

Penn OS

Generated by Doxygen 1.13.2

1 PennOS Group 15 - Spring 2025	1
1.1 Names, PennKeys	1
1.2 Submitted Source Files	1
1.3 Extra Credit	2
1.4 Compilation Instructions	2
1.4.1 For the standalone PennFAT:	2
1.4.2 For the full PennOS:	2
1.5 Overview of Work Accomplished	2
1.6 Description of Code and Code Layout	3
1.7 General Comments	3
2 Class Index	5
2.1 Class List	5
3 File Index	7
3.1 File List	7
4 Class Documentation	9
4.1 buffer Struct Reference	9
4.1.1 Detailed Description	9
4.1.2 Member Data Documentation	9
4.1.2.1 arr	9
4.1.2.2 size	9
4.2 Deque Struct Reference	10
4.2.1 Detailed Description	10
4.2.2 Member Data Documentation	10
4.2.2.1 delete_mem	10
4.2.2.2 front	10
4.2.2.3 size	11
4.2.2.4 tail	11
4.3 dir_entry Struct Reference	11
4.3.1 Detailed Description	11
4.3.2 Member Data Documentation	12
4.3.2.1 first_block	12
4.3.2.2 mtime	12
4.3.2.3 name	12
4.3.2.4 perm	12
4.3.2.5 reserved	12
4.3.2.6 size	13
4.3.2.7 type	13
4.4 fd_node Struct Reference	13
4.4.1 Detailed Description	13
4.4.2 Member Data Documentation	14

4.4.2.1 fd	14
4.4.2.2 mode	14
4.4.2.3 name	14
4.4.2.4 next	14
4.4.2.5 offset	14
4.4.2.6 size	14
4.5 fd_table Struct Reference	14
4.5.1 Detailed Description	15
4.5.2 Member Data Documentation	15
4.5.2.1 head	15
4.6 Job Struct Reference	15
4.6.1 Detailed Description	16
4.6.2 Member Data Documentation	16
4.6.2.1 command	16
4.6.2.2 job_id	16
4.6.2.3 pcmd	16
4.6.2.4 pid	16
4.6.2.5 status	16
4.7 KernelState Struct Reference	17
4.7.1 Detailed Description	17
4.7.2 Member Data Documentation	17
4.7.2.1 curr_process	17
4.7.2.2 curr_thread_num	18
4.7.2.3 current_processes	18
4.7.2.4 dq_BLOCKED	18
4.7.2.5 dq_DEAD	18
4.7.2.6 dq_RUNNING	18
4.7.2.7 dq_STOPPED	18
4.7.2.8 dq_ZOMBIE	18
4.7.2.9 process_quanta	18
4.7.2.10 terminal_owner_pid	18
4.8 Node Struct Reference	19
4.8.1 Detailed Description	19
4.8.2 Member Data Documentation	19
4.8.2.1 data	19
4.8.2.2 next	19
4.8.2.3 prev	20
4.9 parsed_command Struct Reference	20
4.9.1 Detailed Description	20
4.9.2 Member Data Documentation	20
4.9.2.1 commands	20
4.9.2.2 is_background	20

4.9.2.3 is_file_append	20
4.9.2.4 num_commands	21
4.9.2.5 stdin_file	21
4.9.2.6 stdout_file	21
4.10 pcb_t Struct Reference	21
4.10.1 Detailed Description	22
4.10.2 Member Data Documentation	22
4.10.2.1 fd_table	22
4.10.2.2 file_descriptors	22
4.10.2.3 foreground	22
4.10.2.4 name	22
4.10.2.5 pid	22
4.10.2.6 ppid	23
4.10.2.7 priority_level	23
4.10.2.8 status	23
4.10.2.9 status_changed	23
4.10.2.10 stop_signal	23
4.10.2.11 term_signal	23
4.10.2.12 thread	23
4.10.2.13 wake_up_tick	23
4.11 ProcessFDNode Struct Reference	24
4.11.1 Detailed Description	24
4.11.2 Member Data Documentation	24
4.11.2.1 fd_num	24
4.11.2.2 fname	24
4.11.2.3 mode	24
4.11.2.4 next	25
4.11.2.5 offset	25
4.12 spthread_fwd_args_st Struct Reference	25
4.12.1 Member Data Documentation	25
4.12.1.1 actual_arg	25
4.12.1.2 actual_routine	26
4.12.1.3 child_meta	26
4.12.1.4 setup_cond	26
4.12.1.5 setup_done	26
4.12.1.6 setup_mutex	26
4.13 spthread_meta_st Struct Reference	26
4.13.1 Member Data Documentation	26
4.13.1.1 meta_mutex	26
4.13.1.2 state	26
4.13.1.3 suspend_set	27
4.14 spthread_signal_args_st Struct Reference	27

4.14.1 Member Data Documentation	27
4.14.1.1 ack	27
4.14.1.2 shutup_mutex	27
4.14.1.3 signal	27
4.15 sptthread_st Struct Reference	27
4.15.1 Member Data Documentation	28
4.15.1.1 meta	28
4.15.1.2 thread	28
4.16 vec_st Struct Reference	28
4.16.1 Member Data Documentation	28
4.16.1.1 capacity	28
4.16.1.2 data	28
4.16.1.3 ele_dtor_fn	28
4.16.1.4 length	28
5 File Documentation	29
5.1 README.md File Reference	29
5.2 src/fat/err.c File Reference	29
5.2.1 Function Documentation	30
5.2.1.1 error_case()	30
5.2.1.2 f_perror()	30
5.2.2 Variable Documentation	30
5.2.2.1 ERRNO	30
5.3 src/fat/err.h File Reference	30
5.3.1 Macro Definition Documentation	31
5.3.1.1 FILE_NOT_FOUND	31
5.3.1.2 FILE_SYSTEM	31
5.3.1.3 FS_ALREADY_MOUNTED	31
5.3.1.4 FS_NOT_MOUNTED	32
5.3.1.5 FS_SUCCESS	32
5.3.1.6 INVALID_ARGS	32
5.3.1.7 INVALID_FD	32
5.3.1.8 INVALID_OFFSET	32
5.3.1.9 INVALID_OPERATION	32
5.3.1.10 INVALID_PATH	32
5.3.1.11 INVALID_WHENCE	32
5.3.1.12 MEMORY_ERROR	32
5.3.1.13 MKFS	32
5.3.1.14 MOUNT	33
5.3.1.15 NO_SPACE	33
5.3.1.16 NOT_A_DIRECTORY	33
5.3.1.17 PERMISSION_DENIED	33

5.3.1.18 UNMOUNT	33
5.3.2 Function Documentation	33
5.3.2.1 error_case()	33
5.3.2.2 f_perror()	33
5.3.3 Variable Documentation	34
5.3.3.1 ERRNO	34
5.4 err.h	34
5.5 src/fat/fat_core.c File Reference	34
5.5.1 Function Documentation	35
5.5.1.1 convert_block_size()	35
5.5.1.2 create_file()	35
5.5.1.3 format_file_info()	36
5.5.1.4 fs_chmod()	36
5.5.1.5 fs_create()	36
5.5.1.6 fs_list_files()	37
5.5.1.7 fs_mount()	37
5.5.1.8 fs_mv()	37
5.5.1.9 fs_rm()	37
5.5.1.10 fs_touch()	38
5.5.1.11 fs_unmount()	38
5.5.1.12 get_directory()	38
5.5.1.13 get_file_permission()	38
5.5.1.14 index_to_directory()	39
5.5.1.15 k_read_at()	39
5.5.1.16 k_write_at()	39
5.5.1.17 lookup_directory_offset()	40
5.5.1.18 name_to_directory()	40
5.5.1.19 offset_to_directory()	40
5.5.1.20 perm_to_rwx()	41
5.5.1.21 read_file()	41
5.5.1.22 update_directory()	41
5.5.1.23 write_file()	42
5.6 src/fat/fat_core.h File Reference	42
5.6.1 Macro Definition Documentation	44
5.6.1.1 DIRECTORY_FILE	44
5.6.1.2 EOD_FLAG	44
5.6.1.3 FREE_BLOCK	44
5.6.1.4 LAST_BLOCK	44
5.6.1.5 LINK_FILE	44
5.6.1.6 REGULAR_FILE	44
5.6.1.7 SEEK_CUR	44
5.6.1.8 SEEK_END	44

5.6.1.9 SEEK_SET	45
5.6.1.10 UNKNOWN_FILE	45
5.6.2 Typedef Documentation	45
5.6.2.1 Buffer	45
5.6.2.2 Dir_entry	45
5.6.2.3 file_info_callback_t	46
5.6.3 Function Documentation	46
5.6.3.1 convert_block_size()	46
5.6.3.2 create_file()	46
5.6.3.3 format_file_info()	47
5.6.3.4 fs_chmod()	47
5.6.3.5 fs_create()	47
5.6.3.6 fs_list_files()	47
5.6.3.7 fs_mount()	48
5.6.3.8 fs_mv()	48
5.6.3.9 fs_rm()	48
5.6.3.10 fs_touch()	49
5.6.3.11 fs_unmount()	49
5.6.3.12 get_directory()	49
5.6.3.13 get_file_permission()	49
5.6.3.14 index_to_directory()	50
5.6.3.15 k_read_at()	50
5.6.3.16 k_write_at()	50
5.6.3.17 lookup_directory_offset()	51
5.6.3.18 name_to_directory()	51
5.6.3.19 offset_to_directory()	51
5.6.3.20 perm_to_rwx()	52
5.6.3.21 read_file()	52
5.6.3.22 update_directory()	52
5.6.3.23 write_file()	53
5.7 fat_core.h	53
5.8 src/fat/fat_kernel.c File Reference	55
5.8.1 Macro Definition Documentation	56
5.8.1.1 MAX_LINE_LENGTH	56
5.8.2 Function Documentation	56
5.8.2.1 f_cat()	56
5.8.2.2 f_chmod()	56
5.8.2.3 f_cp()	56
5.8.2.4 f_get_permission()	56
5.8.2.5 f_mv()	57
5.8.2.6 f_rm()	57
5.8.2.7 f_touch()	57

5.8.2.8 init()	57
5.8.2.9 k_chmod()	57
5.8.2.10 k_close()	58
5.8.2.11 k_get_permission()	58
5.8.2.12 k_ls()	58
5.8.2.13 k_lseek()	59
5.8.2.14 k_open()	59
5.8.2.15 k_read_at_offset()	59
5.8.2.16 k_unlink()	60
5.8.2.17 k_write_at_offset()	60
5.8.2.18 kf_read()	60
5.8.2.19 kf_write()	61
5.9 src/fat/fat_kernel.h File Reference	61
5.9.1 Macro Definition Documentation	62
5.9.1.1 F_APPEND	62
5.9.1.2 F_READ	63
5.9.1.3 F_WRITE	63
5.9.1.4 SEEK_CUR	63
5.9.1.5 SEEK_END	63
5.9.1.6 SEEK_SET	63
5.9.1.7 STDERR_FILENO	63
5.9.1.8 STDIN_FILENO	63
5.9.1.9 STDOUT_FILENO	63
5.9.2 Function Documentation	63
5.9.2.1 f_cat()	63
5.9.2.2 f_chmod()	64
5.9.2.3 f_cp()	64
5.9.2.4 f_get_permission()	64
5.9.2.5 f_mv()	64
5.9.2.6 f_rm()	65
5.9.2.7 f_touch()	65
5.9.2.8 init()	65
5.9.2.9 k_chmod()	65
5.9.2.10 k_close()	66
5.9.2.11 k_get_permission()	67
5.9.2.12 k_ls()	67
5.9.2.13 k_lseek()	67
5.9.2.14 k_open()	67
5.9.2.15 k_read_at_offset()	68
5.9.2.16 k_unlink()	68
5.9.2.17 k_write_at_offset()	68
5.9.2.18 kf_read()	69

5.9.2.19 <code>kf_write()</code>	69
5.10 <code>fat_kernel.h</code>	70
5.11 <code>src/fat/fd_table.c</code> File Reference	71
5.11.1 Function Documentation	71
5.11.1.1 <code>add_fd()</code>	71
5.11.1.2 <code>initialize_fd_table()</code>	72
5.11.1.3 <code>lookup_add_position()</code>	72
5.11.1.4 <code>lookup_fd()</code>	72
5.11.1.5 <code>remove_fd()</code>	72
5.12 <code>src/fat/fd_table.h</code> File Reference	73
5.12.1 Macro Definition Documentation	74
5.12.1.1 <code>F_APPEND</code>	74
5.12.1.2 <code>F_READ</code>	74
5.12.1.3 <code>F_WRITE</code>	74
5.12.2 Typedef Documentation	74
5.12.2.1 <code>FD_Node</code>	74
5.12.2.2 <code>FD_Table</code>	74
5.12.3 Function Documentation	75
5.12.3.1 <code>add_fd()</code>	75
5.12.3.2 <code>initialize_fd_table()</code>	75
5.12.3.3 <code>lookup_add_position()</code>	75
5.12.3.4 <code>lookup_fd()</code>	76
5.12.3.5 <code>remove_fd()</code>	76
5.13 <code>fd_table.h</code>	76
5.14 <code>src/fat/pennfat.c</code> File Reference	77
5.14.1 Function Documentation	77
5.14.1.1 <code>main()</code>	77
5.15 <code>src/fat/pennfat.h</code> File Reference	78
5.15.1 Macro Definition Documentation	78
5.15.1.1 <code>MAX_LINE_LENGTH</code>	78
5.16 <code>pennfat.h</code>	79
5.17 <code>src/kernel/calls/kernel-call.c</code> File Reference	79
5.17.1 Function Documentation	80
5.17.1.1 <code>k_get_current_process()</code>	80
5.17.1.2 <code>k_get_current_process_pid()</code>	80
5.17.1.3 <code>k_get_lowest_pid()</code>	80
5.17.1.4 <code>k_get_proc()</code>	80
5.17.1.5 <code>k_get_process_by_pid()</code>	81
5.17.1.6 <code>k_proc_cleanup()</code>	81
5.17.1.7 <code>k_proc_create()</code>	81
5.17.1.8 <code>k_read()</code>	81
5.17.1.9 <code>k_release_fd()</code>	82

5.17.1.10 k_write()	82
5.18 src/kernel/calls/kernel-call.h File Reference	83
5.18.1 Function Documentation	84
5.18.1.1 k_get_current_process()	84
5.18.1.2 k_get_current_process_pid()	84
5.18.1.3 k_get_proc()	84
5.18.1.4 k_get_process_by_pid()	85
5.18.1.5 k_proc_cleanup()	85
5.18.1.6 k_proc_create()	85
5.18.1.7 k_read()	85
5.18.1.8 k_release_fd()	85
5.18.1.9 k_write()	86
5.19 kernel-call.h	86
5.20 src/kernel/calls/sys-call.c File Reference	87
5.20.1 Function Documentation	88
5.20.1.1 map_fat_error_to_p_errno()	88
5.20.1.2 s_chmod()	88
5.20.1.3 s_close()	88
5.20.1.4 s_exit()	88
5.20.1.5 s_get_permission()	88
5.20.1.6 s_getpid()	89
5.20.1.7 s_itos()	89
5.20.1.8 s_kill()	89
5.20.1.9 s_ls()	89
5.20.1.10 s_lseek()	90
5.20.1.11 s_nice()	90
5.20.1.12 s_nice_pid()	90
5.20.1.13 s_open()	91
5.20.1.14 s_ps()	91
5.20.1.15 s_read()	91
5.20.1.16 s_register_end()	92
5.20.1.17 s_set_terminal_owner()	92
5.20.1.18 s_sleep()	92
5.20.1.19 s_spawn()	92
5.20.1.20 s_unlink()	93
5.20.1.21 s_waitpid()	93
5.20.1.22 s_waitpid_helper()	93
5.20.1.23 s_write()	93
5.21 src/kernel/calls/sys-call.h File Reference	94
5.21.1 Macro Definition Documentation	95
5.21.1.1 P_WAIT_FLAG_SINGALED	95
5.21.1.2 P_WAIT_FLAG_STOPPED	95

5.21.1.3 P_WAIT_SIG_MASK	96
5.21.1.4 P_WAIT_STATUS_MACROS_H	96
5.21.1.5 P_WIFEXITED	96
5.21.1.6 P_WIFSIGNALED	96
5.21.1.7 P_WIFSTOPPED	96
5.21.1.8 P_WSTOPSIG	96
5.21.1.9 P_WTERMSIG	97
5.21.2 Function Documentation	97
5.21.2.1 s_chmod()	97
5.21.2.2 s_close()	97
5.21.2.3 s_exit()	97
5.21.2.4 s_get_permission()	97
5.21.2.5 s_getpid()	98
5.21.2.6 s_kill()	98
5.21.2.7 s_ls()	98
5.21.2.8 s_lseek()	98
5.21.2.9 s_nice()	99
5.21.2.10 s_nice_pid()	99
5.21.2.11 s_open()	99
5.21.2.12 s_ps()	100
5.21.2.13 s_read()	100
5.21.2.14 s_register_end()	100
5.21.2.15 s_set_terminal_owner()	100
5.21.2.16 s_sleep()	101
5.21.2.17 s_spawn()	101
5.21.2.18 s_unlink()	101
5.21.2.19 s_waitpid()	102
5.21.2.20 s_write()	102
5.21.3 Variable Documentation	103
5.21.3.1 P_ERRNO	103
5.22 sys-call.h	103
5.23 src/kernel/calls/user-call.c File Reference	104
5.23.1 Function Documentation	105
5.23.1.1 stoi()	105
5.23.1.2 u_bg()	105
5.23.1.3 u_busy()	106
5.23.1.4 u_cat()	106
5.23.1.5 u_chmod()	106
5.23.1.6 u_cp()	106
5.23.1.7 u_echo()	107
5.23.1.8 u_fg()	107
5.23.1.9 u_jobs()	107

5.23.1.10 u_kill()	107
5.23.1.11 u_logout()	107
5.23.1.12 u_ls()	108
5.23.1.13 u_man()	108
5.23.1.14 u_mv()	108
5.23.1.15 u_nice()	108
5.23.1.16 u_nice_pid()	108
5.23.1.17 u_orphan_child()	109
5.23.1.18 u_orphanify()	109
5.23.1.19 u_ps()	109
5.23.1.20 u_rm()	109
5.23.1.21 u_sleep()	110
5.23.1.22 u_touch()	110
5.23.1.23 u_zombify()	110
5.23.1.24 zombie_child()	110
5.24 src/kernel/calls/user-call.h File Reference	110
5.24.1 Macro Definition Documentation	112
5.24.1.1 MAX_MESSAGE_SIZE	112
5.24.2 Function Documentation	112
5.24.2.1 stoi()	112
5.24.2.2 u_bg()	112
5.24.2.3 u_busy()	112
5.24.2.4 u_cat()	113
5.24.2.5 u_chmod()	113
5.24.2.6 u_cp()	113
5.24.2.7 u_crash()	114
5.24.2.8 u_echo()	114
5.24.2.9 u_fg()	114
5.24.2.10 u_hang()	114
5.24.2.11 u_jobs()	114
5.24.2.12 u_kill()	115
5.24.2.13 u_logout()	115
5.24.2.14 u_ls()	115
5.24.2.15 u_man()	115
5.24.2.16 u_mv()	115
5.24.2.17 u_nice()	116
5.24.2.18 u_nice_pid()	116
5.24.2.19 u_nohang()	116
5.24.2.20 u_orphan_child()	116
5.24.2.21 u_orphanify()	116
5.24.2.22 u_ps()	117
5.24.2.23 u_recur()	117

5.24.2.24 <code>u_rm()</code>	117
5.24.2.25 <code>u_sleep()</code>	117
5.24.2.26 <code>u_touch()</code>	117
5.24.2.27 <code>u_zombify()</code>	118
5.24.2.28 <code>zombie_child()</code>	118
5.25 <code>user-call.h</code>	118
5.26 <code>src/kernel/kernel.c</code> File Reference	119
5.26.1 Macro Definition Documentation	120
5.26.1.1 <code>_POSIX_C_SOURCE</code>	120
5.26.2 Function Documentation	120
5.26.2.1 <code>add_process_to_run_queue()</code>	120
5.26.2.2 <code>add_process_to_scheduler()</code>	120
5.26.2.3 <code>add_process_to_zombie_queue()</code>	120
5.26.2.4 <code>alarm_handler()</code>	121
5.26.2.5 <code>check_blocked_processes()</code>	121
5.26.2.6 <code>get_kernel_ticks()</code>	121
5.26.2.7 <code>getKernelState()</code>	121
5.26.2.8 <code>int_handler()</code>	122
5.26.2.9 <code>kernel_set_up()</code>	122
5.26.2.10 <code>remove_process_from_run_queue()</code>	122
5.26.2.11 <code>remove_process_from_zombie_queue()</code>	122
5.26.2.12 <code>start_kernel()</code>	123
5.26.2.13 <code>stop_handler()</code>	123
5.26.3 Variable Documentation	123
5.26.3.1 <code>g_shutdown_requested</code>	123
5.26.3.2 <code>k</code>	123
5.26.3.3 <code>kernel_ticks</code>	123
5.27 <code>src/kernel/kernel.h</code> File Reference	124
5.27.1 Macro Definition Documentation	125
5.27.1.1 <code>MAX_PROC</code>	125
5.27.1.2 <code>PROCESS_QUANTA</code>	125
5.27.2 Typedef Documentation	125
5.27.2.1 <code>KernelState</code>	125
5.27.3 Function Documentation	125
5.27.3.1 <code>add_process_to_run_queue()</code>	125
5.27.3.2 <code>add_process_to_scheduler()</code>	126
5.27.3.3 <code>add_process_to_zombie_queue()</code>	126
5.27.3.4 <code>check_blocked_processes()</code>	126
5.27.3.5 <code>get_kernel_ticks()</code>	127
5.27.3.6 <code>getKernelState()</code>	127
5.27.3.7 <code>kernel_set_up()</code>	127
5.27.3.8 <code>remove_process_from_run_queue()</code>	127

5.27.3.9 remove_process_from_zombie_queue()	128
5.27.3.10 start_kernel()	128
5.27.4 Variable Documentation	128
5.27.4.1 g_shutdown_requested	128
5.28 kernel.h	129
5.29 src/kernel/p_errno.c File Reference	129
5.29.1 Function Documentation	130
5.29.1.1 u_perror()	130
5.29.2 Variable Documentation	130
5.29.2.1 P_ERRNO	130
5.30 src/kernel/p_errno.h File Reference	131
5.30.1 Macro Definition Documentation	131
5.30.1.1 P_ERRNO_ALREADY_MOUNTED	131
5.30.1.2 P_ERRNO_ECHILD	132
5.30.1.3 P_ERRNO_EINVAL	132
5.30.1.4 P_ERRNO_ESRCH	132
5.30.1.5 P_ERRNO_FILE_NOT_FOUND	132
5.30.1.6 P_ERRNO_INTERNAL	132
5.30.1.7 P_ERRNO_INVALID_ARG	132
5.30.1.8 P_ERRNO_INVALID_FD	132
5.30.1.9 P_ERRNO_INVALID_OFFSET	132
5.30.1.10 P_ERRNO_INVALID_OPERATION	132
5.30.1.11 P_ERRNO_INVALID_WHENCE	132
5.30.1.12 P_ERRNO_NO_SPACE	133
5.30.1.13 P_ERRNO_NOT_A_DIRECTORY	133
5.30.1.14 P_ERRNO_NOT_MOUNTED	133
5.30.1.15 P_ERRNO_PERMISSION	133
5.30.1.16 P_ERRNO_SUCCESS	133
5.30.1.17 P_ERRNO_UNKNOWN	133
5.30.1.18 P_ERRNO_WRITE_CONFLICT	133
5.30.2 Function Documentation	133
5.30.2.1 u_perror()	133
5.30.3 Variable Documentation	134
5.30.3.1 P_ERRNO	134
5.31 p_errno.h	134
5.32 src/kernel/pcb.c File Reference	134
5.32.1 Function Documentation	135
5.32.1.1 pcb_add_fd()	135
5.32.1.2 pcb_create()	135
5.32.1.3 pcb_destroy()	136
5.32.1.4 pcb_get_fd()	136
5.32.1.5 pcb_initialize_fd_table()	136

5.32.1.6 pcb_remove_fd()	136
5.32.1.7 pcb_set_fd()	137
5.33 src/kernel/pcb.h File Reference	137
5.33.1 Typedef Documentation	138
5.33.1.1 pcb_t	138
5.33.1.2 ProcessFDNode	139
5.33.2 Function Documentation	139
5.33.2.1 pcb_add_fd()	139
5.33.2.2 pcb_get_fd()	139
5.33.2.3 pcb_initialize_fd_table()	139
5.33.2.4 pcb_remove_fd()	140
5.33.2.5 pcb_set_fd()	140
5.34 pcb.h	141
5.35 src/penn-shell/execute_command.c File Reference	141
5.35.1 Function Documentation	142
5.35.1.1 execute_command()	142
5.35.1.2 execute_script_file()	142
5.35.1.3 execute_single_command()	143
5.35.1.4 handle_shell_builtin()	143
5.36 src/penn-shell/execute_command.h File Reference	143
5.36.1 Function Documentation	144
5.36.1.1 execute_command()	144
5.37 execute_command.h	144
5.38 src/penn-shell/parser.c File Reference	145
5.38.1 Macro Definition Documentation	145
5.38.1.1 JUMP_OUT	145
5.38.2 Function Documentation	146
5.38.2.1 parse_command()	146
5.38.2.2 print_parsed_command()	146
5.38.2.3 print_parsed_command_without_end()	146
5.38.2.4 print_parser_errcode()	147
5.39 src/penn-shell/parser.h File Reference	147
5.39.1 Macro Definition Documentation	148
5.39.1.1 EXPECT_COMMANDS	148
5.39.1.2 EXPECT_INPUT_FILENAME	149
5.39.1.3 EXPECT_OUTPUT_FILENAME	149
5.39.1.4 UNEXPECTED_AMPERSAND	149
5.39.1.5 UNEXPECTED_FILE_INPUT	149
5.39.1.6 UNEXPECTED_FILE_OUTPUT	149
5.39.1.7 UNEXPECTED_PIPELINE	149
5.39.2 Function Documentation	149
5.39.2.1 parse_command()	149

5.39.2.2 print_parsed_command()	150
5.39.2.3 print_parsed_command_without_end()	150
5.39.2.4 print_parser_errcode()	150
5.40 parser.h	150
5.41 src/penn-shell/penn-shell.c File Reference	151
5.41.1 Macro Definition Documentation	152
5.41.1.1 _POSIX_C_SOURCE	152
5.41.1.2 LINE_BUFFER_CHUNK_SIZE	152
5.41.2 Function Documentation	152
5.41.2.1 add_job()	152
5.41.2.2 find_job_by_id()	153
5.41.2.3 find_job_by_pid()	153
5.41.2.4 find_last_job()	153
5.41.2.5 find_last_stopped_job()	154
5.41.2.6 free_job()	154
5.41.2.7 initialize_job_control()	154
5.41.2.8 read_line_from_fd()	154
5.41.2.9 reconstruct_command()	154
5.41.2.10 remove_job_by_pid()	154
5.41.2.11 shell()	155
5.41.3 Variable Documentation	155
5.41.3.1 job_list	155
5.41.3.2 next_job_id	155
5.41.3.3 shell_pid	155
5.41.3.4 terminal_controller_pid	155
5.42 src/penn-shell/penn-shell.h File Reference	156
5.42.1 Macro Definition Documentation	157
5.42.1.1 MAX_MESSAGE_SIZE	157
5.42.1.2 PENSHELL_H	158
5.42.1.3 SHELL_PROMPT	158
5.42.2 Typedef Documentation	158
5.42.2.1 Job	158
5.42.2.2 pid_t	158
5.42.3 Enumeration Type Documentation	158
5.42.3.1 JobStatus	158
5.42.4 Function Documentation	158
5.42.4.1 add_job()	158
5.42.4.2 find_job_by_id()	159
5.42.4.3 find_job_by_pid()	159
5.42.4.4 find_last_job()	159
5.42.4.5 find_last_stopped_job()	160
5.42.4.6 free_job()	160

5.42.4.7 initialize_job_control()	160
5.42.4.8 reconstruct_command()	160
5.42.4.9 remove_job_by_pid()	160
5.42.4.10 shell()	161
5.42.5 Variable Documentation	161
5.42.5.1 job_list	161
5.42.5.2 next_job_id	161
5.42.5.3 shell_pid	161
5.42.5.4 terminal_controller_pid	161
5.43 penn-shell.h	162
5.44 src/penn-shell/stress.c File Reference	162
5.44.1 Function Documentation	163
5.44.1.1 u_crash()	163
5.44.1.2 u_hang()	163
5.44.1.3 u_nohang()	164
5.44.1.4 u_recur()	164
5.45 src/penn-shell/stress.h File Reference	164
5.45.1 Function Documentation	165
5.45.1.1 u_crash()	165
5.45.1.2 u_hang()	165
5.45.1.3 u_nohang()	165
5.45.1.4 u_recur()	165
5.46 stress.h	165
5.47 src/penn_os/pennos.c File Reference	166
5.47.1 Function Documentation	166
5.47.1.1 main()	166
5.47.1.2 torta()	166
5.48 src/penn_os/pennos.h File Reference	166
5.49 pennos.h	166
5.50 src/util/deque.c File Reference	167
5.50.1 Function Documentation	168
5.50.1.1 clear_deque()	168
5.50.1.2 deque_contains()	168
5.50.1.3 deque_get_back()	168
5.50.1.4 deque_get_front()	168
5.50.1.5 deque_get_nth_elem()	169
5.50.1.6 deque_new()	169
5.50.1.7 deque_pop_back()	169
5.50.1.8 deque_pop_front()	170
5.50.1.9 deque_push_back()	170
5.50.1.10 deque_push_front()	170
5.50.1.11 deque_remove_nth_elem()	170

5.50.1.12 deque_remove_specific()	171
5.50.1.13 deque_size()	171
5.51 src/util/deque.h File Reference	172
5.51.1 Macro Definition Documentation	173
5.51.1.1 DEQUE_H	173
5.51.2 Typedef Documentation	173
5.51.2.1 Deque	173
5.51.2.2 Node	174
5.51.3 Function Documentation	174
5.51.3.1 clear_deque()	174
5.51.3.2 deque_contains()	174
5.51.3.3 deque_get_back()	174
5.51.3.4 deque_get_front()	175
5.51.3.5 deque_get_nth_elem()	175
5.51.3.6 deque_new()	175
5.51.3.7 deque_pop_back()	176
5.51.3.8 deque_pop_front()	176
5.51.3.9 deque_push_back()	176
5.51.3.10 deque_push_front()	177
5.51.3.11 deque_remove_nth_elem()	177
5.51.3.12 deque_remove_specific()	177
5.51.3.13 deque_size()	178
5.52 deque.h	178
5.53 src/util/logger/logger.c File Reference	179
5.53.1 Function Documentation	179
5.53.1.1 init_logger()	179
5.53.1.2 log_event()	180
5.54 src/util/logger/logger.h File Reference	181
5.54.1 Function Documentation	182
5.54.1.1 init_logger()	182
5.54.1.2 log_event()	183
5.55 logger.h	183
5.56 src/util/spthread.c File Reference	183
5.56.1 Macro Definition Documentation	185
5.56.1.1 _GNU_SOURCE	185
5.56.1.2 MILLISEC_IN_NANO	185
5.56.1.3 SPTHREAD_RUNNING_STATE	185
5.56.1.4 SPTHREAD_SIG_CONTINUE	185
5.56.1.5 SPTHREAD_SIG_SUSPEND	185
5.56.1.6 SPTHREAD_SUSPENDED_STATE	185
5.56.1.7 SPTHREAD_TERMINATED_STATE	185
5.56.2 Typedef Documentation	185

5.56.2.1 pthread_fn	185
5.56.2.2 spthread_fwd_args	185
5.56.2.3 spthread_meta_t	186
5.56.2.4 spthread_signal_args	186
5.56.3 Function Documentation	186
5.56.3.1 spthread_cancel()	186
5.56.3.2 spthread_continue()	186
5.56.3.3 spthread_create()	186
5.56.3.4 spthread_disable_interrupts_self()	186
5.56.3.5 spthread_enable_interrupts_self()	186
5.56.3.6 spthread_equal()	186
5.56.3.7 spthread_exit()	187
5.56.3.8 spthread_join()	187
5.56.3.9 spthread_self()	187
5.56.3.10 spthread_suspend()	187
5.56.3.11 spthread_suspend_self()	187
5.57 src/util/spthread.h File Reference	187
5.57.1 Macro Definition Documentation	188
5.57.1.1 SIGPTHD	188
5.57.2 Typedef Documentation	189
5.57.2.1 spthread_t	189
5.57.3 Function Documentation	189
5.57.3.1 spthread_cancel()	189
5.57.3.2 spthread_continue()	189
5.57.3.3 spthread_create()	189
5.57.3.4 spthread_disable_interrupts_self()	189
5.57.3.5 spthread_enable_interrupts_self()	189
5.57.3.6 spthread_equal()	189
5.57.3.7 spthread_exit()	189
5.57.3.8 spthread_join()	190
5.57.3.9 spthread_self()	190
5.57.3.10 spthread_suspend()	190
5.57.3.11 spthread_suspend_self()	190
5.58 spthread.h	190
5.59 src/util/types/process-status.h File Reference	193
5.59.1 Macro Definition Documentation	193
5.59.1.1 pid_t	193
5.59.2 Enumeration Type Documentation	193
5.59.2.1 ProcessStatus	193
5.60 process-status.h	194
5.61 src/util/types/signals.h File Reference	194
5.61.1 Macro Definition Documentation	195

5.61.1.1 P_SIGCONT	195
5.61.1.2 P_SIGSTOP	195
5.61.1.3 P_SIGTERM	195
5.62 signals.h	195
5.63 src/util/Vec.c File Reference	195
5.63.1 Function Documentation	196
5.63.1.1 vec_clear()	196
5.63.1.2 vec_destroy()	196
5.63.1.3 vec_erase()	196
5.63.1.4 vec_get()	196
5.63.1.5 vec_insert()	196
5.63.1.6 vec_new()	196
5.63.1.7 vec_pop_back()	197
5.63.1.8 vec_push_back()	197
5.63.1.9 vec_resize()	197
5.63.1.10 vec_set()	197
5.64 src/util/Vec.h File Reference	198
5.64.1 Macro Definition Documentation	199
5.64.1.1 vec_capacity	199
5.64.1.2 vec_is_empty	199
5.64.1.3 vec_len	199
5.64.2 Typedef Documentation	199
5.64.2.1 ptr_dtor_fn	199
5.64.2.2 ptr_t	199
5.64.2.3 Vec	200
5.64.3 Function Documentation	200
5.64.3.1 vec_clear()	200
5.64.3.2 vec_destroy()	200
5.64.3.3 vec_erase()	200
5.64.3.4 vec_get()	200
5.64.3.5 vec_insert()	200
5.64.3.6 vec_new()	200
5.64.3.7 vec_pop_back()	201
5.64.3.8 vec_push_back()	201
5.64.3.9 vec_resize()	201
5.64.3.10 vec_set()	201
5.65 Vec.h	202

Chapter 1

PennOS Group 15 - Spring 2025

1.1 Names, PennKeys

- Roshan Bellary: rbellary
- Jefferson Ding: tyding
- Nikita Mounier: nmounier
- Praneel Varshney: pvarsh

1.2 Submitted Source Files

- `src/fat/` - PennFAT filesystem implementation
 - `pennfat.c`, `pennfat.h` - Main PennFAT program
 - `fat_core.c`, `fat_core.h` - Core filesystem functions
 - `fat_kernel.c`, `fat_kernel.h` - Kernel-level filesystem functions
 - `fd_table.c`, `fd_table.h` - File descriptor table implementation
 - `err.c`, `err.h` - Error handling for filesystem
- `src/kernel/` - PennOS kernel implementation
 - `kernel.c`, `kernel.h` - Core kernel functionality
 - `pcb.c`, `pcb.h` - Process Control Block implementation
 - `p_errno.h`, `u_perror.c` - Error handling for the OS
 - `calls/` - System and user-level calls
 - * `kernel-call.c`, `kernel-call.h` - Kernel-level calls
 - * `sys-call.c`, `sys-call.h` - System calls
 - * `user-call.c`, `user-call.h` - User-level commands
- `src/penn-shell/` - PennOS shell implementation
 - `penn-shell.c`, `penn-shell.h` - Shell implementation
 - `parser.c`, `parser.h` - Command line parser
 - `execute_command.c`, `execute_command.h` - Command execution
 - `stress.c`, `stress.h` - Shell commands for testing

- `src/util/` - Utility functions and data structures
 - `Vec.c`, `Vec.h` - Vector implementation
 - `deque.c`, `deque.h` - `Deque` implementation
 - `spthread.c`, `spthread.h` - Special thread library
 - `logger/` - Logging functionality
 - `types/` - Type definitions
- `src/pennos.c` - Main PennOS entry point

1.3 Extra Credit

We did not implement any of the extra credit options.

1.4 Compilation Instructions

1.4.1 For the standalone PennFAT:

1. Change to the fat directory: `cd src/fat`
2. Build the PennFAT executable: `make`
3. Run the standalone PennFAT: `./bin/pennfat`
4. Create a filesystem: `mkfs <filesystem_name> <blocks_in_fat> <block_size_↵ config>`
 - Example: `mkfs fs.img 10 4` (creates a filesystem with 10 blocks in FAT and 4096-byte blocks)

1.4.2 For the full PennOS:

1. Create a filesystem using the standalone PennFAT (see instructions above)
2. Move the created filesystem file to the root directory
3. From the root directory, compile PennOS: `make`
4. Run PennOS with: `./bin/pennos fatfs <filesystem_file> [log_fname]`
 - Example: `./bin/pennos fatfs test_fs pennos.log`

1.5 Overview of Work Accomplished

We've implemented PennOS, a complete UNIX-like operating system. It runs as a user-level program on a host OS. Some noteworthy components are:

1. **PennFAT Filesystem:** A FAT-based filesystem implementation that supports file creation, deletion, reading, writing, permission management. Note that this filesystem is stored as a single file on the host OS.
2. **Kernel and Scheduler:** A round-robin scheduler with priority support. Manages processes as 'sptreads' and handles create, block, terminate, and scheduling.
3. **Process Management:** Full process lifecycle support, including zombies, orphans, and cleanup. Also supports parent-child relationships.
4. **Shell:** A command-line shell that supports built-in commands, I/O redirection, job control, and shell scripts.
5. **Error Handling:** Error handling was done similar to UNIX's `errno` and `perror`.

1.6 Description of Code and Code Layout

See the Submitted Source Files section for information on what each file contains.

As for the overall layout, we structured the OS into four key layers:

- **Core Infrastructure:** Basic utilities like `Vec`, `deque`, and the custom `spthread` library for lightweight threading.
- **Kernel Layer:** Handles core OS responsibilities—process management, scheduling, and kernel-level calls.
- **System Call Interface:** Bridges user programs and the kernel, isolating privilege boundaries and internal logic.
- **User-Level Applications:** Includes the shell, command parsing, and user-facing built-ins.

Some implementation decisions that we made were:

- **Abstraction:** we used consistent prefixes (`k_`, `s_`, `u_`) to denote kernel, system, and user-level functions.
- **Process Control Block (PCB):** Stores process metadata—PID, state, priority, file descriptors—used across scheduling and management.
- **Scheduler:** Priority-based with multiple queues for different process states (running, blocked, zombie, etc.).
- **Filesystem Integration:** Kernel interfaces cleanly with the FAT filesystem, maintaining a strict separation.
- **Error Handling:** Unified error system modeled after `errno/perror` for consistent diagnostics.

1.7 General Comments

No other comments.

Chapter 2

Class Index

2.1 Class List

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

buffer	9
Deque	
Double-ended queue (deque) structure	10
dir_entry	11
fd_node	13
fd_table	14
Job	
Struct for job	15
KernelState	
The kernel state	17
Node	
Node structure for doubly-linked list used in Deque	19
parsed_command	
Struct parsed_command stored all necessary information needed for penn-shell	20
pcb_t	
The Process Control Block (PCB) structure	21
ProcessFDNode	
The Process File Descriptor Node structure	24
spthread_fwd_args_st	25
spthread_meta_st	26
spthread_signal_args_st	27
spthread_st	27
vec_st	28

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

src/fat/err.c	29
src/fat/err.h	30
src/fat/fat_core.c	34
src/fat/fat_core.h	42
src/fat/fat_kernel.c	55
src/fat/fat_kernel.h	61
src/fat/fd_table.c	71
src/fat/fd_table.h	73
src/fat/pennfat.c	77
src/fat/pennfat.h	78
src/kernel/kernel.c	119
src/kernel/kernel.h	124
src/kernel/p_errno.c	129
src/kernel/p_errno.h	131
src/kernel/pcb.c	134
src/kernel/pcb.h	137
src/kernel/calls/kernel-call.c	79
src/kernel/calls/kernel-call.h	83
src/kernel/calls/sys-call.c	87
src/kernel/calls/sys-call.h	94
src/kernel/calls/user-call.c	104
src/kernel/calls/user-call.h	110
src/penn-shell/execute_command.c	141
src/penn-shell/execute_command.h	143
src/penn-shell/parser.c	145
src/penn-shell/parser.h	147
src/penn-shell/penn-shell.c	151
src/penn-shell/penn-shell.h	156
src/penn-shell/stress.c	162
src/penn-shell/stress.h	164
src/penn_os/pennos.c	166
src/penn_os/pennos.h	166
src/util/deque.c	167
src/util/deque.h	172
src/util/spthread.c	183

src/util/ spthread.h	187
src/util/ Vec.c	195
src/util/ Vec.h	198
src/util/logger/ logger.c	179
src/util/logger/ logger.h	181
src/util/types/ process-status.h	193
src/util/types/ signals.h	194

Chapter 4

Class Documentation

4.1 buffer Struct Reference

```
#include <fat_core.h>
```

Public Attributes

- `uint8_t * arr`
- `int size`

4.1.1 Detailed Description

buffer structure for holding data

Parameters

<i>arr</i>	pointer to data array
<i>size</i>	size of buffer in bytes

4.1.2 Member Data Documentation

4.1.2.1 arr

```
uint8_t* buffer::arr
```

4.1.2.2 size

```
int buffer::size
```

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

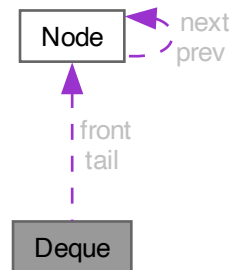
- `src/fat/fat_core.h`

4.2 Deque Struct Reference

Double-ended queue (deque) structure.

```
#include <deque.h>
```

Collaboration diagram for Deque:



Public Attributes

- [Node](#) * [front](#)
- [Node](#) * [tail](#)
- int [size](#)
- void(* [delete_mem](#))(void *)

4.2.1 Detailed Description

Double-ended queue (deque) structure.

