chdir		 	 	 	 399
5.206.1	.2 SYS_chpr		 	 	 	 399
5.206.1	.3 SYS_close		 	 	 	 399
5.206.1	.4 SYS_cps		 	 	 	 399
5.206.1	.5 SYS_dup		 	 	 	 400
5.206.1	.6 SYS_exec		 	 	 	 400
5.206.1	.7 SYS_exit		 	 	 	 400

5.206.1.8 SYS_fork	0
5.206.1.9 SYS_fstat	0
5.206.1.10 SYS_getpid	0
5.206.1.11 SYS_getprocs	1
5.206.1.12 SYS_halt	1
5.206.1.13 SYS_kill	1
5.206.1.14 SYS_link	1
5.206.1.15 SYS_mkdir	1
5.206.1.16 SYS_mknod	1
5.206.1.17 SYS_open	2
5.206.1.18 SYS_pipe	2
5.206.1.19 SYS_read	2
5.206.1.20 SYS_sbrk	2
5.206.1.21 SYS_sleep	2
5.206.1.22 SYS_unlink	2
5.206.1.23 SYS_uptime	3
5.206.1.24 SYS_wait	3
5.206.1.25 SYS_write	3
5.207 syscall.h	3
5.208 sysfile.c File Reference	4
5.208.1 Function Documentation	4
5.208.1.1 argfd()	4
5.208.1.2 create()	5
5.208.1.3 fdalloc()	6
5.208.1.4 isdirempty()	6
5.208.1.5 sys_chdir()	6
5.208.1.6 sys_close()	7
5.208.1.7 sys_dup()	7
5.208.1.8 sys_exec()	7
5.208.1.9 sys_fstat()	8
5.208.1.10 sys_link()	8
5.208.1.11 sys_mkdir()	9
5.208.1.12 sys_mknod()	9
5.208.1.13 sys_open()	9
5.208.1.14 sys_pipe()	0
5.208.1.15 sys_read()	0
5.208.1.16 sys_unlink()	1
5.208.1.17 sys_write()	2
5.209 sysfile.c	2
5.210 sysfile.d File Reference	7
5.211 sysfile.d	7
5.212 sysproc.c File Reference	7

5.212.1 Function Documentation	18
5.212.1.1 sys_chpr()	18
5.212.1.2 sys_cps()	18
5.212.1.3 sys_exit()	19
5.212.1.4 sys_fork()	19
5.212.1.5 sys_getpid()	19
5.212.1.6 sys_getprocs()	19
5.212.1.7 sys_halt()	20
5.212.1.8 sys_kill()	20
5.212.1.9 sys_sbrk()	20
5.212.1.10 sys_sleep()	21
5.212.1.11 sys_uptime()	21
5.212.1.12 sys_wait()	21
5.212.2 Variable Documentation	21
5.212.2.1 lock	22
5.212.2.2 proc	22
5.212.2.3	22
5.213 sysproc.c	22
5.214 sysproc.d File Reference	24
5.215 sysproc.d	24
5.216 trap.c File Reference	24
5.216.1 Function Documentation	25
5.216.1.1 idtinit()	25
5.216.1.2 trap()	25
5.216.1.3 tvinit()	26
5.216.2 Variable Documentation	26
5.216.2.1 idt	27
5.216.2.2 ticks	27
5.216.2.3 tickslock	27
5.216.2.4 vectors	27
5.217 trap.c	28
5.218 trap.d File Reference	29
5.219 trap.d	29
5.220 traps.h File Reference	29
5.220.1 Macro Definition Documentation	30
5.220.1.1 IRQ_COM1	30
5.220.1.2 IRQ_ERROR	30
5.220.1.3 IRQ_IDE	30
5.220.1.4 IRQ_KBD	30
5.220.1.5 IRQ_SPURIOUS	31
5.220.1.6 IRQ_TIMER	31
5.220.1.7 T_ALIGN	31

	5.220.1.8 T_BOUND	31
	5.220.1.9 T_BRKPT	31
	5.220.1.10 T_DBLFLT	31
	5.220.1.11 T_DEBUG	32
	5.220.1.12 T_DEFAULT	32
	5.220.1.13 T_DEVICE	32
	5.220.1.14 T_DIVIDE	32
	5.220.1.15 T_FPERR	32
	5.220.1.16 T_GPFLT	32
	5.220.1.17 T_ILLOP	33
	5.220.1.18 T_IRQ0	33
	5.220.1.19 T_MCHK	33
	5.220.1.20 T_NMI	33
	5.220.1.21 T_OFLOW	33
	5.220.1.22 T_PGFLT	33
	5.220.1.23 T_SEGNP	34
	5.220.1.24 T_SIMDERR	34
	5.220.1.25 T_STACK	34
	5.220.1.26 T_SYSCALL	34
	5.220.1.27 T_TSS	34
5.221 traps.h		35
5.222 types.h	File Reference	35
5.222.1	Typedef Documentation	35
	5.222.1.1 pde_t	35
	5.222.1.2 uchar	36
	5.222.1.3 uint	36
	5.222.1.4 ushort	36
5.223 types.h	43	36
5.224 uart.c F	File Reference	36
5.224.1	Macro Definition Documentation	37
	5.224.1.1 COM1	37
5.224.2	Function Documentation	37
	5.224.2.1 uartgetc()	37
	5.224.2.2 uartinit()	38
	5.224.2.3 uartintr()	38
	5.224.2.4 uartputc()	38
5.224.3	Variable Documentation	39
	5.224.3.1 uart	39
5.225 uart.c		39
5.226 uart.d F	File Reference	40
5.227 uart.d		40
5 228 ulib c F	ile Reference	4٥

5.228.1 Function Documentation	0
5.228.1.1 atoi()	.1
5.228.1.2 gets()	-1
5.228.1.3 memmove()	-1
5.228.1.4 memset()	2
5.228.1.5 stat()	2
5.228.1.6 strchr()	2
5.228.1.7 strcmp()	3
5.228.1.8 strcpy()	3
5.228.1.9 strlen()	3
5.229 ulib.c	4
5.230 ulib.d File Reference	5
5.231 ulib.d	5
5.232 umalloc.c File Reference	5
5.232.1 Typedef Documentation	6
5.232.1.1 Align	6
5.232.1.2 Header	6
5.232.2 Function Documentation	6
5.232.2.1 free()	6
5.232.2.2 malloc()	.7
5.232.2.3 morecore()	.7
5.232.3 Variable Documentation	.7
5.232.3.1 base	8
5.232.3.2 freep	8
5.233 umalloc.c	8
5.234 umalloc.d File Reference	.9
5.235 umalloc.d	9
5.236 uproc.h File Reference	9
5.237 uproc.h	9
5.238 user.h File Reference	0
5.238.1 Function Documentation	0
5.238.1.1 atoi()	1
5.238.1.2 chdir()	1
5.238.1.3 chpr()	1
5.238.1.4 close()	1
5.238.1.5 cps()	2
5.238.1.6 dup()	2
5.238.1.7 exec()	2
5.238.1.8 exit()	3
5.238.1.9 fork()	4
5.238.1.10 free()	5
5.238.1.11 fstat()	5

5.238.1.12 getpid()	455
5.238.1.13 getprocs()	456
5.238.1.14 gets()	456
5.238.1.15 halt()	456
5.238.1.16 kill()	456
5.238.1.17 link()	457
5.238.1.18 malloc()	457
5.238.1.19 memmove()	457
5.238.1.20 memset()	458
5.238.1.21 mkdir()	458
5.238.1.22 mknod()	458
5.238.1.23 open()	458
5.238.1.24 pipe()	459
5.238.1.25 printf()	459
5.238.1.26 read()	459
5.238.1.27 sbrk()	459
5.238.1.28 sleep()	459
5.238.1.29 stat()	460
5.238.1.30 strchr()	460
5.238.1.31 strcmp()	460
5.238.1.32 strcpy()	461
5.238.1.33 strlen()	461
5.238.1.34 unlink()	461
5.238.1.35 uptime()	461
5.238.1.36 wait()	462
5.238.1.37 write()	462
5.239 user.h	462
5.240 usertests.c File Reference	463
5.240.1 Macro Definition Documentation	464
5.240.1.1 BIG	464
5.240.1.2 RTC_ADDR	465
5.240.1.3 RTC_DATA	465
5.240.2 Function Documentation	465
5.240.2.1 argptest()	465
5.240.2.2 bigargtest()	465
5.240.2.3 bigdir()	466
5.240.2.4 bigfile()	467
5.240.2.5 bigwrite()	467
5.240.2.6 bsstest()	468
5.240.2.7 concreate()	468
5.240.2.8 createdelete()	469
5.240.2.9 createtest()	470

5.240.2.10 dirfile()	471
5.240.2.11 dirtest()	472
5.240.2.12 exectest()	472
5.240.2.13 exitiputtest()	473
5.240.2.14 exitwait()	473
5.240.2.15 forktest()	474
5.240.2.16 fourfiles()	474
5.240.2.17 fourteen()	475
5.240.2.18 fsfull()	476
5.240.2.19 iputtest()	476
5.240.2.20 iref()	477
5.240.2.21 linktest()	478
5.240.2.22 linkunlink()	478
5.240.2.23 main()	479
5.240.2.24 mem()	480
5.240.2.25 openiputtest()	480
5.240.2.26 opentest()	481
5.240.2.27 pipe1()	481
5.240.2.28 preempt()	482
5.240.2.29 rand()	483
5.240.2.30 rmdot()	483
5.240.2.31 sbrktest()	484
5.240.2.32 sharedfd()	485
5.240.2.33 subdir()	486
5.240.2.34 uio()	488
5.240.2.35 unlinkread()	489
5.240.2.36 validateint()	490
5.240.2.37 validatetest()	490
5.240.2.38 writetest()	
5.240.2.39 writetest1()	491
5.240.3 Variable Documentation	492
5.240.3.1 buf	492
5.240.3.2 echoargv	
5.240.3.3 name	
5.240.3.4 randstate	493
5.240.3.5 stdout	493
5.240.3.6 uninit	493
5.241 usertests.c	494
5.242 usertests.d File Reference	
5.243 usertests.d	
5.244 vm.c File Reference	
5.244.1 Function Documentation	515

5.244.1.1 allocuvm()	516
5.244.1.2 clearpteu()	516
5.244.1.3 copyout()	517
5.244.1.4 copyuvm()	517
5.244.1.5 deallocuvm()	518
5.244.1.6 freevm()	518
5.244.1.7 inituvm()	519
5.244.1.8 kvmalloc()	519
5.244.1.9 loaduvm()	519
5.244.1.10 mappages()	520
5.244.1.11 seginit()	520
5.244.1.12 setupkvm()	521
5.244.1.13 switchkvm()	521
5.244.1.14 switchuvm()	521
5.244.1.15 uva2ka()	522
5.244.1.16 walkpgdir()	522
5.244.2 Variable Documentation	522
5.244.2.1 data	522
5.244.2.2 kmap	523
5.244.2.3 kpgdir	523
5.245 vm.c	523
5.246 vm.d File Reference	528
5.247 vm.d	528
5.248 wc.c File Reference	528
5.248.1 Function Documentation	528
5.248.1.1 main()	528
5.248.1.2 wc()	529
5.248.2 Variable Documentation	529
5.248.2.1 buf	529
5.249 wc.c	529
5.250 wc.d File Reference	530
5.251 wc.d	530
5.252 x86.h File Reference	530
5.252.1 Function Documentation	531
5.252.1.1 cli()	531
5.252.1.2 inb()	531
5.252.1.3 insl()	531
5.252.1.4 lcr3()	532
5.252.1.5 lgdt()	532
5.252.1.6 lidt()	532
5.252.1.7 loadgs()	533
5.252.1.8 ltr()	533

Index											541
5.258 zombie.d			 	 	 		 	 			 539
5.257 zombie.d File Re	eference		 	 	 		 	 			 539
5.256 zombie.c			 	 	 		 	 			 538
5.255.1.1	main()		 	 	 		 	 			 538
5.255.1 Function	Documentati	on .	 	 	 		 	 			 538
5.255 zombie.c File Re	ference		 	 	 		 	 			 538
5.254 xv6.dox File Refe	erence		 	 	 		 	 			 538
5.253 x86.h			 	 	 		 	 			 536
5.252.1.1	7 xchg()		 	 	 		 	 			 535
5.252.1.1	6 stosl()		 	 	 		 	 			 535
5.252.1.1	5 stosb()		 	 	 		 	 			 535
5.252.1.1	4 sti()		 	 	 		 	 			 534
5.252.1.1	3 readeflags	()	 	 	 		 	 			 534
5.252.1.1	2 rcr2()		 	 	 		 	 			 534
5.252.1.1	1 outw()		 	 	 		 	 			 534
5.252.1.1	0 outsl()		 	 	 		 	 	 		 533
5.252.1.9	outb()		 	 	 		 	 			 533

Chapter 1

ENVIRONMENT PREPARETION

1.1 LINUX

- 1. sudo apt-get install git wget qemu aqemu
- 2. sudo apt-get install libc6-dev:i386 gcc
- 3. sudo apt-get install gdb
- 4. git clone https://github.com/subhrendu1987/xv6-public/
- 5. cd xv6-public && make qemu
- 6. If you have 64 bit OS there is a chance Makefile will not be able to find qemu. Check using the command make qemu-nox and see the output. In that case you should edit the Makefile at line 54 and add the following code: QEMU = $qemu-system-x86_64$
- 1. make qemu-gdb
- 1. Open new terminal cd xv6-public && gdb ./kernel

1.2 BUILDING AND RUNNING XV6

- To build xv6 on an x86 ELF machine (like Linux or FreeBSD), run "make".
- On non-x86 or non-ELF machines (like OS X, even on x86), you may need to install a cross-compiler gcc suite capable of producing x86 ELF binaries (see https://pdos.csail.mit.edu/6.828/).
- Then run "make TOOLPREFIX=i386-jos-elf-". Now install the QEMU PC simulator and run "make qemu".

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

backcmd	. 9
buf	. 10
cmd	. 12
context	. 13
cpu	. 14
devsw	. 16
dinode	. 17
dirent	. 19
elfhdr	. 20
execcmd	. 23
file	. 24
gatedesc	. 26
header	. 28
inode	. 30
ioapic	. 32
kmap	. 33
listcmd	. 35
log	. 36
logheader	. 37
mp	. 38
mpconf	. 40
mpioapic	. 43
mpproc	. 44
pipe	. 46
pipecmd	. 48
proc	. 49
proghdr	. 52
redircmd	. 54
rtcdate	. 56
run	. 58
segdesc	. 58
sleeplock	. 61
spinlock	. 62
stat	. 63
superblock	. 65
taskstate	. 67
trapframe	. 74
uproc	. 80

4 Class Index

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

asm.h	81
bio.c	82
bio.d	88
bootasm.d	88
bootmain.c	88
bootmain.d	91
buf.h	92
cat.c	93
cat.d	95
console.c	95
console.d	07
date.h	07
defs.h	07
echo.c	64
echo.d	65
elf.h	65
entryother.d	67
exec.c	67
exec.d	70
exit.c	71
exit.d	71
fcntl.h	72
file.c	73
file.d	79
file.h	79
foo.c	80
foo.d	82
forktest.c 1	82
forktest.d 1	85
fs.c	85
fs.d	206
fs.h	206
grep.c	209
grep.d	213
ide c	11

6 File Index

ide.d	21
init.c	21
	23
initcode.d	23
ioapic.c	23
ioapic.d	28
kalloc.c	28
kalloc.d	32
kbd.c	32
	34
kbd.h	34
kill.c	43
kill.d	44
lapic.c	44
lapic.d	59
In.c	59
In.d	60
log.c	60
log.d	67
	68
ls.d	71
main.c	71
main.d	75
memide.c	75
memlayout.h	78
mkdir.c	80
mkdir.d	31
mkfs.c	31
	93
mp.c	03
·	80
·	09
nice.c	
	12
	12
	15
	16
	17
pipe.d	21
printf.c	21
•	24
	24
•	43
•	43
•	45
•	46
•	46
•	48
•	48
	49
	49
	66
	66
	67
	68
·	70
	70
·	
spinlock.c	70

3.1 File List 7

spinlock.d	75
·	75
	75
	77
	78
	78
	83
•	83
•	98
•	98
	04
	17
7-1	17
	24
	24
· ·	29
	29
	35
	36
	40
	40
	45
	45
	49
	49
	50
	63
	14
	15
	28
	28
	30
	30
	38
zombie.d 53	39

8 File Index

Chapter 4

Class Documentation

4.1 backcmd

Public Attributes

- struct cmd * cmd
- int type

4.1.1 Detailed Description

Definition at line 47 of file sh.c.

4.1.2 Member Data Documentation

4.1.2.1 cmd

```
struct cmd* backcmd::cmd
```

Definition at line 49 of file sh.c.

Referenced by backcmd(), nulterminate(), and runcmd().

4.1.2.2 type

int backcmd::type

Definition at line 48 of file sh.c.

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

• sh.c

4.2 buf

#include <buf.h>

Public Attributes

- · uint blockno
- uchar data [BSIZE]
- uint dev
- · int flags
- struct sleeplock lock
- struct buf * next
- struct buf * prev
- struct buf * qnext
- · uint refcnt

4.2.1 Detailed Description

Definition at line 1 of file buf.h.

4.2.2 Member Data Documentation

4.2.2.1 blockno

uint buf::blockno

Definition at line 4 of file buf.h.

Referenced by bget(), bread(), iderw(), idestart(), and log_write().

4.2.2.2 data

uchar buf::data[BSIZE]

Definition at line 10 of file buf.h.

Referenced by balloc(), bfree(), bmap(), bzero(), ialloc(), ideintr(), iderw(), idestart(), ilock(), install_trans(), itrunc(), iupdate(), read_head(), readi(), readsb(), write_head(), write_log(), and writei().

4.2 buf 11

4.2.2.3 dev

```
uint buf::dev
```

Definition at line 3 of file buf.h.

Referenced by balloc(), bfree(), bget(), bread(), bzero(), iderw(), idestart(), and readsb().

4.2.2.4 flags

```
int buf::flags
```

Definition at line 2 of file buf.h.

Referenced by bget(), bread(), bwrite(), ideintr(), iderw(), idestart(), and log_write().

4.2.2.5 lock

```
struct sleeplock buf::lock
```

Definition at line 5 of file buf.h.

Referenced by bget(), binit(), brelse(), bwrite(), and iderw().

4.2.2.6 next

```
struct buf* buf::next
```

Definition at line 8 of file buf.h.

Referenced by bget(), binit(), and brelse().

4.2.2.7 prev

```
struct buf* buf::prev
```

Definition at line 7 of file buf.h.

Referenced by bget(), binit(), and brelse().

4.2.2.8 qnext

```
struct buf* buf::qnext
```

Definition at line 9 of file buf.h.

Referenced by ideintr(), and iderw().

4.2.2.9 refcnt

```
uint buf::refcnt
```

Definition at line 6 of file buf.h.

Referenced by bget(), and brelse().

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

• buf.h

4.3 cmd

Public Attributes

int type

4.3.1 Detailed Description

Definition at line 16 of file sh.c.

4.3.2 Member Data Documentation

4.3.2.1 type

```
int cmd::type
```

Definition at line 17 of file sh.c.

Referenced by backcmd(), execcmd(), listcmd(), nulterminate(), pipecmd(), redircmd(), and runcmd().

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

• sh.c

4.4 context

4.4 context

```
#include c.h>
```

Public Attributes

- · uint ebp
- uint ebx
- · uint edi
- uint eip
- · uint esi

4.4.1 Detailed Description

Definition at line 27 of file proc.h.

4.4.2 Member Data Documentation

4.4.2.1 ebp

uint context::ebp

Definition at line 31 of file proc.h.

Referenced by procdump().

4.4.2.2 ebx

uint context::ebx

Definition at line 30 of file proc.h.

4.4.2.3 edi

uint context::edi

Definition at line 28 of file proc.h.

4.4.2.4 eip

```
uint context::eip
```

Definition at line 32 of file proc.h.

Referenced by allocproc().

4.4.2.5 esi

```
uint context::esi
```

Definition at line 29 of file proc.h.

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

• proc.h

4.5 cpu

#include c.h>

Public Attributes

- · uchar apicid
- struct segdesc gdt [NSEGS]
- int intena
- int ncli
- struct proc * proc
- struct context * scheduler
- volatile uint started
- · struct taskstate ts

4.5.1 Detailed Description

Definition at line 2 of file proc.h.

4.5.2 Member Data Documentation

4.5 cpu 15

4.5.2.1 apicid

```
uchar cpu::apicid
```

Definition at line 3 of file proc.h.

Referenced by mpinit(), mycpu(), and startothers().

4.5.2.2 gdt

```
struct segdesc cpu::gdt[NSEGS]
```

Definition at line 6 of file proc.h.

Referenced by seginit(), and switchuvm().

4.5.2.3 intena

int cpu::intena

Definition at line 9 of file proc.h.

Referenced by pushcli(), and sched().

4.5.2.4 ncli

int cpu::ncli

Definition at line 8 of file proc.h.

Referenced by pushcli().

4.5.2.5 proc

struct proc* cpu::proc

Definition at line 10 of file proc.h.

Referenced by myproc(), and scheduler().

4.5.2.6 scheduler

```
struct context* cpu::scheduler
```

Definition at line 4 of file proc.h.

Referenced by scheduler().

4.5.2.7 started

```
volatile uint cpu::started
```

Definition at line 7 of file proc.h.

Referenced by startothers().

4.5.2.8 ts

```
struct taskstate cpu::ts
```

Definition at line 5 of file proc.h.

Referenced by switchuvm().

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

· proc.h

4.6 devsw

```
#include <file.h>
```

Public Attributes

```
• int(* read )(struct inode *, char *, int)
```

• int(* write)(struct inode *, char *, int)

4.6.1 Detailed Description

Definition at line 30 of file file.h.

4.7 dinode 17

4.6.2 Member Data Documentation

4.6.2.1 read

```
int(* devsw::read) (struct inode *, char *, int)
```

Definition at line 31 of file file.h.

Referenced by consoleinit(), and readi().

4.6.2.2 write

```
int(* devsw::write) (struct inode *, char *, int)
```

Definition at line 32 of file file.h.

Referenced by consoleinit(), and writei().

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

• file.h

4.7 dinode

```
#include <fs.h>
```

Public Attributes

- uint addrs [NDIRECT+1]
- short major
- short minor
- short nlink
- · uint size
- short type

4.7.1 Detailed Description

Definition at line 29 of file fs.h.

4.7.2 Member Data Documentation

4.7.2.1 addrs

```
uint dinode::addrs[NDIRECT+1]
```

Definition at line 35 of file fs.h.

Referenced by iappend(), ilock(), and iupdate().

4.7.2.2 major

```
short dinode::major
```

Definition at line 31 of file fs.h.

Referenced by ilock(), and iupdate().

4.7.2.3 minor

```
short dinode::minor
```

Definition at line 32 of file fs.h.

Referenced by ilock(), and iupdate().

4.7.2.4 nlink

short dinode::nlink

Definition at line 33 of file fs.h.

Referenced by ialloc(), ilock(), and iupdate().

4.7.2.5 size

uint dinode::size

Definition at line 34 of file fs.h.

Referenced by ialloc(), iappend(), ilock(), iupdate(), and main().

4.8 dirent 19

4.7.2.6 type

```
short dinode::type
```

Definition at line 30 of file fs.h.

Referenced by ialloc(), ilock(), and iupdate().

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

fs.h

4.8 dirent

```
#include <fs.h>
```

Public Attributes

- ushort inum
- char name [DIRSIZ]

4.8.1 Detailed Description

Definition at line 53 of file fs.h.

4.8.2 Member Data Documentation

4.8.2.1 inum

```
ushort dirent::inum
```

Definition at line 54 of file fs.h.

Referenced by dirlink(), dirlookup(), ls(), and main().

4.8.2.2 name

```
char dirent::name[DIRSIZ]
```

Definition at line 55 of file fs.h.

Referenced by dirlink(), ls(), and main().

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

fs.h

4.9 elfhdr

#include <elf.h>

Public Attributes

- ushort ehsize
- uchar elf [12]
- uint entry
- uint flags
- ushort machine
- uint magic
- ushort phentsize
- ushort phnum
- uint phoff
- · ushort shentsize
- ushort shnum
- · uint shoff
- ushort shstrndx
- · ushort type
- uint version

4.9.1 Detailed Description

Definition at line 6 of file elf.h.

4.9.2 Member Data Documentation

4.9.2.1 ehsize

ushort elfhdr::ehsize

Definition at line 16 of file elf.h.

4.9.2.2 elf

uchar elfhdr::elf[12]

Definition at line 8 of file elf.h.

Referenced by bootmain(), and exec().

4.9 elfhdr 21

4.9.2.3 entry

```
uint elfhdr::entry
```

Definition at line 12 of file elf.h.

Referenced by bootmain(), and exec().

4.9.2.4 flags

```
uint elfhdr::flags
```

Definition at line 15 of file elf.h.

4.9.2.5 machine

```
ushort elfhdr::machine
```

Definition at line 10 of file elf.h.

4.9.2.6 magic

```
uint elfhdr::magic
```

Definition at line 7 of file elf.h.

Referenced by exec().

4.9.2.7 phentsize

```
ushort elfhdr::phentsize
```

Definition at line 17 of file elf.h.

4.9.2.8 phnum

ushort elfhdr::phnum

Definition at line 18 of file elf.h.

Referenced by bootmain(), and exec().

4.9.2.9 phoff

```
uint elfhdr::phoff
```

Definition at line 13 of file elf.h.

Referenced by bootmain(), and exec().

4.9.2.10 shentsize

```
ushort elfhdr::shentsize
```

Definition at line 19 of file elf.h.

4.9.2.11 shnum

ushort elfhdr::shnum

Definition at line 20 of file elf.h.

4.9.2.12 shoff

uint elfhdr::shoff

Definition at line 14 of file elf.h.

4.9.2.13 shstrndx

ushort elfhdr::shstrndx

Definition at line 21 of file elf.h.

4.9.2.14 type

ushort elfhdr::type

Definition at line 9 of file elf.h.

4.10 execcmd 23

4.9.2.15 version

```
uint elfhdr::version
```

Definition at line 11 of file elf.h.

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

• elf.h

4.10 execcmd

Public Attributes

- char * argv [MAXARGS]
- char * eargv [MAXARGS]
- int type

4.10.1 Detailed Description

Definition at line 20 of file sh.c.

4.10.2 Member Data Documentation

4.10.2.1 argv

```
char* execcmd::argv[MAXARGS]
```

Definition at line 22 of file sh.c.

Referenced by nulterminate(), and runcmd().

4.10.2.2 eargv

char* execcmd::eargv[MAXARGS]

Definition at line 23 of file sh.c.

Referenced by nulterminate().

4.10.2.3 type

```
int execcmd::type
```

Definition at line 21 of file sh.c.

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

• sh.c

4.11 file

```
#include <file.h>
```

Public Types

enum { FD_NONE , FD_PIPE , FD_INODE }

Public Attributes

- struct inode * ip
- uint off
- struct pipe * pipe
- char readable
- int ref
- enum file:: { ... } type
- char writable

4.11.1 Detailed Description

Definition at line 1 of file file.h.

4.11.2 Member Enumeration Documentation

4.11.2.1 anonymous enum

anonymous enum

Enumerator

FD_NONE	
FD_PIPE	
FD_INODE	

4.11 file 25

```
Definition at line 2 of file file.h.
00002 { FD_NONE, FD_PIPE, FD_INODE } type;
```

4.11.3 Member Data Documentation

4.11.3.1 ip

```
struct inode* file::ip
```

Definition at line 7 of file file.h.

Referenced by fileclose(), fileread(), filestat(), filewrite(), and sys_open().

4.11.3.2 off

uint file::off

Definition at line 8 of file file.h.

Referenced by fileread(), filewrite(), and sys_open().

4.11.3.3 pipe

```
struct pipe* file::pipe
```

Definition at line 6 of file file.h.

Referenced by fileclose(), fileread(), and filewrite().

4.11.3.4 readable

char file::readable

Definition at line 4 of file file.h.

Referenced by fileread(), and sys_open().

4.11.3.5 ref

```
int file::ref
```

Definition at line 3 of file file.h.

Referenced by filealloc(), fileclose(), and filedup().

4.11.3.6

```
enum { \dots } file::type
```

Referenced by fileclose(), fileread(), filestat(), filewrite(), and sys_open().

4.11.3.7 writable

```
char file::writable
```

Definition at line 5 of file file.h.

Referenced by fileclose(), filewrite(), and sys_open().

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

• file.h

4.12 gatedesc

```
#include <mmu.h>
```

Public Attributes

- uint args: 5
- uint cs: 16
- uint dpl: 2
- uint off_15_0: 16
- uint off_31_16: 16
- uint p: 1
- uint rsv1: 3
- uint s: 1
- uint type: 4

4.12 gatedesc 27

4.12.1 Detailed Description

Definition at line 148 of file mmu.h.

4.12.2 Member Data Documentation

4.12.2.1 args

```
uint gatedesc::args
```

Definition at line 151 of file mmu.h.

4.12.2.2 cs

uint gatedesc::cs

Definition at line 150 of file mmu.h.

4.12.2.3 dpl

```
uint gatedesc::dpl
```

Definition at line 155 of file mmu.h.

4.12.2.4 off_15_0

```
uint gatedesc::off_15_0
```

Definition at line 149 of file mmu.h.

4.12.2.5 off_31_16

uint gatedesc::off_31_16

Definition at line 157 of file mmu.h.

4.12.2.6 p

```
uint gatedesc::p
```

Definition at line 156 of file mmu.h.

Referenced by lidt().

4.12.2.7 rsv1

```
uint gatedesc::rsv1
```

Definition at line 152 of file mmu.h.

4.12.2.8 s

```
uint gatedesc::s
```

Definition at line 154 of file mmu.h.

4.12.2.9 type

```
uint gatedesc::type
```

Definition at line 153 of file mmu.h.

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

• mmu.h

4.13 header

Public Attributes

```
struct {
    union header * ptr
    uint size
    } s
```

• Align x

4.13 header 29

4.13.1 Detailed Description

Definition at line 11 of file umalloc.c.

4.13.2 Member Data Documentation

4.13.2.1 ptr

```
union header* header::ptr
```

Definition at line 13 of file umalloc.c.

Referenced by free(), and malloc().

4.13.2.2

```
struct { ... } header::s
```

Referenced by free(), malloc(), and morecore().

4.13.2.3 size

```
uint header::size
```

Definition at line 14 of file umalloc.c.

Referenced by free(), malloc(), and morecore().

4.13.2.4 x

```
Align header::x
```

Definition at line 16 of file umalloc.c.

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

• umalloc.c

4.14 inode

#include <file.h>

Public Attributes

- uint addrs [NDIRECT+1]
- · uint dev
- · uint inum
- struct sleeplock lock
- short major
- short minor
- short nlink
- int ref
- · uint size
- short type
- int valid

4.14.1 Detailed Description

Definition at line 13 of file file.h.

4.14.2 Member Data Documentation

4.14.2.1 addrs

uint inode::addrs[NDIRECT+1]

Definition at line 25 of file file.h.

Referenced by bmap(), ilock(), itrunc(), and iupdate().

4.14.2.2 dev

uint inode::dev

Definition at line 14 of file file.h.

Referenced by bmap(), create(), dirlookup(), iget(), iinit(), ilock(), itrunc(), iupdate(), readi(), stati(), sys_link(), and writei().

4.14 inode 31

4.14.2.3 inum

```
uint inode::inum
```

Definition at line 15 of file file.h.

Referenced by create(), dirlink(), dirlookup(), ialloc(), iget(), ilock(), iupdate(), stati(), and sys_link().

4.14.2.4 lock

```
struct sleeplock inode::lock
```

Definition at line 17 of file file.h.

Referenced by ilock(), iput(), and iunlock().

4.14.2.5 major

```
short inode::major
```

Definition at line 21 of file file.h.

Referenced by create(), ilock(), iupdate(), readi(), sys_mknod(), and writei().

4.14.2.6 minor

short inode::minor

Definition at line 22 of file file.h.

Referenced by create(), ilock(), iupdate(), and sys_mknod().

4.14.2.7 nlink

short inode::nlink

Definition at line 23 of file file.h.

Referenced by create(), ilock(), iput(), iupdate(), stati(), sys_link(), and sys_unlink().

4.14.2.8 ref

```
int inode::ref
```

Definition at line 16 of file file.h.

Referenced by idup(), iget(), ilock(), iput(), and iunlock().

4.14.2.9 size

```
uint inode::size
```

Definition at line 24 of file file.h.

Referenced by dirlink(), dirlookup(), ilock(), isdirempty(), itrunc(), iupdate(), readi(), stati(), and writei().

4.14.2.10 type

```
short inode::type
```

Definition at line 20 of file file.h.

Referenced by create(), dirlookup(), ilock(), iput(), iupdate(), namex(), readi(), stati(), sys_chdir(), sys_link(), sys_open(), sys_unlink(), and writei().

4.14.2.11 valid

int inode::valid

Definition at line 18 of file file.h.

Referenced by iget(), ilock(), and iput().

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

• file.h

4.15 ioapic

Public Attributes

- uint data
- uint pad [3]
- uint reg

4.16 kmap 33

4.15.1 Detailed Description

Definition at line 28 of file ioapic.c.

4.15.2 Member Data Documentation

4.15.2.1 data

```
uint ioapic::data
```

Definition at line 31 of file ioapic.c.

Referenced by ioapicread(), and ioapicwrite().

4.15.2.2 pad

```
uint ioapic::pad[3]
```

Definition at line 30 of file ioapic.c.

4.15.2.3 reg

```
uint ioapic::reg
```

Definition at line 29 of file ioapic.c.

Referenced by ioapicread(), and ioapicwrite().

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

· ioapic.c

4.16 kmap

Public Attributes

- int perm
- uint phys_end
- uint phys_start
- void * virt

4.16.1 Detailed Description

Definition at line 105 of file vm.c.

4.16.2 Member Data Documentation

4.16.2.1 perm

```
int kmap::perm
```

Definition at line 109 of file vm.c.

Referenced by setupkvm().

4.16.2.2 phys_end

```
uint kmap::phys_end
```

Definition at line 108 of file vm.c.

Referenced by setupkvm().

4.16.2.3 phys_start

```
uint kmap::phys_start
```

Definition at line 107 of file vm.c.

Referenced by setupkvm().

4.16.2.4 virt

void* kmap::virt

Definition at line 106 of file vm.c.

Referenced by setupkvm().

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

vm.c

4.17 listcmd 35

4.17 listcmd

Public Attributes

```
    struct cmd * left
```

- struct cmd * right
- int type

4.17.1 Detailed Description

Definition at line 41 of file sh.c.

4.17.2 Member Data Documentation

4.17.2.1 left

```
struct cmd* listcmd::left
```

Definition at line 43 of file sh.c.

Referenced by listcmd(), nulterminate(), and runcmd().

4.17.2.2 right

```
struct cmd* listcmd::right
```

Definition at line 44 of file sh.c.

Referenced by listcmd(), nulterminate(), and runcmd().

4.17.2.3 type

```
int listcmd::type
```

Definition at line 42 of file sh.c.

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

• sh.c

4.18 log

Public Attributes

- int committing
- int dev
- · struct logheader Ih
- struct spinlock lock
- · int outstanding
- int size
- int start

4.18.1 Detailed Description

Definition at line 39 of file log.c.

4.18.2 Member Data Documentation

4.18.2.1 committing

```
int log::committing
```

Definition at line 44 of file log.c.

Referenced by begin_op(), and end_op().

4.18.2.2 dev

```
int log::dev
```

Definition at line 45 of file log.c.

Referenced by initlog(), install_trans(), read_head(), write_head(), and write_log().

4.18.2.3 lh

```
struct logheader log::lh
```

Definition at line 46 of file log.c.

Referenced by begin_op(), commit(), install_trans(), log_write(), read_head(), recover_from_log(), write_head(), and write_log().

4.19 logheader 37

4.18.2.4 lock

```
struct spinlock log::lock
```

Definition at line 40 of file log.c.

Referenced by begin_op(), end_op(), initlog(), and log_write().

4.18.2.5 outstanding

```
int log::outstanding
```

Definition at line 43 of file log.c.

Referenced by begin_op(), end_op(), and log_write().

4.18.2.6 size

```
int log::size
```

Definition at line 42 of file log.c.

Referenced by initlog(), and log_write().

4.18.2.7 start

```
int log::start
```

Definition at line 41 of file log.c.

Referenced by initlog(), install_trans(), read_head(), write_head(), and write_log().

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

• log.c

4.19 logheader

Public Attributes

- int block [LOGSIZE]
- int n

4.19.1 Detailed Description

Definition at line 34 of file log.c.

4.19.2 Member Data Documentation

4.19.2.1 block

```
int logheader::block[LOGSIZE]
```

Definition at line 36 of file log.c.

Referenced by install_trans(), log_write(), read_head(), write_head(), and write_log().

4.19.2.2 n

int logheader::n

Definition at line 35 of file log.c.

Referenced by begin_op(), commit(), install_trans(), log_write(), read_head(), recover_from_log(), write_head(), and write_log().

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

· log.c

4.20 mp

#include <mp.h>

Public Attributes

- uchar checksum
- · uchar imcrp
- · uchar length
- void * physaddr
- uchar reserved [3]
- uchar signature [4]
- uchar specrev
- uchar type

4.20 mp 39

4.20.1 Detailed Description

Definition at line 3 of file mp.h.

4.20.2 Member Data Documentation

4.20.2.1 checksum

uchar mp::checksum

Definition at line 8 of file mp.h.

4.20.2.2 imcrp

uchar mp::imcrp

Definition at line 10 of file mp.h.

Referenced by mpinit().

4.20.2.3 length

uchar mp::length

Definition at line 6 of file mp.h.

4.20.2.4 physaddr

void* mp::physaddr

Definition at line 5 of file mp.h.

Referenced by mpconfig().

4.20.2.5 reserved

```
uchar mp::reserved[3]
```

Definition at line 11 of file mp.h.

4.20.2.6 signature

```
uchar mp::signature[4]
```

Definition at line 4 of file mp.h.

4.20.2.7 specrev

```
uchar mp::specrev
```

Definition at line 7 of file mp.h.

4.20.2.8 type

```
uchar mp::type
```

Definition at line 9 of file mp.h.

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

• mp.h

4.21 mpconf

#include <mp.h>

Public Attributes

- uchar checksum
- ushort entry
- uint * lapicaddr
- · ushort length
- ushort oemlength
- uint * oemtable
- uchar product [20]
- · uchar reserved
- uchar signature [4]
- uchar version
- uchar xchecksum
- · ushort xlength

4.21 mpconf 41

4.21.1 Detailed Description

Definition at line 14 of file mp.h.

4.21.2 Member Data Documentation

4.21.2.1 checksum

uchar mpconf::checksum

Definition at line 18 of file mp.h.

4.21.2.2 entry

ushort mpconf::entry

Definition at line 22 of file mp.h.

4.21.2.3 lapicaddr

uint* mpconf::lapicaddr

Definition at line 23 of file mp.h.

Referenced by mpinit().

4.21.2.4 length

ushort mpconf::length

Definition at line 16 of file mp.h.

Referenced by mpconfig(), and mpinit().

4.21.2.5 oemlength

```
ushort mpconf::oemlength
```

Definition at line 21 of file mp.h.

4.21.2.6 oemtable

```
uint* mpconf::oemtable
```

Definition at line 20 of file mp.h.

4.21.2.7 product

```
uchar mpconf::product[20]
```

Definition at line 19 of file mp.h.

4.21.2.8 reserved

uchar mpconf::reserved

Definition at line 26 of file mp.h.

4.21.2.9 signature

uchar mpconf::signature[4]

Definition at line 15 of file mp.h.

4.21.2.10 version

uchar mpconf::version

Definition at line 17 of file mp.h.

Referenced by mpconfig().

4.22 mpioapic 43

4.21.2.11 xchecksum

```
uchar mpconf::xchecksum
```

Definition at line 25 of file mp.h.

4.21.2.12 xlength

```
ushort mpconf::xlength
```

Definition at line 24 of file mp.h.

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

• mp.h

4.22 mpioapic

#include <mp.h>

Public Attributes

- uint * addr
- uchar apicno
- · uchar flags
- uchar type
- · uchar version

4.22.1 Detailed Description

Definition at line 40 of file mp.h.

4.22.2 Member Data Documentation

4.22.2.1 addr

uint* mpioapic::addr

Definition at line 45 of file mp.h.

4.22.2.2 apicno

```
uchar mpioapic::apicno
```

Definition at line 42 of file mp.h.

4.22.2.3 flags

```
uchar mpioapic::flags
```

Definition at line 44 of file mp.h.

4.22.2.4 type

```
uchar mpioapic::type
```

Definition at line 41 of file mp.h.

4.22.2.5 version

```
uchar mpioapic::version
```

Definition at line 43 of file mp.h.

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

• mp.h

4.23 mpproc

#include <mp.h>

Public Attributes

- · uchar apicid
- uint feature
- uchar flags
- uchar reserved [8]
- uchar signature [4]
- uchar type
- · uchar version

4.23 mpproc 45

4.23.1 Detailed Description

Definition at line 29 of file mp.h.

4.23.2 Member Data Documentation

4.23.2.1 apicid

uchar mpproc::apicid

Definition at line 31 of file mp.h.

4.23.2.2 feature

uint mpproc::feature

Definition at line 36 of file mp.h.

4.23.2.3 flags

uchar mpproc::flags

Definition at line 33 of file mp.h.

4.23.2.4 reserved

uchar mpproc::reserved[8]

Definition at line 37 of file mp.h.

4.23.2.5 signature

uchar mpproc::signature[4]

Definition at line 35 of file mp.h.

4.23.2.6 type

```
uchar mpproc::type
```

Definition at line 30 of file mp.h.

4.23.2.7 version

```
uchar mpproc::version
```

Definition at line 32 of file mp.h.

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

• mp.h

4.24 pipe

Public Attributes

- char data [PIPESIZE]
- struct spinlock lock
- · uint nread
- · uint nwrite
- int readopen
- · int writeopen

4.24.1 Detailed Description

Definition at line 13 of file pipe.c.

4.24.2 Member Data Documentation

4.24.2.1 data

char pipe::data[PIPESIZE]

Definition at line 15 of file pipe.c.

Referenced by piperead(), and pipewrite().

4.24 pipe 47

4.24.2.2 lock

```
struct spinlock pipe::lock
```

Definition at line 14 of file pipe.c.

Referenced by pipealloc(), pipeclose(), piperead(), and pipewrite().

4.24.2.3 nread

```
uint pipe::nread
```

Definition at line 16 of file pipe.c.

Referenced by pipealloc(), pipeclose(), piperead(), and pipewrite().

4.24.2.4 nwrite

```
uint pipe::nwrite
```

Definition at line 17 of file pipe.c.

Referenced by pipealloc(), pipeclose(), piperead(), and pipewrite().

4.24.2.5 readopen

```
int pipe::readopen
```

Definition at line 18 of file pipe.c.

Referenced by pipealloc(), pipeclose(), and pipewrite().

4.24.2.6 writeopen

```
int pipe::writeopen
```

Definition at line 19 of file pipe.c.

Referenced by pipealloc(), pipeclose(), and piperead().

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

pipe.c

4.25 pipecmd

Public Attributes

```
    struct cmd * left
```

- struct cmd * right
- int type

4.25.1 Detailed Description

Definition at line 35 of file sh.c.

4.25.2 Member Data Documentation

4.25.2.1 left

```
struct cmd* pipecmd::left
```

Definition at line 37 of file sh.c.

Referenced by nulterminate(), pipecmd(), and runcmd().

4.25.2.2 right

```
struct cmd* pipecmd::right
```

Definition at line 38 of file sh.c.

Referenced by nulterminate(), pipecmd(), and runcmd().

4.25.2.3 type

```
int pipecmd::type
```

Definition at line 36 of file sh.c.

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

• sh.c

4.26 proc 49

4.26 proc

```
#include c.h>
```

Public Attributes

- void * chan
- struct context * context
- struct inode * cwd
- int killed
- char * kstack
- char name [16]
- struct file * ofile [NOFILE]
- struct proc * parent
- pde_t * pgdir
- int pid
- int priority
- enum procstate state
- uint sz
- struct trapframe * tf

4.26.1 Detailed Description

Definition at line 38 of file proc.h.

4.26.2 Member Data Documentation

4.26.2.1 chan

void* proc::chan

Definition at line 47 of file proc.h.

Referenced by sleep(), wakeup(), and wakeup1().

4.26.2.2 context

struct context* proc::context

Definition at line 46 of file proc.h.

Referenced by allocproc(), procdump(), sched(), and scheduler().

4.26.2.3 cwd

```
struct inode* proc::cwd
```

Definition at line 50 of file proc.h.

Referenced by exit(), fork(), sys_chdir(), and userinit().

4.26.2.4 killed

```
int proc::killed
```

Definition at line 48 of file proc.h.

Referenced by kill(), sys_sleep(), trap(), and wait().

4.26.2.5 kstack

```
char* proc::kstack
```

Definition at line 41 of file proc.h.

Referenced by allocproc(), fork(), switchuvm(), and wait().

4.26.2.6 name

```
char proc::name[16]
```

Definition at line 51 of file proc.h.

Referenced by cps(), exec(), fork(), procdump(), sys_getprocs(), syscall(), userinit(), and wait().

4.26.2.7 ofile

```
struct file* proc::ofile[NOFILE]
```

Definition at line 49 of file proc.h.

Referenced by exit(), fdalloc(), fork(), sys_close(), and sys_pipe().

4.26 proc 51

4.26.2.8 parent

```
struct proc* proc::parent
```

Definition at line 44 of file proc.h.

Referenced by exit(), fork(), sys_getprocs(), and wait().

4.26.2.9 pgdir

```
pde_t* proc::pgdir
```

Definition at line 40 of file proc.h.

Referenced by exec(), fork(), growproc(), switchuvm(), userinit(), and wait().

4.26.2.10 pid

int proc::pid

Definition at line 43 of file proc.h.

Referenced by acquiresleep(), allocproc(), chpr(), cps(), fork(), holdingsleep(), kill(), procdump(), sys_chpr(), sys_getpid(), sys_getprocs(), sys_kill(), syscall(), and wait().

4.26.2.11 priority

int proc::priority

Definition at line 52 of file proc.h.

Referenced by chpr(), and cps().

4.26.2.12 state

enum procstate proc::state

Definition at line 42 of file proc.h.

Referenced by allocproc(), cps(), exit(), fork(), kill(), procdump(), sched(), scheduler(), sleep(), sys_getprocs(), userinit(), wait(), wakeup1(), and yield().

4.26.2.13 sz

```
uint proc::sz
```

Definition at line 39 of file proc.h.

Referenced by argptr(), exec(), fetchint(), fetchstr(), fork(), growproc(), sys_sbrk(), and userinit().

4.26.2.14 tf

```
struct trapframe* proc::tf
```

Definition at line 45 of file proc.h.

Referenced by allocproc(), argint(), exec(), fork(), syscall(), trap(), and userinit().

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

• proc.h

4.27 proghdr

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

Public Attributes

- uint align
- uint filesz
- · uint flags
- uint memsz
- uint off
- uint paddr
- uint type
- · uint vaddr

4.27.1 Detailed Description

Definition at line 25 of file elf.h.

4.27.2 Member Data Documentation

4.27 proghdr 53

4.27.2.1 align

```
uint proghdr::align
```

Definition at line 33 of file elf.h.

4.27.2.2 filesz

```
uint proghdr::filesz
```

Definition at line 30 of file elf.h.

Referenced by bootmain().

4.27.2.3 flags

uint proghdr::flags

Definition at line 32 of file elf.h.

4.27.2.4 memsz

uint proghdr::memsz

Definition at line 31 of file elf.h.

Referenced by bootmain().

4.27.2.5 off

uint proghdr::off

Definition at line 27 of file elf.h.

Referenced by bootmain().

4.27.2.6 paddr

```
uint proghdr::paddr
```

Definition at line 29 of file elf.h.

Referenced by bootmain().

4.27.2.7 type

```
uint proghdr::type
```

Definition at line 26 of file elf.h.

4.27.2.8 vaddr

```
uint proghdr::vaddr
```

Definition at line 28 of file elf.h.

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

• elf.h

4.28 redircmd

Public Attributes

- struct cmd * cmd
- char * efile
- int fd
- char * file
- int mode
- int type

4.28.1 Detailed Description

Definition at line 26 of file sh.c.

4.28.2 Member Data Documentation

4.28 redircmd 55

4.28.2.1 cmd

```
struct cmd* redircmd::cmd
```

Definition at line 28 of file sh.c.

Referenced by nulterminate(), redircmd(), and runcmd().

4.28.2.2 efile

```
char* redircmd::efile
```

Definition at line 30 of file sh.c.

Referenced by nulterminate(), and redircmd().

4.28.2.3 fd

int redircmd::fd

Definition at line 32 of file sh.c.

Referenced by redircmd(), and runcmd().

4.28.2.4 file

char* redircmd::file

Definition at line 29 of file sh.c.

Referenced by redircmd(), and runcmd().

4.28.2.5 mode

int redircmd::mode

Definition at line 31 of file sh.c.

Referenced by redircmd(), and runcmd().

4.28.2.6 type

```
int redircmd::type
```

Definition at line 27 of file sh.c.

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

• sh.c

4.29 rtcdate

```
#include <date.h>
```

Public Attributes

- uint day
- · uint hour
- · uint minute
- uint month
- · uint second
- uint year

4.29.1 Detailed Description

Definition at line 1 of file date.h.

4.29.2 Member Data Documentation

4.29.2.1 day

uint rtcdate::day

Definition at line 5 of file date.h.

Referenced by cmostime().

4.29 rtcdate 57

4.29.2.2 hour

```
uint rtcdate::hour
```

Definition at line 4 of file date.h.

Referenced by cmostime().

4.29.2.3 minute

```
uint rtcdate::minute
```

Definition at line 3 of file date.h.

Referenced by cmostime().

4.29.2.4 month

```
uint rtcdate::month
```

Definition at line 6 of file date.h.

Referenced by cmostime().

4.29.2.5 second

```
uint rtcdate::second
```

Definition at line 2 of file date.h.

Referenced by cmostime().

4.29.2.6 year

```
uint rtcdate::year
```

Definition at line 7 of file date.h.

Referenced by cmostime().

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

· date.h

4.30 run

Public Attributes

struct run * next

4.30.1 Detailed Description

Definition at line 16 of file kalloc.c.

4.30.2 Member Data Documentation

4.30.2.1 next

```
struct run* run::next
```

Definition at line 17 of file kalloc.c.

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

• kalloc.c

4.31 segdesc

```
#include <mmu.h>
```

Public Attributes

```
• uint avl: 1
```

- uint base_15_0: 16
- uint base_23_16: 8
- uint base_31_24: 8
- uint db: 1
- uint dpl: 2
- uint g: 1
- uint lim_15_0: 16
- uint lim_19_16: 4
- uint p: 1
- uint rsv1: 1
- uint s: 1
- uint type: 4

4.31 segdesc 59

4.31.1 Detailed Description

Definition at line 26 of file mmu.h.

4.31.2 Member Data Documentation

4.31.2.1 avl

uint segdesc::avl

Definition at line 35 of file mmu.h.

4.31.2.2 base_15_0

uint segdesc::base_15_0

Definition at line 28 of file mmu.h.

4.31.2.3 base_23_16

uint segdesc::base_23_16

Definition at line 29 of file mmu.h.

4.31.2.4 base_31_24

uint segdesc::base_31_24

Definition at line 39 of file mmu.h.

4.31.2.5 db

uint segdesc::db

Definition at line 37 of file mmu.h.

4.31.2.6 dpl

```
uint segdesc::dpl
```

Definition at line 32 of file mmu.h.

4.31.2.7 g

```
uint segdesc::g
```

Definition at line 38 of file mmu.h.

4.31.2.8 lim_15_0

```
uint segdesc::lim_15_0
```

Definition at line 27 of file mmu.h.

4.31.2.9 lim_19_16

```
uint segdesc::lim_19_16
```

Definition at line 34 of file mmu.h.

4.31.2.10 p

```
uint segdesc::p
```

Definition at line 33 of file mmu.h.

Referenced by lgdt().

4.31.2.11 rsv1

uint segdesc::rsv1

Definition at line 36 of file mmu.h.

4.32 sleeplock 61

4.31.2.12 s

```
uint segdesc::s
```

Definition at line 31 of file mmu.h.

Referenced by switchuvm().

4.31.2.13 type

```
uint segdesc::type
```

Definition at line 30 of file mmu.h.

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

• mmu.h

4.32 sleeplock

```
#include <sleeplock.h>
```

Public Attributes

- struct spinlock lk
- uint locked
- char * name
- int pid

4.32.1 Detailed Description

Definition at line 2 of file sleeplock.h.

4.32.2 Member Data Documentation

4.32.2.1 lk

```
struct spinlock sleeplock::lk
```

Definition at line 4 of file sleeplock.h.

Referenced by acquiresleep(), holdingsleep(), initsleeplock(), and releasesleep().

4.32.2.2 locked

```
uint sleeplock::locked
```

Definition at line 3 of file sleeplock.h.

Referenced by acquiresleep(), holdingsleep(), initsleeplock(), and releasesleep().

4.32.2.3 name

```
char* sleeplock::name
```

Definition at line 7 of file sleeplock.h.

Referenced by initsleeplock().

4.32.2.4 pid

```
int sleeplock::pid
```

Definition at line 8 of file sleeplock.h.

Referenced by acquiresleep(), holdingsleep(), initsleeplock(), and releasesleep().

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

• sleeplock.h

4.33 spinlock

```
#include <spinlock.h>
```

Public Attributes

- struct cpu * cpu
- · uint locked
- char * name
- uint pcs [10]

4.33.1 Detailed Description

Definition at line 2 of file spinlock.h.

4.34 stat 63

4.33.2 Member Data Documentation

4.33.2.1 cpu

```
struct cpu* spinlock::cpu
```

Definition at line 7 of file spinlock.h.

Referenced by acquire(), holding(), initlock(), and release().

4.33.2.2 locked

```
uint spinlock::locked
```

Definition at line 3 of file spinlock.h.

Referenced by acquire(), holding(), initlock(), and release().

4.33.2.3 name

```
char* spinlock::name
```

Definition at line 6 of file spinlock.h.

Referenced by initlock().

4.33.2.4 pcs

```
uint spinlock::pcs[10]
```

Definition at line 8 of file spinlock.h.

Referenced by acquire(), panic(), and release().

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

• spinlock.h

4.34 stat

#include <stat.h>

Public Attributes

- int dev
- uint ino
- short nlink
- · uint size
- short type

4.34.1 Detailed Description

Definition at line 5 of file stat.h.

4.34.2 Member Data Documentation

4.34.2.1 dev

int stat::dev

Definition at line 7 of file stat.h.

Referenced by stati().

4.34.2.2 ino

uint stat::ino

Definition at line 8 of file stat.h.

Referenced by Is(), and stati().

4.34.2.3 nlink

short stat::nlink

Definition at line 9 of file stat.h.

Referenced by stati().

4.35 superblock 65

4.34.2.4 size

```
uint stat::size
```

Definition at line 10 of file stat.h.

Referenced by Is(), and stati().

4.34.2.5 type

```
short stat::type
```

Definition at line 6 of file stat.h.

Referenced by Is(), and stati().

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

• stat.h

4.35 superblock

```
#include <fs.h>
```

Public Attributes

- uint bmapstart
- uint inodestart
- · uint logstart
- uint nblocks
- uint ninodes
- uint nlog
- uint size

4.35.1 Detailed Description

Definition at line 14 of file fs.h.

4.35.2 Member Data Documentation

4.35.2.1 bmapstart

```
uint superblock::bmapstart
```

Definition at line 21 of file fs.h.

Referenced by balloc(), iinit(), and main().

4.35.2.2 inodestart

```
uint superblock::inodestart
```

Definition at line 20 of file fs.h.

Referenced by iinit(), and main().

4.35.2.3 logstart

```
uint superblock::logstart
```

Definition at line 19 of file fs.h.

Referenced by iinit(), initlog(), and main().

4.35.2.4 nblocks

```
uint superblock::nblocks
```

Definition at line 16 of file fs.h.

Referenced by iinit(), and main().

4.35.2.5 ninodes

uint superblock::ninodes

Definition at line 17 of file fs.h.

Referenced by ialloc(), iinit(), and main().

4.36 taskstate 67

4.35.2.6 nlog

```
uint superblock::nlog
```

Definition at line 18 of file fs.h.

Referenced by iinit(), initlog(), and main().

4.35.2.7 size

```
uint superblock::size
```

Definition at line 15 of file fs.h.

Referenced by balloc(), iinit(), and main().

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

fs.h

4.36 taskstate

#include <mmu.h>

Public Attributes

- void * cr3
- · ushort cs
- · ushort ds
- uint eax
- uint * ebp
- uint ebx
- uint ecx
- uint edi
- uint edx
- · uint eflags
- uint * eip
- ushort es
- uint esi
- uint * esp
- uint esp0
- uint * esp1
- uint * esp2
- ushort fs
- · ushort gs
- · ushort iomb
- ushort ldt
- · uint link

68 Class Documentation

- ushort padding1
- ushort padding10
- ushort padding2
- ushort padding3
- ushort padding4
- · ushort padding5
- ushort padding6
- ushort padding7
- ushort padding8
- ushort padding9
- ushort ss
- ushort ss0
- ushort ss1
- ushort ss2
- ushort t

4.36.1 Detailed Description

Definition at line 107 of file mmu.h.

4.36.2 Member Data Documentation

4.36.2.1 cr3

void* taskstate::cr3

Definition at line 118 of file mmu.h.

4.36.2.2 cs

ushort taskstate::cs

Definition at line 131 of file mmu.h.

4.36.2.3 ds

ushort taskstate::ds

Definition at line 135 of file mmu.h.

4.36 taskstate 69

4.36.2.4 eax

uint taskstate::eax

Definition at line 121 of file mmu.h.

4.36.2.5 ebp

uint* taskstate::ebp

Definition at line 126 of file mmu.h.

4.36.2.6 ebx

uint taskstate::ebx

Definition at line 124 of file mmu.h.

4.36.2.7 ecx

uint taskstate::ecx

Definition at line 122 of file mmu.h.

4.36.2.8 edi

uint taskstate::edi

Definition at line 128 of file mmu.h.

4.36.2.9 edx

uint taskstate::edx

Definition at line 123 of file mmu.h.

70 Class Documentation

4.36.2.10 eflags

```
uint taskstate::eflags
```

Definition at line 120 of file mmu.h.

4.36.2.11 eip

```
uint* taskstate::eip
```

Definition at line 119 of file mmu.h.

4.36.2.12 es

ushort taskstate::es

Definition at line 129 of file mmu.h.

4.36.2.13 esi

uint taskstate::esi

Definition at line 127 of file mmu.h.

4.36.2.14 esp

uint* taskstate::esp

Definition at line 125 of file mmu.h.

4.36.2.15 esp0

uint taskstate::esp0

Definition at line 109 of file mmu.h.

Referenced by switchuvm().

4.36 taskstate 71

4.36.2.16 esp1

```
uint* taskstate::esp1
```

Definition at line 112 of file mmu.h.

4.36.2.17 esp2

```
uint* taskstate::esp2
```

Definition at line 115 of file mmu.h.

4.36.2.18 fs

ushort taskstate::fs

Definition at line 137 of file mmu.h.

4.36.2.19 gs

ushort taskstate::gs

Definition at line 139 of file mmu.h.

4.36.2.20 iomb

ushort taskstate::iomb

Definition at line 144 of file mmu.h.

Referenced by switchuvm().

4.36.2.21 ldt

ushort taskstate::ldt

Definition at line 141 of file mmu.h.

72 Class Documentation

4.36.2.22 link

uint taskstate::link

Definition at line 108 of file mmu.h.

4.36.2.23 padding1

ushort taskstate::padding1

Definition at line 111 of file mmu.h.

4.36.2.24 padding10

ushort taskstate::padding10

Definition at line 142 of file mmu.h.

4.36.2.25 padding2

ushort taskstate::padding2

Definition at line 114 of file mmu.h.

4.36.2.26 padding3

ushort taskstate::padding3

Definition at line 117 of file mmu.h.

4.36.2.27 padding4

ushort taskstate::padding4

Definition at line 130 of file mmu.h.

4.36 taskstate 73

4.36.2.28 padding5

ushort taskstate::padding5

Definition at line 132 of file mmu.h.

4.36.2.29 padding6

ushort taskstate::padding6

Definition at line 134 of file mmu.h.

4.36.2.30 padding7

ushort taskstate::padding7

Definition at line 136 of file mmu.h.

4.36.2.31 padding8

ushort taskstate::padding8

Definition at line 138 of file mmu.h.

4.36.2.32 padding9

ushort taskstate::padding9

Definition at line 140 of file mmu.h.

4.36.2.33 ss

ushort taskstate::ss

Definition at line 133 of file mmu.h.

74 Class Documentation

4.36.2.34 ss0

ushort taskstate::ss0

Definition at line 110 of file mmu.h.

Referenced by switchuvm().

4.36.2.35 ss1

ushort taskstate::ss1

Definition at line 113 of file mmu.h.

4.36.2.36 ss2

ushort taskstate::ss2

Definition at line 116 of file mmu.h.

4.36.2.37 t

ushort taskstate::t

Definition at line 143 of file mmu.h.

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

• mmu.h

4.37 trapframe

#include <x86.h>

4.37 trapframe 75

Public Attributes

- · ushort cs
- · ushort ds
- uint eax
- uint ebp
- uint ebx
- uint ecx
- · uint edi
- uint edx
- uint eflags
- · uint eip
- uint err
- · ushort es
- uint esi
- uint esp
- · ushort fs
- ushort gs
- uint oesp
- ushort padding1
- · ushort padding2
- ushort padding3
- ushort padding4
- ushort padding5
- ushort padding6
- · ushort ss
- uint trapno

4.37.1 Detailed Description

Definition at line 150 of file x86.h.

4.37.2 Member Data Documentation

4.37.2.1 cs

ushort trapframe::cs

Definition at line 175 of file x86.h.

Referenced by trap(), and userinit().

76 Class Documentation

4.37.2.2 ds

```
ushort trapframe::ds
```

Definition at line 168 of file x86.h.

Referenced by userinit().

4.37.2.3 eax

```
uint trapframe::eax
```

Definition at line 159 of file x86.h.

Referenced by fork(), and syscall().

4.37.2.4 ebp

```
uint trapframe::ebp
```

Definition at line 154 of file x86.h.

4.37.2.5 ebx

```
uint trapframe::ebx
```

Definition at line 156 of file x86.h.

4.37.2.6 ecx

```
uint trapframe::ecx
```

Definition at line 158 of file x86.h.

4.37.2.7 edi

uint trapframe::edi

Definition at line 152 of file x86.h.

4.37 trapframe 77

4.37.2.8 edx

```
uint trapframe::edx
```

Definition at line 157 of file x86.h.

4.37.2.9 eflags

```
uint trapframe::eflags
```

Definition at line 177 of file x86.h.

Referenced by userinit().

4.37.2.10 eip

```
uint trapframe::eip
```

Definition at line 174 of file x86.h.

Referenced by exec(), trap(), and userinit().

4.37.2.11 err

uint trapframe::err

Definition at line 173 of file x86.h.

Referenced by trap().

4.37.2.12 es

ushort trapframe::es

Definition at line 166 of file x86.h.

Referenced by userinit().

78 Class Documentation

4.37.2.13 esi

```
uint trapframe::esi
```

Definition at line 153 of file x86.h.

4.37.2.14 esp

```
uint trapframe::esp
```

Definition at line 180 of file x86.h.

Referenced by argint(), exec(), and userinit().

4.37.2.15 fs

```
ushort trapframe::fs
```

Definition at line 164 of file x86.h.

4.37.2.16 gs

```
ushort trapframe::gs
```

Definition at line 162 of file x86.h.

4.37.2.17 oesp

```
uint trapframe::oesp
```

Definition at line 155 of file x86.h.

4.37.2.18 padding1

ushort trapframe::padding1

Definition at line 163 of file x86.h.

4.37 trapframe 79

4.37.2.19 padding2

```
ushort trapframe::padding2
```

Definition at line 165 of file x86.h.

4.37.2.20 padding3

```
ushort trapframe::padding3
```

Definition at line 167 of file x86.h.

4.37.2.21 padding4

ushort trapframe::padding4

Definition at line 169 of file x86.h.

4.37.2.22 padding5

ushort trapframe::padding5

Definition at line 176 of file x86.h.

4.37.2.23 padding6

ushort trapframe::padding6

Definition at line 182 of file x86.h.

4.37.2.24 ss

ushort trapframe::ss

Definition at line 181 of file x86.h.

Referenced by userinit().

80 Class Documentation

4.37.2.25 trapno

```
uint trapframe::trapno
```

Definition at line 170 of file x86.h.

Referenced by trap().

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

• x86.h

4.38 uproc

```
#include <uproc.h>
```

Public Attributes

- char name [16]
- int pid
- int ppid

4.38.1 Detailed Description

Definition at line 1 of file uproc.h.

4.38.2 Member Data Documentation

4.38.2.1 name

```
char uproc::name[16]
```

Definition at line 4 of file uproc.h.

4.38.2.2 pid

```
int uproc::pid
```

Definition at line 2 of file uproc.h.

Referenced by main(), and sys_getprocs().

4.38.2.3 ppid

```
int uproc::ppid
```

Definition at line 3 of file uproc.h.

Referenced by main(), and sys_getprocs().

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

• uproc.h

Chapter 5

File Documentation

5.1 asm.h File Reference

Macros

- #define SEG_ASM(type, base, lim)
- #define SEG_NULLASM
- #define STA_R 0x2
- #define STA_W 0x2
- #define STA_X 0x8

5.1.1 Macro Definition Documentation

5.1.1.1 SEG_ASM

5.1.1.2 SEG_NULLASM

Definition at line 11 of file asm.h.

```
#define SEG_NULLASM

Value:
    .word 0, 0;
    .byte 0, 0, 0, 0
```

Definition at line 5 of file asm.h.

5.1.1.3 STA_R

```
#define STA_R 0x2
```

Definition at line 18 of file asm.h.

5.1.1.4 STA_W

```
#define STA_W 0x2
```

Definition at line 17 of file asm.h.

5.1.1.5 STA X

```
#define STA_X 0x8
```

Definition at line 16 of file asm.h.

5.2 asm.h

Go to the documentation of this file.

```
00001 //
00002 // assembler macros to create x86 segments
00003 //
00005 #define SEG_NULLASM
              .word 0, 0;
.byte 0, 0, 0, 0
00006
00007
80000
00009 // The 0xC0 means the limit is in 4096-byte units
00010 // and (for executable segments) 32-bit mode.
00011 #define SEG_ASM(type, base, lim)
                .word (((lim) » 12) & Oxffff), ((base) & Oxffff); \
.byte (((base) » 16) & Oxff), (0x90 | (type)), \
(0xC0 | (((lim) » 28) & Oxf)), (((base) » 24) & Oxff)
00012
00013
00014
00015
00016 #define STA_X
                                       // Executable segment
00017 #define STA_W
                            0x2
                                        // Writeable (non-executable segments)
                         0x2
00018 #define STA_R
                                       // Readable (executable segments)
```

5.3 bio.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "fs.h"
#include "buf.h"
```

5.3 bio.c File Reference 83

Functions

- static struct buf * bget (uint dev, uint blockno)
- void binit (void)
- struct buf * bread (uint dev, uint blockno)
- void brelse (struct buf *b)
- void bwrite (struct buf *b)

Variables

```
    struct {
        struct buf buf [NBUF]
        struct buf head
        struct spinlock lock
    } bcache
```

5.3.1 Function Documentation

5.3.1.1 bget()

```
static struct buf * bget (
                uint dev.
                uint blockno ) [static]
Definition at line 62 of file bio.c.
00063
00064
         struct buf *b;
00065
00066
        acquire(&bcache.lock);
00067
        // Is the block already cached?
00068
        for(b = bcache.head.next; b != &bcache.head; b = b->next) {
  if(b->dev == dev && b->blockno == blockno) {
00069
00070
00071
             b->refcnt++;
00072
             release(&bcache.lock);
00073
             acquiresleep(&b->lock);
00074
             return b;
00075
00076
        }
00078
        // Not cached; recycle an unused buffer.
00079
         // Even if refcnt==0, B_DIRTY indicates a buffer is in use
08000
         \ensuremath{//} because log.c has modified it but not yet committed it.
        for(b = bcache.head.prev; b != &bcache.head; b = b->prev){
  if(b->refcnt == 0 && (b->flags & B_DIRTY) == 0) {
00081
00082
             b->dev = dev;
00083
00084
             b->blockno = blockno;
00085
             b \rightarrow flags = 0;
             b->refcnt = 1;
00086
00087
             release(&bcache.lock);
00088
             acquiresleep(&b->lock);
00089
             return b:
00090
00091
00092
        panic("bget: no buffers");
00093 }
```

Referenced by bread().

5.3.1.2 binit()

```
void binit (
                 void )
Definition at line 39 of file bio.c.
00040 {
00041
         struct buf *b;
00042
00043
        initlock(&bcache.lock, "bcache");
00044
00045 //PAGEBREAK!
        // Create linked list of buffers
00046
00047
        bcache.head.prev = &bcache.head;
bcache.head.next = &bcache.head;
00048
        for(b = bcache.buf; b < bcache.buf+NBUF; b++) {</pre>
         b->next = bcache.head.next;
00050
          b->prev = &bcache.head;
00051
         initsleeplock(&b->lock, "buffer");
bcache.head.next->prev = b;
00052
00053
00054
          bcache.head.next = b;
```

5.3.1.3 bread()

00055

Definition at line 97 of file bio.c.

```
00098 {
00099    struct buf *b;
00100
00101    b = bget(dev, blockno);
00102    if((b->flags & B_VALID) == 0) {
010103        iderw(b);
00104    }
00105    return b;
00106 }
```

Referenced by balloc(), bfree(), bmap(), bzero(), ialloc(), ilock(), install_trans(), itrunc(), iupdate(), read_head(), readi(), readsb(), write_head(), write_log(), and writei().

5.3.1.4 brelse()

```
void brelse ( {\tt struct\ buf\ *\ b\ )}
```

Definition at line 121 of file bio.c.

```
00122 {
         if(!holdingsleep(&b->lock))
00124
          panic("brelse");
00125
00126
        releasesleep(&b->lock);
00127
00128
         acquire(&bcache.lock);
00129
         b->refcnt--;
        if (b->refcnt == 0) {
   // no one is waiting for it.
00130
00131
00132
          b->next->prev = b->prev;
b->prev->next = b->next;
00133
00134
         b->next = bcache.head.next;
00135
          b->prev = &bcache.head;
```

5.3 bio.c File Reference 85

Referenced by balloc(), bfree(), bmap(), bzero(), ialloc(), ilock(), install_trans(), itrunc(), iupdate(), read_head(), readi(), readsb(), write_head(), write_log(), and writei().

5.3.1.5 bwrite()

```
void bwrite ( \mathsf{struct}\ \mathsf{buf}\ *\ b\ )
```

Definition at line 110 of file bio.c.

```
00111 {
00112     if(!holdingsleep(&b->lock))
00113     panic("bwrite");
00114     b->flags |= B_DIRTY;
00115     iderw(b);
00116 }
```

Referenced by install_trans(), write_head(), and write_log().

5.3.2 Variable Documentation

5.3.2.1

```
struct { ... } bcache
```

Referenced by bget(), binit(), and brelse().

5.3.2.2 buf

```
struct buf buf[NBUF]
```

Definition at line 31 of file bio.c.

5.3.2.3 head

```
struct buf head
```

Definition at line 35 of file bio.c.

5.3.2.4 lock

```
struct spinlock lock
```

Definition at line 30 of file bio.c.

Referenced by holding().

5.4 bio.c

```
00001 // Buffer cache.
00002 //
00003 // The buffer cache is a linked list of buf structures holding
00004 // cached copies of disk block contents. Caching disk blocks
00005 // in memory reduces the number of disk reads and also provides
00006 \text{ // a synchronization point for disk blocks used by multiple processes.}
00007 //
00008 // Interface:
00009 // * To get a buffer for a particular disk block, call bread.
00010 // * After changing buffer data, call bwrite to write it to disk.
00011 // * When done with the buffer, call brelse.
00012 // \star Do not use the buffer after calling brelse.
00013 // * Only one process at a time can use a buffer,
00014 //
             so do not keep them longer than necessary.
00015 //
00016 // The implementation uses two state flags internally:
00017 // * B_VALID: the buffer data has been read from the disk.
00018 // \star B_DIRTY: the buffer data has been modified
00019 //
             and needs to be written to disk.
00020
00021 #include "types.h"
00022 #include "defs.h"
00023 #include "param.h"
00024 #include "spinlock.h"
00025 #include "sleeplock.h"
00026 #include "fs.h
00027 #include "buf.h"
00028
00029 struct {
00030 struct spinlock lock;
00031
       struct buf buf[NBUF];
00032
00034 // head.next is most recently used.
struct buf head.
       // Linked list of all buffers, through prev/next.
00036 } bcache;
00037
00038 void
00039 binit (void)
00040 {
00041
       struct buf *b;
00042
00043 initlock(&bcache.lock, "bcache");
00044
00045 //PAGEBREAK!
00046 // Create linked list of buffers
        bcache.head.prev = &bcache.head;
bcache.head.next = &bcache.head;
00047
00048
00049
        for(b = bcache.buf; b < bcache.buf+NBUF; b++) {</pre>
        b->next = bcache.head.next;
b->prev = &bcache.head;
00050
00051
          initsleeplock(&b->lock, "buffer");
00052
          bcache.head.next->prev = b;
00053
00054
          bcache.head.next = b;
00055 }
00056 }
00057
00058 // Look through buffer cache for block on device dev.
00059 // If not found, allocate a buffer.
00060 // In either case, return locked buffer.
00061 static struct buf*
00062 bget (uint dev, uint blockno)
00063 {
00064
        struct buf *b:
00065
00066
       acquire(&bcache.lock);
```

5.4 bio.c 87

```
00068
        // Is the block already cached?
00069
        for(b = bcache.head.next; b != &bcache.head; b = b->next) {
00070
         if(b->dev == dev && b->blockno == blockno) {
            b->refcnt++;
00071
00072
            release (&bcache.lock);
00073
            acquiresleep(&b->lock);
00074
            return b;
00075
00076
        }
00077
00078
        // Not cached; recycle an unused buffer.
        // Even if refcnt==0, B_DIRTY indicates a buffer is in use
00079
08000
        // because log.c has modified it but not yet committed it.
00081
        for(b = bcache.head.prev; b != &bcache.head; b = b->prev) {
00082
         if(b->refcnt == 0 && (b->flags & B_DIRTY) == 0) {
00083
            b->dev = dev;
00084
            b->blockno = blockno;
            b \rightarrow flags = 0;
00085
00086
            b->refcnt = 1;
00087
            release(&bcache.lock);
00088
            acquiresleep(&b->lock);
00089
            return b;
00090
00091
00092
        panic("bget: no buffers");
00093 }
00094
00095 // Return a locked buf with the contents of the indicated block.
00096 struct buf*
00097 bread(uint dev, uint blockno)
00098 {
00099
       struct buf *b;
00100
       b = bget(dev, blockno);
if((b->flags & B_VALID) == 0) {
00101
       iderw(b);
00102
00103
00105
        return b;
00106 }
00107
00108 // Write b's contents to disk. Must be locked.
00109 void
00110 bwrite(struct buf *b)
00111 {
00112
        if(!holdingsleep(&b->lock))
00113
         panic("bwrite");
00114
       b->flags |= B_DIRTY;
       iderw(b);
00115
00116 }
00117
00118 // Release a locked buffer.
00119 // Move to the head of the MRU list.
00120 void
00121 brelse(struct buf *b)
00122 {
00123
       if(!holdingsleep(&b->lock))
00124
        panic("brelse");
00125
00126
       releasesleep(&b->lock);
00127
        acquire(&bcache.lock);
00128
00129
        b->refcnt--;
00130
        if (b->refcnt == 0) {
         // no one is waiting for it.
00131
         b->next->prev = b->prev;
b->prev->next = b->next;
00132
00133
00134
          b->next = bcache.head.next;
          b->prev = &bcache.head;
00135
00136
          bcache.head.next->prev = b;
00137
         bcache.head.next = b;
00138
00139
00140
        release (&bcache.lock);
00141 }
00142 //PAGEBREAK!
00143 // Blank page.
00144
```

5.5 bio.d File Reference

5.6 bio.d

```
Go to the documentation of this file.
00001 bio.o: bio.c /usr/include/stdc-predef.h types.h defs.h param.h spinlock.h \
00002 sleeplock.h fs.h buf.h
```

5.7 bootasm.d File Reference

5.8 bootasm.d

```
Go to the documentation of this file.
00001 bootasm.o: bootasm.S asm.h memlayout.h mmu.h
```

5.9 bootmain.c File Reference

```
#include "types.h"
#include "elf.h"
#include "x86.h"
#include "memlayout.h"
```

Macros

• #define SECTSIZE 512

Functions

- void bootmain (void)
- void readsect (void *dst, uint offset)
- void readseg (uchar *, uint, uint)
- void waitdisk (void)

5.9.1 Macro Definition Documentation

5.9.1.1 **SECTSIZE**

```
#define SECTSIZE 512
```

Definition at line 13 of file bootmain.c.

5.9.2 Function Documentation

5.9.2.1 bootmain()

```
void bootmain (
     void )
```

Definition at line 18 of file bootmain.c.

```
00020
        struct elfhdr *elf;
        struct proghdr *ph, *eph;
00021
00022
        void (*entry)(void);
00023
        uchar* pa;
00024
00025
        elf = (struct elfhdr*)0x10000; // scratch space
00026
00027
        // Read 1st page off disk
00028
        readseg((uchar*)elf, 4096, 0);
00029
00030
        // Is this an ELF executable?
00031
        if (elf->magic != ELF_MAGIC)
00032
          return; // let bootasm.S handle error
00033
00034
        // Load each program segment (ignores ph flags).
        ph = (struct proghdr*)((uchar*)elf + elf->phoff);
eph = ph + elf->phnum;
00035
00036
        for(; ph < eph; ph++) {
  pa = (uchar*)ph->paddr;
00037
00038
00039
          readseg(pa, ph->filesz, ph->off);
         if(ph->memsz > ph->filesz)
stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
00040
00041
00042
00043
00044
        // Call the entry point from the ELF header.
00045
        // Does not return!
00046
       entry = (void(*)(void))(elf->entry);
       entry();
00047
00048 }
```

5.9.2.2 readsect()

Definition at line 60 of file bootmain.c.

```
00062
          // Issue command.
00063
          waitdisk();
         outb(0x1F2, 1);
00064
                                // count = 1
         outb(0x1F3, offset);
outb(0x1F4, offset » 8);
outb(0x1F5, offset » 16);
00065
00066
         outb(0x1F6, (offset » 24) | 0xE0);
outb(0x1F7, 0x20); // cmd 0x20 - read sectors
00068
00069
00070
00071
         // Read data.
00072
         waitdisk();
00073
         insl(0x1F0, dst, SECTSIZE/4);
00074 }
```

Referenced by readseg().

5.9.2.3 readseg()

```
void readseg (
              uchar * pa,
              uint count,
              uint offset )
Definition at line 79 of file bootmain.c.
00081
        uchar* epa;
00082
00083
        epa = pa + count;
00084
00085
       // Round down to sector boundary.
00086
       pa -= offset % SECTSIZE;
00087
88000
       // Translate from bytes to sectors; kernel starts at sector 1.
00089
       offset = (offset / SECTSIZE) + 1;
00090
00091
       // If this is too slow, we could read lots of sectors at a time.
       // We'd write more to memory than asked, but it doesn't matter --
00092
00093
        // we load in increasing order.
00094
        for(; pa < epa; pa += SECTSIZE, offset++)</pre>
00095
          readsect(pa, offset);
00096 }
```

Referenced by bootmain().

5.9.2.4 waitdisk()

```
void waitdisk (
     void )
```

Definition at line 51 of file bootmain.c.

Referenced by readsect().

5.10 bootmain.c

```
00001 // Boot loader.
00003 // Part of the boot block, along with bootasm.S, which calls bootmain().
00004 \ensuremath{//} bootasm.S has put the processor into protected 32-bit mode.
00005 // bootmain() loads an ELF kernel image from the disk starting at
00006 // sector 1 and then jumps to the kernel entry routine.
00007
00008 #include "types.h"
00000 #include cypes.n
00000 #include "elf.h"
00010 #include "x86.h"
00011 #include "memlayout.h"
00012
00013 #define SECTSIZE 512
00014
00015 void readseg(uchar*, uint, uint);
00016
00017 void
00018 bootmain(void)
00019 {
00020 struct elfhdr *elf;
00021 struct proghdr *ph, *eph;
```

```
void (*entry)(void);
00023
        uchar* pa;
00024
        elf = (struct elfhdr*)0x10000; // scratch space
00025
00026
00027
        // Read 1st page off disk
        readseg((uchar*)elf, 4096, 0);
00029
00030
        // Is this an ELF executable?
        if(elf->magic != ELF_MAGIC)
  return; // let bootasm.S handle error
00031
00032
00033
00034
        // Load each program segment (ignores ph flags).
00035
        ph = (struct proghdr*)((uchar*)elf + elf->phoff);
00036
        eph = ph + elf->phnum;
00037
        for(; ph < eph; ph++) {</pre>
00038
         pa = (uchar*)ph->paddr;
         readseg(pa, ph->filesz, ph->off);
if(ph->memsz > ph->filesz)
00039
00040
00041
            stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
00042
00043
00044
        \ensuremath{//} Call the entry point from the ELF header.
        // Does not return!
00045
00046
        entry = (void(*)(void))(elf->entry);
00047
        entry();
00048 }
00049
00050 void
00051 waitdisk(void)
00052 {
00053
        // Wait for disk ready.
00054
        while ((inb(0x1F7) & 0xC0) != 0x40)
00055
00056 }
00057
00058 // Read a single sector at offset into dst.
00060 readsect(void *dst, uint offset)
00061 {
00062
        // Issue command.
00063
        waitdisk();
                           // count = 1
00064
        outb(0x1F2, 1);
00065
        outb(0x1F3, offset);
00066
        outb(0x1F4, offset » 8);
00067
        outb(0x1F5, offset » 16);
       outb(0x1F6, (offset » 24) | 0xE0);
outb(0x1F7, 0x20); // cmd 0x20 - read sectors
00068
00069
00070
00071
        // Read data.
00072
        waitdisk();
00073 insl(0x1F0, dst, SECTSIZE/4);
00074 }
00075
00076 // Read 'count' bytes at 'offset' from kernel into physical address 'pa'.
00077 // Might copy more than asked.
00079 readseg(uchar* pa, uint count, uint offset)
00080 {
00081
        uchar* epa;
00082
00083
        epa = pa + count;
00084
00085
        // Round down to sector boundary.
00086
        pa -= offset % SECTSIZE;
00087
00088
        // Translate from bytes to sectors; kernel starts at sector 1.
offset = (offset / SECTSIZE) + 1;
00089
00090
           If this is too slow, we could read lots of sectors at a time.
00092
        // We'd write more to memory than asked, but it doesn't matter --
00093
        // we load in increasing order.
00094
        for(; pa < epa; pa += SECTSIZE, offset++)</pre>
00095
          readsect(pa, offset);
00096 }
```

5.11 bootmain.d File Reference

5.12 bootmain.d

```
00001 bootmain.o: bootmain.c types.h elf.h x86.h memlayout.h
```

5.13 buf.h File Reference

Classes

struct buf

Macros

- #define B_DIRTY 0x4
- #define B_VALID 0x2

5.13.1 Macro Definition Documentation

5.13.1.1 B DIRTY

```
#define B_DIRTY 0x4
```

Definition at line 13 of file buf.h.

5.13.1.2 B_VALID

```
#define B_VALID 0x2
```

Definition at line 12 of file buf.h.

5.14 buf.h

```
00001 struct buf {
00002 int flags;
00003 uint dev;
00004 uint blockno;
00005 struct sleeplock lock;
00006 uint refcnt;
00007 struct buf *prev; // LRU cache list
00008 struct buf *next;
00009 struct buf *qnext; // disk queue
00010 uchar data[BSIZE];
00011 };
00012 #define B_VALID 0x2 // buffer has been read from disk
00013 #define B_DIRTY 0x4 // buffer needs to be written to disk
00014
```

5.15 cat.c File Reference 93

5.15 cat.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

```
    void cat (int fd)
```

• int main (int argc, char *argv[])

Variables

• char buf [512]

5.15.1 Function Documentation

5.15.1.1 cat()

```
void cat ( \quad \text{int } fd \ )
```

Definition at line 8 of file cat.c.

Referenced by main().

5.15.1.2 main()

```
int main (
                   int argc,
                   char * argv[] )
Definition at line 25 of file cat.c.
00026 {
          int fd, i;
00027
00028
00029
          if (argc <= 1) {</pre>
           cat(0);
exit();
00030
00031
00032
00033
         for(i = 1; i < argc; i++) {
  if((fd = open(argv[i], 0)) < 0) {
    printf(1, "cat: cannot open %s\n", argv[i]);
    exit():</pre>
00034
00035
00036
00037
               exit();
00038
00039
            cat (fd);
           close(fd);
00040
00041 }
00042 exit();
00043 }
```

5.15.2 Variable Documentation

5.15.2.1 buf

```
char buf[512]
```

Definition at line 5 of file cat.c.

5.16 cat.c

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004
00005 char buf[512];
00006
00007 void
00008 cat(int fd)
00009 {
00010
00011
           while((n = read(fd, buf, sizeof(buf))) > 0) {
   if (write(1, buf, n) != n) {
      printf(1, "cat: write error\n");
      exit();
00012
00013
00014
00015
            }
00016
          }
if (n < 0) {
    printf(1, "cat: read error\n");
    evit():</pre>
00017
00018
00019
00020
              exit();
00021
00022 }
00023
00024 int
00025 main(int argc, char *argv[])
00026 {
00027
           int fd, i;
00028
```

5.17 cat.d File Reference 95

```
if (argc <= 1) {</pre>
         cat(0);
00030
00031
          exit();
00032 }
00033
        for(i = 1; i < argc; i++) {
  if((fd = open(argv[i], 0)) < 0) {</pre>
00034
00036
           printf(1, "cat: cannot open %s\n", argv[i]);
00037
00038
          cat (fd);
00039
00040
          close (fd);
00041
00042 exit();
00043 }
```

5.17 cat.d File Reference

5.18 cat.d

```
Go to the documentation of this file.
00001 cat.o: cat.c /usr/include/stdc-predef.h types.h stat.h user.h
```

5.19 console.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "traps.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "fs.h"
#include "file.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
```

Macros

- #define BACKSPACE 0x100
- #define C(x) ((x)-'@')
- #define CRTPORT 0x3d4
- #define INPUT_BUF 128

Functions

- static void cgaputc (int c)
- void consoleinit (void)
- void consoleintr (int(*getc)(void))
- int consoleread (struct inode *ip, char *dst, int n)
- int consolewrite (struct inode *ip, char *buf, int n)
- static void consputc (int)
- void cprintf (char *fmt,...)
- void panic (char *s)
- static void printint (int xx, int base, int sign)

Variables

```
    struct {
        struct spinlock lock
        int locking
    } cons
    static ushort * crt = (ushort*)P2V(0xb8000)
    struct {
        char buf [INPUT_BUF]
        uint e
        uint r
        uint w
    } input
    static int panicked = 0
```

5.19.1 Macro Definition Documentation

5.19.1.1 BACKSPACE

```
#define BACKSPACE 0x100
```

Definition at line 127 of file console.c.

5.19.1.2 C

```
#define C( x ) ((x)-'@')
```

Definition at line 189 of file console.c.

5.19.1.3 CRTPORT

```
#define CRTPORT 0x3d4
```

Definition at line 128 of file console.c.

5.19.1.4 INPUT_BUF

```
#define INPUT_BUF 128
```

Definition at line 181 of file console.c.

5.19.2 Function Documentation

5.19.2.1 cgaputc()

```
static void cgaputc (
               int c) [static]
Definition at line 132 of file console.c.
00134
        int pos;
00135
       // Cursor position: col + 80*row.
outb(CRTPORT, 14);
pos = inb(CRTPORT+1) « 8;
00136
00137
00138
00139
        outb (CRTPORT, 15);
00140
        pos |= inb(CRTPORT+1);
00141
        if(c == '\n')
pos += 80 - pos%80;
00142
00143
00144
        else if(c == BACKSPACE){
00145
          if(pos > 0) --pos;
00146
00147
          crt[pos++] = (c&0xff) | 0x0700; // black on white
00148
00149
        if(pos < 0 || pos > 25*80)
         panic("pos under/overflow");
00150
00151
00152
        if((pos/80) >= 24){
                               // Scroll up.
00153
        memmove(crt, crt+80, sizeof(crt[0]) *23*80);
00154
         pos -= 80;
00155
          memset(crt+pos, 0, sizeof(crt[0])*(24*80 - pos));
00156
00157
```

Referenced by consputc().

outb(CRTPORT, 14);

outb(CRTPORT+1, pos>8);
outb(CRTPORT, 15);

outb(CRTPORT+1, pos); crt[pos] = ' ' | 0x0700;

00158

00159

00160

00161 00162 00163 }

5.19.2.2 consoleinit()

```
void consoleinit (
     void )
```

Definition at line 289 of file console.c.

```
00290 {
00291     initlock(&cons.lock, "console");
00292
00293     devsw[CONSOLE].write = consolewrite;
00294     devsw[CONSOLE].read = consoleread;
00295     cons.locking = 1;
00296
00297     ioapicenable(IRQ_KBD, 0);
00298 }
```

5.19.2.3 consoleintr()

```
void consoleintr (
               int(*)(void) getc)
Definition at line 192 of file console.c.
00193 {
00194
        int c. doprocdump = 0:
00195
00196
        acquire(&cons.lock);
00197
        while((c = getc()) >= 0){
00198
          switch(c){
00199
          case C('P'):
                         // Process listing.
            // procdump() locks cons.lock indirectly; invoke later
00200
00201
             doprocdump = 1;
00202
            break;
00203
          case C('U'): // Kill line.
00204
            while(input.e != input.w &&
                   input.buf[(input.e-1) % INPUT_BUF] != '\n'){
00205
00206
               input.e--;
              consputc (BACKSPACE);
00207
00208
00209
            break;
00210
          case C('H'): case '\x7f': // Backspace
00211
            if(input.e != input.w){
00212
              input.e--;
               conspute (BACKSPACE);
00213
00214
00215
            break;
00216
           if(c != 0 && input.e-input.r < INPUT_BUF) {
    c = (c == '\r') ? '\n' : c;</pre>
00217
00218
               input.buf[input.e++ % INPUT_BUF] = c;
00219
00220
              consputc(c);
if(c == '\n' || c == C('D') || input.e == input.r+INPUT_BUF) {
  input.w = input.e;
00221
00222
00223
                 wakeup(&input.r);
00224
              }
00225
00226
            break:
00227
          }
00228
00229
        release(&cons.lock);
        if (doprocdump) {
00230
00231
          procdump(); // now call procdump() wo. cons.lock held
00232
00233 }
```

Referenced by kbdintr(), and uartintr().

5.19.2.4 consoleread()

Definition at line 236 of file console.c.

```
00237 {
00238
        uint target;
00239
        int c;
00240
00241
        iunlock(ip);
00242
       target = n;
00243
        acquire(&cons.lock);
00244
        while (n > 0) {
00245
         while(input.r == input.w) {
00246
            if (myproc() ->killed) {
00247
             release (&cons.lock);
00248
              ilock(ip);
00249
              return -1:
00250
00251
            sleep(&input.r, &cons.lock);
```

```
00253
            c = input.buf[input.r++ % INPUT_BUF];
            if(c == C('D')){ // EOF

if(n < target){
    // Save ^D for next time, to make sure
    // caller gets a 0-byte result.</pre>
00254
00255
00256
00257
00258
                 input.r--;
00259
00260
              break;
00261
            *dst++ = c;
00262
00263
            --n;
00264
            if(c == '\n')
00265
              break;
00266
00267
         release(&cons.lock);
00268
         ilock(ip);
00269
         return target - n;
00271 }
```

Referenced by consoleinit().

5.19.2.5 consolewrite()

Definition at line 274 of file console.c.

```
00275 {
00276
00277
00278
        iunlock(ip);
00279
        acquire(&cons.lock);
        for(i = 0; i < n; i++)
  consputc(buf[i] & 0xff);</pre>
00280
00281
00282
        release(&cons.lock);
        ilock(ip);
00284
00285
        return n;
00286 }
```

Referenced by consoleinit().

5.19.2.6 consputc()

```
void conspute ( \quad \quad \text{int } c \text{ ) } \quad [\text{static}]
```

Definition at line 166 of file console.c.

```
00167 {
00168
        if (panicked) {
        cli();
00169
00170
          for(;;)
00171
00172
00173
00174
        if (c == BACKSPACE) {
00175
         uartputc('\b'); uartputc(' '); uartputc('\b');
00176
00177
         uartputc(c);
00178
       cgaputc(c);
00179 }
```

Referenced by consoleintr(), consolewrite(), cprintf(), and printint().

5.19.2.7 cprintf()

Definition at line 55 of file console.c.

```
00056
00057
         int i, c, locking;
00058
        uint *argp;
00059
        char *s;
00060
        locking = cons.locking;
00061
00062
        if (locking)
00063
          acquire(&cons.lock);
00064
00065
        if (fmt == 0)
          panic("null fmt");
00066
00067
        argp = (uint*) (void*) (&fmt + 1);
for(i = 0; (c = fmt[i] & 0xff) != 0; i++) {
  if(c != '%') {
00068
00069
00070
00071
            consputc(c);
00072
            continue;
00073
00074
          c = fmt[++i] & 0xff;
00075
          if(c == 0)
00076
            break;
00077
           switch(c){
00078
          case 'd':
           printint(*argp++, 10, 1);
00079
08000
          break; case 'x':
00081
          case 'p':
00082
          printint(*argp++, 16, 0);
break;
case 's':
00083
00084
00085
           if((s = (char*)*argp++) == 0)
    s = "(null)";
00086
00087
00088
            for(; *s; s++)
00089
              consputc(*s);
00090
            break;
          case '%':
00091
00092
           consputc('%');
00093
            break;
00094
           default:
00095
            // Print unknown % sequence to draw attention.
00096
             consputc('%');
00097
             consputc(c);
00098
            break;
00099
00100
        }
00101
00102
        if (locking)
00103
          release(&cons.lock);
00104 }
```

Referenced by allocuvm(), cps(), exec(), iinit(), ioapicinit(), panic(), procdump(), syscall(), and trap().

5.19.2.8 panic()

Definition at line 107 of file console.c.

```
00108 {
00109    int i;
00110    uint pcs[10];
00111
00112    cli();
00113    cons.locking = 0;
00114    // use lapiccpunum so that we can call panic from mycpu()
00115    cprintf("lapicid %d: panic: ", lapicid());
```

```
00116    cprintf(s);
00117    cprintf("\n");
00118    getcallerpcs(&s, pcs);
00119    for(i=0; i<10; i++)
00120    cprintf(" %p", pcs[i]);
00121    panicked = 1; // freeze other CPU
00122    for(;;)
00123    ;
00124 }</pre>
```

Referenced by acquire(), balloc(), bfree(), bget(), bmap(), brelse(), bwrite(), cgaputc(), clearpteu(), copyuvm(), cprintf(), create(), deallocuvm(), dirlink(), dirlookup(), end_op(), exit(), fileclose(), filedup(), fileread(), filewrite(), freevm(), ialloc(), iderw(), idestart(), iget(), ilock(), initlog(), inituvm(), isdirempty(), iunlock(), kfree(), loaduvm(), log_write(), mappages(), mpinit(), mycpu(), popcli(), release(), sched(), setupkvm(), sleep(), switchuvm(), sys_unlink(), trap(), and userinit().

5.19.2.9 printint()

Definition at line 28 of file console.c.

```
00029 {
        static char digits[] = "0123456789abcdef";
00030
00031
        char buf[16];
00032
        int i;
00033
00034
00035
        if(sign \&\& (sign = xx < 0))
00036
        x = -xx; else
00037
00038
          x = xx;
00039
00040
00041
        buf[i++] = digits[x % base];
}while((x /= base) != 0);
00042
00043
00044
00045
        if(sign)
00046
          buf[i++] = '-';
00047
        while (--i >= 0)
00048
           consputc(buf[i]);
00049
00050 }
```

Referenced by cprintf().

5.19.3 Variable Documentation

5.19.3.1 buf

```
char buf[INPUT_BUF]
```

Definition at line 183 of file console.c.

5.19.3.2

```
struct { ... } cons [static]
```

Referenced by consoleinit(), consoleread(), consolewrite(), cprintf(), and panic().

5.19.3.3 crt

```
ushort* crt = (ushort*)P2V(0xb8000) [static]
```

Definition at line 129 of file console.c.

Referenced by cgaputc().

5.19.3.4 e

uint e

Definition at line 186 of file console.c.

Referenced by mpinit(), and mpsearch1().

5.19.3.5

```
struct { ... } input
```

Referenced by consoleintr(), and consoleread().

5.19.3.6 lock

```
struct spinlock lock
```

Definition at line 23 of file console.c.

5.19.3.7 locking

int locking

Definition at line 24 of file console.c.

Referenced by cprintf().

5.20 console.c 103

5.19.3.8 panicked

```
int panicked = 0 [static]
```

Definition at line 20 of file console.c.

Referenced by consputc(), and panic().

5.19.3.9 r

```
uint r
```

Definition at line 184 of file console.c.

Referenced by cmostime(), fileread(), filewrite(), fill_rtcdate(), holding(), holdingsleep(), idewait(), iput(), kalloc(), kfree(), and stat().

5.19.3.10 w

```
uint w
```

Definition at line 185 of file console.c.

Referenced by wc().

5.20 console.c

Go to the documentation of this file.

```
00001 // Console input and output. 00002 // Input is from the keyboard or serial port. 00003 // Output is written to the screen and serial port.
00004
00005 #include "types.h"
00006 #include "defs.h"
00007 #include "param.h"
00008 #include "traps.h"
00009 #include "spinlock.h"
00010 #include "sleeplock.h"
00011 #include "fs.h"
00012 #include "file.h"
00013 #include "memlayout.h"
00014 #include "mmu.h"
00015 #include "proc.h"
00016 #include "x86.h"
00017
00018 static void consputc(int);
00019
00020 static int panicked = 0;
00021
00022 static struct {
            struct spinlock lock;
00024 int locking;
00025 } cons;
00026
00027 static void
00028 printint(int xx, int base, int sign)
             static char digits[] = "0123456789abcdef";
```

```
00031
        char buf[16];
00032
00033
        uint x;
00034
        if(sign && (sign = xx < 0))
00035
        x = -xx; else
00036
00038
          x = xx;
00039
        i = 0;
00040
00041
        do{
         buf[i++] = digits[x % base];
00042
00043
        }while((x /= base) != 0);
00044
00045
        if(sign)
         buf[i++] = '-';
00046
00047
00048
       while (--i >= 0)
00049
         consputc(buf[i]);
00050 }
00051 //PAGEBREAK: 50
00052
00053 // Print to the console. only understands %d, %x, %p, %s.
00054 void
00055 cprintf(char *fmt, ...)
00056 {
00057
        int i, c, locking;
        uint *argp;
char *s;
00058
00059
00060
00061
        locking = cons.locking;
00062
        if (locking)
00063
          acquire(&cons.lock);
00064
00065
        if (fmt == 0)
          panic("null fmt");
00066
00067
00068
        argp = (uint*)(void*)(&fmt + 1);
        for(i = 0; (c = fmt[i] & 0xff) != 0; i++) {
  if(c != '%') {
00069
00070
00071
            consputc(c);
00072
            continue;
00073
00074
          c = fmt[++i] & 0xff;
00075
          if(c == 0)
00076
            break;
          switch(c) {
case 'd':
00077
00078
          printint(*argp++, 10, 1);
00079
00080
            break:
00081
          case 'x':
00082
          case 'p':
          printint(*argp++, 16, 0);
break;
case 's':
00083
00084
00085
00086
          if((s = (char*)*argp++) == 0)
s = "(null)";
00087
00088
           for(; *s; s++)
00089
              consputc(*s);
          break;
case '%':
00090
00091
00092
           consputc('%');
00093
            break;
00094
00095
           // Print unknown % sequence to draw attention.
00096
            consputc('%');
00097
            consputc(c);
00098
            break:
00099
00100
        }
00101
00102
        if (locking)
00103
          release(&cons.lock);
00104 }
00105
00106 void
00107 panic(char *s)
00108 {
00109
        int i;
00110
        uint pcs[10];
00111
00112
        cli();
00113
        cons.locking = 0;
00114
        // use lapiccpunum so that we can call panic from mycpu()
        cprintf("lapicid %d: panic: ", lapicid());
00115
        cprintf(s);
cprintf("\n");
00116
00117
```

5.20 console.c 105

```
getcallerpcs(&s, pcs);
        for(i=0; i<10; i++)
    cprintf(" %p", pcs[i]);
panicked = 1; // freeze other CPU</pre>
00119
00120
00121
00122
        for(;;)
00123
         ;
00124 }
00125
00126 //PAGEBREAK: 50
00127 #define BACKSPACE 0x100
00128 #define CRTPORT 0x3d4
00129 static ushort *crt = (ushort*)P2V(0xb8000); // CGA memory
00130
00131 static void
00132 cgaputc(int c)
00133 {
00134
        int pos;
00135
00136
        // Cursor position: col + 80*row.
        outb(CRTPORT, 14);
pos = inb(CRTPORT+1) « 8;
00137
00138
00139
         outb (CRTPORT, 15);
        pos |= inb(CRTPORT+1);
00140
00141
00142
        if(c == '\n')
          pos += 80 - pos%80;
00143
        else if(c == BACKSPACE){
00144
00145
           if(pos > 0) --pos;
00146
        } els
00147
           crt[pos++] = (c&0xff) | 0x0700; // black on white
00148
00149
        if (pos < 0 || pos > 25*80)
00150
          panic("pos under/overflow");
00151
        if((pos/80) >= 24){ // Scroll up.
  memmove(crt, crt+80, sizeof(crt[0])*23*80);
  pos -= 80;
00152
00153
00154
00155
           memset(crt+pos, 0, sizeof(crt[0])*(24*80 - pos));
00156
00157
        outb(CRTPORT, 14);
outb(CRTPORT+1, pos>8);
00158
00159
        outb (CRTPORT, 15);
00160
        outb(CRTPORT+1, pos);
crt[pos] = ' ' | 0x0700;
00161
00162
00163 }
00164
00165 void
00166 consputc(int c)
00167 {
00168
        if (panicked) {
00169
         cli();
00170
           for(;;)
00171
00172
00173
        if (c == BACKSPACE) {
00175
          uartputc('\b'); uartputc(' '); uartputc('\b');
        } else
00176
00177
          uartputc(c);
00178
        cgaputc(c);
00179 }
00180
00181 #define INPUT_BUF 128
00182 struct {
00183 char buf[INPUT_BUF];
      uint r; // Read index
uint w; // Write index
uint e; // Edit index
00184
00185
00186
00187 } input;
00188
00189 #define C(x) ((x)-'0') // Control-x
00190
00191 void
00192 consoleintr(int (*getc)(void))
00193 {
00194
        int c, doprocdump = 0;
00195
00196
        acquire(&cons.lock);
00197
        while((c = getc()) >= 0){
00198
          switch(c){
           case C('P'): // Process listing.
00199
00200
             // procdump() locks cons.lock indirectly; invoke later
00201
             doprocdump = 1;
           break; case C('U'): // Kill line.
00202
00203
00204
             while (input.e != input.w &&
```

```
input.buf[(input.e-1) % INPUT_BUF] != '\n'){
               input.e--;
00206
00207
               consputc (BACKSPACE);
00208
00209
            break:
           case C('H'): case '\x7f': // Backspace
00210
00211
            if(input.e != input.w){
00212
               input.e--;
00213
               consputc(BACKSPACE);
00214
00215
            break:
00216
          default:
            if(c != 0 && input.e-input.r < INPUT_BUF){
    c = (c == '\r') ? '\n' : c;</pre>
00217
00218
               input.buf[input.e++ % INPUT_BUF] = c;
00219
               consputc(c);
if(c == '\n' || c == C('D') || input.e == input.r+INPUT_BUF){
  input.w = input.e;
00220
00221
00222
                 wakeup(&input.r);
00223
00224
               }
00225
00226
             break;
00227
          }
00228
00229
        release (&cons.lock);
00230
        if (doprocdump) {
00231
          procdump(); // now call procdump() wo. cons.lock held
00232
00233 }
00234
00235 int
00236 consoleread(struct inode *ip, char *dst, int n)
00237 {
00238
        uint target;
00239
        int c;
00240
00241
        iunlock(ip);
00242
        target = n;
00243
        acquire(&cons.lock);
00244
        while (n > 0) {
00245
          while(input.r == input.w){
            if (myproc() ->killed) {
00246
              release (&cons.lock);
00247
00248
               ilock(ip);
00249
               return -1;
00250
00251
             sleep(&input.r, &cons.lock);
00252
00253
           c = input.buf[input.r++ % INPUT BUF];
          if(c == C('D')) { // EOF
   if (n < target) {</pre>
00254
00255
              // Save ^D for next time, to make sure
// caller gets a 0-byte result.
00256
00257
00258
               input.r--;
00259
00260
            break;
00261
00262
          *dst++ = c;
00263
          --n;
          if(c == '\n')
00264
00265
            break;
00266
00267
        release (&cons.lock);
00268
        ilock(ip);
00269
00270
        return target - n;
00271 }
00272
00273 int
00274 consolewrite(struct inode *ip, char *buf, int n)
00275 {
00276
        int i;
00277
00278
        iunlock(ip);
00279
        acquire (&cons.lock);
        for(i = 0; i < n; i++)
00280
00281
          consputc(buf[i] & 0xff);
00282
        release(&cons.lock);
00283
        ilock(ip);
00284
00285
        return n;
00286 }
00287
00288 void
00289 consoleinit (void)
00290 {
00291
        initlock(&cons.lock, "console");
```

5.21 console.d File Reference

5.22 console.d

Go to the documentation of this file.

5.23 date.h File Reference

Classes

struct rtcdate

5.24 date.h

Go to the documentation of this file.

```
00001 struct rtcdate {
00002 uint second;
00003 uint minute;
00004 uint hour;
00005 uint day;
00006 uint month;
00007 uint year;
00008 };
```

5.25 defs.h File Reference

Macros

#define NELEM(x) (sizeof(x)/sizeof((x)[0]))

Functions

```
    void acquire (struct spinlock *)

    void acquiresleep (struct sleeplock *)

    int allocuvm (pde t*, uint, uint)

int argint (int, int *)
• int argptr (int, char **, int)
int argstr (int, char **)
void begin_op ()

    void binit (void)

struct buf * bread (uint, uint)

    void brelse (struct buf *)

    void bwrite (struct buf *)

• int chpr (int pid, int priority)

    void clearpteu (pde t *pgdir, char *uva)

    void cmostime (struct rtcdate *r)

    void consoleinit (void)

    void consoleintr (int(*)(void))

    int copyout (pde_t *, uint, void *, uint)

pde_t * copyuvm (pde_t *, uint)
void cprintf (char *,...)
• int cps (void)
• int cpuid (void)
• int deallocuvm (pde t*, uint, uint)

    int dirlink (struct inode *, char *, uint)

    struct inode * dirlookup (struct inode *, char *, uint *)

· void end_op ()
• int exec (char *, char **)

    void exit (void)

int fetchint (uint, int *)
int fetchstr (uint, char **)

    struct file * filealloc (void)

    void fileclose (struct file *)

    struct file * filedup (struct file *)

· void fileinit (void)
• int fileread (struct file *, char *, int n)

    int filestat (struct file *, struct stat *)

• int filewrite (struct file *, char *, int n)
• int fork (void)
void freevm (pde_t *)
void getcallerpcs (void *, uint *)

    int getprocs (int max, struct uproc *)

• int growproc (int)

    int holding (struct spinlock *)

• int holdingsleep (struct sleeplock *)
• struct inode * ialloc (uint, short)

    void ideinit (void)

· void ideintr (void)
void iderw (struct buf *)
• void idtinit (void)
struct inode * idup (struct inode *)
· void iinit (int dev)

    void ilock (struct inode *)

    void initlock (struct spinlock *, char *)
```

void initlog (int dev)

```
    void initsleeplock (struct sleeplock *, char *)

void inituvm (pde_t *, char *, uint)

    void ioapicenable (int irq, int cpu)

    void ioapicinit (void)

    void iput (struct inode *)

    void iunlock (struct inode *)

    void iunlockput (struct inode *)

    void iupdate (struct inode *)

    char * kalloc (void)

· void kbdintr (void)
void kfree (char *)

    int kill (int)

void kinit1 (void *, void *)

    void kinit2 (void *, void *)

    void kvmalloc (void)

    void lapiceoi (void)

    int lapicid (void)

    void lapicinit (void)

    void lapicstartap (uchar, uint)

    int loaduvm (pde_t *, char *, struct inode *, uint, uint)

    void log_write (struct buf *)

    int memcmp (const void *, const void *, uint)

    void * memmove (void *, const void *, uint)

void * memset (void *, int, uint)
· void microdelay (int)

    void mpinit (void)

    struct cpu * mycpu (void)

    struct proc * myproc ()

int namecmp (const char *, const char *)

    struct inode * namei (char *)

    struct inode * nameiparent (char *, char *)

    void panic (char *) attribute ((noreturn))

    void picenable (int)

    void picinit (void)

    void pinit (void)

    int pipealloc (struct file **, struct file **)

    void pipeclose (struct pipe *, int)

    int piperead (struct pipe *, char *, int)

    int pipewrite (struct pipe *, char *, int)

    void popcli (void)

    void procdump (void)

    void pushcli (void)

    int readi (struct inode *, char *, uint, uint)

    void readsb (int dev, struct superblock *sb)

    void release (struct spinlock *)

    void releasesleep (struct sleeplock *)

    char * safestrcpy (char *, const char *, int)

    void sched (void)

    void scheduler (void) __attribute__((noreturn))

· void seginit (void)

    void setproc (struct proc *)

    pde t * setupkvm (void)

    void sleep (void *, struct spinlock *)
```

void stati (struct inode *, struct stat *)

int strlen (const char *)

```
• int strncmp (const char *, const char *, uint)
```

- char * strncpy (char *, const char *, int)
- void switchkvm (void)
- void switchuvm (struct proc *)
- void swtch (struct context **, struct context *)
- void syscall (void)
- void timerinit (void)
- · void tvinit (void)
- void uartinit (void)
- void uartintr (void)
- · void uartputc (int)
- void userinit (void)
- char * uva2ka (pde_t *, char *)
- int wait (void)
- void wakeup (void *)
- int writei (struct inode *, char *, uint, uint)
- void yield (void)

Variables

- · uchar ioapicid
- int ismp
- volatile uint * lapic
- · uint ticks
- struct spinlock tickslock

5.25.1 Macro Definition Documentation

5.25.1.1 NELEM

Definition at line 195 of file defs.h.

5.25.2 Function Documentation

5.25.2.1 acquire()

```
void acquire (
               struct spinlock * lk)
Definition at line 25 of file spinlock.c.
        pushcli(); // disable interrupts to avoid deadlock.
00028
        if (holding(lk))
00029
          panic("acquire");
00030
00031
        // The xchg is atomic.
00032
       while(xchg(&lk->locked, 1) != 0)
00033
         ;
00034
00035
       \ensuremath{//} Tell the C compiler and the processor to not move loads or stores
00036
       // past this point, to ensure that the critical section's memory
00037
       // references happen after the lock is acquired.
00038
        __sync_synchronize();
00039
        // Record info about lock acquisition for debugging.
00041
       lk->cpu = mycpu();
00042
        getcallerpcs(&lk, lk->pcs);
00043 }
```

Referenced by acquiresleep(), allocproc(), begin_op(), bget(), brelse(), chpr(), consoleintr(), consoleread(), consolewrite(), cprintf(), cps(), end_op(), exit(), filealloc(), fileclose(), filedup(), fork(), holdingsleep(), ideintr(), iderw(), idup(), iget(), iput(), kalloc(), kfree(), kill(), log_write(), pipeclose(), piperead(), pipewrite(), releasesleep(), scheduler(), sleep(), sys_sleep(), sys_uptime(), trap(), userinit(), wait(), wakeup(), and yield().

5.25.2.2 acquiresleep()

```
void acquiresleep ( {\tt struct \ sleeplock \ * \it lk} \ )
```

Definition at line 23 of file sleeplock.c.

Referenced by bget(), ilock(), and iput().

5.25.2.3 allocuvm()

Definition at line 222 of file vm.c.

