

PennOS

1

Generated by Doxygen 1.9.1



<b>1 Data Structure Index</b>	<b>1</b>
1.1 Data Structures	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Data Structure Documentation</b>	<b>5</b>
3.1 directory_entry Struct Reference	5
3.1.1 Field Documentation	5
3.1.1.1 _BUFFER_	5
3.1.1.2 firstBlock	5
3.1.1.3 mtime	6
3.1.1.4 name	6
3.1.1.5 perm	6
3.1.1.6 size	6
3.1.1.7 type	6
3.2 file Struct Reference	6
3.2.1 Field Documentation	7
3.2.1.1 file_id	7
3.2.1.2 filename	7
3.2.1.3 fileptr_head	7
3.2.1.4 next	7
3.2.1.5 wr_pid	7
3.3 fileptr Struct Reference	7
3.3.1 Field Documentation	8
3.3.1.1 next	8
3.3.1.2 pid	8
3.3.1.3 ptr	8
3.4 job Struct Reference	8
3.4.1 Field Documentation	9
3.4.1.1 done	9
3.4.1.2 job_id	9
3.4.1.3 next	9
3.4.1.4 pid	9
3.4.1.5 stop_order	9
3.5 parsed_command Struct Reference	9
3.5.1 Detailed Description	10
3.5.2 Field Documentation	10
3.5.2.1 commands	10
3.5.2.2 is_background	10
3.5.2.3 is_file_append	10
3.5.2.4 num_commands	10
3.5.2.5 stdin_file	10

3.5.2.6 stdout_file	11
3.6 PCB Struct Reference	11
3.6.1 Field Documentation	11
3.6.1.1 children	11
3.6.1.2 context	11
3.6.1.3 fileDescriptors	12
3.6.1.4 name	12
3.6.1.5 next	12
3.6.1.6 numChildren	12
3.6.1.7 parent_pid	12
3.6.1.8 pid	12
3.6.1.9 priority	12
3.6.1.10 status	13
3.7 point Struct Reference	13
3.7.1 Field Documentation	13
3.7.1.1 first	13
3.7.1.2 second	13
<b>4 File Documentation</b>	<b>15</b>
4.1 src/filesystem/filesystem.c File Reference	15
4.1.1 Detailed Description	16
4.1.2 Macro Definition Documentation	16
4.1.2.1 BETWEEN_INCL	17
4.1.2.2 F_CANREAD	17
4.1.2.3 F_CANWRITE	17
4.1.2.4 F_HASPERM	17
4.1.2.5 MIN	17
4.1.3 Function Documentation	17
4.1.3.1 create_file_entry()	17
4.1.3.2 create_fileptr()	18
4.1.3.3 delete_file_entry()	18
4.1.3.4 delete_fileptr()	19
4.1.3.5 f_chmod()	19
4.1.3.6 f_close()	19
4.1.3.7 f_cp()	20
4.1.3.8 f_isatty()	20
4.1.3.9 f_ls()	20
4.1.3.10 f_lseek()	21
4.1.3.11 f_mount()	21
4.1.3.12 f_mv()	22
4.1.3.13 f_open()	22
4.1.3.14 f_print()	22

4.1.3.15 <code>f_read()</code>	23
4.1.3.16 <code>f_rm()</code>	23
4.1.3.17 <code>f_touch()</code>	24
4.1.3.18 <code>f_unlink()</code>	24
4.1.3.19 <code>f_unmount()</code>	24
4.1.3.20 <code>f_write()</code>	25
4.1.3.21 <code>find_file_entry()</code>	25
4.1.3.22 <code>find_file_entry_by_file_id()</code>	26
4.1.3.23 <code>find_file_entry_by_filename()</code>	26
4.1.3.24 <code>find_unused_fd()</code>	26
4.1.3.25 <code>get_fileptr()</code>	26
4.1.3.26 <code>get_fileptr_ptr()</code>	27
4.1.3.27 <code>is_duplicate_fd()</code>	27
4.1.3.28 <code>print_fileptr_pids_all()</code>	27
4.1.3.29 <code>process_create_fileptrs()</code>	28
4.1.3.30 <code>process_delete_fileptrs()</code>	28
4.1.3.31 <code>valid_perm()</code>	28
4.1.4 Variable Documentation	29
4.1.4.1 <code>current_pcb</code>	29
4.1.4.2 <code>next_file_id</code>	29
4.1.4.3 <code>open_files</code>	29
4.2 <code>src/filesystem/filesystem.h</code> File Reference	29
4.2.1 Macro Definition Documentation	31
4.2.1.1 <code>F_APPEND</code>	31
4.2.1.2 <code>F_READ</code>	31
4.2.1.3 <code>F_SEEK_CURR</code>	31
4.2.1.4 <code>F_SEEK_END</code>	31
4.2.1.5 <code>F_SEEK_SET</code>	31
4.2.1.6 <code>F_STDERR</code>	31
4.2.1.7 <code>F_STDIN</code>	31
4.2.1.8 <code>F_STDOUT</code>	32
4.2.1.9 <code>F_WRITE</code>	32
4.2.1.10 <code>FPERM_EXEC</code>	32
4.2.1.11 <code>FPERM_READ</code>	32
4.2.1.12 <code>FPERM_WRIT</code>	32
4.2.2 Typedef Documentation	32
4.2.2.1 <code>file_t</code>	32
4.2.2.2 <code>fileptr_t</code>	32
4.2.3 Function Documentation	32
4.2.3.1 <code>f_chmod()</code>	32
4.2.3.2 <code>f_close()</code>	33
4.2.3.3 <code>f_cp()</code>	33

4.2.3.4 <a href="#">f_ls()</a> . . . . .	34
4.2.3.5 <a href="#">f_lseek()</a> . . . . .	34
4.2.3.6 <a href="#">f_mount()</a> . . . . .	35
4.2.3.7 <a href="#">f_mv()</a> . . . . .	35
4.2.3.8 <a href="#">f_open()</a> . . . . .	36
4.2.3.9 <a href="#">f_print()</a> . . . . .	37
4.2.3.10 <a href="#">f_read()</a> . . . . .	38
4.2.3.11 <a href="#">f_rm()</a> . . . . .	39
4.2.3.12 <a href="#">f_touch()</a> . . . . .	39
4.2.3.13 <a href="#">f_unlink()</a> . . . . .	40
4.2.3.14 <a href="#">f_unmount()</a> . . . . .	40
4.2.3.15 <a href="#">f_write()</a> . . . . .	41
4.2.3.16 <a href="#">find_file_entry_by_file_id()</a> . . . . .	41
4.2.3.17 <a href="#">print_fileptr_pids_all()</a> . . . . .	41
4.2.3.18 <a href="#">process_create_fileptrs()</a> . . . . .	42
4.2.3.19 <a href="#">process_delete_fileptrs()</a> . . . . .	42
4.3 <a href="#">src/kernel/kernel-functions.c</a> File Reference . . . . .	42
4.3.1 Function Documentation . . . . .	43
4.3.1.1 <a href="#">k_process_cleanup()</a> . . . . .	43
4.3.1.2 <a href="#">k_process_create()</a> . . . . .	43
4.3.1.3 <a href="#">k_process_deep_cleanup()</a> . . . . .	43
4.3.1.4 <a href="#">k_process_kill()</a> . . . . .	44
4.4 <a href="#">src/kernel/kernel-functions.h</a> File Reference . . . . .	44
4.4.1 Function Documentation . . . . .	44
4.4.1.1 <a href="#">k_process_cleanup()</a> . . . . .	44
4.4.1.2 <a href="#">k_process_create()</a> . . . . .	45
4.4.1.3 <a href="#">k_process_deep_cleanup()</a> . . . . .	45
4.4.1.4 <a href="#">k_process_kill()</a> . . . . .	45
4.5 <a href="#">src/kernel/PCB.c</a> File Reference . . . . .	46
4.6 <a href="#">src/kernel/PCB.h</a> File Reference . . . . .	46
4.6.1 Macro Definition Documentation . . . . .	47
4.6.1.1 <a href="#">MAX_FDS</a> . . . . .	47
4.6.1.2 <a href="#">NOFILE</a> . . . . .	47
4.6.1.3 <a href="#">STACKSIZE</a> . . . . .	47
4.6.1.4 <a href="#">STDERR_ID</a> . . . . .	47
4.6.1.5 <a href="#">STDIN_ID</a> . . . . .	48
4.6.1.6 <a href="#">STDOUT_ID</a> . . . . .	48
4.6.2 Typedef Documentation . . . . .	48
4.6.2.1 <a href="#">PCB</a> . . . . .	48
4.6.3 Function Documentation . . . . .	48
4.6.3.1 <a href="#">addPCBToList()</a> . . . . .	48
4.6.3.2 <a href="#">count_running()</a> . . . . .	49

4.6.3.3 count_running_priority()	49
4.6.3.4 createPCB()	49
4.6.3.5 findPCBByContext()	50
4.6.3.6 findPCBByPID()	51
4.6.3.7 getLength()	51
4.6.3.8 k_free()	51
4.6.3.9 removePCBFromList()	52
4.6.4 Variable Documentation	53
4.6.4.1 next_pid	53
4.6.4.2 pcb_list	53
4.7 src/kernel/puser-functions.c File Reference	53
4.7.1 Function Documentation	54
4.7.1.1 p_exit()	54
4.7.1.2 p_kill()	54
4.7.1.3 p_nice()	54
4.7.1.4 p_sleep()	55
4.7.1.5 p_spawn()	55
4.7.1.6 p_waitpid()	55
4.7.1.7 W_WIFCONTINUED()	56
4.7.1.8 W_WIFEXITED()	56
4.7.1.9 W_WIFSIGNALED()	56
4.7.1.10 W_WIFSTOPPED()	57
4.7.2 Variable Documentation	57
4.7.2.1 current_pcb	57
4.7.2.2 ticks	57
4.8 src/kernel/puser-functions.h File Reference	57
4.8.1 Function Documentation	58
4.8.1.1 p_exit()	58
4.8.1.2 p_kill()	58
4.8.1.3 p_nice()	58
4.8.1.4 p_sleep()	59
4.8.1.5 p_spawn()	59
4.8.1.6 p_waitpid()	60
4.8.1.7 W_WIFCONTINUED()	60
4.8.1.8 W_WIFEXITED()	60
4.8.1.9 W_WIFSIGNALED()	61
4.8.1.10 W_WIFSTOPPED()	61
4.8.2 Variable Documentation	61
4.8.2.1 current_pcb	61
4.8.2.2 ticks	61
4.9 src/kernel/scheduler.c File Reference	61
4.9.1 Function Documentation	62

4.9.1.1 alarmHandler()	62
4.9.1.2 freeStacks()	62
4.9.1.3 reaper()	63
4.9.1.4 scheduler()	63
4.9.1.5 setAlarmHandler()	63
4.9.1.6 setTimer()	63
4.9.1.7 start_scheduler()	64
4.9.2 Variable Documentation	64
4.9.2.1 activeContext	64
4.9.2.2 centisecond	64
4.9.2.3 mainContext	64
4.9.2.4 reaperContext	64
4.9.2.5 schedulerContext	64
4.10 src/kernel/scheduler.h File Reference	65
4.10.1 Function Documentation	65
4.10.1.1 init_scheduler()	65
4.10.1.2 setAlarmHandler()	65
4.10.1.3 setTimer()	65
4.10.1.4 start_scheduler()	66
4.10.2 Variable Documentation	66
4.10.2.1 reaperContext	66
4.10.2.2 schedulerContext	66
4.11 src/logger/logger.c File Reference	66
4.11.1 Function Documentation	67
4.11.1.1 log_blocked_event()	67
4.11.1.2 log_continued_event()	67
4.11.1.3 log_create_event()	68
4.11.1.4 log_exited_event()	68
4.11.1.5 log_nice_event()	68
4.11.1.6 log_orphan_event()	69
4.11.1.7 log_schedule_event()	69
4.11.1.8 log_signaled_event()	70
4.11.1.9 log_stopped_event()	70
4.11.1.10 log_unblocked_event()	70
4.11.1.11 log_waited_event()	71
4.11.1.12 log_zombie_event()	71
4.11.2 Variable Documentation	71
4.11.2.1 logfile	71
4.12 src/logger/logger.h File Reference	71
4.12.1 Function Documentation	72
4.12.1.1 log_blocked_event()	72
4.12.1.2 log_continued_event()	73



4.12.1.3 log_create_event()	73
4.12.1.4 log_exited_event()	73
4.12.1.5 log_nice_event()	74
4.12.1.6 log_orphan_event()	74
4.12.1.7 log_schedule_event()	74
4.12.1.8 log_signaled_event()	75
4.12.1.9 log_stopped_event()	75
4.12.1.10 log_unblocked_event()	75
4.12.1.11 log_waited_event()	77
4.12.1.12 log_zombie_event()	77
4.12.2 Variable Documentation	77
4.12.2.1 logfile	77
4.13 src/pennfat/fat.c File Reference	78
4.13.1 Function Documentation	79
4.13.1.1 add_file()	79
4.13.1.2 build_chain()	79
4.13.1.3 delete_chain()	80
4.13.1.4 fill_chain()	80
4.13.1.5 find_file()	81
4.13.1.6 fs_cat()	81
4.13.1.7 fs_chmod()	82
4.13.1.8 fs_cp()	82
4.13.1.9 fs_cp_mode()	83
4.13.1.10 fs_getmeta()	83
4.13.1.11 fs_ls()	84
4.13.1.12 fs_ls_single()	84
4.13.1.13 fs_mark_deleted()	84
4.13.1.14 fs_mount()	85
4.13.1.15 fs_mv()	85
4.13.1.16 fs_rm()	86
4.13.1.17 fs_touch()	86
4.13.1.18 fs_unmount()	86
4.13.1.19 get_free_block()	87
4.13.1.20 mem_idx()	87
4.13.1.21 read_chain()	88
4.13.1.22 valid_filename()	88
4.13.1.23 write_file()	88
4.13.2 Variable Documentation	89
4.13.2.1 BITS_PER_BYTE	89
4.13.2.2 BYTE_SIZE	89
4.13.2.3 DEFAULT_PERMISSIONS	89
4.13.2.4 DIR_ENTRY_SIZE	89

4.13.2.5 FILENAME_DEL_INUSE . . . . .	89
4.13.2.6 FILENAME_DEL_UNUSED . . . . .	90
4.13.2.7 FILENAME_ENDDIR . . . . .	90
4.13.2.8 FILEPERM_EX . . . . .	90
4.13.2.9 FILEPERM_NONE . . . . .	90
4.13.2.10 FILEPERM_RD . . . . .	90
4.13.2.11 FILEPERM_WR . . . . .	90
4.13.2.12 FILETYPE_DIRECTORY . . . . .	90
4.13.2.13 FILETYPE_FILE . . . . .	90
4.13.2.14 FILETYPE_LINK . . . . .	91
4.13.2.15 FILETYPE_UNKNOWN . . . . .	91
4.13.2.16 LASTBLOCK . . . . .	91
4.13.2.17 ROOTDIR . . . . .	91
4.14 src/pennfat/fat.h File Reference . . . . .	91
4.14.1 Macro Definition Documentation . . . . .	92
4.14.1.1 BLOCK_SIZE . . . . .	92
4.14.1.2 FAT_BLOCKS . . . . .	93
4.14.2 Typedef Documentation . . . . .	93
4.14.2.1 dir_entry_t . . . . .	93
4.14.2.2 point_t . . . . .	93
4.14.3 Function Documentation . . . . .	93
4.14.3.1 find_file() . . . . .	93
4.14.3.2 fs_cat() . . . . .	94
4.14.3.3 fs_chmod() . . . . .	95
4.14.3.4 fs_cp() . . . . .	96
4.14.3.5 fs_cp_mode() . . . . .	96
4.14.3.6 fs_getmeta() . . . . .	98
4.14.3.7 fs_ls() . . . . .	99
4.14.3.8 fs_ls_single() . . . . .	100
4.14.3.9 fs_mark_deleted() . . . . .	100
4.14.3.10 fs_mount() . . . . .	101
4.14.3.11 fs_mv() . . . . .	101
4.14.3.12 fs_rm() . . . . .	102
4.14.3.13 fs_touch() . . . . .	103
4.14.3.14 fs_unmount() . . . . .	103
4.14.3.15 read_chain() . . . . .	104
4.14.3.16 valid_filename() . . . . .	105
4.14.4 Variable Documentation . . . . .	105
4.14.4.1 BITS_PER_BYTE . . . . .	105
4.14.4.2 BYTE_SIZE . . . . .	106
4.14.4.3 DEFAULT_PERMISSIONS . . . . .	106
4.14.4.4 DIR_ENTRY_SIZE . . . . .	106

4.14.4.5	FILENAME_DEL_INUSE	106
4.14.4.6	FILENAME_DEL_UNUSED	106
4.14.4.7	FILENAME_ENDDIR	106
4.14.4.8	FILEPERM_EX	106
4.14.4.9	FILEPERM_NONE	106
4.14.4.10	FILEPERM_RD	107
4.14.4.11	FILEPERM_WR	107
4.14.4.12	FILETYPE_DIRECTORY	107
4.14.4.13	FILETYPE_FILE	107
4.14.4.14	FILETYPE_LINK	107
4.14.4.15	FILETYPE_UNKNOWN	107
4.14.4.16	LASTBLOCK	107
4.14.4.17	ROOTDIR	107
4.15	src/pennfat/pennfat.c File Reference	108
4.15.1	Macro Definition Documentation	108
4.15.1.1	CONTINUE	108
4.15.2	Function Documentation	108
4.15.2.1	all_files_exist()	109
4.15.2.2	correct_argc()	109
4.15.2.3	main()	109
4.15.2.4	valid_fs_mounted()	109
4.16	src/pennfat/safe.c File Reference	109
4.16.1	Function Documentation	110
4.16.1.1	safe_close()	110
4.16.1.2	safe_lseek()	110
4.16.1.3	safe_mmap()	111
4.16.1.4	safe_msync()	111
4.16.1.5	safe_munmap()	112
4.16.1.6	safe_open()	112
4.16.1.7	safe_read()	112
4.16.1.8	safe_write()	113
4.17	src/pennfat/safe.h File Reference	113
4.17.1	Function Documentation	114
4.17.1.1	safe_close()	114
4.17.1.2	safe_lseek()	114
4.17.1.3	safe_mmap()	114
4.17.1.4	safe_msync()	115
4.17.1.5	safe_munmap()	115
4.17.1.6	safe_open()	116
4.17.1.7	safe_read()	116
4.17.1.8	safe_write()	117
4.18	src/pennos.c File Reference	117

4.18.1 Function Documentation	117
4.18.1.1 main()	117
4.19 src/shell/job-list.c File Reference	118
4.19.1 Function Documentation	118
4.19.1.1 job_find_by_jobid()	118
4.19.1.2 job_get_last()	119
4.19.1.3 job_print()	119
4.19.1.4 jobs_insert()	119
4.19.1.5 jobs_push()	120
4.19.1.6 jobs_remove()	120
4.20 src/shell/job-list.h File Reference	120
4.20.1 Macro Definition Documentation	121
4.20.1.1 NOT_STOPPED	121
4.20.2 Typedef Documentation	121
4.20.2.1 job_t	121
4.20.3 Function Documentation	121
4.20.3.1 job_find_by_jobid()	121
4.20.3.2 job_get_last()	122
4.20.3.3 job_print()	123
4.20.3.4 jobs_insert()	123
4.20.3.5 jobs_push()	124
4.20.3.6 jobs_remove()	124
4.21 src/shell/pennos-shell.c File Reference	125
4.21.1 Macro Definition Documentation	126
4.21.1.1 CONTINUE	126
4.21.1.2 PROMPT	126
4.21.2 Function Documentation	126
4.21.2.1 cull_background()	127
4.21.2.2 cull_helper()	127
4.21.2.3 debug_print_jobs()	127
4.21.2.4 empty_reaped()	127
4.21.2.5 execute_command()	127
4.21.2.6 execute_script()	128
4.21.2.7 orphan_child()	128
4.21.2.8 pennos_shell()	128
4.21.2.9 shell_busy()	128
4.21.2.10 shell_cat()	128
4.21.2.11 shell_chmod()	129
4.21.2.12 shell_cp()	129
4.21.2.13 shell_echo()	129
4.21.2.14 shell_kill()	129
4.21.2.15 shell_ls()	129

4.21.2.16 shell_mv()	129
4.21.2.17 shell_orphanify()	130
4.21.2.18 shell_ps()	130
4.21.2.19 shell_rm()	130
4.21.2.20 shell_sleep()	130
4.21.2.21 shell_touch()	130
4.21.2.22 shell_zombify()	130
4.21.2.23 spawn_command()	131
4.21.2.24 stop_handler()	131
4.21.2.25 term_handler()	131
4.21.2.26 zombie_child()	131
4.21.3 Variable Documentation	131
4.21.3.1 background	131
4.21.3.2 current_jobid	131
4.21.3.3 foreground_job	132
4.21.3.4 head	132
4.21.3.5 jobid_ctr	132
4.21.3.6 MAN_COMMANDS	132
4.21.3.7 MAX_ARGUMENTS	132
4.21.3.8 n_reaped	133
4.21.3.9 reaped	133
4.21.3.10 stop_order	133
4.21.3.11 stop_trigger	133
4.22 src/shell/pennos-shell.h File Reference	133
4.22.1 Function Documentation	133
4.22.1.1 pennos_shell()	133
4.23 src/util/globals.c File Reference	134
4.23.1 Variable Documentation	134
4.23.1.1 fat	134
4.23.1.2 fs_fd	134
4.24 src/util/globals.h File Reference	134
4.24.1 Macro Definition Documentation	135
4.24.1.1 S_SIGCHLD	135
4.24.1.2 S_SIGCONT	135
4.24.1.3 S_SIGSTOP	135
4.24.1.4 S_SIGTERM	135
4.24.1.5 T_BLOCKED	136
4.24.1.6 T_RUNNING	136
4.24.1.7 T_STOPPED	136
4.24.1.8 T_ZOMBIED	136
4.24.2 Variable Documentation	136
4.24.2.1 fat	136

4.24.2.2 fs_fd	136
4.25 src/util/p-errno.c File Reference	136
4.25.1 Function Documentation	137
4.25.1.1 err_string()	137
4.25.1.2 p_perror()	137
4.25.2 Variable Documentation	137
4.25.2.1 ERRNO	138
4.26 src/util/p-errno.h File Reference	138
4.26.1 Macro Definition Documentation	138
4.26.1.1 ERR_F_CLOSE_TERMINAL	138
4.26.1.2 ERR_F_LSEEK_OOB	138
4.26.1.3 ERR_F_LSEEK_TERMINAL	139
4.26.1.4 ERR_F_OPEN_CREATE_READ	139
4.26.1.5 ERR_F_OPEN_INVALID_MODE	139
4.26.1.6 ERR_F_OPEN_INVALID_PERMS	139
4.26.1.7 ERR_F_OPEN_WRITE_INUSE	139
4.26.1.8 ERR_F_READ_TERM_OUT	139
4.26.1.9 ERR_F_UNLINK_NOT_FOUND	139
4.26.1.10 ERR_F_WRITE_RDONLY	139
4.26.1.11 ERR_F_WRITE_TERM_IN	140
4.26.1.12 ERR_FS_FILE_NOT_FOUND	140
4.26.1.13 ERR_NONE	140
4.26.1.14 ERR_P_KILL_NULL_PROCESS	140
4.26.1.15 ERR_P_NICE_NULL_PROCESS	140
4.26.1.16 ERR_P_SPAWN_NULL_CHILD	140
4.26.1.17 ERR_P_SPAWN_NULL_STACK	140
4.26.1.18 ERR_P_WAITPID_NULL_CHILD	140
4.26.2 Function Documentation	140
4.26.2.1 p_perror()	140
4.27 src/util/parser.h File Reference	141
4.27.1 Macro Definition Documentation	141
4.27.1.1 EXPECT_COMMANDS	142
4.27.1.2 EXPECT_INPUT_FILENAME	142
4.27.1.3 EXPECT_OUTPUT_FILENAME	142
4.27.1.4 UNEXPECTED_AMPERSAND	142
4.27.1.5 UNEXPECTED_FILE_INPUT	142
4.27.1.6 UNEXPECTED_FILE_OUTPUT	142
4.27.1.7 UNEXPECTED_PIPELINE	142
4.27.2 Function Documentation	142
4.27.2.1 parse_command()	143
4.27.2.2 print_parsed_command()	143
4.28 src/util/safe-user.c File Reference	143

4.28.1 Function Documentation	143
4.28.1.1 safe_f_close()	144
4.28.1.2 safe_f_lseek()	144
4.28.1.3 safe_f_open()	144
4.28.1.4 safe_f_print()	144
4.28.1.5 safe_f_read()	144
4.28.1.6 safe_f_unlink()	144
4.28.1.7 safe_f_write()	145
4.29 src/util/safe-user.h File Reference	145
4.29.1 Function Documentation	145
4.29.1.1 safe_f_close()	145
4.29.1.2 safe_f_lseek()	145
4.29.1.3 safe_f_open()	145
4.29.1.4 safe_f_print()	146
4.29.1.5 safe_f_read()	146
4.29.1.6 safe_f_unlink()	146
4.29.1.7 safe_f_write()	146
4.30 src/util/util.c File Reference	146
4.30.1 Function Documentation	147
4.30.1.1 get_argc()	147
4.30.1.2 safe_malloc()	147
4.30.1.3 safe_signal()	147
4.30.2 Variable Documentation	148
4.30.2.1 ERRBUFFER_SIZE	148
4.30.2.2 IOBUFFER_SIZE	148
4.31 src/util/util.h File Reference	148
4.31.1 Macro Definition Documentation	148
4.31.1.1 PRINT	149
4.31.1.2 PRINTE	149
4.31.2 Function Documentation	149
4.31.2.1 get_argc()	149
4.31.2.2 safe_malloc()	149
4.31.2.3 safe_signal()	150
4.31.3 Variable Documentation	150
4.31.3.1 ERRBUFFER_SIZE	150
4.31.3.2 IOBUFFER_SIZE	150





# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">directory_entry</a>	5
<a href="#">file</a>	6
<a href="#">fileptr</a>	7
<a href="#">job</a>	8
<a href="#">parsed_command</a>	9
<a href="#">PCB</a>	11
<a href="#">point</a>	13



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

src/ <a href="#">pennos.c</a> . . . . .	117
src/filesystem/ <a href="#">filesystem.c</a>	
This is a brief description of the contents of the file . . . . .	15
src/filesystem/ <a href="#">filesystem.h</a> . . . . .	29
src/kernel/ <a href="#">kernel-functions.c</a> . . . . .	42
src/kernel/ <a href="#">kernel-functions.h</a> . . . . .	44
src/kernel/ <a href="#">PCB.c</a> . . . . .	46
src/kernel/ <a href="#">PCB.h</a> . . . . .	46
src/kernel/ <a href="#">puser-functions.c</a> . . . . .	53
src/kernel/ <a href="#">puser-functions.h</a> . . . . .	57
src/kernel/ <a href="#">scheduler.c</a> . . . . .	61
src/kernel/ <a href="#">scheduler.h</a> . . . . .	65
src/logger/ <a href="#">logger.c</a> . . . . .	66
src/logger/ <a href="#">logger.h</a> . . . . .	71
src/pennfat/ <a href="#">fat.c</a> . . . . .	78
src/pennfat/ <a href="#">fat.h</a> . . . . .	91
src/pennfat/ <a href="#">pennfat.c</a> . . . . .	108
src/pennfat/ <a href="#">safe.c</a> . . . . .	109
src/pennfat/ <a href="#">safe.h</a> . . . . .	113
src/shell/ <a href="#">job-list.c</a> . . . . .	118
src/shell/ <a href="#">job-list.h</a> . . . . .	120
src/shell/ <a href="#">pennos-shell.c</a> . . . . .	125
src/shell/ <a href="#">pennos-shell.h</a> . . . . .	133
src/util/ <a href="#">globals.c</a> . . . . .	134
src/util/ <a href="#">globals.h</a> . . . . .	134
src/util/ <a href="#">p-errno.c</a> . . . . .	136
src/util/ <a href="#">p-errno.h</a> . . . . .	138
src/util/ <a href="#">parser.h</a> . . . . .	141
src/util/ <a href="#">safe-user.c</a> . . . . .	143
src/util/ <a href="#">safe-user.h</a> . . . . .	145
src/util/ <a href="#">util.c</a> . . . . .	146
src/util/ <a href="#">util.h</a> . . . . .	148



## Chapter 3

# Data Structure Documentation

### 3.1 directory\_entry Struct Reference

```
#include <fat.h>
```

#### Data Fields

- char [name](#) [32]
- uint32\_t [size](#)
- uint16\_t [firstBlock](#)
- uint8\_t [type](#)
- uint8\_t [perm](#)
- time\_t [mtime](#)
- char [\\_BUFFER\\_](#) [16]

#### 3.1.1 Field Documentation

##### 3.1.1.1 [\\_BUFFER\\_](#)

```
char directory_entry::_BUFFER_[16]
```

##### 3.1.1.2 [firstBlock](#)

```
uint16_t directory_entry::firstBlock
```

### 3.1.1.3 mtime

```
time_t directory_entry::mtime
```

### 3.1.1.4 name

```
char directory_entry::name[32]
```

### 3.1.1.5 perm

```
uint8_t directory_entry::perm
```

### 3.1.1.6 size

```
uint32_t directory_entry::size
```

### 3.1.1.7 type

```
uint8_t directory_entry::type
```

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

- [src/pennfat/fat.h](#)

## 3.2 file Struct Reference

```
#include <filesystem.h>
```

Collaboration diagram for file:

### Data Fields

- [char filename](#) [32]
- [int file\\_id](#)
- [int wr\\_pid](#)
- [struct fileptr \\* fileptr\\_head](#)
- [struct file \\* next](#)

### 3.2.1 Field Documentation

#### 3.2.1.1 file\_id

```
int file::file_id
```

#### 3.2.1.2 filename

```
char file::filename[32]
```

#### 3.2.1.3 fileptr\_head

```
struct fileptr* file::fileptr_head
```

#### 3.2.1.4 next

```
struct file* file::next
```

#### 3.2.1.5 wr\_pid

```
int file::wr_pid
```

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

- [src/filesystem/filesystem.h](#)

## 3.3 fileptr Struct Reference

```
#include <filesystem.h>
```

Collaboration diagram for fileptr:

## Data Fields

- int [pid](#)
- int [ptr](#)
- [fileptr\\_t](#) \* [next](#)

### 3.3.1 Field Documentation

#### 3.3.1.1 next

```
fileptr\_t* fileptr::next
```

#### 3.3.1.2 pid

```
int fileptr::pid
```

#### 3.3.1.3 ptr

```
int fileptr::ptr
```

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

- [src/filesystem/filesystem.h](#)

## 3.4 job Struct Reference

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

Collaboration diagram for job:

## Data Fields

- int [job\\_id](#)
- int [pid](#)
- int [stop\\_order](#)
- bool [done](#)
- [job\\_t](#) \* [next](#)



### 3.4.1 Field Documentation

#### 3.4.1.1 done

```
bool job::done
```

#### 3.4.1.2 job\_id

```
int job::job_id
```

#### 3.4.1.3 next

```
job_t* job::next
```

#### 3.4.1.4 pid

```
int job::pid
```

#### 3.4.1.5 stop\_order

```
int job::stop_order
```

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

- [src/shell/job-list.h](#)

## 3.5 parsed\_command Struct Reference

```
#include <parser.h>
```

## Data Fields

- bool [is\\_background](#)
- bool [is\\_file\\_append](#)
- const char \* [stdin\\_file](#)
- const char \* [stdout\\_file](#)
- size\_t [num\\_commands](#)
- char \*\* [commands](#) []

### 3.5.1 Detailed Description

struct [parsed\\_command](#) stored all necessary information needed for penn-shell.

### 3.5.2 Field Documentation

#### 3.5.2.1 commands

```
char** parsed_command::commands[ ]
```

#### 3.5.2.2 is\_background

```
bool parsed_command::is_background
```

#### 3.5.2.3 is\_file\_append

```
bool parsed_command::is_file_append
```

#### 3.5.2.4 num\_commands

```
size_t parsed_command::num_commands
```

#### 3.5.2.5 stdin\_file

```
const char* parsed_command::stdin_file
```

### 3.5.2.6 stdout\_file

```
const char* parsed_command::stdout_file
```

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

- [src/util/parser.h](#)

## 3.6 PCB Struct Reference

```
#include <PCB.h>
```

Collaboration diagram for PCB:

### Data Fields

- char \* [name](#)
- ucontext\_t \* [context](#)
- pid\_t [parent\\_pid](#)
- pid\_t [pid](#)
- pid\_t [children](#) [10000]
- int [numChildren](#)
- int [fileDescriptors](#) [MAX\_FDS]
- int [priority](#)
- int [status](#)
- struct PCB \* [next](#)

### 3.6.1 Field Documentation

#### 3.6.1.1 children

```
pid_t PCB::children[10000]
```

#### 3.6.1.2 context

```
ucontext_t* PCB::context
```

### 3.6.1.3 fileDescriptors

```
int PCB::fileDescriptors[MAX_FDS]
```

### 3.6.1.4 name

```
char* PCB::name
```

### 3.6.1.5 next

```
struct PCB* PCB::next
```

### 3.6.1.6 numChildren

```
int PCB::numChildren
```

### 3.6.1.7 parent\_pid

```
pid_t PCB::parent_pid
```

### 3.6.1.8 pid

```
pid_t PCB::pid
```

### 3.6.1.9 priority

```
int PCB::priority
```

### 3.6.1.10 status

```
int PCB::status
```

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

- [src/kernel/PCB.h](#)

## 3.7 point Struct Reference

```
#include <fat.h>
```

### Data Fields

- int [first](#)
- int [second](#)

### 3.7.1 Field Documentation

#### 3.7.1.1 first

```
int point::first
```

#### 3.7.1.2 second

```
int point::second
```

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

- [src/pennfat/fat.h](#)



## Chapter 4

# File Documentation

### 4.1 src/filesystem/filesystem.c File Reference

This is a brief description of the contents of the file.

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include "filesystem.h"
#include "../util/globals.h"
#include "../util/util.h"
#include "../util/p-errno.h"
#include "../kernel/PCB.h"
#include "../pennfat/fat.h"
#include "../pennfat/safe.h"
```

Include dependency graph for filesystem.c:

#### Macros

- #define [MIN](#)(a, b) (((a) < (b)) ? (a) : (b))
- #define [BETWEEN\\_INCL](#)(value, lower, upper) ((value) >= (lower) && (value) <= (upper))
- #define [F\\_HASPERM](#)(perm, mask) (((perm) & (mask)) != 0)
- #define [F\\_CANREAD](#)(perm) ([F\\_HASPERM](#)(perm, [FILEPERM\\_RD](#)))
- #define [F\\_CANWRITE](#)(perm) (([F\\_HASPERM](#)(perm, [FILEPERM\\_WR](#))) && ([F\\_CANREAD](#)(perm)))

#### Functions

- void [create\\_fileptr](#) ([fileptr\\_t](#) \*\*fileptr\_head, int pid, int ptr)  
*create & insert a file pointer into the fileptr list*
- void [delete\\_fileptr](#) ([fileptr\\_t](#) \*\*fileptr\_head, int pid)
- [fileptr\\_t](#) \* [get\\_fileptr](#) ([fileptr\\_t](#) \*fileptr\_head, int pid)
- int [get\\_fileptr\\_ptr](#) ([fileptr\\_t](#) \*fileptr\_head, int pid)
- int [create\\_file\\_entry](#) (const char \*filename, int mode, [dir\\_entry\\_t](#) \*dir\_entry)
- void [delete\\_file\\_entry](#) (int file\_id)
- bool [valid\\_perm](#) (uint8\_t perm, int mode)
- [file\\_t](#) \* [find\\_file\\_entry\\_by\\_file\\_id](#) (int file\_id)
- [file\\_t](#) \* [find\\_file\\_entry\\_by\\_filename](#) (const char \*filename)

- bool `find_file_entry` (int fd, int \*file\_id\_ptr, `file_t` \*file\_entry)
- void `process_create_fileptrs` (`PCB` \*pcb)
- void `process_delete_fileptrs` (`PCB` \*pcb)
- void `print_fileptr_pids_all` ()
- int `find_unused_fd` (`PCB` \*pcb)
- bool `is_duplicate_fd` (`PCB` \*pcb, int file\_id)
- bool `f_isatty` (int fd)
- int `f_open` (const char \*fname, int mode)  
*Opens or creates a file and returns a file descriptor.*
- int `f_read` (int fd, int n, char \*buf)  
*Reads data from a file descriptor.*
- int `f_write` (int fd, const char \*str, int n)  
*Writes data to a file descriptor.*
- int `f_close` (int fd)  
*Closes a file descriptor.*
- int `f_unlink` (const char \*fname)  
*Unlinks (deletes) a file.*
- int `f_seek` (int fd, int offset, int whence)  
*Moves the file pointer to a specified position within a file.*
- void `f_ls` (const char \*filename)  
*Lists information about files in the file system.*
- void `f_touch` (char \*filenames[], int n)
- int `f_print` (const char \*str)  
*Prints a string to the standard error (stderr).*
- int `f_mount` (char \*fs\_name, uint16\_t \*\*fat)  
*Mounts a file system.*
- void `f_unmount` (uint16\_t \*\*fat, int fs\_fd)  
*Unmounts a file system.*
- void `f_mv` (char \*src, char \*dest)  
*Moves or renames a file or directory.*
- void `f_cp` (char \*src, char \*dest)  
*Copies a file or directory.*
- void `f_rm` (char \*filenames[], int n)  
*Removes (deletes) files or directories.*
- void `f_chmod` (char \*filename, int perms)  
*Changes the permissions of a file or directory.*

## Variables

- int `next_file_id` = 0
- `file_t` \* `open_files` = NULL
- `PCB` \* `current_pcb`

### 4.1.1 Detailed Description

This is a brief description of the contents of the file.

### 4.1.2 Macro Definition Documentation



#### 4.1.2.1 BETWEEN\_INCL

```
#define BETWEEN_INCL(  
    value,  
    lower,  
    upper ) ((value) >= (lower) && (value) <= (upper))
```

#### 4.1.2.2 F\_CANREAD

```
#define F_CANREAD(  
    perm ) (F_HASPERM(perm, FILEPERM_RD))
```

#### 4.1.2.3 F\_CANWRITE

```
#define F_CANWRITE(  
    perm ) ((F_HASPERM(perm, FILEPERM_WR)) && (F_CANREAD(perm)))
```

#### 4.1.2.4 F\_HASPERM

```
#define F_HASPERM(  
    perm,  
    mask ) (((perm) & (mask)) != 0)
```

#### 4.1.2.5 MIN

```
#define MIN(  
    a,  
    b ) ((a) < (b)) ? (a) : (b)
```

### 4.1.3 Function Documentation

#### 4.1.3.1 create\_file\_entry()

```
int create_file_entry (  
    const char * filename,  
    int mode,  
    dir_entry_t * dir_entry )
```

create & insert a file entry into open\_files

**Parameters**

<i>filename</i>	the file name
<i>mode</i>	the open mode
<i>dir_entry</i>	the file's fat directory entry

**Returns**

the *file\_id*

**4.1.3.2 create\_fileptr()**

```
void create_fileptr (
    fileptr_t ** fileptr_head,
    int pid,
    int ptr )
```

create & insert a file pointer into the fileptr list

**Parameters**

<i>fileptr_head</i>	the head of the fileptr list
<i>pid</i>	the process id for the fileptr
<i>ptr</i>	the pointer/offset

**Returns**

none

**4.1.3.3 delete\_file\_entry()**

```
void delete_file_entry (
    int file_id )
```

delete the specified file entry from *open\_files*; call when *file\_entry.fileptr\_head* == NULL

**Parameters**

<i>file_id</i>	the file id
----------------	-------------

**Returns**

none

#### 4.1.3.4 delete\_fileptr()

```
void delete_fileptr (
    fileptr_t ** fileptr_head,
    int pid )
```

delete the specified file pointer from the fileptr list

##### Parameters

<i>fileptr_head</i>	the head of the fileptr list
<i>pid</i>	the process id whose fileptr should be deleted

##### Returns

none

#### 4.1.3.5 f\_chmod()

```
void f_chmod (
    char * filename,
    int perms )
```

Changes the permissions of a file or directory.

##### Parameters

<i>filename</i>	The name of the file or directory.
<i>perms</i>	The new permissions to set for the file or directory.

#### 4.1.3.6 f\_close()

```
int f_close (
    int fd )
```

Closes a file descriptor.

This function closes the specified file descriptor, releasing associated resources. If the file descriptor represents a terminal, an error is returned. The function then finds the corresponding file entry in open files, updates the process's file descriptor table, and deallocates file pointers if this is the last instance of the file descriptor for the process. If the process had write access, it is revoked when the last instance is closed. If no process is using the file, the file entry is deleted.

**Parameters**

<i>fd</i>	The file descriptor to close.
-----------	-------------------------------

**Returns**

On success, returns 0. On failure, returns -1, and the global variable `ERRNO` is set accordingly.

**4.1.3.7 f\_cp()**

```
void f_cp (
    char * src,
    char * dest )
```

Copies a file or directory.

**Parameters**

<i>src</i>	The source path of the file or directory.
<i>dest</i>	The destination path for the copied file or directory.

**4.1.3.8 f\_isatty()**

```
bool f_isatty (
    int fd )
```

check whether a `fd` refers to the terminal

**Parameters**

<i>fd</i>	the file descriptor
-----------	---------------------

**Returns**

true if `fd` is either `F_STDIN`, `F_STDOUT`, or `F_STDERR`; false otherwise

**4.1.3.9 f\_ls()**

```
void f_ls (
    const char * filename )
```

Lists information about files in the file system.

If the given filename is NULL, this function lists information about all files in the file system. Otherwise, it lists information about the specified file. It locates the file entry, and if found, displays information about the file using the `fs_ls_single` function. If the file is not found, `ERR_FS_FILE_NOT_FOUND` is set in `ERRNO`.

#### Parameters

<i>filename</i>	The name of the file to list information about. If NULL, lists information about all files.
-----------------	---

#### 4.1.3.10 `f_lseek()`

```
int f_lseek (
    int fd,
    int offset,
    int whence )
```

Moves the file pointer to a specified position within a file.

This function adjusts the file pointer for the specified file descriptor, allowing seeking to a new position within the file. If the file descriptor represents a terminal, an error is returned. The function then locates the corresponding file entry, retrieves the current file pointer position, and calculates the new offset based on the specified 'whence' parameter. If the new offset is within the file bounds, the file pointer is updated, and the new position is returned. Otherwise, an error is returned.

#### Parameters

<i>fd</i>	The file descriptor to seek within.
<i>offset</i>	The offset to move the file pointer.
<i>whence</i>	The reference position for the offset ( <code>F SEEK_CURR</code> , <code>F SEEK_END</code> , or <code>F SEEK_SET</code> ).

#### Returns

On success, returns the new file pointer position. On failure, returns -1, and the global variable `ERRNO` is set accordingly.

#### 4.1.3.11 `f_mount()`

```
int f_mount (
    char * fs_name,
    uint16_t ** fat )
```

Mounts a file system.

#### Parameters

<i>fs_name</i>	The name of the file system to mount.
<i>fat</i>	A pointer to the file allocation table (FAT) array.

**Returns**

On success, returns 0. On failure, returns -1.

**4.1.3.12 f\_mv()**

```
void f_mv (
    char * src,
    char * dest )
```

Moves or renames a file or directory.

**Parameters**

<i>src</i>	The source path of the file or directory.
<i>dest</i>	The destination path for the file or directory.

**4.1.3.13 f\_open()**

```
int f_open (
    const char * fname,
    int mode )
```

Opens or creates a file and returns a file descriptor.

**Parameters**

<i>fname</i>	The name of the file to open or create.
<i>mode</i>	The mode in which to open the file (F_READ, F_WRITE, or F_APPEND).

**Returns**

The file descriptor on success, or -1 on failure with ERRNO set.

**Note**

If the file already exists, the function checks permissions and handles multiple processes attempting to open the same file.

If the file does not exist, it is created, and the open files list is updated.

**4.1.3.14 f\_print()**

```
int f_print (
    const char * str )
```

Prints a string to the standard error (stderr).

**Parameters**

<i>str</i>	The string to be printed.
------------	---------------------------

**Returns**

On success, returns the number of bytes written. On failure, returns -1.

**4.1.3.15 f\_read()**

```
int f_read (
    int fd,
    int n,
    char * buf )
```

Reads data from a file descriptor.

This function reads data from the specified file descriptor and stores it in the provided buffer. If the file descriptor represents STDIN, data is read from the terminal input. If it represents STDOUT or STDERR, an error is returned. For regular file descriptors, the corresponding file entry is located, and data is read from the file into a temporary buffer.

**Parameters**

<i>fd</i>	The file descriptor to read from.
<i>n</i>	The number of bytes to read.
<i>buf</i>	The buffer to store the read data.

**Returns**

On success, the number of bytes read is returned. On failure, -1 is returned, and the global variable `ERRNO` is set accordingly. If the end of the file is reached (EOF), 0 is returned.

**4.1.3.16 f\_rm()**

```
void f_rm (
    char * filenames[],
    int n )
```

Removes (deletes) files or directories.

**Parameters**

<i>filenames</i>	An array of strings containing the names of the files or directories to be removed.
<i>n</i>	The number of filenames in the array.

#### 4.1.3.17 f\_touch()

```
void f_touch (
    char * filenames[],
    int n )
```

Creates empty files with the specified names.

##### Parameters

<i>filenames</i>	An array of strings containing the names of the files to be created.
<i>n</i>	The number of filenames in the array.

#### 4.1.3.18 f\_unlink()

```
int f_unlink (
    const char * fname )
```

Unlinks (deletes) a file.

This function unlinks (deletes) the specified file, releasing associated resources. It first locates the file entry by filename and checks if the current process is accessing the file. If so, it removes the file pointer and updates the write access status. If no more processes are accessing the file, the file entry is deleted, and the file system is updated. If the file is still in use by another process, it is marked as deleted in the file system.

##### Parameters

<i>fname</i>	The name of the file to unlink.
--------------	---------------------------------

##### Returns

On success, returns 0. On failure, returns -1, and the global variable ERRNO is set accordingly.

#### 4.1.3.19 f\_unmount()

```
void f_unmount (
    uint16_t ** fat,
    int fs_fd )
```

Unmounts a file system.



## Parameters

<i>fat</i>	A pointer to the file allocation table (FAT) array.
<i>fs</i> ↔ <i>_fd</i>	The file descriptor of the file system to unmount.

4.1.3.20 `f_write()`

```
int f_write (
    int fd,
    const char * str,
    int n )
```

Writes data to a file descriptor.

This function writes data to the specified file descriptor based on the given parameters. If the file descriptor represents STDOUT or STDERR, the data is output to the terminal. For regular file descriptors, the function locates the corresponding file entry, checks for write access, and updates the file content accordingly.

## Parameters

<i>fd</i>	The file descriptor to write to.
<i>str</i>	The string containing the data to be written.
<i>n</i>	The number of bytes to write.

## Returns

On success, returns the actual number of bytes written. On failure, returns -1, and the global variable ERRNO is set accordingly.

4.1.3.21 `find_file_entry()`

```
bool find_file_entry (
    int fd,
    int * file_id_ptr,
    file_t * file_entry )
```

find a file by *fd*

## Parameters

<i>file_id_ptr</i>	set to the file id, if the file is found
<i>file_entry</i>	set to the file entry, if the file is found

**Returns**

true if the file is found, false & print otherwise

**4.1.3.22 find\_file\_entry\_by\_file\_id()**

```
file_t* find_file_entry_by_file_id (
    int file_id )
```

**4.1.3.23 find\_file\_entry\_by\_filename()**

```
file_t* find_file_entry_by_filename (
    const char * filename )
```

**4.1.3.24 find\_unused\_fd()**

```
int find_unused_fd (
    PCB * pcb )
```

get the first unused file descriptor (for `f_open`)

**Parameters**

<i>pcb</i>	the calling process
------------	---------------------

**Returns**

the first unused fd, or `-1` if the fd table is full

**4.1.3.25 get\_fileptr()**

```
fileptr_t* get_fileptr (
    fileptr_t * fileptr_head,
    int pid )
```

get a file pointer struct for a process

**Parameters**

<i>fileptr_head</i>	the head of the fileptr list
<i>pid</i>	the process id to find a fileptr for

**Returns**

the file pointer struct, or NULL if `pid` was not found

**4.1.3.26 get\_fileptr\_ptr()**

```
int get_fileptr_ptr (
    fileptr_t * fileptr_head,
    int pid )
```

get a file pointer for a process

**Parameters**

<i>fileptr_head</i>	the head of the fileptr list
<i>pid</i>	the process id to find a fileptr for

**Returns**

the file pointer, or -1 if `pid` was not found

**4.1.3.27 is\_duplicate\_fd()**

```
bool is_duplicate_fd (
    PCB * pcb,
    int file_id )
```

check if a process already has the file open; use this to check whether to create a file pointer

**Parameters**

<i>pcb</i>	the process PCB
<i>file_id</i>	the file id (from <code>file_t.file_id</code> )

**Returns**

true if `file_id` is already in the file descriptor table of `pcb`, false otherwise

**4.1.3.28 print\_fileptr\_pids\_all()**

```
void print_fileptr_pids_all ( )
```

DEBUG: print all file pointers & their pids

**Returns**

none

**4.1.3.29 process\_create\_fileptrs()**

```
void process_create_fileptrs (
    PCB * pcb )
```

create file pointers for each unique file\_id (called by p\_spawn)

**Parameters**

<i>pcb</i>	the process <a href="#">PCB</a>
------------	---------------------------------

**Returns**

none

**4.1.3.30 process\_delete\_fileptrs()**

```
void process_delete_fileptrs (
    PCB * pcb )
```

delete file pointers of each unique file\_id (called by p\_kill)

**Parameters**

<i>pcb</i>	the process <a href="#">PCB</a>
------------	---------------------------------

**Returns**

none

**4.1.3.31 valid\_perm()**

```
bool valid_perm (
    uint8_t perm,
    int mode )
```

check if permissions requested match a file's allowed permissions

## Parameters

<i>perm</i>	the allowed permissions
<i>mode</i>	the permissions requested

## Returns

true if *perm* allows a file to be opened with *mode*, false otherwise

## 4.1.4 Variable Documentation

### 4.1.4.1 current\_pcb

```
PCB* current_pcb [extern]
```

### 4.1.4.2 next\_file\_id

```
int next_file_id = 0
```

### 4.1.4.3 open\_files

```
file_t* open_files = NULL
```

## 4.2 src/filesystem/filesystem.h File Reference