The [Deque](#) supports insertion and removal of elements from both ends. It uses a doubly-linked list of [Node](#) structures. The `delete_mem` function pointer is used to free memory for stored data.

4.2.2 Member Data Documentation

4.2.2.1 `delete_mem`

```
void(* Deque::delete_mem) (void *)
```

Function pointer to free element memory.

4.2.2.2 `front`

```
Node* Deque::front
```

Pointer to the front node.

4.2.2.3 size

```
int Deque::size
```

Number of elements in the deque.

4.2.2.4 tail

```
Node* Deque::tail
```

Pointer to the tail node.

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

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

4.3 dir_entry Struct Reference

```
#include <fat_core.h>
```

Public Attributes

- char [name](#) [32]
- uint32_t [size](#)
- uint16_t [first_block](#)
- uint8_t [type](#)
- uint8_t [perm](#)
- time_t [mtime](#)
- long double [reserved](#)

4.3.1 Detailed Description

directory entry structure representing a file in the filesystem

Parameters

<i>name</i>	file name <code>name[0]</code> is a special marker <ul style="list-style-type: none">• 0: end of directory• 1: deleted entry; the file is also deleted• 2: deleted entry; the file is still being used
<i>size</i>	number of bytes in file
<i>first_block</i>	first block number of file (undefined if 0 size)

Parameters

<i>type</i>	type of file <ul style="list-style-type: none">• 0: unknown• 1: regular file• 2: directory file• 3: symbolic link
<i>perm</i>	file permissions <ul style="list-style-type: none">• 0: none• 2: write only• 4: read only• 5: read and executable (shell scripts)• 6: read and write• 7: read, write, and executable
<i>mtime</i>	creation/modification time
<i>reserved</i>	reserved bytes (last 16)

4.3.2 Member Data Documentation

4.3.2.1 first_block

```
uint16_t dir_entry::first_block
```

4.3.2.2 mtime

```
time_t dir_entry::mtime
```

4.3.2.3 name

```
char dir_entry::name[32]
```

4.3.2.4 perm

```
uint8_t dir_entry::perm
```

4.3.2.5 reserved

```
long double dir_entry::reserved
```

4.3.2.6 size

```
uint32_t dir_entry::size
```

4.3.2.7 type

```
uint8_t dir_entry::type
```

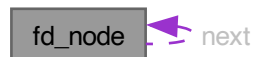
The documentation for this struct was generated from the following file:

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

4.4 fd_node Struct Reference

```
#include <fd_table.h>
```

Collaboration diagram for fd_node:



Public Attributes

- `char * name`
- `int fd`
- `int offset`
- `int size`
- `int mode`
- `struct fd_node * next`

4.4.1 Detailed Description

node struct to represent each file in the fd table

Parameters

<i>name</i>	file name
<i>fd</i>	fd number
<i>offset</i>	pointer offset
<i>size</i>	file size
<i>mode</i>	mode of file (0: read, 1: write, 2: append)
<i>next</i>	pointer to next node in list

4.4.2 Member Data Documentation

4.4.2.1 fd

```
int fd_node::fd
```

4.4.2.2 mode

```
int fd_node::mode
```

4.4.2.3 name

```
char* fd_node::name
```

4.4.2.4 next

```
struct fd_node* fd_node::next
```

4.4.2.5 offset

```
int fd_node::offset
```

4.4.2.6 size

```
int fd_node::size
```

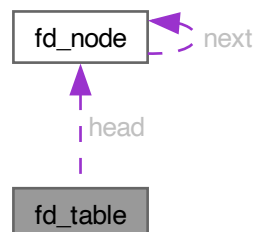
The documentation for this struct was generated from the following file:

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

4.5 fd_table Struct Reference

```
#include <fd_table.h>
```

Collaboration diagram for fd_table:



Public Attributes

- [FD_Node](#) * [head](#)

4.5.1 Detailed Description

fd table struct

Parameters

<i>head</i>	head of the fd_table which is a linkedlist of fd_nodes
-------------	--

4.5.2 Member Data Documentation

4.5.2.1 head

```
FD\_Node* fd_table::head
```

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

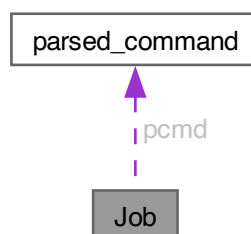
- [src/fat/fd_table.h](#)

4.6 Job Struct Reference

struct for job

```
#include <penn-shell.h>
```

«««< HEAD Collaboration diagram for Job: ===== Collaboration diagram for Job:»»»> 1bee6ba39be93d5183b34551923906b3a4d3



Public Attributes

- int [job_id](#)
- [pid_t](#) pid
- char * [command](#)
- [JobStatus](#) status
- struct [parsed_command](#) * [pcmd](#)

4.6.1 Detailed Description

struct for job

4.6.2 Member Data Documentation

4.6.2.1 command

```
char* Job::command
```

4.6.2.2 job_id

```
int Job::job_id
```

4.6.2.3 pcmd

```
struct parsed\_command* Job::pcmd
```

4.6.2.4 pid

```
pid\_t Job::pid
```

4.6.2.5 status

```
JobStatus Job::status
```

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

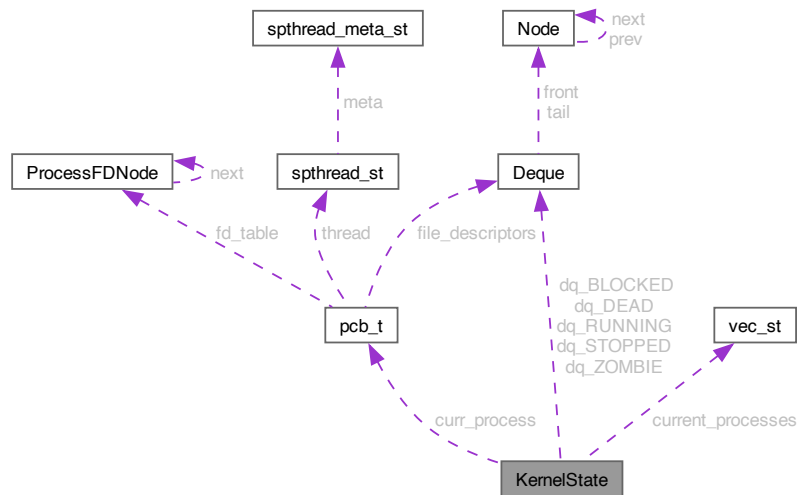
- src/penn-shell/[penn-shell.h](#)

4.7 KernelState Struct Reference

The kernel state.

```
#include <kernel.h>
```

Collaboration diagram for KernelState:



Public Attributes

- `Deque * dq_RUNNING [3]`
- `Deque * dq_ZOMBIE`
- `Deque * dq_DEAD`
- `Deque * dq_BLOCKED`
- `Deque * dq_STOPPED`
- `int curr_thread_num`
- `int process_quanta`
- `pcb_t * curr_process`
- `Vec current_processes`
- `pid_t terminal_owner_pid`

4.7.1 Detailed Description

The kernel state.

This structure contains all the state information for the kernel.

4.7.2 Member Data Documentation

4.7.2.1 curr_process

`pcb_t*` `KernelState::curr_process`

4.7.2.2 curr_thread_num

```
int KernelState::curr_thread_num
```

4.7.2.3 current_processes

```
Vec KernelState::current_processes
```

4.7.2.4 dq_BLOCKED

```
Deque* KernelState::dq_BLOCKED
```

4.7.2.5 dq_DEAD

```
Deque* KernelState::dq_DEAD
```

4.7.2.6 dq_RUNNING

```
Deque* KernelState::dq_RUNNING[3]
```

4.7.2.7 dq_STOPPED

```
Deque* KernelState::dq_STOPPED
```

4.7.2.8 dq_ZOMBIE

```
Deque* KernelState::dq_ZOMBIE
```

4.7.2.9 process_quanta

```
int KernelState::process_quanta
```

4.7.2.10 terminal_owner_pid

```
pid_t KernelState::terminal_owner_pid
```

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

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

4.8 Node Struct Reference

[Node](#) structure for doubly-linked list used in [Deque](#).

```
#include <deque.h>
```

Collaboration diagram for Node:



Public Attributes

- void * [data](#)
- struct [Node](#) * [next](#)
- struct [Node](#) * [prev](#)

4.8.1 Detailed Description

[Node](#) structure for doubly-linked list used in [Deque](#).

This structure represents a node in the deque, holding a pointer to data and pointers to the next and previous nodes in the list.

4.8.2 Member Data Documentation

4.8.2.1 data

```
void* Node::data
```

Pointer to the data stored in the node.

4.8.2.2 next

```
struct Node* Node::next
```

Pointer to the next node in the deque.

4.8.2.3 prev

```
struct Node* Node::prev
```

Pointer to the previous node in the deque.

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

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

4.9 parsed_command Struct Reference

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

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

Public Attributes

- bool [is_background](#)
- bool [is_file_append](#)
- const char * [stdin_file](#)
- const char * [stdout_file](#)
- size_t [num_commands](#)
- char ** [commands](#) []

4.9.1 Detailed Description

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

4.9.2 Member Data Documentation

4.9.2.1 commands

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

4.9.2.2 is_background

```
bool parsed_command::is_background
```

4.9.2.3 is_file_append

```
bool parsed_command::is_file_append
```

4.9.2.4 num_commands

```
size_t parsed_command::num_commands
```

4.9.2.5 stdin_file

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

4.9.2.6 stdout_file

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

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

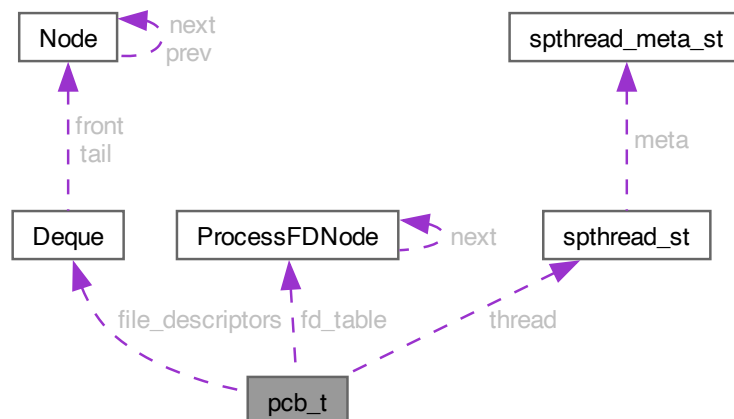
- src/penn-shell/[parser.h](#)

4.10 pcb_t Struct Reference

The Process Control Block (PCB) structure.

```
#include <pcb.h>
```

Collaboration diagram for pcb_t:



Public Attributes

- [pid_t](#) pid
- [pid_t](#) ppid
- int [priority_level](#)
- int [term_signal](#)
- int [stop_signal](#)
- [ProcessStatus](#) status
- bool [status_changed](#)
- unsigned long [wake_up_tick](#)
- [Deque](#) * [file_descriptors](#)
- [ProcessFDNode](#) * [fd_table](#)
- [spthread_t](#) * [thread](#)
- char * [name](#)
- bool [foreground](#)

4.10.1 Detailed Description

The Process Control Block (PCB) structure.

This structure represents a process control block (PCB) in the kernel.

4.10.2 Member Data Documentation

4.10.2.1 fd_table

```
ProcessFDNode* pcb_t::fd_table
```

4.10.2.2 file_descriptors

```
Deque* pcb_t::file_descriptors
```

4.10.2.3 foreground

```
bool pcb_t::foreground
```

4.10.2.4 name

```
char* pcb_t::name
```

4.10.2.5 pid

```
pid\_t pcb_t::pid
```

4.10.2.6 ppid

```
pid_t pcb_t::ppid
```

4.10.2.7 priority_level

```
int pcb_t::priority_level
```

4.10.2.8 status

```
ProcessStatus pcb_t::status
```

4.10.2.9 status_changed

```
bool pcb_t::status_changed
```

4.10.2.10 stop_signal

```
int pcb_t::stop_signal
```

4.10.2.11 term_signal

```
int pcb_t::term_signal
```

4.10.2.12 thread

```
spthread_t* pcb_t::thread
```

4.10.2.13 wake_up_tick

```
unsigned long pcb_t::wake_up_tick
```

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

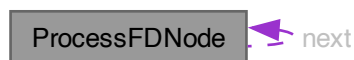
- [src/kernel/pcb.h](#)

4.11 ProcessFDNode Struct Reference

The Process File Descriptor [Node](#) structure.

```
#include <pcb.h>
```

```
«««< HEAD Collaboration diagram for ProcessFDNode: ===== Collaboration diagram for ProcessFDNode:»»»>
1bee6ba39be93d5183b34551923906b3a4d38e38
```



Public Attributes

- int [fd_num](#)
- char [fname](#) [32]
- int [mode](#)
- int [offset](#)
- struct [ProcessFDNode](#) * [next](#)

4.11.1 Detailed Description

The Process File Descriptor [Node](#) structure.

This structure represents a file descriptor node in the process's file descriptor table.

4.11.2 Member Data Documentation

4.11.2.1 fd_num

```
int ProcessFDNode::fd_num
```

4.11.2.2 fname

```
char ProcessFDNode::fname[32]
```

4.11.2.3 mode

```
int ProcessFDNode::mode
```

4.11.2.4 next

```
struct ProcessFDNode* ProcessFDNode::next
```

4.11.2.5 offset

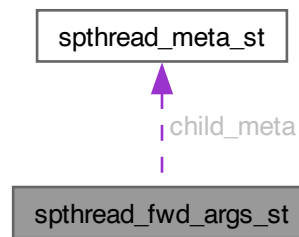
```
int ProcessFDNode::offset
```

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

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

4.12 spthread_fwd_args_st Struct Reference

Collaboration diagram for spthread_fwd_args_st:



Public Attributes

- [pthread_fn](#) `actual_routine`
- `void *` [actual_arg](#)
- `bool` [setup_done](#)
- `pthread_mutex_t` [setup_mutex](#)
- `pthread_cond_t` [setup_cond](#)
- `spthread_meta_t *` [child_meta](#)

4.12.1 Member Data Documentation

4.12.1.1 actual_arg

```
void* spthread_fwd_args_st::actual_arg
```

4.12.1.2 actual_routine

```
pthread_fn spthread_fwd_args_st::actual_routine
```

4.12.1.3 child_meta

```
spthread_meta_t* spthread_fwd_args_st::child_meta
```

4.12.1.4 setup_cond

```
pthread_cond_t spthread_fwd_args_st::setup_cond
```

4.12.1.5 setup_done

```
bool spthread_fwd_args_st::setup_done
```

4.12.1.6 setup_mutex

```
pthread_mutex_t spthread_fwd_args_st::setup_mutex
```

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

- [src/util/spthread.c](#)

4.13 spthread_meta_st Struct Reference

Public Attributes

- sigset_t [suspend_set](#)
- volatile sig_atomic_t [state](#)
- pthread_mutex_t [meta_mutex](#)

4.13.1 Member Data Documentation

4.13.1.1 meta_mutex

```
pthread_mutex_t spthread_meta_st::meta_mutex
```

4.13.1.2 state

```
volatile sig_atomic_t spthread_meta_st::state
```


4.13.1.3 `suspend_set`

```
sigset_t spthread_meta_st::suspend_set
```

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

- [src/util/spthread.c](#)

4.14 `spthread_signal_args_st` Struct Reference

Public Attributes

- const int [signal](#)
- volatile sig_atomic_t [ack](#)
- pthread_mutex_t [shutup_mutex](#)

4.14.1 Member Data Documentation

4.14.1.1 `ack`

```
volatile sig_atomic_t spthread_signal_args_st::ack
```

4.14.1.2 `shutup_mutex`

```
pthread_mutex_t spthread_signal_args_st::shutup_mutex
```

4.14.1.3 `signal`

```
const int spthread_signal_args_st::signal
```

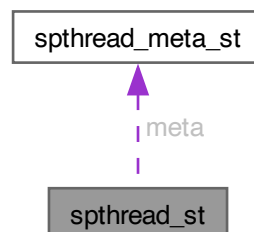
The documentation for this struct was generated from the following file:

- [src/util/spthread.c](#)

4.15 `spthread_st` Struct Reference

```
#include <spthread.h>
```

Collaboration diagram for `spthread_st`:



Public Attributes

- [pthread_t](#) [thread](#)
- [spthread_meta_t](#) * [meta](#)

4.15.1 Member Data Documentation

4.15.1.1 meta

```
spthread\_meta\_t* spthread\_st::meta
```

4.15.1.2 thread

```
pthread\_t spthread\_st::thread
```

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

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

4.16 vec_st Struct Reference

```
#include <Vec.h>
```

Public Attributes

- [ptr_t](#) * [data](#)
- [size_t](#) [length](#)
- [size_t](#) [capacity](#)
- [ptr_dtor_fn](#) [ele_dtor_fn](#)

4.16.1 Member Data Documentation

4.16.1.1 capacity

```
size\_t vec\_st::capacity
```

4.16.1.2 data

```
ptr\_t* vec\_st::data
```

4.16.1.3 ele_dtor_fn

```
ptr\_dtor\_fn vec\_st::ele\_dtor\_fn
```

4.16.1.4 length

```
size\_t vec\_st::length
```

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

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

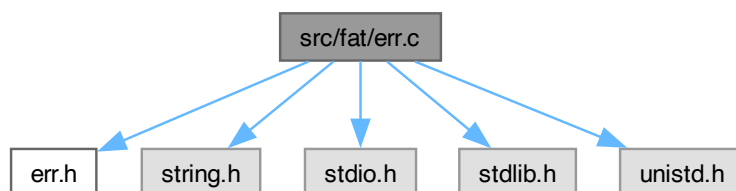
Chapter 5

File Documentation

5.1 README.md File Reference

5.2 src/fat/err.c File Reference

```
#include "err.h"  
#include <string.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
Include dependency graph for err.c:
```



Functions

- char * `error_case` (void)
Returns a string description of the current error.
- void `f_perror` (char *message)
Prints an error message to the standard error stream.

Variables

- int `ERRNO` = 0

5.2.1 Function Documentation

5.2.1.1 error_case()

```
char * error_case (  
    void )
```

Returns a string description of the current error.

Returns

A string description of the current error.

5.2.1.2 f_perror()

```
void f_perror (  
    char * message)
```

Prints an error message to the standard error stream.

Parameters

<i>message</i>	The message to print.
----------------	-----------------------

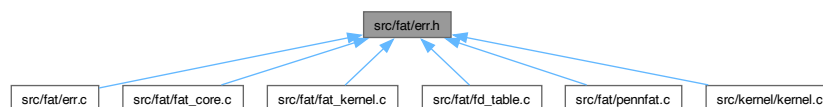
5.2.2 Variable Documentation

5.2.2.1 ERRNO

```
int ERRNO = 0
```

5.3 src/fat/err.h File Reference

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



Macros

- `#define FS_SUCCESS` 0
- `#define MKFS` 1
- `#define MOUNT` 2
- `#define UNMOUNT` 3
- `#define FILE_NOT_FOUND` 4
- `#define FILE_SYSTEM` 5
- `#define MEMORY_ERROR` 6
- `#define INVALID_FD` 7
- `#define PERMISSION_DENIED` 8
- `#define INVALID_ARGS` 9
- `#define INVALID_PATH` 10
- `#define INVALID_OFFSET` 11
- `#define INVALID_WHENCE` 12
- `#define INVALID_OPERATION` 13
- `#define FS_NOT_MOUNTED` 14
- `#define FS_ALREADY_MOUNTED` 15
- `#define NOT_A_DIRECTORY` 16
- `#define NO_SPACE` 17

Functions

- `char * error_case` (void)
Returns a string description of the current error.
- `void f_perror` (char *message)
Prints an error message to the standard error stream.

Variables

- `int ERRNO`

5.3.1 Macro Definition Documentation

5.3.1.1 FILE_NOT_FOUND

```
#define FILE_NOT_FOUND 4
```

5.3.1.2 FILE_SYSTEM

```
#define FILE_SYSTEM 5
```

5.3.1.3 FS_ALREADY_MOUNTED

```
#define FS_ALREADY_MOUNTED 15
```

5.3.1.4 FS_NOT_MOUNTED

```
#define FS_NOT_MOUNTED 14
```

5.3.1.5 FS_SUCCESS

```
#define FS_SUCCESS 0
```

5.3.1.6 INVALID_ARGS

```
#define INVALID_ARGS 9
```

5.3.1.7 INVALID_FD

```
#define INVALID_FD 7
```

5.3.1.8 INVALID_OFFSET

```
#define INVALID_OFFSET 11
```

5.3.1.9 INVALID_OPERATION

```
#define INVALID_OPERATION 13
```

5.3.1.10 INVALID_PATH

```
#define INVALID_PATH 10
```

5.3.1.11 INVALID_WHENCE

```
#define INVALID_WHENCE 12
```

5.3.1.12 MEMORY_ERROR

```
#define MEMORY_ERROR 6
```

5.3.1.13 MKFS

```
#define MKFS 1
```

5.3.1.14 MOUNT

```
#define MOUNT 2
```

5.3.1.15 NO_SPACE

```
#define NO_SPACE 17
```

5.3.1.16 NOT_A_DIRECTORY

```
#define NOT_A_DIRECTORY 16
```

5.3.1.17 PERMISSION_DENIED

```
#define PERMISSION_DENIED 8
```

5.3.1.18 UNMOUNT

```
#define UNMOUNT 3
```

5.3.2 Function Documentation

5.3.2.1 error_case()

```
char * error_case (  
    void )
```

Returns a string description of the current error.

Returns

A string description of the current error.

5.3.2.2 f_perror()

```
void f_perror (  
    char * message)
```

Prints an error message to the standard error stream.

Parameters

<i>message</i>	The message to print.
----------------	-----------------------

5.3.3 Variable Documentation

5.3.3.1 ERRNO

```
int ERRNO [extern]
```

5.4 err.h

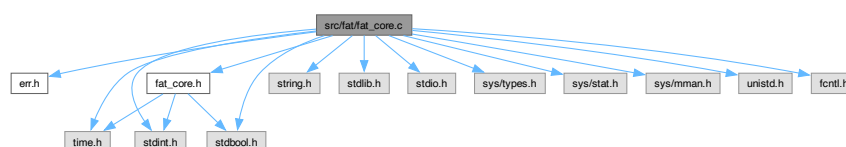
[Go to the documentation of this file.](#)

```
00001 #ifndef ERR_H
00002 #define ERR_H
00003
00004 #define FS_SUCCESS 0
00005 #define MKFS 1
00006 #define MOUNT 2
00007 #define UNMOUNT 3
00008 #define FILE_NOT_FOUND 4
00009 #define FILE_SYSTEM 5
00010 #define MEMORY_ERROR 6
00011 #define INVALID_FD 7
00012 #define PERMISSION_DENIED 8
00013 #define INVALID_ARGS 9
00014 #define INVALID_PATH 10
00015 #define INVALID_OFFSET 11
00016 #define INVALID_WHENCE 12
00017 #define INVALID_OPERATION 13
00018 #define FS_NOT_MOUNTED 14
00019 #define FS_ALREADY_MOUNTED 15
00020 #define NOT_A_DIRECTORY 16
00021 #define NO_SPACE 17
00022
00023 extern int ERRNO;
00024
00030 char *error_case(void);
00031
00037 void f_perror(char *message);
00038
00039 #endif
```

5.5 src/fat/fat_core.c File Reference

```
#include "err.h"
#include "fat_core.h"
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>
```

Include dependency graph for fat_core.c:



Functions

- int [convert_block_size](#) (int input)
- int [fs_create](#) (char *fs_name, int new_fat_block_count, int block_config)
- int [fs_mount](#) (char *fs_name)
- int [fs_unmount](#) ()
- [Dir_entry](#) [get_directory](#) (void)
- int [lookup_directory_offset](#) (char *name, int block)
- [Dir_entry](#) [offset_to_directory](#) (int offset)
- int [update_directory](#) ([Dir_entry](#) ent, int offset)
- [Dir_entry](#) [name_to_directory](#) (char *name)
- [Dir_entry](#) [index_to_directory](#) (int index)
- int [get_file_permission](#) (char *name)
- void [perm_to_rwx](#) (int perm, char *permission_str)
- void [format_file_info](#) ([Dir_entry](#) entry, char *buffer)
- void [fs_list_files](#) (const char *filename, [file_info_callback_t](#) callback, void *user_data)
List files in the filesystem, invoking a callback for each entry.
- int [create_file](#) (char *filename, uint8_t type)
- int [fs_touch](#) (char *file_name)
- int [fs_rm](#) (char *file_name)
- int [fs_mv](#) (char *source, char *dest)
- int [read_file](#) ([Dir_entry](#) ent, uint8_t *arr, int num_bytes, int buff_pos, int file_pos)
- int [write_file](#) ([Dir_entry](#) ent, uint8_t *arr, int num_bytes, int start)
- int [k_read_at](#) ([Dir_entry](#) ent, char *buf, int n, int offset)
- int [k_write_at](#) ([Dir_entry](#) ent, const char *str, int n, int offset)
- int [fs_chmod](#) (char *filename, uint8_t new_perm)

5.5.1 Function Documentation

5.5.1.1 [convert_block_size\(\)](#)

```
int convert_block_size (
    int input)
```

maps block size configuration to actual byte size

Parameters

<i>input</i>	block size configuration (0-4)
--------------	--------------------------------

Returns

size in bytes or -1 if invalid input

5.5.1.2 [create_file\(\)](#)

```
int create_file (
    char * filename,
    uint8_t type)
```

creates a new file with specified type

Parameters

<i>filename</i>	name of the file to create
<i>type</i>	file type (regular, directory, etc.)

Returns

1 on success, -1 on failure

5.5.1.3 format_file_info()

```
void format_file_info (  
    Dir_entry entry,  
    char * buffer)
```

5.5.1.4 fs_chmod()

```
int fs_chmod (  
    char * filename,  
    uint8_t new_perm)
```

changes permissions of a file

Parameters

<i>filename</i>	name of the file
<i>new_perm</i>	new permission value

Returns

1 on success, -1 on failure

5.5.1.5 fs_create()

```
int fs_create (  
    char * fs_name,  
    int new_fat_block_count,  
    int block_config)
```

creates a new pennfat filesystem

Parameters

<i>fs_name</i>	name of the filesystem file
<i>new_fat_block_count</i>	number of blocks in FAT region
<i>block_config</i>	block size configuration (0-4)

Returns

1 on success, -1 on failure

5.5.1.6 fs_list_files()

```
void fs_list_files (
    const char * filename,
    file_info_callback_t callback,
    void * user_data)
```

List files in the filesystem, invoking a callback for each entry.

Parameters

<i>filename</i>	If not NULL, only list that file; else list all files.
<i>callback</i>	Function called for each file entry found.
<i>user_data</i>	Opaque pointer passed to callback.

5.5.1.7 fs_mount()

```
int fs_mount (
    char * fs_name)
```

mounts a pennfat filesystem into memory

Parameters

<i>fs_name</i>	name of the filesystem file to mount
----------------	--------------------------------------

Returns

0 on success, -1 on failure

5.5.1.8 fs_mv()

```
int fs_mv (
    char * source,
    char * dest)
```

renames a file in the filesystem

Parameters

<i>source</i>	original filename
<i>dest</i>	new filename

Returns

1 on success, -1 on failure

5.5.1.9 fs_rm()

```
int fs_rm (
    char * file_name)
```

removes a file from the filesystem

Parameters

<i>file_name</i>	name of the file to remove
------------------	----------------------------

Returns

1 on success, -1 on failure

5.5.1.10 fs_touch()

```
int fs_touch (
    char * file_name)
```

creates a file or updates its timestamp if it exists

Parameters

<i>file_name</i>	name of the file to touch
------------------	---------------------------

Returns

1 on success, -1 on failure

5.5.1.11 fs_unmount()

```
int fs_unmount (
    void )
```

unmounts the currently mounted filesystem

Returns

0 on success, -1 on failure

5.5.1.12 get_directory()

```
Dir_entry get_directory (
    void )
```

gets the root directory entry

Returns

root directory entry

5.5.1.13 get_file_permission()

```
int get_file_permission (
    char * name)
```

gets permission value for a file

Parameters

<i>name</i>	filename
-------------	----------

Returns

permission value or -1 if file not found

5.5.1.14 index_to_directory()

```
Dir_entry index_to_directory (  
    int index)
```

gets directory entry by index

Parameters

<i>index</i>	index of directory entry
--------------	--------------------------

Returns

directory entry at specified index or empty entry if not found

5.5.1.15 k_read_at()

```
int k_read_at (  
    Dir_entry ent,  
    char * buf,  
    int n,  
    int offset)
```

reads data from a file with explicit offset

Parameters

<i>ent</i>	directory entry for the file
<i>arr</i>	buffer to store read data
<i>num_bytes</i>	number of bytes to read
<i>file_pos</i>	position in file to start reading from

Returns

number of bytes read or -1 on error

5.5.1.16 k_write_at()

```
int k_write_at (  
    Dir_entry ent,  
    const char * str,  
    int n,  
    int offset)
```

writes data to a file with explicit offset

Parameters

<i>ent</i>	directory entry for the file
<i>str</i>	buffer containing data to write
<i>n</i>	number of bytes to write
<i>offset</i>	position in file to start writing at

Returns

number of bytes written or -1 on error

5.5.1.17 lookup_directory_offset()

```
int lookup_directory_offset (  
    char * name,  
    int block)
```

finds the location of a directory entry by name

Parameters

<i>name</i>	filename to find
<i>block</i>	block to start search from

Returns

offset of directory entry or -1 if not found

5.5.1.18 name_to_directory()

```
Dir_entry name_to_directory (  
    char * name)
```

gets directory entry by filename

Parameters

<i>name</i>	filename to find
-------------	------------------

Returns

directory entry for specified file or empty entry if not found

5.5.1.19 offset_to_directory()

```
Dir_entry offset_to_directory (  
    int offset)
```

gets directory entry at specified offset

Parameters

<i>offset</i>	file offset of directory entry
---------------	--------------------------------

Returns

directory entry at specified offset

5.5.1.20 perm_to_rwx()

```
void perm_to_rwx (  
    int perm,  
    char * permission_str)
```

converts permission value to string representation

Parameters

<i>perm</i>	permission value
<i>permission_str</i>	output string buffer (should be at least 4 bytes)

5.5.1.21 read_file()

```
int read_file (  
    Dir_entry ent,  
    uint8_t * arr,  
    int num_bytes,  
    int buff_pos,  
    int file_pos)
```

reads data from a file

Parameters

<i>ent</i>	directory entry for the file
<i>arr</i>	buffer to store read data
<i>num_bytes</i>	number of bytes to read
<i>buff_pos</i>	starting position in buffer
<i>file_pos</i>	starting position in file

Returns

number of bytes read or -1 on error

5.5.1.22 update_directory()

```
int update_directory (  
    Dir_entry ent,  
    int offset)
```

updates a directory entry

Parameters

<i>ent</i>	updated directory entry
<i>offset</i>	offset of entry to update

Returns

1 on success, -1 on failure

5.5.1.23 write_file()

```
int write_file (
    Dir_entry ent,
    uint8_t * arr,
    int num_bytes,
    int start)
```

writes data to a file

Parameters

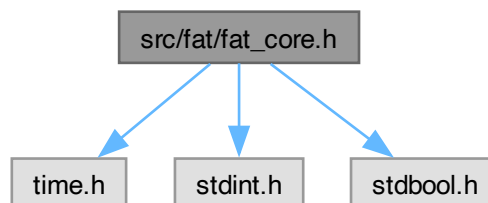
<i>ent</i>	directory entry for the file
<i>arr</i>	buffer containing data to write
<i>num_bytes</i>	number of bytes to write
<i>start</i>	starting position in file

Returns

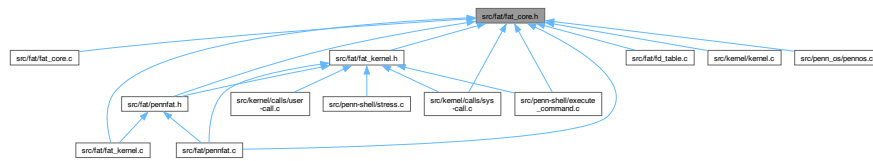
number of bytes written or -1 on error

5.6 src/fat/fat_core.h File Reference

```
#include <time.h>
#include <stdint.h>
#include <stdbool.h>
Include dependency graph for fat_core.h:
```



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



Classes

- struct [dir_entry](#)
- struct [buffer](#)

Macros

- `#define UNKNOWN_FILE 0`
- `#define REGULAR_FILE 1`
- `#define DIRECTORY_FILE 2`
- `#define LINK_FILE 4`
- `#define EOD_FLAG 0x00`
- `#define LAST_BLOCK 0xFFFF`
- `#define FREE_BLOCK 0x0000`
- `#define SEEK_SET 0`
- `#define SEEK_CUR 1`
- `#define SEEK_END 2`

Typedefs

- `typedef struct dir_entry Dir_entry`
- `typedef struct buffer Buffer`
- `typedef void(* file_info_callback_t) (const Dir_entry *, void *)`

Functions

- `int convert_block_size (int input)`
- `int fs_create (char *fs_name, int new_fat_block_count, int block_config)`
- `int fs_mount (char *fs_name)`
- `int fs_unmount (void)`
- `int fs_touch (char *file_name)`
- `int fs_rm (char *file_name)`
- `int fs_mv (char *source, char *dest)`
- `int create_file (char *filename, uint8_t type)`
- `int read_file (Dir_entry ent, uint8_t *arr, int num_bytes, int buff_pos, int file_pos)`
- `int write_file (Dir_entry ent, uint8_t *arr, int num_bytes, int start)`
- `int k_read_at (Dir_entry ent, char *buf, int n, int offset)`
- `int k_write_at (Dir_entry ent, const char *str, int n, int offset)`
- `Dir_entry get_directory (void)`
- `int lookup_directory_offset (char *name, int block)`
- `Dir_entry offset_to_directory (int offset)`

- [Dir_entry name_to_directory](#) (char *name)
- [Dir_entry index_to_directory](#) (int index)
- void [fs_list_files](#) (const char *filename, [file_info_callback_t](#) callback, void *user_data)
List files in the filesystem, invoking a callback for each entry.
- void [format_file_info](#) ([Dir_entry](#) entry, char *buffer)
- int [update_directory](#) ([Dir_entry](#) ent, int offset)
- void [perm_to_rwx](#) (int perm, char *permission_str)
- int [get_file_permission](#) (char *name)
- int [fs_chmod](#) (char *filename, uint8_t new_perm)

5.6.1 Macro Definition Documentation

5.6.1.1 DIRECTORY_FILE

```
#define DIRECTORY_FILE 2
```

5.6.1.2 EOD_FLAG

```
#define EOD_FLAG 0x00
```

5.6.1.3 FREE_BLOCK

```
#define FREE_BLOCK 0x0000
```

5.6.1.4 LAST_BLOCK

```
#define LAST_BLOCK 0xFFFF
```

5.6.1.5 LINK_FILE

```
#define LINK_FILE 4
```

5.6.1.6 REGULAR_FILE

```
#define REGULAR_FILE 1
```

5.6.1.7 SEEK_CUR

```
#define SEEK_CUR 1
```

5.6.1.8 SEEK_END

```
#define SEEK_END 2
```

5.6.1.9 SEEK_SET

```
#define SEEK_SET 0
```

5.6.1.10 UNKNOWN_FILE

```
#define UNKNOWN_FILE 0
```

5.6.2 Typedef Documentation

5.6.2.1 Buffer

```
typedef struct buffer Buffer
```

buffer structure for holding data

Parameters

<i>arr</i>	pointer to data array
<i>size</i>	size of buffer in bytes

5.6.2.2 Dir_entry

```
typedef struct dir\_entry Dir_entry
```

directory entry structure representing a file in the filesystem

Parameters

<i>name</i>	file name <code>name[0]</code> is a special marker <ul style="list-style-type: none">• 0: end of directory• 1: deleted entry; the file is also deleted• 2: deleted entry; the file is still being used
<i>size</i>	number of bytes in file
<i>first_block</i>	first block number of file (undefined if 0 size)
<i>type</i>	type of file <ul style="list-style-type: none">• 0: unknown• 1: regular file• 2: directory file• 3: symbolic link

Parameters

<i>perm</i>	file permissions <ul style="list-style-type: none"> • 0: none • 2: write only • 4: read only • 5: read and executable (shell scripts) • 6: read and write • 7: read, write, and executable
<i>mtime</i>	creation/modification time
<i>reserved</i>	reserved bytes (last 16)

5.6.2.3 file_info_callback_t

```
typedef void(* file_info_callback_t) (const Dir_entry *, void *)
```

5.6.3 Function Documentation

5.6.3.1 convert_block_size()

```
int convert_block_size (
    int input)
```

maps block size configuration to actual byte size

Parameters

<i>input</i>	block size configuration (0-4)
--------------	--------------------------------

Returns

size in bytes or -1 if invalid input

5.6.3.2 create_file()

```
int create_file (
    char * filename,
    uint8_t type)
```

creates a new file with specified type

Parameters

<i>filename</i>	name of the file to create
<i>type</i>	file type (regular, directory, etc.)

Returns

1 on success, -1 on failure

5.6.3.3 format_file_info()

```
void format_file_info (
    Dir_entry entry,
    char * buffer)
```

5.6.3.4 fs_chmod()

```
int fs_chmod (
    char * filename,
    uint8_t new_perm)
```

changes permissions of a file

Parameters

<i>filename</i>	name of the file
<i>new_perm</i>	new permission value

Returns

1 on success, -1 on failure

5.6.3.5 fs_create()

```
int fs_create (
    char * fs_name,
    int new_fat_block_count,
    int block_config)
```

creates a new pennfat filesystem

Parameters

<i>fs_name</i>	name of the filesystem file
<i>new_fat_block_count</i>	number of blocks in FAT region
<i>block_config</i>	block size configuration (0-4)

Returns

1 on success, -1 on failure

5.6.3.6 fs_list_files()

```
void fs_list_files (
    const char * filename,
    file_info_callback_t callback,
    void * user_data)
```

List files in the filesystem, invoking a callback for each entry.

Parameters

<i>filename</i>	If not NULL, only list that file; else list all files.
<i>callback</i>	Function called for each file entry found.
<i>user_data</i>	Opaque pointer passed to callback.