```
if(newsz < oldsz)</pre>
00230
         return oldsz;
00231
        a = PGROUNDUP(oldsz);
00232
        for(; a < newsz; a += PGSIZE) {
  mem = kalloc();
  if(mem == 0) {</pre>
00233
00234
00236
            cprintf("allocuvm out of memory\n");
00237
             deallocuvm(pgdir, newsz, oldsz);
00238
            return 0;
          }
00239
          memset(mem, 0, PGSIZE);
00240
          if (mappages (pgdir, (char*)a, PGSIZE, V2P (mem), PTE_W|PTE_U) < 0) {</pre>
00241
           cprintf("allocuvm out of memory (2)\n");
00242
00243
             deallocuvm(pgdir, newsz, oldsz);
00244
            kfree (mem);
00245
             return 0:
       }
00246
00248
        return newsz;
00249 }
```

Referenced by exec(), and growproc().

5.25.2.4 argint()

Definition at line 50 of file syscall.c.

```
00051 {
00052    return fetchint((myproc()->tf->esp) + 4 + 4*n, ip);
00053 }
```

Referenced by argfd(), argptr(), argstr(), sys_chpr(), sys_exec(), sys_getprocs(), sys_kill(), sys_mknod(), sys_open(), sys_read(), sys_sbrk(), sys_sleep(), and sys_write().

5.25.2.5 argptr()

Definition at line 59 of file syscall.c.

```
00061
00062
       struct proc *curproc = myproc();
00063
00064
       if(argint(n, \&i) < 0)
00065
         return -1;
00066
       if(size < 0 || (uint)i >= curproc->sz || (uint)i+size > curproc->sz)
00067
         return -1;
00068
       *pp = (char*)i;
00069
       return 0;
00070 }
```

Referenced by sys_fstat(), sys_getprocs(), sys_pipe(), sys_read(), and sys_write().

5.25.2.6 argstr()

Definition at line 77 of file syscall.c.

```
00078 {
00079    int addr;
00080    if(argint(n, &addr) < 0)
00081        return -1;
00082    return fetchstr(addr, pp);
00083 }</pre>
```

Referenced by sys_chdir(), sys_exec(), sys_link(), sys_mkdir(), sys_mknod(), sys_open(), and sys_unlink().

5.25.2.7 begin_op()

```
void begin_op ( )
```

Definition at line 126 of file log.c.

```
acquire(&log.lock);
00129
          while(1){
00130
           if(log.committing){
           sleep(&log, &log.lock);
} else if(log.lh.n + (log.outstanding+1) *MAXOPBLOCKS > LOGSIZE){
// this op might exhaust log space; wait for commit.
00131
00132
00133
              sleep(&log, &log.lock);
00135
00136
              log.outstanding += 1;
00137
               release(&log.lock);
00138
              break;
00139
            }
00140
       }
00141 }
```

Referenced by exec(), exit(), fileclose(), filewrite(), sys_chdir(), sys_link(), sys_mkdir(), sys_mknod(), sys_open(), and sys_unlink().

5.25.2.8 binit()

```
void binit (
     void )
```

Definition at line 39 of file bio.c.

```
00040 {
00041
        struct buf *b;
00042
        initlock(&bcache.lock, "bcache");
00043
00044
00045 //PAGEBREAK!
00046
        // Create linked list of buffers
        bcache.head.prev = &bcache.head;
bcache.head.next = &bcache.head;
00047
00048
00049
        for(b = bcache.buf; b < bcache.buf+NBUF; b++) {</pre>
        b->next = bcache.head.next;
00050
         b->prev = &bcache.head;
00051
00052
          initsleeplock(&b->lock, "buffer");
00053
          bcache.head.next->prev = b;
00054
          bcache.head.next = b;
00055
00056 }
```

5.25.2.9 bread()

```
struct buf * bread (
               uint dev,
              uint blockno )
Definition at line 97 of file bio.c.
00098 {
00099
        struct buf *b;
00100
        b = bget(dev, blockno);
00101
00102
        if((b->flags & B_VALID) == 0) {
00103
         iderw(b);
00104
00105
       return b;
00106 }
```

Referenced by balloc(), bfree(), bmap(), bzero(), ialloc(), ilock(), install_trans(), itrunc(), iupdate(), read_head(), readi(), readsb(), write_head(), write_log(), and writei().

5.25.2.10 brelse()

void brelse (

00136

00137

00138 00139 00140

00141 }

}

```
struct buf * b )
Definition at line 121 of file bio.c.
00122 {
         if(!holdingsleep(&b->lock))
           panic("brelse");
00125
00126
        releasesleep(&b->lock);
00127
00128
         acquire (&bcache.lock);
00129
         b->refcnt--;
00130
        if (b->refcnt == 0) {
00131
          // no one is waiting for it.
          b->next->prev = b->prev;
b->prev->next = b->next;
00132
00133
          b->next = bcache.head.next;
b->prev = &bcache.head;
00134
00135
```

bcache.head.next->prev = b;

bcache.head.next = b;

release (&bcache.lock);

Referenced by balloc(), bfree(), bmap(), bzero(), ialloc(), ilock(), install_trans(), itrunc(), iupdate(), read_head(), readi(), readsb(), write_head(), write_log(), and writei().

5.25.2.11 bwrite()

```
00111 {
00112     if(!holdingsleep(&b->lock))
00113         panic("bwrite");
00114     b->flags |= B_DIRTY;
00115     iderw(b);
00116 }
```

Referenced by install_trans(), write_head(), and write_log().

5.25.2.12 chpr()

```
int chpr (
                      int pid,
                      int priority )
Definition at line 559 of file proc.c.
           struct proc *p;
acquire(&ptable.lock);
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if(p->pid == pid) {
    p->priority = priority;
    break;
}
00561
00562
00563
00564
00565
00566
                 break;
00567
00568
         release(&ptable.lock);
return pid;
00569
00571 }
```

Referenced by main(), and sys_chpr().

5.25.2.13 clearpteu()

```
void clearpteu (
          pde_t * pgdir,
          char * uva )
```

Definition at line 303 of file vm.c.

```
00304 {
00305    pte_t *pte;
00306
00307    pte = walkpgdir(pgdir, uva, 0);
00308    if(pte == 0)
00309        panic("clearpteu");
00310    *pte &= ~PTE_U;
00311 }
```

Referenced by exec().

5.25.2.14 cmostime()

```
void cmostime ( {\tt struct\ rtcdate\ *\ r\ )}
```

Definition at line 196 of file lapic.c.

```
00197 {
00198
       struct rtcdate t1, t2;
00199
       int sb, bcd;
00200
00201
        sb = cmos_read(CMOS_STATB);
00202
       bcd = (sb & (1 & 2)) == 0;
00203
00204
00205
        // make sure CMOS doesn't modify time while we read it
00206
        for(;;) {
        fill_rtcdate(&t1);
if(cmos_read(CMOS_STATA) & CMOS_UIP)
00207
00208
         continue;
fill_rtcdate(&t2);
00209
00210
00211
         if (memcmp(&t1, &t2, sizeof(t1)) == 0)
00212
            break;
00213
```

```
00214
        // convert
00215
00216
        if(bcd) {
00217 #define
                   CONV(x)
                                (t1.x = ((t1.x \gg 4) * 10) + (t1.x & 0xf))
         CONV (second);
00218
00219
           CONV (minute);
          CONV (hour );
CONV (day );
00220
00221
00222
           CONV (month );
          CONV(year );
ndef CONV
00223
00224 #undef
00225 }
00226
00227 *r = t1;
00228 r->year += 2000;
00229 }
```

5.25.2.15 consoleinit()

```
void consoleinit (
     void )
```

Definition at line 289 of file console.c.

```
00290 {
00291    initlock(&cons.lock, "console");
00292
00293    devsw[CONSOLE].write = consolewrite;
00294    devsw[CONSOLE].read = consoleread;
00295    cons.locking = 1;
00296
00297    ioapicenable(IRO_KBD, 0);
00298 }
```

5.25.2.16 consoleintr()

```
void consoleintr (
          int(*)(void) getc )
```

Definition at line 192 of file console.c.

```
00193 {
00194
        int c, doprocdump = 0;
00195
00196
        acquire(&cons.lock);
00197
        while((c = getc()) >= 0){
00198
          switch(c){
          case C('P'): // Process listing.
  // procdump() locks cons.lock indirectly; invoke later
  doprocdump = 1;
00199
00200
00201
00202
            break:
00203
           case C('U'): // Kill line.
00204
            while(input.e != input.w &&
                   input.buf[(input.e-1) % INPUT_BUF] != '\n'){
00205
               input.e--;
consputc(BACKSPACE);
00206
00207
00208
00209
             break;
00210
           case C('H'): case '\x7f': // Backspace
00211
           if(input.e != input.w){
00212
               input.e--;
               consputc(BACKSPACE);
00213
00214
00215
             break;
00216
           default:
            if(c != 0 && input.e-input.r < INPUT_BUF) {
    c = (c == '\r') ? '\n' : c;
    input.buf[input.e++ % INPUT_BUF] = c;</pre>
00217
00218
00219
00220
               00221
00222
                 input.w = input.e;
```

Referenced by kbdintr(), and uartintr().

5.25.2.17 copyout()

Definition at line 366 of file vm.c.

```
00367 {
00368
         char *buf, *pa0;
00369
        uint n, va0;
00370
00371
        buf = (char*)p;
        while(len > 0){
        va0 = (uint)PGROUNDDOWN(va);
pa0 = uva2ka(pgdir, (char*)va0);
if(pa0 == 0)
00373
00374
00375
00376
         return -1;
n = PGSIZE - (va - va0);
00377
         if(n > len)
00378
00379
           n = len;
00380
          memmove(pa0 + (va - va0), buf, n);
          len -= n;
buf += n;
00381
00382
        va = va0 + PGSIZE;
00383
00384
00385
        return 0;
00386 }
```

Referenced by exec().

5.25.2.18 copyuvm()

Definition at line 316 of file vm.c.

```
panic("copyuvm: page not present");
00330
          pa = PTE_ADDR(*pte);
00331
          flags = PTE_FLAGS(*pte);
          if((mem = kalloc()) == 0)
00332
            goto bad;
00333
00334
          memmove(mem, (char*)P2V(pa), PGSIZE);
00335
          if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0) {</pre>
00336
            kfree (mem);
            goto bad;
00337
00338
       }
00339
00340
        return d:
00341
00342 bad:
00343
       freevm(d);
00344
        return 0;
00345 }
```

Referenced by fork().

5.25.2.19 cprintf()

Definition at line 55 of file console.c.

```
int i, c, locking;
00057
        uint *argp;
char *s;
00058
00059
00060
00061
        locking = cons.locking;
00062
        if (locking)
00063
          acquire(&cons.lock);
00064
00065
        if (fmt == 0)
00066
         panic("null fmt");
00067
00068
        argp = (uint*)(void*)(&fmt + 1);
        for(i = 0; (c = fmt[i] & 0xff) != 0; i++) {
  if(c != '%') {
00069
00070
00071
            consputc(c);
00072
            continue;
00073
00074
          c = fmt[++i] & Oxff;
00075
          if(c == 0)
00076
            break;
00077
          switch(c){
          case 'd':
00078
          printint(*argp++, 10, 1);
break;
00079
08000
          case 'x':
case 'p':
00081
00082
          printint(*argp++, 16, 0);
00083
00084
            break;
00085
          case 's':
           if((s = (char*)*argp++) == 0)
    s = "(null)";
00086
00087
00088
            for(; *s; s++)
00089
              consputc(*s);
          break; case '%':
00090
00091
00092
           consputc('%');
00093
00094
          default:
            // Print unknown % sequence to draw attention.
consputc('%');
00095
00096
00097
             consputc(c);
00098
             break;
00099
00100
00101
        if(locking)
00102
00103
          release(&cons.lock);
```

Referenced by allocuvm(), cps(), exec(), iinit(), ioapicinit(), panic(), procdump(), syscall(), and trap().

5.25.2.20 cps()

```
int cps ( void )
```

Definition at line 537 of file proc.c.

```
00538 {
00539 struct proc *p;
00540 //Enables interrupts on this processor.
00541 sti();
00542
00543 //Loop over process table looking for process with pid.
00544 acquire(&ptable.lock);
00545 cprintf("name \t pid \t state \t priority \n");
00546 for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
00547    if(p->state == SLEEPING)
00548         cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name,p->pid,p->priority);
00549    else if(p->state == RUNNING)
00550         cprintf("%s \t %d \t RUNNING \t %d \n ", p->name,p->pid,p->priority);
00551    else if(p->state == RUNNABLE)
00552    cprintf("%s \t %d \t RUNNABLE \t %d \n ", p->name,p->pid,p->priority);
00553 }
00554    release(&ptable.lock);
00555    return 23;
00556 }
```

Referenced by main(), and sys_cps().

5.25.2.21 cpuid()

```
int cpuid (
     void )
```

Definition at line 31 of file proc.c.

```
00031 {
00032 return mycpu()-cpus;
00033 }
```

Referenced by seginit(), and trap().

5.25.2.22 deallocuvm()

```
int deallocuvm (
          pde_t * pgdir,
          uint oldsz,
          uint newsz )
```

Definition at line 256 of file vm.c.

```
00257 {
         pte_t *pte;
00259
         uint a, pa;
00260
00261
        if (newsz >= oldsz)
         return oldsz;
00262
00263
00264
         a = PGROUNDUP(newsz);
00265
         for(; a < oldsz; a += PGSIZE) {</pre>
         pte = walkpgdir(pgdir, (char*)a, 0);
00266
00267
           if(!pte)
           a = PGADDR(PDX(a) + 1, 0, 0) - PGSIZE;
else if((*pte & PTE_P) != 0) {
  pa = PTE_ADDR(*pte);
00268
00269
00270
              if(pa == 0)
```

Referenced by allocuvm(), freevm(), and growproc().

5.25.2.23 dirlink()

```
int dirlink (
                struct inode * dp,
                char * name,
                uint inum )
Definition at line 552 of file fs.c.
00554
00555
         struct dirent de;
00556
        struct inode *ip;
00557
00558
        // Check that name is not present.
00559
        if((ip = dirlookup(dp, name, 0)) != 0){
         iput(ip);
return -1;
00560
00561
00562
00563
        // Look for an empty dirent.
for(off = 0; off < dp->size; off += sizeof(de)) {
   if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
00564
00565
00566
00567
             panic("dirlink read");
00568
         if(de.inum == 0)
00569
             break;
00570
00571
        strncpy(de.name, name, DIRSIZ);
00573
        de.inum = inum;
        if(writei(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
00574
00575
         panic("dirlink");
00576
00577
        return 0;
00578 }
```

Referenced by create(), and sys_link().

5.25.2.24 dirlookup()

```
00526 {
00527     uint off, inum;
00528     struct dirent de;
00529
00530     if(dp->type != T_DIR)
00531         panic("dirlookup not DIR");
00532
00533     for(off = 0; off < dp->size; off += sizeof(de)) {
        if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
```

```
panic("dirlookup read");
00536
          if(de.inum == 0)
00537
            continue;
         if(namecmp(name, de.name) == 0){
00538
00539
          // entry matches path element
00540
           if(poff)
00541
             *poff = off;
00542
           inum = de.inum;
00543
           return iget(dp->dev, inum);
00544
       }
00545
00546
00547
       return 0;
00548 }
```

Referenced by create(), dirlink(), namex(), and sys unlink().

5.25.2.25 end_op()

```
void end_op ( )
```

Definition at line 146 of file log.c.

```
00147 {
00148
         int do_commit = 0;
00149
00150
         acquire(&log.lock);
00151
        log.outstanding -= 1;
00152
        if(log.committing)
00153
           panic("log.committing");
00154
        if(log.outstanding == 0){
        do_commit = 1;
log.committing = 1;
} else {
00155
00156
00157
          // begin_op() may be waiting for log space,
00158
          // and decrementing log.outstanding has decreased
// the amount of reserved space.
00159
00160
00161
           wakeup(&log);
00162
00163
        release(&log.lock);
00164
00165
         if (do_commit) {
          // call commit w/o holding locks, since not allowed
// to sleep with locks.
commit();
00166
00167
00168
00169
           acquire(&log.lock);
00170
           log.committing = 0;
00171
           wakeup(&log);
00172
           release(&log.lock);
00173
00174 }
```

Referenced by exec(), exit(), fileclose(), filewrite(), sys_chdir(), sys_link(), sys_mkdir(), sys_mknod(), sys_open(), and sys_unlink().

5.25.2.26 exec()

Definition at line 11 of file exec.c.

```
00012 {
00013    char *s, *last;
00014    int i, off;
00015    uint argc, sz, sp, ustack[3+MAXARG+1];
00016    struct elfhdr elf;
00017    struct inode *ip;
```

```
00018
        struct proghdr ph;
00019
        pde_t *pgdir, *oldpgdir;
00020
        struct proc *curproc = myproc();
00021
00022
        begin_op();
00023
        if((ip = namei(path)) == 0){
00024
00025
          end_op();
00026
          cprintf("exec: fail\n");
00027
          return -1;
00028
00029
        ilock(ip);
00030
        pgdir = 0;
00031
00032
        // Check ELF header
00033
        if(readi(ip, (char*)&elf, 0, sizeof(elf)) != sizeof(elf))
00034
          goto had:
00035
        if(elf.magic != ELF_MAGIC)
00036
          goto bad;
00037
00038
        if((pgdir = setupkvm()) == 0)
00039
          goto bad;
00040
00041
        \ensuremath{//} Load program into memory.
00042
        sz = 0;
00043
        for(i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){</pre>
00044
          if(readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
00045
            goto bad;
          if(ph.type != ELF_PROG_LOAD)
00046
00047
            continue;
00048
          if(ph.memsz < ph.filesz)</pre>
00049
             goto bad;
00050
           if(ph.vaddr + ph.memsz < ph.vaddr)</pre>
00051
            goto bad;
00052
          if((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
00053
            goto bad;
00054
          if(ph.vaddr % PGSIZE != 0)
            goto bad;
00056
          if(loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz) < 0)</pre>
00057
            goto bad;
00058
        iunlockput(ip);
00059
00060
        end op();
00061
        ip = 0;
00062
00063
        // Allocate two pages at the next page boundary.
00064
        // Make the first inaccessible. Use the second as the user stack.
00065
        sz = PGROUNDUP(sz);
00066
        if((sz = allocuvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
          goto bad;
00067
00068
        clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
00069
00070
00071
        \ensuremath{//} Push argument strings, prepare rest of stack in ustack.
00072
        for(argc = 0; argv[argc]; argc++) {
  if(argc >= MAXARG)
00073
00074
            goto bad;
          sp = (sp - (strlen(argv[argc]) + 1)) & ~3;
if(copyout(pgdir, sp, argv[argc], strlen(argv[argc]) + 1) < 0)</pre>
00075
00076
00077
             goto bad:
00078
          ustack[3+argc] = sp;
00079
08000
        ustack[3+argc] = 0;
00081
        ustack[0] = 0xfffffffff; // fake return PC
00082
        ustack[1] = argc;
ustack[2] = sp - (argc+1)*4; // argv pointer
00083
00084
00085
00086
        sp -= (3+argc+1) * 4;
00087
        if (copyout (pgdir, sp, ustack, (3+argc+1) *4) < 0)</pre>
00088
00089
00090
        // Save program name for debugging.
        for(last=s=path; *s; s++)
  if(*s == '/')
00091
         if(*s ==
00092
00093
             last = s+1;
00094
        safestrcpy(curproc->name, last, sizeof(curproc->name));
00095
00096
        \ensuremath{//} Commit to the user image.
00097
        oldpgdir = curproc->pgdir;
00098
        curproc->pgdir = pgdir;
00099
        curproc->sz = sz;
00100
        curproc->tf->eip = elf.entry; // main
        curproc->tf->esp = sp;
00101
00102
        switchuvm(curproc);
00103
        freevm(oldpgdir);
00104
        return 0:
```

Referenced by bigargtest(), exectest(), main(), runcmd(), and sys_exec().

5.25.2.27 exit()

```
void exit (
     void )
```

Definition at line 228 of file proc.c.

```
00230
        struct proc *curproc = myproc();
00231
        struct proc *p;
00232
        int fd:
00233
        if(curproc == initproc)
00235
          panic("init exiting");
00236
00237
        // Close all open files.
        for(fd = 0; fd < NOFILE; fd++) {
  if(curproc->ofile[fd]) {
00238
00239
00240
             fileclose(curproc->ofile[fd]);
00241
             curproc->ofile[fd] = 0;
00242
00243
00244
00245
        begin_op();
00246
        iput (curproc->cwd);
00247
        end_op();
00248
        curproc->cwd = 0;
00249
00250
        acquire(&ptable.lock);
00251
00252
        // Parent might be sleeping in wait().
        wakeup1 (curproc->parent);
00254
00255
        // Pass abandoned children to init.
00256
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){</pre>
          if (p->parent == curproc) {
  p->parent = initproc;
00257
00258
             if (p->state == ZOMBIE)
00259
00260
               wakeup1(initproc);
00261
00262
00263
00264
        \ensuremath{//} Jump into the scheduler, never to return.
        curproc->state = ZOMBIE;
00266
00267
        panic("zombie exit");
00268 }
```

Referenced by argptest(), bigargtest(), bigdir(), bigdir(), bigwrite(), bsstest(), cat(), concreate(), createdelete(), dirfile(), dirtest(), exectest(), exitiputtest(), exitwait(), forktest(), fourfiles(), fourteen(), iputtest(), iref(), linktest(), linkunlink(), main(), mem(), openiputtest(), opentest(), panic(), pipe1(), rmdot(), rsect(), runcmd(), sbrktest(), sharedfd(), subdir(), sys_exit(), trap(), uio(), unlinkread(), validatetest(), wc(), writetest(), writetest1(), and wsect().

5.25.2.28 fetchint()

```
int fetchint (
              uint addr,
              int * ip)
Definition at line 18 of file syscall.c.
       struct proc *curproc = myproc();
00020
00021
00022
        if(addr >= curproc->sz || addr+4 > curproc->sz)
```

Referenced by argint(), and sys_exec().

*ip = *(int*)(addr);

return -1;

5.25.2.29 fetchstr()

υ0025 return 0; 00026 }

00023

00024

```
int fetchstr (
             uint addr,
            char ** pp )
```

Definition at line 32 of file syscall.c.

```
00034
       char *s, *ep;
00035
       struct proc *curproc = myproc();
00036
00037
       if (addr >= curproc->sz)
         return -1;
00038
00039
       *pp = (char*)addr;
00040
       ep = (char*)curproc->sz;
00041
       for(s = *pp; s < ep; s++) {
00042
        if(*s == 0)
00043
           return s - *pp;
00044
       }
00045
       return -1;
00046 }
```

Referenced by argstr(), and sys_exec().

5.25.2.30 filealloc()

```
struct file * filealloc (
            void )
```

Definition at line 27 of file file.c.

```
00028 {
00029
         struct file *f;
00030
00031
         acquire(&ftable.lock);
        for(f = ftable.file; f < ftable.file + NFILE; f++) {
   if(f->ref == 0) {
    f->ref = 1;
}
00032
00033
00034
00035
             release(&ftable.lock);
00036
             return f;
00037
          }
00038
00039
        release(&ftable.lock);
00040
        return 0;
00041 }
```

Referenced by pipealloc(), and sys_open().

5.25.2.31 fileclose()

```
void fileclose (
                 struct file * f )
Definition at line 57 of file file.c.
00058 4
00059
         struct file ff;
00060
00061
         acquire(&ftable.lock);
         if(f->ref < 1)
  panic("fileclose");</pre>
00062
00063
00064
         if(--f->ref > 0){
          release(&ftable.lock);
00065
00066
           return;
00067
        ff = *f;
f->ref = 0;
f->type = FD_NONE;
00068
00069
00070
00071
         release (&ftable.lock);
00072
00073
         if(ff.type == FD_PIPE)
         pipeclose(ff.pipe, ff.writable);
else if(ff.type == FD_INODE){
00074
00075
         begin_op();
iput(ff.ip);
00076
00077
00078
           end_op();
00079 }
00080 }
```

Referenced by exit(), pipealloc(), sys_close(), sys_open(), and sys_pipe().

5.25.2.32 filedup()

Referenced by fork(), and sys_dup().

5.25.2.33 fileinit()

00053 }

00023 }

5.25.2.34 fileread()

```
int fileread (
                 struct file * f,
                 char * addr,
                 int n)
Definition at line 97 of file file.c.
00098 {
00099
         int r;
00100
         if(f->readable == 0)
00101
         return -1;
if(f->type == FD_PIPE)
00102
00103
00104
           return piperead(f->pipe, addr, n);
        if(f->type == FD_INODE) {
  ilock(f->ip);
  if((r = readi(f->ip, addr, f->off, n)) > 0)
    f->off += r;
00105
00106
00107
00108
00109
           iunlock(f->ip);
00110
          return r;
00111
00112 panic("fileread");
00113 }
```

Referenced by sys_read().

5.25.2.35 filestat()

Definition at line 84 of file file.c.

```
00085 {
00086     if(f->type == FD_INODE) {
00087          ilock(f->ip);
00088          stati(f->ip, st);
00089          iunlock(f->ip);
00090          return 0;
00091     }
00092     return -1;
00093 }
```

Referenced by sys fstat().

5.25.2.36 filewrite()

```
int filewrite (  \mbox{struct file } * \ f, \\ \mbox{char } * \ addr, \\ \mbox{int } n \ )
```

Definition at line 118 of file file.c.

```
00119 {
00120    int r;
00121
00122    if(f->writable == 0)
00123        return -1;
00124    if(f->type == FD_PIPE)
00125        return pipewrite(f->pipe, addr, n);
00126    if(f->type == FD_INODE) {
```

```
// write a few blocks at a time to avoid exceeding
00128
            // the maximum log transaction size, including
00129
            // i-node, indirect block, allocation blocks,
            \ensuremath{//} and 2 blocks of slop for non-aligned writes.
00130
            // this really belongs lower down, since writei()
// might be writing a device like the console.
00131
00132
00133
            int max = ((MAXOPBLOCKS-1-1-2) / 2) * 512;
00134
            int i = 0;
            while (i < n) {
  int n1 = n - i;
  if (n1 > max)
     n1 = max;
00135
00136
00137
00138
00139
00140
              begin_op();
00141
               ilock(f->ip);
              if ((r = writei(f->ip, addr + i, f->off, n1)) > 0)
  f->off += r;
00142
00143
               iunlock(f->ip);
00144
00145
              end_op();
00146
00147
              if(r < 0)
              break;
if(r != n1)
00148
00149
              panic("short filewrite");
i += r;
00150
00151
00152
00153
            return i == n ? n : -1;
00155 panic("filewrite");
00156 }
00154
```

Referenced by sys_write().

5.25.2.37 fork()

```
int fork (
     void )
```

Definition at line 181 of file proc.c.

```
00182 {
00183
        int i, pid;
00184
        struct proc *np;
00185
        struct proc *curproc = myproc();
00186
00187
        // Allocate process.
00188
        if((np = allocproc()) == 0){
00189
         return -1;
00190
00191
        // Copy process state from proc.
if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
00192
00193
00194
         kfree(np->kstack);
00195
          np->kstack = 0;
          np->state = UNUSED;
00196
00197
          return -1;
00198
        np->sz = curproc->sz;
np->parent = curproc;
00199
00200
00201
        *np->tf = *curproc->tf;
00202
00203
        // Clear %eax so that fork returns 0 in the child.
00204
        np->tf->eax = 0;
00205
00206
        for(i = 0; i < NOFILE; i++)</pre>
        if(curproc->ofile[i])
00207
00208
            np->ofile[i] = filedup(curproc->ofile[i]);
00209
        np->cwd = idup(curproc->cwd);
00210
00211
        safestrcpy(np->name, curproc->name, sizeof(curproc->name));
00212
00213
        pid = np->pid;
00214
00215
        acquire(&ptable.lock);
00216
00217
        np->state = RUNNABLE;
00218
00219
        release(&ptable.lock);
```

Referenced by bigargtest(), concreate(), createdelete(), exitiputtest(), exitwait(), fork1(), fork1(), fourfiles(), linkunlink(), main(), mem(), openiputtest(), pipe1(), preempt(), sbrktest(), sharedfd(), sys_fork(), uio(), and validatetest().

5.25.2.38 freevm()

```
void freevm (
          pde_t * pgdir )
```

Definition at line 284 of file vm.c.

```
00285 {
00286
            uint i;
00287
00288
            if(pgdir == 0)
00289
               panic("freevm: no pgdir");
           panic("Ireevm: no pgdir");
deallocuvm(pgdir, KERNBASE, 0);
for(i = 0; i < NPDENTRIES; i++){
   if(pgdir[i] & PTE_P){
      char * v = P2V(PTE_ADDR(pgdir[i]));
}</pre>
00290
00291
00292
00293
                   kfree(v);
00295
00296
00297
            kfree((char*)pgdir);
00298 }
```

Referenced by copyuvm(), exec(), setupkvm(), and wait().

5.25.2.39 getcallerpcs()

```
void getcallerpcs (
    void * ,
    uint * )
```

Referenced by panic(), and procdump().

5.25.2.40 getprocs()

```
int getprocs (
                int max,
                struct uproc * )
```

Referenced by main().

5.25.2.41 growproc()

```
int growproc (
              int n)
Definition at line 159 of file proc.c.
00160 {
00161
       uint sz;
       struct proc *curproc = myproc();
00162
00163
00164
       sz = curproc->sz;
       if(n > 0){
00165
       if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
00166
       return -1;
} else if (n < 0) {
00167
00168
       if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
00169
00170
           return -1;
00171
00172
       curproc->sz = sz;
00173
       switchuvm(curproc);
```

Referenced by sys_sbrk().

return 0;

00174

00175 }

5.25.2.42 holding()

```
int holding ( {\tt struct\ spinlock\ *\ lock\ })
```

Definition at line 90 of file spinlock.c.

```
00091 {
00092    int r;
00093    pushcli();
00094    r = lock->locked && lock->cpu == mycpu();
00095    popcli();
00096    return r;
00097 }
```

Referenced by acquire(), release(), and sched().

5.25.2.43 holdingsleep()

```
int holdingsleep ( {\tt struct \ sleeplock \ * \it lk} \ )
```

Definition at line 45 of file sleeplock.c.

```
00046 {
00047    int r;
00048
00049    acquire(&lk->lk);
00050    r = lk->locked && (lk->pid == myproc()->pid);
00051    release(&lk->lk);
00052    return r;
```

Referenced by brelse(), bwrite(), iderw(), and iunlock().

5.25.2.44 ialloc()

```
00197
              int inum;
             struct buf *bp;
struct dinode *dip;
00198
00199
00200
00201
              for(inum = 1; inum < sb.ninodes; inum++) {</pre>
                bor(inum = 1; inum < sb.ninodes; inum++){
  bp = bread(dev, IBLOCK(inum, sb));
  dip = (struct dinode*)bp->data + inum%IPB;
  if(dip->type == 0){ // a free inode
   memset(dip, 0, sizeof(*dip));
  dip->type = type;
  log_write(bp); // mark it allocated on the disk
  broles(bp);
00202
00203
00204
00205
00206
00207
00208
                     brelse(bp);
00209
                     return iget(dev, inum);
00210
00211
                 brelse(bp);
             1
00212
             panic("ialloc: no inodes");
00213
00214 }
```

Referenced by create().

5.25.2.45 ideinit()

```
void ideinit (
     void )
```

Definition at line 51 of file ide.c.

```
00052 {
00053
00054
         initlock(&idelock, "ide");
ioapicenable(IRQ_IDE, ncpu - 1);
00055
00056
00057
         idewait(0);
00058
00059
         // Check if disk 1 is present
         outb(0x1f6, 0xe0 | (1«4));
for(i=0; i<1000; i++){</pre>
00060
00061
          if(inb(0x1f7) != 0) {
  havedisk1 = 1;
00062
00063
00064
              break;
00065
00066
00067
00068
         // Switch back to disk 0.
00069
         outb(0x1f6, 0xe0 | (0«4));
```

5.25.2.46 ideintr()

```
void ideintr (
     void )
```

Definition at line 104 of file ide.c.

```
00105 {
00106 struct buf *b;
00107
```

```
// First queued buffer is the active request.
00109
         acquire(&idelock);
00110
00111
         if((b = idequeue) == 0){
          release(&idelock);
00112
00113
           return:
00114
00115
         idequeue = b->qnext;
00116
00117
         \ensuremath{//} Read data if needed.
        if(!(b->flags & B_DIRTY) && idewait(1) >= 0)
insl(0x1f0, b->data, BSIZE/4);
00118
00119
00120
00121
         // Wake process waiting for this buf.
        b->flags |= B_VALID;
b->flags &= ~B_DIRTY;
00122
00123
00124
        wakeup(b);
00125
        // Start disk on next buf in queue.
00127
        if(idequeue != 0)
00128
           idestart (idequeue);
00129
00130
        release (&idelock);
00131 }
```

Referenced by trap().

5.25.2.47 iderw()

```
void iderw (
               struct buf * b )
Definition at line 138 of file ide.c.
00139 {
        struct buf **pp;
00140
00141
        if(!holdingsleep(&b->lock))
00143
          panic("iderw: buf not locked");
00144
        if((b->flags & (B_VALID|B_DIRTY)) == B_VALID)
00145
         panic("iderw: nothing to do");
       if(b->dev != 0 && !havedisk1)
  panic("iderw: ide disk 1 not present");
00146
00147
00148
00149
       acquire(&idelock); //DOC:acquire-lock
00150
00151
        // Append b to idequeue.
00152
        b->qnext = 0;
        for (pp=&idequeue; *pp; pp=&(*pp)->qnext) //DOC:insert-queue
00153
00154
00155
        *pp = b;
00156
00157
       // Start disk if necessary.
00158
       if(idequeue == b)
00159
         idestart(b);
00160
00161
       // Wait for request to finish.
00162
       while((b->flags & (B_VALID|B_DIRTY)) != B_VALID){
00163
         sleep(b, &idelock);
00164
00165
00166
00167
       release(&idelock);
00168 }
```

Referenced by bread(), and bwrite().

5.25.2.48 idtinit()

```
void idtinit (
void )

Definition at line 30 of file trap.c.
```

5.25.2.49 idup()

00032

00033 }

00283 }

lidt(idt, sizeof(idt));

Referenced by fork(), and namex().

5.25.2.50 iinit()

```
void iinit ( \quad \quad \text{int } dev \; )
```

Definition at line 172 of file fs.c.

Referenced by forkret().

5.25.2.51 ilock()

```
void ilock (
                 struct inode * ip )
Definition at line 288 of file fs.c.
00290
         struct buf *bp;
00291
         struct dinode *dip;
00292
         if(ip == 0 || ip->ref < 1)
  panic("ilock");</pre>
00293
00294
00295
00296
         acquiresleep(&ip->lock);
00297
         if(ip->valid == 0){
00298
           bp = bread(ip->dev, IBLOCK(ip->inum, sb));
00299
           dip = (struct dinode*)bp->data + ip->inum%IPB;
00300
00301
            ip->type = dip->type;
00302
            ip->major = dip->major;
           ip->minor = dip->minor;
ip->nlink = dip->nlink;
ip->size = dip->size;
00303
00304
00305
            memmove(ip->addrs, dip->addrs, sizeof(ip->addrs));
00306
00307
           brelse(bp);
00308
            ip->valid = 1;
           if(ip->type == 0)
  panic("ilock: no type");
00309
00310
00311
00312 }
```

Referenced by consoleread(), consolewrite(), create(), exec(), fileread(), filestat(), filewrite(), namex(), sys_chdir(), sys_link(), sys_open(), and sys_unlink().

5.25.2.52 initlock()

```
void initlock (
          struct spinlock * lk,
           char * name )
```

Definition at line 13 of file spinlock.c.

```
00014 {
00015    1k->name = name;
00016    1k->locked = 0;
00017    1k->cpu = 0;
00018 }
```

Referenced by binit(), consoleinit(), fileinit(), ideinit(), iinit(), initlog(), initsleeplock(), kinit1(), pinit(), pinit(), and tvinit().

5.25.2.53 initlog()

```
void initlog (
int dev )
```

Definition at line 54 of file log.c.

```
00055 {
00056     if (sizeof(struct logheader) >= BSIZE)
00057          panic("initlog: too big logheader");
00058
00059          struct superblock sb;
00060          initlock(&log.lock, "log");
00061          readsb(dev, &sb);
00062          log.start = sb.logstart;
00063          log.size = sb.nlog;
00064          log.dev = dev;
00065          recover_from_log();
00066 }
```

Referenced by forkret().

5.25.2.54 initsleeplock()

Referenced by binit(), and iinit().

lk->name = name;

 $1k \rightarrow locked = 0;$

5.25.2.55 inituvm()

00019 1k->pid = 0;

00017

00018

00020 }

Definition at line 183 of file vm.c.

```
00184 {
00185
        char *mem;
00186
00187
        if(sz >= PGSIZE)
00188
         panic("inituvm: more than a page");
00189
        mem = kalloc();
        memset (mem, 0, PGSIZE);
mappages (pgdir, 0, PGSIZE, V2P (mem), PTE_W|PTE_U);
00190
00191
        memmove(mem, init, sz);
00192
00193 }
```

Referenced by userinit().

5.25.2.56 ioapicenable()

```
void ioapicenable ( \inf \ \mathit{irq}, \inf \ \mathit{cpu} \ )
```

Definition at line 68 of file ioapic.c.

Referenced by consoleinit(), ideinit(), and uartinit().

5.25.2.57 ioapicinit()

```
void ioapicinit (
               void )
Definition at line 49 of file ioapic.c.
00050 {
        int i, id, maxintr;
00051
00052
00053
        ioapic = (volatile struct ioapic*)IOAPIC;
       maxintr = (ioapicread(REG_VER) » 16) & 0xFF;
00055
        id = ioapicread(REG_ID) >> 24;
00056
       if(id != ioapicid)
00057
          cprintf("ioapicinit: id isn't equal to ioapicid; not a MP\n");
00058
00059
       // Mark all interrupts edge-triggered, active high, disabled,
00060
        // and not routed to any CPUs.
        for(i = 0; i <= maxintr; i++) {</pre>
00062
          ioapicwrite(REG_TABLE+2*i, INT_DISABLED | (T_IRQ0 + i));
00063
          ioapicwrite(REG_TABLE+2*i+1, 0);
00064
00065 }
```

5.25.2.58 iput()

```
void iput (
               struct inode * ip )
Definition at line 332 of file fs.c.
00333 4
00334
        acquiresleep(&ip->lock);
        if(ip->valid && ip->nlink == 0){
00335
00336
          acquire(&icache.lock);
00337
          int r = ip->ref;
00338
          release(&icache.lock);
          if(r == 1){ // inode has no links and no other references: truncate and free.
00339
00340
00341
            itrunc(ip);
00342
            ip->type = 0;
00343
            iupdate(ip);
00344
            ip->valid = 0;
00345
          }
00346
00347
        releasesleep(&ip->lock);
00348
00349
        acquire(&icache.lock);
00350
        ip->ref--;
00351
       release(&icache.lock);
00352 }
```

Referenced by dirlink(), exit(), fileclose(), iunlockput(), namex(), sys chdir(), and sys link().

5.25.2.59 iunlock()

Referenced by consoleread(), consolewrite(), fileread(), filestat(), filewrite(), iunlockput(), namex(), sys_chdir(), sys_link(), and sys_open().

5.25.2.60 iunlockput()

```
void iunlockput ( {\tt struct\ inode}\ *\ ip\ ) Definition at line 356 of file fs.c.
```

```
00357 {
00358    iunlock(ip);
00359    iput(ip);
00360 }
```

Referenced by create(), exec(), namex(), sys_chdir(), sys_link(), sys_mkdir(), sys_mknod(), sys_open(), and sys_unlink().

5.25.2.61 iupdate()

```
void iupdate ( {\tt struct\ inode\ *\ ip\ )}
```

Definition at line 221 of file fs.c.

```
00222 {
00223
         struct buf *bp;
00224
         struct dinode *dip;
00225
        bp = bread(ip->dev, IBLOCK(ip->inum, sb));
00227
        dip = (struct dinode*)bp->data + ip->inum%IPB;
        dip->type = ip->type;
dip->major = ip->major;
dip->minor = ip->minor;
00228
00229
00230
         dip->nlink = ip->nlink;
dip->size = ip->size;
00231
00232
00233
         memmove(dip->addrs, ip->addrs, sizeof(ip->addrs));
00234
         log_write(bp);
00235
         brelse(bp);
00236 }
```

Referenced by create(), iput(), itrunc(), sys_link(), sys_unlink(), and writei().

5.25.2.62 kalloc()

```
char * kalloc (
     void )
```

Definition at line 83 of file kalloc.c.

```
00084 {
        struct run *r;
00086
00087
        if(kmem.use_lock)
        acquire(&kmem.lock);
r = kmem.freelist;
00088
00089
00090
        if(r)
00091
          kmem.freelist = r->next;
00092
        if(kmem.use_lock)
00093
          release(&kmem.lock);
00094
        return (char*)r;
00095 }
```

Referenced by allocproc(), allocuvm(), copyuvm(), inituvm(), pipealloc(), setupkvm(), startothers(), and walkpgdir().

5.25.2.63 kbdintr()

Referenced by trap().

5.25.2.64 kfree()

```
void kfree (
                char * v )
Definition at line 60 of file kalloc.c.
00061 {
00062
         struct run *r;
00063
        if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP)
  panic("kfree");
00064
00065
00066
00067
        // Fill with junk to catch dangling refs.
00068
        memset(v, 1, PGSIZE);
00069
00070
00071
        if (kmem.use_lock)
        acquire(&kmem.lock);
r = (struct run*)v;
00072
         r->next = kmem.freelist;
00073
```

Referenced by allocuvm(), copyuvm(), deallocuvm(), fork(), freerange(), freevm(), pipealloc(), pipeclose(), and wait().

5.25.2.65 kill()

00074

00075

00076

00077 }

```
int kill ( int \ pid \ )
```

Definition at line 480 of file proc.c.

kmem.freelist = r;

release(&kmem.lock);

if(kmem.use_lock)

```
00481 {
00482
         struct proc *p;
00483
00484
         acquire(&ptable.lock);
00485
         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
           if (p->pid == pid) {
  p->killed = 1;
00486
00487
             // Wake process from sleep if necessary.
if(p->state == SLEEPING)
00488
00489
00490
               p->state = RUNNABLE;
00491
              release(&ptable.lock);
00492
             return 0;
00493
           }
00494
00495
        release(&ptable.lock);
00496
         return -1;
00497 }
```

Referenced by main(), mem(), preempt(), sbrktest(), sys_kill(), and validatetest().

5.25.2.66 kinit1()

```
void kinit1 (
            void * vstart,
            void * vend )
```

Definition at line 32 of file kalloc.c.

```
00034 initlock(&kmem.lock, "kmem");

00035 kmem.use_lock = 0;

00036 freerange(vstart, vend);
```

5.25.2.67 kinit2()

```
void kinit2 (
            void * vstart,
            void * vend )
```

Definition at line 40 of file kalloc.c.

```
00041 {
00042     freerange(vstart, vend);
00043     kmem.use_lock = 1;
00044 }
```

5.25.2.68 kvmalloc()

```
void kvmalloc (
          void )
```

Definition at line 141 of file vm.c.

5.25.2.69 lapiceoi()

```
void lapiceoi (
          void )
```

Definition at line 110 of file lapic.c.

```
00111 {
00112
         if(lapic)
00113
          lapicw(EOI, 0);
00114 }
```

Referenced by trap().

5.25.2.70 lapicid()

Referenced by mycpu(), and panic().

5.25.2.71 lapicinit()

```
void lapicinit (
    void )
```

Definition at line 55 of file lapic.c.

```
00057
         if(!lapic)
00058
            return;
00059
         // Enable local APIC; set spurious interrupt vector.
lapicw(SVR, ENABLE | (T_IRQ0 + IRQ_SPURIOUS));
00060
00061
00062
00063
         // The timer repeatedly counts down at bus frequency
00064
          // from lapic[TICR] and then issues an interrupt.
         // If xv6 cared more about precise timekeeping,
00065
00066
         // TICR would be calibrated using an external time source.
         lapicw(TDCR, X1);
lapicw(TIMER, PERIODIC | (T_IRQ0 + IRQ_TIMER));
lapicw(TICR, 10000000);
00067
00068
00069
00070
         // Disable logical interrupt lines.
lapicw(LINTO, MASKED);
lapicw(LINT1, MASKED);
00071
00072
00073
00074
00075
         // Disable performance counter overflow interrupts
00076
         // on machines that provide that interrupt entry.
00077
         if(((lapic[VER] \gg 16) \& 0xFF) >= 4)
00078
            lapicw(PCINT, MASKED);
00079
         // Map error interrupt to IRQ_ERROR.
lapicw(ERROR, T_IRQ0 + IRQ_ERROR);
08000
00081
00082
00083
         // Clear error status register (requires back-to-back writes).
00084
         lapicw(ESR, 0);
         lapicw(ESR, 0);
00085
00086
00087
         // Ack any outstanding interrupts.
00088
         lapicw(EOI, 0);
00089
         // Send an Init Level De-Assert to synchronise arbitration {\tt ID's.}
00090
         lapicw(ICRHI, 0);
lapicw(ICRLO, BCAST | INIT | LEVEL);
while(lapic[ICRLO] & DELIVS)
00091
00092
00093
00094
00095
         \ensuremath{//} Enable interrupts on the APIC (but not on the processor).
00096
00097
         lapicw(TPR, 0);
00098 }
```

Referenced by mpenter().

5.25.2.72 lapicstartap()

```
void lapicstartap (
                uchar apicid,
                uint addr )
Definition at line 129 of file lapic.c.
00130 {
00131
         int i;
00132
         ushort *wrv;
00133
00134
         // "The BSP must initialize CMOS shutdown code to OAH \,
00135
        // and the warm reset vector (DWORD based at 40:67) to point at
        // the AP startup code prior to the [universal startup algorithm]."
outb(CMOS_PORT, 0xF); // offset 0xF is shutdown code
outb(CMOS_PORT+1, 0x0A);
00136
00137
00138
00139
         wrv = (ushort*)P2V((0x40*4 | 0x67)); // Warm reset vector
00140
         wrv[0] = 0;
         wrv[1] = addr » 4;
00141
00142
00143
         // "Universal startup algorithm."
        // Send INIT (level-triggered) interrupt to reset other CPU.
00144
        lapicw(ICRHI, apicid«24);
lapicw(ICRLO, INIT | LEVEL | ASSERT);
00146
00147
         microdelay(200);
00148
         lapicw(ICRLO, INIT | LEVEL);
00149
                               // should be 10ms, but too slow in Bochs!
        microdelay(100);
00150
00151
        // Send startup IPI (twice!) to enter code.
00152
        // Regular hardware is supposed to only accept a STARTUP
00153
         // when it is in the halted state due to an INIT. So the second
00154
         // should be ignored, but it is part of the official Intel algorithm.
        // Bochs complains about the second one. Too bad for Bochs. for (i = 0; i < 2; i++) {
00155
00156
         lapicw(ICRHI, apicid«24);
lapicw(ICRLO, STARTUP | (addr»12));
00157
00158
00159
           microdelay(200);
00160
00161 }
```

Referenced by startothers().

5.25.2.73 loaduvm()

int loaduvm (

```
pde_t * pgdir,
                  char * addr,
                  struct inode * ip,
                  uint offset,
                  uint sz )
Definition at line 198 of file vm.c.
00199 {
00200
          uint i, pa, n;
00201
         pte_t *pte;
00202
         if((uint) addr % PGSIZE != 0)
  panic("loaduvm: addr must be page aligned");
for(i = 0; i < sz; i += PGSIZE) {
  if((pte = walkpgdir(pgdir, addr+i, 0)) == 0)</pre>
00203
00204
00205
              panic("loaduvm: address should exist");
00208
            pa = PTE_ADDR(*pte);
00209
            if(sz - i < PGSIZE)</pre>
00210
              n = sz - i;
00211
            else
00212
              n = PGSIZE;
00213
             if(readi(ip, P2V(pa), offset+i, n) != n)
00214
              return -1;
00215
00216
          return 0;
00217 }
```

Referenced by exec().

5.25.2.74 log_write()

```
void log_write (
                 struct buf * b )
Definition at line 214 of file log.c.
00215 {
00216
         int i;
00217
00218
         if (log.lh.n >= LOGSIZE || log.lh.n >= log.size - 1)
         panic("too big a transaction");
if (log.outstanding < 1)
panic("log_write outside of trans");</pre>
00220
00221
00222
         acquire(&log.lock);
for (i = 0; i < log.lh.n; i++) {
  if (log.lh.block[i] == b->blockno) // log absorbtion
00223
00224
00225
00226
00227
         log.lh.block[i] = b->blockno;
00228
00229
        if (i == log.lh.n)
           log.lh.n++;
00230
00231
         b->flags |= B_DIRTY; // prevent eviction
00232 release(&log.lock);
00233 }
```

Referenced by balloc(), bfree(), bmap(), bzero(), ialloc(), iupdate(), and writei().

5.25.2.75 memcmp()

Definition at line 16 of file string.c.

```
00017
00018
        const uchar *s1, *s2;
00020
        s1 = v1;
00021 s2 = v2;
        while (n-- > 0) {
00022
        if(*s1 != *s2)
return *s1 - *s2;
00023
00024
00025 s1++, s2++;
00026 }
00027
00028
       return 0;
00029 }
```

Referenced by cmostime(), mpconfig(), and mpsearch1().

5.25.2.76 memmove()

Definition at line 32 of file string.c.

```
00033 {
00034 const char *s;
```

```
char *d;
00036
00037
        s = src;
        d = dst;
00038
       if(s < d && s + n > d) {
s += n;
00039
00040
00041
         d += n;
00042
00043
            *--d = *--s;
00044
       } else
00045
        while (n-- > 0)
           *d++ = *s++;
00046
00047
00048
       return dst;
00049 }
```

Referenced by cgaputc(), copyout(), copyouvm(), fmtname(), grep(), iderw(), ilock(), inituvm(), install_trans(), iupdate(), ls(), main(), memcpy(), readi(), readsb(), skipelem(), startothers(), write_log(), and writei().

5.25.2.77 memset()

```
void * memset (
     void * dst,
     int c,
     uint n)
```

Definition at line 5 of file string.c.

Referenced by allocproc(), allocuvm(), backcmd(), bigfile(), bzero(), cgaputc(), concreate(), execcmd(), fmtname(), fourfiles(), getcmd(), ialloc(), inituvm(), kfree(), listcmd(), main(), pipecmd(), redircmd(), setupkvm(), sharedfd(), sys_exec(), sys_unlink(), userinit(), and walkpgdir().

5.25.2.78 microdelay()

```
void microdelay ( \quad \text{int } us \ )
```

Definition at line 119 of file lapic.c.

```
00120 {
00121 }
```

Referenced by cmos_read(), lapicstartap(), and uartputc().

5.25.2.79 mpinit()

```
void mpinit (
                 void )
Definition at line 92 of file mp.c.
00093 +
00094
         uchar *p, *e;
int ismp;
struct mp *mp;
00096
00097
         struct mpconf *conf;
         struct mpproc *proc;
00098
00099
         struct mpioapic *ioapic;
00100
00101
         if((conf = mpconfig(&mp)) == 0)
           panic("Expect to run on an SMP");
00103
00104
         lapic = (uint*)conf->lapicaddr;
00105
         for(p=(uchar*)(conf+1), e=(uchar*)conf+conf->length; p<e; ){</pre>
00106
           switch(*p){
           case MPPROC:
00107
00108
             proc = (struct mpproc*)p;
00109
              if(ncpu < NCPU) {</pre>
00110
                cpus[ncpu].apicid = proc->apicid; // apicid may differ from ncpu
00111
                ncpu++;
00112
             p += sizeof(struct mpproc);
00113
00114
              continue;
00115
           case MPIOAPIC:
            ioapic = (struct mpioapic*)p;
ioapicid = ioapic->apicno;
00116
00117
            p += sizeof(struct mpioapic);
00118
             continue;
00119
00120
           case MPBUS:
00121
           case MPIOINTR:
00122
           case MPLINTR:
00123
            p += 8;
              continue;
00124
00125
           default:
00126
            ismp = 0;
00127
              break;
00128
           }
00129
         if(!ismp)
00130
           panic("Didn't find a suitable machine");
00131
00132
00133
         if (mp->imcrp) {
          // Bochs doesn't support IMCR, so this doesn't run on Bochs.
// But it would on real hardware.
outb(0x22, 0x70); // Select IMCR
outb(0x23, inb(0x23) | 1); // Mask external interrupts.
00134
00135
00136
00137
00138
00139 }
```

5.25.2.80 mycpu()

struct cpu * mycpu (

```
void )
Definition at line 38 of file proc.c.
00039 {
       int apicid, i;
00041
00042
       if(readeflags()&FL_IF)
00043
         panic("mycpu called with interrupts enabled\n");
00044
00045
       apicid = lapicid();
       // APIC IDs are not guaranteed to be contiguous. Maybe we should have
00047
       // a reverse map, or reserve a register to store &cpus[i].
00048
        for (i = 0; i < ncpu; ++i) {
00049
        if (cpus[i].apicid == apicid)
00050
            return &cpus[i];
00051
       panic("unknown apicid\n");
```

Referenced by acquire(), cpuid(), holding(), myproc(), popcli(), pushcli(), sched(), sched(), startothers(), and switchuvm().

5.25.2.81 myproc()

```
struct proc * myproc ( )
Definition at line 58 of file proc.c.
00058
00059
         struct cpu *c;
00060
        struct proc *p;
pushcli();
00061
00062
        c = mycpu();
00063
        p = c->proc;
00064
        popcli();
00065
         return p;
00066 }
```

Referenced by acquiresleep(), argfd(), argint(), argptr(), consoleread(), exec(), exit(), fdalloc(), fetchint(), fork(), growproc(), holdingsleep(), namex(), piperead(), pipewrite(), sched(), sleep(), sys_chdir(), sys_close(), sys_getpid(), sys_pipe(), sys_sleep(), sys_sleep(), sys_call(), trap(), wait(), and yield().

5.25.2.82 namecmp()

Referenced by dirlookup(), and sys unlink().

5.25.2.83 namei()

Referenced by exec(), sys_chdir(), sys_link(), sys_open(), and userinit().

5.25.2.84 nameiparent()

Referenced by create(), namex(), sys_link(), and sys_unlink().

5.25.2.85 panic()

```
void panic (
                  char * s)
Definition at line 107 of file console.c.
00109
00110
         uint pcs[10];
00111
00112
         cli();
         cons.locking = 0;
00113
          // use lapiccpunum so that we can call panic from mycpu()
00115
          cprintf("lapicid %d: panic: ", lapicid());
00116
         cprintf(s);
         cprintf("\n");
00117
         getcallerpcs(&s, pcs);
for(i=0; i<10; i++)
   cprintf(" %p", pcs[i]);
panicked = 1; // freeze other CPU</pre>
00118
00119
00120
00121
00122
          for(;;)
00123
```

Referenced by acquire(), balloc(), bfree(), bget(), bmap(), brelse(), bwrite(), cgaputc(), clearpteu(), copyuvm(), cprintf(), create(), deallocuvm(), dirlink(), dirlookup(), end_op(), exit(), fileclose(), filedup(), fileread(), filewrite(), fork1(), freevm(), ialloc(), iderw(), idestart(), iget(), ilock(), initlog(), initlowm(), isdirempty(), iunlock(), kfree(), loaduvm(), log_write(), mappages(), mpinit(), mycpu(), parseblock(), parsecmd(), parseexec(), parseredirs(), popcli(), release(), runcmd(), sched(), setupkvm(), sleep(), switchuvm(), sys_unlink(), trap(), and userinit().

5.25.2.86 picenable()

00124 }

```
void picenable (
    int )
```

5.25.2.87 picinit()

```
void picinit (
     void )
```

Definition at line 11 of file picirq.c.

5.25.2.88 pinit()

```
void pinit ( void )
```

Definition at line 24 of file proc.c.

```
00025 {
00026    initlock(&ptable.lock, "ptable");
00027 }
```

5.25.2.89 pipealloc()

```
int pipealloc (
                  struct file ** f0,
                  struct file ** f1 )
Definition at line 23 of file pipe.c.
00024
00025
          struct pipe *p;
00026
00027
          p = 0;
         *f0 = *f1 = 0;
00028
00029
          if((\star f0 = filealloc()) == 0 \mid \mid (\star f1 = filealloc()) == 0)
00030
            goto bad;
          if((p = (struct pipe*)kalloc()) == 0)
00031
          goto bad;
00032
00033
         p->readopen = 1;
00034
         p->writeopen = 1;
         p > wiredepen = 1,
p -> nwrite = 0;
p -> nread = 0;
initlock(&p->lock, "pipe");
(*f0) -> type = FD_PIPE;
(*f0) -> readable = 1;
00035
00036
00037
00038
00039
00040
         (*f0) ->writable = 0;
         (*f0) ->pipe = p;
(*f1) ->type = FD_PIPE;
(*f1) ->readable = 0;
00041
00042
00043
00044
         (*f1)->writable = 1;
00045
         (*f1)->pipe = p;
00046
         return 0;
00047
00048 //PAGEBREAK: 20
00049 bad:
00050
         if(p)
00051
           kfree((char*)p);
00052
          if(*f0)
00053
            fileclose(*f0);
         if(*f1)
00054
00055
           fileclose(*f1);
00056
         return -1;
00057 }
```

Referenced by sys_pipe().

5.25.2.90 pipeclose()

Definition at line 60 of file pipe.c.

```
00061 {
00062
        acquire(&p->lock);
00063
        if (writable) {
00064
         p->writeopen = 0;
00065
           wakeup(&p->nread);
00066
        } else {
         p->readopen = 0;
wakeup(&p->nwrite);
00067
00068
00069
        if(p->readopen == 0 && p->writeopen == 0){
  release(&p->lock);
00070
00071
00072
           kfree((char*)p);
00073
         } else
00074
           release(&p->lock);
00075 }
```

Referenced by fileclose().

5.25.2.91 piperead()

```
int piperead (
               struct pipe * p,
               char * addr,
               int n)
Definition at line 101 of file pipe.c.
00102 {
00103
00104
00105
        acquire(&p->lock);
        while(p->nread == p->nwrite && p->writeopen){ //DOC: pipe-empty
00106
         if (myproc() ->killed) {
00108
           release(&p->lock);
00109
            return -1;
00110
         sleep(&p->nread, &p->lock); //DOC: piperead-sleep
00111
00112
00113
        for(i = 0; i < n; i++) { //DOC: piperead-copy</pre>
        if (p->nread == p->nwrite)
00114
00115
           break;
         addr[i] = p->data[p->nread++ % PIPESIZE];
00116
00117
00118
       wakeup(&p->nwrite); //DOC: piperead-wakeup
00119
       release(&p->lock);
```

Referenced by fileread().

return i;

00120

00121 }

5.25.2.92 pipewrite()

```
int pipewrite (
          struct pipe * p,
          char * addr,
          int n )
```

Definition at line 79 of file pipe.c.

```
00080 {
00081
         int i:
00082
00083
         acquire(&p->lock);
00084
         for(i = 0; i < n; i++) {
          while(p->nwrite == p->nread + PIPESIZE){ //DOC: pipewrite-full
  if(p->readopen == 0 || myproc()->killed){
00085
00086
00087
               release(&p->lock);
00088
               return -1;
00089
00090
             wakeup(&p->nread);
00091
             sleep(&p->nwrite, &p->lock); //DOC: pipewrite-sleep
00092
00093
          p->data[p->nwrite++ % PIPESIZE] = addr[i];
00094
00095
         wakeup(&p->nread); //DOC: pipewrite-wakeup1
00096
         release(&p->lock);
00097
         return n;
00098 }
```

Referenced by filewrite().

5.25.2.93 popcli()

```
void popcli (
               void )
Definition at line 117 of file spinlock.c.
00118 {
        if(readeflags()&FL_IF)
00120
          panic("popcli - interruptible");
00121
        if(--mycpu()->ncli < 0)</pre>
          panic("popcli");
00122
00123
        if (mycpu()->ncli == 0 && mycpu()->intena)
00124
          sti();
00125 }
```

Referenced by holding(), myproc(), release(), and switchuvm().

5.25.2.94 procdump()

```
void procdump (
                 void )
Definition at line 504 of file proc.c.
00505 {
         static char *states[] = {
[UNUSED] "unused",
00507
                       "embryo",
"sleep ",
00508
          [EMBRYO]
00509
         [SLEEPING]
                      "runble",
00510
         [RUNNABLE]
00511
         [RUNNING]
                       "run
         [ZOMBIE]
                       "zombie"
00513
00514
         int i;
00515
         struct proc *p;
         char *state;
00516
00517
         uint pc[10];
00518
00519
         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
00520
           if(p->state == UNUSED)
00521
             continue;
           if(p->state >= 0 && p->state < NELEM(states) && states[p->state])
00522
00523
             state = states[p->state];
           else
00524
00525
             state = "???";
           cprintf("%d %s %s", p->pid, state, p->name);
00526
00527
           if(p->state == SLEEPING){
            getcallerpcs((uint*)p->context->ebp+2, pc);
for(i=0; i<10 && pc[i] != 0; i++)
    cprintf(" %p", pc[i]);</pre>
00528
00529
00530
00531
00532
           cprintf("\n");
         }
00533
```

Referenced by consoleintr().

5.25.2.95 pushcli()

00534 }

```
void pushcli (
               void )
Definition at line 105 of file spinlock.c.
00106 {
00107
        int eflags;
00108
00109
        eflags = readeflags();
00110
        cli();
00111
        if (mycpu()->ncli == 0)
00112
         mycpu()->intena = eflags & FL_IF;
        mycpu()->ncli += 1;
00113
00114 }
```

Referenced by acquire(), holding(), myproc(), and switchuvm().

5.25.2.96 readi()

```
int readi (
                 struct inode * ip,
                 char * dst,
                 uint off,
                 uint n )
Definition at line 453 of file fs.c.
00454 {
00455
         uint tot, m;
00456
         struct buf *bp;
00457
00458
         if(ip->type == T_DEV){
         if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].read)
return =1.
00459
00460
             return -1;
00461
           return devsw[ip->major].read(ip, dst, n);
00462
00463
00464
         if(off > ip->size || off + n < off)</pre>
        return -1;
if (off + n > ip->size)
n = ip->size - off;
00465
00466
00467
00468
00469
        for(tot=0; tot<n; tot+=m, off+=m, dst+=m) {</pre>
        bp = bread(ip->dev, bmap(ip, off/BSIZE));
m = min(n - tot, BSIZE - off%BSIZE);
memmove(dst, bp->data + off%BSIZE, m);
00470
00471
00472
00473
          brelse(bp);
00474
        }
00475
        return n;
00476 }
```

Referenced by dirlink(), dirlookup(), exec(), fileread(), isdirempty(), and loaduvm().

5.25.2.97 readsb()

```
void readsb (  & \text{int } \textit{dev,} \\ & \text{struct superblock} \, * \, \textit{sb} \, \, ) \\
```

Definition at line 32 of file fs.c.

```
00033 {
00034     struct buf *bp;
00035
00036     bp = bread(dev, 1);
00037     memmove(sb, bp->data, sizeof(*sb));
00038     brelse(bp);
00039 }
```

Referenced by iinit(), and initlog().

5.25.2.98 release()

```
void release ( {\tt struct\ spinlock}\ *\ lk\ )
```

Definition at line 47 of file spinlock.c.

```
1k -> pcs[0] = 0;
         1k \rightarrow cpu = 0;
00053
00054
00055
         // Tell the C compiler and the processor to not move loads or stores
00056
         // past this point, to ensure that all the stores in the critical
00057
         // section are visible to other cores before the lock is released.
         // Both the C compiler and the hardware may re-order loads and
00059
         // stores; __sync_synchronize() tells them both not to.
         __sync_synchronize();
00060
00061
         // Release the lock, equivalent to lk->locked = 0.
00062
         // This code can't use a C assignment, since it might
// not be atomic. A real OS would use C atomics here.
asm volatile("mov1 $0, %0" : "+m" (lk->locked) : );
00063
00064
00065
00066
00067
         popcli();
00068 }
```

Referenced by acquiresleep(), allocproc(), begin_op(), bget(), brelse(), chpr(), consoleintr(), consoleread(), consolewrite(), cprintf(), cps(), end_op(), filealloc(), fileclose(), filedup(), fork(), forkret(), holdingsleep(), ideintr(), iderw(), idup(), iget(), iput(), kalloc(), kfree(), kill(), log_write(), pipeclose(), piperead(), pipewrite(), releasesleep(), scheduler(), sleep(), sys_sleep(), sys_uptime(), trap(), userinit(), wait(), wakeup(), and yield().

5.25.2.99 releasesleep()

```
void releasesleep ( {\tt struct \ sleeplock \ * \it lk} \ )
```

Definition at line 35 of file sleeplock.c.

Referenced by brelse(), iput(), and iunlock().

5.25.2.100 safestrcpy()

```
\begin{array}{c} \text{char * safestrcpy (} \\ \text{char * $s$,} \\ \text{const char * $t$,} \\ \text{int $n$ )} \end{array}
```

Definition at line 83 of file string.c.

Referenced by exec(), fork(), and userinit().

5.25.2.101 sched()

```
void sched (
                void )
Definition at line 366 of file proc.c.
00367
00368
        int intena;
        struct proc *p = myproc();
00369
00370
        if(!holding(&ptable.lock))
  panic("sched ptable.lock");
if(mycpu()->ncli != 1)
00371
00372
00373
          panic("sched locks");
00374
00375
        if(p->state == RUNNING)
00376
          panic("sched running");
00377
        if(readeflags()&FL_IF)
00378
          panic("sched interruptible");
00379
        intena = mycpu()->intena;
00380
        swtch(&p->context, mycpu()->scheduler);
        mycpu()->intena = intena;
00381
```

Referenced by exit(), sleep(), and yield().

5.25.2.102 scheduler()

00382 }

```
void scheduler (
     void )
```

Definition at line 323 of file proc.c.

```
00324 {
         struct proc *p;
00326
         struct cpu *c = mycpu();
00327
         c->proc = 0;
00328
00329
         for(;;) {
00330
           // Enable interrupts on this processor.
00331
           sti();
00332
00333
           \ensuremath{//} Loop over process table looking for process to run.
00334
            acquire(&ptable.lock);
           for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
  if (p->state != RUNNABLE)
00335
00336
00337
                continue;
00338
             // Switch to chosen process. It is the process's job // to release ptable.lock and then reacquire it // before jumping back to us.
00339
00340
00341
00342
             c->proc = p;
             switchuvm(p);
p->state = RUNNING;
00343
00344
00345
00346
              swtch(&(c->scheduler), p->context);
00347
             switchkvm();
00348
00349
              // Process is done running for now.
00350
              // It should have changed its p->state before coming back.
00351
              c->proc = 0;
00352
00353
           release(&ptable.lock);
00354
00355
         }
00356 }
```

Referenced by sched().

5.25.2.103 seginit()

```
void seginit (
    void )
```

Definition at line 16 of file vm.c.

```
00017
00018
               struct cpu *c;
00020
               // Map "logical" addresses to virtual addresses using identity map.
              // Cannot share a CODE descriptor for both kernel and user // because it would have to have DPL_USR, but the CPU forbids
00021
00022
              // an interrupt from CPL=0 to DPL=3.
00023
              // an interrupt from CPL=U to DPL=3.
c = &cpus[cpuid()];
c->gdt[SEG_KCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, 0);
c->gdt[SEG_KDATA] = SEG(STA_W, 0, 0xffffffff, 0);
c->gdt[SEG_UCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, DPL_USER);
c->gdt[SEG_UDATA] = SEG(STA_W, 0, 0xffffffff, DPL_USER);
d-/c->cdt [SEG_UDATA] = SEG(STA_W, 0, 0xffffffff, DPL_USER);
00024
00026
00027
00028
00029
               lgdt(c->gdt, sizeof(c->gdt));
00030 }
```

Referenced by mpenter().

5.25.2.104 setproc()

```
void setproc (
          struct proc * )
```

5.25.2.105 setupkvm()

Definition at line 119 of file vm.c.

```
00120 {
00121
       pde_t *pgdir;
00122
       struct kmap *k;
00123
       if((pgdir = (pde_t*)kalloc()) == 0)
  return 0;
00124
00125
       memset(pgdir, 0, PGSIZE);
if (P2V(PHYSTOP) > (void*)DEVSPACE)
00126
00127
         panic("PHYSTOP too high");
00128
00129
       for(k = kmap; k < &kmap[NELEM(kmap)]; k++)</pre>
        00130
00131
00132
           freevm(pgdir);
00133
           return 0;
00134
00135
       return pgdir;
00136 }
```

Referenced by copyuvm(), exec(), kvmalloc(), and userinit().

5.25.2.106 sleep()

```
void sleep (
               void * chan,
               struct spinlock * lk)
Definition at line 418 of file proc.c.
00419
00420
        struct proc *p = myproc();
00421
00422
        if(p == 0)
00423
         panic("sleep");
00424
00425
       if(1k == 0)
         panic("sleep without lk");
00426
00427
00428
       // Must acquire ptable.lock in order to
00429
       // change p->state and then call sched.
00430
       // Once we hold ptable.lock, we can be
       // guaranteed that we won't miss any wakeup
// (wakeup runs with ptable.lock locked),
00431
00432
        00433
00434
00435
00436
          release(lk);
00437
       // Go to sleep.
00438
       p->chan = chan;
p->state = SLEEPING;
00439
00440
00441
00442
00443
00444
       // Tidy up.
00445
        p->chan = 0;
00446
00447
       // Reacquire original lock.
00448
       if(lk != &ptable.lock){ //DOC: sleeplock2
00449
        release(&ptable.lock);
00450
          acquire(lk);
       }
00451
00452 }
```

Referenced by acquiresleep(), begin_op(), consoleread(), iderw(), main(), openiputtest(), piperead(), pipewrite(), sbrktest(), sys_sleep(), validatetest(), and wait().

5.25.2.107 stati()

Definition at line 440 of file fs.c.

Referenced by filestat().

5.25.2.108 strlen()

```
int strlen ( {\rm const\ char}\ *\ s\ )
```

Definition at line 97 of file string.c.

```
00098 {
00099 int n;
00100
00101 for(n = 0; s[n]; n++)
00102 ;
00103 return n;
00104 }
```

Referenced by exec(), fmtname(), ls(), main(), parsecmd(), and printf().

5.25.2.109 strncmp()

Definition at line 59 of file string.c.

Referenced by namecmp().

5.25.2.110 strncpy()

```
\begin{array}{c} \text{char * strncpy (} \\ \text{char * $s$,} \\ \text{const char * $t$,} \\ \text{int $n$ )} \end{array}
```

Definition at line 69 of file string.c.

```
00070 {
00071    char *os;
00072
00073    os = s;
00074    while (n-- > 0 && (*s++ = *t++) != 0)
00075    ;
00076    while (n-- > 0)
00077     *s++ = 0;
00078    return os;
00079 }
```

Referenced by dirlink(), main(), and sys_getprocs().

5.25.2.111 switchkvm()

Referenced by kvmalloc(), mpenter(), and scheduler().

5.25.2.112 switchuvm()

void switchuvm (

00166 00167 mycpu()->gdt[SEG_TSS] = SEG16(STS_T32A, &mycpu()->ts, 00168 sizeof(mycpu()->ts)-1, 0);mycpu()->gdt[SEG_TSS].s = 0;
mycpu()->ts.ss0 = SEG_KDATA « 3;
mycpu()->ts.esp0 = (uint)p->kstack + KSTACKSIZE;
// setting IOPL=0 in eflags *and* iomb beyond the tss segment limit 00169 00170 00171 00173 // forbids I/O instructions (e.g., inb and outb) from user space 00174 mycpu()->ts.iomb = (ushort) 0xFFFF; ltr(SEG_TSS « 3);
lcr3(V2P(p->pgdir)); // switch to process's address space 00175 00176 00177 popcli(); 00178 }

Referenced by exec(), growproc(), and scheduler().

5.25.2.113 swtch()

Referenced by sched(), and scheduler().

5.25.2.114 syscall()

```
void syscall (
     void )
```

Definition at line 140 of file syscall.c.

```
00141 {
00142
        int num;
00143
        struct proc *curproc = myproc();
00144
00145
        num = curproc->tf->eax;
        if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
  curproc->tf->eax = syscalls[num]();
00146
00147
00148 } else {
        cprintf("%d %s: unknown sys call %d\n",
00150
                   curproc->pid, curproc->name, num);
00151 (00152 )
00153 }
         curproc->tf->eax = -1;
```

Referenced by trap().

5.25.2.115 timerinit()

```
void timerinit (
    void )
```

5.25.2.116 tvinit()

```
void tvinit (
     void )
```

Definition at line 18 of file trap.c.

```
00019 {
00020    int i;
00021
00022    for(i = 0; i < 256; i++)
00023         SETGATE(idt[i], 0, SEG_KCODE«3, vectors[i], 0);
00024    SETGATE(idt[T_SYSCALL], 1, SEG_KCODE«3, vectors[T_SYSCALL], DPL_USER);
00025
00026    initlock(&tickslock, "time");
00027 }
```

5.25.2.117 uartinit()

```
void uartinit (
    void )
```

Definition at line 20 of file uart.c.

```
00029
        outb(COM1+0, 115200/9600);
        outb(COM1+1, 0);
outb(COM1+3, 0x03);
00030
                                // Lock divisor, 8 data bits.
00031
        outb(COM1+4, 0);
00032
00033
                              // Enable receive interrupts.
        outb(COM1+1, 0x01);
00034
        // If status is 0xFF, no serial port.
00036
        if(inb(COM1+5) == 0xFF)
00037
         return;
00038
        uart = 1;
00039
00040
        // Acknowledge pre-existing interrupt conditions;
        // enable interrupts.
00041
00042
        inb(COM1+2);
00043
        inb(COM1+0);
00044
        ioapicenable(IRQ_COM1, 0);
00045
00046
        // Announce that we're here.
for(p="xv6...\n"; *p; p++)
00048
         uartputc(*p);
00049 }
```

5.25.2.118 uartintr()

```
void uartintr (
     void )
```

Definition at line 74 of file uart.c.

```
00075 {
00076 consoleintr(uartgetc);
00077 }
```

Referenced by trap().

5.25.2.119 uartputc()

```
void uartputc ( \quad \text{int } c \ )
```

Definition at line 52 of file uart.c.

```
00053 {
00054    int i;
00055
00056    if(!uart)
00057     return;
00058    for(i = 0; i < 128 && !(inb(COM1+5) & 0x20); i++)
00059     microdelay(10);
00060    outb(COM1+0, c);
00061 }
```

Referenced by consputc(), and uartinit().

5.25.2.120 userinit()

```
void userinit (
void )

Definition at line 121 of file proc.c.
```

```
00122 {
```

```
00123
          struct proc *p;
00124
         extern char _binary_initcode_start[], _binary_initcode_size[];
00125
00126
         p = allocproc();
00127
         initproc = p;
if((p->pgdir = setupkvm()) == 0)
00128
00129
            panic("userinit: out of memory?");
00131
         inituvm(p->pgdir, _binary_initcode_start, (int)_binary_initcode_size);
00132
         p->sz = PGSIZE;
         memset(p->tf, 0, sizeof(*p->tf));
p->tf->cs = (SEG_UCODE « 3) | DPL_USER;
p->tf->ds = (SEG_UDATA « 3) | DPL_USER;
00133
00134
00135
00136
         p->tf->es = p->tf->ds;
00137
         p\rightarrow tf\rightarrow ss = p\rightarrow tf\rightarrow ds;
         p->tf->eflags = FL_IF;
p->tf->esp = PGSIZE;
00138
00139
00140
         p->tf->eip = 0; // beginning of initcode.S
00141
00142
         safestrcpy(p->name, "initcode", sizeof(p->name));
00143
         p->cwd = namei("/");
00144
         // this assignment to p->state lets other cores // run this process. the acquire forces the above \,
00145
00146
         // writes to be visible, and the lock is also needed // because the assignment might not be atomic.
00147
00148
00149
         acquire(&ptable.lock);
00150
00151
         p->state = RUNNABLE;
00152
00153
         release(&ptable.lock);
00154 }
```

5.25.2.121 uva2ka()

Definition at line 350 of file vm.c.

```
00351 {
00352
       pte_t *pte;
00353
00354
       pte = walkpgdir(pgdir, uva, 0);
00355
       if((*pte & PTE_P) == 0)
         return 0;
00357
       if((*pte & PTE_U) == 0)
00358
         return 0;
00359
       return (char*)P2V(PTE_ADDR(*pte));
00360 }
```

Referenced by copyout().

5.25.2.122 wait()

```
int wait (
                void )
Definition at line 273 of file proc.c.
00274
00275
         struct proc *p;
        int havekids, pid;
struct proc *curproc = myproc();
00277
00278
00279
         acquire(&ptable.lock);
00280
        for(;;){
  // Scan through table looking for exited children.
00281
           havekids = 0;
00283
           for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
00284
             if(p->parent != curproc)
             continue;
havekids = 1;
00285
00286
00287
             if(p->state == ZOMBIE){
00288
               // Found one.
                pid = p->pid;
00290
               kfree (p->kstack);
00291
                p->kstack = 0;
               freevm(p->pgdir);
p->pid = 0;
00292
00293
00294
               p->parent = 0;
00295
               p->name[0] = 0;
00296
               p->killed = 0;
               p->state = UNUSED;
00297
                release(&ptable.lock);
00298
00299
                return pid;
00300
00301
00302
00303
           // No point waiting if we don't have any children.
00304
           if(!havekids || curproc->killed){
00305
            release (&ptable.lock);
00306
             return -1;
00307
00308
           // Wait for children to exit. (See wakeup1 call in proc_exit.)
sleep(curproc, &ptable.lock); //DOC: wait-sleep
00309
00310
```

Referenced by bigargtest(), concreate(), createdelete(), exitiputtest(), exitwait(), forktest(), fourfiles(), linkunlink(), main(), mem(), openiputtest(), pipe1(), preempt(), runcmd(), sbrktest(), sharedfd(), sys_wait(), uio(), and validatetest().

5.25.2.123 wakeup()

}

00311 00312 }

```
void wakeup (
     void * chan )
```

Definition at line 469 of file proc.c.

Referenced by consoleintr(), end_op(), ideintr(), pipeclose(), piperead(), pipewrite(), releasesleep(), and trap().

5.25.2.124 writei()

```
int writei (
                 struct inode * ip,
                  char * src,
                  uint off,
                 uint n )
Definition at line 482 of file fs.c.
00483 {
00484
00485
         struct buf *bp;
00486
00487
         if(ip->type == T_DEV){
         if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].write)
00488
00489
             return -1;
00490
           return devsw[ip->major].write(ip, src, n);
00491
00492
         if(off > ip->size || off + n < off)</pre>
00493
         return -1;
if (off + n > MAXFILE*BSIZE)
00494
00495
00496
           return -1;
00497
00498
         for(tot=0; tot<n; tot+=m, off+=m, src+=m) {</pre>
         bp = bread(ip->dev, bmap(ip, off/BSIZE));
m = min(n - tot, BSIZE - off%BSIZE);
memmove(bp->data + off%BSIZE, src, m);
00499
00500
00501
00502
            log_write(bp);
00503
           brelse(bp);
00504
00505
         if(n > 0 && off > ip->size) {
  ip->size = off;
  iupdate(ip);
00506
00507
00508
00509
```

Referenced by dirlink(), filewrite(), and sys_unlink().

5.25.2.125 yield()

00510

00511 }

```
void yield (
     void )
```

return n;

Definition at line 386 of file proc.c.

```
00387 {
00388    acquire(&ptable.lock); //DOC: yieldlock
00389    myproc()->state = RUNNABLE;
00390    sched();
00391    release(&ptable.lock);
00392 }
```

Referenced by trap().

5.25.3 Variable Documentation

5.25.3.1 ioapicid

```
uchar ioapicid [extern]
```

Definition at line 16 of file mp.c.

Referenced by ioapicinit(), and mpinit().

5.25.3.2 ismp

```
int ismp [extern]
```

Referenced by mpinit().

5.25.3.3 lapic

```
volatile uint* lapic [extern]
```

Definition at line 44 of file lapic.c.

Referenced by lapiceoi(), lapicid(), lapicinit(), lapicw(), and mpinit().

5.25.3.4 ticks

```
uint ticks [extern]
```

Definition at line 15 of file trap.c.

Referenced by sys_sleep(), sys_uptime(), and trap().

5.25.3.5 tickslock

```
struct spinlock tickslock [extern]
```

Definition at line 14 of file trap.c.

Referenced by sys_sleep(), sys_uptime(), trap(), and tvinit().

5.26 defs.h

```
00001 struct buf;
00002 struct context;
00003 struct file;
00004 struct inode;
00005 struct pipe;
00006 struct proc;
00007 struct rtcdate;
00008 struct spinlock;
00009 struct sleeplock;
00010 struct stat;
00011 struct superblock;
00012 struct uproc;
00013
00014 // bio.c
00015 void
                      binit (void);
00016 struct buf*
                       bread(uint, uint);
00017 void
                       brelse(struct buf*);
00018 void
                       bwrite(struct buf*);
00019
00020 // console.c
00021 void
                       consoleinit (void);
00022 void
                       cprintf(char*, ...);
00023 void
                        consoleintr(int(*)(void));
00024 void
                       panic(char*) __attribute__((noreturn));
00025
00026 // exec.c
00027 int
                       exec(char*, char**);
00028
00029 // file.c
00030 struct file* filealloc(void);
00031 void
                       fileclose(struct file*);
00032 struct file*
                        filedup(struct file*);
00032 Scract
                       fileinit(void);
                      fileread(struct file*, char*, int n);
filestat(struct file*, struct stat*);
filewrite(struct file*, char*, int n);
00034 int
00035 int
00036 int
00037
00038 // fs.c
                    readsb(int dev, struct superblock *sb);
dirlink(struct inode*, char*, uint);
00039 void
00040 int
00041 struct inode* dirlookup(struct inode*, char*, uint*);
00042 struct inode*
                        ialloc(uint, short);
00043 struct inode*
                        idup(struct inode*);
00044 void
                        iinit(int dev);
                      ilock(struct inode*);
00045 void
               iput(struct inode*);
iput(struct inode*);
iunlock(struct inode*);
iunlockput(struct inode*);
iupdate(struct inode*);
00046 void
00047 void
00048 void
00049 void
00050 int
                        namecmp(const char*, const char*);
00051 struct inode*
                       namei(char*);
00052 struct inode* nameiparent(char*, char*);
                       readi(struct inode*, char*, uint, uint);
stati(struct inode*, struct stat*);
00053 int
00054 void
00055 int
                       writei(struct inode*, char*, uint, uint);
00056
00057 // ide.c
00058 void
                      ideinit(void);
00059 void
                        ideintr(void);
00060 void
                       iderw(struct buf*);
00061
00062 // ioapic.c
00063 void
                       ioapicenable(int irq, int cpu);
00064 extern uchar
                       ioapicid;
00065 void
                       ioapicinit (void);
00066
00067 // kalloc.c
00068 char*
                      kalloc(void);
                       kfree(char*);
00070 void
                       kinit1(void*, void*);
00071 void
                       kinit2(void*, void*);
00072
00073 // kbd.c
00074 void
                       kbdintr(void);
00075
00076 // lapic.c
00077 void
                       cmostime(struct rtcdate *r);
00078 int
                        lapicid(void);
00079 extern volatile uint* lapic;
00080 void lapiceoi(void);
00081 void
                        lapicinit (void);
00082 void
                        lapicstartap(uchar, uint);
```

5.26 defs.h 163

```
00083 void
                      microdelay(int);
00084
00085 // log.c
00086 void
                       initlog(int dev);
00087 void
                       log_write(struct buf*);
begin_op();
00088 void
00089 void
                       end_op();
00090
00091 // mp.c
00092 extern int
                       ismp;
                      mpinit(void);
00093 void
00094
00095 // picirq.c
00096 void
                       picenable(int);
00097 void
                       picinit(void);
00098
00099 // pipe.c
00100 int
                       pipealloc(struct file**, struct file**);
                       pipeclose(struct pipe*, int);
piperead(struct pipe*, char*, int);
00101 void
00102 int
00103 int
                       pipewrite(struct pipe*, char*, int);
00104
00105 //PAGEBREAK: 16
00106 // proc.c
00107 int
                       cpuid(void);
00108 void
                       exit(void);
00109 int
                       fork(void);
00110 int
                       growproc(int);
00111 int
                       kill(int);
00112 struct cpu*
                       mycpu (void);
00113 struct proc*
                       myproc();
00114 void
                       pinit (void);
00115 void
                       procdump(void);
00116 void
                       scheduler(void) __attribute__((noreturn));
00117 void
                       sched(void);
00118 void
                       setproc(struct proc*);
00119 void
                      sleep(void*, struct spinlock*);
00120 void
                      userinit (void);
00121 int
                      wait(void);
00122 void
                      wakeup(void*);
00123 void
                      yield(void);
00124 int
                       chpr(int pid, int priority);
00125 int
                       cps (void):
00126 int
                      getprocs(int max, struct uproc*);
00127 //int
                         halt(void);
00128
00129 // swtch.S
00130 void
                      swtch(struct context**, struct context*);
00131
00132 // spinlock.c
00133 void
                       acquire(struct spinlock*);
00134 void
                       getcallerpcs(void*, uint*);
00135 int
                       holding(struct spinlock*);
00136 void
                      initlock(struct spinlock*, char*);
00137 void
                       release(struct spinlock*);
00138 void
                      pushcli(void);
00139 void
                      popcli(void);
00140
00141 // sleeplock.c
00142 void
                       acquiresleep(struct sleeplock*);
00143 void
                       releasesleep(struct sleeplock*);
holdingsleep(struct sleeplock*);
00144 int
00145 void
                       initsleeplock(struct sleeplock*, char*);
00146
00147 // string.c
00148 int
                      memcmp(const void*, const void*, uint);
                      memmove(void*, const void*, uint);
memset(void*, int, uint);
00149 void*
00150 void*
00151 char*
                       safestrcpy(char*, const char*, int);
                       strlen(const char*);
00152 int
00153 int
                       strncmp(const char*, const char*, uint);
00154 char*
                       strncpy(char*, const char*, int);
00155
00156 // syscall.c
00157 int
                       argint(int, int*);
argptr(int, char**, int);
00158 int
00159 int
                       argstr(int, char**);
00160 int
                       fetchint(uint, int*);
00161 int
                       fetchstr(uint, char**);
00162 void
                       syscall (void);
00163
00164 // timer.c
00165 void
                       timerinit(void);
00166
00167 // trap.c
00168 void
                       idtinit(void);
00169 extern uint
                       ticks:
```

```
tvinit(void);
00171 extern struct spinlock tickslock;
00172
00173 // uart.c
00173 ,,
00174 void
00175 void
00176 void
                                   uartinit(void);
                                     uartintr(void);
                                    uartputc(int);
00177
00178 // vm.c
                        seginit(void);
kvmalloc(void);
setupkvm(void);
uva2ka(pde_t*, char*);
allocuvm(pde_t*, uint, uint);
deallocuvm(pde_t*, uint, uint);
freevm(pde_t*);
inituvm(pde_t*, char*, uint);
loaduvm(pde_t*, char*, struct inode*, uint, uint);
copyuvm(pde_t*, uint);
switchuvm(struct proc*);
switchkvm(void);
copyout(pde_t*, uint, void*, uint);
clearpteu(pde_t *pgdir, char *uva);
00179 void
00180 void
00181 pde_t*
00182 char*
00183 int
00184 int
00185 void
00186 void
00187 int
00188 pde_t*
00189 void
00190 void
00191 int
00192 void
00193
00194 // number of elements in fixed-size array
00195 #define NELEM(x) (sizeof(x)/sizeof((x)[0]))
```

5.27 echo.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

• int main (int argc, char *argv[])

5.27.1 Function Documentation

5.27.1.1 main()

```
int main ( \label{eq:int_argc} \text{int } \arg c, \label{eq:char} \operatorname{char} \ \ast \ \arg v[\ ] \ )
```

Definition at line 6 of file echo.c.

```
00007 {
00008    int i;
00009
00010    for(i = 1; i < argc; i++)
00011         printf(1, "%s%s", argv[i], i+1 < argc ? " " : "\n");
00012    exit();
00013 }
```

5.28 echo.c 165

5.28 echo.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004
00005 int
00006 main(int argc, char *argv[])
00007 {
00008  int i;
00009
00010  for(i = 1; i < argc; i++)
00011     printf(1, "%s%s", argv[i], i+1 < argc ? " " : "\n");
00012  exit();
00013 }</pre>
```

5.29 echo.d File Reference

5.30 echo.d

```
Go to the documentation of this file.
```

00001 echo.o: echo.c /usr/include/stdc-predef.h types.h stat.h user.h

5.31 elf.h File Reference

Classes

- struct elfhdr
- · struct proghdr

Macros

- #define ELF_MAGIC 0x464C457FU
- #define ELF_PROG_FLAG_EXEC 1
- #define ELF_PROG_FLAG_READ 4
- #define ELF_PROG_FLAG_WRITE 2
- #define ELF_PROG_LOAD 1

5.31.1 Macro Definition Documentation

5.31.1.1 ELF_MAGIC

#define ELF_MAGIC 0x464C457FU

Definition at line 3 of file elf.h.

5.31.1.2 ELF_PROG_FLAG_EXEC

```
#define ELF_PROG_FLAG_EXEC 1
```

Definition at line 40 of file elf.h.

5.31.1.3 ELF_PROG_FLAG_READ

```
#define ELF_PROG_FLAG_READ 4
```

Definition at line 42 of file elf.h.

5.31.1.4 ELF_PROG_FLAG_WRITE

```
#define ELF_PROG_FLAG_WRITE 2
```

Definition at line 41 of file elf.h.

5.31.1.5 ELF_PROG_LOAD

```
#define ELF_PROG_LOAD 1
```

Definition at line 37 of file elf.h.

5.32 elf.h

```
00001 // Format of an ELF executable file
00002
00003 #define ELF_MAGIC 0x464C457FU // "\xspacex7FELF" in little endian
00004
00005 // File header
00006 struct elfhdr {
00007 uint magic;
00008 uchar elf[12
                       // must equal ELF_MAGIC
        uchar elf[12];
00009
        ushort type;
00010
        ushort machine;
        uint version;
uint entry;
00011
00012
00013
        uint phoff;
00014
00015
        uint flags;
00016
        ushort ehsize;
00017
        ushort phentsize;
ushort phnum;
ushort shentsize;
00018
00020
        ushort shnum;
00021
        ushort shstrndx;
00022 };
00023
00024 // Program section header
00025 struct proghdr {
00026 uint type;
```

```
00027
         uint off;
00028
         uint vaddr;
00029
         uint paddr;
00030 uint filesz;
00031 uint memsz;
00032 uint flags;
00033 uint align;
00034 };
00035
00035 // Values for Proghdr type 00037 #define ELF_PROG_LOAD
00038
00039 // Flag bits for Proghdr flags
00040 #define ELF_PROG_FLAG_EXEC
00041 #define ELF_PROG_FLAG_WRITE
00042 #define ELF_PROG_FLAG_READ
```

5.33 entryother.d File Reference

5.34 entryother.d

```
Go to the documentation of this file.
```

```
00001 entryother.o: entryother.S asm.h memlayout.h mmu.h
```

5.35 exec.c File Reference

```
#include "types.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "defs.h"
#include "x86.h"
#include "elf.h"
```

Functions

int exec (char *path, char **argv)

5.35.1 Function Documentation

5.35.1.1 exec()

```
int exec (
                char * path,
                char ** argv )
Definition at line 11 of file exec.c.
00012 {
00013
        char *s, *last;
00014
        int i, off;
00015
        uint argc, sz, sp, ustack[3+MAXARG+1];
        struct elfhdr elf;
struct inode *ip;
00016
00017
00018
        struct proghdr ph;
        pde_t *pgdir, *oldpgdir;
struct proc *curproc = myproc();
00019
00020
00021
00022
        begin_op();
00023
00024
        if((ip = namei(path)) == 0){
00025
         end_op();
00026
          cprintf("exec: fail\n");
00027
          return -1;
00028
        ilock(ip);
00029
        pgdir = 0;
00030
00031
00032
        // Check ELF header
00033
        if(readi(ip, (char*)&elf, 0, sizeof(elf)) != sizeof(elf))
00034
          goto bad;
        if(elf.magic != ELF_MAGIC)
00035
00036
         goto bad;
00037
00038
        if((pgdir = setupkvm()) == 0)
00039
          goto bad;
00040
00041
        // Load program into memory.
00042
        sz = 0:
00043
        for(i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){</pre>
00044
          if(readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
00045
             goto bad;
00046
          if(ph.type != ELF_PROG_LOAD)
00047
             continue;
00048
          if(ph.memsz < ph.filesz)</pre>
00049
            goto bad;
00050
          if (ph.vaddr + ph.memsz < ph.vaddr)</pre>
00051
00052
          if((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
00053
            goto bad;
          if (ph.vaddr % PGSIZE != 0)
00054
00055
            goto bad;
          if(loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz) < 0)</pre>
00056
00057
            goto bad;
00058
00059
        iunlockput(ip);
00060
        end_op();
ip = 0;
00061
00062
00063
        // Allocate two pages at the next page boundary.
00064
        // Make the first inaccessible. Use the second as the user stack.
00065
        sz = PGROUNDUP(sz);
00066
        if((sz = allocuvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
00067
          goto bad;
00068
        clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
00069
        sp = sz;
00070
00071
        \ensuremath{//} Push argument strings, prepare rest of stack in ustack.
        for(argc = 0; argv[argc]; argc++) {
  if(argc >= MAXARG)
00072
00073
00074
            goto bad;
00075
          sp = (sp - (strlen(argv[argc]) + 1)) & ~3;
00076
          if(copyout(pgdir, sp, argv[argc], strlen(argv[argc]) + 1) < 0)</pre>
00077
            goto bad;
00078
          ustack[3+argc] = sp;
00079
        ustack[3+argc] = 0;
08000
00081
00082
        ustack[0] = 0xfffffffff; // fake return PC
        ustack[1] = argc;
ustack[2] = sp - (argc+1)*4; // argv pointer
00083
00084
00085
00086
        sp -= (3+argc+1) * 4;
00087
        if(copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)</pre>
00088
          goto bad;
```

5.36 exec.c 169

```
00090
        // Save program name for debugging.
00091
        for(last=s=path; *s; s++)
        if(*s == '/')
00092
            last = s+1;
00093
00094
        safestrcpy(curproc->name, last, sizeof(curproc->name));
00096
        \ensuremath{//} Commit to the user image.
00097
        oldpgdir = curproc->pgdir;
00098
        curproc->pgdir = pgdir;
        curproc->sz = sz;
00099
        curproc->tf->eip = elf.entry; // main
curproc->tf->esp = sp;
00100
00101
00102
        switchuvm(curproc);
00103
        freevm(oldpgdir);
00104
        return 0;
00105
00106
       bad:
        if(pgdir)
00108
          freevm(pgdir);
00109
        if(ip){
00110
          iunlockput(ip);
00111
          end_op();
00112
00113
        return -1;
00114 }
```

Referenced by bigargtest(), exectest(), main(), runcmd(), and sys_exec().

5.36 exec.c

```
00001 #include "types.h"
00002 #include "param.h"
00003 #include "memlayout.h"
00004 #include "mmu.h"
00005 #include "proc.h"
00006 #include "defs.h"
00007 #include "x86.h"
00008 #include "elf.h"
00009
00010 int
00011 exec(char *path, char **argv)
00012 {
00013
        char *s, *last;
        int i, off;
00014
        uint argc, sz, sp, ustack[3+MAXARG+1];
00016
        struct elfhdr elf;
00017
        struct inode *ip;
00018
        struct proghdr ph;
00019
        pde_t *pgdir, *oldpgdir;
struct proc *curproc = myproc();
00020
00021
00022
        begin_op();
00023
00024
        if((ip = namei(path)) == 0){
         end_op();
cprintf("exec: fail\n");
00025
00026
00027
          return -1;
00028
        ilock(ip);
00029
00030
        pgdir = 0;
00031
00032
        // Check ELF header
        if(readi(ip, (char*)&elf, 0, sizeof(elf)) != sizeof(elf))
00033
00034
           goto bad;
00035
        if(elf.magic != ELF_MAGIC)
00036
           goto bad;
00037
00038
        if((pgdir = setupkvm()) == 0)
00039
          goto bad;
00041
         // Load program into memory.
00042
        sz = 0;
00043
        for(i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){</pre>
          if(readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
goto bad;
00044
00045
00046
           if(ph.type != ELF_PROG_LOAD)
00047
00048
           if(ph.memsz < ph.filesz)</pre>
```

```
00049
             goto bad;
00050
           if(ph.vaddr + ph.memsz < ph.vaddr)</pre>
00051
             goto bad;
           if((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
00052
             goto bad;
00053
           if (ph.vaddr % PGSIZE != 0)
00054
00055
            goto bad;
00056
           if(loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz) < 0)</pre>
00057
             goto bad;
00058
00059
        iunlockput(ip);
00060
        end_op();
00061
        ip = 0;
00062
00063
         \ensuremath{//} Allocate two pages at the next page boundary.
00064
         \ensuremath{//} Make the first inaccessible. Use the second as the user stack.
00065
        sz = PGROUNDUP(sz);
00066
        if((sz = allocuvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
          goto bad;
00067
00068
         clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
00069
00070
00071
        \ensuremath{//} Push argument strings, prepare rest of stack in ustack.
00072
        for(argc = 0; argv[argc]; argc++) {
  if(argc >= MAXARG)
00073
00074
            goto bad;
          sp = (sp - (strlen(argv[argc]) + 1)) & ~3;
if(copyout(pgdir, sp, argv[argc], strlen(argv[argc]) + 1) < 0)</pre>
00075
00076
00077
             goto bad;
00078
          ustack[3+argc] = sp;
00079
08000
        ustack[3+argc] = 0;
00081
        ustack[0] = 0xfffffffff; // fake return PC
00082
        ustack[1] = argc;
ustack[2] = sp - (argc+1)*4; // argv pointer
00083
00084
00085
        sp -= (3+argc+1) * 4;
00087
        if(copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)</pre>
88000
00089
00090
        // Save program name for debugging.
        for(last=s=path; *s; s++)
  if(*s == '/')
00091
00092
             last = s+1;
00093
00094
        safestrcpy(curproc->name, last, sizeof(curproc->name));
00095
00096
        // Commit to the user image.
00097
        oldpgdir = curproc->pgdir;
        curproc->pgdir = pgdir;
00098
00099
        curproc->sz = sz;
00100
        curproc->tf->eip = elf.entry; // main
        curproc->tf->esp = sp;
00101
00102
        switchuvm(curproc);
00103
        freevm(oldpgdir);
00104
        return 0;
00105
00106 bad:
        if(pgdir)
00107
00108
          freevm(pgdir);
        if(ip){
00109
         iunlockput(ip);
00110
00111
          end_op();
00112
00113
        return -1;
00114 }
```

5.37 exec.d File Reference

5.38 exec.d

```
00001 exec.o: exec.c /usr/include/stdc-predef.h types.h param.h memlayout.h \backslash 00002 mmu.h proc.h defs.h x86.h elf.h
```

5.39 exit.c File Reference 171

5.39 exit.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

• int main (int argc, char *argv[])

5.39.1 Function Documentation

5.39.1.1 main()

```
int main (
          int argc,
          char * argv[] )
```

Definition at line 10 of file exit.c.

```
00011 {
00012 halt();
00013 exit();
00014 }
```

5.40 exit.c

Go to the documentation of this file.

```
00001 // Shuts down the system by using the halt() system call
00002 // to send a special signal to QEMU.
00003 // Added by Bill Katsak
00004 // Copied from: http://pdos.csail.mit.edu/6.828/2012/homework/xv6-syscall.html
00005
00006 #include "types.h"
00007 #include "stat.h"
00008 #include "user.h"
00009
00010 int main(int argc, char *argv[])
00011 {
00012   halt();
00013   exit();
00014 }
```

5.41 exit.d File Reference

5.42 exit.d

```
00001 exit.o: exit.c /usr/include/stdc-predef.h types.h stat.h user.h
```

5.43 fcntl.h File Reference

Macros

- #define O CREATE 0x200
- #define O_RDONLY 0x000
- #define O_RDWR 0x002
- #define O_WRONLY 0x001

5.43.1 Macro Definition Documentation

5.43.1.1 O_CREATE

#define O_CREATE 0x200

Definition at line 4 of file fcntl.h.

5.43.1.2 O_RDONLY

#define O_RDONLY 0x000

Definition at line 1 of file fcntl.h.

5.43.1.3 O_RDWR

#define O_RDWR 0x002

Definition at line 3 of file fcntl.h.

5.43.1.4 O_WRONLY

#define O_WRONLY 0x001

Definition at line 2 of file fcntl.h.

5.44 fcntl.h 173

5.44 fcntl.h

Go to the documentation of this file.

```
00001 #define O_RDONLY 0x000
00002 #define O_WRONLY 0x001
00003 #define O_RDWR 0x002
00004 #define O_CREATE 0x200
```

file.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "fs.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "file.h"
```

Functions

```
    struct file * filealloc (void)

• void fileclose (struct file *f)
• struct file * filedup (struct file *f)

    void fileinit (void)

• int fileread (struct file *f, char *addr, int n)
```

- int filestat (struct file *f, struct stat *st)
- int filewrite (struct file *f, char *addr, int n)

Variables

```
• struct devsw devsw [NDEV]
struct {
    struct file file [NFILE]
    struct spinlock lock
 } ftable
```

5.45.1 Function Documentation

5.45.1.1 filealloc()

```
struct file * filealloc (
                 void )
Definition at line 27 of file file.c.
00029
         struct file *f;
00030
         acquire(&ftable.lock);
for(f = ftable.file; f < ftable.file + NFILE; f++){
  if(f->ref == 0) {
   f->ref = 1;
00031
00032
00033
00034
00035
              release(&ftable.lock);
00036
00037
00038
00039
         release(&ftable.lock);
00040
        return 0;
00041 }
```

Referenced by pipealloc(), and sys_open().

5.45.1.2 fileclose()

void fileclose (

```
struct file * f)
Definition at line 57 of file file.c.
00059
         struct file ff;
00060
00061
         acquire(&ftable.lock);
00062
        if(f->ref < 1)
  panic("fileclose");</pre>
00063
00064
         if(--f->ref > 0){
00065
          release(&ftable.lock);
00066
00067
        ff = *f;
f->ref = 0;
f->type = FD_NONE;
00068
00069
00070
00071
         release (&ftable.lock);
00072
00073
         if(ff.type == FD_PIPE)
        pipeclose(ff.pipe, ff.writable);
else if(ff.type == FD_INODE){
00074
00075
         begin_op();
iput(ff.ip);
00076
00077
00078
           end_op();
00079
        }
00080 }
```

Referenced by exit(), pipealloc(), sys_close(), sys_open(), and sys_pipe().

5.45.1.3 filedup()

```
struct file * filedup (
               struct file * f )
Definition at line 45 of file file.c.
00046 {
00047
        acquire(&ftable.lock);
00048
        if (f->ref < 1)</pre>
00049
          panic("filedup");
00050
        f->ref++;
00051
        release(&ftable.lock);
00052
       return f;
00053 }
```

Referenced by fork(), and sys_dup().

5.45 file.c File Reference 175

5.45.1.4 fileinit()

5.45.1.5 fileread()

```
int fileread (
          struct file * f,
          char * addr,
          int n )
```

Definition at line 97 of file file.c.

```
00098 {
00099
00100
00101
           if(f->readable == 0)
00102
           return -1;
if(f->type == FD_PIPE)
00103
           return piperead(f->pipe, addr, n);
if(f->type == FD_INODE){
00104
00105
           il(r->type == rD_INODE,;
ilock(f->ip);
if((r = readi(f->ip, addr, f->off, n)) > 0)
    f->off += r;
iunlock(f->ip);
00106
00107
00108
00109
coilOck(f
coilO return r;
00111 }
00111
00112 panic("fileread");
00113 }
```

Referenced by sys_read().

5.45.1.6 filestat()

Definition at line 84 of file file.c.

```
00085 {
00086     if(f->type == FD_INODE){
00087         ilock(f->ip);
00088         stati(f->ip, st);
00089         iunlock(f->ip);
00090         return 0;
00091    }
00092    return -1;
00093 }
```

Referenced by sys_fstat().

5.45.1.7 filewrite()

int filewrite (

```
struct file * f,
                    char * addr,
                    int n)
Definition at line 118 of file file.c.
00119
00120
           int r;
00121
00122
           if(f->writable == 0)
00123
              return -1;
           if(f->type == FD_PIPE)
00124
           return pipewrite(f->pipe, addr, n);
if(f->type == FD_INODE){
   // write a few blocks at a time to avoid exceeding
00125
00126
00127
00128
             // the maximum log transaction size, including
             // i-node, indirect block, allocation blocks,
// and 2 blocks of slop for non-aligned writes.
// this really belongs lower down, since writei()
// might be writing a device like the console.
00129
00130
00131
00132
             int max = ((MAXOPBLOCKS-1-1-2) / 2) * 512;
int i = 0;
00133
00134
             while (i < n) {
  int n1 = n - i;
  if (n1 > max)
     n1 = max;
00135
00136
00137
00138
00139
00140
                begin_op();
00141
                 ilock(f->ip);
                if ((r = writei(f->ip, addr + i, f->off, n1)) > 0)
f->off += r;
00142
00143
                 iunlock(f->ip);
00144
00145
                end_op();
00146
00147
                if(r < 0)
00148
                    break;
                if(r != n1)
  panic("short filewrite");
i += r;
00149
00150
00151
00152
00153
              return i == n ? n : -1;
00155 panic("filewrite");
00156 }
```

Referenced by sys_write().

5.45.2 Variable Documentation

5.45.2.1 devsw

struct devsw devsw[NDEV]

Definition at line 13 of file file.c.

5.45.2.2 file

struct file file[NFILE]

Definition at line 16 of file file.c.

5.46 file.c 177

5.45.2.3

```
struct { ... } ftable
```

Referenced by filealloc(), fileclose(), filedup(), and fileinit().

5.45.2.4 lock

```
struct spinlock lock
```

Definition at line 15 of file file.c.

5.46 file.c

Go to the documentation of this file.

```
00001 //
00002 // File descriptors
00003 //
00004
00004
00005 #include "types.h"
00006 #include "defs.h"
00007 #include "param.h"
00008 #include "fs.h"
00009 #include "spinlock.h"
00010 #include "sleeplock.h"
00011 #include "file.h"
00012
00013 struct devsw devsw[NDEV];
00014 struct {
00015 struct spinlock lock;
00016 struct file file[NFILE];
00017 } ftable;
00018
00019 void
00020 fileinit(void)
00021 {
00022
         initlock(&ftable.lock, "ftable");
00023 }
00024
00025 // Allocate a file structure.
00026 struct file*
00027 filealloc(void)
00028 {
00029
          struct file *f;
00030
00031
          acquire(&ftable.lock);
          for(f = ftable.file; f < ftable.file + NFILE; f++){
  if(f->ref == 0) {
   f->ref = 1;
00032
00033
00034
00035
               release(&ftable.lock);
00036
               return f;
00037
00038
00039
         release(&ftable.lock);
00040
          return 0;
00041 }
00042
00043 // Increment ref count for file f.
00044 struct file*
00045 filedup(struct file *f)
00046 {
00047 acquire(&ftable.lock);
         if(f->ref < 1)
00048
           panic("filedup");
00049
         f->ref++;
00050
00051
00052
         release(&ftable.lock);
          return f;
00053 }
00055 // Close file f. (Decrement ref count, close when reaches 0.)
```

```
00056 void
00057 fileclose(struct file *f)
00058 {
00059
        struct file ff;
00060
00061
        acquire(&ftable.lock);
       if(f->ref < 1)</pre>
00062
00063
          panic("fileclose");
00064
        if(--f->ref > 0){
        release(&ftable.lock);
00065
00066
         return:
00067
00068
       ff = *f;
00069
        f \rightarrow ref = 0;
00070
        f->type = FD_NONE;
00071
        release(&ftable.lock);
00072
00073
        if(ff.type == FD PIPE)
         pipeclose(ff.pipe, ff.writable);
00074
00075
        else if(ff.type == FD_INODE){
        begin_op();
00076
00077
          iput(ff.ip);
00078
         end_op();
00079
00080 }
00081
00082 // Get metadata about file f.
00083 int
00084 filestat(struct file *f, struct stat *st)
00085 {
00086
       if(f->type == FD_INODE) {
00087
         ilock(f->ip);
00088
          stati(f->ip, st);
00089
          iunlock(f->ip);
00090
         return 0;
00091
00092
       return -1;
00094
00095 // Read from file f.
00096 int
00097 fileread(struct file *f, char *addr, int n)
00098 {
00099
        int r;
00100
00101
       if(f->readable == 0)
       return -1;
if(f->type == FD_PIPE)
00102
00103
       return piperead(f->pipe, addr, n);
if(f->type == FD_INODE){
00104
00105
        ilock(f->ip);
if((r = readi(f->ip, addr, f->off, n)) > 0)
00106
00107
00108
            f->off += r;
         iunlock(f->ip);
00109
00110
          return r;
00111
00112
       panic("fileread");
00113 }
00114
00115 //PAGEBREAK!
00116 // Write to file f.
00117 int
00118 filewrite(struct file *f, char *addr, int n)
00119 {
00120
        int r;
00121
00122
        if(f->writable == 0)
00123
          return -1:
        if(f->type == FD_PIPE)
00124
          return pipewrite(f->pipe, addr, n);
00125
00126
        if(f->type == FD_INODE){
00127
         // write a few blocks at a time to avoid exceeding
          \ensuremath{//} the maximum log transaction size, including
00128
          // i-node, indirect block, allocation blocks,
00129
          // and 2 blocks of slop for non-aligned writes.
00130
00131
          // this really belongs lower down, since writei()
00132
          // might be writing a device like the console.
00133
          int max = ((MAXOPBLOCKS-1-1-2) / 2) * 512;
          int i = 0;
00134
          while(i < n){</pre>
00135
           int n1 = n - i;
00136
            if(n1 > max)
00137
00138
              n1 = max;
00139
00140
            begin_op();
00141
            ilock(f->ip);
00142
            if ((r = writei(f->ip, addr + i, f->off, n1)) > 0)
```

5.47 file.d File Reference 179

5.47 file.d File Reference

5.48 file.d

```
Go to the documentation of this file.
00001 file.o: file.c /usr/include/stdc-predef.h types.h defs.h param.h fs.h \
00002 spinlock.h sleeplock.h file.h
```

5.49 file.h File Reference

Classes

- struct devsw
- struct file
- struct inode

Macros

• #define CONSOLE 1

Variables

• struct devsw devsw []

5.49.1 Macro Definition Documentation

5.49.1.1 CONSOLE

#define CONSOLE 1

Definition at line 37 of file file.h.

5.49.2 Variable Documentation

5.49.2.1 devsw

```
struct devsw devsw[] [extern]
```

Definition at line 13 of file file.c.

5.50 file.h

Go to the documentation of this file.

