

PennOS

Generated by Doxygen 1.9.6

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 bgNode Struct Reference	5
3.1.1 Field Documentation	5
3.1.1.1 cm	5
3.1.1.2 groupsize	5
3.1.1.3 jid	5
3.1.1.4 next	6
3.1.1.5 self_pgjid	6
3.1.1.6 self_pid	6
3.1.1.7 state	6
3.2 j_queue Struct Reference	6
3.2.1 Field Documentation	6
3.2.1.1 head	6
3.2.1.2 size	7
3.2.1.3 tail	7
3.3 Job Struct Reference	7
3.3.1 Field Documentation	7
3.3.1.1 cmd	7
3.3.1.2 count_finished	7
3.3.1.3 ground	8
3.3.1.4 group_id	8
3.3.1.5 job_id	8
3.3.1.6 next	8
3.3.1.7 prev	8
3.3.1.8 rawCmd	8
3.3.1.9 status	8
3.4 jobQueue Struct Reference	9
3.4.1 Field Documentation	9
3.4.1.1 front	9
3.4.1.2 q_size	9
3.4.1.3 rear	9
3.5 parsed_command Struct Reference	9
3.5.1 Detailed Description	10
3.5.2 Field Documentation	10
3.5.2.1 commands	10
3.5.2.2 is_background	10
3.5.2.3 is_file_append	10

3.5.2.4 num_commands	10
3.5.2.5 stdin_file	10
3.5.2.6 stdout_file	10
3.6 Pcb Struct Reference	11
3.6.1 Field Documentation	11
3.6.1.1 changed_child_pcb	11
3.6.1.2 children_pcb	11
3.6.1.3 cmd	12
3.6.1.4 exited_child	12
3.6.1.5 fd	12
3.6.1.6 fg	12
3.6.1.7 has_tc	12
3.6.1.8 name	12
3.6.1.9 next	12
3.6.1.10 next_child	12
3.6.1.11 number_of_children	13
3.6.1.12 parent_pcb	13
3.6.1.13 pgid	13
3.6.1.14 pid	13
3.6.1.15 ppid	13
3.6.1.16 prev	13
3.6.1.17 prev_child	13
3.6.1.18 priority	13
3.6.1.19 reading	14
3.6.1.20 sleep_time	14
3.6.1.21 sleeping	14
3.6.1.22 state_change	14
3.6.1.23 stateChangeType	14
3.6.1.24 status	14
3.6.1.25 uc	14
3.6.1.26 waiting	14
3.6.1.27 waiting_on	15
3.6.1.28 zombies	15
3.7 queue Struct Reference	15
3.7.1 Field Documentation	15
3.7.1.1 head	15
3.7.1.2 size	15
3.7.1.3 tail	16
3.8 superblock Struct Reference	16
3.8.1 Field Documentation	16
3.8.1.1 Block_size	16
3.8.1.2 DBoffset	16

3.8.1.3 FAT_tablesize	16
3.8.1.4 fd	17
3.8.1.5 num_data_Blocks	17
3.8.1.6 num_FATBlocks	17
3.8.1.7 rdsiz	17
4 File Documentation	19
4.1 errors.c File Reference	19
4.1.1 Function Documentation	19
4.1.1.1 set_errno()	19
4.2 errors.h File Reference	19
4.2.1 Enumeration Type Documentation	19
4.2.1.1 ERRORS	19
4.2.2 Function Documentation	20
4.2.2.1 p_perror()	20
4.2.2.2 set_errno()	20
4.3 errors.h	20
4.4 f.c File Reference	20
4.4.1 Macro Definition Documentation	21
4.4.1.1 F_SEEK_CUR	21
4.4.1.2 F_SEEK_END	22
4.4.1.3 F_SEEK_SET	22
4.4.1.4 INPUT_SIZE	22
4.4.2 Function Documentation	22
4.4.2.1 check_allzero()	22
4.4.2.2 chmod1()	22
4.4.2.3 contain_file()	22
4.4.2.4 create_newfile()	23
4.4.2.5 data_write()	23
4.4.2.6 f_cat()	23
4.4.2.7 f_chmod()	23
4.4.2.8 f_close()	23
4.4.2.9 f_copy()	24
4.4.2.10 f_ls()	24
4.4.2.11 f_lseek()	24
4.4.2.12 f_mv()	24
4.4.2.13 f_open()	24
4.4.2.14 f_read()	24
4.4.2.15 f_rm()	25
4.4.2.16 f_touch()	25
4.4.2.17 f_unlink()	25
4.4.2.18 f_unmount()	25

4.4.2.19 f_write()	25
4.4.2.20 fatfs_init()	25
4.4.2.21 find_available_block()	26
4.4.2.22 rm()	26
4.4.2.23 touch()	26
4.4.2.24 update_dir_entry()	26
4.4.2.25 update_fat()	26
4.4.3 Variable Documentation	26
4.4.3.1 FAT_block	26
4.4.3.2 rd	27
4.4.3.3 sb	27
4.5 f.h File Reference	27
4.5.1 Function Documentation	27
4.5.1.1 __attribute__()	28
4.5.1.2 f_cat()	28
4.5.1.3 f_close()	28
4.5.1.4 f_copy()	28
4.5.1.5 f_ls()	28
4.5.1.6 f_lseek()	28
4.5.1.7 f_mv()	29
4.5.1.8 f_open()	29
4.5.1.9 f_read()	29
4.5.1.10 f_rm()	29
4.5.1.11 f_touch()	29
4.5.1.12 f_unlink()	29
4.5.1.13 f_unmount()	30
4.5.1.14 f_write()	30
4.5.1.15 fatfs_init()	30
4.6 f.h	30
4.7 j_queue.c File Reference	31
4.7.1 Function Documentation	31
4.7.1.1 create_j_queue()	31
4.7.1.2 find_job_with_id()	31
4.7.1.3 find_last_stopped_job()	32
4.7.1.4 free_j_queue()	32
4.7.1.5 pop_job()	32
4.7.1.6 print_j_list()	32
4.7.1.7 push_job()	32
4.7.1.8 remove_job()	32
4.8 j_queue.h File Reference	32
4.8.1 Function Documentation	33
4.8.1.1 create_j_queue()	33

4.8.1.2 find_job_with_id()	33
4.8.1.3 find_last_stopped_job()	33
4.8.1.4 free_j_queue()	33
4.8.1.5 pop_job()	34
4.8.1.6 print_j_list()	34
4.8.1.7 push_job()	34
4.8.1.8 remove_job()	34
4.9 j_queue.h	34
4.10 Job.c File Reference	35
4.10.1 Function Documentation	35
4.10.1.1 create_job()	35
4.10.1.2 free_job()	35
4.11 Job.h File Reference	35
4.11.1 Enumeration Type Documentation	36
4.11.1.1 jobGround	36
4.11.1.2 jobStatus	36
4.11.2 Function Documentation	36
4.11.2.1 create_job()	36
4.11.2.2 free_job()	37
4.12 Job.h	37
4.13 kernel.c File Reference	37
4.13.1 Typedef Documentation	39
4.13.1.1 q	39
4.13.2 Function Documentation	39
4.13.2.1 free_queues()	39
4.13.2.2 get_active_context()	39
4.13.2.3 get_active_pcb()	39
4.13.2.4 get_blocked_queue()	39
4.13.2.5 get_clock_ticks()	40
4.13.2.6 get_counter()	40
4.13.2.7 get_how_ended()	40
4.13.2.8 get_log_fp()	40
4.13.2.9 get_next_queue()	40
4.13.2.10 get_next_valid_queue()	40
4.13.2.11 get_num_of_ready_processes()	40
4.13.2.12 get_pcb()	41
4.13.2.13 get_pcb_with_pgid()	41
4.13.2.14 get_queue_with_priority()	41
4.13.2.15 get_scheduler_context()	41
4.13.2.16 get_shell_context()	41
4.13.2.17 idle_process()	41
4.13.2.18 inc_clock()	41

4.13.2.19	init_scheduler_context()	42
4.13.2.20	initialise_queues()	42
4.13.2.21	k_block_sigset()	42
4.13.2.22	k_boot_kernel()	42
4.13.2.23	k_exit()	42
4.13.2.24	k_process_block()	42
4.13.2.25	k_process_cleanup()	43
4.13.2.26	k_process_create()	43
4.13.2.27	k_process_kill()	43
4.13.2.28	k_sleep()	44
4.13.2.29	k_tcset()	44
4.13.2.30	k_unblock_sigset()	44
4.13.2.31	k_waitpid()	44
4.13.2.32	make_context()	45
4.13.2.33	print_all_queues_info()	45
4.13.2.34	scheduler()	45
4.13.2.35	set_active_pcb_and_context()	45
4.13.2.36	set_how_ended()	46
4.13.2.37	setup_idle_process()	46
4.13.2.38	setup_log()	46
4.13.2.39	switch_to_scheduler()	46
4.13.3	Variable Documentation	46
4.13.3.1	log_fp	46
4.13.3.2	pid_count	46
4.13.3.3	signal_set	46
4.14	kernel.h File Reference	47
4.14.1	Enumeration Type Documentation	48
4.14.1.1	HOW_ENDED	48
4.14.2	Function Documentation	48
4.14.2.1	boot_kernel()	48
4.14.2.2	get_active_pcb()	48
4.14.2.3	get_blocked_queue()	48
4.14.2.4	get_clock_ticks()	48
4.14.2.5	get_counter()	49
4.14.2.6	get_log_fp()	49
4.14.2.7	get_num_of_ready_processes()	49
4.14.2.8	get_pcb()	49
4.14.2.9	get_pcbs_with_pgid()	49
4.14.2.10	get_queue_with_priority()	49
4.14.2.11	get_scheduler_context()	49
4.14.2.12	get_shell_context()	50
4.14.2.13	idle_process()	50

4.14.2.14 k_boot_kernel()	50
4.14.2.15 k_exit()	50
4.14.2.16 k_process_block()	50
4.14.2.17 k_process_cleanup()	51
4.14.2.18 k_process_create()	51
4.14.2.19 k_process_kill()	51
4.14.2.20 k_sleep()	51
4.14.2.21 k_tcset()	52
4.14.2.22 k_waitpid()	52
4.14.2.23 make_context()	53
4.14.2.24 setup_log()	53
4.14.2.25 switch_to_scheduler()	53
4.14.2.26 wait_log()	53
4.15 kernel.h	53
4.16 parser.h File Reference	54
4.16.1 Macro Definition Documentation	55
4.16.1.1 EXPECT_COMMANDS	55
4.16.1.2 EXPECT_INPUT_FILENAME	55
4.16.1.3 EXPECT_OUTPUT_FILENAME	55
4.16.1.4 UNEXPECTED_AMPERSAND	55
4.16.1.5 UNEXPECTED_FILE_INPUT	55
4.16.1.6 UNEXPECTED_FILE_OUTPUT	55
4.16.1.7 UNEXPECTED_PIPELINE	55
4.16.2 Function Documentation	56
4.16.2.1 parse_command()	56
4.16.2.2 print_parsed_command()	56
4.17 parser.h	56
4.18 pcb.c File Reference	57
4.18.1 Function Documentation	57
4.18.1.1 add_child_to_parent_q()	58
4.18.1.2 add_child_to_parent_zombies()	58
4.18.1.3 create_pcb()	58
4.18.1.4 create_pcb_no_context()	58
4.18.1.5 delete_child_pcb()	58
4.18.1.6 free_pcb()	59
4.18.1.7 remove_child_pcb()	59
4.19 pcb.h File Reference	59
4.19.1 Enumeration Type Documentation	60
4.19.1.1 status	60
4.19.2 Function Documentation	60
4.19.2.1 add_child_to_parent_q()	60
4.19.2.2 add_child_to_parent_zombies()	61

4.19.2.3 create_pcb()	61
4.19.2.4 create_pcb_no_context()	61
4.19.2.5 delete_child_pcb()	61
4.19.2.6 free_pcb()	62
4.19.2.7 remove_child_pcb()	62
4.20 pcb.h	62
4.21 PennFAT.c File Reference	63
4.21.1 Macro Definition Documentation	65
4.21.1.1 INPUT_SIZE	65
4.21.2 Function Documentation	65
4.21.2.1 bg()	65
4.21.2.2 cat()	65
4.21.2.3 cat_a()	65
4.21.2.4 cat_file()	65
4.21.2.5 cat_w()	66
4.21.2.6 check_allzero()	66
4.21.2.7 check_current()	66
4.21.2.8 check_finished()	66
4.21.2.9 check_redirection()	66
4.21.2.10 chmod1()	66
4.21.2.11 contain_file()	67
4.21.2.12 copy()	67
4.21.2.13 create_newfile()	67
4.21.2.14 createQueue()	67
4.21.2.15 creatNode()	67
4.21.2.16 data_write()	67
4.21.2.17 deQueue()	68
4.21.2.18 enQueue()	68
4.21.2.19 fg()	68
4.21.2.20 find_available_block()	68
4.21.2.21 freeq()	68
4.21.2.22 INThandler()	68
4.21.2.23 jobs()	69
4.21.2.24 ls()	69
4.21.2.25 main()	69
4.21.2.26 mkfs()	69
4.21.2.27 mount()	69
4.21.2.28 mv()	69
4.21.2.29 print_status()	70
4.21.2.30 removeQueue()	70
4.21.2.31 rm()	70
4.21.2.32 search_byjid()	70