```
#include <stdint.h>
#include "../kernel/PCB.h"
```

Include dependency graph for filesystem.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [fileptr](#)
- struct [file](#)

## Macros

- `#define F_STDIN 0`
- `#define F_STDOUT 1`
- `#define F_STDERR 2`
- `#define F_WRITE 0`
- `#define F_READ 1`
- `#define F_APPEND 2`
- `#define F_SEEK_SET 0`
- `#define F_SEEK_CURR 1`
- `#define F_SEEK_END 2`
- `#define FPERM_READ 0b100`
- `#define FPERM_WRIT 0b010`
- `#define FPERM_EXEC 0b001`

## Typedefs

- `typedef struct fileptr fileptr_t`
- `typedef struct file file_t`

## Functions

- `int f_open (const char *fname, int mode)`  
*Opens or creates a file and returns a file descriptor.*
- `int f_read (int fd, int n, char *buf)`  
*Reads data from a file descriptor.*
- `int f_write (int fd, const char *str, int n)`  
*Writes data to a file descriptor.*
- `int f_close (int fd)`  
*Closes a file descriptor.*
- `int f_unlink (const char *fname)`  
*Unlinks (deletes) a file.*
- `int f_lseek (int fd, int offset, int whence)`  
*Moves the file pointer to a specified position within a file.*
- `void f_ls (const char *filename)`  
*Lists information about files in the file system.*
- `void f_touch (char *filenames[], int n)`
- `int f_print (const char *str)`  
*Prints a string to the standard error (stderr).*
- `int f_mount (char *fs_name, uint16_t **fat)`  
*Mounts a file system.*
- `void f_unmount (uint16_t **fat, int fs_fd)`  
*Unmounts a file system.*
- `void f_mv (char *src, char *dest)`  
*Moves or renames a file or directory.*
- `void f_cp (char *src, char *dest)`  
*Copies a file or directory.*
- `void f_rm (char *filenames[], int n)`  
*Removes (deletes) files or directories.*
- `void f_chmod (char *filename, int perms)`  
*Changes the permissions of a file or directory.*
- `void print_fileptr_pids_all ()`
- `void process_create_fileptrs (PCB *pcb)`
- `void process_delete_fileptrs (PCB *pcb)`
- `file_t * find_file_entry_by_file_id (int file_id)`

## 4.2.1 Macro Definition Documentation

### 4.2.1.1 F\_APPEND

```
#define F_APPEND 2
```

### 4.2.1.2 F\_READ

```
#define F_READ 1
```

### 4.2.1.3 F\_SEEK\_CURR

```
#define F_SEEK_CURR 1
```

### 4.2.1.4 F\_SEEK\_END

```
#define F_SEEK_END 2
```

### 4.2.1.5 F\_SEEK\_SET

```
#define F_SEEK_SET 0
```

### 4.2.1.6 F\_STDERR

```
#define F_STDERR 2
```

### 4.2.1.7 F\_STDIN

```
#define F_STDIN 0
```

#### 4.2.1.8 F\_STDOUT

```
#define F_STDOUT 1
```

#### 4.2.1.9 F\_WRITE

```
#define F_WRITE 0
```

#### 4.2.1.10 FPERM\_EXEC

```
#define FPERM_EXEC 0b001
```

#### 4.2.1.11 FPERM\_READ

```
#define FPERM_READ 0b100
```

#### 4.2.1.12 FPERM\_WRIT

```
#define FPERM_WRIT 0b010
```

### 4.2.2 Typedef Documentation

#### 4.2.2.1 file\_t

```
typedef struct file file_t
```

#### 4.2.2.2 fileptr\_t

```
typedef struct fileptr fileptr_t
```

### 4.2.3 Function Documentation

#### 4.2.3.1 f\_chmod()

```
void f_chmod (
    char * filename,
    int perms )
```

Changes the permissions of a file or directory.



## Parameters

<i>filename</i>	The name of the file or directory.
<i>perms</i>	The new permissions to set for the file or directory.

**4.2.3.2 f\_close()**

```
int f_close (
    int fd )
```

Closes a file descriptor.

close a file descriptor

## Parameters

<i>fd</i>	the file descriptor to close
-----------	------------------------------

## Returns

0 on success, -1 on error

This function closes the specified file descriptor, releasing associated resources. If the file descriptor represents a terminal, an error is returned. The function then finds the corresponding file entry in open files, updates the process's file descriptor table, and deallocates file pointers if this is the last instance of the file descriptor for the process. If the process had write access, it is revoked when the last instance is closed. If no process is using the file, the file entry is deleted.

## Parameters

<i>fd</i>	The file descriptor to close.
-----------	-------------------------------

## Returns

On success, returns 0. On failure, returns -1, and the global variable ERRNO is set accordingly.

**4.2.3.3 f\_cp()**

```
void f_cp (
    char * src,
    char * dest )
```

Copies a file or directory.

## Parameters

<i>src</i>	The source path of the file or directory.
<i>dest</i>	The destination path for the copied file or directory.

**4.2.3.4 f\_ls()**

```
void f_ls (
    const char * filename )
```

Lists information about files in the file system.

list a file in the current directory

## Parameters

<i>filename</i>	the file to list, or <code>NULL</code> to list all files in the current directory
-----------------	---

## Returns

none

If the given filename is `NULL`, this function lists information about all files in the file system. Otherwise, it lists information about the specified file. It locates the file entry, and if found, displays information about the file using the `fs_ls_single` function. If the file is not found, `ERR_FS_FILE_NOT_FOUND` is set in `ERRNO`.

## Parameters

<i>filename</i>	The name of the file to list information about. If <code>NULL</code> , lists information about all files.
-----------------	---

**4.2.3.5 f\_lseek()**

```
int f_lseek (
    int fd,
    int offset,
    int whence )
```

Moves the file pointer to a specified position within a file.

reposition the file pointer

## Parameters

<i>fd</i>	the file
<i>offset</i>	the offset
<i>whence</i>	Either <code>F SEEK SET</code> , <code>F SEEK CURR</code> , or <code>F SEEK END</code> . If <code>F SEEK SET</code> : file pointer is set to offset. If <code>F SEEK CURR</code> : file pointer is set to offset + the current file pointer position. If <code>F SEEK END</code> : file pointer is set to offset + file size.

**Returns**

the new file pointer position, or `-1` on error

This function adjusts the file pointer for the specified file descriptor, allowing seeking to a new position within the file. If the file descriptor represents a terminal, an error is returned. The function then locates the corresponding file entry, retrieves the current file pointer position, and calculates the new offset based on the specified 'whence' parameter. If the new offset is within the file bounds, the file pointer is updated, and the new position is returned. Otherwise, an error is returned.

**Parameters**

<i>fd</i>	The file descriptor to seek within.
<i>offset</i>	The offset to move the file pointer.
<i>whence</i>	The reference position for the offset ( <code>F SEEK_CURR</code> , <code>F SEEK_END</code> , or <code>F SEEK_SET</code> ).

**Returns**

On success, returns the new file pointer position. On failure, returns `-1`, and the global variable `ERRNO` is set accordingly.

**4.2.3.6 f\_mount()**

```
int f_mount (
    char * fs_name,
    uint16_t ** fat )
```

Mounts a file system.

**Parameters**

<i>fs_name</i>	The name of the file system to mount.
<i>fat</i>	A pointer to the file allocation table (FAT) array.

**Returns**

On success, returns `0`. On failure, returns `-1`.

**4.2.3.7 f\_mv()**

```
void f_mv (
    char * src,
    char * dest )
```

Moves or renames a file or directory.

## Parameters

<i>src</i>	The source path of the file or directory.
<i>dest</i>	The destination path for the file or directory.

**4.2.3.8 f\_open()**

```
int f_open (
    const char * fname,
    int mode )
```

Opens or creates a file and returns a file descriptor.

open a file name *fname* with the mode *mode* and return a file descriptor

## Parameters

<i>fname</i>	the filename to open
<i>mode</i>	Either F_WRITE, F_READ, or F_APPEND. If F_WRITE: reading and writing, truncate if the file exists and create it otherwise; only 1 file instance of F_WRITE mode can exist. If F_READ: read only. If F_APPEND: reading and writing, do not truncate if the file exists; file pointer is set to end of file.

## Returns

a file descriptor on success, -1 otherwise

## Parameters

<i>fname</i>	The name of the file to open or create.
<i>mode</i>	The mode in which to open the file (F_READ, F_WRITE, or F_APPEND).

## Returns

The file descriptor on success, or -1 on failure with ERRNO set.

## Note

If the file already exists, the function checks permissions and handles multiple processes attempting to open the same file.

If the file does not exist, it is created, and the open files list is updated.

#### 4.2.3.9 f\_print()

```
int f_print (
    const char * str )
```

Prints a string to the standard error (stderr).

print to terminal (F\_STDERR)

**Parameters**

<i>str</i>	the string to print (use <code>snprintf</code> to format)
------------	---

**Returns**

number of bytes written (including `\0`) on success, `-1` on error

**Parameters**

<i>str</i>	The string to be printed.
------------	---------------------------

**Returns**

On success, returns the number of bytes written. On failure, returns `-1`.

**4.2.3.10 `f_read()`**

```
int f_read (
    int fd,
    int n,
    char * buf )
```

Reads data from a file descriptor.

read from a file

**Parameters**

<i>fd</i>	the file descriptor to read from
<i>n</i>	number of bytes to read
<i>buf</i>	buffer to read into

**Returns**

number of bytes read (including `\0`) on success, `0` if EOF is reached, `-1` on error

This function reads data from the specified file descriptor and stores it in the provided buffer. If the file descriptor represents `STDIN`, data is read from the terminal input. If it represents `STDOUT` or `STDERR`, an error is returned. For regular file descriptors, the corresponding file entry is located, and data is read from the file into a temporary buffer.

**Parameters**

<i>fd</i>	The file descriptor to read from.
<i>n</i>	The number of bytes to read.
<i>buf</i>	The buffer to store the read data.

**Returns**

On success, the number of bytes read is returned. On failure, -1 is returned, and the global variable `ERRNO` is set accordingly. If the end of the file is reached (EOF), 0 is returned.

**4.2.3.11 `f_rm()`**

```
void f_rm (
    char * filenames[],
    int n )
```

Removes (deletes) files or directories.

**Parameters**

<i>filenames</i>	An array of strings containing the names of the files or directories to be removed.
<i>n</i>	The number of filenames in the array.

**4.2.3.12 `f_touch()`**

```
void f_touch (
    char * filenames[],
    int n )
```

list a file in the current directory

**Parameters**

<i>filenames</i>	files to touch
<i>n</i>	number of files to touch

**Returns**

none

Creates empty files with the specified names.

**Parameters**

<i>filenames</i>	An array of strings containing the names of the files to be created.
<i>n</i>	The number of filenames in the array.

#### 4.2.3.13 `f_unlink()`

```
int f_unlink (
    const char * fname )
```

Unlinks (deletes) a file.

remove a file, if it exists & is not in use

##### Parameters

<i>fname</i>	the file to remove
--------------	--------------------

##### Returns

0 on success, -1 otherwise

This function unlinks (deletes) the specified file, releasing associated resources. It first locates the file entry by filename and checks if the current process is accessing the file. If so, it removes the file pointer and updates the write access status. If no more processes are accessing the file, the file entry is deleted, and the file system is updated. If the file is still in use by another process, it is marked as deleted in the file system.

##### Parameters

<i>fname</i>	The name of the file to unlink.
--------------	---------------------------------

##### Returns

On success, returns 0. On failure, returns -1, and the global variable ERRNO is set accordingly.

#### 4.2.3.14 `f_unmount()`

```
void f_unmount (
    uint16_t ** fat,
    int fs_fd )
```

Unmounts a file system.

##### Parameters

<i>fat</i>	A pointer to the file allocation table (FAT) array.
<i>fs</i> ↔ <i>_fd</i>	The file descriptor of the file system to unmount.



#### 4.2.3.15 f\_write()

```
int f_write (
    int fd,
    const char * str,
    int n )
```

Writes data to a file descriptor.

write to a file & increment file pointer

##### Parameters

<i>fd</i>	the file descriptor to write to
<i>str</i>	the string to write from
<i>n</i>	number of bytes to write

##### Returns

number of bytes written (including `\0`) on success, `-1` on error

This function writes data to the specified file descriptor based on the given parameters. If the file descriptor represents `STDOUT` or `STDERR`, the data is output to the terminal. For regular file descriptors, the function locates the corresponding file entry, checks for write access, and updates the file content accordingly.

##### Parameters

<i>fd</i>	The file descriptor to write to.
<i>str</i>	The string containing the data to be written.
<i>n</i>	The number of bytes to write.

##### Returns

On success, returns the actual number of bytes written. On failure, returns `-1`, and the global variable `ERRNO` is set accordingly.

#### 4.2.3.16 find\_file\_entry\_by\_file\_id()

```
file_t* find_file_entry_by_file_id (
    int file_id )
```

#### 4.2.3.17 print\_fileptr\_pids\_all()

```
void print_fileptr_pids_all ( )
```

DEBUG: print all file pointers & their pids

##### Returns

none

#### 4.2.3.18 process\_create\_fileptrs()

```
void process_create_fileptrs (
    PCB * pcb )
```

create file pointers for each unique file\_id (called by p\_spawn)

##### Parameters

<i>pcb</i>	the process PCB
------------	-----------------

##### Returns

none

#### 4.2.3.19 process\_delete\_fileptrs()

```
void process_delete_fileptrs (
    PCB * pcb )
```

delete file pointers of each unique file\_id (called by p\_kill)

##### Parameters

<i>pcb</i>	the process PCB
------------	-----------------

##### Returns

none

### 4.3 src/kernel/kernel-functions.c File Reference

```
#include "PCB.h"
#include "kernel-functions.h"
#include "../logger/logger.h"
#include "../util/globals.h"
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include "stdio.h"
```

Include dependency graph for kernel-functions.c:

#### Functions

- PCB \* k\_process\_create (PCB \*parent)
- int k\_process\_kill (PCB \*process, int signal)
- void k\_process\_deep\_cleanup (PCB \*process)
- void k\_process\_cleanup (PCB \*process)

## 4.3.1 Function Documentation

### 4.3.1.1 k\_process\_cleanup()

```
void k_process_cleanup (
    PCB * process )
```

frees PCB `process`

#### Parameters

<i>process</i>	pointer of PCB to be freed
----------------	----------------------------

#### Returns

none

### 4.3.1.2 k\_process\_create()

```
PCB* k_process_create (
    PCB * parent )
```

creates PCB and adds it to global PCB list

#### Parameters

<i>parent</i>	the parent of PCB to be created, if it exists
---------------	---

#### Returns

created PCB

### 4.3.1.3 k\_process\_deep\_cleanup()

```
void k_process_deep_cleanup (
    PCB * process )
```

frees PCB `process` and all of its descendants

#### Parameters

<i>process</i>	pointer of PCB (and its descendants) to be freed
----------------	--

**Returns**

none

**4.3.1.4 k\_process\_kill()**

```
int k_process_kill (
    PCB * process,
    int signal )
```

sends signal `signal` to inputted `PCB` `process`

**Parameters**

<i>process</i>	pointer of <code>PCB</code> to send signal to
<i>signal</i>	signal to send

**Returns**

0 on success; 1 on failure

**4.4 src/kernel/kernel-functions.h File Reference**

```
#include "PCB.h"
```

Include dependency graph for kernel-functions.h: This graph shows which files directly or indirectly include this file:

**Functions**

- `PCB * k_process_create (PCB *parent)`
- `int k_process_kill (PCB *process, int signal)`
- `void k_process_deep_cleanup (PCB *process)`
- `void k_process_cleanup (PCB *process)`

**4.4.1 Function Documentation****4.4.1.1 k\_process\_cleanup()**

```
void k_process_cleanup (
    PCB * process )
```

frees `PCB` `process`

## Parameters

<i>process</i>	pointer of <a href="#">PCB</a> to be freed
----------------	--

## Returns

none

#### 4.4.1.2 [k\\_process\\_create\(\)](#)

```
PCB* k_process_create (
    PCB * parent )
```

creates [PCB](#) and adds it to global [PCB](#) list

## Parameters

<i>parent</i>	the parent of <a href="#">PCB</a> to be created, if it exists
---------------	---

## Returns

created [PCB](#)

#### 4.4.1.3 [k\\_process\\_deep\\_cleanup\(\)](#)

```
void k_process_deep_cleanup (
    PCB * process )
```

frees [PCB](#) *process* and all of its descendants

## Parameters

<i>process</i>	pointer of <a href="#">PCB</a> (and its descendants) to be freed
----------------	--

## Returns

none

#### 4.4.1.4 [k\\_process\\_kill\(\)](#)

```
int k_process_kill (
    PCB * process,
    int signal )
```

sends signal `signal` to inputted `PCB` process

#### Parameters

<i>process</i>	pointer of <code>PCB</code> to send signal to
<i>signal</i>	signal to send

#### Returns

0 on success; 1 on failure

## 4.5 src/kernel/PCB.c File Reference

```
#include "PCB.h"
#include "scheduler.h"
#include <stdio.h>
#include "../util/globals.h"
#include <valgrind/valgrind.h>
Include dependency graph for PCB.c:
```

## 4.6 src/kernel/PCB.h File Reference

```
#include <ucontext.h>
#include <sys/types.h>
#include <stdbool.h>
#include <limits.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
Include dependency graph for PCB.h: This graph shows which files directly or indirectly include this file:
```

### Data Structures

- struct `PCB`

### Macros

- #define `STACKSIZE` 4096 \* 256
- #define `MAX_FDS` 1024
- #define `NOFILE` -1
- #define `STDIN_ID` -2
- #define `STDOUT_ID` -3
- #define `STDERR_ID` -4

### Typedefs

- typedef struct `PCB` `PCB`

## Functions

- void `k_free` (`PCB *pcb`)
- `PCB *` `createPCB` (`PCB *parent`)
- void `addPCBToList` (`PCB **list`, `PCB *pcb`)
- void `removePCBFromList` (`PCB **list`, `PCB *pcb`)
- `PCB *` `findPCBByPID` (`pid_t pid`)
- `PCB *` `findPCBByContext` (`ucontext_t *context`)
- int `getLength` (`PCB *list`)
- int `count_running` (`PCB *head`)
- int `count_running_priority` (`PCB *head`, int prio)

## Variables

- `PCB *` `pcb_list`
- `pid_t` `next_pid`

### 4.6.1 Macro Definition Documentation

#### 4.6.1.1 MAX\_FDS

```
#define MAX_FDS 1024
```

#### 4.6.1.2 NOFILE

```
#define NOFILE -1
```

#### 4.6.1.3 STACKSIZE

```
#define STACKSIZE 4096 * 256
```

#### 4.6.1.4 STDERR\_ID

```
#define STDERR_ID -4
```

#### 4.6.1.5 STDIN\_ID

```
#define STDIN_ID -2
```

#### 4.6.1.6 STDOUT\_ID

```
#define STDOUT_ID -3
```

### 4.6.2 Typedef Documentation

#### 4.6.2.1 PCB

```
typedef struct PCB PCB
```

### 4.6.3 Function Documentation

#### 4.6.3.1 addPCBToList()

```
void addPCBToList (
    PCB ** head,
    PCB * pcb )
```

add a process to the PCBList

##### Parameters

<i>list</i>	the PCBList
<i>pcb</i>	the process PCB to add

##### Returns

none

adds a given a PCB pcb to list

##### Parameters

<i>head</i>	pointer to head of circular linked list
<i>pcb</i>	the pcb we want to add



**Returns**

None

**4.6.3.2 count\_running()**

```
int count_running (
    PCB * head )
```

count number of T\_RUNNING processes in a circular linked list

**Parameters**

<i>head</i>	pointer to the head of circular linked list
-------------	---

**Returns**

number of T\_RUNNING processes

**4.6.3.3 count\_running\_priority()**

```
int count_running_priority (
    PCB * head,
    int prio )
```

count number of T\_RUNNING processes with desired priority *prio* in a circular linked list

**Parameters**

<i>head</i>	pointer to the head of circular linked list
<i>prio</i>	desired priority (-1, 0, or 1)

**Returns**

number relevant processes

**4.6.3.4 createPCB()**

```
PCB* createPCB (
    PCB * Parent )
```

create a PCB

**Parameters**

<i>parent</i>	the parent process <a href="#">PCB</a>
---------------	--

**Returns**

the created [PCB](#)

creates [PCB](#) if parent `Parent` isn't NULL, `parent_pid` and `priority` will be inherited instantiated [PCB](#) will also be added to `Parent` 's array of children

**Parameters**

<i>Parent</i>	of parent (if it exists) for the newly created <a href="#">PCB</a> .
---------------	--

**Returns**

returns the newly created [PCB](#).

**4.6.3.5 findPCBByContext()**

```
PCB* findPCBByContext (
    ucontext_t * context )
```

find a [PCB](#) by `ucontext` in the global [PCB](#) list

**Parameters**

<i>context</i>	the process <code>ucontext</code> to find
----------------	---

**Returns**

the [PCB](#), or NULL if it was not found

finds [PCB](#) with desired `ucontext_t context`

**Parameters**

<i>context</i>	the pointer of the given context
----------------	----------------------------------

**Returns**

[PCB](#) with desired context, if it exists

#### 4.6.3.6 findPCBByPID()

```
PCB* findPCBByPID (
    pid_t pid )
```

find a [PCB](#) by pid in the global [PCB](#) list

##### Parameters

<i>pid</i>	the process pid to find
------------	-------------------------

##### Returns

the [PCB](#), or NULL if it was not found

finds [PCB](#) with desired pid *pid*

##### Parameters

<i>pid</i>	the pid of the process we want to find
------------	--

##### Returns

[PCB](#) with desired pid, if it exists

#### 4.6.3.7 getLength()

```
int getLength (
    PCB * head )
```

gets length of circular linked list

##### Parameters

<i>head</i>	pointer to the head of circular linked list
-------------	---

##### Returns

the length of the pcb list

#### 4.6.3.8 k\_free()

```
void k_free (
    PCB * process )
```

free memory of a [PCB](#)

**Parameters**

<i>pcb</i>	the pcb
------------	---------

**Returns**

none

frees inputted **PCB** process

**Parameters**

<i>process</i>	pointer to <b>PCB</b> to be freed
----------------	-----------------------------------

**Returns**

none

**4.6.3.9 removePCBFromList()**

```
void removePCBFromList (  
    PCB ** head,  
    PCB * pcb )
```

remove a process from the PCBList

**Parameters**

<i>list</i>	the PCBList
<i>pcb</i>	the process <b>PCB</b> to remove

**Returns**

none

removes a given a **PCB** pcb from a given list

**Parameters**

<i>head</i>	pointer to head of circular linked list
<i>pcb</i>	the pcb we want to remove

**Returns**

None

## 4.6.4 Variable Documentation

### 4.6.4.1 next\_pid

```
pid_t next_pid [extern]
```

### 4.6.4.2 pcb\_list

```
PCB* pcb_list [extern]
```

## 4.7 src/kernel/puser-functions.c File Reference

```
#include "kernel-functions.h"
#include "scheduler.h"
#include "../filesystem/filesystem.h"
#include "../logger/logger.h"
#include "../util/globals.h"
#include "../util/p-errno.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <ucontext.h>
#include <valgrind/valgrind.h>
Include dependency graph for puser-functions.c:
```

## Functions

- int [p\\_spawn](#) (void(\*func)(), char \*argv[], int fd0, int fd1)
- pid\_t [p\\_waitpid](#) (pid\_t pid, int \*wstatus, bool nohang)
- int [p\\_kill](#) (pid\_t pid, int sig)
- int [p\\_nice](#) (pid\_t pid, int priority)
- void [p\\_sleep](#) (unsigned int time)
- void [p\\_exit](#) (void)
- bool [W\\_WIFEXITED](#) (int status)
- bool [W\\_WIFSTOPPED](#) (int status)
- bool [W\\_WIFCONTINUED](#) (int status)
- bool [W\\_WIFSIGNALED](#) (int status)

## Variables

- [PCB](#) \* [current\\_pcb](#) = NULL
- int [ticks](#) = 0

## 4.7.1 Function Documentation

### 4.7.1.1 p\_exit()

```
void p_exit (
    void )
```

exits current PCB unconditionally

#### Returns

none

### 4.7.1.2 p\_kill()

```
int p_kill (
    pid_t pid,
    int sig )
```

sends signal *sig* to PCB with pid *pid*

#### Parameters

<i>pid</i>	pid of PCB to send signal to
<i>sig</i>	signal to send

#### Returns

0 on sucess; -1 on failure

### 4.7.1.3 p\_nice()

```
int p_nice (
    pid_t pid,
    int priority )
```

changes priority of PCB with pid *pid* to inputted priority *priority*

#### Parameters

<i>pid</i>	pid of PCB to change priority of
<i>priority</i>	priority to change to

**Returns**

0 on success; -1 on failure

**4.7.1.4 p\_sleep()**

```
void p_sleep (
    unsigned int time )
```

blocks current [PCB](#) for time ticks

**Parameters**

<i>time</i>	ticks to block for
-------------	--------------------

**Returns**

none

**4.7.1.5 p\_spawn()**

```
int p_spawn (
    void(*)() func,
    char * argv[],
    int fd0,
    int fd1 )
```

spawns [PCB](#) with start function *func* , input arguments *argv* , and I/O *fd0* / *fd1*

**Parameters**

<i>func</i>	start function of <a href="#">PCB</a>
<i>argv</i>	arguments of func
<i>fd0</i>	F_STDIN file descriptor
<i>fd1</i>	F_STDOUT file descriptor

**Returns**

pid of spawned [PCB](#) on success; -1 on failure

**4.7.1.6 p\_waitpid()**

```
pid_t p_waitpid (
    pid_t pid,
```

```
int * wstatus,  
bool nohang )
```

if `nohang` is false, waits until relevant [PCB\(s\)](#) changes state if `nohang` is true, returns immediately

#### Parameters

<i>pid</i>	pid of <a href="#">PCB</a> to wait for; if -1, wait for any child of current <a href="#">PCB</a>
<i>wstatus</i>	pointer to integer to store status in
<i>nohang</i>	true to return immediately, false to block parent <a href="#">PCB</a>

#### Returns

0 on success, -1 on failure

#### 4.7.1.7 W\_WIFCONTINUED()

```
bool W_WIFCONTINUED (  
    int status )
```

#### Returns

true if the child was continued by a signal

#### 4.7.1.8 W\_WIFEXITED()