5.6.3.7 fs_mount()

```
int fs_mount (  
    char * fs_name)
```

mounts a pennfat filesystem into memory

Parameters

<i>fs_name</i>	name of the filesystem file to mount
----------------	--------------------------------------

Returns

0 on success, -1 on failure

5.6.3.8 fs_mv()

```
int fs_mv (  
    char * source,  
    char * dest)
```

renames a file in the filesystem

Parameters

<i>source</i>	original filename
<i>dest</i>	new filename

Returns

1 on success, -1 on failure

5.6.3.9 fs_rm()

```
int fs_rm (  
    char * file_name)
```

removes a file from the filesystem

Parameters

<i>file_name</i>	name of the file to remove
------------------	----------------------------

Returns

1 on success, -1 on failure

5.6.3.10 fs_touch()

```
int fs_touch (
    char * file_name)
```

creates a file or updates its timestamp if it exists

Parameters

<i>file_name</i>	name of the file to touch
------------------	---------------------------

Returns

1 on success, -1 on failure

5.6.3.11 fs_unmount()

```
int fs_unmount (
    void )
```

unmounts the currently mounted filesystem

Returns

0 on success, -1 on failure

5.6.3.12 get_directory()

```
Dir_entry get_directory (
    void )
```

gets the root directory entry

Returns

root directory entry

5.6.3.13 get_file_permission()

```
int get_file_permission (
    char * name)
```

gets permission value for a file

Parameters

<i>name</i>	filename
-------------	----------

Returns

permission value or -1 if file not found

5.6.3.14 index_to_directory()

```
Dir_entry index_to_directory (  
    int index)
```

gets directory entry by index

Parameters

<i>index</i>	index of directory entry
--------------	--------------------------

Returns

directory entry at specified index or empty entry if not found

5.6.3.15 k_read_at()

```
int k_read_at (  
    Dir_entry ent,  
    char * buf,  
    int n,  
    int offset)
```

reads data from a file with explicit offset

Parameters

<i>ent</i>	directory entry for the file
<i>arr</i>	buffer to store read data
<i>num_bytes</i>	number of bytes to read
<i>file_pos</i>	position in file to start reading from

Returns

number of bytes read or -1 on error

5.6.3.16 k_write_at()

```
int k_write_at (  
    Dir_entry ent,  
    const char * str,  
    int n,  
    int offset)
```

writes data to a file with explicit offset

Parameters

<i>ent</i>	directory entry for the file
<i>str</i>	buffer containing data to write
<i>n</i>	number of bytes to write
<i>offset</i>	position in file to start writing at

Returns

number of bytes written or -1 on error

5.6.3.17 lookup_directory_offset()

```
int lookup_directory_offset (  
    char * name,  
    int block)
```

finds the location of a directory entry by name

Parameters

<i>name</i>	filename to find
<i>block</i>	block to start search from

Returns

offset of directory entry or -1 if not found

5.6.3.18 name_to_directory()

```
Dir_entry name_to_directory (  
    char * name)
```

gets directory entry by filename

Parameters

<i>name</i>	filename to find
-------------	------------------

Returns

directory entry for specified file or empty entry if not found

5.6.3.19 offset_to_directory()

```
Dir_entry offset_to_directory (  
    int offset)
```

gets directory entry at specified offset

Parameters

<i>offset</i>	file offset of directory entry
---------------	--------------------------------

Returns

directory entry at specified offset

5.6.3.20 perm_to_rwx()

```
void perm_to_rwx (  
    int perm,  
    char * permission_str)
```

converts permission value to string representation

Parameters

<i>perm</i>	permission value
<i>permission_str</i>	output string buffer (should be at least 4 bytes)

5.6.3.21 read_file()

```
int read_file (  
    Dir_entry ent,  
    uint8_t * arr,  
    int num_bytes,  
    int buff_pos,  
    int file_pos)
```

reads data from a file

Parameters

<i>ent</i>	directory entry for the file
<i>arr</i>	buffer to store read data
<i>num_bytes</i>	number of bytes to read
<i>buff_pos</i>	starting position in buffer
<i>file_pos</i>	starting position in file

Returns

number of bytes read or -1 on error

5.6.3.22 update_directory()

```
int update_directory (  
    Dir_entry ent,  
    int offset)
```

updates a directory entry

Parameters

<i>ent</i>	updated directory entry
<i>offset</i>	offset of entry to update

Returns

1 on success, -1 on failure

5.6.3.23 write_file()

```
int write_file (  
    Dir_entry ent,  
    uint8_t * arr,  
    int num_bytes,  
    int start)
```

writes data to a file

Parameters

<i>ent</i>	directory entry for the file
<i>arr</i>	buffer containing data to write
<i>num_bytes</i>	number of bytes to write
<i>start</i>	starting position in file

Returns

number of bytes written or -1 on error

5.7 fat_core.h

[Go to the documentation of this file.](#)

```
00001 #ifndef FAT_CORE_H  
00002 #define FAT_CORE_H  
00003  
00004 #include <time.h>  
00005 #include <stdint.h>  
00006 #include <stdbool.h>  
00007  
00008 /* File types */  
00009 #define UNKNOWN_FILE 0  
00010 #define REGULAR_FILE 1  
00011 #define DIRECTORY_FILE 2  
00012 #define LINK_FILE 4  
00013  
00014 /* Special flags */  
00015 #define EOD_FLAG 0x00  
00016  
00017 /* Block status values */  
00018 #define LAST_BLOCK 0xFFFF  
00019 #define FREE_BLOCK 0x0000  
00020  
00021 /* File positioning constants */  
00022 #define SEEK_SET 0  
00023 #define SEEK_CUR 1  
00024 #define SEEK_END 2  
00025
```

```

00050 typedef struct dir_entry
00051 {
00052     char name[32];
00053     uint32_t size;
00054     uint16_t first_block;
00055     uint8_t type;
00056     uint8_t perm;
00057     time_t mtime;
00058     long double reserved;
00059 } Dir_entry;
00060
00066 typedef struct buffer
00067 {
00068     uint8_t *arr;
00069     int size;
00070 } Buffer;
00071
00072 /*
00073  * Filesystem core management functions
00074  */
00075
00081 int convert_block_size(int input);
00082
00090 int fs_create(char *fs_name, int new_fat_block_count, int block_config);
00091
00097 int fs_mount(char *fs_name);
00098
00103 int fs_unmount(void);
00104
00105 /*
00106  * File operation functions
00107  */
00108
00114 int fs_touch(char *file_name);
00115
00121 int fs_rm(char *file_name);
00122
00129 int fs_mv(char *source, char *dest);
00130
00137 int create_file(char *filename, uint8_t type);
00138
00139 /*
00140  * File data read/write functions
00141  */
00142
00152 int read_file(Dir_entry ent, uint8_t *arr, int num_bytes, int buff_pos, int file_pos);
00153
00162 int write_file(Dir_entry ent, uint8_t *arr, int num_bytes, int start);
00163
00172 int k_read_at(Dir_entry ent, char *buf, int n, int offset);
00173
00182 int k_write_at(Dir_entry ent, const char *str, int n, int offset);
00183
00184 /*
00185  * Directory entry management functions
00186  */
00187
00192 Dir_entry get_directory(void);
00193
00200 int lookup_directory_offset(char *name, int block);
00201
00207 Dir_entry offset_to_directory(int offset);
00208
00214 Dir_entry name_to_directory(char *name);
00215
00221 Dir_entry index_to_directory(int index);
00222
00223 // Callback type for file listing
00224 typedef void (*file_info_callback_t)(const Dir_entry *, void *);
00225
00232 void fs_list_files(const char *filename, file_info_callback_t callback, void *user_data);
00233
00234 // Formats a Dir_entry into a printable string for ls output
00235 void format_file_info(Dir_entry entry, char *buffer);
00236
00243 int update_directory(Dir_entry ent, int offset);
00244
00245 /*
00246  * Utility functions
00247  */
00248
00254 void perm_to_rwx(int perm, char *permission_str);
00255
00261 int get_file_permission(char *name);
00262
00269 int fs_chmod(char *filename, uint8_t new_perm);
00270

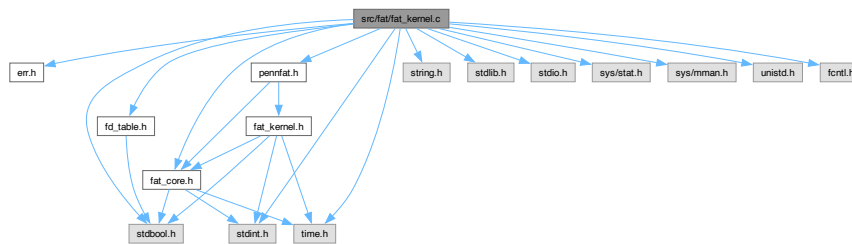
```

```
00271 #endif
```

5.8 src/fat/fat_kernel.c File Reference

```
#include "err.h"
#include "fd_table.h"
#include "pennfat.h"
#include "fat_core.h"
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>
```

Include dependency graph for fat_kernel.c:



Macros

- #define `MAX_LINE_LENGTH` 4096

Functions

- void `init` ()
- int `k_open` (const char *fname, int mode)
- int `kf_read` (int fd, int n, char *buf)
- int `kf_write` (int fd, const char *str, int n)
- int `k_read_at_offset` (int fd, int n, char *buf, int offset)
- int `k_write_at_offset` (int fd, const char *str, int n, int offset)
- int `k_close` (int fd)
- void `k_unlink` (const char *fname)
- void `k_lseek` (int fd, int offset, int whence)
- void `k_ls` (const char *filename)
- void `f_touch` (char *args[])
- void `f_rm` (char *args[])
- void `f_mv` (char *args[])
- void `f_cat` (char *args[], int fd_in, int fd_out)
- void `f_cp` (char *args[])
- int `f_get_permission` (char *file_name)
- int `k_chmod` (const char *fname, uint8_t new_perm)
- void `f_chmod` (char *args[])
- int `k_get_permission` (const char *fname)

5.8.1 Macro Definition Documentation

5.8.1.1 MAX_LINE_LENGTH

```
#define MAX_LINE_LENGTH 4096
```

5.8.2 Function Documentation

5.8.2.1 f_cat()

```
void f_cat (  
    char * args[],  
    int fd_in,  
    int fd_out)
```

concatenates files or displays input

Parameters

<i>args</i>	command arguments array (args[0] is command name)
<i>fd_in</i>	input file descriptor
<i>fd_out</i>	output file descriptor

5.8.2.2 f_chmod()

```
void f_chmod (  
    char * args[])
```

changes file permissions

Parameters

<i>args</i>	command arguments array (args[0] is command name)
-------------	---

5.8.2.3 f_cp()

```
void f_cp (  
    char * args[])
```

copies a file

Parameters

<i>args</i>	command arguments array (args[0] is command name)
-------------	---

5.8.2.4 f_get_permission()

```
int f_get_permission (  
    char * file_name)
```

gets permission value for a file

Parameters

<i>file_name</i>	name of the file
------------------	------------------

Returns

permission value or -1 if file not found

5.8.2.5 f_mv()

```
void f_mv (  
    char * args[])
```

renames a file

Parameters

<i>args</i>	command arguments array (args[0] is command name)
-------------	---

5.8.2.6 f_rm()

```
void f_rm (  
    char * args[])
```

removes files from the filesystem

Parameters

<i>args</i>	command arguments array (args[0] is command name)
-------------	---

5.8.2.7 f_touch()

```
void f_touch (  
    char * args[])
```

creates empty files or updates timestamps of existing files

Parameters

<i>args</i>	command arguments array (args[0] is command name)
-------------	---

5.8.2.8 init()

```
void init (  
    void )
```

initializes the pennfat system

5.8.2.9 k_chmod()

```
int k_chmod (  
    const char * fname,  
    uint8_t new_perm)
```

changes permissions of a file

Parameters

<i>fname</i>	name of the file
<i>new_perm</i>	new permission value

Returns

1 on success, -1 on failure

5.8.2.10 k_close()

```
int k_close (  
    int fd)
```

closes a file descriptor

Parameters

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

Returns

0 on success, negative value on failure

5.8.2.11 k_get_permission()

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

gets permission value for a file

Parameters

<i>fname</i>	name of the file
--------------	------------------

Returns

permission value or -1 if file not found

5.8.2.12 k_ls()

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

lists file(s) in the filesystem

Parameters

<i>filename</i>	specific file to list, or NULL for all files
-----------------	--

5.8.2.13 k_lseek()

```
void k_lseek (  
    int fd,  
    int offset,  
    int whence)
```

repositions the file offset

Parameters

<i>fd</i>	file descriptor
<i>offset</i>	offset value
<i>whence</i>	reference position (SEEK_SET, SEEK_CUR, SEEK_END)

5.8.2.14 k_open()

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

opens a file with specified mode

Parameters

<i>fname</i>	name of the file to open
<i>mode</i>	access mode (F_READ, F_WRITE, F_APPEND)

Returns

file descriptor on success, negative value on failure

5.8.2.15 k_read_at_offset()

```
int k_read_at_offset (  
    int fd,  
    int n,  
    char * buf,  
    int offset)
```

reads data from a file at specified offset

Parameters

<i>fd</i>	file descriptor
<i>n</i>	maximum number of bytes to read
<i>buf</i>	buffer to store read data
<i>offset</i>	position in file to start reading from

Returns

number of bytes read, 0 on EOF, negative value on error

5.8.2.16 k_unlink()

```
void k_unlink (  
    const char * fname)
```

removes a file from the filesystem

Parameters

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

5.8.2.17 k_write_at_offset()

```
int k_write_at_offset (  
    int fd,  
    const char * str,  
    int n,  
    int offset)
```

writes data to a file at specified offset

Parameters

<i>fd</i>	file descriptor
<i>str</i>	buffer containing data to write
<i>n</i>	number of bytes to write
<i>offset</i>	position in file to start writing at

Returns

number of bytes written, negative value on error

5.8.2.18 kf_read()

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

reads data from a file

Parameters

<i>fd</i>	file descriptor
<i>n</i>	maximum number of bytes to read
<i>buf</i>	buffer to store read data

Returns

number of bytes read, 0 on EOF, negative value on error

5.8.2.19 kf_write()

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

writes data to a file

Parameters

<i>fd</i>	file descriptor
<i>str</i>	buffer containing data to write
<i>n</i>	number of bytes to write

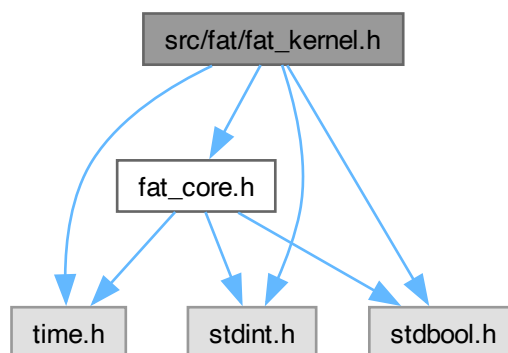
Returns

number of bytes written, negative value on error

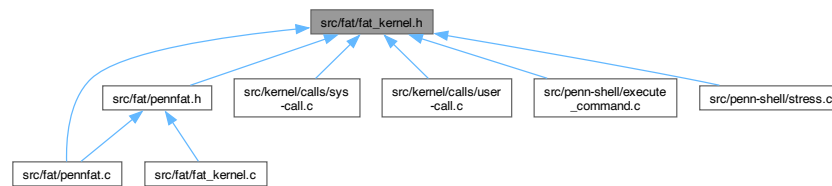
5.9 src/fat/fat_kernel.h File Reference

```
#include "fat_core.h"
#include <time.h>
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for fat_kernel.h:



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



Macros

- `#define F_READ 0`
- `#define F_WRITE 1`
- `#define F_APPEND 2`
- `#define STDIN_FILENO 0`
- `#define STDOUT_FILENO 1`
- `#define STDERR_FILENO 2`
- `#define SEEK_SET 0`
- `#define SEEK_CUR 1`
- `#define SEEK_END 2`

Functions

- `void init (void)`
- `int k_open (const char *fname, int mode)`
- `int kf_read (int fd, int n, char *buf)`
- `int kf_write (int fd, const char *str, int n)`
- `int k_read_at_offset (int fd, int n, char *buf, int offset)`
- `int k_write_at_offset (int fd, const char *str, int n, int offset)`
- `int k_close (int fd)`
- `void k_unlink (const char *fname)`
- `void k_lseek (int fd, int offset, int whence)`
- `void k_ls (const char *filename)`
- `void f_touch (char *args[])`
- `void f_rm (char *args[])`
- `void f_mv (char *args[])`
- `void f_cat (char *args[], int fd_in, int fd_out)`
- `void f_cp (char *args[])`
- `void f_chmod (char *args[])`
- `int f_get_permission (char *file_name)`
- `int k_chmod (const char *fname, uint8_t new_perm)`
- `int k_get_permission (const char *fname)`

5.9.1 Macro Definition Documentation

5.9.1.1 F_APPEND

```
#define F_APPEND 2
```

5.9.1.2 F_READ

```
#define F_READ 0
```

5.9.1.3 F_WRITE

```
#define F_WRITE 1
```

5.9.1.4 SEEK_CUR

```
#define SEEK_CUR 1
```

5.9.1.5 SEEK_END

```
#define SEEK_END 2
```

5.9.1.6 SEEK_SET

```
#define SEEK_SET 0
```

5.9.1.7 STDERR_FILENO

```
#define STDERR_FILENO 2
```

5.9.1.8 STDIN_FILENO

```
#define STDIN_FILENO 0
```

5.9.1.9 STDOUT_FILENO

```
#define STDOUT_FILENO 1
```

5.9.2 Function Documentation

5.9.2.1 f_cat()

```
void f_cat (  
    char * args[],  
    int fd_in,  
    int fd_out)
```

concatenates files or displays input

Parameters

<i>args</i>	command arguments array (args[0] is command name)
<i>fd_in</i>	input file descriptor
<i>fd_out</i>	output file descriptor

5.9.2.2 f_chmod()

```
void f_chmod (  
    char * args[])
```

changes file permissions

Parameters

<i>args</i>	command arguments array (args[0] is command name)
-------------	---

5.9.2.3 f_cp()

```
void f_cp (  
    char * args[])
```

copies a file

Parameters

<i>args</i>	command arguments array (args[0] is command name)
-------------	---

5.9.2.4 f_get_permission()

```
int f_get_permission (  
    char * file_name)
```

gets permission value for a file

Parameters

<i>file_name</i>	name of the file
------------------	------------------

Returns

permission value or -1 if file not found

5.9.2.5 f_mv()

```
void f_mv (  
    char * args[])
```

renames a file

Parameters

<i>args</i>	command arguments array (<i>args</i> [0] is command name)
-------------	--

5.9.2.6 f_rm()

```
void f_rm (
    char * args[])
```

removes files from the filesystem

Parameters

<i>args</i>	command arguments array (<i>args</i> [0] is command name)
-------------	--

5.9.2.7 f_touch()

```
void f_touch (
    char * args[])
```

creates empty files or updates timestamps of existing files

Parameters

<i>args</i>	command arguments array (<i>args</i> [0] is command name)
-------------	--

5.9.2.8 init()

```
void init (
    void )
```

initializes the pennfat system

5.9.2.9 k_chmod()

```
int k_chmod (
    const char * fname,
    uint8_t new_perm)
```

changes permissions of a file

Parameters

<i>fname</i>	name of the file
<i>new_perm</i>	new permission value

Returns

1 on success, -1 on failure

5.9.2.10 `k_close()`

```
int k_close (  
    int fd)
```

closes a file descriptor

Parameters

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

Returns

0 on success, negative value on failure

5.9.2.11 k_get_permission()

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

gets permission value for a file

Parameters

<i>fname</i>	name of the file
--------------	------------------

Returns

permission value or -1 if file not found

5.9.2.12 k_ls()

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

lists file(s) in the filesystem

Parameters

<i>filename</i>	specific file to list, or NULL for all files
-----------------	--

5.9.2.13 k_lseek()

```
void k_lseek (  
    int fd,  
    int offset,  
    int whence)
```

repositions the file offset

Parameters

<i>fd</i>	file descriptor
<i>offset</i>	offset value
<i>whence</i>	reference position (SEEK_SET, SEEK_CUR, SEEK_END)

5.9.2.14 k_open()

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

opens a file with specified mode

Parameters

<i>fname</i>	name of the file to open
<i>mode</i>	access mode (F_READ, F_WRITE, F_APPEND)

Returns

file descriptor on success, negative value on failure

5.9.2.15 k_read_at_offset()

```
int k_read_at_offset (  
    int fd,  
    int n,  
    char * buf,  
    int offset)
```

reads data from a file at specified offset

Parameters

<i>fd</i>	file descriptor
<i>n</i>	maximum number of bytes to read
<i>buf</i>	buffer to store read data
<i>offset</i>	position in file to start reading from

Returns

number of bytes read, 0 on EOF, negative value on error

5.9.2.16 k_unlink()

```
void k_unlink (  
    const char * fname)
```

removes a file from the filesystem

Parameters

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

5.9.2.17 k_write_at_offset()

```
int k_write_at_offset (  
    int fd,  
    const char * str,  
    int n,  
    int offset)
```

writes data to a file at specified offset

Parameters

<i>fd</i>	file descriptor
<i>str</i>	buffer containing data to write
<i>n</i>	number of bytes to write
<i>offset</i>	position in file to start writing at

Returns

number of bytes written, negative value on error

5.9.2.18 kf_read()

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

reads data from a file

Parameters

<i>fd</i>	file descriptor
<i>n</i>	maximum number of bytes to read
<i>buf</i>	buffer to store read data

Returns

number of bytes read, 0 on EOF, negative value on error

5.9.2.19 kf_write()

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

writes data to a file

Parameters

<i>fd</i>	file descriptor
<i>str</i>	buffer containing data to write
<i>n</i>	number of bytes to write

Returns

number of bytes written, negative value on error

5.10 fat_kernel.h

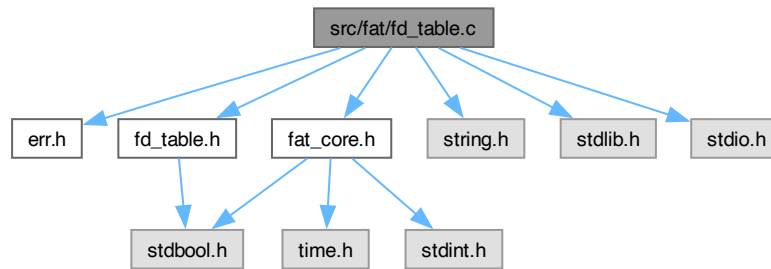
[Go to the documentation of this file.](#)

```
00001 #ifndef FAT_KERNEL_H
00002 #define FAT_KERNEL_H
00003
00004 #include "fat_core.h"
00005 #include <time.h>
00006 #include <stdint.h>
00007 #include <stdbool.h>
00008
00009 /* File opening modes */
00010 #define F_READ 0
00011 #define F_WRITE 1
00012 #define F_APPEND 2
00013
00014 /* Standard file descriptors */
00015 #define STDIN_FILENO 0
00016 #define STDOUT_FILENO 1
00017 #define STDERR_FILENO 2
00018
00019 /* File positioning constants */
00020 #define SEEK_SET 0
00021 #define SEEK_CUR 1
00022 #define SEEK_END 2
00023
00027 void init(void);
00028
00029 /*
00030  * File descriptor management functions
00031  */
00032
00039 int k_open(const char *fname, int mode);
00040
00048 int kf_read(int fd, int n, char *buf);
00049
00057 int kf_write(int fd, const char *str, int n);
00058
00067 int k_read_at_offset(int fd, int n, char *buf, int offset);
00068
00077 int k_write_at_offset(int fd, const char *str, int n, int offset);
00078
00084 int k_close(int fd);
00085
00090 void k_unlink(const char *fname);
00091
00098 void k_lseek(int fd, int offset, int whence);
00099
00100 /*
00101  * File listing functions
00102  */
00103
00108 void k_ls(const char *filename);
00109
00110 /*
00111  * File manipulation command functions
00112  */
00113
00118 void f_touch(char *args[]);
00119
00124 void f_rm(char *args[]);
00125
00130 void f_mv(char *args[]);
00131
00138 void f_cat(char *args[], int fd_in, int fd_out);
00139
00144 void f_cp(char *args[]);
00145
00150 void f_chmod(char *args[]);
00151
00157 int f_get_permission(char *file_name);
00158
00165 int k_chmod(const char *fname, uint8_t new_perm);
00166
00172 int k_get_permission(const char *fname);
00173
00174 #endif
```

5.11 src/fat/fd_table.c File Reference

```
#include "err.h"
#include "fd_table.h"
#include "fat_core.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
```

Include dependency graph for fd_table.c:



Functions

- void [initialize_fd_table](#) (FD_Table *fd_table)
- FD_Node * [lookup_add_position](#) (FD_Table *fd_table)
- FD_Node * [add_fd](#) (FD_Table *fd_table, char *name, int mode)
- FD_Node * [remove_fd](#) (FD_Table *fd_table, int fd)
- FD_Node * [lookup_fd](#) (FD_Table *fd_table, int fd)

5.11.1 Function Documentation

5.11.1.1 add_fd()

```
FD_Node * add_fd (
    FD_Table * fd_table,
    char * name,
    int mode)
```

add new [fd_node](#) to [fd_table](#)

Parameters

fd_table	pointer to fd_table linked list
name	name of file being added
mode	mode of file being added

Returns

[fd_node](#) that was added (or NULL on error)

5.11.1.2 initialize_fd_table()

```
void initialize_fd_table (  
    FD_Table * fd_table)
```

initialize a fd table for a process

Parameters

<code>fd_table</code>	pointer to <code>fd_table</code>
-----------------------	----------------------------------

5.11.1.3 lookup_add_position()

```
FD_Node * lookup_add_position (  
    FD_Table * fd_table)
```

finds correct position in the linked list to insert a new `fd_node` done in order to maintain sequential FD numbers with no gaps

Parameters

<code>fd_table</code>	Pointer to the file descriptor table
-----------------------	--------------------------------------

Returns

pointer to the node after which to insert, or NULL to insert at head

5.11.1.4 lookup_fd()

```
FD_Node * lookup_fd (  
    FD_Table * fd_table,  
    int fd)
```

find the `fd_node` with the fd number supplied as argument

Parameters

<code>fd_table</code>	<code>fd_table</code>
<code>fd</code>	fd number that is being searched for

Returns

`fd_node` being searched for (or NULL if it's not present)

5.11.1.5 remove_fd()

```
FD_Node * remove_fd (  
    FD_Table * fd_table,  
    int fd)
```

remove `fd_node` with given fd number from `fd_table` linked list

Parameters

<i>fd_table</i>	<i>fd_table</i>
<i>fd</i>	fd number being removed

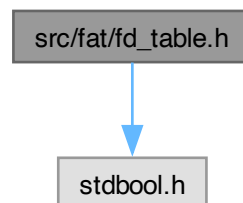
Returns

fd_node that was removed (or NULL on error)

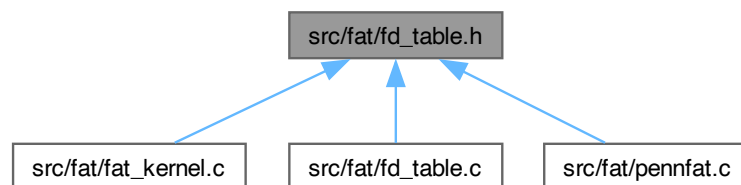
5.12 src/fat/fd_table.h File Reference

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

Include dependency graph for fd_table.h:



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



Classes

- struct *fd_node*
- struct *fd_table*

Macros

- `#define F_READ 0`
- `#define F_WRITE 1`
- `#define F_APPEND 2`

Typedefs

- `typedef struct fd_node FD_Node`
- `typedef struct fd_table FD_Table`

Functions

- `void initialize_fd_table (FD_Table *fd_table)`
- `FD_Node * lookup_add_position (FD_Table *fd_table)`
- `FD_Node * add_fd (FD_Table *fd_table, char *name, int mode)`
- `FD_Node * remove_fd (FD_Table *fd_table, int fd)`
- `FD_Node * lookup_fd (FD_Table *fd_table, int fd)`

5.12.1 Macro Definition Documentation

5.12.1.1 F_APPEND

```
#define F_APPEND 2
```

5.12.1.2 F_READ

```
#define F_READ 0
```

5.12.1.3 F_WRITE

```
#define F_WRITE 1
```

5.12.2 Typedef Documentation

5.12.2.1 FD_Node

```
typedef struct fd_node FD_Node
```

node struct to represent each file in the fd table

Parameters

<i>name</i>	file name
<i>fd</i>	fd number
<i>offset</i>	pointer offset
<i>size</i>	file size
<i>mode</i>	mode of file (0: read, 1: write, 2: append)
<i>next</i>	pointer to next node in list

5.12.2.2 FD_Table

```
typedef struct fd_table FD_Table
```

fd table struct

Parameters

<i>head</i>	head of the fd_table which is a linkedlist of fd_nodes
-------------	--

5.12.3 Function Documentation

5.12.3.1 [add_fd\(\)](#)

```
FD_Node * add_fd (  
    FD_Table * fd_table,  
    char * name,  
    int mode)
```

add new [fd_node](#) to [fd_table](#)

Parameters

<i>fd_table</i>	pointer to fd_table linked list
<i>name</i>	name of file being added
<i>mode</i>	mode of file being added

Returns

[fd_node](#) that was added (or NULL on error)

5.12.3.2 [initialize_fd_table\(\)](#)

```
void initialize_fd_table (  
    FD_Table * fd_table)
```

initialize a fd table for a process

Parameters

<i>fd_table</i>	pointer to fd_table
-----------------	-------------------------------------

5.12.3.3 [lookup_add_position\(\)](#)

```
FD_Node * lookup_add_position (  
    FD_Table * fd_table)
```

finds correct position in the linked list to insert a new [fd_node](#) done in order to maintain sequential FD numbers with no gaps

Parameters

<i>fd_table</i>	Pointer to the file descriptor table
-----------------	--------------------------------------

Returns

pointer to the node after which to insert, or NULL to insert at head

5.12.3.4 lookup_fd()

```
FD_Node * lookup_fd (
    FD_Table * fd_table,
    int fd)
```

find the `fd_node` with the fd number supplied as argument

Parameters

<code>fd_table</code>	<code>fd_table</code>
<code>fd</code>	fd number that is being searched for

Returns

`fd_node` being searched for (or NULL if it's not present)

5.12.3.5 remove_fd()

```
FD_Node * remove_fd (
    FD_Table * fd_table,
    int fd)
```

remove `fd_node` with given fd number from `fd_table` linked list

Parameters

<code>fd_table</code>	<code>fd_table</code>
<code>fd</code>	fd number being removed

Returns

`fd_node` that was removed (or NULL on error)

5.13 fd_table.h

[Go to the documentation of this file.](#)

```
00001 #ifndef FD_TABLE_H
00002 #define FD_TABLE_H
00003
00004 #include <stdbool.h>
00005
00006 struct dir_entry;
00007
00008 #define F_READ 0
00009 #define F_WRITE 1
00010 #define F_APPEND 2
00011
00012
00023 typedef struct fd_node {
00024     char* name;
00025     int fd;
00026     int offset;
00027     int size;
00028     int mode;
00029     struct fd_node* next;
```

```

00030 } FD_Node;
00031
00036 typedef struct fd_table {
00037     FD_Node* head;
00038 } FD_Table;
00039
00044 void initialize_fd_table(FD_Table* fd_table);
00045
00052 FD_Node* lookup_add_position(FD_Table* fd_table);
00053
00061 FD_Node* add_fd(FD_Table* fd_table, char* name, int mode);
00062
00069 FD_Node* remove_fd(FD_Table* fd_table, int fd);
00070
00077 FD_Node* lookup_fd(FD_Table* fd_table, int fd);
00078
00079 #endif

```

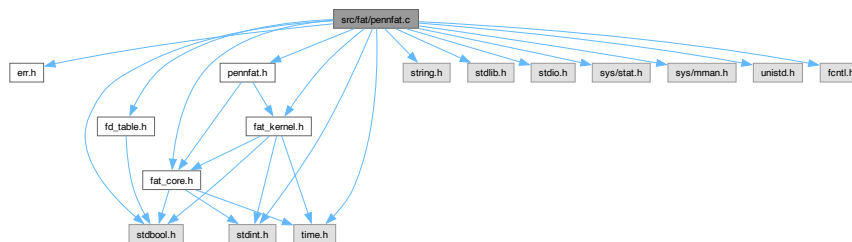
5.14 src/fat/pennfat.c File Reference

```

#include "err.h"
#include "fd_table.h"
#include "pennfat.h"
#include "fat_core.h"
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>
#include "fat_kernel.h"

```

Include dependency graph for pennfat.c:



Functions

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

5.14.1 Function Documentation

5.14.1.1 main()

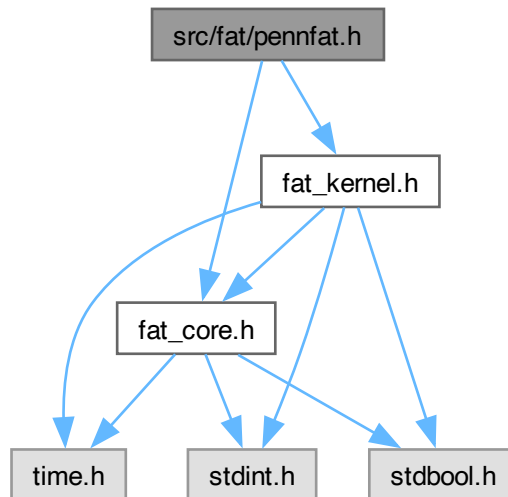
```

int main (
    int argc,
    char ** argv)

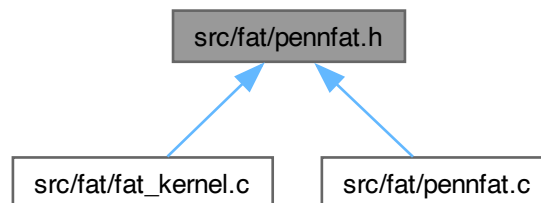
```

5.15 src/fat/pennfat.h File Reference

```
#include "fat_core.h"
#include "fat_kernel.h"
Include dependency graph for pennfat.h:
```



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



Macros

- `#define MAX_LINE_LENGTH 4096`

5.15.1 Macro Definition Documentation

5.15.1.1 MAX_LINE_LENGTH

```
#define MAX_LINE_LENGTH 4096
```


Releases a process's hold on a file descriptor. This involves finding the corresponding global file table entry, decrementing its reference count, and potentially closing the underlying file/resource if the count reaches zero. It also updates the process's local FD table to mark the descriptor as closed.

- `pcb_t * k_get_proc (pid_t pid)`
Get a process by its PID.
- `int k_write (int fd, const char *buf, int n)`
Write a message to the standard output.
- `int k_read (char *store_result, int len, int fd)`
Read from standard input.
- `pcb_t * k_get_process_by_pid (pid_t pid)`
Get the current process running in the kernel.

5.17.1 Function Documentation

5.17.1.1 k_get_current_process()

```
pcb_t * k_get_current_process ()
```

Get the current process running in the kernel.

Returns

Reference to the current PCB.

5.17.1.2 k_get_current_process_pid()

```
pid_t k_get_current_process_pid ()
```

Get the pid of the current process.

Returns

The PID of the current process.

5.17.1.3 k_get_lowest_pid()

```
int k_get_lowest_pid ()
```

5.17.1.4 k_get_proc()

```
pcb_t * k_get_proc (
    pid_t pid)
```

Get a process by its PID.

Parameters

<i>pid</i>	The PID of the process to retrieve.
------------	-------------------------------------

Returns

A pointer to the process control block (PCB) of the process, or NULL if not found.

A pointer to the process control block (PCB) of the process, or NULL if not found.

5.17.1.5 k_get_process_by_pid()

```
pcb_t * k_get_process_by_pid (  
    pid_t pid)
```

Get the current process running in the kernel.

Returns

Reference to the current PCB.

5.17.1.6 k_proc_cleanup()

```
void k_proc_cleanup (  
    pcb_t * proc)
```

Clean up a terminated/finished thread's resources. This may include freeing the PCB, handling children, etc.

5.17.1.7 k_proc_create()

```
pcb_t * k_proc_create (  
    pcb_t * parent)
```

Create a new child process, inheriting applicable properties from the parent.

Returns

Reference to the child PCB.

5.17.1.8 k_read()

```
int k_read (  
    char * store_result,  
    int len,  
    int fd)
```

Read form standard input.

Parameters

<i>store_result</i>	The buffer to store the read result.
<i>len</i>	The length of the buffer.
<i>fd</i>	The file descriptor to read from (not used in this implementation).

Returns

The number of bytes read.

5.17.1.9 k_release_fd()

```
int k_release_fd (
    pcb_t * proc,
    int local_fd)
```

Releases a process's hold on a file descriptor. This involves finding the corresponding global file table entry, decrementing its reference count, and potentially closing the underlying file/resource if the count reaches zero. It also updates the process's local FD table to mark the descriptor as closed.

Parameters

<i>proc</i>	The process control block (PCB) of the process closing the FD.
<i>local_fd</i>	The file descriptor number <i>local</i> to the given process.

Returns

0 on success, -1 on error (e.g., invalid *local_fd*, FD not open by this process).

0 on success, -1 on error (e.g., invalid *local_fd*, FD not open by this process).

5.17.1.10 k_write()

```
int k_write (
    int fd,
    const char * buf,
    int n)
```

Write a message to the standard output.

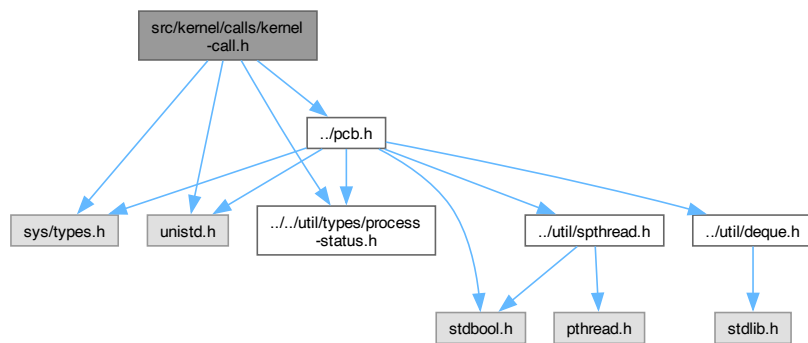
Parameters

<i>msg</i>	The message to write.
<i>fd</i>	The file descriptor to write to (not used in this implementation).

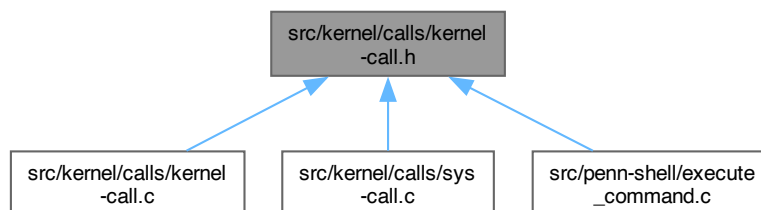
5.18 src/kernel/calls/kernel-call.h File Reference

```
#include <sys/types.h>
#include <unistd.h>
#include "../util/types/process-status.h"
#include "../pcb.h"
```

Include dependency graph for kernel-call.h:



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



Functions

- `pid_t k_get_current_process ()`
Get the pid of the current process.
- `pcb_t * k_get_current_process ()`
Get the current process running in the kernel.
- `pcb_t * k_get_process_by_pid (pid_t pid)`
Get the current process running in the kernel.
- `pcb_t * k_proc_create (pcb_t *parent)`
Create a new child process, inheriting applicable properties from the parent.
- `void k_proc_cleanup (pcb_t *proc)`
Clean up a terminated/finished thread's resources. This may include freeing the PCB, handling children, etc.
- `int k_release_fd (pcb_t *proc, int local_fd)`

Releases a process's hold on a file descriptor. This involves finding the corresponding global file table entry, decrementing its reference count, and potentially closing the underlying file/resource if the count reaches zero. It also updates the process's local FD table to mark the descriptor as closed.

- `pcb_t * k_get_proc (pid_t pid)`
Get a process by its PID.
- `int k_read (char *store_result, int len, int fd)`
Read from standard input.
- `int k_write (int fd, const char *buf, int n)`
Write a message to the standard output.

5.18.1 Function Documentation

5.18.1.1 `k_get_current_process()`

```
pcb_t * k_get_current_process ()
```

Get the current process running in the kernel.

Returns

Reference to the current PCB.

5.18.1.2 `k_get_current_process_pid()`

```
pid_t k_get_current_process_pid ()
```

Get the pid of the current process.

Returns

The PID of the current process.

5.18.1.3 `k_get_proc()`

```
pcb_t * k_get_proc (
    pid_t pid)
```

Get a process by its PID.

Parameters

<code>pid</code>	The PID of the process to retrieve.
------------------	-------------------------------------

Returns

A pointer to the process control block (PCB) of the process, or NULL if not found.

A pointer to the process control block (PCB) of the process, or NULL if not found.

5.18.1.4 k_get_process_by_pid()

```
pcb_t * k_get_process_by_pid (  
    pid_t pid)
```

Get the current process running in the kernel.

Returns

Reference to the current PCB.

5.18.1.5 k_proc_cleanup()

```
void k_proc_cleanup (  
    pcb_t * proc)
```

Clean up a terminated/finished thread's resources. This may include freeing the PCB, handling children, etc.

5.18.1.6 k_proc_create()

```
pcb_t * k_proc_create (  
    pcb_t * parent)
```

Create a new child process, inheriting applicable properties from the parent.

Returns

Reference to the child PCB.

5.18.1.7 k_read()

```
int k_read (  
    char * store_result,  
    int len,  
    int fd)
```

Read form standard input.

Parameters

<i>store_result</i>	The buffer to store the read result.
<i>len</i>	The length of the buffer.
<i>fd</i>	The file descriptor to read from (not used in this implementation).

Returns

The number of bytes read.

5.18.1.8 k_release_fd()

```
int k_release_fd (  
    pcb_t * proc,  
    int local_fd)
```

Releases a process's hold on a file descriptor. This involves finding the corresponding global file table entry, decrementing its reference count, and potentially closing the underlying file/resource if the count reaches zero. It also updates the process's local FD table to mark the descriptor as closed.

Parameters

<i>proc</i>	The process control block (PCB) of the process closing the FD.
<i>local</i> ↔ <i>_fd</i>	The file descriptor number <i>local</i> to the given process.

Returns

0 on success, -1 on error (e.g., invalid *local_fd*, FD not open by this process).

0 on success, -1 on error (e.g., invalid *local_fd*, FD not open by this process).

5.18.1.9 k_write()

```
int k_write (
    int fd,
    const char * buf,
    int n)
```

Write a message to the standard output.

Parameters

<i>msg</i>	The message to write.
<i>fd</i>	The file descriptor to write to (not used in this implementation).

5.19 kernel-call.h

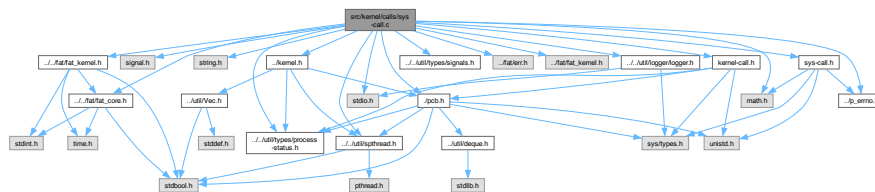
[Go to the documentation of this file.](#)