```
00001 struct file {
00002 enum { FD_NONE, FD_PIPE, FD_INODE } type;
         int ref; // reference count
00004
         char readable;
00005
         char writable;
00006    struct pipe *pipe;
00007    struct inode *ip;
00008    uint off;
00009 };
00010
00011
00012 // in-memory copy of an inode 00013 struct inode {
00014 uint dev;
00015 uint inum;
                                   // Device number
                                  // Inode number
// Reference count
         uint inum;
00016
        struct sleeplock lock; // protects everything below here int valid; // inode has been read from disk?
00017
00018
00019
00020
         short type;
                                 // copy of disk inode
00021
         short major;
00022
         short minor;
00023
         short nlink;
00024 uint size;
         uint addrs[NDIRECT+1];
00025
00026 };
00028 // table mapping major device number to
00029 // device functions
00030 struct devsw {
00031 int (*read)(struct inode*, char*, int);
00032 int (*write)(struct inode*, char*, int);
00033 };
00035 extern struct devsw devsw[];
00036
00037 #define CONSOLE 1
```

5.51 foo.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
```

Functions

• int main (int argc, char *argv[])

5.52 foo.c 181

5.51.1 Function Documentation

5.51.1.1 main()

```
int main (
               int argc,
               char * argv[] )
Definition at line 6 of file foo.c.
00006
00007
        int pid;
        int k, n;
80000
00009
        int x, z;
00010
       if (argc != 2) {
   printf(2, "usage: %s n\n", argv[0]);
}
00011
00012
00013
00014
00015
        n = atoi(argv[1]);
00016
       for ( k = 0; k < n; k++ ) {
  pid = fork ();
  if ( pid < 0 ) {
    printf(1, "%d failed in fork!\n", getpid());</pre>
00017
00018
00019
00020
00021
            exit();
         } else if (pid == 0) {
   // child
00022
00023
            00024
00025
00026
00028
                printf(1, "[pid=%d] %d \n",getpid(),z);
00029
00030
            }
00031
00032
            exit();
00033
          }
00034 }
00035
00036
       wait();

       for (k = 0; k < n; k++) {
00037
00038
00039
00040
       exit();
00041 }
```

5.52 foo.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00002 #Include stat.n
00003 #include "user.h"
00004 #include "fcntl.h"
00005
00006 int main(int argc, char *argv[]) { 00007 int pid; 00008 int k, n;
00009
            int x, z;
00010
            if (argc != 2) {
   printf(2, "usage: %s n\n", argv[0]);
}
00011
00012
00013
00014
00015
            n = atoi(argv[1]);
00016
            for ( k = 0; k < n; k++ ) {
  pid = fork ();
  if ( pid < 0 ) {
    printf(1, "%d failed in fork!\n", getpid());</pre>
00017
00018
00019
00020
               exit();
} else if (pid == 0) {
00021
00022
```

```
00024
00025
00026
00027
00028
00030
00031
00032
        exit();
00033
      }
00034 }
00035
wait();
00038
00039
00036 for (k = 0; k < n; k++) {
00039
00040 exit();
00041 }
```

5.53 foo.d File Reference

5.54 foo.d

```
Go to the documentation of this file.
```

00001 foo.o: foo.c /usr/include/stdc-predef.h types.h stat.h user.h fcntl.h

5.55 forktest.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Macros

#define N 1000

Functions

- void forktest (void)
- int main (void)
- void printf (int fd, const char *s,...)

5.55.1 Macro Definition Documentation

5.55.1.1 N

#define N 1000

Definition at line 8 of file forktest.c.

5.55.2 Function Documentation

5.55.2.1 forktest()

```
void forktest (
     void )
```

Definition at line 17 of file forktest.c.

```
00019
                                         int n, pid;
00020
                                       printf(1, "fork test\n");
00021
00022
                                        for (n=0; n<N; n++) {</pre>
00023
                                       pid = fork();
if (pid < 0)
 00024
00025
                                            break;
if (pid == 0)
00026
00027
00028
                                                             exit();
00029
00030
00031
                                         if(n == N) {
                                         printf(1, "fork claimed to work N times!\n", N);
00032
00033
00034
                                                   exit();
00035
                                        for(; n > 0; n--) {
  if(wait() < 0) {
     rest f(); n = 0;
     rest f(); n = 0;

 00036
 00037
                                              printf(1, "wait stopped early\n");
exit();
 00038
00039
                                        }
00040
00041
 00042
                                       if(wait() != -1) {
  printf(1, "wait got too many\n");
  evit():
 00043
 00044
                                     exit();
}
00045
00046
00047
00047
00048 printf(1, "fork test OK\n");
00049 }
```

Referenced by main().

5.55.2.2 main()

```
int main (
     void )
```

Definition at line 52 of file forktest.c.

```
00053 {
00054 forktest();
00055 exit();
00056 }
```

5.55.2.3 printf()

00012 {
00013 write(fd, s, strlen(s));
00014 }

Referenced by argptest(), balloc(), bigargtest(), bigdir(), bigdir(), bigwrite(), bsstest(), cat(), concreate(), createdelete(), createdelete(), dirfile(), dirtest(), exectest(), exitiputtest(), exitwait(), forktest(), fourfiles(), fourteen(), fsfull(), getcmd(), iputtest(), iref(), linktest(), linkunlink(), ls(), main(), mem(), openiputtest(), opentest(), panic(), parsecmd(), pipe1(), preempt(), rmdot(), runcmd(), sbrktest(), sharedfd(), subdir(), uio(), unlinkread(), validatetest(), wc(), writetest(), and writetest1().

5.56 forktest.c

Go to the documentation of this file.

```
00001 // Test that fork fails gracefully.
00002 // Tiny executable so that the limit can be filling the proc table.
00003
00004 #include "types.h"
00005 #include "stat.h
00006 #include "user.h"
00007
00008 #define N 1000
00009
00010 void
00011 printf(int fd, const char *s, ...)
00012 {
00013
        write(fd, s, strlen(s));
00014 }
00015
00016 void
00017 forktest (void)
00018 {
00019
        int n, pid;
00020
00021
       printf(1, "fork test\n");
00022
00023
        for (n=0; n<N; n++) {</pre>
         pid = fork();
00024
00025
          if (pid < 0)
00026
            break;
          if(pid == 0)
00027
00028
            exit();
00029
00030
00031
        if(n == N) {
00032
         printf(1, "fork claimed to work N times!\n", N);
00033
          exit();
00034
00035
        for(; n > 0; n--) {
00036
         if(wait() < 0){</pre>
00038
            printf(1, "wait stopped early\n");
00039
            exit();
00040
00041
00042
00043
        if (wait() != -1) {
00044
         printf(1, "wait got too many\n");
00045
00046
00047
       printf(1, "fork test OK\n");
00048
00049 }
00050
00051 int
00052 main(void)
00053 {
00054
        forktest();
00055
       exit();
00056 }
```

5.57 forktest.d File Reference

5.58 forktest.d

Go to the documentation of this file.

```
00001 forktest.o: forktest.c /usr/include/stdc-predef.h types.h stat.h user.h
```

5.59 fs.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "stat.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "fs.h"
#include "buf.h"
#include "file.h"
```

Macros

#define min(a, b) ((a) < (b) ? (a) : (b))

Functions

- · static uint balloc (uint dev)
- static void bfree (int dev, uint b)
- static uint bmap (struct inode *ip, uint bn)
- static void bzero (int dev, int bno)
- int dirlink (struct inode *dp, char *name, uint inum)
- struct inode * dirlookup (struct inode *dp, char *name, uint *poff)
- struct inode * ialloc (uint dev, short type)
- struct inode * idup (struct inode *ip)
- static struct inode * iget (uint dev, uint inum)
- void iinit (int dev)
- void ilock (struct inode *ip)
- void iput (struct inode *ip)
- static void itrunc (struct inode *)
- void iunlock (struct inode *ip)
- void iunlockput (struct inode *ip)
- void iupdate (struct inode *ip)
- int namecmp (const char *s, const char *t)
- struct inode * namei (char *path)
- struct inode * nameiparent (char *path, char *name)
- static struct inode * namex (char *path, int nameiparent, char *name)
- int readi (struct inode *ip, char *dst, uint off, uint n)
- void readsb (int dev, struct superblock *sb)
- static char * skipelem (char *path, char *name)
- void stati (struct inode *ip, struct stat *st)
- int writei (struct inode *ip, char *src, uint off, uint n)

Variables

```
    struct {
        struct inode inode [NINODE]
        struct spinlock lock
        } icache
```

· struct superblock sb

5.59.1 Macro Definition Documentation

5.59.1.1 min

Definition at line 24 of file fs.c.

5.59.2 Function Documentation

5.59.2.1 balloc()

Definition at line 57 of file fs.c.

```
00058 {
              int b, bi, m;
struct buf *bp;
00059
00060
00061
00063
               for(b = 0; b < sb.size; b += BPB) {</pre>
               bror(D = U; D < sb.size; b += BPB) {
    bp = bread(dev, BBLOCK(b, sb));
    for(bi = 0; bi < BPB && b + bi < sb.size; bi++) {
        m = 1 « (bi % 8);
        if((bp->data[bi/8] & m) == 0) { // Is block free?
        bp->data[bi/8] |= m; // Mark block in use.
        log write(bp):
00064
00065
00066
00067
00068
00069
                           log_write(bp);
                          brelse(bp);
bzero(dev, b + bi);
return b + bi;
00070
00071
00072
00073
00074
00075
                   brelse(bp);
00077 panic("balloc: out of blocks");
00078 }
00076
```

Referenced by bmap().

5.59 fs.c File Reference 187

5.59.2.2 bfree()

```
static void bfree (
                int dev,
                uint b ) [static]
```

Definition at line 82 of file fs.c.

```
00083 {
         struct buf *bp;
00085
        int bi, m;
00086
00087
        bp = bread(dev, BBLOCK(b, sb));
        bi = b % BPB;
m = 1 « (bi % 8);
00088
00089
00090
        if((bp->data[bi/8] \& m) == 0)
        panic("freeing free block");
bp->data[bi/8] &= ~m;
00091
00092
00093
         log_write(bp);
00094 brelse(bp);
00095 }
```

Referenced by itrunc().

5.59.2.3 bmap()

Definition at line 373 of file fs.c.

```
00374 {
           uint addr, *a;
00376
           struct buf *bp;
00377
00378
           if(bn < NDIRECT) {</pre>
             if((addr = ip->addrs[bn]) == 0)
  ip->addrs[bn] = addr = balloc(ip->dev);
00379
00380
00381
             return addr;
00382
00383
          bn -= NDIRECT;
00384
00385
           if (bn < NINDIRECT) {</pre>
            // Load indirect block, allocating if necessary.
if((addr = ip->addrs[NDIRECT]) == 0)
00386
00387
00388
                ip->addrs[NDIRECT] = addr = balloc(ip->dev);
             bp = bread(ip->dev, addr);
a = (uint*)bp->data;
if((addr = a[bn]) == 0){
   a[bn] = addr = balloc(ip->dev);
   log_write(bp);
00389
00390
00391
00392
00393
00394
00395
             brelse(bp);
00396
              return addr;
00397
00399 panic("bmap: out of range");
00400 }
00398
```

Referenced by readi(), and writei().

5.59.2.4 bzero()

```
static void bzero (
                int dev,
                int bno ) [static]
Definition at line 43 of file fs.c.
00044 {
        struct buf *bp;
00046
00047
        bp = bread(dev, bno);
        memset(bp->data, 0, BSIZE);
log_write(bp);
00048
```

Referenced by balloc(), ialloc(), and main().

5.59.2.5 dirlink()

00050 brelse(bp);

00049

00561 00562 00563

00569

00570

00578 }

00051 }

```
int dirlink (
               struct inode * dp,
              char * name,
               uint inum )
Definition at line 552 of file fs.c.
00553 {
00554
        int off;
00555
        struct dirent de;
00556
        struct inode *ip;
00557
00558
       // Check that name is not present.
00559
        if((ip = dirlookup(dp, name, 0)) != 0){
        iput(ip);
return -1;
00560
```

00571 00572 strncpy(de.name, name, DIRSIZ); 00573 de.inum = inum; if(writei(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
panic("dirlink"); 00574 00575 00576 00577 return 0;

// Look for an empty dirent.
for(off = 0; off < dp->size; off += sizeof(de)) {
 if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
 panic("dirlink read");
}

Referenced by create(), and sys_link().

if(de.inum == 0)

break;

5.59 fs.c File Reference 189

5.59.2.6 dirlookup()

```
struct inode * dirlookup (
                struct inode * dp,
                char * name,
                uint * poff )
Definition at line 525 of file fs.c.
00526 {
00527
         uint off, inum;
00528
         struct dirent de;
00529
        if (dp->type != T_DIR)
  panic("dirlookup not DIR");
00530
00532
00533
        for(off = 0; off < dp->size; off += sizeof(de)){
         if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
panic("dirlookup read");
if(de.inum == 0)
00534
00535
00536
00537
             continue;
00538
           if(namecmp(name, de.name) == 0) {
00539
            // entry matches path element
00540
             if(poff)
              *poff = off;
00541
             inum = de.inum;
00542
00543
             return iget(dp->dev, inum);
00544
00545
00546
00547
        return 0;
00548 }
```

Referenced by create(), dirlink(), namex(), and sys_unlink().

5.59.2.7 ialloc()

Definition at line 195 of file fs.c.

```
00196 {
00197
         int inum;
00198
         struct buf *bp;
         struct dinode *dip;
00199
00200
00201
         for(inum = 1; inum < sb.ninodes; inum++) {</pre>
00202
          bp = bread(dev, IBLOCK(inum, sb));
00203
            dip = (struct dinode*)bp->data + inum%IPB;
            if (dip->type == 0) {    // a free inode
    memset(dip, 0, sizeof(*dip));
    dip->type = type;
    log_write(bp);    // mark it allocated on the disk
00204
00205
00206
00207
00208
               brelse(bp);
00209
               return iget(dev, inum);
00210
00211
            brelse(bp);
00212
         panic("ialloc: no inodes");
00213
```

Referenced by create().

5.59.2.8 idup()

Referenced by fork(), and namex().

5.59.2.9 iget()

Definition at line 242 of file fs.c.

```
00244
         struct inode *ip, *empty;
00245
00246
         acquire(&icache.lock);
00247
00248
         // Is the inode already cached?
00249
         empty = 0;
00250
         for(ip = &icache.inode[0]; ip < &icache.inode[NINODE]; ip++){</pre>
           if(ip->ref > 0 && ip->dev == dev && ip->inum == inum) {
00251
            ip->ref++;
release(&icache.lock);
00252
00253
00254
             return ip;
00255
00256
           if(empty == 0 && ip->ref == 0) // Remember empty slot.
00257
              empty = ip;
00258
00259
        // Recycle an inode cache entry.
if(empty == 0)
  panic("iget: no inodes");
00260
00261
00262
00263
00264
        ip = empty;
        ip->dev = dev;
ip->inum = inum;
00265
00266
00267
         ip->ref = 1;
        ip->valid = 0;
release(&icache.lock);
00268
00269
00270
00271
00272 }
         return ip;
```

Referenced by dirlookup(), ialloc(), and namex().

5.59 fs.c File Reference 191

5.59.2.10 iinit()

```
void iinit (
                 int dev )
Definition at line 172 of file fs.c.
00173
00174
         int i = 0;
00175
00176
         initlock(&icache.lock, "icache");
00177
         for(i = 0; i < NINODE; i++) {</pre>
          initsleeplock(&icache.inode[i].lock, "inode");
00178
00179
00180
        readsb(dev, &sb);
00182
        cprintf("sb: size %d nblocks %d ninodes %d nlog %d logstart %d\
00183 inodestart %d bmap start %d\n", sb.size, sb.nblocks, 00184 sb.ninodes, sb.nlog, sb.logstart, sb.inodestart,
00185
                  sb.bmapstart);
00186 }
```

Referenced by forkret().

5.59.2.11 ilock()

void ilock (

```
struct inode * ip )
Definition at line 288 of file fs.c.
00289 {
00290
         struct buf *bp;
00291
         struct dinode *dip;
00292
         if(ip == 0 || ip->ref < 1)
  panic("ilock");</pre>
00293
00294
00295
00296
         acquiresleep(&ip->lock);
00297
00298
         if(ip->valid == 0){
00299
           bp = bread(ip->dev, IBLOCK(ip->inum, sb));
00300
            dip = (struct dinode*)bp->data + ip->inum%IPB;
00301
           ip->type = dip->type;
           ip->cype = dip->cype,
ip->major = dip->major;
ip->minor = dip->minor;
00302
00303
00304
           ip->nlink = dip->nlink;
00305
           ip->size = dip->size;
00306
            memmove(ip->addrs, dip->addrs, sizeof(ip->addrs));
           brelse(bp);
ip->valid = 1;
00307
00308
           if (ip->type == 0)
panic("ilock: no type");
00309
00310
00311
00312 }
```

Referenced by consoleread(), consolewrite(), create(), exec(), fileread(), filestat(), filewrite(), namex(), sys_chdir(), sys_link(), sys_open(), and sys_unlink().

5.59.2.12 iput()

```
void iput (
               struct inode * ip )
Definition at line 332 of file fs.c.
00333 4
00334
        acquiresleep(&ip->lock);
00335
        if(ip->valid && ip->nlink == 0) {
00336
          acquire(&icache.lock);
00337
          int r = ip -> ref;
          release(&icache.lock);
00338
          if(r == 1){
   // inode has no links and no other references: truncate and free.
00339
00340
00341
             itrunc(ip);
00342
            ip->type = 0;
00343
            iupdate(ip);
            ip->valid = 0;
00344
00345
          }
00346
00347
       releasesleep(&ip->lock);
00348
00349
        acquire(&icache.lock);
00350
        ip->ref--;
00351
       release(&icache.lock);
00352 }
```

Referenced by dirlink(), exit(), fileclose(), iunlockput(), namex(), sys_chdir(), and sys_link().

5.59.2.13 itrunc()

```
static void itrunc ( {\tt struct\ inode\ *\ ip\ )} \quad [{\tt static}]
```

Definition at line 408 of file fs.c.

```
00409 {
         int i, j;
struct buf *bp;
00410
00411
00412
         uint *a;
00413
00414
         for(i = 0; i < NDIRECT; i++) {</pre>
         if(ip->addrs[i]){
00415
             bfree(ip->dev, ip->addrs[i]);
ip->addrs[i] = 0;
00416
00417
00418
           }
00419
00420
00421
         if (ip->addrs[NDIRECT]) {
           bp = bread(ip->dev, ip->addrs[NDIRECT]);
a = (uint*)bp->data;
00422
00423
           for(j = 0; j < NINDIRECT; j++) {
   if(a[j])</pre>
00424
00425
00426
                bfree(ip->dev, a[j]);
00427
00428
            brelse(bp);
           bfree(ip->dev, ip->addrs[NDIRECT]);
ip->addrs[NDIRECT] = 0;
00429
00430
00431
00432
00433
         ip->size = 0;
00434
         iupdate(ip);
00435 }
```

Referenced by iput().

5.59 fs.c File Reference 193

5.59.2.14 iunlock()

Referenced by consoleread(), consolewrite(), fileread(), filestat(), filewrite(), iunlockput(), namex(), sys_chdir(), sys_link(), and sys_open().

5.59.2.15 iunlockput()

Referenced by create(), exec(), namex(), $sys_chdir()$, $sys_link()$, $sys_mkdir()$, $sys_mknod()$, $sys_open()$, and $sys_unlink()$.

5.59.2.16 iupdate()

00360 }

```
void iupdate ( {\tt struct\ inode\ *\ ip\ )}
```

Definition at line 221 of file fs.c.

```
00223
         struct buf *bp;
00224
         struct dinode *dip;
00225
         bp = bread(ip->dev, IBLOCK(ip->inum, sb));
00226
00227
         dip = (struct dinode*)bp->data + ip->inum%IPB;
         dip->type = ip->type;
dip->major = ip->major;
00228
00229
         dip >major = ip >major,
dip->minor = ip->minor;
dip->nlink = ip->nlink;
dip->size = ip->size;
00230
00231
00232
00233
          memmove(dip->addrs, ip->addrs, sizeof(ip->addrs));
00234
          log_write(bp);
00235
         brelse(bp);
00236 }
```

Referenced by create(), iput(), itrunc(), sys_link(), sys_unlink(), and writei().

5.59.2.17 namecmp()

Referenced by dirlookup(), and sys unlink().

5.59.2.18 namei()

00664 }

Referenced by exec(), sys_chdir(), sys_link(), sys_open(), and userinit().

5.59.2.19 nameiparent()

Definition at line 667 of file fs.c.

Referenced by create(), namex(), sys_link(), and sys_unlink().

5.59 fs.c File Reference 195

5.59.2.20 namex()

```
static struct inode * namex (
               char * path,
               int nameiparent,
               char * name ) [static]
Definition at line 625 of file fs.c.
00626
00627
        struct inode *ip, *next;
00628
00629
        if(*path == '/')
         ip = iget(ROOTDEV, ROOTINO);
00630
        else
00631
00632
          ip = idup(myproc()->cwd);
00633
       while((path = skipelem(path, name)) != 0){
00634
         ilock(ip);
00635
00636
          if(ip->type != T_DIR) {
00637
           iunlockput(ip);
00638
            return 0;
00639
00640
          if(nameiparent && *path == ' \setminus 0') {
00641
            // Stop one level early.
00642
            iunlock(ip);
00643
            return ip;
00644
          if((next = dirlookup(ip, name, 0)) == 0){
00645
00646
            iunlockput(ip);
00647
            return 0;
00648
00649
          iunlockput(ip);
00650
          ip = next;
00651
00652
        if (nameiparent) {
00653
         iput(ip);
00654
          return 0;
00655
00656
        return ip;
00657 }
```

Referenced by namei(), and nameiparent().

5.59.2.21 readi()

int readi (

```
struct inode * ip,
                char * dst,
                uint off,
                uint n )
Definition at line 453 of file fs.c.
00454 {
00455
         uint tot, m;
00456
        struct buf *bp;
00457
         if(ip->type == T_DEV) {
00458
          if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].read)
00459
00460
00461
           return devsw[ip->major].read(ip, dst, n);
00462
00463
        if(off > ip->size || off + n < off)
  return -1;</pre>
00464
00465
00466
         if(off + n > ip->size)
00467
          n = ip -> size - off;
00468
00469
        for(tot=0; tot<n; tot+=m, off+=m, dst+=m) {</pre>
          bp = bread(ip->dev, bmap(ip, off/BSIZE));
m = min(n - tot, BSIZE - off%BSIZE);
00470
00471
```

memmove(dst, bp->data + off%BSIZE, m);

00472

```
00473 brelse(bp);
00474 }
00475 return n;
00476 }
```

Referenced by dirlink(), dirlookup(), exec(), fileread(), isdirempty(), and loaduvm().

5.59.2.22 readsb()

00036 00037 00038

00039 }

bp = bread(dev, 1);
memmove(sb, bp->data, sizeof(*sb));

Referenced by iinit(), and initlog().

brelse(bp);

5.59.2.23 skipelem()

Definition at line 596 of file fs.c.

```
00597 {
00598
        char *s;
00599
        int len;
00600
00601
        while(*path == '/')
00602
         path++;
        if(*path == 0)
00603
        return 0;
s = path;
00604
00605
        while (*path != '/' && *path != 0)
00606
00607
          path++;
        len = path - s;
if (len >= DIRSIZ)
00608
00609
          memmove(name, s, DIRSIZ);
00610
        else {
00611
        memmove(name, s, len);
name[len] = 0;
00612
00613
00614
00615
        while(*path == '/')
00616
         path++;
00617
        return path;
00618 }
```

Referenced by namex().

5.59 fs.c File Reference 197

5.59.2.24 stati()

Referenced by filestat().

5.59.2.25 writei()

```
int writei (
          struct inode * ip,
          char * src,
          uint off,
          uint n )
```

Definition at line 482 of file fs.c.

```
00483 {
00484
        uint tot, m;
00485
        struct buf *bp;
00486
        if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].write)
00488
00489
            return -1;
          return devsw[ip->major].write(ip, src, n);
00490
00491
00492
        if(off > ip->size || off + n < off)</pre>
       return -1;
if (off + n > MAXFILE*BSIZE)
00494
00495
00496
        return -1;
00497
00498
       for(tot=0; tot<n; tot+=m, off+=m, src+=m) {</pre>
       bp = bread(ip->dev, bmap(ip, off/BSIZE));
m = min(n - tot, BSIZE - off%BSIZE);
00499
00500
00501
         memmove(bp->data + off%BSIZE, src, m);
00502
          log_write(bp);
00503
          brelse(bp);
00504
       }
00505
00506
        if(n > 0 && off > ip->size) {
        ip->size = off;
00507
00508
          iupdate(ip);
00509
        }
00510
        return n;
00511 }
```

Referenced by dirlink(), filewrite(), and sys_unlink().

5.59.3 Variable Documentation

5.59.3.1

```
struct { ... } icache
```

Referenced by idup(), iget(), iinit(), and iput().

5.59.3.2 inode

```
struct inode inode[NINODE]
```

Definition at line 168 of file fs.c.

5.59.3.3 lock

```
struct spinlock lock
```

Definition at line 167 of file fs.c.

5.59.3.4 sb

```
struct superblock sb
```

Definition at line 28 of file fs.c.

Referenced by balloc(), bfree(), cmostime(), ialloc(), iinit(), ilock(), initlog(), iupdate(), and readsb().

5.60 fs.c

Go to the documentation of this file.

```
00001 // File system implementation. Five layers: 00002 // + Blocks: allocator for raw disk blocks.
                + Log: crash recovery for multi-step updates.
00003 //
                + Files: inode allocator, reading, writing, metadata.
+ Directories: inode with special contents (list of other inodes!)
+ Names: paths like /usr/rtm/xv6/fs.c for convenient naming.
00005 //
00006 //
00007 //
00008 // This file contains the low-level file system manipulation 00009 // routines. The (higher-level) system call implementations
00010 // are in sysfile.c.
00011
00012 #include "types.h"
00013 #include "defs.h"
00015 #include "param.h"
00015 #include "stat.h"
00016 #include "mmu.h"
00017 #include "proc.h"
00018 #include "spinlock.h"
00019 #include "sleeplock.h"
00020 #include "fs.h"
00021 #include "buf.h"
00022 #include "file.h"
00023
```

5.60 fs.c 199

```
00024 \#define min(a, b) ((a) < (b) ? (a) : (b))
00025 static void itrunc(struct inode*);
00026 // there should be one superblock per disk device, but we run with
00027 // only one device
00028 struct superblock sb;
00029
00030 // Read the super block.
00031 void
00032 readsb(int dev, struct superblock *sb)
00033 {
00034
        struct buf *bp;
00035
00036
        bp = bread(dev, 1);
00037
        memmove(sb, bp->data, sizeof(*sb));
00038
       brelse(bp);
00039 }
00040
00041 // Zero a block.
00042 static void
00043 bzero(int dev, int bno)
00044 {
00045
        struct buf *bp;
00046
        bp = bread(dev, bno);
00047
00048
        memset(bp->data, 0, BSIZE);
        log_write(bp);
00050 brelse(bp);
00051 }
00052
00053 // Blocks.
00054
00055 // Allocate a zeroed disk block.
00056 static uint
00057 balloc(uint dev)
00058 {
        int b, bi, m;
00059
00060
        struct buf *bp;
00061
00062
        for(b = 0; b < sb.size; b += BPB) {
   bp = bread(dev, BBLOCK(b, sb));
   for(bi = 0; bi < BPB && b + bi < sb.size; bi++) {</pre>
00063
00064
00065
            m = 1 \ll (bi \% 8);
00066
             if((bp->data[bi/8] & m) == 0){ // Is block free?
00067
00068
              bp->data[bi/8] |= m; // Mark block in use.
00069
               log_write(bp);
00070
               brelse(bp);
               bzero(dev, b + bi);
return b + bi;
00071
00072
00073
00074
00075
          brelse(bp);
00076
00077
       panic("balloc: out of blocks");
00078 }
00079
00080 // Free a disk block.
00081 static void
00082 bfree(int dev, uint b)
00083 {
        struct buf *bp;
00084
00085
        int bi, m;
00086
00087
        bp = bread(dev, BBLOCK(b, sb));
        bi = b % BPB;
00088
        m = 1 « (bi % 8);
00089
00090
        if((bp->data[bi/8] \& m) == 0)
          panic("freeing free block");
00091
00092
        bp->data[bi/8] &= ~m;
        log_write(bp);
00093
00094 brelse(bp);
00095 }
00096
00097 // Inodes.
00098 //
00099 // An inode describes a single unnamed file.
00100 // The inode disk structure holds metadata: the file's type,
00101 // its size, the number of links referring to it, and the
00102 // list of blocks holding the file's content.
00103 //
00104 // The inodes are laid out sequentially on disk at 00105 // sb.startinode. Each inode has a number, indicating its
00106 // position on the disk.
00107 //
00108 \!\!\!// The kernel keeps a cache of in-use inodes in memory
00109 // to provide a place for synchronizing access 00110 // to inodes used by multiple processes. The cached
```

```
00111 // inodes include book-keeping information that is
00112 // not stored on disk: ip->ref and ip->valid.
00113 //
00114 ^{\prime\prime} An inode and its in-memory representation go through a 00115 ^{\prime\prime} sequence of states before they can be used by the 00116 ^{\prime\prime} rest of the file system code.
00117 //
00118 // * Allocation: an inode is allocated if its type (on disk)
00119 //
          is non-zero. ialloc() allocates, and iput() frees if
00120 //
           the reference and link counts have fallen to zero.
00121 //
00122 // \star Referencing in cache: an entry in the inode cache
00123 //
           is free if ip->ref is zero. Otherwise ip->ref tracks
            the number of in-memory pointers to the entry (open
00124 //
00125 //
            files and current directories). iget() finds or
00126 //
            creates a cache entry and increments its ref; iput()
00127 //
           decrements ref.
00128 //
00129 // * Valid: the information (type, size, &c) in an inode
00130 //
           cache entry is only correct when ip->valid is 1.
00131 //
            ilock() reads the inode from
00132 //
            the disk and sets ip->valid, while iput() clears
           ip->valid if ip->ref has fallen to zero.
00133 //
00134 //
00135 // * Locked: file system code may only examine and modify
         the information in an inode and its content if it
00136 //
00137 //
           has first locked the inode.
00138 //
00139 // Thus a typical sequence is:
          ip = iget(dev, inum)
00140 //
00141 //
           ilock(ip)
00142 //
            ... examine and modify ip->xxx ...
00143 //
            iunlock(ip)
00144 //
           iput(ip)
00145 //
00146 // ilock() is separate from iget() so that system calls can
00147 // get a long-term reference to an inode (as for an open file)
00148 // and only lock it for short periods (e.g., in read()).
00149 // The separation also helps avoid deadlock and races during
00150 // pathname lookup. iget() increments ip->ref so that the inode
00151 // stays cached and pointers to it remain valid.
00152 //
00153 \!\!\!// Many internal file system functions expect the caller to
00154 // have locked the inodes involved; this lets callers create
00155 // multi-step atomic operations.
00156 //
00157 // The icache.lock spin-lock protects the allocation of icache
00158 \!\!\!// entries. Since ip->ref indicates whether an entry is free,
00159 // and ip->dev and ip->inum indicate which i-node an entry
00160 // holds, one must hold icache.lock while using any of those fields.
00161 //
00162 // An ip->lock sleep-lock protects all ip-> fields other than ref,
00163 // dev, and inum. One must hold ip->lock in order to 00164 // read or write that inode's ip->valid, ip->size, ip->type, &c.
00165
00166 struct {
00167 struct spinlock lock;
00168
        struct inode inode[NINODE];
00169 } icache;
00170
00171 void
00172 iinit(int dev)
00173 {
00174
00175
00176
        initlock(&icache.lock, "icache");
        for(i = 0; i < NINODE; i++) {</pre>
00177
          initsleeplock(&icache.inode[i].lock, "inode");
00178
00179
00180
00181
        readsb(dev, &sb);
00182
        cprintf("sb: size %d nblocks %d ninodes %d nlog %d logstart %d\
00183 inodestart %d bmap start %d\n", sb.size, sb.nblocks,
                sb.ninodes, sb.nlog, sb.logstart, sb.inodestart,
00184
00185
                sb.bmapstart);
00186 }
00187
00188 static struct inode* iget(uint dev, uint inum);
00189
00190 //PAGEBREAK!
00191 // Allocate an inode on device dev.
00192 // Mark it as allocated by giving it type type.
00193 // Returns an unlocked but allocated and referenced inode.
00194 struct inode*
00195 ialloc(uint dev, short type)
00196 {
00197
       int inum:
```

5.60 fs.c 201

```
struct buf *bp;
00199
         struct dinode *dip;
00200
         for(inum = 1; inum < sb.ninodes; inum++){
  bp = bread(dev, IBLOCK(inum, sb));
  dip = (struct dinode*)bp->data + inum%IPB;
  if(dip->type == 0){ // a free inode
00201
00202
00203
00205
             memset(dip, 0, sizeof(*dip));
00206
             dip->type = type;
             log_write(bp);
                                // mark it allocated on the disk
00207
             brelse(bp);
00208
00209
             return iget(dev, inum);
00210
00211
           brelse(bp);
00212
00213
        panic("ialloc: no inodes");
00214 }
00215
00216 // Copy a modified in-memory inode to disk.
00217 // Must be called after every change to an ip->xxx field 00218 // that lives on disk, since i-node cache is write-through.
00219 // Caller must hold ip->lock.
00220 void
00221 iupdate(struct inode *ip)
00222 {
00223
        struct buf *bp;
00224
        struct dinode *dip;
00225
00226
        bp = bread(ip->dev, IBLOCK(ip->inum, sb));
         dip = (struct dinode*)bp->data + ip->inum%IPB;
00227
        dip = (struct dimode*);
dip->type = ip->type;
dip->major = ip->major;
dip->minor = ip->minor;
00228
00229
00230
00231
         dip->nlink = ip->nlink;
         dip->size = ip->size;
00232
         memmove(dip->addrs, ip->addrs, sizeof(ip->addrs));
00233
00234
         log_write(bp);
00235
        brelse(bp);
00236 }
00237
00238 // Find the inode with number inum on device dev
00239 // and return the in-memory copy. Does not lock
00240 // the inode and does not read it from disk.
00241 static struct inode*
00242 iget (uint dev, uint inum)
00243 {
00244
        struct inode *ip, *empty;
00245
00246
        acquire (&icache.lock);
00247
00248
         // Is the inode already cached?
00249
         empty = 0;
00250
         for(ip = &icache.inode[0]; ip < &icache.inode[NINODE]; ip++){</pre>
00251
          if(ip->ref > 0 && ip->dev == dev && ip->inum == inum) {
00252
             ip->ref++;
00253
             release (&icache.lock);
00254
             return ip;
00255
           00256
00257
00258
00259
00260
         // Recycle an inode cache entry.
00261
         if(empty == 0)
00262
          panic("iget: no inodes");
00263
00264
        ip = empty;
        ip->dev = dev;
ip->inum = inum;
00265
00266
         ip->ref = 1;
00267
00268
         ip->valid = 0;
00269
        release(&icache.lock);
00270
00271
         return ip;
00272 }
00273
00274 // Increment reference count for ip.
00275 // Returns ip to enable ip = idup(ip1) idiom.
00276 struct inode*
00277 idup(struct inode *ip)
00278 {
00279
        acquire(&icache.lock);
        ip->ref++;
00280
00281
        release(&icache.lock);
00282
        return ip;
00283 }
00284
```

```
00285 // Lock the given inode.
00286 // Reads the inode from disk if necessary.
00287 void
00288 ilock(struct inode *ip)
00289 {
00290
        struct buf *bp;
        struct dinode *dip;
00291
00292
00293
        if(ip == 0 || ip->ref < 1)</pre>
          panic("ilock");
00294
00295
00296
        acquiresleep(&ip->lock);
00297
        if(ip->valid == 0){
00298
00299
          bp = bread(ip->dev, IBLOCK(ip->inum, sb));
00300
          dip = (struct dinode*)bp->data + ip->inum%IPB;
          ip->type = dip->type;
ip->major = dip->major;
00301
00302
          ip->minor = dip->minor;
00303
00304
          ip->nlink = dip->nlink;
00305
          ip->size = dip->size;
00306
          memmove(ip->addrs, dip->addrs, sizeof(ip->addrs));
00307
          brelse(bp);
          ip->valid = 1;
if(ip->type == 0)
00308
00309
00310
            panic("ilock: no type");
00311
00312 }
00313
00314 // Unlock the given inode.
00315 void
00316 iunlock(struct inode *ip)
00317 {
00318
        if(ip == 0 || !holdingsleep(&ip->lock) || ip->ref < 1)</pre>
00319
          panic("iunlock");
00320
00321
       releasesleep(&ip->lock);
00323
00324 // Drop a reference to an in-memory inode.
00325 // If that was the last reference, the inode cache entry can
00326 // be recycled.
00327 // If that was the last reference and the inode has no links
00328 // to it, free the inode (and its content) on disk.
00329 // All calls to iput() must be inside a transaction in
00330 // case it has to free the inode.
00331 void
00332 iput(struct inode *ip)
00333 {
00334
        acquiresleep(&ip->lock);
00335
        if(ip->valid && ip->nlink == 0) {
00336
          acquire(&icache.lock);
00337
          int r = ip -> ref;
00338
          release (&icache.lock);
          if(r == 1){ // inode has no links and no other references: truncate and free.
00339
00340
00341
             itrunc(ip);
00342
             ip->type = 0;
00343
             iupdate(ip);
00344
             ip->valid = 0;
00345
          }
00346
00347
        releasesleep(&ip->lock);
00348
00349
        acquire(&icache.lock);
00350
       ip->ref--;
00351
        release(&icache.lock);
00352 }
00353
00354 // Common idiom: unlock, then put.
00355 void
00356 iunlockput(struct inode *ip)
00357 {
        iunlock(ip);
00358
00359
       iput(ip);
00360 }
00361
00362 //PAGEBREAK!
00363 // Inode content
00364 //
00365 // The content (data) associated with each inode is stored
00366 // in blocks on the disk. The first NDIRECT block numbers 00367 // are listed in ip->addrs[]. The next NINDIRECT blocks are
00368 // listed in block ip->addrs[NDIRECT].
00369
00370 // Return the disk block address of the nth block in inode ip.
00371 // If there is no such block, bmap allocates one.
```

5.60 fs.c 203

```
00372 static uint
00373 bmap(struct inode *ip, uint bn)
00374 {
00375
        uint addr, *a;
00376
        struct buf *bp;
00377
00378
        if(bn < NDIRECT) {</pre>
00379
          if((addr = ip->addrs[bn]) == 0)
00380
            ip->addrs[bn] = addr = balloc(ip->dev);
00381
          return addr;
00382
00383
        bn -= NDIRECT:
00384
00385
        if (bn < NINDIRECT) {</pre>
00386
          // Load indirect block, allocating if necessary.
00387
          if((addr = ip->addrs[NDIRECT]) == 0)
            ip->addrs[NDIRECT] = addr = balloc(ip->dev);
00388
          bp = bread(ip->dev, addr);
00389
          a = (uint*)bp->data;
00390
          if((addr = a[bn]) == 0) {
   a[bn] = addr = balloc(ip->dev);
00391
00392
00393
            log_write(bp);
00394
00395
          brelse(bp);
00396
          return addr;
00397
00398
00399
        panic("bmap: out of range");
00400 }
00401
00402 // Truncate inode (discard contents).
00403 // Only called when the inode has no links
00404 // to it (no directory entries referring to it)
00405 // and has no in-memory reference to it (is
00406 // not an open file or current directory).
00407 static void
00408 itrunc(struct inode *ip)
        int i, j;
struct buf *bp;
00410
00411
00412
        uint *a;
00413
        for(i = 0; i < NDIRECT; i++) {</pre>
00414
         if(ip->addrs[i]){
00415
            bfree(ip->dev, ip->addrs[i]);
ip->addrs[i] = 0;
00416
00417
00418
00419
00420
        if (ip->addrs[NDIRECT]) {
00421
          bp = bread(ip->dev, ip->addrs[NDIRECT]);
00423
          a = (uint*)bp->data;
00424
          for(j = 0; j < NINDIRECT; j++){</pre>
00425
           <u>if</u>(a[j])
00426
              bfree(ip->dev, a[j]);
00427
          brelse(bp);
00429
          bfree(ip->dev, ip->addrs[NDIRECT]);
          ip->addrs[NDIRECT] = 0;
00430
00431
00432
00433
        ip->size = 0;
00434
        iupdate(ip);
00435 }
00436
00437 // Copy stat information from inode.
00438 // Caller must hold ip->lock.
00439 void
00440 stati(struct inode *ip, struct stat *st)
00441 {
00442 st->dev = ip->dev;
00443
        st->ino = ip->inum;
       st->type = ip->type;
st->nlink = ip->nlink;
00444
00445
       st->size = ip->size;
00446
00447 }
00448
00449 //PAGEBREAK!
00450 // Read data from inode.
00451 // Caller must hold ip->lock.
00452 int
00453 readi(struct inode *ip, char *dst, uint off, uint n)
00454 {
00455
        uint tot, m;
00456
       struct buf *bp;
00457
00458
        if(ip->tvpe == T DEV){
```

```
if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].read)
00460
00461
          return devsw[ip->major].read(ip, dst, n);
00462
00463
00464
        if(off > ip->size || off + n < off)</pre>
          return -1;
00465
00466
        if(off + n > ip->size)
00467
         n = ip -> size - off;
00468
00469
        for (tot=0; tot<n; tot+=m, off+=m, dst+=m) {</pre>
         bp = bread(ip->dev, bmap(ip, off/BSIZE));
m = min(n - tot, BSIZE - off%BSIZE);
00470
00471
00472
          memmove(dst, bp->data + off%BSIZE, m);
00473
          brelse(bp);
00474
00475
        return n:
00476 }
00478 // PAGEBREAK!
00479 // Write data to inode.
00480 // Caller must hold ip->lock.
00481 int
00482 writei(struct inode *ip, char *src, uint off, uint n)
00483 {
00484
        uint tot, m;
00485
        struct buf *bp;
00486
        if(ip->type == T_DEV){
00487
         if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].write)
00488
00489
            return -1;
00490
          return devsw[ip->major].write(ip, src, n);
00491
00492
00493
        if(off > ip->size \mid \mid off + n < off)
        return -1;
if (off + n > MAXFILE*BSIZE)
00494
00495
         return -1;
00497
00498
        for(tot=0; tot<n; tot+=m, off+=m, src+=m) {</pre>
        bp = bread(ip->dev, bmap(ip, off/BSIZE));
m = min(n - tot, BSIZE - off%BSIZE);
memmove(bp->data + off%BSIZE, src, m);
00499
00500
00501
00502
           log_write(bp);
00503
          brelse(bp);
00504
00505
        if(n > 0 && off > ip->size) {
00506
         ip->size = off;
00507
00508
          iupdate(ip);
00509
00510
        return n;
00511 }
00512
00513 //PAGEBREAK!
00514 // Directories
00516 int
00517 namecmp(const char *s, const char *t)
00518 {
00519
        return strncmp(s, t, DIRSIZ);
00520 }
00522 // Look for a directory entry in a directory.
00523 // If found, set *poff to byte offset of entry.
00524 struct inode*
00525 dirlookup(struct inode *dp, char *name, uint *poff)
00526 {
00527
       uint off, inum;
00528
        struct dirent de;
00529
00530
        if (dp->type != T_DIR)
00531
        panic("dirlookup not DIR");
00532
00533
        for(off = 0; off < dp->size; off += sizeof(de)){
00534
         if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
00535
             panic("dirlookup read");
00536
           if(de.inum == 0)
00537
             continue:
          if(namecmp(name, de.name) == 0) {
  // entry matches path element
00538
00539
             if (poff)
00541
               *poff = off;
00542
            inum = de.inum;
00543
            return iget(dp->dev, inum);
00544
00545
       }
```

5.60 fs.c 205

```
00547
        return 0;
00548 }
00549
00550 // Write a new directory entry (name, inum) into the directory dp.
00551 int
00552 dirlink(struct inode *dp, char *name, uint inum)
00553 {
00554
        int off;
00555
        struct dirent de;
00556
        struct inode *ip;
00557
00558
        // Check that name is not present.
00559
        if((ip = dirlookup(dp, name, 0)) != 0){
00560
         iput(ip);
00561
          return -1;
00562
00563
00564
        // Look for an empty dirent.
        for (off = 0; off < dp->size; off += sizeof(de)) {
00565
00566
          if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
00567
             panic("dirlink read");
           if(de.inum == 0)
00568
00569
            break;
00570
00571
00572
        strncpy(de.name, name, DIRSIZ);
00573
        de.inum = inum;
        if(writei(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
  panic("dirlink");
00574
00575
00576
00577
        return 0;
00578 }
00579
00580 //PAGEBREAK!
00581 // Paths
00582
00583 // Copy the next path element from path into name.
00584 // Return a pointer to the element following the copied one.
00585 // The returned path has no leading slashes,
00586 // so the caller can check *path=='\' to see if the name is the last one.
00587 // If no name to remove, return 0.
00588 //
00589 // Examples:
         skipelem("a/bb/c", name) = "bb/c", setting name = "a"
skipelem("///a//bb", name) = "bb", setting name = "a"
skipelem("a", name) = "", setting name = "a"
skipelem("", name) = skipelem("///", name) = 0
00590 //
00591 //
00592 //
00593 //
00594 //
00595 static char*
00596 skipelem(char *path, char *name)
00597 {
00598
        char *s;
00599
       int len;
00600
00601
        while(*path == '/')
00602
          path++;
00603
        if(*path == 0)
00604
          return 0;
00605
        s = path;
        while (*path != '/' && *path != 0)
00606
00607
          path++;
00608
        len = path - s;
00609
        if(len >= DIRSIZ)
00610
          memmove(name, s, DIRSIZ);
00611
        else {
        memmove(name, s, len);
00612
00613
          name[len] = 0;
00614
00615
        while(*path == '/')
00616
          path++;
00617
        return path;
00618 }
00619
00620 // Look up and return the inode for a path name.
00621 // If parent != 0, return the inode for the parent and copy the final
00622 // path element into name, which must have room for DIRSIZ bytes.
00623 // Must be called inside a transaction since it calls iput().
00624 static struct inode*
00625 namex(char *path, int nameiparent, char *name)
00626 {
00627
        struct inode *ip, *next;
00628
00629
        if (*path == '/')
00630
          ip = iget(ROOTDEV, ROOTINO);
        else
00631
00632
          ip = idup(myproc()->cwd);
```

```
00634
        while((path = skipelem(path, name)) != 0){
00635
         if(ip->type != T_DIR) {
00636
00637
           iunlockput(ip);
00638
            return 0:
00639
00640
          if(nameiparent && *path == ' \setminus 0'){
          // Stop one level early.
00641
00642
            iunlock(ip);
00643
           return ip;
00644
          if((next = dirlookup(ip, name, 0)) == 0){
00645
00646
           iunlockput(ip);
00647
            return 0;
00648
          iunlockput(ip);
00649
00650
         ip = next;
00651
00652
       if (nameiparent) {
00653
         iput(ip);
00654
          return 0;
00655
00656
       return ip;
00657 }
00658
00659 struct inode*
00660 namei(char *path)
00661 {
       char name[DIRSIZ];
00662
00663
       return namex(path, 0, name);
00664 }
00665
00666 struct inode*
00667 nameiparent(char *path, char *name)
00668 {
00669
        return namex(path, 1, name);
```

5.61 fs.d File Reference

5.62 fs.d

Go to the documentation of this file.

00001 fs.o: fs.c /usr/include/stdc-predef.h types.h defs.h param.h stat.h mmu.h \setminus 00002 proc.h spinlock.h sleeplock.h fs.h buf.h file.h

5.63 fs.h File Reference

Classes

- struct dinode
- · struct dirent
- · struct superblock

Macros

- #define BBLOCK(b, sb) (b/BPB + sb.bmapstart)
- #define BPB (BSIZE*8)
- #define BSIZE 512
- #define DIRSIZ 14
- #define IBLOCK(i, sb) ((i) / IPB + sb.inodestart)
- #define IPB (BSIZE / sizeof(struct dinode))
- #define MAXFILE (NDIRECT + NINDIRECT)
- #define NDIRECT 12
- #define NINDIRECT (BSIZE / sizeof(uint))
- #define ROOTINO 1

5.63 fs.h File Reference 207

5.63.1 Macro Definition Documentation

5.63.1.1 BBLOCK

```
#define BBLOCK( b, \\ sb \ ) \ (b/{\tt BPB} \ + \ {\tt sb.bmapstart})
```

Definition at line 48 of file fs.h.

5.63.1.2 BPB

```
#define BPB (BSIZE*8)
```

Definition at line 45 of file fs.h.

5.63.1.3 BSIZE

#define BSIZE 512

Definition at line 6 of file fs.h.

5.63.1.4 DIRSIZ

#define DIRSIZ 14

Definition at line 51 of file fs.h.

5.63.1.5 IBLOCK

Definition at line 42 of file fs.h.

5.63.1.6 IPB

```
#define IPB (BSIZE / sizeof(struct dinode))
```

Definition at line 39 of file fs.h.

5.63.1.7 MAXFILE

```
#define MAXFILE (NDIRECT + NINDIRECT)
```

Definition at line 26 of file fs.h.

5.63.1.8 NDIRECT

#define NDIRECT 12

Definition at line 24 of file fs.h.

5.63.1.9 **NINDIRECT**

```
#define NINDIRECT (BSIZE / sizeof(uint))
```

Definition at line 25 of file fs.h.

5.63.1.10 ROOTINO

#define ROOTINO 1

Definition at line 5 of file fs.h.

5.64 fs.h 209

5.64 fs.h

Go to the documentation of this file.

```
00001 // On-disk file system format.
00002 // Both the kernel and user programs use this header file.
00004
00005 #define ROOTINO 1 // root i-number 00006 #define BSIZE 512 // block size
00007
00008 // Disk layout:
00009 // [ boot block | super block | log | inode blocks |
00010 //
                                                      free bit map | data blocks]
00011 //
00012 // mkfs computes the super block and builds an initial file system. The
00013 // super block describes the disk layout:
00014 struct superblock {
      uint size;
00015
                              // Size of file system image (blocks)
00016
        uint nblocks;
                             // Number of data blocks
00017
        uint ninodes;
                             // Number of inodes.
                             // Number of log blocks
00018
        uint nlog;
00019 uint logstart; // Block number of first log block
00020 uint inodestart; // Block number of first inode block
00021
        uint bmapstart;
                             // Block number of first free map block
00022 };
00023
00024 #define NDIRECT 12
00025 #define NINDIRECT (BSIZE / sizeof(uint))
00026 #define MAXFILE (NDIRECT + NINDIRECT)
00028 // On-disk inode structure
00029 struct dinode {
00030 short type;
                                 // File type
00031
        short major;
                                 // Major device number (T_DEV only)
00032
                                // Minor device number (T_DEV only)
        short minor:
00033
                                // Number of links to inode in file system
        short nlink;
00034 uint size; // Size of file (bytes)
00035 uint addrs[NDIRECT+1]; // Data block addresses
00036 };
00037
00038 // Inodes per block.
                              (BSIZE / sizeof(struct dinode))
00039 #define IPB
00040
00041 // Block containing inode i
00042 #define IBLOCK(i, sb) ((i) / IPB + sb.inodestart)
00043
00044 // Bitmap bits per block
00045 #define BPB
                              (BSIZE*8)
00046
00047 // Block of free map containing bit for block b
00048 #define BBLOCK(b, sb) (b/BPB + sb.bmapstart)
00049
00050 // Directory is a file containing a sequence of dirent structures.
00051 #define DIRSIZ 14
00052
00053 struct dirent {
00054 ushort inum;
00055 char name[DIRSIZ];
00056 };
00057
```

5.65 grep.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

```
• void grep (char *pattern, int fd)
```

- int main (int argc, char *argv[])
- int match (char *, char *)
- int matchhere (char *, char *)
- int matchstar (int, char *, char *)

Variables

• char buf [1024]

5.65.1 Function Documentation

5.65.1.1 grep()

```
void grep ( \mbox{char} \ * \ pattern, \mbox{int} \ fd \ )
```

Definition at line 11 of file grep.c.

```
00012 {
00013
         int n, m;
00014
         char *p, *q;
00015
         m = 0;
while((n = read(fd, buf+m, sizeof(buf)-m-1)) > 0){
00016
00017
00018
          m += n;
           buf[m] = '\0';
00019
           p = buf;
while((q = strchr(p, '\n')) != 0){
00020
00021
             *q = 0;
if (match (pattern, p)) {
    *q = '\n';
    write(1) = ...
00022
00023
00025
               write(1, p, q+1 - p);
00026
             p = q+1;
00027
00028
00029
           if(p == buf)
00030
             m = 0;
00031
            if (m > 0) {
            m -= p - buf;
memmove(buf, p, m);
00032
00033
00034
00035 }
```

Referenced by main().

5.65.1.2 main()

```
int main (
          int argc,
          char * argv[] )
```

Definition at line 39 of file grep.c.

```
00040 {
00041
         int fd, i;
00042
         char *pattern;
00043
00044
         if(argc <= 1) {
   printf(2, "usage: grep pattern [file ...]\n");</pre>
00045
00046
           exit();
00047
00048
        pattern = argv[1];
00049
00050
        if(argc <= 2){</pre>
         grep(pattern, 0);
exit();
00051
00052
00053
```

5.65.1.3 match()

```
int match ( \label{eq:char} \mbox{char} \ * \ re, \mbox{char} \ * \ text \ )
```

Definition at line 73 of file grep.c.

```
00074 {
00075     if(re[0] == '^')
00076     return matchhere(re+1, text);
do{ // must look at empty string
00078     if(matchhere(re, text))
00079     return 1;
00080     }
while(*text++ != '\0');
return 0;
00081 }
```

Referenced by grep().

5.65.1.4 matchhere()

```
int matchhere ( {\rm char} \, * \, re, {\rm char} \, * \, text \, )
```

Definition at line 85 of file grep.c.

Referenced by match(), matchhere(), and matchstar().

5.65.1.5 matchstar()

```
int matchstar (
              int c,
              char * re,
              char * text )
Definition at line 99 of file grep.c.
        do{ // a * matches zero or more instances
00101
         if(matchhere(re, text))
00102
00103
            return 1;
       }while(*text!='\0' && (*text++==c || c=='.'));
00104
00105
       return 0;
00106 }
```

Referenced by matchhere().

5.65.2 Variable Documentation

5.65.2.1 buf

```
char buf[1024]
```

Definition at line 7 of file grep.c.

Referenced by grep().

5.66 grep.c

```
Go to the documentation of this file.
```

```
00001 // Simple grep. Only supports ^ . * $ operators.
00002
00003 #include "types.h"
00004 #include "stat.h"
00005 #include "user.h"
00006
00007 char buf[1024];
00008 int match(char*, char*);
00009
00010 void

00011 grep(char *pattern, int fd)

00012 {
00013
         int n, m;
00014 char *p, *q;
00015
         m = 0;
while((n = read(fd, buf+m, sizeof(buf)-m-1)) > 0){
  m += n;
  buf[m] = '\0';
00016
00017
00018
00019
00020
00021
            while ((q = strchr(p, '\n')) != 0) {
00022
              \star q = 0;
             if (match (pattern, p)) {
  *q = '\n';
00023
00024
00025
                write(1, p, q+1 - p);
00026
00027
              p = q+1;
00028
00029
           if (p == buf)
m = 0;
00030
00031
            <u>if</u>(m > 0) {
00032
             m -= p - buf;
```

```
memmove(buf, p, m);
00034
00035 }
00036 }
00037
00038 int
00039 main(int argc, char *argv[])
00040 {
00041
        int fd, i;
00042
        char *pattern;
00043
00044
        if (argc <= 1) {</pre>
        printf(2, "usage: grep pattern [file ...]\n");
exit();
00045
00046
00047
00048
       pattern = argv[1];
00049
00050
        if (argc <= 2) {
         grep(pattern, 0);
00052
          exit();
00053
00054
        for(i = 2; i < argc; i++) {
  if((fd = open(argv[i], 0)) < 0) {
    printf(1, "grep: cannot open %s\n", argv[i]);</pre>
00055
00056
00057
            exit();
00059
00060
          grep(pattern, fd);
00061
          close(fd);
00062
00063
        exit();
00064 }
00065
00066 // Regexp matcher from Kernighan & Pike,
00067 // The Practice of Programming, Chapter 9.
00068
00069 int matchhere(char*, char*);
00070 int matchstar(int, char*, char*);
00071
00072 int
00073 match(char *re, char *text)
00074 {
        if (re[0] == '^')
00075
          return matchhere(re+1, text);
        do{ // must look at empty string
        if (matchhere (re, text))
00078
00079
            return 1;
00080 }while(*text++ != '\0');
00081 return 0;
00082 }
00084 // matchhere: search for re at beginning of text
00085 int matchhere(char *re, char *text)
00086 {
        if(re[0] == '\0')
00087
00088
        return 1;
if(re[1] == '*')
00090
          return matchstar(re[0], re+2, text);
        if(re[0] == '$' && re[1] == '\0')
  return *text == '\0';
if(*text!='\0' && (re[0]=='.' || re[0]==*text))
00091
00092
00093
00094
         return matchhere(re+1, text+1);
00095
        return 0;
00096 }
00097
00098 // matchstar: search for c*re at beginning of text
00099 int matchstar(int c, char *re, char *text)
00100 {
00101 do{ // a * matches zero or more instances
        if (matchhere (re, text))
00103
             return 1;
        }while(*text!='\0' && (*text++==c || c=='.'));
00104
00105
       return 0;
00106 }
00107
```

5.67 grep.d File Reference

5.68 grep.d

Go to the documentation of this file.

```
00001 grep.o: grep.c /usr/include/stdc-predef.h types.h stat.h user.h
```

5.69 ide.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
#include "traps.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "fs.h"
#include "buf.h"
```

Macros

- #define IDE BSY 0x80
- #define IDE_CMD_RDMUL 0xc4
- #define IDE_CMD_READ 0x20
- #define IDE_CMD_WRITE 0x30
- #define IDE_CMD_WRMUL 0xc5
- #define IDE DF 0x20
- #define IDE DRDY 0x40
- #define IDE_ERR 0x01
- #define SECTOR_SIZE 512

Functions

- void ideinit (void)
- void ideintr (void)
- void iderw (struct buf *b)
- static void idestart (struct buf *)
- static int idewait (int checkerr)

Variables

- static int havedisk1
- · static struct spinlock idelock
- static struct buf * idequeue

5.69.1 Macro Definition Documentation

5.69 ide.c File Reference 215

5.69.1.1 IDE_BSY

#define IDE_BSY 0x80

Definition at line 17 of file ide.c.

5.69.1.2 IDE_CMD_RDMUL

#define IDE_CMD_RDMUL 0xc4

Definition at line 24 of file ide.c.

5.69.1.3 IDE_CMD_READ

#define IDE_CMD_READ 0x20

Definition at line 22 of file ide.c.

5.69.1.4 IDE_CMD_WRITE

#define IDE_CMD_WRITE 0x30

Definition at line 23 of file ide.c.

5.69.1.5 IDE_CMD_WRMUL

#define IDE_CMD_WRMUL 0xc5

Definition at line 25 of file ide.c.

5.69.1.6 IDE_DF

#define IDE_DF 0x20

Definition at line 19 of file ide.c.

5.69.1.7 IDE_DRDY

```
#define IDE_DRDY 0x40
```

Definition at line 18 of file ide.c.

5.69.1.8 IDE_ERR

```
#define IDE_ERR 0x01
```

Definition at line 20 of file ide.c.

5.69.1.9 SECTOR_SIZE

```
#define SECTOR_SIZE 512
```

Definition at line 16 of file ide.c.

5.69.2 Function Documentation

5.69.2.1 ideinit()

```
void ideinit (
     void )
```

Definition at line 51 of file ide.c.

```
00052 {
00053
00054
             int i;
           initlock(&idelock, "ide");
ioapicenable(IRO_IDE, ncpu - 1);
idewait(0);
00055
00056
00057
00058
           // Check if disk 1 is present
outb(0x1f6, 0xe0 | (1«4));
for(i=0; i<1000; i++) {
   if(inb(0x1f7) != 0) {
      havedisk1 = 1;
      havel;
}</pre>
00059
00060
00061
00062
00063
           }
00064
                   break;
00065
00067
00068
           // Switch back to disk 0.
00069 outb(0x1f6, 0xe0 | (0«4));
00070 }
```

5.69 ide.c File Reference 217

5.69.2.2 ideintr()

```
void ideintr (
                void )
Definition at line 104 of file ide.c.
00105 +
00106
         struct buf *b;
00107
00108
         // First queued buffer is the active request.
00109
         acquire(&idelock);
00110
00111
         if((b = idequeue) == 0){
          release(&idelock);
00112
00113
           return;
00114
00115
         idequeue = b->qnext;
00116
00117
        // Read data if needed.
if(!(b->flags & B_DIRTY) && idewait(1) >= 0)
00118
00119
          insl(0x1f0, b->data, BSIZE/4);
00120
00121
         \ensuremath{//} Wake process waiting for this buf.
        b->flags |= B_VALID;
b->flags &= ~B_DIRTY;
00122
00123
00124
        wakeup(b);
00125
00126
        // Start disk on next buf in queue.
00127
        if(idequeue != 0)
00128
          idestart(idequeue);
00130 release(&idelock);
00131 }
00129
```

5.69.2.3 iderw()

```
void iderw ( {\tt struct\ buf\ *\ b\ )}
```

Definition at line 138 of file ide.c.

```
00139
        struct buf **pp;
00140
00141
00142
        if(!holdingsleep(&b->lock))
00143
          panic("iderw: buf not locked");
00144
        if((b->flags & (B_VALID|B_DIRTY)) == B_VALID)
        panic("iderw: nothing to do");
if(b->dev != 0 && !havedisk1)
panic("iderw: ide disk 1 not present");
00145
00146
00147
00148
00149
        acquire(&idelock); //DOC:acquire-lock
00150
00151
        // Append b to idequeue.
00152
        b->qnext = 0;
        for(pp=&idequeue; *pp; pp=&(*pp)->qnext) //DOC:insert-queue
00153
00154
        *pp = b;
00155
00156
00157
        // Start disk if necessary.
00158
        if(idequeue == b)
          idestart(b);
00159
00160
00161
        // Wait for request to finish.
00162
        while((b->flags & (B_VALID|B_DIRTY)) != B_VALID){
       sleep(b, &idelock);
00163
00164
00165
00166
00167
        release(&idelock);
00168 }
```

5.69.2.4 idestart()

```
static void idestart (
                   struct buf * b ) [static]
Definition at line 74 of file ide.c.
00075 {
00076
          if(b == 0)
00077
            panic("idestart");
          if (b->blockno >= FSSIZE)
00079
            panic("incorrect blockno");
         int sector_per_block = BSIZE/SECTOR_SIZE;
int sector = b->blockno * sector_per_block;
int read_cmd = (sector_per_block == 1) ? IDE_CMD_READ : IDE_CMD_RDMUL;
int write_cmd = (sector_per_block == 1) ? IDE_CMD_WRITE : IDE_CMD_WRMUL;
08000
00081
00082
00083
00084
00085
          if (sector_per_block > 7) panic("idestart");
00086
00087
          idewait(0):
         outb(0x3f6, 0); // generate interrupt
00088
00089
         outb(0x1f2, sector_per_block); // number of sectors
00090
         outb(0x1f3, sector & 0xff);
00091
00092
          outb(0x1f4, (sector » 8) & 0xff);
          outb(0x1f5, (sector » 16) & 0xff);
         outb(0x1f6, 0xe0 | ((b->dev&1) «4) | ((sector»24)&0x0f));
if(b->flags & B_DIRTY){
  outb(0x1f7, write_cmd);
  outs1(0x1f0, b->data, BSIZE/4);
00093
00094
00095
00096
00097
          } else
00098
            outb(0x1f7, read_cmd);
         }
00099
00100 }
```

Referenced by ideintr(), and iderw().

5.69.2.5 idewait()

static int idewait (

```
int checkerr ) [static]

Definition at line 39 of file ide.c.

00040 {
00041    int r;
00042
00043    while(((r = inb(0x1f7)) & (IDE_BSY|IDE_DRDY)) != IDE_DRDY)
00044    ;
00045    if(checkerr && (r & (IDE_DF|IDE_ERR)) != 0)
        return -1;
00047    return 0;
```

Referenced by ideinit(), ideintr(), and idestart().

5.69.3 Variable Documentation

5.69.3.1 havedisk1

```
int havedisk1 [static]
```

Definition at line 34 of file ide.c.

Referenced by ideinit(), and iderw().

5.70 ide.c 219

5.69.3.2 idelock

```
struct spinlock idelock [static]
```

Definition at line 31 of file ide.c.

Referenced by ideinit(), ideintr(), and iderw().

5.69.3.3 idequeue

```
struct buf* idequeue [static]
```

Definition at line 32 of file ide.c.

Referenced by ideintr(), and iderw().

5.70 ide.c

```
Go to the documentation of this file.
```

```
00001 // Simple PIO-based (non-DMA) IDE driver code.
00002
00003 #include "types.h"
00004 #include "defs.h"
00005 #include "param.h"
00006 #include "memlayout.h"
00007 #include "mmu.h"
00008 #include "proc.h"
00009 #include "x86.h"
00010 #include "traps.h"
00011 #include "spinlock.h"
00012 #include "sleeplock.h"
00013 #include "fs.h"
00014 #include "buf.h"
00015
00016 #define SECTOR_SIZE
                               512
00017 #define IDE_BSY
                               0x80
00018 #define IDE_DRDY
00019 #define IDE_DF
00020 #define IDE_ERR
                               0x01
00021
00022 #define IDE_CMD_READ 0x20
00023 #define IDE_CMD_WRITE 0x30
00024 #define IDE_CMD_RDMUL 0xc4
00025 #define IDE_CMD_WRMUL 0xc5
00026
00027 // idequeue points to the buf now being read/written to the disk.
00028 // idequeue->qnext points to the next buf to be processed.
00029 // You must hold idelock while manipulating queue.
00031 static struct spinlock idelock;
00032 static struct buf *idequeue;
00033
00034 static int havedisk1;
00035 static void idestart(struct buf*);
00036
00037 // Wait for IDE disk to become ready.
00038 static int
00039 idewait(int checkerr)
00040 {
00041
        int r;
00042
00043
        while(((r = inb(0x1f7)) & (IDE_BSY|IDE_DRDY)) != IDE_DRDY)
00044
00045
        if(checkerr && (r & (IDE_DF|IDE_ERR)) != 0)
00046
          return -1;
00047
        return 0;
00048 }
00049
```

```
00050 void
00051 ideinit (void)
00052 {
00053
         int i:
00054
00055
         initlock(&idelock, "ide");
         ioapicenable(IRQ_IDE, ncpu - 1);
00056
00057
         idewait(0);
00058
        // Check if disk 1 is present
outb(0x1f6, 0xe0 | (1«4));
for(i=0; i<1000; i++) {
  if(inb(0x1f7) != 0) {</pre>
00059
00060
00061
00062
00063
              havedisk1 = 1;
00064
              break;
00065
00066
00067
00068
         // Switch back to disk 0.
        outb(0x1f6, 0xe0 | (0«4));
00069
00070 }
00071
00072 // Start the request for b. Caller must hold idelock.
00073 static void
00074 idestart(struct buf *b)
00075 {
00076
         if(b == 0)
00077
           panic("idestart");
00078
         if(b->blockno >= FSSIZE)
          panic("incorrect blockno");
00079
08000
         int sector_per_block = BSIZE/SECTOR_SIZE;
00081
         int sector = b->blockno * sector_per_block;
00082
         int read_cmd = (sector_per_block == 1) ? IDE_CMD_READ : IDE_CMD_RDMUL;
00083
         int write_cmd = (sector_per_block == 1) ? IDE_CMD_WRITE : IDE_CMD_WRMUL;
00084
         if (sector_per_block > 7) panic("idestart");
00085
00086
         idewait(0);
00088
         outb(0x3f6, 0); // generate interrupt
00089
         outb(0x1f2, sector_per_block); // number of sectors
00090
         outb(0x1f3, sector & 0xff);
        outb(0x1f4, (sector & UXII);
outb(0x1f4, (sector » 8) & 0xff);
outb(0x1f5, (sector » 16) & 0xff);
outb(0x1f6, 0xe0 | ((b->dev&1) «4) | ((sector » 24) & 0x0f));
if(b->flags & B_DIRTY){
  outb(0x1f7, write_cmd);
  outb(0x1f7, b->deta_b_COUTA(A));
}
00091
00092
00093
00094
00095
00096
           outsl(0x1f0, b->data, BSIZE/4);
        outb(0x1f7, read_cmd);
}
00097
00098
00099
00100 }
00101
00102 // Interrupt handler.
00103 void
00104 ideintr(void)
00105 {
00106
00107
00108
         // First queued buffer is the active request.
00109
         acquire (&idelock);
00110
00111
         if((b = idequeue) == 0){
          release (&idelock);
00112
00113
           return;
00114
00115
         idequeue = b->qnext;
00116
00117
         // Read data if needed.
         if(!(b->flags & B_DIRTY) && idewait(1) >= 0)
00118
           insl(0x1f0, b->data, BSIZE/4);
00119
00120
00121
         // Wake process waiting for this buf.
         b->flags |= B_VALID;
b->flags &= ~B_DIRTY;
00122
00123
00124
         wakeup(b);
00125
00126
         // Start disk on next buf in queue.
00127
         if(idequeue != 0)
00128
           idestart (idequeue);
00129
00130
         release(&idelock);
00131 }
00132
00133 //PAGEBREAK!
00134 // Sync buf with disk.
00135 // If B_DIRTY is set, write buf to disk, clear B_DIRTY, set B_VALID.
00136 // Else if B_VALID is not set, read buf from disk, set B_VALID.
```

5.71 ide.d File Reference 221

```
00137 void
00138 iderw(struct buf *b)
00139 {
00140
       struct buf **pp;
00141
00142
       if(!holdingsleep(&b->lock))
         panic("iderw: buf not locked");
00144
       if((b->flags & (B_VALID|B_DIRTY)) == B_VALID)
00145
         panic("iderw: nothing to do");
       if (b->dev != 0 && !havedisk1)
00146
        panic("iderw: ide disk 1 not present");
00147
00148
00149
       acquire(&idelock); //DOC:acquire-lock
00150
00151
       // Append b to idequeue.
00152
       b->qnext = 0;
       for(pp=&idequeue; *pp; pp=&(*pp)->qnext) //DOC:insert-queue
00153
00154
       *pp = b;
00155
00156
00157
       // Start disk if necessary.
00158
       if(idequeue == b)
        idestart(b);
00159
00160
00161
       // Wait for request to finish.
00162
       while((b->flags & (B_VALID|B_DIRTY)) != B_VALID) {
      sleep(b, &idelock);
}
00163
00164
00165
00166
00167
       release(&idelock);
00168 }
```

5.71 ide.d File Reference

5.72 ide.d

```
Go to the documentation of this file.
```

```
00001 ide.o: ide.c /usr/include/stdc-predef.h types.h defs.h param.h \ 00002 memlayout.h mmu.h proc.h x86.h traps.h spinlock.h sleeplock.h fs.h buf.h
```

5.73 init.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
```

Functions

• int main (void)

Variables

char * argv [] = { "sh", 0 }

5.73.1 Function Documentation

5.73.1.1 main()

```
int main (
                  void )
Definition at line 11 of file init.c.
          int pid, wpid;
00013
00014
          if(open("console", O_RDWR) < 0){
  mknod("console", 1, 1);
  open("console", O_RDWR);</pre>
00015
00016
00017
00018
         dup(0); // stdout
dup(0); // stderr
00019
00020
00021
00022
          for(;;) {
          printf(1, "init: starting sh\n");
pid = fork():
00024
            pid = fork();
            if(pid < 0) {
  printf(1, "init: fork failed\n");</pre>
00025
00026
00027
               exit();
00028
00029
            if (pid == 0) {
              exec("sh", argv);
printf(1, "init: exec sh failed\n");
00030
00031
00032
               exit();
00033
00034
            while((wpid=wait()) >= 0 && wpid != pid)
00035
               printf(1, "zombie!\n");
00036 }
00037 }
```

5.73.2 Variable Documentation

5.73.2.1 argv

```
char* argv[] = { "sh", 0 }
```

Definition at line 8 of file init.c.

Referenced by exec(), main(), and sys_exec().

5.74 init.c

Go to the documentation of this file.

```
00001 // init: The initial user-level program
00002
00003 #include "types.h"
00003 #Include types.n
00004 #include "stat.h"
00005 #include "user.h"
00006 #include "fcntl.h"
00007
00008 char *argv[] = { "sh", 0 };
00009
00010 int
00011 main(void)
00012 {
00013
          int pid, wpid;
00014
          if(open("console", O_RDWR) < 0) {
  mknod("console", 1, 1);
  open("console", O_RDWR);</pre>
00015
00016
00017
00018
```

5.75 init.d File Reference 223

```
dup(0); // stdout
dup(0); // stderr
00020
00021
00022
         for(;;) {
            printf(1, "init: starting sh\n");
pid = fork();
00023
00024
            if(pid < 0){
  printf(1, "init: fork failed\n");</pre>
00026
00027
               exit();
00028
00029
            if(pid == 0){
              exec("sh", argv);
printf(1, "init: exec sh failed\n");
00030
00031
00032
              exit();
00033
            while((wpid=wait()) >= 0 && wpid != pid)
printf(1, "zombie!\n");
00034
00035
00036
00037 }
```

5.75 init.d File Reference

5.76 init.d

Go to the documentation of this file.

00001 init.o: init.c /usr/include/stdc-predef.h types.h stat.h user.h fcntl.h

5.77 initcode.d File Reference

5.78 initcode.d

Go to the documentation of this file.

00001 initcode.o: initcode.S syscall.h traps.h

5.79 ioapic.c File Reference

```
#include "types.h"
#include "defs.h"
#include "traps.h"
```

Classes

struct ioapic

Macros

- #define INT_ACTIVELOW 0x00002000
- #define INT_DISABLED 0x00010000
- #define INT LEVEL 0x00008000
- #define INT LOGICAL 0x00000800
- #define IOAPIC 0xFEC00000
- #define REG ID 0x00
- #define REG_TABLE 0x10
- #define REG_VER 0x01

Functions

- void ioapicenable (int irq, int cpunum)
- void ioapicinit (void)
- static uint ioapicread (int reg)
- static void ioapicwrite (int reg, uint data)

Variables

• volatile struct ioapic * ioapic

5.79.1 Macro Definition Documentation

5.79.1.1 INT_ACTIVELOW

#define INT_ACTIVELOW 0x00002000

Definition at line 22 of file ioapic.c.

5.79.1.2 INT_DISABLED

#define INT_DISABLED 0x00010000

Definition at line 20 of file ioapic.c.

5.79.1.3 INT_LEVEL

#define INT_LEVEL 0x00008000

Definition at line 21 of file ioapic.c.

5.79.1.4 INT_LOGICAL

#define INT_LOGICAL 0x00000800

Definition at line 23 of file ioapic.c.

5.79.1.5 IOAPIC

```
#define IOAPIC 0xFEC00000
```

Definition at line 9 of file ioapic.c.

5.79.1.6 REG_ID

```
#define REG_ID 0x00
```

Definition at line 11 of file ioapic.c.

5.79.1.7 **REG_TABLE**

```
#define REG_TABLE 0x10
```

Definition at line 13 of file ioapic.c.

5.79.1.8 REG_VER

```
#define REG_VER 0x01
```

Definition at line 12 of file ioapic.c.

5.79.2 Function Documentation

5.79.2.1 ioapicenable()

```
void ioapicenable (
    int irq,
    int cpunum )
```

Definition at line 68 of file ioapic.c.

Referenced by consoleinit(), ideinit(), and uartinit().

5.79.2.2 ioapicinit()

```
void ioapicinit (
    void )
```

Definition at line 49 of file ioapic.c.

```
00050 {
00051
         int i, id, maxintr;
00053
         ioapic = (volatile struct ioapic*)IOAPIC;
00054
         maxintr = (ioapicread(REG_VER) » 16) & 0xFF;
         id = ioapicread(REG_ID) >> 24;
00055
        if (id != ioapicid)
    cprintf("ioapicinit: id isn't equal to ioapicid; not a MP\n");
00056
00057
00058
00059
         // Mark all interrupts edge-triggered, active high, disabled,
00060
         // and not routed to any CPUs.
         for(i = 0; i <= maxintr; i++) {
  ioapicwrite(REG_TABLE+2*i, INT_DISABLED | (T_IRQ0 + i));
  ioapicwrite(REG_TABLE+2*i+1, 0);</pre>
00061
00062
00063
00064 }
00065 }
```

5.79.2.3 ioapicread()

Definition at line 35 of file ioapic.c.

```
00036 {
00037    ioapic->reg = reg;
00038    return ioapic->data;
00039 }
```

Referenced by ioapicinit().

5.79.2.4 ioapicwrite()

```
static void ioapicwrite (
          int reg,
          uint data ) [static]
```

Definition at line 42 of file ioapic.c.

Referenced by ioapicenable(), and ioapicinit().

5.79.3 Variable Documentation

5.80 ioapic.c 227

5.79.3.1 ioapic

```
volatile struct ioapic* ioapic
```

Definition at line 25 of file ioapic.c.

5.80 ioapic.c

Go to the documentation of this file.

```
00001 // The I/O APIC manages hardware interrupts for an SMP system. 00002 // http://www.intel.com/design/chipsets/datashts/29056601.pdf
00003 // See also picirq.c.
00005 #include "types.h"
00006 #include "defs.h"
00007 #include "traps.h"
80000
00009 #define IOAPIC 0xFEC00000 // Default physical address of IO APIC
00010
00011 #define REG_ID 0x00 // Register index: ID 00012 #define REG_VER 0x01 // Register index: version 00013 #define REG_TABLE 0x10 // Redirection table base
00014
00015 // The redirection table starts at REG_TABLE and uses
00016 // two registers to configure each interrupt.
00017 // The first (low) register in a pair contains configuration bits.
00018 // The second (high) register contains a bitmask telling which
00019 // CPUs can serve that interrupt.
00020 #define INT_DISABLED 0x00010000 // Interrupt disabled
00021 #define INT_LEVEL 0x00008000 // Level-triggered (vs edge-)
00022 #define INT_ACTIVELOW 0x00002000 // Active low (vs high)
00023 #define INT_LOGICAL 0x00000800 // Destination is CPU id (vs APIC ID)
00024
00025 volatile struct ioapic *ioapic;
00026
00027 // IO APIC MMIO structure: write reg, then read or write data.
00028 struct ioapic {
        uint reg;
00030
         uint pad[3];
00031
         uint data;
00032 };
00033
00034 static uint
00035 ioapicread(int reg)
00036 {
00037 ioapic->reg = reg;
00038 return ioapic->data;
00039 }
00040
00041 static void
00042 ioapicwrite(int reg, uint data)
00043 {
00044
         ioapic->reg = reg;
00045
         ioapic->data = data;
00046 }
00047
00048 void
00049 ioapicinit (void)
00050 {
00051
         int i, id, maxintr;
00052
00053
         ioapic = (volatile struct ioapic*)IOAPIC;
         maxintr = (ioapicread(REG_VER) » 16) & 0xFF;
00054
00055
         id = ioapicread(REG_ID) >> 24;
00056
         if(id != ioapicid)
00057
            cprintf("ioapicinit: id isn't equal to ioapicid; not a MP\n");
00058
00059
         // Mark all interrupts edge-triggered, active high, disabled,
00060
         // and not routed to any CPUs.
00061
         for(i = 0; i <= maxintr; i++) {</pre>
00062
            ioapicwrite(REG_TABLE+2*i, INT_DISABLED | (T_IRQ0 + i));
00063
            ioapicwrite(REG_TABLE+2*i+1, 0);
00064
00065 }
00066
00067 void
00068 ioapicenable(int irq, int cpunum)
00069 {
```

```
00070 // Mark interrupt edge-triggered, active high, 00071 // enabled, and routed to the given cpunum, 00072 // which happens to be that cpu's APIC ID. 00073 ioapicwrite(REG_TABLE+2*irq, T_IRQ0 + irq); 00074 ioapicwrite(REG_TABLE+2*irq+1, cpunum « 24); 00075 }
```

5.81 ioapic.d File Reference

5.82 ioapic.d

```
Go to the documentation of this file.
```

```
00001 ioapic.o: ioapic.c /usr/include/stdc-predef.h types.h defs.h traps.h
```

5.83 kalloc.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "spinlock.h"
```

Classes

• struct run

Functions

```
• void freerange (void *vstart, void *vend)
```

- char * kalloc (void)
- void kfree (char *v)
- void kinit1 (void *vstart, void *vend)
- void kinit2 (void *vstart, void *vend)

Variables

```
    char end []
    struct {
        struct run * freelist
        struct spinlock lock
        int use_lock
    } kmem
```

5.83.1 Function Documentation

5.83.1.1 freerange()

Referenced by kinit1(), and kinit2().

5.83.1.2 kalloc()

```
char * kalloc (
     void )
```

Definition at line 83 of file kalloc.c.

```
00084 {
00085
       struct run *r;
00086
       if(kmem.use_lock)
00088
         acquire(&kmem.lock);
00089
       r = kmem.freelist;
00090
       if(r)
00091
         kmem.freelist = r->next;
00092
        if (kmem.use_lock)
00093
         release(&kmem.lock);
00094
       return (char*)r;
00095 }
```

Referenced by allocproc(), allocuvm(), copyuvm(), inituvm(), pipealloc(), setupkvm(), startothers(), and walkpgdir().

5.83.1.3 kfree()

Definition at line 60 of file kalloc.c.

```
00062
00063
        if((uint) v % PGSIZE || v < end || V2P(v) >= PHYSTOP)
00064
00065
          panic("kfree");
00066
00067
        // Fill with junk to catch dangling refs.
00068
        memset(v, 1, PGSIZE);
00069
00070
        if(kmem.use_lock)
00071
        acquire(&kmem.lock);
r = (struct run*)v;
00072
00073
        r->next = kmem.freelist;
00074
        kmem.freelist = r;
00075
        if (kmem.use_lock)
00076
           release(&kmem.lock);
00077 }
```

Referenced by allocuvm(), copyuvm(), deallocuvm(), fork(), freerange(), freevm(), pipealloc(), pipeclose(), and wait().

5.83.1.4 kinit1()

```
void kinit1 (
     void * vstart,
     void * vend )
```

Definition at line 32 of file kalloc.c.

5.83.1.5 kinit2()

Definition at line 40 of file kalloc.c.

```
00041 {
00042          freerange(vstart, vend);
00043          kmem.use_lock = 1;
00044 }
```

5.83.2 Variable Documentation

5.83.2.1 end

```
char end[] [extern]
```

Referenced by kfree().

5.83.2.2 freelist

```
struct run* freelist
```

Definition at line 23 of file kalloc.c.

5.83.2.3

```
struct { ... } kmem
```

Referenced by kalloc(), kfree(), kinit1(), and kinit2().

5.84 kalloc.c 231

5.83.2.4 lock

```
struct spinlock lock
```

Definition at line 21 of file kalloc.c.

5.83.2.5 use_lock

```
int use_lock
```

Definition at line 22 of file kalloc.c.

5.84 kalloc.c

Go to the documentation of this file.

```
00001 // Physical memory allocator, intended to allocate 00002 // memory for user processes, kernel stacks, page table pages,
00003 // and pipe buffers. Allocates 4096-byte pages.
00004
00005 #include "types.h"
00006 #include "defs.h"
00007 #include "param.h"
00000 #include "memlayout.h"
00009 #include "mmu.h"
00010 #include "spinlock.h"
00011
00012 void freerange(void *vstart, void *vend);
00013 extern char end[]; // first address after kernel loaded from ELF file 00014 // defined by the kernel linker script in kernel.ld
00015
00016 struct run {
00017
         struct run *next;
00018 };
00019
00020 struct {
00021 struct spinlock lock;
00022 int use_lock;
          struct run *freelist;
00023
00024 } kmem;
00025
00026 // Initialization happens in two phases.
00027 // 1. main() calls kinit1() while still using entrypgdir to place just 00028 // the pages mapped by entrypgdir on free list. 00029 // 2. main() calls kinit2() with the rest of the physical pages
00030 // after installing a full page table that maps them on all cores.
00031 void
00032 kinit1(void *vstart, void *vend)
00033 {
00034 initlock(&kmem.lock
00035 kmem.use_lock = 0;
         initlock(&kmem.lock, "kmem");
00036
         freerange(vstart, vend);
00037 }
00038
00039 void
00040 kinit2(void *vstart, void *vend)
00041 {
00042
         freerange(vstart, vend);
00043
         kmem.use_lock = 1;
00044 }
00045
00046 void
00047 freerange(void *vstart, void *vend)
00048 {
00049 char *p;
         p = (char*)PGROUNDUP((uint)vstart);
for(; p + PGSIZE <= (char*)vend; p += PGSIZE)</pre>
00050
00051
00052
           kfree(p);
00053 }
00054 //PAGEBREAK: 21
00055 // Free the page of physical memory pointed at by v_{\star}
```

```
00056 // which normally should have been returned by a
00057 // call to kalloc(). (The exception is when 00058 // initializing the allocator; see kinit above.)
00059 void
00060 kfree(char *v)
00061 {
00062
         struct run *r;
00063
00064
         if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP)
00065
          panic("kfree");
00066
        // Fill with junk to catch dangling refs.
memset(v, 1, PGSIZE);
00067
00068
00069
00070
         if(kmem.use_lock)
         acquire(&kmem.lock);
r = (struct run*)v;
00071
00072
00073
         r->next = kmem.freelist;
         kmem.freelist = r;
00075
         if(kmem.use_lock)
00076
          release(&kmem.lock);
00077 }
00078
00079 // Allocate one 4096-byte page of physical memory.
00080 // Returns a pointer that the kernel can use.
00081 // Returns 0 if the memory cannot be allocated.
00082 char*
00083 kalloc(void)
00084 {
         struct run *r;
00085
00086
00087
         if (kmem.use_lock)
00088
           acquire(&kmem.lock);
00089
         r = kmem.freelist;
00090
         if(r)
           kmem.freelist = r->next;
00091
00092
         if(kmem.use_lock)
           release(&kmem.lock);
00094
         return (char*)r;
00095 }
00096
```

5.85 kalloc.d File Reference

5.86 kalloc.d

```
Go to the documentation of this file.
```

5.87 kbd.c File Reference

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "kbd.h"
```

Functions

- · int kbdgetc (void)
- · void kbdintr (void)

5.87 kbd.c File Reference 233

5.87.1 Function Documentation

5.87.1.1 kbdgetc()

```
int kbdgetc (
     void )
```

Definition at line 7 of file kbd.c.

```
00009
         static uint shift;
00010
         static uchar *charcode[4] = {
00011
           normalmap, shiftmap, ctlmap, ctlmap
00012
00013
         uint st, data, c;
00014
00015
         st = inb(KBSTATP);
         if((st & KBS_DIB) == 0)
00016
00017
           return -1;
         data = inb(KBDATAP);
00018
00019
00020
        if (data == 0xE0) {
         shift |= E0ESC;
return 0;
00021
00022
        } else if(data & 0x80){
  // Key released
  data = (shift & E0ESC ? data : data & 0x7F);
00023
00024
00025
         shift &= ~(shiftcode[data] | EOESC);
00026
00027
           return 0;
00028
        } else if(shift & EOESC){
         // Last character was an E0 escape; or with 0x80
data |= 0x80;
00029
00030
00031
           shift &= ~E0ESC;
00032
00033
        shift |= shiftcode[data];
shift ^= togglecode[data];
00034
00035
        c = charcode[shift & (CTL | SHIFT)][data];
if(shift & CAPSLOCK){
00036
00037
        if('a' <= c && c <= 'z')
   c += 'A' - 'a';
else if('A' <= c && c <= 'Z')
   c += 'a' - 'A';</pre>
00038
00039
00040
00041
00042
00043
         return c;
00044 }
```

Referenced by kbdintr().

5.87.1.2 kbdintr()

```
void kbdintr (
     void )
```

Definition at line 47 of file kbd.c.

```
00048 {
00049          consoleintr(kbdgetc);
00050 }
```

Referenced by trap().

5.88 kbd.c

```
Go to the documentation of this file.
```

```
00001 #include "types.h'
00002 #include "x86.h"
00003 #include "defs.h"
00004 #include "kbd.h"
00005
00006 int
00007 kbdgetc(void)
00008 {
00009
         static uint shift:
         static uchar *charcode[4] = {
00010
00011
           normalmap, shiftmap, ctlmap, ctlmap
00012
00013
         uint st, data, c;
00014
         st = inb(KBSTATP);
00015
         if((st & KBS_DIB) == 0)
  return -1;
00016
00017
00018
         data = inb(KBDATAP);
00019
         if(data == 0xE0) {
    shift |= E0ESC;
    return 0;
00020
00021
            return 0;
00022
         } else if(data & 0x80){
         // Key released
data = (shift & E0ESC ? data : data & 0x7F);
shift &= ~(shiftcode[data] | E0ESC);
00024
00025
00026
        return 0;
} else if(shift & EOESC){
00027
00028
         // Last character was an EO escape; or with 0x80
00030
           data |= 0x80;
00031
           shift &= ~E0ESC;
00032
00033
         shift |= shiftcode[data];
shift ^= togglecode[data];
00034
00035
00036
         c = charcode[shift & (CTL | SHIFT)][data];
00037
         if(shift & CAPSLOCK) {
         if ('a' <= c && c <= 'z')
c += 'A' - 'a';
else if ('A' <= c && c <= 'Z')
c += 'a' - 'A';
00038
00039
00040
00042 }
00043 return c;
00044 }
00045
00046 void
00047 kbdintr(void)
00049
         consoleintr(kbdgetc);
00050 }
```

5.89 kbd.d File Reference

5.90 kbd.d

```
Go to the documentation of this file.
```

```
00001 kbd.o: kbd.c /usr/include/stdc-predef.h types.h x86.h defs.h kbd.h
```

5.91 kbd.h File Reference

Macros

- #define ALT (1<<2)
- #define C(x) (x '@')

5.91 kbd.h File Reference 235

- #define CAPSLOCK (1<<3)
- #define CTL (1<<1)
- #define E0ESC (1<<6)
- #define KBDATAP 0x60
- #define KBS_DIB 0x01
- #define KBSTATP 0x64
- #define KEY_DEL 0xE9
- #define KEY_DN 0xE3
- #define KEY_END 0xE1
- #define KEY HOME 0xE0
- #define KEY_INS 0xE8
- #define KEY_LF 0xE4
- #define KEY PGDN 0xE7
- #define KEY_PGUP 0xE6
- #define KEY_RT 0xE5
- #define KEY UP 0xE2
- #define NO 0
- #define NUMLOCK (1<<4)
- #define SCROLLLOCK (1<<5)
- #define SHIFT (1<<0)

Variables

- static uchar ctlmap [256]
- static uchar normalmap [256]
- static uchar shiftcode [256]
- static uchar shiftmap [256]
- static uchar togglecode [256]

5.91.1 Macro Definition Documentation

5.91.1.1 ALT

```
#define ALT (1 << 2)
```

Definition at line 11 of file kbd.h.

5.91.1.2 C

```
#define C( x ) (x - '@')
```

Definition at line 32 of file kbd.h.

5.91.1.3 CAPSLOCK

```
#define CAPSLOCK (1<<3)</pre>
```

Definition at line 13 of file kbd.h.

5.91.1.4 CTL

```
\#define CTL (1<<1)
```

Definition at line 10 of file kbd.h.

5.91.1.5 E0ESC

```
#define E0ESC (1 << 6)
```

Definition at line 17 of file kbd.h.

5.91.1.6 KBDATAP

#define KBDATAP 0x60

Definition at line 5 of file kbd.h.

5.91.1.7 KBS DIB

#define KBS_DIB 0x01

Definition at line 4 of file kbd.h.

5.91.1.8 KBSTATP

#define KBSTATP 0x64

Definition at line 3 of file kbd.h.

5.91 kbd.h File Reference 237

5.91.1.9 KEY_DEL

#define KEY_DEL 0xE9

Definition at line 29 of file kbd.h.

5.91.1.10 KEY_DN

#define KEY_DN 0xE3

Definition at line 23 of file kbd.h.

5.91.1.11 KEY_END

#define KEY_END 0xE1

Definition at line 21 of file kbd.h.

5.91.1.12 KEY_HOME

#define KEY_HOME 0xE0

Definition at line 20 of file kbd.h.

5.91.1.13 KEY_INS

#define KEY_INS 0xE8

Definition at line 28 of file kbd.h.

5.91.1.14 KEY_LF

#define KEY_LF 0xE4

Definition at line 24 of file kbd.h.

5.91.1.15 KEY_PGDN

#define KEY_PGDN 0xE7

Definition at line 27 of file kbd.h.

5.91.1.16 KEY_PGUP

#define KEY_PGUP 0xE6

Definition at line 26 of file kbd.h.

5.91.1.17 KEY_RT

#define KEY_RT 0xE5

Definition at line 25 of file kbd.h.

5.91.1.18 KEY_UP

#define KEY_UP 0xE2

Definition at line 22 of file kbd.h.

5.91.1.19 NO

#define NO 0

Definition at line 7 of file kbd.h.

5.91.1.20 NUMLOCK

#define NUMLOCK (1<<4)

Definition at line 14 of file kbd.h.

5.91 kbd.h File Reference 239

5.91.1.21 SCROLLLOCK

```
#define SCROLLLOCK (1<<5)
```

Definition at line 15 of file kbd.h.

5.91.1.22 SHIFT

```
\#define SHIFT (1<<0)
```

Definition at line 9 of file kbd.h.

5.91.2 Variable Documentation

5.91.2.1 ctlmap

```
uchar ctlmap[256] [static]
```

Initial value:

```
NO,
                                        NO,
                                                                                NO,
C('U'),
C('A'),
C('L'),
NO, NO,
C('Q'), C('W'),
C('O'), C('P'),
                           NO,
                                        NO,
                                                      NO,
                                                                   NO,
                                                                                              NO,
                                                                                             C('I'),
C('S'),
NO,
C('V'),
                                                   C('I'),
'\r',
C('J'),
                           C('E'),
                                        C('R'),
                                                                   C('Y'),
                                      NO, '\r', C('H'), C('J'), C('\\'), C('Z'),
                          NO,
                                                                   NO,
            C('F'),
                          C('G'),
NO,
C('D'),
C('B'),
             C('N'), C('M'), NO,
[0x9C] '\r',
[0xB5] C('/'),
[0xC8] KEY_UP, [0xD0] KEY_DN, [0xC9] KEY_PGUP, [0xD1] KEY_PGDN,
[0xCB] KEY_LF,
                           [0xCD] KEY_RT,
[0x97] KEY_HOME, [0xCF] KEY_END,
[0xD2] KEY_INS,
                           [0xD3] KEY_DEL
```

Definition at line 95 of file kbd.h.

Referenced by kbdgetc().

5.91.2.2 normalmap

```
uchar normalmap[256] [static]
```

Initial value:

```
'1',
'9',
'e',
'[',
'g',
        ′8′,
′w′,
'd', '1,
'\", '\", 1
'b', 'n',
                                           NO,
                NO,
                         NO,
                                  NO,
                                                   NO,
      NO,
                NO,
                         NO,
NO,
'8',
'2',
                                  NO,
                                          NO,
                                                   NO,
                                  NO,
                                          NO,
                                                   NO,
                                                           NO,
[0x9C] '\n',
[0xB5] '/',
 [0xC8] KEY_UP,
                          [0xD0] KEY_DN,
 [0xC9] KEY_PGUP,
                        [0xD1] KEY_PGDN,
[0xCB] KEY_LF, [0xCD] KEY_RT, [0x97] KEY_HOME, [0xCF] KEY_END,
[0xD2] KEY_INS,
                          [0xD3] KEY_DEL
```

Definition at line 51 of file kbd.h.

Referenced by kbdgetc().

5.91.2.3 shiftcode

```
uchar shiftcode[256] [static]
```

Initial value:

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

Definition at line 34 of file kbd.h.

Referenced by kbdgetc().

5.91.2.4 shiftmap

```
uchar shiftmap[256] [static]
```

Initial value:

5.92 kbd.h 241

```
'', NO,
                     NO,
                                           NO,
                            NO,
                                   NO,
NO,
'8',
'2',
      NO, NO, NO, '9', '-', '4', '3', '0', '.',
                                                  '7',
                                    NO,
                                                  NO,
[0x9C] '\n',
[0xB5] '/',
[0xC8] KEY_UP,
                     [0xD0] KEY_DN,
[0xC9] KEY_PGUP,
                     [0xD1] KEY_PGDN,
[0xCB] KEY_LF,
                      [0xCD] KEY_RT,
[0x97] KEY_HOME,
                      [0xCF] KEY END,
[0xD2] KEY_INS,
                     [0xD3] KEY_DEL
```

Definition at line 73 of file kbd.h.

Referenced by kbdgetc().

5.91.2.5 togglecode

```
uchar togglecode[256] [static]

Initial value:
=
{
  [0x3A] CAPSLOCK,
  [0x45] NUMLOCK,
  [0x46] SCROLLLOCK
```

Definition at line 44 of file kbd.h.

Referenced by kbdgetc().

5.92 kbd.h

Go to the documentation of this file.

```
00001 // PC keyboard interface constants
00002
                                       // kbd controller status port(I)
// kbd data in buffer
// kbd data port(I)
00003 #define KBSTATP
                                0x64
00004 #define KBS_DIB
                                0x01
00005 #define KBDATAP
                               0×60
00006
00007 #define NO
80000
00009 #define SHIFT
00010 #define CTL
                                (1«1)
00011 #define ALT
                                (1 < 2)
00012
00013 #define CAPSLOCK
                                (1«3)
00014 #define NUMLOCK
00015 #define SCROLLLOCK
00016
00017 #define E0ESC
                                (1 < 6)
00018
00019 // Special keycodes
00020 #define KEY_HOME
00021 #define KEY_END
00022 #define KEY_UP
                                0xE2
00023 #define KEY_DN
                                0xE3
00024 #define KEY_LF
                                0xE4
00025 #define KEY_RT
                                0xE5
00026 #define KEY_PGUP
                                0xE6
00027 #define KEY_PGDN
00028 #define KEY_INS
                                0xE8
00029 #define KEY_DEL
                                0xE9
00030
00031 // C('A') == Control-A
00032 #define C(x) (x - '@')
00033
```

```
00034 static uchar shiftcode[256] =
00035 {
00036
         [0x1D] CTL,
00037
         [0x2A] SHIFT,
00038
         [0x36] SHIFT,
00039
         [0x38] ALT,
         [0x9D] CTL,
00041
         [0xB8] ALT
00042 };
00043
00044 static uchar togglecode[256] =
00045 {
00046
         [0x3A] CAPSLOCK,
00047
         [0x45] NUMLOCK,
00048
         [0x46] SCROLLLOCK
00049 };
00050
00051 static uchar normalmap[256] =
00052 {
                                                    '5', '6', // 0x00
'\b', '\t',
'u', 'i', // 0x10
00053
                0x1B, '1',
        '\b',
                                             ' =' ,
00054
                                             'y',
00055
                                                    'a',
                                                           's',
                                            NO,
00056
                                                    'l',
                                             ′k′,
00057
                                                                  // 0x20
00058
                                                    NO,
00059
                                                                  // 0x30
00060
                                             NO,
                                                    NO,
                                                           NO,
                              NO,
                                            NO,
         NO, NO, NO,
'8', '9', '-',
'2', '3', '0',
                                                          '7',
'1',
00061
                                     NO,
                                                    NO,
                                                                   // 0x40
                                            '6',
                                                    '+',
                                     151,
00062
                                    NO, NO,
00063
                                                    NO.
                                                          NO.
                                                                  // 0x50
         [0x9C] '\n',
[0xB5] '/',
00064
                              // KP_Enter
00065
                               // KP_Div
00066
          [0xC8] KEY_UP,
                               [0xD0] KEY_DN,
00067
          [0xC9] KEY_PGUP,
                              [0xD1] KEY_PGDN,
         [0xCB] KEY_LF,
[0x97] KEY_HOME,
00068
                               [0xCD] KEY_RT,
                              [0xCF] KEY END.
00069
00070
         [0xD2] KEY_INS,
                              [0xD3] KEY_DEL
00071 };
00072
00073 static uchar shiftmap[256] =
00074 {
                                     '#',
'_'',
'_T',
'\n'
                                                    '%',
'\b',
'U',
                                                           '^',
'\t',
                               '@',
                                             '$',
00075
         NO,
                                                                  // 0x00
                033.
                                            '+',
'Y',
                       '(',
'E',
                              ')',
'R',
                ′ *′,
         '&',
00076
         'Q',
00077
                                                                 // 0x10
         'o',
                              '}',
00078
                'P',
                       ' {',
                                             NO,
                                                           's',
         'D',
                'F',
                                     'J',
'Z',
'>',
00079
                       'G',
                              'H',
                                             ′K′,
                                                    'L',
                                                                  // 0x20
                              '|',
'<',
00080
                       NO,
                                             'X',
                                                    'C',
                                                           'V'.
                                                           , <sub>*</sub>,
         'B',
               'N',
                       'M',
00081
                                                    NO,
                                                                  // 0x30
                              NO,
         NO,
                       NO,
                                     NO,
                                                           NO,
00082
                                            NO.
                                                    NO.
                       NO,
                NO,
         NO,
                              NO,
                                     NO,
00083
                                             NO.
                                                                  // 0x40
                                                    NO.
         NO, NO, NO, NO, '8', '9', '-', '2', '3', '0', [0x9C] '\n', [0xB5] '/', [0xC8] KEY_UP,
00084
                      ′0′,
                             '.', NO,
// KP_Enter
00085
                                                    NO,
                                                                   // 0x50
00086
00087
                               // KP_Div
00088
                               [0xD0] KEY_DN,
00089
         [0xC9] KEY_PGUP,
                              [0xD1] KEY_PGDN,
                               [0xCD] KEY_RT,
00090
         [0xCB] KEY_LF,
00091
         [0x97] KEY_HOME,
                               [0xCF] KEY_END,
00092
         [0xD2] KEY_INS,
                             [0xD3] KEY_DEL
00093 };
00094
00095 static uchar ctlmap[256] =
00096 {
                                         NO,
00097
                    NO,
                                                    NO,
00098
         NO,
                    NO,
                               NO,
                                         NO,
                                                    NO,
                                                               NO,
                                                                          NO,
                                                                                    NO,
         C('Q'), C('W'),
C('O'), C('P'),
                                       C('R'), C('T'),
NO, '\r',
C('H'), C('J'),
                                                               C('Y'),
                                                                          C('U'),
                                                                                    C('I'),
C('S'),
                              C('E'),
00099
                                                                         C('A'),
C('L'),
00100
                              NO,
C('G'),
                                                              NO,
C('K'),
         C('D'), C('F'),
00101
                                                                                    NO.
                                         C('\\'), C('Z'),
00102
                    NO,
                              NO,
                                                               C('X'), C('C'),
                                                                                    C('V'),
         NO.
         C('B'), C('N'
[0x9C] '\r',
[0xB5] C('/'),
                                                    NO,
                   C('N'),
                              C('M'),
                                         NO,
                                                               C('/'),
                                                                                    NO,
00103
                                                                         NO,
                              // KP_Enter
// KP_Div
00104
00105
         [0xC8] KEY_UP,
                              [0xD0] KEY_DN,
[0xD1] KEY_PGDN,
00106
         [0xC9] KEY_PGUP,
00107
00108
         [0xCB] KEY_LF,
                               [0xCD] KEY_RT,
         [0x97] KEY_HOME,
                               [0xCF] KEY_END,
00109
00110
         [0xD2] KEY_INS,
                               [0xD3] KEY_DEL
00111 };
00112
```

5.93 kill.c File Reference 243

5.93 kill.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

• int main (int argc, char **argv)

5.93.1 Function Documentation

5.93.1.1 main()

```
int main (
          int argc,
          char ** argv )
```

Definition at line 6 of file kill.c.

```
00007 {
00008    int i;
00009
00010    if(argc < 2) {
00011        printf(2, "usage: kill pid...\n");
00012        exit();
00013    }
00014    for(i=1; i<argc; i++)
00015        kill(atoi(argv[i]));
00016    exit();
00017 }</pre>
```

5.94 kill.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004
00005 int
00006 main(int argc, char **argv)
00007 {
00009
         if(argc < 2) {
  printf(2, "usage: kill pid...\n");
  exit();</pre>
00010
00011
00012
00013
00014
         for(i=1; i<argc; i++)</pre>
00015
         kill(atoi(argv[i]));
00016 exit();
00017 }
```

5.95 kill.d File Reference

5.96 kill.d

```
Go to the documentation of this file.
00001 kill.o: kill.c /usr/include/stdc-predef.h types.h stat.h user.h
```

5.97 lapic.c File Reference

```
#include "param.h"
#include "types.h"
#include "defs.h"
#include "date.h"
#include "memlayout.h"
#include "traps.h"
#include "mmu.h"
#include "x86.h"
```

Macros

- #define ASSERT 0x00004000
- #define BCAST 0x00080000
- #define BUSY 0x00001000
- #define CMOS PORT 0x70
- #define CMOS RETURN 0x71
- #define CMOS_STATA 0x0a
- #define CMOS_STATB 0x0b
- #define CMOS_UIP (1 << 7)
- #define CONV(x) (t1.x = ((t1.x >> 4) * 10) + (t1.x & 0xf))
- #define DAY 0x07
- #define DEASSERT 0x00000000
- #define DELIVS 0x00001000
- #define ENABLE 0x00000100
- #define EOI (0x00B0/4)
- #define ERROR (0x0370/4)
- #define ESR (0x0280/4)
- #define FIXED 0x00000000
- #define HOURS 0x04
- #define ICRHI (0x0310/4)
- #define ICRLO (0x0300/4)
- #define ID (0x0020/4)
- #define INIT 0x00000500
- #define LEVEL 0x00008000
- #define LINT0 (0x0350/4)
- #define LINT1 (0x0360/4)
- #define MASKED 0x00010000
- #define MINS 0x02
- #define MONTH 0x08
- #define PCINT (0x0340/4)

- #define PERIODIC 0x00020000
- #define SECS 0x00
- #define STARTUP 0x00000600
- #define SVR (0x00F0/4)
- #define TCCR (0x0390/4)
- #define TDCR (0x03E0/4)
- #define TICR (0x0380/4)
- #define TIMER (0x0320/4)
- #define TPR (0x0080/4)
- #define VER (0x0030/4)
- #define X1 0x0000000B
- #define YEAR 0x09

Functions

- static uint cmos_read (uint reg)
- void cmostime (struct rtcdate *r)
- static void fill_rtcdate (struct rtcdate *r)
- void lapiceoi (void)
- int lapicid (void)
- void lapicinit (void)
- void lapicstartap (uchar apicid, uint addr)
- static void lapicw (int index, int value)
- void microdelay (int us)

Variables

• volatile uint * lapic

5.97.1 Macro Definition Documentation

5.97.1.1 ASSERT

#define ASSERT 0x00004000

Definition at line 25 of file lapic.c.

5.97.1.2 BCAST

#define BCAST 0x00080000

Definition at line 28 of file lapic.c.

5.97.1.3 BUSY

#define BUSY 0x00001000

Definition at line 29 of file lapic.c.

5.97.1.4 CMOS_PORT

 $\#define CMOS_PORT 0x70$

Definition at line 123 of file lapic.c.

5.97.1.5 CMOS_RETURN

#define CMOS_RETURN 0x71

Definition at line 124 of file lapic.c.

5.97.1.6 CMOS_STATA

#define CMOS_STATA 0x0a

Definition at line 163 of file lapic.c.

5.97.1.7 CMOS STATB

#define CMOS_STATB 0x0b

Definition at line 164 of file lapic.c.

5.97.1.8 CMOS_UIP

#define CMOS_UIP (1 << 7)

Definition at line 165 of file lapic.c.

5.97.1.9 CONV

```
#define CONV(  x \ ) \ ({\tt t1.x} \ = \ (({\tt t1.x} \ >> \ 4) \ * \ 10) \ + \ ({\tt t1.x} \ \& \ 0{\tt xf}))
```

5.97.1.10 DAY

#define DAY 0x07

Definition at line 170 of file lapic.c.

5.97.1.11 DEASSERT

#define DEASSERT 0x0000000

Definition at line 26 of file lapic.c.

5.97.1.12 DELIVS

#define DELIVS 0x00001000

Definition at line 24 of file lapic.c.

5.97.1.13 ENABLE

#define ENABLE 0x00000100

Definition at line 19 of file lapic.c.

5.97.1.14 EOI

#define EOI (0x00B0/4)

Definition at line 17 of file lapic.c.

5.97.1.15 ERROR

```
#define ERROR (0x0370/4)
```

Definition at line 38 of file lapic.c.

5.97.1.16 ESR

```
\#define ESR (0x0280/4)
```

Definition at line 20 of file lapic.c.

5.97.1.17 FIXED

#define FIXED 0x00000000

Definition at line 30 of file lapic.c.

5.97.1.18 HOURS

#define HOURS 0x04

Definition at line 169 of file lapic.c.

5.97.1.19 ICRHI

#define ICRHI (0x0310/4)

Definition at line 31 of file lapic.c.

5.97.1.20 ICRLO

#define ICRLO (0x0300/4)

Definition at line 21 of file lapic.c.

5.97.1.21 ID

#define ID (0x0020/4)

Definition at line 14 of file lapic.c.

5.97.1.22 INIT

#define INIT 0x00000500

Definition at line 22 of file lapic.c.

5.97.1.23 LEVEL

#define LEVEL 0x00008000

Definition at line 27 of file lapic.c.

5.97.1.24 LINT0

#define LINTO (0x0350/4)

Definition at line 36 of file lapic.c.

5.97.1.25 LINT1

#define LINT1 (0x0360/4)

Definition at line 37 of file lapic.c.

5.97.1.26 MASKED

#define MASKED 0x00010000

Definition at line 39 of file lapic.c.

5.97.1.27 MINS

#define MINS 0x02

Definition at line 168 of file lapic.c.

5.97.1.28 MONTH

#define MONTH 0x08

Definition at line 171 of file lapic.c.

5.97.1.29 PCINT

#define PCINT (0x0340/4)

Definition at line 35 of file lapic.c.

5.97.1.30 PERIODIC

#define PERIODIC 0x00020000

Definition at line 34 of file lapic.c.

5.97.1.31 SECS

#define SECS 0x00

Definition at line 167 of file lapic.c.

5.97.1.32 STARTUP

#define STARTUP 0x00000600

Definition at line 23 of file lapic.c.

5.97.1.33 SVR

```
#define SVR (0x00F0/4)
```

Definition at line 18 of file lapic.c.

5.97.1.34 TCCR

```
\#define\ TCCR\ (0x0390/4)
```

Definition at line 41 of file lapic.c.

5.97.1.35 TDCR

```
#define TDCR (0x03E0/4)
```

Definition at line 42 of file lapic.c.

5.97.1.36 TICR

```
#define TICR (0x0380/4)
```

Definition at line 40 of file lapic.c.

5.97.1.37 TIMER

```
#define TIMER (0x0320/4)
```

Definition at line 32 of file lapic.c.

5.97.1.38 TPR

#define TPR (0x0080/4)

Definition at line 16 of file lapic.c.

5.97.1.39 VER

```
#define VER (0x0030/4)
```

Definition at line 15 of file lapic.c.

5.97.1.40 X1

```
#define X1 0x000000B
```

Definition at line 33 of file lapic.c.

5.97.1.41 YEAR

```
#define YEAR 0x09
```

Definition at line 172 of file lapic.c.

5.97.2 Function Documentation

5.97.2.1 cmos_read()

Definition at line 175 of file lapic.c.