```
bool W_WIFEXITED (  
    int status )
```

#### Returns

true if the child terminated normally, that is, by a call to [p\\_exit](#) or by returning

#### 4.7.1.9 W\_WIFSIGNALED()

```
bool W_WIFSIGNALED (  
    int status )
```

#### Returns

true if the child was terminated by a signal



#### 4.7.1.10 W\_WIFSTOPPED()

```
bool W_WIFSTOPPED (
    int status )
```

##### Returns

true if the child was stopped by a signal

### 4.7.2 Variable Documentation

#### 4.7.2.1 current\_pcb

```
PCB* current_pcb = NULL
```

#### 4.7.2.2 ticks

```
int ticks = 0
```

## 4.8 src/kernel/puser-functions.h File Reference

```
#include "kernel-functions.h"
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <ucontext.h>
```

Include dependency graph for puser-functions.h: This graph shows which files directly or indirectly include this file:

### Functions

- int [p\\_spawn](#) (void(\*func)(), char \*argv[], int fd0, int fd1)
- pid\_t [p\\_waitpid](#) (pid\_t pid, int \*wstatus, bool nohang)
- int [p\\_kill](#) (pid\_t pid, int sig)
- int [p\\_nice](#) (pid\_t pid, int priority)
- void [p\\_sleep](#) (unsigned int time)
- void [p\\_exit](#) (void)
- bool [W\\_WIFEXITED](#) (int status)
- bool [W\\_WIFSTOPPED](#) (int status)
- bool [W\\_WIFCONTINUED](#) (int status)
- bool [W\\_WIFSIGNALED](#) (int status)

## Variables

- `PCB * current_pcb`
- `int ticks`

## 4.8.1 Function Documentation

### 4.8.1.1 `p_exit()`

```
void p_exit (
    void )
```

exits current `PCB` unconditionally

#### Returns

none

### 4.8.1.2 `p_kill()`

```
int p_kill (
    pid_t pid,
    int sig )
```

sends signal `sig` to `PCB` with pid `pid`

#### Parameters

<i>pid</i>	pid of <code>PCB</code> to send signal to
<i>sig</i>	signal to send

#### Returns

0 on sucess; -1 on failure

### 4.8.1.3 `p_nice()`

```
int p_nice (
    pid_t pid,
    int priority )
```

changes priority of `PCB` with pid `pid` to inputted priority `priority`

## Parameters

<i>pid</i>	pid of <a href="#">PCB</a> to change priority of
<i>priority</i>	priority to change to

## Returns

0 on sucess; -1 on failure

#### 4.8.1.4 p\_sleep()

```
void p_sleep (
    unsigned int time )
```

blocks current [PCB](#) for time ticks

## Parameters

<i>time</i>	ticks to block for
-------------	--------------------

## Returns

none

#### 4.8.1.5 p\_spawn()

```
int p_spawn (
    void(*)() func,
    char * argv[],
    int fd0,
    int fd1 )
```

spawns [PCB](#) with start function `func` , input arguments `argv` , and I/O `fd0` / `fd1`

## Parameters

<i>func</i>	start function of <a href="#">PCB</a>
<i>argv</i>	arguments of func
<i>fd0</i>	F_STDIN file descriptor
<i>fd1</i>	F_STDOUT file descriptor

## Returns

pid of spawned [PCB](#) on success; -1 on failure

#### 4.8.1.6 p\_waitpid()

```
pid_t p_waitpid (
    pid_t pid,
    int * wstatus,
    bool nohang )
```

if `nohang` is false, waits until relevant [PCB\(s\)](#) changes state if `nohang` is true, returns immediately

##### Parameters

<i>pid</i>	pid of <a href="#">PCB</a> to wait for; if -1, wait for any child of current <a href="#">PCB</a>
<i>wstatus</i>	pointer to integer to store status in
<i>nohang</i>	true to return immediately, false to block parent <a href="#">PCB</a>

##### Returns

0 on success, -1 on failure

#### 4.8.1.7 W\_WIFCONTINUED()

```
bool W_WIFCONTINUED (
    int status )
```

##### Returns

true if the child was continued by a signal

#### 4.8.1.8 W\_WIFEXITED()

```
bool W_WIFEXITED (
    int status )
```

##### Returns

true if the child terminated normally, that is, by a call to [p\\_exit](#) or by returning

#### 4.8.1.9 W\_WIFSIGNALED()

```
bool W_WIFSIGNALED (
    int status )
```

##### Returns

true if the child was terminated by a signal

#### 4.8.1.10 W\_WIFSTOPPED()

```
bool W_WIFSTOPPED (
    int status )
```

##### Returns

true if the child was stopped by a signal

### 4.8.2 Variable Documentation

#### 4.8.2.1 current\_pcb

```
PCB* current_pcb [extern]
```

#### 4.8.2.2 ticks

```
int ticks [extern]
```

## 4.9 src/kernel/scheduler.c File Reference

```
#include "puser-functions.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <fcntl.h>
#include <ucontext.h>
#include <sys/time.h>
#include "../util/globals.h"
#include "../filesystem/filesystem.h"
#include "../logger/logger.h"
#include <time.h>
#include <valgrind/valgrind.h>
Include dependency graph for scheduler.c:
```

## Functions

- static void [scheduler](#) (void)
- static void [reaper](#) ()
- static void [alarmHandler](#) (int signum)
- static void [setAlarmHandler](#) (void)
- static void [setTimer](#) (void)
- static void [freeStacks](#) (void)
- void [start\\_scheduler](#) ()

## Variables

- static ucontext\_t [mainContext](#)
- ucontext\_t [schedulerContext](#)
- ucontext\_t [reaperContext](#)
- static ucontext\_t \* [activeContext](#) = NULL
- static const int [centisecond](#) = 10000

### 4.9.1 Function Documentation

#### 4.9.1.1 alarmHandler()

```
static void alarmHandler (  
    int signum ) [static]
```

alarm handler that is invoked when alarm is signalled at every quanta increments ticks and sets current context to scheduler, effectively allowing the scheduler to run at every quanta

#### Returns

none

#### 4.9.1.2 freeStacks()

```
static void freeStacks (  
    void ) [static]
```

frees all contexts prior to exit

#### Returns

none

#### 4.9.1.3 reaper()

```
static void reaper ( ) [static]
```

reaper function that runs at termination of [PCB](#) increments ticks and sets current context back to scheduler

##### Returns

none

#### 4.9.1.4 scheduler()

```
static void scheduler (
    void ) [static]
```

scheduler function - function to be run at every tick decides which [PCB](#) to be run for the remainder of current tick

##### Returns

none

#### 4.9.1.5 setAlarmHandler()

```
static void setAlarmHandler (
    void ) [static]
```

sets [alarmHandler](#) to be called when alarm is signalled

##### Returns

none

#### 4.9.1.6 setTimer()

```
static void setTimer (
    void ) [static]
```

sets timer to invoke alarm every centisecond

##### Returns

none

#### 4.9.1.7 start\_scheduler()

```
void start_scheduler ( )
```

initializes and start scheduler

Returns

none

### 4.9.2 Variable Documentation

#### 4.9.2.1 activeContext

```
ucontext_t* activeContext = NULL [static]
```

#### 4.9.2.2 centisecond

```
const int centisecond = 10000 [static]
```

#### 4.9.2.3 mainContext

```
ucontext_t mainContext [static]
```

#### 4.9.2.4 reaperContext

```
ucontext_t reaperContext
```

#### 4.9.2.5 schedulerContext

```
ucontext_t schedulerContext
```



## 4.10 src/kernel/scheduler.h File Reference

```
#include "puser-functions.h"
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <fcntl.h>
#include <ucontext.h>
#include <sys/time.h>
```

Include dependency graph for scheduler.h: This graph shows which files directly or indirectly include this file:

### Functions

- void [init\\_scheduler](#) ()
- void [start\\_scheduler](#) ()
- void [setAlarmHandler](#) ()
- void [setTimer](#) ()

### Variables

- ucontext\_t [schedulerContext](#)
- ucontext\_t [reaperContext](#)

### 4.10.1 Function Documentation

#### 4.10.1.1 [init\\_scheduler\(\)](#)

```
void init_scheduler ( )
```

#### 4.10.1.2 [setAlarmHandler\(\)](#)

```
void setAlarmHandler ( )
```

#### 4.10.1.3 [setTimer\(\)](#)

```
void setTimer ( )
```

#### 4.10.1.4 start\_scheduler()

```
void start_scheduler ( )
```

initializes and start scheduler

##### Returns

none

### 4.10.2 Variable Documentation

#### 4.10.2.1 reaperContext

```
ucontext_t reaperContext [extern]
```

#### 4.10.2.2 schedulerContext

```
ucontext_t schedulerContext [extern]
```

## 4.11 src/logger/logger.c File Reference

```
#include "logger.h"
#include "stdio.h"
#include "../kernel/puser-functions.h"
Include dependency graph for logger.c:
```

### Functions

- void [log\\_schedule\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs a scheduling event.*
- void [log\\_create\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs a process creation event.*
- void [log\\_signaled\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs a signalling event.*
- void [log\\_exited\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs an exiting event (natural termination via [p\\_exit](#)) event.*
- void [log\\_zombie\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs zombie event.*
- void [log\\_orphan\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs orphan event.*
- void [log\\_waited\\_event](#) (int pid, int prio, char \*process\_name)

*Logs waiting event.*

- void [log\\_nice\\_event](#) (int pid, int old\_prio, int new\_prio, char \*process\_name)

*Logs a priority change event.*

- void [log\\_blocked\\_event](#) (int pid, int prio, char \*process\_name)

*Logs a blocked event (via waitpid).*

- void [log\\_unblocked\\_event](#) (int pid, int prio, char \*process\_name)

*Logs an unblocked event.*

- void [log\\_stopped\\_event](#) (int pid, int prio, char \*process\_name)

*Logs a stopped event (via signalling).*

- void [log\\_continued\\_event](#) (int pid, int prio, char \*process\_name)

*Logs a continued event.*

## Variables

- FILE \* [logfile](#)

### 4.11.1 Function Documentation

#### 4.11.1.1 [log\\_blocked\\_event\(\)](#)

```
void log_blocked_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a blocked event (via waitpid).

##### Parameters

<i>pid</i>	Process ID of the blocked process.
<i>prio</i>	Priority of the process.
<i>process_name</i>	Name of the process.

#### 4.11.1.2 [log\\_continued\\_event\(\)](#)

```
void log_continued_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a continued event.

## Parameters

<i>pid</i>	Process ID of the continued process.
<i>prio</i>	Priority of the process.
<i>process_name</i>	Name of the process.

**4.11.1.3 log\_create\_event()**

```
void log_create_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a process creation event.

## Parameters

<i>pid</i>	Process ID of the created process.
<i>prio</i>	Priority of the created process.
<i>process_name</i>	Name of the created process.

**4.11.1.4 log\_exited\_event()**

```
void log_exited_event (
    int pid,
    int prio,
    char * process_name )
```

Logs an exiting event (natural termination via [p\\_exit](#)) event.

## Parameters

<i>pid</i>	Process ID of the exited process.
<i>prio</i>	Priority of the exited process.
<i>process_name</i>	Name of the exited process.

**4.11.1.5 log\_nice\_event()**

```
void log_nice_event (
    int pid,
    int old_prio,
```

```
int new_prio,  
char * process_name )
```

Logs a priority change event.

#### Parameters

<i>pid</i>	Process ID of the changed process.
<i>old_prio</i>	Old priority of the process.
<i>new_prio</i>	New priority of the process.
<i>process_name</i>	Name of the process.

#### 4.11.1.6 log\_orphan\_event()

```
void log_orphan_event (  
    int pid,  
    int prio,  
    char * process_name )
```

Logs orphan event.

#### Parameters

<i>pid</i>	Process ID of the orphaned process.
<i>prio</i>	Priority of the orphaned process.
<i>process_name</i>	Name of the orphaned process.

#### 4.11.1.7 log\_schedule\_event()

```
void log_schedule_event (  
    int pid,  
    int prio,  
    char * process_name )
```

Logs a scheduling event.

#### Parameters

<i>pid</i>	Process ID of the scheduled process.
<i>prio</i>	Priority of the scheduled process.
<i>process_name</i>	Name of the scheduled process.

#### 4.11.1.8 log\_signaled\_event()

```
void log_signaled_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a signalling event.

##### Parameters

<i>pid</i>	Process ID of the signalled process.
<i>prio</i>	Priority of the signalled process.
<i>process_name</i>	Name of the signalled process.

#### 4.11.1.9 log\_stopped\_event()

```
void log_stopped_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a stopped event (via signalling).

##### Parameters

<i>pid</i>	Process ID of the stopped process.
<i>prio</i>	Priority of the process.
<i>process_name</i>	Name of the process.

#### 4.11.1.10 log\_unblocked\_event()

```
void log_unblocked_event (
    int pid,
    int prio,
    char * process_name )
```

Logs an unblocked event.

##### Parameters

<i>pid</i>	Process ID of the unblocked process.
<i>prio</i>	Priority of the process.
<i>process_name</i>	Name of the process.

#### 4.11.1.11 log\_waited\_event()

```
void log_waited_event (
    int pid,
    int prio,
    char * process_name )
```

Logs waiting event.

##### Parameters

<i>pid</i>	Process ID of the waiting process.
<i>prio</i>	Priority of the waiting process.
<i>process_name</i>	Name of the waiting process.

#### 4.11.1.12 log\_zombie\_event()

```
void log_zombie_event (
    int pid,
    int prio,
    char * process_name )
```

Logs zombie event.

##### Parameters

<i>pid</i>	Process ID of the zombied process.
<i>prio</i>	Priority of the zombied process.
<i>process_name</i>	Name of the zombied process.

### 4.11.2 Variable Documentation

#### 4.11.2.1 logfile

```
FILE* logfile
```

## 4.12 src/logger/logger.h File Reference

```
#include "stdio.h"
```

Include dependency graph for logger.h: This graph shows which files directly or indirectly include this file:

## Functions

- void [log\\_schedule\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs a scheduling event.*
- void [log\\_create\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs a process creation event.*
- void [log\\_signaled\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs a signalling event.*
- void [log\\_exited\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs an exiting event (natural termination via [p\\_exit](#)) event.*
- void [log\\_zombie\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs zombie event.*
- void [log\\_orphan\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs orphan event.*
- void [log\\_waited\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs waiting event.*
- void [log\\_nice\\_event](#) (int pid, int old\_prio, int new\_prio, char \*process\_name)  
*Logs a priority change event.*
- void [log\\_blocked\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs a blocked event (via [waitpid](#)).*
- void [log\\_unblocked\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs an unblocked event.*
- void [log\\_stopped\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs a stopped event (via signalling).*
- void [log\\_continued\\_event](#) (int pid, int prio, char \*process\_name)  
*Logs a continued event.*

## Variables

- FILE \* [logfile](#)

### 4.12.1 Function Documentation

#### 4.12.1.1 [log\\_blocked\\_event\(\)](#)

```
void log_blocked_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a blocked event (via [waitpid](#)).

#### Parameters

<i>pid</i>	Process ID of the blocked process.
<i>prio</i>	Priority of the process.
<i>process_name</i>	Name of the process.



#### 4.12.1.2 log\_continued\_event()

```
void log_continued_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a continued event.

##### Parameters

<i>pid</i>	Process ID of the continued process.
<i>prio</i>	Priority of the process.
<i>process_name</i>	Name of the process.

#### 4.12.1.3 log\_create\_event()

```
void log_create_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a process creation event.

##### Parameters

<i>pid</i>	Process ID of the created process.
<i>prio</i>	Priority of the created process.
<i>process_name</i>	Name of the created process.

#### 4.12.1.4 log\_exited\_event()

```
void log_exited_event (
    int pid,
    int prio,
    char * process_name )
```

Logs an exiting event (natural termination via [p\\_exit](#)) event.

##### Parameters

<i>pid</i>	Process ID of the exited process.
<i>prio</i>	Priority of the exited process.
<i>process_name</i>	Name of the exited process.

#### 4.12.1.5 log\_nice\_event()

```
void log_nice_event (
    int pid,
    int old_prio,
    int new_prio,
    char * process_name )
```

Logs a priority change event.

##### Parameters

<i>pid</i>	Process ID of the changed process.
<i>old_prio</i>	Old priority of the process.
<i>new_prio</i>	New priority of the process.
<i>process_name</i>	Name of the process.

#### 4.12.1.6 log\_orphan\_event()

```
void log_orphan_event (
    int pid,
    int prio,
    char * process_name )
```

Logs orphan event.

##### Parameters

<i>pid</i>	Process ID of the ophaned process.
<i>prio</i>	Priority of the ophaned process.
<i>process_name</i>	Name of the ophaned process.

#### 4.12.1.7 log\_schedule\_event()

```
void log_schedule_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a scheduling event.

## Parameters

<i>pid</i>	Process ID of the scheduled process.
<i>prio</i>	Priority of the scheduled process.
<i>process_name</i>	Name of the scheduled process.

**4.12.1.8 log\_signaled\_event()**

```
void log_signaled_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a signalling event.

## Parameters

<i>pid</i>	Process ID of the signalled process.
<i>prio</i>	Priority of the signalled process.
<i>process_name</i>	Name of the signalled process.

**4.12.1.9 log\_stopped\_event()**

```
void log_stopped_event (
    int pid,
    int prio,
    char * process_name )
```

Logs a stopped event (via signalling).

## Parameters

<i>pid</i>	Process ID of the stopped process.
<i>prio</i>	Priority of the process.
<i>process_name</i>	Name of the process.

**4.12.1.10 log\_unblocked\_event()**

```
void log_unblocked_event (
    int pid,
```

```
int prio,  
char * process_name )
```

Logs an unblocked event.

## Parameters

<i>pid</i>	Process ID of the unblocked process.
<i>prio</i>	Priority of the process.
<i>process_name</i>	Name of the process.

**4.12.1.11 log\_waited\_event()**

```
void log_waited_event (
    int pid,
    int prio,
    char * process_name )
```

Logs waiting event.

## Parameters

<i>pid</i>	Process ID of the waiting process.
<i>prio</i>	Priority of the waiting process.
<i>process_name</i>	Name of the waiting process.

**4.12.1.12 log\_zombie\_event()**

```
void log_zombie_event (
    int pid,
    int prio,
    char * process_name )
```

Logs zombie event.

## Parameters

<i>pid</i>	Process ID of the zombied process.
<i>prio</i>	Priority of the zombied process.
<i>process_name</i>	Name of the zombied process.

**4.12.2 Variable Documentation****4.12.2.1 logfile**

```
FILE* logfile [extern]
```

## 4.13 src/pennfat/fat.c File Reference

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <time.h>
#include "fat.h"
#include "safe.h"
```

Include dependency graph for fat.c:

### Functions

- int [mem\\_idx](#) (uint16\_t \*fat, int block\_idx)
- int [get\\_free\\_block](#) (uint16\_t \*fat)
- void [delete\\_chain](#) (uint16\_t \*fat, int head)
- void [build\\_chain](#) (uint16\_t \*fat, int fs\_fd, int curr\_block, char \*data, int n\_bytes)
- int [fill\\_chain](#) (uint16\_t \*fat, int fs\_fd, int head, int chain\_size, char \*buffer, int buffer\_size)
- void [add\\_file](#) (uint16\_t \*fat, int fs\_fd, int dir\_head, const char \*filename)
- void [write\\_file](#) (uint16\_t \*fat, int fs\_fd, [point\\_t](#) location, [dir\\_entry\\_t](#) entry)
- void [read\\_chain](#) (uint16\_t \*fat, int fs\_fd, int head, char \*buffer, int chain\_bytes)
- bool [find\\_file](#) (uint16\_t \*fat, int fs\_fd, int dir\_head, const char \*filename, [point\\_t](#) \*loc, [dir\\_entry\\_t](#) \*ret)
- bool [valid\\_filename](#) (char \*str)
- void [fs\\_getmeta](#) (uint16\_t \*fat, int fs\_fd, int \*n\_blocks, int \*block\_size)
- int [fs\\_mount](#) (char \*fs\_name, uint16\_t \*\*fat)
- void [fs\\_unmount](#) (uint16\_t \*\*fat, int fs\_fd)
- bool [fs\\_touch](#) (uint16\_t \*fat, int fs\_fd, const char \*target)
- bool [fs\\_mv](#) (uint16\_t \*fat, int fs\_fd, const char \*old\_name, const char \*new\_name)
- bool [fs\\_mark\\_deleted](#) (uint16\_t \*fat, int fs\_fd, const char \*target)
- bool [fs\\_rm](#) (uint16\_t \*fat, int fs\_fd, const char \*target)
- char \* [fs\\_cat](#) (uint16\_t \*fat, int fs\_fd, int input\_mode, int output\_mode, char \*input\_str, char \*input\_files[], char \*output\_file)
- bool [fs\\_cp](#) (uint16\_t \*fat, int fs\_fd, const char \*source, const char \*dest)
- bool [fs\\_cp\\_mode](#) (uint16\_t \*fat, int fs\_fd, const char \*source, const char \*dest, bool host\_in, bool host\_out)
- void [fs\\_ls\\_single](#) ([dir\\_entry\\_t](#) \*entry)
- void [fs\\_ls](#) (uint16\_t \*fat, int fs\_fd)
- uint8\_t [fs\\_chmod](#) (uint16\_t \*fat, int fs\_fd, const char \*target, uint8\_t permissions)

### Variables

- const int [DIR\\_ENTRY\\_SIZE](#) = 64
- const int [ROOTDIR](#) = 1
- const int [DEFAULT\\_PERMISSIONS](#) = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH
- const int [LASTBLOCK](#) = 0xFFFF
- const int [BITS\\_PER\\_BYTE](#) = 8
- const int [BYTE\\_SIZE](#) = 1 << [BITS\\_PER\\_BYTE](#)
- const int [FILETYPE\\_UNKNOWN](#) = 0
- const int [FILETYPE\\_FILE](#) = 1

- const int FILETYPE\_DIRECTORY = 2
- const int FILETYPE\_LINK = 4
- const int FILENAME\_ENDDIR = 0
- const int FILENAME\_DEL\_UNUSED = 1
- const int FILENAME\_DEL\_INUSE = 2
- const int FILEPERM\_NONE = 0
- const int FILEPERM\_RD = 0b100
- const int FILEPERM\_WR = 0b010
- const int FILEPERM\_EX = 0b001

## 4.13.1 Function Documentation

### 4.13.1.1 add\_file()

```
void add_file (
    uint16_t * fat,
    int fs_fd,
    int dir_head,
    const char * filename )
```

add a new empty file to the directory, allocating new blocks as necessary

#### Parameters

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>dirhead</i>	the first index of the directory chain (ROOTDIR or 1 for the root dir)
<i>filename</i>	the file to add

#### Returns

none

### 4.13.1.2 build\_chain()

```
void build_chain (
    uint16_t * fat,
    int fs_fd,
    int curr_block,
    char * data,
    int n_bytes )
```

allocates data as a FAT chain

**Parameters**

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>curr_block</i>	first index of the chain to build
<i>data</i>	what to copy to memory
<i>n_bytes</i>	length of <i>data</i>

**Returns**

none

**4.13.1.3 delete\_chain()**

```
void delete_chain (
    uint16_t * fat,
    int head )
```

traverse down the fat chain, marking them all as deleted

**Parameters**

<i>fat</i>	filesystem
<i>head</i>	the first index of the chain

**Returns**

none

**4.13.1.4 fill\_chain()**

```
int fill_chain (
    uint16_t * fat,
    int fs_fd,
    int head,
    int chain_size,
    char * buffer,
    int buffer_size )
```

fill a FAT chain with data

**Parameters**

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>head</i>	the first index of the chain
<i>buffer</i>	what to read from
<i>buffer_size</i>	length of <i>buffer</i>



**Returns**

the number of bytes written

**4.13.1.5 find\_file()**

```
bool find_file (
    uint16_t * fat,
    int fs_fd,
    int dir_head,
    const char * filename,
    point_t * loc,
    dir_entry_t * ret )
```

Finds a file or directory in the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>dir_head</i>	Index of the first block in the directory.
<i>filename</i>	Name of the file or directory to find.
<i>loc</i>	Pointer to a point_t structure to store the location (block index and entry index).
<i>ret</i>	Pointer to a dir_entry_t structure to store the found directory entry.

**Returns**

Returns true if the file or directory is found; otherwise, false.

**4.13.1.6 fs\_cat()**

```
char* fs_cat (
    uint16_t * fat,
    int fs_fd,
    int input_mode,
    int output_mode,
    char * input_str,
    char * input_files[],
    char * output_file )
```

Concatenates input strings or files and outputs the result to the terminal or a file.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>input_mode</i>	Mode for input: 0 for input string, >0 for input files.
<i>output_mode</i>	Mode for output: 0 for stdout, 1 for file overwrite, 2 for file append.
<i>input_str</i>	Input string to concatenate (used when input_mode is 0).
<i>input_files</i>	Array of input file names (used when input_mode is >0).
<i>output_file</i>	Name of the output file (used when output_mode is 1 or 2).

**Returns**

Returns the concatenated output string if `output_mode` is 0; otherwise, returns NULL.

**4.13.1.7 fs\_chmod()**

```
uint8_t fs_chmod (
    uint16_t * fat,
    int fs_fd,
    const char * target,
    uint8_t permissions )
```

Changes the permissions of a file or directory in the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>target</i>	Name of the file or directory to change permissions.
<i>permissions</i>	New permissions to set.

**Returns**

Returns the old permissions before the change.

**4.13.1.8 fs\_cp()**

```
bool fs_cp (
    uint16_t * fat,
    int fs_fd,
    const char * source,
    const char * dest )
```

Copies a file or directory to a destination in the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>source</i>	Name of the source file or directory to copy.
<i>dest</i>	Name of the destination file or directory.

**Returns**

Returns true if the copy is successful; otherwise, false.