```
00001 #include <sys/types.h>
00002 #include <unistd.h>
00003 #include "../util/types/process-status.h"
00004 #include "../pcb.h"
00005
00010 pid_t k_get_current_process_pid();
00011
00017 pcb_t *k_get_current_process();
00018
00024
00025 pcb_t *k_get_process_by_pid(pid_t pid);
00032 pcb_t *k_proc_create(pcb_t *parent);
00033
00038 void k_proc_cleanup(pcb_t *proc);
00039
00054 int k_release_fd(pcb_t *proc, int local_fd);
00055 int k_release_fd(pcb_t *proc, int local_fd);
00056
00065 pcb_t *k_get_proc(pid_t pid);
00066
00075 int k_read(char *store_result, int len, int fd);
00076
00083
00084 int k_write(int fd, const char *buf, int n);
```

5.20 src/kernel/calls/sys-call.c File Reference

```
#include <math.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include "../fat/fat_core.h"
#include "../fat/fat_kernel.h"
#include "../util/logger/logger.h"
#include "../util/spthread.h"
#include "../util/types/process-status.h"
#include "../util/types/signals.h"
#include "../fat/err.h"
#include "../fat/fat_kernel.h"
#include "../kernel.h"
#include "../p_errno.h"
#include "../pcb.h"
#include "kernel-call.h"
#include "sys-call.h"
```

Include dependency graph for sys-call.c:



Functions

- `int map_fat_error_to_p_errno (int fat_errno)`
- `int s_open (const char *fname, int mode)`
- `int s_read (int fd, char *buf, int n)`
- `int s_write (int fd, const char *str, int n)`
- `int s_close (int fd)`
- `int s_lseek (int fd, int offset, int whence)`
- `int s_unlink (const char *fname)`
- `void s_ls (const char *filename)`
- `pid_t s_spawn (void *(*func)(void *), char *argv[], int fd0, int fd1, int foreground)`
- `pid_t s_waitpid_helper (pid_t curr_proc_pid, pid_t pid, int *wstatus)`
- `pid_t s_waitpid (pid_t pid, int *wstatus, int nohang)`
- `void s_sleep (unsigned int ticks)`
- `int s_kill (pid_t pid, int signal)`
- `int s_set_terminal_owner (pid_t pid)`
- `void s_nice_pid (int prio, int pid)`
- `void s_exit (void)`
- `int s_nice (pid_t pid, int priority)`
- `char * s_itos (int n)`
- `void s_ps ()`
- `void s_register_end ()`
- `int s_chmod (const char *fname, const char *mode_str)`
- `int s_get_permission (const char *fname)`
- `pid_t s_getpid (void)`

5.20.1 Function Documentation

5.20.1.1 `map_fat_error_to_p_errno()`

```
int map_fat_error_to_p_errno (  
    int fat_errno)
```

5.20.1.2 `s_chmod()`

```
int s_chmod (  
    const char * fname,  
    const char * mode_str)
```

changes permissions of a file

Parameters

<i>fname</i>	name of the file
<i>mode_str</i>	mode string (e.g., "+r", "-wx")

Returns

0 on success, -1 on failure

5.20.1.3 `s_close()`

```
int s_close (  
    int fd)
```

closes a file descriptor

Parameters

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

Returns

0 on success, negative value on failure

5.20.1.4 `s_exit()`

```
void s_exit (  
    void )
```

terminates calling process

5.20.1.5 `s_get_permission()`

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

gets the permission bits for a file.

Parameters

<i>fname</i>	name of the file.
--------------	-------------------

Returns

permission value (int) on success, -1 on failure (sets P_ERRNO).

5.20.1.6 s_getpid()

```
pid_t s_getpid (  
    void )
```

gets the process ID of the calling process

Returns

process ID of calling process

5.20.1.7 s_itos()

```
char * s_itos (  
    int n)
```

5.20.1.8 s_kill()

```
int s_kill (  
    pid_t pid,  
    int signal)
```

sends a signal to a process

Parameters

<i>pid</i>	process ID of target process
<i>signal</i>	signal number (P_SIGTERM, P_SIGSTOP, P_SIGCONT)

Returns

0 on success, -1 on error

5.20.1.9 s_ls()

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

lists file(s) in filesystem

Parameters

<i>filename</i>	specific file to list, or NULL for all files
-----------------	--

5.20.1.10 s_lseek()

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

repositions file offset of open file descriptor

Parameters

<i>fd</i>	file descriptor
<i>offset</i>	offset value
<i>whence</i>	reference position (SEEK_SET, SEEK_CUR, SEEK_END)

Returns

new offset on success, negative value on error

5.20.1.11 s_nice()

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

sets scheduling priority of a process

Parameters

<i>pid</i>	process ID of target
<i>priority</i>	new priority value (0, 1, or 2)

Returns

0 on success, -1 on failure

5.20.1.12 s_nice_pid()

```
void s_nice_pid (  
    int prio,  
    int pid)
```

Sets the priority of a process

Parameters

<i>prio</i>	new priority value (0, 1, or 2)
<i>pid</i>	process ID of target process

5.20.1.13 s_open()

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

opens a file with specified mode

Parameters

<i>fname</i>	name of file to open
<i>mode</i>	access mode (F_READ, F_WRITE, or F_APPEND)

Returns

file descriptor on success, negative value on error

5.20.1.14 s_ps()

```
void s_ps (
    void )
```

displays information about all processes

5.20.1.15 s_read()

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

reads data from a file descriptor

Parameters

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

Returns

number of bytes read, 0 on EOF, negative value on error

5.20.1.16 s_register_end()

```
void s_register_end (
    void )
```

registers calling process as finished

5.20.1.17 s_set_terminal_owner()

```
int s_set_terminal_owner (
    pid_t pid)
```

sets the terminal owner

Parameters

<i>pid</i>	process ID of target
------------	----------------------

5.20.1.18 s_sleep()

```
void s_sleep (
    unsigned int ticks)
```

suspends calling process for specified clock ticks

Parameters

<i>ticks</i>	duration of sleep in clock ticks (must be > 0)
--------------	--

5.20.1.19 s_spawn()

```
pid_t s_spawn (
    void *(* func ) (void *),
    char * argv[],
    int fd0,
    int fd1,
    int foreground)
```

creates a child process executing specified function

Parameters

<i>func</i>	function to execute
<i>argv</i>	null-terminated array of args (command name as argv[0])
<i>fd0</i>	input file descriptor
<i>fd1</i>	output file descriptor
<i>foreground</i>	true for foreground execution, false for background

Returns

process ID of child, -1 on error

5.20.1.20 s_unlink()

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

removes a file from filesystem if not in use

Parameters

<i>fname</i>	name of file to remove
--------------	------------------------

Returns

0 on success, -1 on error

5.20.1.21 s_waitpid()

```
pid_t s_waitpid (  
    pid_t pid,  
    int * wstatus,  
    int nohang)
```

waits for a child process to change state

Parameters

<i>pid</i>	process ID of child to wait for (-1 for any child)
<i>wstatus</i>	pointer to store exit status information
<i>nohang</i>	if true, return immediately if no child has exited

Returns

process ID of state-changed child, 0 if nohang and no change, -1 on error

5.20.1.22 s_waitpid_helper()

```
pid_t s_waitpid_helper (  
    pid_t curr_proc_pid,  
    pid_t pid,  
    int * wstatus)
```

5.20.1.23 s_write()

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

writes data to a file descriptor

Parameters

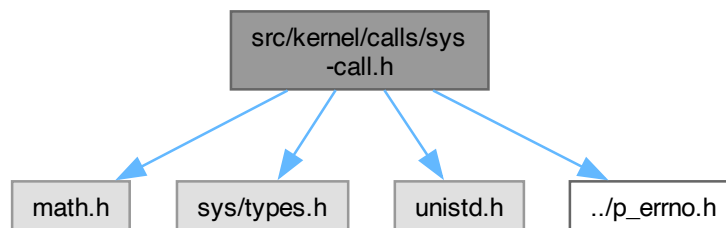
<i>fd</i>	file descriptor to write to
<i>str</i>	buffer containing data to write
<i>n</i>	number of bytes to write

Returns

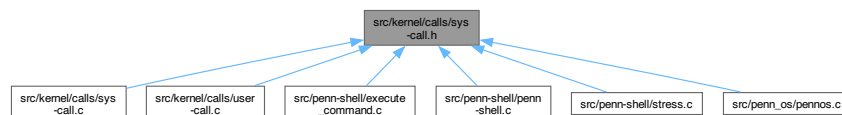
number of bytes written, negative value on error

5.21 src/kernel/calls/sys-call.h File Reference

```
#include <math.h>
#include <sys/types.h>
#include <unistd.h>
#include "../p_errno.h"
Include dependency graph for sys-call.h:
```



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

**Macros**

- `#define P_WAIT_STATUS_MACROS_H`
- `#define P_WAIT_FLAG_STOPPED 0x100`
- `#define P_WAIT_FLAG_SIGNALED 0x200`
- `#define P_WAIT_SIG_MASK 0xFF`
- `#define P_WIFEXITED(status)`

Returns true if the child terminated normally (via `s_exit` or `return`).

- `#define P_WIFSIGNALED(status)`
Returns true if the child process was terminated by a signal.
- `#define P_WTERMSIG(status)`
Returns the number of the signal that caused the child to terminate. (Only valid if `P_WIFSIGNALED(status)` is true).
- `#define P_WIFSTOPPED(status)`
Returns true if the child process was stopped by delivery of a signal.
- `#define P_WSTOPSIG(status)`
Returns the number of the signal which caused the child to stop. (Only valid if `P_WIFSTOPPED(status)` is true).

Functions

- `int s_open (const char *fname, int mode)`
- `int s_close (int fd)`
- `int s_read (int fd, char *buf, int n)`
- `int s_write (int fd, const char *str, int n)`
- `int s_lseek (int fd, int offset, int whence)`
- `int s_unlink (const char *fname)`
- `void s_ls (const char *filename)`
- `pid_t s_spawn (void *(*func)(void *), char *argv[], int fd0, int fd1, int foreground)`
- `pid_t s_waitpid (pid_t pid, int *wstatus, int nohang)`
- `int s_kill (pid_t pid, int signal)`
- `void s_exit (void)`
- `int s_nice (pid_t pid, int priority)`
- `void s_sleep (unsigned int ticks)`
- `void s_ps (void)`
- `void s_register_end (void)`
- `void s_nice_pid (int prio, int pid)`
- `pid_t s_getpid (void)`
- `int s_chmod (const char *fname, const char *mode_str)`
- `int s_get_permission (const char *fname)`
- `int s_set_terminal_owner (pid_t pid)`

Variables

- `int P_ERRNO`
Error number for the last system call.

5.21.1 Macro Definition Documentation

5.21.1.1 P_WAIT_FLAG_SIGNALED

```
#define P_WAIT_FLAG_SIGNALED 0x200
```

5.21.1.2 P_WAIT_FLAG_STOPPED

```
#define P_WAIT_FLAG_STOPPED 0x100
```

5.21.1.3 P_WAIT_SIG_MASK

```
#define P_WAIT_SIG_MASK 0xFF
```

5.21.1.4 P_WAIT_STATUS_MACROS_H

```
#define P_WAIT_STATUS_MACROS_H
```

5.21.1.5 P_WIFEXITED

```
#define P_WIFEXITED(  
    status)
```

Value:

```
((status) & (P_WAIT_FLAG_STOPPED | P_WAIT_FLAG_SIGNALED)) == 0)
```

Returns true if the child terminated normally (via `s_exit` or `return`).

5.21.1.6 P_WIFSIGNALED

```
#define P_WIFSIGNALED(  
    status)
```

Value:

```
((status) & P_WAIT_FLAG_SIGNALED) != 0)
```

Returns true if the child process was terminated by a signal.

5.21.1.7 P_WIFSTOPPED

```
#define P_WIFSTOPPED(  
    status)
```

Value:

```
((status) & P_WAIT_FLAG_STOPPED) != 0)
```

Returns true if the child process was stopped by delivery of a signal.

5.21.1.8 P_WSTOPSIG

```
#define P_WSTOPSIG(  
    status)
```

Value:

```
((status) & P_WAIT_SIG_MASK)
```

Returns the number of the signal which caused the child to stop. (Only valid if `P_WIFSTOPPED(status)` is true).

5.21.1.9 P_WTERMSIG

```
#define P_WTERMSIG(  
    status)
```

Value:

```
((status) & P_WAIT_SIG_MASK)
```

Returns the number of the signal that caused the child to terminate. (Only valid if `P_WIFSIGNALED(status)` is true).

5.21.2 Function Documentation

5.21.2.1 s_chmod()

```
int s_chmod (  
    const char * fname,  
    const char * mode_str)
```

changes permissions of a file

Parameters

<i>fname</i>	name of the file
<i>mode_str</i>	mode string (e.g., "+r", "-wx")

Returns

0 on success, -1 on failure

5.21.2.2 s_close()

```
int s_close (  
    int fd)
```

closes a file descriptor

Parameters

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

Returns

0 on success, negative value on failure

5.21.2.3 s_exit()

```
void s_exit (  
    void )
```

terminates calling process

5.21.2.4 s_get_permission()

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

gets the permission bits for a file.

Parameters

<i>fname</i>	name of the file.
--------------	-------------------

Returns

permission value (int) on success, -1 on failure (sets P_ERRNO).

5.21.2.5 s_getpid()

```
pid_t s_getpid (  
    void )
```

gets the process ID of the calling process

Returns

process ID of calling process

5.21.2.6 s_kill()

```
int s_kill (  
    pid_t pid,  
    int signal)
```

sends a signal to a process

Parameters

<i>pid</i>	process ID of target process
<i>signal</i>	signal number (P_SIGTERM, P_SIGSTOP, P_SIGCONT)

Returns

0 on success, -1 on error

5.21.2.7 s_ls()

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

lists file(s) in filesystem

Parameters

<i>filename</i>	specific file to list, or NULL for all files
-----------------	--

5.21.2.8 s_lseek()

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

repositions file offset of open file descriptor

Parameters

<i>fd</i>	file descriptor
<i>offset</i>	offset value
<i>whence</i>	reference position (SEEK_SET, SEEK_CUR, SEEK_END)

Returns

new offset on success, negative value on error

5.21.2.9 s_nice()

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

sets scheduling priority of a process

Parameters

<i>pid</i>	process ID of target
<i>priority</i>	new priority value (0, 1, or 2)

Returns

0 on success, -1 on failure

5.21.2.10 s_nice_pid()

```
void s_nice_pid (  
    int prio,  
    int pid)
```

Sets the priority of a process

Parameters

<i>prio</i>	new priority value (0, 1, or 2)
<i>pid</i>	process ID of target process

5.21.2.11 s_open()

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

opens a file with specified mode

Parameters

<i>fname</i>	name of file to open
<i>mode</i>	access mode (F_READ, F_WRITE, or F_APPEND)

Returns

file descriptor on success, negative value on error

5.21.2.12 s_ps()

```
void s_ps (
    void )
```

displays information about all processes

5.21.2.13 s_read()

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

reads data from a file descriptor

Parameters

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

Returns

number of bytes read, 0 on EOF, negative value on error

5.21.2.14 s_register_end()

```
void s_register_end (
    void )
```

registers calling process as finished

5.21.2.15 s_set_terminal_owner()

```
int s_set_terminal_owner (
    pid_t pid)
```

sets the terminal owner

Parameters

<i>pid</i>	process ID of target
------------	----------------------

5.21.2.16 s_sleep()

```
void s_sleep (
    unsigned int ticks)
```

suspends calling process for specified clock ticks

Parameters

<i>ticks</i>	duration of sleep in clock ticks (must be > 0)
--------------	--

5.21.2.17 s_spawn()

```
pid_t s_spawn (
    void *(* func ) (void *),
    char * argv[],
    int fd0,
    int fd1,
    int foreground)
```

creates a child process executing specified function

Parameters

<i>func</i>	function to execute
<i>argv</i>	null-terminated array of args (command name as argv[0])
<i>fd0</i>	input file descriptor
<i>fd1</i>	output file descriptor
<i>foreground</i>	true for foreground execution, false for background

Returns

process ID of child, -1 on error

5.21.2.18 s_unlink()

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

removes a file from filesystem if not in use

Parameters

<i>fname</i>	name of file to remove
--------------	------------------------

Returns

0 on success, -1 on error

5.21.2.19 s_waitpid()

```
pid_t s_waitpid (  
    pid_t pid,  
    int * wstatus,  
    int nohang)
```

waits for a child process to change state

Parameters

<i>pid</i>	process ID of child to wait for (-1 for any child)
<i>wstatus</i>	pointer to store exit status information
<i>nohang</i>	if true, return immediately if no child has exited

Returns

process ID of state-changed child, 0 if nohang and no change, -1 on error

5.21.2.20 s_write()

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

writes data to a file descriptor

Parameters

<i>fd</i>	file descriptor to write to
<i>str</i>	buffer containing data to write
<i>n</i>	number of bytes to write

Returns

number of bytes written, negative value on error

5.21.3 Variable Documentation

5.21.3.1 P_ERRNO

```
int P_ERRNO [extern]
```

Error number for the last system call.

5.22 sys-call.h

[Go to the documentation of this file.](#)

```
00001 #ifndef SYS_CALL_H
00002 #define SYS_CALL_H
00003
00004 #include <math.h>
00005 #include <sys/types.h>
00006 #include <unistd.h>
00007 #include "../p_errno.h"
00008
00009 // File system call
00013 extern int P_ERRNO;
00014
00021 int s_open(const char *fname, int mode);
00022
00028 int s_close(int fd);
00029
00037 int s_read(int fd, char *buf, int n);
00038
00046 int s_write(int fd, const char *str, int n);
00047
00055 int s_lseek(int fd, int offset, int whence);
00056
00062 int s_unlink(const char *fname);
00063
00068 void s_ls(const char *filename);
00069
00079 pid_t s_spawn(void (*func)(void *),
00080               char *argv[],
00081               int fd0,
00082               int fd1,
00083               int foreground);
00084
00093 pid_t s_waitpid(pid_t pid, int *wstatus, int nohang);
00094
00101 int s_kill(pid_t pid, int signal);
00102
00106 void s_exit(void);
00107
00114 int s_nice(pid_t pid, int priority);
00115
00120 void s_sleep(unsigned int ticks);
00121
00125 void s_ps(void);
00126
00130 void s_register_end(void);
00131
00138 void s_nice_pid(int prio, int pid);
00139
00144 pid_t s_getpid(void);
00145
00152 int s_chmod(const char *fname, const char *mode_str); // Pass mode string
00153
00159 int s_get_permission(const char *fname);
00160
00165 int s_set_terminal_owner(pid_t pid);
00166
00167 #ifndef P_WAIT_STATUS_MACROS_H // Include guard for these macros
00168 #define P_WAIT_STATUS_MACROS_H
00169
00170 #define P_WAIT_FLAG_STOPPED 0x100 // Bit 8 indicates stopped by a signal
00171 #define P_WAIT_FLAG_SIGNALED 0x200 // Bit 9 indicates terminated by a signal
00172 #define P_WAIT_SIG_MASK 0xFF // Lower 8 bits for signal number
00173
00177 #define P_WIFEXITED(status) (((status) & (P_WAIT_FLAG_STOPPED | P_WAIT_FLAG_SIGNALED)) == 0)
00178
00182 #define P_WIFSIGNALED(status) (((status) & P_WAIT_FLAG_SIGNALED) != 0)
```

```

00183
00188 #define P_WTERMSIG(status) ((status) & P_WAIT_SIG_MASK)
00189
00193 #define P_WIFSTOPPED(status) (((status) & P_WAIT_FLAG_STOPPED) != 0)
00194
00199 #define P_WSTOPSIG(status) ((status) & P_WAIT_SIG_MASK)
00200
00201 #endif // P_WAIT_STATUS_MACROS_H
00202
00203 #endif /* SYS_CALL_H */

```

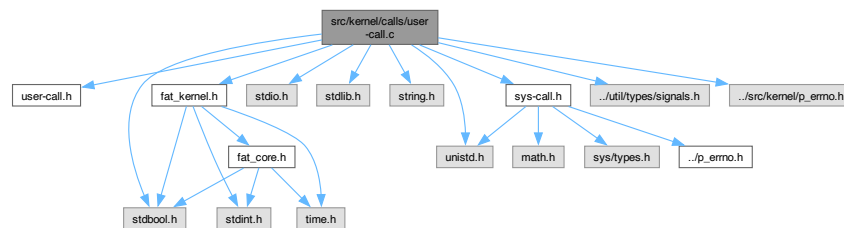
5.23 src/kernel/calls/user-call.c File Reference

```

#include "user-call.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "../util/types/signals.h"
#include "fat_kernel.h"
#include "sys-call.h"
#include "../src/kernel/p_errno.h"

```

Include dependency graph for user-call.c:



Functions

- void * [u_cat](#) (void *arg)
The usual cat program.
- void * [u_ps](#) (void *arg)
List all processes on PennOS, displaying PID, PPID, priority, status, and command name.
- void * [u_kill](#) (void *arg)
Sends a specified signal to a list of processes. If a signal name is not specified, default to "term". Valid signals are -term, -stop, and -cont.
- void * [u_zombify](#) (void *arg)
Used to test zombifying functionality of your kernel.
- void * [zombie_child](#) (void *arg)
Helper for zombify.
- void * [u_nice](#) (void *arg)
Spawn a new process for command and set its priority to priority.
- void * [u_nice_pid](#) (void *arg)
Adjust the priority level of an existing process.
- void * [u_orphanify](#) (void *arg)

- Used to test orphanifying functionality of your kernel.*

 - void * `u_orphan_child` (void *arg)

Helper for orphanify.
- int `stoi` (char *str)

Converts a string to an integer.
- void * `u_sleep` (void *arg)

Sleep for n seconds.
- void * `u_busy` (void *arg)

Busy wait indefinitely. It can only be interrupted via signals.
- void * `u_echo` (void *arg)

Echo back an input string.
- void * `u_ls` (void *arg)

Lists all files in the working directory. For extra credit, it should support relative and absolute file paths.
- void * `u_touch` (void *arg)

For each file, create an empty file if it doesn't exist, else update its timestamp.
- void * `u_mv` (void *arg)

Rename a file. If the `dst_file` file already exists, overwrite it.
- void * `u_cp` (void *arg)
- void * `u_rm` (void *arg)

Remove a list of files. Treat each file in the list as a separate transaction. (i.e. if removing file1 fails, still attempt to remove file2, file3, etc.)
- void * `u_bg` (void *arg)

Resumes the most recently stopped job in the background, or the job specified by `job_id`.
- void * `u_fg` (void *arg)

Brings the most recently stopped or background job to the foreground, or the job specified by `job_id`.
- void * `u_man` (void *arg)

Lists all available commands.
- void * `u_jobs` (void *arg)

Lists all jobs.
- void * `u_logout` (void *arg)

Exits the shell and shutdown PennOS.
- void * `u_chmod` (void *arg)

Change permissions of a file. There's no need to error if a permission being added already exists, or if a permission being removed is already not granted.

5.23.1 Function Documentation

5.23.1.1 stoi()

```
int stoi (
    char * str)
```

Converts a string to an integer.

5.23.1.2 u_bg()

```
void * u_bg (
    void * arg)
```

Resumes the most recently stopped job in the background, or the job specified by `job_id`.

Example Usage: `bg` Example Usage: `bg 2` (`job_id` is 2)

5.23.1.3 `u_busy()`

```
void * u_busy (
    void * arg)
```

Busy wait indefinitely. It can only be interrupted via signals.

Example Usage: `busy`

5.23.1.4 `u_cat()`

```
void * u_cat (
    void * arg)
```

The usual `cat` program.

If `files` `arg` is provided, concatenate these files and print to stdout. If `files` `arg` is *not* provided, read from stdin and print back to stdout.

Example Usage: `cat f1 f2` (concatenates `f1` and `f2` and print to stdout) Example Usage: `cat f1 f2 < f3` (concatenates `f1` and `f2` and prints to stdout, ignores `f3`) Example Usage: `cat < f3` (concatenates `f3`, prints to stdout)

5.23.1.5 `u_chmod()`

```
void * u_chmod (
    void * arg)
```

Change permissions of a file. There's no need to error if a permission being added already exists, or if a permission being removed is already not granted.

Print appropriate error message if:

- `file` is not a file that exists
- `perms` is invalid

Example Usage: `chmod +x file` (adds executable permission to file) Example Usage: `chmod +rw file` (adds read + write permissions to file) Example Usage: `chmod -wx file` (removes write + executable permissions from file)

5.23.1.6 `u_cp()`

```
void * u_cp (
    void * arg)
```

Copy a file. If the `dst_file` file already exists, overwrite it.

Print appropriate error message if:

- `src_file` is not a file that exists
- `src_file` does not have read permissions
- `dst_file` file already exists but does not have write permissions

Example Usage: `cp src_file dst_file`

5.23.1.7 u_echo()

```
void * u_echo (
    void * arg)
```

Echo back an input string.

Example Usage: echo Hello World

5.23.1.8 u_fg()

```
void * u_fg (
    void * arg)
```

Brings the most recently stopped or background job to the foreground, or the job specified by job_id.

Example Usage: fg Example Usage: fg 2 (job_id is 2)

5.23.1.9 u_jobs()

```
void * u_jobs (
    void * arg)
```

Lists all jobs.

Example Usage: jobs

5.23.1.10 u_kill()

```
void * u_kill (
    void * arg)
```

Sends a specified signal to a list of processes. If a signal name is not specified, default to "term". Valid signals are -term, -stop, and -cont.

Example Usage: kill 1 2 3 (sends term to processes 1, 2, and 3) Example Usage: kill -term 1 2 (sends term to processes 1 and 2) Example Usage: kill -stop 1 2 (sends stop to processes 1 and 2) Example Usage: kill -cont 1 (sends cont to process 1)

5.23.1.11 u_logout()

```
void * u_logout (
    void * arg)
```

Exits the shell and shutdowns PennOS.

Example Usage: logout

5.23.1.12 `u_ls()`

```
void * u_ls (
    void * arg)
```

Lists all files in the working directory. For extra credit, it should support relative and absolute file paths.

Example Usage: `ls` (regular credit) Example Usage: `ls ../../foo/./bar/sample` (only for EC)

5.23.1.13 `u_man()`

```
void * u_man (
    void * arg)
```

Lists all available commands.

Example Usage: `man`

5.23.1.14 `u_mv()`

```
void * u_mv (
    void * arg)
```

Rename a file. If the `dst_file` file already exists, overwrite it.

Print appropriate error message if:

- `src_file` is not a file that exists
- `src_file` does not have read permissions
- `dst_file` file already exists but does not have write permissions

Example Usage: `mv src_file dst_file`

5.23.1.15 `u_nice()`

```
void * u_nice (
    void * arg)
```

Spawn a new process for `command` and set its priority to `priority`.

1. Adjust the priority level of an existing process.

Example Usage: `nice 2 cat f1 f2 f3` (spawns `cat` with priority 2)

5.23.1.16 `u_nice_pid()`

```
void * u_nice_pid (
    void * arg)
```

Adjust the priority level of an existing process.

Parameters

<i>prio</i>	The new priority level (0, 1, or 2).
<i>pid</i>	The PID of the target process.

5.23.1.17 u_orphan_child()

```
void * u_orphan_child (  
    void * arg)
```

Helper for orphanify.

5.23.1.18 u_orphanify()

```
void * u_orphanify (  
    void * arg)
```

Used to test orphanifying functionality of your kernel.

Example Usage: orphanify

5.23.1.19 u_ps()

```
void * u_ps (  
    void * arg)
```

List all processes on PennOS, displaying PID, PPID, priority, status, and command name.

Example Usage: ps

5.23.1.20 u_rm()

```
void * u_rm (  
    void * arg)
```

Remove a list of files. Treat each file in the list as a separate transaction. (i.e. if removing file1 fails, still attempt to remove file2, file3, etc.)

Print appropriate error message if:

- `file` is not a file that exists

Example Usage: rm f1 f2 f3 f4 f5

5.23.1.21 `u_sleep()`

```
void * u_sleep (
    void * arg)
```

Sleep for *n* seconds.

Note that you'll have to convert the number of seconds to the correct number of ticks.

Example Usage: `sleep 10`

5.23.1.22 `u_touch()`

```
void * u_touch (
    void * arg)
```

For each file, create an empty file if it doesn't exist, else update its timestamp.

Example Usage: `touch f1 f2 f3 f4 f5`

5.23.1.23 `u_zombify()`

```
void * u_zombify (
    void * arg)
```

Used to test zombifying functionality of your kernel.

Example Usage: `zombify`

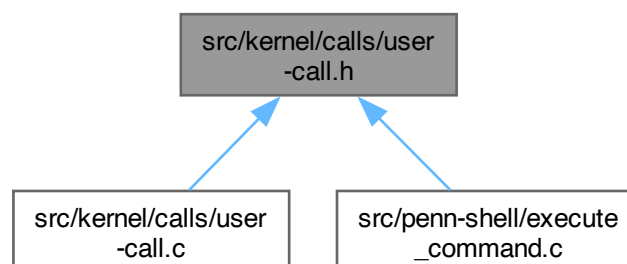
5.23.1.24 `zombie_child()`

```
void * zombie_child (
    void * arg)
```

Helper for `zombify`.

5.24 `src/kernel/calls/user-call.h` File Reference

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



Macros

- #define `MAX_MESSAGE_SIZE` 4096

Functions

- void * `u_cat` (void *arg)
The usual `cat` program.
- void * `u_sleep` (void *arg)
Sleep for `n` seconds.
- void * `u_busy` (void *arg)
Busy wait indefinitely. It can only be interrupted via signals.
- void * `u_echo` (void *arg)
Echo back an input string.
- void * `u_ls` (void *arg)
Lists all files in the working directory. For extra credit, it should support relative and absolute file paths.
- void * `u_touch` (void *arg)
For each file, create an empty file if it doesn't exist, else update its timestamp.
- void * `u_mv` (void *arg)
Rename a file. If the `dst_file` file already exists, overwrite it.
- void * `u_cp` (void *arg)
- void * `u_rm` (void *arg)
Remove a list of files. Treat each file in the list as a separate transaction. (i.e. if removing `file1` fails, still attempt to remove `file2`, `file3`, etc.)
- void * `u_chmod` (void *arg)
Change permissions of a file. There's no need to error if a permission being added already exists, or if a permission being removed is already not granted.
- void * `u_ps` (void *arg)
List all processes on PennOS, displaying PID, PPID, priority, status, and command name.
- void * `u_kill` (void *arg)
Sends a specified signal to a list of processes. If a signal name is not specified, default to "term". Valid signals are -term, -stop, and -cont.
- void * `zombie_child` (void *arg)
Helper for zombify.
- void * `u_zombify` (void *arg)
Used to test zombifying functionality of your kernel.
- void * `u_orphan_child` (void *arg)
Helper for orphanify.
- void * `u_orphanify` (void *arg)
Used to test orphanifying functionality of your kernel.
- void * `u_nice` (void *arg)
Spawn a new process for `command` and set its priority to `priority`.
- void * `u_nice_pid` (void *arg)
Adjust the priority level of an existing process.
- void * `u_man` (void *arg)
Lists all available commands.
- void * `u_bg` (void *arg)
Resumes the most recently stopped job in the background, or the job specified by `job_id`.
- void * `u_fg` (void *arg)
Brings the most recently stopped or background job to the foreground, or the job specified by `job_id`.
- void * `u_jobs` (void *arg)

Lists all jobs.

- void * [u_logout](#) (void *arg)

Exits the shell and shutdown PennOS.

- int [stoi](#) (char *str)

Converts a string to an integer.

- void * [u_hang](#) (void *arg)

Test command that spawns 10 child processes and waits on them (blocking). Useful for testing the scheduler and wait functionality.

- void * [u_nohang](#) (void *arg)

Test command that spawns 10 child processes and waits on them (non-blocking). Useful for testing the scheduler with non-blocking waits.

- void * [u_recur](#) (void *arg)

Test command that recursively spawns 26 processes (Gen_A through Gen_Z). Useful for testing recursive spawning and deep process hierarchies.

- void * [u_crash](#) (void *arg)

Test command that writes a large pattern to a file and then crashes. Useful for testing file system durability and crash recovery. Requires the filesystem to be able to hold at least 5480 bytes in a file.

5.24.1 Macro Definition Documentation

5.24.1.1 MAX_MESSAGE_SIZE

```
#define MAX_MESSAGE_SIZE 4096
```

5.24.2 Function Documentation

5.24.2.1 stoi()

```
int stoi (
    char * str)
```

Converts a string to an integer.

5.24.2.2 u_bg()

```
void * u_bg (
    void * arg)
```

Resumes the most recently stopped job in the background, or the job specified by job_id.

Example Usage: bg Example Usage: bg 2 (job_id is 2)

5.24.2.3 u_busy()

```
void * u_busy (
    void * arg)
```

Busy wait indefinitely. It can only be interrupted via signals.

Example Usage: busy

5.24.2.4 u_cat()

```
void * u_cat (
    void * arg)
```

The usual `cat` program.

If `files` arg is provided, concatenate these files and print to stdout. If `files` arg is *not* provided, read from stdin and print back to stdout.

Example Usage: `cat f1 f2` (concatenates `f1` and `f2` and print to stdout) Example Usage: `cat f1 f2 < f3` (concatenates `f1` and `f2` and prints to stdout, ignores `f3`) Example Usage: `cat < f3` (concatenates `f3`, prints to stdout)

5.24.2.5 u_chmod()

```
void * u_chmod (
    void * arg)
```

Change permissions of a file. There's no need to error if a permission being added already exists, or if a permission being removed is already not granted.

Print appropriate error message if:

- `file` is not a file that exists
- `perms` is invalid

Example Usage: `chmod +x file` (adds executable permission to file) Example Usage: `chmod +rw file` (adds read + write permissions to file) Example Usage: `chmod -wx file` (removes write + executable permissions from file)

5.24.2.6 u_cp()

```
void * u_cp (
    void * arg)
```

Copy a file. If the `dst_file` file already exists, overwrite it.

Print appropriate error message if:

- `src_file` is not a file that exists
- `src_file` does not have read permissions
- `dst_file` file already exists but does not have write permissions

Example Usage: `cp src_file dst_file`

5.24.2.7 `u_crash()`

```
void * u_crash (  
    void * arg)
```

Test command that writes a large pattern to a file and then crashes. Useful for testing file system durability and crash recovery. Requires the filesystem to be able to hold at least 5480 bytes in a file.

Example Usage: `crash`

5.24.2.8 `u_echo()`

```
void * u_echo (  
    void * arg)
```

Echo back an input string.

Example Usage: `echo Hello World`

5.24.2.9 `u_fg()`

```
void * u_fg (  
    void * arg)
```

Brings the most recently stopped or background job to the foreground, or the job specified by `job_id`.

Example Usage: `fg` Example Usage: `fg 2` (`job_id` is 2)

5.24.2.10 `u_hang()`

```
void * u_hang (  
    void * arg)
```

Test command that spawns 10 child processes and waits on them (blocking). Useful for testing the scheduler and wait functionality.

Example Usage: `hang`

5.24.2.11 `u_jobs()`

```
void * u_jobs (  
    void * arg)
```

Lists all jobs.

Example Usage: `jobs`

5.24.2.12 u_kill()

```
void * u_kill (
    void * arg)
```

Sends a specified signal to a list of processes. If a signal name is not specified, default to "term". Valid signals are -term, -stop, and -cont.

Example Usage: kill 1 2 3 (sends term to processes 1, 2, and 3) Example Usage: kill -term 1 2 (sends term to processes 1 and 2) Example Usage: kill -stop 1 2 (sends stop to processes 1 and 2) Example Usage: kill -cont 1 (sends cont to process 1)

5.24.2.13 u_logout()

```
void * u_logout (
    void * arg)
```

Exits the shell and shutdowns PennOS.

Example Usage: logout

5.24.2.14 u_ls()

```
void * u_ls (
    void * arg)
```

Lists all files in the working directory. For extra credit, it should support relative and absolute file paths.

Example Usage: ls (regular credit) Example Usage: ls ../../foo/./bar/sample (only for EC)

5.24.2.15 u_man()

```
void * u_man (
    void * arg)
```

Lists all available commands.

Example Usage: man

5.24.2.16 u_mv()

```
void * u_mv (
    void * arg)
```

Rename a file. If the `dst_file` file already exists, overwrite it.

Print appropriate error message if:

- `src_file` is not a file that exists
- `src_file` does not have read permissions
- `dst_file` file already exists but does not have write permissions

Example Usage: mv `src_file` `dst_file`

5.24.2.17 `u_nice()`

```
void * u_nice (
    void * arg)
```

Spawn a new process for `command` and set its priority to `priority`.

1. Adjust the priority level of an existing process.

Example Usage: `nice 2 cat f1 f2 f3` (spawns cat with priority 2)

5.24.2.18 `u_nice_pid()`

```
void * u_nice_pid (
    void * arg)
```

Adjust the priority level of an existing process.

Example Usage: `nice_pid 0 123` (sets priority 0 to PID 123)

Parameters

<i>prio</i>	The new priority level (0, 1, or 2).
<i>pid</i>	The PID of the target process.

5.24.2.19 `u_nohang()`

```
void * u_nohang (
    void * arg)
```

Test command that spawns 10 child processes and waits on them (non-blocking). Useful for testing the scheduler with non-blocking waits.

Example Usage: `nohang`

5.24.2.20 `u_orphan_child()`

```
void * u_orphan_child (
    void * arg)
```

Helper for orphanify.

5.24.2.21 `u_orphanify()`

```
void * u_orphanify (
    void * arg)
```

Used to test orphanifying functionality of your kernel.

Example Usage: `orphanify`

5.24.2.22 u_ps()

```
void * u_ps (
    void * arg)
```

List all processes on PennOS, displaying PID, PPID, priority, status, and command name.

Example Usage: ps

5.24.2.23 u_recur()

```
void * u_recur (
    void * arg)
```

Test command that recursively spawns 26 processes (Gen_A through Gen_Z). Useful for testing recursive spawning and deep process hierarchies.

Example Usage: recur

5.24.2.24 u_rm()

```
void * u_rm (
    void * arg)
```

Remove a list of files. Treat each file in the list as a separate transaction. (i.e. if removing file1 fails, still attempt to remove file2, file3, etc.)

Print appropriate error message if:

- `file` is not a file that exists

Example Usage: rm f1 f2 f3 f4 f5

5.24.2.25 u_sleep()

```
void * u_sleep (
    void * arg)
```

Sleep for `n` seconds.

Note that you'll have to convert the number of seconds to the correct number of ticks.

Example Usage: sleep 10

5.24.2.26 u_touch()

```
void * u_touch (
    void * arg)
```

For each file, create an empty file if it doesn't exist, else update its timestamp.

Example Usage: touch f1 f2 f3 f4 f5

5.24.2.27 u_zombify()

```
void * u_zombify (
    void * arg)
```

Used to test zombifying functionality of your kernel.

Example Usage: zombify

5.24.2.28 zombie_child()

```
void * zombie_child (
    void * arg)
```

Helper for zombify.

5.25 user-call.h

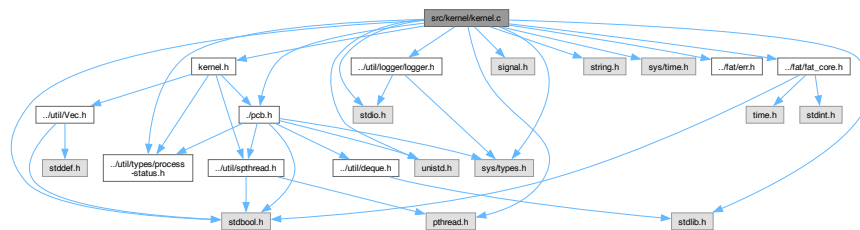
[Go to the documentation of this file.](#)

```
00001 #ifndef MAX_MESSAGE_SIZE
00002 #define MAX_MESSAGE_SIZE 4096
00003 #endif
00014 void* u_cat(void* arg);
00015
00024 void* u_sleep(void* arg);
00025
00032 void* u_busy(void* arg);
00033
00039 void* u_echo(void* arg);
00040
00048 void* u_ls(void* arg);
00049
00056 void* u_touch(void* arg);
00057
00068 void* u_mv(void* arg);
00069
00080 void* u_cp(void* arg);
00081
00092 void* u_rm(void* arg);
00093
00108 void* u_chmod(void* arg);
00109
00116 void* u_ps(void* arg);
00117
00128 void* u_kill(void* arg);
00129
00133 void* zombie_child(void* arg);
00134
00140 void* u_zombify(void* arg);
00141
00145 void* u_orphan_child(void* arg);
00146
00152 void* u_orphanify(void* arg);
00153
00160 void* u_nice(void* arg);
00161
00167 void* u_nice_pid(void* arg);
00168
00174 void* u_man(void* arg);
00175
00183 void* u_bg(void* arg);
00184
00192 void* u_fg(void* arg);
00193
00199 void* u_jobs(void* arg);
00200
00206 void* u_logout(void* arg);
00207
00212 int stoi(char* str);
00213
00220 void* u_hang(void* arg);
00221
00228 void* u_nohang(void* arg);
00229
00236 void* u_recur(void* arg);
00237
00245 void* u_crash(void* arg);
```

5.26 src/kernel/kernel.c File Reference