```
00176 {
00177 outb(CMOS_PORT, reg);
00178 microdelay(200);
00179
00180 return inb(CMOS_RETURN);
00181 }
```

Referenced by cmostime(), and fill_rtcdate().

5.97.2.2 cmostime()

```
void cmostime (
               struct rtcdate * r )
Definition at line 196 of file lapic.c.
00198
        struct rtcdate t1, t2;
00199
        int sb, bcd;
00200
00201
        sb = cmos_read(CMOS_STATB);
00202
00203
        bcd = (sb & (1 & 2)) == 0;
00204
00205
        // make sure CMOS doesn't modify time while we read it
00206
        for(;;) {
        fill_rtcdate(&t1);
00207
         if(cmos_read(CMOS_STATA) & CMOS_UIP)
    continue;
00208
00209
         fill_rtcdate(&t2);
00210
         if(memcmp(&t1, &t2, sizeof(t1)) == 0)
00211
00212
            break;
        }
00213
00214
00215
        // convert
       if (bcd) {
00216
00217 #define
                  CONV(x)
                                (t1.x = ((t1.x \gg 4) * 10) + (t1.x & 0xf))
00217 #define con.,
00218 CONV(second);
00219 CONV(minute);
          CONV (hour );
CONV (day );
00220
         CONV (day
00221
        CONV (month );
CONV (year );
00222
00223
00224 #undef
                  CONV
00225 }
00226
00227 *r = t1;
00228 r->year += 2000;
00229 }
```

5.97.2.3 fill_rtcdate()

```
00185 {
00186    r->second = cmos_read(SECS);
00187    r->minute = cmos_read(MINS);
00188    r->hour = cmos_read(HOURS);
00189    r->day = cmos_read(DAY);
00190    r->month = cmos_read(MONTH);
00191    r->year = cmos_read(YEAR);
```

Referenced by cmostime().

5.97.2.4 lapiceoi()

```
void lapiceoi (
     void )
```

Definition at line 110 of file lapic.c.

```
00111 {
00112    if(lapic)
00113         lapicw(EOI, 0);
00114 }
```

Referenced by trap().

5.97.2.5 lapicid()

Referenced by mycpu(), and panic().

5.97.2.6 lapicinit()

00106 }

```
void lapicinit (
     void )
```

Definition at line 55 of file lapic.c.

```
00057
          if(!lapic)
00058
             return;
00059
          // Enable local APIC; set spurious interrupt vector.
lapicw(SVR, ENABLE | (T_IRQ0 + IRQ_SPURIOUS));
00060
00061
00062
00063
          \ensuremath{//} The timer repeatedly counts down at bus frequency
          // from lapic[TICR] and then issues an interrupt.
// If xv6 cared more about precise timekeeping,
00064
00065
00066
          // TICR would be calibrated using an external time source.
          lapicw(TDCR, X1);
lapicw(TIMER, PERIODIC | (T_IRQ0 + IRQ_TIMER));
lapicw(TICR, 10000000);
00067
00068
00069
00070
          // Disable logical interrupt lines.
lapicw(LINTO, MASKED);
lapicw(LINT1, MASKED);
00071
00072
00073
00074
00075
          // Disable performance counter overflow interrupts
00076
          \ensuremath{//} on machines that provide that interrupt entry.
00077
          if(((lapic[VER] \gg 16) \& 0xFF) >= 4)
00078
             lapicw(PCINT, MASKED);
00079
          // Map error interrupt to IRQ_ERROR.
lapicw(ERROR, T_IRQ0 + IRQ_ERROR);
08000
00081
00082
00083
          // Clear error status register (requires back-to-back writes).
          lapicw(ESR, 0);
00085
          lapicw(ESR, 0);
00086
00087
          // Ack any outstanding interrupts.
00088
          lapicw(EOI, 0);
00089
          // Send an Init Level De-Assert to synchronise arbitration {\tt ID's.}
00090
00091
          lapicw(ICRHI, 0);
lapicw(ICRLO, BCAST | INIT | LEVEL);
while(lapic[ICRLO] & DELIVS)
00092
00093
00094
00095
          \ensuremath{//} Enable interrupts on the APIC (but not on the processor).
00096
00097
          lapicw(TPR, 0);
00098 }
```

Referenced by mpenter().

5.97.2.7 lapicstartap()

```
void lapicstartap (
                uchar apicid,
                uint addr )
Definition at line 129 of file lapic.c.
00131
00132
        ushort *wrv;
00133
         // "The BSP must initialize CMOS shutdown code to OAH
00134
00135
         // and the warm reset vector (DWORD based at 40:67) to point at
         // the AP startup code prior to the [universal startup algorithm]."
00136
        outb(CMOS_PORT, 0xF); // offset 0xF is shutdown code
outb(CMOS_PORT+1, 0x0A);
00137
00138
        wrv = (ushort*)P2V((0x40«4 | 0x67)); // Warm reset vector
wrv[0] = 0;
wrv[1] = addr >> 4;
00139
00140
00141
00142
00143
         // "Universal startup algorithm."
00144
         // Send INIT (level-triggered) interrupt to reset other CPU.
00145
         lapicw(ICRHI, apicid«24);
00146
        lapicw(ICRLO, INIT | LEVEL | ASSERT);
        microdelay(200);
lapicw(ICRLO, INIT | LEVEL);
00147
00148
00149
        microdelay(100);
                               // should be 10ms, but too slow in Bochs!
00150
00151
         // Send startup IPI (twice!) to enter code.
00152
         // Regular hardware is supposed to only accept a STARTUP \,
        // when it is in the halted state due to an INIT. So the second // should be ignored, but it is part of the official Intel algorithm.
00153
00154
         // Bochs complains about the second one. Too bad for Bochs. for(i = 0; i < 2; i++) {
00155
00156
00157
           lapicw(ICRHI, apicid«24);
           lapicw(ICRLO, STARTUP | (addr»12));
00158
00159
           microdelay(200);
00160
00161 }
```

Referenced by startothers().

5.97.2.8 lapicw()

```
static void lapicw (
                int index,
                int value ) [static]
```

Definition at line 48 of file lapic.c.

```
00049 {
00050    lapic[index] = value;
00051    lapic[ID]; // wait for write to finish, by reading
00052 }
```

Referenced by lapiceoi(), lapicinit(), and lapicstartap().

5.97.2.9 microdelay()

00120 { 00121 }

Referenced by cmos_read(), lapicstartap(), and uartputc().

5.97.3 Variable Documentation

5.97.3.1 lapic

```
volatile uint* lapic
```

Definition at line 44 of file lapic.c.

Referenced by lapiceoi(), lapicid(), lapicinit(), lapicw(), and mpinit().

5.98 lapic.c

Go to the documentation of this file.

```
00001 // The local APIC manages internal (non-I/O) interrupts.
00002 // See Chapter 8 & Appendix C of Intel processor manual volume 3.
00003
00004 #include "param.h"
00005 #include "types.h"
00006 #include "defs.h'
00007 #include "date.h"
00008 #include "memlayout.h"
00009 #include "traps.h"
00010 #include "mmu.h"
00011 #include "x86.h"
00012
00013 // Local APIC registers, divided by 4 for use as uint[] indices.
00014 #define ID (0x0020/4) // ID
00015 #define VER
                            (0x0030/4)
                                           // Version
                                          // Task Priority
// EOI
// Spurious Inte
00016 #define TPR
                            (0x0080/4)
00017 #define EOI
                          (0x00B0/4)
                                           // Spurious Interrupt Vector
00018 #define SVR
                           (0x00F0/4)
00019
         #define ENABLE
                               0x00000100
                                                 // Unit Enable
00020 #define ESR (0x0280/4) // Error Status
00021 #define ICRLO (0x0300/4) // Interrupt Command
                                 0x00000500 // INIT/RESET
0x00000600 // Startup IPI
0x00001000 // Delivery status
0x00004000 // Assert interrupt (vs deassert)
00022 #define INIT
00023
         #define STARTUP
00024
         #define DELIVS
00025
         #define ASSERT
        #define DEASSERT 0x00000000
#define LEVEL 0x00008000
00026
00027
                                                  // Level triggered
00028
         #define BCAST
                                 0x00080000
                                                  // Send to all APICs, including self.
00029
         #define BUSY
                                 0x00001000
         #define FIXED
00030
                                 0×00000000
00031 #define ICRHI (0x0310/4) // Interrupt Command [63:32]
00032 #define TIMER (0x0320/4) // Local Vector Table 0 (TIMER)
         00033 #define X1
00034
00034 #define PERIODIC 0x0020000 // Perfordic

00035 #define PCINT (0x0340/4) // Performance Counter LVT

00036 #define LINT0 (0x0350/4) // Local Vector Table 1 (LINT0)

00037 #define LINT1 (0x0360/4) // Local Vector Table 2 (LINT1)

00038 #define ERROR (0x0370/4) // Local Vector Table 3 (ERROR)
00039
         #define MASKED
                                 0x00010000
                                                  // Interrupt masked
00040 #define TICR (0x0380/4) // Timer Initial Count
00041 #define TCCR (0x0390/4) // Timer Current Count
                                          // Timer Divide Configuration
00042 #define TDCR
                           (0x03E0/4)
00043
00044 volatile uint *lapic; // Initialized in mp.c
00046 //PAGEBREAK!
00047 static void
00048 lapicw(int index, int value)
00049 {
00050
         lapic[index] = value;
         lapic[ID]; // wait for write to finish, by reading
00052 }
00053
00054 void
00055 lapicinit (void)
00056 {
00057
         if(!lapic)
00058
            return;
```

5.98 lapic.c 257

```
00060
         // Enable local APIC; set spurious interrupt vector.
00061
         lapicw(SVR, ENABLE | (T_IRQ0 + IRQ_SPURIOUS));
00062
         // The timer repeatedly counts down at bus frequency // from lapic[TICR] and then issues an interrupt.
00063
00064
         // If xv6 cared more about precise timekeeping,
00066
         // TICR would be calibrated using an external time source.
00067
         lapicw(TDCR, X1);
         lapicw(TIMER, PERIODIC | (T_IRQ0 + IRQ_TIMER));
lapicw(TICR, 10000000);
00068
00069
00070
00071
         // Disable logical interrupt lines.
         lapicw(LINT0, MASKED);
lapicw(LINT1, MASKED);
00072
00073
00074
00075
         // Disable performance counter overflow interrupts
00076
         // on machines that provide that interrupt entry. if(((lapic[VER]»16) & 0xFF) >= 4)
00077
00078
           lapicw(PCINT, MASKED);
00079
08000
         // Map error interrupt to IRQ_ERROR.
00081
         lapicw(ERROR, T_IRQ0 + IRQ_ERROR);
00082
00083
         // Clear error status register (requires back-to-back writes).
00084
         lapicw(ESR, 0);
00085
         lapicw(ESR, 0);
00086
00087
         // Ack any outstanding interrupts.
         lapicw(EOI, 0);
00088
00089
00090
         // Send an Init Level De-Assert to synchronise arbitration ID's.
         lapicw(ICRHI, 0);
lapicw(ICRLO, BCAST | INIT | LEVEL);
00091
00092
00093
         while(lapic[ICRLO] & DELIVS)
00094
00095
00096
         // Enable interrupts on the APIC (but not on the processor).
00097
         lapicw(TPR, 0);
00098 }
00099
00100 int.
00101 lapicid(void)
00102 {
00103
        if (!lapic)
00104
           return 0;
00105
        return lapic[ID] » 24;
00106 }
00107
00108 // Acknowledge interrupt.
00109 void
00110 lapiceoi(void)
00111 {
00112
        if(lapic)
           lapicw(EOI, 0);
00113
00114 }
00116 // Spin for a given number of microseconds.
00117 // On real hardware would want to tune this dynamically.
00118 void
00119 microdelay(int us)
00120 {
00121
00122
00123 #define CMOS_PORT
                               0x70
00124 #define CMOS_RETURN 0x71
00125
00126 // Start additional processor running entry code at addr. 00127 // See Appendix B of MultiProcessor Specification.
00128 void
00129 lapicstartap(uchar apicid, uint addr)
00130 {
00131
         int i;
00132
        ushort *wrv;
00133
00134
         // "The BSP must initialize CMOS shutdown code to OAH
00135
         // and the warm reset vector (DWORD based at 40:67) to point at
00136
         // the AP startup code prior to the [universal startup algorithm]."
         outb(CMOS_PORT, 0xF); // offset 0xF is shutdown code
outb(CMOS_PORT+1, 0x0A);
wrv = (ushort*)P2V((0x40«4 | 0x67)); // Warm reset vector
00137
00138
00139
         wrv[0] = 0;
wrv[1] = addr » 4;
00140
00141
00142
00143
         // "Universal startup algorithm."  
        /// Send INIT (level-triggered) interrupt to reset other CPU.
lapicw(ICRHI, apicid«24);
00144
00145
```

```
lapicw(ICRLO, INIT | LEVEL | ASSERT);
00147
        microdelay(200);
        lapicw(ICRLO, INIT | LEVEL);
00148
00149
        microdelay(100);
                             // should be 10ms, but too slow in Bochs!
00150
00151
        // Send startup IPI (twice!) to enter code.
        // Regular hardware is supposed to only accept a STARTUP
00153
        // when it is in the halted state due to an INIT. So the second
00154
        \ensuremath{//} should be ignored, but it is part of the official Intel algorithm.
00155
        \ensuremath{//} Bochs complains about the second one. Too bad for Bochs.
        for(i = 0; i < 2; i++) {
    lapicw(ICRHI, apicid«24);
    lapicw(ICRLO, STARTUP | (addr»12));</pre>
00156
00157
00158
00159
         microdelay(200);
00160 }
00161 }
00162
00163 #define CMOS_STATA
                            0x0a
00164 #define CMOS_STATB 0x0b
00165 #define CMOS_UIP (1 « 7)
                                       // RTC update in progress
00166
00167 #define SECS
                       0x00
00168 #define MINS
                       0x02
00169 #define HOURS
                       0 \times 0.4
00170 #define DAY
                       0x07
00171 #define MONTH
                       0x08
00172 #define YEAR
                       0x09
00173
00174 static uint
00175 cmos_read(uint reg)
00176 {
00177 outb(CMOS_PORT, reg);
00178 microdelay(200);
00179
00180
       return inb(CMOS_RETURN);
00181 }
00182
00183 static void
00184 fill_rtcdate(struct rtcdate *r)
00185 {
00186
       r->second = cmos_read(SECS);
       r->minute = cmos_read(MINS);
00187
       r->hour = cmos_read(HOURS);
r->day = cmos_read(DAY);
00188
00189
00190
       r->month = cmos_read(MONTH);
00191
        r->year = cmos_read(YEAR);
00192 }
00193
00194 // gemu seems to use 24-hour GWT and the values are BCD encoded
00195 void
00196 cmostime(struct rtcdate *r)
00197 {
00198
       struct rtcdate t1, t2;
00199
       int sb, bcd;
00200
00201
       sb = cmos read(CMOS STATB);
00202
00203
        bcd = (sb & (1 & 2)) == 0;
00204
00205
        // make sure CMOS doesn't modify time while we read it
00206
        for(;;) {
        fill_rtcdate(&t1);
00207
00208
         if(cmos_read(CMOS_STATA) & CMOS_UIP)
00209
              continue;
00210
         fill_rtcdate(&t2);
00211
        if(memcmp(&t1, &t2, sizeof(t1)) == 0)
00212
            break;
00213
00214
        // convert
00216
        if (bcd) {
00217 #define
                 CONV(x)
                              (t1.x = ((t1.x \gg 4) * 10) + (t1.x & 0xf))
        CONV (second);
00218
          CONV(minute);
00219
          CONV (hour );
00220
00221
          CONV (day
00222
          CONV (month );
00223
         CONV(year);
                 CONV
00224 #undef
00225
       }
00226
        *r = t1;
00228 r->year += 2000;
00229 }
```

5.99 lapic.d File Reference

5.100 lapic.d

Go to the documentation of this file.

```
00001 lapic.o: lapic.c /usr/include/stdc-predef.h param.h types.h defs.h date.h \setminus 00002 memlayout.h traps.h mmu.h x86.h
```

5.101 In.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

int main (int argc, char *argv[])

5.101.1 Function Documentation

5.101.1.1 main()

int main (

5.102 In.c

00014 exit(); 00015 }

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004
00005 int
00006 main(int argc, char *argv[])
00007 {
00008    if(argc != 3) {
        printf(2, "Usage: ln old new\n");
00010        exit();
00011    }
00012    if(link(argv[1], argv[2]) < 0)
00013        printf(2, "link %s %s: failed\n", argv[1], argv[2]);
00014    exit();
00015 }</pre>
```

5.103 In.d File Reference

5.104 In.d

```
Go to the documentation of this file.
```

00001 ln.o: ln.c /usr/include/stdc-predef.h types.h stat.h user.h

5.105 log.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "fs.h"
#include "buf.h"
```

Classes

- struct log
- struct logheader

Functions

- void begin_op (void)
- static void commit ()
- void end_op (void)
- void initlog (int dev)
- static void install_trans (void)
- void log_write (struct buf *b)
- static void read_head (void)
- static void recover_from_log (void)
- static void write_head (void)
- static void write_log (void)

Variables

• struct log log

5.105.1 Function Documentation

5.105.1.1 begin_op()

```
void begin_op (
     void )
```

Definition at line 126 of file log.c.

```
00127 {
00128
         acquire(&log.lock);
00129
         while(1){
00130
           if(log.committing){
             sleep(&log, &log.lock);
           } else if(log.lh.n + (log.outstanding+1)*MAXOPBLOCKS > LOGSIZE){
// this op might exhaust log space; wait for commit.
00132
00133
00134
             sleep(&log, &log.lock);
00135
           } else {
00136
              log.outstanding += 1;
00137
              release(&log.lock);
00138
              break;
00139
00140 }
00141 }
```

Referenced by exec(), exit(), fileclose(), filewrite(), sys_chdir(), sys_link(), sys_mkdir(), sys_mknod(), sys_open(), and sys_unlink().

5.105.1.2 commit()

```
static void commit ( ) [static]
```

Definition at line 193 of file log.c.

```
00194 {
00195
        if (log.lh.n > 0) {
                          . // Write modified blocks from cache to log
00196
         write_log();
00197
         write_head();
                          // Write header to disk -- the real commit
00198
          install_trans(); // Now install writes to home locations
00199
         log.lh.n = 0;
         write_head();
                         // Erase the transaction from the log
00200
00201
00202 }
```

Referenced by end_op().

5.105.1.3 end_op()

```
void end_op (
     void )
```

Definition at line 146 of file log.c.

```
00147
00148
          int do_commit = 0;
00149
00150
          acquire(&log.lock);
00151
          log.outstanding -= 1;
00152
          if(log.committing)
            panic("log.committing");
00153
          if (log.outstanding == 0) {
  do_commit = 1;
00154
00155
00156
            log.committing = 1;
00157
          } else {
           // begin_op() may be waiting for log space,
// and decrementing log.outstanding has decreased
// the amount of reserved space.
00158
00159
00160
00161
            wakeup(&log);
00162
```

```
release(&log.lock);
00164
00165
        if (do_commit) {
        // call commit w/o holding locks, since not allowed
// to sleep with locks.
00166
00167
00168
          commit();
00169
          acquire(&log.lock);
00170
          log.committing = 0;
00171
          wakeup(&log);
00172
          release(&log.lock);
        }
00173
00174 }
```

Referenced by exec(), exit(), fileclose(), filewrite(), sys_chdir(), sys_link(), sys_mkdir(), sys_mknod(), sys_open(), and sys_unlink().

5.105.1.4 initlog()

```
void initlog ( \quad \text{int } \textit{dev} \; )
```

Definition at line 54 of file log.c.

```
00055 {
00056     if (sizeof(struct logheader) >= BSIZE)
00057          panic("initlog: too big logheader");
00058          struct superblock sb;
00060          initlock(&log.lock, "log");
00061          readsb(dev, &sb);
00062          log.start = sb.logstart;
00063          log.size = sb.nlog;
00064          log.dev = dev;
00065          recover_from_log();
00066 }
```

Referenced by forkret().

5.105.1.5 install_trans()

Definition at line 70 of file log.c.

```
00071 {
00072
            int tail;
00073
00074
            for (tail = 0; tail < log.lh.n; tail++) {</pre>
           struct buf *lbuf = bread(log.dev, log.start+tail+1); // read log block
struct buf *dbuf = bread(log.dev, log.lh.block[tail]); // read dst
memmove(dbuf->data, lbuf->data, BSIZE); // copy block to dst
00075
00076
00077
              bwrite(dbuf); // write dst to disk
00079
              brelse(lbuf);
08000
              brelse(dbuf);
00081
00082 }
```

Referenced by commit(), and recover_from_log().

5.105.1.6 log_write()

```
void log_write (
                  struct buf * b )
Definition at line 214 of file log.c.
00216
00217
         if (log.lh.n >= LOGSIZE || log.lh.n >= log.size - 1)
  panic("too big a transaction");
if (log.outstanding < 1)
  panic("log_write outside of trans");</pre>
00218
00219
00220
00221
00222
00223
         acquire(&log.lock);
00224
         for (i = 0; i < log.lh.n; i++) {</pre>
          if (log.lh.block[i] == b->blockno) // log absorbtion
00225
00226
              break;
00227
         log.lh.block[i] = b->blockno;
00228
00229
         if (i == log.lh.n)
           log.lh.n++;
00230
        b->flags |= B_DIRTY; // prevent eviction
00231
00232 release(&log.lock);
00233 }
```

Referenced by balloc(), bfree(), bmap(), bzero(), ialloc(), iupdate(), and writei().

5.105.1.7 read_head()

Definition at line 86 of file log.c.

Referenced by recover_from_log().

5.105.1.8 recover from log()

Definition at line 116 of file log.c.

```
00117 {
00118     read_head();
00119     install_trans(); // if committed, copy from log to disk
00120     log.lh.n = 0;
00121     write_head(); // clear the log
00122 }
```

Referenced by initlog().

5.105.1.9 write_head()

```
static void write_head (
                    void ) [static]
Definition at line 102 of file log.c.
00103 4
          struct buf *buf = bread(log.dev, log.start);
struct logheader *hb = (struct logheader *) (buf->data);
00104
00106
          hb->n = log.lh.n;

for (i = 0; i < log.lh.n; i++) {

  hb->block[i] = log.lh.block[i];
00107
00108
00109
00110
           bwrite(buf);
00112
         brelse(buf);
00113 }
```

Referenced by commit(), and recover_from_log().

5.105.1.10 write_log()

Definition at line 178 of file log.c.

```
00180
           int tail;
00181
           for (tail = 0; tail < log.lh.n; tail++) {
   struct buf *to = bread(log.dev, log.start+tail+1); // log block
   struct buf *from = bread(log.dev, log.lh.block[tail]); // cache block</pre>
00182
00183
00184
              memmove(to->data, from->data, BSIZE);
00186
              bwrite(to); // write the log
00187
              brelse(from);
00188
              brelse(to);
00189
00190 }
```

Referenced by commit().

5.105.2 Variable Documentation

5.105.2.1 log

```
struct log log
```

Definition at line 48 of file log.c.

5.106 log.c 265

5.106 log.c

Go to the documentation of this file.

```
00001 #include "types.h 00002 #include "defs.h"
00003 #include "param.h"
00004 #include "spinlock.h"
00005 #include "sleeplock.h"
00006 #include "fs.h"
00007 #include "buf.h"
80000
00009 // Simple logging that allows concurrent FS system calls.
00010 //
00011 // A log transaction contains the updates of multiple FS system
00012 // calls. The logging system only commits when there are
00013 \ensuremath{//} no FS system calls active. Thus there is never
00014 \ensuremath{//} any reasoning required about whether a commit might
00015 // write an uncommitted system call's updates to disk.
00016 //
00017 // A system call should call begin_op()/end_op() to mark
00018 // its start and end. Usually begin_op() just increments
00019 \ensuremath{//} the count of in-progress FS system calls and returns.
00020 // But if it thinks the log is close to running out, it
00021 // sleeps until the last outstanding end_op() commits.
00023 // The log is a physical re-do log containing disk blocks.
00024 // The on-disk log format:
00025 //
           header block, containing block #s for block A, B, C, ...
00026 //
            block A
00027 //
            block B
00028 //
           block C
00029 //
00030 // Log appends are synchronous.
00031
00032 // Contents of the header block, used for both the on-disk header block
00033 // and to keep track in memory of logged block# before commit.
00034 struct logheader {
00035 int n;
00036 int block[LOGSIZE];
00037 };
00038
00039 struct log {
00040
        struct spinlock lock;
00041
        int start;
00042
        int size;
00043
        int outstanding; // how many FS sys calls are executing.
00044
        int committing; // in commit(), please wait.
00045
        int dev;
00046
        struct logheader lh;
00047 };
00048 struct log log;
00049
00050 static void recover_from_log(void);
00051 static void commit();
00052
00053 void
00054 initlog(int dev)
00055 {
00056
        if (sizeof(struct logheader) >= BSIZE)
00057
         panic("initlog: too big logheader");
00058
00059
        struct superblock sb;
        initlock(&log.lock, "log");
00060
00061
        readsb(dev, &sb);
        log.start = sb.logstart;
log.size = sb.nlog;
00062
00063
        log.dev = dev:
00064
00065
        recover_from_log();
00066 }
00067
00068 // Copy committed blocks from log to their home location
00069 static void
00070 install_trans(void)
00071 {
00072
        int tail;
00074
         for (tail = 0; tail < log.lh.n; tail++) {</pre>
          struct buf *lbuf = bread(log.dev, log.start+tail+1); // read log block
struct buf *dbuf = bread(log.dev, log.lh.block[tail]); // read dst
memmove(dbuf->data, lbuf->data, BSIZE); // copy block to dst
00075
00076
00077
00078
           bwrite(dbuf); // write dst to disk
00079
           brelse(lbuf);
08000
           brelse(dbuf);
00081
00082 }
```

```
00084 // Read the log header from disk into the in-memory log header
00085 static void
00086 read_head(void)
00087 {
00088
        struct buf *buf = bread(log.dev, log.start);
        struct logheader *lh = (struct logheader *) (buf->data);
00090
00091
        log.lh.n = lh->n;
        for (i = 0; i < log.lh.n; i++) {
  log.lh.block[i] = lh->block[i];
}
00092
00093
00094
00095
       brelse(buf);
00095
00097
00098 // Write in-memory log header to disk.
00099 // This is the true point at which the 00100 // current transaction commits.
00101 static void
00102 write_head(void)
00103 {
00104
        struct buf *buf = bread(log.dev, log.start);
        struct logheader *hb = (struct logheader *) (buf->data);
00105
00106
        int i;
00107
        hb \rightarrow n = log.lh.n;
        for (i = 0; i < log.lh.n; i++) {
00108
00109
          hb->block[i] = log.lh.block[i];
00110
00111
        bwrite(buf);
00112 brelse(buf);
00113 }
00114
00115 static void
00116 recover_from_log(void)
00117 {
00118
        read_head();
00119
        install_trans(); // if committed, copy from log to disk
        log.lh.n = 0;
00121
        write_head(); // clear the log
00122 }
00123
00124 // called at the start of each FS system call.
00125 void
00126 begin_op(void)
00127 {
00128
        acquire(&log.lock);
00129
        while(1){
00130
         if(log.committing){
          sleep(&log, &log.lock);
} else if(log.lh.n + (log.outstanding+1)*MAXOPBLOCKS > LOGSIZE){
00131
00132
00133
            // this op might exhaust log space; wait for commit.
00134
            sleep(&log, &log.lock);
00135
          } else {
00136
           log.outstanding += 1;
00137
            release(&log.lock);
00138
            break;
00139
00140
00141 }
00142
00143 // called at the end of each FS system call.
00144 // commits if this was the last outstanding operation.
00145 void
00146 end_op(void)
00147 {
00148
        int do_commit = 0;
00149
00150
        acquire(&log.lock);
00151
        log.outstanding -= 1;
        if(log.committing)
00152
00153
          panic("log.committing");
00154
        if(log.outstanding == 0){
         do_commit = 1;
00155
00156
          log.committing = 1;
00157
00158
         // begin_op() may be waiting for log space,
00159
          // and decrementing log.outstanding has decreased
           // the amount of reserved space.
00160
00161
          wakeup(&log);
00162
00163
        release(&log.lock);
00164
00165
         if (do_commit) {
          // call commit w/o holding locks, since not allowed
00166
           // to sleep with locks.
00167
00168
          commit();
00169
          acquire(&log.lock);
```

```
log.committing = 0;
00171
          wakeup(&log);
00172
          release(&log.lock);
       }
00173
00174 }
00175
00176 // Copy modified blocks from cache to log.
00177 static void
00178 write_log(void)
00179 {
00180
        int tail:
00181
00182
        for (tail = 0; tail < log.lh.n; tail++) {</pre>
        struct buf *to = bread(log.dev, log.start+tail+1); // log block
00183
00184
         struct buf *from = bread(log.dev, log.lh.block[tail]); // cache block
          memmove(to->data, from->data, BSIZE);
00185
00186
         bwrite(to); // write the log
         brelse(from);
00187
        brelse(to);
00188
00189
        }
00190 }
00191
00192 static void
00193 commit()
00194 {
00195
        if (log.lh.n > 0) {
                          // Write modified blocks from cache to log
// Write header to disk -- the real commit
00196
         write_log();
00197
          write_head();
          install_trans(); // Now install writes to home locations
00198
00199
          log.lh.n = 0;
00200
                          // Erase the transaction from the log
         write head();
00201
        }
00202 }
00203
00204 \// Caller has modified b->data and is done with the buffer.
00205 // Record the block number and pin in the cache with B_DIRTY.
00206 // commit()/write_log() will do the disk write.
00208 // log_write() replaces bwrite(); a typical use is:
00209 // bp = bread(...)
00210 //
           modify bp->data[]
           log_write(bp)
00211 //
00212 //
           brelse(bp)
00213 void
00214 log_write(struct buf *b)
00215 {
00216
00217
        if (log.lh.n >= LOGSIZE || log.lh.n >= log.size - 1)
00218
        panic("too big a transaction");
if (log.outstanding < 1)</pre>
00219
00221
        panic("log_write outside of trans");
00222
00223
        acquire(&log.lock);
        for (i = 0; i < log.lh.n; i++) {
  if (log.lh.block[i] == b->blockno) // log absorbtion
00224
00225
            break;
00227
00228
       log.lh.block[i] = b->blockno;
00229
       if (i == log.lh.n)
         log.lh.n++;
00230
       b->flags |= B_DIRTY; // prevent eviction
00231
00232
       release(&log.lock);
00233 }
00234
```

5.107 log.d File Reference

5.108 log.d

```
Go to the documentation of this file.
```

```
00001 log.o: log.c /usr/include/stdc-predef.h types.h defs.h param.h spinlock.h \ 00002 sleeplock.h fs.h buf.h
```

5.109 Is.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fs.h"
```

Functions

- char * fmtname (char *path)
- void ls (char *path)
- int main (int argc, char *argv[])

5.109.1 Function Documentation

5.109.1.1 fmtname()

```
char * fmtname (
                char * path )
Definition at line 7 of file Is.c.
00008 {
00009
         static char buf[DIRSIZ+1];
00010
        char *p;
00011
00012
         // Find first character after last slash.
        for(p=path+strlen(path); p >= path && *p != '/'; p--)
00013
00014
00015
00016
        // Return blank-padded name.
if (strlen(p) >= DIRSIZ)
00017
00018
00019
          return p;
        memmove(buf, p, strlen(p));
memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
00020
00021
00022
        return buf;
00023 }
```

5.109.1.2 ls()

Referenced by Is().

5.110 ls.c 269

```
return;
00036
00037
          if(fstat(fd, &st) < 0){
  printf(2, "ls: cannot stat %s\n", path);
  close(fd);</pre>
00038
00039
00040
00041
            return;
00042
00043
00044
          switch(st.type) {
00045
          case T_FILE:
          printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);
00046
00047
            break;
00048
00049
           if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){
  printf(1, "ls: path too long\n");
00050
00051
00052
               break;
00053
            strcpy(buf, path);
p = buf+strlen(buf);
*p++ = '/';
while(read(fd, &de, sizeof(de)) == sizeof(de)){
   if(de.inum == 0)
00054
00055
00056
00057
00058
00059
                 continue;
00060
               memmove(p, de.name, DIRSIZ);
p[DIRSIZ] = 0;
00061
               if(stat(buf, &st) < 0){
  printf(1, "ls: cannot stat %s\n", buf);</pre>
00062
00063
00064
                  continue;
00065
00066
               printf(1, "%s %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);
00067
00068
            break;
00069
00070
         close(fd);
00071 }
```

Referenced by main().

5.109.1.3 main()

```
int main (
          int argc,
          char * argv[] )
```

Definition at line 74 of file Is.c.

```
00075 {
00076
         int i;
00077
        if(argc < 2){
   ls(".");</pre>
00078
00079
00080
           exit();
00081
00082
        for(i=1; i<argc; i++)</pre>
00083
          ls(argv[i]);
00084
        exit();
00085 }
```

5.110 ls.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004 #include "fs.h"
00005
00006 char*
00007 fmtname(char *path)
00008 {
00009 static char buf[DIRSIZ+1];
```

```
00010
        char *p;
00011
00012
         // Find first character after last slash.
         for(p=path+strlen(path); p >= path && *p != '/'; p--)
00013
00014
00015
         p++;
00016
00017
         // Return blank-padded name.
00018
        if(strlen(p) >= DIRSIZ)
00019
          return p;
        memmove(buf, p, strlen(p));
memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
00020
00021
00022
        return buf;
00023 }
00024
00025 void
00026 ls(char *path)
00027 {
00028
        char buf[512], *p;
00029
         int fd;
00030
        struct dirent de;
00031
         struct stat st;
00032
         if((fd = open(path, 0)) < 0){
  printf(2, "ls: cannot open %s\n", path);
  return;</pre>
00033
00034
00035
           return;
00036
00037
         if(fstat(fd, &st) < 0){
  printf(2, "ls: cannot stat %s\n", path);
  close(fd);</pre>
00038
00039
00040
00041
           return;
00042
00043
00044
         switch(st.type) {
00045
        printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);
hreak:
         case T_FILE:
00046
00047
           break;
00048
00049
           if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf) {
00050
00051
             printf(1, "ls: path too long\n");
00052
             break:
00053
           strcpy(buf, path);
p = buf+strlen(buf);
00054
00055
            *p++ = '/';
00056
           while(read(fd, &de, sizeof(de)) == sizeof(de)){
00057
00058
             if(de.inum == 0)
00059
               continue;
             memmove(p, de.name, DIRSIZ);
p[DIRSIZ] = 0;
00060
00061
              if(stat(buf, &st) < 0){
  printf(1, "ls: cannot stat %s\n", buf);</pre>
00062
00063
00064
                continue;
00065
00066
             printf(1, "%s %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);
00067
00068
           break;
00069
00070
        close (fd):
00071 }
00072
00073 int
00074 main(int argc, char *argv[])
00075 {
00076
        int i;
00077
         if(argc < 2){
   ls(".");
   exit();</pre>
00078
00079
00080
00081
00082
         for(i=1; i<argc; i++)</pre>
00083
          ls(argv[i]);
00084
        exit();
00085 }
```

5.111 Is.d File Reference 271

5.111 Is.d File Reference

5.112 Is.d

```
Go to the documentation of this file.
```

```
00001 ls.o: ls.c /usr/include/stdc-predef.h types.h stat.h user.h fs.h
```

5.113 main.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
```

Functions

- __attribute__ ((__aligned__(PGSIZE)))
- static void mpenter (void)
- static void mpmain (void)
- static void startothers (void)

Variables

• pde_t entrypgdir []

5.113.1 Function Documentation

```
5.113.1.1 __attribute__()
```

```
__attribute__ (
                     (__aligned__(PGSIZE)) )
Definition at line 102 of file main.c.
            // Map VA's [0, 4MB) to PA's [0, 4MB)
          [0] = (0) | PTE_P | PTE_W | PTE_PS,

// Map VA's [KERNBASE, KERNBASE+4MB) to PA's [0, 4MB)

[KERNBASE»PDXSHIFT] = (0) | PTE_P | PTE_W | PTE_PS,
00105
00106
00107
```

00108 };

5.113.1.2 mpenter()

00045

00046

00047

00048 }

Referenced by startothers().

seginit();

mpmain();

lapicinit();

5.113.1.3 mpmain()

Definition at line 10 of file main.c.

```
00019 {
00020
       kinit1(end, P2V(4*1024*1024)); // phys page allocator
       00021
       mpinit();
lapicinit();
00022
                     // interrupt controller
00023
00024
       seginit();
                       // segment descriptors
00025
       picinit();
                        // disable pic
       ioapicinit();  // another interrupt controller
consoleinit();  // console hardware
uartinit();  // serial port
00026
00027
00028
                        // serial port
// process table
// trap vectors
00029
       pinit();
       tvinit();
00030
00031
       binit();
                        // buffer cache
                       // file table
// disk
// start other processors
       fileinit();
00032
00033
       ideinit();
       00034
00035
00036
00037
00038 }
```

Referenced by mpenter().

5.113.1.4 startothers()

Definition at line 64 of file main.c.

```
00065 {
00066
        extern uchar _binary_entryother_start[], _binary_entryother_size[];
00067
        uchar *code;
00068
        struct cpu *c;
00069
       char *stack;
00070
00071
        // Write entry code to unused memory at 0x7000.
00072
       // The linker has placed the image of entryother.S in
       // _binary_entryother_start.
code = P2V(0x7000);
00073
00074
00075
       memmove(code, _binary_entryother_start, (uint)_binary_entryother_size);
00076
00077
       for(c = cpus; c < cpus+ncpu; c++) {</pre>
00078
       if(c == mycpu()) // We've started already.
```

5.114 main.c 273

```
continue;
08000
00081
            // Tell entryother.S what stack to use, where to enter, and what
           // pgdir to use. We cannot use kpgdir yet, because the AP processor // is running in low memory, so we use entrypgdir for the APs too.
00082
00083
00084
            stack = kalloc();
            *(void**)(code-4) = stack + KSTACKSIZE;
00086
            \star (void(\star\star) (void)) (code-8) = mpenter;
00087
            *(int**)(code-12) = (void *) V2P(entrypgdir);
00088
00089
            lapicstartap(c->apicid, V2P(code));
00090
00091
            // wait for cpu to finish mpmain()
00092
           while(c->started == 0)
00093
00094
00095 }
```

5.113.2 Variable Documentation

5.113.2.1 entrypgdir

```
pde_t entrypgdir[]
```

Definition at line 60 of file main.c.

Referenced by startothers().

5.114 main.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "defs.h"
00002 #Include dels.n
00003 #include "param.h"
00004 #include "memlayout.h"
00005 #include "mmu.h"
00006 #include "proc.h"
00007 #include "x86.h"
80000
00009 static void startothers (void);
00010 static void mpmain(void) __attribute__((noreturn));
00011 extern pde_t *kpgdir;
00012 extern char end[]; // first address after kernel loaded from ELF file
00014 // Bootstrap processor starts running {\tt C} code here.
00015 // Allocate a real stack and switch to it, first
00016 // doing some setup required for memory allocator to work.
00017 int
00018 main(void)
00019 {
00020
        kinit1(end, P2V(4*1024*1024)); // phys page allocator
        00021
00022
        mpinit();
lapicinit();
                            // interrupt controller
00023
                            // segment descriptors
00024
        seginit();
00025
                            // disable pic
        picinit();
                            // another interrupt controller
// console hardware
00026
        ioapicinit();
00027
        consoleinit();
                            // serial port
00028
        uartinit();
        pinit();
                            // process table
00029
                            // trap vectors
00030
         tvinit();
                            // buffer cache
00031
        binit();
        fileinit();  // file table
ideinit();  // disk
startothers();  // start other processors
kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come after startothers()
00032
00033
00034
00035
00036
        userinit(); // first user process
                            // finish this processor's setup
00037
        mpmain();
```

```
00038 }
00039
00040 // Other CPUs jump here from entryother.S.
00041 static void
00042 mpenter (void)
00043 {
00044
00045
        seginit();
00046
        lapicinit();
00047
        mpmain();
00048 }
00049
00050 // Common CPU setup code.
00051 static void
00052 mpmain(void)
00053 {
        cprintf("cpu%d: starting %dn", cpuid(), cpuid());
00054
00055
        idtinit();  // load idt register
xchg(&(mycpu()->started), 1); // tell startothers() we're up
00057
        scheduler();
                           // start running processes
00058 }
00059
00060 pde_t entrypgdir[]; // For entry.S
00061
00062 // Start the non-boot (AP) processors.
00063 static void
00064 startothers(void)
00065 {
00066
        extern uchar _binary_entryother_start[], _binary_entryother_size[];
00067
        uchar *code;
struct cpu *c;
00068
00069
        char *stack;
00070
00071
        // Write entry code to unused memory at 0x7000.
        // The linker has placed the image of entryother.S in
// _binary_entryother_start.
code = P2V(0x7000);
00072
00073
00074
        memmove(code, _binary_entryother_start, (uint)_binary_entryother_size);
00076
        for(c = cpus; c < cpus+ncpu; c++){
  if(c == mycpu()) // We've started already.</pre>
00077
00078
00079
             continue:
00080
00081
          // Tell entryother.S what stack to use, where to enter, and what
          // pgdir to use. We cannot use kpgdir yet, because the AP processor // is running in low memory, so we use entrypgdir for the APs too.
00082
00083
          stack = kalloc();
*(void**) (code-4) = stack + KSTACKSIZE;
00084
00085
00086
           *(void(**)(void))(code-8) = mpenter;
           *(int**)(code-12) = (void *) V2P(entrypgdir);
00087
00088
00089
           lapicstartap(c->apicid, V2P(code));
00090
00091
           // wait for cpu to finish \ensuremath{\mathsf{mpmain}}\,()
00092
          while(c->started == 0)
00093
            ;
00094
00095 }
00096
00097 \ensuremath{//} The boot page table used in entry.S and entryother.S.
00098 \!\!\!// Page directories (and page tables) must start on page boundaries,
00099 // hence the __aligned__ attribute.
00100 // PTE_PS in a page directory entry enables 4Mbyte pages.
00101
00102
        _attribute__((__aligned__(PGSIZE)))
// Map VA's [KERNBASE, KERNBASE+4MB) to PA's [0, 4MB)
00106
        [KERNBASE»PDXSHIFT] = (0) | PTE_P | PTE_W | PTE_PS,
00107
00108 };
00109
00110 //PAGEBREAK!
00111 // Blank page.
00112 //PAGEBREAK!
00113 // Blank page.
00114 //PAGEBREAK!
00115 // Blank page.
00116
```

5.115 main.d File Reference

5.116 main.d

Go to the documentation of this file.

```
00001 main.o: main.c /usr/include/stdc-predef.h types.h defs.h param.h \setminus 00002 memlayout.h mmu.h proc.h x86.h
```

5.117 memide.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
#include "traps.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "fs.h"
#include "buf.h"
```

Functions

- void ideinit (void)
- · void ideintr (void)
- void iderw (struct buf *b)

Variables

- uchar _binary_fs_img_size []
- uchar _binary_fs_img_start []
- · static int disksize
- static uchar * memdisk

5.117.1 Function Documentation

5.117.1.1 ideinit()

```
void ideinit (
     void )
```

Definition at line 22 of file memide.c.

5.117.1.2 ideintr()

```
void ideintr (
    void )
```

Definition at line 30 of file memide.c.

```
00031 {
00032 // no-op
00033 }
```

Referenced by trap().

5.117.1.3 iderw()

```
void iderw ( \label{eq:struct_buf} \text{struct buf } *\ b \ )
```

Definition at line 39 of file memide.c.

```
00040 {
00041
00042
00043
         if(!holdingsleep(&b->lock))
         panic("iderw: buf not locked");
if((b->flags & (B_VALID|B_DIRTY)) == B_VALID)
panic("iderw: nothing to do");
00044
00045
00046
         if(b->dev != 1)
  panic("iderw: request not for disk 1");
00047
00048
         if(b->blockno >= disksize)
00049
00050
00051
          panic("iderw: block out of range");
00052
        p = memdisk + b->blockno*BSIZE;
00053
        if(b->flags & B_DIRTY) {
         b->flags &= ~B_DIRTY;
00055
00056
            memmove(p, b->data, BSIZE);
         } else
00057
00058 memmove(b->data, p, BSIZE);
00059 b->flags |= B_VALID;
00060 }
```

Referenced by bread(), and bwrite().

5.117.2 Variable Documentation

5.117.2.1 _binary_fs_img_size

```
uchar _binary_fs_img_size[]
```

Definition at line 16 of file memide.c.

Referenced by ideinit().

5.118 memide.c 277

5.117.2.2 _binary_fs_img_start

```
uchar _binary_fs_img_start[] [extern]
```

Referenced by ideinit().

5.117.2.3 disksize

```
int disksize [static]
```

Definition at line 18 of file memide.c.

Referenced by ideinit(), and iderw().

5.117.2.4 memdisk

```
uchar* memdisk [static]
```

Definition at line 19 of file memide.c.

Referenced by ideinit(), and iderw().

5.118 memide.c

Go to the documentation of this file.

```
00001 // Fake IDE disk; stores blocks in memory.
00002 // Useful for running kernel without scratch disk.
00003
00004 #include "types.h"
00005 #include "defs.h"
00006 #include "param.h"
00007 #include "mmu.h"
00007 #Include "mind.h"
00008 #include "proc.h"
00009 #include "x86.h"
00010 #include "traps.h"
00011 #include "spinlock.h"
00012 #include "sleeplock.h"
00013 #include "fs.h"
00014 #include "buf.h"
00015
00016 extern uchar _binary_fs_img_start[], _binary_fs_img_size[];
00017
00018 static int disksize;
00019 static uchar *memdisk;
00020
00021 void
00022 ideinit (void)
00023 {
00024 memdisk = _binary_fs_img_start;
00025 disksize = (uint)_binary_fs_img_size/BSIZE;
00026 }
00027
00028 // Interrupt handler.
00029 void
00030 ideintr(void)
00031 {
00032
          // no-op
00033 }
00034
00035 // Sync buf with disk.
```

```
00036 // If B_DIRTY is set, write buf to disk, clear B_DIRTY, set B_VALID.
00037 // Else if B_VALID is not set, read buf from disk, set B_VALID.
00038 void
00039 iderw(struct buf *b)
00040 {
00041
        uchar *p;
00042
00043
       if(!holdingsleep(&b->lock))
00044
         panic("iderw: buf not locked");
       if((b->flags & (B_VALID|B_DIRTY)) == B_VALID)
panic("iderw: nothing to do");
00045
00046
       if(b->dev != 1)
00047
         panic("iderw: request not for disk 1");
00048
       if(b->blockno >= disksize)
00049
00050
        panic("iderw: block out of range");
00051
00052 p = memdisk + b->blockno*BSIZE;
00053
00054 if (b->flags & B_DIRTY) {
       b->flags &= ~B_DIRTY;
memmove(p, b->data, BSIZE);
00055
00056
00057
       } else
         memmove(b->data, p, BSIZE);
00058
00059
       b->flags |= B_VALID;
00060 }
```

5.119 memlayout.h File Reference

Macros

- #define DEVSPACE 0xFE000000
- #define EXTMEM 0x100000
- #define KERNBASE 0x80000000
- #define KERNLINK (KERNBASE+EXTMEM)
- #define P2V(a) ((void *)(((char *) (a)) + KERNBASE))
- #define P2V_WO(x) ((x) + KERNBASE)
- #define PHYSTOP 0xE000000
- #define V2P(a) (((uint) (a)) KERNBASE)
- #define V2P_WO(x) ((x) KERNBASE)

5.119.1 Macro Definition Documentation

5.119.1.1 DEVSPACE

#define DEVSPACE 0xFE000000

Definition at line 5 of file memlayout.h.

5.119.1.2 EXTMEM

#define EXTMEM 0x100000

Definition at line 3 of file memlayout.h.

5.119.1.3 KERNBASE

```
#define KERNBASE 0x8000000
```

Definition at line 8 of file memlayout.h.

5.119.1.4 KERNLINK

```
#define KERNLINK (KERNBASE+EXTMEM)
```

Definition at line 9 of file memlayout.h.

5.119.1.5 P2V

Definition at line 12 of file memlayout.h.

5.119.1.6 P2V_WO

```
#define P2V_WO( x ) ((x) + KERNBASE)
```

Definition at line 15 of file memlayout.h.

5.119.1.7 PHYSTOP

```
#define PHYSTOP 0xE000000
```

Definition at line 4 of file memlayout.h.

5.119.1.8 V2P

Definition at line 11 of file memlayout.h.

5.119.1.9 V2P_WO

```
#define V2P_W0( x ) ((x) - KERNBASE)
```

Definition at line 14 of file memlayout.h.

5.120 memlayout.h

Go to the documentation of this file.

```
00001 // Memory layout
00002
00003 #define EXTMEM 0x100000
                                               // Start of extended memory
                                            // Top physical memory
// Other devices are at high addresses
00004 #define PHYSTOP 0xE000000
00005 #define DEVSPACE 0xFE000000
00006
00007 // Key addresses for address space layout (see kmap in vm.c for layout)
00008 #define KERNBASE 0x80000000 // First kernel virtual address 00009 #define KERNLINK (KERNBASE+EXTMEM) // Address where kernel is linked
00010
00011 #define V2P(a) (((uint) (a)) - KERNBASE)
00012 \#define P2V(a) ((void *)(((char *) (a)) + KERNBASE))
00013
00014 \#define V2P\_WO(x) ((x) - KERNBASE) // same as V2P, but without casts
00015 \#define P2V_WO(x) ((x) + KERNBASE) // same as P2V, but without casts
```

5.121 mkdir.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

• int main (int argc, char *argv[])

5.121.1 Function Documentation

5.121.1.1 main()

```
int main (
                int argc,
                char * argv[])
Definition at line 6 of file mkdir.c.
00007 {
80000
         int i:
00009
00010
         if (argc < 2) {</pre>
        printf(2, "Usage: mkdir files...\n");
00011
00012
00013
00014
00015
         for(i = 1; i < argc; i++) {</pre>
         if(mkdir(argv[i]) < 0){
  printf(2, "mkdir: %s failed to create\n", argv[i]);</pre>
00017
00018
00019
00020
00021
00022
        exit();
00023 }
```

5.122 mkdir.c 281

5.122 mkdir.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004
00005 int
00006 main(int argc, char *argv[])
00007 {
00008
         int i;
00009
00010
          if(argc < 2){
         printf(2, "Usage: mkdir files...\n");
exit():
00011
00012
            exit();
00013
00014
         for(i = 1; i < argc; i++) {
   if(mkdir(argv[i]) < 0) {
     printf(2, "mkdir: %s failed to create\n", argv[i]);</pre>
00015
00016
00017
00018
00019
00020
00021
00022
         exit();
00023 }
```

5.123 mkdir.d File Reference

5.124 mkdir.d

Go to the documentation of this file.

00001 mkdir.o: mkdir.c /usr/include/stdc-predef.h types.h stat.h user.h

5.125 mkfs.c File Reference

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <assert.h>
#include "types.h"
#include "fs.h"
#include "stat.h"
#include "param.h"
```

Macros

- #define min(a, b) ((a) < (b) ? (a) : (b))
- #define NINODES 200
- · #define stat xv6 stat
- #define static_assert(a, b) do { switch (0) case 0: case (a): ; } while (0)

Functions

- void balloc (int)
- uint ialloc (ushort type)
- void iappend (uint inum, void *p, int n)
- int main (int argc, char *argv[])
- void rinode (uint inum, struct dinode *ip)
- void rsect (uint sec, void *buf)
- void winode (uint, struct dinode *)
- void wsect (uint, void *)
- uint xint (uint x)
- ushort xshort (ushort x)

Variables

- uint freeblock
- uint freeinode = 1
- int fsfd
- int nbitmap = FSSIZE/(BSIZE*8) + 1
- int nblocks
- int ninodeblocks = NINODES / IPB + 1
- int nlog = LOGSIZE
- int nmeta
- struct superblock sb
- char zeroes [BSIZE]

5.125.1 Macro Definition Documentation

5.125.1.1 min

Definition at line 253 of file mkfs.c.

5.125.1.2 NINODES

#define NINODES 200

Definition at line 18 of file mkfs.c.

5.125.1.3 stat

```
#define stat xv6_stat
```

Definition at line 8 of file mkfs.c.

5.125.1.4 static_assert

Definition at line 15 of file mkfs.c.

5.125.2 Function Documentation

5.125.2.1 balloc()

```
void balloc ( \quad \text{ int } used \ )
```

Definition at line 238 of file mkfs.c.

```
00239 {
00240
         uchar buf[BSIZE];
00241
        int i;
00242
00243
        printf("balloc: first %d blocks have been allocated\n", used);
        assert(used < BSIZE*8);
bzero(buf, BSIZE);
for(i = 0; i < used; i++){</pre>
00244
00245
00246
00247
          buf[i/8] = buf[i/8] | (0x1 « (i%8));
00248
00249
        printf("balloc: write bitmap block at sector %d\n", sb.bmapstart);
00250 wsect(sb.bmapstart, buf);
00251 }
```

Referenced by main().

5.125.2.2 ialloc()

Definition at line 224 of file mkfs.c.

```
00225 {
00226
         uint inum = freeinode++;
00227
         struct dinode din;
00228
00229
         bzero(&din, sizeof(din));
        din.type = xshort(type);
din.nlink = xshort(1);
00230
00231
00232
         din.size = xint(0);
00233
         winode(inum, &din);
00234
         return inum;
00235 }
```

Referenced by main().

5.125.2.3 iappend()

```
void iappend (
                  uint inum,
                  void *p,
                  int n)
Definition at line 256 of file mkfs.c.
00257 {
         char *p = (char*)xp;
uint fbn, off, n1;
struct dinode din;
00258
00259
00260
00261
         char buf[BSIZE];
00262
         uint indirect[NINDIRECT];
00263
         uint x;
00264
00265
         rinode(inum, &din);
         off = xint(din.size);
00266
         oil = xint(din.size);
// printf("append inum %d at off %d sz %d\n", inum, off, n);
while(n > 0) {
  fbn = off / BSIZE;
  assert(fbn < MAXFILE);</pre>
00267
00268
00269
00270
            if(fbn < NDIRECT) {</pre>
00271
00272
              if(xint(din.addrs[fbn]) == 0){
00273
                 din.addrs[fbn] = xint(freeblock++);
00274
00275
              x = xint(din.addrs[fbn]);
00276
            } else {
00277
              if(xint(din.addrs[NDIRECT]) == 0){
                din.addrs[NDIRECT] = xint(freeblock++);
00278
00279
00280
               rsect(xint(din.addrs[NDIRECT]), (char*)indirect);
               if(indirect[fbn - NDIRECT] == 0){
  indirect[fbn - NDIRECT] = xint(freeblock++);
  wsect(xint(din.addrs[NDIRECT]), (char*)indirect);
00281
00282
00283
00284
00285
              x = xint(indirect[fbn-NDIRECT]);
00286
00287
            n1 = min(n, (fbn + 1) * BSIZE - off);
            rsect(x, buf);
bcopy(p, buf + off - (fbn * BSIZE), n1);
00288
00289
00290
            wsect(x, buf);
00291
            n -= n1;
00292
            off += n1;
00293
            p += n1;
00294
00295
         din.size = xint(off);
         winode(inum, &din);
00296
```

Referenced by main().

00297 }

5.125.2.4 main()

```
int main (
          int argc,
          char * argv[] )
```

Definition at line 68 of file mkfs.c.

```
00069 {
00070
       int i, cc, fd;
00071
       uint rootino, inum, off;
00072
       struct dirent de:
00073
       char buf[BSIZE];
00074
       struct dinode din;
00075
00076
       static_assert(sizeof(int) == 4, "Integers must be 4 bytes!");
00077
00078
00079
       if(argc < 2){
08000
       fprintf(stderr, "Usage: mkfs fs.img files...\n");
00081
         exit(1);
```

```
00082
        }
00083
00084
        assert((BSIZE % sizeof(struct dinode)) == 0);
00085
        assert((BSIZE % sizeof(struct dirent)) == 0);
00086
00087
        fsfd = open(argv[1], O_RDWR|O_CREAT|O_TRUNC, 0666);
        if(fsfd < 0){
00089
          perror(argv[1]);
00090
          exit(1);
00091
00092
00093
        // 1 fs block = 1 disk sector
        nmeta = 2 + nlog + ninodeblocks + nbitmap;
nblocks = FSSIZE - nmeta;
00094
00095
00096
00097
        sb.size = xint(FSSIZE);
00098
        sb.nblocks = xint(nblocks);
sb.ninodes = xint(NINODES);
00099
        sb.nlog = xint(nlog);
00100
00101
        sb.logstart = xint(2);
00102
        sb.inodestart = xint(2+nlog);
00103
        sb.bmapstart = xint(2+nlog+ninodeblocks);
00104
        printf("nmeta %d (boot, super, log blocks %u inode blocks %u, bitmap blocks %u) blocks %d total
00105
        %d\n",
00106
                nmeta, nlog, ninodeblocks, nbitmap, nblocks, FSSIZE);
00107
00108
        freeblock = nmeta;
                                 // the first free block that we can allocate
00109
00110
        for(i = 0; i < FSSIZE; i++)</pre>
00111
          wsect(i, zeroes);
00112
00113
        memset(buf, 0, sizeof(buf));
00114
        memmove(buf, &sb, sizeof(sb));
00115
        wsect(1, buf);
00116
00117
        rootino = ialloc(T DIR);
00118
        assert(rootino == ROOTINO);
00119
00120
        bzero(&de, sizeof(de));
        de.inum = xshort(rootino);
strcpy(de.name, ".");
00121
00122
        iappend(rootino, &de, sizeof(de));
00123
00124
00125
        bzero(&de, sizeof(de));
00126
        de.inum = xshort(rootino);
00127
        strcpy(de.name, "..");
00128
        iappend(rootino, &de, sizeof(de));
00129
00130
        for(i = 2; i < argc; i++) {</pre>
00131
          assert(index(argv[i], '/') == 0);
00132
00133
           if((fd = open(argv[i], 0)) < 0){</pre>
00134
           perror(argv[i]);
00135
             exit(1);
          }
00136
00138
           // Skip leading _ in name when writing to file system.
00139
           // The binaries are named \_{\rm rm}, \_{\rm cat}, etc. to keep the
00140
           // build operating system from trying to execute them
          // in place of system binaries like rm and cat.
if(argv[i][0] == '_')
00141
00142
00143
             ++argv[i];
00144
00145
           inum = ialloc(T_FILE);
00146
00147
           bzero(&de, sizeof(de));
00148
           de.inum = xshort(inum);
          strncpy(de.name, argv[i], DIRSIZ);
iappend(rootino, &de, sizeof(de));
00149
00150
00151
00152
          while((cc = read(fd, buf, sizeof(buf))) > 0)
00153
            iappend(inum, buf, cc);
00154
00155
          close(fd);
00156
00157
00158
        // fix size of root inode \operatorname{dir}
00159
        rinode(rootino, &din);
00160
        off = xint(din.size);
        off = ((off/BSIZE) + 1) * BSIZE;
00161
        din.size = xint(off);
00162
00163
        winode (rootino, &din);
00164
00165
        balloc(freeblock);
00166
00167
        exit(0);
```

```
00168 }
```

5.125.2.5 rinode()

```
void rinode (
                uint inum,
                struct dinode * ip )
Definition at line 198 of file mkfs.c.
00199 {
00200
        char buf[BSIZE];
00201
        uint bn;
00202
        struct dinode *dip;
00203
00204
        bn = IBLOCK(inum, sb);
        rsect(bn, buf);
dip = ((struct dinode*)buf) + (inum % IPB);
*ip = *dip;
00205
00206
00207
00208 }
```

Referenced by iappend(), and main().

5.125.2.6 rsect()

00214 perror("lseek");
00215 exit(1);
00216 }
00217 if(read(fsfd, buf, BSIZE) != BSIZE) {
00218 perror("read");
00219 exit(1);
00220 }
00221 }

Referenced by iappend(), rinode(), and winode().

5.125.2.7 winode()

```
void winode (
          uint inum,
          struct dinode * ip )
```

Definition at line 184 of file mkfs.c.

```
00185 {
00186
        char buf[BSIZE];
        uint bn;
struct dinode *dip;
00187
00188
00189
00190
        bn = IBLOCK(inum, sb);
00191
        rsect(bn, buf);
        dip = ((struct dinode*)buf) + (inum % IPB);
*dip = *ip;
00192
00193
00194
        wsect(bn, buf);
00195 }
```

Referenced by ialloc(), iappend(), and main().

5.125.2.8 wsect()

Referenced by balloc(), iappend(), main(), and winode().

5.125.2.9 xint()

00181 }

```
uint xint (
          uint x )
```

Definition at line 56 of file mkfs.c.

Referenced by ialloc(), iappend(), and main().

5.125.2.10 xshort()

```
ushort xshort (
     ushort x )
```

Definition at line 46 of file mkfs.c.

```
00047 {
00048     ushort y;
00049     uchar *a = (uchar*)&y;
00050     a[0] = x;
00051     a[1] = x » 8;
00052     return y;
00053 }
```

Referenced by ialloc(), and main().

5.125.3 Variable Documentation

5.125.3.1 freeblock

```
uint freeblock
```

Definition at line 33 of file mkfs.c.

Referenced by iappend(), and main().

5.125.3.2 freeinode

```
uint freeinode = 1
```

Definition at line 32 of file mkfs.c.

Referenced by ialloc().

5.125.3.3 fsfd

int fsfd

Definition at line 29 of file mkfs.c.

Referenced by main(), rsect(), and wsect().

5.125.3.4 nbitmap

```
int nbitmap = FSSIZE/(BSIZE*8) + 1
```

Definition at line 23 of file mkfs.c.

Referenced by main().

5.125.3.5 nblocks

int nblocks

Definition at line 27 of file mkfs.c.

Referenced by main().

5.125.3.6 ninodeblocks

```
int ninodeblocks = NINODES / IPB + 1
```

Definition at line 24 of file mkfs.c.

Referenced by main().

5.125.3.7 nlog

```
int nlog = LOGSIZE
```

Definition at line 25 of file mkfs.c.

Referenced by main().

5.125.3.8 nmeta

int nmeta

Definition at line 26 of file mkfs.c.

Referenced by main().

5.125.3.9 sb

```
struct superblock sb
```

Definition at line 30 of file mkfs.c.

Referenced by balloc(), main(), rinode(), and winode().

5.125.3.10 zeroes

```
char zeroes[BSIZE]
```

Definition at line 31 of file mkfs.c.

Referenced by main().

5.126 mkfs.c

Go to the documentation of this file.

```
00001 #include <stdio.h>
00002 #include <unistd.h>
00003 #include <stdlib.h>
00004 #include <string.h>
00005 #include <fcntl.h>
00006 #include <assert.h>
00007
00008 \#define stat xv6_stat // avoid clash with host struct stat
00009 #include "types.h"
00010 #include "fs.h"
00011 #include "stat.h"
00012 #include "param.h"
00013
00014 #ifndef static_assert
00015 #define static_assert(a, b) do { switch (0) case 0: case (a): ; } while (0)
00016 #endif
00017
00018 #define NINODES 200
00019
00020 // Disk layout:
00021 // [ boot block | sb block | log | inode blocks | free bit map | data blocks ]
00023 int nbitmap = FSSIZE/(BSIZE*8) + 1;
00024 int ninodeblocks = NINODES / IPB + 1;
00025 int nlog = LOGSIZE;
00026 int nmeta; // Number of meta blocks (boot, sb, nlog, inode, bitmap) 00027 int nblocks; // Number of data blocks
00028
00029 int fsfd;
00030 struct superblock sb;
00031 char zeroes[BSIZE];
00032 uint freeinode = 1;
00033 uint freeblock;
00034
00035
00036 void balloc(int);
00037 void wsect(uint, void*);
00038 void winode(uint, struct dinode*);
00039 void rinode(uint inum, struct dinode *ip);
00040 void rsect(uint sec, void *buf);
00041 uint ialloc(ushort type);
00042 void iappend(uint inum, void *p, int n);
00043
00044 // convert to intel byte order
00045 ushort
00046 xshort(ushort x)
00047 {
00048
        ushort y;
00049
        uchar *a = (uchar*) &y;
00050 a[0] = x;
00051 a[1] = x \gg 8;
        return y;
00052
00053 }
00054
00055 uint
00056 xint(uint x)
00057 {
00058
        uint y;
        uchar *a = (uchar*) &y;
a[0] = x;
00059
00061
        a[1] = x \gg 8;
00062
        a[2] = x \gg 16;
        a[3] = x \gg 24;
00063
        return y;
00064
00065 }
00066
00067 int
00068 main(int argc, char *argv[])
00069 {
00070
        int i, cc, fd;
00071
        uint rootino, inum, off;
struct dirent de;
00072
         char buf[BSIZE];
00074
        struct dinode din;
00075
00076
         static_assert(sizeof(int) == 4, "Integers must be 4 bytes!");
00077
00078
00079
08000
          fprintf(stderr, "Usage: mkfs fs.img files...\n");
00081
           exit(1);
00082
```

5.126 mkfs.c 291

```
00083
00084
         assert((BSIZE % sizeof(struct dinode)) == 0);
00085
         assert((BSIZE % sizeof(struct dirent)) == 0);
00086
00087
         fsfd = open(argv[1], O_RDWR|O_CREAT|O_TRUNC, 0666);
00088
         if (fsfd < 0) {
00089
          perror(argv[1]);
00090
           exit(1);
00091
00092
        // 1 fs block = 1 disk sector
00093
        nmeta = 2 + nlog + ninodeblocks + nbitmap;
nblocks = FSSIZE - nmeta;
00094
00095
00096
00097
         sb.size = xint(FSSIZE);
        sb.nblocks = xint(nblocks);
sb.ninodes = xint(NINODES);
00098
00099
00100
         sb.nlog = xint(nlog);
         sb.logstart = xint(2);
00101
00102
         sb.inodestart = xint(2+nlog);
00103
         sb.bmapstart = xint(2+nlog+ninodeblocks);
00104
00105
        printf("nmeta %d (boot, super, log blocks %u inode blocks %u, bitmap blocks %u) blocks %d total
        %d\n",
00106
                 nmeta, nlog, ninodeblocks, nbitmap, nblocks, FSSIZE);
00107
00108
         freeblock = nmeta;
                                  // the first free block that we can allocate
00109
         for(i = 0; i < FSSIZE; i++)</pre>
00110
00111
          wsect(i, zeroes);
00112
00113
        memset(buf, 0, sizeof(buf));
00114
        memmove(buf, &sb, sizeof(sb));
00115
         wsect(1, buf);
00116
        rootino = ialloc(T_DIR);
00117
00118
        assert (rootino == ROOTINO);
00119
00120
         bzero(&de, sizeof(de));
        de.inum = xshort(rootino);
strcpy(de.name, ".");
00121
00122
00123
        iappend(rootino, &de, sizeof(de));
00124
00125
        bzero(&de, sizeof(de));
00126
        de.inum = xshort(rootino);
00127
         strcpy(de.name, "..");
00128
        iappend(rootino, &de, sizeof(de));
00129
        for(i = 2; i < argc; i++) {</pre>
00130
          assert(index(argv[i], '/') == 0);
00131
00132
00133
           if((fd = open(argv[i], 0)) < 0){
00134
            perror(argv[i]);
00135
             exit(1);
00136
00137
00138
           // Skip leading _ in name when writing to file system.
           // The binaries are named _rm, _cat, etc. to keep the // build operating system from trying to execute them
00139
00140
           // in place of system binaries like rm and cat.
if(argv[i][0] == '_')
00141
00142
00143
             ++arqv[i];
00144
00145
           inum = ialloc(T_FILE);
00146
00147
           bzero(&de, sizeof(de));
00148
           de.inum = xshort(inum);
           strncpy(de.name, argv[i], DIRSIZ);
iappend(rootino, &de, sizeof(de));
00149
00150
00151
00152
           while((cc = read(fd, buf, sizeof(buf))) > 0)
00153
             iappend(inum, buf, cc);
00154
00155
           close(fd);
00156
00157
00158
         // fix size of root inode dir
00159
         rinode(rootino, &din);
        off = xint(din.size);
off = ((off/BSIZE) + 1) * BSIZE;
00160
00161
        din.size = xint(off);
00162
00163
         winode (rootino, &din);
00164
00165
        balloc(freeblock);
00166
00167
        exit(0);
00168 }
```

```
00169
00170 void
00171 wsect(uint sec, void *buf)
00172 {
        if(lseek(fsfd, sec * BSIZE, 0) != sec * BSIZE){
  perror("lseek");
  evit(1):
00173
00174
00175
         exit(1);
00176
00177
       if(write(fsfd, buf, BSIZE) != BSIZE) {
        perror("write");
exit(1);
00178
00179
00180 }
00181 }
00182
00183 void
00184 winode(uint inum, struct dinode *ip)
00185 {
00186
       char buf[BSIZE];
       uint bn;
00187
00188
       struct dinode *dip;
00189
00190
       bn = IBLOCK(inum, sb);
00191
       rsect(bn, buf);
       dip = ((struct dinode*)buf) + (inum % IPB);
*dip = *ip;
00192
00193
00194
       wsect(bn, buf);
00195 }
00196
00197 void
00198 rinode(uint inum, struct dinode *ip)
00199 {
00200
       char buf[BSIZE];
00201
       uint bn;
00202
       struct dinode *dip;
00203
       bn = IBLOCK(inum, sb);
00204
       rsect(bn, buf);
dip = ((struct dinode*)buf) + (inum % IPB);
00205
00206
00207
       *ip = *dip;
00208 }
00209
00210 void
00211 rsect(uint sec, void *buf)
00212 {
00213
        if(lseek(fsfd, sec * BSIZE, 0) != sec * BSIZE){
         perror("lseek");
00214
00215
          exit(1);
00216
       if(read(fsfd, buf, BSIZE) != BSIZE) {
00217
        perror("read");
00218
00219
          exit(1);
00220
00221 }
00222
00223 uint
00224 ialloc(ushort type)
00225 {
00226
       uint inum = freeinode++;
00227
       struct dinode din;
00228
00229
       bzero(&din, sizeof(din));
00230
       din.type = xshort(type);
00231
       din.nlink = xshort(1);
00232
       din.size = xint(0);
00233
       winode(inum, &din);
00234
       return inum;
00235 }
00236
00237 void
00238 balloc(int used)
00239 {
00240
       uchar buf[BSIZE];
00241
       int i;
00242
00243
       printf("balloc: first %d blocks have been allocated\n", used);
00244
        assert (used < BSIZE*8);
00245
        bzero(buf, BSIZE);
00246
       for(i = 0; i < used; i++) {</pre>
         buf[i/8] = buf[i/8] | (0x1 « (i%8));
00247
00248
00249
       printf("balloc: write bitmap block at sector %d\n", sb.bmapstart);
00250
       wsect(sb.bmapstart, buf);
00251 }
00252
00253 \#define min(a, b) ((a) < (b) ? (a) : (b))
00254
00255 void
```

```
00256 iappend(uint inum, void *xp, int n)
00257 {
00258
        char *p = (char*)xp;
00259
        uint fbn, off, n1;
        struct dinode din;
00260
        char buf[BSIZE];
00261
00262
        uint indirect[NINDIRECT];
00263
00264
00265
        rinode(inum, &din);
00266
        off = xint(din.size);
       // printf("append inum %d at off %d sz %d\n", inum, off, n); while (n > 0) {
00267
00268
00269
        fbn = off / BSIZE;
00270
          assert(fbn < MAXFILE);
00271
          if(fbn < NDIRECT) {</pre>
            if(xint(din.addrs[fbn]) == 0){
00272
00273
              din.addrs[fbn] = xint(freeblock++);
00275
            x = xint(din.addrs[fbn]);
00276
00277
            if(xint(din.addrs[NDIRECT]) == 0){
00278
              din.addrs[NDIRECT] = xint(freeblock++);
00279
00280
            rsect(xint(din.addrs[NDIRECT]), (char*)indirect);
            if(indirect[fbn - NDIRECT] == 0){
  indirect[fbn - NDIRECT] = xint(freeblock++);
00282
00283
              wsect(xint(din.addrs[NDIRECT]), (char*)indirect);
00284
00285
            x = xint(indirect[fbn-NDIRECT]);
00286
00287
          n1 = min(n, (fbn + 1) * BSIZE - off);
          rsect(x, buf);
bcopy(p, buf + off - (fbn * BSIZE), n1);
00288
00289
00290
          wsect(x, buf);
00291
          n -= n1;
00292
          off += n1;
          p += n1;
00294
00295
       din.size = xint(off);
00296
       winode(inum, &din);
00297 }
```

5.127 mmu.h File Reference

Classes

- struct gatedesc
- struct segdesc
- · struct taskstate

Macros

- #define CR0_PE 0x00000001
- #define CR0_PG 0x80000000
- #define CR0_WP 0x00010000
- #define CR4 PSE 0x00000010
- #define DPL USER 0x3
- #define FL IF 0x00000200
- #define NPDENTRIES 1024
- #define NPTENTRIES 1024
- #define NSEGS 6
- #define PDX(va) (((uint)(va) >> PDXSHIFT) & 0x3FF)
- #define PDXSHIFT 22
- #define PGADDR(d, t, o) ((uint)((d) << PDXSHIFT | (t) << PTXSHIFT | (o)))
- #define PGROUNDDOWN(a) (((a)) & ∼(PGSIZE-1))
- #define PGROUNDUP(sz) (((sz)+PGSIZE-1) & ~(PGSIZE-1))

- #define PGSIZE 4096
- #define PTE_ADDR(pte) ((uint)(pte) & ~0xFFF)
- #define PTE_FLAGS(pte) ((uint)(pte) & 0xFFF)
- #define PTE P 0x001
- #define PTE_PS 0x080
- #define PTE_U 0x004
- #define PTE_W 0x002
- #define PTX(va) (((uint)(va) >> PTXSHIFT) & 0x3FF)
- #define PTXSHIFT 12
- #define SEG(type, base, lim, dpl)
- #define SEG16(type, base, lim, dpl)
- #define SEG_KCODE 1
- #define SEG_KDATA 2
- #define SEG_TSS 5
- #define SEG_UCODE 3
- #define SEG UDATA 4
- #define SETGATE(gate, istrap, sel, off, d)
- #define STA_R 0x2
- #define STA_W 0x2
- #define STA_X 0x8
- #define STS IG32 0xE
- #define STS_T32A 0x9
- #define STS_TG32 0xF

Typedefs

typedef uint pte_t

5.127.1 Macro Definition Documentation

5.127.1.1 CR0_PE

#define CR0_PE 0x0000001

Definition at line 8 of file mmu.h.

5.127.1.2 CR0 PG

#define CR0_PG 0x80000000

Definition at line 10 of file mmu.h.

5.127.1.3 CR0_WP

#define CR0_WP 0x00010000

Definition at line 9 of file mmu.h.

5.127.1.4 CR4_PSE

#define CR4_PSE 0x00000010

Definition at line 12 of file mmu.h.

5.127.1.5 DPL_USER

#define DPL_USER 0x3

Definition at line 53 of file mmu.h.

5.127.1.6 FL_IF

#define FL_IF 0x00000200

Definition at line 5 of file mmu.h.

5.127.1.7 NPDENTRIES

#define NPDENTRIES 1024

Definition at line 83 of file mmu.h.

5.127.1.8 NPTENTRIES

#define NPTENTRIES 1024

Definition at line 84 of file mmu.h.

5.127.1.9 NSEGS

```
#define NSEGS 6
```

Definition at line 22 of file mmu.h.

5.127.1.10 PDX

```
#define PDX(  va \ ) \ (((uint)(va) >> PDXSHIFT) \& 0x3FF)
```

Definition at line 74 of file mmu.h.

5.127.1.11 PDXSHIFT

```
#define PDXSHIFT 22
```

Definition at line 88 of file mmu.h.

5.127.1.12 PGADDR

Definition at line 80 of file mmu.h.

5.127.1.13 PGROUNDDOWN

```
#define PGROUNDDOWN( a \ ) \ (((a)) \ \& \ \sim (PGSIZE-1))
```

Definition at line 91 of file mmu.h.

5.127.1.14 PGROUNDUP

```
#define PGROUNDUP( sz \ ) \ (((sz) + pgsize - 1) \ \& \ \sim (pgsize - 1))
```

Definition at line 90 of file mmu.h.

5.127.1.15 PGSIZE

```
#define PGSIZE 4096
```

Definition at line 85 of file mmu.h.

5.127.1.16 PTE_ADDR

```
#define PTE_ADDR( pte \ ) \ \ ((uint) \ (pte) \ \& \ {\sim} 0xFFF)
```

Definition at line 100 of file mmu.h.

5.127.1.17 PTE_FLAGS

Definition at line 101 of file mmu.h.

5.127.1.18 PTE_P

```
#define PTE_P 0x001
```

Definition at line 94 of file mmu.h.

5.127.1.19 PTE_PS

```
#define PTE_PS 0x080
```

Definition at line 97 of file mmu.h.

5.127.1.20 PTE_U

```
#define PTE_U 0x004
```

Definition at line 96 of file mmu.h.

5.127.1.21 PTE_W

```
#define PTE_W 0x002
```

Definition at line 95 of file mmu.h.

5.127.1.22 PTX

```
#define PTX( va \ ) \ (\mbox{((uint)(va)} >> \mbox{PTXSHIFT)} \ \& \ \mbox{0x3FF)}
```

Definition at line 77 of file mmu.h.

5.127.1.23 PTXSHIFT

```
#define PTXSHIFT 12
```

Definition at line 87 of file mmu.h.

5.127.1.24 SEG

Value:

Definition at line 43 of file mmu.h.

5.127.1.25 SEG16

Value:

Definition at line 47 of file mmu.h.

5.127.1.26 SEG_KCODE

#define SEG_KCODE 1

Definition at line 15 of file mmu.h.

5.127.1.27 SEG_KDATA

#define SEG_KDATA 2

Definition at line 16 of file mmu.h.

5.127.1.28 SEG_TSS

#define SEG_TSS 5

Definition at line 19 of file mmu.h.

5.127.1.29 SEG_UCODE

#define SEG_UCODE 3

Definition at line 17 of file mmu.h.

5.127.1.30 SEG_UDATA

#define SEG_UDATA 4

Definition at line 18 of file mmu.h.

5.127.1.31 SETGATE

Value:

```
{
  (gate).off_15_0 = (uint)(off) & 0xffff;
  (gate).cs = (sel);
  (gate).args = 0;
  (gate).rsv1 = 0;
  (gate).type = (istrap) ? STS_TG32 : STS_IG32;
  (gate).s = 0;
  (gate).dpl = (d);
  (gate).p = 1;
  (gate).off_31_16 = (uint)(off) » 16;
  \}
```

Definition at line 168 of file mmu.h.

5.127.1.32 STA_R

#define STA_R 0x2

Definition at line 58 of file mmu.h.

5.127.1.33 STA_W

#define STA_W 0x2

Definition at line 57 of file mmu.h.

5.127.1.34 STA_X

#define STA_X 0x8

Definition at line 56 of file mmu.h.

5.127.1.35 STS_IG32

#define STS_IG32 0xE

Definition at line 62 of file mmu.h.

5.128 mmu.h 301

5.127.1.36 STS_T32A

```
#define STS_T32A 0x9
```

Definition at line 61 of file mmu.h.

5.127.1.37 STS_TG32

```
#define STS_TG32 0xF
```

Definition at line 63 of file mmu.h.

5.127.2 Typedef Documentation

5.127.2.1 pte_t

```
typedef uint pte_t
```

Definition at line 104 of file mmu.h.

5.128 mmu.h

Go to the documentation of this file.

```
00001 // This file contains definitions for the
00002 // x86 memory management unit (MMU).
00004 // Eflags register
                                  0x00000200
00005 #define FL_IF
                                                    // Interrupt Enable
00006
00007 // Control Register flags
// Protection Enable
00009 #define CRO_WP
                                                    // Write Protect
00010 #define CR0_PG
                                  0x80000000
                                                    // Paging
00011
                                                    // Page size extension
00012 #define CR4_PSE
                                 0x00000010
00013
00014 // various segment selectors.
00014 // Various Segment Selectors.

00015 #define SEG_KCODE 1 // kernel code

00016 #define SEG_KDATA 2 // kernel data+stack

00017 #define SEG_UCODE 3 // user code

00018 #define SEG_UDATA 4 // user data+stack

00019 #define SEG_TSS 5 // this process's task state
00020
00021 // cpu->gdt[NSEGS] holds the above segments.
00022 #define NSEGS
00023
00024 #ifndef __ASSEMBLER_
00025 // Segment Descriptor
00034
        uint lim_19_16 : 4; // High bits of segment limit
```

```
uint avl : 1;
uint rsvl : 1;
uint db : 1;
vint a : 1;
// Reserved
// Reserved
// 0 = 16-bit segment, 1 = 32-bit segment
// Granularity: limit scaled by 4K when set
00036
00037
00038
00039
00040 };
00042 // Normal segment
00043 #define SEG(type, base, lim, dpl) (struct segdesc)
00044 { ((lim) » 12) & 0xffff, (uint) (base) & 0xfff, 00045 ((uint) (base) » 16) & 0xff, type, 1, dpl, 1, 00046 (uint) (lim) » 28, 0, 0, 1, 1, (uint) (base) » 24 } 00047 #define SEG16(type, base, lim, dpl) (struct segdesc)
00048 { (lim) & Oxffff, (uint) (base) & Oxffff, 
00049 ((uint) (base) » 16) & Oxff, type, 1, dpl, 1,
00050
         (uint)(lim) » 16, 0, 0, 1, 0, (uint)(base) » 24 }
00051 #endif
00052
00053 #define DPL_USER 0x3
                                         // User DPL
00054
00055 // Application segment type bits
00056 #define STA_X 0x8 // Executable segment
00057 #define STA_W 0x2 // Writeable (non-executable segments)
00058 #define STA_R
                                          // Readable (executable segments)
                                0x2
00059
00060 // System segment type bits
00061 #define STS_T32A 0x9
00062 #define STS_IG32 0xE
                                        // Available 32-bit TSS
// 32-bit Interrupt Gate
// 32-bit Trap Gate
00062 #define STS_IG32
00063 #define STS_TG32
                                0xF
00064
00065 // A virtual address 'la' has a three-part structure as follows:
00066 //
00067 // +------10------+
00068 // | Page Directory | Page Table | Offset within Page | 00069 // | Index | Index |
00069 // | Index | Index | 00070 // +------
00071 // \--- PDX(va) --/ \--- PTX(va) --/
00073 // page directory index
00074 #define PDX(va)
                                      (((uint)(va) » PDXSHIFT) & 0x3FF)
00075
00076 // page table index
00077 #define PTX(va)
                                     (((uint)(va) » PTXSHIFT) & 0x3FF)
00079 // construct virtual address from indexes and offset
00080 \#define PGADDR(d, t, o) ((uint)((d) \ll PDXSHIFT | (t) \ll PTXSHIFT | (o)))
00081
00082 // Page directory and page table constants.
00083 #define NPDENTRIES 1024 // # directory entries per page directory
00084 #define NPTENTRIES 1024 // # PTEs per page table
00085 #define PGSIZE 4096 // bytes mapped by a page
00086
00087 #define PTXSHIFT 12 // offset of PTX in a linear address 00088 #define PDXSHIFT 22 // offset of PDX in a linear address
00089
00090 #define PGROUNDUP(sz) (((sz)+PGSIZE-1) & ~(PGSIZE-1))
00091 #define PGROUNDDOWN(a) (((a)) & ~(PGSIZE-1))
00092
00093 // Page table/directory entry flags.
                          0x001 // Present
0x002 // Writeable
00094 #define PTE_P
00095 #define PTE W
                                     0x004 // User
0x080 // Page Size
00096 #define PTE U
00097 #define PTE_PS
00098
00099 // Address in page table or page directory entry
00100 #define PTE_ADDR(pte) ((uint)(pte) & ~0xFFF)
00101 #define PTE_FLAGS(pte) ((uint)(pte) & 0xFFF)
00102
00103 #ifndef __ASSEMBLER_
00104 typedef uint pte_t;
00105
00106 // Task state segment format
00107 struct taskstate {
                                  // Old ts selector
00108 uint link;
00109
                                  // Stack pointers and segment selectors
          uint esp0;
00110
                                  // after an increase in privilege level
         ushort ss0;
00111
          ushort padding1;
00112
          uint *esp1;
00113
          ushort ss1:
         ushort padding2;
00114
00115
          uint *esp2;
00116
          ushort ss2;
00117
          ushort padding3;
00118
          void *cr3;
                                   // Page directory base
00119
         uint *eip;
                                  // Saved state from last task switch
00120
         uint eflags;
00121
                                  // More saved state (registers)
         uint eax:
```

```
uint ecx;
00123
00124
         uint ebx;
00125
         uint *esp;
00126
         uint *ebp;
00127
         uint esi:
00128
         uint edi;
00129
         ushort es;
                               // Even more saved state (segment selectors)
00130
         ushort padding4;
00131
         ushort cs;
00132
         ushort padding5;
00133
         ushort ss:
         ushort padding6;
00134
00135
         ushort ds;
00136
         ushort padding7;
00137
         ushort fs;
00138
         ushort padding8;
00139
         ushort gs;
00140
        ushort padding9;
00141
         ushort ldt;
00142
         ushort padding10;
00143
         ushort t;
                               // Trap on task switch
00144
        ushort iomb;
                              // I/O map base address
00145 };
00146
00147 // Gate descriptors for interrupts and traps
00148 struct gatedesc {
00149 uint off_15_0 : 16; // low 16 bits of offset in segment
                            // code segment selector
// # args, 0 for interrupt/trap gates
// reserved(should be zero I guess)
00150
         uint cs : 16;
00151
         uint args : 5;
                           // # drys, o for interrupt/riap gates
// reserved(should be zero I guess)
// type(STS_{IG32,TG32})
// must be 0 (system)
// descriptor(meaning new) privilege level
00152
         uint rsv1 : 3;
00153
         uint type : 4;
00154
         uint s : 1;
00155
         uint dpl : 2;
        uint p: 1; // Present
uint off_31_16: 16; // high bits of offset in segment
00156
00157
00158 };
00160 // Set up a normal interrupt/trap gate descriptor.
00161 // - istrap: 1 for a trap (= exception) gate, 0 for an interrupt gate.
00162 // interrupt gate clears FL_IF, trap gate leaves FL_IF alone
00163 // - sel: Code segment selector for interrupt/trap handler
00164 // - off: Offset in code segment for interrupt/trap handler
00165 // - dpl: Descriptor Privilege Level
00166 // the privilege level required for software to invoke
00167 //
                  this interrupt/trap gate explicitly using an int instruction.
00168 #define SETGATE(gate, istrap, sel, off, d)
00169 {
         (gate).off_15_0 = (uint)(off) & 0xffff;
00170
         (gate).cs = (sel);
00171
         (gate).args = 0;
00172
00173
         (gate).rsv1 = 0;
00174
         (gate).type = (istrap) ? STS_TG32 : STS_IG32;
00175
         (gate).s = 0;
00176
         (gate).dpl = (d);
00177
         (qate).p = 1;
         (gate).off_31_16 = (uint)(off) » 16;
00179 }
00180
00181 #endif
```

5.129 mp.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mp.h"
#include "x86.h"
#include "mmu.h"
#include "proc.h"
```

Functions

static struct mpconf * mpconfig (struct mp **pmp)

- void mpinit (void)
- static struct mp * mpsearch (void)
- static struct mp * mpsearch1 (uint a, int len)
- static uchar sum (uchar *addr, int len)

Variables

- struct cpu cpus [NCPU]
- · uchar ioapicid
- int ncpu

5.129.1 Function Documentation

5.129.1.1 mpconfig()

```
static struct mpconf * mpconfig (
               struct mp ** pmp ) [static]
Definition at line 73 of file mp.c.
00074 {
00075
        struct mpconf *conf;
00076
        struct mp *mp;
00077
00078
        if((mp = mpsearch()) == 0 || mp->physaddr == 0)
00079
          return 0;
        conf = (struct mpconf*) P2V((uint) mp->physaddr);
if(memcmp(conf, "PCMP", 4) != 0)
08000
00081
00082
          return 0;
00083
        if(conf->version != 1 && conf->version != 4)
00084
           return 0:
        if (sum((uchar*)conf, conf->length) != 0)
00085
00086
         return 0;
00087
        *pmp = mp;
return conf;
00089 }
```

Referenced by mpinit().

5.129.1.2 mpinit()

```
void mpinit (
     void )
```

Definition at line 92 of file mp.c.

```
00093 {
00094
        uchar *p, *e;
00095
        int ismp;
       struct mp *mp;
struct mpconf *conf;
00096
00097
00098
        struct mpproc *proc;
00099
        struct mpioapic *ioapic;
00100
00101
        if((conf = mpconfig(&mp)) == 0)
00102
         panic("Expect to run on an SMP");
        ismp = 1;
00103
00104 lapic = (uint*)conf->lapicaddr;
00105
       for (p=(uchar*) (conf+1), e=(uchar*) conf+conf->length; p<e; ) {</pre>
```

```
00106
             switch(*p){
00107
            case MPPROC:
00108
               proc = (struct mpproc*)p;
               if(ncpu < NCPU) {
00109
00110
                 cpus[ncpu].apicid = proc->apicid; // apicid may differ from ncpu
00111
                 ncpu++;
00112
00113
             p += sizeof(struct mpproc);
00114
               continue;
00115
            case MPIOAPIC:
             ioapic = (struct mpioapic*)p;
ioapicid = ioapic->apicno;
00116
00117
             p += sizeof(struct mpioapic);
continue;
00118
00119
00120
            case MPBUS:
00121
            case MPIOINTR:
            case MPLINTR:
00122
             p += 8;
00123
00124
               continue;
00125
            default:
00126
               ismp = 0;
00127
               break;
00128
            }
00129
00130
          if(!ismp)
00131
            panic("Didn't find a suitable machine");
00132
00133
          if (mp->imcrp) {
            If(mp > inderp) {
    // Bochs doesn't support IMCR, so this doesn't run on Bochs.
    // But it would on real hardware.
    outb(0x22, 0x70);    // Select IMCR
    outb(0x23, inb(0x23) | 1);    // Mask external interrupts.
00134
00135
00136
00137
00138
00139 }
```

5.129.1.3 mpsearch()

```
static struct mp * mpsearch ( void ) [static]
```

Definition at line 49 of file mp.c.

```
00050 {
        uchar *bda;
00052
        uint p;
00053
        struct mp *mp;
00054
00055
        bda = (uchar *) P2V(0x400);
        if ((p = ((bda[0x0F] «8) | bda[0x0E]) « 4)){
00056
         if((mp = mpsearch1(p, 1024)))
00057
00058
            return mp;
00059
        } else {
00060
        p = ((bda[0x14] \times 8) | bda[0x13]) *1024;
00061
          if((mp = mpsearch1(p-1024, 1024)))
00062
            return mp;
00063
00064
        return mpsearch1(0xF0000, 0x10000);
00065 }
```

Referenced by mpconfig().

5.129.1.4 mpsearch1()

Definition at line 31 of file mp.c.

00032 {

```
00033    uchar *e, *p, *addr;
00034
00035    addr = P2V(a);
00036    e = addr+len;
00037    for(p = addr; p < e; p += sizeof(struct mp))
00038         if(memcmp(p, "_MP_", 4) == 0 && sum(p, sizeof(struct mp)) == 0)
00039         return (struct mp*)p;
00040    return 0;</pre>
```

Referenced by mpsearch().

5.129.1.5 sum()

Definition at line 19 of file mp.c.

```
00020 {
00021 int i, sum;
00022
00023 sum = 0;
00024 for(i=0; i<len; i++)
00025 sum += addr[i];
00026 return sum;
00027 }
```

Referenced by mpconfig(), mpsearch1(), and sum().

5.129.2 Variable Documentation

5.129.2.1 cpus

```
struct cpu cpus[NCPU]
```

Definition at line 14 of file mp.c.

Referenced by cpuid(), mpinit(), mycpu(), seginit(), and startothers().

5.129.2.2 ioapicid

```
uchar ioapicid
```

Definition at line 16 of file mp.c.

Referenced by ioapicinit(), and mpinit().

5.130 mp.c 307

5.129.2.3 ncpu

```
int ncpu
```

Definition at line 15 of file mp.c.

Referenced by ideinit(), mpinit(), mycpu(), and startothers().

5.130 mp.c

Go to the documentation of this file.

```
00001 // Multiprocessor support
00002 // Search memory for MP description structures.
00003 // http://developer.intel.com/design/pentium/datashts/24201606.pdf
00004
00005 #include "types.h"
00006 #include "defs.h"
00007 #include "param.h"
00008 #include "memlayout.h"
00000 #include "mp.h"
00010 #include "x86.h"
00011 #include "mmu.h"
00012 #include "proc.h"
00013
00014 struct cpu cpus[NCPU];
00015 int ncpu;
00016 uchar ioapicid;
00017
00018 static uchar
00019 sum(uchar *addr, int len)
00020 {
00021
         int i, sum;
00022
00023
00024
         for(i=0; i<len; i++)</pre>
00025
          sum += addr[i];
00026
        return sum:
00027 }
00029 // Look for an MP structure in the len bytes at addr.
00030 static struct mp*
00031 mpsearch1(uint a, int len)
00032 {
00033
         uchar *e, *p, *addr;
00034
00035
         addr = P2V(a);
00036
         e = addr+len;
         for(p = addr; p < e; p += sizeof(struct mp))
  if(memcmp(p, "_MP_", 4) == 0 && sum(p, sizeof(struct mp)) == 0)
    return (struct mp*)p;</pre>
00037
00038
00039
00040
         return 0;
00041 }
00042
00043 // Search for the MP Floating Pointer Structure, which according to the
00044 // spec is in one of the following three locations: 00045 // 1) in the first KB of the EBDA;
00046 // 2) in the last KB of system base memory;
00047 // 3) in the BIOS ROM between 0xE0000 and 0xFFFFF.
00048 static struct mp*
00049 mpsearch(void)
00050 {
00051
         uchar *bda;
00052
         uint p;
00053
         struct mp *mp;
00054
00055
         bda = (uchar *) P2V(0x400);
         if((p = ((bda[0x0F] < 8) | bda[0x0E]) < 4)){}
00056
00057
          if((mp = mpsearch1(p, 1024)))
00058
              return mp;
00059
         } else {
         p = ((bda[0x14] \times 8) | bda[0x13]) *1024;
00060
           if((mp = mpsearch1(p-1024, 1024)))
  return mp;
00061
00062
00063
00064
         return mpsearch1 (0xF0000, 0x10000);
00065 }
00066
```

```
00067 // Search for an MP configuration table. For now,
00068 // don't accept the default configurations (physaddr == 0).
00069 // Check for correct signature, calculate the checksum and,
00070 // if correct, check the version.
00071 // To do: check extended table checksum.
00072 static struct mpconf*
00073 mpconfig(struct mp **pmp)
00074 {
00075
        struct mpconf *conf;
00076
        struct mp *mp;
00077
00078
        if((mp = mpsearch()) == 0 || mp->physaddr == 0)
          return 0;
00079
        conf = (struct mpconf*) P2V((uint) mp->physaddr);
if(memcmp(conf, "PCMP", 4) != 0)
08000
00081
00082
           return 0;
        if(conf->version != 1 && conf->version != 4)
00083
00084
          return 0;
        if (sum((uchar*)conf, conf->length) != 0)
00085
00086
         return 0;
00087
        *pmp = mp;
00088
        return conf;
00089 }
00090
00091 void
00092 mpinit (void)
00093 {
00094
        uchar *p, *e;
00095
        int ismp;
00096
        struct mp *mp;
00097
        struct mpconf *conf;
00098
        struct mpproc *proc;
00099
        struct mpioapic *ioapic;
00100
        if((conf = mpconfig(&mp)) == 0)
  panic("Expect to run on an SMP");
00101
00102
        ismp = 1;
00103
        lapic = (uint*)conf->lapicaddr;
00105
        for(p=(uchar*)(conf+1), e=(uchar*)conf+conf->length; p<e; ){</pre>
00106
         switch(*p){
00107
          case MPPROC:
00108
             proc = (struct mpproc*)p;
             if (ncpu < NCPU) {
00109
00110
               cpus[ncpu].apicid = proc->apicid; // apicid may differ from ncpu
00111
00112
00113
            p += sizeof(struct mpproc);
          continue; case MPIOAPIC:
00114
00115
           ioapic = (struct mpioapic*)p;
00116
             ioapicid = ioapic->apicno;
00117
            p += sizeof(struct mpioapic);
00118
00119
             continue;
00120
          case MPBUS:
00121
          case MPIOINTR:
          case MPLINTR:
00122
           p += 8;
00124
             continue;
00125
           default:
00126
            ismp = 0;
00127
            break:
00128
          }
00129
00130
00131
          panic("Didn't find a suitable machine");
00132
00133
        if (mp->imcrp) {
         // Bochs doesn't support IMCR, so this doesn't run on Bochs.
// But it would on real hardware.
outb(0x22, 0x70); // Select IMCR
00134
00135
00137
          outb(0x23, inb(0x23) \mid 1); // Mask external interrupts.
00138
00139 }
```

5.131 mp.d File Reference

5.132 mp.d

Go to the documentation of this file.

00001 mp.o: mp.c /usr/include/stdc-predef.h types.h defs.h param.h memlayout.h \backslash 00002 mp.h x86.h mmu.h proc.h

5.133 mp.h File Reference

Classes

- struct mp
- struct mpconf
- struct mpioapic
- struct mpproc

Macros

- #define MPBOOT 0x02
- #define MPBUS 0x01
- #define MPIOAPIC 0x02
- #define MPIOINTR 0x03
- #define MPLINTR 0x04
- #define MPPROC 0x00

5.133.1 Macro Definition Documentation

5.133.1.1 MPBOOT

#define MPBOOT 0x02

Definition at line 34 of file mp.h.

5.133.1.2 MPBUS

#define MPBUS 0x01

Definition at line 50 of file mp.h.

5.133.1.3 MPIOAPIC

#define MPIOAPIC 0x02

Definition at line 51 of file mp.h.

5.133.1.4 MPIOINTR

```
#define MPIOINTR 0x03
```

Definition at line 52 of file mp.h.

5.133.1.5 MPLINTR

```
#define MPLINTR 0x04
```

Definition at line 53 of file mp.h.

5.133.1.6 MPPROC

```
#define MPPROC 0x00
```

Definition at line 49 of file mp.h.

5.134 mp.h

Go to the documentation of this file.

```
00001 // See MultiProcessor Specification Version 1.[14]
                                           // floating pointer
// "_MP_"
00003 struct mp {
00004 uchar signature[4];
           void *physaddr;
                                                        // phys addr of MP config table
00005
00006 uchar length;
00007 uchar specrev;
                                                       // [14]
80000
           uchar checksum;
                                                      // all bytes must add up to 0
// MP system config type
          uchar type;
uchar imcrp;
00009
00010
00011
           uchar reserved[3];
00012 };
00013
00014 struct mpconf {
                                           // configuration table header
// total table length
                                           // total table length
// [14]
// all bytes must add up to 0
// product id
// OEM table pointer
// OEM table length
// entry count
// address of local APIC
// extended table length
// extended table checksum
          uchar checksum;
          uchar product[20];
00019
00020 uint *oemtable;
00021
00022
          ushort oemlength;
          ushort entry;
uint *lapicaddr;
ushort xlength;
00023
00024
00025 uchar xchecksum;
00026 uchar reserved;
00027 };
00028
00029 struct mpproc { // processor table entry 00030 uchar type; // entry type (0) 00031 uchar apicid; // local APIC id 00032 uchar version; // local APIC ver 00033 uchar flags: // CPU flags
                                                        // local APIC verison
          uchar version; // local AFIC version
uchar flags; // CPU flags
#define MPBOOT 0x02 // This proc is the bootstrap processor.
uchar signature[4]; // CPU signature
uint feature; // feature flags from CPUID instruction
00033 uchar flags;
00034
00035
00036
00037
          uint feature;
           uchar reserved[8];
00038 };
00040 struct mpioapic { // I/O APIC table entry
```

5.135 nice.c File Reference 311

```
00041
            uchar type;
                                                               // entry type (2)
00042
            uchar apicno;
                                                               // I/O APIC id
                                                               // I/O APIC version
00043
            uchar version;
00043 uchar flags;
00044 uchar flags;
00045 uint *addr;
                                                                // I/O APIC flags
                                                              // I/O APIC address
00046 };
00048 // Table entry types
00049 #define MPPROC 0x00 // One per processor
00050 #define MPBUS 0x01 // One per bus
00051 #define MPIOAPIC 0x02 // One per I/O APIC
00052 #define MPIOINTR 0x03 // One per bus interrupt source
00053 #define MPLINTR 0x04 // One per system interrupt source
00054
00055 //PAGEBREAK!
00056 // Blank page.
```

5.135 nice.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
```

Functions

• int main (int argc, char *argv[])

5.135.1 Function Documentation

5.135.1.1 main()

Definition at line 7 of file nice.c.

```
00008 {
00009    int priority, pid;
00010    if(argc < 3){
00011        printf(2, "Usage: nice pid priority\n");
00012        exit();
00013    }
00014    pid = atoi(argv[1]);
00015    priority = atoi(argv[2]);
00016    if (priority < 0 || priority > 20) {
        printf(2, "Invalid priority (0-20)!\n");
00018        exit();
00019    }
00020    chpr(pid, priority);
00021    exit();
00022 }
```

5.136 nice.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004 #include "fcntl.h"
00005
00006 int
00007 main(int argc, char *argv[])
00008 {
00009
            int priority, pid;
00010 if (argc < 3) {
           printf(2,"Usage: nice pid priority\n");
exit();
00012
00013
00013  }
00014  pid = atoi(argv[1]);
00015  priority = atoi(argv[2]);
00016  if (priority < 0 || priority > 20) {
00017     printf(2, "Invalid priority (0-20)!\n");
00018     orith();
00018
               exit();
00019 } 00020 chpr(pid, priority);
cnpr(pic
ou021 exit();
00022 }
```

5.137 nice.d File Reference

5.138 nice.d

```
Go to the documentation of this file.
```

```
00001 nice.o: nice.c /usr/include/stdc-predef.h types.h stat.h user.h fcntl.h
```

5.139 param.h File Reference

Macros

- #define FSSIZE 1000
- #define KSTACKSIZE 4096
- #define LOGSIZE (MAXOPBLOCKS*3)
- #define MAXARG 32
- #define MAXOPBLOCKS 10
- #define NBUF (MAXOPBLOCKS*3)
- #define NCPU 8
- #define NDEV 10
- #define NFILE 100
- #define NINODE 50
- #define NOFILE 16
- #define NPROC 64
- #define ROOTDEV 1

5.139.1 Macro Definition Documentation

5.139.1.1 FSSIZE

#define FSSIZE 1000

Definition at line 13 of file param.h.

5.139.1.2 KSTACKSIZE

#define KSTACKSIZE 4096

Definition at line 2 of file param.h.

5.139.1.3 LOGSIZE

#define LOGSIZE (MAXOPBLOCKS*3)

Definition at line 11 of file param.h.

5.139.1.4 MAXARG

#define MAXARG 32

Definition at line 9 of file param.h.

5.139.1.5 MAXOPBLOCKS

#define MAXOPBLOCKS 10

Definition at line 10 of file param.h.

5.139.1.6 NBUF

#define NBUF (MAXOPBLOCKS*3)

Definition at line 12 of file param.h.

5.139.1.7 NCPU

#define NCPU 8

Definition at line 3 of file param.h.

5.139.1.8 NDEV

#define NDEV 10

Definition at line 7 of file param.h.

5.139.1.9 NFILE

#define NFILE 100

Definition at line 5 of file param.h.

5.139.1.10 NINODE

#define NINODE 50

Definition at line 6 of file param.h.

5.139.1.11 NOFILE

#define NOFILE 16

Definition at line 4 of file param.h.

5.139.1.12 NPROC

#define NPROC 64

Definition at line 1 of file param.h.

315 5.140 param.h

5.139.1.13 ROOTDEV

```
#define ROOTDEV 1
```

Definition at line 8 of file param.h.

5.140 param.h

Go to the documentation of this file.

```
Go to the documentation of this file.

00001 #define NPROC 64 // maximum number of processes

00002 #define KSTACKSIZE 4096 // size of per-process kernel stack

00003 #define NCPU 8 // maximum number of CPUs

00004 #define NOFILE 16 // open files per process

00005 #define NFILE 100 // open files per system

00006 #define NINODE 50 // maximum number of active i-nodes

00007 #define NDEV 10 // maximum major device number

00008 #define ROOTDEV 1 // device number of file system root disk

00009 #define MAXARG 32 // max exec arguments

00010 #define MAXOPBLOCKS 10 // max # of blocks any FS op writes

00011 #define LOGSIZE (MAXOPBLOCKS+3) // max data blocks in on-disk log

00012 #define NBUF (MAXOPBLOCKS+3) // size of disk block cache

00013 #define FSSIZE 1000 // size of file system in blocks

00014
     00014
```

5.141 picirq.c File Reference

```
#include "types.h"
#include "x86.h"
#include "traps.h"
```

Macros

- #define IO_PIC1 0x20
- #define IO_PIC2 0xA0

Functions

void picinit (void)

5.141.1 Macro Definition Documentation

5.141.1.1 IO_PIC1

```
#define IO_PIC1 0x20
```

Definition at line 6 of file picirq.c.

5.141.1.2 IO_PIC2

```
#define IO_PIC2 0xA0
```

Definition at line 7 of file picirq.c.

5.141.2 Function Documentation

5.141.2.1 picinit()

```
void picinit (
     void )
```

Definition at line 11 of file picirq.c.

5.142 picirq.c

Go to the documentation of this file.

5.143 picirq.d File Reference

5.144 picirq.d

Go to the documentation of this file.

00001 picirq.o: picirq.c /usr/include/stdc-predef.h types.h x86.h traps.h

5.145 pipe.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "mmu.h"
#include "proc.h"
#include "fs.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "file.h"
```

Classes

• struct pipe

Macros

• #define PIPESIZE 512

Functions

- int pipealloc (struct file **f0, struct file **f1)
- void pipeclose (struct pipe *p, int writable)
- int piperead (struct pipe *p, char *addr, int n)
- int pipewrite (struct pipe *p, char *addr, int n)

5.145.1 Macro Definition Documentation

5.145.1.1 PIPESIZE

```
#define PIPESIZE 512
```

Definition at line 11 of file pipe.c.

5.145.2 Function Documentation

5.145.2.1 pipealloc()

```
int pipealloc (
                  struct file ** f0,
                  struct file ** f1 )
Definition at line 23 of file pipe.c.
00024
00025
          struct pipe *p;
00026
00027
          p = 0;
         *f0 = *f1 = 0;
00028
00029
          if((\star f0 = filealloc()) == 0 \mid \mid (\star f1 = filealloc()) == 0)
00030
            goto bad;
          if((p = (struct pipe*)kalloc()) == 0)
00031
          goto bad;
00032
00033
         p->readopen = 1;
00034
         p->writeopen = 1;
         p > wiredepen = 1,
p -> nwrite = 0;
p -> nread = 0;
initlock(&p->lock, "pipe");
(*f0) -> type = FD_PIPE;
(*f0) -> readable = 1;
00035
00036
00037
00038
00039
00040
         (*f0) ->writable = 0;
         (*f0) ->pipe = p;
(*f1) ->type = FD_PIPE;
(*f1) ->readable = 0;
00041
00042
00043
00044
         (*f1)->writable = 1;
00045
         (*f1)->pipe = p;
00046
         return 0;
00047
00048 //PAGEBREAK: 20
00049 bad:
00050
         if(p)
00051
           kfree((char*)p);
00052
          if(*f0)
00053
            fileclose(*f0);
         if(*f1)
00054
00055
           fileclose(*f1);
00056
         return -1;
00057 }
```

Referenced by sys_pipe().

5.145.2.2 pipeclose()

Definition at line 60 of file pipe.c.

```
00061 {
00062
        acquire(&p->lock);
00063
        if (writable) {
00064
         p->writeopen = 0;
00065
           wakeup(&p->nread);
00066
        } else {
         p->readopen = 0;
wakeup(&p->nwrite);
00067
00068
00069
        if(p->readopen == 0 && p->writeopen == 0){
  release(&p->lock);
00070
00071
00072
           kfree((char*)p);
00073
         } else
00074
           release(&p->lock);
00075 }
```

Referenced by fileclose().

5.145.2.3 piperead()

```
int piperead (
               struct pipe * p,
               char * addr,
               int n)
Definition at line 101 of file pipe.c.
00102 {
00103
00104
00105
        acquire(&p->lock);
        while(p->nread == p->nwrite && p->writeopen){ //DOC: pipe-empty
00106
         if (myproc() ->killed) {
00108
           release(&p->lock);
00109
            return -1;
00110
         sleep(&p->nread, &p->lock); //DOC: piperead-sleep
00111
00112
00113
        for(i = 0; i < n; i++) { //DOC: piperead-copy</pre>
        if (p->nread == p->nwrite)
00114
00115
           break;
         addr[i] = p->data[p->nread++ % PIPESIZE];
00116
00117
00118
       wakeup(&p->nwrite); //DOC: piperead-wakeup
00119
       release(&p->lock);
00120
       return i;
00121 }
```

Referenced by fileread().