4.21.2.33 sig_handler()	70
4.21.2.34 sigchld_handler()	70
4.21.2.35 sigtstp_handler()	70
4.21.2.36 stringcpy()	71
4.21.2.37 touch()	71
4.21.2.38 unmount()	71
4.21.2.39 update_dir_entry()	71
4.21.2.40 update_fat()	71
4.21.3 Variable Documentation	71
4.21.3.1 back	71
4.21.3.2 c	72
4.21.3.3 currentjob	72
4.21.3.4 FAT_block	72
4.21.3.5 gq	72
4.21.3.6 jobid	72
4.21.3.7 pid	72
4.21.3.8 pid1	72
4.21.3.9 rd	72
4.21.3.10 sb	73
4.22 PennFAT.h File Reference	73
4.22.1 Function Documentation	73
4.22.1.1 __attribute__()	74
4.22.1.2 cat()	74
4.22.1.3 check_allzero()	74
4.22.1.4 chmod1()	74
4.22.1.5 contain_file()	74
4.22.1.6 copy()	74
4.22.1.7 data_write()	75
4.22.1.8 find_available_block()	75
4.22.1.9 ls()	75
4.22.1.10 mkfs()	75
4.22.1.11 mount()	76
4.22.1.12 mv()	76
4.22.1.13 print_status()	76
4.22.1.14 rm()	76
4.22.1.15 touch()	76
4.22.1.16 unmount()	76
4.22.1.17 update_dir_entry()	77
4.22.1.18 update_fat()	77
4.23 PennFAT.h	77
4.24 queue.c File Reference	78
4.24.1 Function Documentation	78

4.24.1.1 create_queue()	78
4.24.1.2 free_child_queue()	78
4.24.1.3 free_queue()	79
4.24.1.4 front()	79
4.24.1.5 get_child_pcb_with_pid()	79
4.24.1.6 get_pcb_with_pid()	80
4.24.1.7 get_pcb_with_pgid_in_queue()	80
4.24.1.8 is_empty()	80
4.24.1.9 pop()	80
4.24.1.10 pop_child()	80
4.24.1.11 print_queue()	80
4.24.1.12 print_queue_info()	81
4.24.1.13 push()	81
4.24.1.14 push_to_front()	81
4.24.1.15 remove_pcb()	81
4.25 queue.h File Reference	81
4.25.1 Function Documentation	82
4.25.1.1 add_to_blocked_list()	82
4.25.1.2 create_queue()	82
4.25.1.3 free_child_queue()	82
4.25.1.4 free_queue()	83
4.25.1.5 front()	83
4.25.1.6 get_child_pcb_with_pid()	83
4.25.1.7 get_pcb_with_pid()	84
4.25.1.8 get_pcb_with_pgid_in_queue()	84
4.25.1.9 is_empty()	84
4.25.1.10 pop()	84
4.25.1.11 print_queue()	84
4.25.1.12 print_queue_info()	84
4.25.1.13 push()	85
4.25.1.14 remove_pcb()	85
4.25.2 Variable Documentation	85
4.25.2.1 blocked_list	85
4.26 queue.h	85
4.27 shell.c File Reference	86
4.27.1 Macro Definition Documentation	87
4.27.1.1 MAX_LEN	87
4.27.2 Function Documentation	87
4.27.2.1 fg_pipeline()	87
4.27.2.2 get_arg_count()	87
4.27.2.3 get_args_from_cmd()	87
4.27.2.4 handle_command()	87

4.27.2.5 <code>handle_sigint()</code>	88
4.27.2.6 <code>handle_sigstsp()</code>	88
4.27.2.7 <code>interactive_shell()</code>	88
4.27.2.8 <code>main()</code>	88
4.27.2.9 <code>newline()</code>	88
4.27.2.10 <code>non_interactive()</code>	88
4.27.2.11 <code>parse()</code>	88
4.27.2.12 <code>poll_background_jobs()</code>	89
4.27.2.13 <code>print_finished()</code>	89
4.27.2.14 <code>prompt()</code>	89
4.27.3 Variable Documentation	89
4.27.3.1 <code>job_id</code>	89
4.27.3.2 <code>jobs_list</code>	89
4.27.3.3 <code>mode</code>	89
4.27.3.4 <code>start</code>	89
4.28 <code>shell.h</code> File Reference	90
4.28.1 Enumeration Type Documentation	90
4.28.1.1 <code>SHELL_MODE</code>	90
4.28.2 Function Documentation	90
4.28.2.1 <code>fg_pipeline()</code>	90
4.28.2.2 <code>parse()</code>	91
4.28.2.3 <code>poll_background_jobs()</code>	91
4.28.2.4 <code>print_finished()</code>	91
4.29 <code>shell.h</code>	91
4.30 <code>user.c</code> File Reference	91
4.30.1 Function Documentation	92
4.30.1.1 <code>custom_read()</code>	93
4.30.1.2 <code>dummy()</code>	93
4.30.1.3 <code>echo()</code>	93
4.30.1.4 <code>kill_as_process()</code>	93
4.30.1.5 <code>orphan_child()</code>	94
4.30.1.6 <code>orphanify()</code>	94
4.30.1.7 <code>p_boot_kernel()</code>	94
4.30.1.8 <code>p_exit()</code>	94
4.30.1.9 <code>p_getpid()</code>	94
4.30.1.10 <code>p_kill()</code>	95
4.30.1.11 <code>p_nice()</code>	95
4.30.1.12 <code>p_sleep()</code>	95
4.30.1.13 <code>p_spawn()</code>	96
4.30.1.14 <code>p_switch_to_scheduler()</code>	96
4.30.1.15 <code>p_tcset()</code>	96
4.30.1.16 <code>p_waitpid()</code>	96

4.30.1.17 p_waitpid_old()	96
4.30.1.18 print_all_process_info()	97
4.30.1.19 sleeps()	97
4.30.1.20 spawn_shell()	97
4.30.1.21 W_WIFCONTINUED()	97
4.30.1.22 W_WIFEXITED()	97
4.30.1.23 W_WIFSIGNALED()	97
4.30.1.24 W_WIFSTOPPED()	98
4.30.1.25 wait_log()	98
4.30.1.26 zombie_child()	98
4.30.1.27 zombify()	98
4.31 user.h File Reference	98
4.31.1 Enumeration Type Documentation	99
4.31.1.1 CHANGE	99
4.31.1.2 S_SIGNALS	100
4.31.2 Function Documentation	100
4.31.2.1 dummy()	100
4.31.2.2 echo()	100
4.31.2.3 interactive_shell()	101
4.31.2.4 k_block_sigset()	101
4.31.2.5 k_unblock_sigset()	101
4.31.2.6 kill_as_process()	101
4.31.2.7 orphan_child()	101
4.31.2.8 orphanify()	102
4.31.2.9 p_boot_kernel()	102
4.31.2.10 p_exit()	102
4.31.2.11 p_getpid()	102
4.31.2.12 p_kill()	102
4.31.2.13 p_nice()	103
4.31.2.14 p_sleep()	103
4.31.2.15 p_spawn()	103
4.31.2.16 p_switch_to_scheduler()	104
4.31.2.17 p_tcset()	104
4.31.2.18 p_waitpid()	104
4.31.2.19 print_all_process_info()	104
4.31.2.20 sleeps()	104
4.31.2.21 spawn_shell()	105
4.31.2.22 W_WIFEXITED()	105
4.31.2.23 W_WIFSIGNALED()	105
4.31.2.24 W_WIFSTOPPED()	105
4.31.2.25 zombie_child()	105
4.31.2.26 zombify()	105

4.32 user.h	106
4.33 utils.c File Reference	107
4.33.1 Function Documentation	107
4.33.1.1 debug()	107
4.33.1.2 extractInt()	107
4.33.1.3 memsetter()	107
4.33.1.4 myStrNCpy()	108
4.33.1.5 print_out()	108
4.33.1.6 string_copy()	108
4.34 utils.h File Reference	108
4.34.1 Macro Definition Documentation	108
4.34.1.1 DEBUG	109
4.34.2 Function Documentation	109
4.34.2.1 debug()	109
4.34.2.2 extractInt()	109
4.34.2.3 memsetter()	109
4.34.2.4 myStrNCpy()	109
4.34.2.5 print_out()	109
4.34.2.6 string_copy()	110
4.35 utils.h	110
Index	111

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

bgNode	5
j_queue	6
Job	7
jobQueue	9
parsed_command	9
Pcb	11
queue	15
superblock	16

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

errors.c	19
errors.h	19
f.c	20
f.h	27
j_queue.c	31
j_queue.h	32
Job.c	35
Job.h	35
kernel.c	37
kernel.h	47
parser.h	54
pcb.c	57
pcb.h	59
PennFAT.c	63
PennFAT.h	73
queue.c	78
queue.h	81
shell.c	86
shell.h	90
user.c	91
user.h	98
utils.c	107
utils.h	108

Chapter 3

Data Structure Documentation

3.1 bgNode Struct Reference

Data Fields

- int [self_pid](#)
- int [self_pgid](#)
- int [jid](#)
- int [state](#)
- int [groupsize](#)
- char * [cm](#)
- struct [bgNode](#) * [next](#)

3.1.1 Field Documentation

3.1.1.1 cm

```
char* cm
```

3.1.1.2 groupsize

```
int groupsize
```

3.1.1.3 jid

```
int jid
```

3.1.1.4 next

```
struct bgNode* next
```

3.1.1.5 self_pgid

```
int self_pgid
```

3.1.1.6 self_pid

```
int self_pid
```

3.1.1.7 state

```
int state
```

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

- [PennFAT.c](#)

3.2 j_queue Struct Reference

```
#include <j_queue.h>
```

Data Fields

- struct [Job](#) * [head](#)
- struct [Job](#) * [tail](#)
- int [size](#)

3.2.1 Field Documentation

3.2.1.1 head

```
struct Job* head
```

3.2.1.2 size

```
int size
```

3.2.1.3 tail

```
struct Job* tail
```

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

- [j_queue.h](#)

3.3 Job Struct Reference

```
#include <Job.h>
```

Data Fields

- struct [parsed_command](#) * cmd
- char * [rawCmd](#)
- int [group_id](#)
- struct [Job](#) * [next](#)
- struct [Job](#) * [prev](#)
- int [job_id](#)
- enum [jobStatus](#) status
- enum [jobGround](#) ground
- int [count_finished](#)

3.3.1 Field Documentation

3.3.1.1 cmd

```
struct parsed\_command* cmd
```

3.3.1.2 count_finished

```
int count_finished
```

3.3.1.3 ground

```
enum jobGround ground
```

3.3.1.4 group_id

```
int group_id
```

3.3.1.5 job_id

```
int job_id
```

3.3.1.6 next

```
struct Job* next
```

3.3.1.7 prev

```
struct Job* prev
```

3.3.1.8 rawCmd

```
char* rawCmd
```

3.3.1.9 status

```
enum jobStatus status
```

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

- [Job.h](#)

3.4 jobQueue Struct Reference

Data Fields

- struct [bgNode](#) * [front](#)
- struct [bgNode](#) * [rear](#)
- int [q_size](#)

3.4.1 Field Documentation

3.4.1.1 front

```
struct bgNode* front
```

3.4.1.2 q_size

```
int q_size
```

3.4.1.3 rear

```
struct bgNode * rear
```

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

- [PennFAT.c](#)

3.5 parsed_command Struct Reference

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

Data Fields

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

3.5.1 Detailed Description

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

3.5.2 Field Documentation

3.5.2.1 commands

```
char** commands[ ]
```

3.5.2.2 is_background

```
bool is_background
```

3.5.2.3 is_file_append

```
bool is_file_append
```

3.5.2.4 num_commands

```
size_t num_commands
```

3.5.2.5 stdin_file

```
const char* stdin_file
```

3.5.2.6 stdout_file

```
const char* stdout_file
```

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

- [parser.h](#)

3.6 Pcb Struct Reference

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

Data Fields

- `ucontext_t * uc`
- `pid_t pid`
- `pid_t ppid`
- `pid_t pgid`
- `int priority`
- `int status`
- `int stateChangeType`
- `int fd [2]`
- `struct Pcb * next`
- `struct Pcb * prev`
- `struct Pcb * next_child`
- `struct Pcb * prev_child`
- `struct queue * children_pcb`
- `struct queue * zombies`
- `struct Pcb * parent_pcb`
- `int number_of_children`
- `int exited_child`
- `char * name`
- `bool sleeping`
- `unsigned int sleep_time`
- `bool waiting`
- `pid_t waiting_on`
- `struct parsed_command * cmd`
- `bool state_change`
- `struct Pcb * changed_child_pcb`
- `bool has_tc`
- `bool fg`
- `bool reading`

3.6.1 Field Documentation

3.6.1.1 `changed_child_pcb`

```
struct Pcb* changed_child_pcb
```

3.6.1.2 `children_pcb`

```
struct queue* children_pcb
```

3.6.1.3 cmd

```
struct parsed\_command* cmd
```

3.6.1.4 exited_child

```
int exited_child
```

3.6.1.5 fd

```
int fd[2]
```

3.6.1.6 fg

```
bool fg
```

3.6.1.7 has_tc

```
bool has_tc
```

3.6.1.8 name

```
char* name
```

3.6.1.9 next

```
struct Pcb* next
```

3.6.1.10 next_child

```
struct Pcb* next_child
```

3.6.1.11 number_of_children

```
int number_of_children
```

3.6.1.12 parent_pcb

```
struct Pcb* parent_pcb
```

3.6.1.13 pgid

```
pid_t pgid
```

3.6.1.14 pid

```
pid_t pid
```

3.6.1.15 ppid

```
pid_t ppid
```

3.6.1.16 prev

```
struct Pcb* prev
```

3.6.1.17 prev_child

```
struct Pcb* prev_child
```

3.6.1.18 priority

```
int priority
```

3.6.1.19 reading

```
bool reading
```

3.6.1.20 sleep_time

```
unsigned int sleep_time
```

3.6.1.21 sleeping

```
bool sleeping
```

3.6.1.22 state_change

```
bool state_change
```

3.6.1.23 stateChangeType

```
int stateChangeType
```

3.6.1.24 status

```
int status
```

3.6.1.25 uc

```
ucontext_t* uc
```

3.6.1.26 waiting

```
bool waiting
```

3.6.1.27 waiting_on

```
pid_t waiting_on
```

3.6.1.28 zombies

```
struct queue* zombies
```

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

- [pcb.h](#)