```
#include "kernel.h"
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include "../fat/err.h"
#include "../fat/fat_core.h"
#include "../util/logger/logger.h"
#include "../util/types/process-status.h"
#include "pcb.h"
```

Include dependency graph for kernel.c:



Macros

- `#define _POSIX_C_SOURCE 200809L`

Functions

- unsigned long `get_kernel_ticks ()`
Get the kernel ticks.
- void `alarm_handler (int signum)`
- `KernelState *` `getKernelState ()`
Get the kernel state.
- void `stop_handler (int signum)`
- void `int_handler (int signum)`
- void `kernel_set_up ()`
Initialize the kernel.
- void `start_kernel ()`
Start the kernel.
- void `check_blocked_processes ()`
Check if a process is blocked by sleep and wake it up if the time has come.
- void `add_process_to_scheduler (pcb_t *proc)`
Add a process to the appropriate run queue based on its priority. Also set its place in the pid pool.
- void `add_process_to_run_queue (pcb_t *proc)`
Add a process to the appropriate run queue based on its priority.

- void `remove_process_from_run_queue` (`pcb_t *proc`)
Remove a process from its current run queue.
- void `add_process_to_zombie_queue` (`pcb_t *proc`)
Add a process to the zombie queue.
- void `remove_process_from_zombie_queue` (`pcb_t *proc`)
Remove a process from the zombie queue.

Variables

- volatile unsigned long `kernel_ticks` = 0
- volatile bool `g_shutdown_requested` = false
Global variable to track if a shutdown has been requested.
- struct `KernelState` `k`

5.26.1 Macro Definition Documentation

5.26.1.1 `_POSIX_C_SOURCE`

```
#define _POSIX_C_SOURCE 200809L
```

5.26.2 Function Documentation

5.26.2.1 `add_process_to_run_queue()`

```
void add_process_to_run_queue (  
    pcb_t * proc)
```

Add a process to the appropriate run queue based on its priority.

Parameters

<code>proc</code>	The process to enqueue. Must not be NULL.
-------------------	---

5.26.2.2 `add_process_to_scheduler()`

```
void add_process_to_scheduler (  
    pcb_t * proc)
```

Add a process to the appropriate run queue based on its priority. Also set its place in the pid pool.

This schedules the process to be picked up by the kernel scheduler during its round-robin cycle. Must only be called on RUNNING processes.

Parameters

<code>proc</code>	The process to enqueue. Must not be NULL.
-------------------	---

5.26.2.3 `add_process_to_zombie_queue()`

```
void add_process_to_zombie_queue (  
    pcb_t * proc)
```

Add a process to the zombie queue.

Zombie processes have terminated but are waiting to be reaped by their parent. This function registers such processes into the `dq_ZOMBIE` queue.

Parameters

<i>proc</i>	The process to mark as zombie. Must not be NULL.
-------------	--

Add a process to the zombie queue.

Zombie processes have terminated but are waiting to be reaped by their parent. This function registers such processes into the dq_ZOMBIE queue.

Parameters

<i>proc</i>	The process to mark as zombie. Must not be NULL.
-------------	--

5.26.2.4 alarm_handler()

```
void alarm_handler (  
    int signum)
```

5.26.2.5 check_blocked_processes()

```
void check_blocked_processes ()
```

Check if a process is blocked by sleep and wake it up if the time has come.

5.26.2.6 get_kernel_ticks()

```
unsigned long get_kernel_ticks ()
```

Get the kernel ticks.

Returns

unsigned long The number of ticks since the kernel started.

5.26.2.7 getKernelState()

```
KernelState * getKernelState ()
```

Get the kernel state.

Returns

KernelState* The kernel state.

5.26.2.8 int_handler()

```
void int_handler (
    int signum)
```

5.26.2.9 kernel_set_up()

```
void kernel_set_up ()
```

Initialize the kernel.

This function is called once at kernel startup and should perform any necessary initialization tasks.

5.26.2.10 remove_process_from_run_queue()

```
void remove_process_from_run_queue (
    pcb_t * proc)
```

Remove a process from its current run queue.

This is typically used when a process blocks, is killed, exits, or is reprioritized. The function will look for the process by PID within its current priority-level queue and remove it if found.

Parameters

<i>proc</i>	The process to remove. Must not be NULL.
-------------	--

Remove a process from its current run queue.

This is typically used when a process blocks, is killed, exits, or is reprioritized. The function will look for the process by PID within its current priority-level queue and remove it if found.

Parameters

<i>proc</i>	The process to remove. Must not be NULL.
-------------	--

5.26.2.11 remove_process_from_zombie_queue()

```
void remove_process_from_zombie_queue (
    pcb_t * proc)
```

Remove a process from the zombie queue.

This is typically done when the parent reaps a zombie child via `s_waitpid`. The function matches by PID and removes the process.

Parameters

<i>proc</i>	The zombie process to remove. Must not be NULL.
-------------	---

Remove a process from the zombie queue.

This is typically done when the parent reaps a zombie child via `s_waitpid`. The function matches by PID and removes the process.

Parameters

<i>proc</i>	The zombie process to remove. Must not be NULL.
-------------	---

5.26.2.12 start_kernel()

```
void start_kernel ()
```

Start the kernel.

This function is called once at kernel startup and should perform any necessary initialization tasks.

5.26.2.13 stop_handler()

```
void stop_handler (  
    int signum)
```

5.26.3 Variable Documentation

5.26.3.1 g_shutdown_requested

```
volatile bool g_shutdown_requested = false
```

Global variable to track if a shutdown has been requested.

5.26.3.2 k

```
struct KernelState k
```

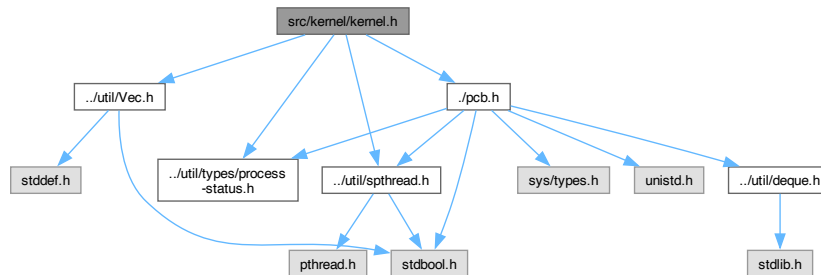
5.26.3.3 kernel_ticks

```
volatile unsigned long kernel_ticks = 0
```

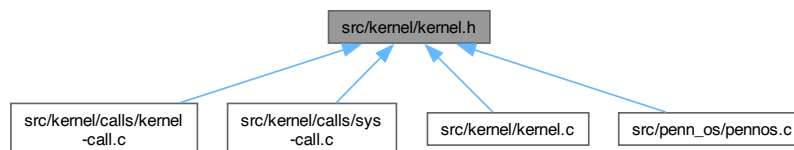
5.27 src/kernel/kernel.h File Reference

```
#include "../util/Vec.h"
#include "../util/spthread.h"
#include "../util/types/process-status.h"
#include "../pcb.h"
```

Include dependency graph for kernel.h:



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



Classes

- struct [KernelState](#)
The kernel state.

Macros

- #define [PROCESS_QUANTA](#) 19
- #define [MAX_PROC](#) 100

Typedefs

- typedef struct KernelState [KernelState](#)
The kernel state.

Functions

- `KernelState * getKernelState ()`
Get the kernel state.
- `void start_kernel ()`
Start the kernel.
- `void kernel_set_up ()`
Initialize the kernel.
- `void add_process_to_scheduler (pcb_t *proc)`
Add a process to the appropriate run queue based on its priority. Also set its place in the pid pool.
- `void add_process_to_run_queue (pcb_t *proc)`
Add a process to the appropriate run queue based on its priority.
- `void remove_process_from_run_queue (pcb_t *proc)`
Remove a process from its current run queue.
- `void add_process_to_zombie_queue (pcb_t *proc)`
Add a process to the zombie queue.
- `void remove_process_from_zombie_queue (pcb_t *proc)`
Remove a process from the zombie queue.
- `void check_blocked_processes ()`
Check if a process is blocked by sleep and wake it up if the time has come.
- `unsigned long get_kernel_ticks ()`
Get the kernel ticks.

Variables

- volatile bool `g_shutdown_requested`
Global variable to track if a shutdown has been requested.

5.27.1 Macro Definition Documentation

5.27.1.1 MAX_PROC

```
#define MAX_PROC 100
```

5.27.1.2 PROCESS_QUANTA

```
#define PROCESS_QUANTA 19
```

5.27.2 Typedef Documentation

5.27.2.1 KernelState

```
typedef struct KernelState KernelState
```

The kernel state.

This structure contains all the state information for the kernel.

5.27.3 Function Documentation

5.27.3.1 add_process_to_run_queue()

```
void add_process_to_run_queue (  
    pcb_t * proc)
```

Add a process to the appropriate run queue based on its priority.

Parameters

<i>proc</i>	The process to enqueue. Must not be NULL.
-------------	---

5.27.3.2 add_process_to_scheduler()

```
void add_process_to_scheduler (  
    pcb_t * proc)
```

Add a process to the appropriate run queue based on its priority. Also set its place in the pid pool.

This schedules the process to be picked up by the kernel scheduler during its round-robin cycle. Must only be called on RUNNING processes.

Parameters

<i>proc</i>	The process to enqueue. Must not be NULL.
-------------	---

5.27.3.3 add_process_to_zombie_queue()

```
void add_process_to_zombie_queue (  
    pcb_t * proc)
```

Add a process to the zombie queue.

Zombie processes have terminated but are waiting to be reaped by their parent. This function registers such processes into the dq_ZOMBIE queue.

Parameters

<i>proc</i>	The process to mark as zombie. Must not be NULL.
-------------	--

Add a process to the zombie queue.

Zombie processes have terminated but are waiting to be reaped by their parent. This function registers such processes into the dq_ZOMBIE queue.

Parameters

<i>proc</i>	The process to mark as zombie. Must not be NULL.
-------------	--

5.27.3.4 check_blocked_processes()

```
void check_blocked_processes ()
```

Check if a process is blocked by sleep and wake it up if the time has come.

5.27.3.5 get_kernel_ticks()

```
unsigned long get_kernel_ticks ()
```

Get the kernel ticks.

Returns

unsigned long The number of ticks since the kernel started.

5.27.3.6 getKernelState()

```
KernelState * getKernelState ()
```

Get the kernel state.

Returns

KernelState* The kernel state.

5.27.3.7 kernel_set_up()

```
void kernel_set_up ()
```

Initialize the kernel.

This function is called once at kernel startup and should perform any necessary initialization tasks.

5.27.3.8 remove_process_from_run_queue()

```
void remove_process_from_run_queue (  
    pcb_t * proc)
```

Remove a process from its current run queue.

This is typically used when a process blocks, is killed, exits, or is reprioritized. The function will look for the process by PID within its current priority-level queue and remove it if found.

Parameters

<i>proc</i>	The process to remove. Must not be NULL.
-------------	--

Remove a process from its current run queue.

This is typically used when a process blocks, is killed, exits, or is reprioritized. The function will look for the process by PID within its current priority-level queue and remove it if found.

Parameters

<i>proc</i>	The process to remove. Must not be NULL.
-------------	--

5.27.3.9 remove_process_from_zombie_queue()

```
void remove_process_from_zombie_queue (  
    pcb_t * proc)
```

Remove a process from the zombie queue.

This is typically done when the parent reaps a zombie child via `s_waitpid`. The function matches by PID and removes the process.

Parameters

<i>proc</i>	The zombie process to remove. Must not be NULL.
-------------	---

Remove a process from the zombie queue.

This is typically done when the parent reaps a zombie child via `s_waitpid`. The function matches by PID and removes the process.

Parameters

<i>proc</i>	The zombie process to remove. Must not be NULL.
-------------	---

5.27.3.10 start_kernel()

```
void start_kernel ()
```

Start the kernel.

This function is called once at kernel startup and should perform any necessary initialization tasks.

5.27.4 Variable Documentation**5.27.4.1 g_shutdown_requested**

```
volatile bool g_shutdown_requested [extern]
```

Global variable to track if a shutdown has been requested.

5.28 kernel.h

[Go to the documentation of this file.](#)

```

00001 #include "../util/Vec.h"
00002 #include "../util/spthread.h"
00003 #include "../util/types/process-status.h"
00004 #include "../pcb.h"
00005 #ifndef PROCESS_QUANTA
00006 #define PROCESS_QUANTA 19
00007 #endif
00008
00009 #ifndef MAX_PROC
00010 #define MAX_PROC 100
00011 #endif
00012
00018 typedef struct KernelState
00019 {
00020     Deque *dq_RUNNING[3];
00021     Deque *dq_ZOMBIE;
00022     Deque *dq_DEAD; // Ensure this exists if used, otherwise remove
00023     Deque *dq_BLOCKED;
00024     Deque *dq_STOPPED; // Make sure this is uncommented/added
00025     int curr_thread_num;
00026     int process_quanta;
00027     pcb_t *curr_process;
00028     Vec current_processes;
00029     pid_t terminal_owner_pid; // Add terminal owner tracking
00030 } KernelState;
00031
00035 extern volatile bool g_shutdown_requested;
00036
00042 KernelState *getKernelState();
00043
00050 void start_kernel();
00051
00058 void kernel_set_up();
00059
00069 void add_process_to_scheduler(pcb_t *proc);
00070
00077 void add_process_to_run_queue(pcb_t *proc);
00078
00089 void remove_process_from_run_queue(pcb_t *proc);
00090
00100 void add_process_to_zombie_queue(pcb_t *proc);
00101
00110 void remove_process_from_zombie_queue(pcb_t *proc);
00111
00120 void check_blocked_processes();
00121
00130 unsigned long get_kernel_ticks();

```

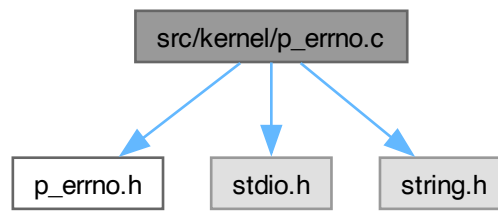
5.29 src/kernel/p_errno.c File Reference

```

#include "p_errno.h"
#include <stdio.h>
#include <string.h>

```

Include dependency graph for p_errno.c:



Functions

- void `u_perror` (const char *msg)
Print an error message to the standard error stream.

Variables

- int `P_ERRNO` = 0
Error number for the last system call.

5.29.1 Function Documentation

5.29.1.1 u_perror()

```
void u_perror (  
    const char * msg)
```

Print an error message to the standard error stream.

Parameters

<code>msg</code>	The message to print.
------------------	-----------------------

5.29.2 Variable Documentation

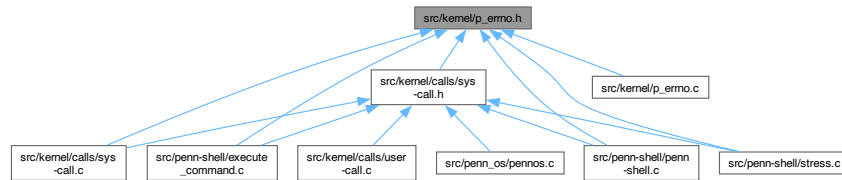
5.29.2.1 P_ERRNO

```
int P_ERRNO = 0
```

Error number for the last system call.

5.30 src/kernel/p_errno.h File Reference

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



Macros

- `#define P_ERRNO_SUCCESS 0`
- `#define P_ERRNO_FILE_NOT_FOUND 1`
- `#define P_ERRNO_PERMISSION 2`
- `#define P_ERRNO_WRITE_CONFLICT 3`
- `#define P_ERRNO_INTERNAL 4`
- `#define P_ERRNO_INVALID_ARG 5`
- `#define P_ERRNO_NO_SPACE 6`
- `#define P_ERRNO_NOT_MOUNTED 7`
- `#define P_ERRNO_ALREADY_MOUNTED 8`
- `#define P_ERRNO_INVALID_FD 9`
- `#define P_ERRNO_NOT_A_DIRECTORY 10`
- `#define P_ERRNO_INVALID_OPERATION 11`
- `#define P_ERRNO_INVALID_WHENCE 12`
- `#define P_ERRNO_INVALID_OFFSET 13`
- `#define P_ERRNO_ESRCH 14`
- `#define P_ERRNO_EINVAL 15`
- `#define P_ERRNO_ECHILD 16`
- `#define P_ERRNO_UNKNOWN 255`

Functions

- void `u_perror` (const char *msg)
Print an error message to the standard error stream.

Variables

- int `P_ERRNO`
Error number for the last system call.

5.30.1 Macro Definition Documentation

5.30.1.1 P_ERRNO_ALREADY_MOUNTED

```
#define P_ERRNO_ALREADY_MOUNTED 8
```

5.30.1.2 P_ERRNO_ECHILD

```
#define P_ERRNO_ECHILD 16
```

5.30.1.3 P_ERRNO_EINVAL

```
#define P_ERRNO_EINVAL 15
```

5.30.1.4 P_ERRNO_ESRCH

```
#define P_ERRNO_ESRCH 14
```

5.30.1.5 P_ERRNO_FILE_NOT_FOUND

```
#define P_ERRNO_FILE_NOT_FOUND 1
```

5.30.1.6 P_ERRNO_INTERNAL

```
#define P_ERRNO_INTERNAL 4
```

5.30.1.7 P_ERRNO_INVALID_ARG

```
#define P_ERRNO_INVALID_ARG 5
```

5.30.1.8 P_ERRNO_INVALID_FD

```
#define P_ERRNO_INVALID_FD 9
```

5.30.1.9 P_ERRNO_INVALID_OFFSET

```
#define P_ERRNO_INVALID_OFFSET 13
```

5.30.1.10 P_ERRNO_INVALID_OPERATION

```
#define P_ERRNO_INVALID_OPERATION 11
```

5.30.1.11 P_ERRNO_INVALID_WHENCE

```
#define P_ERRNO_INVALID_WHENCE 12
```

5.30.1.12 P_ERRNO_NO_SPACE

```
#define P_ERRNO_NO_SPACE 6
```

5.30.1.13 P_ERRNO_NOT_A_DIRECTORY

```
#define P_ERRNO_NOT_A_DIRECTORY 10
```

5.30.1.14 P_ERRNO_NOT_MOUNTED

```
#define P_ERRNO_NOT_MOUNTED 7
```

5.30.1.15 P_ERRNO_PERMISSION

```
#define P_ERRNO_PERMISSION 2
```

5.30.1.16 P_ERRNO_SUCCESS

```
#define P_ERRNO_SUCCESS 0
```

5.30.1.17 P_ERRNO_UNKNOWN

```
#define P_ERRNO_UNKNOWN 255
```

5.30.1.18 P_ERRNO_WRITE_CONFLICT

```
#define P_ERRNO_WRITE_CONFLICT 3
```

5.30.2 Function Documentation

5.30.2.1 u_perror()

```
void u_perror (  
    const char * msg)
```

Print an error message to the standard error stream.

Parameters

<i>msg</i>	The message to print.
------------	-----------------------

5.30.3 Variable Documentation

5.30.3.1 P_ERRNO

```
int P_ERRNO [extern]
```

Error number for the last system call.

5.31 p_errno.h

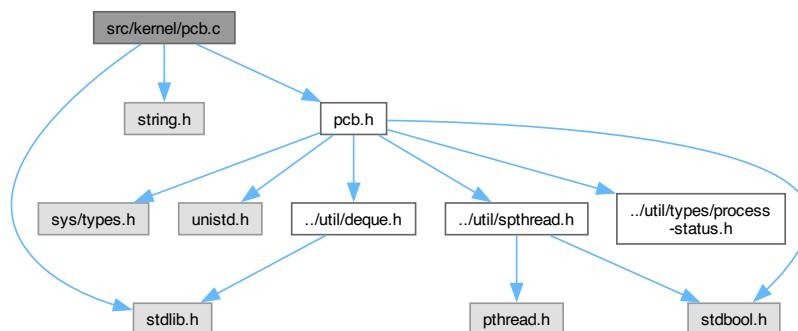
[Go to the documentation of this file.](#)

```
00001 #ifndef P_ERRNO_H
00002 #define P_ERRNO_H
00003
00004 #define P_ERRNO_SUCCESS 0           // No error
00005 #define P_ERRNO_FILE_NOT_FOUND 1    // File not found (maps from FAT: FILE_NOT_FOUND)
00006 #define P_ERRNO_PERMISSION 2        // Permission denied (maps from FAT: PERMISSION_DENIED)
00007 #define P_ERRNO_WRITE_CONFLICT 3    // File open for writing elsewhere
00008 #define P_ERRNO_INTERNAL 4          // General internal kernel/OS error (maps from FAT: MEMORY_ERROR,
FILE_SYSTEM)
00009 #define P_ERRNO_INVALID_ARG 5        // Invalid argument provided to syscall (e.g., bad priority)
(maps from FAT: INVALID_ARGS)
00010 #define P_ERRNO_NO_SPACE 6           // No space left on device (maps from FAT: NO_SPACE)
00011 #define P_ERRNO_NOT_MOUNTED 7        // Filesystem not mounted (maps from FAT: FS_NOT_MOUNTED)
00012 #define P_ERRNO_ALREADY_MOUNTED 8    // Filesystem already mounted (maps from FAT: FS_ALREADY_MOUNTED)
00013 #define P_ERRNO_INVALID_FD 9         // Invalid file descriptor (maps from FAT: INVALID_FD)
00014 #define P_ERRNO_NOT_A_DIRECTORY 10   // Path is not a directory (maps from FAT: NOT_A_DIRECTORY)
00015 #define P_ERRNO_INVALID_OPERATION 11 // Operation not permitted (e.g., closing stdin, bad mode) (maps
from FAT: INVALID_OPERATION)
00016 #define P_ERRNO_INVALID_WHENCE 12    // Invalid 'whence' argument for lseek (maps from FAT:
INVALID_WHENCE)
00017 #define P_ERRNO_INVALID_OFFSET 13    // Invalid 'offset' argument for lseek (maps from FAT:
INVALID_OFFSET)
00018 #define P_ERRNO_ESRCH 14             // No such process
00019 #define P_ERRNO_EINVAL 15            // Invalid argument (general, e.g., bad signal)
00020 #define P_ERRNO_ECHILD 16           // No child processes (for waitpid)
00021 #define P_ERRNO_UNKNOWN 255         // Unknown error
00022
00023 extern int P_ERRNO; // Make sure this is declared extern @Nikita pls
00024
00030 void u_perror(const char *msg);
00031
00032 #endif // P_ERRNO_H
```

5.32 src/kernel/pcb.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include "pcb.h"
```

Include dependency graph for pcb.c:



Functions

- void `pcb_initialize_fd_table` (`pcb_t` *pcb)
Initialize the file descriptor table for a process.
- int `pcb_add_fd` (`pcb_t` *pcb, int fd, const char *fname, int mode, int offset)
Add a file descriptor to the process's file descriptor table.
- int `pcb_remove_fd` (`pcb_t` *pcb, int fd)
Remove a file descriptor from the process's file descriptor table.
- `ProcessFDNode` * `pcb_get_fd` (`pcb_t` *pcb, int fd_num)
Get a file descriptor from the process's file descriptor table.
- `pcb_t` * `pcb_create` (`pid_t` pid, `pid_t` ppid, int priority_level, char *name, bool foreground)
- void `pcb_destroy` (`pcb_t` *pcb)
- int `pcb_set_fd` (`pcb_t` *pcb, int fd_num, const char *fname, int mode, int offset)
Set a file descriptor in the process's file descriptor table.

5.32.1 Function Documentation

5.32.1.1 `pcb_add_fd()`

```
int pcb_add_fd (
    pcb_t * pcb,
    int kernel_fd,
    const char * fname,
    int mode,
    int offset)
```

Add a file descriptor to the process's file descriptor table.

Parameters

<i>pcb</i>	The process control block to add the file descriptor to.
<i>kernel_↵ _fd</i>	The kernel file descriptor number.
<i>fname</i>	The name of the file.
<i>mode</i>	The mode of the file.
<i>offset</i>	The offset of the file.

Returns

int The file descriptor number.

5.32.1.2 `pcb_create()`

```
pcb_t * pcb_create (
    pid_t pid,
    pid_t ppid,
    int priority_level,
    char * name,
    bool foreground)
```

5.32.1.3 pcb_destroy()

```
void pcb_destroy (  
    pcb_t * pcb)
```

5.32.1.4 pcb_get_fd()

```
ProcessFDNode * pcb_get_fd (  
    pcb_t * pcb,  
    int fd_num)
```

Get a file descriptor from the process's file descriptor table.

Parameters

<i>pcb</i>	The process control block to get the file descriptor from.
<i>fd_num</i>	The file descriptor number.

Returns

ProcessFDNode* The file descriptor node.

5.32.1.5 pcb_initialize_fd_table()

```
void pcb_initialize_fd_table (  
    pcb_t * pcb)
```

Initialize the file descriptor table for a process.

Parameters

<i>pcb</i>	The process control block to initialize.
------------	--

5.32.1.6 pcb_remove_fd()

```
int pcb_remove_fd (  
    pcb_t * pcb,  
    int fd_num)
```

Remove a file descriptor from the process's file descriptor table.

Parameters

<i>pcb</i>	The process control block to remove the file descriptor from.
<i>fd_num</i>	The file descriptor number.

Returns

int The file descriptor number.

5.32.1.7 pcb_set_fd()

```
int pcb_set_fd (
    pcb_t * pcb,
    int fd_num,
    const char * fname,
    int mode,
    int offset)
```

Set a file descriptor in the process's file descriptor table.

Parameters

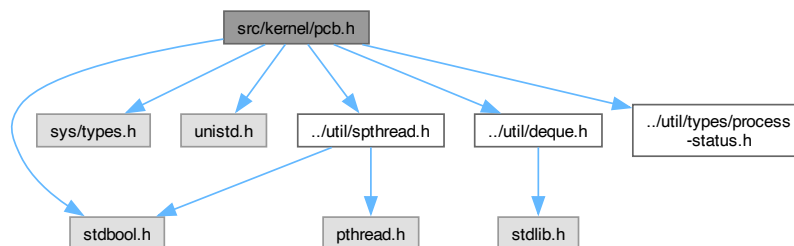
<i>pcb</i>	The process control block to set the file descriptor in.
<i>fd_num</i>	The file descriptor number.
<i>fname</i>	The name of the file.
<i>mode</i>	The mode of the file.
<i>offset</i>	The offset of the file.

Returns

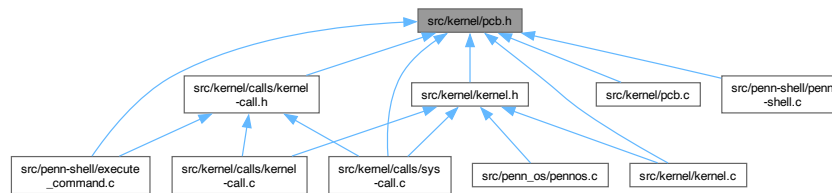
int The file descriptor number.

5.33 src/kernel/pcb.h File Reference

```
#include <stdbool.h>
#include <sys/types.h>
#include <unistd.h>
#include "../util/deque.h"
#include "../util/spthread.h"
#include "../util/types/process-status.h"
Include dependency graph for pcb.h:
```



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



Classes

- struct [ProcessFDNode](#)
The Process File Descriptor *Node* structure.
- struct [pcb_t](#)
The Process Control Block (PCB) structure.

Typedefs

- typedef struct ProcessFDNode [ProcessFDNode](#)
The Process File Descriptor *Node* structure.
- typedef struct pcb_t [pcb_t](#)
The Process Control Block (PCB) structure.

Functions

- void [pcb_initialize_fd_table](#) ([pcb_t](#) *pcb)
Initialize the file descriptor table for a process.
- int [pcb_add_fd](#) ([pcb_t](#) *pcb, int kernel_fd, const char *fname, int mode, int offset)
Add a file descriptor to the process's file descriptor table.
- [ProcessFDNode](#) * [pcb_get_fd](#) ([pcb_t](#) *pcb, int fd_num)
Get a file descriptor from the process's file descriptor table.
- int [pcb_remove_fd](#) ([pcb_t](#) *pcb, int fd_num)
Remove a file descriptor from the process's file descriptor table.
- int [pcb_set_fd](#) ([pcb_t](#) *pcb, int fd_num, const char *fname, int mode, int offset)
Set a file descriptor in the process's file descriptor table.

5.33.1 Typedef Documentation

5.33.1.1 pcb_t

```
typedef struct pcb_t pcb_t
```

The Process Control Block (PCB) structure.

This structure represents a process control block (PCB) in the kernel.

5.33.1.2 ProcessFDNode

```
typedef struct ProcessFDNode ProcessFDNode
```

The Process File Descriptor [Node](#) structure.

This structure represents a file descriptor node in the process's file descriptor table.

5.33.2 Function Documentation

5.33.2.1 pcb_add_fd()

```
int pcb_add_fd (
    pcb_t * pcb,
    int kernel_fd,
    const char * fname,
    int mode,
    int offset)
```

Add a file descriptor to the process's file descriptor table.

Parameters

<i>pcb</i>	The process control block to add the file descriptor to.
<i>kernel_fd</i>	The kernel file descriptor number.
<i>fname</i>	The name of the file.
<i>mode</i>	The mode of the file.
<i>offset</i>	The offset of the file.

Returns

int The file descriptor number.

5.33.2.2 pcb_get_fd()

```
ProcessFDNode * pcb_get_fd (
    pcb_t * pcb,
    int fd_num)
```

Get a file descriptor from the process's file descriptor table.

Parameters

<i>pcb</i>	The process control block to get the file descriptor from.
<i>fd_num</i>	The file descriptor number.

Returns

ProcessFDNode* The file descriptor node.

5.33.2.3 pcb_initialize_fd_table()

```
void pcb_initialize_fd_table (
    pcb_t * pcb)
```

Initialize the file descriptor table for a process.

Parameters

<i>pcb</i>	The process control block to initialize.
------------	--

5.33.2.4 pcb_remove_fd()

```
int pcb_remove_fd (  
    pcb_t * pcb,  
    int fd_num)
```

Remove a file descriptor from the process's file descriptor table.

Parameters

<i>pcb</i>	The process control block to remove the file descriptor from.
<i>fd_num</i>	The file descriptor number.

Returns

int The file descriptor number.

5.33.2.5 pcb_set_fd()

```
int pcb_set_fd (  
    pcb_t * pcb,  
    int fd_num,  
    const char * fname,  
    int mode,  
    int offset)
```

Set a file descriptor in the process's file descriptor table.

Parameters

<i>pcb</i>	The process control block to set the file descriptor in.
<i>fd_num</i>	The file descriptor number.
<i>fname</i>	The name of the file.
<i>mode</i>	The mode of the file.
<i>offset</i>	The offset of the file.

Returns

int The file descriptor number.

5.34 pcb.h

[Go to the documentation of this file.](#)

```

00001 #ifndef PCB_H
00002 #define PCB_H
00003
00004 #include <stdbool.h>
00005 #include <sys/types.h>
00006 #include <unistd.h>
00007 #include "../util/deque.h"
00008 #include "../util/spthread.h"
00009 #include "../util/types/process-status.h"
00010
00011 // The Process Control Block (PCB) structure for kernel processes
00012 // This should match the current implementation as used in sys-call.c and
00013 // kernel-call.c
00014
00021 typedef struct ProcessFDNode
00022 {
00023     int fd_num; // process-level FD
00024     char fname[32];
00025     int mode;
00026     int offset;
00027     struct ProcessFDNode *next;
00028 } ProcessFDNode;
00029
00035 typedef struct pcb_t
00036 {
00037     pid_t pid; // Process ID
00038     pid_t ppid; // Parent Process ID
00039     int priority_level; // Priority level (0, 1, 2)
00040     int term_signal; // Signal that terminated the process
00041     int stop_signal; // Signal that stopped the process
00042     ProcessStatus status; // Process status (enum)
00043     bool status_changed; // Flag to indicate status change
00044     unsigned long wake_up_tick; // Tick count when sleep should end
00045     Deque *file_descriptors; // Set of file descriptors
00046     ProcessFDNode *fd_table; // Linked list of file descriptor nodes
00047     spthread_t *thread; // Pointer to the thread of execution
00048     char *name; // Process name
00049     bool foreground; // Foreground/background process
00050 } pcb_t;
00051
00057 void pcb_initialize_fd_table(pcb_t *pcb);
00058
00069 int pcb_add_fd(pcb_t *pcb, int kernel_fd, const char *fname, int mode, int offset);
00070
00078 ProcessFDNode *pcb_get_fd(pcb_t *pcb, int fd_num);
00079
00087 int pcb_remove_fd(pcb_t *pcb, int fd_num);
00088
00099 int pcb_set_fd(pcb_t *pcb, int fd_num, const char *fname, int mode, int offset);
00100
00101 #endif // PCB_H

```

5.35 src/penn-shell/execute_command.c File Reference

```

#include "execute_command.h"
#include <ctype.h>
#include <errno.h>
#include <limits.h>
#include <signals.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "../fat/fat_core.h"
#include "../fat/fat_kernel.h"
#include "../kernel/calls/kernel-call.h"
#include "../kernel/calls/sys-call.h"
#include "../kernel/calls/user-call.h"
#include "../kernel/pcb.h"

```


5.35.1.3 execute_single_command()

```
pid_t execute_single_command (  
    struct parsed_command * cmd,  
    int cmd_index,  
    int input_fd,  
    int output_fd,  
    const char * full_command_line)
```

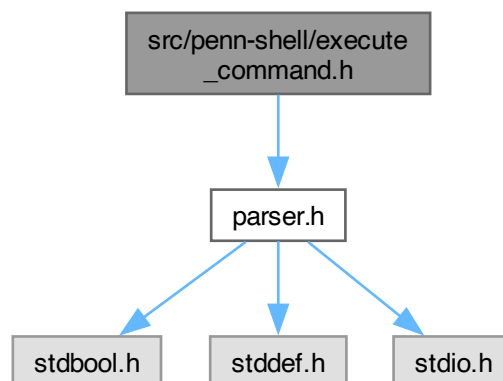
5.35.1.4 handle_shell_builtin()

```
int handle_shell_builtin (  
    struct parsed_command * cmd)
```

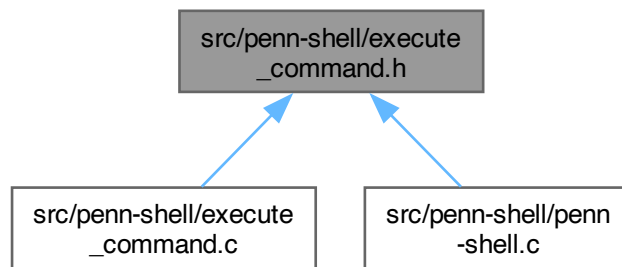
5.36 src/penn-shell/execute_command.h File Reference

```
#include "parser.h"
```

Include dependency graph for execute_command.h:



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



Functions

- int `execute_command` (struct `parsed_command` *cmd)
Execute a parsed command via Penn-Shell.

5.36.1 Function Documentation

5.36.1.1 `execute_command()`

```
int execute_command (  
    struct parsed_command * cmd)
```

Execute a parsed command via Penn-Shell.

Parameters

<code>cmd</code>	The parsed command to execute.
------------------	--------------------------------

Returns

int The exit status of the command.

5.37 `execute_command.h`

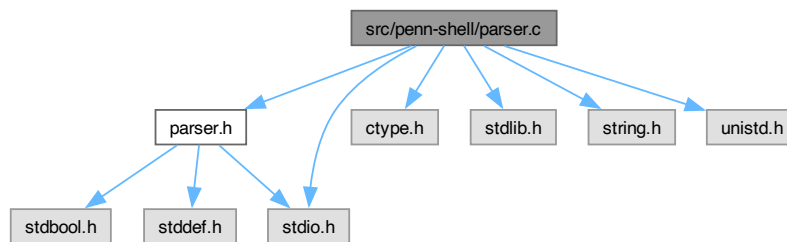
[Go to the documentation of this file.](#)

```
00001 #include "parser.h"  
00002  
00009 int execute_command(struct parsed_command *cmd);
```

5.38 src/penn-shell/parser.c File Reference

```
#include "parser.h"
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
```

Include dependency graph for parser.c:



Macros

- #define [JUMP_OUT](#)(code)

Functions

- int [parse_command](#) (const char *const cmd_line, struct [parsed_command](#) **const result)
Parse a command line into a [parsed_command](#) structure.
- void [print_parsed_command](#) (const struct [parsed_command](#) *const cmd)
Print a parsed command line.
- void [print_parsed_command_without_end](#) (struct [parsed_command](#) *const cmd)
Print a parsed command line without the trailing newline.
- void [print_parser_errcode](#) (FILE *output, int err_code)
Print a debugging message for a parser error code.

5.38.1 Macro Definition Documentation

5.38.1.1 JUMP_OUT

```
#define JUMP_OUT(  
    code)
```

Value:

```
do {  
    ret_code = code; \
    goto PROCESS_ERROR; \
} while (0)
```

5.38.2 Function Documentation

5.38.2.1 `parse_command()`

```
int parse_command (
    const char * cmd_line,
    struct parsed\_command ** result)
```

Parse a command line into a [parsed_command](#) structure.

Parameters

<i>cmd_line</i>	The command line to parse.
<i>result</i>	A pointer to a pointer to a parsed_command structure.

Returns

int 0 on success, -1 on failure.

layout of memory for struct [parsed_command](#) bool is_background; bool is_file_append;

const char *stdin_file; const char *stdout_file;

size_t num_commands;

commands are pointers to arguments char **commands[num_commands];

below are hidden in memory **

arguments are pointers to original_string + num_commands because all argv are null-terminated char *arguments[total_strings + num_commands];

original_string is a copy of the cmdline but with each token null-terminated char *original_string;

5.38.2.2 `print_parsed_command()`

```
void print_parsed_command (
    const struct parsed\_command * cmd)
```

Print a parsed command line.

Parameters

<i>cmd</i>	The parsed command to print.
------------	------------------------------

5.38.2.3 `print_parsed_command_without_end()`

```
void print_parsed_command_without_end (
    struct parsed\_command * cmd)
```

Print a parsed command line without the trailing newline.

Parameters

<i>cmd</i>	The parsed command to print.
------------	------------------------------

5.38.2.4 print_parser_errcode()

```
void print_parser_errcode (  
    FILE * output,  
    int err_code)
```

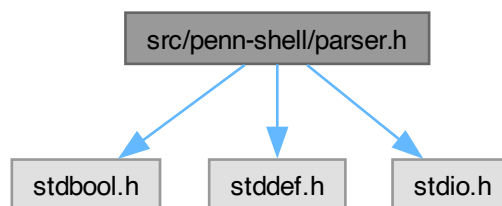
Print a debugging message for a parser error code.

Parameters

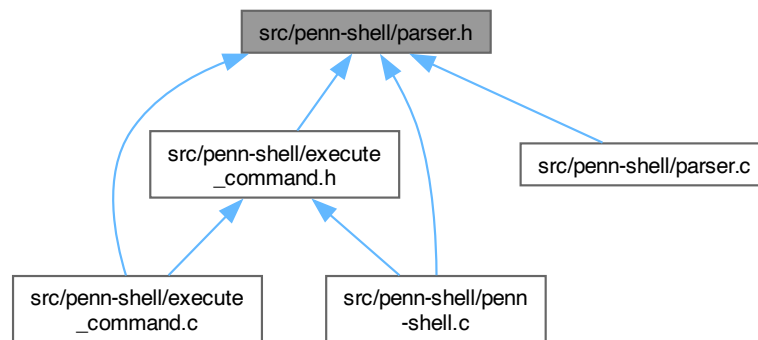
<i>output</i>	The output stream to print to.
<i>err_code</i>	The error code to print.

5.39 src/penn-shell/parser.h File Reference

```
#include <stdbool.h>  
#include <stddef.h>  
#include <stdio.h>  
Include dependency graph for parser.h:
```



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



Classes

- struct `parsed_command`
struct `parsed_command` stored all necessary information needed for penn-shell.

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)`
Parse a command line into a `parsed_command` structure.
- `void print_parsed_command (const struct parsed_command *cmd)`
Print a parsed command line.
- `void print_parser_errcode (FILE *output, int err_code)`
Print a debugging message for a parser error code.
- `void print_parsed_command_without_end (struct parsed_command *cmd)`
Print a parsed command line without the trailing newline.

5.39.1 Macro Definition Documentation

5.39.1.1 EXPECT_COMMANDS

```
#define EXPECT_COMMANDS 7
```

5.39.1.2 EXPECT_INPUT_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

5.39.1.3 EXPECT_OUTPUT_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

5.39.1.4 UNEXPECTED_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

5.39.1.5 UNEXPECTED_FILE_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

5.39.1.6 UNEXPECTED_FILE_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

5.39.1.7 UNEXPECTED_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

5.39.2 Function Documentation

5.39.2.1 parse_command()

```
int parse_command (
    const char * cmd_line,
    struct parsed_command ** result)
```

Parse a command line into a [parsed_command](#) structure.

Parameters

<i>cmd_line</i>	The command line to parse.
<i>result</i>	A pointer to a pointer to a parsed_command structure.

Returns

int 0 on success, -1 on failure.

layout of memory for struct [parsed_command](#) bool is_background; bool is_file_append;

const char *stdin_file; const char *stdout_file;

size_t num_commands;

commands are pointers to arguments char **commands[num_commands];

below are hidden in memory **

arguments are pointers to original_string + num_commands because all argv are null-terminated char *arguments[total_strings + num_commands];

original_string is a copy of the cmdline but with each token null-terminated char *original_string;

5.39.2.2 `print_parsed_command()`

```
void print_parsed_command (
    const struct parsed_command * cmd)
```

Print a parsed command line.

Parameters

<i>cmd</i>	The parsed command to print.
------------	------------------------------

5.39.2.3 `print_parsed_command_without_end()`

```
void print_parsed_command_without_end (
    struct parsed_command * cmd)
```

Print a parsed command line without the trailing newline.