#### 4.13.1.9 fs\_cp\_mode()

```
bool fs_cp_mode (
    uint16_t * fat,
    int fs_fd,
    const char * source,
    const char * dest,
    bool host_in,
    bool host_out )
```

Copies a file or directory with specified input and output modes in the filesystem.

##### Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>source</i>	Name of the source file or directory to copy.
<i>dest</i>	Name of the destination file or directory.
<i>host_in</i>	Input mode: true for hostOS to PennFAT, false for PennFAT to PennFAT.
<i>host_out</i>	Output mode: true for PennFAT to hostOS, false for PennFAT to PennFAT.

##### Returns

Returns true if the copy is successful; otherwise, false.

#### 4.13.1.10 fs\_getmeta()

```
void fs_getmeta (
    uint16_t * fat,
    int fs_fd,
    int * n_blocks,
    int * block_size )
```

Retrieves metadata information from the filesystem.

##### Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>n_blocks</i>	Pointer to an integer to store the number of blocks in the filesystem.
<i>block_size</i>	Pointer to an integer to store the block size in bytes.

##### Returns

None.

#### 4.13.1.11 fs\_ls()

```
void fs_ls (
    uint16_t * fat,
    int fs_fd )
```

Displays information about all directory entries in the filesystem.

##### Parameters

<i>fat</i>	Pointer to FAT.
<i>fs↔ _fd</i>	File descriptor of the filesystem.

##### Returns

None.

#### 4.13.1.12 fs\_ls\_single()

```
void fs_ls_single (
    dir_entry_t * entry )
```

Displays information about a single directory entry.

##### Parameters

<i>entry</i>	Pointer to the directory entry.
--------------	---------------------------------

##### Returns

None.

#### 4.13.1.13 fs\_mark\_deleted()

```
bool fs_mark_deleted (
    uint16_t * fat,
    int fs_fd,
    const char * target )
```

Marks a file or directory as deleted in the filesystem.

##### Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>target</i>	Name of the file or directory to mark as deleted.

### Returns

Returns true if the file or directory is successfully marked as deleted; otherwise, false.

#### 4.13.1.14 fs\_mount()

```
int fs_mount (
    char * fs_name,
    uint16_t ** fat )
```

Mounts a file system.

### Parameters

<i>fs_name</i>	The name of the file system to mount.
<i>fat</i>	Pointer to FAT.

### Returns

Returns the file descriptor of the mounted file system on success. On failure, returns -1.

#### 4.13.1.15 fs\_mv()

```
bool fs_mv (
    uint16_t * fat,
    int fs_fd,
    const char * old_name,
    const char * new_name )
```

Moves or renames a file in the filesystem.

### Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>old_name</i>	Name of the file to move or rename.
<i>new_name</i>	New name or path for the file.

### Returns

Returns true if the file is successfully moved or renamed; otherwise, false.

#### 4.13.1.16 fs\_rm()

```
bool fs_rm (
    uint16_t * fat,
    int fs_fd,
    const char * target )
```

Removes (deletes) a file or directory from the filesystem.

##### Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>target</i>	Name of the file or directory to remove.

##### Returns

Returns true if the file or directory is successfully removed; otherwise, false.

#### 4.13.1.17 fs\_touch()

```
bool fs_touch (
    uint16_t * fat,
    int fs_fd,
    const char * target )
```

Creates or updates a file in the filesystem.

##### Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>target</i>	Name of the file to create or update.

##### Returns

Returns true if a new file is created; otherwise, false if an existing file is updated.

#### 4.13.1.18 fs\_unmount()

```
void fs_unmount (
    uint16_t ** fat,
    int fs_fd )
```

Unmounts a file system.

## Parameters

<i>fat</i>	Pointer to FAT.
<i>fs↔ _fd</i>	File descriptor of the filesystem.

## Returns

None.

**4.13.1.19 get\_free\_block()**

```
int get_free_block (
    uint16_t * fat )
```

search for an open block (where value is 0)

## Parameters

<i>fat</i>	filesystem
------------	------------

## Returns

the block index on success, 0 on failure

**4.13.1.20 mem\_idx()**

```
int mem_idx (
    uint16_t * fat,
    int block_idx )
```

get the address in memory of the specified block index

## Parameters

<i>fat</i>	filesystem
<i>block_idx</i>	block index

## Returns

the address/index in memory of `block_idx`

**4.13.1.21 read\_chain()**

```
void read_chain (
    uint16_t * fat,
    int fs_fd,
    int head,
    char * buffer,
    int chain_bytes )
```

Reads a FAT chain from the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>head</i>	Index of the first block in the chain.
<i>buffer</i>	Buffer to store the read data.
<i>chain_bytes</i>	Total number of bytes to read from the chain.

**Returns**

None.

**4.13.1.22 valid\_filename()**

```
bool valid_filename (
    char * str )
```

Checks if a string is a valid filename.

**Parameters**

<i>str</i>	The string to be checked.
------------	---------------------------

**Returns**

Returns true if the string is a valid filename; otherwise, false.

**4.13.1.23 write\_file()**

```
void write_file (
    uint16_t * fat,
    int fs_fd,
    point_t location,
    dir_entry_t entry )
```

seek and write a file block to memory



## Parameters

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>location</i>	point in memory
<i>entry</i>	the file block

## Returns

none

## 4.13.2 Variable Documentation

### 4.13.2.1 BITS\_PER\_BYTE

```
const int BITS_PER_BYTE = 8
```

### 4.13.2.2 BYTE\_SIZE

```
const int BYTE_SIZE = 1 << BITS_PER_BYTE
```

### 4.13.2.3 DEFAULT\_PERMISSIONS

```
const int DEFAULT_PERMISSIONS = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH
```

### 4.13.2.4 DIR\_ENTRY\_SIZE

```
const int DIR_ENTRY_SIZE = 64
```

### 4.13.2.5 FILENAME\_DEL\_INUSE

```
const int FILENAME_DEL_INUSE = 2
```

#### 4.13.2.6 FILENAME\_DEL\_UNUSED

```
const int FILENAME_DEL_UNUSED = 1
```

#### 4.13.2.7 FILENAME\_ENDDIR

```
const int FILENAME_ENDDIR = 0
```

#### 4.13.2.8 FILEPERM\_EX

```
const int FILEPERM_EX = 0b001
```

#### 4.13.2.9 FILEPERM\_NONE

```
const int FILEPERM_NONE = 0
```

#### 4.13.2.10 FILEPERM\_RD

```
const int FILEPERM_RD = 0b100
```

#### 4.13.2.11 FILEPERM\_WR

```
const int FILEPERM_WR = 0b010
```

#### 4.13.2.12 FILETYPE\_DIRECTORY

```
const int FILETYPE_DIRECTORY = 2
```

#### 4.13.2.13 FILETYPE\_FILE

```
const int FILETYPE_FILE = 1
```

#### 4.13.2.14 FILETYPE\_LINK

```
const int FILETYPE_LINK = 4
```

#### 4.13.2.15 FILETYPE\_UNKNOWN

```
const int FILETYPE_UNKNOWN = 0
```

#### 4.13.2.16 LASTBLOCK

```
const int LASTBLOCK = 0xFFFF
```

#### 4.13.2.17 ROOTDIR

```
const int ROOTDIR = 1
```

## 4.14 src/pennfat/fat.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for fat.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [directory\\_entry](#)
- struct [point](#)

### Macros

- #define [FAT\\_BLOCKS](#)(x) ((x) >> [BITS\\_PER\\_BYTE](#))
- #define [BLOCK\\_SIZE](#)(x) ([BYTE\\_SIZE](#) << ((x) & 0xFF))

### Typedefs

- typedef struct [directory\\_entry](#) [dir\\_entry\\_t](#)
- typedef struct [point](#) [point\\_t](#)

## Functions

- void [read\\_chain](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), int [head](#), char \*buffer, int chain\_bytes)
- bool [find\\_file](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), int dir\_head, const char \*filename, [point\\_t](#) \*loc, [dir\\_entry\\_t](#) \*ret)
- bool [valid\\_filename](#) (char \*str)
- void [fs\\_getmeta](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), int \*n\_blocks, int \*block\_size)
- int [fs\\_mount](#) (char \*fs\_name, uint16\_t \*\*[fat](#))
- void [fs\\_unmount](#) (uint16\_t \*\*[fat](#), int [fs\\_fd](#))
- bool [fs\\_touch](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), const char \*target)
- bool [fs\\_mv](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), const char \*old\_name, const char \*new\_name)
- bool [fs\\_rm](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), const char \*target)
- bool [fs\\_mark\\_deleted](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), const char \*target)
- char \* [fs\\_cat](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), int input\_mode, int output\_mode, char \*input\_str, char \*input\_files[], char \*output\_file)
- bool [fs\\_cp](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), const char \*source, const char \*dest)
- bool [fs\\_cp\\_mode](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), const char \*source, const char \*dest, bool host\_in, bool host\_out)
- void [fs\\_ls\\_single](#) ([dir\\_entry\\_t](#) \*entry)
- void [fs\\_ls](#) (uint16\_t \*[fat](#), int [fs\\_fd](#))
- uint8\_t [fs\\_chmod](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), const char \*target, uint8\_t permissions)

## Variables

- const int [DIR\\_ENTRY\\_SIZE](#)
- const int [ROOTDIR](#)
- const int [DEFAULT\\_PERMISSIONS](#)
- const int [LASTBLOCK](#)
- const int [BITS\\_PER\\_BYTE](#)
- const int [BYTE\\_SIZE](#)
- const int [FILETYPE\\_UNKNOWN](#)
- const int [FILETYPE\\_FILE](#)
- const int [FILETYPE\\_DIRECTORY](#)
- const int [FILETYPE\\_LINK](#)
- const int [FILENAME\\_ENDDIR](#)
- const int [FILENAME\\_DEL\\_UNUSED](#)
- const int [FILENAME\\_DEL\\_INUSE](#)
- const int [FILEPERM\\_NONE](#)
- const int [FILEPERM\\_RD](#)
- const int [FILEPERM\\_WR](#)
- const int [FILEPERM\\_EX](#)

### 4.14.1 Macro Definition Documentation

#### 4.14.1.1 BLOCK\_SIZE

```
#define BLOCK_SIZE(  
    x ) (BYTE_SIZE << ((x) & 0xFF))
```

### 4.14.1.2 FAT\_BLOCKS

```
#define FAT_BLOCKS(
    x ) ((x) >> BITS_PER_BYTE)
```

## 4.14.2 Typedef Documentation

### 4.14.2.1 dir\_entry\_t

```
typedef struct directory_entry dir_entry_t
```

### 4.14.2.2 point\_t

```
typedef struct point point_t
```

## 4.14.3 Function Documentation

### 4.14.3.1 find\_file()

```
bool find_file (
    uint16_t * fat,
    int fs_fd,
    int dir_head,
    const char * filename,
    point_t * loc,
    dir_entry_t * ret )
```

search for a filename in the directory use NULL for `loc` and `ret` to simply check if the file exists

#### Parameters

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>dir_head</i>	the first index of the directory chain (ROOTDIR or 1 for the root dir)
<i>filename</i>	the file to search for
<i>loc</i>	will be set to the block ( <code>loc.first</code> ) & entry number ( <code>loc.second</code> ) of the file, if the file is found
<i>ret</i>	the directory entry of the file, if the file is found

**Returns**

`true` if the file is found, `false` otherwise

Finds a file or directory in the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>dir_head</i>	Index of the first block in the directory.
<i>filename</i>	Name of the file or directory to find.
<i>loc</i>	Pointer to a <code>point_t</code> structure to store the location (block index and entry index).
<i>ret</i>	Pointer to a <code>dir_entry_t</code> structure to store the found directory entry.

**Returns**

Returns `true` if the file or directory is found; otherwise, `false`.

**4.14.3.2 fs\_cat()**

```
char* fs_cat (
    uint16_t * fat,
    int fs_fd,
    int input_mode,
    int output_mode,
    char * input_str,
    char * input_files[],
    char * output_file )
```

cat a number of files or the `input_str`, output to buffer or another file

**Parameters**

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>input_mode</i>	0 for <code>stdin</code> , otherwise the number of files in <code>input_files</code>
<i>output_mode</i>	0 for <code>stdout</code> , 1 for overwrite, 2 for append
<i>input_str</i>	if <code>input_mode</code> is 0, the input
<i>input_files</i>	if <code>input_mode</code> is not 0, the files to concatenate
<i>output_file</i>	if <code>output_mode</code> is not 0, the file to output to

**Returns**

if `output_mode` is 0, the output buffer

Concatenates input strings or files and outputs the result to the terminal or a file.

## Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>input_mode</i>	Mode for input: 0 for input string, >0 for input files.
<i>output_mode</i>	Mode for output: 0 for stdout, 1 for file overwrite, 2 for file append.
<i>input_str</i>	Input string to concatenate (used when <i>input_mode</i> is 0).
<i>input_files</i>	Array of input file names (used when <i>input_mode</i> is >0).
<i>output_file</i>	Name of the output file (used when <i>output_mode</i> is 1 or 2).

## Returns

Returns the concatenated output string if *output\_mode* is 0; otherwise, returns NULL.

## 4.14.3.3 fs\_chmod()

```
uint8_t fs_chmod (
    uint16_t * fat,
    int fs_fd,
    const char * target,
    uint8_t permissions )
```

change permissions

## Parameters

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>target</i>	the file to change permissions of
<i>permissions</i>	the new permissions

## Returns

the old permissions

Changes the permissions of a file or directory in the filesystem.

## Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>target</i>	Name of the file or directory to change permissions.
<i>permissions</i>	New permissions to set.

**Returns**

Returns the old permissions before the change.

**4.14.3.4 fs\_cp()**

```
bool fs_cp (
    uint16_t * fat,
    int fs_fd,
    const char * source,
    const char * dest )
```

copy a file

**Parameters**

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>source</i>	the file to copy from
<i>dest</i>	the file to copy to; created if necessary, or overwritten if not

**Returns**

true on success, false if source was not found

Copies a file or directory to a destination in the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>source</i>	Name of the source file or directory to copy.
<i>dest</i>	Name of the destination file or directory.

**Returns**

Returns true if the copy is successful; otherwise, false.

**4.14.3.5 fs\_cp\_mode()**

```
bool fs_cp_mode (
    uint16_t * fat,
    int fs_fd,
    const char * source,
    const char * dest,
```



```
bool host_in,  
bool host_out )
```

copy a file (may input/output with host OS)

**Parameters**

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>source</i>	the file to copy from
<i>dest</i>	the file to copy to; created if necessary, or overwritten if not
<i>host_in</i>	if <code>true</code> , input file from host OS
<i>host_out</i>	if <code>true</code> , output file to host OS

**Returns**

`true` on success, `false` if `source` was not found

Copies a file or directory with specified input and output modes in the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>source</i>	Name of the source file or directory to copy.
<i>dest</i>	Name of the destination file or directory.
<i>host_in</i>	Input mode: <code>true</code> for hostOS to PennFAT, <code>false</code> for PennFAT to PennFAT.
<i>host_out</i>	Output mode: <code>true</code> for PennFAT to hostOS, <code>false</code> for PennFAT to PennFAT.

**Returns**

Returns `true` if the copy is successful; otherwise, `false`.

**4.14.3.6 fs\_getmeta()**

```
void fs_getmeta (
    uint16_t * fat,
    int fs_fd,
    int * n_blocks,
    int * block_size )
```

get the metadata of a filesystem

**Parameters**

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>n_blocks</i>	set to the number of blocks in FAT
<i>block_size</i>	set to the size of a block

**Returns**

none

Retrieves metadata information from the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>n_blocks</i>	Pointer to an integer to store the number of blocks in the filesystem.
<i>block_size</i>	Pointer to an integer to store the block size in bytes.

**Returns**

None.

**4.14.3.7 fs\_ls()**

```
void fs_ls (
    uint16_t * fat,
    int fs_fd )
```

list directory

**Parameters**

<i>fat</i>	filesystem
<i>fs↔ _fd</i>	filesystem file descriptor

**Returns**

none

Displays information about all directory entries in the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs↔ _fd</i>	File descriptor of the filesystem.

**Returns**

None.

#### 4.14.3.8 fs\_ls\_single()

```
void fs_ls_single (
    dir_entry_t * entry )
```

list information for a single file

##### Parameters

<i>entry</i>	the directory entry for the file
--------------	----------------------------------

##### Returns

none

Displays information about a single directory entry.

##### Parameters

<i>entry</i>	Pointer to the directory entry.
--------------	---------------------------------

##### Returns

None.

#### 4.14.3.9 fs\_mark\_deleted()

```
bool fs_mark_deleted (
    uint16_t * fat,
    int fs_fd,
    const char * target )
```

mark a file as deleted with `FILENAME_DEL_INUSE, -2`; it is still in use by another process, but shouldn't be able to be accessed anew

##### Parameters

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>target</i>	the file to mark

##### Returns

`true` if the file was marked as deleted, `false` otherwise

Marks a file or directory as deleted in the filesystem.

## Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>target</i>	Name of the file or directory to mark as deleted.

## Returns

Returns true if the file or directory is successfully marked as deleted; otherwise, false.

**4.14.3.10 fs\_mount()**

```
int fs_mount (
    char * fs_name,
    uint16_t ** fat )
```

mount a filesystem & map fat to memory

## Parameters

<i>fs_name</i>	filesystem filename
<i>fat</i>	set to the filesytem

## Returns

the filesystem file descriptor (*fs\_fd*)

Mounts a file system.

## Parameters

<i>fs_name</i>	The name of the file system to mount.
<i>fat</i>	Pointer to FAT.

## Returns

Returns the file descriptor of the mounted file system on success. On failure, returns -1.

**4.14.3.11 fs\_mv()**

```
bool fs_mv (
    uint16_t * fat,
    int fs_fd,
    const char * old_name,
    const char * new_name )
```

rename a file

**Parameters**

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>old_name</i>	the file to rename
<i>new_name</i>	the new name for <i>old_name</i>

**Returns**

`true` if the file is renamed (and found), `false` otherwise

Moves or renames a file in the filesystem.

**Parameters**

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>old_name</i>	Name of the file to move or rename.
<i>new_name</i>	New name or path for the file.

**Returns**

Returns `true` if the file is successfully moved or renamed; otherwise, `false`.

**4.14.3.12 fs\_rm()**

```
bool fs_rm (
    uint16_t * fat,
    int fs_fd,
    const char * target )
```

delete a file

**Parameters**

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>target</i>	the file to delete

**Returns**

`true` if the file was deleted, `false` otherwise

Removes (deletes) a file or directory from the filesystem.

## Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>target</i>	Name of the file or directory to remove.

## Returns

Returns true if the file or directory is successfully removed; otherwise, false.

**4.14.3.13 fs\_touch()**

```
bool fs_touch (
    uint16_t * fat,
    int fs_fd,
    const char * target )
```

touch a file; if it exists, update the timestamp, otherwise create the file

## Parameters

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>target</i>	the target filename

## Returns

true if the file is created, false otherwise

Creates or updates a file in the filesystem.

## Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>target</i>	Name of the file to create or update.

## Returns

Returns true if a new file is created; otherwise, false if an existing file is updated.

**4.14.3.14 fs\_unmount()**

```
void fs_unmount (
    uint16_t ** fat,
    int fs_fd )
```

unmount a filesystem

#### Parameters

<i>fat</i>	pointer to the filesystem; will be unmapped from memory
<i>fs↔ _fd</i>	filesystem file descriptor; will be closed

#### Returns

none

Unmounts a file system.

#### Parameters

<i>fat</i>	Pointer to FAT.
<i>fs↔ _fd</i>	File descriptor of the filesystem.

#### Returns

None.

#### 4.14.3.15 read\_chain()

```
void read_chain (
    uint16_t * fat,
    int fs_fd,
    int head,
    char * buffer,
    int chain_bytes )
```

read a FAT chain

#### Parameters

<i>fat</i>	filesystem
<i>fs_fd</i>	filesystem file descriptor
<i>head</i>	the first index of the chain
<i>buffer</i>	what to read into; assume this has enough space
<i>chain_bytes</i>	length of the chain

#### Returns

none

Reads a FAT chain from the filesystem.



## Parameters

<i>fat</i>	Pointer to FAT.
<i>fs_fd</i>	File descriptor of the filesystem.
<i>head</i>	Index of the first block in the chain.
<i>buffer</i>	Buffer to store the read data.
<i>chain_bytes</i>	Total number of bytes to read from the chain.

## Returns

None.

### 4.14.3.16 valid\_filename()

```
bool valid_filename (
    char * str )
```

check whether a filename is valid

## Parameters

<i>str</i>	filename
------------	----------

## Returns

`true` if `str` is POSIX-valid (consists of [A-Za-z0-9.\_-]) and 32 bytes null-terminated (31 characters), `false` & print otherwise

Checks if a string is a valid filename.

## Parameters

<i>str</i>	The string to be checked.
------------	---------------------------

## Returns

Returns true if the string is a valid filename; otherwise, false.

## 4.14.4 Variable Documentation

### 4.14.4.1 BITS\_PER\_BYTE

```
const int BITS_PER_BYTE [extern]
```

#### 4.14.4.2 BYTE\_SIZE

```
const int BYTE_SIZE [extern]
```

#### 4.14.4.3 DEFAULT\_PERMISSIONS

```
const int DEFAULT_PERMISSIONS [extern]
```

#### 4.14.4.4 DIR\_ENTRY\_SIZE

```
const int DIR_ENTRY_SIZE [extern]
```

#### 4.14.4.5 FILENAME\_DEL\_INUSE

```
const int FILENAME_DEL_INUSE [extern]
```

#### 4.14.4.6 FILENAME\_DEL\_UNUSED

```
const int FILENAME_DEL_UNUSED [extern]
```

#### 4.14.4.7 FILENAME\_ENDDIR

```
const int FILENAME_ENDDIR [extern]
```

#### 4.14.4.8 FILEPERM\_EX

```
const int FILEPERM_EX [extern]
```

#### 4.14.4.9 FILEPERM\_NONE

```
const int FILEPERM_NONE [extern]
```

#### 4.14.4.10 FILEPERM\_RD

```
const int FILEPERM_RD [extern]
```

#### 4.14.4.11 FILEPERM\_WR

```
const int FILEPERM_WR [extern]
```

#### 4.14.4.12 FILETYPE\_DIRECTORY

```
const int FILETYPE_DIRECTORY [extern]
```

#### 4.14.4.13 FILETYPE\_FILE

```
const int FILETYPE_FILE [extern]
```

#### 4.14.4.14 FILETYPE\_LINK

```
const int FILETYPE_LINK [extern]
```

#### 4.14.4.15 FILETYPE\_UNKNOWN

```
const int FILETYPE_UNKNOWN [extern]
```

#### 4.14.4.16 LASTBLOCK

```
const int LASTBLOCK [extern]
```

#### 4.14.4.17 ROOTDIR

```
const int ROOTDIR [extern]
```

## 4.15 src/pennfat/pennfat.c File Reference

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <time.h>
#include <unistd.h>
#include <limits.h>
#include <errno.h>
#include "fat.h"
#include "safe.h"
#include "../util/parser.h"
#include "../util/util.h"
Include dependency graph for pennfat.c:
```

### Macros

- #define [CONTINUE](#) {free(command); continue;}

### Functions

- bool [correct\\_argc](#) (int expected, int actual)
- bool [valid\\_fs\\_mounted](#) (int [fs\\_fd](#))
- bool [all\\_files\\_exist](#) (uint16\_t \*[fat](#), int [fs\\_fd](#), char \*argv[], int first\_file\_arg, int last\_file\_arg)
- int [main](#) (int argc, char \*argv[])

### 4.15.1 Macro Definition Documentation

#### 4.15.1.1 CONTINUE

```
#define CONTINUE {free(command); continue;}
```

### 4.15.2 Function Documentation

#### 4.15.2.1 all\_files\_exist()

```
bool all_files_exist (
    uint16_t * fat,
    int fs_fd,
    char * argv[],
    int first_file_arg,
    int last_file_arg )
```

#### 4.15.2.2 correct\_argc()

```
bool correct_argc (
    int expected,
    int actual )
```

#### 4.15.2.3 main()

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

#### 4.15.2.4 valid\_fs\_mounted()

```
bool valid_fs_mounted (
    int fs_fd )
```

## 4.16 src/pennfat/safe.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/types.h>
#include "safe.h"
Include dependency graph for safe.c:
```

## Functions

- int [safe\\_open](#) (const char \*pathname, int flags, mode\_t mode)
- void [safe\\_close](#) (int fd)
- int [safe\\_read](#) (int fd, void \*buf, size\_t count)
- void [safe\\_write](#) (int fd, const void \*buf, size\_t count)
- off\_t [safe\\_lseek](#) (int fd, off\_t offset, int whence)
- void [safe\\_msync](#) (void \*addr, size\_t length, int flags)
- void \* [safe\\_mmap](#) (void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset)
- void [safe\\_munmap](#) (void \*addr, size\_t length)

### 4.16.1 Function Documentation

#### 4.16.1.1 [safe\\_close\(\)](#)

```
void safe_close (  
    int fd )
```

Closes a file descriptor with error handling.

##### Parameters

<i>fd</i>	The file descriptor to close.
-----------	-------------------------------

##### Returns

None.

#### 4.16.1.2 [safe\\_lseek\(\)](#)

```
off_t safe_lseek (  
    int fd,  
    off_t offset,  
    int whence )
```

Performs a seek operation on a file descriptor with error handling.

##### Parameters

<i>fd</i>	The file descriptor to seek.
<i>offset</i>	The offset to move the file pointer.
<i>whence</i>	The reference point for the offset.

**Returns**

Returns the resulting offset location if successful; otherwise, exits with an error message.

**4.16.1.3 safe\_mmap()**

```
void* safe_mmap (
    void * addr,
    size_t length,
    int prot,
    int flags,
    int fd,
    off_t offset )
```

Maps files or devices into memory with error handling.