3.7 queue Struct Reference

```
#include <queue.h>
```

Data Fields

- struct [Pcb](#) * [head](#)
- struct [Pcb](#) * [tail](#)
- int [size](#)

3.7.1 Field Documentation

3.7.1.1 head

```
struct Pcb* head
```

3.7.1.2 size

```
int size
```

3.7.1.3 tail

```
struct Pcb* tail
```

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

- [queue.h](#)

3.8 superblock Struct Reference

```
#include <f.h>
```

Data Fields

- int [fd](#)
- int [Block_size](#)
- int [DBoffset](#)
- int [num_data_Blocks](#)
- int [num_FATBlocks](#)
- int [FAT_tablesize](#)
- int [rdsiz](#)

3.8.1 Field Documentation

3.8.1.1 Block_size

```
int Block_size
```

3.8.1.2 DBoffset

```
int DBoffset
```

3.8.1.3 FAT_tablesize

```
int FAT_tablesize
```


3.8.1.4 fd

```
int fd
```

3.8.1.5 num_data_Blocks

```
int num_data_Blocks
```

3.8.1.6 num_FATBlocks

```
int num_FATBlocks
```

3.8.1.7 rdsiz

```
int rdsiz
```

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

- [f.h](#)
- [PennFAT.h](#)

Chapter 4

File Documentation

4.1 errors.c File Reference

```
#include "errors.h"
```

Functions

- void [set_errno](#) (enum [ERRORS](#) num)

4.1.1 Function Documentation

4.1.1.1 set_errno()

```
void set_errno (  
    enum ERRORS num )
```

4.2 errors.h File Reference

Enumerations

- enum [ERRORS](#) { [DEFAULT](#) , [PID_INVALID](#) }

Functions

- void [p_perror](#) (char *str)
- void [set_errno](#) (enum [ERRORS](#) num)

4.2.1 Enumeration Type Documentation

4.2.1.1 ERRORS

```
enum ERRORS
```

Enumerator

DEFAULT	
PID_INVALID	

4.2.2 Function Documentation

4.2.2.1 p_perror()

```
void p_perror (
    char * str )
```

4.2.2.2 set_errno()

```
void set_errno (
    enum ERRORS num )
```

4.3 errors.h

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

```
00001 #ifndef ERR_H
00002 #define ERR_H
00003
00004 enum ERRORS
00005 {
00006     DEFAULT,
00007     PID_INVALID
00008 };
00009
00010 static int error_number = 0;
00011
00012 void p_perror(char *str);
00013 void set_errno(enum ERRORS num);
00014
00015 #endif
```

4.4 f.c File Reference

```
#include "f.h"
#include <fcntl.h>
#include <sys/mman.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include "parser.h"
#include <errno.h>
#include <time.h>
#include <sys/stat.h>
```

Macros

- `#define INPUT_SIZE 10000`
- `#define F_SEEK_SET 0`
- `#define F_SEEK_CUR 1`
- `#define F_SEEK_END 2`

Functions

- `bool check_allzero (char *entry)`
- `int find_available_block ()`
- `void update_fat (int cur, int block)`
- `void update_dir_entry (struct Root_directory_entry *entry, int delete)`
- `void create_newfile (char *fn)`
- `int contain_file (char *filename)`
- `void touch (char **file_name, int fsize)`
- `void rm (char *fn, int delete)`
- `void data_write (char *fn, char *data, int write_size)`
- `void chmod1 (char *filename, char *perm)`
- `int f_cat (char **cmd, int filenum)`
- `int fatfs_init (char *fatfs)`
- `int f_open (const char *fname, int mode)`
- `int f_read (int fd, int n, char *buf)`
- `int f_write (int fd, const char *str, int n)`
- `int f_close (int fd)`
- `int f_lseek (int fd, int offset, int whence)`
- `int f_unlink (const char *fname)`
- `int f_ls (const char *filename)`
- `int f_rm (char *file)`
- `int f_mv (char *source, char *dest)`
- `int f_copy (char *source, char *dest1)`
- `int f_touch (char **file_name, int fsize)`
- `int f_unmount ()`
- `int f_chmod ()`

Variables

- `struct superblock sb`
- `struct Root_directory_entry * rd`
- `int16_t * FAT_block`

4.4.1 Macro Definition Documentation

4.4.1.1 F_SEEK_CUR

```
#define F_SEEK_CUR 1
```

4.4.1.2 F_SEEK_END

```
#define F_SEEK_END 2
```

4.4.1.3 F_SEEK_SET

```
#define F_SEEK_SET 0
```

4.4.1.4 INPUT_SIZE

```
#define INPUT_SIZE 10000
```

4.4.2 Function Documentation

4.4.2.1 check_allzero()

```
bool check_allzero (  
    char * entry )
```

4.4.2.2 chmod1()

```
void chmod1 (  
    char * filename,  
    char * perm )
```

4.4.2.3 contain_file()

```
int contain_file (  
    char * filename )
```

4.4.2.4 create_newfile()

```
void create_newfile (
    char * fn )
```

f1 f2 f3 f4

4.4.2.5 data_write()

```
void data_write (
    char * fn,
    char * data,
    int write_size )
```

1 root dir->filename, firstblock first block->data: first data, FAT_table[firstblock]=nextblock use size to write.

FAT_block update

update fat block in file

appif

cur is the last block

write to data

4.4.2.6 f_cat()

```
int f_cat (
    char ** cmd,
    int filenum )
```

4.4.2.7 f_chmod()

```
int f_chmod ( )
```

4.4.2.8 f_close()

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

4.4.2.9 `f_copy()`

```
int f_copy (
    char * source,
    char * dest1 )
```

4.4.2.10 `f_ls()`

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

4.4.2.11 `f_lseek()`

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

4.4.2.12 `f_mv()`

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

4.4.2.13 `f_open()`

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

`F_WRITE`

`F_APPEND`

4.4.2.14 `f_read()`

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


4.4.2.15 f_rm()

```
int f_rm (
    char * file )
```

4.4.2.16 f_touch()

```
int f_touch (
    char ** file_name,
    int fsize )
```

check if the file already exist(check rd)

4.4.2.17 f_unlink()

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

4.4.2.18 f_unmount()

```
int f_unmount ( )
```

4.4.2.19 f_write()

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

FAT_block update

update fat block in file

write to data

4.4.2.20 fatfs_init()

```
int fatfs_init (
    char * fatfs )
```

rdsiz wrong!!

4.4.2.21 find_available_block()

```
int find_available_block ( )
```

4.4.2.22 rm()

```
void rm (
    char * fn,
    int delete )
```

4.4.2.23 touch()

```
void touch (
    char ** file_name,
    int fsize )
```

check if the file already exist(check rd)

4.4.2.24 update_dir_entry()

```
void update_dir_entry (
    struct Root_directory_entry * entry,
    int delete )
```

4.4.2.25 update_fat()

```
void update_fat (
    int cur,
    int block )
```

4.4.3 Variable Documentation

4.4.3.1 FAT_block

```
int16_t* FAT_block
```

4.4.3.2 rd

```
struct Root_directory_entry* rd
```

4.4.3.3 sb

```
struct superblock sb
```

4.5 f.h File Reference

```
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
```

Data Structures

- struct [superblock](#)

Functions

- struct [__attribute__\(\(__packed__\)\)](#) Root_directory_entry
- int [f_open](#) (const char *fname, int [mode](#))
- int [f_read](#) (int fd, int n, char *buf)
- int [f_unlink](#) (const char *fname)
- int [f_write](#) (int fd, const char *str, int n)
- int [f_close](#) (int fd)
- int [f_ls](#) (const char *filename)
- int [f_lseek](#) (int fd, int offset, int whence)
- int [f_touch](#) (char **file_name, int fsize)
- int [f_copy](#) (char *source, char *dest1)
- int [f_unmount](#) ()
- int [f_cat](#) (char **cmd, int filenum)
- int [f_rm](#) (char *file)
- int [f_mv](#) (char *source, char *dest)
- int [fatfs_init](#) (char *fatfs)

4.5.1 Function Documentation

4.5.1.1 `__attribute__()`

```
struct __attribute__ (  
    (__packed__ )
```

4.5.1.2 `f_cat()`

```
int f_cat (  
    char ** cmd,  
    int filenum )
```

4.5.1.3 `f_close()`

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

4.5.1.4 `f_copy()`

```
int f_copy (  
    char * source,  
    char * dest1 )
```

4.5.1.5 `f_ls()`

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

4.5.1.6 `f_lseek()`

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

4.5.1.7 f_mv()

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

4.5.1.8 f_open()

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

F_WRITE

F_APPEND

4.5.1.9 f_read()

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

4.5.1.10 f_rm()

```
int f_rm (
    char * file )
```

4.5.1.11 f_touch()

```
int f_touch (
    char ** file_name,
    int fsize )
```

check if the file already exist(check rd)

4.5.1.12 f_unlink()

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

4.5.1.13 f_unmount()

```
int f_unmount ( )
```

4.5.1.14 f_write()

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

FAT_block update

update fat block in file

write to data

4.5.1.15 fatfs_init()

```
int fatfs_init (
    char * fatfs )
```

rdsiz wrong!!

4.6 f.h

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

```
00001 #ifndef F_H
00002 #define F_H
00003 #include <stdio.h>
00004 #include <unistd.h>
00005 #include <ctype.h>
00006 #include <stdlib.h>
00007 #include <string.h>
00008 #include <stddef.h>
00009 #include <stdbool.h>
00010 #include <stdint.h>
00011 struct superblock {
00012     int fd;
00013     int Block_size; //total amount of blocks of virtual disk
00014     //int16_t RdirIndex; // root directory block index
00015     int DBoffset; // data block start index
00016     int num_data_Blocks; // amount of data blocks
00017     int num_FATBlocks; // number of blocks for FAT
00018     int FAT_tablesize;
00019     int rdsiz;
00020 };
00021 };
00022 struct __attribute__((__packed__)) Root_directory_entry {
00023     char name[32];
00024     uint32_t size;
00025     uint16_t firstBlock;
00026     uint8_t type;
00027     uint8_t perm;
00028     time_t mtime;
00029     char remain[16];
00030     int cursor;
00031 };
00032 };
00033
00034 int f_open(const char *fname, int mode);
```

```

00035 int f_read(int fd, int n, char *buf);
00036 int f_unlink(const char *fname);
00037 int f_write(int fd, const char *str, int n);
00038 int f_close(int fd);
00039 int f_ls(const char *filename);
00040 int f_lseek(int fd, int offset, int whence);
00041 int f_touch(char**file_name,int fsize);
00042 int f_copy(char *source, char *dest1);
00043 int f_unmount();
00044 int f_cat(char **cmd, int filenum);
00045 int f_rm(char *file);
00046 int f_mv(char *source, char *dest);
00047 int f_copy(char *source, char *dest1);
00048 int f_touch(char**file_name,int fsize);
00049 int fatfs_init(char *fatfs);
00050
00051
00052 #endif

```

4.7 j_queue.c File Reference

```

#include "j_queue.h"
#include <stdio.h>

```

Functions

- struct [j_queue](#) * [create_j_queue](#) ()
- void [push_job](#) (struct [j_queue](#) *q, struct [Job](#) *j)
- struct [Job](#) * [pop_job](#) (struct [j_queue](#) *q)
- void [free_j_queue](#) (struct [j_queue](#) *q)
- struct [Job](#) * [find_job_with_id](#) (struct [j_queue](#) *q, int [job_id](#))
- struct [Job](#) * [find_last_stopped_job](#) (struct [j_queue](#) *jobs_list)
- void [remove_job](#) (struct [j_queue](#) *q, struct [Job](#) *p)
- void [print_j_list](#) (struct [j_queue](#) *ll)

4.7.1 Function Documentation

4.7.1.1 create_j_queue()

```
struct j\_queue * create_j_queue ( )
```

4.7.1.2 find_job_with_id()

```
struct Job * find_job_with_id (
    struct j\_queue * q,
    int job\_id )
```

4.7.1.3 find_last_stopped_job()

```
struct Job * find_last_stopped_job (
    struct j_queue * jobs_list )
```

4.7.1.4 free_j_queue()

```
void free_j_queue (
    struct j_queue * q )
```

4.7.1.5 pop_job()

```
struct Job * pop_job (
    struct j_queue * q )
```

4.7.1.6 print_j_list()

```
void print_j_list (
    struct j_queue * ll )
```

4.7.1.7 push_job()

```
void push_job (
    struct j_queue * q,
    struct Job * j )
```

4.7.1.8 remove_job()

```
void remove_job (
    struct j_queue * q,
    struct Job * p )
```

4.8 j_queue.h File Reference

```
#include "Job.h"
```


Data Structures

- struct [j_queue](#)

Functions

- struct [j_queue](#) * [create_j_queue](#) ()
- void [push_job](#) (struct [j_queue](#) *q, struct [Job](#) *j)
- struct [Job](#) * [pop_job](#) (struct [j_queue](#) *q)
- void [free_j_queue](#) (struct [j_queue](#) *q)
- struct [Job](#) * [find_job_with_id](#) (struct [j_queue](#) *q, int [job_id](#))
- struct [Job](#) * [find_last_stopped_job](#) (struct [j_queue](#) *[jobs_list](#))
- void [remove_job](#) (struct [j_queue](#) *q, struct [Job](#) *p)
- void [print_j_list](#) (struct [j_queue](#) *ll)

4.8.1 Function Documentation

4.8.1.1 [create_j_queue\(\)](#)

```
struct j\_queue * create\_j\_queue ( )
```

4.8.1.2 [find_job_with_id\(\)](#)

```
struct Job * find\_job\_with\_id (  
    struct j\_queue * q,  
    int job_id )
```

4.8.1.3 [find_last_stopped_job\(\)](#)

```
struct Job * find\_last\_stopped\_job (  
    struct j\_queue * jobs_list )
```