Parameters

<i>cmd</i>	The parsed command to print.
------------	------------------------------

5.39.2.4 `print_parser_errcode()`

```
void print_parser_errcode (
    FILE * output,
    int err_code)
```

Print a debugging message for a parser error code.

Parameters

<i>output</i>	The output stream to print to.
<i>err_code</i>	The error code to print.

5.40 `parser.h`

[Go to the documentation of this file.](#)

```
00001 /* Penn-Shell Parser
00002     hanbangw, 21fa */
00003
00004 #pragma once
00005
00006 #include <stdbool.h>
00007 #include <stddef.h>
00008 #include <stdio.h>
00009
00010 #define UNEXPECTED_FILE_INPUT 1
00011 #define UNEXPECTED_FILE_OUTPUT 2
00012 #define UNEXPECTED_PIPELINE 3
00013 #define UNEXPECTED_AMPERSAND 4
00014 #define EXPECT_INPUT_FILENAME 5
```

```

00015 #define EXPECT_OUTPUT_FILENAME 6
00016 #define EXPECT_COMMANDS 7
00017
00023 struct parsed_command
00024 {
00025     // indicates the command shall be executed in background
00026     // (ends with an ampersand '&')
00027     bool is_background;
00028
00029     // indicates if the stdout_file shall be opened in append mode
00030     // ignore this value when stdout_file is NULL
00031     bool is_file_append;
00032
00033     // filename for redirecting input from
00034     const char *stdin_file;
00035
00036     // filename for redirecting output to
00037     const char *stdout_file;
00038
00039     // number of commands (pipeline stages)
00040     size_t num_commands;
00041
00042     // an array to a list of arguments
00043     // size of `commands` is `num_commands`
00044     char **commands[];
00045 };
00046
00054 int parse_command(const char *cmd_line, struct parsed_command **result);
00055
00061 void print_parsed_command(const struct parsed_command *cmd);
00062
00069 void print_parser_errcode(FILE *output, int err_code);
00070
00076 void print_parsed_command_without_end(struct parsed_command *cmd);

```

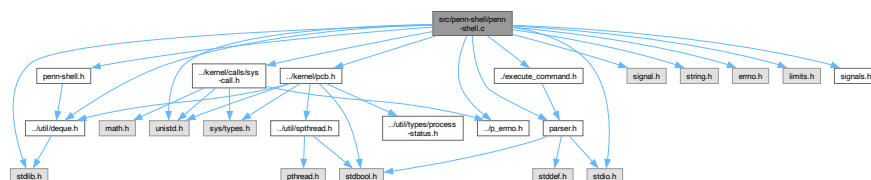
5.41 src/penn-shell/penn-shell.c File Reference

```

#include "penn-shell.h"
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "../kernel/calls/sys-call.h"
#include "../kernel/p_errno.h"
#include "../execute_command.h"
#include "parser.h"
#include "../kernel/pcb.h"
#include "../util/deque.h"
#include <errno.h>
#include <limits.h>
#include <signals.h>

```

Include dependency graph for penn-shell.c:



Macros

- #define `_POSIX_C_SOURCE` 200809L
- #define `LINE_BUFFER_CHUNK_SIZE` 128

Functions

- void `free_job` (void *data)
Frees memory associated with a job.
- void `initialize_job_control` ()
Initializes job control mechanisms.
- Job * `find_job_by_pid` (pid_t pid)
Finds a job by its process ID.
- Job * `find_job_by_id` (int job_id)
Finds a job by its job ID.
- Job * `find_last_job` ()
Finds the most recently added job.
- Job * `find_last_stopped_job` ()
Finds the most recently stopped job.
- char * `reconstruct_command` (char *argv[])
Reconstructs a command string from an argument vector.
- void `add_job` (pid_t pid, struct `parsed_command` *cmd, JobStatus status)
Adds a new job to the job list.
- void `remove_job_by_pid` (pid_t pid)
Removes a job from the job list by its process ID.
- int `read_line_from_fd` (int fd, char **line_buffer, size_t *buffer_size, size_t *line_len)
- void * `shell` (void *args)
Main shell function to be run as a thread.

Variables

- Deque * `job_list`
Global job list.
- int `next_job_id`
Next available job ID.
- pid_t `shell_pid`
Process ID of the shell.
- pid_t `terminal_controller_pid` = -1
Process ID of the terminal controller.

5.41.1 Macro Definition Documentation

5.41.1.1 _POSIX_C_SOURCE

```
#define _POSIX_C_SOURCE 200809L
```

5.41.1.2 LINE_BUFFER_CHUNK_SIZE

```
#define LINE_BUFFER_CHUNK_SIZE 128
```

5.41.2 Function Documentation

5.41.2.1 add_job()

```
void add_job (
    pid_t pid,
    struct parsed_command * cmd,
    JobStatus status)
```

Adds a new job to the job list.

Parameters

<i>pid</i>	Process ID of the new job.
<i>cmd</i>	Parsed command structure for the job.
<i>status</i>	Initial status of the job.

5.41.2.2 find_job_by_id()

```
Job * find_job_by_id (  
    int job_id)
```

Finds a job by its job ID.

Parameters

<i>job_id</i>	Job ID to search for.
---------------	-----------------------

Returns

Job* Pointer to the found job, or NULL if not found.

5.41.2.3 find_job_by_pid()

```
Job * find_job_by_pid (  
    pid_t pid)
```

Finds a job by its process ID.

Parameters

<i>pid</i>	Process ID to search for.
------------	---------------------------

Returns

Job* Pointer to the found job, or NULL if not found.

5.41.2.4 find_last_job()

```
Job * find_last_job ()
```

Finds the most recently added job.

Returns

Job* Pointer to the last job, or NULL if no jobs exist.

5.41.2.5 find_last_stopped_job()

```
Job * find_last_stopped_job ()
```

Finds the most recently stopped job.

Returns

Job* Pointer to the last stopped job, or NULL if no stopped jobs exist.

5.41.2.6 free_job()

```
void free_job (  
    void * data)
```

Frees memory associated with a job.

Parameters

<i>data</i>	Pointer to the job to be freed.
-------------	---------------------------------

5.41.2.7 initialize_job_control()

```
void initialize_job_control ()
```

Initializes job control mechanisms.

5.41.2.8 read_line_from_fd()

```
int read_line_from_fd (  
    int fd,  
    char ** line_buffer,  
    size_t * buffer_size,  
    size_t * line_len)
```

5.41.2.9 reconstruct_command()

```
char * reconstruct_command (  
    char * argv[])
```

Reconstructs a command string from an argument vector.

Parameters

<i>argv</i>	Array of command arguments.
-------------	-----------------------------

Returns

char* Reconstructed command string.

5.41.2.10 remove_job_by_pid()

```
void remove_job_by_pid (  
    pid_t pid)
```

Removes a job from the job list by its process ID.

Parameters

<i>pid</i>	Process ID of the job to remove.
------------	----------------------------------

5.41.2.11 shell()

```
void * shell (  
    void * args)
```

Main shell function to be run as a thread.

Parameters

<i>args</i>	Arguments passed to the shell thread.
-------------	---------------------------------------

Returns

void* Return value of the thread.

5.41.3 Variable Documentation

5.41.3.1 job_list

```
Deque* job_list
```

Global job list.

5.41.3.2 next_job_id

```
int next_job_id
```

Next available job ID.

5.41.3.3 shell_pid

```
pid_t shell_pid
```

Process ID of the shell.

5.41.3.4 terminal_controller_pid

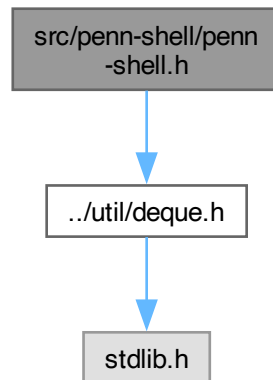
```
pid_t terminal_controller_pid = -1
```

Process ID of the terminal controller.

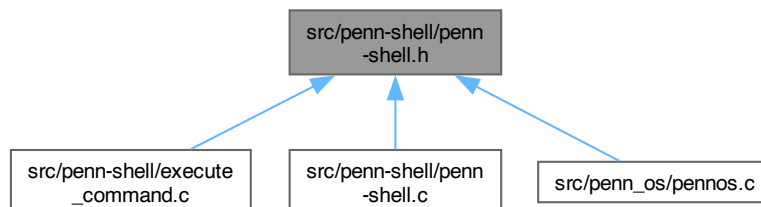
5.42 src/penn-shell/penn-shell.h File Reference

```
#include "../util/deque.h"
```

Include dependency graph for penn-shell.h:



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



Classes

- struct [Job](#)
struct for job

Macros

- #define [PENSHELL_H](#)
- #define [MAX_MESSAGE_SIZE](#) 4096
- #define [SHELL_PROMPT](#) "\$"

Typedefs

- typedef int [pid_t](#)
- typedef struct Job [Job](#)
struct for job

Enumerations

- enum [JobStatus](#) { [JOB_STATUS_RUNNING](#) , [JOB_STATUS_STOPPED](#) , [JOB_STATUS_DONE](#) }
enum for job status

Functions

- void * [shell](#) (void *args)
Main shell function to be run as a thread.
- void [free_job](#) (void *data)
Frees memory associated with a job.
- void [initialize_job_control](#) ()
Initializes job control mechanisms.
- [Job](#) * [find_job_by_pid](#) ([pid_t](#) pid)
Finds a job by its process ID.
- [Job](#) * [find_job_by_id](#) (int job_id)
Finds a job by its job ID.
- [Job](#) * [find_last_job](#) ()
Finds the most recently added job.
- [Job](#) * [find_last_stopped_job](#) ()
Finds the most recently stopped job.
- void [add_job](#) ([pid_t](#) pid, struct [parsed_command](#) *cmd, [JobStatus](#) status)
Adds a new job to the job list.
- void [remove_job_by_pid](#) ([pid_t](#) pid)
Removes a job from the job list by its process ID.
- char * [reconstruct_command](#) (char *argv[])
Reconstructs a command string from an argument vector.

Variables

- [Deque](#) * [job_list](#)
Global job list.
- int [next_job_id](#)
Next available job ID.
- [pid_t](#) [shell_pid](#)
Process ID of the shell.
- [pid_t](#) [terminal_controller_pid](#)
Process ID of the terminal controller.

5.42.1 Macro Definition Documentation

5.42.1.1 MAX_MESSAGE_SIZE

```
#define MAX_MESSAGE_SIZE 4096
```

5.42.1.2 PENSHELL_H

```
#define PENSHELL_H
```

5.42.1.3 SHELL_PROMPT

```
#define SHELL_PROMPT "$"
```

5.42.2 Typedef Documentation

5.42.2.1 Job

```
typedef struct Job Job
```

struct for job

5.42.2.2 pid_t

```
typedef int pid_t
```

5.42.3 Enumeration Type Documentation

5.42.3.1 JobStatus

```
enum JobStatus
```

enum for job status

Enumerator

JOB_STATUS_RUNNING	
JOB_STATUS_STOPPED	
JOB_STATUS_DONE	

5.42.4 Function Documentation

5.42.4.1 add_job()

```
void add_job (  
    pid_t pid,  
    struct parsed_command * cmd,  
    JobStatus status)
```

Adds a new job to the job list.

Parameters

<i>pid</i>	Process ID of the new job.
<i>cmd</i>	Parsed command structure for the job.
<i>status</i>	Initial status of the job.

5.42.4.2 find_job_by_id()

```
Job * find_job_by_id (  
    int job_id)
```

Finds a job by its job ID.

Parameters

<i>job↔ _id</i>	Job ID to search for.
---------------------	-----------------------

Returns

Job* Pointer to the found job, or NULL if not found.

5.42.4.3 find_job_by_pid()

```
Job * find_job_by_pid (  
    pid_t pid)
```

Finds a job by its process ID.

Parameters

<i>pid</i>	Process ID to search for.
------------	---------------------------

Returns

Job* Pointer to the found job, or NULL if not found.

5.42.4.4 find_last_job()

```
Job * find_last_job ()
```

Finds the most recently added job.

Returns

Job* Pointer to the last job, or NULL if no jobs exist.

5.42.4.5 find_last_stopped_job()

```
Job * find_last_stopped_job ()
```

Finds the most recently stopped job.

Returns

Job* Pointer to the last stopped job, or NULL if no stopped jobs exist.

5.42.4.6 free_job()

```
void free_job (  
    void * data)
```

Frees memory associated with a job.

Parameters

<i>data</i>	Pointer to the job to be freed.
-------------	---------------------------------

5.42.4.7 initialize_job_control()

```
void initialize_job_control ()
```

Initializes job control mechanisms.

5.42.4.8 reconstruct_command()

```
char * reconstruct_command (  
    char * argv[])
```

Reconstructs a command string from an argument vector.

Parameters

<i>argv</i>	Array of command arguments.
-------------	-----------------------------

Returns

char* Reconstructed command string.

5.42.4.9 remove_job_by_pid()

```
void remove_job_by_pid (  
    pid_t pid)
```

Removes a job from the job list by its process ID.

Parameters

<i>pid</i>	Process ID of the job to remove.
------------	----------------------------------

5.42.4.10 shell()

```
void * shell (  
    void * args)
```

Main shell function to be run as a thread.

Parameters

<i>args</i>	Arguments passed to the shell thread.
-------------	---------------------------------------

Returns

void* Return value of the thread.

5.42.5 Variable Documentation**5.42.5.1 job_list**

```
Deque* job_list [extern]
```

Global job list.

5.42.5.2 next_job_id

```
int next_job_id [extern]
```

Next available job ID.

5.42.5.3 shell_pid

```
pid_t shell_pid [extern]
```

Process ID of the shell.

5.42.5.4 terminal_controller_pid

```
pid_t terminal_controller_pid [extern]
```

Process ID of the terminal controller.

5.43 penn-shell.h

[Go to the documentation of this file.](#)

```

00001 #include "../util/deque.h" // Include deque if using it for job list
00002
00003 #ifndef PENSHELL_H
00004 #define PENSHELL_H
00005 // #include "../util/types/process-status.h"
00006 #ifndef MAX_MESSAGE_SIZE
00007 #define MAX_MESSAGE_SIZE 4096
00008 #endif
00009
00010 #ifndef SHELL_PROMPT
00011 #define SHELL_PROMPT "$"
00012 #endif
00013
00014 #ifndef pid_t
00015 typedef int pid_t;
00016 #endif
00017
00018 #ifndef JobStatus
00019
00024 typedef enum
00025 {
00026     JOB_STATUS_RUNNING,
00027     JOB_STATUS_STOPPED,
00028     JOB_STATUS_DONE // Transient state before removal
00029 } JobStatus;
00030 #endif
00031
00032 #ifndef Job
00033
00038 typedef struct Job
00039 {
00040     int job_id;
00041     pid_t pid;
00042     char *command; // Store a copy of the command line
00043     JobStatus status;
00044     struct parsed_command *pcmd; // Store the parsed command for potential restart/display
00045 } Job;
00046 #endif
00047
00053 void *shell(void *args);
00054
00059 void free_job(void *data);
00060
00064 void initialize_job_control();
00065
00071 Job *find_job_by_pid(pid_t pid);
00072
00078 Job *find_job_by_id(int job_id);
00079
00084 Job *find_last_job();
00085
00090 Job *find_last_stopped_job();
00091
00098 void add_job(pid_t pid, struct parsed_command *cmd, JobStatus status);
00099
00104 void remove_job_by_pid(pid_t pid);
00105
00111 char *reconstruct_command(char *argv[]);
00112
00114 extern Deque *job_list;
00115
00117 extern int next_job_id;
00118
00120 extern pid_t shell_pid;
00121
00123 extern pid_t terminal_controller_pid;
00124
00125 #endif

```

5.44 src/penn-shell/stress.c File Reference

```

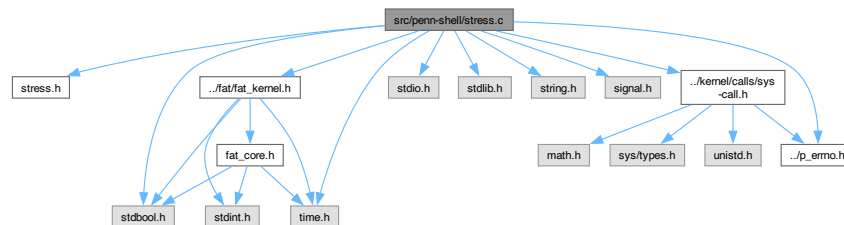
#include "stress.h"
#include <stdbool.h>
#include <stdio.h>

```



```
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include "../kernel/calls/sys-call.h"
#include "../kernel/p_errno.h"
#include "../fat/fat_kernel.h"
```

Include dependency graph for stress.c:



Functions

- void * [u_hang](#) (void *arg)
Test command that spawns 10 child processes and waits on them (blocking). Useful for testing the scheduler and wait functionality.
- void * [u_nohang](#) (void *arg)
Test command that spawns 10 child processes and waits on them (non-blocking). Useful for testing the scheduler with non-blocking waits.
- void * [u_recur](#) (void *arg)
Test command that recursively spawns 26 processes (Gen_A through Gen_Z). Useful for testing recursive spawning and deep process hierarchies.
- void * [u_crash](#) (void *arg)
Test command that writes a large pattern to a file and then crashes. Useful for testing file system durability and crash recovery. Requires the filesystem to be able to hold at least 5480 bytes in a file.

5.44.1 Function Documentation

5.44.1.1 u_crash()

```
void * u_crash (
    void * arg)
```

Test command that writes a large pattern to a file and then crashes. Useful for testing file system durability and crash recovery. Requires the filesystem to be able to hold at least 5480 bytes in a file.

Example Usage: crash

5.44.1.2 u_hang()

```
void * u_hang (
    void * arg)
```

Test command that spawns 10 child processes and waits on them (blocking). Useful for testing the scheduler and wait functionality.

Example Usage: hang

5.44.1.3 u_nohang()

```
void * u_nohang (
    void * arg)
```

Test command that spawns 10 child processes and waits on them (non-blocking). Useful for testing the scheduler with non-blocking waits.

Example Usage: nohang

5.44.1.4 u_recur()

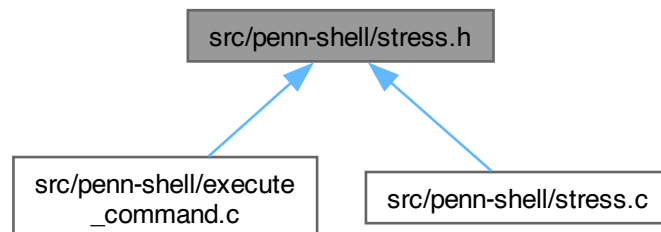
```
void * u_recur (
    void * arg)
```

Test command that recursively spawns 26 processes (Gen_A through Gen_Z). Useful for testing recursive spawning and deep process hierarchies.

Example Usage: recur

5.45 src/penn-shell/stress.h File Reference

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



Functions

- void * [u_hang](#) (void *)
Test command that spawns 10 child processes and waits on them (blocking). Useful for testing the scheduler and wait functionality.
- void * [u_nohang](#) (void *)
Test command that spawns 10 child processes and waits on them (non-blocking). Useful for testing the scheduler with non-blocking waits.
- void * [u_recur](#) (void *)
Test command that recursively spawns 26 processes (Gen_A through Gen_Z). Useful for testing recursive spawning and deep process hierarchies.
- void * [u_crash](#) (void *)
Test command that writes a large pattern to a file and then crashes. Useful for testing file system durability and crash recovery. Requires the filesystem to be able to hold at least 5480 bytes in a file.

5.45.1 Function Documentation

5.45.1.1 u_crash()

```
void * u_crash (  
    void * arg)
```

Test command that writes a large pattern to a file and then crashes. Useful for testing file system durability and crash recovery. Requires the filesystem to be able to hold at least 5480 bytes in a file.

Example Usage: crash

5.45.1.2 u_hang()

```
void * u_hang (  
    void * arg)
```

Test command that spawns 10 child processes and waits on them (blocking). Useful for testing the scheduler and wait functionality.

Example Usage: hang

5.45.1.3 u_nohang()

```
void * u_nohang (  
    void * arg)
```

Test command that spawns 10 child processes and waits on them (non-blocking). Useful for testing the scheduler with non-blocking waits.

Example Usage: nohang

5.45.1.4 u_recur()

```
void * u_recur (  
    void * arg)
```

Test command that recursively spawns 26 processes (Gen_A through Gen_Z). Useful for testing recursive spawning and deep process hierarchies.

Example Usage: recur

5.46 stress.h

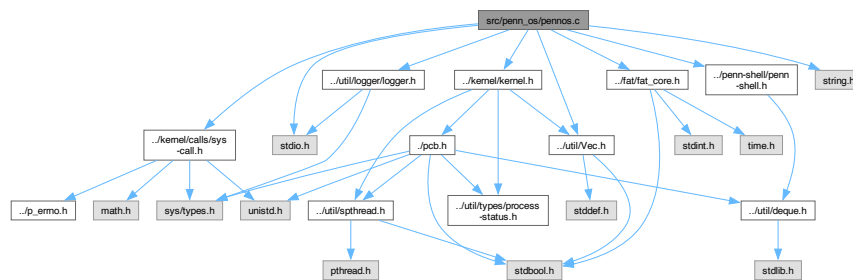
[Go to the documentation of this file.](#)

```
00001 #ifndef STRESS_H_  
00002 #define STRESS_H_  
00003  
00004 void* u_hang(void*);  
00005 void* u_nohang(void*);  
00006 void* u_recur(void*);  
00007  
00008 // this one requires the fs to hold at least 5480 bytes for a file.  
00009 void* u_crash(void*);  
00010  
00011 #endif
```

5.47 src/penn_os/pennos.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "../fat/fat_core.h"
#include "../kernel/calls/sys-call.h"
#include "../kernel/kernel.h"
#include "../penn-shell/penn-shell.h"
#include "../util/Vec.h"
#include "../util/logger/logger.h"
```

Include dependency graph for pennos.c:



Functions

- void * [torta](#) (void *args)
- int [main](#) (int argc, char *argv[])

5.47.1 Function Documentation

5.47.1.1 main()

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

5.47.1.2 torta()

```
void * torta (
    void * args)
```

5.48 src/penn_os/pennos.h File Reference

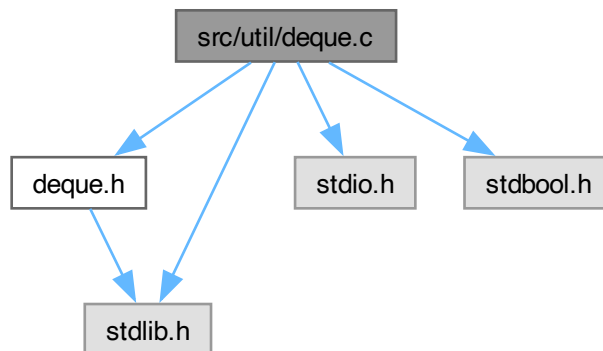
5.49 pennos.h

[Go to the documentation of this file.](#)

```
00001 #ifndef PENNOS_H
00002 #define PENNOS_H
00003
00004 #endif
```

5.50 src/util/deque.c File Reference

```
#include "deque.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
Include dependency graph for deque.c:
```



Functions

- `Deque * deque_new (void(*func)(void *))`
Create a new deque.
- `int deque_size (Deque *q)`
Get the number of elements in the deque.
- `void deque_push_front (Deque *q, void *value)`
Insert an element at the front of the deque.
- `void deque_push_back (Deque *q, void *value)`
Insert an element at the back of the deque.
- `void * deque_get_nth_elem (Deque *q, int n)`
Get the data at the *n*th position in the deque (0-based index).
- `void * deque_remove_nth_elem (Deque *q, int n)`
Remove and return the data at the *n*th position in the deque (0-based index).
- `void * deque_get_front (Deque *q)`
Get the data at the front of the deque without removing it.
- `void * deque_pop_front (Deque *q)`
Remove and return the data at the front of the deque.
- `void * deque_get_back (Deque *q)`
Get the data at the back of the deque without removing it.
- `void * deque_pop_back (Deque *q)`
Remove and return the data at the back of the deque.
- `void clear_deque (Deque *q)`
Remove all elements from the deque and free their memory using `delete_mem`.
- `void * deque_remove_specific (Deque *q, void *data)`
Remove and return the first occurrence of a specific value from the deque.
- `bool deque_contains (Deque *q, void *data)`
Check if the deque contains a specific value.

5.50.1 Function Documentation

5.50.1.1 clear_deque()

```
void clear_deque (  
    Deque * q)
```

Remove all elements from the deque and free their memory using delete_mem.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

5.50.1.2 deque_contains()

```
bool deque_contains (  
    Deque * q,  
    void * value)
```

Check if the deque contains a specific value.

Parameters

<i>q</i>	Pointer to the Deque .
<i>value</i>	Pointer to the data to search for.

Returns

true if value is found, false otherwise or if q is NULL.

5.50.1.3 deque_get_back()

```
void * deque_get_back (  
    Deque * q)
```

Get the data at the back of the deque without removing it.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

Returns

Pointer to the data at the back, or NULL if deque is empty or q is NULL.

5.50.1.4 deque_get_front()

```
void * deque_get_front (  
    Deque * q)
```

Get the data at the front of the deque without removing it.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

Returns

Pointer to the data at the front, or NULL if deque is empty or *q* is NULL.

5.50.1.5 deque_get_nth_elem()

```
void * deque_get_nth_elem (  
    Deque * q,  
    int n)
```

Get the data at the *n*th position in the deque (0-based index).

Parameters

<i>q</i>	Pointer to the Deque .
<i>n</i>	Index of the element to retrieve.

Returns

Pointer to the data at position *n*, or NULL if out of bounds or *q* is NULL.

5.50.1.6 deque_new()

```
Deque * deque_new (  
    void(* func ) (void *))
```

Create a new deque.

Parameters

<i>func</i>	Function pointer to free memory for stored data (can be NULL).
-------------	--

Returns

Pointer to the created [Deque](#), or NULL on allocation failure.

5.50.1.7 deque_pop_back()

```
void * deque_pop_back (  
    Deque * q)
```

Remove and return the data at the back of the deque.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

Returns

Pointer to the removed data, or NULL if deque is empty or *q* is NULL.

5.50.1.8 deque_pop_front()

```
void * deque_pop_front (  
    Deque * q)
```

Remove and return the data at the front of the deque.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

Returns

Pointer to the removed data, or NULL if deque is empty or *q* is NULL.

5.50.1.9 deque_push_back()

```
void deque_push_back (  
    Deque * q,  
    void * value)
```

Insert an element at the back of the deque.

Parameters

<i>q</i>	Pointer to the Deque .
<i>value</i>	Pointer to the data to insert.

5.50.1.10 deque_push_front()

```
void deque_push_front (  
    Deque * q,  
    void * value)
```

Insert an element at the front of the deque.

Parameters

<i>q</i>	Pointer to the Deque .
<i>value</i>	Pointer to the data to insert.

5.50.1.11 deque_remove_nth_elem()

```
void * deque_remove_nth_elem (  
    Deque * q,  
    int n)
```

Remove and return the data at the *n*th position in the deque (0-based index).

Parameters

<i>q</i>	Pointer to the Deque .
<i>n</i>	Index of the element to remove.

Returns

Pointer to the removed data, or NULL if out of bounds or *q* is NULL.

5.50.1.12 deque_remove_specific()

```
void * deque_remove_specific (  
    Deque * q,  
    void * value)
```

Remove and return the first occurrence of a specific value from the deque.

Parameters

<i>q</i>	Pointer to the Deque .
<i>value</i>	Pointer to the data to remove.

Returns

Pointer to the removed data, or NULL if not found or *q* is NULL.

5.50.1.13 deque_size()

```
int deque_size (  
    Deque * q)
```

Get the number of elements in the deque.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

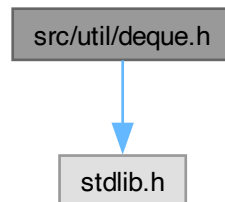
Returns

Number of elements in the deque, or 0 if *q* is NULL.

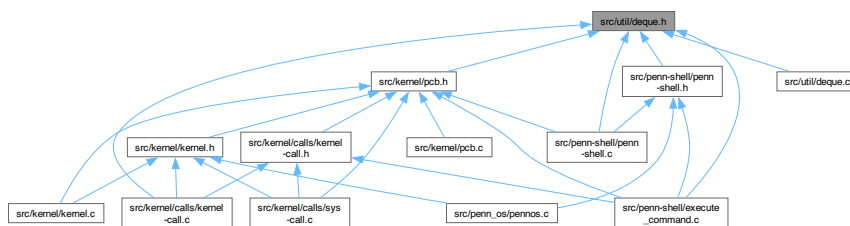
5.51 src/util/deque.h File Reference

```
#include <stdlib.h>
```

Include dependency graph for deque.h:



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



Classes

- struct [Node](#)
[Node](#) structure for doubly-linked list used in [Deque](#).
- struct [Deque](#)
Double-ended queue (deque) structure.

Macros

- `#define` [DEQUE_H](#)

Typedefs

- typedef struct Node [Node](#)
[Node](#) structure for doubly-linked list used in [Deque](#).
- typedef struct Deque [Deque](#)
Double-ended queue (deque) structure.

Functions

- [Deque](#) * [deque_new](#) (void(*func)(void *))
Create a new deque.
- int [deque_size](#) ([Deque](#) *q)
Get the number of elements in the deque.
- void [deque_push_front](#) ([Deque](#) *q, void *value)
Insert an element at the front of the deque.
- void [deque_push_back](#) ([Deque](#) *q, void *value)
Insert an element at the back of the deque.
- void * [deque_get_front](#) ([Deque](#) *q)
Get the data at the front of the deque without removing it.
- void * [deque_pop_front](#) ([Deque](#) *q)
Remove and return the data at the front of the deque.
- void * [deque_get_back](#) ([Deque](#) *q)
Get the data at the back of the deque without removing it.
- void * [deque_pop_back](#) ([Deque](#) *q)
Remove and return the data at the back of the deque.
- void [clear_deque](#) ([Deque](#) *q)
Remove all elements from the deque and free their memory using delete_mem.
- void * [deque_get_nth_elem](#) ([Deque](#) *q, int n)
Get the data at the nth position in the deque (0-based index).
- void * [deque_remove_nth_elem](#) ([Deque](#) *q, int n)
Remove and return the data at the nth position in the deque (0-based index).
- void * [deque_remove_specific](#) ([Deque](#) *q, void *value)
Remove and return the first occurrence of a specific value from the deque.
- bool [deque_contains](#) ([Deque](#) *q, void *value)
Check if the deque contains a specific value.

5.51.1 Macro Definition Documentation

5.51.1.1 DEQUE_H

```
#define DEQUE_H
```

5.51.2 Typedef Documentation

5.51.2.1 Deque

```
typedef struct Deque Deque
```

Double-ended queue (deque) structure.

The [Deque](#) supports insertion and removal of elements from both ends. It uses a doubly-linked list of [Node](#) structures. The delete_mem function pointer is used to free memory for stored data.

5.51.2.2 Node

```
typedef struct Node Node
```

[Node](#) structure for doubly-linked list used in [Deque](#).

This structure represents a node in the deque, holding a pointer to data and pointers to the next and previous nodes in the list.

5.51.3 Function Documentation

5.51.3.1 clear_deque()

```
void clear_deque (  
    Deque * q)
```

Remove all elements from the deque and free their memory using delete_mem.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

5.51.3.2 deque_contains()

```
bool deque_contains (  
    Deque * q,  
    void * value)
```

Check if the deque contains a specific value.

Parameters

<i>q</i>	Pointer to the Deque .
<i>value</i>	Pointer to the data to search for.

Returns

true if value is found, false otherwise or if q is NULL.

5.51.3.3 deque_get_back()

```
void * deque_get_back (  
    Deque * q)
```

Get the data at the back of the deque without removing it.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

Returns

Pointer to the data at the back, or NULL if deque is empty or *q* is NULL.

5.51.3.4 deque_get_front()

```
void * deque_get_front (  
    Deque * q)
```

Get the data at the front of the deque without removing it.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

Returns

Pointer to the data at the front, or NULL if deque is empty or *q* is NULL.

5.51.3.5 deque_get_nth_elem()

```
void * deque_get_nth_elem (  
    Deque * q,  
    int n)
```

Get the data at the *n*th position in the deque (0-based index).

Parameters

<i>q</i>	Pointer to the Deque .
<i>n</i>	Index of the element to retrieve.

Returns

Pointer to the data at position *n*, or NULL if out of bounds or *q* is NULL.

5.51.3.6 deque_new()

```
Deque * deque_new (  
    void(* func) (void *))
```

Create a new deque.

Parameters

<i>func</i>	Function pointer to free memory for stored data (can be NULL).
-------------	--

Returns

Pointer to the created [Deque](#), or NULL on allocation failure.

5.51.3.7 deque_pop_back()

```
void * deque_pop_back (  
    Deque * q)
```

Remove and return the data at the back of the deque.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

Returns

Pointer to the removed data, or NULL if deque is empty or q is NULL.

5.51.3.8 deque_pop_front()

```
void * deque_pop_front (  
    Deque * q)
```

Remove and return the data at the front of the deque.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

Returns

Pointer to the removed data, or NULL if deque is empty or q is NULL.

5.51.3.9 deque_push_back()

```
void deque_push_back (  
    Deque * q,  
    void * value)
```

Insert an element at the back of the deque.

Parameters

<i>q</i>	Pointer to the Deque .
<i>value</i>	Pointer to the data to insert.

5.51.3.10 deque_push_front()

```
void deque_push_front (  
    Deque * q,  
    void * value)
```

Insert an element at the front of the deque.

Parameters

<i>q</i>	Pointer to the Deque .
<i>value</i>	Pointer to the data to insert.

5.51.3.11 deque_remove_nth_elem()

```
void * deque_remove_nth_elem (  
    Deque * q,  
    int n)
```

Remove and return the data at the nth position in the deque (0-based index).

Parameters

<i>q</i>	Pointer to the Deque .
<i>n</i>	Index of the element to remove.

Returns

Pointer to the removed data, or NULL if out of bounds or q is NULL.

5.51.3.12 deque_remove_specific()

```
void * deque_remove_specific (  
    Deque * q,  
    void * value)
```

Remove and return the first occurrence of a specific value from the deque.

Parameters

<i>q</i>	Pointer to the Deque .
<i>value</i>	Pointer to the data to remove.

Returns

Pointer to the removed data, or NULL if not found or q is NULL.

5.51.3.13 deque_size()

```
int deque_size (
    Deque * q)
```

Get the number of elements in the deque.

Parameters

<i>q</i>	Pointer to the Deque .
----------	--

Returns

Number of elements in the deque, or 0 if q is NULL.

5.52 deque.h

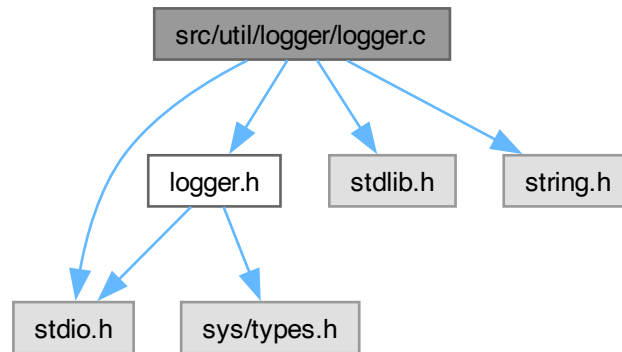
[Go to the documentation of this file.](#)

```
00001 #include <stdlib.h>
00002 // Node structure (reused for all types)
00003 #ifndef NODE_H
00004 #define NODE_H
00011 typedef struct Node {
00012     void *data;
00013     struct Node *next;
00014     struct Node *prev;
00015 } Node;
00016 #endif
00017
00018 #ifndef DEQUE_H
00019 #define DEQUE_H
00020 // Deque structure
00028 typedef struct Deque {
00029     Node *front;
00030     Node *tail;
00031     int size;
00032     void (*delete_mem)(void *);
00033 } Deque;
00034 #endif
00041 Deque *deque_new(void (*func)(void *));
00042
00049 int deque_size(Deque *q);
00050
00057 void deque_push_front(Deque *q, void *value);
00058
00065 void deque_push_back(Deque *q, void *value);
00066
00073 void *deque_get_front(Deque *q);
00074
00081 void *deque_pop_front(Deque *q);
00082
00089 void *deque_get_back(Deque *q);
00090
00097 void *deque_pop_back(Deque *q);
00098
00104 void clear_deque(Deque *q);
00105
00113 void *deque_get_nth_elem(Deque *q, int n);
00114
00122 void *deque_remove_nth_elem(Deque *q, int n);
00123
00131 void *deque_remove_specific(Deque *q, void *value);
00132
00140 bool deque_contains(Deque *q, void *value);
```


5.53 src/util/logger/logger.c File Reference

```
#include "logger.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for logger.c:



Functions

- void `init_logger` (const char *log_filename)
Initializes the logger.
- void `log_event` (unsigned long tick, const char *event, `pid_t` pid, int priority, const char *name)
Logs an event.

5.53.1 Function Documentation

5.53.1.1 init_logger()

```
void init_logger (
    const char * log_filename)
```

Initializes the logger.

Parameters

<code>log_filename</code>	The name of the log file.
---------------------------	---------------------------

5.53.1.2 log_event()

```
void log_event (
    unsigned long tick,
    const char * event,
    pid_t pid,
    int priority,
    const char * name)
```

Logs an event.

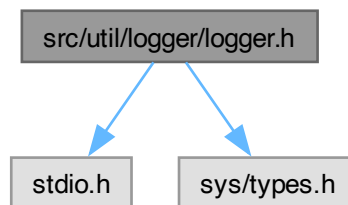
Parameters

<i>tick</i>	The tick count.
<i>event</i>	The event description.
<i>pid</i>	The process ID.
<i>priority</i>	The priority level.
<i>name</i>	The process name.

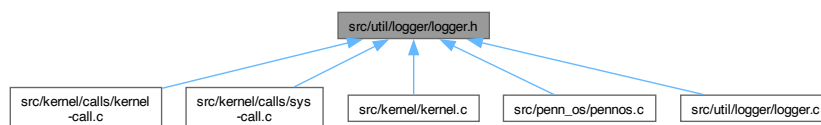
5.54 src/util/logger/logger.h File Reference

```
#include <stdio.h>
#include <sys/types.h>
```

Include dependency graph for logger.h:



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



Functions

- void `init_logger` (const char *log_filename)
Initializes the logger.
- void `log_event` (unsigned long tick, const char *event, `pid_t` pid, int priority, const char *name)
Logs an event.

5.54.1 Function Documentation

5.54.1.1 `init_logger()`

```
void init_logger (  
    const char * log_filename)
```

Initializes the logger.

Parameters

<i>log_filename</i>	The name of the log file.
---------------------	---------------------------

5.54.1.2 log_event()

```
void log_event (
    unsigned long tick,
    const char * event,
    pid_t pid,
    int priority,
    const char * name)
```

Logs an event.

Parameters

<i>tick</i>	The tick count.
<i>event</i>	The event description.
<i>pid</i>	The process ID.
<i>priority</i>	The priority level.
<i>name</i>	The process name.