**Parameters**

<i>addr</i>	The desired starting address for the mapping.
<i>length</i>	The length of the mapping.
<i>prot</i>	The desired memory protection of the mapping.
<i>flags</i>	The type of the mapping.
<i>fd</i>	The file descriptor of the file to map.
<i>offset</i>	The offset within the file to start the mapping.

**Returns**

The starting address of the mapped region.

**4.16.1.4 safe\_msync()**

```
void safe_msync (
    void * addr,
    size_t length,
    int flags )
```

Synchronizes changes to a file mapping with error handling.

**Parameters**

<i>addr</i>	The starting address of the file mapping.
<i>length</i>	The length of the file mapping.
<i>flags</i>	Flags specifying the type of synchronization.

**Returns**

None.

**4.16.1.5 safe\_munmap()**

```
void safe_munmap (
    void * addr,
    size_t length )
```

Unmaps files or devices from memory with error handling.

**Parameters**

<i>addr</i>	The starting address of the mapped region.
<i>length</i>	The length of the mapped region.

**4.16.1.6 safe\_open()**

```
int safe_open (
    const char * pathname,
    int flags,
    mode_t mode )
```

Opens a file with error handling.

**Parameters**

<i>pathname</i>	The path of the file to open.
<i>flags</i>	Flags specifying the mode of opening.
<i>mode</i>	The permissions to set if the file is created.

**Returns**

Returns the file descriptor if successful; otherwise, exits with an error message.

**4.16.1.7 safe\_read()**

```
int safe_read (
    int fd,
    void * buf,
    size_t count )
```

Reads data from a file descriptor with error handling.



## Parameters

<i>fd</i>	The file descriptor to read from.
<i>buf</i>	The buffer to store the read data.
<i>count</i>	The number of bytes to read.

## Returns

Returns the number of bytes read if successful; otherwise, exits with an error message.

4.16.1.8 **safe\_write()**

```
void safe_write (
    int fd,
    const void * buf,
    size_t count )
```

Writes data to a file descriptor with error handling.

## Parameters

<i>fd</i>	The file descriptor to write to.
<i>buf</i>	The buffer containing the data to write.
<i>count</i>	The number of bytes to write.

## Returns

None.

4.17 **src/pennfat/safe.h File Reference**

This graph shows which files directly or indirectly include this file:

## Functions

- int [safe\\_open](#) (const char \*, int, mode\_t)
- void [safe\\_close](#) (int)
- int [safe\\_read](#) (int, void \*, size\_t)
- void [safe\\_write](#) (int, const void \*, size\_t)
- off\_t [safe\\_lseek](#) (int fd, off\_t offset, int whence)
- void [safe\\_msync](#) (void \*addr, size\_t length, int flags)
- void \* [safe\\_mmap](#) (void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset)
- void [safe\\_munmap](#) (void \*addr, size\_t length)

## 4.17.1 Function Documentation

### 4.17.1.1 `safe_close()`

```
void safe_close (
    int fd )
```

Closes a file descriptor with error handling.

#### Parameters

<i>fd</i>	The file descriptor to close.
-----------	-------------------------------

#### Returns

None.

### 4.17.1.2 `safe_lseek()`

```
off_t safe_lseek (
    int fd,
    off_t offset,
    int whence )
```

Performs a seek operation on a file descriptor with error handling.

#### Parameters

<i>fd</i>	The file descriptor to seek.
<i>offset</i>	The offset to move the file pointer.
<i>whence</i>	The reference point for the offset.

#### Returns

Returns the resulting offset location if successful; otherwise, exits with an error message.

### 4.17.1.3 `safe_mmap()`

```
void* safe_mmap (
    void * addr,
    size_t length,
```

```
int prot,  
int flags,  
int fd,  
off_t offset )
```

Maps files or devices into memory with error handling.

#### Parameters

<i>addr</i>	The desired starting address for the mapping.
<i>length</i>	The length of the mapping.
<i>prot</i>	The desired memory protection of the mapping.
<i>flags</i>	The type of the mapping.
<i>fd</i>	The file descriptor of the file to map.
<i>offset</i>	The offset within the file to start the mapping.

#### Returns

The starting address of the mapped region.

#### 4.17.1.4 safe\_msync()

```
void safe_msync (  
    void * addr,  
    size_t length,  
    int flags )
```

Synchronizes changes to a file mapping with error handling.

#### Parameters

<i>addr</i>	The starting address of the file mapping.
<i>length</i>	The length of the file mapping.
<i>flags</i>	Flags specifying the type of synchronization.

#### Returns

None.

#### 4.17.1.5 safe\_munmap()

```
void safe_munmap (  
    void * addr,  
    size_t length )
```

Unmaps files or devices from memory with error handling.

## Parameters

<i>addr</i>	The starting address of the mapped region.
<i>length</i>	The length of the mapped region.

**4.17.1.6 safe\_open()**

```
int safe_open (
    const char * pathname,
    int flags,
    mode_t mode )
```

Opens a file with error handling.

## Parameters

<i>pathname</i>	The path of the file to open.
<i>flags</i>	Flags specifying the mode of opening.
<i>mode</i>	The permissions to set if the file is created.

## Returns

Returns the file descriptor if successful; otherwise, exits with an error message.

**4.17.1.7 safe\_read()**

```
int safe_read (
    int fd,
    void * buf,
    size_t count )
```

Reads data from a file descriptor with error handling.

## Parameters

<i>fd</i>	The file descriptor to read from.
<i>buf</i>	The buffer to store the read data.
<i>count</i>	The number of bytes to read.

## Returns

Returns the number of bytes read if successful; otherwise, exits with an error message.

#### 4.17.1.8 safe\_write()

```
void safe_write (
    int fd,
    const void * buf,
    size_t count )
```

Writes data to a file descriptor with error handling.

##### Parameters

<i>fd</i>	The file descriptor to write to.
<i>buf</i>	The buffer containing the data to write.
<i>count</i>	The number of bytes to write.

##### Returns

None.

## 4.18 src/pennos.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "util/globals.h"
#include "shell/pennos-shell.h"
#include "pennfat/fat.h"
#include "kernel/PCB.h"
#include "kernel/scheduler.h"
#include "logger/logger.h"
Include dependency graph for pennos.c:
```

### Functions

- int [main](#) (int argc, char \*argv[])

#### 4.18.1 Function Documentation

##### 4.18.1.1 main()

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

Entry point for PennOS. Initializes the logger, filesystem, and spawns the shell process.

**Parameters**

<i>argc</i>	The number of command-line arguments.
<i>argv</i>	An array of command-line arguments.

**Returns**

Returns 1 if the number of command-line arguments is less than 2.

**4.19 src/shell/job-list.c File Reference**

```
#include <stdlib.h>
#include <stdio.h>
#include "job-list.h"
#include "../util/util.h"
#include "../util/p-errno.h"
#include "../filesystem/filesystem.h"
#include "../kernel/puser-functions.h"
Include dependency graph for job-list.c:
```

**Functions**

- [job\\_t \\* job\\_find\\_by\\_jobid](#) ([job\\_t \\*\\*head](#), int target\_job\_id)
- int [job\\_get\\_last](#) ([job\\_t \\*\\*head](#))
- [job\\_t \\* jobs\\_push](#) ([job\\_t \\*\\*head](#), int job\_id, int pid, int [stop\\_order](#))
- void [jobs\\_insert](#) ([job\\_t \\*\\*head](#), [job\\_t \\*new\\_job](#))
- [job\\_t \\* jobs\\_remove](#) ([job\\_t \\*\\*head](#), int target\_job\_id)
- void [job\\_print](#) ([job\\_t \\*job](#))

**4.19.1 Function Documentation****4.19.1.1 job\_find\_by\_jobid()**

```
job_t* job_find_by_jobid (
    job_t ** head,
    int target_job_id )
```

Finds a job by its job ID in the linked list of jobs.

**Parameters**

<i>head</i>	A pointer to the head of the linked list of jobs.
<i>target_job_id</i>	The job ID to search for.

**Returns**

Returns a pointer to the found job or NULL if not found.

**4.19.1.2 job\_get\_last()**

```
int job_get_last (
    job_t ** head )
```

Gets the job ID of the last job in the linked list.

**Parameters**

<i>head</i>	A pointer to the head of the linked list of jobs.
-------------	---

**Returns**

Returns the job ID of the last job or -1 if the list is empty.

**4.19.1.3 job\_print()**

```
void job_print (
    job_t * job )
```

Prints information about a job.

**Parameters**

<i>job</i>	A pointer to the job structure to be printed.
------------	---

**4.19.1.4 jobs\_insert()**

```
void jobs_insert (
    job_t ** head,
    job_t * new_job )
```

Inserts a new job into the linked list in ascending order based on job ID.

**Parameters**

<i>head</i>	A pointer to the head of the linked list of jobs.
<i>new_job</i>	A pointer to the new job to be inserted.

#### 4.19.1.5 jobs\_push()

```
job_t* jobs_push (
    job_t ** head,
    int job_id,
    int pid,
    int stop_order )
```

Pushes a new job to the end of the linked list.

##### Parameters

<i>head</i>	A pointer to the head of the linked list of jobs.
<i>job_id</i>	The job ID of the new job.
<i>pid</i>	The process ID associated with the job.
<i>stop_order</i>	The order in which the job should be stopped.

##### Returns

Returns a pointer to the newly added job.

#### 4.19.1.6 jobs\_remove()

```
job_t* jobs_remove (
    job_t ** head,
    int target_job_id )
```

Removes a job with the specified job ID from the linked list.

##### Parameters

<i>head</i>	A pointer to the head of the linked list of jobs.
<i>target_job_id</i>	The job ID of the job to be removed.

##### Returns

A pointer to the removed job, or NULL if not found.

## 4.20 src/shell/job-list.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for job-list.h: This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [job](#)

## Macros

- `#define` [NOT\\_STOPPED](#) -1

## Typedefs

- typedef struct [job](#) [job\\_t](#)

## Functions

- [job\\_t](#) \* [job\\_find\\_by\\_jobid](#) ([job\\_t](#) \*\*[head](#), int [target\\_job\\_id](#))
- int [job\\_get\\_last](#) ([job\\_t](#) \*\*[head](#))
- [job\\_t](#) \* [jobs\\_push](#) ([job\\_t](#) \*\*[head](#), int [job\\_id](#), int [pid](#), int [stop\\_order](#))
- void [jobs\\_insert](#) ([job\\_t](#) \*\*[head](#), [job\\_t](#) \*[new\\_job](#))
- [job\\_t](#) \* [jobs\\_remove](#) ([job\\_t](#) \*\*[head](#), int [target\\_job\\_id](#))
- void [job\\_print](#) ([job\\_t](#) \*[job](#))

### 4.20.1 Macro Definition Documentation

#### 4.20.1.1 NOT\_STOPPED

```
#define NOT_STOPPED -1
```

### 4.20.2 Typedef Documentation

#### 4.20.2.1 job\_t

```
typedef struct job job_t
```

### 4.20.3 Function Documentation

#### 4.20.3.1 job\_find\_by\_jobid()

```
job_t* job_find_by_jobid (  
    job_t ** head,  
    int target_job_id )
```

find a job struct by [job\\_id](#)

**Parameters**

<i>head</i>	the job linkedlist
<i>target_job↔ _id</i>	the job_id to find

**Returns**

the job struct if it was found, or `NULL` otherwise

Finds a job by its job ID in the linked list of jobs.

**Parameters**

<i>head</i>	A pointer to the head of the linked list of jobs.
<i>target_job↔ _id</i>	The job ID to search for.

**Returns**

Returns a pointer to the found job or `NULL` if not found.

**4.20.3.2 job\_get\_last()**

```
int job_get_last (
    job_t ** head )
```

get last process (most recent background process)

**Parameters**

<i>head</i>	the job linkedlist
-------------	--------------------

**Returns**

the last job\_id, or `-1` if the list is empty

Gets the job ID of the last job in the linked list.

**Parameters**

<i>head</i>	A pointer to the head of the linked list of jobs.
-------------	---

**Returns**

Returns the job ID of the last job or `-1` if the list is empty.

### 4.20.3.3 job\_print()

```
void job_print (
    job_t * job )
```

DEBUG: print job info

#### Parameters

<i>job</i>	the job
------------	---------

Prints information about a job.

#### Parameters

<i>job</i>	A pointer to the job structure to be printed.
------------	---

### 4.20.3.4 jobs\_insert()

```
void jobs_insert (
    job_t ** head,
    job_t * new_job )
```

insert a job struct at the correct position to maintain sortedness by job\_id

#### Parameters

<i>head</i>	the job linkedlist
<i>new_job</i>	the job struct

#### Returns

none

Inserts a new job into the linked list in ascending order based on job ID.

#### Parameters

<i>head</i>	A pointer to the head of the linked list of jobs.
<i>new_job</i>	A pointer to the new job to be inserted.

#### 4.20.3.5 jobs\_push()

```
job_t* jobs_push (
    job_t ** head,
    int job_id,
    int pid,
    int stop_order )
```

create a job and push it onto the linkedlist

##### Parameters

<i>head</i>	the job linkedlist
<i>job_id</i>	the new job_id
<i>pid</i>	the job's pid mapping
<i>stop_order</i>	-1 or NOT_STOPPED if foreground process; main stop_order if background process

##### Returns

the created job struct

Pushes a new job to the end of the linked list.

##### Parameters

<i>head</i>	A pointer to the head of the linked list of jobs.
<i>job_id</i>	The job ID of the new job.
<i>pid</i>	The process ID associated with the job.
<i>stop_order</i>	The order in which the job should be stopped.

##### Returns

Returns a pointer to the newly added job.

#### 4.20.3.6 jobs\_remove()

```
job_t* jobs_remove (
    job_t ** head,
    int target_job_id )
```

remove the specified job and return it

##### Parameters

<i>head</i>	the job linkedlist
<i>target_job_id</i>	the job_id to find

**Returns**

the job struct if it was found, or `NULL` otherwise

Removes a job with the specified job ID from the linked list.

**Parameters**

<i>head</i>	A pointer to the head of the linked list of jobs.
<i>target_job↔ _id</i>	The job ID of the job to be removed.

**Returns**

A pointer to the removed job, or `NULL` if not found.

## 4.21 src/shell/pennos-shell.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "../util/parser.h"
#include "../util/util.h"
#include "../util/globals.h"
#include "../util/p-errno.h"
#include "../util/safe-user.h"
#include "../filesystem/filesystem.h"
#include "../kernel/puser-functions.h"
#include "../logger/logger.h"
#include "job-list.h"
```

Include dependency graph for pennos-shell.c:

**Macros**

- `#define PROMPT "$ "`
- `#define CONTINUE {free(command); continue;}`

**Functions**

- void `stop_handler` (int signal)
- void `term_handler` (int signal)
- void `shell_cat` (int argc, char \*argv[])
- void `shell_sleep` (int argc, char \*argv[])
- void `shell_busy` (int argc, char \*argv[])
- void `shell_echo` (int argc, char \*argv[])
- void `shell_ls` (int argc, char \*argv[])
- void `shell_touch` (int argc, char \*argv[])
- void `shell_mv` (int argc, char \*argv[])
- void `shell_cp` (int argc, char \*argv[])
- void `shell_rm` (int argc, char \*argv[])
- void `shell_chmod` (int argc, char \*argv[])
- void `shell_ps` (int argc, char \*argv[])

- void [shell\\_kill](#) (int argc, char \*argv[])
- void [zombie\\_child](#) ()
- void [shell\\_zombify](#) (int argc, char \*argv[])
- void [orphan\\_child](#) ()
- void [shell\\_orphanify](#) (int argc, char \*argv[])
- void [cull\\_helper](#) (job\_t \*job)
- void [cull\\_background](#) ()
- void [debug\\_print\\_jobs](#) ()
- void [empty\\_reaped](#) ()
- int [spawn\\_command](#) (char \*command[], int in\_fd, int out\_fd)
- int [execute\\_command](#) (char \*command[], const char \*in\_filename, const char \*out\_filename, bool append↵\_mode)
- int [execute\\_script](#) (char \*command\_in[], const char \*in\_filename, const char \*out\_filename, bool append↵\_mode)
- void [pennos\\_shell](#) (int argc, char \*argv[])

## Variables

- const int [MAX\\_ARGUMENTS](#) = 1000
- const char \* [MAN\\_COMMANDS](#)
- job\_t \* [foreground\\_job](#) = NULL
- int [current\\_jobid](#)
- int [jobid\\_ctr](#) = 1
- int [stop\\_order](#) = 1
- job\_t \* [head](#) = NULL
- job\_t \*\* [background](#) = &[head](#)
- int [n\\_reaped](#) = 0
- job\_t \* [reaped](#) [1000]
- bool [stop\\_trigger](#) = false

## 4.21.1 Macro Definition Documentation

### 4.21.1.1 CONTINUE

```
#define CONTINUE {free(command); continue;}
```

### 4.21.1.2 PROMPT

```
#define PROMPT "$ "
```

## 4.21.2 Function Documentation

#### 4.21.2.1 `cull_background()`

```
void cull_background ( )
```

#### 4.21.2.2 `cull_helper()`

```
void cull_helper (
    job_t * job )
```

#### 4.21.2.3 `debug_print_jobs()`

```
void debug_print_jobs ( )
```

#### 4.21.2.4 `empty_reaped()`

```
void empty_reaped ( )
```

#### 4.21.2.5 `execute_command()`

```
int execute_command (
    char * command[],
    const char * in_filename,
    const char * out_filename,
    bool append_mode )
```

start a (single) independent pennOS process & return the pid; does not wait for the process

##### Parameters

<i>command</i>	the command to parse & execute
<i>in_filename</i>	NULL for F_STDIN, otherwise the input file
<i>out_filename</i>	NULL for F_STDOUT, otherwise the output file
<i>append_mode</i>	whether to append to out_filename; ignore if out_filename == NULL

##### Returns

the pid of the spawned process

#### 4.21.2.6 execute\_script()

```
int execute_script (
    char * command_in[],
    const char * in_filename,
    const char * out_filename,
    bool append_mode )
```

#### 4.21.2.7 orphan\_child()

```
void orphan_child ( )
```

#### 4.21.2.8 pennos\_shell()

```
void pennos_shell (
    int argc,
    char * argv[] )
```

shell function to be called upon startup

##### Parameters

<i>argc</i>	TODO: does this need any args?
<i>argv</i>	TODO

##### Returns

nonzero on error, 1 on success

#### 4.21.2.9 shell\_busy()

```
void shell_busy (
    int argc,
    char * argv[] )
```

#### 4.21.2.10 shell\_cat()

```
void shell_cat (
    int argc,
    char * argv[] )
```



#### 4.21.2.11 shell\_chmod()

```
void shell_chmod (
    int argc,
    char * argv[] )
```

#### 4.21.2.12 shell\_cp()

```
void shell_cp (
    int argc,
    char * argv[] )
```

#### 4.21.2.13 shell\_echo()

```
void shell_echo (
    int argc,
    char * argv[] )
```

#### 4.21.2.14 shell\_kill()

```
void shell_kill (
    int argc,
    char * argv[] )
```

#### 4.21.2.15 shell\_ls()

```
void shell_ls (
    int argc,
    char * argv[] )
```

#### 4.21.2.16 shell\_mv()

```
void shell_mv (
    int argc,
    char * argv[] )
```

**4.21.2.17 shell\_orphanify()**

```
void shell_orphanify (
    int argc,
    char * argv[] )
```

**4.21.2.18 shell\_ps()**

```
void shell_ps (
    int argc,
    char * argv[] )
```

**4.21.2.19 shell\_rm()**

```
void shell_rm (
    int argc,
    char * argv[] )
```

**4.21.2.20 shell\_sleep()**

```
void shell_sleep (
    int argc,
    char * argv[] )
```

**4.21.2.21 shell\_touch()**

```
void shell_touch (
    int argc,
    char * argv[] )
```

**4.21.2.22 shell\_zombify()**

```
void shell_zombify (
    int argc,
    char * argv[] )
```

#### 4.21.2.23 `spawn_command()`

```
int spawn_command (
    char * command[],
    int in_fd,
    int out_fd )
```

#### 4.21.2.24 `stop_handler()`

```
void stop_handler (
    int signal )
```

#### 4.21.2.25 `term_handler()`

```
void term_handler (
    int signal )
```

#### 4.21.2.26 `zombie_child()`

```
void zombie_child ( )
```

### 4.21.3 Variable Documentation

#### 4.21.3.1 `background`

```
job_t** background = &head
```

#### 4.21.3.2 `current_jobid`

```
int current_jobid
```



#### 4.21.3.8 n\_reaped

```
int n_reaped = 0
```

#### 4.21.3.9 reaped

```
job_t* reaped[1000]
```

#### 4.21.3.10 stop\_order

```
int stop_order = 1
```

#### 4.21.3.11 stop\_trigger

```
bool stop_trigger = false
```

## 4.22 src/shell/pennos-shell.h File Reference

This graph shows which files directly or indirectly include this file:

### Functions

- void [pennos\\_shell](#) (int argc, char \*argv[])

### 4.22.1 Function Documentation

#### 4.22.1.1 pennos\_shell()

```
void pennos_shell (  
    int argc,  
    char * argv[] )
```

shell function to be called upon startup

**Parameters**

<i>argc</i>	TODO: does this need any args?
<i>argv</i>	TODO

**Returns**

nonzero on error, 1 on success

## 4.23 src/util/globals.c File Reference

```
#include <stdint.h>  
Include dependency graph for globals.c:
```

**Variables**

- int [fs\\_fd](#)
- uint16\_t \* [fat](#)

### 4.23.1 Variable Documentation

#### 4.23.1.1 fat

```
uint16_t* fat
```

#### 4.23.1.2 fs\_fd

```
int fs_fd
```

## 4.24 src/util/globals.h File Reference

```
#include <stdio.h>  
#include <stdint.h>  
#include <unistd.h>  
Include dependency graph for globals.h: This graph shows which files directly or indirectly include this file:
```

## Macros

- `#define S_SIGSTOP 123 /* a thread receiving this signal should be stopped */`
- `#define S_SIGCONT 456 /* a thread receiving this signal should be continued */`
- `#define S_SIGTERM 789 /* a thread receiving this signal should be terminated */`
- `#define S_SIGCHLD 812 /* a thread receiving this signal should be terminated */`
- `#define T_RUNNING 111`
- `#define T_STOPPED 222`
- `#define T_BLOCKED 333`
- `#define T_ZOMBIED 444`

## Variables

- `int fs_fd`
- `uint16_t * fat`

### 4.24.1 Macro Definition Documentation

#### 4.24.1.1 S\_SIGCHLD

```
#define S_SIGCHLD 812 /* a thread receiving this signal should be terminated */
```

#### 4.24.1.2 S\_SIGCONT

```
#define S_SIGCONT 456 /* a thread receiving this signal should be continued */
```

#### 4.24.1.3 S\_SIGSTOP

```
#define S_SIGSTOP 123 /* a thread receiving this signal should be stopped */
```

#### 4.24.1.4 S\_SIGTERM

```
#define S_SIGTERM 789 /* a thread receiving this signal should be terminated */
```

#### 4.24.1.5 T\_BLOCKED

```
#define T_BLOCKED 333
```

#### 4.24.1.6 T\_RUNNING

```
#define T_RUNNING 111
```

#### 4.24.1.7 T\_STOPPED

```
#define T_STOPPED 222
```

#### 4.24.1.8 T\_ZOMBIED

```
#define T_ZOMBIED 444
```

### 4.24.2 Variable Documentation

#### 4.24.2.1 fat

```
uint16_t* fat [extern]
```

#### 4.24.2.2 fs\_fd

```
int fs_fd [extern]
```

## 4.25 src/util/p-errno.c File Reference

```
#include <stdio.h>
#include "p-errno.h"
#include "util.h"
#include "../filesystem/filesystem.h"
Include dependency graph for p-errno.c:
```



## Functions

- char \* `err_string` (int `errno`)
- void `p_perror` (const char \*`message`)

## Variables

- int `ERRNO` = `ERR_NONE`

### 4.25.1 Function Documentation

#### 4.25.1.1 `err_string()`

```
char* err_string (  
    int errno )
```

Returns a string description for the given error code.

##### Parameters

<i>errno</i>	The error code.
--------------	-----------------

##### Returns

A string description for the error code.

#### 4.25.1.2 `p_perror()`

```
void p_perror (  
    const char * message )
```

Print an error message along with the corresponding error string.