4.8.1.4 [free_j_queue\(\)](#)

```
void free\_j\_queue (  
    struct j\_queue * q )
```

4.8.1.5 pop_job()

```
struct Job * pop_job (
    struct j_queue * q )
```

4.8.1.6 print_j_list()

```
void print_j_list (
    struct j_queue * ll )
```

4.8.1.7 push_job()

```
void push_job (
    struct j_queue * q,
    struct Job * j )
```

4.8.1.8 remove_job()

```
void remove_job (
    struct j_queue * q,
    struct Job * p )
```

4.9 j_queue.h

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

```
00001 #ifndef JQ_H
00002 #define JQ_H
00003
00004 #include "Job.h"
00005
00006 struct j_queue
00007 {
00008     struct Job *head;
00009     struct Job *tail;
00010     int size;
00011 };
00012
00013 struct j_queue *create_j_queue();
00014 void push_job(struct j_queue *q, struct Job *j);
00015 struct Job *pop_job(struct j_queue *q);
00016 void free_j_queue(struct j_queue *q);
00017 struct Job *find_job_with_id(struct j_queue *q, int job_id);
00018 struct Job *find_last_stopped_job(struct j_queue *jobs_list);
00019 void remove_job(struct j_queue *q, struct Job *p);
00020 void print_j_list(struct j_queue *ll);
00021 #endif
```

4.10 Job.c File Reference

```
#include "Job.h"
#include "utils.h"
```

Functions

- struct [Job](#) * [create_job](#) (struct [parsed_command](#) *cmd, char *rawCmd, int group_id, enum [jobStatus](#) status, enum [jobGround](#) ground)
- void [free_job](#) (struct [Job](#) *job)

4.10.1 Function Documentation

4.10.1.1 [create_job\(\)](#)

```
struct Job * create_job (
    struct parsed\_command * cmd,
    char * rawCmd,
    int group_id,
    enum jobStatus status,
    enum jobGround ground )
```

4.10.1.2 [free_job\(\)](#)

```
void free_job (
    struct Job * job )
```

4.11 Job.h File Reference

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

Data Structures

- struct [Job](#)

Enumerations

- enum [jobStatus](#) { [J_RUNNING](#) , [J_STOPPED](#) , [J_FINISHED](#) , [J_TERMINATED](#) }
- enum [jobGround](#) { [BG](#) , [FG](#) }

Functions

- struct `Job` * `create_job` (struct `parsed_command` *cmd, char *rawCmd, int group_id, enum `jobStatus` status, enum `jobGround` ground)
- void `free_job` (struct `Job` *job)

4.11.1 Enumeration Type Documentation

4.11.1.1 jobGround

enum `jobGround`

Enumerator

BG	
FG	

4.11.1.2 jobStatus

enum `jobStatus`

Enumerator

J_RUNNING	
J_STOPPED	
J_FINISHED	
J_TERMINATED	

4.11.2 Function Documentation

4.11.2.1 create_job()

```
struct Job * create_job (  
    struct parsed_command * cmd,  
    char * rawCmd,  
    int group_id,  
    enum jobStatus status,  
    enum jobGround ground )
```

4.11.2.2 free_job()

```
void free_job (
    struct Job * job )
```

4.12 Job.h

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

```
00001 #ifndef JOB_H
00002 #define JOB_H
00003
00004 #include "parser.h"
00005 #include <stdlib.h>
00006
00007 static int JOBCOUNT = 1;
00008
00009 enum jobStatus
00010 {
00011     J_RUNNING,
00012     J_STOPPED,
00013     J_FINISHED,
00014     J_TERMINATED
00015 };
00016 enum jobGround
00017 {
00018     BG,
00019     FG
00020 };
00021
00022 struct Job
00023 {
00024     struct parsed_command *cmd;
00025     char *rawCmd; // the unparsed command as a single string
00026     int group_id;
00027     struct Job *next;
00028     struct Job *prev;
00029     int job_id;
00030     enum jobStatus status;
00031     enum jobGround ground;
00032     int count_finished;
00033 };
00034
00035 struct Job *create_job(struct parsed_command *cmd,
00036                       char *rawCmd,
00037                       int group_id,
00038                       enum jobStatus status,
00039                       enum jobGround ground);
00040 void free_job(struct Job *job);
00041 #endif
```

4.13 kernel.c File Reference

```
#include "kernel.h"
#include <valgrind/valgrind.h>
```

Typedefs

- typedef struct [queue q](#)

Functions

- struct [queue](#) * [get_next_queue](#) ()
- int [get_clock_ticks](#) ()
 - Getter for clock ticks.*
- int [get_how_ended](#) ()
- void [set_how_ended](#) (int how)
- void [make_context](#) (ucontext_t *ucp, void(*func)(), int argc, char *argv[], bool link_to_scheduler)
- int [get_counter](#) ()
- void [initialise_queues](#) ()
- struct [Pcb](#) * [get_pcb](#) (pid_t pi)
- struct [queue](#) * [get_pcb_with_pgid](#) (pid_t pgid)
- void [print_all_queues_info](#) ()
- void [inc_clock](#) ()
- struct [queue](#) * [get_queue_with_priority](#) (int priority)
- void [free_queues](#) ()
- struct [Pcb](#) * [get_active_pcb](#) ()
- ucontext_t * [get_active_context](#) ()
- void [set_active_pcb_and_context](#) (struct [Pcb](#) *p)
- int [get_num_of_ready_processes](#) ()
- void [setup_log](#) ()
- FILE * [get_log_fp](#) ()
- void [setup_idle_process](#) ()
- void [idle_process](#) ()
- struct [queue](#) * [get_blocked_queue](#) ()
- struct [queue](#) * [get_next_valid_queue](#) ()
- void [scheduler](#) ()
- struct [Pcb](#) * [k_process_create](#) (struct [Pcb](#) *parent)
- void [k_process_block](#) (struct [Pcb](#) *p, struct [Pcb](#) *cp)
 - Blocks the process.*
- pid_t [k_waitpid](#) (struct [Pcb](#) *calling_pcb, struct [Pcb](#) *child_pcb, int *wstatus, pid_t pid, bool nohang)
 - Kernel side function used for waiting on a child process to terminate or stop.*
- void [k_exit](#) (struct [Pcb](#) *calling_pcb)
 - Kernel side assist function for p_exit.*
- int [k_process_kill](#) (struct [Pcb](#) *process, int signal)
 - Delivers signal to specified process.*
- void [k_process_cleanup](#) (struct [Pcb](#) *process)
 - Recursively cleanup a process's child tree when the process is terminated.*
- void [k_sleep](#) (unsigned int ticks)
 - Put the currently executing process to sleep for a specified number of timer ticks.*
- ucontext_t * [get_scheduler_context](#) (void)
- ucontext_t * [get_shell_context](#) (void)
- void [init_scheduler_context](#) ()
- void [switch_to_scheduler](#) ()
 - Saves active context and switches to scheduler context.*
- void [k_boot_kernel](#) ()
- void [k_block_sigset](#) (sigset_t *mask)
- void [k_unblock_sigset](#) (sigset_t *mask)
- int [k_tcset](#) (pid_t pid, pid_t *tc_pid)
 - Kernel side function used for setting terminal control.*

Variables

- pid_t `pid_count` = 1
- FILE * `log_fp`
- sigset_t `signal_set`

4.13.1 Typedef Documentation

4.13.1.1 q

```
typedef struct queue q
```

4.13.2 Function Documentation

4.13.2.1 free_queues()

```
void free_queues ( )
```

4.13.2.2 get_active_context()

```
ucontext_t * get_active_context ( )
```

4.13.2.3 get_active_pcb()

```
struct Pcb * get_active_pcb ( )
```

4.13.2.4 get_blocked_queue()

```
struct queue * get_blocked_queue ( )
```

4.13.2.5 `get_clock_ticks()`

```
int get_clock_ticks ( )
```

Getter for clock ticks.

4.13.2.6 `get_counter()`

```
int get_counter ( )
```

4.13.2.7 `get_how_ended()`

```
int get_how_ended ( )
```

4.13.2.8 `get_log_fp()`

```
FILE * get_log_fp ( )
```

4.13.2.9 `get_next_queue()`

```
struct queue * get_next_queue ( )
```

4.13.2.10 `get_next_valid_queue()`

```
struct queue * get_next_valid_queue ( )
```

4.13.2.11 `get_num_of_ready_processes()`

```
int get_num_of_ready_processes ( )
```


4.13.2.12 get_pcb()

```
struct Pcb * get_pcb (
    pid_t pi )
```

4.13.2.13 get_pcb_with_pgid()

```
struct queue * get_pcb_with_pgid (
    pid_t pgid )
```

4.13.2.14 get_queue_with_priority()

```
struct queue * get_queue_with_priority (
    int priority )
```

4.13.2.15 get_scheduler_context()

```
ucontext_t * get_scheduler_context (
    void )
```

4.13.2.16 get_shell_context()

```
ucontext_t * get_shell_context (
    void )
```

4.13.2.17 idle_process()

```
void idle_process ( )
```

4.13.2.18 inc_clock()

```
void inc_clock ( )
```

4.13.2.19 init_scheduler_context()

```
void init_scheduler_context ( )
```

4.13.2.20 initialise_queues()

```
void initialise_queues ( )
```

4.13.2.21 k_block_sigset()

```
void k_block_sigset (
    sigset_t * mask )
```

4.13.2.22 k_boot_kernel()

```
void k_boot_kernel ( )
```

4.13.2.23 k_exit()

```
void k_exit (
    struct Pcb * calling_pcb )
```

Kernel side assist function for p_exit.

Parameters

<i>calling_pcb</i>	PCB of the process calling p_exit
--------------------	-----------------------------------

4.13.2.24 k_process_block()

```
void k_process_block (
    struct Pcb * p,
    struct Pcb * cp )
```

Blocks the process.

Parameters

<i>p</i>	Refers to the PCB of the process to be blocked
<i>cp</i>	Optional parameter referring to child process being waited on leading to parent getting blocked

4.13.2.25 k_process_cleanup()

```
void k_process_cleanup (
    struct Pcb * process )
```

Recursively cleanup a process's child tree when the process is terminated.

Parameters

<i>process</i>	Pointer to the process's PCB
----------------	------------------------------

4.13.2.26 k_process_create()

```
struct Pcb * k_process_create (
    struct Pcb * parent )
```

4.13.2.27 k_process_kill()

```
int k_process_kill (
    struct Pcb * process,
    int signal )
```

Delivers signal to specified process.

Parameters

<i>process</i>	The process being signaled
<i>signal</i>	The kind of signal being delivered

Returns

CHANGE 0, NO CHANGE 1 where CHANGE indicates that a process changed state as a result of the signal

4.13.2.28 k_sleep()

```
void k_sleep (
    unsigned int ticks )
```

Put the currently executing process to sleep for a specified number of timer ticks.

Parameters

<i>ticks</i>	The number of timer ticks to sleep for
--------------	--

4.13.2.29 k_tcset()

```
int k_tcset (
    pid_t pid,
    pid_t * tc_pid )
```

Kernel side function used for setting terminal control.

Parameters

<i>pid</i>	PID of the process being given terminal control
<i>tc_pid</i>	Used for tracking which process (with this PID) has terminal control

Returns

0 on success and -1 on error

4.13.2.30 k_unblock_sigset()

```
void k_unblock_sigset (
    sigset_t * mask )
```

4.13.2.31 k_waitpid()

```
pid_t k_waitpid (
    struct Pcb * calling_pcb,
    struct Pcb * child_pcb,
    int * wstatus,
    pid_t pid,
    bool nohang )
```

Kernel side function used for waiting on a child process to terminate or stop.

Parameters

<i>calling_pcb</i>	Pointer to the PCB of the process that is waiting on the child process
<i>child_pcb</i>	Pointer to the PCB of the child process being waited on
<i>wstatus</i>	Pointer to an integer variable used to store the status of the child process
<i>pid</i>	PID of the child process being waited on
<i>nohang</i>	If true, return immediately if the child process is not terminated or stopped

Returns

0 if the child process is not terminated or stopped and nohang is true, or the PID of the child process if it is terminated or stopped

4.13.2.32 make_context()

```
void make_context (
    ucontext_t * ucp,
    void(*)() func,
    int argc,
    char * argv[],
    bool link_to_scheduler )
```

4.13.2.33 print_all_queues_info()

```
void print_all_queues_info ( )
```

4.13.2.34 scheduler()

```
void scheduler ( )
```

4.13.2.35 set_active_pcb_and_context()

```
void set_active_pcb_and_context (
    struct Pcb * p )
```

4.13.2.36 `set_how_ended()`

```
void set_how_ended (
    int how )
```

4.13.2.37 `setup_idle_process()`

```
void setup_idle_process ( )
```

4.13.2.38 `setup_log()`

```
void setup_log ( )
```

4.13.2.39 `switch_to_scheduler()`

```
void switch_to_scheduler ( )
```

Saves active context and switches to scheduler context.