5.145.2.4 pipewrite()

```
int pipewrite (
          struct pipe * p,
          char * addr,
          int n )
```

Definition at line 79 of file pipe.c.

```
00080 {
00081
         int i:
00082
00083
         acquire(&p->lock);
00084
         for(i = 0; i < n; i++) {
          while(p->nwrite == p->nread + PIPESIZE){ //DOC: pipewrite-full
  if(p->readopen == 0 || myproc()->killed){
00085
00086
00087
               release(&p->lock);
00088
               return -1;
00089
00090
             wakeup(&p->nread);
00091
             sleep(&p->nwrite, &p->lock); //DOC: pipewrite-sleep
00092
00093
          p->data[p->nwrite++ % PIPESIZE] = addr[i];
00094
00095
         wakeup(&p->nread); //DOC: pipewrite-wakeup1
00096
         release(&p->lock);
00097
         return n;
00098 }
```

Referenced by filewrite().

5.146 pipe.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "defs.h"
00003 #include "param.h"
00004 #include "mmu.h"
00005 #include "proc.h"
00006 #include "fs.h"
00006 #include "IS.N
00007 #include "spinlock.h"
00008 #include "sleeplock.h"
00009 #include "file.h"
00011 #define PIPESIZE 512
00012
00013 struct pipe {
00014 struct spinlock lock;
         char data[PIPESIZE];
00015
00016 uint nread; // number of bytes read
00017 uint nwrite; // number of bytes written
00018 int readopen; // read fd is still open
00019 int writeopen; // write fd is still open
00020 };
00021
00022 int
00023 pipealloc(struct file **f0, struct file **f1)
00024 {
00025
         struct pipe *p;
00026
         p = 0;
*f0 = *f1 = 0;
00027
00028
         if((\star f0 = filealloc()) == 0 \mid | (\star f1 = filealloc()) == 0)
00030
            goto bad;
00031
         if((p = (struct pipe*)kalloc()) == 0)
00032
           goto bad;
00033
         p->readopen = 1;
00034
         p->writeopen = 1;
         p->nwrite = 0;
p->nread = 0;
00035
00036
00037
          initlock(&p->lock, "pipe");
00038
         (*f0) \rightarrow type = FD_PIPE;
         (*f0) \rightarrow readable = 1;
00039
         (*f0)->writable = 0;
00040
         (*f0) \rightarrow pipe = p;
00041
00042
         (*f1)->type = FD_PIPE;
         (*f1) -> readable = 0;
(*f1) -> writable = 1;
00043
00044
00045
         (*f1)->pipe = p;
00046
         return 0;
00047
00048 //PAGEBREAK: 20
00049 bad:
00050
00051
           kfree((char*)p);
          <u>if</u>(*f0)
00052
00053
           fileclose(*f0);
00054
          if(*f1)
00055
           fileclose(*f1);
00056
         return -1;
00057 }
00058
00059 void
00060 pipeclose(struct pipe *p, int writable)
00061 {
00062
         acquire(&p->lock);
00063
          if (writable) {
          p->writeopen = 0;
00064
00065
            wakeup(&p->nread);
00066
         } else {
         p->readopen = 0;
00067
00068
            wakeup(&p->nwrite);
00069
         if (p->readopen == 0 && p->writeopen == 0) {
00070
          release(&p->lock);
00071
00072
            kfree((char*)p);
         } else
00074
           release(&p->lock);
00075 }
00076
00077 //PAGEBREAK: 40
00078 int
00079 pipewrite(struct pipe *p, char *addr, int n)
00080 {
00081
         int i;
00082
```

```
acquire(&p->lock);
00084
       for(i = 0; i < n; i++) {
         while(p->nwrite == p->nread + PIPESIZE){ //DOC: pipewrite-full
  if(p->readopen == 0 || myproc()->killed){
00085
00086
00087
             release(&p->lock);
00088
             return -1;
00089
00090
            wakeup(&p->nread);
00091
           sleep(&p->nwrite, &p->lock); //DOC: pipewrite-sleep
00092
00093
         p->data[p->nwrite++ % PIPESIZE] = addr[i];
00094
00095
       wakeup(&p->nread); //DOC: pipewrite-wakeup1
00096
       release(&p->lock);
00097
00098 }
00099
00100 int
00101 piperead(struct pipe *p, char *addr, int n)
00102 {
00103
00104
00105
       acquire(&p->lock);
       while(p->nread == p->nwrite && p->writeopen){    //DOC: pipe-empty
00106
00107
         if (myproc() ->killed) {
           release(&p->lock);
00109
00110
00111
         sleep(&p->nread, &p->lock); //DOC: piperead-sleep
00112
00113
       if(p->nread == p->nwrite)
00114
00115
00116
         addr[i] = p->data[p->nread++ % PIPESIZE];
00117
       wakeup(&p->nwrite); //DOC: piperead-wakeup
00118
       release(&p->lock);
00119
00120
       return i;
00121 }
```

5.147 pipe.d File Reference

5.148 pipe.d

```
Go to the documentation of this file.
```

```
00001 pipe.o: pipe.c /usr/include/stdc-predef.h types.h defs.h param.h mmu.h \ 00002 proc.h fs.h spinlock.h sleeplock.h file.h
```

5.149 printf.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

- void printf (int fd, const char *fmt,...)
- static void printint (int fd, int xx, int base, int sgn)
- static void putc (int fd, char c)

5.149.1 Function Documentation

5.149.1.1 printf()

```
void printf (
                  int fd,
                  const char * fmt,
                    ...)
Definition at line 40 of file printf.c.
00041 {
00042
         char *s;
int c, i, state;
uint *ap;
00043
00044
00045
00046
00047
          ap = (uint*)(void*)&fmt + 1;
         for(i = 0; fmt[i]; i++) {
    c = fmt[i] & 0xff;
    if(state == 0) {
        if(c == '%') {
00048
00049
00050
00051
00052
                state = '%';
00053
               } else {
                putc(fd, c);
00054
00055
00056
            } else if(state == '%'){
              if(c == 'd'){
00058
                printint(fd, *ap, 10, 1);
              ap++;
} else if(c == 'x' || c == 'p'){
printint(fd, *ap, 16, 0);
00059
00060
00061
00062
              ap++;
} else if(c == 's'){
00063
00064
                s = (char*)*ap;
00065
                 ap++;
                 if(s == 0)
s = "(null)";
00066
00067
                 while (*s != 0) {
putc(fd, *s);
00068
00069
00070
                   s++;
00071
               } else if(c == 'c'){
  putc(fd, *ap);
00072
00073
00074
              ap++;
} else if(c == '%'){
00075
00076
                putc(fd, c);
00077
                // Unknown % sequence. Print it to draw attention.
putc(fd, '%');
00078
00079
00080
                 putc(fd, c);
00081
00082
               state = 0;
00083
00084 }
00085 }
```

5.149.1.2 printint()

```
static void printint (
            int fd,
             int xx,
            int base,
            int sgn ) [static]
```

Definition at line 12 of file printf.c. 00013 {

5.150 printf.c 323

```
static char digits[] = "0123456789ABCDEF";
00015
        char buf[16];
00016
        int i, neg;
00017
        uint x;
00018
        neg = 0;
00019
        if(sgn && xx < 0){
        neg = 1;
x = -xx;
00021
00022
00023
        } else {
00024
       x = xx;
00025
00026
00027
        i = 0;
00028
        do{
        buf[i++] = digits[x % base];
}while((x /= base) != 0);
if(neg)
00029
00030
00031
          buf[i++] = '-';
00033
00034
        while (--i >= 0)
          putc(fd, buf[i]);
00035
00036 }
```

Referenced by printf().

5.149.1.3 putc()

```
static void putc (  \mbox{int } fd, \\ \mbox{char } c \mbox{ ) [static]}
```

Definition at line 6 of file printf.c.

```
00007 {
00008 write(fd, &c, 1);
00009 }
```

Referenced by printf(), and printint().

5.150 printf.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004
00005 static void
00006 putc(int fd, char c) 00007 {
80000
        write(fd, &c, 1);
00009 }
00010
00011 static void
00012 printint(int fd, int xx, int base, int sgn) 00013 {
00014
        static char digits[] = "0123456789ABCDEF";
        char buf[16];
00015
00016
        int i, neg;
00017
        uint x;
00018
00019
        neg = 0;
        if(sgn && xx < 0) {
  neg = 1;
  x = -xx;
00020
00021
00022
00023
        } else {
00024
          x = xx;
00025
00026
00027
        i = 0;
00028
        do {
```

```
buf[i++] = digits[x % base];
       }while((x /= base) != 0);
00030
00031
       if(neg)
        buf[i++] = '-';
00032
00033
00034 while (--i >= 0)
        putc(fd, buf[i]);
00036 }
00037
00038 // Print to the given fd. Only understands %d, %x, %p, %s.
00039 void
00040 printf(int fd, const char *fmt, ...)
00041 {
00042
00043
       int c, i, state;
00044
       uint *ap;
00045
00046
       state = 0;
       ap = (uint*)(void*)&fmt + 1;
00048
       for(i = 0; fmt[i]; i++){
        c = fmt[i] & 0xff;
00049
          if(state == 0) {
   if(c == '%') {
     state = '%';
}
00050
00051
00052
00053
            } else {
00054
             putc(fd, c);
00055
00056
         } else if(state == '%'){
00057
            if(c == 'd'){
             printint(fd, *ap, 10, 1);
00058
00059
            ap++;
} else if(c == 'x' || c == 'p'){
00060
00061
            printint(fd, *ap, 16, 0);
00062
            } else if(c == 's'){
s = (char*)*ap;
00063
00064
             ap++;
if(s == 0)
s = "(null)";
00065
00066
00067
00068
              while(*s != 0){
00069
               putc(fd, *s);
00070
                s++;
00071
00072
          } else if(c == 'c'){
00073
            putc(fd, *ap);
00074
              ap++;
00075
           } else if(c == '%'){
00076
              putc(fd, c);
00077
            } else {
             // Unknown % sequence. Print it to draw attention.
00078
              putc(fd, '%');
08000
              putc(fd, c);
00081
00082
            state = 0;
00083
00084
       }
00085 }
```

5.151 printf.d File Reference

5.152 printf.d

Go to the documentation of this file.

00001 printf.o: printf.c /usr/include/stdc-predef.h types.h stat.h user.h

5.153 proc.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
```

```
#include "mmu.h"
#include "x86.h"
#include "proc.h"
#include "spinlock.h"
```

Functions

- static struct proc * allocproc (void)
- int chpr (int pid, int priority)
- int cps ()
- int cpuid ()
- · void exit (void)
- int fork (void)
- void forkret (void)
- int growproc (int n)
- int kill (int pid)
- struct cpu * mycpu (void)
- struct proc * myproc (void)
- void pinit (void)
- void procdump (void)
- void sched (void)
- void scheduler (void)
- void sleep (void *chan, struct spinlock *lk)
- void trapret (void)
- void userinit (void)
- int wait (void)
- void wakeup (void *chan)
- static void wakeup1 (void *chan)
- void yield (void)

Variables

```
• static struct proc * initproc
```

```
• int nextpid = 1
```

```
    struct {
        struct spinlock lock
        struct proc proc [NPROC]
    } ptable
```

5.153.1 Function Documentation

5.153.1.1 allocproc()

```
static struct proc * allocproc (
                void ) [static]
Definition at line 74 of file proc.c.
00076
         struct proc *p;
00077
         char *sp;
00078
00079
        acquire (&ptable.lock);
08000
00081
         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)</pre>
          if (p->state == UNUSED)
00082
00083
             goto found;
00084
        release(&ptable.lock);
00085
00086
        return 0;
00087
00088 found:
00089
        p->state = EMBRYO;
00090
        p->pid = nextpid++;
00091
00092
        release (&ptable.lock);
00093
00094
        // Allocate kernel stack.
00095
         if((p->kstack = kalloc()) == 0){
         p->state = UNUSED;
00096
00097
           return 0;
00098
00099
         sp = p->kstack + KSTACKSIZE;
00100
00101
        // Leave room for trap frame.
00102
         sp -= sizeof *p->tf;
         p->tf = (struct trapframe*)sp;
00103
00104
        // Set up new context to start executing at forkret, // which returns to trapret.
00105
00106
00107
        sp -= 4;
00108
         *(uint*)sp = (uint)trapret;
00109
        sp -= sizeof *p->context;
p->context = (struct context*)sp;
memset(p->context, 0, sizeof *p->context);
p->context->eip = (uint)forkret;
00110
00111
00112
00113
00114
00115
        return p;
00116 }
```

Referenced by fork(), and userinit().

5.153.1.2 chpr()

Definition at line 559 of file proc.c.

```
00560 {
00561
        struct proc *p;
00562
        acquire(&ptable.lock);
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
   if(p->pid == pid) {
00563
00564
00565
            p->priority = priority;
00566
             break;
00567
          }
00568
00569
        release(&ptable.lock);
00570
        return pid;
00571 }
```

Referenced by main(), and sys_chpr().

5.153.1.3 cps()

```
00539 struct proc *p;
00540 //Enables interrupts on this processor.
00541 sti();
00542
00543 //Loop over process table looking for process with pid.
00544 acquire(&ptable.lock);
00545 cprintf("name \t pid \t state \t priority \n");
00546 for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
    if(p->state == SLEEPING)
        cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name,p->pid,p->priority);
00549        cprintf("%s \t %d \t RUNNING \t %d \n ", p->name,p->pid,p->priority);
00550        cprintf("%s \t %d \t RUNNING \t %d \n ", p->name,p->pid,p->priority);
00551        else if(p->state == RUNNABLE)
00552        cprintf("%s \t %d \t RUNNABLE \t %d \n ", p->name,p->pid,p->priority);
00553 }
00554 release(&ptable.lock);
00555 return 23;
00556 }
```

Referenced by main(), and sys_cps().

5.153.1.4 cpuid()

```
int cpuid (
     void )
```

Definition at line 31 of file proc.c.

```
00031 {
00032 return mycpu()-cpus;
00033 }
```

Referenced by seginit(), and trap().

5.153.1.5 exit()

```
void exit (
     void )
```

Definition at line 228 of file proc.c.

```
00229 {
00230
        struct proc *curproc = myproc();
00231
        struct proc *p;
00232
        int fd;
00233
00234
        if(curproc == initproc)
00235
        panic("init exiting");
00236
        // Close all open files.
for(fd = 0; fd < NOFILE; fd++){</pre>
00237
00238
         if(curproc->ofile[fd]){
00239
00240
            fileclose(curproc->ofile[fd]);
00241
             curproc->ofile[fd] = 0;
00242
00243
00244
00245
       begin_op();
        iput (curproc->cwd);
```

```
00247
         end_op();
         curproc->cwd = 0;
00248
00249
00250
         acquire(&ptable.lock);
00251
00252
         // Parent might be sleeping in wait().
         wakeup1 (curproc->parent);
00254
00255
         // Pass abandoned children to init.
00256
         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
           if(p->parent == curproc){
  p->parent = initproc;
  if(p->state == ZOMBIE)
00257
00258
00259
00260
                wakeup1(initproc);
00261
00262
00263
00264
         \ensuremath{//} Jump into the scheduler, never to return.
         curproc->state = ZOMBIE;
00265
         sched();
         panic("zombie exit");
00267
00268 }
```

Referenced by argptest(), bigargtest(), bigdir(), bigdir(), bigwrite(), bsstest(), cat(), concreate(), createdelete(), dirfile(), dirtest(), exectest(), exitiputtest(), exitwait(), forktest(), fourfiles(), fourteen(), iputtest(), iref(), linktest(), linkunlink(), main(), mem(), openiputtest(), opentest(), panic(), pipe1(), rmdot(), rsect(), runcmd(), sbrktest(), sharedfd(), subdir(), sys_exit(), trap(), uio(), unlinkread(), validatetest(), wc(), writetest(), writetest1(), and wsect().

5.153.1.6 fork()

```
int fork (
     void )
```

Definition at line 181 of file proc.c.

```
00182 {
00183
        int i, pid;
00184
        struct proc *np;
00185
        struct proc *curproc = myproc();
00186
00187
        // Allocate process.
00188
        if((np = allocproc()) == 0) {
00189
          return -1;
00190
00191
00192
        // Copy process state from proc.
00193
        if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
          kfree (np->kstack);
00194
          np->kstack = 0;
np->state = UNUSED;
00195
00196
00197
          return -1:
00198
00199
        np->sz = curproc->sz;
00200
        np->parent = curproc;
00201
        *np->tf = *curproc->tf;
00202
        // Clear %eax so that fork returns 0 in the child.
00203
00204
        np->tf->eax = 0;
00205
00206
        for(i = 0; i < NOFILE; i++)</pre>
00207
        if(curproc->ofile[i])
            np->ofile[i] = filedup(curproc->ofile[i]);
00208
00209
        np->cwd = idup(curproc->cwd);
00210
        safestrcpy(np->name, curproc->name, sizeof(curproc->name));
00212
00213
        pid = np->pid;
00214
00215
        acquire (&ptable.lock);
00216
00217
        np->state = RUNNABLE;
00218
00219
        release(&ptable.lock);
00220
00221
        return pid;
00222 }
```

Referenced by bigargtest(), concreate(), createdelete(), exitiputtest(), exitwait(), fork1(), fork1(), fourfiles(), linkunlink(), main(), mem(), openiputtest(), pipe1(), preempt(), sbrktest(), sharedfd(), sys_fork(), uio(), and validatetest().

5.153.1.7 forkret()

```
void forkret (
    void )
```

Definition at line 397 of file proc.c.

```
00399
        static int first = 1;
        \ensuremath{//} Still holding ptable.lock from scheduler.
00400
00401
        release(&ptable.lock);
00402
00403
        if (first) {
00404
         // Some initialization functions must be run in the context
00405
         // of a regular process (e.g., they call sleep), and thus cannot
          // be run from main().
00406
         first = 0;
00407
          iinit (ROOTDEV);
00408
          initlog(ROOTDEV);
00409
00411
00412
       // Return to "caller", actually trapret (see allocproc).
00413 }
```

Referenced by allocproc().

5.153.1.8 growproc()

```
int growproc ( \quad \text{ int } n \ )
```

Definition at line 159 of file proc.c.

```
00160 {
00161
00162
        struct proc *curproc = myproc();
00163
00164
       sz = curproc->sz;
00165
       if(n > 0){
        if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
00166
           return -1;
00167
00168
       } else if(n < 0){</pre>
        if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
00169
00170
           return -1;
00171
00172
       curproc->sz = sz;
00173
        switchuvm(curproc);
00174
        return 0;
00175 }
```

Referenced by sys_sbrk().

5.153.1.9 kill()

```
int kill (
                 int pid )
Definition at line 480 of file proc.c.
         struct proc *p;
00483
00484
         acquire (&ptable.lock);
         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){</pre>
00485
           if (p->pid == pid) {
   p->killed = 1;
00486
00488
              // Wake process from sleep if necessary.
             if(p->state == SLEEPING)
p->state = RUNNABLE;
00489
00490
00491
              release (&ptable.lock);
00492
             return 0;
00493
          }
00494
00495
         release(&ptable.lock);
         return -1;
00496
```

Referenced by main(), mem(), preempt(), sbrktest(), sys_kill(), and validatetest().

5.153.1.10 mycpu()

00497 }

Definition at line 38 of file proc.c.

```
00040
        int apicid, i;
00041
00042
        if (readeflags() &FL_IF)
00043
          panic("mycpu called with interrupts enabled\n");
00044
00045
        apicid = lapicid();
00046
        // APIC IDs are not guaranteed to be contiguous. Maybe we should have
00047
        // a reverse map, or reserve a register to store &cpus[i].
        for (i = 0; i < ncpu; ++i) {
   if (cpus[i].apicid == apicid)</pre>
00048
00049
00050
             return &cpus[i];
00051
00052
        panic("unknown apicid\n");
00053 }
```

Referenced by acquire(), cpuid(), holding(), myproc(), popcli(), pushcli(), sched(), sched(), startothers(), and switchuvm().

5.153.1.11 myproc()

Definition at line 58 of file proc.c.

```
00058
00059
        struct cpu *c;
        struct proc *p;
00060
00061
        pushcli();
00062
        c = mycpu();
        p = c->proc;
00063
00064
        popcli();
00065
        return p;
00066 }
```

Referenced by acquiresleep(), argfd(), argint(), argptr(), consoleread(), exec(), exit(), fdalloc(), fetchint(), fetchstr(), fork(), growproc(), holdingsleep(), namex(), piperead(), pipewrite(), sched(), sleep(), sys_chdir(), sys_close(), sys_getpid(), sys_pipe(), sys_sbrk(), sys_sleep(), syscall(), trap(), wait(), and yield().

5.153.1.12 pinit()

5.153.1.13 procdump()

```
void procdump (
                void )
Definition at line 504 of file proc.c.
00505 {
        static char *states[] = {
00507
         [UNUSED]
                      "unused",
00508
         [EMBRYO]
                      "embryo",
         [SLEEPING] "sleep",
00509
         [RUNNABLE] "runble",
00510
                     "run
00511
        [RUNNING]
                     "zombie"
00512
         [ZOMBIE]
00513
00514
        int i;
00515
        struct proc *p;
00516
        char *state;
00517
        uint pc[10];
00518
00519
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
00520
         if(p->state == UNUSED)
00521
             continue;
00522
          if(p->state >= 0 && p->state < NELEM(states) && states[p->state])
00523
            state = states[p->state];
           else
00524
            state = "???";
           cprintf("%d %s %s", p->pid, state, p->name);
00526
00527
           if(p->state == SLEEPING){
           getcallerpcs((uint*)p->context->ebp+2, pc);
for(i=0; i<10 && pc[i] != 0; i++)
   cprintf(" %p", pc[i]);</pre>
00528
00529
00530
00531
00532
          cprintf("\n");
00533
        }
00534 }
```

Referenced by consoleintr().

5.153.1.14 sched()

void sched (

```
void )
Definition at line 366 of file proc.c.
00367 {
00368
        int intena;
        struct proc *p = myproc();
00369
00370
00371
        if(!holding(&ptable.lock))
00372
          panic("sched ptable.lock");
        if (mycpu()->ncli != 1)
00373
00374
          panic("sched locks");
00375
        if(p->state == RUNNING)
00376
          panic ("sched running");
        if(readeflags()&FL_IF)
00378
          panic("sched interruptible");
00379
        intena = mycpu()->intena;
        swtch(&p->context, mycpu()->scheduler);
mycpu()->intena = intena;
00380
00381
00382 }
```

Referenced by exit(), sleep(), and yield().

5.153.1.15 scheduler()

```
void scheduler (
                void )
Definition at line 323 of file proc.c.
00325
         struct proc *p;
00326
         struct cpu *c = mycpu();
         c->proc = 0;
00327
00328
00329
00330
          // Enable interrupts on this processor.
00331
00332
00333
           // Loop over process table looking for process to run.
00334
           acquire(&ptable.lock);
           for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
   if(p->state != RUNNABLE)
00335
00336
00337
               continue;
00338
             // Switch to chosen process. It is the process's job // to release ptable.lock and then reacquire it
00339
00340
             // before jumping back to us.
00341
00342
             c->proc = p;
             switchuvm(p);
p->state = RUNNING;
00343
00344
00345
00346
             swtch(&(c->scheduler), p->context);
00347
             switchkvm();
00348
00349
             // Process is done running for now.
00350
              // It should have changed its p->state before coming back.
00351
             c \rightarrow proc = 0;
00352
00353
           release (&ptable.lock);
00354
00355
        }
```

Referenced by sched().

00356 }

5.153.1.16 sleep()

Definition at line 418 of file proc.c.

```
00419 {
00420
          struct proc *p = myproc();
00421
00422
          if(p == 0)
00423
            panic("sleep");
00424
00425
          if(1k == 0)
           panic("sleep without lk");
00426
00427
00428
          // Must acquire ptable.lock in order to
          // change p->state and then call sched.
00429
00430
          // Once we hold ptable.lock, we can be
          // guaranteed that we won't miss any wakeup
// (wakeup runs with ptable.lock locked),
00431
00432
          // (waked runs with plants.lock locked),
// so it's okay to release lk.
if(lk != &ptable.lock) { //DOC: sleeplock1
    acquire(&ptable.lock); //DOC: sleeplock1
00433
00434
00435
00436
           release(lk);
00437
          // Go to sleep.
00438
00439
          p->chan = chan;
p->state = SLEEPING;
00440
00441
00442
          sched();
```

Referenced by acquiresleep(), begin_op(), consoleread(), iderw(), main(), openiputtest(), piperead(), pipewrite(), sbrktest(), sys_sleep(), validatetest(), and wait().

5.153.1.17 trapret()

```
void trapret (
     void )
```

Referenced by allocproc().

5.153.1.18 userinit()

```
void userinit (
     void )
```

Definition at line 121 of file proc.c.

```
00122 {
00123
        extern char _binary_initcode_start[], _binary_initcode_size[];
00124
00125
00126
        p = allocproc();
00127
        initproc = p;
00128
        if((p->pgdir = setupkvm()) == 0)
panic("userinit: out of memory?");
00129
00130
00131
        inituvm(p->pgdir, _binary_initcode_start, (int)_binary_initcode_size);
00132
        p->sz = PGSIZE;
00133
        memset(p->tf, 0, sizeof(*p->tf));
        p->tf->cs = (SEG_UCODE « 3) | DPL_USER;
p->tf->ds = (SEG_UDATA « 3) | DPL_USER;
00134
00135
        p->tf->es = p->tf->ds;
00136
        p->tf->ss = p->tf->ds;
00137
        p->tf->eflags = FL_IF;
00138
00139
        p->tf->esp = PGSIZE;
00140
        p->tf->eip = 0; // beginning of initcode.S
00141
        safestrcpy(p->name, "initcode", sizeof(p->name));
p->cwd = namei("/");
00142
00143
00144
        // this assignment to p->state lets other cores
00146
        // run this process. the acquire forces the above
00147
        // writes to be visible, and the lock is also needed
00148
        // because the assignment might not be atomic.
00149
        acquire(&ptable.lock);
00150
00151
        p->state = RUNNABLE;
00152
00153
        release(&ptable.lock);
00154 }
```

5.153.1.19 wait()

```
int wait (
                void )
Definition at line 273 of file proc.c.
00274
00275
         struct proc *p;
        int havekids, pid;
struct proc *curproc = myproc();
00277
00278
00279
         acquire(&ptable.lock);
00280
        for(;;){
  // Scan through table looking for exited children.
00281
           havekids = 0;
00283
           for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
00284
             if(p->parent != curproc)
             continue;
havekids = 1;
00285
00286
00287
             if(p->state == ZOMBIE){
00288
               // Found one.
                pid = p->pid;
00290
               kfree (p->kstack);
00291
                p->kstack = 0;
               freevm(p->pgdir);
p->pid = 0;
00292
00293
00294
               p->parent = 0;
00295
               p->name[0] = 0;
00296
               p->killed = 0;
               p->state = UNUSED;
00297
00298
                release (&ptable.lock);
00299
                return pid;
00300
00301
00302
00303
           // No point waiting if we don't have any children.
00304
           if(!havekids || curproc->killed){
00305
            release(&ptable.lock);
00306
             return -1;
00308
           // Wait for children to exit. (See wakeup1 call in proc_exit.)
sleep(curproc, &ptable.lock); //DOC: wait-sleep
00309
00310
        }
```

Referenced by bigargtest(), concreate(), createdelete(), exitiputtest(), exitwait(), forktest(), fourfiles(), linkunlink(), main(), mem(), openiputtest(), pipe1(), preempt(), runcmd(), sbrktest(), sharedfd(), sys_wait(), uio(), and validatetest().

5.153.1.20 wakeup()

00311 00312 }

```
void wakeup (
     void * chan )
```

Definition at line 469 of file proc.c.

```
00470 {
00471          acquire(&ptable.lock);
00472          wakeupl(chan);
00473          release(&ptable.lock);
00474 }
```

Referenced by consoleintr(), end_op(), ideintr(), pipeclose(), piperead(), pipewrite(), releasesleep(), and trap().

5.153.1.21 wakeup1()

```
static void wakeup1 (
     void * chan ) [static]
```

Definition at line 458 of file proc.c.

Referenced by exit(), and wakeup().

5.153.1.22 yield()

```
void yield (
     void )
```

Definition at line 386 of file proc.c.

```
00387 {
00388    acquire(&ptable.lock); //DOC: yieldlock
00389    myproc()->state = RUNNABLE;
00390    sched();
00391    release(&ptable.lock);
00392 }
```

Referenced by trap().

5.153.2 Variable Documentation

5.153.2.1 initproc

```
struct proc* initproc [static]
```

Definition at line 15 of file proc.c.

Referenced by exit(), and userinit().

5.153.2.2 lock

```
struct spinlock lock
```

Definition at line 11 of file proc.c.

5.153.2.3 nextpid

```
int nextpid = 1
```

Definition at line 17 of file proc.c.

Referenced by allocproc().

5.153.2.4 proc

```
struct proc proc[NPROC]
```

Definition at line 12 of file proc.c.

5.153.2.5

```
struct { ... } ptable
```

Referenced by allocproc(), chpr(), cps(), exit(), fork(), forkret(), kill(), pinit(), procdump(), sched(), scheduler(), sleep(), userinit(), wait(), wakeup(), wakeup1(), and yield().

5.154 proc.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "defs.h"
00003 #include "param.h"
00004 #include "memlayout.h"
00004 #include "mmu.h"
00006 #include "x86.h"
00007 #include "proc.h"
00008 #include "spinlock.h"
00009
00010 struct {
00011 struct spinlock lock;
00012 struct proc proc[NPRO
         struct proc proc[NPROC];
00013 } ptable;
00014
00015 static struct proc *initproc;
00016
00017 int nextpid = 1;
00018 extern void forkret(void);
00019 extern void trapret (void);
00020
00021 static void wakeup1 (void *chan);
00022
00023 void
00024 pinit (void)
00025 {
         initlock(&ptable.lock, "ptable");
00026
00027 }
00028
00029 // Must be called with interrupts disabled
00030 int
00031 cpuid() {
00032
         return mycpu()-cpus;
00033 }
00034
00035 // Must be called with interrupts disabled to avoid the caller being
00036 // rescheduled between reading lapicid and running through the loop.
```

5.154 proc.c 337

```
00037 struct cpu*
00038 mycpu(void)
00039 {
00040
        int apicid, i;
00041
00042
        if (readeflags() &FL_IF)
         panic("mycpu called with interrupts enabled\n");
00044
00045
        apicid = lapicid();
       // APIC IDs are not guaranteed to be contiguous. Maybe we should have
00046
        // a reverse map, or reserve a register to store &cpus[i].
00047
        for (i = 0; i < ncpu; ++i) {
  if (cpus[i].apicid == apicid)</pre>
00048
00049
00050
           return &cpus[i];
00051
00052 panic("unknown apicid\n");
00053 }
00054
00055 // Disable interrupts so that we are not rescheduled
00056 // while reading proc from the cpu structure
00057 struct proc*
00058 myproc(void) {
00059
       struct cpu *c;
00060
       struct proc *p;
00061
       pushcli();
00062
       c = mycpu();
       p = c->proc;
00063
       popcli();
00064
00065
        return p;
00066 }
00067
00068 //PAGEBREAK: 32
00069 // Look in the process table for an UNUSED proc.
00070 // If found, change state to EMBRYO and initialize
00071 // state required to run in the kernel.
00072 // Otherwise return 0.
00073 static struct proc*
00074 allocproc(void)
00075 {
00076
       struct proc *p;
00077
       char *sp;
00078
00079
       acquire(&ptable.lock);
08000
00081
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)</pre>
00082
          if(p->state == UNUSED)
00083
           goto found;
00084
00085
        release (&ptable.lock);
00086
        return 0:
00087
00088 found:
        p->state = EMBRYO;
00089
00090
        p->pid = nextpid++;
00091
00092
        release(&ptable.lock);
00093
00094
        // Allocate kernel stack.
00095
        if((p->kstack = kalloc()) == 0){
        p->state = UNUSED;
00096
00097
          return 0:
00098
00099
        sp = p->kstack + KSTACKSIZE;
00100
00101
        // Leave room for trap frame.
00102
        sp -= sizeof *p->tf;
00103
        p->tf = (struct trapframe*)sp;
00104
00105
        // Set up new context to start executing at forkret,
        // which returns to trapret.
00106
00107
        sp -= 4;
00108
        *(uint*)sp = (uint)trapret;
00109
00110
       sp -= sizeof *p->context;
       p->context = (struct context*)sp;
00111
00112
        memset(p->context, 0, sizeof *p->context);
00113
       p->context->eip = (uint)forkret;
00114
00115
       return p;
00116 }
00117
00118 //PAGEBREAK: 32
00119 // Set up first user process.
00120 void
00121 userinit(void)
00122 {
00123
       struct proc *p;
```

```
extern char _binary_initcode_start[], _binary_initcode_size[];
00125
         p = allocproc();
00126
00127
00128
        initproc = p;
if((p->pqdir = setupkvm()) == 0)
00129
           panic("userinit: out of memory?");
00130
00131
         inituvm(p->pgdir, _binary_initcode_start, (int)_binary_initcode_size);
00132
         p->sz = PGSIZE;
        memset(p->tf, 0, sizeof(*p->tf));
p->tf->cs = (SEG_UCODE « 3) | DPL_USER;
p->tf->ds = (SEG_UDATA « 3) | DPL_USER;
00133
00134
00135
        p->tf->ds = (050_05)ffff
p->tf->es = p->tf->ds;
p->tf->ss = p->tf->ds;
00136
00137
00138
         p->tf->eflags = FL_IF;
         p->tf->esp = PGSIZE;
00139
         p->tf->eip = 0; // beginning of initcode.S
00140
00141
        safestrcpy(p->name, "initcode", sizeof(p->name));
00142
00143
        p->cwd = namei("/");
00144
00145
         // this assignment to p->state lets other cores
        // run this process. the acquire forces the above
// writes to be visible, and the lock is also needed
// because the assignment might not be atomic.
00146
00147
00148
        acquire(&ptable.lock);
00150
00151
        p->state = RUNNABLE;
00152
00153
        release(&ptable.lock);
00154 }
00155
00156 // Grow current process's memory by n bytes.
00157 // Return 0 on success, -1 on failure.
00158 int
00159 growproc(int n)
00160 {
        uint sz;
00161
00162
        struct proc *curproc = myproc();
00163
00164
        sz = curproc->sz;
00165
        if(n > 0){
         if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
00166
             return -1;
00167
        } else if(n < 0){
00168
00169
          if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
00170
             return -1;
00171
00172
        curproc->sz = sz:
00173
        switchuvm(curproc);
00174
        return 0;
00175 }
00176
00177 // Create a new process copying p as the parent. 00178 // Sets up stack to return as if from system call.
00179 // Caller must set state of returned proc to RUNNABLE.
00180 int
00181 fork (void)
00182 {
        int i, pid;
struct proc *np;
00183
00184
00185
        struct proc *curproc = myproc();
00186
00187
         // Allocate process.
00188
         if((np = allocproc()) == 0){
00189
          return -1;
00190
00191
00192
         // Copy process state from proc.
         if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
00193
00194
          kfree(np->kstack);
00195
           np->kstack = 0;
          np->state = UNUSED;
00196
           return -1;
00197
00198
00199
         np->sz = curproc->sz;
00200
         np->parent = curproc;
00201
         *np->tf = *curproc->tf;
00202
00203
         // Clear %eax so that fork returns 0 in the child.
        np->tf->eax = 0;
00204
00205
00206
         for(i = 0; i < NOFILE; i++)</pre>
         if(curproc->ofile[i])
00207
00208
             np->ofile[i] = filedup(curproc->ofile[i]);
00209
         np->cwd = idup(curproc->cwd);
00210
```

5.154 proc.c 339

```
safestrcpy(np->name, curproc->name, sizeof(curproc->name));
00212
00213
        pid = np->pid;
00214
00215
        acquire(&ptable.lock);
00216
00217
        np->state = RUNNABLE;
00218
00219
        release(&ptable.lock);
00220
00221
        return pid;
00222 }
00223
00224 // Exit the current process. Does not return.
00225 // An exited process remains in the zombie state
00226 \ensuremath{//} until its parent calls wait() to find out it exited.
00227 void
00228 exit (void)
00229 {
00230
        struct proc *curproc = myproc();
00231
        struct proc *p;
00232
        int fd;
00233
        if(curproc == initproc)
00234
00235
          panic("init exiting");
00236
00237
         \ensuremath{//} Close all open files.
00238
        for(fd = 0; fd < NOFILE; fd++) {</pre>
00239
          if(curproc->ofile[fd]){
             fileclose(curproc->ofile[fd]);
00240
00241
             curproc->ofile[fd] = 0;
00242
          }
00243
00244
00245
        begin_op();
00246
        iput (curproc->cwd);
00247
        end op();
00248
        curproc->cwd = 0;
00249
00250
        acquire(&ptable.lock);
00251
00252
        // Parent might be sleeping in wait().
00253
        wakeup1(curproc->parent);
00254
00255
         // Pass abandoned children to init.
00256
         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
          if (p->parent == curproc) {
  p->parent = initproc;
00257
00258
             if (p->state == ZOMBIE)
00259
00260
               wakeup1(initproc);
00261
          }
00262
00263
        // Jump into the scheduler, never to return.
curproc->state = ZOMBIE;
00264
00265
00266
        sched();
00267
        panic("zombie exit");
00268 }
00269
00270 \ensuremath{//} Wait for a child process to exit and return its pid.
00271 // Return -1 if this process has no children.
00272 int
00273 wait (void)
00274 {
        struct proc *p;
00275
        int havekids, pid;
struct proc *curproc = myproc();
00276
00277
00278
00279
        acquire(&ptable.lock);
        for(j;){
   // Scan through table looking for exited children.
00280
00281
00282
          havekids = 0;
           for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){</pre>
00283
             if (p->parent != curproc)
   continue;
00284
00285
00286
             havekids = 1;
00287
             if(p->state == ZOMBIE){
00288
               // Found one.
00289
               pid = p->pid;
               kfree (p->kstack);
00290
               p->kstack = 0;
00291
00292
               freevm(p->pgdir);
00293
               p->pid = 0;
00294
               p->parent = 0;
               p->name[0] = 0;
p->killed = 0;
00295
00296
               p->state = UNUSED;
00297
```

```
release(&ptable.lock);
00299
               return pid;
00300
            }
          }
00301
00302
00303
           // No point waiting if we don't have any children.
           if(!havekids || curproc->killed){
00305
            release(&ptable.lock);
00306
             return -1;
00307
00308
          // Wait for children to exit. (See wakeupl call in proc_exit.)
sleep(curproc, &ptable.lock); //DOC: wait-sleep
00309
00310
00311
00312 }
00313
00314 //PAGEBREAK: 42
00315 // Per-CPU process scheduler.
00316 // Each CPU calls scheduler() after setting itself up.
00317 // Scheduler never returns. It loops, doing:
00318 // - choose a process to run
00319 // - swtch to start running that process
00320 // - eventually that process transfers control
00321 //
               via swtch back to the scheduler.
00322 void
00323 scheduler(void)
00324 {
00325
        struct proc *p;
00326
        struct cpu *c = mycpu();
        c->proc = 0;
00327
00328
00329
        for(;;) {
00330
          // Enable interrupts on this processor.
00331
           sti();
00332
           \ensuremath{//} Loop over process table looking for process to run.
00333
00334
           acquire(&ptable.lock);
           for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
00335
00336
            if(p->state != RUNNABLE)
00337
               continue;
00338
             // Switch to chosen process. It is the process's job // to release ptable.lock and then reacquire it
00339
00340
             // before jumping back to us.
00341
             c->proc = p;
00342
00343
             switchuvm(p);
00344
            p->state = RUNNING;
00345
00346
             swtch(&(c->scheduler), p->context);
00347
             switchkvm();
00348
00349
             // Process is done running for now.
00350
             // It should have changed its p->state before coming back.
00351
            c->proc = 0;
00352
00353
          release(&ptable.lock);
00354
00355
        }
00356 }
00357
00358 // Enter scheduler. Must hold only ptable.lock
00359 // and have changed proc->state. Saves and restores
00360 // intena because intena is a property of this
00361 // kernel thread, not this CPU. It should
00362 // be proc->intena and proc->ncli, but that would
00363 // break in the few places where a lock is held but
00364 // there's no process.
00365 void
00366 sched(void)
00367 {
00368
        int intena;
00369
        struct proc *p = myproc();
00370
00371
        if(!holding(&ptable.lock))
        panic("sched ptable.lock");
if(mycpu()->ncli != 1)
00372
00373
00374
          panic("sched locks");
00375
        if(p->state == RUNNING)
00376
          panic("sched running");
        if (readeflags() &FL_IF)
00377
00378
          panic ("sched interruptible");
00379
        intena = mycpu()->intena;
        swtch(&p->context, mycpu()->scheduler);
mycpu()->intena = intena;
00380
00381
00382 }
00383
00384 // Give up the CPU for one scheduling round.
```

5.154 proc.c 341

```
00385 void
00386 yield(void)
00387 {
         acquire(&ptable.lock); //DOC: yieldlock
00388
00389
         myproc()->state = RUNNABLE;
00390
         sched();
00391
        release(&ptable.lock);
00392 }
00393
00394 // A fork child's very first scheduling by scheduler() 00395 // will swtch here. "Return" to user space.
00396 void
00397 forkret (void)
00398 {
00399
         static int first = 1;
00400
         // Still holding ptable.lock from scheduler.
00401
         release (&ptable.lock);
00402
00403
          // Some initialization functions must be run in the context
// of a regular process (e.g., they call sleep), and thus cannot
// be run from main().
00404
00405
00406
00407
           first = 0;
           iinit (ROOTDEV);
00408
00409
            initlog(ROOTDEV);
00410
00411
00412
         // Return to "caller", actually trapret (see allocproc).
00413 }
00414
00415 // Atomically release lock and sleep on chan. 00416 // Reacquires lock when awakened.
00417 void
00418 sleep(void *chan, struct spinlock *lk)
00419 {
00420
         struct proc *p = myproc();
00421
         if(p == 0)
00423
          panic("sleep");
00424
00425
         if(1k == 0)
         panic("sleep without lk");
00426
00427
00428
         // Must acquire ptable.lock in order to
         // change p->state and then call sched.
00429
00430
         // Once we hold ptable.lock, we can be
         // guaranteed that we won't miss any wakeup
// (wakeup runs with ptable.lock locked),
00431
00432
         // (match plant and plant stocker);
// so it's okay to release lk.
if(lk != &ptable.lock) {    //DOC: sleeplock0
    acquire(&ptable.lock);    //DOC: sleeplock1
00433
00434
00435
00436
           release(lk);
00437
00438
         // Go to sleep.
        p->chan = chan;
p->state = SLEEPING;
00439
00440
00441
00442
00443
         // Tidy up.
00444
00445
         p->chan = 0;
00446
00447
         // Reacquire original lock.
00448
         if(lk != &ptable.lock) { //DOC: sleeplock2
00449
          release(&ptable.lock);
00450
           acquire(lk);
00451
00452 }
00453
00454 //PAGEBREAK!
00455 // Wake up all processes sleeping on chan.
00456 // The ptable lock must be held.
00457 static void
00458 wakeup1(void *chan)
00459 {
00460
        struct proc *p;
00461
00462
         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)</pre>
           if(p->state == SLEEPING && p->chan == chan)
p->state = RUNNABLE;
00463
00464
00465 }
00466
00467 // Wake up all processes sleeping on chan.
00468 void
00469 wakeup(void *chan)
00470 {
00471
        acquire(&ptable.lock);
```

```
wakeup1(chan);
00473
        release(&ptable.lock);
00474 }
00475
00476 // Kill the process with the given pid. 00477 // Process won't exit until it returns
00478 // to user space (see trap in trap.c).
00479 int
00480 kill(int pid)
00481 {
00482
        struct proc *p;
00483
00484
         acquire(&ptable.lock);
00485
         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
00486
          if (p->pid == pid) {
             p->killed = 1;
00487
              // Wake process from sleep if necessary.
00488
              if (p->state == SLEEPING)
00489
               p->state = RUNNABLE;
00490
00491
             release(&ptable.lock);
00492
             return 0;
00493
          }
00494
        release(&ptable.lock);
00495
00496
        return -1;
00497 }
00498
00499 //PAGEBREAK: 36
00500 // Print a process listing to console. For debugging. 00501 // Runs when user types {\rm ^{2}P} on console.
00502 // No lock to avoid wedging a stuck machine further.
00503 void
00504 procdump(void)
00505 {
         static char *states[] = {
[UNUSED] "unused",
00506
         [UNUSED]
00507
         [EMBRYO]
                       "embryo",
00508
         [SLEEPING]
                      "sleep ",
00510
         [RUNNABLE]
                      "runble",
00511
         [RUNNING]
                      "run
00512
         [ZOMBIE]
                       "zombie"
00513
        };
int i;
00514
00515
        struct proc *p;
00516
         char *state;
00517
         uint pc[10];
00518
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if(p->state == UNUSED)
00519
00520
00521
             continue:
           if(p->state >= 0 && p->state < NELEM(states) && states[p->state])
00523
             state = states[p->state];
00524
           else
           state = "???";
cprintf("%d %s %s", p->pid, state, p->name);
if(p->state == SLEEPING) {
00525
00526
00527
            getcallerpcs((uint*)p->context->ebp+2, pc);
00529
              for(i=0; i<10 && pc[i] != 0; i++)
00530
               cprintf(" %p", pc[i]);
00531
           cprintf("\n");
00532
00533
        }
00534 }
00535
00536 int
00537 cps()
00538 {
00539 struct proc *p;
00540 //Enables interrupts on this processor.
00541 sti();
00542
00543 //Loop over process table looking for process with pid.
00544 acquire(&ptable.lock);
00545 cprintf("name \t pid \t state \t priority \n");

00546 for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {

00547 if(p->state == SLEEPING)
00548
          cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name,p->pid,p->priority);
00549
        else if(p->state == RUNNING)
          cprintf("%s \t %d \t RUNNING \t %d \n ", p->name,p->pid,p->priority);
00550
00551
        else if (p->state == RUNNABLE)
           cprintf("%s \t %d \t RUNNABLE \t %d \n ", p->name,p->pid,p->priority);
00552
00554 release(&ptable.lock);
00555 return 23;
00556 }
00557
00558 int
```

5.155 proc.d File Reference

5.156 proc.d

```
Go to the documentation of this file.

00001 proc.o: proc.c /usr/include/stdc-predef.h types.h defs.h param.h \
00002 memlayout.h mmu.h x86.h proc.h spinlock.h
```

5.157 proc.h File Reference

Classes

- struct context
- struct cpu
- struct proc

Enumerations

```
    enum procstate {
        UNUSED, EMBRYO, SLEEPING, RUNNABLE,
        RUNNING, ZOMBIE}
```

Variables

- struct cpu cpus [NCPU]
- int ncpu

5.157.1 Enumeration Type Documentation

5.157.1.1 procstate

```
enum procstate
```

Enumerator

UNUSED	
EMBRYO	
SLEEPING	
RUNNABLE	
RUNNING	
ZOMBIE	

```
Definition at line 35 of file proc.h.
00035 { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
```

5.157.2 Variable Documentation

5.157.2.1 cpus

```
struct cpu cpus[NCPU] [extern]
```

Definition at line 14 of file mp.c.

Referenced by cpuid(), mpinit(), mycpu(), seginit(), and startothers().

5.157.2.2 ncpu

```
int ncpu [extern]
```

Definition at line 15 of file mp.c.

Referenced by ideinit(), mpinit(), mycpu(), and startothers().

5.158 proc.h

Go to the documentation of this file.

```
00001 // Per-CPU state
00002 struct cpu {
00003 uchar apicid;
00004 struct context *scheduler;
00005 struct taskstate ts;
                                        // Local APIC ID
                                       // swtch() here to enter scheduler
                                        // Used by x86 to find stack for interrupt
00006
        struct segdesc gdt[NSEGS];
                                        // x86 global descriptor table
        volatile uint started;
                                        // Has the CPU started?
// Depth of pushcli nesting.
00007
80000
        int ncli;
00009
                                        // Were interrupts enabled before pushcli?
        int intena:
00010
                                        // The process running on this cpu or null
        struct proc *proc;
00011 };
00012
00013 extern struct cpu cpus[NCPU];
00014 extern int ncpu;
00015
00016 //PAGEBREAK: 17
00017 // Saved registers for kernel context switches.
00018 // Don't need to save all the segment registers (%cs, etc),
```

```
00019 // because they are constant across kernel contexts.
00020 // Don't need to save %eax, %ecx, %edx, because the
00021 // x86 convention is that the caller has saved them.
00022 // Contexts are stored at the bottom of the stack they
00023 // describe; the stack pointer is the address of the context.
00024 // The layout of the context matches the layout of the stack in swtch.S
00025 // at the "Switch stacks" comment. Switch doesn't save eip explicitly,
00026 // but it is on the stack and allocproc() manipulates it.
00027 struct context {
00028 uint edi;
00029
        uint esi:
00030
        uint ebx:
00031
        uint ebp;
00032 uint eip;
00033 };
00034
00035 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
00036
00037 // Per-process state
00038 struct proc {
00039 uint sz;
                                          // Size of process memory (bytes)
       pde_t* pgdir;
char *kstack;
00040
                                          // Page table
00041
                                          // Bottom of kernel stack for this process
                                         // Process state
00042
        enum procstate state;
00043
                                         // Process ID
        int pid;
00044 struct proc *parent;
                                         // Parent process
                                         // Trap frame for current syscall
00045
        struct trapframe *tf;
                                        // swtch() here to run process
// If non-zero, sleeping on chan
00046
        struct context *context;
00047
        void *chan;
00048 int killed; // If non-zero 00049 struct file *ofile[NOFILE]; // Open files
                                          // If non-zero, have been killed
       struct inode *cwd;
00050
                                         // Current directory
00051 char name[16];
00052 int priority;
                                         // Process name (debugging)
        int priority;
                                          // Process priority
00053 };
00054
00055 // Process memory is laid out contiguously, low addresses first:
00056 // text
00057 //
            original data and bss
00058 //
            fixed-size stack
00059 //
           expandable heap
```

5.159 ps.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
```

Functions

• int main (void)

5.159.1 Function Documentation

5.159.1.1 main()

5.160 ps.c

Go to the documentation of this file.

```
00001 #include "types.h"

00002 #include "stat.h"

00003 #include "user.h"

00004 #include "fcntl.h"

00005

00006 int main(void) {

00007 cps();

00008 exit();

00009 }
```

5.161 ps.d File Reference

5.162 ps.d

Go to the documentation of this file.

```
00001 ps.o: ps.c /usr/include/stdc-predef.h types.h stat.h user.h fcntl.h
```

5.163 pstree.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "uproc.h"
```

Macros

• #define MAXPROC 64

Functions

• int main (int argc, char *argv[])

5.163.1 Macro Definition Documentation

5.163.1.1 MAXPROC

```
#define MAXPROC 64
```

Definition at line 5 of file pstree.c.

5.164 pstree.c 347

5.163.2 Function Documentation

5.163.2.1 main()

```
int main (
                 int argc,
                 char * argv[] )
Definition at line 8 of file pstree.c.
00008
00009
         // declare a uproc struct with an allocated size of 64
         struct uproc *prs = malloc(MAXPROC*sizeof(struct uproc));
00010
00011
00012
         // call getprocs in kernel space and get the amount of processes running back
00013
         int num = getprocs(MAXPROC, prs);
00014
00015
         // getprocs will return -1 if an error is received. Handle errors here
00016
         if (num == -1) {
00017
           printf(1, "Kernel returned an error when getting processes. \n");
00018
           exit();
00019
00020
         // create a counter
00021
00022
         int i=0;
00023
00024
         // loop through uproc structs that were created in the kernel space for each process
00025
         for(; i < num; i++) +</pre>
00026
          // format output to the user
00027
           if(prs[i].pid == 1) {
           // if the pid is 1 then this is init
printf(1, "%s[%d] \n", prs[i].name, prs[i].pid-1);
}else if(prs[i].ppid == 1){
00028
00029
00030
              // if parent pid is 1, then it is a child to parent init
printf(1, " %s[%d] \n", prs[i].name, prs[i].pid-1);
00031
00032
00033
           }else {
              // if parent pid is not 1, then it is a grandchild to init and child to another process
printf(1, " %s[%d] \n", prs[i].name, prs[i].pid-1);
00034
00035
00036
00037
00038
00039
         // exit after pstree command is done
00040
        exit();
00041 }
```

5.164 pstree.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004 #include "uproc.h"
00005 #define MAXPROC 64
00006
00007 int
00008 main(int argc, char *argv[]){
       // declare a uproc struct with an allocated size of 64
00009
       struct uproc *prs = malloc(MAXPROC*sizeof(struct uproc));
00010
00011
00012
        // call getprocs in kernel space and get the amount of processes running back
00013
        int num = getprocs(MAXPROC, prs);
00014
00015
        // getprocs will return -1 if an error is received. Handle errors here
00016
        if(num == -1) {
         printf(1, "Kernel returned an error when getting processes. \n");
00017
00018
          exit();
00019
00020
00021
        // create a counter
00022
        int i=0:
00023
       // loop through uproc structs that were created in the kernel space for each process
```

```
for(; i< num; i++) {</pre>
              // format output to the user
if(prs[i].pid == 1) {
00026
00027
                // if the pid is 1 then this is init
printf(1, "%s[%d] \n", prs[i].name, prs[i].pid-1);
}else if(prs[i].ppid == 1) {
   // if parent pid is 1, then it is a child to parent init
   printf(1, " %s[%d] \n", prs[i].name, prs[i].pid-1);
00028
00029
00030
00032
                 }else {
// if parent pid is not 1, then it is a grandchild to init and child to another process
printf(1, " %s[%d] \n", prs[i].name, prs[i].pid-1);
00033
00034
00035
00036
00037
00038
00039
             // exit after pstree command is done
00040 exit();
00041 }
```

5.165 pstree.d File Reference

5.166 pstree.d

```
Go to the documentation of this file.
```

```
00001 pstree.o: pstree.c /usr/include/stdc-predef.h types.h stat.h user.h \backslash 00002 uproc.h
```

5.167 README.md File Reference

5.168 rm.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

• int main (int argc, char *argv[])

5.168.1 Function Documentation

5.169 rm.c 349

5.168.1.1 main()

```
int main (
                 int argc,
                 char * argv[])
Definition at line 6 of file rm.c.
00007 {
80000
00009
         if(argc < 2) {
  printf(2, "Usage: rm files...\n");
  exit();</pre>
00010
00011
00012
00013
00014
00015
         for(i = 1; i < argc; i++) {</pre>
        if(unlink(argv[i]) < 0){
  printf(2, "rm: %s failed to delete\n", argv[i]);</pre>
00016
00017
00018
              break;
00019
           }
00020 }
00021
00022
        exit();
00023 }
```

5.169 rm.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004
00005 int
00006 main(int argc, char *argv[])
00007 {
00008
         int i;
00009
00010
         if(argc < 2){</pre>
         printf(2, "Usage: rm files...\n");
00011
00012 exit();
00013 }
         for(i = 1; i < argc; i++) {
   if(unlink(argv[i]) < 0) {
     printf(2, "rm: %s failed to delete\n", argv[i]);</pre>
00015
00016
00018
00019
00020 }
00021
00022
         exit();
00023 }
```

5.170 rm.d File Reference

5.171 rm.d

```
Go to the documentation of this file.
```

```
00001 rm.o: rm.c /usr/include/stdc-predef.h types.h stat.h user.h
```

5.172 sh.c File Reference

```
#include "types.h"
#include "user.h"
#include "fcntl.h"
```

Classes

- struct backcmd
- struct cmd
- · struct execomd
- struct listcmd
- struct pipecmd
- · struct redircmd

Macros

- #define BACK 5
- #define EXEC 1
- #define LIST 4
- #define MAXARGS 10
- #define PIPE 3
- #define REDIR 2

Functions

- struct cmd * backcmd (struct cmd *subcmd)
- struct cmd * execcmd (void)
- int fork1 (void)
- int getcmd (char *buf, int nbuf)
- int gettoken (char **ps, char *es, char **q, char **eq)
- struct cmd * listcmd (struct cmd *left, struct cmd *right)
- int main (void)
- struct cmd * nulterminate (struct cmd *)
- void panic (char *)
- struct cmd * parseblock (char **ps, char *es)
- struct cmd * parsecmd (char *)
- struct cmd * parseexec (char **, char *)
- struct cmd * parseline (char **, char *)
- struct cmd * parsepipe (char **, char *)
- struct cmd * parseredirs (struct cmd *cmd, char **ps, char *es)
- int peek (char **ps, char *es, char *toks)
- struct cmd * pipecmd (struct cmd *left, struct cmd *right)
- struct cmd * redircmd (struct cmd *subcmd, char *file, char *efile, int mode, int fd)
- void runcmd (struct cmd *cmd)

Variables

- char symbols [] = "<|>&;()"
- char whitespace [] = " \t\r\n\v"

5.172.1 Macro Definition Documentation

5.172 sh.c File Reference 351

5.172.1.1 BACK

#define BACK 5

Definition at line 12 of file sh.c.

5.172.1.2 EXEC

#define EXEC 1

Definition at line 8 of file sh.c.

5.172.1.3 LIST

#define LIST 4

Definition at line 11 of file sh.c.

5.172.1.4 MAXARGS

#define MAXARGS 10

Definition at line 14 of file sh.c.

5.172.1.5 PIPE

#define PIPE 3

Definition at line 10 of file sh.c.

5.172.1.6 REDIR

#define REDIR 2

Definition at line 9 of file sh.c.

5.172.2 Function Documentation

5.172.2.1 backcmd()

Definition at line 249 of file sh.c.

5.172.2.2 execcmd()

Definition at line 196 of file sh.c.

```
00197 {
00198    struct execcmd *cmd;
00199
00200    cmd = malloc(sizeof(*cmd));
00201    memset(cmd, 0, sizeof(*cmd));
00202    cmd->type = EXEC;
00203    return (struct cmd*)cmd;
00204 }
```

5.172.2.3 fork1()

```
int fork1 (
     void )
```

Definition at line 182 of file sh.c.

```
00183 {
00184    int pid;
00185
00186    pid = fork();
00187    if(pid == -1)
00188         panic("fork");
00189    return pid;
00190 }
```

Referenced by main(), and runcmd().

5.172 sh.c File Reference 353

5.172.2.4 getcmd()

Referenced by main().

5.172.2.5 gettoken()

Definition at line 266 of file sh.c.

```
00267 {
00268
        char *s;
        int ret;
00270
00271
        s = *ps;
        while(s < es && strchr(whitespace, *s))
s++;</pre>
00272
00273
00274
        if(q)
        *q = s;
ret = *s;
00275
00276
00277
        switch(*s){
00278
        case 0:
00279
        break; case '|':
00280
        case '(':
00281
00282
        case ')':
00283
        case '&':
00284
        case '<':
00285
00286
        s++;
break;
case '>':
00287
00288
         s++;
00289
          if(*s == '>') {
  ret = '+';
00290
00291
00292
            s++;
00293
           }
00294
           break;
00295
         ret = 'a';
00296
          while(s < es && !strchr(whitespace, *s) && !strchr(symbols, *s))</pre>
00297
00298
            s++;
00299
          break;
00300
00301
        if (eq)
00302
          *eq = s;
00303
00304
        while(s < es && strchr(whitespace, *s))</pre>
00305
        s++;
*ps = s;
00306
00307
        return ret;
00308 }
```

Referenced by parseblock(), parseexec(), parseline(), parsepipe(), and parseredirs().

5.172.2.6 listcmd()

```
struct cmd * listcmd (
                struct cmd * left,
                struct cmd * right )
Definition at line 236 of file sh.c.
         struct listemd *cmd;
00238
00239
        cmd = malloc(sizeof(*cmd));
00240
        memset(cmd, 0, sizeof(*cmd));
cmd->type = LIST;
cmd->left = left;
00241
00242
00243
        cmd->right = right;
00244
00245
        return (struct cmd*)cmd;
00246 }
```

5.172.2.7 main()

```
int main (
     void )
```

Definition at line 145 of file sh.c.

```
00146 {
00147
          static char buf[100];
00148
          int fd:
00149
          // Ensure that three file descriptors are open.
while((fd = open("console", O_RDWR))) >= 0) {
  if(fd >= 3) {
00150
00152
00153
                close(fd);
00154
                break;
00155
00156
00157
00158
          // Read and run input commands.
          while(getcmd(buf, sizeof(buf)) >= 0){
  if(buf[0] == 'c' && buf[1] == 'd' && buf[2] == ' '){
00159
00160
               // Chdir must be called by the parent, not the child. buf[strlen(buf)-1] = 0; // chop \n
00161
00162
               if(chdir(buf+3) < 0)
printf(2, "cannot cd %s\n", buf+3);</pre>
00163
00164
00165
00166
            if(fork1() == 0)
00167
               runcmd(parsecmd(buf));
00168
00169
             wait();
00170
00171 exit();
00172 }
```

5.172.2.8 nulterminate()

Definition at line 450 of file sh.c.

```
00451 {
00452 int i;
00453 struct backemd *bemd;
00454 struct execomd *ecmd;
00455 struct listemd *lemd;
00456 struct pipecmd *pemd;
00457 struct rediremd *remd;
```

5.172 sh.c File Reference 355

```
00458
00459
        if(cmd == 0)
00460
          return 0;
00461
        switch(cmd->type){
00462
00463
       case EXEC:
        ecmd = (struct execomd*)cmd;
00464
00465
         for(i=0; ecmd->argv[i]; i++)
00466
           *ecmd->eargv[i] = 0;
00467
         break;
00468
        case REDIR:
00469
        rcmd = (struct redircmd*)cmd;
nulterminate(rcmd->cmd);
00470
00471
00472
          *rcmd->efile = 0;
00473
00474
00475
       case PIPE:
        pcmd = (struct pipecmd*)cmd;
00477
          nulterminate(pcmd->left);
00478
          nulterminate(pcmd->right);
00479
         break;
00480
00481
        case LIST:
       lcmd = (struct listcmd*)cmd;
nulterminate(lcmd->left);
00482
00484
         nulterminate(lcmd->right);
00485
00486
00487
       case BACK:
        bcmd = (struct backcmd*)cmd;
00488
00489
          nulterminate(bcmd->cmd);
00490
         break;
00491
00492
       return cmd;
00493 }
```

Referenced by nulterminate(), and parsecmd().

5.172.2.9 panic()

Referenced by fork1(), parseblock(), parsecmd(), parseexec(), parseredirs(), and runcmd().

5.172.2.10 parseblock()

```
struct cmd * parseblock (
                 char ** ps,
                  char * es)
Definition at line 400 of file sh.c.
00401 {
         struct cmd *cmd;
00403
00404
         if(!peek(ps, es, "("))
00405
           panic("parseblock");
         gettoken(ps, es, 0, 0);
cmd = parseline(ps, es);
if(!peek(ps, es, ")"))
panic("syntax - missing )");
00406
00407
00408
00409
00410
         gettoken(ps, es, 0, 0);
00411
         cmd = parseredirs(cmd, ps, es);
00412
         return cmd;
00413 }
```

Referenced by parseexec().

5.172.2.11 parsecmd()

```
struct cmd * parsecmd (
                    char * s)
Definition at line 328 of file sh.c.
00329 +
00330
           char *es;
00331
          struct cmd *cmd;
00332
          es = s + strlen(s);
cmd = parseline(&s, es);
peek(&s, es, "");
if(s != es){
   printf(2, "leftovers: %s\n", s);
   panic("syntax");
00333
00334
00335
00336
00338
00339
00340
          nulterminate(cmd);
00341
           return cmd;
```

Referenced by main().

00342 }

5.172.2.12 parseexec()

Definition at line 416 of file sh.c.

```
00417 {
00418
         char *q, *eq;
00419
        int tok, argc;
00420
        struct execomd *cmd;
struct cmd *ret;
00421
00422
00423
        if (peek (ps, es, "("))
00424
          return parseblock(ps, es);
00425
00426
        ret = execcmd();
00427
        cmd = (struct execcmd*)ret;
00428
00429
        ret = parseredirs(ret, ps, es);
while(!peek(ps, es, "|)&;")){
00430
00431
          if((tok=gettoken(ps, es, &q, &eq)) == 0)
00432
00433
             break;
          if(tok != 'a')
00434
00435
            panic("syntax");
00436
           cmd->argv[argc] = q;
           cmd->eargv[argc] = eq;
00437
00438
           argc++;
          if(argc >= MAXARGS)
  panic("too many args");
00439
00440
00441
          ret = parseredirs(ret, ps, es);
00442 }
        cmd->argv[argc] = 0;
cmd->eargv[argc] = 0;
00443
00444
00445
        return ret;
00446 }
```

Referenced by parsepipe().

5.172 sh.c File Reference 357

5.172.2.13 parseline()

Definition at line 345 of file sh.c.

```
00346 {
           struct cmd *cmd;
00347
00348
           cmd = parsepipe(ps, es);
while(peek(ps, es, "&")){
  gettoken(ps, es, 0, 0);
00349
00350
00351
00352
              cmd = backcmd(cmd);
00353
          if(peek(ps, es, ";")){
  gettoken(ps, es, 0, 0);
  cmd = listcmd(cmd, parseline(ps, es));
00354
00355
00356
00357
00358
          return cmd;
00359 }
```

Referenced by parseblock(), parsecmd(), and parseline().

5.172.2.14 parsepipe()

Definition at line 362 of file sh.c.

Referenced by parseline(), and parsepipe().

5.172.2.15 parseredirs()

Definition at line 375 of file sh.c.

```
00376 {
00377
          int tok;
00378
         char *q, *eq;
00379
00380
         while(peek(ps, es, "<>")){
         tok = gettoken(ps, es, 0, 0);
if(gettoken(ps, es, &q, &eq) != 'a')
panic("missing file for redirection");
00381
00382
00383
00384
           switch(tok){
            case '<':
00385
```

```
cmd = redircmd(cmd, q, eq, O_RDONLY, 0);
  break;
case '>':
00387
00388
          cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
00389
00390
          break;
case '+': // »
00391
00392
          cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
00393
            break;
00394
00395
00396
       return cmd;
00397 }
```

Referenced by parseblock(), and parseexec().

5.172.2.16 peek()

Definition at line 311 of file sh.c.

```
00312 {
00313    char *s;
00314
00315    s = *ps;
00316    while(s < es && strchr(whitespace, *s))
00317    s++;
00318    *ps = s;
00319    return *s && strchr(toks, *s);</pre>
```

Referenced by parseblock(), parsecmd(), parseexec(), parseline(), parsepipe(), and parseredirs().

5.172.2.17 pipecmd()

Definition at line 223 of file sh.c.

5.172 sh.c File Reference 359

5.172.2.18 redircmd()

```
struct cmd * redircmd (
                struct cmd * subcmd,
                char * file,
                char * efile,
                int mode,
                int fd )
Definition at line 207 of file sh.c.
00208
00209
         struct redircmd *cmd;
00210
00211
        cmd = malloc(sizeof(*cmd));
        memset(cmd, 0, sizeof(*cmd));
cmd->type = REDIR;
cmd->cmd = subcmd;
00212
00213
00214
        cmd->file = file;
00215
        cmd->efile = efile;
00216
        cmd->mode = mode;
cmd->fd = fd;
00217
00218
00219
        return (struct cmd*)cmd;
00220 }
```

5.172.2.19 runcmd()

void runcmd (

```
struct cmd * cmd )
Definition at line 58 of file sh.c.
00059 {
00060
         int p[2];
00061
        struct backemd *bemd;
00062
        struct execomd *ecmd;
00063
        struct listcmd *lcmd;
00064
        struct pipecmd *pcmd;
00065
        struct redircmd *rcmd;
00066
00067
        if(cmd == 0)
00068
          exit();
00069
00070
        switch(cmd->type){
00071
        default:
          panic("runcmd");
00072
00073
00074
        case EXEC:
         ecmd = (struct execcmd*)cmd;
00075
00076
          if(ecmd->argv[0] == 0)
00077
          exec(ecmd->argv[0], ecmd->argv);
printf(2, "exec %s failed\n", ecmd->argv[0]);
00078
00079
00080
           break:
00081
00082
        case REDIR:
00083
         rcmd = (struct redircmd*)cmd;
00084
           close(rcmd->fd);
           if(open(rcmd->file, rcmd->mode) < 0){
  printf(2, "open %s failed\n", rcmd->file);
00085
00086
00087
             exit();
00088
00089
           runcmd(rcmd->cmd);
00090
          break;
00091
        case LIST:
00092
          lcmd = (struct listcmd*)cmd;
if(fork1() == 0)
00093
00094
00095
             runcmd(lcmd->left);
00096
          wait();
00097
00098
          runcmd(lcmd->right);
          break;
00099
00100
        case PIPE:
00101
          pcmd = (struct pipecmd*)cmd;
```

```
if(pipe(p) < 0)
  panic("pipe");
if(fork1() == 0){</pre>
00103
00104
00105
              close(1);
               dup(p[1]);
close(p[0]);
00106
00107
00108
               close(p[1]);
00109
               runcmd(pcmd->left);
00110
            if(fork1() == 0){
  close(0);
00111
00112
              dup(p[0]);
close(p[0]);
00113
00114
00115
               close(p[1]);
00116
               runcmd(pcmd->right);
00117
            close(p[0]);
close(p[1]);
00118
00119
00120
            wait();
00121
            wait();
00122
00123
         case BACK:
  bcmd = (struct backcmd*)cmd;
  if(fork1() == 0)
00124
00125
00126
00127
              runcmd(bcmd->cmd);
00128
00129
00130 exit();
00131 }
```

Referenced by main(), and runcmd().

5.172.3 Variable Documentation

5.172.3.1 symbols

```
char symbols[] = "<|>&;()"
```

Definition at line 263 of file sh.c.

Referenced by gettoken().

5.172.3.2 whitespace

```
char whitespace[] = " \t \n \v"
```

Definition at line 262 of file sh.c.

Referenced by gettoken(), and peek().

5.173 sh.c 361

5.173 sh.c

Go to the documentation of this file.

```
00001 // Shell.
00002
00003 #include "types.h"
00004 #include "user.h"
00005 #include "fcntl.h"
00006
00007 // Parsed command representation
00008 #define EXEC
00009 #define REDIR 2
00010 #define PIPE 3
00011 #define LIST
00012 #define BACK
00013
00014 #define MAXARGS 10
00015
00016 struct cmd { 00017 int type;
00018 };
00019
00020 struct execond {
00021
        int type;
       char *argv[MAXARGS];
00022
00023
        char *eargv[MAXARGS];
00024 };
00025
00026 struct redircmd {
00027 int type;
00028
        struct cmd *cmd;
00029
        char *file;
00030
        char *efile;
       int mode;
int fd;
00031
00032
00033 };
00034
00035 struct pipecmd {
00036 int type;
00037 struct cmd *left;
00038 struct cmd *right;
00039 };
00040
00041 struct listcmd {
00042 int type;
00043 struct cmd *left;
00044 struct cmd *right;
00045 };
00046
00047 struct backemd {
00048 int type;
00049 struct cm
        struct cmd *cmd;
00050 };
00051
00052 int fork1 (void); // Fork but panics on failure.
00053 void panic(char*);
00054 struct cmd *parsecmd(char*);
00055
00056 // Execute cmd. Never returns.
00057 void
00058 runcmd(struct cmd *cmd)
00059 {
00060
        int p[2];
00061
        struct backemd *bemd;
00062
        struct execomd *ecmd;
00063
        struct listemd *lemd;
00064
        struct pipecmd *pcmd;
00065
        struct redircmd *rcmd;
00066
00067
        if(cmd == 0)
00068
          exit();
00069
00070
        switch(cmd->type) {
00071
        default:
00072
          panic("runcmd");
00073
00074
        case EXEC:
         ecmd = (struct execcmd*)cmd;
00075
00076
          if(ecmd->argv[0] == 0)
00077
            exit();
          exec(ecmd->argv[0], ecmd->argv);
printf(2, "exec %s failed\n", ecmd->argv[0]);
00078
00079
08000
00081
00082
        case REDIR:
```

```
rcmd = (struct redircmd*)cmd;
00084
           close(rcmd->fd);
00085
           if(open(rcmd->file, rcmd->mode) < 0){</pre>
             printf(2, "open %s failed\n", rcmd->file);
00086
00087
              exit();
00088
00089
           runcmd(rcmd->cmd);
00090
00091
00092
         case LIST:
          lcmd = (struct listcmd*)cmd;
if(fork1() == 0)
00093
00094
00095
             runcmd(lcmd->left);
00096
00097
           runcmd(lcmd->right);
00098
           break;
00099
00100
        case PIPE:
         pcmd = (struct pipecmd*)cmd;
00101
00102
           if (pipe (p) < 0)</pre>
00103
             panic("pipe");
00104
           if(fork1() == 0){
00105
             close(1);
00106
             dup(p[1]);
00107
             close(p[0]);
00108
              close(p[1]);
00109
              runcmd(pcmd->left);
00110
           if(fork1() == 0){
00111
00112
             close(0);
             dup(p[0]);
00113
00114
             close(p[0]);
00115
             close(p[1]);
00116
             runcmd(pcmd->right);
00117
00118
           close(p[0]);
00119
           close(p[1]);
           wait();
00121
           wait();
00122
           break;
00123
00124
         case BACK:
00125
         bcmd = (struct backcmd*)cmd;
           if(fork1() == 0)
00126
00127
             runcmd(bcmd->cmd);
00128
           break;
00129
00130
        exit();
00131 }
00132
00133 int
00134 getcmd(char *buf, int nbuf)
00135 {
00136 printf(2, "$ ");

00137 memset(buf, 0, nbuf);

00138 gets(buf, nbuf);

00139 if(buf[0] == 0) // EOF
00140
          return -1;
00141
        return 0;
00142 }
00143
00144 int
00145 main(void)
00146 {
00147
         static char buf[100];
00148
        int fd;
00149
00150
         // Ensure that three file descriptors are open.
         while((fd = open("console", O_RDWR)) >= 0){
00151
00152
          if(fd >= 3){
00153
              close(fd);
00154
             break;
00155
          }
00156
00157
00158
         // Read and run input commands.
         while(getcmd(buf, sizeof(buf)) >= 0){
  if(buf[0] == 'c' && buf[1] == 'd' && buf[2] == ' '){
00159
00160
             // Chdir must be called by the parent, not the child. buf[strlen(buf)-1] = 0; // chop \n
00161
00162
             if(chdir(buf+3) < 0)
printf(2, "cannot cd %s\n", buf+3);</pre>
00163
00164
00165
              continue;
00166
00167
           if(fork1() == 0)
             runcmd(parsecmd(buf));
00168
           wait();
00169
```

5.173 sh.c 363

```
00171 exit();
00172 }
00173
00174 void
00175 panic(char *s)
00176 {
00177 printf(2, "%s\n", s);
00178 exit();
00179 }
00180
00181 int
00182 fork1(void)
00183 {
00184
        int pid;
00185
        pid = fork();
00186
        if (pid == -1)
00187
        panic("fork");
00188
00189 return pid;
00190 }
00191
00192 //PAGEBREAK!
00193 // Constructors
00194
00195 struct cmd*
00196 execcmd(void)
00197 {
00198
        struct execomd *cmd;
00199
00200
        cmd = malloc(sizeof(*cmd));
       memset(cmd, 0, sizeof(*cmd));
cmd->type = EXEC;
00201
00202
00203 return (struct cmd*)cmd;
00204 }
00205
00206 struct cmd*
00207 redircmd(struct cmd *subcmd, char *file, char *efile, int mode, int fd)
00208 {
00209
        struct redircmd *cmd;
00210
00211
        cmd = malloc(sizeof(*cmd));
        memset(cmd, 0, sizeof(*cmd));
cmd->type = REDIR;
00212
00213
00214
        cmd->cmd = subcmd;
00215
        cmd->file = file;
00216
        cmd->efile = efile:
        cmd->mode = mode;
cmd->fd = fd;
00217
00218
00219
        return (struct cmd*)cmd;
00220 }
00221
00222 struct cmd*
00223 pipecmd(struct cmd *left, struct cmd *right) 00224 {
00225
        struct pipecmd *cmd;
00227
        cmd = malloc(sizeof(*cmd));
        memset(cmd, 0, sizeof(*cmd));
cmd->type = PIPE;
cmd->left = left;
00228
00229
00230
        cmd->right = right;
00231
00232
        return (struct cmd*)cmd;
00233 }
00234
00235 struct cmd*
00236 listcmd(struct cmd *left, struct cmd *right)
00237 {
00238
        struct listemd *emd;
00239
00240
        cmd = malloc(sizeof(*cmd));
00241
        memset(cmd, 0, sizeof(*cmd));
        cmd->type = LIST;
cmd->left = left;
00242
00243
        cmd->right = right;
00244
00245
        return (struct cmd*)cmd;
00246 }
00247
00248 struct cmd*
00249 backcmd(struct cmd *subcmd)
00250 {
00251
        struct backemd *emd;
00252
00253
        cmd = malloc(sizeof(*cmd));
        memset(cmd, 0, sizeof(*cmd));
cmd->type = BACK;
cmd->cmd = subcmd;
00254
00255
00256
```

```
return (struct cmd*)cmd;
00258 }
00259 //PAGEBREAK!
00260 // Parsing
00261
00262 char whitespace[] = " \t\r\n\v";
00263 char symbols[] = "<|>&;()";
00264
00265 int
00266 gettoken(char **ps, char *es, char **q, char **eq)
00267 {
00268
        char *s;
00269
        int ret;
00270
00271
        s = *ps;
        while(s < es && strchr(whitespace, *s))</pre>
00272
00273
          s++;
00274
        if(q)
        *q = s;
ret = *s;
00276
00277
        switch(*s){
00278
        case 0:
00279
        break; case '|':
00280
00281
        case '(':
00282
        case ')':
00283
        case '&':
00284
        case '<':
00285
        s++;
break;
00286
00287
        case '>':
00288
         s++;
00289
          if(*s == '>') {
  ret = '+';
00290
00291
00292
             s++;
00293
00294
           break;
00295
        default:
         ret = 'a';
00296
00297
           while(s < es && !strchr(whitespace, *s) && !strchr(symbols, *s))</pre>
            s++;
00298
00299
          break:
00300
00301
        if (eq)
00302
          *eq = s;
00303
        while(s < es && strchr(whitespace, *s))</pre>
00304
00305
          s++;
00306
        *ps = s;
00307
        return ret;
00308 }
00309
00310 int.
00311 peek(char **ps, char *es, char *toks)
00312 {
00313
        char *s;
00314
00315
00316
        while(s < es && strchr(whitespace, *s))</pre>
00317
         s++;
00318
        *ps = s;
00319
        return *s && strchr(toks, *s);
00320 }
00321
00322 struct cmd *parseline(char**, char*);
00323 struct cmd *parsepipe(char**, char*);
00324 struct cmd *parseexec(char**, char*);
00325 struct cmd *nulterminate(struct cmd*);
00326
00327 struct cmd*
00328 parsecmd(char *s)
00329 {
00330
        char *es;
00331
        struct cmd *cmd;
00332
00333
        es = s + strlen(s);
        cmd = parseline(&s, es);
peek(&s, es, "");
00334
00335
00336
        if(s != es) {
  printf(2, "leftovers: %s\n", s);
00337
00338
          panic("syntax");
00339
00340
        nulterminate(cmd);
00341
        return cmd;
00342 }
00343
```

5.173 sh.c 365

```
00344 struct cmd*
00345 parseline(char **ps, char *es)
00346 {
00347
         struct cmd *cmd;
00348
        cmd = parsepipe(ps, es);
while(peek(ps, es, "&")){
00349
00351
          gettoken(ps, es, 0, 0);
00352
           cmd = backcmd(cmd);
00353
         if(peek(ps, es, ";")){
  gettoken(ps, es, 0, 0);
  cmd = listcmd(cmd, parseline(ps, es));
00354
00355
00356
00357
00358
         return cmd;
00359 }
00360
00361 struct cmd*
00362 parsepipe(char **ps, char *es)
00363 {
00364
         struct cmd *cmd;
00365
        cmd = parseexec(ps, es);
if(peek(ps, es, "|")){
  gettoken(ps, es, 0, 0);
  cmd = pipecmd(cmd, parsepipe(ps, es));
00366
00367
00368
00369
00370
00371
         return cmd;
00372 }
00373
00374 struct cmd*
00375 parseredirs(struct cmd *cmd, char **ps, char *es)
00376 {
00377
         int tok;
00378
        char *q, *eq;
00379
        while (peek (ps, es, "<>")) {
  tok = gettoken (ps, es, 0, 0);
00380
00382
           if(gettoken(ps, es, &q, &eq) != 'a')
00383
             panic("missing file for redirection");
00384
           switch(tok) {
case '<':</pre>
00385
           cmd = redircmd(cmd, q, eq, O_RDONLY, 0);
00386
00387
             break;
           case '>':
00388
00389
            cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
           break;
case '+': // »
00390
00391
             cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
00392
00393
             break:
00394
           }
00395
00396
         return cmd;
00397 }
00398
00399 struct cmd*
00400 parseblock(char **ps, char *es)
00401 {
00402
        struct cmd *cmd;
00403
        if(!peek(ps, es, "("))
00404
00405
          panic("parseblock");
        gettoken(ps, es, 0, 0);
cmd = parseline(ps, es);
00406
00407
         if(!peek(ps, es, ")"))
panic("syntax - missing)");
00408
00409
00410
         gettoken(ps, es, 0, 0);
00411
         cmd = parseredirs(cmd, ps, es);
00412
         return cmd:
00413 }
00414
00415 struct cmd*
00416 parseexec(char **ps, char *es)
00417 {
00418
         char *q, *eq;
00419
        int tok, argc;
00420
         struct execomd *cmd;
00421
        struct cmd *ret;
00422
         if(peek(ps, es, "("))
00423
          return parseblock (ps, es);
00424
00425
00426
        ret = execcmd();
00427
         cmd = (struct execcmd*)ret;
00428
         argc = 0;
00429
00430
        ret = parseredirs(ret, ps, es);
```

```
while(!peek(ps, es, "|)&;")){
         if((tok=gettoken(ps, es, &q, &eq)) == 0)
00433
            break;
          if(tok != 'a')
00434
            panic("syntax");
00435
00436
          cmd->argv[argc] = q;
00437
          cmd->eargv[argc] = eq;
00438
          argc++;
00439
          if(argc >= MAXARGS)
            panic("too many args");
00440
          ret = parseredirs(ret, ps, es);
00441
00442 }
00443 cmd->argv[argc] = 0;

00444 cmd->eargv[argc] = 0;

00445 return ret;
00446 }
00447
00448 // NUL-terminate all the counted strings.
00449 struct cmd*
00450 nulterminate(struct cmd *cmd)
00451 {
00452
        int i;
       struct backemd *bemd;
00453
00454
       struct execcmd *ecmd;
struct listcmd *lcmd;
00455
       struct pipecmd *pcmd;
00457
        struct redircmd *rcmd;
00458
        if(cmd == 0)
00459
         return 0;
00460
00461
00462
        switch(cmd->type) {
00463 case EXEC:
        ecmd = (struct execcmd*)cmd;
for(i=0; ecmd->argv[i]; i++)
00464
00465
            *ecmd->eargv[i] = 0;
00466
00467
         break;
00468
00469
       case REDIR:
        rcmd = (struct redircmd*)cmd;
nulterminate(rcmd->cmd);
00470
00471
00472
          *rcmd->efile = 0;
         break;
00473
00474
00475
        case PIPE:
        pcmd = (struct pipecmd*)cmd;
00476
00477
          nulterminate(pcmd->left);
         nulterminate(pcmd->right);
break;
00478
00479
00480
00481
       case LIST:
        lcmd = (struct listcmd*)cmd;
nulterminate(lcmd->left);
00482
00483
         nulterminate(lcmd->right);
break;
00484
00485
00486
        case BACK:
        bcmd = (struct backcmd*)cmd;
00488
00489
          nulterminate(bcmd->cmd);
00490
          break;
        }
00491
00492
        return cmd;
00493 }
```

5.174 sh.d File Reference

5.175 sh.d

Go to the documentation of this file.

00001 sh.o: sh.c /usr/include/stdc-predef.h types.h user.h fcntl.h $\,$

5.176 shutdown.c File Reference

```
#include "types.h"
#include "stat.h"
```

5.177 shutdown.c 367

```
#include "user.h"
```

Functions

```
• int main (int argc, char *argv[])
```

5.176.1 Function Documentation

5.176.1.1 main()

```
int main (
          int argc,
          char * argv[] )
```

Definition at line 10 of file shutdown.c.

5.177 shutdown.c

Go to the documentation of this file.

```
00001 // Shuts down the system by using the halt() system call
00002 // to send a special signal to QEMU.
00003 // Added by Bill Katsak
00004 // Copied from: http://pdos.csail.mit.edu/6.828/2012/homework/xv6-syscall.html
00005
00006 #include "types.h"
00007 #include "stat.h"
00008 #include "user.h"
00009
00010 int main(int argc, char *argv[])
00011 {
00012  halt();
00013  exit();
00014 }
```

5.178 shutdown.d File Reference

5.179 shutdown.d

Go to the documentation of this file.

```
00001 shutdown.o: shutdown.c /usr/include/stdc-predef.h types.h stat.h user.h
```

5.180 sleeplock.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "x86.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "sleeplock.h"
```

Functions

- void acquiresleep (struct sleeplock *lk)
- int holdingsleep (struct sleeplock *lk)
- void initsleeplock (struct sleeplock *lk, char *name)
- void releasesleep (struct sleeplock *lk)

5.180.1 Function Documentation

5.180.1.1 acquiresleep()

Referenced by bget(), ilock(), and iput().

5.180.1.2 holdingsleep()

Referenced by brelse(), bwrite(), iderw(), and iunlock().

5.181 sleeplock.c 369

5.180.1.3 initsleeplock()

Referenced by binit(), and iinit().

5.180.1.4 releasesleep()

Referenced by brelse(), iput(), and iunlock().

5.181 sleeplock.c

00042 }

Go to the documentation of this file.

```
00001 // Sleeping locks
00002
00003 #include "types.h"
00004 #include "defs.h"
00005 #include "param.h"
00006 #include "x86.h"
00007 #include "memlayout.h"
00008 #include "mmu.h"
00009 #include "proc.h"
00010 #include "spinlock.h"
00011 #include "sleeplock.h"
00012
00013 void
00014 initsleeplock(struct sleeplock *lk, char *name)
00015 {
00016 initlock(&lk->lk, "sleep lock");

00017 lk->name = name;

00018 lk->locked = 0;

00019 lk->pid = 0;
00020 }
00021
00022 void
00023 acquiresleep(struct sleeplock *lk)
00024 {
00025
          acquire(&lk->lk);
00026 while (1k->locked)
00027
            sleep(lk, &lk->lk);
00028
         1k->locked = 1;
1k->pid = myproc()->pid;
00029
00030
00031
         release(&lk->lk);
00032 }
```

```
00033
00034 void
00035 releasesleep(struct sleeplock *lk)
00036 {
      acquire(&lk->lk);
lk->locked = 0;
lk->pid = 0;
00037
00038
00040 wakeup(lk);
00041 release(&lk->lk);
00042 }
00043
00044 int
00045 holdingsleep(struct sleeplock *lk)
00046 {
00047
00048
        acquire(&lk->lk);
00049
00050
        r = lk->locked && (lk->pid == myproc()->pid);
        release(&lk->lk);
vuu52 return r;
00053 }
00057
00054
00055
00056
```

5.182 sleeplock.d File Reference

5.183 sleeplock.d

Go to the documentation of this file.

```
00001 sleeplock.o: sleeplock.c /usr/include/stdc-predef.h types.h defs.h \ 00002 param.h x86.h memlayout.h mmu.h proc.h spinlock.h sleeplock.h
```

5.184 sleeplock.h File Reference

Classes

· struct sleeplock

5.185 sleeplock.h

Go to the documentation of this file.

5.186 spinlock.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "x86.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
```

Functions

- void acquire (struct spinlock *lk)
- void getcallerpcs (void *v, uint pcs[])
- int holding (struct spinlock *lock)
- void initlock (struct spinlock *lk, char *name)
- void popcli (void)
- void pushcli (void)
- void release (struct spinlock *lk)

5.186.1 Function Documentation

5.186.1.1 acquire()

```
void acquire (
                struct spinlock * lk )
Definition at line 25 of file spinlock.c.
00027
        pushcli(); // disable interrupts to avoid deadlock.
00028
         if(holding(lk))
          panic("acquire");
00029
00030
00031
        // The xchq is atomic.
00032
        while(xchg(&lk->locked, 1) != 0)
00033
00034
        \ensuremath{//} Tell the C compiler and the processor to not move loads or stores
00035
        // past this point, to ensure that the critical section's memory
// references happen after the lock is acquired.
00036
00037
00038
        __sync_synchronize();
00039
00040
         // Record info about lock acquisition for debugging.
00041
        lk->cpu = mycpu();
        getcallerpcs(&lk, lk->pcs);
00042
00043 }
```

Referenced by acquiresleep(), allocproc(), begin_op(), bget(), brelse(), chpr(), consoleintr(), consoleread(), consolewrite(), cprintf(), cps(), end_op(), exit(), filealloc(), fileclose(), filedup(), fork(), holdingsleep(), ideintr(), iderw(), idup(), iget(), iput(), kalloc(), kfree(), kill(), log_write(), pipeclose(), piperead(), pipewrite(), releasesleep(), scheduler(), sleep(), sys_sleep(), sys_uptime(), trap(), userinit(), wait(), wakeup(), and yield().

5.186.1.2 getcallerpcs()

```
void getcallerpcs (
               void * v,
               uint pcs[] )
Definition at line 72 of file spinlock.c.
00073 {
00074
        uint *ebp;
00075
        int i;
00076
        ebp = (uint*)v - 2;
00078
        for(i = 0; i < 10; i++) {
00079
          if(ebp == 0 || ebp < (uint*)KERNBASE || ebp == (uint*)Oxffffffff)</pre>
00080
                                // saved %eip
00081
         pcs[i] = ebp[1];
          ebp = (uint*)ebp[0]; // saved %ebp
00082
00083
        for(; i < 10; i++)</pre>
00084
```

Referenced by acquire().

pcs[i] = 0;

00085

00086 }

5.186.1.3 holding()

Referenced by acquire(), release(), and sched().

5.186.1.4 initlock()

Definition at line 13 of file spinlock.c.

Referenced by binit(), consoleinit(), fileinit(), ideinit(), iinit(), initlog(), initsleeplock(), kinit1(), pinit(), pinit(), and tvinit().

5.186.1.5 popcli()

```
void popcli (
     void )
```

Definition at line 117 of file spinlock.c.

```
00118 {
00119     if(readeflags()&FL_IF)
00120         panic("popcli - interruptible");
00121     if(--mycpu()->ncli < 0)
00122         panic("popcli");
00123     if(mycpu()->ncli == 0 && mycpu()->intena)
00124         sti();
00125 }
```

Referenced by holding(), myproc(), release(), and switchuvm().

5.187 spinlock.c 373

5.186.1.6 pushcli()

```
void pushcli (
     void )
```

Definition at line 105 of file spinlock.c.

Referenced by acquire(), holding(), myproc(), and switchuvm().

5.186.1.7 release()

Definition at line 47 of file spinlock.c.

```
00048 {
00049
          if(!holding(lk))
            panic("release");
00050
00051
00052
          1k - pcs[0] = 0;
00053
          1k \rightarrow cpu = 0;
00054
00055
         // Tell the C compiler and the processor to not move loads or stores \ensuremath{\text{--}}
00056
         // past this point, to ensure that all the stores in the critical
// section are visible to other cores before the lock is released.
00057
00058
          \ensuremath{//} Both the C compiler and the hardware may re-order loads and
00059
         // stores; __sync_synchronize() tells them both not to.
00060
          __sync_synchronize();
00061
00062
          // Release the lock, equivalent to lk \rightarrow locked = 0.
00063
         // This code can't use a C assignment, since it might
// not be atomic. A real OS would use C atomics here.
00065
          asm volatile("mov1 $0, $0" : "+m" (lk->locked) : );
00066
          popcli();
00067
00068 }
```

Referenced by acquiresleep(), allocproc(), begin_op(), bget(), brelse(), chpr(), consoleintr(), consoleread(), consolewrite(), cprintf(), cps(), end_op(), filealloc(), fileclose(), filedup(), fork(), forkret(), holdingsleep(), ideintr(), iderw(), idup(), iget(), iput(), kalloc(), kfree(), kill(), log_write(), pipeclose(), piperead(), pipewrite(), releasesleep(), scheduler(), sleep(), sys_sleep(), sys_uptime(), trap(), userinit(), wait(), wakeup(), and yield().

5.187 spinlock.c

Go to the documentation of this file.

```
00001 // Mutual exclusion spin locks.
00002
00003 #include "types.h"
00004 #include "defs.h"
00005 #include "param.h"
00006 #include "x86.h"
00007 #include "memlayout.h"
00008 #include "mmu.h"
00009 #include "proc.h"
00010 #include "spinlock.h"
00011
00012 void
```

```
00013 initlock(struct spinlock *lk, char *name)
00014 {
00015
        1k->name = name;
        1k \rightarrow 1ocked = 0;
00016
00017
        1k \rightarrow cpu = 0;
00018 }
00020 // Acquire the lock.
00021 // Loops (spins) until the lock is acquired.
00022 // Holding a lock for a long time may cause
00023 // other CPUs to waste time spinning to acquire it.
00024 void
00025 acquire(struct spinlock *lk)
00026 {
00027
        pushcli(); // disable interrupts to avoid deadlock.
        if (holding(lk))
  panic("acquire");
00028
00029
00030
00031
        // The xchg is atomic.
00032
        while(xchg(&lk->locked, 1) != 0)
00033
00034
        // Tell the C compiler and the processor to not move loads or stores \,
00035
00036
        // past this point, to ensure that the critical section's memory
00037
        // references happen after the lock is acquired.
00038
        __sync_synchronize();
00039
00040
         // Record info about lock acquisition for debugging.
00041
        lk->cpu = mycpu();
        getcallerpcs(&lk, lk->pcs);
00042
00043 }
00044
00045 // Release the lock.
00046 void
00047 release(struct spinlock *lk)
00048 {
00049
        if(!holding(lk))
          panic("release");
00051
00052
        1k -> pcs[0] = 0;
00053
        1k \rightarrow cpu = 0;
00054
00055
        // Tell the C compiler and the processor to not move loads or stores
        // past this point, to ensure that all the stores in the critical
00056
        // section are visible to other cores before the lock is released.
00057
00058
        \ensuremath{//} Both the C compiler and the hardware may re-order loads and
00059
        // stores; _
                      _sync_synchronize() tells them both not to.
00060
        __sync_synchronize();
00061
00062
        // Release the lock, equivalent to lk\rightarrow locked = 0.
00063
        // This code can't use a C assignment, since it might
        // not be atomic. A real OS would use C atomics here.
asm volatile("mov1 $0, %0" : "+m" (lk->locked) : );
00064
00065
00066
        popcli();
00067
00068 }
00070 // Record the current call stack in pcs[] by following the %ebp chain.
00071 void
00072 getcallerpcs(void *v, uint pcs[])
00073 {
00074
        uint *ebp;
00075
        int i;
00076
00077
        ebp = (uint*)v - 2;
00078
        for(i = 0; i < 10; i++) {</pre>
         if(ebp == 0 || ebp < (uint*)KERNBASE || ebp == (uint*)Oxffffffff)</pre>
00079
08000
            break:
00081
          pcs[i] = ebp[1];
                                  // saved %eip
          ebp = (uint*)ebp[0]; // saved %ebp
00082
00083
00084
        for(; i < 10; i++)</pre>
00085
          pcs[i] = 0;
00086 }
00087
00088 // Check whether this cpu is holding the lock.
00089 int
00090 holding(struct spinlock *lock)
00091 {
00092
        int r:
00093
        pushcli();
00094
        r = lock->locked && lock->cpu == mycpu();
        popcli();
00095
00096
         return r;
00097 }
00098
00099
```

```
00100 // Pushcli/popcli are like cli/sti except that they are matched:
00101 // it takes two popcli to undo two pushcli. Also, if interrupts
00102 // are off, then pushcli, popcli leaves them off.
00103
00104 void
00105 pushcli(void)
00106 {
00107
        int eflags;
00108
00109
       eflags = readeflags();
00110
        cli();
       if (mycpu()->ncli == 0)
  mycpu()->intena = eflags & FL_IF;
00111
00112
00113 mycpu()->ncli += 1;
00114 }
00115
00116 void
00117 popcli(void)
00118 {
00119
       if (readeflags() &FL_IF)
00120
          panic("popcli - interruptible");
       if(--mycpu()->ncli < 0)
panic("popcli");</pre>
00121
00122
       if(mycpu()->ncli == 0 && mycpu()->intena)
sti();
00123
00124
00125 }
00126
```

5.188 spinlock.d File Reference

5.189 spinlock.d

```
Go to the documentation of this file.
```

```
00001 spinlock.o: spinlock.c /usr/include/stdc-predef.h types.h defs.h param.h \setminus 00002 x86.h memlayout.h mmu.h proc.h spinlock.h
```

5.190 spinlock.h File Reference

Classes

struct spinlock

5.191 spinlock.h

Go to the documentation of this file.

```
00001 // Mutual exclusion lock.
00002 struct spinlock {
00003
      uint locked;
                            // Is the lock held?
00004
        // For debugging:
00005
00006
       char *name;
                            // Name of lock.
                           // The cpu holding the lock.
// The call stack (an array of program counters)
        struct cpu *cpu;
00007
80000
        uint pcs[10];
00009
                            // that locked the lock.
00010 };
00011
```

5.192 stat.h File Reference

Classes

struct stat

Macros

```
• #define T_DEV 3
```

- #define T_DIR 1
- #define T FILE 2

5.192.1 Macro Definition Documentation

5.192.1.1 T_DEV

```
#define T_DEV 3
```

Definition at line 3 of file stat.h.

5.192.1.2 T_DIR

```
#define T_DIR 1
```

Definition at line 1 of file stat.h.

5.192.1.3 T_FILE

```
#define T_FILE 2
```

Definition at line 2 of file stat.h.

5.193 stat.h

Go to the documentation of this file.

```
00001 #define T_DIR 1 // Directory
00002 #define T_FILE 2 // File
00003 #define T_DEV 3 // Device
00004
00005 struct stat {
00006 short type; // Type of file
00007 int dev; // File system's disk device
00008 uint ino; // Inode number
00009 short nlink; // Number of links to file
00010 uint size; // Size of file in bytes
00011 };
```

5.194 stressfs.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fs.h"
#include "fcntl.h"
```

Functions

• int main (int argc, char *argv[])

5.194.1 Function Documentation

5.194.1.1 main()

```
int main (
          int argc,
          char * argv[] )
```

Definition at line 17 of file stressfs.c.

```
00019
           int fd, i;
           char path[] = "stressfs0";
00020
00021
          char data[512];
00022
          printf(1, "stressfs starting\n");
memset(data, 'a', sizeof(data));
00023
00024
00025
          for(i = 0; i < 4; i++)
  if(fork() > 0)
  break:
00026
00027
00028
              break;
00029
00030
          printf(1, "write %d\n", i);
00031
          path[8] += i;
00032
00033 fd = open(path, O_CREATE | O_RDWR);

00034 for(i = 0; i < 20; i++)

00035 // printf(fd, "%d\n", i);

00036 write(fd, data, sizeof(data));
00037
          close(fd);
00038
          printf(1, "read\n");
00039
00040
          fd = open(path, O_RDONLY);
for (i = 0; i < 20; i++)
  read(fd, data, sizeof(data));</pre>
00041
00042
00043
00044
          close(fd);
00045
00046
           wait();
00047
00048
          exit();
00049 }
```

5.195 stressfs.c

```
Go to the documentation of this file.
```

```
00001 // Demonstrate that moving the "acquire" in iderw after the loop that 00002 // appends to the idequeue results in a race.
00004 // For this to work, you should also add a spin within iderw's
00005 // idequeue traversal loop. Adding the following demonstrated a panic
00006 // after about 5 runs of stressfs in QEMU on a 2.1GHz CPU: 00007 // for (i = 0; i < 40000; i++)
00008 //
                  asm volatile("");
00009
00010 #include "types.h"
00010 #include "stat.h"
00011 #include "stat.h"
00012 #include "user.h"
00013 #include "fs.h"
00014 #include "fcntl.h"
00015
00016 int
00017 main(int argc, char *argv[])
00018 {
00019 int fd, i;
00020 char path[] = "stressfs0";
00021
         char data[512];
00022
         printf(1, "stressfs starting\n");
memset(data, 'a', sizeof(data));
00023
00024
00025
00026
          for(i = 0; i < 4; i++)
          if(fork() > 0)
00027
00028
               break:
00029
00030
         printf(1, "write %d\n", i);
00031
         path[8] += i;
00032
00032 path[0] 7-1,

00033 fd = open(path, O_CREATE | O_RDWR);

00034 for(i = 0; i < 20; i++)

00035 // printf(fd, "%d\n", i);
00035 // printf(fd, "%d\n", 1,,
00036 write(fd, data, sizeof(data));
00037
         close(fd);
00038
         printf(1, "read\n");
00039
00040
00041
          fd = open(path, O_RDONLY);
00042
          for (i = 0; i < 20; i++)
00043
             read(fd, data, sizeof(data));
00044
         close(fd);
00045
00046
          wait():
00047
00048
          exit();
00049 }
```

5.196 stressfs.d File Reference

5.197 stressfs.d

Go to the documentation of this file.

```
00001 stressfs.o: stressfs.c /usr/include/stdc-predef.h types.h stat.h user.h \backslash 00002 \, fs.h fcntl.h
```

5.198 string.c File Reference

```
#include "types.h"
#include "x86.h"
```

Functions

```
int memcmp (const void *v1, const void *v2, uint n)
void * memcpy (void *dst, const void *src, uint n)
void * memmove (void *dst, const void *src, uint n)
void * memset (void *dst, int c, uint n)
char * safestrcpy (char *s, const char *t, int n)
int strlen (const char *s)
int strncmp (const char *p, const char *q, uint n)
char * strncpy (char *s, const char *t, int n)
```

5.198.1 Function Documentation

5.198.1.1 memcmp()

```
int memcmp (  \mbox{const void} * v1, \\ \mbox{const void} * v2, \\ \mbox{uint } n \; )
```

Definition at line 16 of file string.c.

```
00017 {
00018
        const uchar *s1, *s2;
00019
00020
        s1 = v1;
00021
       s2 = v2;
00022
       while (n-- > 0) {
        if(*s1 != *s2)
return *s1 - *s2;
00023
00024
00025
         s1++, s2++;
00026 }
00027
00028
       return 0;
00029 }
```

Referenced by cmostime(), mpconfig(), and mpsearch1().

5.198.1.2 memcpy()

Definition at line 53 of file string.c.

```
00054 {
00055     return memmove(dst, src, n);
00056 }
```

5.198.1.3 memmove()

```
00034
        const char *s;
00035
       char *d;
00036
00037
       s = src;
       d = dst;
00038
       if(s < d && s + n > d) {
        s += n;
00040
         d += n;
00041
        while (n-- > 0)
00042
00043
           *--d = *--s;
00044
       } else
00045
        while (n-- > 0)
00046
           *d++ = *s++;
00047
00048
       return dst;
00049 }
```

Referenced by cgaputc(), copyout(), copyouvm(), fmtname(), grep(), iderw(), ilock(), inituvm(), install_trans(), iupdate(), ls(), main(), memcpy(), readi(), readsb(), skipelem(), startothers(), write_log(), and writei().

5.198.1.4 memset()

```
void * memset (  \mbox{void} * \mbox{$dst$,} \\ \mbox{int $c$,} \\ \mbox{uint $n$} )
```

Definition at line 5 of file string.c.

5.198.1.5 safestrcpy()

```
\label{eq:char} \begin{array}{c} \text{char * safestrcpy (} \\ \text{char * $s$,} \\ \text{const char * $t$,} \\ \text{int $n$ )} \end{array}
```

Definition at line 83 of file string.c.

```
00084 {
00085
        char *os;
00086
00087
        os = s;
00088
        <u>if</u> (n <= 0)
00089
          return os;
00090
        while (--n > 0 \&\& (*s++ = *t++) != 0)
00091
        *s = 0;
00092
00093
        return os;
00094 }
```

Referenced by exec(), fork(), and userinit().

5.198.1.6 strlen()

5.198.1.7 strncmp()

```
int strncmp (  \mbox{const char} \ * \ p , \\ \mbox{const char} \ * \ q , \\ \mbox{uint } n \ )
```

Definition at line 59 of file string.c.

Referenced by namecmp().

5.198.1.8 strncpy()

```
\begin{array}{c} \text{char} * \text{strncpy (} \\ \\ \text{char} * s, \\ \\ \text{const char} * t, \\ \\ \text{int } n \text{ )} \end{array}
```

Definition at line 69 of file string.c.

Referenced by dirlink(), main(), and sys_getprocs().

5.199 string.c

Go to the documentation of this file.

```
00001 #include "types.h 00002 #include "x86.h"
00003
00004 void*
00005 memset(void *dst, int c, uint n)
00006 {
       if ((int)dst%4 == 0 && n%4 == 0){
00007
        c &= 0xFF;
80000
00009
          stosl(dst, (c«24)|(c«16)|(c«8)|c, n/4);
00010 } else
00011
         stosb(dst, c, n);
00012
       return dst;
00013 }
00014
00015 int
00016 memcmp(const void *v1, const void *v2, uint n)
00017 {
00018
        const uchar *s1, *s2;
00019
        s1 = v1;
00020
        s2 = v2;
00021
        while (n-- > 0) {
00022
        if(*s1 != *s2)
00024
            return *s1 - *s2;
00025
         s1++, s2++;
00026 }
00027
00028
        return 0;
00029 }
00030
00031 void*
00032 memmove(void *dst, const void *src, uint n)
00033 {
00034 const char *s;
00035
        char *d;
00036
00037
        s = src;
        d = dst;
00038
        if (s < d && s + n > d) {
    s += n;
    d += n;
00039
00040
         while (n-- > 0)
00042
00043
            *--d = *--s;
00044
        } else
        while (n-- > 0)
00045
            *d++ = *s++;
00046
00047
00048
       return dst;
00049 }
00050
00051 // memcpy exists to placate GCC. Use memmove.
00052 void*
00053 memcpy(void *dst, const void *src, uint n)
00054 {
00055
        return memmove(dst, src, n);
00056 }
00057
00058 int
00059 strncmp(const char *p, const char *q, uint n)
00061 while(n > 0 && *p && *p == *q)
00062 n--, p++, q++;
       n--, p++, q++;
if (n == 0)
00063
         return 0:
00064
        return (uchar) *p - (uchar) *q;
00065
00066 }
00067
00068 char*
00069 strncpy(char *s, const char *t, int n)
00070 {
00071
        char *os:
00072
        os = s;
00074
        while (n-- > 0 \&\& (*s++ = *t++) != 0)
00075
        while(n-- > 0)
   *s++ = 0;
00076
00077
00078
        return os;
00079 }
08000
00081 // Like strncpy but guaranteed to NUL-terminate.
00082 char*
```

```
00083 safestrcpy(char *s, const char *t, int n)
00085
        char *os;
00086
        os = s;
00087
       if (n <= 0)
00088
         return os;
00090
       while (--n > 0 && (*s++ = *t++) != 0)
       ;
*s = 0;
00091
00092
00093
       return os;
00094 }
00095
00096 int
00097 strlen(const char *s)
00098 {
00099
       int n:
00100
       for(n = 0; s[n]; n++)
00102
00103
       return n;
00104 }
00105
```

5.200 string.d File Reference

5.201 string.d

```
Go to the documentation of this file.
```

00001 string.o: string.c /usr/include/stdc-predef.h types.h x86.h

5.202 syscall.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
#include "syscall.h"
```

Functions

- int argint (int n, int *ip)
- int argptr (int n, char **pp, int size)
- int argstr (int n, char **pp)
- int fetchint (uint addr, int *ip)
- int fetchstr (uint addr, char **pp)
- int sys_chdir (void)
- int sys_chpr (void)
- int sys_close (void)
- int sys_cps (void)
- int sys_dup (void)
- int sys_exec (void)
- int sys_exit (void)

- int sys_fork (void)
- int sys_fstat (void)
- int sys_getpid (void)
- int sys_getprocs (void)
- int sys_halt (void)
- int sys_kill (void)
- int sys_link (void)
- int sys_mkdir (void)
- int sys_mknod (void)
- int sys_open (void)
- int sys_pipe (void)
- int sys_read (void)
- int sys_sbrk (void)
- int sys_sleep (void)
- int sys_unlink (void)
- int sys_uptime (void)
- int sys_wait (void)
- int sys_write (void)
- void syscall (void)

Variables

• static int(* syscalls [])(void)

5.202.1 Function Documentation

5.202.1.1 argint()

Definition at line 50 of file syscall.c.

```
00051 {
00052    return fetchint((myproc()->tf->esp) + 4 + 4*n, ip);
00053 }
```

Referenced by argfd(), argstr(), argstr(), sys_chpr(), sys_exec(), sys_getprocs(), sys_kill(), sys_mknod(), sys_open(), sys_read(), sys_sbrk(), sys_sleep(), and sys_write().

5.202.1.2 argptr()

```
int argptr (
    int n,
    char ** pp,
    int size )
```

Definition at line 59 of file syscall.c.

```
00060 {
00061
00062
        struct proc *curproc = myproc();
00063
00064
       if(argint(n, \&i) < 0)
       return -1;
if(size < 0 || (uint)i >= curproc->sz || (uint)i+size > curproc->sz)
00066
00067
          return -1;
00068
       *pp = (char*)i;
00069
        return 0;
00070 }
```

Referenced by sys_fstat(), sys_getprocs(), sys_pipe(), sys_read(), and sys_write().

5.202.1.3 argstr()

Definition at line 77 of file syscall.c.

```
00078 {
00079    int addr;
00080    if(argint(n, &addr) < 0)
00081        return -1;
00082    return fetchstr(addr, pp);
00083 }</pre>
```

Referenced by sys_chdir(), sys_exec(), sys_link(), sys_mkdir(), sys_mknod(), sys_open(), and sys_unlink().

5.202.1.4 fetchint()

Definition at line 18 of file syscall.c.

```
00019 {
00020     struct proc *curproc = myproc();
00021
00022     if(addr >= curproc->sz || addr+4 > curproc->sz)
00023         return -1;
00024     *ip = *(int*)(addr);
00025     return 0;
00026 }
```

Referenced by argint(), and sys_exec().

5.202.1.5 fetchstr()

```
int fetchstr (
               uint addr,
               char ** pp )
Definition at line 32 of file syscall.c.
00033 {
        char *s, *ep;
00035
        struct proc *curproc = myproc();
00036
        if(addr >= curproc->sz)
00038
          return -1;
        *pp = (char*)addr;
ep = (char*)curproc->sz;
00039
```

Referenced by argstr(), and sys_exec().