5.55 logger.h

[Go to the documentation of this file.](#)

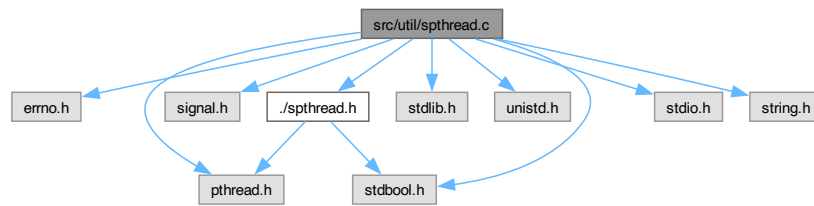
```
00001 #ifndef LOGGER_H
00002 #define LOGGER_H
00003
00004 #include <stdio.h>
00005 #include <sys/types.h>
00006
00012 void init_logger(const char *log_filename);
00013
00023 void log_event(unsigned long tick, const char *event,
00024               pid_t pid, int priority, const char *name);
00025
00026 #endif
```

5.56 src/util/spthread.c File Reference

```
#include <errno.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include " ./spthread.h"
#include <stdio.h>
```

```
#include <string.h>
```

Include dependency graph for `spthread.c`:



Classes

- struct `spthread_fwd_args_st`
- struct `spthread_signal_args_st`
- struct `spthread_meta_st`

Macros

- `#define _GNU_SOURCE`
- `#define MILLISEC_IN_NANO 100000`
- `#define SPTHREAD_RUNNING_STATE 0`
- `#define SPTHREAD_SUSPENDED_STATE 1`
- `#define SPTHREAD_TERMINATED_STATE 2`
- `#define SPTHREAD_SIG_SUSPEND -1`
- `#define SPTHREAD_SIG_CONTINUE -2`

Typedefs

- `typedef void (* pthread_fn) (void *)`
- `typedef struct spthread_fwd_args_st spthread_fwd_args`
- `typedef struct spthread_signal_args_st spthread_signal_args`
- `typedef struct spthread_meta_st spthread_meta_t`

Functions

- `int spthread_create (spthread_t *thread, const pthread_attr_t *attr, pthread_fn start_routine, void *arg)`
- `int spthread_suspend (spthread_t thread)`
- `int spthread_suspend_self ()`
- `int spthread_continue (spthread_t thread)`
- `int spthread_cancel (spthread_t thread)`
- `bool spthread_self (spthread_t *thread)`
- `int spthread_join (spthread_t thread, void **retval)`
- `void spthread_exit (void *status)`
- `bool spthread_equal (spthread_t first, spthread_t second)`
- `int spthread_disable_interrupts_self ()`
- `int spthread_enable_interrupts_self ()`

5.56.1 Macro Definition Documentation

5.56.1.1 `_GNU_SOURCE`

```
#define _GNU_SOURCE
```

5.56.1.2 `MILISEC_IN_NANO`

```
#define MILISEC_IN_NANO 100000
```

5.56.1.3 `SPTHREAD_RUNNING_STATE`

```
#define SPTHREAD_RUNNING_STATE 0
```

5.56.1.4 `SPTHREAD_SIG_CONTINUE`

```
#define SPTHREAD_SIG_CONTINUE -2
```

5.56.1.5 `SPTHREAD_SIG_SUSPEND`

```
#define SPTHREAD_SIG_SUSPEND -1
```

5.56.1.6 `SPTHREAD_SUSPENDED_STATE`

```
#define SPTHREAD_SUSPENDED_STATE 1
```

5.56.1.7 `SPTHREAD_TERMINATED_STATE`

```
#define SPTHREAD_TERMINATED_STATE 2
```

5.56.2 Typedef Documentation

5.56.2.1 `pthread_fn`

```
typedef void *(* pthread_fn) (void *)
```

5.56.2.2 `spthread_fwd_args`

```
typedef struct spthread\_fwd\_args\_st spthread\_fwd\_args
```

5.56.2.3 `spthread_meta_t`

```
typedef struct spthread_meta_st spthread_meta_t
```

5.56.2.4 `spthread_signal_args`

```
typedef struct spthread_signal_args_st spthread_signal_args
```

5.56.3 Function Documentation

5.56.3.1 `spthread_cancel()`

```
int spthread_cancel (  
    spthread_t thread)
```

5.56.3.2 `spthread_continue()`

```
int spthread_continue (  
    spthread_t thread)
```

5.56.3.3 `spthread_create()`

```
int spthread_create (  
    spthread_t * thread,  
    const pthread_attr_t * attr,  
    pthread_fn start_routine,  
    void * arg)
```

5.56.3.4 `spthread_disable_interrupts_self()`

```
int spthread_disable_interrupts_self ()
```

5.56.3.5 `spthread_enable_interrupts_self()`

```
int spthread_enable_interrupts_self ()
```

5.56.3.6 `spthread_equal()`

```
bool spthread_equal (  
    spthread_t first,  
    spthread_t second)
```


5.56.3.7 pthread_exit()

```
void pthread_exit (  
    void * status)
```

5.56.3.8 pthread_join()

```
int pthread_join (  
    pthread_t thread,  
    void ** retval)
```

5.56.3.9 pthread_self()

```
bool pthread_self (  
    pthread_t * thread)
```

5.56.3.10 pthread_suspend()

```
int pthread_suspend (  
    pthread_t thread)
```

5.56.3.11 pthread_suspend_self()

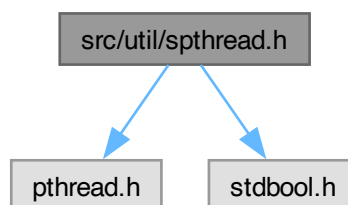
```
int pthread_suspend_self ()
```

5.57 src/util/spthread.h File Reference

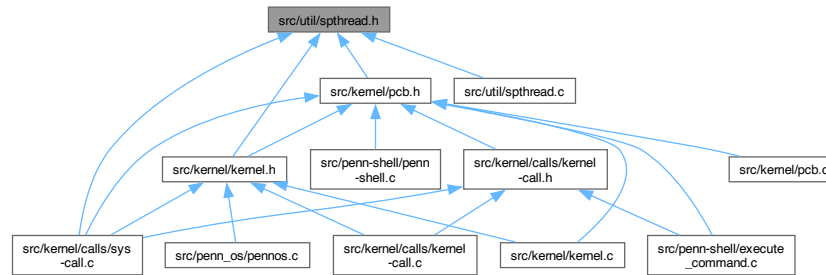
```
#include <pthread.h>
```

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

Include dependency graph for pthread.h:



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



Classes

- struct [spthread_st](#)

Macros

- #define [SIGPTHD](#) SIGUSR1

Typedefs

- typedef struct [spthread_st](#) [spthread_t](#)

Functions

- int [spthread_create](#) ([spthread_t](#) *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg)
- int [spthread_suspend](#) ([spthread_t](#) thread)
- int [spthread_suspend_self](#) ()
- int [spthread_continue](#) ([spthread_t](#) thread)
- int [spthread_cancel](#) ([spthread_t](#) thread)
- bool [spthread_self](#) ([spthread_t](#) *thread)
- int [spthread_join](#) ([spthread_t](#) thread, void **retval)
- void [spthread_exit](#) (void *status)
- bool [spthread_equal](#) ([spthread_t](#) first, [spthread_t](#) second)
- int [spthread_disable_interrupts_self](#) ()
- int [spthread_enable_interrupts_self](#) ()

5.57.1 Macro Definition Documentation

5.57.1.1 SIGPTHD

```
#define SIGPTHD SIGUSR1
```

5.57.2 Typedef Documentation

5.57.2.1 pthread_t

```
typedef struct pthread_st pthread_t
```

5.57.3 Function Documentation

5.57.3.1 pthread_cancel()

```
int pthread_cancel (  
    pthread_t thread)
```

5.57.3.2 pthread_continue()

```
int pthread_continue (  
    pthread_t thread)
```

5.57.3.3 pthread_create()

```
int pthread_create (  
    pthread_t * thread,  
    const pthread_attr_t * attr,  
    void *(* start_routine ) (void *),  
    void * arg)
```

5.57.3.4 pthread_disable_interrupts_self()

```
int pthread_disable_interrupts_self ()
```

5.57.3.5 pthread_enable_interrupts_self()

```
int pthread_enable_interrupts_self ()
```

5.57.3.6 pthread_equal()

```
bool pthread_equal (  
    pthread_t first,  
    pthread_t second)
```

5.57.3.7 pthread_exit()

```
void pthread_exit (  
    void * status)
```

5.57.3.8 `spthread_join()`

```
int pthread_join (
    pthread_t thread,
    void ** retval)
```

5.57.3.9 `spthread_self()`

```
bool pthread_self (
    pthread_t * thread)
```

5.57.3.10 `spthread_suspend()`

```
int pthread_suspend (
    pthread_t thread)
```

5.57.3.11 `spthread_suspend_self()`

```
int pthread_suspend_self ()
```

5.58 `spthread.h`

[Go to the documentation of this file.](#)

```
00001 #ifndef SPHTHREAD_H_
00002 #define SPHTHREAD_H_
00003
00004 #include <pthread.h>
00005 #include <stdbool.h>
00006
00007 // CAUTION: according to `man 7 pthread`:
00008 //
00009 //   On older Linux kernels, SIGUSR1 and SIGUSR2
00010 //   are used. Applications must avoid the use of whichever set of
00011 //   signals is employed by the implementation.
00012 //
00013 // This may not work on other linux versions
00014
00015 // SIGNAL PTHREAD
00016 // NOTE: if within a created pthread you change
00017 // the behaviour of SIGUSR1, then you will not be able
00018 // to suspend and continue a pthread
00019 #define SIGPTHD SIGUSR1
00020
00021 // declares a struct, but the internals of the
00022 // struct cannot be seen by functions outside of pthread.c
00023 typedef struct pthread_meta_st pthread_meta_t;
00024
00025 // The pthread wrapper struct.
00026 // Sometimes you may have to access the inner pthread member
00027 // but you shouldn't need to do that
00028 typedef struct pthread_st {
00029     pthread_t thread;
00030     pthread_meta_t* meta;
00031 } pthread_t;
00032
00033 // NOTE:
00034 // None of these are signal safe
00035 // Also note that most of these functions are not safe to suspension,
00036 // meaning that if the thread calling these is an pthread and is suspended
00037 // in the middle of pthread_continue or pthread_suspend, then it may not work.
00038 //
00039 // Make sure that the calling thread cannot be suspended before calling these
00040 // functions. Exceptions to this are pthread_exit(), pthread_self() and if a
```

```

00041 // thread is continuing or suspending itself.
00042
00043 // spthread_create:
00044 // this function works similar to pthread_create, except for two differences.
00045 // 1) the created pthread is able to be asynchronously suspended, and continued
00046 //    using the functions:
00047 //    - spthread_suspend
00048 //    - spthread_continue
00049 // 2) The created pthread will be suspended before it executes the specified
00050 //    routine. It must first be continued with `spthread_continue` before
00051 //    it will start executing.
00052 //
00053 // It is worth noting that this function is not signal safe.
00054 // In other words, it should not be called from a signal handler.
00055 //
00056 // to avoid repetition, see pthread_create(3) for details
00057 // on arguments and return values as they are the same here.
00058 int spthread_create(spthread_t* thread,
00059                    const pthread_attr_t* attr,
00060                    void* (*start_routine)(void*),
00061                    void* arg);
00062
00063 // The spthread_suspend function will signal to the
00064 // specified thread to suspend execution.
00065 //
00066 // Calling spthread_suspend on an already suspended
00067 // thread does not do anything.
00068 //
00069 // It is worth noting that this function is not signal safe.
00070 // In other words, it should not be called from a signal handler.
00071 //
00072 // args:
00073 // - pthread_t thread: the thread we want to suspend
00074 //   This thread must be created using the spthread_create() function,
00075 //   if created by some other function, the behaviour is undefined.
00076 //
00077 // returns:
00078 // - 0 on success
00079 // - EAGAIN if the thread could not be signaled
00080 // - ENOSYS if not supported on this system
00081 // - ESRCH if the thread specified is not a valid pthread
00082 int spthread_suspend(spthread_t thread);
00083
00084 // The spthread_suspend_self function will cause the calling
00085 // thread (which should be created by spthread_create) to suspend
00086 // itself.
00087 //
00088 // returns:
00089 // - 0 on success
00090 // - EAGAIN if the thread could not be signaled
00091 // - ENOSYS if not supported on this system
00092 // - ESRCH if the calling thread is not an spthread
00093 int spthread_suspend_self();
00094
00095 // The spthread_continue function will signal to the
00096 // specified thread to resume execution if suspended.
00097 //
00098 // Calling spthread_continue on an already non-suspended
00099 // thread does not do anything.
00100 //
00101 // It is worth noting that this function is not signal safe.
00102 // In other words, it should not be called from a signal handler.
00103 //
00104 // args:
00105 // - spthread_t thread: the thread we want to continue
00106 //   This thread must be created using the spthread_create() function,
00107 //   if created by some other function, the behaviour is undefined.
00108 //
00109 // returns:
00110 // - 0 on success
00111 // - EAGAIN if the thread could not be signaled
00112 // - ENOSYS if not supported on this system
00113 // - ESRCH if the thread specified is not a valid pthread
00114 int spthread_continue(spthread_t thread);
00115
00116 // The spthread_cancel function will send a
00117 // cancellation request to the specified thread.
00118 //
00119 // as of now, this function is identical to pthread_cancel(3)
00120 // so to avoid repetition, you should look there.
00121 //
00122 // Here are a few things that are worth highlighting:
00123 // - it is worth noting that it is a cancellation __request__
00124 //   the thread may not terminate immediately, instead the
00125 //   thread is checked whenever it calls a function that is
00126 //   marked as a cancellation point. At those points, it will
00127 //   start the cancellation procedure

```

```

00128 // - to make sure all things are de-allocated properly on
00129 //   normal exiting of the thread and when it is cancelled,
00130 //   you should mark a deferred de-allocation with
00131 //   pthread_cleanup_push(3).
00132 //   consider the following example:
00133 //
00134 //       void* thread_routine(void* arg) {
00135 //           int* num = malloc(sizeof(int));
00136 //           pthread_cleanup_push(&free, num);
00137 //           return NULL;
00138 //       }
00139 //
00140 //   this program will allocate an integer on the heap
00141 //   and mark that data to be de-allocated on cleanup.
00142 //   This means that when the thread returns from the
00143 //   routine specified in spthread_create, free will
00144 //   be called on num. This will also happen if the thread
00145 //   is cancelled and not able to be exited normally.
00146 //
00147 //   Another function that should be used in conjunction
00148 //   is pthread_cleanup_pop(3). I will leave that
00149 //   to you to read more on.
00150 //
00151 // It is worth noting that this function is not signal safe.
00152 // In other words, it should not be called from a signal handler.
00153 //
00154 // args:
00155 // - spthread_t thread: the thread we want to cancel.
00156 //   This thread must be created using the spthread_create() function,
00157 //   if created by some other function, the behaviour is undefined.
00158 //
00159 // returns:
00160 // - 0 on success
00161 // - ESRCH if the thread specified is not a valid pthread
00162 int spthread_cancel(spthread_t thread);
00163
00164 // Can be called by a thread to get two peices of information:
00165 // 1. Whether or not the calling thread is an spthread (true or false)
00166 // 2. The spthread_t of the calling thread, if it is an spthread_t
00167 //
00168 // almost always the function will be called like this:
00169 // spthread_t self;
00170 // bool i_am_spthread = spthread_self(&self);
00171 //
00172 // args:
00173 // - spthread_t* thread: the output parameter to get the spthread_t
00174 //   representing the calling thread, if it is an spthread
00175 //
00176 // returns:
00177 // - true if the calling thread is an spthread_t
00178 // - false otherwise.
00179 bool spthread_self(spthread_t* thread);
00180
00181 // The equivalent of pthread_join but for spthread
00182 // To make sure all resources are cleaned up appropriately
00183 // sptreads that are created must at some ppoint have spthread_join
00184 // called on them. Do not use pthread_join on an spthread.
00185 //
00186 // to avoid repetition, see pthread_join(3) for details
00187 // on arguments and return values as they are the same as this function.
00188 int spthread_join(spthread_t thread, void** retval);
00189
00190 // The equivalent of pthread_exit but for spthread
00191 // spthread_exit must be used by sptreads instead of pthread_exit.
00192 // Otherwise, calls to spthread_join or other functions (like spthread_suspend)
00193 // may not work as intended.
00194 //
00195 // to avoid repetition, see pthread_exit(3) for details
00196 // on arguments and return values as they are the same as this function.
00197 void spthread_exit(void* status);
00198
00199 // The equivalent of pthread_equal but for spthread.
00200 // It two spthread_t's describe the same thread, returns a
00201 // non-zero value; otherwise it returns 0.
00202 bool spthread_equal(spthread_t first, spthread_t second);
00203
00204 // Calling this function from an spthread prevents it from
00205 // being suspended until re-enabled by the sibling function
00206 // "spthread_enable_interrupts_self".
00207 //
00208 // This is done by blocking the SIG_PTHD signal
00209 //
00210 // returns 0 on success, or -1 on error
00211 int spthread_disable_interrupts_self();
00212
00213 // Calling this function from an spthread re-enables it to
00214 // being suspendable. Should be called after it's sibling function

```

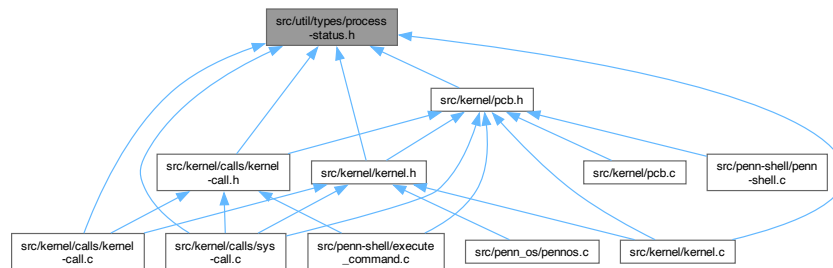
```

00215 // "spthread_disable_interrupts_self".
00216 //
00217 // This is done by unblocking the SIG_PTHD signal
00218 //
00219 // returns 0 on success, or -1 on error
00220 int spthread_enable_interrupts_self();
00221
00222 #endif // SPTHREAD_H_

```

5.59 src/util/types/process-status.h File Reference

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



Macros

- `#define pid_t int`

Enumerations

- `enum ProcessStatus {
PROCESS_STATUS_RUNNING , PROCESS_STATUS_WAITING , PROCESS_STATUS_ZOMBIE ,
PROCESS_STATUS_STOPPED ,
PROCESS_STATUS_BLOCKED , PROCESS_STATUS_DEAD }`

5.59.1 Macro Definition Documentation

5.59.1.1 pid_t

```
#define pid_t int
```

5.59.2 Enumeration Type Documentation

5.59.2.1 ProcessStatus

```
enum ProcessStatus
```

Enumerator

PROCESS_STATUS_RUNNING	
PROCESS_STATUS_WAITING	
PROCESS_STATUS_ZOMBIE	
PROCESS_STATUS_STOPPED	
PROCESS_STATUS_BLOCKED	
PROCESS_STATUS_DEAD	

5.60 process-status.h

[Go to the documentation of this file.](#)

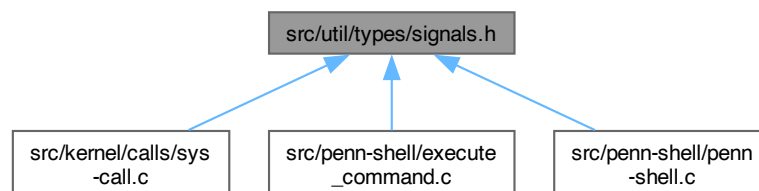
```

00001 // Process status enum
00002 // This enum is used to represent the status of a process in the system.
00003 // It is used in the PCB (Process Control Block) structure to indicate the
00004 // current state of the process.
00005 // The enum values are used to determine the scheduling and execution
00006 // behavior of the process
00007 #ifndef PROCESS_STATUS_H
00008 #define PROCESS_STATUS_H
00009 typedef enum {
00010     PROCESS_STATUS_RUNNING,
00011     PROCESS_STATUS_WAITING,
00012     PROCESS_STATUS_ZOMBIE,
00013     PROCESS_STATUS_STOPPED,
00014     PROCESS_STATUS_BLOCKED,
00015     PROCESS_STATUS_DEAD
00016 } ProcessStatus;
00017 #endif
00018
00019 #ifndef pid_t
00020 #define pid_t int
00021 #endif

```

5.61 src/util/types/signals.h File Reference

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



Macros

- `#define P_SIGTERM 1`
- `#define P_SIGSTOP 2`
- `#define P_SIGCONT 3`

5.61.1 Macro Definition Documentation

5.61.1.1 P_SIGCONT

```
#define P_SIGCONT 3
```

5.61.1.2 P_SIGSTOP

```
#define P_SIGSTOP 2
```

5.61.1.3 P_SIGTERM

```
#define P_SIGTERM 1
```

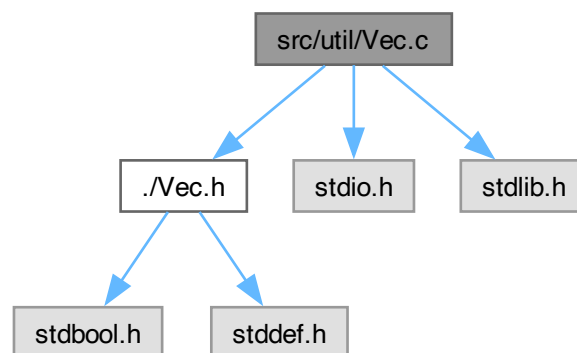
5.62 signals.h

[Go to the documentation of this file.](#)

```
00001 #ifndef PENNOS_SIGNALS_H
00002 #define PENNOS_SIGNALS_H
00003
00004 #define P_SIGTERM 1
00005 #define P_SIGSTOP 2
00006 #define P_SIGCONT 3
00007
00008 #endif
```

5.63 src/util/Vec.c File Reference

```
#include "../Vec.h"
#include <stdio.h>
#include <stdlib.h>
Include dependency graph for Vec.c:
```



Functions

- `Vec vec_new` (`size_t initial_capacity`, `ptr_dtor_fn ele_dtor_fn`)
- `ptr_t vec_get` (`Vec *self`, `size_t index`)
- `void vec_set` (`Vec *self`, `size_t index`, `ptr_t new_ele`)
- `void vec_push_back` (`Vec *self`, `ptr_t new_ele`)
- `bool vec_pop_back` (`Vec *self`)
- `void vec_insert` (`Vec *self`, `size_t index`, `ptr_t new_ele`)
- `void vec_erase` (`Vec *self`, `size_t index`)
- `void vec_resize` (`Vec *self`, `size_t new_capacity`)
- `void vec_clear` (`Vec *self`)
- `void vec_destroy` (`Vec *self`)

5.63.1 Function Documentation

5.63.1.1 `vec_clear()`

```
void vec_clear (  
    Vec * self)
```

5.63.1.2 `vec_destroy()`

```
void vec_destroy (  
    Vec * self)
```

5.63.1.3 `vec_erase()`

```
void vec_erase (  
    Vec * self,  
    size_t index)
```

5.63.1.4 `vec_get()`

```
ptr_t vec_get (  
    Vec * self,  
    size_t index)
```

5.63.1.5 `vec_insert()`

```
void vec_insert (  
    Vec * self,  
    size_t index,  
    ptr_t new_ele)
```

5.63.1.6 `vec_new()`

```
Vec vec_new (  
    size_t initial_capacity,  
    ptr_dtor_fn ele_dtor_fn)
```

Creates a new empty `Vec(tor)` with the specified `initial_capacity` and specified function to clean up elements in the vector.

Parameters

<i>initial_capacity</i>	the initial capacity of the newly created vector, non negative
<i>ele_dtor_fn</i>	a function pointer to a function that takes in a ptr_t (a vector element) and cleans it up. This is commonly just <code>free</code> but custom functions can be passed in. NULL can also be passed in to specify that there is no cleanup function that needs to be called on each element.

Returns

a newly created vector with specified capacity, 0 length and the specified element destructor (cleanup) function.

Postcondition

if memory allocation fails, the function will panic.

5.63.1.7 vec_pop_back()

```
bool vec_pop_back (  
    Vec * self)
```

5.63.1.8 vec_push_back()

```
void vec_push_back (  
    Vec * self,  
    ptr_t new_ele)
```

5.63.1.9 vec_resize()

```
void vec_resize (  
    Vec * self,  
    size_t new_capacity)
```

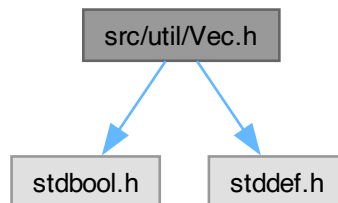
5.63.1.10 vec_set()

```
void vec_set (  
    Vec * self,  
    size_t index,  
    ptr_t new_ele)
```

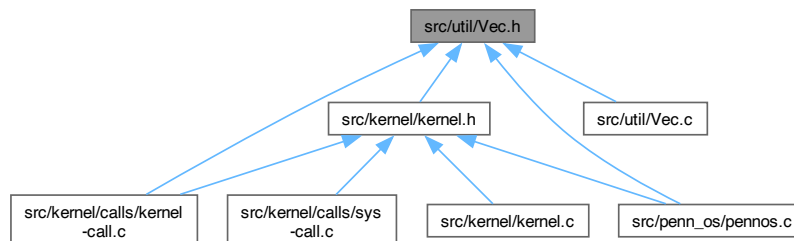
5.64 src/util/Vec.h File Reference

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

Include dependency graph for Vec.h:



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



Classes

- struct [vec_st](#)

Macros

- #define [vec_capacity](#)(vec)
- #define [vec_len](#)(vec)
- #define [vec_is_empty](#)(vec)

Typedefs

- typedef void * [ptr_t](#)
- typedef void(* [ptr_dtor_fn](#)) ([ptr_t](#))
- typedef struct [vec_st](#) [Vec](#)

Functions

- [Vec](#) [vec_new](#) (size_t initial_capacity, [ptr_dtor_fn](#) ele_dtor_fn)
- [ptr_t](#) [vec_get](#) ([Vec](#) *self, size_t index)
- void [vec_set](#) ([Vec](#) *self, size_t index, [ptr_t](#) new_ele)
- void [vec_push_back](#) ([Vec](#) *self, [ptr_t](#) new_ele)
- bool [vec_pop_back](#) ([Vec](#) *self)
- void [vec_insert](#) ([Vec](#) *self, size_t index, [ptr_t](#) new_ele)
- void [vec_erase](#) ([Vec](#) *self, size_t index)
- void [vec_resize](#) ([Vec](#) *self, size_t new_capacity)
- void [vec_clear](#) ([Vec](#) *self)
- void [vec_destroy](#) ([Vec](#) *self)

5.64.1 Macro Definition Documentation

5.64.1.1 [vec_capacity](#)

```
#define vec_capacity(  
    vec)
```

Value:

```
((vec)->capacity)
```

5.64.1.2 [vec_is_empty](#)

```
#define vec_is_empty(  
    vec)
```

Value:

```
((vec)->length == 0)
```

5.64.1.3 [vec_len](#)

```
#define vec_len(  
    vec)
```

Value:

```
((vec)->length)
```

5.64.2 Typedef Documentation

5.64.2.1 [ptr_dtor_fn](#)

```
typedef void(* ptr_dtor_fn) (ptr\_t)
```

5.64.2.2 [ptr_t](#)

```
typedef void* ptr\_t
```

5.64.2.3 Vec

```
typedef struct vec_st Vec
```

5.64.3 Function Documentation

5.64.3.1 vec_clear()

```
void vec_clear (  
    Vec * self)
```

5.64.3.2 vec_destroy()

```
void vec_destroy (  
    Vec * self)
```

5.64.3.3 vec_erase()

```
void vec_erase (  
    Vec * self,  
    size_t index)
```

5.64.3.4 vec_get()

```
ptr_t vec_get (  
    Vec * self,  
    size_t index)
```

5.64.3.5 vec_insert()

```
void vec_insert (  
    Vec * self,  
    size_t index,  
    ptr_t new_ele)
```

5.64.3.6 vec_new()

```
Vec vec_new (  
    size_t initial_capacity,  
    ptr_dtor_fn ele_dtor_fn)
```

Creates a new empty [Vec\(tor\)](#) with the specified `initial_capacity` and specified function to clean up elements in the vector.

Parameters

<i>initial_capacity</i>	the initial capacity of the newly created vector, non negative
<i>ele_dtor_fn</i>	a function pointer to a function that takes in a ptr_t (a vector element) and cleans it up. This is commonly just <code>free</code> but custom functions can be passed in. NULL can also be passed in to specify that there is no cleanup function that needs to be called on each element.

Returns

a newly created vector with specified capacity, 0 length and the specified element destructor (cleanup) function.

Postcondition

if memory allocation fails, the function will panic.

5.64.3.7 vec_pop_back()

```
bool vec_pop_back (  
    Vec * self)
```

5.64.3.8 vec_push_back()

```
void vec_push_back (  
    Vec * self,  
    ptr_t new_ele)
```

5.64.3.9 vec_resize()

```
void vec_resize (  
    Vec * self,  
    size_t new_capacity)
```

5.64.3.10 vec_set()

```
void vec_set (  
    Vec * self,  
    size_t index,  
    ptr_t new_ele)
```

5.65 Vec.h

[Go to the documentation of this file.](#)

```

00001 #ifndef VEC_H_
00002 #define VEC_H_
00003
00004 #include <stdbool.h>
00005 #include <stddef.h> // for size_t
00006
00007 typedef void* ptr_t;
00008 typedef void (*ptr_dtor_fn)(ptr_t);
00009
00010 typedef struct vec_st {
00011     ptr_t* data;
00012     size_t length;
00013     size_t capacity;
00014     ptr_dtor_fn ele_dtor_fn;
00015 } Vec;
00016
00033 Vec vec_new(size_t initial_capacity, ptr_dtor_fn ele_dtor_fn);
00034
00035 /* Returns the current capacity of the Vec
00036  * Written as a function-like macro
00037  *
00038  * @param vec, a pointer to the vector we want to grab the capacity of.
00039  */
00040 // TODO: finish this macro
00041 #define vec_capacity(vec) ((vec)->capacity)
00042
00043 /* Returns the current length of the Vec
00044  * written as a function-like macro
00045  *
00046  * @param vec, a pointer to the vector we want to grab the len of.
00047  */
00048 // TODO: finish this macro
00049 #define vec_len(vec) ((vec)->length)
00050 /* Checks if the Vec is empty
00051  * written as a function-like macro
00052  *
00053  * @param vec, a pointer to the vector we want to check emptiness of.
00054  */
00055 // TODO: finish this macro
00056 #define vec_is_empty(vec) ((vec)->length == 0)
00057
00058 /* Gets the specified element of the Vec
00059  *
00060  * @param self a pointer to the vector who's element we want to get.
00061  * @param index the index of the element to get.
00062  * @returns the element at the specified index.
00063  * @pre Assumes self points to a valid vector. If the index is >= self->length
00064  * then this function will panic()
00065  */
00066 ptr_t vec_get(Vec* self, size_t index);
00067
00068 /* Sets the specified element of the Vec to the specified value
00069  *
00070  * @param self a pointer to the vector who's element we want to set.
00071  * @param index the index of the element to set.
00072  * @param new_ele the value we want to set the element at that index to
00073  * @returns the element at the specified index.
00074  * @pre Assumes self points to a valid vector. If the index is >= self->length
00075  * then this function will panic()
00076  */
00077 void vec_set(Vec* self, size_t index, ptr_t new_ele);
00078
00079 /* Appends the given element to the end of the Vec
00080  *
00081  * @param self a pointer to the vector we are pushing onto
00082  * @param new_ele the value we want to add to the end of the container
00083  * @pre Assumes self points to a valid vector.
00084  * @post If a resize is needed and it fails, then this function will panic()
00085  * @post If after the operation the new length is greater than the old capacity
00086  * then a reallocation takes place and all elements are copied over.
00087  * Capacity is doubled. If initial capacity is zero, it is resized to
00088  * capacity 1. Any pointers to elements prior to this reallocation are
00089  * invalidated.
00090  */
00091 void vec_push_back(Vec* self, ptr_t new_ele);
00092
00093 /* Removes and destroys the last element of the Vec
00094  *
00095  * @param self a pointer to the vector we are popping.
00096  * @returns true iff an element was removed.
00097  * @pre Assumes self points to a valid vector.
00098  * @post The capacity of self stays the same. The removed element is

```



```

00099  * destructed (cleaned up) as specified by the dtor_fn provided in vec_new.
00100  */
00101  bool vec_pop_back(Vec* self);
00102
00103  /* Inserts an element at the specified location in the container
00104  *
00105  * @param self      a pointer to the vector we want to insert into.
00106  * @param index      the index of the element we want to insert at.
00107  *                  Elements at this index and after it are "shifted" up
00108  *                  one position. If index is equal to the length, then we insert
00109  *                  at the end of the vector.
00110  * @param new_ele    the value we want to insert
00111  * @pre Assumes self points to a valid vector. If the index is > self->length
00112  * then this function will panic().
00113  * @post If after the operation the new length is greater than the old capacity
00114  * then a reallocation takes place and all elements are copied over. Capacity is
00115  * doubled. Any pointers to elements prior to this reallocation are invalidated.
00116  */
00117  void vec_insert(Vec* self, size_t index, ptr_t new_ele);
00118
00119  /* Erases an element at the specified valid location in the container
00120  *
00121  * @param self      a pointer to the vector we want to erase from.
00122  * @param index      the index of the element we want to erase at. Elements
00123  *                  after this index are "shifted" down one position.
00124  * @pre Assumes self points to a valid vector. If the index is >= self->length
00125  * then this function will panic().
00126  */
00127  void vec_erase(Vec* self, size_t index);
00128
00129  /* Resizes the container to a new specified capacity.
00130  * Does nothing if new_capacity <= self->length
00131  *
00132  * @param self      a pointer to the vector we want to resize.
00133  * @param new_capacity the new capacity of the vector.
00134  * @pre Assumes self points to a valid vector.
00135  * @post If a resize takes place, then a reallocation takes place and all
00136  * elements are copied over. Any pointers to elements prior to this
00137  * reallocation are invalidated.
00138  * @post The removed elements are destructed (cleaned up).
00139  */
00140  void vec_resize(Vec* self, size_t new_capacity);
00141
00142  /* Erases all elements from the container.
00143  * After this, the length of the vector is zero.
00144  * Capacity of the vector is unchanged.
00145  *
00146  * @param self a pointer to the vector we want to clear.
00147  * @pre Assumes self points to a valid vector.
00148  * @post The removed elements are destructed (cleaned up).
00149  */
00150  void vec_clear(Vec* self);
00151
00152  /* Destruct the vector.
00153  * All elements are destructed and storage is deallocated.
00154  * Must set capacity and length to zero. Data is set to NULL.
00155  *
00156  * @param self a pointer to the vector we want to destruct.
00157  * @pre Assumes self points to a valid vector.
00158  * @post The removed elements are destructed (cleaned up)
00159  * and data storage deallocated.
00160  */
00161  void vec_destroy(Vec* self);
00162
00163  #endif // VEC_H_

```


Index

- `_GNU_SOURCE`
 - `spthread.c`, [185](#)
 - `_POSIX_C_SOURCE`
 - `kernel.c`, [120](#)
 - `penn-shell.c`, [152](#)
- `ack`
 - `spthread_signal_args_st`, [27](#)
- `actual_arg`
 - `spthread_fwd_args_st`, [25](#)
- `actual_routine`
 - `spthread_fwd_args_st`, [25](#)
- `add_fd`
 - `fd_table.c`, [71](#)
 - `fd_table.h`, [75](#)
- `add_job`
 - `penn-shell.c`, [152](#)
 - `penn-shell.h`, [158](#)
- `add_process_to_run_queue`
 - `kernel.c`, [120](#)
 - `kernel.h`, [125](#)
- `add_process_to_scheduler`
 - `kernel.c`, [120](#)
 - `kernel.h`, [126](#)
- `add_process_to_zombie_queue`
 - `kernel.c`, [120](#)
 - `kernel.h`, [126](#)
- `alarm_handler`
 - `kernel.c`, [121](#)
- `arr`
 - `buffer`, [9](#)
- `Buffer`
 - `fat_core.h`, [45](#)
- `buffer`, [9](#)
 - `arr`, [9](#)
 - `size`, [9](#)
- `capacity`
 - `vec_st`, [28](#)
- `check_blocked_processes`
 - `kernel.c`, [121](#)
 - `kernel.h`, [126](#)
- `child_meta`
 - `spthread_fwd_args_st`, [26](#)
- `clear_deque`
 - `deque.c`, [168](#)
 - `deque.h`, [174](#)
- `command`
 - `Job`, [16](#)
- `commands`
 - `parsed_command`, [20](#)
- `convert_block_size`
 - `fat_core.c`, [35](#)
 - `fat_core.h`, [46](#)
- `create_file`
 - `fat_core.c`, [35](#)
 - `fat_core.h`, [46](#)
- `curr_process`
 - `KernelState`, [17](#)
- `curr_thread_num`
 - `KernelState`, [17](#)
- `current_processes`
 - `KernelState`, [18](#)
- `data`
 - `Node`, [19](#)
 - `vec_st`, [28](#)
- `delete_mem`
 - `Deque`, [10](#)
- `Deque`, [10](#)
 - `delete_mem`, [10](#)
 - `deque.h`, [173](#)
 - `front`, [10](#)
 - `size`, [10](#)
 - `tail`, [11](#)
- `deque.c`
 - `clear_deque`, [168](#)
 - `deque_contains`, [168](#)
 - `deque_get_back`, [168](#)
 - `deque_get_front`, [168](#)
 - `deque_get_nth_elem`, [169](#)
 - `deque_new`, [169](#)
 - `deque_pop_back`, [169](#)
 - `deque_pop_front`, [170](#)
 - `deque_push_back`, [170](#)
 - `deque_push_front`, [170](#)
 - `deque_remove_nth_elem`, [170](#)
 - `deque_remove_specific`, [171](#)
 - `deque_size`, [171](#)
- `deque.h`
 - `clear_deque`, [174](#)
 - `Deque`, [173](#)
 - `deque_contains`, [174](#)
 - `deque_get_back`, [174](#)
 - `deque_get_front`, [175](#)
 - `deque_get_nth_elem`, [175](#)
 - `DEQUE_H`, [173](#)
 - `deque_new`, [175](#)
 - `deque_pop_back`, [176](#)

- deque_pop_front, 176
 - deque_push_back, 176
 - deque_push_front, 177
 - deque_remove_nth_elem, 177
 - deque_remove_specific, 177
 - deque_size, 177
 - Node, 173
- deque_contains
 - deque.c, 168
 - deque.h, 174
- deque_get_back
 - deque.c, 168
 - deque.h, 174
- deque_get_front
 - deque.c, 168
 - deque.h, 175
- deque_get_nth_elem
 - deque.c, 169
 - deque.h, 175
- DEQUE_H
 - deque.h, 173
- deque_new
 - deque.c, 169
 - deque.h, 175
- deque_pop_back
 - deque.c, 169
 - deque.h, 176
- deque_pop_front
 - deque.c, 170
 - deque.h, 176
- deque_push_back
 - deque.c, 170
 - deque.h, 176
- deque_push_front
 - deque.c, 170
 - deque.h, 177
- deque_remove_nth_elem
 - deque.c, 170
 - deque.h, 177
- deque_remove_specific
 - deque.c, 171
 - deque.h, 177
- deque_size
 - deque.c, 171
 - deque.h, 177
- Dir_entry
 - fat_core.h, 45
- dir_entry, 11
 - first_block, 12
 - mtime, 12
 - name, 12
 - perm, 12
 - reserved, 12
 - size, 12
 - type, 13
- DIRECTORY_FILE
 - fat_core.h, 44
- dq_BLOCKED
 - KernelState, 18
- dq_DEAD
 - KernelState, 18
- dq_RUNNING
 - KernelState, 18
- dq_STOPPED
 - KernelState, 18
- dq_ZOMBIE
 - KernelState, 18
- ele_dtor_fn
 - vec_st, 28
- EOD_FLAG
 - fat_core.h, 44
- err.c
 - ERRNO, 30
 - error_case, 30
 - f_perror, 30
- err.h
 - ERRNO, 34
 - error_case, 33
 - f_perror, 33
 - FILE_NOT_FOUND, 31
 - FILE_SYSTEM, 31
 - FS_ALREADY_MOUNTED, 31
 - FS_NOT_MOUNTED, 31
 - FS_SUCCESS, 32
 - INVALID_ARGS, 32
 - INVALID_FD, 32
 - INVALID_OFFSET, 32
 - INVALID_OPERATION, 32
 - INVALID_PATH, 32
 - INVALID_WHENCE, 32
 - MEMORY_ERROR, 32
 - MKFS, 32
 - MOUNT, 32
 - NO_SPACE, 33
 - NOT_A_DIRECTORY, 33
 - PERMISSION_DENIED, 33
 - UNMOUNT, 33
- ERRNO
 - err.c, 30
 - err.h, 34
- error_case
 - err.c, 30
 - err.h, 33
- execute_command
 - execute_command.c, 142
 - execute_command.h, 144
- execute_command.c
 - execute_command, 142
 - execute_script_file, 142
 - execute_single_command, 142
 - handle_shell_builtin, 143
- execute_command.h
 - execute_command, 144
- execute_script_file
 - execute_command.c, 142
- execute_single_command

- execute_command.c, [142](#)
- EXPECT_COMMANDS
 - parser.h, [148](#)
- EXPECT_INPUT_FILENAME
 - parser.h, [148](#)
- EXPECT_OUTPUT_FILENAME
 - parser.h, [149](#)
- F_APPEND
 - fat_kernel.h, [62](#)
 - fd_table.h, [74](#)
- f_cat
 - fat_kernel.c, [56](#)
 - fat_kernel.h, [63](#)
- f_chmod
 - fat_kernel.c, [56](#)
 - fat_kernel.h, [64](#)
- f_cp
 - fat_kernel.c, [56](#)
 - fat_kernel.h, [64](#)
- f_get_permission
 - fat_kernel.c, [56](#)
 - fat_kernel.h, [64](#)
- f_mv
 - fat_kernel.c, [57](#)
 - fat_kernel.h, [64](#)
- f_perror
 - err.c, [30](#)
 - err.h, [33](#)
- F_READ
 - fat_kernel.h, [62](#)
 - fd_table.h, [74](#)
- f_rm
 - fat_kernel.c, [57](#)
 - fat_kernel.h, [65](#)
- f_touch
 - fat_kernel.c, [57](#)
 - fat_kernel.h, [65](#)
- F_WRITE
 - fat_kernel.h, [63](#)
 - fd_table.h, [74](#)
- fat_core.c
 - convert_block_size, [35](#)
 - create_file, [35](#)
 - format_file_info, [36](#)
 - fs_chmod, [36](#)
 - fs_create, [36](#)
 - fs_list_files, [36](#)
 - fs_mount, [37](#)
 - fs_mv, [37](#)
 - fs_rm, [37](#)
 - fs_touch, [38](#)
 - fs_unmount, [38](#)
 - get_directory, [38](#)
 - get_file_permission, [38](#)
 - index_to_directory, [39](#)
 - k_read_at, [39](#)
 - k_write_at, [39](#)
 - lookup_directory_offset, [40](#)
 - name_to_directory, [40](#)
 - offset_to_directory, [40](#)
 - perm_to_rwx, [41](#)
 - read_file, [41](#)
 - update_directory, [41](#)
 - write_file, [42](#)
- fat_core.h
 - Buffer, [45](#)
 - convert_block_size, [46](#)
 - create_file, [46](#)
 - Dir_entry, [45](#)
 - DIRECTORY_FILE, [44](#)
 - EOD_FLAG, [44](#)
 - file_info_callback_t, [46](#)
 - format_file_info, [46](#)
 - FREE_BLOCK, [44](#)
 - fs_chmod, [47](#)
 - fs_create, [47](#)
 - fs_list_files, [47](#)
 - fs_mount, [48](#)
 - fs_mv, [48](#)
 - fs_rm, [48](#)
 - fs_touch, [49](#)
 - fs_unmount, [49](#)
 - get_directory, [49](#)
 - get_file_permission, [49](#)
 - index_to_directory, [50](#)
 - k_read_at, [50](#)
 - k_write_at, [50](#)
 - LAST_BLOCK, [44](#)
 - LINK_FILE, [44](#)
 - lookup_directory_offset, [51](#)
 - name_to_directory, [51](#)
 - offset_to_directory, [51](#)
 - perm_to_rwx, [52](#)
 - read_file, [52](#)
 - REGULAR_FILE, [44](#)
 - SEEK_CUR, [44](#)
 - SEEK_END, [44](#)
 - SEEK_SET, [44](#)
 - UNKNOWN_FILE, [45](#)
 - update_directory, [52](#)
 - write_file, [53](#)
- fat_kernel.c
 - f_cat, [56](#)
 - f_chmod, [56](#)
 - f_cp, [56](#)
 - f_get_permission, [56](#)
 - f_mv, [57](#)
 - f_rm, [57](#)
 - f_touch, [57](#)
 - init, [57](#)
 - k_chmod, [57](#)
 - k_close, [58](#)
 - k_get_permission, [58](#)
 - k_ls, [58](#)
 - k_lseek, [59](#)
 - k_open, [59](#)