4.13.3 Variable Documentation

4.13.3.1 `log_fp`

```
FILE* log_fp
```

4.13.3.2 `pid_count`

```
pid_t pid_count = 1
```

4.13.3.3 `signal_set`

```
sigset_t signal_set
```

4.14 kernel.h File Reference

```
#include "queue.h"
#include "user.h"
#include <stdio.h>
```

Enumerations

- enum [HOW_ENDED](#) { [TIMED_OUT](#) , [FIN_RUN](#) }

Functions

- void [idle_process](#) ()
- struct [Pcb](#) * [get_active_pcb](#) ()
- int [get_counter](#) ()
- void [boot_kernel](#) ()
- void [setup_log](#) ()
- FILE * [get_log_fp](#) ()
- int [get_num_of_ready_processes](#) ()
- struct [Pcb](#) * [k_process_create](#) (struct [Pcb](#) *parent)
- int [k_process_kill](#) (struct [Pcb](#) *process, int signal)
Delivers signal to specified process.
- void [k_process_cleanup](#) (struct [Pcb](#) *process)
Recursively cleanup a process's child tree when the process is terminated.
- void [k_sleep](#) (unsigned int ticks)
Put the currently executing process to sleep for a specified number of timer ticks.
- struct [queue](#) * [get_queue_with_priority](#) (int priority)
- struct [queue](#) * [get_blocked_queue](#) ()
- struct [Pcb](#) * [get_pcb](#) (pid_t pi)
- ucontext_t * [get_scheduler_context](#) (void)
- ucontext_t * [get_shell_context](#) (void)
- void [make_context](#) (ucontext_t *ucp, void(*func)(), int argc, char *argv[], bool link_to_scheduler)
- struct [queue](#) * [get_pcb_with_pgid](#) (pid_t pgid)
- int [get_clock_ticks](#) ()
Getter for clock ticks.
- void [switch_to_scheduler](#) ()
Saves active context and switches to scheduler context.
- void [k_process_block](#) (struct [Pcb](#) *p, struct [Pcb](#) *cp)
Blocks the process.
- void [k_exit](#) (struct [Pcb](#) *calling_pcb)
Kernel side assist function for p_exit.
- int [k_tcset](#) (pid_t pid, pid_t *tc_pid)
Kernel side function used for setting terminal control.
- pid_t [k_waitpid](#) (struct [Pcb](#) *calling_pcb, struct [Pcb](#) *child_pcb, int *wstatus, pid_t pid, bool nohang)
Kernel side function used for waiting on a child process to terminate or stop.
- void [wait_log](#) (struct [Pcb](#) *calling_pcb)
- void [k_boot_kernel](#) ()

4.14.1 Enumeration Type Documentation

4.14.1.1 HOW_ENDED

enum [HOW_ENDED](#)

Enumerator

TIMED_OUT	
FIN_RUN	

4.14.2 Function Documentation

4.14.2.1 boot_kernel()

```
void boot_kernel ( )
```

4.14.2.2 get_active_pcb()

```
struct Pcb * get_active_pcb ( )
```

4.14.2.3 get_blocked_queue()

```
struct queue * get_blocked_queue ( )
```

4.14.2.4 get_clock_ticks()

```
int get_clock_ticks ( )
```

Getter for clock ticks.

4.14.2.5 get_counter()

```
int get_counter ( )
```

4.14.2.6 get_log_fp()

```
FILE * get_log_fp ( )
```

4.14.2.7 get_num_of_ready_processes()

```
int get_num_of_ready_processes ( )
```

4.14.2.8 get_pcb()

```
struct Pcb * get_pcb (
    pid_t pi )
```

4.14.2.9 get_pcbs_with_pgid()

```
struct queue * get_pcbs_with_pgid (
    pid_t pgid )
```

4.14.2.10 get_queue_with_priority()

```
struct queue * get_queue_with_priority (
    int priority )
```

4.14.2.11 get_scheduler_context()

```
ucontext_t * get_scheduler_context (
    void )
```

4.14.2.12 get_shell_context()

```
ucontext_t * get_shell_context (
    void )
```

4.14.2.13 idle_process()

```
void idle_process ( )
```

4.14.2.14 k_boot_kernel()

```
void k_boot_kernel ( )
```

4.14.2.15 k_exit()

```
void k_exit (
    struct Pcb * calling_pcb )
```

Kernel side assist function for p_exit.

Parameters

<i>calling_pcb</i>	PCB of the process calling p_exit
--------------------	-----------------------------------

4.14.2.16 k_process_block()

```
void k_process_block (
    struct Pcb * p,
    struct Pcb * cp )
```

Blocks the process.

Parameters

<i>p</i>	Refers to the PCB of the process to be blocked
<i>cp</i>	Optional parameter referring to child process being waited on leading to parent getting blocked

4.14.2.17 k_process_cleanup()

```
void k_process_cleanup (
    struct Pcb * process )
```

Recursively cleanup a process's child tree when the process is terminated.

Parameters

<i>process</i>	Pointer to the process's PCB
----------------	------------------------------

4.14.2.18 k_process_create()

```
struct Pcb * k_process_create (
    struct Pcb * parent )
```

4.14.2.19 k_process_kill()

```
int k_process_kill (
    struct Pcb * process,
    int signal )
```

Delivers signal to specified process.

Parameters

<i>process</i>	The process being signaled
<i>signal</i>	The kind of signal being delivered

Returns

CHANGE 0, NO CHANGE 1 where CHANGE indicates that a process changed state as a result of the signal

4.14.2.20 k_sleep()

```
void k_sleep (
    unsigned int ticks )
```

Put the currently executing process to sleep for a specified number of timer ticks.

Parameters

<i>ticks</i>	The number of timer ticks to sleep for
--------------	--

4.14.2.21 k_tcset()

```
int k_tcset (
    pid_t pid,
    pid_t * tc_pid )
```

Kernel side function used for setting terminal control.

Parameters

<i>pid</i>	PID of the process being given terminal control
<i>tc_pid</i>	Used for tracking which process (with this PID) has terminal control

Returns

0 on success and -1 on error

4.14.2.22 k_waitpid()

```
pid_t k_waitpid (
    struct Pcb * calling_pcb,
    struct Pcb * child_pcb,
    int * wstatus,
    pid_t pid,
    bool nohang )
```

Kernel side function used for waiting on a child process to terminate or stop.

Parameters

<i>calling_pcb</i>	Pointer to the PCB of the process that is waiting on the child process
<i>child_pcb</i>	Pointer to the PCB of the child process being waited on
<i>wstatus</i>	Pointer to an integer variable used to store the status of the child process
<i>pid</i>	PID of the child process being waited on
<i>nohang</i>	If true, return immediately if the child process is not terminated or stopped

Returns

0 if the child process is not terminated or stopped and nohang is true, or the PID of the child process if it is terminated or stopped

4.14.2.23 make_context()

```
void make_context (
    ucontext_t * ucp,
    void(*)() func,
    int argc,
    char * argv[],
    bool link_to_scheduler )
```

4.14.2.24 setup_log()

```
void setup_log ( )
```

4.14.2.25 switch_to_scheduler()

```
void switch_to_scheduler ( )
```

Saves active context and switches to scheduler context.

4.14.2.26 wait_log()

```
void wait_log (
    struct Pcb * calling_pcb )
```

4.15 kernel.h

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

```
00001 #ifndef KER_H
00002 #define KER_H
00003 #include "queue.h"
00004 #include "user.h"
00005 #include <stdio.h>
00006 enum HOW_ENDED {
00007     TIMED_OUT,
00008     FIN_RUN
00009 };
00010
00011
00012 static sigset_t intmask;
00013 void idle_process();
00014 struct Pcb *get_active_pcb();
00015 int get_counter();
00016 void boot_kernel();
00017 void setup_log();
00018 FILE *get_log_fp();
00019 int get_num_of_ready_processes();
00020 struct Pcb *k_process_create(struct Pcb *parent);
00021
```

```

00028 int k_process_kill(struct Pcb *process, int signal);
00029
00034 void k_process_cleanup(struct Pcb *process);
00035
00040 void k_sleep(unsigned int ticks);
00041 struct queue *get_queue_with_priority(int priority);
00042 struct queue *get_blocked_queue();
00043 struct Pcb *get_pcb(pid_t pi); // Check all queues for the pcb with given pid
00044
00045 ucontext_t *get_scheduler_context(void);
00046
00047 ucontext_t *get_shell_context(void);
00048 // void initialise_queues();
00049 // void setup_idle_process();
00050 // void init_scheduler_context();
00051
00052
00053 void make_context(ucontext_t *ucp, void (*func)(), int argc, char *argv[], bool link_to_scheduler);
00054 struct queue *get_pcb_with_pgid(pid_t pgid);
00055
00059 int get_clock_ticks();
00060
00064 void switch_to_scheduler();
00065
00066
00067
00068
00074 void k_process_block(struct Pcb *p, struct Pcb *cp);
00075
00080 void k_exit(struct Pcb *calling_pcb);
00081
00088 int k_tcset(pid_t pid, pid_t *tc_pid);
00089
00099 pid_t k_waitpid(struct Pcb *calling_pcb, struct Pcb *child_pcb, int *wstatus, pid_t pid, bool nohang);
00100
00101
00102 void wait_log(struct Pcb *calling_pcb);
00103 static pid_t current_pid = 0;
00104
00105 void k_boot_kernel();
00106 #endif

```

4.16 parser.h File Reference

```

#include <stddef.h>
#include <stdbool.h>

```

Data Structures

- struct [parsed_command](#)

Macros

- #define [UNEXPECTED_FILE_INPUT](#) 1
- #define [UNEXPECTED_FILE_OUTPUT](#) 2
- #define [UNEXPECTED_PIPELINE](#) 3
- #define [UNEXPECTED_AMPERSAND](#) 4
- #define [EXPECT_INPUT_FILENAME](#) 5
- #define [EXPECT_OUTPUT_FILENAME](#) 6
- #define [EXPECT_COMMANDS](#) 7

Functions

- int [parse_command](#) (const char *cmd_line, struct [parsed_command](#) **result)
- void [print_parsed_command](#) (const struct [parsed_command](#) *cmd)

4.16.1 Macro Definition Documentation

4.16.1.1 EXPECT_COMMANDS

```
#define EXPECT_COMMANDS 7
```

4.16.1.2 EXPECT_INPUT_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

4.16.1.3 EXPECT_OUTPUT_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

4.16.1.4 UNEXPECTED_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

4.16.1.5 UNEXPECTED_FILE_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

4.16.1.6 UNEXPECTED_FILE_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

4.16.1.7 UNEXPECTED_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

4.16.2 Function Documentation

4.16.2.1 parse_command()

```
int parse_command (
    const char * cmd_line,
    struct parsed_command ** result )
```

Arguments: `cmd_line`: a null-terminated string that is the command line `result`: a non-null pointer to a `struct parsed_command *`

Return value (int): an error code which can be, 0: parser finished succesfully -1: parser encountered a system call error 1-7: parser specific error, see error type above

This function will parse the given `cmd_line` and store the parsed information into a `struct parsed_command`. The memory needed for the struct will be allocated by this function, and the pointer to the memory will be stored into the given `*result`.

You can directly use the result in system calls. See demo for more information.

If the function returns a successful value (0), a `struct parsed_command` is guaranteed to be allocated and stored in the given `*result`. It is the caller's responsibility to free the given pointer using `free(3)`.

Otherwise, no `struct parsed_command` is allocated and `*result` is unchanged. If a system call error (-1) is returned, the caller can use `errno(3)` or `perror(3)` to gain more information about the error.

4.16.2.2 print_parsed_command()

```
void print_parsed_command (
    const struct parsed_command * cmd )
```

4.17 parser.h

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

```
00001 /* Penn-Shell Parser
00002     hanbangw, 21fa */
00003
00004 #pragma once
00005
00006 #include <stddef.h>
00007 #include <stdbool.h>
00008
00009 /* Here defines all possible parser errors */
00010 // parser encountered an unexpected file input token '<'
00011 #define UNEXPECTED_FILE_INPUT 1
00012
00013 // parser encountered an unexpected file output token '>'
00014 #define UNEXPECTED_FILE_OUTPUT 2
00015
00016 // parser encountered an unexpected pipeline token '|'
00017 #define UNEXPECTED_PIPELINE 3
00018
00019 // parser encountered an unexpected ampersand token '&'
00020 #define UNEXPECTED_AMPERSAND 4
00021
00022 // parser didn't find input filename following '<'
00023 #define EXPECT_INPUT_FILENAME 5
00024
```



```

00025 // parser didn't find output filename following '>' or '»'
00026 #define EXPECT_OUTPUT_FILENAME 6
00027
00028 // parser didn't find any commands or arguments where it expects one
00029 #define EXPECT_COMMANDS 7
00030
00035 struct parsed_command {
00036     // indicates the command shall be executed in background
00037     // (ends with an ampersand '&')
00038     bool is_background;
00039
00040     // indicates if the stdout_file shall be opened in append mode
00041     // ignore this value when stdout_file is NULL
00042     bool is_file_append;
00043
00044     // filename for redirecting input from
00045     const char *stdin_file;
00046
00047     // filename for redirecting output to
00048     const char *stdout_file;
00049
00050     // number of commands (pipeline stages)
00051     size_t num_commands;
00052
00053     // an array to a list of arguments
00054     // size of 'commands' is 'num_commands'
00055     char **commands[];
00056 };
00057
00083 int parse_command(const char *cmd_line, struct parsed_command **result);
00084
00085
00086 /* This is a debugging function used for outputting a parsed command line. */
00087 void print_parsed_command(const struct parsed_command *cmd);

```

4.18 pcb.c File Reference

```

#include "pcb.h"
#include <stdio.h>

```

Functions

- struct `Pcb` * `create_pcb` (ucontext_t *uc, pid_t pid, pid_t ppid, pid_t pgid, int priority, int status, struct `parsed_command` *cmd)
- struct `Pcb` * `create_pcb_no_context` (pid_t pid, pid_t ppid, pid_t pgid, int priority, int status)
- void `free_pcb` (struct `Pcb` *p)
- void `add_child_to_parent_q` (struct `Pcb` *parent_pcb, struct `Pcb` *child_pcb)
- void `remove_child_pcb` (struct `Pcb` *parent_pcb, struct `Pcb` *child_pcb)
Removes a child process from the list of children of a parent process but does not free the memory.
- void `delete_child_pcb` (struct `Pcb` *parent_pcb, struct `Pcb` *child_pcb)
Removes a child process from the list of children of a parent process and frees the associated memory.
- void `add_child_to_parent_zombies` (struct `Pcb` *parent_pcb, struct `Pcb` *child_pcb)
Adds a child process's PCB to a parent process's zombie queue.