, -- -= 0)
return s - *pp;
}

return -1;

5.202.1.6 sys_chdir()

00040 00041

00042

00043 00044 00045

00046 }

```
int sys_chdir (
            void )
```

Definition at line 372 of file sysfile.c.

```
00374
         char *path;
00375
         struct inode *ip;
00376
         struct proc *curproc = myproc();
00377
00378
         begin_op();
00379
         if (argstr(0, &path) < 0 || (ip = namei(path)) == 0){</pre>
         end_op();
return -1;
00380
00381
00382
        ilock(ip);
if(ip->type != T_DIR){
  iunlockput(ip);
  end_op();
  return -1;
00383
00384
00385
00386
00387
00388
        iunlock(ip);
iput(curproc->cwd);
00389
00390
00391
         end_op();
         curproc->cwd = ip;
00392
00393
         return 0;
00394 }
```

5.202.1.7 sys_chpr()

```
int sys_chpr (
          void )
```

Definition at line 125 of file sysproc.c.

```
00126 {
00127 int pid, pr;
00128
       if(argint(0, &pid) < 0)</pre>
00129
         return -1;
       if(argint(1, &pr) < 0)
00130
00131
        return -1;
00132
00133
       return chpr(pid, pr);
00134 }
```

5.202.1.8 sys_close()

```
int sys_close (
     void )
```

Definition at line 94 of file sysfile.c.

```
00095 {
00096 int fd;
00097 struct file *f;
00098
00099 if(argfd(0, &fd, &f) < 0)
00100 return -1;
00101 myproc()->ofile[fd] = 0;
00102 fileclose(f);
00103 return 0;
```

5.202.1.9 sys_cps()

```
int sys_cps (
     void )
```

Definition at line 119 of file sysproc.c.

```
00120 {
00121 return cps();
00122 }
```

5.202.1.10 sys_dup()

```
int sys_dup (
     void )
```

Definition at line 56 of file sysfile.c.

5.202.1.11 sys_exec()

```
int sys_exec (
     void )
```

Definition at line 397 of file sysfile.c.

```
00398 {
          char *path, *argv[MAXARG];
          int i;
00401
         uint uargv, uarg;
00402
         return -1;
}
          if (argstr(0, &path) < 0 || argint(1, (int*)&uargv) < 0){</pre>
00403
00404
00405
         memset(argv, 0, sizeof(argv));
for(i=0;; i++) {
   if(i >= NELEM(argv))
00406
00407
00408
            return -1;
if (fetchint (uargv+4*i, (int*) & uarg) < 0)</pre>
00409
00410
            return -1;
if (uarg == 0) {
argv[i] = 0;
00411
00412
00413
00414
              break;
00415
00416
            if(fetchstr(uarg, &argv[i]) < 0)</pre>
00417
              return -1;
00419 return exec(path, argv);
00420 }
00418 }
```

5.202.1.12 sys_exit()

```
int sys_exit (
     void )
```

Definition at line 25 of file sysproc.c.

5.202.1.13 sys_fork()

```
int sys_fork (
     void )
```

Definition at line 19 of file sysproc.c.

```
00020 {
00021 return fork();
00022 }
```

5.202.1.14 sys_fstat()

```
int sys_fstat (
     void )
```

Definition at line 107 of file sysfile.c.

```
00108 {
00109     struct file *f;
00110     struct stat *st;
00111
00112     if(argfd(0, 0, &f) < 0 || argptr(1, (void*)&st, sizeof(*st)) < 0)
00113          return -1;
00114     return filestat(f, st);
00115 }</pre>
```

5.202.1.15 sys_getpid()

```
int sys_getpid (
     void )
```

Definition at line 48 of file sysproc.c.

```
00049 {
00050 return myproc()->pid;
00051 }
```

5.202.1.16 sys_getprocs()

```
int sys_getprocs (
     void )
```

Definition at line 137 of file sysproc.c.

```
00137
00138
         // declare local variables for max uproc size, struct uproc, and counter
        int max;
00140
        struct uproc *p;
00141
        int i=0;
00142
00143
        // if argint has trouble, return error -1
if(argint(0,&max) < 0)</pre>
00144
00145
          return -1;
00146
00147
        // if argptr for allocating struct size has trouble, return -1
00148
        if(argptr(1,(char**)&p, max*sizeof(struct uproc)) < 0)</pre>
00149
          return -1;
00150
00151
        // create pointer to ptable processes
00152
        struct proc *ptr = ptable.proc;
00153
00154
         // loop through ptable
00155
         for(; ptr < &ptable.proc[NPROC]; ptr++)</pre>
00156
00157
           if(!(ptr->state == UNUSED))
00158
00159
             // if the process in ptable is not UNUSED, assign pid, parent pid, and name to uproc
            p[i].pid = ptr->pid;
p[i].ppid = ptr->parent->pid;
00160
00161
             strncpy(p[i].name, ptr->name, 16);
// add 1 to the process counter
00162
00163
00164
             i++;
00165
00166
00167
        \ensuremath{//} return the number of processes that are not UNUSED
00168
00169
        return i;
00170 }
```

5.202.1.17 sys_halt()

```
int sys_halt (
     void )
```

Definition at line 111 of file sysproc.c.

```
00112 {
00113    outb(0xf4, 0x00);
00114    return 0;
00115 }
```

5.202.1.18 sys_kill()

```
int sys_kill (
     void )
```

Definition at line 38 of file sysproc.c.

```
00039 {
00040 int pid;
00041
00042 if(argint(0, &pid) < 0)
00043 return -1;
00044 return kill(pid);
00045 }
```

5.202.1.19 sys_link()

```
int sys_link (
     void )
```

Definition at line 119 of file sysfile.c.

```
00120 {
         char name[DIRSIZ], \starnew, \starold;
00121
00122
        struct inode *dp, *ip;
00123
00124
        if(argstr(0, &old) < 0 || argstr(1, &new) < 0)</pre>
00125
          return -1;
00126
00127
        begin_op();
        if((ip = namei(old)) == 0) {
  end_op();
00128
00129
00130
           return -1:
00131
00132
00133
        ilock(ip);
        if (ip->type == T_DIR) {
  iunlockput(ip);
00134
00135
00136
          end_op();
00137
          return -1;
00138
00139
00140
        ip->nlink++;
00141
        iupdate(ip);
00142
        iunlock(ip);
00143
00144
        if((dp = nameiparent(new, name)) == 0)
        goto bad;
ilock(dp);
00145
        if(dp->dev != ip->dev || dirlink(dp, name, ip->inum) < 0){
  iunlockput(dp);
  gets bad;</pre>
00146
00147
00148
00149
          goto bad;
00150
00151
         iunlockput(dp);
00152
        iput(ip);
00153
00154
        end_op();
00155
00156
        return 0;
00157
00158 bad:
        ilock(ip);
00159
00160
        ip->nlink--;
00161
        iupdate(ip);
00162
        iunlockput(ip);
00163
        end_op();
00164
00165 }
```

5.202.1.20 sys_mkdir()

```
int sys_mkdir (
              void )
Definition at line 336 of file sysfile.c.
00337 {
       char *path;
00339
       struct inode *ip;
00340
00341
       begin_op();
00342
       if(argstr(0, &path) < 0 || (ip = create(path, T_DIR, 0, 0)) == 0){</pre>
        end_op();
00343
00344
         return -1;
00345
00346
       iunlockput(ip);
00347 end_op();
```

5.202.1.21 sys_mknod()

return 0;

```
int sys_mknod (
     void )
```

00348

00349 }

Definition at line 352 of file sysfile.c.

```
00353 {
00354
            struct inode *ip;
00355
            char *path;
00356
            int major, minor;
00357
           begin_op();
if((argstr(0, &path)) < 0 ||
    argint(1, &major) < 0 ||
    argint(2, &minor) < 0 ||
    (ip = create(path, T_DEV, major, minor)) == 0){</pre>
00358
00359
00360
00361
00362
00363
               end_op();
00364
               return -1;
00365 }
00365 iunlockput(ip);
00367 end_op();
00368 return 0;
00369 }
```

5.202.1.22 sys_open()

```
int sys_open (
     void )
```

Definition at line 286 of file sysfile.c.

```
00287 {
00288
        char *path;
        int fd, omode;
struct file *f;
00289
00290
00291
        struct inode *ip;
00292
        if(argstr(0, &path) < 0 || argint(1, &omode) < 0)</pre>
00293
00294
          return -1;
00295
00296
        begin_op();
00297
        if (omode & O_CREATE) {
00298
        ip = create(path, T_FILE, 0, 0);
if(ip == 0){
00299
00300
00301
            end_op();
00302
             return -1;
```

```
00303
00304
        } else {
          if((ip = namei(path)) == 0){
00305
00306
           end_op();
00307
            return -1;
00308
00309
          ilock(ip);
00310
          if(ip->type == T_DIR && omode != O_RDONLY) {
00311
           iunlockput(ip);
00312
             end_op();
00313
            return -1;
00314
00315
        }
00316
00317
        if((f = filealloc()) == 0 \mid \mid (fd = fdalloc(f)) < 0){
00318
            fileclose(f);
00319
00320
          iunlockput(ip);
00321
         end_op();
00322
          return -1;
00323
00324
        iunlock(ip);
00325
        end_op();
00326
00327
        f->type = FD_INODE;
00328 f->ip = ip;
00329 f->off = 0;
        f->readable = !(omode & O_WRONLY);
f->writable = (omode & O_WRONLY) || (omode & O_RDWR);
00330
00331
00332
        return fd;
00333 }
```

5.202.1.23 sys_pipe()

```
int sys_pipe (
     void )
```

Definition at line 423 of file sysfile.c.

```
00424 {
          int *fd;
struct file *rf, *wf;
int fd0, fd1;
00425
00427
00428
00429
          if(argptr(0, (void*)&fd, 2*sizeof(fd[0])) < 0)</pre>
00430
          return -1;
if (pipealloc(&rf, &wf) < 0)</pre>
00431
00432
            return -1;
00433
          fd0 = -1;
         if((fd0 = fdalloc(rf)) < 0 || (fd1 = fdalloc(wf)) < 0){
   if(fd0 >= 0)
       myproc() -> ofile[fd0] = 0;
00434
00435
00436
           fileclose(rf);
00437
           fileclose(wf);
return -1;
00438
00439
00440
         fd[0] = fd0;
fd[1] = fd1;
00441
00442
00443
         return 0;
00444 }
```

5.202.1.24 sys_read()

```
int sys_read (
     void )
```

Definition at line 70 of file sysfile.c.

```
00071 {
00072 struct file *f;
00073 int n;
```

```
00074 char *p;

00075

00076 if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)

00077 return -1;

00078 return fileread(f, p, n);

00079 }
```

5.202.1.25 sys_sbrk()

```
int sys_sbrk (
     void )
```

Definition at line 54 of file sysproc.c.

```
00055 {
00056
        int addr;
00057
        int n;
00058
        if(argint(0, &n) < 0)</pre>
00059
00060
           return -1;
        addr = myproc()->sz;
00061
        if (growproc(n) < 0)
  return -1;</pre>
00062
00063
00064
        return addr;
00065 }
```

5.202.1.26 sys_sleep()

```
int sys_sleep (
     void )
```

Definition at line 68 of file sysproc.c.

```
00069 {
          int n;
00071
         uint ticks0;
00072
00073
         if(argint(0, &n) < 0)
         return -1;
acquire(&tickslock);
ticks0 = ticks;
while(ticks - ticks0 < n){
  if(myproc()->killed){
00074
00075
00076
00077
00078
             release(&tickslock);
00079
08000
              return -1;
00081
00082
           sleep(&ticks, &tickslock);
00083
00084
         release(&tickslock);
00085
          return 0;
00086 }
```

5.202.1.27 sys_unlink()

```
int sys_unlink (
                void )
Definition at line 185 of file sysfile.c.
00186 {
        struct inode *ip, *dp;
00188
         struct dirent de;
00189
         char name[DIRSIZ], *path;
00190
        uint off;
00191
00192
        if(argstr(0, &path) < 0)</pre>
00193
          return -1;
00194
00195
        begin_op();
00196
        if((dp = nameiparent(path, name)) == 0){
00197
         end_op();
00198
          return -1;
00199
00200
00201
        ilock(dp);
00202
        // Cannot unlink "." or "..".
if (namecmp(name, ".") == 0 || namecmp(name, "..") == 0)
00203
00204
00205
         goto bad;
00206
00207
        if((ip = dirlookup(dp, name, &off)) == 0)
00208
           goto bad;
        ilock(ip);
00209
00210
00211
         if(ip->nlink < 1)</pre>
00212
          panic("unlink: nlink < 1");</pre>
00213
         if(ip->type == T_DIR && !isdirempty(ip)){
         iunlockput(ip);
00214
00215
          goto bad;
00216
00217
00218
        memset(&de, 0, sizeof(de));
if(writei(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
00219
00220
          panic("unlink: writei");
         if(ip->type == T_DIR) {
  dp->nlink--;
  iupdate(dp);
00221
00222
00223
00224
00225
         iunlockput(dp);
00226
00227
        ip->nlink--;
00228
        iupdate(ip);
00229
        iunlockput(ip);
00230
00231
        end_op();
00232
00233
        return 0;
00234
00235 bad:
        iunlockput(dp);
00236
00237
        end_op();
00238
00239 }
```

5.202.1.28 sys_uptime()

```
int sys_uptime (
     void )
```

Definition at line 91 of file sysproc.c.

```
00092 {
00093     uint xticks;
00094
00095     acquire(&tickslock);
00096     xticks = ticks;
00097     release(&tickslock);
00098     return xticks;
00099 }
```

5.202.1.29 sys_wait()

```
int sys_wait (
     void )
```

Definition at line 32 of file sysproc.c.

```
00033 {
00034     return wait();
00035 }
```

5.202.1.30 sys_write()

```
int sys_write (
     void )
```

Definition at line 82 of file sysfile.c.

5.202.1.31 syscall()

```
void syscall (
     void )
```

Definition at line 140 of file syscall.c.

```
00141 {
00142    int num;
00143    struct proc *curproc = myproc();
00144

00145    num = curproc->tf->eax;
00146    if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
        curproc->tf->eax = syscalls[num]();
00147         curproc->tf->eax = syscalls[num]();
00148    } else {
        curproc->pid, curproc->name, num);
        curproc->pid;
00150         curproc->tf->eax = -1;
00151         curproc->tf->eax = -1;
00153 }
```

Referenced by trap().

5.202.2 Variable Documentation

5.202.2.1 syscalls

Initial value:

```
[SYS_fork]
              sys_fork,
[SYS_exit]
[SYS_wait]
              sys_exit,
              svs wait,
[SYS_pipe]
              sys pipe,
[SYS_read]
              sys_read,
[SYS_kill]
              sys_kill,
[SYS_exec]
              sys_exec,
[SYS_fstat]
             sys_fstat,
[SYS_chdir]
             sys_chdir,
[SYS_dup]
              sys_dup,
[SYS_getpid] sys_getpid,
[SYS_sbrk]
              sys_sbrk,
[SYS_sleep]
             sys_sleep,
[SYS_uptime] sys_uptime,
[SYS_open]
[SYS_write]
              sys_open,
             sys_write,
[SYS_mknod]
             sys_mknod,
[SYS_unlink] sys_unlink,
[SYS_link]
              sys_link,
[SYS_mkdir]
             sys_mkdir,
[SYS_close]
             sys_close,
[SYS_halt]
              sys halt,
[SYS_cps]
             svs cps,
[SYS_chpr]
            sys_chpr,
[SYS_getprocs] sys_getprocs,
```

Definition at line 111 of file syscall.c.

Referenced by syscall().

5.203 syscall.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "defs.h"
00003 #include "param.h"
00004 #include "memlayout.h"
00005 #include "mmu.h"
00006 #include "proc.h"
00007 #include "x86.h"
00008 #include "syscall.h"
00010 // User code makes a system call with INT T_SYSCALL.
00011 // System call number in %eax.
00012 // Arguments on the stack, from the user call to the C
00013 // library system call function. The saved user %esp points
00014 // to a saved program counter, and then the first argument.
00016 // Fetch the int at addr from the current process.
00017 int
00018 fetchint(uint addr, int *ip)
00019 {
00020
          struct proc *curproc = myproc();
00021
00022
          if (addr >= curproc->sz || addr+4 > curproc->sz)
00023
             return -1;
         *ip = *(int*)(addr);
00024
00025
         return 0;
00026 }
00027
00028 // Fetch the nul-terminated string at addr from the current process.
00029 // Doesn't actually copy the string - just sets *pp to point at it.
00030 // Returns length of string, not including nul.
00031 int
00032 fetchstr(uint addr, char **pp)
00033 {
00034 char *s, *ep;
00035 struct proc *curproc = myproc();
```

5.203 syscall.c 397

```
00037
        if(addr >= curproc->sz)
00038
          return -1;
00039
        *pp = (char*)addr;
        ep = (char*)curproc->sz;
00040
        for(s = *pp; s < ep; s++) {
  if(*s == 0)</pre>
00041
00042
00043
           return s - *pp;
00044
00045
        return -1;
00046 }
00047
00048 // Fetch the nth 32-bit system call argument.
00050 argint(int n, int *ip)
00051 {
       return fetchint((myproc()->tf->esp) + 4 + 4*n, ip);
00052
00053 }
00055 // Fetch the nth word-sized system call argument as a pointer
00056 // to a block of memory of size bytes. Check that the pointer
00057 // lies within the process address space.
00058 int.
00059 argptr(int n, char **pp, int size)
00060 {
00061
00062
       struct proc *curproc = myproc();
00063
00064
       if(argint(n, \&i) < 0)
00065
       return -1;
if(size < 0 || (uint) i >= curproc->sz || (uint) i+size > curproc->sz)
00066
00067
         return -1;
00068
       *pp = (char*)i;
00069
       return 0;
00070 }
00071
00072 // Fetch the nth word-sized system call argument as a string pointer.
00073 // Check that the pointer is valid and the string is nul-terminated.
00074 // (There is no shared writable memory, so the string can't change
00075 // between this check and being used by the kernel.)
00076 int
00077 argstr(int n, char **pp)
00078 {
00079
       int addr;
      if (argint(n, &addr) < 0)</pre>
08000
00081
          return -1;
00082 return fetchstr(addr, pp);
00083 }
00084
00085 extern int svs chdir(void);
00086 extern int sys_close(void);
00087 extern int sys_dup(void);
00088 extern int sys_exec(void);
00089 extern int sys_exit(void);
00090 extern int sys_fork(void);
00091 extern int sys_fstat(void);
00092 extern int sys_getpid(void);
00093 extern int sys_kill(void);
00094 extern int sys_link(void);
00095 extern int sys_mkdir(void);
00096 extern int sys_mknod(void);
00097 extern int sys_open(void);
00098 extern int sys_pipe(void);
00099 extern int sys_read(void);
00100 extern int sys_sbrk(void);
00101 extern int sys_sleep(void);
00102 extern int sys_unlink(void);
00103 extern int sys_wait (void);
00104 extern int sys_write(void);
00105 extern int sys_uptime(void);
00106 extern int sys_halt(void);
00107 extern int sys_cps(void);
00108 extern int sys_chpr(void);
00109 extern int sys_getprocs(void);
00110
00111 static int (*syscalls[])(void) = {
00112 [SYS_fork]
                  sys_fork,
00113 [SYS_exit]
                    sys_exit,
00114 [SYS_wait]
                    sys_wait,
00115 [SYS_pipe]
                    svs pipe,
00116 [SYS_read]
                    sys read,
00117 [SYS_kill]
                    sys_kill,
00118 [SYS_exec]
                     sys_exec,
                     sys_fstat,
00119 [SYS_fstat]
00120 [SYS_chdir]
                    sys_chdir,
00121 [SYS_dup]
                    sys_dup,
00122 [SYS_getpid] sys_getpid,
```

```
00123 [SYS_sbrk]
                     sys_sbrk,
00124 [SYS_sleep] sys_sleep,
00125 [SYS_uptime] sys_uptime,
00126 [SYS_open]
                     sys_open,
00127 [SYS_write]
00128 [SYS_mknod]
                     sys_write,
                     svs mknod.
00129 [SYS_unlink] sys_unlink,
00130 [SYS_link]
                      sys_link,
00131 [SYS_mkdir]
                      sys_mkdir,
00132 [SYS_close]
                      sys_close,
00133 [SYS_halt]
                     sys_halt,
00134 [SYS_cps]
                     sys_cps,
00135 [SYS_chpr]
                      sys chpr,
00136 [SYS_getprocs] sys_getprocs,
00137 };
00138
00139 void
00140 syscall (void)
00141 {
00142
        int num;
00143
        struct proc *curproc = myproc();
00144
00145
        num = curproc->tf->eax;
        if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {</pre>
00146
00147
          curproc->tf->eax = syscalls[num]();
        } else {
        cprintf("%d %s: unknown sys call %d\n",
00149
00150
                   curproc->pid, curproc->name, num);
00151
          curproc \rightarrow tf \rightarrow eax = -1;
00152 }
00153 }
```

5.204 syscall.d File Reference

5.205 syscall.d

```
Go to the documentation of this file.
```

```
00001 syscall.o: syscall.c /usr/include/stdc-predef.h types.h defs.h param.h \backslash 00002 memlayout.h mmu.h proc.h x86.h syscall.h
```

5.206 syscall.h File Reference

Macros

- #define SYS_chdir 9
- #define SYS_chpr 24
- #define SYS_close 21
- #define SYS_cps 23
- #define SYS_dup 10
- #define SYS_exec 7
- #define SYS_exit 2
- #define SYS_fork 1
- #define SYS_fstat 8
- #define SYS_getpid 11
- #define SYS_getprocs 25
- #define SYS_halt 22
- #define SYS_kill 6
- #define SYS_link 19
- #define SYS_mkdir 20
- #define SYS_mknod 17
- #define SYS_open 15

- #define SYS_pipe 4
- #define SYS_read 5
- #define SYS_sbrk 12
- #define SYS_sleep 13
- #define SYS_unlink 18
- #define SYS_uptime 14
- #define SYS_wait 3
- #define SYS_write 16

5.206.1 Macro Definition Documentation

5.206.1.1 SYS_chdir

```
#define SYS_chdir 9
```

Definition at line 10 of file syscall.h.

5.206.1.2 SYS_chpr

```
#define SYS_chpr 24
```

Definition at line 25 of file syscall.h.

5.206.1.3 SYS_close

```
#define SYS_close 21
```

Definition at line 22 of file syscall.h.

5.206.1.4 SYS_cps

#define SYS_cps 23

Definition at line 24 of file syscall.h.

5.206.1.5 SYS_dup

```
#define SYS_dup 10
```

Definition at line 11 of file syscall.h.

5.206.1.6 SYS_exec

```
#define SYS_exec 7
```

Definition at line 8 of file syscall.h.

5.206.1.7 SYS_exit

```
#define SYS_exit 2
```

Definition at line 3 of file syscall.h.

5.206.1.8 SYS_fork

```
#define SYS_fork 1
```

Definition at line 2 of file syscall.h.

5.206.1.9 SYS fstat

```
#define SYS_fstat 8
```

Definition at line 9 of file syscall.h.

5.206.1.10 SYS_getpid

#define SYS_getpid 11

Definition at line 12 of file syscall.h.

5.206.1.11 SYS_getprocs

#define SYS_getprocs 25

Definition at line 26 of file syscall.h.

5.206.1.12 SYS_halt

#define SYS_halt 22

Definition at line 23 of file syscall.h.

5.206.1.13 SYS_kill

#define SYS_kill 6

Definition at line 7 of file syscall.h.

5.206.1.14 SYS_link

#define SYS_link 19

Definition at line 20 of file syscall.h.

5.206.1.15 SYS_mkdir

#define SYS_mkdir 20

Definition at line 21 of file syscall.h.

5.206.1.16 SYS_mknod

#define SYS_mknod 17

Definition at line 18 of file syscall.h.

5.206.1.17 SYS_open

```
#define SYS_open 15
```

Definition at line 16 of file syscall.h.

5.206.1.18 SYS_pipe

```
#define SYS_pipe 4
```

Definition at line 5 of file syscall.h.

5.206.1.19 SYS_read

```
#define SYS_read 5
```

Definition at line 6 of file syscall.h.

5.206.1.20 SYS_sbrk

```
#define SYS_sbrk 12
```

Definition at line 13 of file syscall.h.

5.206.1.21 SYS_sleep

```
#define SYS_sleep 13
```

Definition at line 14 of file syscall.h.

5.206.1.22 SYS_unlink

#define SYS_unlink 18

Definition at line 19 of file syscall.h.

5.207 syscall.h 403

5.206.1.23 SYS_uptime

```
#define SYS_uptime 14
```

Definition at line 15 of file syscall.h.

5.206.1.24 SYS_wait

```
#define SYS_wait 3
```

Definition at line 4 of file syscall.h.

5.206.1.25 SYS_write

```
#define SYS_write 16
```

Definition at line 17 of file syscall.h.

5.207 syscall.h

Go to the documentation of this file.

```
00001 // System call numbers
00002 #define SYS_fork 1
00003 #define SYS_exit
00004 #define SYS_wait
00005 #define SYS_pipe
00006 #define SYS_read
00000 #define SYS_kill
00008 #define SYS_exec
00009 #define SYS_fstat
00010 #define SYS_chdir
00011 #define SYS_dup
00012 #define SYS_getpid 11
00013 #define SYS_sbrk 12
00014 #define SYS_sleep
00015 #define SYS_uptime 14
00016 #define SYS_open
00017 #define SYS_write
00018 #define SYS_mknod 17
00019 #define SYS_unlink 18
00020 #define SYS_link 19
00021 #define SYS_mkdir 20
00022 #define SYS_close 21
00023 #define SYS_halt
00024 #define SYS_cps 23
00025 #define SYS_chpr 24
00026 #define SYS_getprocs 25
00027
```

5.208 sysfile.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "stat.h"
#include "mmu.h"
#include "proc.h"
#include "fs.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "file.h"
#include "fcntl.h"
```

Functions

```
    static int argfd (int n, int *pfd, struct file **pf)
```

- static struct inode * create (char *path, short type, short major, short minor)
- static int fdalloc (struct file *f)
- static int isdirempty (struct inode *dp)
- int sys_chdir (void)
- int sys_close (void)
- int sys_dup (void)
- int sys_exec (void)
- int sys_fstat (void)
- int sys_link (void)
- int sys_mkdir (void)
- int sys_mknod (void)
- int sys_open (void)
- int sys_pipe (void)
- int sys_read (void)
- int sys unlink (void)
- int sys_write (void)

5.208.1 Function Documentation

5.208.1.1 argfd()

int fd; 00025 struct file *f;

00024

00026 00027

```
static int argfd (
              int * pfd,
              struct file ** pf ) [static]
Definition at line 22 of file sysfile.c.
```

if(argint(n, &fd) < 0)</pre>

```
00029 if (fd < 0 || fd >= NOFILE || (f=myproc()->ofile[fd]) == 0)
```

```
00030 return -1;

00031 if(pfd)

00032 *pfd = fd;

00033 if(pf)

00034 *pf = f;

00035 return 0;

00036 }
```

Referenced by sys_close(), sys_dup(), sys_fstat(), sys_read(), and sys_write().

5.208.1.2 create()

Definition at line 242 of file sysfile.c.

```
00243 {
00244
         struct inode *ip, *dp;
00245
         char name[DIRSIZ];
00246
00247
        if((dp = nameiparent(path, name)) == 0)
return 0;
00248
00249
        ilock(dp);
00250
        if((ip = dirlookup(dp, name, 0)) != 0){
  iunlockput(dp);
}
00251
00252
00253
          ilock(ip);
          if(type == T_FILE && ip->type == T_FILE)
return ip;
00254
00255
00256
           iunlockput(ip);
00257
00258
00259
00260
        if((ip = ialloc(dp->dev, type)) == 0)
00261
         panic("create: ialloc");
00262
00263
        ilock(ip);
00264
        ip->major = major;
        ip->minor = minor;
ip->nlink = 1;
00265
00266
00267
        iupdate(ip);
00268
00269
        if(type == T_DIR){ // Create . and .. entries.
         dp->nlink++; // for ".."
iupdate(dp);
00270
00271
          // No ip->nlink++ for ".": avoid cyclic ref count.
if(dirlink(ip, ".", ip->inum) < 0 || dirlink(ip, "..", dp->inum) < 0)</pre>
00272
00273
00274
             panic("create dots");
00275
00276
00277
        if(dirlink(dp, name, ip->inum) < 0)</pre>
00278
         panic("create: dirlink");
00279
00280
        iunlockput(dp);
00281
00282
        return ip;
00283 }
```

Referenced by sys_mkdir(), sys_mknod(), and sys_open().

5.208.1.3 fdalloc()

```
static int fdalloc (
                struct file * f ) [static]
Definition at line 41 of file sysfile.c.
00042 {
00043
        int fd:
00044
        struct proc *curproc = myproc();
00045
00046
        for (fd = 0; fd < NOFILE; fd++) {</pre>
        if(curproc->ofile[fd] == 0) {
  curproc->ofile[fd] = f;
00047
00048
             return fd;
00049
00050
          }
      }
return -1;
00051
00052
00053 }
```

Referenced by sys_dup(), sys_open(), and sys_pipe().

5.208.1.4 isdirempty()

```
static int isdirempty ( {\tt struct\ inode\ *\ dp\ )} \quad [{\tt static}]
```

Definition at line 169 of file sysfile.c.

```
00170 {
00171
         int off;
00172
         struct dirent de;
00173
        for(off=2*sizeof(de); off<dp->size; off+=sizeof(de)){
00174
         if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
panic("isdirempty: readi");
if(de i=== isdirempty);
00175
00176
           if(de.inum != 0)
00178
             return 0;
00179
00180
        return 1;
00181 }
```

Referenced by sys_unlink().

5.208.1.5 sys_chdir()

```
int sys_chdir (
     void )
```

Definition at line 372 of file sysfile.c.

```
00373 {
00374
       char *path;
00375
       struct inode *ip;
       struct proc *curproc = myproc();
00376
00377
00378
        begin_op();
00379
       if (argstr(0, &path) < 0 || (ip = namei(path)) == 0) {</pre>
00380
         end_op();
00381
          return -1;
00382
00383
       ilock(ip);
00384
       if(ip->type != T_DIR){
        iunlockput(ip);
end_op();
00385
00386
00387
          return -1;
00388
       iunlock(ip);
00389
00390
       iput (curproc->cwd);
00391
        end_op();
00392
        curproc->cwd = ip;
00393
        return 0;
00394 }
```

5.208.1.6 sys_close()

```
int sys_close (
     void )
```

Definition at line 94 of file sysfile.c.

```
00095 {
00096    int fd;
00097    struct file *f;
00098
00099    if(argfd(0, &fd, &f) < 0)
0100        return -1;
00101    myproc()->ofile[fd] = 0;
0102    fileclose(f);
00103    return 0;
00104 }
```

5.208.1.7 sys_dup()

```
int sys_dup (
     void )
```

Definition at line 56 of file sysfile.c.

```
00057 {
00058
        struct file *f;
00059
       int fd;
00060
00061
       if(argfd(0, 0, &f) < 0)
00062
       return -1;
if((fd=fdalloc(f)) < 0)</pre>
00063
00064
          return -1;
00065 filedup(f);
00066 return fd;
00067 }
```

5.208.1.8 sys_exec()

```
int sys_exec (
     void )
```

Definition at line 397 of file sysfile.c.

```
00398 {
00399
         char *path, *argv[MAXARG];
00400
00401
         uint uargv, uarg;
00402
00403
         if(argstr(0, &path) < 0 || argint(1, (int*)&uargv) < 0){</pre>
        return -1;
00404
00405
        memset(argv, 0, sizeof(argv));
for(i=0;; i++){
00406
00407
         if(i >= NELEM(argv))
   return -1;
00408
00409
          if(fetchint(uargv+4*i, (int*)&uarg) < 0)
return -1;
if(uarg == 0) {</pre>
00410
00411
00412
           argv[i] = 0;
break;
00413
00414
00415
          if(fetchstr(uarg, &argv[i]) < 0)
  return -1;</pre>
00416
00417
00418
        }
00419 return exec(path, argv);
00420 }
```

5.208.1.9 sys_fstat()

```
int sys_fstat (
               void )
Definition at line 107 of file sysfile.c.
00108 {
00109
        struct file *f;
00110
       struct stat *st;
00111
00112
       if(argfd(0, 0, &f) < 0 \mid \mid argptr(1, (void*)&st, sizeof(*st)) < 0)
00113
         return -1:
       return filestat(f, st);
00114
00115 }
```

5.208.1.10 sys_link()

```
int sys_link (
     void )
```

Definition at line 119 of file sysfile.c.

```
00120 {
        char name[DIRSIZ], *new, *old;
struct inode *dp, *ip;
00121
00122
00123
00124
        if(argstr(0, &old) < 0 || argstr(1, &new) < 0)</pre>
00125
00126
        begin_op();
00127
00128
        if((ip = namei(old)) == 0){
00129
        end_op();
00130
          return -1;
00131
00132
00133
        ilock(ip);
        if (ip->type == T_DIR) {
  iunlockput(ip);
  end_op();
00134
00135
00136
00137
          return -1;
00138
00139
00140
        ip->nlink++;
00141
        iupdate(ip);
00142
        iunlock(ip);
00143
00144
        if((dp = nameiparent(new, name)) == 0)
           goto bad;
00145
        ilock(dp);
if(dp->dev != ip->dev || dirlink(dp, name, ip->inum) < 0){</pre>
00146
00147
00148
         iunlockput (dp);
00149
          goto bad;
00150
        iunlockput(dp);
00151
00152
        iput(ip);
00153
00154
        end_op();
00155
00156
        return 0;
00157
00158 bad:
        ilock(ip);
00159
00160
        ip->nlink--;
00161
        iupdate(ip);
00162
        iunlockput(ip);
00163
        end_op();
00164
        return -1;
00165 }
```

5.208.1.11 sys_mkdir()

```
int sys_mkdir (
void )
```

Definition at line 336 of file sysfile.c.

```
00337 {
       char *path;
00339
       struct inode *ip;
00340
00341
       begin_op();
00342
       if(argstr(0, &path) < 0 || (ip = create(path, T_DIR, 0, 0)) == 0){</pre>
        end_op();
00343
00344
         return -1;
00345
00346
       iunlockput(ip);
00347 end_op();
00348
       return 0;
00349 }
```

5.208.1.12 sys_mknod()

```
int sys_mknod (
     void )
```

Definition at line 352 of file sysfile.c.

```
00353 {
00354
             struct inode *ip;
00355
            char *path;
00356
            int major, minor;
00357
            begin_op();
if((argstr(0, &path)) < 0 ||
    argint(1, &major) < 0 ||
    argint(2, &minor) < 0 ||
    (ip = create(path, T_DEV, major, minor)) == 0){</pre>
00358
00359
00360
00361
00362
00363
               end_op();
00364
               return -1;
00365 }
00365 iunlockput(ip);
00367 end_op();
00368 return 0;
00369 }
```

5.208.1.13 sys_open()

```
int sys_open (
     void )
```

Definition at line 286 of file sysfile.c.

```
00287 {
00288
        char *path;
        int fd, omode;
struct file *f;
00289
00290
00291
        struct inode *ip;
00292
        if(argstr(0, &path) < 0 || argint(1, &omode) < 0)</pre>
00293
00294
          return -1;
00295
00296
        begin_op();
00297
        if (omode & O_CREATE) {
00298
        ip = create(path, T_FILE, 0, 0);
if(ip == 0){
00299
00300
00301
            end_op();
00302
             return -1;
```

```
00303
00304
        } else {
          if((ip = namei(path)) == 0){
00305
00306
           end_op();
00307
            return -1;
00308
00309
          ilock(ip);
00310
          if(ip->type == T_DIR && omode != O_RDONLY) {
00311
           iunlockput(ip);
00312
             end_op();
00313
            return -1;
00314
00315
        }
00316
00317
        if((f = filealloc()) == 0 \mid \mid (fd = fdalloc(f)) < 0){
00318
            fileclose(f);
00319
00320
          iunlockput(ip);
00321
         end_op();
00322
          return -1;
00323
00324
        iunlock(ip);
00325
        end_op();
00326
00327
        f->type = FD_INODE;
00328 f->ip = ip;
00329 f->off = 0;
        f->readable = !(omode & O_WRONLY);
f->writable = (omode & O_WRONLY) || (omode & O_RDWR);
00330
00331
00332
        return fd;
00333 }
```

5.208.1.14 sys_pipe()

```
int sys_pipe (
     void )
```

Definition at line 423 of file sysfile.c.

```
00424 {
          int *fd;
struct file *rf, *wf;
int fd0, fd1;
00425
00427
00428
00429
          if(argptr(0, (void*)&fd, 2*sizeof(fd[0])) < 0)</pre>
00430
          return -1;
if (pipealloc(&rf, &wf) < 0)</pre>
00431
00432
            return -1;
00433
          fd0 = -1;
         if((fd0 = fdalloc(rf)) < 0 || (fd1 = fdalloc(wf)) < 0){
   if(fd0 >= 0)
       myproc() -> ofile[fd0] = 0;
00434
00435
00436
           fileclose(rf);
00437
           fileclose(wf);
return -1;
00438
00439
00440
00441
         fd[0] = fd0;
fd[1] = fd1;
00442
00443
         return 0;
00444 }
```

5.208.1.15 sys_read()

```
int sys_read (
     void )
```

Definition at line 70 of file sysfile.c.

```
00071 {
00072 struct file *f;
00073 int n;
```

5.208.1.16 sys unlink()

```
int sys_unlink (
     void )
```

Definition at line 185 of file sysfile.c.

```
00186 {
         struct inode *ip, *dp;
struct dirent de;
00187
00188
         char name[DIRSIZ], *path;
00189
00190
         uint off;
00191
00192
         if(argstr(0, &path) < 0)</pre>
00193
          return -1;
00194
00195
         begin_op();
00196
         if ((dp = nameiparent(path, name)) == 0) {
00197
          end_op();
00198
           return -1;
00199
00200
00201
         ilock(dp);
00202
         // Cannot unlink "." or "..".
if(namecmp(name, ".") == 0 || namecmp(name, "..") == 0)
00203
00204
00205
          goto bad;
00206
00207
         if((ip = dirlookup(dp, name, &off)) == 0)
  goto bad;
00208
00209
         ilock(ip);
00210
00211
         if(ip->nlink < 1)</pre>
         panic("unlink: nlink < 1");
if(ip->type == T_DIR && !isdirempty(ip)){
00212
00213
         iunlockput(ip);
goto bad;
00214
00215
00216
00217
         memset(&de, 0, sizeof(de));
if(writei(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
panic("unlink: writei");
00218
00219
00220
         if(ip->type == T_DIR) {
  dp->nlink--;
  in-day ();
00221
00222
00223
           iupdate(dp);
00224
00225
         iunlockput(dp);
00226
00227
         ip->nlink--;
00228
         iupdate(ip);
00229
         iunlockput(ip);
00230
00231
         end_op();
00232
00233
         return 0;
00234
00235 bad:
00236
        iunlockput(dp);
00237
         end_op();
00238
         return -1;
00239 }
```

5.208.1.17 sys_write()

```
int sys_write (
               void )
Definition at line 82 of file sysfile.c.
        struct file *f;
00084
00085
        int n;
00086
       char *p;
00087
00088
       if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)</pre>
00089
         return -1;
00090
       return filewrite(f, p, n);
00091 }
```

5.209 sysfile.c

Go to the documentation of this file.

```
00001 //
00002 // File-system system calls.
00003 // Mostly argument checking, since we don't trust
00004 // user code, and calls into file.c and fs.c.
00005 //
00006
00007 #include "types.h"
00008 #include "defs.h"
00009 #include "param.h"
00010 #include "stat.h"
00011 #include "mmu.h"
00012 #include "proc.h"
00013 #include "fs.h"
00014 #include "spinlock.h"
00015 #include "sleeplock.h"
00016 #include "file.h"
00017 #include "fcntl.h"
00018
00019 // Fetch the nth word-sized system call argument as a file descriptor
00020 // and return both the descriptor and the corresponding struct file.
00021 static int
00022 argfd(int n, int *pfd, struct file **pf)
00023 {
       int fd;
struct file *f;
00024
00025
00026
00027
        if(argint(n, &fd) < 0)</pre>
00028
          return -1;
        if(fd < 0 || fd >= NOFILE || (f=myproc()->ofile[fd]) == 0)
00029
00030
           return -1;
        if (pfd)
00031
00032
          *pfd = fd;
00033
        if(pf)
00034
          *pf = f;
00035
        return 0;
00036 }
00037
00038 // Allocate a file descriptor for the given file.
00039 // Takes over file reference from caller on success.
00040 static int
00041 fdalloc(struct file *f)
00042 {
00043
        int fd;
00044
        struct proc *curproc = myproc();
00045
         for (fd = 0; fd < NOFILE; fd++) {</pre>
00046
         if(curproc->ofile[fd] == 0){
00047
00048
           curproc->ofile[fd] = f;
00049
             return fd;
00050
          }
00051 }
00052
        return -1;
00053 }
00054
00055 int
00056 sys_dup(void)
00057 {
00058 struct file *f;
00059 int fd;
```

5.209 sysfile.c 413

```
00060
00061
        if(argfd(0, 0, &f) < 0)
00062
           return -1;
        if((fd=fdalloc(f)) < 0)</pre>
00063
00064
           return -1;
        filedup(f);
00065
00066
        return fd;
00067 }
00068
00069 int
00070 sys_read(void)
00071 {
00072
        struct file *f;
00073
        int n;
00074
        char *p;
00075
00076
        if(argfd(0, 0, &f) < 0 \mid | argint(2, &n) < 0 \mid | argptr(1, &p, n) < 0)
00077
        return -1;
return fileread(f, p, n);
00078
00079 }
08000
00081 int
00082 sys_write(void)
00083 {
00084
        struct file *f;
00085
        int n;
        char *p;
00086
00087
        if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)</pre>
00088
00089
        return -1;
return filewrite(f, p, n);
00090
00091 }
00092
00093 int
00094 sys_close(void)
00095 {
00096
        int fd;
        struct file *f;
00098
00099
        if(argfd(0, &fd, &f) < 0)</pre>
        return -1;
myproc()->ofile[fd] = 0;
00100
00101
        fileclose(f);
00102
00103
        return 0;
00104 }
00105
00106 int
00107 sys_fstat(void)
00108 {
00109
        struct file *f;
00110
        struct stat *st;
00111
00112
        if(argfd(0, 0, &f) < 0 \mid \mid argptr(1, (void*)&st, sizeof(*st)) < 0)
        return -1;
return filestat(f, st);
00113
00114
00115 }
00116
00117 // Create the path new as a link to the same inode as old.
00118 int
00119 sys_link(void)
00120 {
        char name[DIRSIZ], *new, *old;
00121
00122
        struct inode *dp, *ip;
00123
00124
        if(argstr(0, &old) < 0 || argstr(1, &new) < 0)</pre>
00125
          return -1;
00126
00127
        begin_op();
00128
        if((ip = namei(old)) == 0){
00129
         end_op();
00130
          return -1;
00131
00132
        ilock(ip);
00133
        if(ip->type == T_DIR){
  iunlockput(ip);
00134
00135
00136
          end_op();
00137
          return -1;
00138
00139
00140
        ip->nlink++;
00141
        iupdate(ip);
00142
        iunlock(ip);
00143
        if((dp = nameiparent(new, name)) == 0)
  goto bad;
00144
00145
        ilock(dp);
00146
```

```
if(dp->dev != ip->dev || dirlink(dp, name, ip->inum) < 0){</pre>
00148
         iunlockput(dp);
00149
           goto bad;
00150
        iunlockput(dp);
00151
00152
        iput(ip);
00153
00154
        end_op();
00155
00156
        return 0;
00157
00158 bad:
00159
        ilock(ip);
00160
        ip->nlink--
00161
        iupdate(ip);
00162
        iunlockput(ip);
00163
        end_op();
00164
        return -1;
00165 }
00166
00167 // Is the directory dp empty except for "." and ".." ?
00168 static int
00169 isdirempty(struct inode *dp)
00170 {
00171
        int off;
00172
        struct dirent de;
00173
00174
        for(off=2*sizeof(de); off<dp->size; off+=sizeof(de)){
          if(readi(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
panic("isdirempty: readi");
00175
00176
00177
           if(de.inum != 0)
00178
            return 0;
00179
00180
        return 1;
00181 }
00182
00183 //PAGEBREAK!
00184 int
00185 sys_unlink(void)
00186 {
00187
        struct inode *ip, *dp;
00188
        struct dirent de;
00189
        char name[DIRSIZ], *path;
        uint off;
00190
00191
00192
        if(argstr(0, &path) < 0)</pre>
00193
          return -1;
00194
00195
        begin_op();
00196
        if((dp = nameiparent(path, name)) == 0){
00197
          end_op();
00198
00199
00200
00201
        ilock(dp);
00202
        // Cannot unlink "." or "..".

if(namecmp(name, ".") == 0 || namecmp(name, "..") == 0)
00203
00204
00205
          goto bad;
00206
        if((ip = dirlookup(dp, name, &off)) == 0)
00207
          goto bad;
00208
00209
        ilock(ip);
00210
00211
        if(ip->nlink < 1)</pre>
         panic("unlink: nlink < 1");
if(ip->type == T_DIR && !isdirempty(ip)){
  iunlockput(ip);
00212
00213
00214
00215
          goto bad;
00216
00217
00218
        memset(&de, 0, sizeof(de));
        if(writei(dp, (char*)&de, off, sizeof(de)) != sizeof(de))
panic("unlink: writei");
00219
00220
        if (ip->type == T_DIR) {
  dp->nlink--;
00221
00222
00223
          iupdate(dp);
00224
00225
        iunlockput(dp);
00226
00227
        ip->nlink--;
00228
         iupdate(ip);
00229
        iunlockput(ip);
00230
00231
        end_op();
00232
00233
        return 0:
```

5.209 sysfile.c 415

```
00234
00235 bad:
00236
         iunlockput(dp);
00237
        end_op();
00238
        return -1;
00239 }
00240
00241 static struct inode*
00242 create(char *path, short type, short major, short minor)
00243 {
00244
         struct inode *ip, *dp;
00245
        char name[DIRSIZ];
00246
00247
        if((dp = nameiparent(path, name)) == 0)
00248
           return 0;
00249
        ilock(dp);
00250
00251
         if((ip = dirlookup(dp, name, 0)) != 0){
          iunlockput(dp);
00252
00253
           ilock(ip);
00254
           if(type == T_FILE && ip->type == T_FILE)
             return ip;
00255
00256
           iunlockput(ip);
00257
           return 0;
00258
00259
         if((ip = ialloc(dp->dev, type)) == 0)
  panic("create: ialloc");
00260
00261
00262
00263
         ilock(ip);
00264
        ip->major = major;
        ip->minor = minor;
ip->nlink = 1;
00265
00266
00267
         iupdate(ip);
00268
         if(type == T_DIR){    // Create . and .. entries.
    dp->nlink++;    // for ".."
00269
00270
00271
           iupdate(dp);
           // No ip->nlink++ for ".": avoid cyclic ref count.
if(dirlink(ip, ".", ip->inum) < 0 || dirlink(ip, "..", dp->inum) < 0)
00272
00273
00274
             panic("create dots");
00275
00276
        if(dirlink(dp, name, ip->inum) < 0)
  panic("create: dirlink");</pre>
00277
00278
00279
00280
        iunlockput(dp);
00281
00282
        return ip;
00283 }
00284
00285 int
00286 sys_open(void)
00287 {
00288
        char *path;
        int fd, omode;
struct file *f;
00289
00290
00291
        struct inode *ip;
00292
00293
        if(argstr(0, &path) < 0 || argint(1, &omode) < 0)</pre>
00294
          return -1;
00295
00296
        begin_op();
00297
00298
         if (omode & O_CREATE) {
00299
          ip = create(path, T_FILE, 0, 0);
00300
           if(ip == 0){
00301
             end_op();
00302
             return -1:
00303
00304
         } else {
           if((ip = namei(path)) == 0){
00305
            end_op();
00306
00307
             return -1;
00308
           ilock(ip);
00309
00310
           if(ip->type == T_DIR && omode != O_RDONLY) {
00311
             iunlockput(ip);
00312
             end_op();
00313
             return -1;
00314
00315
         }
00316
         if((f = filealloc()) == 0 || (fd = fdalloc(f)) < 0){</pre>
00317
00318
          if(f)
             fileclose(f):
00319
00320
           iunlockput(ip);
```

```
00321
          end_op();
00322
          return -1;
00323
00324
        iunlock(ip);
00325
        end_op();
00326
00327
        f->type = FD_INODE;
00328
        f \rightarrow ip = ip;
        f \rightarrow off = 0;
00329
        f->readable = !(omode & O_WRONLY);
f->writable = (omode & O_WRONLY) || (omode & O_RDWR);
00330
00331
00332
        return fd:
00333 }
00334
00335 int
00336 sys_mkdir(void)
00337 {
00338
        char *path;
struct inode *ip;
00339
00340
00341
         begin_op();
00342
        if(argstr(0, &path) < 0 || (ip = create(path, T_DIR, 0, 0)) == 0){</pre>
00343
        end_op();
return -1;
00344
00345
00346
        iunlockput(ip);
00347
        end_op();
00348
        return 0;
00349 }
00350
00351 int
00352 sys_mknod(void)
00353 {
00354
        struct inode *ip;
00355
         char *path;
00356
        int major, minor;
00357
00358
        begin_op();
        if((argstr(0, &path)) < 0 ||
    argint(1, &major) < 0 ||
    argint(2, &minor) < 0 ||</pre>
00359
00360
00361
00362
            (ip = create(path, T_DEV, major, minor)) == 0){
00363
         end op();
00364
          return -1;
00365
00366
        iunlockput(ip);
00367
        end_op();
00368
        return 0;
00369 }
00370
00371 int
00372 sys_chdir(void)
00373 {
00374
        char *path;
00375
        struct inode *ip;
00376
        struct proc *curproc = myproc();
00377
00378
        begin_op();
00379
        if(argstr(0, &path) < 0 || (ip = namei(path)) == 0){</pre>
00380
          end_op();
00381
           return -1;
00382
00383
        ilock(ip);
00384
        if(ip->type != T_DIR){
00385
         iunlockput(ip);
00386
         end_op();
00387
           return -1;
00388
00389
        iunlock(ip);
00390
        iput (curproc->cwd);
00391
         end_op();
00392
        curproc->cwd = ip;
00393
        return 0;
00394 }
00395
00396 int
00397 sys_exec(void)
00398 {
00399
        char *path, *argv[MAXARG];
00400
        int i;
00401
        uint uargv, uarg;
00402
00403
         if(argstr(0, &path) < 0 || argint(1, (int*)&uargv) < 0){</pre>
00404
          return -1;
00405
        memset(argv, 0, sizeof(argv));
for(i=0;; i++){
00406
00407
```

```
00408
          if(i >= NELEM(argv))
00409
          if (fetchint (uargv+4*i, (int*)&uarg) < 0)</pre>
00410
          return -1;
if(uarg == 0){
00411
00412
          argv[i] = 0;
break;
00413
00414
00415
00416
          if(fetchstr(uarg, &argv[i]) < 0)</pre>
00417
             return -1;
00418 }
00419
        return exec(path, argv);
00420 }
00421
00422 int
00423 sys_pipe(void)
00424 {
00425
        int *fd;
        struct file *rf, *wf;
        int fd0, fd1;
00428
        if(argptr(0, (void*)&fd, 2*sizeof(fd[0])) < 0)</pre>
00429
00430
          return -1;
        if(pipealloc(&rf, &wf) < 0)</pre>
00431
00432
           return -1;
00433 fd0 = -1;
00434
        if((fd0 = fdalloc(rf)) < 0 || (fd1 = fdalloc(wf)) < 0){</pre>
00435
         if(fd0 >= 0)
            myproc()->ofile[fd0] = 0;
00436
          fileclose(rf);
00437
         fileclose(wf);
00438
return -1;
00440 }
00441 fd[0] = fd0;
00442 fd[1] = fd1;
00443
        return 0;
00444 }
```

5.210 sysfile.d File Reference

5.211 sysfile.d

```
Go to the documentation of this file.
```

```
00001 sysfile.o: sysfile.c /usr/include/stdc-predef.h types.h defs.h param.h \ 00002 stat.h mmu.h proc.h fs.h spinlock.h sleeplock.h file.h fcntl.h
```

5.212 sysproc.c File Reference

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "uproc.h"
#include "spinlock.h"
```

Functions

```
• int sys_chpr (void)
```

- int sys_cps (void)
- int sys_exit (void)
- int sys_fork (void)
- int sys_getpid (void)
- int sys_getprocs (void)
- int sys_halt (void)
- int sys_kill (void)
- int sys_sbrk (void)
- int sys_sleep (void)
- int sys_uptime (void)
- int sys_wait (void)

Variables

struct {
 struct spinlock lock
 struct proc proc [NPROC]
 } ptable

5.212.1 Function Documentation

5.212.1.1 sys_chpr()

```
int sys_chpr (
     void )
```

Definition at line 125 of file sysproc.c.

```
00126 {
00127    int pid, pr;
00128    if(argint(0, &pid) < 0)
00129    return -1;
00130    if(argint(1, &pr) < 0)
00131    return -1;
00132
00133    return chpr(pid, pr);
00134 }
```

5.212.1.2 sys_cps()

```
int sys_cps (
     void )
```

Definition at line 119 of file sysproc.c.

```
00120 {
00121 return cps();
00122 }
```

5.212.1.3 sys_exit()

```
int sys_exit (
          void )
```

Definition at line 25 of file sysproc.c.

```
00026 {
00027 exit();
00028 return 0; // not reached
00029 }
```

5.212.1.4 sys_fork()

```
int sys_fork (
     void )
```

Definition at line 19 of file sysproc.c.

```
00020 {
00021 return fork();
00022 }
```

5.212.1.5 sys getpid()

```
int sys_getpid (
     void )
```

Definition at line 48 of file sysproc.c.

```
00049 {
00050 return myproc()->pid;
00051 }
```

5.212.1.6 sys_getprocs()

```
int sys_getprocs (
     void )
```

Definition at line 137 of file sysproc.c.

```
00138
         // declare local variables for max uproc size, struct uproc, and counter
00139
         int max;
        struct uproc *p;
int i=0;
00140
00141
00142
00143
         // if argint has trouble, return error -1
00144
         if(argint(0,&max) < 0)</pre>
00145
         return -1;
00146
00147
        // if argptr for allocating struct size has trouble, return -1
if(argptr(1,(char**)&p, max*sizeof(struct uproc)) < 0)
return -1;</pre>
00148
00149
00150
00151
         // create pointer to ptable processes
00152
        struct proc *ptr = ptable.proc;
00153
00154
        // loop through ptable
00155
        for(; ptr < &ptable.proc[NPROC]; ptr++)</pre>
00156
```

```
if(!(ptr->state == UNUSED))
00158
               \ensuremath{//} if the process in ptable is not UNUSED, assign pid, parent pid, and name to uproc
00159
              p[i].pid = ptr->pid;
p[i].ppid = ptr->parent->pid;
strncpy(p[i].name, ptr->name, 16);
// add 1 to the process counter
00160
00161
00162
00163
00164
00165
         }
00166
00167
00168
          // return the number of processes that are not UNUSED
00169 return i;
00170 }
```

5.212.1.7 sys_halt()

```
int sys_halt (
     void )
```

Definition at line 111 of file sysproc.c.

```
00112 {
00113 outb(0xf4, 0x00);
00114 return 0;
00115 }
```

5.212.1.8 sys_kill()

```
int sys_kill (
     void )
```

Definition at line 38 of file sysproc.c.

```
00039 {
00040 int pid;
00041
00042 if(argint(0, &pid) < 0)
00043 return -1;
00044 return kill(pid);
00045 }
```

5.212.1.9 sys_sbrk()

```
int sys_sbrk (
     void )
```

Definition at line 54 of file sysproc.c.

```
00056
         int addr;
00057
         int n;
00058
         if (argint(0, &n) < 0)
    return -1;</pre>
00059
00060
00061
         addr = myproc()->sz;
         if (growproc(n) < 0)
  return -1;</pre>
00062
00063
         return addr;
00064
00065 }
```

5.212.1.10 sys_sleep()

```
int sys_sleep (
     void )
```

Definition at line 68 of file sysproc.c.

```
00069 {
00070    int n;
00071    uint ticks0;
00072    if(argint(0, &n) < 0)
00073    if(argint(0, &n) < 0)
00074     return -1;
00075    acquire(&tickslock);
00076    ticks0 = ticks;
00077    while(ticks - ticks0 < n) {
    if(myproc()->killed) {
        return -1;
00080         return -1;
00081    }
00082    sleep(&ticks, &tickslock);
00083    }
00084    release(&tickslock);
00085    return 0;
00086 }
```

5.212.1.11 sys_uptime()

```
int sys_uptime (
     void )
```

Definition at line 91 of file sysproc.c.

```
00092 {
00093     uint xticks;
00094
00095     acquire(&tickslock);
00096     xticks = ticks;
00097     release(&tickslock);
00098     return xticks;
00099 }
```

5.212.1.12 sys_wait()

```
int sys_wait (
     void )
```

Definition at line 32 of file sysproc.c.

```
00033 {
00034    return wait();
00035 }
```

5.212.2 Variable Documentation

5.212.2.1 lock

```
struct spinlock lock
```

Definition at line 13 of file sysproc.c.

5.212.2.2 proc

```
struct proc proc[NPROC]
```

Definition at line 14 of file sysproc.c.

5.212.2.3

```
struct { ... } ptable
```

Referenced by sys_getprocs().

5.213 sysproc.c

Go to the documentation of this file.

```
Go to the documentation of the control of the contr
     00011
   00015 } ptable;
     00016
     00017
   00018 int
00019 sys_fork(void)
     00020 {
      00021
                                                         return fork();
      00022 }
     00023
   00024 int
00025 sys_exit(void)
00026 {
   00027 exit();
00028 return
   00028 return 0; // not reached 00029 }
     00030
     00031 int
   00032 sys_wait(void)
00033 {
      00034 return wait();
      00035 }
     00036
   00037 int
00038 sys_kill(void)
  00038 Syru
00039 {
00040 int pid;
```

5.213 sysproc.c 423

```
00041
00042
        if(argint(0, &pid) < 0)</pre>
00043
          return -1;
       return kill (pid);
00044
00045 }
00046
00047 int
00048 sys_getpid(void)
00049 {
00050
        return myproc()->pid;
00051 }
00052
00053 int
00054 sys_sbrk(void)
00055 {
00056
       int addr;
00057
        int n;
00058
00059
        if(argint(0, &n) < 0)</pre>
00060
          return -1;
00061
        addr = myproc()->sz;
00062
        if (growproc(n) < 0)</pre>
00063
         return -1;
00064
        return addr;
00065 }
00066
00067 int
00068 sys_sleep(void)
00069 {
00070
        int n;
00071
        uint ticks0;
00072
00073
        if(argint(0, &n) < 0)
00074
          return -1;
        acquire(&tickslock);
00075
        ticks0 = ticks;
while(ticks - ticks0 < n){</pre>
00076
00077
00078
         if (myproc() ->killed) {
00079
            release(&tickslock);
08000
            return -1;
00081
00082
          sleep(&ticks, &tickslock);
00083
00084
        release(&tickslock);
00085 return 0;
00086 }
00087
00088 // return how many clock tick interrupts have occurred 00089 // since start.
00090 int
00091 sys_uptime(void)
00092 {
00093
        uint xticks;
00094
00095
       acquire(&tickslock);
       xticks = ticks;
00096
00097
        release(&tickslock);
00098
       return xticks;
00099 }
00100
00101 /*int
00102 sys_halt(void)
00103 {
00104
      char *p = "Shutdown";
00105
        for( ; *p; p++)
         outw(0xB004, 0x2000);
00106
00107
        return 0;
00108 }*/
00109
00110 int
00111 sys_halt(void)
00112 {
00113 outb(0xf4, 0x00);
00114
        return 0;
00115 }
00116
00117
00118 int
00119 sys_cps(void)
00120 {
00121
        return cps();
00122 }
00123
00124 int
00125 sys_chpr(void)
00126 {
00127
       int pid, pr;
```

```
if(argint(0, &pid) < 0)</pre>
00129
          return -1;
        if(argint(1, &pr) < 0)</pre>
00130
00131
        return -1;
00132
00133
        return chpr(pid, pr);
00134 }
00135
00136 int
00137 sys_getprocs(void){
00138  // declare local
       // declare local variables for max uproc size, struct uproc, and counter
00139
        int max;
00140
        struct uproc *p;
00141
        int i=0;
00142
00143
        // if argint has trouble, return error -1
00144
        if(argint(0, \&max) < 0)
00145
          return -1;
00147
        // if argptr for allocating struct size has trouble, return -1
00148
        if(argptr(1,(char**)&p, max*sizeof(struct uproc)) < 0)</pre>
00149
          return -1;
00150
        // create pointer to ptable processes
00151
00152
        struct proc *ptr = ptable.proc;
00154
        // loop through ptable
00155
        for(; ptr < &ptable.proc[NPROC]; ptr++)</pre>
00156
00157
          if(!(ptr->state == UNUSED))
00158
00159
             // if the process in ptable is not UNUSED, assign pid, parent pid, and name to uproc
            p[i].pid = ptr->pid;
p[i].ppid = ptr->parent->pid;
00160
00161
00162
             strncpy(p[i].name, ptr->name, 16);
             // add 1 to the process counter
00163
            i++;
00164
00165
00166
00167
00168
        // return the number of processes that are not UNUSED
00169
        return i;
00170 }
00171
00172
00173
```

5.214 sysproc.d File Reference

5.215 sysproc.d

```
Go to the documentation of this file.
```

```
00001 sysproc.o: sysproc.c /usr/include/stdc-predef.h types.h x86.h defs.h \ 00002 date.h param.h memlayout.h mmu.h proc.h uproc.h spinlock.h
```

5.216 trap.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
#include "traps.h"
#include "spinlock.h"
```

Functions

- void idtinit (void)
- void trap (struct trapframe *tf)
- void tvinit (void)

Variables

- struct gatedesc idt [256]
- · uint ticks
- · struct spinlock tickslock
- uint vectors []

5.216.1 Function Documentation

5.216.1.1 idtinit()

```
void idtinit (
            void )
```

Definition at line 30 of file trap.c.

```
00031 {
00032 lidt(idt, sizeof(idt));
00033 }
```

5.216.1.2 trap()

```
void trap (
            struct trapframe * tf)
```

Definition at line 37 of file trap.c.

```
00039
        if(tf->trapno == T_SYSCALL){
        if (myproc() ->killed)
00040
         exit();
myproc()->tf = tf;
00041
00042
        syscall();
if (myproc()->killed)
00043
00044
00045
           exit();
00046
         return;
00047
00048
       00049
00050
00051
00052
           acquire(&tickslock);
00053
           ticks++;
00054
           wakeup(&ticks);
           release (&tickslock);
00055
00056
00057
         lapiceoi();
       break;
case T_IRQ0 + IRQ_IDE:
00058
00059
        ideintr();
00061
         lapiceoi();
00062
         break;
00063
       case T_IRQ0 + IRQ_IDE+1:
```

```
// Bochs generates spurious IDE1 interrupts.
        break;
case T_IRQ0 + IRQ_KBD:
00065
00066
         kbdintr();
00067
00068
           lapiceoi();
00069
           break:
00070
         case T_IRQ0 + IRQ_COM1:
00071
         uartintr();
00072
           lapiceoi();
        break;
case T_IRQ0 + 7:
case T_IRQ0 + IRQ_SPURIOUS:
cprintf("cpu%d: spurious interrupt at %x:%x\n",
00073
00074
00075
00076
00077
                   cpuid(), tf->cs, tf->eip);
00078
           lapiceoi();
00079
          break;
08000
00081
        //PAGEBREAK: 13
00082
        default:
00083
          if(myproc() == 0 || (tf->cs&3) == 0){
00084
             // In kernel, it must be our mistake.
00085
             cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)n",
00086
                      tf->trapno, cpuid(), tf->eip, rcr2());
00087
             panic("trap");
00088
00089
           // In user space, assume process misbehaved.
00090
           cprintf("pid %d %s: trap %d err %d on cpu %d "
00091
                    "eip 0x%x addr 0x%x--kill procn",
          myproc()->pid, myproc()->name, tf->trapno,
    tf->err, cpuid(), tf->eip, rcr2());
myproc()->killed = 1;
00092
00093
00094
00095
00096
00097
         \ensuremath{//} Force process exit if it has been killed and is in user space.
        // (If it is still executing in the kernel, let it keep running
// until it gets to the regular system call return.)
00098
00099
00100
         if(myproc() && myproc()->killed && (tf->cs&3) == DPL_USER)
00102
00103
         // Force process to give up CPU on clock tick.
00104
        // If interrupts were on while locks held, would need to check nlock.
         if (myproc() && myproc()->state == RUNNING &&
00105
00106
           tf->trapno == T_IRQ0+IRQ_TIMER)
00107
           yield();
00108
00109
         \ensuremath{//} Check if the process has been killed since we yielded
00110
         if(myproc() && myproc()->killed && (tf->cs&3) == DPL_USER)
00111
           exit();
00112 }
```

5.216.1.3 tvinit()

```
void tvinit (
     void )
```

Definition at line 18 of file trap.c.

```
00019 {
00020    int i;
00021
00022    for(i = 0; i < 256; i++)
00023         SETGATE(idt[i], 0, SEG_KCODE«3, vectors[i], 0);
00024    SETGATE(idt[T_SYSCALL], 1, SEG_KCODE«3, vectors[T_SYSCALL], DPL_USER);
00025    initlock(&tickslock, "time");
00027 }
```

5.216.2 Variable Documentation

5.216.2.1 idt

```
struct gatedesc idt[256]
```

Definition at line 12 of file trap.c.

Referenced by idtinit(), and tvinit().

5.216.2.2 ticks

```
uint ticks
```

Definition at line 15 of file trap.c.

Referenced by sys_sleep(), sys_uptime(), and trap().

5.216.2.3 tickslock

```
struct spinlock tickslock
```

Definition at line 14 of file trap.c.

Referenced by sys_sleep(), sys_uptime(), trap(), and tvinit().

5.216.2.4 vectors

```
uint vectors[] [extern]
```

Referenced by tvinit().

5.217 trap.c

```
Go to the documentation of this file.
```

```
00001 #include "types.h"
00002 #include "defs.h"
00003 #include "param.h"
00004 #include "memlayout.h"
00005 #include "mmu.h"
00006 #include "proc.h"
00007 #include "x86.h"
00008 #include "traps.h"
00009 #include "spinlock.h"
00011 // Interrupt descriptor table (shared by all CPUs).
00012 struct gatedesc idt[256];
00013 extern uint vectors[]; // in vectors.S: array of 256 entry pointers
00014 struct spinlock tickslock;
00015 uint ticks;
00016
00017 void
00018 tvinit(void)
00019 {
00020
         int i:
00021
         for(i = 0; i < 256; i++)</pre>
        SETGATE (idt[i], 0, SEG_KCODE«3, vectors[i], 0);
SETGATE (idt[T_SYSCALL], 1, SEG_KCODE«3, vectors[T_SYSCALL], DPL_USER);
00023
00024
00025
00026 initlock(&tickslock, "time");
00027 }
00028
00029 void
00030 idtinit (void)
00031 {
00032
         lidt(idt, sizeof(idt));
00033 }
00034
00035 //PAGEBREAK: 41
00036 void
00037 trap(struct trapframe *tf)
00038 {
         if(tf->trapno == T SYSCALL){
00039
         if (myproc() ->killed)
00040
             exit();
00042
           myproc()->tf = tf;
00043
           syscall();
00044
           if(myproc()->killed)
00045
             exit();
00046
           return:
00047
00048
00049
         switch(tf->trapno) {
         case T_IRQ0 + IRQ_TIMER:
  if(cpuid() == 0){
00050
00051
             acquire(&tickslock);
00052
00053
              ticks++:
00054
              wakeup(&ticks);
00055
             release(&tickslock);
00056
00057
           lapiceoi();
         break;
case T_IRQ0 + IRQ_IDE:
ideintr();
00058
00059
00060
00061
            lapiceoi();
00062
           break;
         case T_IRQ0 + IRQ_IDE+1:
   // Bochs generates spurious IDE1 interrupts.
   break;
00063
00064
00065
00066
         case T_IRQ0 + IRQ_KBD:
         kbdintr();
00067
00068
           lapiceoi();
         break;
case T_IRQ0 + IRQ_COM1:
00069
00070
          uartintr();
00071
00072
           lapiceoi();
00073
           break;
00074
         case T_IRQ0 + 7:
         case T_IRQ0 + IRQ_SPURIOUS:
00075
          cprintf("cpu%d: spurious interrupt at x:x\n",
00076
00077
                    cpuid(), tf->cs, tf->eip);
           lapiceoi();
00078
00079
           break;
08000
00081
         //PAGEBREAK: 13
00082
         default:
```

```
if(myproc() == 0 || (tf->cs&3) == 0){}
00084
             // In kernel, it must be our mistake.
00085
              cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)n",
                      tf->trapno, cpuid(), tf->eip, rcr2());
00086
             panic("trap");
00087
00088
            // In user space, assume process misbehaved.
00090
           cprintf("pid %d %s: trap %d err %d on cpu %d "
00091
                     "eip 0x%x addr 0x%x--kill procn",
                    myproc()->pid, myproc()->name, tf->trapno,
tf->err, cpuid(), tf->eip, rcr2());
00092
00093
           myproc()->killed = 1;
00094
00095
00096
00097
         \ensuremath{//} Force process exit if it has been killed and is in user space.
         // (If it is still executing in the kernel, let it keep running
// until it gets to the regular system call return.)
00098
00099
00100
         if(myproc() && myproc()->killed && (tf->cs&3) == DPL_USER)
00101
00102
00103
         // Force process to give up CPU on clock tick.
        /// If interrupts were on while locks held, would need to check nlock.
if(myproc() && myproc()->state == RUNNING &&
00104
00105
00106
            tf->trapno == T_IRQ0+IRQ_TIMER)
00107
           yield();
00109
         \ensuremath{//} Check if the process has been killed since we yielded
00110
         if(myproc() && myproc()->killed && (tf->cs&3) == DPL_USER)
00111
           exit();
00112 }
```

5.218 trap.d File Reference

5.219 trap.d

```
Go to the documentation of this file.

00001 trap.o: trap.c /usr/include/stdc-predef.h types.h defs.h param.h \
00002 memlayout.h mmu.h proc.h x86.h traps.h spinlock.h
```

5.220 traps.h File Reference

Macros

- #define IRQ_COM1 4
- #define IRQ_ERROR 19
- #define IRQ IDE 14
- #define IRQ KBD 1
- #define IRQ_SPURIOUS 31
- #define IRQ_TIMER 0
- #define T_ALIGN 17
- #define T_BOUND 5
- #define T_BRKPT 3
- #define T_DBLFLT 8
- #define T_DEBUG 1
- #define T_DEFAULT 500
- #define T_DEVICE 7
- #define T_DIVIDE 0
- #define T_FPERR 16
- #define T_GPFLT 13
- #define T ILLOP 6
- #define T_IRQ0 32

```
• #define T_MCHK 18
```

- #define T_NMI 2
- #define T_OFLOW 4
- #define T_PGFLT 14
- #define T_SEGNP 11
- #define T_SIMDERR 19
- #define T_STACK 12
- #define T_SYSCALL 64
- #define T_TSS 10

5.220.1 Macro Definition Documentation

5.220.1.1 IRQ_COM1

```
#define IRQ_COM1 4
```

Definition at line 34 of file traps.h.

5.220.1.2 IRQ_ERROR

#define IRQ_ERROR 19

Definition at line 36 of file traps.h.

5.220.1.3 IRQ_IDE

#define IRQ_IDE 14

Definition at line 35 of file traps.h.

5.220.1.4 IRQ_KBD

#define IRQ_KBD 1

Definition at line 33 of file traps.h.

5.220.1.5 IRQ_SPURIOUS

```
#define IRQ_SPURIOUS 31
```

Definition at line 37 of file traps.h.

5.220.1.6 IRQ_TIMER

```
#define IRQ_TIMER 0
```

Definition at line 32 of file traps.h.

5.220.1.7 T_ALIGN

```
#define T_ALIGN 17
```

Definition at line 21 of file traps.h.

5.220.1.8 T_BOUND

```
#define T_BOUND 5
```

Definition at line 9 of file traps.h.

5.220.1.9 T_BRKPT

```
#define T_BRKPT 3
```

Definition at line 7 of file traps.h.

5.220.1.10 T_DBLFLT

```
#define T_DBLFLT 8
```

Definition at line 12 of file traps.h.

5.220.1.11 T_DEBUG

```
#define T_DEBUG 1
```

Definition at line 5 of file traps.h.

5.220.1.12 T_DEFAULT

```
#define T_DEFAULT 500
```

Definition at line 28 of file traps.h.

5.220.1.13 T_DEVICE

```
#define T_DEVICE 7
```

Definition at line 11 of file traps.h.

5.220.1.14 T_DIVIDE

```
#define T_DIVIDE 0
```

Definition at line 4 of file traps.h.

5.220.1.15 T_FPERR

```
#define T_FPERR 16
```

Definition at line 20 of file traps.h.

5.220.1.16 T_GPFLT

#define T_GPFLT 13

Definition at line 17 of file traps.h.

5.220.1.17 T_ILLOP

#define T_ILLOP 6

Definition at line 10 of file traps.h.

5.220.1.18 T_IRQ0

#define T_IRQ0 32

Definition at line 30 of file traps.h.

5.220.1.19 T_MCHK

#define T_MCHK 18

Definition at line 22 of file traps.h.

5.220.1.20 T_NMI

#define T_NMI 2

Definition at line 6 of file traps.h.

5.220.1.21 T OFLOW

#define T_OFLOW 4

Definition at line 8 of file traps.h.

5.220.1.22 T_PGFLT

#define T_PGFLT 14

Definition at line 18 of file traps.h.

5.220.1.23 T_SEGNP

```
#define T_SEGNP 11
```

Definition at line 15 of file traps.h.

5.220.1.24 T_SIMDERR

```
#define T_SIMDERR 19
```

Definition at line 23 of file traps.h.

5.220.1.25 T_STACK

```
#define T_STACK 12
```

Definition at line 16 of file traps.h.

5.220.1.26 T_SYSCALL

#define T_SYSCALL 64

Definition at line 27 of file traps.h.

5.220.1.27 T_TSS

#define T_TSS 10

Definition at line 14 of file traps.h.

5.221 traps.h 435

5.221 traps.h

```
Go to the documentation of this file.
```

```
00001 // x86 trap and interrupt constants.
 00002
 00003 // Processor-defined:
                                                                                                                               // divide error
// debug exception
 00004 #define T_DIVIDE
 00005 #define T_DEBUG
 00006 #define T_NMI
                                                                                                                                   // non-maskable interrupt
 00007 #define T_BRKPT
                                                                                                                                   // breakpoint
00008 #define T_BRKPT
00008 #define T_OFLOW
00009 #define T_BOUND
00010 #define T_ILLOP
00011 #defire T_TT
                                                                                                                                  // overflow
                                                                                                                               // overilow
// bounds check
// illegal opcode
                                                                                                                              // illegal opcode
// device not available
// double fault
// reserved (not used since 486)
// invalid task switch segment
 00011 #define T_DEVICE
00012 #define T_DBLFLT
00013 // #define T_COPROC
00014 #define T_TSS
00015 #define T_SEGNP
                                                                                                           9
                                                                                                        10
                                                                                                                                  // segment not present
                                                                                                         11
00015 #define T_SEGNP 11  // segment not present  
00016 #define T_STACK 12  // stack exception  
00017 #define T_GPFLT 13  // general protection fault  
00018 #define T_PGFLT 14  // page fault  
00019 // #define T_RES 15  // reserved  
00020 #define T_FPERR 16  // floating point error  
00021 #define T_ALIGN 17  // aligment check  
00022 #define T_MCHK 18  // machine check  
00023 #define T_SIMDERR 19  // SIMD floating point error  
00024  
00024 #define T_SIMDERR 19  // SIMD floating point error  
00024  
00025 #define T_SIMDERR 19  // SIMD floating point error  
00026 #define T_SIMDERR 19  // SIMD floating point error  
00027 #define T_SIMDERR 19  // SIMD floating point error  
00028 #define T_SIMDERR 19  // SIMD floating point error  
00029 #define T_SIMDERR 19  // SIMD floating point error  
00020 #define T_SIMDERR 19  // SIMD floating point error  
00020 #define T_SIMDERR 19  // SIMD floating point error  
00020 #define T_SIMDERR 19  // SIMD floating point error  
00020 #define T_SIMDERR 19  // SIMD floating point error  
00020 #define T_SIMDERR 19  // SIMD floating point error  
00020 #define T_SIMDERR 19  // SIMD floating point error  
00020 #define T_SIMDERR 19  // SIMD floating point error  
00020 #define T_SIMDERR 19  // SIMDER  
00020 #define T_SIMDERR   
0002
 00024
 00025 // These are arbitrarily chosen, but with care not to overlap
 00026 // processor defined exceptions or interrupt vectors.
 00027 #define T_SYSCALL 64
00028 #define T_DEFAULT 500
                                                                                                                                 // system call 
// catchall
 00029
  00030 #define T_IRQ0 32 // IRQ 0 corresponds to int T_IRQ
 00031
 00032 #define IRQ_TIMER
                                                                                                           0
 00033 #define IRQ_KBD
 00034 #define IRQ_COM1
 00035 #define IRQ_IDE
 00036 #define IRQ_ERROR
 00037 #define IRQ_SPURIOUS
 00038
```

5.222 types.h File Reference

Typedefs

- typedef uint pde_t
- · typedef unsigned char uchar
- typedef unsigned int uint
- · typedef unsigned short ushort

5.222.1 Typedef Documentation

5.222.1.1 pde_t

```
typedef uint pde_t
```

Definition at line 4 of file types.h.

5.222.1.2 uchar

```
typedef unsigned char uchar
```

Definition at line 3 of file types.h.

5.222.1.3 uint

```
typedef unsigned int uint
```

Definition at line 1 of file types.h.

5.222.1.4 ushort

```
typedef unsigned short ushort
```

Definition at line 2 of file types.h.

5.223 types.h

Go to the documentation of this file.

```
00001 typedef unsigned int uint;
00002 typedef unsigned short ushort;
00003 typedef unsigned char uchar;
00004 typedef uint pde_t;
```

5.224 uart.c File Reference

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "traps.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "fs.h"
#include "file.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
```

Macros

• #define COM1 0x3f8

5.224 uart.c File Reference 437

Functions

- static int uartgetc (void)
- void uartinit (void)
- void uartintr (void)
- void uartputc (int c)

Variables

· static int uart

5.224.1 Macro Definition Documentation

5.224.1.1 COM1

```
#define COM1 0x3f8
```

Definition at line 15 of file uart.c.

5.224.2 Function Documentation

5.224.2.1 uartgetc()

Definition at line 64 of file uart.c.

```
00065 {
00066    if(!uart)
00067    return -1;
00068    if(!(inb(COM1+5) & 0x01))
00069    return -1;
00070    return inb(COM1+0);
00071 }
```

Referenced by uartintr().

5.224.2.2 uartinit()

```
void uartinit (
            void )
Definition at line 20 of file uart.c.
00022
00023
       // Turn off the FIFO
outb(COM1+2, 0);
00024
00025
00026
       00027
00028
00029
00030
       outb(COM1+1, 0);
       outb(COM1+3, 0x03);
outb(COM1+4, 0);
00031
                            // Lock divisor, 8 data bits.
00032
00033
       outb(COM1+1, 0x01);
                            // Enable receive interrupts.
00034
00035
       // If status is 0xFF, no serial port.
       if(inb(COM1+5) == 0xFF)
00036
       return;
uart = 1;
00037
00038
00039
00040
       // Acknowledge pre-existing interrupt conditions;
00041
       // enable interrupts.
00042
       inb(COM1+2);
       inb(COM1+0);
00043
00044
       ioapicenable(IRQ_COM1, 0);
00045
00046
       // Announce that we're here.
00047
       for (p="xv6...\n"; *p; p++)
00048
       uartputc(*p);
```

5.224.2.3 uartintr()

00049 }

```
void uartintr (
     void )
```

Definition at line 74 of file uart.c.

```
00075 {
00076      consoleintr(uartgetc);
00077 }
```

Referenced by trap().

5.224.2.4 uartputc()

```
void uartputc ( \quad \text{int } c \ )
```

Definition at line 52 of file uart.c.

Referenced by consputc(), and uartinit().

5.225 uart.c 439

5.224.3 Variable Documentation

5.224.3.1 uart

```
int uart [static]
```

Definition at line 17 of file uart.c.

Referenced by uartgetc(), uartinit(), and uartputc().

5.225 uart.c

Go to the documentation of this file.

```
00001 // Intel 8250 serial port (UART).
00002
00003 #include "types.h"
00004 #include "defs.h"
00005 #include "param.h"
00006 #include "traps.h"
00007 #include "spinlock.h"
00008 #include "sleeplock.h"
00009 #include "fs.h"
00010 #include "file.h"
00011 #include "mmu.h"
00012 #include "proc.h"
00012 #Include proc.n
00014
00015 #define COM1 0x3f8
00016
00017 static int uart; // is there a uart?
00018
00019 void
00020 uartinit(void)
00021 {
00022
        char *p;
00023
        // Turn off the FIFO
00024
00025
        outb(COM1+2, 0);
00026
00027
        // 9600 baud, 8 data bits, 1 stop bit, parity off.
        outb(COM1+3, 0x80); // Unlock divisor
outb(COM1+0, 115200/9600);
00028
00029
         outb(COM1+1, 0);
00030
        outb(COM1+3, 0x03);
outb(COM1+4, 0);
                                 // Lock divisor, 8 data bits.
00031
00032
00033
        outb(COM1+1, 0x01); // Enable receive interrupts.
00034
00035
         // If status is 0xFF, no serial port.
        if(inb(COM1+5) == 0xFF)
00036
00037
           return:
00038
        uart = 1;
00039
00040
        // Acknowledge pre-existing interrupt conditions;
00041
         // enable interrupts.
00042
        inb(COM1+2);
00043
        inb(COM1+0);
00044
         ioapicenable(IRQ_COM1, 0);
00045
00046
         // Announce that we're here.
00047
         for (p="xv6...\n"; *p; p++)
00048
           uartputc(*p);
00049 }
00050
00051 void
00052 uartputc(int c)
00053 {
00054
         int i;
00055
00056
        if(!uart)
00057
00058
        for(i = 0; i < 128 && !(inb(COM1+5) & 0x20); i++)</pre>
```

```
microdelay(10);
00060 outb(COM1+0, c);
00061 }
00062
00063 static int
00064 uartgetc(void)
00065 {
00066
      if(!uart)
00067
          return -1;
       if(!(inb(COM1+5) & 0x01))
00068
         return -1;
00069
00070 return inb(COM1+0);
00071 }
00072
00073 void
00074 uartintr(void)
00075 {
00076 consoleintr(uartgetc);
00077 }
```

5.226 uart.d File Reference

5.227 uart.d

```
Go to the documentation of this file.
```

```
00001 wart.o: wart.c /wsr/include/stdc-predef.h types.h defs.h param.h traps.h \ 00002 spinlock.h sleeplock.h fs.h file.h mmu.h proc.h x86.h
```

5.228 ulib.c File Reference

```
#include "types.h"
#include "stat.h"
#include "fcntl.h"
#include "user.h"
#include "x86.h"
```

Functions

```
• int atoi (const char *s)
```

- char * gets (char *buf, int max)
- void * memmove (void *vdst, const void *vsrc, int n)
- void * memset (void *dst, int c, uint n)
- int stat (const char *n, struct stat *st)
- char * strchr (const char *s, char c)
- int strcmp (const char *p, const char *q)
- char * strcpy (char *s, const char *t)
- uint strlen (const char *s)

5.228.1 Function Documentation

5.228 ulib.c File Reference 441

5.228.1.1 atoi()

```
int atoi ( {\rm const\ char\ *\ s\ )}
```

Definition at line 85 of file ulib.c.

```
00086 {
00087    int n;
00088
00089    n = 0;
00090    while ('0' <= *s && *s <= '9')
00091    n = n*10 + *s++ - '0';
00092    return n;
```

Referenced by main().

5.228.1.2 gets()

Definition at line 53 of file ulib.c.

```
00054 {
00055
          int i, cc;
00056
         char c;
00057
00058 for(i=0; i+1 < max; ){
         cc = read(0, &c, 1);
if(cc < 1)
00059
00060
00061
               break;
         buf(i++) = c;
if(c == '\n' || c == '\r')
break;
00062
00063
00064
00064 break;

00065 }

00066 buf[i] = '\0';

00067 return buf;

00068 }
```

Referenced by getcmd().

5.228.1.3 memmove()

```
void * memmove (  \mbox{void} * \mbox{$vdst$,} \\ \mbox{const void} * \mbox{$vsrc$,} \\ \mbox{int $n$} )
```

Definition at line 96 of file ulib.c.

5.228.1.4 memset()

```
void * memset (  \mbox{void} * \mbox{$dst$,} \\ \mbox{int $c$,} \\ \mbox{uint $n$} )
```

Definition at line 37 of file ulib.c.

```
00038 {
00039 stosb(dst, c, n);
00040 return dst;
00041 }
```

Referenced by allocproc(), allocuvm(), backcmd(), bigfile(), bzero(), cgaputc(), concreate(), execcmd(), fmtname(), fourfiles(), getcmd(), ialloc(), inituvm(), kfree(), listcmd(), main(), pipecmd(), redircmd(), setupkvm(), sys_exec(), sys_unlink(), userinit(), and walkpgdir().

5.228.1.5 stat()

```
int stat (  \mbox{const char} * n, \\ \mbox{struct stat} * st \; )
```

Definition at line 71 of file ulib.c.

```
00072 {
00073    int fd;
00074    int r;
00075
00076    fd = open(n, O_RDONLY);
00077    if(fd < 0)
00078        return -1;
00079    r = fstat(fd, st);
00080    close(fd);
00081    return r;
00082 }</pre>
```

5.228.1.6 strchr()

```
char * strchr (  \mbox{const char * $s$,} \\ \mbox{char $c$ )}
```

Definition at line 44 of file ulib.c.

Referenced by gettoken(), grep(), peek(), and wc().

443 5.228 ulib.c File Reference

5.228.1.7 strcmp()

```
int strcmp (
           const char *p,
           const char *q)
```

Definition at line 19 of file ulib.c.

```
00020 {
00021    while(*p && *p == *q)
00022    p++, q++;
00023    return (uchar)*p - (uchar)*q;
00024 }
```

5.228.1.8 strcpy()

```
char * strcpy (
            char * s,
            const char * t )
```

Definition at line 8 of file ulib.c.

```
00009 {
00010
           char *os;
00011
00012 os = s;

00013 while((*s++ = *t++) != 0)

00014 ;

00015 return os;
00016 }
```

Referenced by Is(), and main().

5.228.1.9 strlen()

```
uint strlen (
          const char *s)
```

Definition at line 27 of file ulib.c.

```
Dem...
00028 {
00029 int n;
00029
00031
        for(n = 0; s[n]; n++)
00032
00033 return n;
00034 }
```

Referenced by exec(), fmtname(), ls(), main(), parsecmd(), and printf().

5.229 ulib.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "fcntl.h"
00004 #include "user.h"
00005 #include "x86.h"
00006
00007 char*
00008 strcpy(char *s, const char *t)
00009 {
00010 char *os;
00011
00012 os = s;
00013 while((*s++ = *t++) != 0)
00014 ;
          ;
00015 return os;
00016 }
00017
00018 int
00019 strcmp(const char *p, const char *q) 00020 {
00021 while (*p && *p == *q)
00022 p++, q++;
- while (*p && *p == *q)
00022 p++, q++;
00023 return (uchar) *p - (uchar) *q;
00024 }
00025
00026 uint
00027 strlen(const char *s)
00028 {
00029
       int n;
00030
00031 for (n = 0; s[n]; n++)
00032
00033 return n;
00034 }
00035
00036 void*
00037 memset(void *dst, int c, uint n)
00038 {
00039 stosb(dst, c, n);
00040 return dst;
00041 }
00042
00043 char*
00044 strchr(const char *s, char c)
00045 {
00046 for(; *s; s++)
00047 if(*s == c)
00048 return (char*)s;
00049
         return 0;
00050 }
00051
00052 char*
00053 gets(char *buf, int max)
00054 {
00055 int i, cc;
00056 char c;
00057
         for(i=0; i+1 < max; ) {</pre>
00058
         cc = read(0, &c, 1);
if(cc < 1)
00059
00060
00061
              break;
         buf[i++] = c;
if(c == '\n' || c == '\r')
00062
00063
00064
             break;
00065
00066 buf[i] = '\0';
00067 return buf;
00068 }
00069
00070 int
00071 stat(const char *n, struct stat *st)
00072 {
00073
         int fd;
00074 int r;
00075
         fd = open(n, O_RDONLY);
if(fd < 0)</pre>
00076
00077
         return -1;
r = fstat(fd, st);
00078
00079
08000
         close(fd);
00081 return r;
00082 }
```

5.230 ulib.d File Reference 445

```
00083
00084 int
00085 atoi(const char *s)
00086 {
00087
00088
00090 while('0' <= *s && *s <= '9')
00091 n = n*10 + *s++ - '0';
00092 return n;
00093 }
00094
00095 void*
00096 memmove (void *vdst, const void *vsrc, int n)
00097 {
00098 char *dst;
00099 const char *src;
00100
00100

00101 dst = vdst;

00102 src = vsrc;

00103 while(n-- > 0)

00104 *dst++ = *src++;
00104 *dst++ = *3
00105 return vdst;
00106 }
```

5.230 ulib.d File Reference

5.231 ulib.d

```
Go to the documentation of this file.
```

```
00001 ulib.o: ulib.c /usr/include/stdc-predef.h types.h stat.h fcntl.h user.h \setminus 00002 x86.h
```

5.232 umalloc.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "param.h"
```

Classes

• union header

Typedefs

- typedef long Align
- typedef union header Header

Functions

- void free (void *ap)
- void * malloc (uint nbytes)
- static Header * morecore (uint nu)

Variables

- · static Header base
- static Header * freep

5.232.1 Typedef Documentation

5.232.1.1 Align

```
typedef long Align
```

Definition at line 9 of file umalloc.c.

5.232.1.2 Header

```
typedef union header Header
```

Definition at line 19 of file umalloc.c.

5.232.2 Function Documentation

5.232.2.1 free()

```
void free (
     void * ap )
```

Definition at line 25 of file umalloc.c.

Referenced by mem(), and morecore().

5.232.2.2 malloc()

```
void * malloc (
                uint nbytes )
Definition at line 64 of file umalloc.c.
00065 4
00066
         Header *p, *prevp;
00067
        uint nunits;
00068
00069
         nunits = (nbytes + sizeof(Header) - 1)/sizeof(Header) + 1;
00070
        if((prevp = freep) == 0){
         base.s.ptr = freep = prevp = &base;
base.s.size = 0;
00071
00072
00073
00074
        for(p = prevp->s.ptr; ; prevp = p, p = p->s.ptr) {
          if(p->s.size >= nunits){
   if(p->s.size == nunits)
     prevp->s.ptr = p->s.ptr;
00075
00076
00077
00078
             else {
00079
             p->s.size -= nunits;
08000
               p += p->s.size;
00081
              p->s.size = nunits;
00082
             freep = prevp;
return (void*) (p + 1);
00083
00084
00085
           if(p == freep)
00087
             if((p = morecore(nunits)) == 0)
00088
               return 0;
00089
```

Referenced by backcmd(), execcmd(), listcmd(), main(), mem(), pipecmd(), and redircmd().

5.232.2.3 morecore()

00090 }

```
static Header * morecore (
          uint nu ) [static]
```

Definition at line 47 of file umalloc.c.

```
00048 {
00049
        char *p;
00050
       Header *hp;
00051
00052
       if(nu < 4096)
00053
         nu = 4096;
00054 p = sbrk(nu * sizeof(Header));
       if (p == (char*)-1)
  return 0;
00055
00056
       hp = (Header*)p;
00057
00058
        hp->s.size = nu;
00059
        free((void*)(hp + 1));
00060
        return freep;
00061 }
```

Referenced by malloc().

5.232.3 Variable Documentation

5.232.3.1 base

```
Header base [static]
```

Definition at line 21 of file umalloc.c.

Referenced by malloc(), and printint().

5.232.3.2 freep

```
Header* freep [static]
```

Definition at line 22 of file umalloc.c.

Referenced by free(), malloc(), and morecore().

5.233 umalloc.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004 #include "param.h"
00006 // Memory allocator by Kernighan and Ritchie,
00007 // The C programming Language, 2nd ed. Section 8.7.
80000
00009 typedef long Align;
00010
00011 union header {
00012 struct {
00013
         union header *ptr;
00014
          uint size;
00015 } s;
00016
        Align x;
00017 };
00018
00019 typedef union header Header;
00020
00021 static Header base;
00022 static Header *freep;
00023
00024 void
00025 free(void *ap)
00026 {
00027
        Header *bp, *p;
00028
00029
        bp = (Header*)ap - 1;
        for(p = freep; !(bp > p && bp < p->s.ptr); p = p->s.ptr)
if(p >= p->s.ptr && (bp > p || bp < p->s.ptr))
00031
00032
             break;
        if (bp + bp->s.size == p->s.ptr) {
  bp->s.size += p->s.ptr->s.size;
  bp->s.ptr = p->s.ptr->s.ptr;
00033
00034
00035
00036
        } else
00037
          bp->s.ptr = p->s.ptr;
00038
        if(p + p->s.size == bp){
         p->s.size += bp->s.size;
p->s.ptr = bp->s.ptr;
00039
00040
00041
        } else
00042
          p->s.ptr = bp;
00043 freep = p;
00044 }
00045
00046 static Header*
00047 morecore(uint nu)
00048 {
00049
         char *p;
```

```
Header *hp;
00051
00052
       if(nu < 4096)
00053
        nu = 4096;
00054 p = sbrk(nu * sizeof(Header));
       if (p == (char*)-1)
00055
         return 0;
00057 hp = (Header*)p;
00058 hp->s.size = nu;
00059
       free((void*)(hp + 1));
       return freep;
00060
00061 }
00062
00063 void*
00064 malloc(uint nbytes)
00065 {
       Header *p, *prevp;
00066
00067
       uint nunits;
       nunits = (nbytes + sizeof(Header) - 1)/sizeof(Header) + 1;
00070
       if((prevp = freep) == 0){
        base.s.ptr = freep = prevp = &base;
00071
00072
         base.s.size = 0;
00073
00074
       for(p = prevp->s.ptr; ; prevp = p, p = p->s.ptr) {
00075
        if (p->s.size >= nunits) {
00076
           if(p->s.size == nunits)
00077
             prevp->s.ptr = p->s.ptr;
00078
            p->s.size -= nunits;
00079
08000
           p -> p->s.size;
p->s.size = nunits;
}
              p += p->s.size;
00081
00082
           freep = prevp;
return (void*) (p + 1);
00083
00084
00085
00086
         if(p == freep)
if((p = morecore(nunits)) == 0)
00088
              return 0;
00089 }
00090 }
```

5.234 umalloc.d File Reference

5.235 umalloc.d

Go to the documentation of this file.

```
00001 umalloc.o: umalloc.c /usr/include/stdc-predef.h types.h stat.h user.h \backslash 00002 param.h
```

5.236 uproc.h File Reference

Classes

struct uproc

5.237 uproc.h

Go to the documentation of this file.

```
00001 struct uproc {
00002    int pid;
00003    int ppid;
00004    char name[16];
00005 };
```

5.238 user.h File Reference

Functions

```
• int atoi (const char *)
• int chdir (const char *)
• int chpr (int pid, int priority)
• int close (int)
• int cps (void)
• int dup (int)
• int exec (char *, char **)
• int exit (void) __attribute__((noreturn))

    int fork (void)

void free (void *)
• int fstat (int fd, struct stat *)
• int getpid (void)
• int getprocs (int max, struct uproc *)
char * gets (char *, int max)
• int halt (void)
• int kill (int)
• int link (const char *, const char *)

    void * malloc (uint)

    void * memmove (void *, const void *, int)

void * memset (void *, int, uint)
• int mkdir (const char *)
• int mknod (const char *, short, short)
• int open (const char *, int)
int pipe (int *)

    void printf (int, const char *,...)

• int read (int, void *, int)
char * sbrk (int)
• int sleep (int)
• int stat (const char *, struct stat *)
• char * strchr (const char *, char c)
• int strcmp (const char *, const char *)
• char * strcpy (char *, const char *)
• uint strlen (const char *)
• int unlink (const char *)
• int uptime (void)
• int wait (void)
• int write (int, const void *, int)
```

5.238.1 Function Documentation

5.238 user.h File Reference 451

5.238.1.1 atoi()

Referenced by main().

5.238.1.2 chdir()

```
int chdir ( \mbox{const char} \ * \ \ )
```

Referenced by dirfile(), dirtest(), exitiputtest(), iputtest(), iref(), main(), rmdot(), and subdir().

5.238.1.3 chpr()

Definition at line 559 of file proc.c.

```
00560 {
00561
        struct proc *p;
00562
        acquire(&ptable.lock);
00563
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
        if(p->pid == pid) {
00564
00565
          p->priority = priority;
00566
            break;
00567
         }
00568
        release(&ptable.lock);
00570
       return pid;
00571 }
```

5.238.1.4 close()

```
int close (
          int )
```

Referenced by argptest(), bigargtest(), bigdir(), bigdir(), bigwrite(), concreate(), createdelete(), createtest(), dirfile(), fourfiles(), fourteen(), fsfull(), iref(), linktest(), linkunlink(), ls(), main(), opentest(), pipe1(), preempt(), runcmd(), sharedfd(), stat(), subdir(), unlinkread(), writetest(), and writetest1().

5.238.1.5 cps()

```
int cps ( void )
```

Definition at line 537 of file proc.c.

```
00539 struct proc *p;
00540 //Enables interrupts on this processor.
00541 sti();
00542
00543 //Loop over process table looking for process with pid.
00544 acquire(&ptable.lock);
00545 cprintf("name \t pid \t state \t priority \n");
00546 for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
00547    if(p->state == SLEEPING)
           cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name,p->pid,p->priority);
00549
        else if(p->state == RUNNING)
00550
         cprintf("%s \t %d \t RUNNING \t %d \n ", p->name,p->pid,p->priority);
        else if(p->state == RUNNABLE)
    cprintf("%s \t %d \t RUNNABLE \t %d \n ", p->name,p->pid,p->priority);
00551
00552
00553 }
00554 release(&ptable.lock);
00555 return 23;
00556 }
```

5.238.1.6 dup()

```
int dup (
    int )
```

Referenced by main(), and runcmd().

5.238.1.7 exec()

Definition at line 11 of file exec.c.

```
00012 {
00013
        char *s, *last;
       int i, off;
uint argc, sz, sp, ustack[3+MAXARG+1];
struct elfhdr elf;
00014
00015
00016
00017
        struct inode *ip;
00018
        struct proghdr ph;
00019
        pde_t *pgdir, *oldpgdir;
        struct proc *curproc = myproc();
00020
00021
00022
        begin_op();
00023
00024
        if((ip = namei(path)) == 0){
        end_op();
cprintf("exec: fail\n");
00025
00026
00027
         return -1:
00028
00029
        ilock(ip);
00030
       pgdir = 0;
00031
        // Check ELF header
00032
00033
        if(readi(ip, (char*)&elf, 0, sizeof(elf)) != sizeof(elf))
00034
          goto bad;
00035
        if(elf.magic != ELF_MAGIC)
00036
        goto bad;
```

```
00038
        if((pgdir = setupkvm()) == 0)
00039
          goto bad;
00040
00041
        \ensuremath{//} Load program into memory.
00042
        for(i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){</pre>
00044
          if(readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
00045
             goto bad;
           if (ph.type != ELF_PROG_LOAD)
00046
00047
            continue;
          if(ph.memsz < ph.filesz)</pre>
00048
00049
            goto bad;
00050
           if (ph.vaddr + ph.memsz < ph.vaddr)</pre>
00051
            goto bad;
00052
           if((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
00053
            goto bad;
00054
          if(ph.vaddr % PGSIZE != 0)
            goto bad;
00056
           if(loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz) < 0)</pre>
00057
00058
00059
        iunlockput(ip);
00060
        end_op();
00061
        ip = 0;
00062
00063
        // Allocate two pages at the next page boundary.
00064
        \ensuremath{//} Make the first inaccessible. Use the second as the user stack.
00065
        sz = PGROUNDUP(sz);
00066
        if((sz = allocuvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
          goto bad;
00067
00068
        clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
00069
00070
00071
         // Push argument strings, prepare rest of stack in ustack.
        for(argc = 0; argv[argc]; argc++) {
  if(argc >= MAXARG)
00072
00073
            goto bad;
00075
          sp = (sp - (strlen(argv[argc]) + 1)) & ~3;
00076
          if(copyout(pgdir, sp, argv[argc], strlen(argv[argc]) + 1) < 0)</pre>
00077
             goto bad;
00078
          ustack[3+argc] = sp;
00079
08000
        ustack[3+argc] = 0;
00081
00082
        ustack[0] = 0xfffffffff; // fake return PC
        ustack[1] = argc;
ustack[2] = sp - (argc+1)*4; // argv pointer
00083
00084
00085
00086
        sp -= (3+argc+1) * 4;
00087
        if(copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)</pre>
00088
          goto bad;
00089
00090
        // Save program name for debugging.
00091
        for(last=s=path; *s; s++)
  if(*s == '/')
00092
00093
             last = s+1;
00094
        safestrcpy(curproc->name, last, sizeof(curproc->name));
00095
00096
        // Commit to the user image.
00097
        oldpgdir = curproc->pgdir;
00098
        curproc->pgdir = pgdir;
00099
        curproc->sz = sz;
00100
        curproc->tf->eip = elf.entry; // main
        curproc->tf->esp = sp;
00101
00102
        switchuvm(curproc);
00103
        freevm(oldpgdir);
00104
        return 0:
00105
00106 bad:
00107
       if(pgdir)
00108
          freevm(pgdir);
00109
        if(ip){
          iunlockput(ip);
00110
00111
          end_op();
00112
00113
        return -1;
00114 }
```

5.238.1.8 exit()

```
int exit (
```

void)

```
Definition at line 228 of file proc.c.
```

```
00229 +
00230
         struct proc *curproc = myproc();
00231
         struct proc *p;
00232
         int fd;
00233
00234
        if(curproc == initproc)
00235
         panic("init exiting");
00236
         // Close all open files.
00237
00238
        for(fd = 0; fd < NOFILE; fd++) {</pre>
00239
         if(curproc->ofile[fd]){
00240
              fileclose(curproc->ofile[fd]);
00241
             curproc->ofile[fd] = 0;
00242
          }
00243
00244
00245
         begin_op();
00246
         iput (curproc->cwd);
00247
         end_op();
        curproc->cwd = 0;
00248
00249
00250
        acquire(&ptable.lock);
00251
00252
         // Parent might be sleeping in wait().
00253
         wakeup1 (curproc->parent);
00254
00255
         \ensuremath{//} Pass abandoned children to init.
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
   if(p->parent == curproc) {
     p->parent = initproc;
    if(p->state == ZOMBIE)
}
00256
00257
00258
00259
00260
               wakeup1(initproc);
00261
          }
00262
00263
00264
         // Jump into the scheduler, never to return.
00265
         curproc->state = ZOMBIE;
00266
        sched();
00267
         panic("zombie exit");
00268 }
```

5.238.1.9 fork()

```
int fork (
     void )
```

Definition at line 181 of file proc.c.

```
00182 {
00183
         int i, pid;
00184
         struct proc *np;
00185
         struct proc *curproc = myproc();
00186
        // Allocate process.
if((np = allocproc()) == 0){
00187
00188
00189
          return -1;
00190
00191
         // Copy process state from proc.
if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
00192
00193
          kfree(np->kstack);
00194
          np->kstack = 0;
np->state = UNUSED;
00195
00196
00197
          return -1;
00198
00199
        np->sz = curproc->sz;
00200
        np->parent = curproc;
*np->tf = *curproc->tf;
00201
00202
00203
         // Clear %eax so that fork returns 0 in the child.
00204
        np->tf->eax = 0;
00205
00206
         for(i = 0; i < NOFILE; i++)</pre>
00207
         if(curproc->ofile[i])
00208
             np->ofile[i] = filedup(curproc->ofile[i]);
```

```
np->cwd = idup(curproc->cwd);
00210
00211
       safestrcpy(np->name, curproc->name, sizeof(curproc->name));
00212
00213
       pid = np->pid;
00214
00215
       acquire(&ptable.lock);
00216
00217
       np->state = RUNNABLE;
00218
00219
       release(&ptable.lock);
00220
00221
       return pid;
00222 }
```

5.238.1.10 free()

```
void free ( void * ap )
```

Definition at line 25 of file umalloc.c.

```
00026 {
00027
           Header *bp, *p;
00028
          bp = (Header*)ap - 1;
for(p = freep; !(bp > p && bp < p->s.ptr); p = p->s.ptr)
  if(p >= p->s.ptr && (bp > p || bp < p->s.ptr))
00029
00030
00031
00032
                break;
          if (bp + bp->s.size == p->s.ptr) {
    bp->s.size += p->s.ptr->s.size;
    bp->s.size += p->s.ptr->s.size;
00033
00034
00035
             bp->s.ptr = p->s.ptr->s.ptr;
00036 } else
00037
             bp->s.ptr = p->s.ptr;
00038
          if (p + p->s.size == bp) {
           p->s.size += bp->s.size;
p->s.ptr = bp->s.ptr;
00039
00040
00041
          } else
00042
             p->s.ptr = bp;
00043 freep = p;
00044 }
```

Referenced by mem(), and morecore().

5.238.1.11 fstat()

Referenced by Is(), and stat().

5.238.1.12 getpid()

```
int getpid ( void )
```

Referenced by main(), mem(), and sbrktest().

5.238.1.13 getprocs()

```
int getprocs (
                int max,
                struct uproc * )
```

5.238.1.14 gets()

Definition at line 53 of file ulib.c.

```
00054 {
00055
         int i, cc;
00056
         char c;
00057
00058
         for(i=0; i+1 < max; ) {</pre>
00059
         cc = read(0, &c, 1);
00060
           <u>if</u>(cc < 1)
         break;
buf[i++] = c;
if(c == '\n' || c == '\r')
00061
00062
00063
00064
             break;
00065
00066
        buf[i] = ' \setminus 0';
00067
         return buf;
00068 }
```

Referenced by getcmd().

5.238.1.15 halt()

```
int halt (
     void )
```

Referenced by main().

5.238.1.16 kill()

```
int kill ( \quad \quad \text{int } pid \ )
```

Definition at line 480 of file proc.c.

```
00481 {
00482
          struct proc *p;
00483
00484
           acquire(&ptable.lock);
00485
           for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {</pre>
           if(p->pid == pid) {
  p->killed = 1;
  // Wake process from sleep if necessary.
  if(p->state == SLEEPING)
   p->state = RUNNABLE;
00486
00487
00488
00489
00490
00491
                release(&ptable.lock);
00492
                return 0;
00493
            }
00494
00495
          release(&ptable.lock);
return -1;
00496
00497 }
```

5.238.1.17 link()

Referenced by bigdir(), concreate(), dirfile(), iref(), linktest(), linkunlink(), main(), subdir(), and validatetest().

5.238.1.18 malloc()

```
void * malloc (
          uint nbytes )
```

Definition at line 64 of file umalloc.c.

```
00066
        Header *p, *prevp;
00067
        uint nunits;
00068
00069
        nunits = (nbytes + sizeof(Header) - 1)/sizeof(Header) + 1;
        if((prevp = freep) == 0) {
  base.s.ptr = freep = prevp = &base;
00070
00071
00072
          base.s.size = 0;
00073
00074
        for(p = prevp->s.ptr; ; prevp = p, p = p->s.ptr) {
00075
         if(p->s.size >= nunits) {
  if(p->s.size == nunits)
00076
00077
               prevp->s.ptr = p->s.ptr;
00078
00079
             p->s.size -= nunits;
08000
             p-/s.size;
p->s.size = nunits;
}
00081
00082
00083
             freep = prevp;
00084
             return (void*) (p + 1);
00085
00086
           if(p == freep)
00087
             if((p = morecore(nunits)) == 0)
00088
               return 0;
00089
00090 }
```

Referenced by backcmd(), execcmd(), listcmd(), main(), mem(), pipecmd(), and redircmd().

5.238.1.19 memmove()

Definition at line 96 of file ulib.c.

5.238.1.20 memset()

```
void * memset (  \mbox{void} * \mbox{$dst$,} \\ \mbox{int $c$,} \\ \mbox{uint $n$} )
```

Definition at line 5 of file string.c.

5.238.1.21 mkdir()

```
int mkdir ( \label{eq:const_char*} \mbox{const_char} * \mbox{ )}
```

Referenced by dirfile(), dirtest(), exitiputtest(), fourteen(), iputtest(), iref(), main(), openiputtest(), rmdot(), and subdir().

5.238.1.22 mknod()

Referenced by main().

5.238.1.23 open()

```
int open ( \label{eq:const_char} \mbox{const_char} \ *\ \mbox{,} int )
```

Referenced by argptest(), bigargtest(), bigdir(), bigdir(), bigwrite(), concreate(), createdelete(), createdelete(), dirfile(), fourfiles(), fourfiles(), fourfiles(), iref(), linktest(), linkunlink(), ls(), main(), openiputtest(), opentest(), runcmd(), sharedfd(), stat(), subdir(), unlinkread(), writetest(), and writetest1().

5.238 user.h File Reference 459

5.238.1.24 pipe()

```
int pipe (
    int * )
```

5.238.1.25 printf()

```
void printf (
          int fd,
          const char * s,
          ... )
```

Definition at line 11 of file forktest.c.

```
00012 {
00013 write(fd, s, strlen(s));
00014 }
```

Referenced by argptest(), balloc(), bigargtest(), bigdir(), bigdir(), bigwrite(), bsstest(), cat(), concreate(), createdelete(), createdelete(), dirfile(), dirtest(), exectest(), exitiputtest(), exitwait(), forktest(), fourfiles(), fourteen(), fsfull(), getcmd(), iputtest(), iref(), linktest(), linkunlink(), ls(), main(), mem(), openiputtest(), opentest(), panic(), parsecmd(), pipe1(), preempt(), rmdot(), runcmd(), sbrktest(), sharedfd(), subdir(), uio(), unlinkread(), validatetest(), wc(), writetest(), and writetest1().

5.238.1.26 read()

```
int read (
          int ,
          void * ,
          int )
```

Referenced by argptest(), bigfile(), cat(), concreate(), fourfiles(), gets(), grep(), linktest(), ls(), main(), pipe1(), preempt(), rsect(), sbrktest(), sharedfd(), subdir(), unlinkread(), wc(), writetest(), and writetest1().

5.238.1.27 sbrk()

```
char * sbrk (
```

Referenced by argptest(), morecore(), and sbrktest().

5.238.1.28 sleep()

```
int sleep (
          int )
```

5.238.1.29 stat()

```
00073 int fd;

00074 int r;

00075

00076 fd = open(n, O_RDONLY);

00077 if(fd < 0)

00078 return -1;

00079 r = fstat(fd, st);

00080 close(fd);

00081 return r;
```

5.238.1.30 strchr()

```
\label{eq:char_strekn} \begin{array}{c} \text{char * strchr (} \\ \text{const char * $s$,} \\ \text{char $c$ )} \end{array}
```

Definition at line 44 of file ulib.c.

Referenced by gettoken(), grep(), peek(), and wc().