##### Parameters

<i>message</i>	The additional message to print.
----------------	----------------------------------

### 4.25.2 Variable Documentation

#### 4.25.2.1 ERRNO

```
int ERRNO = ERR\_NONE
```

## 4.26 src/util/p-errno.h File Reference

This graph shows which files directly or indirectly include this file:

### Macros

- `#define ERR\_NONE 0`
- `#define ERR\_FS\_FILE\_NOT\_FOUND 1000`
- `#define ERR\_F\_OPEN\_INVALID\_PERMS 1010`
- `#define ERR\_F\_OPEN\_WRITE\_INUSE 1011`
- `#define ERR\_F\_OPEN\_CREATE\_READ 1012`
- `#define ERR\_F\_OPEN\_INVALID\_MODE 1013`
- `#define ERR\_F\_READ\_TERM\_OUT 1020`
- `#define ERR\_F\_WRITE\_TERM\_IN 1030`
- `#define ERR\_F\_WRITE\_ROONLY 1031`
- `#define ERR\_F\_CLOSE\_TERMINAL 1040`
- `#define ERR\_F\_UNLINK\_NOT\_FOUND 1050`
- `#define ERR\_F\_LSEEK\_TERMINAL 1060`
- `#define ERR\_F\_LSEEK\_OOB 1061`
- `#define ERR\_P\_SPAWN\_NULL\_CHILD 2000`
- `#define ERR\_P\_SPAWN\_NULL\_STACK 2001`
- `#define ERR\_P\_WAITPID\_NULL\_CHILD 2010`
- `#define ERR\_P\_KILL\_NULL\_PROCESS 2020`
- `#define ERR\_P\_NICE\_NULL\_PROCESS 2030`

### Functions

- void [p\\_perror](#) (const char \*message)

#### 4.26.1 Macro Definition Documentation

##### 4.26.1.1 ERR\_F\_CLOSE\_TERMINAL

```
#define ERR\_F\_CLOSE\_TERMINAL 1040
```

##### 4.26.1.2 ERR\_F\_LSEEK\_OOB

```
#define ERR\_F\_LSEEK\_OOB 1061
```

#### 4.26.1.3 ERR\_F\_LSEEK\_TERMINAL

```
#define ERR_F_LSEEK_TERMINAL 1060
```

#### 4.26.1.4 ERR\_F\_OPEN\_CREATE\_READ

```
#define ERR_F_OPEN_CREATE_READ 1012
```

#### 4.26.1.5 ERR\_F\_OPEN\_INVALID\_MODE

```
#define ERR_F_OPEN_INVALID_MODE 1013
```

#### 4.26.1.6 ERR\_F\_OPEN\_INVALID\_PERMS

```
#define ERR_F_OPEN_INVALID_PERMS 1010
```

#### 4.26.1.7 ERR\_F\_OPEN\_WRITE\_INUSE

```
#define ERR_F_OPEN_WRITE_INUSE 1011
```

#### 4.26.1.8 ERR\_F\_READ\_TERM\_OUT

```
#define ERR_F_READ_TERM_OUT 1020
```

#### 4.26.1.9 ERR\_F\_UNLINK\_NOT\_FOUND

```
#define ERR_F_UNLINK_NOT_FOUND 1050
```

#### 4.26.1.10 ERR\_F\_WRITE\_ONLY

```
#define ERR_F_WRITE_ONLY 1031
```

#### 4.26.1.11 ERR\_F\_WRITE\_TERM\_IN

```
#define ERR_F_WRITE_TERM_IN 1030
```

#### 4.26.1.12 ERR\_FS\_FILE\_NOT\_FOUND

```
#define ERR_FS_FILE_NOT_FOUND 1000
```

#### 4.26.1.13 ERR\_NONE

```
#define ERR_NONE 0
```

#### 4.26.1.14 ERR\_P\_KILL\_NULL\_PROCESS

```
#define ERR_P_KILL_NULL_PROCESS 2020
```

#### 4.26.1.15 ERR\_P\_NICE\_NULL\_PROCESS

```
#define ERR_P_NICE_NULL_PROCESS 2030
```

#### 4.26.1.16 ERR\_P\_SPAWN\_NULL\_CHILD

```
#define ERR_P_SPAWN_NULL_CHILD 2000
```

#### 4.26.1.17 ERR\_P\_SPAWN\_NULL\_STACK

```
#define ERR_P_SPAWN_NULL_STACK 2001
```

#### 4.26.1.18 ERR\_P\_WAITPID\_NULL\_CHILD

```
#define ERR_P_WAITPID_NULL_CHILD 2010
```

### 4.26.2 Function Documentation

#### 4.26.2.1 p\_perror()

```
void p_perror (
    const char * message )
```

print a message describing the meaning of the value of ERRNO

## Parameters

<i>message</i>	the message to print
----------------	----------------------

## Returns

none

Print an error message along with the corresponding error string.

## Parameters

<i>message</i>	The additional message to print.
----------------	----------------------------------

## 4.27 src/util/parser.h File Reference

```
#include <stddef.h>
#include <stdbool.h>
```

Include dependency graph for parser.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [parsed\\_command](#)

### Macros

- #define [UNEXPECTED\\_FILE\\_INPUT](#) 1
- #define [UNEXPECTED\\_FILE\\_OUTPUT](#) 2
- #define [UNEXPECTED\\_PIPELINE](#) 3
- #define [UNEXPECTED\\_AMPERSAND](#) 4
- #define [EXPECT\\_INPUT\\_FILENAME](#) 5
- #define [EXPECT\\_OUTPUT\\_FILENAME](#) 6
- #define [EXPECT\\_COMMANDS](#) 7

### Functions

- int [parse\\_command](#) (const char \*cmd\_line, struct [parsed\\_command](#) \*\*result)
- void [print\\_parsed\\_command](#) (const struct [parsed\\_command](#) \*cmd)

#### 4.27.1 Macro Definition Documentation

#### 4.27.1.1 EXPECT\_COMMANDS

```
#define EXPECT_COMMANDS 7
```

#### 4.27.1.2 EXPECT\_INPUT\_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

#### 4.27.1.3 EXPECT\_OUTPUT\_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

#### 4.27.1.4 UNEXPECTED\_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

#### 4.27.1.5 UNEXPECTED\_FILE\_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

#### 4.27.1.6 UNEXPECTED\_FILE\_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

#### 4.27.1.7 UNEXPECTED\_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

### 4.27.2 Function Documentation

#### 4.27.2.1 parse\_command()

```
int parse_command (
    const char * cmd_line,
    struct parsed_command ** result )
```

Arguments: `cmd_line`: a null-terminated string that is the command line result: a non-null pointer to a `struct parsed_command *`

Return value (int): an error code which can be, 0: parser finished succesfully -1: parser encountered a system call error 1-7: parser specific error, see error type above

This function will parse the given `cmd_line` and store the parsed information into a `struct parsed_command`. The memory needed for the struct will be allocated by this function, and the pointer to the memory will be stored into the given `*result`.

You can directly use the result in system calls. See demo for more information.

If the function returns a successful value (0), a `struct parsed_command` is guaranteed to be allocated and stored in the given `*result`. It is the caller's responsibility to free the given pointer using `free(3)`.

Otherwise, no `struct parsed_command` is allocated and `*result` is unchanged. If a system call error (-1) is returned, the caller can use `errno(3)` or `perror(3)` to gain more information about the error.

#### 4.27.2.2 print\_parsed\_command()

```
void print_parsed_command (
    const struct parsed_command * cmd )
```

## 4.28 src/util/safe-user.c File Reference

```
#include <stdint.h>
#include "safe-user.h"
#include "p-errno.h"
#include "../kernel/puser-functions.h"
#include "../filesystem/filesystem.h"
```

Include dependency graph for safe-user.c:

### Functions

- int `safe_f_open` (const char \*fname, int mode)
- int `safe_f_read` (int fd, int n, char \*buf)
- int `safe_f_write` (int fd, const char \*str, int n)
- int `safe_f_close` (int fd)
- int `safe_f_unlink` (const char \*fname)
- int `safe_f_lseek` (int fd, int offset, int whence)
- int `safe_f_print` (const char \*str)

#### 4.28.1 Function Documentation

#### 4.28.1.1 `safe_f_close()`

```
int safe_f_close (
    int fd )
```

#### 4.28.1.2 `safe_f_lseek()`

```
int safe_f_lseek (
    int fd,
    int offset,
    int whence )
```

#### 4.28.1.3 `safe_f_open()`

```
int safe_f_open (
    const char * fname,
    int mode )
```

#### 4.28.1.4 `safe_f_print()`

```
int safe_f_print (
    const char * str )
```

#### 4.28.1.5 `safe_f_read()`

```
int safe_f_read (
    int fd,
    int n,
    char * buf )
```

#### 4.28.1.6 `safe_f_unlink()`

```
int safe_f_unlink (
    const char * fname )
```



#### 4.28.1.7 `safe_f_write()`

```
int safe_f_write (
    int fd,
    const char * str,
    int n )
```

## 4.29 src/util/safe-user.h File Reference

```
#include <stdint.h>
```

Include dependency graph for safe-user.h: This graph shows which files directly or indirectly include this file:

### Functions

- int [safe\\_f\\_open](#) (const char \**fname*, int *mode*)
- int [safe\\_f\\_read](#) (int *fd*, int *n*, char \**buf*)
- int [safe\\_f\\_write](#) (int *fd*, const char \**str*, int *n*)
- int [safe\\_f\\_close](#) (int *fd*)
- int [safe\\_f\\_unlink](#) (const char \**fname*)
- int [safe\\_f\\_lseek](#) (int *fd*, int *offset*, int *whence*)
- int [safe\\_f\\_print](#) (const char \**str*)

### 4.29.1 Function Documentation

#### 4.29.1.1 `safe_f_close()`

```
int safe_f_close (
    int fd )
```

#### 4.29.1.2 `safe_f_lseek()`

```
int safe_f_lseek (
    int fd,
    int offset,
    int whence )
```

#### 4.29.1.3 `safe_f_open()`

```
int safe_f_open (
    const char * fname,
    int mode )
```

#### 4.29.1.4 `safe_f_print()`

```
int safe_f_print (
    const char * str )
```

#### 4.29.1.5 `safe_f_read()`

```
int safe_f_read (
    int fd,
    int n,
    char * buf )
```

#### 4.29.1.6 `safe_f_unlink()`

```
int safe_f_unlink (
    const char * fname )
```

#### 4.29.1.7 `safe_f_write()`

```
int safe_f_write (
    int fd,
    const char * str,
    int n )
```

## 4.30 `src/util/util.c` File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
Include dependency graph for util.c:
```

### Functions

- int [get\\_argc](#) (char \*argv[])
- void \* [safe\\_malloc](#) (size\_t size)
- void [safe\\_signal](#) (int signum, void(\*handler)(int))

### Variables

- const int [IOBUFFER\\_SIZE](#) = 10000
- const int [ERRBUFFER\\_SIZE](#) = 1000

## 4.30.1 Function Documentation

### 4.30.1.1 `get_argc()`

```
int get_argc (
    char * argv[] )
```

Get the number of arguments in an array of strings.

#### Parameters

<i>argv</i>	The array of strings.
-------------	-----------------------

#### Returns

The number of arguments.

### 4.30.1.2 `safe_malloc()`

```
void* safe_malloc (
    size_t size )
```

Allocate memory using malloc.

#### Parameters

<i>size</i>	The size of the memory to allocate.
-------------	-------------------------------------

#### Returns

A pointer to the allocated memory.

### 4.30.1.3 `safe_signal()`

```
void safe_signal (
    int signum,
    void(*) (int) handler )
```

Set a signal handler using signal.

**Parameters**

<i>signum</i>	The signal number.
<i>handler</i>	The signal handler function.

## 4.30.2 Variable Documentation

### 4.30.2.1 ERRBUFFER\_SIZE

```
const int ERRBUFFER_SIZE = 1000
```

### 4.30.2.2 IOBUFFER\_SIZE

```
const int IOBUFFER_SIZE = 10000
```

## 4.31 src/util/util.h File Reference

This graph shows which files directly or indirectly include this file:

### Macros

- `#define PRINT(x) {fprintf(stderr, "%d\n", x);}`
- `#define PRINTE {fprintf(stderr, "e\n");}`

### Functions

- `int get_argc (char *argv[])`
- `void * safe_malloc (size_t size)`
- `void safe_signal (int signum, void(*handler)(int))`

### Variables

- `const int IOBUFFER_SIZE`
- `const int ERRBUFFER_SIZE`

### 4.31.1 Macro Definition Documentation

#### 4.31.1.1 PRINT

```
#define PRINT(  
    x ) {fprintf(stderr, "%d\n", x);}
```

#### 4.31.1.2 PRINTE

```
#define PRINTE {fprintf(stderr, "e\n");}
```

### 4.31.2 Function Documentation

#### 4.31.2.1 get\_argc()

```
int get_argc (  
    char * argv[] )
```

Get the number of arguments in an array of strings.

##### Parameters

<i>argv</i>	The array of strings.
-------------	-----------------------

##### Returns

The number of arguments.

#### 4.31.2.2 safe\_malloc()

```
void* safe_malloc (  
    size_t size )
```

Allocate memory using malloc.

##### Parameters

<i>size</i>	The size of the memory to allocate.
-------------	-------------------------------------

##### Returns

A pointer to the allocated memory.

#### 4.31.2.3 `safe_signal()`

```
void safe_signal (
    int signum,
    void(*) (int) handler )
```

Set a signal handler using `signal`.

##### Parameters

<i>signum</i>	The signal number.
<i>handler</i>	The signal handler function.

### 4.31.3 Variable Documentation

#### 4.31.3.1 `ERRBUFFER_SIZE`