4.18.1 Function Documentation

4.18.1.1 add_child_to_parent_q()

```
void add_child_to_parent_q (
    struct Pcb * parent_pcb,
    struct Pcb * child_pcb )
```

4.18.1.2 add_child_to_parent_zombies()

```
void add_child_to_parent_zombies (
    struct Pcb * parent_pcb,
    struct Pcb * child_pcb )
```

Adds a child process's PCB to a parent process's zombie queue.

Parameters

<i>parent_pcb</i>	Pointer to the parent process's PCB
<i>child_pcb</i>	Pointer to the child process's PCB

4.18.1.3 create_pcb()

```
struct Pcb * create_pcb (
    ucontext_t * uc,
    pid_t pid,
    pid_t ppid,
    pid_t pgid,
    int priority,
    int status,
    struct parsed_command * cmd )
```

4.18.1.4 create_pcb_no_context()

```
struct Pcb * create_pcb_no_context (
    pid_t pid,
    pid_t ppid,
    pid_t pgid,
    int priority,
    int status )
```

4.18.1.5 delete_child_pcb()

```
void delete_child_pcb (
    struct Pcb * parent_pcb,
    struct Pcb * child_pcb )
```

Removes a child process from the list of children of a parent process and frees the associated memory.

Parameters

<i>parent_pcb</i>	Pointer to the parent process's PCB
<i>child_pcb</i>	Pointer to the child process's PCB

4.18.1.6 free_pcb()

```
void free_pcb (
    struct Pcb * p )
```

4.18.1.7 remove_child_pcb()

```
void remove_child_pcb (
    struct Pcb * parent_pcb,
    struct Pcb * child_pcb )
```

Removes a child process from the list of children of a parent process but does not free the memory.

Parameters

<i>parent_pcb</i>	Pointer to the parent process's PCB
<i>child_pcb</i>	Pointer to the child process's PCB

4.19 pcb.h File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <ucontext.h>
#include <string.h>
#include <stdbool.h>
#include "parser.h"
#include "queue.h"
```

Data Structures

- struct [Pcb](#)

Enumerations

- enum [status](#) {
 [READY](#) , [RUNNING](#) , [BLOCKED](#) , [STOPPED](#) ,
 [TERMINATED](#) , [FINISHED](#) , [ZOMBIE](#) , [ORPHAN](#) }

Functions

- struct `Pcb` * `create_pcb` (ucontext_t *uc, pid_t `pid`, pid_t ppid, pid_t pgid, int priority, int `status`, struct `parsed_command` *cmd)
- struct `Pcb` * `create_pcb_no_context` (pid_t `pid`, pid_t ppid, pid_t pgid, int priority, int `status`)
- void `free_pcb` (struct `Pcb` *pcb)
- void `add_child_to_parent_q` (struct `Pcb` *parent_pcb, struct `Pcb` *child_pcb)
- void `remove_child_pcb` (struct `Pcb` *parent_pcb, struct `Pcb` *child_pcb)
Removes a child process from the list of children of a parent process but does not free the memory.
- void `delete_child_pcb` (struct `Pcb` *parent_pcb, struct `Pcb` *child_pcb)
Removes a child process from the list of children of a parent process and frees the associated memory.
- void `add_child_to_parent_zombies` (struct `Pcb` *parent_pcb, struct `Pcb` *child_pcb)
Adds a child process's PCB to a parent process's zombie queue.

4.19.1 Enumeration Type Documentation

4.19.1.1 status

enum `status`

Enumerator

READY	
RUNNING	
BLOCKED	
STOPPED	
TERMINATED	
FINISHED	
ZOMBIE	
ORPHAN	

4.19.2 Function Documentation

4.19.2.1 add_child_to_parent_q()

```
void add_child_to_parent_q (
    struct Pcb * parent_pcb,
    struct Pcb * child_pcb )
```

4.19.2.2 add_child_to_parent_zombies()

```
void add_child_to_parent_zombies (
    struct Pcb * parent_pcb,
    struct Pcb * child_pcb )
```

Adds a child process's PCB to a parent process's zombie queue.

Parameters

<i>parent_pcb</i>	Pointer to the parent process's PCB
<i>child_pcb</i>	Pointer to the child process's PCB

4.19.2.3 create_pcb()

```
struct Pcb * create_pcb (
    ucontext_t * uc,
    pid_t pid,
    pid_t ppid,
    pid_t pgid,
    int priority,
    int status,
    struct parsed_command * cmd )
```

4.19.2.4 create_pcb_no_context()

```
struct Pcb * create_pcb_no_context (
    pid_t pid,
    pid_t ppid,
    pid_t pgid,
    int priority,
    int status )
```

4.19.2.5 delete_child_pcb()

```
void delete_child_pcb (
    struct Pcb * parent_pcb,
    struct Pcb * child_pcb )
```

Removes a child process from the list of children of a parent process and frees the associated memory.

Parameters

<i>parent_pcb</i>	Pointer to the parent process's PCB
<i>child_pcb</i>	Pointer to the child process's PCB

4.19.2.6 free_pcb()

```
void free_pcb (
    struct Pcb * pcb )
```

4.19.2.7 remove_child_pcb()

```
void remove_child_pcb (
    struct Pcb * parent_pcb,
    struct Pcb * child_pcb )
```

Removes a child process from the list of children of a parent process but does not free the memory.

Parameters

<i>parent_pcb</i>	Pointer to the parent process's PCB
<i>child_pcb</i>	Pointer to the child process's PCB

4.20 pcb.h

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

```
00001 #ifndef PCB_H
00002 #define PCB_H
00003
00004 #include <unistd.h>
00005 #include <stdlib.h>
00006 #include <ucontext.h>
00007 #include <string.h>
00008 #include <stdbool.h>
00009 #include "parser.h"
00010 #include "queue.h"
00011
00012 enum status {
00013     READY,
00014     RUNNING,
00015     BLOCKED,
00016     STOPPED,
00017     TERMINATED,
00018     FINISHED,
00019     ZOMBIE,
00020     ORPHAN
00021 };
00022
00023 struct Pcb
00024 {
00025     ucontext_t *uc;
00026     pid_t pid;
00027     pid_t ppid;
00028     pid_t pgid;
00029     int priority;
00030     int status;
00031     int stateChangeType; // 0 if no change; 1 if terminated normally; 2 if stopped; 3 if terminated by
    a signal; 4 if continued
00032     int fd[2];
00033     struct Pcb *next;
00034     struct Pcb *prev;
00035     struct Pcb *next_child;
00036     struct Pcb *prev_child;
00037     struct queue *children_pcb;
```

```

00038     struct queue *zombies;
00039     struct Pcb *parent_pcb;
00040     int number_of_children;
00041     int exited_child;
00042     char *name;
00043     bool sleeping;
00044     unsigned int sleep_time;
00045     bool waiting;
00046     pid_t waiting_on;
00047     struct parsed_command *cmd;
00048     bool state_change;
00049     struct Pcb *changed_child_pcb;
00050     bool has_tc;
00051     bool fg;
00052     bool reading;
00053 };
00054
00055 struct Pcb *create_pcb(ucontext_t *uc, pid_t pid, pid_t ppid, pid_t pgid, int priority, int status,
    struct parsed_command *cmd);
00056 struct Pcb *create_pcb_no_context(pid_t pid, pid_t ppid, pid_t pgid, int priority, int status);
00057 void free_pcb(struct Pcb *pcb);
00058 void add_child_to_parent_q(struct Pcb *parent_pcb, struct Pcb *child_pcb);
00059
00065 void remove_child_pcb(struct Pcb *parent_pcb, struct Pcb *child_pcb);
00066
00072 void delete_child_pcb(struct Pcb *parent_pcb, struct Pcb *child_pcb);
00073
00079 void add_child_to_parent_zombies(struct Pcb *parent_pcb, struct Pcb *child_pcb);
00080
00081
00082 #endif

```

4.21 PennFAT.c File Reference

```

#include "PennFAT.h"
#include <fcntl.h>
#include <sys/mman.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include "parser.h"
#include <errno.h>
#include <time.h>
#include <sys/stat.h>

```

Data Structures

- struct [bgNode](#)
- struct [jobQueue](#)

Macros

- #define [INPUT_SIZE](#) 10000

Functions

- void `mkfs` (char *fs_name, int blocks_in_fat, int block_size_config)
- bool `check_allzero` (char *entry)
- void `mount` (char *fs_name)
- void `unmount` ()
- int `contain_file` (char *filename)
- void `create_newfile` (char *fn)
- void `touch` (char **file_name, int fsize)
- void `update_dir_entry` (struct Root_directory_entry *entry, int delete)
- void `print_status` ()
- int `find_available_block` ()
- void `rm` (char *fn, int delete)
- void `update_fat` (int cur, int block)
- void `data_write` (char *fn, char *data, int write_size)
- void `cat_a` (char **cmd, int filenum, int out)
- void `cat_file` (char **cmd, int filenum, int out)
- void `cat_w` (char **cmd, int filenum, int out)
- void `copy` (char *source, char *dest1, bool source_os, bool dest_os)
- void `cat` (char **cm, int filenum)
- void `ls` ()
- void `chmod1` (char *filename, char *perm)
- void `mv` (char *filename, char *name)
- char * `stringcpy` (char *destination, const char *source)
- struct `bgNode` * `creatNode` (int p, int pg, char *c, int a, int state)
- struct `jobQueue` * `createQueue` ()
- void `enQueue` (struct `jobQueue` *q, int p, int pg, char *c, int a, int state)
- void `deQueue` (struct `jobQueue` *q)
- void `removeQueue` (struct `jobQueue` *q, int pg)
- void `freeq` (struct `jobQueue` *q)
- void `sig_handler` (int signo)
- void `INTHandler` (int signo)
- void `sigsttp_handler` (int sig)
- int `check_finished` (struct `jobQueue` *q, int asyn)
- void `check_redirection` (struct `parsed_command` *cmd)
- struct `bgNode` * `search_byjid` (int j, struct `jobQueue` *q)
- void `bg` (char **comd, struct `jobQueue` *q)
- void `fg` (char **comd, struct `jobQueue` *q)
- void `jobs` (struct `jobQueue` *q)
- *jobs no complete!!!!*
- void `check_current` (struct `jobQueue` *q)
- void `sigchld_handler` (int sig)
- int `main` (int argc, char *argv[])

Variables

- struct `superblock` sb
- struct `Root_directory_entry` * rd
- int16_t * `FAT_block`
- int c = 0
- int pid
- int pid1
- int jobid = 0
- struct `bgNode` * currentjob
- int back
- struct `jobQueue` * gq

4.21.1 Macro Definition Documentation

4.21.1.1 INPUT_SIZE

```
#define INPUT_SIZE 10000
```

4.21.2 Function Documentation

4.21.2.1 bg()

```
void bg (
    char ** cmd,
    struct jobQueue * q )
```

4.21.2.2 cat()

```
void cat (
    char ** cm,
    int filenum )
```

4.21.2.3 cat_a()

```
void cat_a (
    char ** cmd,
    int filenum,
    int out )
```

4.21.2.4 cat_file()

```
void cat_file (
    char ** cmd,
    int filenum,
    int out )
```

start to print

4.21.2.5 cat_w()

```
void cat_w (
    char ** cmd,
    int filenum,
    int out )
```

4.21.2.6 check_allzero()

```
bool check_allzero (
    char * entry )
```

4.21.2.7 check_current()

```
void check_current (
    struct jobQueue * q )
```

4.21.2.8 check_finished()

```
int check_finished (
    struct jobQueue * q,
    int asyn )
```

4.21.2.9 check_redirection()

```
void check_redirection (
    struct parsed_command * cmd )
```

4.21.2.10 chmod1()

```
void chmod1 (
    char * filename,
    char * perm )
```

4.21.2.11 contain_file()

```
int contain_file (
    char * filename )
```

4.21.2.12 copy()

```
void copy (
    char * source,
    char * dest1,
    bool source_os,
    bool dest_os )
```

4.21.2.13 create_newfile()

```
void create_newfile (
    char * fn )
```

f1 f2 f3 f4

4.21.2.14 createQueue()

```
struct jobQueue * createQueue ( )
```

4.21.2.15 creatNode()

```
struct bgNode * creatNode (
    int p,
    int pg,
    char * c,
    int a,
    int state )
```

4.21.2.16 data_write()

```
void data_write (
    char * fn,
    char * data,
    int write_size )
```

1 root dir->filename, firstblock first block->data:first data, FAT_table[firstblock]=nextblock use size to write.

FAT_block update

update fat block in file

appif

cur is the last block

write to data

4.21.2.17 deQueue()

```
void deQueue (
    struct jobQueue * q )
```

4.21.2.18 enQueue()

```
void enQueue (
    struct jobQueue * q,
    int p,
    int pg,
    char * c,
    int a,
    int state )
```

4.21.2.19 fg()

```
void fg (
    char ** cmd,
    struct jobQueue * q )
```

4.21.2.20 find_available_block()

```
int find_available_block ( )
```

4.21.2.21 freeq()

```
void freeq (
    struct jobQueue * q )
```

4.21.2.22 INThandler()

```
void INThandler (
    int signo )
```

4.21.2.23 jobs()

```
void jobs (
    struct jobQueue * q )
```

jobs no complete!!!!

4.21.2.24 ls()

```
void ls ( )
```

4.21.2.25 main()

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

not first time

not last time

4.21.2.26 mkfs()

```
void mkfs (
    char * fs_name,
    int blocks_in_fat,
    int block_size_config )
```

4.21.2.27 mount()

```
void mount (
    char * fs_name )
```

rdsiz wrong!!

4.21.2.28 mv()

```
void mv (
    char * filename,
    char * name )
```

4.21.2.29 print_status()

```
void print_status ( )
```

4.21.2.30 removeQueue()

```
void removeQueue (
    struct jobQueue * q,
    int pg )
```

4.21.2.31 rm()