5.238.1.31 strcmp()

```
int strcmp (  \mbox{const char} \ * \ p, \\ \mbox{const char} \ * \ q \ )
```

Definition at line 19 of file ulib.c.

5.238.1.32 strcpy()

Definition at line 8 of file ulib.c.

```
00009 {
00010    char *os;
00011
00012    os = s;
00013    while((*s++ = *t++) != 0)
00014    ;
00015    return os;
00016 }
```

Referenced by Is(), and main().

5.238.1.33 strlen()

```
uint strlen ( const char * s )
```

Definition at line 97 of file string.c.

```
00098 {
00099    int n;
00100
00101    for(n = 0; s[n]; n++)
00102    ;
00103    return n;
00104 }
```

5.238.1.34 unlink()

```
int unlink ( \mbox{const char} \ * \ \ )
```

Referenced by bigargtest(), bigdir(), bigdir(), bigwrite(), concreate(), createdelete(), createtest(), dirfile(), dirtest(), exitiputtest(), fourfiles(), fsfull(), iputtest(), iref(), linktest(), linkunlink(), main(), openiputtest(), rmdot(), sharedfd(), subdir(), unlinkread(), writetest(), and writetest1().

5.238.1.35 uptime()

```
int uptime ( void )
```

5.238.1.36 wait()

```
int wait (
                 void )
Definition at line 273 of file proc.c.
00275
         struct proc *p;
         int havekids, pid;
struct proc *curproc = myproc();
00276
00277
00278
         acquire(&ptable.lock);
00280
         // Scan through table looking for exited children.
00281
00282
           havekids = 0;
           for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
  if(p->parent != curproc)
    continue;
00283
00284
00286
              havekids = 1;
00287
             if(p->state == ZOMBIE){
00288
               // Found one.
00289
                pid = p->pid;
               kfree(p->kstack);
p->kstack = 0;
00290
00291
00292
                freevm(p->pgdir);
00293
               p->pid = 0;
                p->parent = 0;
00294
               p->name[0] = 0;
p->killed = 0;
00295
00296
00297
               p->state = UNUSED;
00298
                release(&ptable.lock);
00299
                return pid;
00300
00301
           }
00302
00303
           // No point waiting if we don't have any children.
00304
           if(!havekids || curproc->killed){
00305
            release(&ptable.lock);
00306
              return -1;
00307
00308
           // Wait for children to exit. (See wakeup1 call in proc_exit.)
sleep(curproc, &ptable.lock); //DOC: wait-sleep
00309
00310
```

5.238.1.37 write()

00311 00312 }

```
int write (
    int ,
    const void * ,
    int )
```

Referenced by bigfile(), bigwrite(), cat(), dirfile(), fourfiles(), fsfull(), grep(), linktest(), main(), pipe1(), printf(), putc(), sbrktest(), sharedfd(), subdir(), unlinkread(), writetest(), writetest1(), and wsect().

5.239 user.h

Go to the documentation of this file.

```
00001 struct stat;
00002 struct rtcdate;
00003 struct uproc;
00004
00005 // system calls
00006 int fork(void);
00007 int exit(void) __attribute__((noreturn));
00008 int wait(void);
```

```
00009 int pipe(int*);
00010 int write(int, const void*, int);
00011 int read(int, void*, int);
00012 int close(int);
00013 int kill(int);
00014 int exec(char*, char**);
00015 int open(const char*, int);
00016 int mknod(const char*, short, short);
00017 int unlink(const char*);
00018 int fstat(int fd, struct stat*);
00019 int link(const char*, const char*);
00020 int mkdir(const char*);
00021 int chdir(const char*);
00022 int dup(int);
00023 int getpid(void);
00024 char* sbrk(int);
00025 int sleep(int);
00026 int uptime (void);
00027 int halt(void);
00028 int cps(void);
00029 int chpr(int pid, int priority);
00030 int getprocs(int max, struct uproc*);
00031
00032 // ulib.c
00033 int stat(const char*, struct stat*);
00034 char* strcpy(char*, const char*);
00035 void *memmove(void*, const void*, int);
00036 char* strchr(const char*, char c);
00037 int strcmp(const char*, const char*);
00038 void printf(int, const char*, ...);
00039 char* gets(char*, int max);
00040 uint strlen(const char*);
00041 void* memset(void*, int, uint);
00042 void* malloc(uint);
00043 void free(void*);
00044 int atoi(const char*);
```

5.240 usertests.c File Reference

```
#include "param.h"
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fs.h"
#include "fcntl.h"
#include "syscall.h"
#include "traps.h"
#include "memlayout.h"
```

Macros

- #define BIG (100*1024*1024)
- #define RTC_ADDR 0x70
- #define RTC_DATA 0x71

Functions

- · void argptest ()
- · void bigargtest (void)
- · void bigdir (void)
- · void bigfile (void)
- · void bigwrite (void)
- void bsstest (void)

- void concreate (void)
- void createdelete (void)
- · void createtest (void)
- void dirfile (void)
- · void dirtest (void)
- · void exectest (void)
- void exitiputtest (void)
- void exitwait (void)
- · void forktest (void)
- · void fourfiles (void)
- · void fourteen (void)
- void fsfull ()
- · void iputtest (void)
- void iref (void)
- void linktest (void)
- void linkunlink ()
- int main (int argc, char *argv[])
- void mem (void)
- void openiputtest (void)
- void opentest (void)
- void pipe1 (void)
- void preempt (void)
- unsigned int rand ()
- void rmdot (void)
- void sbrktest (void)
- void sharedfd (void)
- void subdir (void)
- void uio ()
- void unlinkread (void)
- void validateint (int *p)
- · void validatetest (void)
- · void writetest (void)
- void writetest1 (void)

Variables

- char buf [8192]
- char * echoargv [] = { "echo", "ALL", "TESTS", "PASSED", 0 }
- char name [3]
- unsigned long randstate = 1
- int stdout = 1
- char uninit [10000]

5.240.1 Macro Definition Documentation

5.240.1.1 BIG

#define BIG (100*1024*1024)

5.240.1.2 RTC_ADDR

```
#define RTC_ADDR 0x70
```

5.240.1.3 RTC DATA

```
#define RTC_DATA 0x71
```

5.240.2 Function Documentation

5.240.2.1 argptest()

```
void argptest ( )
```

Definition at line 1727 of file usertests.c.

```
01728 {
01729    int fd;
01730    fd = open("init", O_RDONLY);
01731    if (fd < 0) {
01732        printf(2, "open failed\n");
01733        exit();
01735    read(fd, sbrk(0) - 1, -1);
01736    close(fd);
01737    printf(1, "arg test passed\n");
01738 }</pre>
```

Referenced by main().

5.240.2.2 bigargtest()

```
void bigargtest (
    void )
```

Definition at line 1613 of file usertests.c.

```
01614 {
01615
           int pid, fd;
01616
           unlink("bigarg-ok");
01617
           pid = fork();
if (pid == 0) {
01618
01619
            static char *args[MAXARG];
int i;
01620
01621
            for(i = 0; i < MAXARG-1; i++)
  args[i] = "bigargs test: failed\n"</pre>
01622
01623
              args[MAXARG-1] = 0;
printf(stdout, "bigarg test\n");
01624
01625
              exec("echo", args);
printf(stdout, "bigarg test ok\n");
fd = open("bigarg-ok", O_CREATE);
close(fd);
01626
01627
01628
01629
01630 exit();
01631 } else if(pid < 0){
```

```
printf(stdout, "bigargtest: fork failed\n");
01633
01634
        wait();
01635
        fd = open("bigarg-ok", 0);
if(fd < 0){</pre>
01636
01637
01638
        printf(stdout, "bigarg test failed!\n");
01639
          exit();
01640
01641
        close(fd);
01642
       unlink("bigarg-ok");
01643 }
```

Referenced by main().

5.240.2.3 bigdir()

```
void bigdir (
     void )
```

Definition at line 893 of file usertests.c.

```
00894 {
00895
              int i, fd:
00896
             char name[10];
             printf(1, "bigdir test\n");
00898
             unlink("bd");
00899
00900
              fd = open("bd", O_CREATE);
if(fd < 0){</pre>
00901
00902
              printf(1, "bigdir create failed\n");
exit();
00903
00904
00905
              close(fd);
00906
00907
             for(i = 0; i < 500; i++){
  name[0] = 'x';
  name[1] = '0' + (i / 64);
  name[2] = '0' + (i % 64);
  name[3] = '\0';
  if(link("bd", name) != 0){
    printf(1, "bigdir link failed\n");
  evit():</pre>
00908
00910
00911
00912
00913
00914
00915
                     exit();
00916
                 }
00917
00918
             unlink("bd");
for(i = 0; i < 500; i++){
   name[0] = 'x';
   name[1] = '0' + (i / 64);
   name[2] = '0' + (i % 64);
   name[3] = '\0';
   if(unlink(name) != 0){
      printf(1, "bigdir unlink failed");
      exit():</pre>
00919
00920
00921
00922
00923
00924
00925
00926
00927
                     exit();
00929
00931 printf(1, "bigdir ok\n");
00932 }
00930
```

5.240.2.4 bigfile()

```
void bigfile (
     void )
```

Definition at line 1148 of file usertests.c.

```
01149 {
         int fd, i, total, cc;
01150
01151
        printf(1, "bigfile test\n");
01152
01154
         unlink("bigfile");
         fd = open("bigfile", O_CREATE | O_RDWR);
01155
         if(fd < 0){
01156
         printf(1, "cannot create bigfile");
01157
01158
           exit();
01159
01160
         for(i = 0; i < 20; i++) {</pre>
         memset(buf, i, 600);
if(write(fd, buf, 600) != 600){
  printf(1, "write bigfile failed\n");
01161
01162
01163
01164
              exit();
01165
           }
01166
01167
         close(fd);
01168
         fd = open("bigfile", 0);
01169
01170
         if(fd < 0){</pre>
          printf(1, "cannot open bigfile\n");
01171
01172
           exit();
01173
01174
        total = 0;
        for(i = 0; ; i++) {
  cc = read(fd, buf, 300);
01175
01176
01177
           if(cc < 0){
01178
           printf(1, "read bigfile failed\n");
01179
             exit();
01180
           if(cc == 0)
01181
01182
             break:
           if(cc != 300){
  printf(1, "short read bigfile\n");
  exit();
01183
01184
01185
01186
           if(buf[0] != i/2 || buf[299] != i/2) {
  printf(1, "read bigfile wrong data\n");
01187
01188
01189
             exit();
01190
01191
           total += cc;
01192
01193
         close(fd);
         if(total != 20*600){
01194
         printf(1, "read bigfile wrong total\n");
01195
01196
           exit();
01197
01198
         unlink("bigfile");
01200 printf(1, "bigfile test ok\n");
01201 }
01199
```

Referenced by main().

5.240.2.5 bigwrite()

```
void bigwrite (
     void )
```

Definition at line 1119 of file usertests.c.

```
01120 {
01121    int fd, sz;
01122
01123    printf(1, "bigwrite test\n");
01124
01125    unlink("bigwrite");
```

```
01126 for(sz = 499; sz < 12*512; sz += 471) {
        fd = open("bigwrite", O_CREATE | O_RDWR);
01127
          if(fd < 0) {
  printf(1, "cannot create bigwrite\n");</pre>
01128
01129
01130
            exit();
01131
         int i;
for(i = 0; i < 2; i++){
01132
01133
          int cc = write(fd, buf, sz);
if(cc != sz){
  printf(1, "write(%d) ret %d\n", sz, cc);
01134
01135
01136
01137
               exit();
            }
01138
         close(fd);
01139
01140
         unlink("bigwrite");
01141
01142
01143
       printf(1, "bigwrite ok\n");
01144
01145 }
```

Referenced by main().

5.240.2.6 bsstest()

```
void bsstest (
     void )
```

Definition at line 1595 of file usertests.c.

```
01596 {
01597     int i;
01598
01599     printf(stdout, "bss test\n");
01600     for(i = 0; i < sizeof(uninit); i++) {
01601          if(uninit[i] != '\0') {
                printf(stdout, "bss test failed\n");
01603          exit();
01604     }
01605     }
01606     printf(stdout, "bss test ok\n");
01607 }</pre>
```

Referenced by main().

5.240.2.7 concreate()

```
void concreate (
     void )
```

Definition at line 765 of file usertests.c.

```
00766 {
        char file[3];
00767
       int i, pid, n, fd;
char fa[40];
00768
00769
00770
        struct {
        ushort inum;
char name[14];
00771
00772
00773 } de;
00774
       00775
00776
00777
       for(i = 0; i < 40; i++) {
  file[1] = '0' + i;
  unlink(file);</pre>
00778
00779
00780
00781
        pid = fork();
00782
          if(pid && (i % 3) == 1){
```

```
00783
              link("CO", file);
           } else if(pid == 0 && (i % 5) == 1){
link("CO", file);
00784
00785
           } else {
  fd = open(file, O_CREATE | O_RDWR);
00786
00787
00788
              if(fd < 0){
00789
                printf(1, "concreate create %s failed\n", file);
00790
                exit();
00791
00792
              close(fd);
00793
           if(pid == 0)
00794
00795
             exit();
00796
           else
00797
              wait();
00798
00799
00800
         memset(fa, 0, sizeof(fa));
fd = open(".", 0);
00801
00802
         n = 0;
00803
         while(read(fd, &de, sizeof(de)) > 0){
00804
           if(de.inum == 0)
00805
              continue;
           if (de.name[0] == 'C' && de.name[2] == '\0'){
  i = de.name[1] - '0';
  if (i < 0 || i >= sizeof(fa)){
    printf(1, "concreate weird file %s\n", de.name);
00806
00807
00808
00809
00810
                exit();
00811
              if(fa[i]){
00812
                printf(1, "concreate duplicate file %s\n", de.name);
00813
00814
                exit();
00815
00816
              fa[i] = 1;
00817
              n++;
           }
00818
00819
00820
         close(fd);
00821
         if(n != 40){
   printf(1, "concreate not enough files in directory listing\n");
00822
00823
00824
           exit();
00825
00826
         for(i = 0; i < 40; i++) {
  file[1] = '0' + i;</pre>
00827
00828
00829
           pid = fork();
            if(pid < 0) {
  printf(1, "fork failed\n");</pre>
00830
00831
00832
              exit();
00833
           if(((i % 3) == 0 && pid == 0) ||
((i % 3) == 1 && pid != 0)){
00834
00835
              close(open(file, 0));
00836
00837
              close(open(file, 0));
00838
              close(open(file, 0));
00839
              close(open(file, 0));
00840
00841
              unlink(file);
00842
              unlink(file);
              unlink(file);
00843
00844
             unlink(file);
00845
00846
           if(pid == 0)
00847
             exit();
           else
00848
00849
              wait();
00850
00851
        printf(1, "concreate ok\n");
00853 }
```

5.240.2.8 createdelete()

```
void createdelete (
     void )
```

Definition at line 584 of file usertests.c.

```
enum { N = 20 };
00586
           int pid, i, fd, pi;
char name[32];
00587
00588
00589
           printf(1, "createdelete test\n");
00591
           for(pi = 0; pi < 4; pi++) {
  pid = fork();
  if(pid < 0) {
    printf(1, "fork failed\n");
}</pre>
00592
00593
00594
00595
00596
                 exit();
00597
00598
              if(pid == 0) {
  name[0] = 'p' + pi;
  name[2] = '\0';
  for(i = 0; i < N; i++) {</pre>
00599
00600
00601
00602
                    name[1] = '0' + i;
00603
00604
                     fd = open(name, O_CREATE | O_RDWR);
                     if(fd < 0) {
  printf(1, "create failed\n");</pre>
00605
00606
00607
                       exit();
00608
                    close(fd);
                     if(i > 0 && (i % 2 ) == 0){
  name[1] = '0' + (i / 2);
  if(unlink(name) < 0){</pre>
00610
00611
00612
                         printf(1, "unlink failed\n");
00613
00614
                           exit();
00615
                       }
00616
                    }
00617
00618
                  exit();
00619
              }
00620
           }
00621
00622
            for(pi = 0; pi < 4; pi++) {</pre>
          wait();
}
00623
00624
00625
           name[0] = name[1] = name[2] = 0;
for(i = 0; i < N; i++) {
  for(pi = 0; pi < 4; pi++) {
    name[0] = 'p' + pi;
    name[1] = '0' + i;
}</pre>
00626
00627
00628
00629
00630
                  fd = open(name, 0);
if((i == 0 || i >= N/2) && fd < 0){
    printf(1, "oops createdelete %s didn't exist\n", name);</pre>
00631
00632
00633
00634
                     exit();
                 pelse if((i >= 1 && i < N/2) && fd >= 0){
printf(1, "oops createdelete %s did exist\n", name);
00635
00636
00637
                    exit();
00638
00639
                  if(fd >= 0)
00640
                    close(fd);
00641
           }
00642
00643
           for(i = 0; i < N; i++) {
  for(pi = 0; pi < 4; pi++) {
    name[0] = 'p' + i;
    name[1] = '0' + i;
}</pre>
00644
00645
00646
00647
00648
                  unlink(name);
00649
00650
00651
          printf(1, "createdelete ok\n");
00652
00653 }
```

Referenced by main().

5.240.2.9 createtest()

```
void createtest (
     void )
```

Definition at line 245 of file usertests.c.

```
00246 {
00247
         int i, fd;
00248
         printf(stdout, "many creates, followed by unlink testn");
00249
00250
00251
00252
         name[2] = ' \setminus 0';
         for(i = 0; i < 52; i++) {
  name[1] = '0' + i;</pre>
00253
00254
           fd = open(name, O_CREATE|O_RDWR);
00255
00256
           close(fd);
00257
00258
         name[0] = 'a';
         name[2] = ' \setminus 0';
00259
         for(i = 0; i < 52; i++) {
  name[1] = '0' + i;</pre>
00260
00261
00262
           unlink(name);
00263
00264
        printf(stdout, "many creates, followed by unlink; ok\n");
00265 }
```

Referenced by main().

5.240.2.10 dirfile()

```
void dirfile (
     void )
```

Definition at line 1284 of file usertests.c.

```
01285
01286
01287
01288
        printf(1, "dir vs file\n");
01289
        fd = open("dirfile", O_CREATE);
01290
01291
        if (fd < 0) {</pre>
        printf(1, "create dirfile failed\n");
01292
01293
          exit();
01294
01295
        close(fd);
       if(chdir("dirfile") == 0) {
  printf(1, "chdir dirfile succeeded!\n");
01296
01297
01298
          exit();
01299
01300
        fd = open("dirfile/xx", 0);
        if(fd >= 0) {
   printf(1, "create dirfile/xx succeeded!\n");
01301
01302
01303
          exit();
01304
01305
        fd = open("dirfile/xx", O_CREATE);
01306
       if(fd >= 0){
01307
         printf(1, "create dirfile/xx succeeded!\n");
01308
          exit();
01309
        if (mkdir("dirfile/xx") == 0) {
01310
01311
         printf(1, "mkdir dirfile/xx succeeded!\n");
01312
          exit();
01313
        if(unlink("dirfile/xx") == 0){
01314
         printf(1, "unlink dirfile/xx succeeded!\n");
01315
01316
          exit();
01317
        if(link("README", "dirfile/xx") == 0){
01318
01319
         printf(1, "link to dirfile/xx succeeded!\n");
01320
01321
        if (unlink("dirfile") != 0) {
01322
         printf(1, "unlink dirfile failed!\n");
01323
01324
          exit();
01325
01326
        fd = open(".", O_RDWR);
01327
        if (fd >= 0) {
  printf(1, "open . for writing succeeded!\n");
01328
01329
01330
          exit();
01331
```

```
01332     fd = open(".", 0);
01333     if(write(fd, "x", 1) > 0) {
01334          printf(1, "write . succeeded!\n");
01335          exit();
01336     }
01337     close(fd);
01338
01339     printf(1, "dir vs file OK\n");
01340 }
```

Referenced by main().

5.240.2.11 dirtest()

```
void dirtest (
     void )
```

Definition at line 267 of file usertests.c.

```
00269
         printf(stdout, "mkdir test\n");
00270
         if (mkdir("dir0") < 0) {
  printf(stdout, "mkdir failed\n");</pre>
00271
00272
00273
            exit();
00274
00275
         if(chdir("dir0") < 0) {
  printf(stdout, "chdir dir0 failed\n");
  exit();</pre>
00276
00277
00278
00279
00280
00281
         if(chdir("..") < 0){</pre>
         printf(stdout, "chdir .. failed\n");
exit();
00282
00283
00284 }
00285
         if(unlink("dir0") < 0){
  printf(stdout, "unlink dir0 failed\n");
  exit();</pre>
00286
00287
00288
00289
00290 printf(stdout, "mkdir test ok\n");
00291 }
```

5.240.2.12 exectest()

```
void exectest (
     void )
```

Definition at line 294 of file usertests.c.

```
00295 {
00296    printf(stdout, "exec test\n");
00297    if(exec("echo", echoargy) < 0) {
00298        printf(stdout, "exec echo failed\n");
00299        exit();
00300    }
00301 }</pre>
```

5.240.2.13 exitiputtest()

```
void exitiputtest (
                 void )
Definition at line 43 of file usertests.c.
00044
00045
         int pid;
00046
00047
         printf(stdout, "exitiput test\n");
00048
         pid = fork();
00049
         if(pid < 0) {
  printf(stdout, "fork failed\n");</pre>
00050
00051
00052
           exit();
00053
00054
         if (pid == 0) {
           if (mkdir("iputdir") < 0) {
   printf(stdout, "mkdir failed\n");</pre>
00055
00056
00057
              exit();
00058
            if(chdir("iputdir") < 0){
  printf(stdout, "child chdir failed\n");</pre>
00059
00060
00061
              exit();
00062
            if(unlink("../iputdir") < 0){
  printf(stdout, "unlink ../iputdir failed\n");</pre>
00063
00064
00065
              exit();
00066
00067
           exit();
00068
00069
         wait();
00070
         printf(stdout, "exitiput test ok\n");
```

Referenced by main().

00071 }

5.240.2.14 exitwait()

```
void exitwait (
     void )
```

Definition at line 405 of file usertests.c.

```
00406 {
00407
          int i, pid;
00408
00409
          for(i = 0; i < 100; i++){</pre>
          pid = fork();
if(pid < 0) {
   printf(1, "fork failed\n");</pre>
00410
00411
00412
00413
               return;
00414
00415
            if(pid){
              if(wait() != pid) {
  printf(1, "wait wrong pid\n");
00416
00417
00418
                 return;
00419
00420
            } else {
00421
              exit();
           }
00422
00424 printf(1, "exitwait ok\n");
00425 }
```

5.240.2.15 forktest()

```
void forktest (
     void )
```

Definition at line 1380 of file usertests.c.

```
01381 {
01382
        int n. pid:
01383
        printf(1, "fork test\n");
01384
01385
        for(n=0; n<1000; n++) {
  pid = fork();</pre>
01386
01387
          if(pid < 0)</pre>
01388
01389
         break;
if(pid == 0)
01390
01391
            exit();
01392
01393
        if(n == 1000){
  printf(1, "fork claimed to work 1000 times!\n");
01394
01395
01396
          exit();
01397
01398
        for(; n > 0; n--) {
   if(wait() < 0) {
     printf(1, "wait stopped early\n");</pre>
01399
01400
01401
01402
            exit();
01403
01404
01405
        01406
01407
01408
          exit();
01409
01410
```

Referenced by main().

5.240.2.16 fourfiles()

```
void fourfiles (
    void )
```

Definition at line 518 of file usertests.c.

```
00519 {
        int fd, pid, i, j, n, total, pi;
char *names[] = { "f0", "f1", "f2", "f3" };
00520
00521
        char *fname;
00522
00523
        printf(1, "fourfiles test\n");
00524
00525
00526
        for(pi = 0; pi < 4; pi++) {</pre>
00527
         fname = names[pi];
00528
          unlink(fname);
00529
          pid = fork();
00530
          if (pid < 0) {
  printf(1, "fork failed\n");</pre>
00531
00532
00533
             exit();
00534
00535
00536
          if (pid == 0) {
            fd = open(fname, O_CREATE | O_RDWR);
00537
00538
             if(fd < 0){
00539
             printf(1, "create failed\n");
00540
               exit();
00541
00542
00543
             memset(buf, '0'+pi, 512);
00544
             for(i = 0; i < 12; i++) {
              if((n = write(fd, buf, 500)) != 500){
```

```
printf(1, "write failed %d\n", n);
00547
                exit();
00548
              }
00549
00550
            exit();
00551
          }
00552
00553
00554
        for(pi = 0; pi < 4; pi++) {</pre>
       _ \p1 = \
wait();
00555
00556
00557
00558
        for (i = 0; i < 2; i++) {
00559
         fname = names[i];
00560
          fd = open(fname, 0);
00561
          total = 0;
          while((n = read(fd, buf, sizeof(buf))) > 0){
00562
            for(j = 0; j < n; j++) {
    if(buf[j] != '0'+i) {
00563
00564
00565
                printf(1, "wrong char\n");
00566
                exit();
              }
00567
00568
00569
            total += n;
00570
00571
          close(fd);
00572
          if(total != 12*500){
00573
            printf(1, "wrong length %d\n", total);
00574
             exit();
00575
00576
          unlink(fname);
00577
00578
        printf(1, "fourfiles ok\n");
00579
00580 }
```

5.240.2.17 fourteen()

```
void fourteen (
     void )
```

Definition at line 1204 of file usertests.c.

```
01205 {
01206
        int fd;
01207
       // DIRSIZ is 14.
01208
       printf(1, "fourteen test\n");
01209
01210
01211
       if (mkdir("12345678901234") != 0) {
01212
        printf(1, "mkdir 12345678901234 failed\n");
         exit();
01213
01214
       if (mkdir("12345678901234/123456789012345") != 0) {
01215
        printf(1, "mkdir 12345678901234/123456789012345 failed\n");
01216
01217
         exit();
01218
01219
       fd = open("123456789012345/123456789012345", O_CREATE);
01220
       if (fd < 0) {</pre>
         printf(1, "create 123456789012345/123456789012345/123456789012345 failed\n");
01221
01222
         exit();
01223
01224
       close(fd);
01225
       fd = open("12345678901234/12345678901234/12345678901234", 0);
01226
       if(fd < 0){</pre>
         printf(1, "open 12345678901234/12345678901234/12345678901234 failed\n");
01227
01228
         exit();
01229
01230
       close(fd);
01231
01232
       if (mkdir("12345678901234/12345678901234") == 0) {
        printf(1, "mkdir 12345678901234/12345678901234 succeeded!\n");
01233
01234
         exit();
01235
01236
       if (mkdir("123456789012345/12345678901234") == 0) {
01237
         printf(1, "mkdir 12345678901234/123456789012345 succeeded!\n");
```

```
01238 exit();
01239 }
01240
01241 printf(1, "fourteen ok\n");
01242 }
```

Referenced by main().

5.240.2.18 fsfull()

```
void fsfull ( )
```

Definition at line 1648 of file usertests.c.

```
01649 {
01650
            int nfiles;
            int fsblocks = 0;
01651
01652
           printf(1, "fsfull test\n");
01653
01654
01655
            for(nfiles = 0; ; nfiles++) {
01656
              char name[64];
              name[0] = 'f';
name[1] = '0' + nfiles / 1000;
name[2] = '0' + (nfiles % 1000) / 100;
name[3] = '0' + (nfiles % 100) / 10;
name[4] = '0' + (nfiles % 10);
01657
01658
01659
01660
01661
              name[4] = '0' + (nfiles % 10);
name[5] = '\0';
printf(1, "writing %s\n", name);
int fd = open(name, O_CREATE|O_RDWR);
if(fd < 0) {</pre>
01662
01663
01664
01665
                 printf(1, "open %s failed\n", name);
break;
01666
01667
01668
01669
               int total = 0;
               while(1) {
  int cc = write(fd, buf, 512);
  if(cc < 512)</pre>
01670
01671
01672
                 break;
total += cc;
01673
01674
01675
                  fsblocks++;
01676
               printf(1, "wrote %d bytes\n", total);
01677
01678
               close(fd);
01679
               if(total == 0)
01680
                 break;
01681
01682
           while (nfiles >= 0) {
01683
            char name[64];
name[0] = 'f';
name[1] = '0' + nfiles / 1000;
01684
01685
01686
              name[2] = '0' + nilles / 1000;
name[2] = '0' + (nfiles % 1000) / 100;
name[3] = '0' + (nfiles % 100) / 10;
name[4] = '0' + (nfiles % 10);
name[5] = '\0';
01687
01688
01689
01690
01691
               unlink(name);
01692
              nfiles--;
01693
01694
01695
           printf(1, "fsfull test finished\n");
01696 }
```

5.240.2.19 iputtest()

```
void iputtest (
     void )
```

Definition at line 18 of file usertests.c.

00019 {

```
printf(stdout, "iput test\n");
00021
        if (mkdir("iputdir") < 0){</pre>
00022
         printf(stdout, "mkdir failed\n");
00023
00024
           exit();
00025
        if(chdir("iputdir") < 0) {
  printf(stdout, "chdir iputdir failed\n");
  exit();</pre>
00026
00027
00028
00029
        if (unlink("../iputdir") < 0) {</pre>
00030
         printf(stdout, "unlink ../iputdir failed\n");
00031
00032
          exit();
00033
00034
        if(chdir("/") < 0){</pre>
         printf(stdout, "chdir / failed\n");
00035
00036
           exit();
00037
        printf(stdout, "iput test ok\n");
00039 }
```

5.240.2.20 iref()

```
void iref (
    void )
```

Definition at line 1344 of file usertests.c.

```
01345 {
01346
          int i, fd;
01347
01348
         printf(1, "empty file name\n");
01349
         // the 50 is NINODE
for(i = 0; i < 50 + 1; i++) {
  if(mkdir("irefd") != 0) {</pre>
01350
01351
01352
01353
              printf(1, "mkdir irefd failed\n");
01354
01355
           if(chdir("irefd") != 0) {
  printf(1, "chdir irefd failed\n");
01356
01357
01358
              exit();
01359
01360
01361
            mkdir("");
           link("README", "");
fd = open("", O_CREATE);
if(fd >= 0)
01362
01363
01364
              close(fd);
01365
01366
           fd = open("xx", O_CREATE);
01367
            if(fd >= 0)
01368
              close(fd);
01369
           unlink("xx");
01370
01371
01372
         chdir("/");
01373 printf(1, "empty file name OK\n");
01374 }
```

5.240.2.21 linktest()

```
void linktest (
     void )
```

Definition at line 702 of file usertests.c.

```
00704
00705
         printf(1, "linktest\n");
00706
00707
00708
         unlink("lf1");
00709
         unlink("1f2");
00710
00711
         fd = open("lf1", O_CREATE(O_RDWR);
         if (fd < 0) {
  printf(1, "create lf1 failed\n");</pre>
00712
00713
00714
           exit();
00715
         if(write(fd, "hello", 5) != 5) {
  printf(1, "write lf1 failed\n");
00716
00717
00718
           exit();
00719
00720
         close(fd);
00721
         if(link("lf1", "lf2") < 0){
  printf(1, "link lf1 lf2 failed\n");
  printf();</pre>
00722
00723
00724
           exit();
00725
00726
         unlink("lf1"):
00727
00728
         if(open("1f1", 0) >= 0){
          printf(1, "unlinked lf1 but it is still there!\n");
00729
00730
            exit();
00731
00732
00733
         fd = open("lf2", 0);
00734
         if (fd < 0) {
         printf(1, "open 1f2 failed\n");
exit();
00735
00736
00737
         if(read(fd, buf, sizeof(buf)) != 5){
  printf(1, "read lf2 failed\n");
00738
00739
00740
           exit();
00741
00742
         close(fd);
00743
         if(link("lf2", "lf2") >= 0){
  printf(1, "link lf2 lf2 succeeded! oops\n");
00744
00745
00746
            exit();
00747
00748
         unlink("lf2");
if(link("lf2", "lf1") >= 0){
  printf(1, "link non-existant succeeded! oops\n");
  evit():
00749
00750
00751
00752
           exit();
00753
00754
         if(link(".", "lf1") >= 0){
   printf(1, "link . lf1 succeeded! oops\n");
00755
00756
00757
           exit();
00758
00759
00760
         printf(1, "linktest ok\n");
00761 }
```

Referenced by main().

5.240.2.22 linkunlink()

```
void linkunlink ( )
```

Definition at line 858 of file usertests.c. 00859 {

```
00860
          int pid, i;
00861
          printf(1, "linkunlink test\n");
00862
00863
00864
           unlink("x");
           pid = fork();
00865
           if (pid < 0) {
  printf(1, "fork failed\n");</pre>
00866
00867
00868
             exit();
00869
00870
           unsigned int x = (pid ? 1 : 97);
for(i = 0; i < 100; i++) {
  x = x * 1103515245 + 12345;</pre>
00871
00872
00873
             if((x % 3) == 0){
  close(open("x", O_RDWR | O_CREATE));
} else if((x % 3) == 1) {
  link("cat", "x");
00874
00875
00876
00877
00878
             } else {
00879
                unlink("x");
00880
00881
00882
           if (pid)
00883
00884
             wait();
00886
             exit();
00888 printf(1, "linkunlink ok\n");
00889 }
00887
```

5.240.2.23 main()

```
int main (
          int argc,
          char * argv[] )
```

Definition at line 1749 of file usertests.c.

```
01750 {
01751
        printf(1, "usertests starting\n");
01752
01753
        if(open("usertests.ran", 0) >= 0){
01754
        printf(1, "already ran user tests -- rebuild fs.img\n");
01755
          exit();
01756
       close(open("usertests.ran", O_CREATE));
01757
01758
01759
       argptest();
01760
        createdelete();
01761
       linkunlink();
01762
       concreate();
       fourfiles();
01763
01764
       sharedfd();
01765
01766
       bigargtest();
01767
        bigwrite();
01768
       bigargtest();
01769
       bsstest();
01770
       sbrktest();
01771
       validatetest();
01772
01773
       opentest();
01774
01775
       writetest1();
01776
       createtest();
01777
01778
       openiputtest();
01779
       exitiputtest();
01780
       iputtest();
01781
01782
       mem();
01783
       pipel();
       preempt();
exitwait();
01784
01785
01786
```

```
01787
        rmdot();
01788
        fourteen();
01789
        bigfile();
01790
        subdir();
01791
        linktest();
01792
        unlinkread();
01793
        dirfile();
01794
        iref();
01795
        forktest();
       bigdir(); // slow
01796
01797
01798
       uio();
01799
01800
       exectest();
01801
01802 exit();
01803 }
```

5.240.2.24 mem()

```
void mem (
     void )
```

Definition at line 428 of file usertests.c.

```
00429 {
          void *m1, *m2;
00430
00431
         int pid, ppid;
00432
00433
         printf(1, "mem test\n");
         print( (, mem test(n)),
    ppid = getpid();
    if((pid = fork()) == 0) {
        m1 = 0;
        while((m2 = malloc(10001)) != 0) {
00434
00435
00436
00437
00438
               *(char**)m2 = m1;
00439
               m1 = m2;
00440
            while (m1) {
  m2 = *(char**) m1;
  free (m1);
00441
00442
00443
00444
              m1 = m2;
00445
00446
            m1 = malloc(1024*20);
            if (m1 == 0) {
  printf(1, "couldn't allocate mem?!!\n");
00447
00448
00449
               kill(ppid);
00450
              exit();
00451
00452
            free(m1);
00453
            printf(1, "mem ok\n");
00454
            exit();
00455
         } else {
00456
            wait();
00457
          }
00458 }
```

Referenced by allocuvm(), copyuvm(), inituvm(), and main().

5.240.2.25 openiputtest()

```
void openiputtest (
    void )
```

Definition at line 85 of file usertests.c.

```
00086 {
00087    int pid;
00088
00089    printf(stdout, "openiput test\n");
00090    if(mkdir("oidir") < 0) {</pre>
```

```
printf(stdout, "mkdir oidir failed\n");
00092
           exit();
00093
00094
        pid = fork();
00095
        if (pid < 0) {</pre>
        printf(stdout, "fork failed\n");
exit();
00096
00098
00099
        if (pid == 0) {
         int fd = open("oidir", O_RDWR);
if(fd >= 0){
00100
00101
            printf(stdout, "open directory for write succeeded\n");
00102
00103
             exit();
00104
00105
           exit();
00106
        sleep(1);
if(unlink("oidir") != 0){
  printf(stdout, "unlink failed\n");
00107
00108
00109
00110
          exit();
00111
00112
        wait();
00113 printf(stdout, "openiput test ok\n");
00114 }
```

5.240.2.26 opentest()

```
void opentest (
    void )
```

Definition at line 119 of file usertests.c.

```
00121
         int fd;
00122
        printf(stdout, "open test\n");
00123
        fd = open("echo", 0);
if(fd < 0){
00124
00125
        printf(stdout, "open echo failed!\n");
exit();
00127
00128
        close(fd);
fd = open("doesnotexist", 0);
if(fd >= 0){
00129
00130
00131
00132
         printf(stdout, "open doesnotexist succeeded!\n");
00133
           exit();
00135 printf(stdout, "open test ok\n");
00136 }
00134
```

Referenced by main().

5.240.2.27 pipe1()

```
void pipel (
          void )
```

Definition at line 306 of file usertests.c.

```
00307 {
00308    int fds[2], pid;
00309    int seq, i, n, cc, total;
00310
00311    if(pipe(fds) != 0) {
        printf(1, "pipe() failed\n");
00312         exit();
00314    }
```

```
pid = fork();
         seq = 0;
if (pid == 0) {
00316
00317
          close(fds[0]);
00318
            for (n = 0; n < 5; n++) {
  for (i = 0; i < 1033; i++)
    buf[i] = seq++;
  if (write (fds[1], buf, 1033) != 1033) {</pre>
00319
00320
00321
00322
00323
               printf(1, "pipel oops 1\n");
00324
                  exit();
               }
00325
00326
         exit();
} else if(pid > 0){
00327
00328
00329
            close(fds[1]);
00330
            total = 0;
00331
            cc = 1;
            cc = 1;
while((n = read(fds[0], buf, cc)) > 0){
  for(i = 0; i < n; i++){
    if((buf[i] & 0xff) != (seq++ & 0xff)){
    printf(1, "pipel oops 2\n");
00332
00333
00334
00335
00336
                     return;
00337
                 }
00338
00339
               total += n;
00340
               cc = cc * 2;
00341
               if(cc > sizeof(buf))
00342
                 cc = sizeof(buf);
00343
            if(total != 5 * 1033){
00344
               printf(1, "pipe1 oops 3 total %d\n", total);
00345
00346
               exit();
00347
00348
            close(fds[0]);
00349
            wait();
00350
         } else {
         printf(1, "fork() failed\n");
exit();
00351
00353
00354
         printf(1, "pipe1 ok\n");
00355 }
```

Referenced by main().

5.240.2.28 preempt()

```
void preempt (
     void )
```

Definition at line 359 of file usertests.c.

```
00360 {
00361
        int pid1, pid2, pid3;
00362
        int pfds[2];
00363
        printf(1, "preempt: ");
pid1 = fork();
00364
00365
        if (pid1 == 0)
00366
00367
         for(;;)
00368
00369
        pid2 = fork();
if (pid2 == 0)
00370
00371
         for(;;)
00372
00373
             ;
00374
        pipe(pfds);
pid3 = fork();
00375
00376
         if (pid3 == 0) {
00377
00378
         close(pfds[0]);
if(write(pfds[1], "x", 1) != 1)
00379
00380
             printf(1, "preempt write error");
00381
           close(pfds[1]);
00382
           for(;;)
00383
             ;
00384
        }
00385
00386
        close(pfds[1]);
```

```
if(read(pfds[0], buf, sizeof(buf)) != 1){
        printf(1, "preempt read error");
00388
00389
           return;
00390
        close(pfds[0]);
printf(1, "kill...");
kill(pid1);
00391
00392
00393
00394
        kill(pid2);
00395
        kill(pid3);
        printf(1, "wait... ");
00396
00397
        wait();
00398
        wait();
00399
        wait();
00400
       printf(1, "preempt ok\n");
00401 }
```

5.240.2.29 rand()

```
unsigned int rand ( )
```

Definition at line 1742 of file usertests.c.

```
01743 {
01744    randstate = randstate * 1664525 + 1013904223;
01745    return randstate;
01746 }
```

5.240.2.30 rmdot()

```
void rmdot (
     void )
```

Definition at line 1245 of file usertests.c.

```
01246 {
         printf(1, "rmdot test\n");
if(mkdir("dots") != 0){
01247
01248
         printf(1, "mkdir dots failed\n");
exit();
01249
01250
01251
        if (chdir("dots") != 0) {
01252
          printf(1, "chdir dots failed\n");
01253
01254
           exit();
01255
         if(unlink(".") == 0) {
  printf(1, "rm . worked!\n");
01256
01257
01258
           exit();
01259
         if(unlink("..") == 0) {
  printf(1, "rm .. worked!\n");
  exit();
01260
01261
01262
01263
         if(chdir("/") != 0) {
   printf(1, "chdir / failed\n");
}
01264
01265
01266
           exit();
01267
01268
         if (unlink("dots/.") == 0) {
         printf(1, "unlink dots/. worked!\n");
01269
01270
           exit();
01271
01272
         if (unlink("dots/..") == 0) {
         printf(1, "unlink dots/.. worked!\n");
exit();
01273
01274
01275
01276
         if (unlink("dots") != 0) {
          printf(1, "unlink dots failed!\n");
01277
01278
           exit();
01280 printf(1, "rmdot ok\n");
01281 }
01279
```

5.240.2.31 sbrktest()

```
void sbrktest (
              void )
Definition at line 1415 of file usertests.c.
01416 {
        int fds[2], pid, pids[10], ppid;
01417
01418
        char *a, *b, *c, *lastaddr, *oldbrk, *p, scratch;
01419
       uint amt;
01420
       printf(stdout, "sbrk test\n");
oldbrk = sbrk(0);
01421
01422
01423
01424
       // can one sbrk() less than a page?
01425
        a = sbrk(0);
01426
       int i;
       for(i = 0; i < 5000; i++) {
  b = sbrk(1);</pre>
01427
01428
01429
         if(b != a){
01430
          printf(stdout, "sbrk test failed %d %x %x\n", i, a, b);
01431
           exit();
01432
          *b = 1;
01433
01434
         a = b + 1;
01435
01436
        pid = fork();
01437
        if (pid < 0) {</pre>
01438
         printf(stdout, "sbrk test fork failed\n");
01439
         exit();
01440
01441
       c = sbrk(1);
        c = sbrk(1);
01442
01443
        if(c != a + 1) {
01444
        printf(stdout, "sbrk test failed post-fork\n");
01445
          exit();
01446
        if (pid == 0)
01447
01448
         exit();
01449
        wait();
01450
01451
        // can one grow address space to something big?
01452 #define BIG (100*1024*1024)
       a = sbrk(0);
01453
01454
       amt = (BIG) - (uint)a;
01455
        p = sbrk(amt);
       if (p != a) {
01456
01457
        printf(stdout, "sbrk test failed to grow big address space; enough phys mem?\n");
01458
         exit();
01459
       lastaddr = (char*) (BIG-1);
*lastaddr = 99;
01460
01461
01462
01463
        // can one de-allocate?
01464
        a = sbrk(0);
        c = sbrk(-4096);
01465
        if(c == (char*) 0xffffffff) {
01466
        printf(stdout, "sbrk could not deallocate\n");
01467
01468
          exit();
01469
01470
        c = sbrk(0);
01471
        if(c != a - 4096) {
        printf(stdout, "sbrk deallocation produced wrong address, a %x c %x\n", a, c);
01472
01473
          exit();
01474
01475
01476
        // can one re-allocate that page?
01477
        a = sbrk(0);
        c = sbrk(4096);
01478
01479
        if(c != a || sbrk(0) != a + 4096){
         printf(stdout, "sbrk re-allocation failed, a %x c %x\n", a, c);
01480
01481
          exit();
01482
        if(*lastaddr == 99){}
01483
        // should be zero
01484
         printf(stdout, "sbrk de-allocation didn't really deallocate\n");
01485
01486
          exit();
01487
01488
01489
       a = sbrk(0);
       c = sbrk(-(sbrk(0) - oldbrk));
01490
01491
       if(c != a) {
01492
        printf(stdout, "sbrk downsize failed, a %x c %x\n", a, c);
01493
          exit();
```

```
01494
        }
01495
01496
         // can we read the kernel's memory?
        for(a = (char*) (KERNBASE); a < (char*) (KERNBASE+2000000); a += 50000) {</pre>
01497
          ppid = getpid();
01498
          pid = fork();
01499
01500
          if (pid < 0) {
01501
            printf(stdout, "fork failed\n");
01502
             exit();
01503
01504
          if(pid == 0){
            printf(stdout, "oops could read x = x^n, a, *a);
01505
             kill(ppid);
01506
01507
            exit();
01508
01509
          wait();
01510
01511
01512
        // if we run the system out of memory, does it clean up the last
01513
        // failed allocation?
        if (pipe (fds) != 0) {
  printf(1, "pipe() failed\n");
01514
01515
01516
          exit();
01517
01518
        for(i = 0; i < sizeof(pids)/sizeof(pids[0]); i++){</pre>
01519
         if((pids[i] = fork()) == 0){
01520
            // allocate a lot of memory
01521
             sbrk(BIG - (uint)sbrk(0));
            write(fds[1], "x", 1);
// sit around until killed
for(;;) sleep(1000);
01522
01523
01524
01525
01526
          if (pids[i] != -1)
01527
             read(fds[0], &scratch, 1);
01528
        ^{\prime\prime} // if those failed allocations freed up the pages they did allocate,
01529
        // we'll be able to allocate here
01530
        c = sbrk(4096);
01531
01532
        for(i = 0; i < sizeof(pids)/sizeof(pids[0]); i++){</pre>
01533
         if(pids[i] == -1)
01534
          kill(pids[i]);
01535
01536
          wait();
01537
01538
        if(c == (char*) 0xffffffff) {
         printf(stdout, "failed sbrk leaked memory\n");
01539
01540
          exit();
01541
01542
01543
        if(sbrk(0) > oldbrk)
          sbrk(-(sbrk(0) - oldbrk));
01545
        printf(stdout, "sbrk test OK\n");
01546
01547 }
```

5.240.2.32 sharedfd()

```
void sharedfd (
     void )
```

Definition at line 465 of file usertests.c.

```
00466 {
00467
        int fd, pid, i, n, nc, np;
00468
       char buf[10];
00469
       printf(1, "sharedfd test\n");
00470
00471
00472
       unlink("sharedfd");
00473
       fd = open("sharedfd", O_CREATE|O_RDWR);
00474
       if (fd < 0) {</pre>
        printf(1, "fstests: cannot open sharedfd for writing");
00475
00476
          return;
00477
00478
       pid = fork();
       memset(buf, pid==0?'c':'p', sizeof(buf));
```

```
for(i = 0; i < 1000; i++){</pre>
          if(write(fd, buf, sizeof(buf)) != sizeof(buf)) {
   printf(1, "fstests: write sharedfd failed\n");
00481
00482
00483
              break;
00484
           }
00485
00486
         if (pid == 0)
00487
           exit();
00488
         else
00489
           wait();
         close(fd);
00490
         fd = open("sharedfd", 0);
if(fd < 0){</pre>
00491
00492
00493
          printf(1, "fstests: cannot open sharedfd for reading\n");
00494
00495
00496
         nc = np = 0;
         me = mp = 0,
while((n = read(fd, buf, sizeof(buf))) > 0){
  for(i = 0; i < sizeof(buf); i++){</pre>
00497
00498
00499
             if(buf[i] == 'c')
00500
                nc++;
              if(buf[i] == 'p')
00501
00502
               np++;
00503
           }
00504
00505
         close(fd);
00506
         unlink("sharedfd");
00507
         if (nc == 10000 && np == 10000) {
00508
           printf(1, "sharedfd ok\n");
         } else {
00509
         printf(1, "sharedfd oops %d %d\n", nc, np);
00510
00511
           exit();
00512 }
00513 }
```

Referenced by main().

5.240.2.33 subdir()

```
void subdir (
     void )
```

Definition at line 935 of file usertests.c.

```
00936 {
00937
        int fd, cc;
00938
00939
       printf(1, "subdir test\n");
00940
       unlink("ff");
if(mkdir("dd") != 0){
00941
00942
        printf(1, "subdir mkdir dd failed\n");
00943
00944
00945
00946
        fd = open("dd/ff", O_CREATE | O_RDWR);
00947
00948
        if (fd < 0) {
        printf(1, "create dd/ff failed\n");
00949
00950
         exit();
00951
       write(fd, "ff", 2);
00952
00953
       close(fd);
00954
        if(unlink("dd") >= 0){
00955
        printf(1, "unlink dd (non-empty dir) succeeded!\n");
00956
00957
          exit();
00958
00959
        if (mkdir("/dd/dd") != 0) {
00960
        printf(1, "subdir mkdir dd/dd failed\n");
00961
00962
         exit();
00963
00964
       fd = open("dd/dd/ff", O_CREATE | O_RDWR);
00965
       <u>if</u>(fd < 0){
00966
        printf(1, "create dd/dd/ff failed\n");
00967
00968
          exit();
00969
```

```
write(fd, "FF", 2);
        close(fd);
00971
00972
00973
        fd = open("dd/dd/../ff", 0);
        if(fd < 0){</pre>
00974
         printf(1, "open dd/dd/../ff failed\n");
00975
00976
          exit();
00977
00978
        cc = read(fd, buf, sizeof(buf));
        if(cc != 2 || buf[0] != 'f'){
  printf(1, "dd/dd/../ff wrong content\n");
00979
00980
00981
          exit();
00982
00983
00984
        00985
00986
00987
          exit();
00988
00989
00990
        if(unlink("dd/dd/ff") != 0) {
00991
          printf(1, "unlink dd/dd/ff failed\n");
00992
          exit();
00993
        if(open("dd/dd/ff", O_RDONLY) >= 0){
  printf(1, "open (unlinked) dd/dd/ff succeeded\n");
00994
00995
00996
           exit();
00997
00998
00999
        if(chdir("dd") != 0){
         printf(1, "chdir dd failed\n");
01000
01001
          exit();
01002
01003
        if(chdir("dd/../../dd") != 0) {
01004
          printf(1, "chdir dd/../../dd failed\n");
01005
           exit();
01006
01007
        if(chdir("dd/../../dd") != 0){
01008
         printf(1, "chdir dd/../../dd failed\n");
01009
01010
        if(chdir("./..") != 0) {
  printf(1, "chdir ./.. failed\n");
01011
01012
01013
          exit();
01014
01015
01016
        fd = open("dd/dd/fffff", 0);
01017
        if (fd < 0) {</pre>
          printf(1, "open dd/dd/ffff failed\n");
01018
01019
           exit();
01020
01021
        if(read(fd, buf, sizeof(buf)) != 2){
01022
         printf(1, "read dd/dd/ffff wrong len\n");
01023
          exit();
01024
01025
        close (fd);
01026
01027
        if(open("dd/dd/ff", O_RDONLY) >= 0){
01028
         printf(1, "open (unlinked) dd/dd/ff succeeded!\n");
01029
           exit();
01030
01031
01032
        if (open("dd/ff/ff", O_CREATE|O_RDWR) >= 0) {
01033
         printf(1, "create dd/ff/ff succeeded!\n");
01034
           exit();
01035
        if(open("dd/xx/ff", O_CREATE|O_RDWR) >= 0){
01036
         printf(1, "create dd/xx/ff succeeded!\n");
01037
01038
          exit();
01039
        if(open("dd", O_CREATE) >= 0) {
  printf(1, "create dd succeeded!\n");
01040
01041
01042
          exit();
01043
        if(open("dd", O_RDWR) >= 0) {
  printf(1, "open dd rdwr succeeded!\n");
01044
01045
01046
           exit();
01047
        if(open("dd", O_WRONLY) >= 0){
  printf(1, "open dd wronly succeeded!\n");
01048
01049
01050
           exit();
01051
        if(link("dd/ff/ff", "dd/dd/xx") == 0){
  printf(1, "link dd/ff/ff dd/dd/xx succeeded!\n");
01052
01053
01054
           exit();
01055
01056
         if (link("dd/xx/ff", "dd/dd/xx") == 0) {
```

```
printf(1, "link dd/xx/ff dd/dd/xx succeeded!\n");
01058
01059
         if(link("dd/fff", "dd/dd/fffff") == 0){
  printf(1, "link dd/ff dd/dd/ffff succeeded!\n");
01060
01061
01062
           exit();
01063
01064
         if(mkdir("dd/ff/ff") == 0){
01065
         printf(1, "mkdir dd/ff/ff succeeded!\n");
01066
           exit();
01067
        if(mkdir("dd/xx/ff") == 0){
01068
         printf(1, "mkdir dd/xx/ff succeeded!\n");
01069
01070
01071
         if (mkdir("dd/dd/ffff") == 0) {
  printf(1, "mkdir dd/dd/ffff succeeded!\n");
01072
01073
01074
           exit();
01075
01076
         if (unlink("dd/xx/ff") == 0) {
01077
          printf(1, "unlink dd/xx/ff succeeded!\n");
01078
           exit();
01079
        if(unlink("dd/ff/ff") == 0) {
  printf(1, "unlink dd/ff/ff succeeded!\n");
  ref();
01080
01081
01082
01083
        if(chdir("dd/ff") == 0) {
  printf(1, "chdir dd/ff succeeded!\n");
01084
01085
01086
           exit();
01087
01088
        if(chdir("dd/xx") == 0){
01089
         printf(1, "chdir dd/xx succeeded!\n");
01090
           exit();
01091
01092
01093
         if (unlink("dd/dd/fffff") != 0) {
          printf(1, "unlink dd/dd/ff failed\n");
01095
           exit();
01096
        if(unlink("dd/ff") != 0) {
  printf(1, "unlink dd/ff failed\n");
01097
01098
01099
           exit():
01100
01101
        if (unlink("dd") == 0) {
01102
          printf(1, "unlink non-empty dd succeeded!\n");
01103
           exit();
01104
        if (unlink("dd/dd") < 0) {</pre>
01105
         printf(1, "unlink dd/dd failed\n");
01106
01107
           exit();
01108
01109
         if(unlink("dd") < 0){</pre>
         printf(1, "unlink dd failed\n");
01110
01111
           exit();
01112
01114
        printf(1, "subdir ok\n");
01115 }
```

Referenced by main().

5.240.2.34 uio()

```
void uio ( )
```

Definition at line 1699 of file usertests.c.

```
01700 {
01701
        #define RTC_ADDR 0x70
#define RTC_DATA 0x71
01702
01703
01704
        ushort port = 0;
01705
        uchar val = 0;
        int pid;
01706
01707
01708
        printf(1, "uio test\n");
01709
        pid = fork();
01710
        if (pid == 0) {
```

```
port = RTC_ADDR;
01712
            val = 0x09; /* year */
           /* http://wiki.osdev.org/Inline_Assembly/Examples */
asm volatile("outb %0,%1"::"a"(val), "d" (port));
01713
01714
01715
           port = RTC_DATA;
           asm volatile("inb %1,%0" : "=a" (val) : "d" (port));
01716
01717
           printf(1, "uio: uio succeeded; test FAILED\n");
01718
            exit();
        } else if(pid < 0) {
  printf (1, "fork failed\n");</pre>
01719
01720
01721
           exit();
01722
01723
         wait();
01724 printf(1, "uio test done\n");
01725 }
```

5.240.2.35 unlinkread()

```
void unlinkread (
     void )
```

Definition at line 657 of file usertests.c.

```
00659
         int fd, fd1;
00660
         printf(1, "unlinkread test\n");
fd = open("unlinkread", O_CREATE | O_RDWR);
if(fd < 0){</pre>
00661
00662
00663
         print(1, "create unlinkread failed\n");
00664
00665
00666
         write(fd, "hello", 5);
00667
00668
        close (fd);
00669
00670
        fd = open("unlinkread", O_RDWR);
00671
         if (fd < 0) {</pre>
00672
         printf(1, "open unlinkread failed\n");
00673
           exit();
00674
00675
         if (unlink("unlinkread") != 0) {
         printf(1, "unlink unlinkread failed\n");
00676
00677
           exit();
00678
00679
        fd1 = open("unlinkread", O_CREATE | O_RDWR);
write(fd1, "yyy", 3);
close(fd1);
00680
00681
00682
00683
00684
         if(read(fd, buf, sizeof(buf)) != 5){
00685
          printf(1, "unlinkread read failed");
00686
           exit();
00687
        if(buf[0] != 'h') {
  printf(1, "unlinkread wrong data\n");
00688
00689
           exit();
00690
00691
         if(write(fd, buf, 10) != 10){
  printf(1, "unlinkread write failed\n");
00692
00693
00694
           exit();
00695
00696
         close(fd);
00697
         unlink("unlinkread");
00698
         printf(1, "unlinkread ok\n");
00699 }
```

5.240.2.36 validateint()

```
void validateint ( int \, * \, p \,)
```

Definition at line 1550 of file usertests.c.

```
01551 {
01552    int res;
01553    asm("mov %%esp, %%ebx\n\t"
01554    "mov %3, %*esp\n\t"
01555    "int %2\n\t"
01556    "mov %%ebx, %*esp":
01557    "=a" (res):
01558    "a" (SYS_sleep), "n" (T_SYSCALL), "c" (p):
01559    "ebx");
```

Referenced by validatetest().

5.240.2.37 validatetest()

```
void validatetest (
    void )
```

Definition at line 1563 of file usertests.c.

```
01564 {
01565
        int hi, pid;
01566
        uint p;
01567
        printf(stdout, "validate test\n");
01568
        hi = 1100 * 1024;
01569
01570
01571
        for(p = 0; p <= (uint)hi; p += 4096){
        if((pid = fork()) == 0) {
   // try to crash the kernel by passing in a badly placed integer
01572
01573
             validateint((int*)p);
01574
01575
             exit();
01576
01577
          sleep(0);
01578
           sleep(0);
01579
           kill(pid);
01580
          wait();
01581
01582
          // try to crash the kernel by passing in a bad string pointer
          if(link("nosuchfile", (char*)p) != -1){
  printf(stdout, "link should not succeed\n");
01583
01584
01585
             exit();
01586
01587
01588
01589 printf(stdout, "validate ok\n");
01590 }
```

5.240.2.38 writetest()

```
void writetest (
    void )
```

Definition at line 139 of file usertests.c.

```
00140 {
         int fd:
00141
00142
         int i;
00143
00144
         printf(stdout, "small file test\n");
00145
         fd = open("small", O_CREATE(O_RDWR);
00146
         if (fd >= 0) {
00147
           printf(stdout, "creat small succeeded; ok\n");
00148
         } else {
         printf(stdout, "error: creat small failed!\n");
00149
00150
           exit();
00151
         for(i = 0; i < 100; i++){
   if(write(fd, "aaaaaaaaaa", 10) != 10){
     printf(stdout, "error: write aa %d new file failed\n", i);</pre>
00152
00153
00154
00155
             exit();
00156
00157
           if(write(fd, "bbbbbbbbbbbbbb", 10) != 10) {
00158
             printf(stdout, "error: write bb %d new file failed\n", i);
00159
              exit();
00160
00161
         printf(stdout, "writes ok\n");
00162
00163
         close(fd);
         fd = open("small", O_RDONLY);
if(fd >= 0){
00164
00165
00166
           printf(stdout, "open small succeeded ok\n");
00167
         } else {
         printf(stdout, "error: open small failed!\n");
exit();
00168
00169
00170
00171
         i = read(fd, buf, 2000);
00172
         if(i == 2000) {
           printf(stdout, "read succeeded ok\n");
00173
00174
         } else {
         printf(stdout, "read failed\n");
exit();
00175
00176
00177
00178
        close (fd);
00179
        if(unlink("small") < 0){
  printf(stdout, "unlink small failed\n");</pre>
00180
00181
00182
. 00184 printf(stdout, "small file test ok\n"); 00185 }
00183
```

Referenced by main().

5.240.2.39 writetest1()

```
void writetest1 (
     void )
```

Definition at line 188 of file usertests.c.

```
00189 {
00190
        int i, fd, n;
00191
       printf(stdout, "big files test\n");
00192
00193
       fd = open("big", O_CREATE(O_RDWR);
00194
       if(fd < 0){
00195
        printf(stdout, "error: creat big failed!\n");
00196
00197
         exit();
00198
00199
00200
       for(i = 0; i < MAXFILE; i++) {</pre>
        ((int*)buf)[0] = i;
```

```
if(write(fd, buf, 512) != 512){
  printf(stdout, "error: write big file failed\n", i);
00203
00204
              exit();
00205
00206
00207
        close(fd);
00209
        fd = open("big", O_RDONLY);
if(fd < 0){</pre>
00210
00211
         printf(stdout, "error: open big failed!\n");
00212
00213
           exit();
00214
00215
00216
        n = 0;
        for(;;) {
  i = read(fd, buf, 512);
  if(i == 0) {
00217
00218
00219
             if(n == MAXFILE - 1) {
              printf(stdout, "read only %d blocks from big", n);
exit();
00221
00222
00223
          break;
} else if(i != 512){
printf(stdout, "read failed %d\n", i);
00224
00225
00226
00227
             exit();
00228
          00229
00230
00231
00232
             exit();
00233
00234
           n++;
00235
        close(fd);
if(unlink("big") < 0){
  printf(stdout, "unlink big failed\n");
  exit();</pre>
00236
00237
00238
00240
00241
       printf(stdout, "big files ok\n");
00242 }
```

Referenced by main().

5.240.3 Variable Documentation

5.240.3.1 buf

```
char buf[8192]
```

Definition at line 11 of file usertests.c.

Referenced by pipe1(), and sharedfd().

5.240.3.2 echoargy

```
char* echoargv[] = { "echo", "ALL", "TESTS", "PASSED", 0 }
```

Definition at line 13 of file usertests.c.

Referenced by exectest().

5.240.3.3 name

char name[3]

Definition at line 12 of file usertests.c.

Referenced by bigdir(), concreate(), create(), createdelete(), createtest(), dirlink(), dirlookup(), fsfull(), initlock(), initsleeplock(), main(), namei(), namei(), namex(), skipelem(), sys_getprocs(), sys_link(), sys_unlink(), trap(), and wc().

5.240.3.4 randstate

```
unsigned long randstate = 1
```

Definition at line 1740 of file usertests.c.

Referenced by rand().

5.240.3.5 stdout

int stdout = 1

Definition at line 14 of file usertests.c.

Referenced by bigargtest(), bsstest(), createtest(), dirtest(), exectest(), exitiputtest(), iputtest(), openiputtest(), openiputtest(), sbrktest(), validatetest(), writetest(), and writetest1().

5.240.3.6 uninit

char uninit[10000]

Definition at line 1593 of file usertests.c.

Referenced by bsstest().

5.241 usertests.c

Go to the documentation of this file.

```
00001 #include "param.h"
00002 #include "types.h"
00003 #include "stat.h"
00004 #include "user.h"
00005 #include "fs.h"
00006 #include "fcntl.h"
00007 #include "syscall.h"
00008 #include "traps.h"
00009 #include "memlayout.h"
00011 char buf[8192];
00012 char name[3];
00013 char *echoargv[] = { "echo", "ALL", "TESTS", "PASSED", 0 };
00014 int stdout = 1;
00015
00016 // does chdir() call iput(p->cwd) in a transaction?
00017 void
00018 iputtest(void)
00019 {
        printf(stdout, "iput test\n");
00020
00021
        if (mkdir("iputdir") < 0){</pre>
        printf(stdout, "mkdir failed\n");
exit();
00023
00024
00025
        if(chdir("iputdir") < 0){
  printf(stdout, "chdir iputdir failed\n");</pre>
00026
00027
00028
          exit();
00030
        if(unlink("../iputdir") < 0){</pre>
00031
         printf(stdout, "unlink ../iputdir failed\n");
00032
           exit();
00033
00034
        if (chdir("/") < 0) {</pre>
         printf(stdout, "chdir / failed\n");
00035
00036
00037
00038
        printf(stdout, "iput test ok\n");
00039 }
00040
00041 // does exit() call iput(p->cwd) in a transaction?
00042 void
00043 exitiputtest (void)
00044 {
00045
        int pid;
00046
00047
        printf(stdout, "exitiput test\n");
00048
00049
        pid = fork();
00050
        if (pid < 0) {</pre>
         printf(stdout, "fork failed\n");
00051
00052
          exit();
00053
        if (pid == 0) {
   if (mkdir("iputdir") < 0) {</pre>
00054
00055
00056
            printf(stdout, "mkdir failed\n");
00057
             exit();
00058
00059
          if(chdir("iputdir") < 0){</pre>
            printf(stdout, "child chdir failed\n");
00061
             exit();
00062
           if(unlink("../iputdir") < 0){</pre>
00063
           printf(stdout, "unlink ../iputdir failed\n");
00064
00065
            exit();
00066
00067
          exit();
00068
00069
        wait();
        printf(stdout, "exitiput test ok\n");
00070
00071 }
00072
00073 // does the error path in open() for attempt to write a
00074 // directory call iput() in a transaction?
00075 // needs a hacked kernel that pauses just after the namei()
00076 // call in sys_open():
00077 //
            if((ip = namei(path)) == 0)
00078 //
              return -1;
00079 //
00080 //
               int i;
              for (i = 0; i < 10000; i++)
00081 //
00082 //
                yield();
```

5.241 usertests.c 495

```
00083 //
00084 void
00085 openiputtest (void)
00086 {
00087
        int pid;
00088
        printf(stdout, "openiput test\n");
        if (mkdir("oidir") < 0) {
  printf(stdout, "mkdir oidir failed\n");</pre>
00090
00091
00092
           exit();
00093
        pid = fork();
00094
00095
        if (pid < 0) {</pre>
00096
          printf(stdout, "fork failed\n");
00097
           exit();
00098
        if (pid == 0) {
00099
          int fd = open("oidir", O_RDWR);
if(fd >= 0) {
00100
00101
00102
            printf(stdout, "open directory for write succeeded\n");
00103
00104
          exit();
00105
00106
00107
        sleep(1);
        if(unlink("oidir") != 0) {
  printf(stdout, "unlink failed\n");
00109
00110
          exit();
00111
00112
        wait();
        printf(stdout, "openiput test ok\n");
00113
00114 }
00115
00116 // simple file system tests
00117
00118 void
00119 opentest (void)
00120 {
00121
00122
        printf(stdout, "open test\n");
fd = open("echo", 0);
if(fd < 0){</pre>
00123
00124
00125
         printf(stdout, "open echo failed!\n");
00126
00127
          exit();
00128
00129
        close(fd);
        fd = open("doesnotexist", 0);
if (fd >= 0) {
00130
00131
          printf(stdout, "open doesnotexist succeeded!\n");
00132
00133
          exit();
00134
00135
        printf(stdout, "open test ok\n");
00136 }
00137
00138 void
00139 writetest (void)
00140 {
00141
00142
        int i;
00143
        printf(stdout, "small file test\n");
00144
00145
        fd = open("small", O_CREATE(O_RDWR);
00146
        if (fd >= 0) {
00147
          printf(stdout, "creat small succeeded; ok\n");
00148
        } else {
00149
          printf(stdout, "error: creat small failed!\n");
00150
          exit();
00151
00152
        for(i = 0; i < 100; i++){
00153
          if(write(fd, "aaaaaaaaaa", 10) != 10) {
00154
            printf(stdout, "error: write aa %d new file failed\n", i);
00155
             exit();
00156
          if(write(fd, "bbbbbbbbbb", 10) != 10){
  printf(stdout, "error: write bb %d new file failed\n", i);
00157
00158
00159
             exit();
00160
00161
        printf(stdout, "writes ok\n");
00162
00163
        close(fd);
        fd = open("small", O_RDONLY);
00164
00165
00166
          printf(stdout, "open small succeeded ok\n");
00167
        } else {
          printf(stdout, "error: open small failed!\n");
00168
00169
           exit();
```

```
00170
00171
        i = read(fd, buf, 2000);
00172
        if(i == 2000) {
          printf(stdout, "read succeeded ok\n");
00173
00174
        } else {
        printf(stdout, "read failed\n");
exit();
00175
00176
00177
00178
        close(fd);
00179
        if(unlink("small") < 0){
  printf(stdout, "unlink small failed\n");</pre>
00180
00181
00182
          exit();
00183
00184
        printf(stdout, "small file test ok\n");
00185 }
00186
00187 void
00188 writetest1(void)
00189 {
00190
        int i, fd, n;
00191
        printf(stdout, "big files test\n");
00192
00193
00194
        fd = open("big", O_CREATE|O_RDWR);
00195
        if (fd < 0) {
00196
          printf(stdout, "error: creat big failed!\n");
00197
          exit();
00198
00199
00200
        for(i = 0; i < MAXFILE; i++) {</pre>
00201
          ((int*)buf)[0] = i;
           if(write(fd, buf, 512) != 512){
  printf(stdout, "error: write big file failed\n", i);
00202
00203
00204
             exit();
00205
00206
        }
00207
00208
        close(fd);
00209
00210
        fd = open("big", O_RDONLY);
        if (fd < 0) {
00211
          printf(stdout, "error: open big failed!\n");
00212
00213
          exit();
00214
00215
00216
        n = 0;
00217
        for(;;) {
00218
          i = read(fd, buf, 512);
          if(i == 0){
00219
            if(n == MAXFILE - 1) {
   printf(stdout, "read only %d blocks from big", n);
00221
00222
               exit();
00223
00224
          break;
} else if(i != 512){
  printf(stdout, "read failed %d\n", i);
00225
00227
             exit();
00228
          00229
00230
00231
00232
             exit();
00233
00234
          n++;
00235
00236
        close(fd);
        if(unlink("big") < 0){
  printf(stdout, "unlink big failed\n");</pre>
00237
00238
00239
          exit();
00240
        printf(stdout, "big files ok\n");
00241
00242 }
00243
00244 void
00245 createtest(void)
00246 {
00247
00248
00249
        printf(stdout, "many creates, followed by unlink test\n");
00250
00251
        name[0] = 'a';
00252
        name[2] = ' \setminus 0';
        for(i = 0; i < 52; i++) {
name[1] = '0' + i;
00253
00254
          fd = open(name, O_CREATE|O_RDWR);
close(fd);
00255
00256
```

5.241 usertests.c 497

```
00257
         name[0] = 'a';
name[2] = '\0';
00258
00259
         for(i = 0; i < 52; i++) {
name[1] = '0' + i;
00260
00261
00262
           unlink(name);
00263
00264
         printf(stdout, "many creates, followed by unlink; ok\n");
00265 }
00266
00267 void dirtest (void)
00268 {
00269
         printf(stdout, "mkdir test\n");
00270
00271
         if (mkdir("dir0") < 0) {</pre>
          printf(stdout, "mkdir failed\n");
00272
00273
            exit();
00274
         if(chdir("dir0") < 0){
  printf(stdout, "chdir dir0 failed\n");</pre>
00276
00277
00278
            exit();
00279
00280
00281
         if(chdir("..") < 0){</pre>
         printf(stdout, "chdir .. failed\n");
00283
            exit();
00284
00285
         if(unlink("dir0") < 0){
  printf(stdout, "unlink dir0 failed\n");
  evit():</pre>
00286
00287
00288
           exit();
00290 printf(stdout, "mkdir test ok\n"); 00291 }
00292
00293 void
00294 exectest (void)
00295 {
printf(stdout, "exec test\n");
00297 if(exec("echo", echoargv) < 0){
00298 printf(stdout, "exec echo failed\n");
00299
            exit();
00300
00301 }
00302
00303 // simple fork and pipe read/write
00304
00305 void
00306 pipe1(void)
00307 {
00308
         int fds[2], pid;
00309
         int seq, i, n, cc, total;
00310
         if(pipe(fds) != 0) {
  printf(1, "pipe() failed\n");
  evit().
00311
00312
            exit();
00314
00315
         pid = fork();
         seq = 0;
if (pid == 0) {
00316
00317
           close(fds[0]);
00318
00319
            for (n = 0; n < 5; n++) {
             for(i = 0; i < 1033; i++)
buf[i] = seq++;
00320
00321
              if(write(fds[1], buf, 1033) != 1033){
  printf(1, "pipe1 oops 1\n");
00322
00323
00324
                 exit();
00325
              }
00326
00327
            exit();
         } else if(pid > 0) {
00328
           close(fds[1]);
00329
00330
            total = 0:
00331
            cc = 1;
00332
            while ((n = read(fds[0], buf, cc)) > 0) {
00333
              for(i = 0; i < n; i++){</pre>
                if((buf[i] & 0xff) != (seq++ & 0xff)){
  printf(1, "pipe1 oops 2\n");
00334
00335
00336
                    return:
00337
                 }
00338
00339
               total += n;
00340
               cc = cc * 2;
              if(cc > sizeof(buf))
  cc = sizeof(buf);
00341
00342
00343
```

```
if(total != 5 * 1033) {
00345
            printf(1, "pipel oops 3 total %d\n", total);
00346
              exit();
00347
00348
           close(fds[0]);
00349
           wait();
00350
         } else {
00351
          printf(1, "fork() failed\n");
00352
           exit();
00353
        printf(1, "pipe1 ok\n");
00354
00355 }
00356
00357 // meant to be run w/ at most two CPUs
00358 void
00359 preempt(void) 00360 {
00361
        int pid1, pid2, pid3;
int pfds[2];
00362
00363
00364
         printf(1, "preempt: ");
         pid1 = fork();
if (pid1 == 0)
00365
00366
00367
          for(;;)
00368
             ;
00369
00370
         pid2 = fork();
00371
         if(pid2 == 0)
00372
         for(;;)
00373
            ;
00374
00375
         pipe(pfds);
00376
         pid3 = fork();
00377
         if(pid3 == 0){
00378
          close(pfds[0]);
           if (write(pfds[1], "x", 1) != 1)
  printf(1, "preempt write error");
close(pfds[1]);
00379
00380
00381
00382
           for(;;)
00383
00384
00385
00386
         close(pfds[1]);
         if (read(pfds[0], buf, sizeof(buf)) != 1) {
00387
00388
         printf(1, "preempt read error");
00389
00390
        close(pfds[0]);
printf(1, "kill...");
00391
00392
         kill(pid1);
00393
00394
         kill(pid2);
00395
         kill(pid3);
00396
         printf(1, "wait... ");
00397
         wait();
00398
         wait();
00399
        wait();
00400
        printf(1, "preempt ok\n");
00401 }
00402
00403 // try to find any races between exit and wait
00404 void
00405 exitwait(void)
00406 {
00407
        int i, pid;
00408
        for(i = 0; i < 100; i++){
  pid = fork();
  if(pid < 0){
    printf(1, "fork failed\n");</pre>
00409
00410
00411
00412
00413
             return;
00414
00415
           if(pid){
            if(wait() != pid) {
  printf(1, "wait wrong pid\n");
00416
00417
00418
                return;
00419
00420
           } else
00421
             exit();
           }
00422
00423
        printf(1, "exitwait ok\n");
00424
00425 }
00426
00427 void
00428 mem(void)
00429 {
00430
        void *m1, *m2;
```

```
00431
        int pid, ppid;
00432
         printf(1, "mem test\n");
00433
00434
         ppid = getpid();
         if((pid = fork()) == 0) {
00435
          m1 = 0;
while((m2 = malloc(10001)) != 0){
00436
00438
             *(char**)m2 = m1;
00439
            m1 = m2;
00440
           while (m1) {
00441
            m2 = *(char**)m1;
00442
00443
             free(m1);
00444
            m1 = m2;
00445
00446
           m1 = malloc(1024*20);
           if(m1 == 0){
  printf(1, "couldn't allocate mem?!!\n");
  kill(ppid);
00447
00448
00450
             exit();
00451
00452
           free(m1);
           printf(1, "mem ok\n");
00453
00454
           exit();
00455
        } else
00456
          wait();
00457
00458 }
00459
00460 // More file system tests
00461
00462 // two processes write to the same file descriptor
00463 // is the offset shared? does inode locking work?
00464 void
00465 sharedfd(void)
00466 {
        int fd, pid, i, n, nc, np;
char buf[10];
00467
00469
00470
        printf(1, "sharedfd test\n");
00471
00472
        unlink("sharedfd"):
        fd = open("sharedfd", O_CREATE|O_RDWR);
if(fd < 0){</pre>
00473
00474
00475
         printf(1, "fstests: cannot open sharedfd for writing");
00476
00477
        pid = fork();
00478
         memset(buf, pid==0?'c':'p', sizeof(buf));
for(i = 0; i < 1000; i++){
00479
00480
          if(write(fd, buf, sizeof(buf)) != sizeof(buf)){
  printf(1, "fstests: write sharedfd failed\n");
00482
00483
             break;
00484
          }
00485
00486
         if (pid == 0)
00487
          exit();
00488
00489
          wait();
00490
         close (fd);
        fd = open("sharedfd", 0);
if (fd < 0) {</pre>
00491
00492
         printf(1, "fstests: cannot open sharedfd for reading\n");
00493
00494
           return;
00495
00496
        nc = np = 0;
        while((n = read(fd, buf, sizeof(buf))) > 0){
  for(i = 0; i < sizeof(buf); i++){
    if(buf[i] == 'c')</pre>
00497
00498
00499
00500
                nc++;
00501
              if(buf[i] == 'p')
00502
               np++;
00503
          }
00504
00505
        close(fd);
00506
        unlink("sharedfd");
00507
        if (nc == 10000 && np == 10000) {
00508
          printf(1, "sharedfd ok\n");
00509
        } else {
         printf(1, "sharedfd oops %d %d\n", nc, np);
00510
00511
           exit();
00512
        }
00513 }
00514
\tt 00515 // four processes write different files at the same
00516 // time, to test block allocation.
00517 void
```

```
00518 fourfiles (void)
00519 {
         int fd, pid, i, j, n, total, pi;
char *names[] = { "f0", "f1", "f2", "f3" };
00520
00521
00522
         char *fname:
00523
00524
         printf(1, "fourfiles test\n");
00525
00526
         for(pi = 0; pi < 4; pi++) {</pre>
00527
           fname = names[pi];
           unlink(fname);
00528
00529
00530
           pid = fork();
            if (pid < 0) {
  printf(1, "fork failed\n");</pre>
00531
00532
00533
              exit();
00534
00535
00536
            if (pid == 0) {
00537
             fd = open(fname, O_CREATE | O_RDWR);
00538
              if(fd < 0){
                printf(1, "create failed\n");
00539
00540
                exit();
00541
00542
00543
              memset(buf, '0'+pi, 512);
for(i = 0; i < 12; i++){</pre>
00544
               if((n = write(fd, buf, 500)) != 500){
    printf(1, "write failed %d\n", n);
00545
00546
00547
                   exit();
00548
                }
00549
00550
              exit();
00551
           }
         }
00552
00553
         for(pi = 0; pi < 4; pi++) {</pre>
00554
           wait();
00556
00557
00558
         for(i = 0; i < 2; i++){</pre>
          fname = names[i];
00559
           fd = open(fname, 0);
total = 0;
00560
00561
           total = 0;
while((n = read(fd, buf, sizeof(buf))) > 0){
  for(j = 0; j < n; j++){
    if(buf[j] != '0'+i){
      printf(1, "wrong char\n");
}</pre>
00562
00563
00564
00565
00566
                   exit();
00567
                }
00568
00569
              total += n;
00570
00571
            close(fd);
if(total != 12*500){
00572
00573
             printf(1, "wrong length %d\n", total);
00574
              exit();
00575
00576
            unlink(fname);
00577
00578
         printf(1, "fourfiles ok\n");
00579
00580 }
00581
00582 // four processes create and delete different files in same directory
00583 void
00584 createdelete(void)
00585 {
00586
        enum { N = 20 };
00587
         int pid, i, fd, pi;
00588
         char name[32];
00589
         printf(1, "createdelete test\n");
00590
00591
         for(pi = 0; pi < 4; pi++) {
  pid = fork();</pre>
00592
00593
            if(pid < 0) {
  printf(1, "fork failed\n");</pre>
00594
00595
00596
              exit();
00597
           }
00598
00599
            if(pid == 0){
             name[0] = 'p' + pi;
name[2] = '\0';
00600
00601
              for (i = 0; i < N; i++) {
  name[1] = '0' + i;</pre>
00602
00603
                 fd = open(name, O_CREATE | O_RDWR);
00604
```

```
if(fd < 0){</pre>
00606
                  printf(1, "create failed\n");
00607
                    exit();
00608
00609
                 close (fd);
                 if(i > 0 && (i % 2 ) == 0){
  name[1] = '0' + (i / 2);
00610
00611
00612
                   if(unlink(name) < 0){</pre>
00613
                    printf(1, "unlink failed\n");
00614
                      exit();
                   }
00615
00616
                 }
00617
00618
              exit();
00619
            }
00620
00621
00622
         for(pi = 0; pi < 4; pi++) {</pre>
00623
           wait();
00624
00625
00626
         name[0] = name[1] = name[2] = 0;
         for(i = 0; i < N; i++) {
  for(pi = 0; pi < 4; pi++) {
    name[0] = 'p' + pi;
    name[1] = '0' + i;
</pre>
00627
00628
00629
00630
00631
               fd = open(name, 0);
              if((i == 0 || i >= N/2) && fd < 0){
  printf(1, "oops createdelete %s didn't exist\n", name);</pre>
00632
00633
              exit();
} else if((i >= 1 && i < N/2) && fd >= 0){
00634
00635
00636
                printf(1, "oops createdelete %s did exist\n", name);
00637
                 exit();
00638
00639
               if(fd >= 0)
                 close (fd);
00640
00641
           }
00642
00643
         for(i = 0; i < N; i++) {
  for(pi = 0; pi < 4; pi++) {
    name[0] = 'p' + i;
    name[1] = '0' + i;</pre>
00644
00645
00646
00647
00648
              unlink(name);
00649
00650
00651
        printf(1, "createdelete ok\n");
00652
00653 }
00654
00655 // can I unlink a file and still read it?
00656 void
00657 unlinkread(void)
00658 {
         int fd, fd1:
00659
00660
00661
         printf(1, "unlinkread test\n");
00662
         fd = open("unlinkread", O_CREATE | O_RDWR);
         if(fd < 0){
  printf(1, "create unlinkread failed\n");</pre>
00663
00664
00665
           exit();
00666
00667
         write(fd, "hello", 5);
00668
         close(fd);
00669
00670
         fd = open("unlinkread", O_RDWR);
         <u>if</u>(fd < 0){
00671
          printf(1, "open unlinkread failed\n");
00672
00673
            exit();
00674
00675
         if(unlink("unlinkread") != 0){
00676
           printf(1, "unlink unlinkread failed\n");
00677
           exit();
00678
00679
         fd1 = open("unlinkread", O_CREATE | O_RDWR);
write(fd1, "yyy", 3);
00680
00681
00682
         close(fd1);
00683
         if(read(fd, buf, sizeof(buf)) != 5) {
  printf(1, "unlinkread read failed");
00684
00685
00686
            exit();
00687
         if(buf[0] != 'h') {
  printf(1, "unlinkread wrong data\n");
00688
00689
00690
            exit();
00691
```

```
if (write(fd, buf, 10) != 10) {
        printf(1, "unlinkread write failed\n");
00693
00694
          exit();
00695
00696
        close (fd):
00697
        unlink("unlinkread");
        printf(1, "unlinkread ok\n");
00699 }
00700
00701 void
00702 linktest (void)
00703 {
00704
        int fd;
00705
00706
       printf(1, "linktest\n");
00707
        unlink("lf1");
00708
00709
        unlink("lf2");
00711
        fd = open("lf1", O_CREATE|O_RDWR);
        if (fd < 0) {
00712
          printf(1, "create lf1 failed\n");
00713
00714
          exit();
00715
        if(write(fd, "hello", 5) != 5){
  printf(1, "write lfl failed\n");
00716
00717
00718
          exit();
00719
00720
        close (fd);
00721
        if(link("lf1", "lf2") < 0){
  printf(1, "link lf1 lf2 failed\n");</pre>
00722
00723
00724
          exit();
00725
00726
        unlink("lf1");
00727
        00728
00729
00730
          exit();
00731
00732
        fd = open("1f2", 0);
00733
        if (fd < 0) {
00734
         printf(1, "open 1f2 failed\n");
00735
00736
          exit();
00737
        if(read(fd, buf, sizeof(buf)) != 5) {
  printf(1, "read lf2 failed\n");
00738
00739
00740
         exit();
00741
00742
        close(fd);
00743
        00744
00745
00746
          exit();
00747
00748
00749
        unlink("1f2");
        if(link("lf2", "lf1") >= 0){
  printf(1, "link non-existant succeeded! oops\n");
00750
00751
00752
          exit();
00753
00754
        if(link(".", "lf1") >= 0){
   printf(1, "link . lf1 succeeded! oops\n");
00755
00756
00757
          exit();
00758
00759
       printf(1, "linktest ok\n");
00760
00761 }
00762
00763 // test concurrent create/link/unlink of the same file
00764 void
00765 concreate (void)
00766 {
00767
        char file[3];
00768
        int i, pid, n, fd;
00769
        char fa[40];
00770
        struct {
         ushort inum;
00771
00772
          char name[14];
00773
        } de;
00774
        printf(1, "concreate test\n");
00775
        file[0] = 'C';
file[2] = '\0';
for(i = 0; i < 40; i++){
00776
00777
00778
```

```
file[1] = '0' + i;
           unlink(file);
pid = fork();
00780
00781
           if(pid && (i % 3) == 1) {
  link("CO", file);
} else if(pid == 0 && (i % 5) == 1) {
00782
00783
00784
00785
             link("CO", file);
00786
           } else {
00787
              fd = open(file, O_CREATE | O_RDWR);
             if(fd < 0){
  printf(1, "concreate create %s failed\n", file);</pre>
00788
00789
00790
               exit();
00791
00792
             close(fd);
00793
00794
           if(pid == 0)
00795
             exit();
00796
           else
00797
             wait();
00798
00799
        memset(fa, 0, sizeof(fa));
fd = open(".", 0);
00800
00801
         n = 0;
00802
00803
         while (read(fd, &de, sizeof(de)) > 0) {
00804
          if(de.inum == 0)
00805
           if(de.name[0] == 'C' && de.name[2] == '\0'){
  i = de.name[1] - '0';
00806
00807
             if(i < 0 || i >= sizeof(fa)){
  printf(1, "concreate weird file %s\n", de.name);
00808
00809
00810
                exit();
00811
00812
              <u>if</u>(fa[i]){
               printf(1, "concreate duplicate file %s\n", de.name);
00813
00814
                exit();
00815
             fa[i] = 1;
00817
             n++;
00818
00819
00820
        close (fd);
00821
00822
         if(n != 40){
00823
         printf(1, "concreate not enough files in directory listing\n");
           exit();
00824
00825
00826
00827
         for (i = 0; i < 40; i++) {
          file[1] = '0' + i;
pid = fork();
00828
00829
           if(pid < 0) {
  printf(1, "fork failed\n");</pre>
00830
00831
00832
             exit();
00833
00834
           if(((i % 3) == 0 && pid == 0) ||
              ((i % 3) == 1 && pid != 0)){
00836
             close(open(file, 0));
00837
             close(open(file, 0));
00838
             close(open(file, 0));
00839
             close(open(file, 0));
00840
           } else {
00841
             unlink(file);
00842
             unlink(file);
00843
             unlink(file);
00844
             unlink(file);
00845
00846
           if(pid == 0)
00847
            exit();
00848
           else
00849
             wait();
00850
00851
        printf(1, "concreate ok\n");
00852
00853 }
00854
00855 // another concurrent link/unlink/create test,
00856 // to look for deadlocks.
00857 void
00858 linkunlink()
00859 {
00860
         int pid, i;
00861
        printf(1, "linkunlink test\n");
00862
00863
        unlink("x");
00864
00865
        pid = fork();
```

```
if(pid < 0) {
  printf(1, "fork failed\n");</pre>
00867
00868
             exit();
00869
00870
          unsigned int x = (pid ? 1 : 97);
for(i = 0; i < 100; i++) {</pre>
00871
00873
            x = x * 1103515245 + 12345;
            if((x % 3) == 0) {
  close(open("x", O_RDWR | O_CREATE));
} else if((x % 3) == 1) {
  link("cat", "x");
00874
00875
00876
00877
00878
            } else {
00879
              unlink("x");
00880
00881
00882
00883
          if (pid)
00884
            wait();
00885
         else
00886
          exit();
00887
00888 printf(1, "linkunlink ok\n");
00889 }
00890
00891 // directory that uses indirect blocks
00892 void
00893 bigdir(void)
00894 {
00895
          int i, fd;
00896
         char name[10];
00897
00898
         printf(1, "bigdir test\n");
00899
          unlink("bd");
00900
          fd = open("bd", O_CREATE);
00901
00902
          if (fd < 0) {
           printf(1, "bigdir create failed\n");
00904
            exit();
00905
00906
         close (fd);
00907
          for(i = 0; i < 500; i++){</pre>
00908
           name[0] = 'x';

name[1] = '0' + (i / 64);

name[2] = '0' + (i % 64);
00909
00910
00911
            name[2] = 'U' + (1 % 64);
name[3] = '\0';
if(link("bd", name) != 0){
  printf(1, "bigdir link failed\n");
00912
00913
00914
00915
               exit();
00916
            }
00917
         }
00918
         unlink("bd");

for(i = 0; i < 500; i++){

   name[0] = 'x';

   name[1] = '0' + (i / 64);

   name[2] = '0' + (i % 64);

   name[3] = '\0';

   if(unlink(name)) | -0 0 (
00919
00920
00921
00922
00923
00924
            if(unlink(name) != 0) {
  printf(1, "bigdir unlink failed");
00925
00926
00927
               exit();
00928
            }
00929
00930
00931
         printf(1, "bigdir ok\n");
00932 }
00933
00934 void
00935 subdir(void)
00936 {
00937
          int fd, cc;
00938
         printf(1, "subdir test\n");
00939
00940
00941
          unlink("ff");
00942
          if (mkdir("dd") != 0) {
00943
          printf(1, "subdir mkdir dd failed\n");
00944
            exit();
00945
00946
00947
          fd = open("dd/ff", O_CREATE | O_RDWR);
00948
          if(fd < 0){
           printf(1, "create dd/ff failed\n");
00949
00950
            exit();
00951
00952
         write(fd, "ff", 2);
```

```
close(fd);
00954
        if(unlink("dd") >= 0){
00955
         printf(1, "unlink dd (non-empty dir) succeeded!\n");
00956
00957
          exit();
00958
00959
00960
         if (mkdir("/dd/dd") != 0) {
00961
         printf(1, "subdir mkdir dd/dd failed\n");
00962
           exit();
00963
00964
00965
        fd = open("dd/dd/ff", O_CREATE | O_RDWR);
00966
        if (fd < 0) {
00967
          printf(1, "create dd/dd/ff failed\n");
00968
           exit();
00969
00970
        write(fd, "FF", 2);
00971
        close(fd);
00972
00973
        fd = open("dd/dd/../ff", 0);
00974
        if (fd < 0) {</pre>
         printf(1, "open dd/dd/../ff failed\n");
00975
00976
          exit();
00977
00978
        cc = read(fd, buf, sizeof(buf));
00979
         if(cc != 2 || buf[0] != 'f'){
          printf(1, "dd/dd/../ff wrong content\n");
00980
00981
           exit();
00982
00983
        close (fd);
00984
00985
         if(link("dd/dd/fff", "dd/dd/fffff") != 0){
00986
          printf(1, "link dd/dd/ff dd/dd/ffff failed\n");
00987
           exit();
00988
00989
        if(unlink("dd/dd/ff") != 0){
00991
         printf(1, "unlink dd/dd/ff failed\n");
00992
00993
        if(open("dd/dd/ff", O_RDONLY) >= 0){
  printf(1, "open (unlinked) dd/dd/ff succeeded\n");
00994
00995
00996
          exit();
00997
00998
        if(chdir("dd") != 0) {
  printf(1, "chdir dd failed\n");
00999
01000
01001
          exit();
01002
01003
        if (chdir("dd/../../dd") != 0) {
01004
          printf(1, "chdir dd/../../dd failed\n");
           exit();
01005
01006
        if(chdir("dd/../../dd") != 0){
  printf(1, "chdir dd/../../dd failed\n");
01007
01008
01009
01010
        if(chdir("./..") != 0) {
  printf(1, "chdir ./.. failed\n");
01011
01012
01013
          exit();
01014
01015
01016
        fd = open("dd/dd/fffff", 0);
01017
         if (fd < 0) {</pre>
01018
          printf(1, "open dd/dd/ffff failed\n");
01019
           exit();
01020
01021
        if (read(fd, buf, sizeof(buf)) != 2) {
         printf(1, "read dd/dd/ffff wrong len\n");
01022
01023
01024
01025
        close (fd);
01026
        if(open("dd/dd/ff", O_RDONLY) >= 0){
  printf(1, "open (unlinked) dd/dd/ff succeeded!\n");
01027
01028
01029
           exit();
01030
01031
        if(open("dd/ff/ff", O_CREATE|O_RDWR) >= 0){
01032
          printf(1, "create dd/ff/ff succeeded!\n");
01033
01034
           exit();
01035
01036
         if(open("dd/xx/ff", O_CREATE(O_RDWR) >= 0){
01037
          printf(1, "create dd/xx/ff succeeded!\n");
01038
           exit();
01039
```

```
if(open("dd", O_CREATE) >= 0) {
  printf(1, "create dd succeeded!\n");
01041
01042
            exit();
01043
         if(open("dd", O_RDWR) >= 0) {
  printf(1, "open dd rdwr succeeded!\n");
  ovit();
01044
01045
01046
           exit();
01047
         if(open("dd", O_WRONLY) >= 0) {
  printf(1, "open dd wronly succeeded!\n");
01048
01049
01050
           exit();
01051
         if(link("dd/ff/ff", "dd/dd/xx") == 0){
01052
01053
           printf(1, "link dd/ff/ff dd/dd/xx succeeded!\n");
01054
            exit();
01055
         if(link("dd/xx/ff", "dd/dd/xx") == 0){
  printf(1, "link dd/xx/ff dd/dd/xx succeeded!\n");
01056
01057
01058
01059
         if(link("dd/fff", "dd/dd/fffff") == 0){
  printf(1, "link dd/ff dd/dd/ffff succeeded!\n");
01060
01061
01062
           exit();
01063
01064
         if (mkdir("dd/ff/ff") == 0) {
01065
           printf(1, "mkdir dd/ff/ff succeeded!\n");
01066
            exit();
01067
         if(mkdir("dd/xx/ff") == 0){
  printf(1, "mkdir dd/xx/ff succeeded!\n");
01068
01069
01070
           exit();
01071
01072
         if (mkdir("dd/dd/fffff") == 0) {
01073
           printf(1, "mkdir dd/dd/ffff succeeded!\n");
01074
           exit();
01075
01076
         if (unlink("dd/xx/ff") == 0) {
01077
           printf(1, "unlink dd/xx/ff succeeded!\n");
01078
           exit();
01079
01080
         if (unlink("dd/ff/ff") == 0) {
          printf(1, "unlink dd/ff/ff succeeded!\n");
01081
01082
           exit():
01083
         if(chdir("dd/ff") == 0) {
  printf(1, "chdir dd/ff succeeded!\n");
01084
01085
01086
           exit();
01087
         if (chdir("dd/xx") == 0) {
01088
         printf(1, "chdir dd/xx succeeded!\n");
01089
01090
           exit();
01091
01092
         if(unlink("dd/dd/fffff") != 0) {
  printf(1, "unlink dd/dd/ff failed\n");
01093
01094
01095
           exit();
01096
01097
         if (unlink("dd/ff") != 0) {
01098
          printf(1, "unlink dd/ff failed\n");
01099
            exit();
01100
         if (unlink("dd") == 0) {
01101
01102
           printf(1, "unlink non-empty dd succeeded!\n");
01103
01104
         if(unlink("dd/dd") < 0){
  printf(1, "unlink dd/dd failed\n");</pre>
01105
01106
01107
           exit();
01108
         if(unlink("dd") < 0){</pre>
01109
01110
          printf(1, "unlink dd failed\n");
01111
            exit();
01112
01113
         printf(1, "subdir ok\n");
01114
01115 }
01116
01117 // test writes that are larger than the log.
01118 void
01119 bigwrite(void)
01120 {
         int fd, sz;
01122
        printf(1, "bigwrite test\n");
01123
01124
         unlink("bigwrite");
for(sz = 499; sz < 12*512; sz += 471){
01125
01126
```

```
fd = open("bigwrite", O_CREATE | O_RDWR);
          if(fd < 0){
01128
            printf(1, "cannot create bigwrite\n");
01129
01130
            exit();
01131
01132
          int i:
01133
          for(i = 0; i < 2; i++){
01134
            int cc = write(fd, buf, sz);
            if(cc != sz){
   printf(1, "write(%d) ret %d\n", sz, cc);
01135
01136
01137
               exit();
01138
            }
01139
01140
          close(fd);
01141
          unlink("bigwrite");
01142
01143
        printf(1, "bigwrite ok\n");
01144
01145 }
01146
01147 void
01148 bigfile(void)
01149 {
01150
        int fd, i, total, cc;
01151
01152
       printf(1, "bigfile test\n");
01153
01154
        unlink("bigfile");
        fd = open("bigfile", O_CREATE | O_RDWR);
01155
        if (fd < 0) {
01156
          printf(1, "cannot create bigfile");
01157
01158
          exit();
01159
01160
        for(i = 0; i < 20; i++) {</pre>
01161
          memset(buf, i, 600);
          if(write(fd, buf, 600) != 600){
  printf(1, "write bigfile failed\n");
01162
01163
01164
            exit();
01165
          }
01166
01167
        close(fd);
01168
        fd = open("bigfile", 0);
if(fd < 0){</pre>
01169
01170
01171
         printf(1, "cannot open bigfile\n");
01172
          exit();
01173
01174
        total = 0;
        for(i = 0; ; i++) {
01175
         cc = read(fd, buf, 300);
01176
01177
          if(cc < 0){
01178
            printf(1, "read bigfile failed\n");
01179
             exit();
01180
          if(cc == 0)
01181
01182
            break;
           if(cc != 300){
01184
            printf(1, "short read bigfile\n");
01185
            exit();
01186
          if(buf[0] != i/2 || buf[299] != i/2) {
  printf(1, "read bigfile wrong data\n");
01187
01188
01189
            exit();
01190
01191
          total += cc;
01192
01193
        close(fd);
        if (total != 20*600) {
01194
          printf(1, "read bigfile wrong total\n");
01195
01196
          exit();
01197
01198
        unlink("bigfile");
01199
        printf(1, "bigfile test ok\n");
01200
01201 }
01202
01203 void
01204 fourteen(void)
01205 {
01206
        int fd:
01207
01208
        // DIRSIZ is 14.
01209
        printf(1, "fourteen test\n");
01210
        if(mkdir("12345678901234") != 0){
01211
          printf(1, "mkdir 12345678901234 failed\n");
01212
01213
          exit();
```

```
01214
01215
        if (mkdir("12345678901234/123456789012345") != 0) {
          printf(1, "mkdir 12345678901234/123456789012345 failed\n");
01216
01217
          exit();
01218
        fd = open("123456789012345/123456789012345/123456789012345", O_CREATE);
01219
01220
        if(fd < 0){</pre>
01221
          printf(1, "create 123456789012345/123456789012345/123456789012345 failed\n");
01222
          exit();
01223
01224
        close(fd);
        fd = open("12345678901234/12345678901234/12345678901234", 0);
01225
01226
        if(fd < 0){
01227
         printf(1, "open 12345678901234/12345678901234/12345678901234 failed\n");
01228
          exit();
01229
01230
        close (fd):
01231
        if (mkdir("12345678901234/12345678901234") == 0) {
        printf(1, "mkdir 12345678901234/12345678901234 succeeded!\n");
01233
01234
01235
        if(mkdir("123456789012345/12345678901234") == 0) {
01236
          printf(1, "mkdir 12345678901234/123456789012345 succeeded!\n");
01237
01238
          exit();
01239
01240
01241
        printf(1, "fourteen ok\n");
01242 }
01243
01244 void
01245 rmdot (void)
01246 {
        printf(1, "rmdot test\n");
if(mkdir("dots") != 0){
  printf(1, "mkdir dots failed\n");
01247
01248
01249
          exit();
01250
01251
01252
        if (chdir("dots") != 0) {
01253
         printf(1, "chdir dots failed\n");
01254
          exit();
01255
        if(unlink(".") == 0) {
  printf(1, "rm . worked!\n");
  evit():
01256
01257
01258
          exit();
01259
        if(unlink("..") == 0) {
  printf(1, "rm .. worked!\n");
01260
01261
01262
          exit();
01263
        if(chdir("/") != 0) {
  printf(1, "chdir / failed\n");
01264
01265
01266
           exit();
01267
        if (unlink("dots/.") == 0) {
01268
         printf(1, "unlink dots/. worked!\n");
01269
01270
01271
        if(unlink("dots/..") == 0) {
  printf(1, "unlink dots/.. worked!\n");
01272
01273
01274
          exit():
01275
        if (unlink("dots") != 0) {
        printf(1, "unlink dots failed!\n");
01277
01278
           exit();
01279
        printf(1, "rmdot ok\n");
01280
01281 }
01282
01283 void
01284 dirfile (void)
01285 {
01286
        int fd;
01287
01288
        printf(1, "dir vs file\n");
01289
01290
        fd = open("dirfile", O_CREATE);
01291
        if(fd < 0){</pre>
         printf(1, "create dirfile failed\n");
01292
01293
          exit();
01294
01295
        close(fd);
01296
        if(chdir("dirfile") == 0){
          printf(1, "chdir dirfile succeeded!\n");
01297
01298
          exit();
01299
01300
        fd = open("dirfile/xx", 0);
```

```
if (fd >= 0) {
01302
         printf(1, "create dirfile/xx succeeded!\n");
01303
           exit();
01304
        fd = open("dirfile/xx", O_CREATE);
01305
01306
        if(fd >= 0){
          printf(1, "create dirfile/xx succeeded!\n");
01307
01308
01309
        if(mkdir("dirfile/xx") == 0){
  printf(1, "mkdir dirfile/xx succeeded!\n");
01310
01311
01312
           exit();
01313
01314
        if(unlink("dirfile/xx") == 0){
01315
          printf(1, "unlink dirfile/xx succeeded!\n");
01316
01317
        if(link("README", "dirfile/xx") == 0) {
  printf(1, "link to dirfile/xx succeeded!\n");
01318
01319
01320
           exit();
01321
        if(unlink("dirfile") != 0){
01322
         printf(1, "unlink dirfile failed!\n");
01323
01324
           exit();
01325
01326
01327
        fd = open(".", O_RDWR);
        if(fd >= 0) {
  printf(1, "open . for writing succeeded!\n");
01328
01329
01330
          exit();
01331
        fd = open(".", 0);
if(write(fd, "x", 1) > 0){
  printf(1, "write . succeeded!\n");
01332
01333
01334
01335
          exit();
01336
01337
        close(fd);
01338
01339
        printf(1, "dir vs file OK\n");
01340 }
01341
01342 // test that iput() is called at the end of \_namei()
01343 void
01344 iref(void)
01345 {
01346
        int i, fd;
01347
01348
        printf(1, "empty file name\n");
01349
01350
        // the 50 is NINODE
        for(i = 0; i < 50 + 1; i++) {
  if(mkdir("irefd") != 0) {</pre>
01351
01352
01353
             printf(1, "mkdir irefd failed\n");
01354
             exit();
01355
01356
           if(chdir("irefd") != 0) {
01357
            printf(1, "chdir irefd failed\n");
01358
             exit();
01359
01360
           mkdir("");
01361
           link("README", "");
fd = open("", O_CREATE);
01362
01363
01364
           if(fd >= 0)
01365
             close(fd);
01366
           fd = open("xx", O_CREATE);
01367
           if(fd >= 0)
             close(fd);
01368
01369
           unlink("xx");
01370
01371
        chdir("/");
01372
       printf(1, "empty file name OK\n");
01373
01374 }
01375
01376 // test that fork fails gracefully
01377 // the forktest binary also does this, but it runs out of proc entries first.
01378 \!\!\!// inside the bigger usertests binary, we run out of memory first.
01379 void
01380 forktest(void)
01381 {
01382
        int n, pid;
01383
01384
        printf(1, "fork test\n");
01385
        for (n=0; n<1000; n++) {
  pid = fork();</pre>
01386
01387
```

```
01388
         if(pid < 0)
01389
            break;
          if (pid == 0)
01390
           exit();
01391
01392
01393
01394
        if(n == 1000) {
        printf(1, "fork claimed to work 1000 times!\n");
01395
01396
01397
01398
        for(; n > 0; n--) {
  if(wait() < 0) {</pre>
01399
01400
01401
           printf(1, "wait stopped early\n");
01402
            exit();
01403
01404
01405
01406
        if (wait () != -1) {
        printf(1, "wait got too many\n");
01407
          exit();
01408
01409
01410
       printf(1, "fork test OK\n");
01411
01412 }
01413
01414 void
01415 sbrktest (void)
01416 {
01417
        int fds[2], pid, pids[10], ppid;
01418
       char *a, *b, *c, *lastaddr, *oldbrk, *p, scratch;
01419
       uint amt;
01420
01421
       printf(stdout, "sbrk test\n");
01422
       oldbrk = sbrk(0);
01423
01424
       // can one sbrk() less than a page?
       a = sbrk(0);
01425
01426
        int i;
        for(i = 0; i < 5000; i++){
01427
         b = sbrk(1);
01428
         if(b != a){
01429
           printf(stdout, "sbrk test failed %d %x %x\n", i, a, b);
01430
01431
            exit();
01432
01433
          *b = 1;
01434
         a = b + 1;
01435
       pid = fork();
01436
01437
        if (pid < 0) {
        printf(stdout, "sbrk test fork failed\n");
exit();
01438
01439
01440
01441
       c = sbrk(1);
01442
       c = sbrk(1);
       if(c != a + 1) {
  printf(stdout, "sbrk test failed post-fork\n");
01443
01444
          exit();
01445
01446
01447
       if(pid == 0)
01448
         exit();
       wait();
01449
01450
01451
       // can one grow address space to something big?
01452 #define BIG (100*1024*1024)
01453 a = sbrk(0);
01454
       amt = (BIG) - (uint)a;
       p = sbrk(amt);
if (p != a) {
01455
01456
        printf(stdout, "sbrk test failed to grow big address space; enough phys mem?\n");
01457
01458
01459
       lastaddr = (char*) (BIG-1);
*lastaddr = 99;
01460
01461
01462
01463
       // can one de-allocate?
01464
       a = sbrk(0);
01465
        c = sbrk(-4096);
        if(c == (char*)0xffffffff){
01466
        printf(stdout, "sbrk could not deallocate\n");
01467
01468
         exit();
01469
01470
01471
        if(c != a - 4096){
        printf(stdout, "sbrk deallocation produced wrong address, a %x c %x\n", a, c);
01472
01473
          exit();
01474
```

```
01475
01476
        // can one re-allocate that page?
        a = sbrk(0);
c = sbrk(4096);
01477
01478
        if (c != a || sbrk(0) != a + 4096) {
01479
         printf(stdout, "sbrk re-allocation failed, a %x c %x\n", a, c);
01480
01481
          exit();
01482
01483
        if(*lastaddr == 99) {
01484
          // should be zero
          printf(stdout, "sbrk de-allocation didn't really deallocate\n");
01485
01486
          exit();
01487
01488
01489
        a = sbrk(0);
01490
        c = sbrk(-(sbrk(0) - oldbrk));
01491
        if(c != a) {
        printf(stdout, "sbrk downsize failed, a %x c %x\n", a, c);
01492
01493
          exit();
01494
01495
01496
        // can we read the kernel's memory?
        for(a = (char*) (KERNBASE); a < (char*) (KERNBASE+2000000); a += 50000) {
    ppid = getpid();</pre>
01497
01498
01499
          pid = fork();
01500
          if(pid < 0){</pre>
01501
            printf(stdout, "fork failed\n");
01502
            exit();
01503
           if(pid == 0){
01504
01505
             printf(stdout, "oops could read %x = %x\n", a, *a);
01506
             kill(ppid);
01507
             exit();
01508
01509
          wait();
01510
01511
01512
        // if we run the system out of memory, does it clean up the last
01513
        // failed allocation?
        if (pipe(fds) != 0) {
  printf(1, "pipe() failed\n");
01514
01515
01516
          exit():
01517
01518
        for(i = 0; i < sizeof(pids)/sizeof(pids[0]); i++){</pre>
01519
         if((pids[i] = fork()) == 0){
01520
            // allocate a lot of memory
01521
             sbrk(BIG - (uint)sbrk(0));
             write(fds[1], "x", 1);
// sit around until killed
for(;;) sleep(1000);
01522
01523
01524
01525
01526
          if (pids[i] != -1)
01527
             read(fds[0], &scratch, 1);
01528
        ^{\prime\prime} // if those failed allocations freed up the pages they did allocate,
01529
        // we'll be able to allocate here
01530
01531
        c = sbrk(4096);
01532
        for(i = 0; i < sizeof(pids)/sizeof(pids[0]); i++){</pre>
01533
          if(pids[i] == -1)
01534
          kill(pids[i]);
01535
01536
          wait();
01537
01538
        if(c == (char*) 0xffffffff) {
01539
          printf(stdout, "failed sbrk leaked memory\n");
01540
          exit();
01541
01542
01543
        if(sbrk(0) > oldbrk)
         sbrk(-(sbrk(0) - oldbrk));
01545
01546
       printf(stdout, "sbrk test OK\n");
01547 }
01548
01549 void
01550 validateint(int *p)
01551 {
        int res;
01552
        asm("mov %%esp, %%ebx\n\t"
"mov %3, %%esp\n\t"
01553
01554
             "int %2\n\t"
01555
             "mov %%ebx, %%esp" :
             "=a" (res):
"a" (SYS_sleep), "n" (T_SYSCALL), "c" (p):
01557
01558
             "ebx");
01559
01560 }
01561
```

```
01562 void
01563 validatetest(void)
01564 {
01565
         int hi, pid;
01566
        uint p;
01567
01568
         printf(stdout, "validate test\n");
01569
        hi = 1100 * 1024;
01570
         for(p = 0; p <= (uint)hi; p += 4096){</pre>
01571
         if((pid = fork()) == 0){
01572
            // try to crash the kernel by passing in a badly placed integer
01573
01574
             validateint((int*)p);
01575
01576
01577
           sleep(0);
01578
           sleep(0);
01579
           kill (pid);
01580
           wait();
01581
           // try to crash the kernel by passing in a bad string pointer
if(link("nosuchfile", (char*)p) != -1){
  printf(stdout, "link should not succeed\n");
01582
01583
01584
01585
             exit();
01586
           }
01587 }
01588
01589
        printf(stdout, "validate ok\n");
01590 }
01591
01592 // does unintialized data start out zero?
01593 char uninit[10000];
01594 void
01595 bsstest (void)
01596 {
        int i:
01597
01598
        printf(stdout, "bss test\n");
01599
01600
         for(i = 0; i < sizeof(uninit); i++){</pre>
         if(uninit[i] != '\0'){
   printf(stdout, "bss test failed\n");
01601
01602
01603
             exit();
01604
          }
01605
01606
        printf(stdout, "bss test ok\n");
01607 }
01608
01609 // does exec return an error if the arguments
01610 // are larger than a page? or does it write
01611 // below the stack and wreck the instructions/data?
01612 void
01613 bigargtest(void)
01614 {
01615
        int pid, fd;
01616
01617
        unlink("bigarg-ok");
01618
        pid = fork();
         if (pid == 0) {
01619
         static char *args[MAXARG];
01620
          int i;
for(i = 0; i < MAXARG-1; i++)</pre>
01621
01622
            args[i] = "bigargs test: failed\n
01623
01624
           args[MAXARG-1] = 0;
           printf(stdout, "bigarg test\n");
01625
           exec("echo", args);
printf(stdout, "bigarg test ok\n");
01626
01627
01628
           fd = open("bigarg-ok", O_CREATE);
           close(fd);
01629
01630
           exit();
        } else if(pid < 0){
   printf(stdout, "bigargtest: fork failed\n");</pre>
01631
01632
01633
           exit();
01634
        wait();
01635
01636
        fd = open("bigarg-ok", 0);
01637
        if(fd < 0){
         printf(stdout, "bigarg test failed!\n");
01638
01639
          exit();
01640
01641
        close(fd);
01642
        unlink("bigarg-ok");
01643 }
01644
01645 // what happens when the file system runs out of blocks?
01646 // answer: balloc panics, so this test is not useful.
```

```
01647 void
01648 fsfull()
01649 {
01650
         int nfiles;
01651
        int fsblocks = 0;
01652
        printf(1, "fsfull test\n");
01653
01654
01655
        for(nfiles = 0; ; nfiles++) {
           char name[64];
name[0] = 'f';
name[1] = '0' + nfiles / 1000;
01656
01657
01658
           name[2] = '0' + (nfiles % 1000) / 100;
name[3] = '0' + (nfiles % 100) / 10;
01659
01660
           name[4] = '0' + (nfiles % 10);
name[5] = '\0';
01661
01662
           name[5] = '\0';
printf(1, "writing %s\n", name);
int fd = open(name, O_CREATE|O_RDWR);
if(fd < 0){</pre>
01663
01664
01665
            printf(1, "open %s failed\n", name);
01666
01667
01668
01669
           int total = 0;
01670
           while(1){
01671
             int cc = write(fd, buf, 512);
01672
              if(cc < 512)
01673
01674
             total += cc;
01675
             fsblocks++;
01676
           printf(1, "wrote %d bytes\n", total);
01677
01678
           close(fd);
01679
           if(total == 0)
01680
              break;
01681
01682
01683
         while(nfiles >= 0){
01684
           char name[64];
           name[0] = 'f';
name[1] = '0' + nfiles / 1000;
01685
01686
           name[1] = '0' + nilles / 1000;
name[2] = '0' + (nfiles % 1000) / 100;
name[3] = '0' + (nfiles % 100) / 10;
01687
01688
           name[4] = '0' + (nfiles % 10);
01689
           name[5] = ' \setminus 0';
01690
01691
           unlink(name);
01692
           nfiles--;
01693
01694
        printf(1, "fsfull test finished\n");
01695
01696 }
01697
01698 void
01699 uio()
01700 {
        #define RTC_ADDR 0x70
01701
01702
        #define RTC_DATA 0x71
01703
01704
        ushort port = 0;
01705
        uchar val = 0;
01706
        int pid;
01707
01708
         printf(1, "uio test\n");
01709
         pid = fork();
01710
         if (pid == 0) {
01711
           port = RTC_ADDR;
01712
           val = 0x09; /* year */
01713
           /* http://wiki.osdev.org/Inline Assembly/Examples */
           asm volatile("outb %0,%1"::"a"(val), "d" (port));
01714
01715
           port = RTC_DATA;
01716
           asm volatile("inb %1,%0" : "=a" (val) : "d" (port));
01717
           printf(1, "uio: uio succeeded; test FAILED\n");
01718
           exit();
        } else if(pid < 0){
  printf (1, "fork failed\n");</pre>
01719
01720
01721
           exit();
01722
01723
01724
        printf(1, "uio test done\n");
01725 }
01726
01727 void argptest()
01728 {
01729
         fd = open("init", O_RDONLY);
01730
        if (fd < 0) {
  printf(2, "open failed\n");</pre>
01731
01732
01733
           exit();
```

```
01734
01735
       read(fd, sbrk(0) - 1, -1);
       close(fd);
printf(1, "arg test passed\n");
01736
01737
01738 }
01739
01740 unsigned long randstate = 1;
01741 unsigned int
01742 rand()
01743 {
01744 randstate = randstate * 1664525 + 1013904223;
01745
       return randstate;
01746 }
01747
01748 int
01749 main(int argc, char *argv[])
01750 {
01751
        printf(1, "usertests starting\n");
01753
        if(open("usertests.ran", 0) >= 0){
01754
         printf(1, "already ran user tests -- rebuild fs.img\n");
01755
          exit();
01756
01757
       close(open("usertests.ran", O_CREATE));
01758
01759
       argptest();
01760
        createdelete();
01761
       linkunlink();
01762
       concreate();
01763
       fourfiles();
01764
       sharedfd();
01765
01766
       bigargtest();
01767
       bigwrite();
01768
       bigargtest();
01769
       bsstest();
01770
       sbrktest();
01771
       validatetest();
01772
01773
       opentest();
01774
       writetest();
01775
       writetest1();
01776
       createtest();
01777
01778
       openiputtest();
01779
        exitiputtest();
01780
       iputtest();
01781
01782
       mem();
01783
       pipel();
01784
        preempt();
01785
        exitwait();
01786
01787
01788
       rmdot();
       fourteen();
01789
        bigfile();
01790
        subdir();
01791
        linktest();
01792
        unlinkread();
01793
       dirfile();
01794
       iref();
01795
       forktest();
01796
       bigdir(); // slow
01797
01798
       uio();
01799
01800
       exectest();
01801
01802
       exit();
01803 }
```

5.242 usertests.d File Reference

5.243 usertests.d

Go to the documentation of this file.