```
const int ERRBUFFER_SIZE [extern]
```

#### 4.31.3.2 `IOBUFFER_SIZE`

```
const int IOBUFFER_SIZE [extern]
```

# Index

- [\\_BUFFER\\_](#)
    - [directory\\_entry, 5](#)
- [activeContext](#)
  - [scheduler.c, 64](#)
- [add\\_file](#)
  - [fat.c, 79](#)
- [addPCBToList](#)
  - [PCB.h, 48](#)
- [alarmHandler](#)
  - [scheduler.c, 62](#)
- [all\\_files\\_exist](#)
  - [pennfat.c, 108](#)
- [background](#)
  - [pennos-shell.c, 131](#)
- [BETWEEN\\_INCL](#)
  - [filesystem.c, 16](#)
- [BITS\\_PER\\_BYTE](#)
  - [fat.c, 89](#)
  - [fat.h, 105](#)
- [BLOCK\\_SIZE](#)
  - [fat.h, 92](#)
- [build\\_chain](#)
  - [fat.c, 79](#)
- [BYTE\\_SIZE](#)
  - [fat.c, 89](#)
  - [fat.h, 105](#)
- [centisecond](#)
  - [scheduler.c, 64](#)
- [children](#)
  - [PCB, 11](#)
- [commands](#)
  - [parsed\\_command, 10](#)
- [context](#)
  - [PCB, 11](#)
- [CONTINUE](#)
  - [pennfat.c, 108](#)
  - [pennos-shell.c, 126](#)
- [correct\\_argc](#)
  - [pennfat.c, 109](#)
- [count\\_running](#)
  - [PCB.h, 49](#)
- [count\\_running\\_priority](#)
  - [PCB.h, 49](#)
- [create\\_file\\_entry](#)
  - [filesystem.c, 17](#)
- [create\\_fileptr](#)
  - [filesystem.c, 18](#)
- [createPCB](#)
  - [PCB.h, 49](#)
- [cull\\_background](#)
  - [pennos-shell.c, 126](#)
- [cull\\_helper](#)
  - [pennos-shell.c, 127](#)
- [current\\_jobid](#)
  - [pennos-shell.c, 131](#)
- [current\\_pcb](#)
  - [filesystem.c, 29](#)
  - [puser-functions.c, 57](#)
  - [puser-functions.h, 61](#)
- [debug\\_print\\_jobs](#)
  - [pennos-shell.c, 127](#)
- [DEFAULT\\_PERMISSIONS](#)
  - [fat.c, 89](#)
  - [fat.h, 106](#)
- [delete\\_chain](#)
  - [fat.c, 80](#)
- [delete\\_file\\_entry](#)
  - [filesystem.c, 18](#)
- [delete\\_fileptr](#)
  - [filesystem.c, 19](#)
- [DIR\\_ENTRY\\_SIZE](#)
  - [fat.c, 89](#)
  - [fat.h, 106](#)
- [dir\\_entry\\_t](#)
  - [fat.h, 93](#)
- [directory\\_entry, 5](#)
  - [\\_BUFFER\\_, 5](#)
  - [firstBlock, 5](#)
  - [mtime, 5](#)
  - [name, 6](#)
  - [perm, 6](#)
  - [size, 6](#)
  - [type, 6](#)
- [done](#)
  - [job, 9](#)
- [empty\\_reaped](#)
  - [pennos-shell.c, 127](#)
- [ERR\\_F\\_CLOSE\\_TERMINAL](#)
  - [p-errno.h, 138](#)
- [ERR\\_F\\_LSEEK\\_OOB](#)
  - [p-errno.h, 138](#)
- [ERR\\_F\\_LSEEK\\_TERMINAL](#)
  - [p-errno.h, 138](#)
- [ERR\\_F\\_OPEN\\_CREATE\\_READ](#)
  - [p-errno.h, 139](#)

ERR\_F\_OPEN\_INVALID\_MODE  
     p-errno.h, 139  
 ERR\_F\_OPEN\_INVALID\_PERMS  
     p-errno.h, 139  
 ERR\_F\_OPEN\_WRITE\_INUSE  
     p-errno.h, 139  
 ERR\_F\_READ\_TERM\_OUT  
     p-errno.h, 139  
 ERR\_F\_UNLINK\_NOT\_FOUND  
     p-errno.h, 139  
 ERR\_F\_WRITE\_RDONLY  
     p-errno.h, 139  
 ERR\_F\_WRITE\_TERM\_IN  
     p-errno.h, 139  
 ERR\_FS\_FILE\_NOT\_FOUND  
     p-errno.h, 140  
 ERR\_NONE  
     p-errno.h, 140  
 ERR\_P\_KILL\_NULL\_PROCESS  
     p-errno.h, 140  
 ERR\_P\_NICE\_NULL\_PROCESS  
     p-errno.h, 140  
 ERR\_P\_SPAWN\_NULL\_CHILD  
     p-errno.h, 140  
 ERR\_P\_SPAWN\_NULL\_STACK  
     p-errno.h, 140  
 ERR\_P\_WAITPID\_NULL\_CHILD  
     p-errno.h, 140  
 err\_string  
     p-errno.c, 137  
 ERRBUFFER\_SIZE  
     util.c, 148  
     util.h, 150  
 ERRNO  
     p-errno.c, 137  
 execute\_command  
     pennos-shell.c, 127  
 execute\_script  
     pennos-shell.c, 127  
 EXPECT\_COMMANDS  
     parser.h, 141  
 EXPECT\_INPUT\_FILENAME  
     parser.h, 142  
 EXPECT\_OUTPUT\_FILENAME  
     parser.h, 142  
  
 F\_APPEND  
     filesystem.h, 31  
 F\_CANREAD  
     filesystem.c, 17  
 F\_CANWRITE  
     filesystem.c, 17  
 f\_chmod  
     filesystem.c, 19  
     filesystem.h, 32  
 f\_close  
     filesystem.c, 19  
     filesystem.h, 33  
 f\_cp  
     filesystem.c, 20  
     filesystem.h, 33  
 F\_HASPERM  
     filesystem.c, 17  
 f\_isatty  
     filesystem.c, 20  
 f\_ls  
     filesystem.c, 20  
     filesystem.h, 34  
 f\_lseek  
     filesystem.c, 21  
     filesystem.h, 34  
 f\_mount  
     filesystem.c, 21  
     filesystem.h, 35  
 f\_mv  
     filesystem.c, 22  
     filesystem.h, 35  
 f\_open  
     filesystem.c, 22  
     filesystem.h, 36  
 f\_print  
     filesystem.c, 22  
     filesystem.h, 36  
 F\_READ  
     filesystem.h, 31  
 f\_read  
     filesystem.c, 23  
     filesystem.h, 38  
 f\_rm  
     filesystem.c, 23  
     filesystem.h, 39  
 F\_SEEK\_CURR  
     filesystem.h, 31  
 F\_SEEK\_END  
     filesystem.h, 31  
 F\_SEEK\_SET  
     filesystem.h, 31  
 F\_STDERR  
     filesystem.h, 31  
 F\_STDIN  
     filesystem.h, 31  
 F\_STDOUT  
     filesystem.h, 31  
 f\_touch  
     filesystem.c, 24  
     filesystem.h, 39  
 f\_unlink  
     filesystem.c, 24  
     filesystem.h, 39  
 f\_unmount  
     filesystem.c, 24  
     filesystem.h, 40  
 F\_WRITE  
     filesystem.h, 32  
 f\_write  
     filesystem.c, 25  
     filesystem.h, 40



- fat
  - globals.c, [134](#)
  - globals.h, [136](#)
- fat.c
  - add\_file, [79](#)
  - BITS\_PER\_BYTE, [89](#)
  - build\_chain, [79](#)
  - BYTE\_SIZE, [89](#)
  - DEFAULT\_PERMISSIONS, [89](#)
  - delete\_chain, [80](#)
  - DIR\_ENTRY\_SIZE, [89](#)
  - FILENAME\_DEL\_INUSE, [89](#)
  - FILENAME\_DEL\_UNUSED, [89](#)
  - FILENAME\_ENDDIR, [90](#)
  - FILEPERM\_EX, [90](#)
  - FILEPERM\_NONE, [90](#)
  - FILEPERM\_RD, [90](#)
  - FILEPERM\_WR, [90](#)
  - FILETYPE\_DIRECTORY, [90](#)
  - FILETYPE\_FILE, [90](#)
  - FILETYPE\_LINK, [90](#)
  - FILETYPE\_UNKNOWN, [91](#)
  - fill\_chain, [80](#)
  - find\_file, [81](#)
  - fs\_cat, [81](#)
  - fs\_chmod, [82](#)
  - fs\_cp, [82](#)
  - fs\_cp\_mode, [82](#)
  - fs\_getmeta, [83](#)
  - fs\_ls, [83](#)
  - fs\_ls\_single, [84](#)
  - fs\_mark\_deleted, [84](#)
  - fs\_mount, [85](#)
  - fs\_mv, [85](#)
  - fs\_rm, [85](#)
  - fs\_touch, [86](#)
  - fs\_unmount, [86](#)
  - get\_free\_block, [87](#)
  - LASTBLOCK, [91](#)
  - mem\_idx, [87](#)
  - read\_chain, [87](#)
  - ROOTDIR, [91](#)
  - valid\_filename, [88](#)
  - write\_file, [88](#)
- fat.h
  - BITS\_PER\_BYTE, [105](#)
  - BLOCK\_SIZE, [92](#)
  - BYTE\_SIZE, [105](#)
  - DEFAULT\_PERMISSIONS, [106](#)
  - DIR\_ENTRY\_SIZE, [106](#)
  - dir\_entry\_t, [93](#)
  - FAT\_BLOCKS, [92](#)
  - FILENAME\_DEL\_INUSE, [106](#)
  - FILENAME\_DEL\_UNUSED, [106](#)
  - FILENAME\_ENDDIR, [106](#)
  - FILEPERM\_EX, [106](#)
  - FILEPERM\_NONE, [106](#)
  - FILEPERM\_RD, [106](#)
  - FILEPERM\_WR, [107](#)
  - FILETYPE\_DIRECTORY, [107](#)
  - FILETYPE\_FILE, [107](#)
  - FILETYPE\_LINK, [107](#)
  - FILETYPE\_UNKNOWN, [107](#)
  - find\_file, [93](#)
  - fs\_cat, [94](#)
  - fs\_chmod, [95](#)
  - fs\_cp, [96](#)
  - fs\_cp\_mode, [96](#)
  - fs\_getmeta, [98](#)
  - fs\_ls, [99](#)
  - fs\_ls\_single, [99](#)
  - fs\_mark\_deleted, [100](#)
  - fs\_mount, [101](#)
  - fs\_mv, [101](#)
  - fs\_rm, [102](#)
  - fs\_touch, [103](#)
  - fs\_unmount, [103](#)
  - LASTBLOCK, [107](#)
  - point\_t, [93](#)
  - read\_chain, [104](#)
  - ROOTDIR, [107](#)
  - valid\_filename, [105](#)
- FAT\_BLOCKS
  - fat.h, [92](#)
- file, [6](#)
  - file\_id, [7](#)
  - filename, [7](#)
  - fileptr\_head, [7](#)
  - next, [7](#)
  - wr\_pid, [7](#)
- file\_id
  - file, [7](#)
- file\_t
  - filesystem.h, [32](#)
- fileDescriptors
  - PCB, [11](#)
- filename
  - file, [7](#)
- FILENAME\_DEL\_INUSE
  - fat.c, [89](#)
  - fat.h, [106](#)
- FILENAME\_DEL\_UNUSED
  - fat.c, [89](#)
  - fat.h, [106](#)
- FILENAME\_ENDDIR
  - fat.c, [90](#)
  - fat.h, [106](#)
- FILEPERM\_EX
  - fat.c, [90](#)
  - fat.h, [106](#)
- FILEPERM\_NONE
  - fat.c, [90](#)
  - fat.h, [106](#)
- FILEPERM\_RD
  - fat.c, [90](#)
  - fat.h, [106](#)

- FILEPERM\_WR
  - fat.c, [90](#)
  - fat.h, [107](#)
- fileptr, [7](#)
  - next, [8](#)
  - pid, [8](#)
  - ptr, [8](#)
- fileptr\_head
  - file, [7](#)
- fileptr\_t
  - filesystem.h, [32](#)
- filesystem.c
  - BETWEEN\_INCL, [16](#)
  - create\_file\_entry, [17](#)
  - create\_fileptr, [18](#)
  - current\_pcb, [29](#)
  - delete\_file\_entry, [18](#)
  - delete\_fileptr, [19](#)
  - F\_CANREAD, [17](#)
  - F\_CANWRITE, [17](#)
  - f\_chmod, [19](#)
  - f\_close, [19](#)
  - f\_cp, [20](#)
  - F\_HASPERM, [17](#)
  - f\_isatty, [20](#)
  - f\_ls, [20](#)
  - f\_lseek, [21](#)
  - f\_mount, [21](#)
  - f\_mv, [22](#)
  - f\_open, [22](#)
  - f\_print, [22](#)
  - f\_read, [23](#)
  - f\_rm, [23](#)
  - f\_touch, [24](#)
  - f\_unlink, [24](#)
  - f\_unmount, [24](#)
  - f\_write, [25](#)
  - find\_file\_entry, [25](#)
  - find\_file\_entry\_by\_file\_id, [26](#)
  - find\_file\_entry\_by\_filename, [26](#)
  - find\_unused\_fd, [26](#)
  - get\_fileptr, [26](#)
  - get\_fileptr\_ptr, [27](#)
  - is\_duplicate\_fd, [27](#)
  - MIN, [17](#)
  - next\_file\_id, [29](#)
  - open\_files, [29](#)
  - print\_fileptr\_pids\_all, [27](#)
  - process\_create\_fileptrs, [28](#)
  - process\_delete\_fileptrs, [28](#)
  - valid\_perm, [28](#)
- filesystem.h
  - F\_APPEND, [31](#)
  - f\_chmod, [32](#)
  - f\_close, [33](#)
  - f\_cp, [33](#)
  - f\_ls, [34](#)
  - f\_lseek, [34](#)
  - f\_mount, [35](#)
  - f\_mv, [35](#)
  - f\_open, [36](#)
  - f\_print, [36](#)
  - F\_READ, [31](#)
  - f\_read, [38](#)
  - f\_rm, [39](#)
  - F\_SEEK\_CURR, [31](#)
  - F\_SEEK\_END, [31](#)
  - F\_SEEK\_SET, [31](#)
  - F\_STDERR, [31](#)
  - F\_STDIN, [31](#)
  - F\_STDOUT, [31](#)
  - f\_touch, [39](#)
  - f\_unlink, [39](#)
  - f\_unmount, [40](#)
  - F\_WRITE, [32](#)
  - f\_write, [40](#)
  - file\_t, [32](#)
  - fileptr\_t, [32](#)
  - find\_file\_entry\_by\_file\_id, [41](#)
  - FPERM\_EXEC, [32](#)
  - FPERM\_READ, [32](#)
  - FPERM\_WRIT, [32](#)
  - print\_fileptr\_pids\_all, [41](#)
  - process\_create\_fileptrs, [41](#)
  - process\_delete\_fileptrs, [42](#)
- FILETYPE\_DIRECTORY
  - fat.c, [90](#)
  - fat.h, [107](#)
- FILETYPE\_FILE
  - fat.c, [90](#)
  - fat.h, [107](#)
- FILETYPE\_LINK
  - fat.c, [90](#)
  - fat.h, [107](#)
- FILETYPE\_UNKNOWN
  - fat.c, [91](#)
  - fat.h, [107](#)
- fill\_chain
  - fat.c, [80](#)
- find\_file
  - fat.c, [81](#)
  - fat.h, [93](#)
- find\_file\_entry
  - filesystem.c, [25](#)
- find\_file\_entry\_by\_file\_id
  - filesystem.c, [26](#)
  - filesystem.h, [41](#)
- find\_file\_entry\_by\_filename
  - filesystem.c, [26](#)
- find\_unused\_fd
  - filesystem.c, [26](#)
- findPCBByContext
  - PCB.h, [50](#)
- findPCBByPID
  - PCB.h, [50](#)
- first

- point, [13](#)
- firstBlock
  - directory\_entry, [5](#)
- foreground\_job
  - pennos-shell.c, [131](#)
- FPERM\_EXEC
  - filesystem.h, [32](#)
- FPERM\_READ
  - filesystem.h, [32](#)
- FPERM\_WRIT
  - filesystem.h, [32](#)
- freeStacks
  - scheduler.c, [62](#)
- fs\_cat
  - fat.c, [81](#)
  - fat.h, [94](#)
- fs\_chmod
  - fat.c, [82](#)
  - fat.h, [95](#)
- fs\_cp
  - fat.c, [82](#)
  - fat.h, [96](#)
- fs\_cp\_mode
  - fat.c, [82](#)
  - fat.h, [96](#)
- fs\_fd
  - globals.c, [134](#)
  - globals.h, [136](#)
- fs\_getmeta
  - fat.c, [83](#)
  - fat.h, [98](#)
- fs\_ls
  - fat.c, [83](#)
  - fat.h, [99](#)
- fs\_ls\_single
  - fat.c, [84](#)
  - fat.h, [99](#)
- fs\_mark\_deleted
  - fat.c, [84](#)
  - fat.h, [100](#)
- fs\_mount
  - fat.c, [85](#)
  - fat.h, [101](#)
- fs\_mv
  - fat.c, [85](#)
  - fat.h, [101](#)
- fs\_rm
  - fat.c, [85](#)
  - fat.h, [102](#)
- fs\_touch
  - fat.c, [86](#)
  - fat.h, [103](#)
- fs\_unmount
  - fat.c, [86](#)
  - fat.h, [103](#)
- get\_argc
  - util.c, [147](#)
  - util.h, [149](#)
- get\_fileptr
  - filesystem.c, [26](#)
- get\_fileptr\_ptr
  - filesystem.c, [27](#)
- get\_free\_block
  - fat.c, [87](#)
- getLength
  - PCB.h, [51](#)
- globals.c
  - fat, [134](#)
  - fs\_fd, [134](#)
- globals.h
  - fat, [136](#)
  - fs\_fd, [136](#)
  - S\_SIGCHLD, [135](#)
  - S\_SIGCONT, [135](#)
  - S\_SIGSTOP, [135](#)
  - S\_SIGTERM, [135](#)
  - T\_BLOCKED, [135](#)
  - T\_RUNNING, [136](#)
  - T\_STOPPED, [136](#)
  - T\_ZOMBIED, [136](#)
- head
  - pennos-shell.c, [132](#)
- init\_scheduler
  - scheduler.h, [65](#)
- IOBUFFER\_SIZE
  - util.c, [148](#)
  - util.h, [150](#)
- is\_background
  - parsed\_command, [10](#)
- is\_duplicate\_fd
  - filesystem.c, [27](#)
- is\_file\_append
  - parsed\_command, [10](#)
- job, [8](#)
  - done, [9](#)
  - job\_id, [9](#)
  - next, [9](#)
  - pid, [9](#)
  - stop\_order, [9](#)
- job-list.c
  - job\_find\_by\_jobid, [118](#)
  - job\_get\_last, [119](#)
  - job\_print, [119](#)
  - jobs\_insert, [119](#)
  - jobs\_push, [120](#)
  - jobs\_remove, [120](#)
- job-list.h
  - job\_find\_by\_jobid, [121](#)
  - job\_get\_last, [122](#)
  - job\_print, [123](#)
  - job\_t, [121](#)
  - jobs\_insert, [123](#)
  - jobs\_push, [123](#)
  - jobs\_remove, [124](#)

- NOT\_STOPPED, 121
- job\_find\_by\_jobid
  - job-list.c, 118
  - job-list.h, 121
- job\_get\_last
  - job-list.c, 119
  - job-list.h, 122
- job\_id
  - job, 9
- job\_print
  - job-list.c, 119
  - job-list.h, 123
- job\_t
  - job-list.h, 121
- jobid\_ctr
  - pennos-shell.c, 132
- jobs\_insert
  - job-list.c, 119
  - job-list.h, 123
- jobs\_push
  - job-list.c, 120
  - job-list.h, 123
- jobs\_remove
  - job-list.c, 120
  - job-list.h, 124
- k\_free
  - PCB.h, 51
- k\_process\_cleanup
  - kernel-functions.c, 43
  - kernel-functions.h, 44
- k\_process\_create
  - kernel-functions.c, 43
  - kernel-functions.h, 45
- k\_process\_deep\_cleanup
  - kernel-functions.c, 43
  - kernel-functions.h, 45
- k\_process\_kill
  - kernel-functions.c, 44
  - kernel-functions.h, 45
- kernel-functions.c
  - k\_process\_cleanup, 43
  - k\_process\_create, 43
  - k\_process\_deep\_cleanup, 43
  - k\_process\_kill, 44
- kernel-functions.h
  - k\_process\_cleanup, 44
  - k\_process\_create, 45
  - k\_process\_deep\_cleanup, 45
  - k\_process\_kill, 45
- LASTBLOCK
  - fat.c, 91
  - fat.h, 107
- log\_blocked\_event
  - logger.c, 67
  - logger.h, 72
- log\_continued\_event
  - logger.c, 67
- logger.h, 73
  - log\_create\_event
    - logger.c, 68
    - logger.h, 73
  - log\_exited\_event
    - logger.c, 68
    - logger.h, 73
  - log\_nice\_event
    - logger.c, 68
    - logger.h, 74
  - log\_orphan\_event
    - logger.c, 69
    - logger.h, 74
  - log\_schedule\_event
    - logger.c, 69
    - logger.h, 74
  - log\_signaled\_event
    - logger.c, 69
    - logger.h, 75
  - log\_stopped\_event
    - logger.c, 70
    - logger.h, 75
  - log\_unblocked\_event
    - logger.c, 70
    - logger.h, 75
  - log\_waited\_event
    - logger.c, 71
    - logger.h, 77
  - log\_zombie\_event
    - logger.c, 71
    - logger.h, 77
- logfile
  - logger.c, 71
  - logger.h, 77
- logger.c
  - log\_blocked\_event, 67
  - log\_continued\_event, 67
  - log\_create\_event, 68
  - log\_exited\_event, 68
  - log\_nice\_event, 68
  - log\_orphan\_event, 69
  - log\_schedule\_event, 69
  - log\_signaled\_event, 69
  - log\_stopped\_event, 70
  - log\_unblocked\_event, 70
  - log\_waited\_event, 71
  - log\_zombie\_event, 71
  - logfile, 71
- logger.h
  - log\_blocked\_event, 72
  - log\_continued\_event, 73
  - log\_create\_event, 73
  - log\_exited\_event, 73
  - log\_nice\_event, 74
  - log\_orphan\_event, 74
  - log\_schedule\_event, 74
  - log\_signaled\_event, 75
  - log\_stopped\_event, 75

- log\_unblocked\_event, 75
- log\_waited\_event, 77
- log\_zombie\_event, 77
- logfile, 77
- main
  - pennfat.c, 109
  - pennos.c, 117
- mainContext
  - scheduler.c, 64
- MAN\_COMMANDS
  - pennos-shell.c, 132
- MAX\_ARGUMENTS
  - pennos-shell.c, 132
- MAX\_FDS
  - PCB.h, 47
- mem\_idx
  - fat.c, 87
- MIN
  - filesystem.c, 17
- mtime
  - directory\_entry, 5
- n\_reaped
  - pennos-shell.c, 132
- name
  - directory\_entry, 6
  - PCB, 12
- next
  - file, 7
  - fileptr, 8
  - job, 9
  - PCB, 12
- next\_file\_id
  - filesystem.c, 29
- next\_pid
  - PCB.h, 53
- NOFILE
  - PCB.h, 47
- NOT\_STOPPED
  - job-list.h, 121
- num\_commands
  - parsed\_command, 10
- numChildren
  - PCB, 12
- open\_files
  - filesystem.c, 29
- orphan\_child
  - pennos-shell.c, 128
- p\_errno.c
  - err\_string, 137
  - ERRNO, 137
  - p\_perror, 137
- p\_errno.h
  - ERR\_F\_CLOSE\_TERMINAL, 138
  - ERR\_F\_LSEEK\_OOB, 138
  - ERR\_F\_LSEEK\_TERMINAL, 138
  - ERR\_F\_OPEN\_CREATE\_READ, 139
  - ERR\_F\_OPEN\_INVALID\_MODE, 139
  - ERR\_F\_OPEN\_INVALID\_PERMS, 139
  - ERR\_F\_OPEN\_WRITE\_INUSE, 139
  - ERR\_F\_READ\_TERM\_OUT, 139
  - ERR\_F\_UNLINK\_NOT\_FOUND, 139
  - ERR\_F\_WRITE\_RDONLY, 139
  - ERR\_F\_WRITE\_TERM\_IN, 139
  - ERR\_FS\_FILE\_NOT\_FOUND, 140
  - ERR\_NONE, 140
  - ERR\_P\_KILL\_NULL\_PROCESS, 140
  - ERR\_P\_NICE\_NULL\_PROCESS, 140
  - ERR\_P\_SPAWN\_NULL\_CHILD, 140
  - ERR\_P\_SPAWN\_NULL\_STACK, 140
  - ERR\_P\_WAITPID\_NULL\_CHILD, 140
  - p\_perror, 140
- p\_exit
  - puser-functions.c, 54
  - puser-functions.h, 58
- p\_kill
  - puser-functions.c, 54
  - puser-functions.h, 58
- p\_nice
  - puser-functions.c, 54
  - puser-functions.h, 58
- p\_perror
  - p-errno.c, 137
  - p-errno.h, 140
- p\_sleep
  - puser-functions.c, 55
  - puser-functions.h, 59
- p\_spawn
  - puser-functions.c, 55
  - puser-functions.h, 59
- p\_waitpid
  - puser-functions.c, 55
  - puser-functions.h, 60
- parent\_pid
  - PCB, 12
- parse\_command
  - parser.h, 142
- parsed\_command, 9
  - commands, 10
  - is\_background, 10
  - is\_file\_append, 10
  - num\_commands, 10
  - stdin\_file, 10
  - stdout\_file, 10
- parser.h
  - EXPECT\_COMMANDS, 141
  - EXPECT\_INPUT\_FILENAME, 142
  - EXPECT\_OUTPUT\_FILENAME, 142
  - parse\_command, 142
  - print\_parsed\_command, 143
  - UNEXPECTED\_AMPERSAND, 142
  - UNEXPECTED\_FILE\_INPUT, 142
  - UNEXPECTED\_FILE\_OUTPUT, 142
  - UNEXPECTED\_PIPELINE, 142

- PCB, 11
  - children, 11
  - context, 11
  - fileDescriptors, 11
  - name, 12
  - next, 12
  - numChildren, 12
  - parent\_pid, 12
  - PCB.h, 48
  - pid, 12
  - priority, 12
  - status, 12
- PCB.h
  - addPCBToList, 48
  - count\_running, 49
  - count\_running\_priority, 49
  - createPCB, 49
  - findPCBByContext, 50
  - findPCBByPID, 50
  - getLength, 51
  - k\_free, 51
  - MAX\_FDS, 47
  - next\_pid, 53
  - NOFILE, 47
  - PCB, 48
  - pcb\_list, 53
  - removePCBFromList, 52
  - STACKSIZE, 47
  - STDERR\_ID, 47
  - STDIN\_ID, 47
  - STDOUT\_ID, 48
- pcb\_list
  - PCB.h, 53
- pennfat.c
  - all\_files\_exist, 108
  - CONTINUE, 108
  - correct\_argc, 109
  - main, 109
  - valid\_fs\_mounted, 109
- pennos-shell.c
  - background, 131
  - CONTINUE, 126
  - cull\_background, 126
  - cull\_helper, 127
  - current\_jobid, 131
  - debug\_print\_jobs, 127
  - empty\_reaped, 127
  - execute\_command, 127
  - execute\_script, 127
  - foreground\_job, 131
  - head, 132
  - jobid\_ctr, 132
  - MAN\_COMMANDS, 132
  - MAX\_ARGUMENTS, 132
  - n\_reaped, 132
  - orphan\_child, 128
  - pennos\_shell, 128
  - PROMPT, 126
  - reaped, 133
  - shell\_busy, 128
  - shell\_cat, 128
  - shell\_chmod, 128
  - shell\_cp, 129
  - shell\_echo, 129
  - shell\_kill, 129
  - shell\_ls, 129
  - shell\_mv, 129
  - shell\_orphanify, 129
  - shell\_ps, 130
  - shell\_rm, 130
  - shell\_sleep, 130
  - shell\_touch, 130
  - shell\_zombify, 130
  - spawn\_command, 130
  - stop\_handler, 131
  - stop\_order, 133
  - stop\_trigger, 133
  - term\_handler, 131
  - zombie\_child, 131
- pennos-shell.h
  - pennos\_shell, 133
- pennos.c
  - main, 117
- pennos\_shell
  - pennos-shell.c, 128
  - pennos-shell.h, 133
- perm
  - directory\_entry, 6
- pid
  - fileptr, 8
  - job, 9
  - PCB, 12
- point, 13
  - first, 13
  - second, 13
- point\_t
  - fat.h, 93
- PRINT
  - util.h, 148
- print\_fileptr\_pids\_all
  - filesystem.c, 27
  - filesystem.h, 41
- print\_parsed\_command
  - parser.h, 143
- PRINTE
  - util.h, 149
- priority
  - PCB, 12
- process\_create\_fileptrs
  - filesystem.c, 28
  - filesystem.h, 41
- process\_delete\_fileptrs
  - filesystem.c, 28
  - filesystem.h, 42
- PROMPT
  - pennos-shell.c, 126

- ptr
  - fileptr, 8
- puser-functions.c
  - current\_pcb, 57
  - p\_exit, 54
  - p\_kill, 54
  - p\_nice, 54
  - p\_sleep, 55
  - p\_spawn, 55
  - p\_waitpid, 55
  - ticks, 57
  - W\_WIFCONTINUED, 56
  - W\_WIFEXITED, 56
  - W\_WIFSIGNALED, 56
  - W\_WIFSTOPPED, 56
- puser-functions.h
  - current\_pcb, 61
  - p\_exit, 58
  - p\_kill, 58
  - p\_nice, 58
  - p\_sleep, 59
  - p\_spawn, 59
  - p\_waitpid, 60
  - ticks, 61
  - W\_WIFCONTINUED, 60
  - W\_WIFEXITED, 60
  - W\_WIFSIGNALED, 60
  - W\_WIFSTOPPED, 61
- read\_chain
  - fat.c, 87
  - fat.h, 104
- reaped
  - pennos-shell.c, 133
- reaper
  - scheduler.c, 62
- reaperContext
  - scheduler.c, 64
  - scheduler.h, 66
- removePCBFromList
  - PCB.h, 52
- ROOTDIR
  - fat.c, 91
  - fat.h, 107
- S\_SIGCHLD
  - globals.h, 135
- S\_SIGCONT
  - globals.h, 135
- S\_SIGSTOP
  - globals.h, 135
- S\_SIGTERM
  - globals.h, 135
- safe-user.c
  - safe\_f\_close, 143
  - safe\_f\_lseek, 144
  - safe\_f\_open, 144
  - safe\_f\_print, 144
  - safe\_f\_read, 144
  - safe\_f\_unlink, 144
  - safe\_f\_write, 144
- safe-user.h
  - safe\_f\_close, 145
  - safe\_f\_lseek, 145
  - safe\_f\_open, 145
  - safe\_f\_print, 145
  - safe\_f\_read, 146
  - safe\_f\_unlink, 146
  - safe\_f\_write, 146
- safe.c
  - safe\_close, 110
  - safe\_lseek, 110
  - safe\_mmap, 111
  - safe\_msync, 111
  - safe\_munmap, 112
  - safe\_open, 112
  - safe\_read, 112
  - safe\_write, 113
- safe.h
  - safe\_close, 114
  - safe\_lseek, 114
  - safe\_mmap, 114
  - safe\_msync, 115
  - safe\_munmap, 115
  - safe\_open, 116
  - safe\_read, 116
  - safe\_write, 116
- safe\_close
  - safe.c, 110
  - safe.h, 114
- safe\_f\_close
  - safe-user.c, 143
  - safe-user.h, 145
- safe\_f\_lseek
  - safe-user.c, 144
  - safe-user.h, 145
- safe\_f\_open
  - safe-user.c, 144
  - safe-user.h, 145
- safe\_f\_print
  - safe-user.c, 144
  - safe-user.h, 145
- safe\_f\_read
  - safe-user.c, 144
  - safe-user.h, 146
- safe\_f\_unlink
  - safe-user.c, 144
  - safe-user.h, 146
- safe\_f\_write
  - safe-user.c, 144
  - safe-user.h, 146
- safe\_lseek
  - safe.c, 110
  - safe.h, 114
- safe\_malloc
  - util.c, 147
  - util.h, 149

- safe\_mmap
  - safe.c, [111](#)
  - safe.h, [114](#)
- safe\_msync
  - safe.c, [111](#)
  - safe.h, [115](#)
- safe\_munmap
  - safe.c, [112](#)
  - safe.h, [115](#)
- safe\_open
  - safe.c, [112](#)
  - safe.h, [116](#)
- safe\_read
  - safe.c, [112](#)
  - safe.h, [116](#)
- safe\_signal
  - util.c, [147](#)
  - util.h, [150](#)
- safe\_write
  - safe.c, [113](#)
  - safe.h, [116](#)
- scheduler
  - scheduler.c, [63](#)
- scheduler.c
  - activeContext, [64](#)
  - alarmHandler, [62](#)
  - centisecond, [64](#)
  - freeStacks, [62](#)
  - mainContext, [64](#)
  - reaper, [62](#)
  - reaperContext, [64](#)
  - scheduler, [63](#)
  - schedulerContext, [64](#)
  - setAlarmHandler, [63](#)
  - setTimer, [63](#)
  - start\_scheduler, [63](#)
- scheduler.h
  - init\_scheduler, [65](#)
  - reaperContext, [66](#)
  - schedulerContext, [66](#)
  - setAlarmHandler, [65](#)
  - setTimer, [65](#)
  - start\_scheduler, [65](#)
- schedulerContext
  - scheduler.c, [64](#)
  - scheduler.h, [66](#)
- second
  - point, [13](#)
- setAlarmHandler
  - scheduler.c, [63](#)
  - scheduler.h, [65](#)
- setTimer
  - scheduler.c, [63](#)
  - scheduler.h, [65](#)
- shell\_busy
  - pennos-shell.c, [128](#)
- shell\_cat
  - pennos-shell.c, [128](#)
- shell\_chmod
  - pennos-shell.c, [128](#)
- shell\_cp
  - pennos-shell.c, [129](#)
- shell\_echo
  - pennos-shell.c, [129](#)
- shell\_kill
  - pennos-shell.c, [129](#)
- shell\_ls
  - pennos-shell.c, [129](#)
- shell\_mv
  - pennos-shell.c, [129](#)
- shell\_orphanify
  - pennos-shell.c, [129](#)
- shell\_ps
  - pennos-shell.c, [130](#)
- shell\_rm
  - pennos-shell.c, [130](#)
- shell\_sleep
  - pennos-shell.c, [130](#)
- shell\_touch
  - pennos-shell.c, [130](#)
- shell\_zombify
  - pennos-shell.c, [130](#)
- size
  - directory\_entry, [6](#)
- spawn\_command
  - pennos-shell.c, [130](#)
- src/filesystem/filesystem.c, [15](#)
- src/filesystem/filesystem.h, [29](#)
- src/kernel/kernel-functions.c, [42](#)
- src/kernel/kernel-functions.h, [44](#)
- src/kernel/PCB.c, [46](#)
- src/kernel/PCB.h, [46](#)
- src/kernel/puser-functions.c, [53](#)
- src/kernel/puser-functions.h, [57](#)
- src/kernel/scheduler.c, [61](#)
- src/kernel/scheduler.h, [65](#)
- src/logger/logger.c, [66](#)
- src/logger/logger.h, [71](#)
- src/pennfat/fat.c, [78](#)
- src/pennfat/fat.h, [91](#)
- src/pennfat/pennfat.c, [108](#)
- src/pennfat/safe.c, [109](#)
- src/pennfat/safe.h, [113](#)
- src/pennos.c, [117](#)
- src/shell/job-list.c, [118](#)
- src/shell/job-list.h, [120](#)
- src/shell/pennos-shell.c, [125](#)
- src/shell/pennos-shell.h, [133](#)
- src/util/globals.c, [134](#)
- src/util/globals.h, [134](#)
- src/util/p-errno.c, [136](#)
- src/util/p-errno.h, [138](#)
- src/util/parser.h, [141](#)
- src/util/safe-user.c, [143](#)
- src/util/safe-user.h, [145](#)
- src/util/util.c, [146](#)



- src/util/util.h, [148](#)
- STACKSIZE
  - PCB.h, [47](#)
- start\_scheduler
  - scheduler.c, [63](#)
  - scheduler.h, [65](#)
- status
  - PCB, [12](#)
- STDERR\_ID
  - PCB.h, [47](#)
- stdin\_file
  - parsed\_command, [10](#)
- STDIN\_ID
  - PCB.h, [47](#)
- stdout\_file
  - parsed\_command, [10](#)
- STDOUT\_ID
  - PCB.h, [48](#)
- stop\_handler
  - pennos-shell.c, [131](#)
- stop\_order
  - job, [9](#)
  - pennos-shell.c, [133](#)
- stop\_trigger
  - pennos-shell.c, [133](#)
- T\_BLOCKED
  - globals.h, [135](#)
- T\_RUNNING
  - globals.h, [136](#)
- T\_STOPPED
  - globals.h, [136](#)
- T\_ZOMBIED
  - globals.h, [136](#)
- term\_handler
  - pennos-shell.c, [131](#)
- ticks
  - puser-functions.c, [57](#)
  - puser-functions.h, [61](#)
- type
  - directory\_entry, [6](#)
- UNEXPECTED\_AMPERSAND
  - parser.h, [142](#)
- UNEXPECTED\_FILE\_INPUT
  - parser.h, [142](#)
- UNEXPECTED\_FILE\_OUTPUT
  - parser.h, [142](#)
- UNEXPECTED\_PIPELINE
  - parser.h, [142](#)
- util.c
  - ERRBUFFER\_SIZE, [148](#)
  - get\_argc, [147](#)
  - IOBUFFER\_SIZE, [148](#)
  - safe\_malloc, [147](#)
  - safe\_signal, [147](#)
- util.h
  - ERRBUFFER\_SIZE, [150](#)
  - get\_argc, [149](#)
  - IOBUFFER\_SIZE, [150](#)
  - PRINT, [148](#)
  - PRINTE, [149](#)
  - safe\_malloc, [149](#)
  - safe\_signal, [150](#)
- valid\_filename
  - fat.c, [88](#)
  - fat.h, [105](#)
- valid\_fs\_mounted
  - pennfat.c, [109](#)
- valid\_perm
  - filesystem.c, [28](#)
- W\_WIFCONTINUED
  - puser-functions.c, [56](#)
  - puser-functions.h, [60](#)
- W\_WIFEXITED
  - puser-functions.c, [56](#)
  - puser-functions.h, [60](#)
- W\_WIFSIGNALED
  - puser-functions.c, [56](#)
  - puser-functions.h, [60](#)
- W\_WIFSTOPPED
  - puser-functions.c, [56](#)
  - puser-functions.h, [61](#)
- wr\_pid
  - file, [7](#)
- write\_file
  - fat.c, [88](#)
- zombie\_child
  - pennos-shell.c, [131](#)