```
void rm (
    char * fn,
    int delete )
```

4.21.2.32 search_byjid()

```
struct bgNode * search_byjid (
    int j,
    struct jobQueue * q )
```

4.21.2.33 sig_handler()

```
void sig_handler (
    int signo )
```

4.21.2.34 sigchld_handler()

```
void sigchld_handler (
    int sig )
```

4.21.2.35 sigtstp_handler()

```
void sigtstp_handler (
    int sig )
```

4.21.2.36 strcpy()

```
char * strcpy (
    char * destination,
    const char * source )
```

4.21.2.37 touch()

```
void touch (
    char ** file_name,
    int fsize )
```

check if the file already exist(check rd)

4.21.2.38 unmount()

```
void unmount ( )
```

4.21.2.39 update_dir_entry()

```
void update_dir_entry (
    struct Root_directory_entry * entry,
    int delete )
```

3 FAT[1]=2 FAT[2]=3 FAT[3]=-1

4.21.2.40 update_fat()

```
void update_fat (
    int cur,
    int block )
```

4.21.3 Variable Documentation

4.21.3.1 back

```
int back
```

4.21.3.2 c

```
int c = 0
```

4.21.3.3 currentjob

```
struct bgNode* currentjob
```

4.21.3.4 FAT_block

```
int16_t* FAT_block
```

4.21.3.5 gq

```
struct jobQueue* gq
```

4.21.3.6 jobid

```
int jobid = 0
```

4.21.3.7 pid

```
int pid
```

4.21.3.8 pid1

```
int pid1
```

4.21.3.9 rd

```
struct Root_directory_entry* rd
```


4.21.3.10 sb

```
struct superblock sb
```

4.22 PennFAT.h File Reference

```
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
```

Data Structures

- struct [superblock](#)

Functions

- struct [__attribute__](#) ((__packed__)) Root_directory_entry
- void [mkfs](#) (char *fs_name, int blocks_in_fat, int block_size_config)
- void [update_dir_entry](#) (struct Root_directory_entry *entry, int delete)
- void [update_fat](#) (int cur, int block)
- void [mount](#) (char *fs_name)
- void [unmount](#) ()
- void [touch](#) (char **file_name, int fsize)
- void [data_write](#) (char *fn, char *data, int write_size)
- void [copy](#) (char *source, char *dest1, bool source_os, bool dest_os)
- void [cat](#) (char **cm, int filenum)
- void [ls](#) ()
- void [chmod1](#) (char *filename, char *perm)
- void [mv](#) (char *filename, char *name)
- void [print_status](#) ()
- void [rm](#) (char *fn, int delete)
- int [contain_file](#) (char *filename)
- bool [check_allzero](#) (char *entry)
- int [find_available_block](#) ()

4.22.1 Function Documentation

4.22.1.1 `__attribute__()`

```
struct __attribute__ (  
    (__packed__)
```

4.22.1.2 `cat()`

```
void cat (  
    char ** cm,  
    int filenum )
```

4.22.1.3 `check_allzero()`

```
bool check_allzero (  
    char * entry )
```

4.22.1.4 `chmod1()`

```
void chmod1 (  
    char * filename,  
    char * perm )
```

4.22.1.5 `contain_file()`

```
int contain_file (  
    char * filename )
```

4.22.1.6 `copy()`

```
void copy (  
    char * source,  
    char * dest1,  
    bool source_os,  
    bool dest_os )
```

4.22.1.7 data_write()

```
void data_write (
    char * fn,
    char * data,
    int write_size )
```

1 root dir->filename, firstblock first block->data: first data, FAT_table[firstblock]=nextblock use size to write.

FAT_block update

update fat block in file

appif

cur is the last block

write to data

1 root dir->filename, firstblock first block->data: first data, FAT_table[firstblock]=nextblock use size to write.

FAT_block update

update fat block in file

appif

cur is the last block

write to data

4.22.1.8 find_available_block()

```
int find_available_block ( )
```

4.22.1.9 ls()

```
void ls ( )
```

4.22.1.10 mkfs()

```
void mkfs (
    char * fs_name,
    int blocks_in_fat,
    int block_size_config )
```

4.22.1.11 mount()

```
void mount (
    char * fs_name )
```

rdsiz wrong!!

4.22.1.12 mv()

```
void mv (
    char * filename,
    char * name )
```

4.22.1.13 print_status()

```
void print_status ( )
```

4.22.1.14 rm()

```
void rm (
    char * fn,
    int delete )
```

4.22.1.15 touch()

```
void touch (
    char ** file_name,
    int fsize )
```

check if the file already exist(check rd)

check if the file already exist(check rd)

4.22.1.16 unmount()

```
void unmount ( )
```

4.22.1.17 update_dir_entry()

```
void update_dir_entry (
    struct Root_directory_entry * entry,
    int delete )
```

3 FAT[1]=2 FAT[2]=3 FAT[3]=-1

4.22.1.18 update_fat()

```
void update_fat (
    int cur,
    int block )
```

4.23 PennFAT.h

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

```
00001 #ifndef PENNFAT_H
00002 #define PENNFAT_H
00003
00004 #include <stdio.h>
00005 #include <unistd.h>
00006 #include <ctype.h>
00007 #include <stdlib.h>
00008 #include <string.h>
00009 #include <stddef.h>
00010 #include <stdbool.h>
00011 #include <stdint.h>
00012
00013
00014 struct superblock {
00015     int fd;
00016     int Block_size; //total amount of blocks of virtual disk
00017     //int16_t RdirIndex; // root directory block index
00018     int DBoffset; // data block start index
00019     int num_data_Blocks; // amount of data blocks
00020     int num_FATBlocks; // number of blocks for FAT
00021     int FAT_tablesize;
00022     int rdsiz;
00023 };
00024
00025 struct __attribute__((__packed__)) Root_directory_entry {
00026     char name[32];
00027     uint32_t size;
00028     uint16_t firstBlock;
00029     uint8_t type;
00030     uint8_t perm;
00031     time_t mtime;
00032     char remain[16];
00033     int cursor;
00034 };
00035
00036
00037 void mkfs(char *fs_name, int blocks_in_fat,int block_size_config);
00038 void update_dir_entry(struct Root_directory_entry *entry, int delete);
00039 void update_fat(int cur,int block);
00040 void mount(char *fs_name);
00041 void unmount();
00042 void touch(char **file_name, int fsize);
00043 void data_write(char *fn, char *data, int write_size);
00044 void copy(char *source, char *dest1, bool source_os, bool dest_os);
00045 void cat(char **cm, int filenum);
00046 void ls();
00047 void chmod1(char *filename, char *perm);
00048 void mv(char *filename, char *name);
00049 void print_status();
00050 void rm(char *fn, int delete);
00051 int contain_file(char *filename);
00052 bool check_allzero(char *entry);
00053 int find_available_block();
00054
00055
00056 #endif
00057
```

4.24 queue.c File Reference

```
#include "queue.h"
#include <stdio.h>
#include "kernel.h"
```

Functions

- struct [queue](#) * [create_queue](#) ()
Allocates memory for a queue of PCBs.
- void [push](#) (struct [queue](#) *q, struct [Pcb](#) *new_pcb)
- void [push_to_front](#) (struct [queue](#) *q, struct [Pcb](#) *new_pcb)
- struct [Pcb](#) * [pop](#) (struct [queue](#) *q)
Pops and returns the [Pcb](#) struct pointer of the process at the front of the queue.
- struct [Pcb](#) * [pop_child](#) (struct [queue](#) *q)
- void [print_queue](#) (struct [queue](#) *q)
- void [print_queue_info](#) (struct [queue](#) *q)
- void [free_queue](#) (struct [queue](#) *q)
Frees all PCBs in the queue.
- void [free_child_queue](#) (struct [queue](#) *q)
Frees all PCBs in the children queue.
- struct [Pcb](#) * [front](#) (struct [queue](#) *q)
Returns pointer to front element.
- bool [is_empty](#) (struct [queue](#) *q)
- struct [Pcb](#) * [get_pcb_with_pid](#) (struct [queue](#) *q, pid_t pid)
- void [get_pcb_with_pgid_in_queue](#) (struct [queue](#) *q, pid_t pgid, struct [queue](#) *result)
- void [remove_pcb](#) (struct [queue](#) *q, struct [Pcb](#) *p)
Removes process from the queue but does not free the memory.
- struct [Pcb](#) * [get_child_pcb_with_pid](#) (struct [queue](#) *q, pid_t pid)
Looks for the PCB in children queue of parent PCB.

4.24.1 Function Documentation

4.24.1.1 create_queue()

```
struct queue * create_queue ( )
```

Allocates memory for a queue of PCBs.

Returns

Returns the pointer to the queue struct on success and NULL on failure

4.24.1.2 free_child_queue()

```
void free_child_queue (
    struct queue * q )
```

Frees all PCBs in the children queue.

Parameters

<i>q</i>	Reference to queue
----------	--------------------

4.24.1.3 free_queue()

```
void free_queue (
    struct queue * q )
```

Frees all PCBs in the queue.

Parameters

<i>q</i>	Reference to queue
----------	--------------------

4.24.1.4 front()

```
struct Pcb * front (
    struct queue * q )
```

Returns pointer to front element.

Parameters

<i>q</i>	Reference to queue
----------	--------------------

4.24.1.5 get_child_pcb_with_pid()

```
struct Pcb * get_child_pcb_with_pid (
    struct queue * q,
    pid_t pid )
```

Looks for the PCB in children queue of parent PCB.

Parameters

<i>q</i>	Children queue
<i>pid</i>	PID of child process

4.24.1.6 get_pcb_with_pid()

```
struct Pcb * get_pcb_with_pid (
    struct queue * q,
    pid_t pid )
```

4.24.1.7 get_pcbs_with_pgid_in_queue()

```
void get_pcbs_with_pgid_in_queue (
    struct queue * q,
    pid_t pgid,
    struct queue * result )
```

4.24.1.8 is_empty()

```
bool is_empty (
    struct queue * q )
```

4.24.1.9 pop()

```
struct Pcb * pop (
    struct queue * q )
```

Pops and returns the [Pcb](#) struct pointer of the process at the front of the queue.

Parameters

<i>q</i>	Queue of PCBs
----------	---------------

4.24.1.10 pop_child()

```
struct Pcb * pop_child (
    struct queue * q )
```

4.24.1.11 print_queue()

```
void print_queue (
    struct queue * q )
```


4.24.1.12 print_queue_info()

```
void print_queue_info (
    struct queue * q )
```

4.24.1.13 push()

```
void push (
    struct queue * q,
    struct Pcb * new_pcb )
```

4.24.1.14 push_to_front()

```
void push_to_front (
    struct queue * q,
    struct Pcb * new_pcb )
```

4.24.1.15 remove_pcb()

```
void remove_pcb (
    struct queue * q,
    struct Pcb * p )
```

Removes process from the queue but does not free the memory.

Parameters

<i>q</i>	Queue from which process is to be removed
<i>p</i>	Pointer to the PCB of the process which is to be removed from queue q

4.25 queue.h File Reference

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

Data Structures

- struct queue

Functions

- struct `queue` * `create_queue` ()
Allocates memory for a queue of PCBs.
- void `push` (struct `queue` *`q`, struct `Pcb` *`new_pcb`)
- struct `Pcb` * `pop` (struct `queue` *`q`)
Pops and returns the `Pcb` struct pointer of the process at the front of the queue.
- void `print_queue` (struct `queue` *`q`)
- void `print_queue_info` (struct `queue` *`q`)
- void `free_queue` (struct `queue` *`q`)
Frees all PCBs in the queue.
- void `free_child_queue` (struct `queue` *`q`)
Frees all PCBs in the children queue.
- struct `Pcb` * `front` (struct `queue` *`q`)
Returns pointer to front element.
- bool `is_empty` (struct `queue` *`q`)
- struct `Pcb` * `get_pcb_with_pid` (struct `queue` *`q`, pid_t `pid`)
- struct `Pcb` * `get_child_pcb_with_pid` (struct `queue` *`q`, pid_t `pid`)
Looks for the PCB in children queue of parent PCB.
- void `get_pcb_with_pgid_in_queue` (struct `queue` *`q`, pid_t `pgid`, struct `queue` *`result`)
- void `add_to_blocked_list` (struct `Pcb` *`p`)
- void `remove_pcb` (struct `queue` *`q`, struct `Pcb` *`p`)
Removes process from the queue but does not free the memory.

Variables

- struct `Pcb` * `blocked_list`

4.25.1 Function Documentation

4.25.1.1 add_to_blocked_list()

```
void add_to_blocked_list (
    struct Pcb * p )
```

4.25.1.2 create_queue()

```
struct queue * create_queue ( )
```

Allocates memory for a queue of PCBs.

Returns

Returns the pointer to the queue struct on success and NULL on failure

4.25.1.3 free_child_queue()

```
void free_child_queue (
    struct queue * q )
```

Frees all PCBs in the children queue.

Parameters

<i>q</i>	Reference to queue
----------	--------------------

4.25.1.4 free_queue()

```
void free_queue (  
    struct queue * q )
```

Frees all PCBs in the queue.

Parameters

<i>q</i>	Reference to queue
----------	--------------------

4.25.1.5 front()

```
struct Pcb * front (  
    struct queue * q )
```

Returns pointer to front element.

Parameters

<i>q</i>	Reference to queue
----------	--------------------

4.25.1.6 get_child_pcb_with_pid()

```
struct Pcb * get_child_pcb_with_pid (  
    struct queue * q,  
    pid_t pid )
```

Looks for the PCB in children queue of parent PCB.

Parameters

<i>q</i>	Children queue
<i>pid</i>	PID of child process

4.25.1.7 get_pcb_with_pid()

```
struct Pcb * get_pcb_with_pid (
    struct queue * q,
    pid_t pid )
```

4.25.1.8 get_pcbs_with_pgid_in_queue()