00001 usertests.o: usertests.c /usr/include/stdc-predef.h param.h types.h \setminus 00002 stat.h user.h fs.h fcntl.h syscall.h traps.h memlayout.h

5.244 vm.c File Reference 515

5.244 vm.c File Reference

```
#include "param.h"
#include "types.h"
#include "defs.h"
#include "x86.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "elf.h"
```

Classes

struct kmap

Functions

```
• int allocuvm (pde_t *pgdir, uint oldsz, uint newsz)

    void clearpteu (pde_t *pgdir, char *uva)

• int copyout (pde_t *pgdir, uint va, void *p, uint len)
pde_t * copyuvm (pde_t *pgdir, uint sz)

    int deallocuvm (pde_t *pgdir, uint oldsz, uint newsz)

void freevm (pde_t *pgdir)

    void inituvm (pde_t *pgdir, char *init, uint sz)

    void kvmalloc (void)

• int loaduvm (pde t *pgdir, char *addr, struct inode *ip, uint offset, uint sz)

    static int mappages (pde_t *pgdir, void *va, uint size, uint pa, int perm)

    void seginit (void)

    pde t * setupkvm (void)

· void switchkvm (void)
• void switchuvm (struct proc *p)
char * uva2ka (pde_t *pgdir, char *uva)
• static pte_t * walkpgdir (pde_t *pgdir, const void *va, int alloc)
```

Variables

- char data []
- static struct kmap kmap []
- pde_t * kpgdir

5.244.1 Function Documentation

5.244.1.1 allocuvm()

```
int allocuvm (
               pde_t * pgdir,
               uint oldsz,
               uint newsz )
Definition at line 222 of file vm.c.
00223 {
00224
        char *mem;
00225
        uint a;
00226
00227
        if (newsz >= KERNBASE)
00228
          return 0;
        if (newsz < oldsz)</pre>
00229
00230
          return oldsz;
00231
       a = PGROUNDUP(oldsz);
for(; a < newsz; a += PGSIZE){</pre>
00232
00233
        mem = kalloc();
00234
00235
          if(mem == 0){
           cprintf("allocuvm out of memory\n");
00236
00237
         return 0;
            deallocuvm(pgdir, newsz, oldsz);
00238
00239
00240
          memset(mem, 0, PGSIZE);
00241
         if (mappages (pgdir, (char*)a, PGSIZE, V2P (mem), PTE_W|PTE_U) < 0) {</pre>
00242
            cprintf("allocuvm out of memory (2)\n");
00243
            deallocuvm(pgdir, newsz, oldsz);
00244
            kfree (mem);
00245
            return 0;
00246
```

Referenced by exec(), and growproc().

5.244.1.2 clearpteu()

00248 return newsz;

00247

00249 }

```
void clearpteu (
          pde_t * pgdir,
          char * uva )
```

Definition at line 303 of file vm.c.

```
00304 {
00305    pte_t *pte;
00306
00307    pte = walkpgdir(pgdir, uva, 0);
00308    if(pte == 0)
00309     panic("clearpteu");
00310    *pte &= ~PTE_U;
00311 }
```

Referenced by exec().

5.244 vm.c File Reference 517

5.244.1.3 copyout()

Definition at line 366 of file vm.c.

```
00367 {
00368
         char *buf, *pa0;
00369
         uint n, va0;
00370
         buf = (char*)p;
00371
00372
         while(len > 0){
         va0 = (uint)PGROUNDDOWN(va);
pa0 = uva2ka(pgdir, (char*)va0);
if(pa0 == 0)
00373
00374
00375
           return -1;
n = PGSIZE - (va - va0);
00376
00377
00378
           if(n > len)
n = len;
00379
00380
           memmove(pa0 + (va - va0), buf, n);
00381
           len -= n;
           buf += n;
va = va0 + PGSIZE;
00382
00383
00384
00385
         return 0;
00386 }
```

Referenced by exec().

5.244.1.4 copyuvm()

Definition at line 316 of file vm.c.

```
00317 {
00318
           pde_t *d;
          pte_t *pte;
uint pa, i, flags;
00319
00320
          char *mem;
00321
00322
00323
          if((d = setupkvm()) == 0)
          return 0;
for(i = 0; i < sz; i += PGSIZE) {
  if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
    panic("copyuvm: pte should exist");</pre>
00324
00325
00326
00328
             if(!(*pte & PTE_P))
00329
               panic("copyuvm: page not present");
             pa = PTE_ADDR(*pte);
flags = PTE_FLAGS(*pte);
if((mem = kalloc()) == 0)
00330
00331
00332
00333
                goto bad;
              memmove (mem, (char*)P2V(pa), PGSIZE);
if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0) {</pre>
00334
00335
00336
                kfree (mem);
00337
                goto bad;
00338
             }
00339
00340
          return d;
00341
00342 bad:
           freevm(d);
00343
00344
           return 0;
00345 }
```

Referenced by fork().

5.244.1.5 deallocuvm()

```
int deallocuvm (
                 pde_t * pgdir,
                 uint oldsz,
                 uint newsz )
Definition at line 256 of file vm.c.
00257 {
00258
         pte_t *pte;
00259
         uint a, pa;
00260
00261
         if (newsz >= oldsz)
00262
           return oldsz;
00263
00264
         a = PGROUNDUP(newsz);
        for(; a < oldsz; a += PGSIZE) {
  pte = walkpgdir(pgdir, (char*)a, 0);
}</pre>
00265
00266
00267
           if(!pte)
           a = PGADDR(PDX(a) + 1, 0, 0) - PGSIZE;
else if((*pte & PTE_P) != 0){
00268
00269
           pa = PTE_ADDR(*pte);
if(pa == 0)
00270
00271
              panic("kfree");
char *v = P2V(pa);
00272
00273
00274
              kfree(v);
00275
              *pte = 0;
00276
```

Referenced by allocuvm(), freevm(), and growproc().

5.244.1.6 freevm()

00277 00278

00279 }

```
void freevm (
     pde_t * pgdir )
```

return newsz;

Definition at line 284 of file vm.c.

```
00285 {
00286
         uint i;
00287
00288
         if(pgdir == 0)
00289
           panic("freevm: no pgdir");
00290
         deallocuvm(pgdir, KERNBASE, 0);
         for(i = 0; i < NPDENTRIES; i++) {
  if(pgdir[i] & PTE_P) {</pre>
00291
00292
00293
             char * v = P2V(PTE_ADDR(pgdir[i]));
              kfree(v);
00294
00295
00296
00297
         kfree((char*)pgdir);
00298 }
```

Referenced by copyuvm(), exec(), setupkvm(), and wait().

5.244 vm.c File Reference 519

5.244.1.7 inituvm()

```
void inituvm (
                           pde_t * pgdir,
                          char * init,
                          uint sz )
Definition at line 183 of file vm.c.
00184 {
00185
              char *mem;
00186
00187
              if(sz >= PGSIZE)
00187    if(sz >= PGSIZE)
00188         panic("initurm: more than a page");
00189    mem = kalloc();
00190    memset(mem, 0, PGSIZE);
00191    mappages(pgdir, 0, PGSIZE, V2P(mem), PTE_W|PTE_U);
00192    memmove(mem, init, sz);
```

Referenced by userinit().

00193 }

5.244.1.8 kvmalloc()

```
void kvmalloc (
            void )
```

Definition at line 141 of file vm.c.

```
00143 kpgdir = setupkvm();
```

5.244.1.9 loaduvm()

```
int loaduvm (
            pde_t * pgdir,
            char * addr,
             struct inode * ip,
             uint offset,
             uint sz )
```

Definition at line 198 of file vm.c.

```
00199 {
00200
         uint i, pa, n;
00201
         pte_t *pte;
00202
00203
         if((uint) addr % PGSIZE != 0)
         panic("loaduvm: addr must be page aligned");
for(i = 0; i < sz; i += PGSIZE) {</pre>
00204
00205
         if((pte = walkpgdir(pgdir, addr+i, 0)) == 0)
panic("loaduvm: address should exist");
00207
00208
           pa = PTE_ADDR(*pte);
           if (sz - i < PGSIZE)
n = sz - i;
00209
00210
00211
           else
00212
             n = PGSIZE;
00213
            if(readi(ip, P2V(pa), offset+i, n) != n)
00214
             return -1;
00215
00216
         return 0;
00217 }
```

Referenced by exec().

5.244.1.10 mappages()

```
static int mappages (
               pde_t * pgdir,
                void * va,
                uint size,
               uint pa,
               int perm ) [static]
Definition at line 61 of file vm.c.
00062 {
00063
        char *a, *last;
00064
        pte_t *pte;
00065
00066
        a = (char*)PGROUNDDOWN((uint)va);
        last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
for(;;){
00067
00068
00069
          if((pte = walkpgdir(pgdir, a, 1)) == 0)
00070
            return -1;
          if(*pte & PTE_P)
  panic("remap");
00071
00072
          *pte = pa | perm | PTE_P;
if(a == last)
00073
00074
```

Referenced by allocuvm(), copyuvm(), inituvm(), and setupkvm().

5.244.1.11 seginit()

00075

00076

00077

00078

00080 }

```
void seginit (
     void )
```

Definition at line 16 of file vm.c.

break;

return 0;

a += PGSIZE;

pa += PGSIZE;

```
00017 {
00018
           struct cpu *c;
00019
          // Map "logical" addresses to virtual addresses using identity map.
00020
           // Cannot share a CODE descriptor for both kernel and user
00022
          // because it would have to have DPL_USR, but the CPU forbids
00023
           // an interrupt from CPL=0 to DPL=3.
          c = &cpus[cpuid()];
c->gdt[SEG_KCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, 0);
c->gdt[SEG_UCODE] = SEG(STA_W, 0, 0xffffffff, 0);
c->gdt[SEG_UCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, DPL_USER);
c->gdt[SEG_UDATA] = SEG(STA_W, 0, 0xffffffff, DPL_USER);
00024
00025
00026
00027
00028
00029
           lgdt(c->gdt, sizeof(c->gdt));
00030 }
```

Referenced by mpenter().

5.244 vm.c File Reference 521

5.244.1.12 setupkvm()

```
pde_t * setupkvm (
             void )
Definition at line 119 of file vm.c.
00120 {
00121
       pde_t *pgdir;
       struct kmap *k;
00122
00123
       if((pgdir = (pde_t*)kalloc()) == 0)
        return 0;
00125
      memset(pgdir, 0, PGSIZE);
if (P2V(PHYSTOP) > (void*)DEVSPACE)
00126
00127
        panic("PHYSTOP too high");
00128
       00129
00130
00131
00132
           freevm(pgdir);
00133
00134
        }
00135
      return pgdir;
00136 }
```

Referenced by copyuvm(), exec(), kvmalloc(), and userinit().

5.244.1.13 switchkvm()

Referenced by kvmalloc(), mpenter(), and scheduler().

5.244.1.14 switchuvm()

void switchuvm (

```
struct proc * p )
Definition at line 157 of file vm.c.
00159
00160
          panic("switchuvm: no process");
00161
        if(p->kstack == 0)
          panic("switchuvm: no kstack");
00162
       if(p->pgdir == 0)
00163
00164
         panic ("switchuvm: no pgdir");
00165
00166
00167
       mycpu()->gdt[SEG_TSS] = SEG16(STS_T32A, &mycpu()->ts,
00168
                                        sizeof(mycpu()->ts)-1, 0);
       mycpu()->qdt[SEG_TSS].s = 0;
00169
00170
       mycpu()->ts.ss0 = SEG_KDATA « 3;
00171
       mycpu()->ts.esp0 = (uint)p->kstack + KSTACKSIZE;
00172
        // setting IOPL=0 in eflags *and* iomb beyond the tss segment limit
        // forbids I/O instructions (e.g., inb and outb) from user space
mycpu()->ts.iomb = (ushort) 0xFFFF;
00173
00174
00175
        ltr(SEG_TSS « 3);
00176
        lcr3(V2P(p->pgdir)); // switch to process's address space
00177
        popcli();
00178 }
```

Referenced by exec(), growproc(), and scheduler().

5.244.1.15 uva2ka()

```
char * uva2ka (
          pde_t * pgdir,
          char * uva )
```

Definition at line 350 of file vm.c.

```
00351 {
00352
       pte_t *pte;
00353
00354
       pte = walkpgdir(pgdir, uva, 0);
00355
        if((*pte & PTE_P) == 0)
00356
         return 0;
00357
       if((*pte & PTE_U) == 0)
00358
         return 0:
00359
        return (char*)P2V(PTE_ADDR(*pte));
00360 }
```

Referenced by copyout().

5.244.1.16 walkpgdir()

Definition at line 36 of file vm.c.

```
00037 {
00038
       pde_t *pde;
00039
       pte_t *pgtab;
00040
00041
       pde = &pgdir[PDX(va)];
00042
       if(*pde & PTE_P){
         pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
00043
00044
         if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
00045
00046
            return 0;
00047
          // Make sure all those PTE_P bits are zero.
00048
          memset(pgtab, 0, PGSIZE);
00049
         // The permissions here are overly generous, but they can
         // be further restricted by the permissions in the page table
00050
         // entries, if necessary.
*pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
00051
00052
00053
00054
       return &pgtab[PTX(va)];
00055 }
```

Referenced by clearpteu(), copyuvm(), deallocuvm(), loaduvm(), mappages(), and uva2ka().

5.244.2 Variable Documentation

5.244.2.1 data

```
char data[] [extern]
```

Referenced by inb(), ioapicwrite(), kbdgetc(), main(), outb(), outw(), stosb(), and stosl().

5.245 vm.c 523

5.244.2.2 kmap

5.244.2.3 kpgdir

```
pde_t* kpgdir
```

Definition at line 11 of file vm.c.

Referenced by kvmalloc(), and switchkvm().

5.245 vm.c

Go to the documentation of this file.

```
00001 #include "param.h"
00002 #include "types.h"
00003 #include "defs.h"
00004 #include "x86.h"
00005 #include "memlayout.h"
00006 #include "mmu.h"
00007 #include "proc.h"
00008 #include "elf.h"
00009
00010 extern char data[]; // defined by kernel.ld
00011 pde_t *kpgdir; // for use in scheduler()
00012
00013 // Set up CPU's kernel segment descriptors.
00014 // Run once on entry on each CPU.
00015 void
00016 seginit (void)
00017 {
00018
         struct cpu *c;
00019
00020
         // Map "logical" addresses to virtual addresses using identity map.
00021
         // Cannot share a CODE descriptor for both kernel and user
00022
         // because it would have to have DPL_USR, but the CPU forbids
00023
         // an interrupt from CPL=0 to DPL=3.
        c = &cpus[cpuid()];
c->gdt[SEG_KCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, 0);
c->gdt[SEG_KDATA] = SEG(STA_W, 0, 0xffffffff, 0);
c->gdt[SEG_UCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, DPL_USER);
c->gdt[SEG_UDATA] = SEG(STA_W, 0, 0xffffffff, DPL_USER);
00024
00025
00026
00028
00029
         lgdt(c->gdt, sizeof(c->gdt));
00030 }
00031
00032 // Return the address of the PTE in page table pgdir
00033 // that corresponds to virtual address va. If alloc!=0,
00034 // create any required page table pages.
00035 static pte_t *
00036 walkpgdir(pde_t *pgdir, const void *va, int alloc)
00037 {
00038
         pde_t *pde;
         pte_t *pgtab;
00039
00040
00041
         pde = &pgdir[PDX(va)];
00042
         if(*pde & PTE_P){
00043
           pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
00044
         } else {
00045
          if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
00046
             return 0;
```

```
// Make sure all those PTE_P bits are zero.
00048
           memset(pgtab, 0, PGSIZE);
00049
           // The permissions here are overly generous, but they can
           // be further restricted by the permissions in the page table
00050
           // entries, if necessary.
*pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
00051
00052
00053
00054
        return &pgtab[PTX(va)];
00055 }
00056
00057 // Create PTEs for virtual addresses starting at va that refer to
\tt 00058 // physical addresses starting at pa. va and size might not
00059 // be page-aligned.
00060 static int
00061 mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
00062 {
00063
        char *a, *last;
00064
        pte_t *pte;
00065
        a = (char*)PGROUNDDOWN((uint)va);
00066
00067
         last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
00068
        for(;;) {
00069
          if((pte = walkpgdir(pgdir, a, 1)) == 0)
00070
             return -1;
00071
           if(*pte & PTE_P)
00072
            panic("remap");
00073
           *pte = pa | perm | PTE_P;
00074
          if(a == last)
          break;
a += PGSIZE;
00075
00076
          pa += PGSIZE;
00077
00078
        }
00079
        return 0;
00080 }
00081
00082 // There is one page table per process, plus one that's used when
00083 // a CPU is not running any process (kpgdir). The kernel uses the 00084 // current process's page table during system calls and interrupts;
00085 // page protection bits prevent user code from using the kernel's
00086 // mappings.
00087 //
00088 // setupkvm() and exec() set up every page table like this:
00089 //
00090 //
            0..KERNBASE: user memory (text+data+stack+heap), mapped to
00091 //
                          phys memory allocated by the kernel
00092 //
            KERNBASE..KERNBASE+EXTMEM: mapped to 0..EXTMEM (for I/O space)
00093 //
            KERNBASE+EXTMEM..data: mapped to EXTMEM..V2P(data)
                          for the kernel's instructions and r/o data
00094 //
00095 //
            data..KERNBASE+PHYSTOP: mapped to V2P(data)..PHYSTOP,
00096 //
                                              rw data + free physical memory
00097 //
            0xfe000000..0: mapped direct (devices such as ioapic)
00098 //
00099 // The kernel allocates physical memory for its heap and for user memory
00100 // between V2P(end) and the end of physical memory (PHYSTOP) 00101 // (directly addressable from end..P2V(PHYSTOP)).
00102
00103 // This table defines the kernel's mappings, which are present in
00104 // every process's page table.
00105 static struct kmap {
00106
       void *virt;
        uint phys_start;
00107
00108
        uint phys_end;
00109
        int perm;
00110 \} kmap[] = {
00111
       { (void*) KERNBASE, 0,
                                             EXTMEM.
                                                         PTE_W}, // I/O space
00112 { (void*)KERNLINK, V2P(KERNLINK), V2P(data), 0}, // kern text+rodata
00113 { (void*)data, V2P(data), PHYSTOP, PTE_W}, // kern data+memory
00113 { (void*)data, V2P(data)
00114 { (void*)DEVSPACE, DEVSPACE,
                                              Ο,
                                                          PTE_W}, // more devices
00115 };
00116
00117 // Set up kernel part of a page table.
00118 pde_t*
00119 setupkvm(void)
00120 {
        pde_t *pgdir;
struct kmap *k;
00121
00122
00123
00124
        if((pgdir = (pde_t*)kalloc()) == 0)
00125
          return 0:
        memset(pgdir, 0, PGSIZE);
00126
        if (P2V(PHYSTOP) > (void*)DEVSPACE)
00127
          panic("PHYSTOP too high");
00128
00129
         for(k = kmap; k < &kmap[NELEM(kmap)]; k++)</pre>
           if(mappages(pgdir, k->virt, k->phys_end - k->phys_start,
00130
00131
                        (uint)k \rightarrow phys_start, k \rightarrow perm) < 0) {
             freevm(pgdir);
00132
00133
             return 0:
```

5.245 vm.c 525

```
00135
        return pgdir;
00136 }
00137
00138 // Allocate one page table for the machine for the kernel address
00139 // space for scheduler processes.
00140 void
00141 kvmalloc(void)
00142 {
00143
        kpgdir = setupkvm();
00144
        switchkvm();
00145 }
00146
00147 // Switch h/w page table register to the kernel-only page table,
00148 \!\!\!\!// for when no process is running.
00149 void
00150 switchkym(void)
00151 {
00152
        lcr3(V2P(kpgdir)); // switch to the kernel page table
00153 }
00154
00155 // Switch TSS and h/w page table to correspond to process p.
00156 void
00157 switchuvm(struct proc *p)
00158 {
00159
        if(p == 0)
00160
          panic("switchuvm: no process");
00161
        if(p->kstack == 0)
00162
          panic("switchuvm: no kstack");
        if(p->pgdir == 0)
00163
00164
          panic("switchuvm: no pgdir");
00165
00166
        pushcli();
00167
        mycpu()->gdt[SEG_TSS] = SEG16(STS_T32A, &mycpu()->ts,
00168
                                         sizeof(mycpu()->ts)-1, 0);
        mycpu()->gdt[SEG_TSS].s = 0;
00169
        mycpu()->ts.ss0 = SEG_KDATA « 3;
mycpu()->ts.esp0 = (uint)p->kstack + KSTACKSIZE;
00170
00171
00172
         // setting IOPL=0 in eflags *and* iomb beyond the tss segment limit
00173
         // forbids I/O instructions (e.g., inb and outb) from user space
00174
        mycpu()->ts.iomb = (ushort) 0xFFFF;
        ltr(SEG_TSS « 3);
lcr3(V2P(p->pgdir)); // switch to process's address space
00175
00176
00177
        popcli();
00178 }
00179
00180 // Load the initcode into address 0 of pgdir.
00181 // sz must be less than a page.
00182 void
00183 inituvm(pde_t *pgdir, char *init, uint sz)
00184 {
00185
00186
00187
        if(sz >= PGSIZE)
          panic("inituvm: more than a page");
00188
00189
        mem = kalloc();
        memset(mem, 0, PGSIZE);
00190
00191
        mappages(pgdir, 0, PGSIZE, V2P(mem), PTE_W|PTE_U);
00192
        memmove(mem, init, sz);
00193 }
00194
00195 // Load a program segment into pgdir. addr must be page-aligned 00196 // and the pages from addr to addr+sz must already be mapped.
00197 int
00198 loaduvm(pde_t *pgdir, char *addr, struct inode *ip, uint offset, uint sz)
00199 {
00200
        uint i, pa, n;
00201
        pte_t *pte;
00202
00203
        if((uint) addr % PGSIZE != 0)
00204
          panic("loaduvm: addr must be page aligned");
00205
         for(i = 0; i < sz; i += PGSIZE) +</pre>
00206
         if((pte = walkpgdir(pgdir, addr+i, 0)) == 0)
            panic("loaduvm: address should exist");
00207
          pa = PTE_ADDR(*pte);
00208
00209
          if(sz - i < PGSIZE)
00210
            n = sz - i;
00211
            n = PGSIZE;
00212
           if(readi(ip, P2V(pa), offset+i, n) != n)
00213
00214
             return -1;
00215
00216
        return 0;
00217 }
00218
00219 // Allocate page tables and physical memory to grow process from oldsz to 00220 // newsz, which need not be page aligned. Returns new size or 0 on error.
```

```
00222 allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
00223 {
00224
        char *mem;
00225
        uint a;
00226
00227
        if(newsz >= KERNBASE)
00228
          return 0;
00229
        if(newsz < oldsz)</pre>
00230
         return oldsz;
00231
        a = PGROUNDUP(oldsz);
00232
        for(; a < newsz; a += PGSIZE) {</pre>
00233
00234
         mem = kalloc();
00235
          if (mem == 0) {
           cprintf("allocuvm out of memory\n");
00236
00237
            deallocuvm(pgdir, newsz, oldsz);
00238
            return 0;
00240
          memset(mem, 0, PGSIZE);
00241
          if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){</pre>
00242
            cprintf("allocuvm out of memory (2)\n");
00243
            deallocuvm(pgdir, newsz, oldsz);
00244
            kfree (mem);
00245
            return 0;
00246
          }
00247
00248
       return newsz;
00249 }
00250
00251 // Deallocate user pages to bring the process size from oldsz to
00252 // newsz. oldsz and newsz need not be page-aligned, nor does newsz
00253 // need to be less than oldsz. oldsz can be larger than the actual
00254 // process size. Returns the new process size.
00255 int
00256 deallocuvm(pde_t *pgdir, uint oldsz, uint newsz)
00257 {
       pte_t *pte;
00259
       uint a, pa;
00260
00261
       if(newsz >= oldsz)
        return oldsz;
00262
00263
00264
       a = PGROUNDUP(newsz);
00265
       for(; a < oldsz; a += PGSIZE) {</pre>
00266
        pte = walkpgdir(pgdir, (char*)a, 0);
00267
          if(!pte)
          a = PGADDR(PDX(a) + 1, 0, 0) - PGSIZE;
else if((*pte & PTE_P) != 0){
00268
00269
          pa = PTE_ADDR(*pte);
if(pa == 0)
00270
00271
00272
              panic("kfree");
00273
            char *v = P2V(pa);
00274
            kfree(v);
00275
            *pte = 0;
00276
         }
00277
00278
       return newsz;
00279 }
00280
{\tt 00281} // Free a page table and all the physical memory pages
00282 // in the user part.
00283 void
00284 freevm(pde_t *pgdir)
00285 {
00286
       uint i;
00287
00288
        if(padir == 0)
         panic("freevm: no pgdir");
00289
        deallocuvm(pgdir, KERNBASE, 0);
00291
        for(i = 0; i < NPDENTRIES; i++) {</pre>
00292
          if(pgdir[i] & PTE_P){
00293
            char * v = P2V(PTE_ADDR(pgdir[i]));
            kfree(v);
00294
00295
          }
00296
00297
        kfree((char*)pgdir);
00298 }
00299
00300 // Clear PTE_U on a page. Used to create an inaccessible
00301 // page beneath the user stack.
00302 void
00303 clearpteu(pde_t *pgdir, char *uva)
00304 {
00305
       pte_t *pte;
00306
00307
       pte = walkpgdir(pgdir, uva, 0);
```

5.245 vm.c 527

```
if(pte == 0)
00309
          panic("clearpteu");
00310
        *pte &= ~PTE_U;
00311 }
00312
00313 // Given a parent process's page table, create a copy
00314 // of it for a child.
00315 pde_t*
00316 copyuvm(pde_t *pgdir, uint sz)
00317 {
00318
        pde_t *d;
        pte_t *pte;
00319
        uint pa, i, flags;
char *mem;
00320
00321
00322
00323
        if((d = setupkvm()) == 0)
00324
           return 0;
        for(i = 0; i < sz; i += PGSIZE) {
   if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)</pre>
00325
00326
00327
             panic("copyuvm: pte should exist");
00328
           if(!(*pte & PTE_P))
00329
            panic("copyuvm: page not present");
          pa = PTE_ADDR(*pte);
flags = PTE_FLAGS(*pte);
00330
00331
00332
           if ((mem = kalloc()) == 0)
00333
            goto bad;
00334
           memmove(mem, (char*)P2V(pa), PGSIZE);
00335
           if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0) {</pre>
00336
            kfree (mem);
00337
             goto bad;
00338
00339
00340
        return d;
00341
00342 bad:
        freevm(d);
00343
00344
        return 0;
00346
00347 //PAGEBREAK!
00348 // Map user virtual address to kernel address.
00349 char*
00350 uva2ka(pde_t *pgdir, char *uva)
00351 {
00352
        pte_t *pte;
00353
00354
        pte = walkpgdir(pgdir, uva, 0);
00355
        if((*pte & PTE_P) == 0)
00356
          return 0:
00357
        if((*pte & PTE_U) == 0)
00358
          return 0;
00359
        return (char*)P2V(PTE_ADDR(*pte));
00360 }
00361
00362 // Copy len bytes from p to user address va in page table pgdir. 00363 // Most useful when pgdir is not the current page table.
00364 // uva2ka ensures this only works for PTE_U pages.
00365 int
00366 copyout(pde_t *pgdir, uint va, void *p, uint len)
00367 {
        char *buf, *pa0;
00368
00369
        uint n, va0;
00370
00371
        buf = (char*)p;
00372
        while (len > 0) {
          va0 = (uint)PGROUNDDOWN(va);
pa0 = uva2ka(pgdir, (char*)va0);
00373
00374
           if(pa0 == 0)
00375
            return -1;
00376
00377
          n = PGSIZE - (va - va0);
00378
          if(n > len)
00379
            n = len;
           memmove(pa0 + (va - va0), buf, n);
00380
00381
           len -= n;
          buf += n;
00382
00383
           va = va0 + PGSIZE;
00384
00385
        return 0;
00386 }
00387
00388 //PAGEBREAK!
00389 // Blank page.
00390 //PAGEBREAK!
00391 // Blank page
00392 //PAGEBREAK!
00393 // Blank page.
00394
```

5.246 vm.d File Reference

5.247 vm.d

Go to the documentation of this file.

```
00001 vm.o: vm.c /usr/include/stdc-predef.h param.h types.h defs.h x86.h \ 00002 memlayout.h mmu.h proc.h elf.h
```

5.248 wc.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

- int main (int argc, char *argv[])
- void wc (int fd, char *name)

Variables

char buf [512]

5.248.1 Function Documentation

5.248.1.1 main()

```
int main (
          int argc,
          char * argv[] )
```

Definition at line 36 of file wc.c.

```
00037 {
00038
            int fd, i;
 00039
            if (argc <= 1) {
  wc(0, "");
  exit();</pre>
 00040
 00041
 00042
 00043 }
 00044
           for(i = 1; i < argc; i++){
  if((fd = open(argv[i], 0)) < 0){
    printf(1, "wc: cannot open %s\n", argv[i]);
    exit();</pre>
 00045
 00046
 00047
 00048
00049
               wc(fd, argv[i]);
```

5.249 wc.c 529

5.248.1.2 wc()

```
void wc (
               int fd,
               char * name )
Definition at line 8 of file wc.c.
00009 {
        int i, n;
00010
00011
       int 1, w, c, inword;
00012
00013
        1 = w = c = 0;
       inword = 0;
00014
00015
       while((n = read(fd, buf, sizeof(buf))) > 0){
00016
         for (i=0; i<n; i++) {</pre>
00017
            c++;
00018
            if(buf[i] == '\n')
           1++;
if(strchr(" \r\t\n\v", buf[i]))
inword = 0;
00019
00021
            else if(!inword){
00022
             w++;
00023
00024
              inword = 1;
00025
            }
00026
        }
00027
       if(n < 0){
00028
        printf(1, "wc: read error\n");
exit();
00029
00030
00032 printf(1, "%d %d %d %s\n", 1, w, c, name);
```

Referenced by main().

5.248.2 Variable Documentation

5.248.2.1 buf

```
char buf[512]
```

Definition at line 5 of file wc.c.

5.249 wc.c

Go to the documentation of this file.

```
00001 #include "types.h"
00002 #include "stat.h"
00003 #include "user.h"
00004
00005 char buf[512];
00006
00007 void
00008 wc(int fd, char *name)
00009 {
00010
       int i, n;
       int 1, w, c, inword;
00011
00012
00017
           c++;
00018
            if(buf[i] == '\n')
```

```
1++;
             if(strchr(" \r\t\n\v", buf[i]))
inword = 0;
else if(!inword){
    w++;
00020
00021
00022
00023
00024
                inword = 1;
00025
             }
00026
           }
00027
        if(n < 0){
00028
         printf(1, "wc: read error\n");
00029
00030
           exit();
00031
00032
        printf(1, "%d %d %d %s\n", 1, w, c, name);
00033 }
00034
00035 int
00036 main(int argc, char *argv[])
00037 {
00038
         int fd, i;
00039
         if (argc <= 1) {
  wc(0, "");</pre>
00040
00041
00042
           exit();
00043
00044
00045
         for(i = 1; i < argc; i++) {</pre>
         if((fd = open(argv[i], 0)) < 0){
    printf(1, "wc: cannot open %s\n", argv[i]);</pre>
00046
00047
00048
             exit();
00049
00050
           wc(fd, argv[i]);
00051
          close(fd);
00052
00053 exit();
00054 }
```

5.250 wc.d File Reference

5.251 wc.d

Go to the documentation of this file.

00001 wc.o: wc.c /usr/include/stdc-predef.h types.h stat.h user.h

5.252 x86.h File Reference

Classes

· struct trapframe

Functions

- static void cli (void)
- static uchar inb (ushort port)
- static void insl (int port, void *addr, int cnt)
- static void lcr3 (uint val)
- static void lgdt (struct segdesc *p, int size)
- static void lidt (struct gatedesc *p, int size)
- static void loadgs (ushort v)
- static void ltr (ushort sel)
- static void outb (ushort port, uchar data)

5.252 x86.h File Reference 531

- static void outsl (int port, const void *addr, int cnt)
- static void outw (ushort port, ushort data)
- static uint rcr2 (void)
- static uint readeflags (void)
- static void sti (void)
- static void stosb (void *addr, int data, int cnt)
- static void stosl (void *addr, int data, int cnt)
- static uint xchg (volatile uint *addr, uint newval)

5.252.1 Function Documentation

5.252.1.1 cli()

Referenced by consputc(), panic(), and pushcli().

5.252.1.2 inb()

Referenced by cgaputc(), cmos_read(), ideinit(), idewait(), kbdgetc(), mpinit(), uartgetc(), uartinit(), uartputc(), and waitdisk().

5.252.1.3 insl()

Referenced by ideintr(), and readsect().

5.252.1.4 lcr3()

Referenced by switchkvm(), and switchuvm().

5.252.1.5 lgdt()

Definition at line 63 of file x86.h.

Referenced by seginit().

5.252.1.6 lidt()

Definition at line 77 of file x86.h.

```
00078 {
00079     volatile ushort pd[3];
00080
00081     pd[0] = size-1;
00082     pd[1] = (uint)p;
00083     pd[2] = (uint)p > 16;
00084
00085     asm volatile("lidt (%0)" : : "r" (pd));
00086 }
```

Referenced by idtinit().

5.252 x86.h File Reference 533

5.252.1.7 loadgs()

```
static void loadgs ( ushort\ v\ )\ [inline],\ [static] Definition at line 103 of file x86.h. 00104\ \{\\00105\ asm\ volatile("movw\ \$0,\ \$\$gs"\ :\ "r"\ (v));\\00106\ \}
```

5.252.1.8 ltr()

Referenced by switchuvm().

5.252.1.9 outb()

00025 }

Referenced by cgaputc(), cmos_read(), ideinit(), idestart(), lapicstartap(), mpinit(), picinit(), readsect(), sys_halt(), uartinit(), and uartputc().

5.252.1.10 outsl()

```
static void outsl (
          int port,
          const void * addr,
          int cnt ) [inline], [static]
```

Definition at line 34 of file x86.h.

```
00035 {
00036    asm volatile("cld; rep outsl" :
00037    "=S" (addr), "=c" (cnt) :
00038    "d" (port), "0" (addr), "1" (ent) :
00039    "cc");
```

Referenced by idestart().

5.252.1.11 outw()

5.252.1.12 rcr2()

Definition at line 134 of file x86.h.

```
00135 {
00136     uint val;
00137     asm volatile("movl %%cr2,%0" : "=r" (val));
00138     return val;
00139 }
```

Referenced by trap().

5.252.1.13 readeflags()

Definition at line 95 of file x86.h.

```
00096 {
00097    uint eflags;
00098    asm volatile("pushfl; popl %0" : "=r" (eflags));
00099    return eflags;
00100 }
```

Referenced by mycpu(), popcli(), pushcli(), and sched().

5.252.1.14 sti()

Definition at line 115 of file x86.h.

```
00116 {
00117 asm volatile("sti");
00118 }
```

Referenced by cps(), popcli(), and scheduler().

5.252 x86.h File Reference 535

5.252.1.15 stosb()

Referenced by bootmain(), and memset().

5.252.1.16 stosl()

Definition at line 52 of file x86.h.

```
00053 {
00054     asm volatile("cld; rep stos1" :
00055     "=D" (addr), "=c" (cnt) :
00056     "0" (addr), "1" (cnt), "a" (data) :
00057     "memory", "cc");
```

Referenced by memset().

5.252.1.17 xchg()

Definition at line 121 of file x86.h.

Referenced by acquire().

536 File Documentation

5.253 x86.h

```
Go to the documentation of this file.
00001 // Routines to let C code use special x86 instructions.
00002
00003 static inline uchar
00004 inb(ushort port)
00005 {
00006
         uchar data;
00007
U0009 return data;
00008 asm volatile("in %1,%0" : "=a" (data) : "d" (port));
00011
00012 static inline void
00013 insl(int port, void *addr, int cnt)
00014 {
00015 asm volatile("cld; rep insl" :
                        "=D" (addr), "=c" (cnt):
"d" (port), "0" (addr), "1" (cnt):
"memory", "cc");
00016
00017
00018
00019 }
00020
00021 static inline void
00022 outb(ushort port, uchar data)
00024
        asm volatile("out %0,%1" : : "a" (data), "d" (port));
00025 }
00026
00027 static inline void
00028 outw(ushort port, ushort data)
00030
        asm volatile("out %0,%1" : : "a" (data), "d" (port));
00031 }
00032
00033 static inline void
00034 outsl(int port, const void *addr, int cnt)
00035 {
00036 asm volatile("cld; rep outsl" :
                        "=S" (addr), "=c" (cnt) :
"d" (port), "0" (addr), "1" (cnt) :
00037
00038
                        "cc");
00039
00040 }
00042 static inline void
00043 stosb(void *addr, int data, int cnt)
00044 {
00044 {
00045 asm volatile("cld; rep stosb":
00046 "=D" (addr), "=c" (cnt):
00047 "0" (addr), "1" (cnt), "a" (data):
00048 "memory", "cc");
00049 }
00050
00051 static inline void
00052 stosl(void *addr, int data, int cnt)
00053 {
        asm volatile("cld; rep stosl" :
    "=D" (addr), "=c" (cnt) :
    "0" (addr), "1" (cnt), "a" (data) :
    "memory", "cc");
00055
00056
00057
00058 }
00059
00060 struct segdesc;
00062 static inline void
00063 lgdt(struct segdesc *p, int size)
00064 {
00065
         volatile ushort pd[3];
00066
00067 pd[0] = size-1;
00068
        pd[1] = (uint)p;
00069
        pd[2] = (uint)p \gg 16;
00070
00071
        asm volatile("lgdt (%0)" : : "r" (pd));
00072 }
00074 struct gatedesc;
00075
```

00076 static inline void

00081 pd[0] = size-1; 00082 pd[1] = (uint)p;

00078 {

08000

00077 lidt(struct gatedesc *p, int size)

volatile ushort pd[3];

5.253 x86.h 537

```
00083 pd[2] = (uint)p \gg 16;
00084
       asm volatile("lidt (%0)" : : "r" (pd));
00085
00086 }
00087
00088 static inline void
00089 ltr(ushort sel)
00090 {
00091
        asm volatile("ltr %0" : : "r" (sel));
00092 }
00093
00094 static inline uint
00095 readeflags(void)
00096 {
        uint eflags;
00097
00098 asm volatile("pushfl; popl %0" : "=r" (eflags));
00099 return eflags;
00102 static inline void
00103 loadgs(ushort v)
00104 {
        asm volatile("movw %0, %%gs" : : "r" (v));
00105
00106 }
00107
00108 static inline void
00109 cli(void)
00110 {
00111
        asm volatile("cli");
00112 }
00113
00114 static inline void
00115 sti(void)
00116 {
00117
        asm volatile("sti");
00117
00119
00120 static inline uint
00121 xchg(volatile uint *addr, uint newval)
00122 {
00123
        uint result;
00124
00125 // The + in "+m" denotes a read-modify-write operand.
00126 asm volatile("lock; xchgl %0, %1":
00125
                      "+m" (*addr), "=a" (result) :
"1" (newval) :
00127
00128
                      "cc");
00129
00130
       return result;
00131 }
00132
00133 static inline uint
00134 rcr2(void)
00135 {
00136 uint val;
        asm volatile("movl %%cr2,%0" : "=r" (val));
00137
00138
        return val;
00139 }
00140
00141 static inline void
00142 lcr3(uint val)
00143 {
        asm volatile("mov1 %0,%%cr3" : : "r" (val));
00144
00145 }
00146
00147 //PAGEBREAK: 36
00148 // Layout of the trap frame built on the stack by the 00149 // hardware and by trapasm.S, and passed to trap().
00150 struct trapframe {
00151 // registers as pushed by pusha
00152
        uint edi;
00153
        uint esi;
00154
        uint ebp;
00155
        uint oesp;
                        // useless & ignored
00156
        uint ebx:
00157
        uint edx;
00158
        uint ecx;
00159
        uint eax;
00160
        // rest of trap frame
00161
00162
        ushort gs;
        ushort padding1;
ushort fs;
00163
00164
00165
        ushort padding2;
00166
        ushort es;
00167
        ushort padding3;
00168
        ushort ds;
        ushort padding4;
00169
```

538 File Documentation

```
uint trapno;
00171
        // below here defined by x86 hardware
00172
00173
        uint err;
        uint eip;
00174
00175
        ushort cs;
ushort padding5;
00176
00177
        uint eflags;
00178
        \ensuremath{//} below here only when crossing rings, such as from user to kernel
00179
00180
        uint esp;
00181 ushort ss;
00182 ushort pad
        ushort padding6;
00183 };
```

5.254 xv6.dox File Reference

5.255 zombie.c File Reference

```
#include "types.h"
#include "stat.h"
#include "user.h"
```

Functions

· int main (void)

5.255.1 Function Documentation

5.255.1.1 main()

```
int main (
     void )
```

Definition at line 9 of file zombie.c.

```
00010 {
00011    if(fork() > 0)
00012         sleep(5); // Let child exit before parent.
00013    exit();
00014 }
```

5.256 zombie.c

Go to the documentation of this file.

```
00001 // Create a zombie process that
00002 // must be reparented at exit.
00003
00004 #include "types.h"
00005 #include "stat.h"
00006 #include "user.h"
00007
00008 int
00009 main(void)
00010 {
00011    if(fork() > 0)
00012         sleep(5); // Let child exit before parent.
00013    exit();
00014 }
```

5.257 zombie.d File Reference

5.258 zombie.d

Go to the documentation of this file.
00001 zombie.o: zombie.c /usr/include/stdc-predef.h types.h stat.h user.h

540 File Documentation

Index

attribute	execcmd, 23
main.c, 271	init.c, 222
_binary_fs_img_size	asm.h, 81
memide.c, 276	SEG_ASM, 81
_binary_fs_img_start	SEG_NULLASM, 81
memide.c, 276	STA_R, 81
agguira	STA_W, 82
acquire defs.h, 110	STA_X, 82
spinlock.c, 371	ASSERT
acquiresleep	lapic.c, 245
defs.h, 111	atoi
sleeplock.c, 368	ulib.c, 440
addr	user.h, 450
	avl
mpioapic, 43 addrs	segdesc, 59
	B DIRTY
dinode, 17 inode, 30	buf.h, 92
Align	B_VALID
	buf.h, 92
umalloc.c, 446	BACK
align	_
proghdr, 52	sh.c, 350 backcmd, 9
allocproc	cmd, 9
proc.c, 325	,
allocuvm	sh.c, 352
defs.h, 111	type, 9
vm.c, 515 ALT	BACKSPACE
	console.c, 96 balloc
kbd.h, 235	
apicid	fs.c, 186
cpu, 14	mkfs.c, 283
mpproc, 45	base umalloc.c, 447
apicno	base 15 0
mpioapic, 43 argfd	
sysfile.c, 404	segdesc, 59
	base_23_16 segdesc, 59
argint defs.h, 112	base_31_24
syscall.c, 384	segdesc, 59 BBLOCK
argptest	fs.h, 207
usertests.c, 465	bcache
argptr	
defs.h, 112	bio.c, 85
syscall.c, 384	BCAST
args	lapic.c, 245
gatedesc, 27	begin_op
argstr	defs.h, 113
defs.h, 112	log.c, 260
syscall.c, 385	bfree
argv	fs.c, 186

bget	blockno, 10
bio.c, 83	cat.c, 94
BIG	console.c, 101
usertests.c, 464	data, 10
bigargtest	dev, 10
usertests.c, 465	flags, 11
bigdir	grep.c, 212
usertests.c, 466	lock, 11
bigfile	next, 11
usertests.c, 466	prev, 11
	gnext, 11
bigwrite	
usertests.c, 467	refcnt, 12
binit	usertests.c, 492
bio.c, 83	wc.c, 529
defs.h, 113	buf.h, 92
bio.c, 82	B_DIRTY, 92
bcache, 85	B_VALID, 92
bget, <mark>83</mark>	BUSY
binit, 83	lapic.c, 245
bread, 84	bwrite
brelse, 84	bio.c, 85
buf, 85	defs.h, 114
bwrite, 85	bzero
head, 85	fs.c, 187
lock, 85	
bio.d, 88	C
block	console.c, 96
logheader, 38	kbd.h, 235
blockno	CAPSLOCK
buf, 10	kbd.h, 235
	cat
bmap	cat.c, 93
fs.c, 187	cat.c, 93
bmapstart	buf, 94
superblock, 65	
bootasm.d, 88	cat, 93
bootmain	main, 93
bootmain.c, 89	cat.d, 95
bootmain.c, 88	cgaputc
bootmain, 89	console.c, 97
readsect, 89	chan
readseg, 89	proc, 49
SECTSIZE, 88	chdir
waitdisk, 90	user.h, 451
bootmain.d, 91	checksum
BPB	mp, <mark>39</mark>
fs.h, 207	mpconf, 41
bread	chpr
bio.c, 84	defs.h, 114
	proc.c, 326
defs.h, 113	user.h, 451
brelse	clearpteu
bio.c, 84	defs.h, 115
defs.h, 114	vm.c, 516
BSIZE	cli
fs.h, 207	
bsstest	x86.h, 531
usertests.c, 468	close
buf, 10	user.h, 451
bio.c, 85	cmd, 12
	backcmd, 9

vadivasad F4	
rediremd, 54	console.c, 97
type, 12	defs.h, 116
CMOS_PORT	consoleread
lapic.c, 246	console.c, 98
cmos_read	
lapic.c, 252	console.c, 99
CMOS_RETURN	consputc
lapic.c, 246 CMOS_STATA	console.c, 99
	context, 13
lapic.c, 246 CMOS_STATB	ebp, 13
lapic.c, 246	ebx, 13 edi, 13
CMOS UIP	eip, 13
lapic.c, 246	esi, 14
cmostime	proc, 49
defs.h, 115	CONV
lapic.c, 252	lapic.c, 246
COM1	•
uart.c, 437	copyout defs.h, 117
commit	vm.c, 516
log.c, 261	copyuvm
committing	defs.h, 117
log, 36	vm.c, 517
concreate	cprintf
usertests.c, 468	console.c, 99
cons	defs.h, 118
console.c, 101	cps
CONSOLE	defs.h, 118
file.h, 179	proc.c, 326
console.c, 95	user.h, 451
BACKSPACE, 96	cpu, 14
buf, 101	apicid, 14
C, 96	gdt, 15
cgaputc, 97	intena, 15
cons, 101	ncli, 15
consoleinit, 97	proc, 15
consoleintr, 97	scheduler, 15
consoleread, 98	spinlock, 63
consolewrite, 99	started, 16
conspute, 99	ts, 16
cprintf, 99	cpuid
crt, 102	defs.h, 119
CRTPORT, 96	proc.c, 327
e, 102	cpus
input, 102	mp.c, 306
INPUT BUF, 96	proc.h, 344
lock, 102	CR0 PE
locking, 102	mmu.h, 294
panic, 100	CR0 PG
panicked, 102	mmu.h, 294
printint, 101	CR0_WP
r, 103	mmu.h, 294
w, 103	cr3
console.d, 107	taskstate, 68
consoleinit	CR4_PSE
console.c, 97	mmu.h, 295
defs.h, 116	create
consoleintr	sysfile.c, 405
	-

createdelete	dirlink, 120 dirlookup, 120
usertests.c, 469 createtest	end_op, 121
usertests.c, 470	exec, 121
crt	exit, 123
	fetchint, 123
console.c, 102 CRTPORT	fetchstr, 124
	filealloc, 124
console.c, 96	fileclose, 124
cs gatedesc, 27	filedup, 125
taskstate, 68	fileinit, 125
trapframe, 75	fileread, 125
CTL	filestat, 126
kbd.h, 236	filewrite, 126
ctlmap	fork, 127
kbd.h, 239	freevm, 128
cwd	getcallerpcs, 128
proc, 49	getprocs, 128
proc, 40	growproc, 128
data	holding, 129
buf, 10	holdingsleep, 129
ioapic, 33	ialloc, 129
pipe, 46	ideinit, 130
vm.c, 522	ideintr, 130
date.h, 107	iderw, 131
DAY	idtinit, 131
lapic.c, 247	idup, 132
day	iinit, 132
rtcdate, 56	ilock, 132
db	initlock, 133
segdesc, 59	initlog, 133
deallocuvm	initsleeplock, 133
defs.h, 119	inituvm, 134
vm.c, 517	ioapicenable, 134
DEASSERT	ioapicid, 160
lapic.c, 247	ioapicinit, 134
defs.h, 107	iput, 135
acquire, 110	ismp, 161
acquiresleep, 111	iunlock, 135
allocuvm, 111	iunlockput, 135
argint, 112	iupdate, 136
argptr, 112	kalloc, 136
argstr, 112	kbdintr, 136
begin_op, 113	kfree, 137
binit, 113	kill, 137
bread, 113	kinit1, 137
brelse, 114	kinit2, 138
bwrite, 114	kvmalloc, 138
chpr, 114	lapic, 161
clearpteu, 115	lapiceoi, 138
cmostime, 115	lapicid, 138
consoleinit, 116	lapicinit, 139
consoleintr, 116	lapicstartap, 139
copyout, 117	loaduvm, 140
copyuvm, 117	log_write, 140
cprintf, 118	memcmp, 141
cps, 118	memmove, 141
cpuid, 119	memset, 142
deallocuvm, 119	•

	microdelay, 142	DEV:	SPACE
	mpinit, 142		memlayout.h, 278
	mycpu, 143	devs	w, 1 <mark>6</mark>
	myproc, 143		file.c, 176
	namecmp, 144		file.h, 180
	namei, 144		read, 17
	nameiparent, 144		write, 17
	NELEM, 110		de, 17
	panic, 144		addrs, 17
	picenable, 145		major, 18
	picinit, 145		minor, 18
	pinit, 145		nlink, 18
	pipealloc, 145		size, 18
	pipeclose, 146		type, 18
	piperead, 146	diren	
	pipewrite, 147		inum, 19
	·		
	popcli, 147	dirfile	name, 19
	procdump, 148		
	pushcli, 148		usertests.c, 471
	readi, 148	dirlin	
	readsb, 149		defs.h, 120
	release, 149		fs.c, 188
	releasesleep, 150	dirloc	•
	safestrcpy, 150		defs.h, 120
	sched, 150		fs.c, 188
	scheduler, 151	DIRS	SIZ
	seginit, 151		fs.h, 207
	setproc, 152	dirtes	st
	setupkvm, 152		usertests.c, 472
	sleep, 152	disks	ize
	stati, 153		memide.c, 277
	strlen, 153	dpl	
	strncmp, 154	-	gatedesc, 27
	strncpy, 154		segdesc, 59
	switchkym, 154		_USER
	switchuvm, 155		mmu.h, 295
	swtch, 155	ds	- ,
	syscall, 155		taskstate, 68
	ticks, 161		trapframe, 75
	tickslock, 161	dup	trapiramo, 70
	timerinit, 156	•	user.h, 452
	tvinit, 156		u301.11, 402
	uartinit, 156	е	
	uartintr, 157		console.c, 102
		E0ES	
	uartputc, 157		kbd.h, 236
	userinit, 157	eargy	
	uva2ka, 158	U	execcmd, 23
	wait, 158	eax	CACCOTTIC, 20
	wakeup, 159		taskstate, 68
	writei, 159		
	yield, 160		trapframe, 76
DEL	LIVS	ebp	contout 10
	lapic.c, 247		context, 13
dev			taskstate, 69
	buf, 10		trapframe, 76
	inode, 30	ebx	
	log, 36		context, 13
	stat, 64		taskstate, 69
			trapframe, 76

echo.c, 164	type, 22
main, 164	version, 22
echo.d, 165	EMBRYO
echoargv	proc.h, 344
usertests.c, 492	ENABLE
ecx	lapic.c, 247
taskstate, 69	end
trapframe, 76	kalloc.c, 230
edi	end_op
context, 13	defs.h, 121
taskstate, 69	log.c, 261
trapframe, 76	entry
edx	elfhdr, 20
taskstate, 69	mpconf, 41
trapframe, 76	entryother.d, 167
efile	entrypgdir
redircmd, 55	main.c, 273
eflags	EOI
taskstate, 69	lapic.c, 247
trapframe, 77	err
ehsize	trapframe, 77
elfhdr, 20	ERROR
eip	lapic.c, 247
context, 13	•
•	es tacketete 70
taskstate, 70	taskstate, 70
trapframe, 77	trapframe, 77
elf	esi
elfhdr, 20	context, 14
elf.h, 165	taskstate, 70
ELF_MAGIC, 165	trapframe, 77
ELF_PROG_FLAG_EXEC, 165	esp
ELF_PROG_FLAG_READ, 166	taskstate, 70
ELF_PROG_FLAG_WRITE, 166	trapframe, 78
ELF PROG LOAD, 166	esp0
ELF MAGIC	taskstate, 70
elf.h, 165	esp1
ELF_PROG_FLAG_EXEC	taskstate, 70
elf.h, 165	
	esp2
ELF_PROG_FLAG_READ	taskstate, 71
elf.h, 166	ESR
ELF_PROG_FLAG_WRITE	lapic.c, 248
elf.h, 166	EXEC
ELF_PROG_LOAD	sh.c, 351
elf.h, 166	exec
elfhdr, 20	defs.h, 121
ehsize, 20	exec.c, 167
elf, 20	user.h, 452
entry, 20	exec.c, 167
flags, 21	exec, 167
machine, 21	exec.d, 170
magic, 21	execcmd, 23
phentsize, 21	argv, 23
•	•
phnum, 21	eargv, 23
phoff, 21	sh.c, 352
shentsize, 22	type, 23
shnum, 22	
aboff 22	exectest
shoff, 22	usertests.c, 472
shstrndx, 22	

defs.h, 123	ftable, 176
proc.c, 327	lock, 177
user.h, 453	file.d, 179
exit.c, 171	file.h, 179
main, 171	CONSOLE, 179
exit.d, 171	devsw, 180
exitiputtest	filealloc
usertests.c, 472	defs.h, 124
exitwait	file.c, 173
usertests.c, 473	fileclose
EXTMEM	defs.h, 124
memlayout.h, 278	file.c, 174
•	filedup
fcntl.h, 172	defs.h, 125
O_CREATE, 172	file.c, 174
O_RDONLY, 172	fileinit
O_RDWR, 172	defs.h, 125
O_WRONLY, 172	file.c, 174
fd	fileread
redircmd, 55	defs.h, 125
FD_INODE	file.c, 175
file, 24	filestat
FD NONE	defs.h, 126
_ file, 24	file.c, 175
FD PIPE	filesz
file, 24	
fdalloc	proghdr, 53
sysfile.c, 405	filewrite
feature	defs.h, 126
mpproc, 45	file.c, 175
fetchint	fill_rtcdate
defs.h, 123	lapic.c, 253
	FIXED
syscall.c, 385 fetchstr	lapic.c, 248
	FL_IF
defs.h, 124	mmu.h, 295
syscall.c, 385	flags
file, 24	buf, 11
FD_INODE, 24	elfhdr, 21
FD_NONE, 24	mpioapic, 44
FD_PIPE, 24	mpproc, 45
file.c, 176	proghdr, 53
ip, 25	fmtname
off, 25	ls.c, 268
pipe, 25	foo.c, 180
readable, 25	main, 181
redircmd, 55	foo.d, 182
ref, 25	fork
type, 26	defs.h, 127
writable, 26	proc.c, 328
file.c, 173	user.h, 454
devsw, 176	fork1
file, 176	sh.c, 352
filealloc, 173	forkret
fileclose, 174	proc.c, 329
filedup, 174	forktest
fileinit, 174	forktest.c, 183
fileread, 175	usertests.c, 473
filestat, 175	forktest.c, 182
filewrite, 175	101111031.0, 10Z
,	

forktest, 183	writei, 197
main, 183	fs.d, 206
N, 182	fs.h, 206
printf, 183	BBLOCK, 207
forktest.d, 185	BPB, 207
fourfiles	BSIZE, 207
usertests.c, 474	DIRSIZ, 207
fourteen	IBLOCK, 207
usertests.c, 475	IPB, 207
free	MAXFILE, 208
umalloc.c, 446	NDIRECT, 208
user.h, 455	NINDIRECT, 208
freeblock	ROOTINO, 208
mkfs.c, 287	fsfd
freeinode	mkfs.c, 288
mkfs.c, 288	fsfull
freelist	usertests.c, 476
kalloc.c, 230	FSSIZE
freep	param.h, 312
umalloc.c, 448	fstat
freerange	user.h, 455
kalloc.c, 228	ftable
freevm	file.c, 176
defs.h, 128	
vm.c, 518	g
fs	segdesc, 60
	gatedesc, 26
taskstate, 71	args, 27
trapframe, 78	
fs.c, 185	cs, 27
balloc, 186	dpl, 27
bfree, 186	off_15_0, 27
	off_31_16, 27
bmap, 187	
bzero, 187	p, 27
dirlink, 188	rsv1, <mark>28</mark>
dirlookup, 188	s, 28
ialloc, 189	type, 28
	gdt
icache, 197	
idup, 189	cpu, 15
iget, 190	getcallerpcs
iinit, 190	defs.h, 128
	spinlock.c, 371
ilock, 191	getcmd
inode, 198	-
iput, 191	sh.c, 352
itrunc, 192	getpid
iunlock, 192	user.h, 455
,	getprocs
iunlockput, 193	defs.h, 128
iupdate, 193	
lock, 198	user.h, 455
min, 186	gets
namecmp, 193	ulib.c, 441
•	user.h, 456
namei, 194	
nameiparent, 194	gettoken
namex, 194	sh.c, <mark>353</mark>
readi, 195	grep
	grep.c, 210
readsb, 196	
sb, 198	grep.c, 209
skipelem, 196	buf, 212
stati, 196	grep, 210
otati, 100	main, 210
	,

match, 211	IDE_DRDY, 215
matchhere, 211	IDE ERR, 216
matchstar, 211	ideinit, 216
grep.d, 213	ideintr, 216
growproc	idelock, 218
defs.h, 128	idequeue, 219
	•
proc.c, 329	iderw, 217
gs	idestart, 217
taskstate, 71	idewait, 218
trapframe, 78	SECTOR_SIZE, 216
	ide.d, 221
halt	IDE_BSY
user.h, 456	ide.c, 214
havedisk1	IDE_CMD_RDMUL
ide.c, 218	ide.c, 215
head	IDE_CMD_READ
bio.c, 85	ide.c, 215
Header	IDE CMD WRITE
umalloc.c, 446	ide.c, 215
header, 28	IDE CMD WRMUL
ptr, 29	ide.c, 215
s, 29	IDE DF
size, 29	ide.c, 215
x, 29	IDE DRDY
holding	-
defs.h, 129	ide.c, 215
spinlock.c, 371	IDE_ERR
•	ide.c, 216
holdingsleep	ideinit
defs.h, 129	defs.h, 130
sleeplock.c, 368	ide.c, 216
hour	memide.c, 275
rtcdate, 56	ideintr
HOURS	defs.h, 130
lapic.c, 248	ide.c, 216
	memide.c, 275
ialloc	idelock
defs.h, 129	ide.c, 218
fs.c, 189	idequeue
mkfs.c, 283	ide.c, 219
iappend	iderw
mkfs.c, 283	defs.h, 131
IBLOCK	ide.c, 217
fs.h, 207	memide.c, 276
icache	idestart
fs.c, 197	
ICRHI	ide.c, 217
lapic.c, 248	idewait
ICRLO	ide.c, 218
lapic.c, 248	idt
ID	trap.c, 426
	idtinit
lapic.c, 248	defs.h, 131
ide.c, 214	trap.c, 425
havedisk1, 218	idup
IDE_BSY, 214	defs.h, 132
IDE_CMD_RDMUL, 215	fs.c, 189
IDE_CMD_READ, 215	iget
IDE_CMD_WRITE, 215	fs.c, 190
IDE_CMD_WRMUL, 215	iinit
IDE_DF, 215	

defs.h, 132	ioapic.c, 224
fs.c, 190	INT_LEVEL
ilock	ioapic.c, 224
defs.h, 132	INT_LOGICAL
fs.c, 191	ioapic.c, 224
imcrp	intena
mp, 39	cpu, 15
inb	inum
x86.h, 531	dirent, 19
INIT	inode, 30
lapic.c, 249	IO_PIC1
init.c, 221	picirq.c, 315
argv, <mark>222</mark>	IO_PIC2
main, 221	picirq.c, 315
init.d, 223	IOAPIC
initcode.d, 223	ioapic.c, 224
initlock	ioapic, 32
defs.h, 133	data, 33
spinlock.c, 372	ioapic.c, 226
initlog	pad, 33
defs.h, 133	reg, 33
log.c, 262	ioapic.c, 223
initproc	INT_ACTIVELOW, 224
·	INT_ACTIVELOW, 224
proc.c, 335	_
initsleeplock	INT_LEVEL, 224
defs.h, 133	INT_LOGICAL, 224
sleeplock.c, 368	IOAPIC, 224
inituvm	ioapic, 226
defs.h, 134	ioapicenable, 225
vm.c, 518	ioapicinit, 225
ino	ioapicread, 226
stat, 64	ioapicwrite, 226
inode, 30	REG_ID, 225
addrs, 30	REG_TABLE, 225
dev, 30	REG_VER, 225
fs.c, 198	ioapic.d, 228
inum, 30	ioapicenable
lock, 31	defs.h, 134
major, 31	ioapic.c, 225
minor, 31	ioapicid
nlink, 31	defs.h, 160
ref, 31	mp.c, 306
size, 32	ioapicinit
type, 32	defs.h, 134
valid, 32	ioapic.c, 225
inodestart	ioapicread
superblock, 66	ioapic.c, 226
input 100	ioapicwrite
console.c, 102	ioapic.c, 226
INPUT_BUF	iomb
console.c, 96	taskstate, 71
insl	ip
x86.h, 531	file, 25
install_trans	IPB
log.c, 262	fs.h, 207
INT_ACTIVELOW	iput
ioapic.c, 224	defs.h, 135
INT_DISABLED	fs.c, 191

iputtest	KBS_DIB, 236
usertests.c, 476	KBSTATP, 236
iref	KEY_DEL, 236
usertests.c, 477	KEY_DN, 237
IRQ COM1	KEY_END, 237
traps.h, 430	KEY HOME, 237
IRQ ERROR	KEY INS, 237
_	-
traps.h, 430	KEY_LF, 237
IRQ_IDE	KEY_PGDN, 237
traps.h, 430	KEY_PGUP, 238
IRQ_KBD	KEY_RT, 238
traps.h, 430	KEY_UP, 238
IRQ_SPURIOUS	NO, 238
traps.h, 430	normalmap, 239
IRQ_TIMER	NUMLOCK, 238
traps.h, 431	SCROLLLOCK, 238
isdirempty	SHIFT, 239
sysfile.c, 406	shiftcode, 240
ismp	shiftmap, 240
defs.h, 161	togglecode, 241
itrunc	KBDATAP
fs.c, 192	kbd.h, 236
iunlock	kbdgetc
defs.h, 135	kbd.c, 233
fs.c, 192	kbdintr
iunlockput	defs.h, 136
defs.h, 135	kbd.c, 233
fs.c, 193	KBS DIB
	kbd.h, 236
iupdate	KBSTATP
defs.h, 136	-
fs.c, 193	kbd.h, 236
fs.c, 193	kbd.h, 236 KERNBASE
fs.c, 193 kalloc	kbd.h, 236 KERNBASE memlayout.h, 278
fs.c, 193 kalloc defs.h, 136	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbd.c, 232	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbdgetc, 233	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGDN kbd.h, 237
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbdgetc, 233 kbdintr, 233	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGDN kbd.h, 237
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbd.c, 232 kbdgetc, 233 kbdintr, 233 kbd.d, 234	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGUP kbd.h, 238 KEY_RT
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbd.c, 232 kbdgetc, 233 kbdintr, 233 kbd.d, 234 kbd.h, 234	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGUP kbd.h, 238 KEY_RT kbd.h, 238
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbd.c, 232 kbdgetc, 233 kbdintr, 233 kbd.d, 234 kbd.h, 234 ALT, 235	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGUP kbd.h, 238 KEY_RT kbd.h, 238 KEY_UP
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbd.c, 232 kbdgetc, 233 kbdintr, 233 kbd.d, 234 kbd.h, 234 ALT, 235 C, 235 CAPSLOCK, 235	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGUP kbd.h, 238 KEY_RT kbd.h, 238 KEY_UP kbd.h, 238
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbd.c, 232 kbdgetc, 233 kbdintr, 233 kbd.h, 234 ALT, 235 C, 235 CAPSLOCK, 235 CTL, 236	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGUP kbd.h, 238 KEY_RT kbd.h, 238 KEY_UP kbd.h, 238 Kfree
kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbd.c, 232 kbdgetc, 233 kbdintr, 233 kbd.h, 234 ALT, 235 C, 235 CAPSLOCK, 235 CTL, 236 ctlmap, 239	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGUP kbd.h, 238 KEY_RT kbd.h, 238 KEY_UP kbd.h, 238 Kfree defs.h, 137
fs.c, 193 kalloc defs.h, 136 kalloc.c, 229 kalloc.c, 228 end, 230 freelist, 230 freerange, 228 kalloc, 229 kfree, 229 kinit1, 229 kinit2, 230 kmem, 230 lock, 230 use_lock, 231 kalloc.d, 232 kbd.c, 232 kbdgetc, 233 kbdintr, 233 kbd.h, 234 ALT, 235 C, 235 CAPSLOCK, 235 CTL, 236	kbd.h, 236 KERNBASE memlayout.h, 278 KERNLINK memlayout.h, 279 KEY_DEL kbd.h, 236 KEY_DN kbd.h, 237 KEY_END kbd.h, 237 KEY_HOME kbd.h, 237 KEY_INS kbd.h, 237 KEY_LF kbd.h, 237 KEY_LF kbd.h, 237 KEY_PGDN kbd.h, 237 KEY_PGUP kbd.h, 238 KEY_RT kbd.h, 238 KEY_UP kbd.h, 238 Kfree

kill	ICRLO, 248
defs.h, 137	ID, 248
proc.c, 329	INIT, 249
user.h, 456	lapic, 256
kill.c, 243	lapiceoi, 253
main, 243	lapicid, 253
kill.d, 244	lapicinit, 254
killed	lapicstartap, 254
	lapicw, 255
proc, 50	•
kinit1	LEVEL, 249
defs.h, 137	LINT0, 249
kalloc.c, 229	LINT1, 249
kinit2	MASKED, 249
defs.h, 138	microdelay, 255
kalloc.c, 230	MINS, 249
kmap, 33	MONTH, 250
perm, 34	PCINT, 250
phys_end, 34	PERIODIC, 250
phys_start, 34	SECS, 250
virt, 34	STARTUP, 250
vm.c, <u>522</u>	SVR, 250
kmem	TCCR, 251
kalloc.c, 230	TDCR, 251
kpgdir	TICR, 251
vm.c, 523	TIMER, 251
kstack	TPR, 251
proc, 50	VER, 251
KSTACKSIZE	X1, 252
param.h, 313	YEAR, 252
kvmalloc	lapic.d, 259
defs.h, 138	lapicaddr
vm.c, 519	mpconf, 41
lonio	lapiceoi
lapic	defs.h, 138
defs.h, 161	lapic.c, 253
lapic.c, 256	lapicid
lapic.c, 244	defs.h, 138
ASSERT, 245	lapic.c, 253
BCAST, 245	lapicinit
BUSY, 245	defs.h, 139
CMOS_PORT, 246	lapic.c, 254
cmos_read, 252	lapicstartap
CMOS_RETURN, 246	defs.h, 139
CMOS_STATA, 246	lapic.c, 254
CMOS_STATB, 246	lapicw
CMOS_UIP, 246	lapic.c, 255
cmostime, 252	lcr3
CONV, 246	
DAY, 247	x86.h, 531
DEASSERT, 247	ldt
DELIVS, 247	taskstate, 71
ENABLE, 247	left
EOI, 247	listcmd, 35
	pipecmd, 48
ERROR, 247	length
ESR, 248	mp, 39
fill_rtcdate, 253	mpconf, 41
FIXED, 248	LEVEL
HOURS, 248	lapic.c, 249
ICRHI, 248	-

lgdt	lh, 36
x86.h, 532	lock, 36
lh	log.c, 264
log, 36	outstanding, 37
lidt	size, 37
x86.h, 532	start, 37
lim_15_0	log.c, 260
segdesc, 60	begin_op, 260
lim_19_16	commit, 261
segdesc, 60	end op, 261
link	initlog, 262
taskstate, 71	install_trans, 262
user.h, 456	log, 264
linktest	log_write, 262
	-
usertests.c, 477 linkunlink	read_head, 263
	recover_from_log, 263
usertests.c, 478	write_head, 263
LINTO	write_log, 264
lapic.c, 249	log.d, 267
LINT1	log_write
lapic.c, 249	defs.h, 140
LIST	log.c, 262
sh.c, 351	logheader, 37
listcmd, 35	block, 38
left, 35	n, 38
right, 35	LOGSIZE
sh.c, 353	param.h, 313
type, 35	logstart
lk	superblock, 66
sleeplock, 61	ls
In.c, 259	ls.c, 268
main, 259	ls.c, 268
In.d, 260	fmtname, 268
loadgs	ls, 268
x86.h, 532	main, 269
loaduvm	ls.d, 271
defs.h, 140	ltr
vm.c, 519	x86.h, 533
lock	
bio.c, 85	machine
buf, 11	elfhdr, 21
console.c, 102	magic
file.c, 177	elfhdr, 21
fs.c, 198	main
inode, 31	cat.c, 93
kalloc.c, 230	echo.c, 164
log, 36	exit.c, 171
pipe, 46	foo.c, 181
proc.c, 335	forktest.c, 183
sysproc.c, 421	grep.c, 210
locked	init.c, 221
sleeplock, 61	kill.c, 243
·	In.c, 259
spinlock, 63	Is.c, 269
locking	mkdir.c, 280
console.c, 102	mkfs.c, 284
log, 36	nice.c, 311
committing, 36	ps.c, 345
dev, 36	pstree.c, 347
	police.c, 077

0.40	DEVODAGE 070
rm.c, 348	DEVSPACE, 278
sh.c, 354	EXTMEM, 278
shutdown.c, 367	KERNBASE, 278
stressfs.c, 377	KERNLINK, 279
usertests.c, 479	P2V, 279
wc.c, 528	P2V_WO, 279
zombie.c, 538	PHYSTOP, 279
main.c, 271	V2P, 279
attribute, 271	V2P WO, 279
entrypgdir, 273	memmove
mpenter, 271	
·	defs.h, 141
mpmain, 272	string.c, 379
startothers, 272	ulib.c, 441
main.d, 275	user.h, 457
major	memset
dinode, 18	defs.h, 142
inode, 31	string.c, 380
malloc	ulib.c, 441
umalloc.c, 446	user.h, 457
user.h, 457	memsz
mappages	proghdr, 53
vm.c, 519	microdelay
MASKED	defs.h, 142
_	
lapic.c, 249	lapic.c, 255
match	min
grep.c, 211	fs.c, 186
matchhere	mkfs.c, 282
grep.c, 211	minor
matchstar	dinode, 18
grep.c, 211	inode, 31
MAXARG	MINS
param.h, 313	lapic.c, 249
MAXARGS	minute
sh.c, 351	rtcdate, 57
MAXFILE	mkdir
fs.h, 208	user.h, 458
MAXOPBLOCKS	mkdir.c, 280
param.h, 313	main, 280
MAXPROC	mkdir.d, 281
pstree.c, 346	mkfs.c, 281
mem	balloc, 283
usertests.c, 480	freeblock, 287
memcmp	freeinode, 288
defs.h, 141	fsfd, 288
string.c, 379	ialloc, 283
memcpy	iappend, 283
string.c, 379	main, 284
memdisk	min, 282
memide.c, 277	nbitmap, 288
memide.c, 275	nblocks, 288
_binary_fs_img_size, 276	ninodeblocks, 288
	NINODES, 282
_binary_fs_img_start, 276	
disksize, 277	nlog, 289
ideinit, 275	1 000
	nmeta, 289
ideintr, 275	rinode, 286
ideintr, 275 iderw, 276	rinode, 286 rsect, 286
ideintr, 275 iderw, 276 memdisk, 277	rinode, 286
ideintr, 275 iderw, 276	rinode, 286 rsect, 286

static_assert, 283	length, 39
winode, 286	physaddr, 39
wsect, 286	reserved, 39
xint, 287	signature, 40
xshort, 287	specrev, 40
zeroes, 289	type, 40
mknod	mp.c, 303
user.h, 458	cpus, 306
mmu.h, 293	ioapicid, 306
CR0 PE, 294	mpconfig, 304
CR0 PG, 294	mpinit, 304
CR0_WP, 294	mpsearch, 305
CR4_PSE, 295	mpsearch1, 305
DPL_USER, 295	ncpu, 306
FL_IF, 295	sum, 306
NPDENTRIES, 295	mp.d, 308
NPTENTRIES, 295	mp.h, 309
NSEGS, 295	MPBOOT, 309
PDX, 296	MPBUS, 309
PDXSHIFT, 296	MPIOAPIC, 309
PGADDR, 296	MPIOINTR, 309
PGROUNDDOWN, 296	MPLINTR, 310
PGROUNDUP, 296	MPPROC, 310
PGSIZE, 296	MPBOOT
PTE_ADDR, 297	mp.h, 309
PTE_FLAGS, 297	MPBUS
PTE_P, 297	mp.h, 309
PTE_PS, 297	mpconf, 40
pte_t, 301	checksum, 41
PTE_U, 297	entry, 41
PTE_W, 297	lapicaddr, 41
PTX, 298	length, 41
PTXSHIFT, 298	oemlength, 41
SEG, 298	oemtable, 42
SEG16, 298	product, 42
SEG_KCODE, 298	reserved, 42
SEG_KDATA, 299	signature, 42
SEG_TSS, 299	version, 42
SEG_UCODE, 299	xchecksum, 42
SEG_UDATA, 299	xlength, 43
SETGATE, 299	mpconfig
STA_R, 300	mp.c, 304
STA_W, 300	mpenter
STA X, 300	main.c, 271
STS_IG32, 300	mpinit
STS T32A, 300	defs.h, 142
STS TG32, 301	mp.c, 304
mode	MPIOAPIC
redircmd, 55	mp.h, 309
MONTH	mpioapic, 43
lapic.c, 250	addr, 43
month	apicno, 43
rtcdate, 57	flags, 44
morecore	type, 44
umalloc.c, 447	version, 44
	MPIOINTR
mp, 38	
checksum, 39	mp.h, 309 MPLINTR
imcrp, 39	IVIF LIIV I T

mp.h, 310	proc.h, 344
mpmain	NDEV
main.c, 272 MPPROC	param.h, 314 NDIRECT
mp.h, 310	fs.h, 208
mpproc, 44	NELEM
apicid, 45	defs.h, 110
feature, 45	next
flags, 45	buf, 11
reserved, 45	run, 58
signature, 45	nextpid
type, 45	proc.c, 335
version, 46	NFILE
mpsearch	param.h, 314
mp.c, 305	nice.c, 311
mpsearch1	main, 311
mp.c, 305	nice.d, 312
mycpu	NINDIRECT
defs.h, 143	fs.h, 208
proc.c, 330	NINODE
myproc	param.h, 314
defs.h, 143	ninodeblocks
proc.c, 330	mkfs.c, 288
	NINODES
N () I do	mkfs.c, 282
forktest.c, 182	ninodes
n lambaadar 00	superblock, 66
logheader, 38	nlink
name	dinode, 18
dirent, 19	inode, 31
proc, 50	stat, 64
sleeplock, 62 spinlock, 63	nlog
·	mkfs.c, 289
uproc, 80 usertests.c, 492	superblock, 66
namecmp	nmeta
defs.h, 144	mkfs.c, 289
fs.c. 193	NO
namei	kbd.h, 238
defs.h, 144	NOFILE
fs.c, 194	param.h, 314
nameiparent	normalmap kbd.h, 239
defs.h, 144	NPDENTRIES
fs.c, 194	mmu.h, 295
namex	NPROC
fs.c, 194	param.h, 314
nbitmap	NPTENTRIES
mkfs.c, 288	mmu.h, 295
nblocks	nread
mkfs.c, 288	pipe, 47
superblock, 66	NSEGS
NBUF	mmu.h, 295
param.h, 313	nulterminate
ncli	sh.c, 354
cpu, 15	NUMLOCK
NCPU	kbd.h, 238
param.h, 313	nwrite
ncpu	pipe, 47
mp.c, 306	

O_CREATE	taskstate, 72
fcntl.h, 172	trapframe, 79
O_RDONLY	padding5
fcntl.h, 172	taskstate, 72
O_RDWR	trapframe, 79
fcntl.h, 172	padding6
O_WRONLY	taskstate, 73
fcntl.h, 172	trapframe, 79
oemlength	padding7
mpconf, 41	taskstate, 73
oemtable	padding8
mpconf, 42	taskstate, 73
oesp	padding9
trapframe, 78	taskstate, 73
off	paddr
file, 25	proghdr, 53
proghdr, 53	panic
off_15_0	console.c, 100
gatedesc, 27	defs.h, 144
off_31_16	sh.c, 355
gatedesc, 27 ofile	panicked console.c, 102
proc, 50	param.h, 312
	FSSIZE, 312
open user.h, 458	KSTACKSIZE, 313
openiputtest	LOGSIZE, 313
usertests.c, 480	MAXARG, 313
opentest	MAXOPBLOCKS, 313
usertests.c, 481	NBUF, 313
outb	NCPU, 313
x86.h, 533	NDEV, 314
outsl	NFILE, 314
x86.h, 533	NINODE, 314
outstanding	NOFILE, 314
log, 37	NPROC, 314
outw	ROOTDEV, 314
x86.h, 533	parent
,	proc, 50
p	parseblock
gatedesc, 27	sh.c, 355
segdesc, 60	parsecmd
P2V	sh.c, 355
memlayout.h, 279	parseexec
P2V_WO	sh.c, 356
memlayout.h, 279	parseline
pad	sh.c, 356
ioapic, 33	parsepipe
padding1	sh.c, 357
taskstate, 72	parseredirs
trapframe, 78	sh.c, 357
padding10	PCINT
taskstate, 72	lapic.c, 250
padding2	pcs
taskstate, 72	spinlock, 63
trapframe, 78	pde_t
padding3	types.h, 435
taskstate, 72	PDX
trapframe, 79	mmu.h, 296
padding4	

PDXSHIFT	user.h, 458
mmu.h, 296	writeopen, 47
peek	pipe.c, 317
sh.c, 358	pipealloc, 317
PERIODIC	pipeclose, 318
lapic.c, 250	piperead, 318
perm	PIPESIZE, 317
kmap, 34	pipewrite, 319
PGADDR	pipe.d, 321
mmu.h, 296	pipe1
pgdir	usertests.c, 481
proc, 51	pipealloc
PGROUNDDOWN	defs.h, 145
mmu.h, 296	pipe.c, 317
PGROUNDUP	pipe.c, 317
mmu.h, 296	defs.h, 146
PGSIZE	pipe.c, 318
mmu.h, 296	pipecmd, 48
phentsize	left, 48
elfhdr, 21	right, 48
phnum	sh.c, 358
elfhdr, 21	type, 48
phoff	piperead
elfhdr, 21	defs.h, 146
phys_end	pipe.c, 318
kmap, 34	PIPESIZE
phys_start	pipe.c, 317
kmap, 34	pipewrite
physaddr	defs.h, 147
mp, 39	pipe.c, 319
mp, 39 PHYSTOP	pipe.c, 319 popcli
PHYSTOP	popcli
PHYSTOP memlayout.h, 279	popcli defs.h, 147
PHYSTOP memlayout.h, 279 picenable	popcli defs.h, 147 spinlock.c, 372
PHYSTOP memlayout.h, 279 picenable defs.h, 145	popcli defs.h, 147 spinlock.c, 372 ppid
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 picirq.d, 316 pid proc, 51	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printfnt, 322 printint, 322
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324 printint
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330 PIPE	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324 printint console.c, 101
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330 PIPE sh.c, 351	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printf, 322 printint, 322 putc, 323 printf.d, 324 printint console.c, 101 printf.c, 322
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330 PIPE	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324 printint console.c, 101
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330 PIPE sh.c, 351 pipe, 46 data, 46	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324 printf.d, 324 printint console.c, 101 printf.c, 322 priority proc, 51
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330 PIPE sh.c, 351 pipe, 46	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324 printint console.c, 101 printf.c, 322 priority
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330 PIPE sh.c, 351 pipe, 46 data, 46	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324 printf.d, 324 printint console.c, 101 printf.c, 322 priority proc, 51
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330 PIPE sh.c, 351 pipe, 46 data, 46 file, 25	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324 printint console.c, 101 printf.c, 322 priority proc, 51 proc, 49
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330 PIPE sh.c, 351 pipe, 46 data, 46 file, 25 lock, 46	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324 printint console.c, 101 printf.c, 322 priority proc, 51 proc, 49 chan, 49
PHYSTOP memlayout.h, 279 picenable defs.h, 145 picinit defs.h, 145 picirq.c, 316 picirq.c, 315 IO_PIC1, 315 IO_PIC2, 315 picinit, 316 picirq.d, 316 pid proc, 51 sleeplock, 62 uproc, 80 pinit defs.h, 145 proc.c, 330 PIPE sh.c, 351 pipe, 46 data, 46 file, 25 lock, 46 nread, 47	popcli defs.h, 147 spinlock.c, 372 ppid uproc, 80 preempt usertests.c, 482 prev buf, 11 printf forktest.c, 183 printf.c, 322 user.h, 459 printf.c, 321 printf, 322 printint, 322 putc, 323 printf.d, 324 printf.d, 324 printint console.c, 101 printf.c, 322 priority proc, 51 proc, 49 chan, 49 context, 49

killed, 50	mpconf, 42
kstack, 50	proghdr, 52
name, 50	align, 52
ofile, 50	filesz, 53
parent, 50	flags, 53
pgdir, 51	memsz, 53
pid, 51	off, 53
priority, 51	paddr, 53
proc.c, 336	type, 54
state, 51	vaddr, 54
sysproc.c, 422	ps.c, 345
sz, 51	main, 345
tf, 52	ps.d, 346
proc.c, 324	pstree.c, 346
allocproc, 325	main, 347
chpr, 326	MAXPROC, 346
cps, 326	pstree.d, 348
cpuid, 327	ptable
exit, 327	proc.c, 336
fork, 328	sysproc.c, 422
forkret, 329	PTE_ADDR
growproc, 329	mmu.h, 297
initproc, 335	PTE_FLAGS
kill, 329	mmu.h, 297
lock, 335	PTE_P
mycpu, 330	mmu.h, 297
myproc, 330	PTE_PS
nextpid, 335	mmu.h, 297
pinit, 330	pte_t
proc, 336	mmu.h, 301
procdump, 331	PTE_U
ptable, 336	mmu.h, 297
sched, 331	PTE_W
scheduler, 331	mmu.h, 297
sleep, 332	ptr
trapret, 333	header, 29
userinit, 333	PTX
wait, 333	mmu.h, 298
wakeup, 334	PTXSHIFT
wakeup1, 334	mmu.h, 298
yield, 335	pushcli
proc.d, 343	defs.h, 148
proc.h, 343	spinlock.c, 372
cpus, 344	putc
EMBRYO, 344	printf.c, 323
ncpu, 344	gnovt
procstate, 343	qnext
RUNNABLE, 344	buf, 11
RUNNING, 344	r
SLEEPING, 344	console.c, 103
UNUSED, 344	rand
ZOMBIE, 344	usertests.c, 483
procdump	randstate
defs.h, 148	usertests.c, 493
proc.c, 331	rcr2
procstate	x86.h, 534
proc.h, 343	read
product	devsw, 17
	G0.0, 17

user.h, 459	rinode
read_head	mkfs.c, 286
log.c, 263	rm.c, 348
readable	main, 348
file, 25	rm.d, 349
readeflags	rmdot
x86.h, 534	usertests.c, 483
readi	ROOTDEV
defs.h, 148	param.h, 314
fs.c, 195	ROOTINO
README.md, 348	fs.h, 208
readopen	rsect
pipe, 47	mkfs.c, 286
readsb	rsv1
defs.h, 149	gatedesc, 28
fs.c, 196	segdesc, 60
readsect	RTC_ADDR
bootmain.c, 89	usertests.c, 464
readseg	RTC DATA
bootmain.c, 89	usertests.c, 465
recover from log	rtcdate, 56
<u>~</u>	
log.c, 263	day, 56
REDIR	hour, 56
sh.c, 351	minute, 57
redircmd, 54	month, 57
cmd, 54	second, 57
efile, 55	year, <mark>57</mark>
fd, 55	run, <mark>58</mark>
file, 55	next, 58
mode, 55	runcmd
sh.c, 358	sh.c, 359
type, 55	RUNNABLE
ref	proc.h, 344
file, 25	RUNNING
inode, 31	proc.h, 344
refcnt	
buf, 12	S
reg	gatedesc, 28
ioapic, 33	header, 29
REG ID	segdesc, 60
 ioapic.c, 225	safestrcpy
REG TABLE	defs.h, 150
	•
ioapic.c. 225	string.c, 380
ioapic.c, 225	
REG_VER	string.c, 380
REG_VER ioapic.c, 225	string.c, 380 sb
REG_VER ioapic.c, 225 release	string.c, 380 sb fs.c, 198
REG_VER ioapic.c, 225 release defs.h, 149	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373	string.c, 380 sb fs.c, 198 mkfs.c, 289
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep defs.h, 150	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest usertests.c, 483
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep defs.h, 150 sleeplock.c, 369	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest usertests.c, 483 sched
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep defs.h, 150 sleeplock.c, 369 reserved	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest usertests.c, 483 sched defs.h, 150
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep defs.h, 150 sleeplock.c, 369 reserved mp, 39	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest usertests.c, 483 sched defs.h, 150 proc.c, 331
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep defs.h, 150 sleeplock.c, 369 reserved mp, 39 mpconf, 42	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest usertests.c, 483 sched defs.h, 150 proc.c, 331 scheduler
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep defs.h, 150 sleeplock.c, 369 reserved mp, 39 mpconf, 42 mpproc, 45	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest usertests.c, 483 sched defs.h, 150 proc.c, 331 scheduler cpu, 15
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep defs.h, 150 sleeplock.c, 369 reserved mp, 39 mpconf, 42 mpproc, 45 right	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest usertests.c, 483 sched defs.h, 150 proc.c, 331 scheduler cpu, 15 defs.h, 151
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep defs.h, 150 sleeplock.c, 369 reserved mp, 39 mpconf, 42 mpproc, 45 right listcmd, 35	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest usertests.c, 483 sched defs.h, 150 proc.c, 331 scheduler cpu, 15 defs.h, 151 proc.c, 331
REG_VER ioapic.c, 225 release defs.h, 149 spinlock.c, 373 releasesleep defs.h, 150 sleeplock.c, 369 reserved mp, 39 mpconf, 42 mpproc, 45 right	string.c, 380 sb fs.c, 198 mkfs.c, 289 sbrk user.h, 459 sbrktest usertests.c, 483 sched defs.h, 150 proc.c, 331 scheduler cpu, 15 defs.h, 151

second	LIST, 351
rtcdate, 57	listcmd, 353
SECS	main, 354
lapic.c, 250	MAXARGS, 351
SECTOR_SIZE	nulterminate, 354
ide.c, 216	panic, 355
SECTSIZE	parseblock, 355
bootmain.c, 88	parsecmd, 355
SEG	parseexec, 356
	-
mmu.h, 298	parseline, 356
SEG16	parsepipe, 357
mmu.h, 298	parseredirs, 357
SEG_ASM	peek, 358
asm.h, 81	PIPE, 351
SEG_KCODE	pipecmd, 358
mmu.h, 298	REDIR, 351
SEG_KDATA	redircmd, 358
mmu.h, 299	runcmd, 359
SEG_NULLASM	symbols, 360
asm.h, 81	whitespace, 360
SEG_TSS	sh.d, 366
mmu.h, 299	sharedfd
SEG_UCODE	usertests.c, 485
mmu.h, 299	shentsize
SEG UDATA	elfhdr, 22
mmu.h, 299	SHIFT
segdesc, 58	kbd.h, 239
avl, 59	shiftcode
base_15_0, 59	kbd.h, 240
base_23_16, 59	shiftmap
base_31_24, 59	kbd.h, 240
db, 59	shnum
dpl, 59	elfhdr, 22
•	shoff
g, 60	
lim_15_0, 60	elfhdr, 22
lim_19_16, 60	shstrndx
p, 60	elfhdr, 22
rsv1, 60	shutdown.c, 366
s, 60	main, 367
type, 61	shutdown.d, 367
seginit	signature
defs.h, 151	mp, 40
vm.c, 520	mpconf, 42
SETGATE	mpproc, 45
mmu.h, 299	size
setproc	dinode, 18
defs.h, 152	header, 29
setupkvm	inode, 32
defs.h, 152	log, 37
vm.c, 520	stat, 64
sh.c, 349	superblock, 67
BACK, 350	skipelem
backcmd, 352	fs.c, 196
EXEC, 351	sleep
	•
execcmd, 352	defs.h, 152
fork1, 352	proc.c, 332
getcmd, 352	user.h, 459
gettoken, 353	SLEEPING

1.044	
proc.h, 344	ino, <mark>64</mark>
sleeplock, 61	mkfs.c, 282
lk, 61	nlink, 64
locked, 61	size, 64
name, 62	type, 65
pid, 62	ulib.c, 442
sleeplock.c, 368	user.h, 459
acquiresleep, 368	stat.h, 375
holdingsleep, 368	T_DEV, 376
initsleeplock, 368	T DIR, 376
releasesleep, 369	T FILE, 376
sleeplock.d, 370	state
sleeplock.h, 370	
•	proc, 51
specrev	stati
mp, 40	defs.h, 153
spinlock, 62	fs.c, 196
cpu, 63	static_assert
locked, 63	mkfs.c, 283
name, 63	stdout
pcs, 63	usertests.c, 493
spinlock.c, 370	sti
acquire, 371	x86.h, 534
getcallerpcs, 371	stosb
holding, 371	x86.h, 534
initlock, 372	stosl
popcli, 372	x86.h, 535
pushcli, 372	strchr
release, 373	ulib.c, 442
spinlock.d, 375	user.h, 460
spinlock.h, 375	strcmp
SS	ulib.c, 442
taskstate, 73	user.h, 460
trapframe, 79	strcpy
ss0	ulib.c, 443
taskstate, 73	user.h, 460
ss1	stressfs.c, 377
taskstate, 74	main, 377
ss2	stressfs.d, 378
taskstate, 74	string.c, 378
STA_R	memcmp, 379
asm.h, 81	memcpy, 379
mmu.h, 300	memmove, 379
STA_W	memset, 380
asm.h, 82	safestrcpy, 380
mmu.h, 300	strlen, 380
STA X	strncmp, 381
asm.h, 82	strncpy, 381
mmu.h, 300	string.d, 383
start	strlen
log, 37	defs.h, 153
started	string.c, 380
cpu, 16	ulib.c, 443
startothers	
	user.h, 461
main.c, 272	strncmp
STARTUP	defs.h, 154
lapic.c, 250	string.c, 381
stat, 63	strncpy
dev, 64	defs.h, 154

string.c, 381	sys_exec
STS_IG32	syscall.c, 387
mmu.h, 300	sysfile.c, 407
STS_T32A	SYS_exit
mmu.h, 300	syscall.h, 400
STS_TG32	sys_exit
mmu.h, 301	syscall.c, 388
subdir	sysproc.c, 418
usertests.c, 486	SYS_fork
sum	syscall.h, 400
mp.c, 306	sys_fork
superblock, 65	syscall.c, 388
bmapstart, 65 inodestart, 66	sysproc.c, 419 SYS fstat
	_
logstart, 66 nblocks, 66	syscall.h, 400
ninodes, 66	sys_fstat syscall.c, 388
nlog, 66	sysfile.c, 407
size, 67 SVR	SYS_getpid
	syscall.h, 400
lapic.c, 250 switchkym	sys_getpid
	syscall.c, 388
defs.h, 154	sysproc.c, 419
vm.c, 521	SYS_getprocs
switchuvm	syscall.h, 400
defs.h, 155	sys_getprocs
vm.c, 521	syscall.c, 389
swtch	sysproc.c, 419
defs.h, 155	SYS_halt
symbols	syscall.h, 401
sh.c, 360	sys_halt
SYS_chdir	syscall.c, 389
syscall.h, 399	sysproc.c, 420
sys_chdir	SYS_kill
syscall.c, 386	syscall.h, 401
sysfile.c, 406	sys_kill
SYS_chpr	syscall.c, 389
syscall.h, 399	sysproc.c, 420
sys_chpr	SYS_link
syscall.c, 386	syscall.h, 401
sysproc.c, 418	sys_link
SYS_close	syscall.c, 390
syscall.h, 399 sys close	sysfile.c, 408 SYS_mkdir
<i>,</i> –	
syscall.c, 386	syscall.h, 401
sysfile.c, 406	sys_mkdir
SYS_cps	syscall.c, 390
syscall.h, 399	sysfile.c, 408 SYS_mknod
sys_cps syscall.c, 387	
sysproc.c, 418	syscall.h, 401 sys_mknod
SYS_dup	syscall.c, 391 sysfile.c, 409
syscall.h, 399	
sys_dup	SYS_open
syscall.c, 387	syscall.h, 401
sysfile.c, 407	sys_open
SYS_exec	syscall.c, 391
syscall.h, 400	sysfile.c, 409

SYS_pipe	sys_getpid, 388
syscall.h, 402	sys_getprocs, 389
sys_pipe	sys_halt, 389
syscall.c, 392	sys_kill, 389
sysfile.c, 410	sys_link, 390
SYS read	sys_mkdir, 390
syscall.h, 402	sys_mknod, 391
sys_read	sys_open, 391
syscall.c, 392	sys_pipe, 392
sysfile.c, 410	sys_read, 392
SYS_sbrk	sys_sbrk, <mark>393</mark>
syscall.h, 402	sys_sleep, 393
sys_sbrk	sys_unlink, 393
syscall.c, 393	sys_uptime, 394
sysproc.c, 420	sys_wait, 394
SYS sleep	sys_write, 395
syscall.h, 402	syscall, 395
sys_sleep	syscalls, 395
syscall.c, 393	syscall.d, 398
-	
sysproc.c, 420	syscall.h, 398
SYS_unlink	SYS_chdir, 399
syscall.h, 402	SYS_chpr, 399
sys_unlink	SYS_close, 399
syscall.c, 393	SYS_cps, 399
sysfile.c, 411	SYS_dup, 399
SYS_uptime	SYS_exec, 400
syscall.h, 402	SYS_exit, 400
sys_uptime	SYS fork, 400
syscall.c, 394	SYS fstat, 400
sysproc.c, 421	SYS_getpid, 400
SYS wait	SYS_getprocs, 400
syscall.h, 403	SYS_halt, 401
sys_wait	SYS_kill, 401
syscall.c, 394	SYS_link, 401
sysproc.c, 421	SYS_mkdir, 401
SYS_write	SYS_mknod, 401
syscall.h, 403	SYS_open, 401
sys_write	SYS_pipe, 402
syscall.c, 395	SYS_read, 402
sysfile.c, 411	SYS sbrk, 402
syscall	SYS sleep, 402
defs.h, 155	SYS_unlink, 402
syscall.c, 395	SYS uptime, 402
syscall.c, 383	SYS_wait, 403
argint, 384	SYS write, 403
argptr, 384	syscalls
argstr, 385	syscall.c, 395
fetchint, 385	sysfile.c, 404
fetchstr, 385	argfd, 404
sys_chdir, 386	create, 405
sys_chpr, 386	fdalloc, 405
sys_close, 386	isdirempty, 406
sys_cps, 387	sys_chdir, 406
sys_dup, 387	sys_close, 406
sys_exec, 387	sys_dup, 407
sys_exit, 388	sys_exec, 407
sys_fork, 388	sys_fstat, 407
sys_fstat, 388	sys_link, 408
5,5_10tat, 000	5,5, 100

sys_mkdir, 408	traps.h, 433
sys_mknod, 409	T_MCHK
sys_open, 409	traps.h, 433
sys_pipe, 410	T_NMI
sys_read, 410	traps.h, 433
sys_unlink, 411	T_OFLOW
sys_write, 411	traps.h, 433
sysfile.d, 417	T_PGFLT
sysproc.c, 417	traps.h, 433
lock, 421	T_SEGNP
proc, 422	traps.h, 433
ptable, 422	T_SIMDERR
sys_chpr, 418	traps.h, 434
sys_cps, 418	T_STACK
sys_exit, 418 sys_fork, 419	traps.h, 434 T SYSCALL
sys_getpid, 419	traps.h, 434
sys_getprocs, 419	T TSS
sys_getplocs, 419 sys_halt, 420	traps.h, 434
sys_Hait, 420 sys_kill, 420	taskstate, 67
sys_kiii, 420 sys_sbrk, 420	cr3, 68
sys_sleep, 420	cs, 68
sys_uptime, 421	ds, 68
sys_wait, 421	eax, 68
sysproc.d, 424	ebp, 69
SZ	ebx, 69
proc, 51	ecx, 69
F / -	edi, 69
t	edx, 69
taskstate, 74	eflags, 69
T_ALIGN	eip, 70
traps.h, 431	es, 70
T_BOUND	esi, 70
traps.h, 431	esp, 70
T_BRKPT	esp0, 70
traps.h, 431	esp1, 70
T_DBLFLT	esp2, 71
traps.h, 431	fs, 71
T_DEBUG	gs, 71
traps.h, 431 T DEFAULT	iomb, 71
traps.h, 432	ldt, 71
T DEV	link, 71
stat.h, 376	padding1, 72
T DEVICE	padding10, 72
traps.h, 432	padding2, 72
T DIR	padding3, 72
stat.h, 376	padding4, 72 padding5, 72
T DIVIDE	paddings, 72 padding6, 73
traps.h, 432	padding0, 73
T_FILE	padding8, 73
stat.h, 376	padding9, 73
T_FPERR	ss, 73
traps.h, 432	ss0, 73
T_GPFLT	ss1, 74
traps.h, 432	ss2, 74
T_ILLOP	t, 74
traps.h, 432	TCCR
T_IRQ0	

1.1.054	
lapic.c, 251	trapno
TDCR	trapframe, 79
lapic.c, 251	trapret
tf	proc.c, 333
proc, 52	traps.h, 429
ticks	IRQ_COM1, 430
defs.h, 161	IRQ_ERROR, 430
trap.c, 427	IRQ_IDE, 430
tickslock	IRQ_KBD, 430
defs.h, 161	IRQ SPURIOUS, 430
trap.c, 427	IRQ TIMER, 431
TICR	T_ALIGN, 431
lapic.c, 251	T_BOUND, 431
TIMER	T BRKPT, 431
	-
lapic.c, 251	T_DBLFLT, 431
timerinit	T_DEBUG, 431
defs.h, 156	T_DEFAULT, 432
togglecode	T_DEVICE, 432
kbd.h, 241	T_DIVIDE, 432
TPR	T_FPERR, 432
lapic.c, 251	T_GPFLT, 432
trap	T_ILLOP, 432
trap.c, 425	T_IRQ0, 433
trap.c, 424	T MCHK, 433
idt, 426	T_NMI, 433
idtinit, 425	T_OFLOW, 433
ticks, 427	T PGFLT, 433
tickslock, 427	T_SEGNP, 433
	T SIMDERR, 434
trap, 425	-
tvinit, 426	T_STACK, 434
vectors, 427	T_SYSCALL, 434
trap.d, 429	T_TSS, 434
trapframe, 74	ts
cs, 75	cpu, 16
ds, 75	tvinit
eax, 76	defs.h, 156
ebp, 76	trap.c, 426
ebx, 76	type
ecx, 76	backcmd, 9
edi, 76	cmd, 12
edx, 76	dinode, 18
eflags, 77	elfhdr, 22
eip, 77	execcmd, 23
err, 77	file, 26
es, 77	gatedesc, 28
esi, 77	inode, 32
esp, 78	listcmd, 35
fs, 78	mp, 40
gs, 78	mpioapic, 44
oesp, 78	mpproc, 45
padding1, 78	pipecmd, 48
padding2, 78	proghdr, 54
padding3, 79	redircmd, 55
padding4, 79	segdesc, 61
padding5, 79	stat, 65
padding6, 79	types.h, 435
ss, 79	pde_t, 435
trapno, 79	uchar, 435
Suprior i	3011a1, 100

uint, 436	name, 80
ushort, 436	pid, 80
	ppid, 80
uart	uproc.h, 449
uart.c, 439	uptime
uart.c, 436	user.h, 461
COM1, 437	use_lock
uart, 439	kalloc.c, 231
uartgetc, 437 uartinit, 437	user.h, 450
uartinit, 437	atoi, 450
uartputc, 438	chdir, 451
uart.d, 440	chpr, 451
uartgetc	close, 451
uart.c, 437	cps, 451
uartinit	dup, 452
defs.h, 156	exec, 452
uart.c, 437	exit, 453
uartintr	fork, 454
defs.h, 157	free, 455
uart.c, 438	fstat, 455
uartputc	getpid, 455
defs.h, 157	getprocs, 455
uart.c, 438	gets, 456 halt, 456
uchar	kill, 456
types.h, 435	link, 456
uint	malloc, 457
types.h, 436	memmove, 457
uio	memset, 457
usertests.c, 488	mkdir, 458
ulib.c, 440	mknod, 458
atoi, 440	open, 458
gets, 441	pipe, 458
memmove, 441	printf, 459
memset, 441	read, 459
stat, 442	sbrk, 459
strchr, 442	sleep, 459
strcmp, 442	stat, 459
strcpy, 443	strchr, 460
strlen, 443	strcmp, 460
ulib.d, 445	strcpy, 460
umalloc.c, 445	strlen, 461
Align, 446	unlink, 461
base, 447	uptime, 461
free, 446	wait, 461
freep, 448	write, 462
Header, 446	userinit
malloc, 446	defs.h, 157
morecore, 447	proc.c, 333
umalloc.d, 449	usertests.c, 463
uninit	argptest, 465
usertests.c, 493	BIG, 464
unlink	bigargtest, 465
user.h, 461 unlinkread	bigdir, 466
usertests.c, 489	bigfile, 466
UNUSED	bigwrite, 467
proc.h, 344	bsstest, 468
uproc, 80	buf, 492
uproo, oo	

concreate, 468	vectors
createdelete, 469	trap.c, 427
createtest, 470	VER
dirfile, 471	lapic.c, 251
dirtest, 472	version
echoargv, 492	elfhdr, 22
exectest, 472	mpconf, 42
exitiputtest, 472	mpioapic, 44
exitwait, 473	mpproc, 46
forktest, 473	virt
fourfiles, 474	kmap, 34
fourteen, 475	vm.c, 515
fsfull, 476	allocuvm, 515
iputtest, 476	clearpteu, 516
iref, 477	copyout, 516
linktest, 477	copyuvm, 517
linkunlink, 478	data, 522
main, 479	deallocuvm, 517
mem, 480	freevm, 518
name, 492	inituvm, 518
openiputtest, 480	kmap, 522
opentest, 481	kpgdir, 523
pipe1, 481	kvmalloc, 519
preempt, 482	loaduvm, 519
rand, 483	mappages, 519
randstate, 493	seginit, 520
rmdot, 483	setupkym, 520
RTC_ADDR, 464	switchkym, 521
RTC_DATA, 465	switchuvm, 521
sbrktest, 483	uva2ka, 521
sharedfd, 485 stdout, 493	walkpgdir, 522
subdir, 486	vm.d, 528
uio, 488	W
uninit, 493	console.c, 103
unlink, 493 unlinkread, 489	wait
validateint, 489	defs.h, 158
validatetest, 490	proc.c, 333
writetest, 490	user.h, 461
writetest1, 491	waitdisk
usertests.d, 514	bootmain.c, 90
ushort	wakeup
types.h, 436	defs.h, 159
uva2ka	proc.c, 334
defs.h, 158	wakeup1
vm.c, 521	proc.c, 334
VIII.0, 021	walkpgdir
V2P	vm.c, 522
memlayout.h, 279	wc
V2P_WO	wc.c, 528
memlayout.h, 279	wc.c, 528
vaddr	buf, 529
proghdr, 54	main, 528
valid	wc, 528
inode, 32	wc.d, 530
validateint	whitespace
usertests.c, 489	sh.c, 360
validatetest	winode
usertests.c, 490	mkfs.c, 286

writable file, 26	yield defs.h, 160
write	proc.c, 335
devsw, 17	p. 66.6, 666
user.h, 462	zeroes
write_head	mkfs.c, 289
log.c, 263	ZOMBIE
write_log	proc.h, 344 zombie.c, 538
log.c, 264	main, 538
writei defs.h, 159	zombie.d, 539
fs.c, 197	20
writeopen	
pipe, 47	
writetest	
usertests.c, 490	
writetest1	
usertests.c, 491	
wsect mkfs.c, 286	
11INIS.C, 200	
X	
header, 29	
X1	
lapic.c, 252	
x86.h, 530 cli, 531	
inb, 531	
insl, 531	
lcr3, 531	
lgdt, 532	
lidt, 532	
loadgs, 532	
ltr, 533	
outb, 533 outsl, 533	
outw, 533	
rcr2, 534	
readeflags, 534	
sti, 534	
stosb, 534	
stosl, 535	
xchg, 535 xchecksum	
mpconf, 42	
xchg	
x86.h, 535	
xint	
mkfs.c, 287	
xlength	
mpconf, 43	
xshort mkfs.c, 287	
xv6.dox, 538	
YEAR	
lapic.c, 252	
year	
rtcdate, 57	