- k_read_at_offset, 59
 - k_unlink, 60
 - k_write_at_offset, 60
 - kf_read, 60
 - kf_write, 61
 - MAX_LINE_LENGTH, 56
- fat_kernel.h
 - F_APPEND, 62
 - f_cat, 63
 - f_chmod, 64
 - f_cp, 64
 - f_get_permission, 64
 - f_mv, 64
 - F_READ, 62
 - f_rm, 65
 - f_touch, 65
 - F_WRITE, 63
 - init, 65
 - k_chmod, 65
 - k_close, 65
 - k_get_permission, 67
 - k_ls, 67
 - k_lseek, 67
 - k_open, 67
 - k_read_at_offset, 68
 - k_unlink, 68
 - k_write_at_offset, 68
 - kf_read, 69
 - kf_write, 69
 - SEEK_CUR, 63
 - SEEK_END, 63
 - SEEK_SET, 63
 - STDERR_FILENO, 63
 - STDIN_FILENO, 63
 - STDOUT_FILENO, 63
- fd
 - fd_node, 14
- FD_Node
 - fd_table.h, 74
- fd_node, 13
 - fd, 14
 - mode, 14
 - name, 14
 - next, 14
 - offset, 14
 - size, 14
- fd_num
 - ProcessFDNode, 24
- FD_Table
 - fd_table.h, 74
- fd_table, 14
 - head, 15
 - pcb_t, 22
- fd_table.c
 - add_fd, 71
 - initialize_fd_table, 71
 - lookup_add_position, 72
 - lookup_fd, 72
 - remove_fd, 72
- fd_table.h
 - add_fd, 75
 - F_APPEND, 74
 - F_READ, 74
 - F_WRITE, 74
 - FD_Node, 74
 - FD_Table, 74
 - initialize_fd_table, 75
 - lookup_add_position, 75
 - lookup_fd, 75
 - remove_fd, 76
- file_descriptors
 - pcb_t, 22
- file_info_callback_t
 - fat_core.h, 46
- FILE_NOT_FOUND
 - err.h, 31
- FILE_SYSTEM
 - err.h, 31
- find_job_by_id
 - penn-shell.c, 153
 - penn-shell.h, 159
- find_job_by_pid
 - penn-shell.c, 153
 - penn-shell.h, 159
- find_last_job
 - penn-shell.c, 153
 - penn-shell.h, 159
- find_last_stopped_job
 - penn-shell.c, 153
 - penn-shell.h, 159
- first_block
 - dir_entry, 12
- fname
 - ProcessFDNode, 24
- foreground
 - pcb_t, 22
- format_file_info
 - fat_core.c, 36
 - fat_core.h, 46
- FREE_BLOCK
 - fat_core.h, 44
- free_job
 - penn-shell.c, 154
 - penn-shell.h, 160
- front
 - Deque, 10
- FS_ALREADY_MOUNTED
 - err.h, 31
- fs_chmod
 - fat_core.c, 36
 - fat_core.h, 47
- fs_create
 - fat_core.c, 36
 - fat_core.h, 47
- fs_list_files
 - fat_core.c, 36

- fat_core.h, [47](#)
- fs_mount
 - fat_core.c, [37](#)
 - fat_core.h, [48](#)
- fs_mv
 - fat_core.c, [37](#)
 - fat_core.h, [48](#)
- FS_NOT_MOUNTED
 - err.h, [31](#)
- fs_rm
 - fat_core.c, [37](#)
 - fat_core.h, [48](#)
- FS_SUCCESS
 - err.h, [32](#)
- fs_touch
 - fat_core.c, [38](#)
 - fat_core.h, [49](#)
- fs_unmount
 - fat_core.c, [38](#)
 - fat_core.h, [49](#)
- g_shutdown_requested
 - kernel.c, [123](#)
 - kernel.h, [128](#)
- get_directory
 - fat_core.c, [38](#)
 - fat_core.h, [49](#)
- get_file_permission
 - fat_core.c, [38](#)
 - fat_core.h, [49](#)
- get_kernel_ticks
 - kernel.c, [121](#)
 - kernel.h, [126](#)
- getKernelState
 - kernel.c, [121](#)
 - kernel.h, [127](#)
- handle_shell_builtin
 - execute_command.c, [143](#)
- head
 - fd_table, [15](#)
- index_to_directory
 - fat_core.c, [39](#)
 - fat_core.h, [50](#)
- init
 - fat_kernel.c, [57](#)
 - fat_kernel.h, [65](#)
- init_logger
 - logger.c, [179](#)
 - logger.h, [182](#)
- initialize_fd_table
 - fd_table.c, [71](#)
 - fd_table.h, [75](#)
- initialize_job_control
 - penn-shell.c, [154](#)
 - penn-shell.h, [160](#)
- int_handler
 - kernel.c, [121](#)
- INVALID_ARGS
 - err.h, [32](#)
- INVALID_FD
 - err.h, [32](#)
- INVALID_OFFSET
 - err.h, [32](#)
- INVALID_OPERATION
 - err.h, [32](#)
- INVALID_PATH
 - err.h, [32](#)
- INVALID_WHENCE
 - err.h, [32](#)
- is_background
 - parsed_command, [20](#)
- is_file_append
 - parsed_command, [20](#)
- Job, [15](#)
 - command, [16](#)
 - job_id, [16](#)
 - pcmd, [16](#)
 - penn-shell.h, [158](#)
 - pid, [16](#)
 - status, [16](#)
- job_id
 - Job, [16](#)
- job_list
 - penn-shell.c, [155](#)
 - penn-shell.h, [161](#)
- JOB_STATUS_DONE
 - penn-shell.h, [158](#)
- JOB_STATUS_RUNNING
 - penn-shell.h, [158](#)
- JOB_STATUS_STOPPED
 - penn-shell.h, [158](#)
- JobStatus
 - penn-shell.h, [158](#)
- JUMP_OUT
 - parser.c, [145](#)
- k
 - kernel.c, [123](#)
- k_chmod
 - fat_kernel.c, [57](#)
 - fat_kernel.h, [65](#)
- k_close
 - fat_kernel.c, [58](#)
 - fat_kernel.h, [65](#)
- k_get_current_process
 - kernel-call.c, [80](#)
 - kernel-call.h, [84](#)
- k_get_current_process_pid
 - kernel-call.c, [80](#)
 - kernel-call.h, [84](#)
- k_get_lowest_pid
 - kernel-call.c, [80](#)
- k_get_permission
 - fat_kernel.c, [58](#)
 - fat_kernel.h, [67](#)

- k_get_proc
 - kernel-call.c, 80
 - kernel-call.h, 84
- k_get_process_by_pid
 - kernel-call.c, 81
 - kernel-call.h, 84
- k_ls
 - fat_kernel.c, 58
 - fat_kernel.h, 67
- k_lseek
 - fat_kernel.c, 59
 - fat_kernel.h, 67
- k_open
 - fat_kernel.c, 59
 - fat_kernel.h, 67
- k_proc_cleanup
 - kernel-call.c, 81
 - kernel-call.h, 85
- k_proc_create
 - kernel-call.c, 81
 - kernel-call.h, 85
- k_read
 - kernel-call.c, 81
 - kernel-call.h, 85
- k_read_at
 - fat_core.c, 39
 - fat_core.h, 50
- k_read_at_offset
 - fat_kernel.c, 59
 - fat_kernel.h, 68
- k_release_fd
 - kernel-call.c, 82
 - kernel-call.h, 85
- k_unlink
 - fat_kernel.c, 60
 - fat_kernel.h, 68
- k_write
 - kernel-call.c, 82
 - kernel-call.h, 86
- k_write_at
 - fat_core.c, 39
 - fat_core.h, 50
- k_write_at_offset
 - fat_kernel.c, 60
 - fat_kernel.h, 68
- kernel-call.c
 - k_get_current_process, 80
 - k_get_current_process_pid, 80
 - k_get_lowest_pid, 80
 - k_get_proc, 80
 - k_get_process_by_pid, 81
 - k_proc_cleanup, 81
 - k_proc_create, 81
 - k_read, 81
 - k_release_fd, 82
 - k_write, 82
- kernel-call.h
 - k_get_current_process, 84
- k_get_current_process_pid, 84
- k_get_proc, 84
- k_get_process_by_pid, 84
- k_proc_cleanup, 85
- k_proc_create, 85
- k_read, 85
- k_release_fd, 85
- k_write, 86
- kernel.c
 - _POSIX_C_SOURCE, 120
 - add_process_to_run_queue, 120
 - add_process_to_scheduler, 120
 - add_process_to_zombie_queue, 120
 - alarm_handler, 121
 - check_blocked_processes, 121
 - g_shutdown_requested, 123
 - get_kernel_ticks, 121
 - getKernelState, 121
 - int_handler, 121
 - k, 123
 - kernel_set_up, 122
 - kernel_ticks, 123
 - remove_process_from_run_queue, 122
 - remove_process_from_zombie_queue, 122
 - start_kernel, 123
 - stop_handler, 123
- kernel.h
 - add_process_to_run_queue, 125
 - add_process_to_scheduler, 126
 - add_process_to_zombie_queue, 126
 - check_blocked_processes, 126
 - g_shutdown_requested, 128
 - get_kernel_ticks, 126
 - getKernelState, 127
 - kernel_set_up, 127
 - KernelState, 125
 - MAX_PROC, 125
 - PROCESS_QUANTA, 125
 - remove_process_from_run_queue, 127
 - remove_process_from_zombie_queue, 128
 - start_kernel, 128
- kernel_set_up
 - kernel.c, 122
 - kernel.h, 127
- kernel_ticks
 - kernel.c, 123
- KernelState, 17
 - curr_process, 17
 - curr_thread_num, 17
 - current_processes, 18
 - dq_BLOCKED, 18
 - dq_DEAD, 18
 - dq_RUNNING, 18
 - dq_STOPPED, 18
 - dq_ZOMBIE, 18
 - kernel.h, 125
 - process_quanta, 18
 - terminal_owner_pid, 18

- fat_kernel.c, [60](#)
 - fat_kernel.h, [69](#)
- kf_write
 - fat_kernel.c, [61](#)
 - fat_kernel.h, [69](#)
- LAST_BLOCK
 - fat_core.h, [44](#)
- length
 - vec_st, [28](#)
- LINE_BUFFER_CHUNK_SIZE
 - penn-shell.c, [152](#)
- LINK_FILE
 - fat_core.h, [44](#)
- log_event
 - logger.c, [179](#)
 - logger.h, [183](#)
- logger.c
 - init_logger, [179](#)
 - log_event, [179](#)
- logger.h
 - init_logger, [182](#)
 - log_event, [183](#)
- lookup_add_position
 - fd_table.c, [72](#)
 - fd_table.h, [75](#)
- lookup_directory_offset
 - fat_core.c, [40](#)
 - fat_core.h, [51](#)
- lookup_fd
 - fd_table.c, [72](#)
 - fd_table.h, [75](#)
- main
 - pennfat.c, [77](#)
 - pennos.c, [166](#)
- map_fat_error_to_p_errno
 - sys-call.c, [88](#)
- MAX_LINE_LENGTH
 - fat_kernel.c, [56](#)
 - pennfat.h, [78](#)
- MAX_MESSAGE_SIZE
 - penn-shell.h, [157](#)
 - user-call.h, [112](#)
- MAX_PROC
 - kernel.h, [125](#)
- MEMORY_ERROR
 - err.h, [32](#)
- meta
 - spthread_st, [28](#)
- meta_mutex
 - spthread_meta_st, [26](#)
- MILISEC_IN_NANO
 - spthread.c, [185](#)
- MKFS
 - err.h, [32](#)
- mode
 - fd_node, [14](#)
- ProcessFDNode, [24](#)
- MOUNT
 - err.h, [32](#)
- mtime
 - dir_entry, [12](#)
- name
 - dir_entry, [12](#)
 - fd_node, [14](#)
 - pcb_t, [22](#)
- name_to_directory
 - fat_core.c, [40](#)
 - fat_core.h, [51](#)
- next
 - fd_node, [14](#)
 - Node, [19](#)
 - ProcessFDNode, [24](#)
- next_job_id
 - penn-shell.c, [155](#)
 - penn-shell.h, [161](#)
- NO_SPACE
 - err.h, [33](#)
- Node, [19](#)
 - data, [19](#)
 - deque.h, [173](#)
 - next, [19](#)
 - prev, [19](#)
- NOT_A_DIRECTORY
 - err.h, [33](#)
- num_commands
 - parsed_command, [20](#)
- offset
 - fd_node, [14](#)
 - ProcessFDNode, [25](#)
- offset_to_directory
 - fat_core.c, [40](#)
 - fat_core.h, [51](#)
- P_ERRNO
 - p_errno.c, [130](#)
 - p_errno.h, [134](#)
 - sys-call.h, [103](#)
- p_errno.c
 - P_ERRNO, [130](#)
 - u_perror, [130](#)
- p_errno.h
 - P_ERRNO, [134](#)
 - P_ERRNO_ALREADY_MOUNTED, [131](#)
 - P_ERRNO_ECHILD, [131](#)
 - P_ERRNO_EINVAL, [132](#)
 - P_ERRNO_ESRCH, [132](#)
 - P_ERRNO_FILE_NOT_FOUND, [132](#)
 - P_ERRNO_INTERNAL, [132](#)
 - P_ERRNO_INVALID_ARG, [132](#)
 - P_ERRNO_INVALID_FD, [132](#)
 - P_ERRNO_INVALID_OFFSET, [132](#)
 - P_ERRNO_INVALID_OPERATION, [132](#)
 - P_ERRNO_INVALID_WHENCE, [132](#)

- P_ERRNO_NO_SPACE, [132](#)
- P_ERRNO_NOT_A_DIRECTORY, [133](#)
- P_ERRNO_NOT_MOUNTED, [133](#)
- P_ERRNO_PERMISSION, [133](#)
- P_ERRNO_SUCCESS, [133](#)
- P_ERRNO_UNKNOWN, [133](#)
- P_ERRNO_WRITE_CONFLICT, [133](#)
- u_perror, [133](#)
- P_ERRNO_ALREADY_MOUNTED
 - p_errno.h, [131](#)
- P_ERRNO_ECHILD
 - p_errno.h, [131](#)
- P_ERRNO_EINVAL
 - p_errno.h, [132](#)
- P_ERRNO_ESRCH
 - p_errno.h, [132](#)
- P_ERRNO_FILE_NOT_FOUND
 - p_errno.h, [132](#)
- P_ERRNO_INTERNAL
 - p_errno.h, [132](#)
- P_ERRNO_INVALID_ARG
 - p_errno.h, [132](#)
- P_ERRNO_INVALID_FD
 - p_errno.h, [132](#)
- P_ERRNO_INVALID_OFFSET
 - p_errno.h, [132](#)
- P_ERRNO_INVALID_OPERATION
 - p_errno.h, [132](#)
- P_ERRNO_INVALID_WHENCE
 - p_errno.h, [132](#)
- P_ERRNO_NO_SPACE
 - p_errno.h, [132](#)
- P_ERRNO_NOT_A_DIRECTORY
 - p_errno.h, [133](#)
- P_ERRNO_NOT_MOUNTED
 - p_errno.h, [133](#)
- P_ERRNO_PERMISSION
 - p_errno.h, [133](#)
- P_ERRNO_SUCCESS
 - p_errno.h, [133](#)
- P_ERRNO_UNKNOWN
 - p_errno.h, [133](#)
- P_ERRNO_WRITE_CONFLICT
 - p_errno.h, [133](#)
- P_SIGCONT
 - signals.h, [195](#)
- P_SIGSTOP
 - signals.h, [195](#)
- P_SIGTERM
 - signals.h, [195](#)
- P_WAIT_FLAG_SIGNALED
 - sys-call.h, [95](#)
- P_WAIT_FLAG_STOPPED
 - sys-call.h, [95](#)
- P_WAIT_SIG_MASK
 - sys-call.h, [95](#)
- P_WAIT_STATUS_MACROS_H
 - sys-call.h, [96](#)
- P_WIFEXITED
 - sys-call.h, [96](#)
- P_WIFSIGNALED
 - sys-call.h, [96](#)
- P_WIFSTOPPED
 - sys-call.h, [96](#)
- P_WSTOPSIG
 - sys-call.h, [96](#)
- P_WTERMSIG
 - sys-call.h, [96](#)
- parse_command
 - parser.c, [146](#)
 - parser.h, [149](#)
- parsed_command, [20](#)
 - commands, [20](#)
 - is_background, [20](#)
 - is_file_append, [20](#)
 - num_commands, [20](#)
 - stdin_file, [21](#)
 - stdout_file, [21](#)
- parser.c
 - JUMP_OUT, [145](#)
 - parse_command, [146](#)
 - print_parsed_command, [146](#)
 - print_parsed_command_without_end, [146](#)
 - print_parser_errcode, [147](#)
- parser.h
 - EXPECT_COMMANDS, [148](#)
 - EXPECT_INPUT_FILENAME, [148](#)
 - EXPECT_OUTPUT_FILENAME, [149](#)
 - parse_command, [149](#)
 - print_parsed_command, [149](#)
 - print_parsed_command_without_end, [150](#)
 - print_parser_errcode, [150](#)
 - UNEXPECTED_AMPERSAND, [149](#)
 - UNEXPECTED_FILE_INPUT, [149](#)
 - UNEXPECTED_FILE_OUTPUT, [149](#)
 - UNEXPECTED_PIPELINE, [149](#)
- pcb.c
 - pcb_add_fd, [135](#)
 - pcb_create, [135](#)
 - pcb_destroy, [135](#)
 - pcb_get_fd, [136](#)
 - pcb_initialize_fd_table, [136](#)
 - pcb_remove_fd, [136](#)
 - pcb_set_fd, [136](#)
- pcb.h
 - pcb_add_fd, [139](#)
 - pcb_get_fd, [139](#)
 - pcb_initialize_fd_table, [139](#)
 - pcb_remove_fd, [140](#)
 - pcb_set_fd, [140](#)
 - pcb_t, [138](#)
 - ProcessFDNode, [138](#)
- pcb_add_fd
 - pcb.c, [135](#)
 - pcb.h, [139](#)
- pcb_create

- pcb.c, 135
- pcb_destroy
 - pcb.c, 135
- pcb_get_fd
 - pcb.c, 136
 - pcb.h, 139
- pcb_initialize_fd_table
 - pcb.c, 136
 - pcb.h, 139
- pcb_remove_fd
 - pcb.c, 136
 - pcb.h, 140
- pcb_set_fd
 - pcb.c, 136
 - pcb.h, 140
- pcb_t, 21
 - fd_table, 22
 - file_descriptors, 22
 - foreground, 22
 - name, 22
 - pcb.h, 138
 - pid, 22
 - ppid, 22
 - priority_level, 23
 - status, 23
 - status_changed, 23
 - stop_signal, 23
 - term_signal, 23
 - thread, 23
 - wake_up_tick, 23
- pcmd
 - Job, 16
- penn-shell.c
 - _POSIX_C_SOURCE, 152
 - add_job, 152
 - find_job_by_id, 153
 - find_job_by_pid, 153
 - find_last_job, 153
 - find_last_stopped_job, 153
 - free_job, 154
 - initialize_job_control, 154
 - job_list, 155
 - LINE_BUFFER_CHUNK_SIZE, 152
 - next_job_id, 155
 - read_line_from_fd, 154
 - reconstruct_command, 154
 - remove_job_by_pid, 154
 - shell, 155
 - shell_pid, 155
 - terminal_controller_pid, 155
- penn-shell.h
 - add_job, 158
 - find_job_by_id, 159
 - find_job_by_pid, 159
 - find_last_job, 159
 - find_last_stopped_job, 159
 - free_job, 160
 - initialize_job_control, 160
 - Job, 158
 - job_list, 161
 - JOB_STATUS_DONE, 158
 - JOB_STATUS_RUNNING, 158
 - JOB_STATUS_STOPPED, 158
 - JobStatus, 158
 - MAX_MESSAGE_SIZE, 157
 - next_job_id, 161
 - PENSHELL_H, 157
 - pid_t, 158
 - reconstruct_command, 160
 - remove_job_by_pid, 160
 - shell, 161
 - shell_pid, 161
 - SHELL_PROMPT, 158
 - terminal_controller_pid, 161
- pennfat.c
 - main, 77
- pennfat.h
 - MAX_LINE_LENGTH, 78
- PennOS Group 15 - Spring 2025, 1
- pennos.c
 - main, 166
 - torta, 166
- PENSHELL_H
 - penn-shell.h, 157
- perm
 - dir_entry, 12
- perm_to_rwx
 - fat_core.c, 41
 - fat_core.h, 52
- PERMISSION_DENIED
 - err.h, 33
- pid
 - Job, 16
 - pcb_t, 22
- pid_t
 - penn-shell.h, 158
 - process-status.h, 193
- ppid
 - pcb_t, 22
- prev
 - Node, 19
- print_parsed_command
 - parser.c, 146
 - parser.h, 149
- print_parsed_command_without_end
 - parser.c, 146
 - parser.h, 150
- print_parser_errcode
 - parser.c, 147
 - parser.h, 150
- priority_level
 - pcb_t, 23
- process-status.h
 - pid_t, 193
 - PROCESS_STATUS_BLOCKED, 194
 - PROCESS_STATUS_DEAD, 194

- PROCESS_STATUS_RUNNING, [194](#)
- PROCESS_STATUS_STOPPED, [194](#)
- PROCESS_STATUS_WAITING, [194](#)
- PROCESS_STATUS_ZOMBIE, [194](#)
- ProcessStatus, [193](#)
- PROCESS_QUANTA
 - kernel.h, [125](#)
- process_quanta
 - KernelState, [18](#)
- PROCESS_STATUS_BLOCKED
 - process-status.h, [194](#)
- PROCESS_STATUS_DEAD
 - process-status.h, [194](#)
- PROCESS_STATUS_RUNNING
 - process-status.h, [194](#)
- PROCESS_STATUS_STOPPED
 - process-status.h, [194](#)
- PROCESS_STATUS_WAITING
 - process-status.h, [194](#)
- PROCESS_STATUS_ZOMBIE
 - process-status.h, [194](#)
- ProcessFDNode, [24](#)
 - fd_num, [24](#)
 - fname, [24](#)
 - mode, [24](#)
 - next, [24](#)
 - offset, [25](#)
 - pcb.h, [138](#)
- ProcessStatus
 - process-status.h, [193](#)
- pthread_fn
 - spthread.c, [185](#)
- ptr_dtor_fn
 - Vec.h, [199](#)
- ptr_t
 - Vec.h, [199](#)
- read_file
 - fat_core.c, [41](#)
 - fat_core.h, [52](#)
- read_line_from_fd
 - penn-shell.c, [154](#)
- README.md, [29](#)
- reconstruct_command
 - penn-shell.c, [154](#)
 - penn-shell.h, [160](#)
- REGULAR_FILE
 - fat_core.h, [44](#)
- remove_fd
 - fd_table.c, [72](#)
 - fd_table.h, [76](#)
- remove_job_by_pid
 - penn-shell.c, [154](#)
 - penn-shell.h, [160](#)
- remove_process_from_run_queue
 - kernel.c, [122](#)
 - kernel.h, [127](#)
- remove_process_from_zombie_queue
 - kernel.c, [122](#)
- kernel.h, [128](#)
- reserved
 - dir_entry, [12](#)
- s_chmod
 - sys-call.c, [88](#)
 - sys-call.h, [97](#)
- s_close
 - sys-call.c, [88](#)
 - sys-call.h, [97](#)
- s_exit
 - sys-call.c, [88](#)
 - sys-call.h, [97](#)
- s_get_permission
 - sys-call.c, [88](#)
 - sys-call.h, [97](#)
- s_getpid
 - sys-call.c, [89](#)
 - sys-call.h, [98](#)
- s_itos
 - sys-call.c, [89](#)
- s_kill
 - sys-call.c, [89](#)
 - sys-call.h, [98](#)
- s_ls
 - sys-call.c, [89](#)
 - sys-call.h, [98](#)
- s_lseek
 - sys-call.c, [90](#)
 - sys-call.h, [98](#)
- s_nice
 - sys-call.c, [90](#)
 - sys-call.h, [99](#)
- s_nice_pid
 - sys-call.c, [90](#)
 - sys-call.h, [99](#)
- s_open
 - sys-call.c, [91](#)
 - sys-call.h, [99](#)
- s_ps
 - sys-call.c, [91](#)
 - sys-call.h, [100](#)
- s_read
 - sys-call.c, [91](#)
 - sys-call.h, [100](#)
- s_register_end
 - sys-call.c, [91](#)
 - sys-call.h, [100](#)
- s_set_terminal_owner
 - sys-call.c, [92](#)
 - sys-call.h, [100](#)
- s_sleep
 - sys-call.c, [92](#)
 - sys-call.h, [101](#)
- s_spawn
 - sys-call.c, [92](#)
 - sys-call.h, [101](#)
- s_unlink
 - sys-call.c, [92](#)

- sys-call.h, 101
- s_waitpid
 - sys-call.c, 93
 - sys-call.h, 102
- s_waitpid_helper
 - sys-call.c, 93
- s_write
 - sys-call.c, 93
 - sys-call.h, 102
- SEEK_CUR
 - fat_core.h, 44
 - fat_kernel.h, 63
- SEEK_END
 - fat_core.h, 44
 - fat_kernel.h, 63
- SEEK_SET
 - fat_core.h, 44
 - fat_kernel.h, 63
- setup_cond
 - spthread_fwd_args_st, 26
- setup_done
 - spthread_fwd_args_st, 26
- setup_mutex
 - spthread_fwd_args_st, 26
- shell
 - penn-shell.c, 155
 - penn-shell.h, 161
- shell_pid
 - penn-shell.c, 155
 - penn-shell.h, 161
- SHELL_PROMPT
 - penn-shell.h, 158
- shutup_mutex
 - spthread_signal_args_st, 27
- signal
 - spthread_signal_args_st, 27
- signals.h
 - P_SIGCONT, 195
 - P_SIGSTOP, 195
 - P_SIGTERM, 195
- SIGPTH
 - spthread.h, 188
- size
 - buffer, 9
 - Deque, 10
 - dir_entry, 12
 - fd_node, 14
- spthread.c
 - _GNU_SOURCE, 185
 - MILISEC_IN_NANO, 185
 - pthread_fn, 185
 - spthread_cancel, 186
 - spthread_continue, 186
 - spthread_create, 186
 - spthread_disable_interrupts_self, 186
 - spthread_enable_interrupts_self, 186
 - spthread_equal, 186
 - spthread_exit, 186
- spthread_fwd_args, 185
- spthread_join, 187
- spthread_meta_t, 185
- SPTHREAD_RUNNING_STATE, 185
- spthread_self, 187
- SPTHREAD_SIG_CONTINUE, 185
- SPTHREAD_SIG_SUSPEND, 185
- spthread_signal_args, 186
- spthread_suspend, 187
- spthread_suspend_self, 187
- SPTHREAD_SUSPENDED_STATE, 185
- SPTHREAD_TERMINATED_STATE, 185
- spthread.h
 - SIGPTH, 188
 - spthread_cancel, 189
 - spthread_continue, 189
 - spthread_create, 189
 - spthread_disable_interrupts_self, 189
 - spthread_enable_interrupts_self, 189
 - spthread_equal, 189
 - spthread_exit, 189
 - spthread_join, 189
 - spthread_self, 190
 - spthread_suspend, 190
 - spthread_suspend_self, 190
 - spthread_t, 189
- spthread_cancel
 - spthread.c, 186
 - spthread.h, 189
- spthread_continue
 - spthread.c, 186
 - spthread.h, 189
- spthread_create
 - spthread.c, 186
 - spthread.h, 189
- spthread_disable_interrupts_self
 - spthread.c, 186
 - spthread.h, 189
- spthread_enable_interrupts_self
 - spthread.c, 186
 - spthread.h, 189
- spthread_equal
 - spthread.c, 186
 - spthread.h, 189
- spthread_exit
 - spthread.c, 186
 - spthread.h, 189
- spthread_fwd_args
 - spthread.c, 185
- spthread_fwd_args_st, 25
 - actual_arg, 25
 - actual_routine, 25
 - child_meta, 26
 - setup_cond, 26
 - setup_done, 26
 - setup_mutex, 26
- spthread_join
 - spthread.c, 187

- spthread.h, 189
- spthread_meta_st, 26
 - meta_mutex, 26
 - state, 26
 - suspend_set, 26
- spthread_meta_t
 - spthread.c, 185
- SPTHREAD_RUNNING_STATE
 - spthread.c, 185
- spthread_self
 - spthread.c, 187
 - spthread.h, 190
- SPTHREAD_SIG_CONTINUE
 - spthread.c, 185
- SPTHREAD_SIG_SUSPEND
 - spthread.c, 185
- spthread_signal_args
 - spthread.c, 186
- spthread_signal_args_st, 27
 - ack, 27
 - shutup_mutex, 27
 - signal, 27
- spthread_st, 27
 - meta, 28
 - thread, 28
- spthread_suspend
 - spthread.c, 187
 - spthread.h, 190
- spthread_suspend_self
 - spthread.c, 187
 - spthread.h, 190
- SPTHREAD_SUSPENDED_STATE
 - spthread.c, 185
- spthread_t
 - spthread.h, 189
- SPTHREAD_TERMINATED_STATE
 - spthread.c, 185
- src/fat/err.c, 29
- src/fat/err.h, 30, 34
- src/fat/fat_core.c, 34
- src/fat/fat_core.h, 42, 53
- src/fat/fat_kernel.c, 55
- src/fat/fat_kernel.h, 61, 70
- src/fat/fd_table.c, 71
- src/fat/fd_table.h, 73, 76
- src/fat/pennfat.c, 77
- src/fat/pennfat.h, 78, 79
- src/kernel/calls/kernel-call.c, 79
- src/kernel/calls/kernel-call.h, 83, 86
- src/kernel/calls/sys-call.c, 87
- src/kernel/calls/sys-call.h, 94, 103
- src/kernel/calls/user-call.c, 104
- src/kernel/calls/user-call.h, 110, 118
- src/kernel/kernel.c, 119
- src/kernel/kernel.h, 124, 129
- src/kernel/p_errno.c, 129
- src/kernel/p_errno.h, 131, 134
- src/kernel/pcb.c, 134
- src/kernel/pcb.h, 137, 141
- src/penn-shell/execute_command.c, 141
- src/penn-shell/execute_command.h, 143, 144
- src/penn-shell/parser.c, 145
- src/penn-shell/parser.h, 147, 150
- src/penn-shell/penn-shell.c, 151
- src/penn-shell/penn-shell.h, 156, 162
- src/penn-shell/stress.c, 162
- src/penn-shell/stress.h, 164, 165
- src/penn_os/pennos.c, 166
- src/penn_os/pennos.h, 166
- src/util/deque.c, 167
- src/util/deque.h, 172, 178
- src/util/logger/logger.c, 179
- src/util/logger/logger.h, 181, 183
- src/util/spthread.c, 183
- src/util/spthread.h, 187, 190
- src/util/types/process-status.h, 193, 194
- src/util/types/signals.h, 194, 195
- src/util/Vec.c, 195
- src/util/Vec.h, 198, 202
- start_kernel
 - kernel.c, 123
 - kernel.h, 128
- state
 - spthread_meta_st, 26
- status
 - Job, 16
 - pcb_t, 23
- status_changed
 - pcb_t, 23
- STDERR_FILENO
 - fat_kernel.h, 63
- stdin_file
 - parsed_command, 21
- STDIN_FILENO
 - fat_kernel.h, 63
- stdout_file
 - parsed_command, 21
- STDOUT_FILENO
 - fat_kernel.h, 63
- stoi
 - user-call.c, 105
 - user-call.h, 112
- stop_handler
 - kernel.c, 123
- stop_signal
 - pcb_t, 23
- stress.c
 - u_crash, 163
 - u_hang, 163
 - u_nohang, 163
 - u_recur, 164
- stress.h
 - u_crash, 165
 - u_hang, 165
 - u_nohang, 165
 - u_recur, 165

- suspend_set
 - sphthread_meta_st, 26
- sys-call.c
 - map_fat_error_to_p_errno, 88
 - s_chmod, 88
 - s_close, 88
 - s_exit, 88
 - s_get_permission, 88
 - s_getpid, 89
 - s_itos, 89
 - s_kill, 89
 - s_ls, 89
 - s_lseek, 90
 - s_nice, 90
 - s_nice_pid, 90
 - s_open, 91
 - s_ps, 91
 - s_read, 91
 - s_register_end, 91
 - s_set_terminal_owner, 92
 - s_sleep, 92
 - s_spawn, 92
 - s_unlink, 92
 - s_waitpid, 93
 - s_waitpid_helper, 93
 - s_write, 93
- sys-call.h
 - P_ERRNO, 103
 - P_WAIT_FLAG_SIGNALED, 95
 - P_WAIT_FLAG_STOPPED, 95
 - P_WAIT_SIG_MASK, 95
 - P_WAIT_STATUS_MACROS_H, 96
 - P_WIFEXITED, 96
 - P_WIFSIGNALED, 96
 - P_WIFSTOPPED, 96
 - P_WSTOPSIG, 96
 - P_WTERMSIG, 96
 - s_chmod, 97
 - s_close, 97
 - s_exit, 97
 - s_get_permission, 97
 - s_getpid, 98
 - s_kill, 98
 - s_ls, 98
 - s_lseek, 98
 - s_nice, 99
 - s_nice_pid, 99
 - s_open, 99
 - s_ps, 100
 - s_read, 100
 - s_register_end, 100
 - s_set_terminal_owner, 100
 - s_sleep, 101
 - s_spawn, 101
 - s_unlink, 101
 - s_waitpid, 102
 - s_write, 102
- tail
 - Deque, 11
- term_signal
 - pcb_t, 23
- terminal_controller_pid
 - penn-shell.c, 155
 - penn-shell.h, 161
- terminal_owner_pid
 - KernelState, 18
- thread
 - pcb_t, 23
 - sphthread_st, 28
- torta
 - pennos.c, 166
- type
 - dir_entry, 13
- u_bg
 - user-call.c, 105
 - user-call.h, 112
- u_busy
 - user-call.c, 105
 - user-call.h, 112
- u_cat
 - user-call.c, 106
 - user-call.h, 112
- u_chmod
 - user-call.c, 106
 - user-call.h, 113
- u_cp
 - user-call.c, 106
 - user-call.h, 113
- u_crash
 - stress.c, 163
 - stress.h, 165
 - user-call.h, 113
- u_echo
 - user-call.c, 106
 - user-call.h, 114
- u_fg
 - user-call.c, 107
 - user-call.h, 114
- u_hang
 - stress.c, 163
 - stress.h, 165
 - user-call.h, 114
- u_jobs
 - user-call.c, 107
 - user-call.h, 114
- u_kill
 - user-call.c, 107
 - user-call.h, 114
- u_logout
 - user-call.c, 107
 - user-call.h, 115
- u_ls
 - user-call.c, 107
 - user-call.h, 115
- u_man
 - user-call.c, 108

- user-call.h, 115
- u_mv
 - user-call.c, 108
 - user-call.h, 115
- u_nice
 - user-call.c, 108
 - user-call.h, 115
- u_nice_pid
 - user-call.c, 108
 - user-call.h, 116
- u_nohang
 - stress.c, 163
 - stress.h, 165
 - user-call.h, 116
- u_orphan_child
 - user-call.c, 109
 - user-call.h, 116
- u_orphanify
 - user-call.c, 109
 - user-call.h, 116
- u_perror
 - p_errno.c, 130
 - p_errno.h, 133
- u_ps
 - user-call.c, 109
 - user-call.h, 116
- u_recur
 - stress.c, 164
 - stress.h, 165
 - user-call.h, 117
- u_rm
 - user-call.c, 109
 - user-call.h, 117
- u_sleep
 - user-call.c, 109
 - user-call.h, 117
- u_touch
 - user-call.c, 110
 - user-call.h, 117
- u_zombify
 - user-call.c, 110
 - user-call.h, 117
- UNEXPECTED_AMPERSAND
 - parser.h, 149
- UNEXPECTED_FILE_INPUT
 - parser.h, 149
- UNEXPECTED_FILE_OUTPUT
 - parser.h, 149
- UNEXPECTED_PIPELINE
 - parser.h, 149
- UNKNOWN_FILE
 - fat_core.h, 45
- UNMOUNT
 - err.h, 33
- update_directory
 - fat_core.c, 41
 - fat_core.h, 52
- user-call.c
- stoi, 105
- u_bg, 105
- u_busy, 105
- u_cat, 106
- u_chmod, 106
- u_cp, 106
- u_echo, 106
- u_fg, 107
- u_jobs, 107
- u_kill, 107
- u_logout, 107
- u_ls, 107
- u_man, 108
- u_mv, 108
- u_nice, 108
- u_nice_pid, 108
- u_orphan_child, 109
- u_orphanify, 109
- u_ps, 109
- u_rm, 109
- u_sleep, 109
- u_touch, 110
- u_zombify, 110
- zombie_child, 110
- user-call.h
 - MAX_MESSAGE_SIZE, 112
 - stoi, 112
 - u_bg, 112
 - u_busy, 112
 - u_cat, 112
 - u_chmod, 113
 - u_cp, 113
 - u_crash, 113
 - u_echo, 114
 - u_fg, 114
 - u_hang, 114
 - u_jobs, 114
 - u_kill, 114
 - u_logout, 115
 - u_ls, 115
 - u_man, 115
 - u_mv, 115
 - u_nice, 115
 - u_nice_pid, 116
 - u_nohang, 116
 - u_orphan_child, 116
 - u_orphanify, 116
 - u_ps, 116
 - u_recur, 117
 - u_rm, 117
 - u_sleep, 117
 - u_touch, 117
 - u_zombify, 117
 - zombie_child, 118
- Vec
 - Vec.h, 199
- Vec.c
 - vec_clear, 196

- vec_destroy, [196](#)
- vec_erase, [196](#)
- vec_get, [196](#)
- vec_insert, [196](#)
- vec_new, [196](#)
- vec_pop_back, [197](#)
- vec_push_back, [197](#)
- vec_resize, [197](#)
- vec_set, [197](#)
- Vec.h
 - ptr_dtor_fn, [199](#)
 - ptr_t, [199](#)
 - Vec, [199](#)
 - vec_capacity, [199](#)
 - vec_clear, [200](#)
 - vec_destroy, [200](#)
 - vec_erase, [200](#)
 - vec_get, [200](#)
 - vec_insert, [200](#)
 - vec_is_empty, [199](#)
 - vec_len, [199](#)
 - vec_new, [200](#)
 - vec_pop_back, [201](#)
 - vec_push_back, [201](#)
 - vec_resize, [201](#)
 - vec_set, [201](#)
- vec_capacity
 - Vec.h, [199](#)
- vec_clear
 - Vec.c, [196](#)
 - Vec.h, [200](#)
- vec_destroy
 - Vec.c, [196](#)
 - Vec.h, [200](#)
- vec_erase
 - Vec.c, [196](#)
 - Vec.h, [200](#)
- vec_get
 - Vec.c, [196](#)
 - Vec.h, [200](#)
- vec_insert
 - Vec.c, [196](#)
 - Vec.h, [200](#)
- vec_is_empty
 - Vec.h, [199](#)
- vec_len
 - Vec.h, [199](#)
- vec_new
 - Vec.c, [196](#)
 - Vec.h, [200](#)
- vec_pop_back
 - Vec.c, [197](#)
 - Vec.h, [201](#)
- vec_push_back
 - Vec.c, [197](#)
 - Vec.h, [201](#)
- vec_resize
 - Vec.c, [197](#)
- Vec.h, [201](#)
- vec_set
 - Vec.c, [197](#)
 - Vec.h, [201](#)
- vec_st, [28](#)
 - capacity, [28](#)
 - data, [28](#)
 - ele_dtor_fn, [28](#)
 - length, [28](#)
- wake_up_tick
 - pcb_t, [23](#)
- write_file
 - fat_core.c, [42](#)
 - fat_core.h, [53](#)
- zombie_child
 - user-call.c, [110](#)
 - user-call.h, [118](#)