```
void get_pcbs_with_pgid_in_queue (
    struct queue * q,
    pid_t pgid,
    struct queue * result )
```

4.25.1.9 is_empty()

```
bool is_empty (
    struct queue * q )
```

4.25.1.10 pop()

```
struct Pcb * pop (
    struct queue * q )
```

Pops and returns the `Pcb` struct pointer of the process at the front of the queue.

Parameters

<i>q</i>	Queue of PCBs
----------	---------------

4.25.1.11 print_queue()

```
void print_queue (
    struct queue * q )
```

4.25.1.12 print_queue_info()

```
void print_queue_info (
    struct queue * q )
```

4.25.1.13 push()

```
void push (
    struct queue * q,
    struct Pcb * new_pcb )
```

4.25.1.14 remove_pcb()

```
void remove_pcb (
    struct queue * q,
    struct Pcb * p )
```

Removes process from the queue but does not free the memory.

Parameters

<i>q</i>	Queue from which process is to be removed
<i>p</i>	Pointer to the PCB of the process which is to be removed from queue q

4.25.2 Variable Documentation**4.25.2.1 blocked_list**

```
struct Pcb* blocked_list [extern]
```

4.26 queue.h

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

```
00001 #ifndef Q_H
00002 #define Q_H
00003
00004 #include "pcb.h"
00005 #include <signal.h>
00006 struct queue
00007 {
00008     struct Pcb *head;
00009     struct Pcb *tail;
00010     int size;
00011 };
00012
00015 struct queue *create_queue();
00016
00017 extern struct Pcb *blocked_list;
00018
00019 void push(struct queue *q, struct Pcb *new_pcb);
00020
00021 struct Pcb *pop(struct queue *q); // Removes and returns front element
00022 void print_queue(struct queue *q);
00023 void print_queue_info(struct queue *q);
00024
00027 void free_queue(struct queue *q);
00028
```

```

00031 void free_child_queue(struct queue *q);
00032
00033
00038 struct Pcb *front(struct queue *q);
00039
00040
00041 bool is_empty(struct queue *q);
00042
00043 struct Pcb *get_pcb_with_pid(struct queue *q, pid_t pid);
00044
00050 struct Pcb *get_child_pcb_with_pid(struct queue *q, pid_t pid);
00051
00052
00053 void get_pcbs_with_pgid_in_queue(struct queue *q, pid_t pgid, struct queue *result);
00054 // void push_child(struct Pcb *parent_pcb, struct Pcb *child_pcb);
00055
00056 void add_to_blocked_list(struct Pcb *p);
00057
00061 void remove_pcb(struct queue *q, struct Pcb *p);
00062 #endif

```

4.27 shell.c File Reference

```
#include "shell.h"
```

Macros

- #define `MAX_LEN` 512

Functions

- void `prompt` ()
- void `newline` ()
- void `handle_sigint` (int signal)
- void `handle_sigstsp` (int signal)
- int `get_arg_count` (struct `parsed_command` *cmd)
- void `get_args_from_cmd` (struct `parsed_command` *cmd, int arg_count, char *args[])
- void `non_interactive` ()
- int `handle_command` (struct `parsed_command` *cmd, char *raw_cmd, void(*func)(), char *args[])
- int `parse` (char *str)
- void `interactive_shell` ()
- void `fg_pipeline` (struct `Job` *j)
- void `print_finished` ()
- void `poll_background_jobs` ()
- int `main` (int argc, char *argv[])

Variables

- int `mode` = `INTERACTIVE`
- int `job_id` = 1
- bool `start` = true
- struct `j_queue` * `jobs_list`

4.27.1 Macro Definition Documentation

4.27.1.1 MAX_LEN

```
#define MAX_LEN 512
```

4.27.2 Function Documentation

4.27.2.1 fg_pipeline()

```
void fg_pipeline (
    struct Job * j )
```

4.27.2.2 get_arg_count()

```
int get_arg_count (
    struct parsed_command * cmd )
```

4.27.2.3 get_args_from_cmd()

```
void get_args_from_cmd (
    struct parsed_command * cmd,
    int arg_count,
    char * args[ ] )
```

4.27.2.4 handle_command()

```
int handle_command (
    struct parsed_command * cmd,
    char * raw_cmd,
    void(*)() func,
    char * args[ ] )
```

4.27.2.5 `handle_sigint()`

```
void handle_sigint (
    int signal )
```

4.27.2.6 `handle_sigtstp()`

```
void handle_sigtstp (
    int signal )
```

4.27.2.7 `interactive_shell()`

```
void interactive_shell ( )
```

4.27.2.8 `main()`

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

4.27.2.9 `newline()`

```
void newline ( )
```

4.27.2.10 `non_interactive()`

```
void non_interactive ( )
```

4.27.2.11 `parse()`

```
int parse (
    char * str )
```


4.27.2.12 poll_background_jobs()

```
void poll_background_jobs ( )
```

4.27.2.13 print_finished()

```
void print_finished ( )
```

4.27.2.14 prompt()

```
void prompt ( )
```

4.27.3 Variable Documentation

4.27.3.1 job_id

```
int job_id = 1
```

4.27.3.2 jobs_list

```
struct j_queue* jobs_list
```

4.27.3.3 mode

```
int mode = INTERACTIVE
```

4.27.3.4 start

```
bool start = true
```

4.28 shell.h File Reference

```
#include "user.h"
#include "Job.h"
#include "j_queue.h"
#include "f.h"
#include "utils.h"
```

Enumerations

- enum [SHELL_MODE](#) { [INTERACTIVE](#) , [SHELL_SCRIPT](#) }

Functions

- void [print_finished](#) ()
- void [poll_background_jobs](#) ()
- void [fg_pipeline](#) (struct [Job](#) *j)
- int [parse](#) (char *str)

4.28.1 Enumeration Type Documentation

4.28.1.1 SHELL_MODE

```
enum SHELL\_MODE
```

Enumerator

INTERACTIVE	
SHELL_SCRIPT	

4.28.2 Function Documentation

4.28.2.1 fg_pipeline()

```
void fg_pipeline (
    struct Job * j )
```

4.28.2.2 parse()

```
int parse (
    char * str )
```

4.28.2.3 poll_background_jobs()

```
void poll_background_jobs ( )
```

4.28.2.4 print_finished()

```
void print_finished ( )
```

4.29 shell.h

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

```
00001 #ifndef SHELL_H
00002 #define SHELL_H
00003
00004 #include "user.h"
00005 #include "Job.h"
00006 #include "j_queue.h"
00007 #include "f.h"
00008 #include "utils.h"
00009
00010 enum SHELL_MODE {
00011     INTERACTIVE,
00012     SHELL_SCRIPT
00013 };
00014
00015 void print_finished();
00016 void poll_background_jobs();
00017 void fg_pipeline(struct Job *j);
00018 int parse(char *str);
00019
00020 #endif
```

4.30 user.c File Reference

```
#include "user.h"
#include "kernel.h"
```

Functions

- bool [W_WIFEXITED](#) (int [status](#))
- bool [W_WIFSTOPPED](#) (int [status](#))
- bool [W_WIFSIGNALED](#) (int [status](#))
- bool [W_WIFCONTINUED](#) (int [status](#))
- int [p_nice](#) (pid_t [pid](#), int priority)
Change the priority of a process.
- void [p_sleep](#) (unsigned int ticks)
Put the calling process to sleep for a specified number of ticks.
- void [sleeps](#) (char *ticks)
Wrapper function for p_sleep that takes a string argument representing the number of ticks to sleep for.
- void [echo](#) (char *argStr)
Print a string argument to the console and terminate the calling process.
- void [kill_as_process](#) (pid_t [pid](#), int sig)
Send a signal to a process with a given PID and terminate the calling process.
- void [print_all_process_info](#) ()
Print information about all processes in the system to the console and terminate the calling process.
- pid_t [p_spawn](#) (void(*func)(), char *argv[], int fd0, int fd1, bool [fg](#))
- void [wait_log](#) (struct [Pcb](#) *calling_pcb)
- pid_t [p_waitpid](#) (pid_t [pid](#), int *wstatus, bool nohang)
- pid_t [p_waitpid_old](#) (pid_t [pid](#), int *wstatus, bool nohang)
- int [p_kill](#) (pid_t [pid](#), int sig)
sends the signal sig to the thread referenced by pid.
- void [p_exit](#) (void)
Exits thread unconditionally.
- int [p_tcset](#) (pid_t [pid](#))
Gives terminal control to process specified by pid.
- void [custom_read](#) (int fd, char *buf, ssize_t n)
- void [dummy](#) (char *str)
Dummy function for testing.
- void [spawn_shell](#) ()
Creates the shell process.
- pid_t [p_getpid](#) ()
Returns active process's PID.
- void [p_switch_to_scheduler](#) ()
Saves active context and switches to scheduler context.
- void [p_boot_kernel](#) ()
- void [zombie_child](#) ()
Function that does nothing, used to create a zombie child process.
- void [zombify](#) ()
Create a zombie child process that does nothing and keep the current process running indefinitely.
- void [orphan_child](#) ()
Function that loops indefinitely, used to create an orphan child process.
- void [orphanify](#) ()
Create an orphan child process that loops indefinitely and return immediately.

4.30.1 Function Documentation

4.30.1.1 custom_read()

```
void custom_read (
    int fd,
    char * buf,
    ssize_t n )
```

4.30.1.2 dummy()

```
void dummy (
    char * str )
```

Dummy function for testing.

Parameters

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

4.30.1.3 echo()

```
void echo (
    char * argStr )
```

Print a string argument to the console and terminate the calling process.

Parameters

<i>argStr</i>	The string to print to the console
---------------	------------------------------------

4.30.1.4 kill_as_process()

```
void kill_as_process (
    pid_t pid,
    int sig )
```

Send a signal to a process with a given PID and terminate the calling process.

Parameters

<i>pid</i>	The PID of the process to send the signal to
<i>sig</i>	The signal to send

4.30.1.5 orphan_child()

```
void orphan_child ( )
```

Function that loops indefinitely, used to create an orphan child process.

4.30.1.6 orphanify()

```
void orphanify ( )
```

Create an orphan child process that loops indefinitely and return immediately.

4.30.1.7 p_boot_kernel()

```
void p_boot_kernel ( )
```

4.30.1.8 p_exit()

```
void p_exit (
    void )
```

Exits thread unconditionally.

Parameters

<i>None</i>	
-------------	--

4.30.1.9 p_getpid()

```
pid_t p_getpid ( )
```

Returns active process's PID.

Returns

PID of current process

4.30.1.10 p_kill()

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

sends the signal sig to the thread referenced by pid.

Parameters

<i>pid</i>	pid of process to which we wish to send signal
<i>sig</i>	signal to be sent to process referenced by pid

Returns

It returns 0 on success, -1 on error.

4.30.1.11 p_nice()

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

Change the priority of a process.

Parameters

<i>pid</i>	The process ID of the target process
<i>priority</i>	The new priority of the target process

Returns

0 on success, -1 if the process with the given PID does not exist

4.30.1.12 p_sleep()

```
void p_sleep (
    unsigned int ticks )
```

Put the calling process to sleep for a specified number of ticks.

Parameters

<i>ticks</i>	The number of timer ticks to sleep for
--------------	--

4.30.1.13 p_spawn()

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

4.30.1.14 p_switch_to_scheduler()

```
void p_switch_to_scheduler ( )
```

Saves active context and switches to scheduler context.

4.30.1.15 p_tcset()

```
int p_tcset (
    pid_t pid )
```

Gives terminal control to process specified by pid.

Parameters

<i>pid</i>	Process PID
------------	-------------

4.30.1.16 p_waitpid()

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

4.30.1.17 p_waitpid_old()

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


4.30.1.18 print_all_process_info()

```
void print_all_process_info ( )
```

Print information about all processes in the system to the console and terminate the calling process.

4.30.1.19 sleeps()

```
void sleeps (
    char * ticks )
```

Wrapper function for p_sleep that takes a string argument representing the number of ticks to sleep for.

Parameters

<i>ticks</i>	A string representing the number of timer ticks to sleep for
--------------	--

4.30.1.20 spawn_shell()

```
void spawn_shell ( )
```

Creates the shell process.

4.30.1.21 W_WIFCONTINUED()

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

4.30.1.22 W_WIFEXITED()

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

4.30.1.23 W_WIFSIGNALED()

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

4.30.1.24 W_WIFSTOPPED()

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

4.30.1.25 wait_log()

```
void wait_log (
    struct Pcb * calling_pcb )
```

4.30.1.26 zombie_child()

```
void zombie_child ( )
```

Function that does nothing, used to create a zombie child process.

4.30.1.27 zombify()

```
void zombify ( )
```

Create a zombie child process that does nothing and keep the current process running indefinitely.

4.31 user.h File Reference

```
#include "pcb.h"
#include "queue.h"
#include "errors.h"
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <ucontext.h>
#include <unistd.h>
#include <signal.h>
```

Enumerations

- enum [S_SIGNALS](#) { [S_SIGTERM](#) , [S_SIGCONT](#) , [S_SIGSTOP](#) , [S_SIGCHLD](#) }
- enum [CHANGE](#) { [CHANGED](#) , [NO_CHANGE](#) }

Functions

- bool [W_WIFEXITED](#) (int [status](#))
- bool [W_WIFSTOPPED](#) (int [status](#))
- bool [W_WIFSIGNALED](#) (int [status](#))
- pid_t [p_spawn](#) (void(*func)(), char *argv[], int fd0, int fd1, bool [fg](#))
- pid_t [p_waitpid](#) (pid_t [pid](#), int *wstatus, bool nohang)
- int [p_kill](#) (pid_t [pid](#), int sig)
sends the signal sig to the thread referenced by pid.
- void [p_exit](#) (void)
Exits thread unconditionally.
- int [p_nice](#) (pid_t [pid](#), int priority)
Change the priority of a process.
- void [p_sleep](#) (unsigned int ticks)
Put the calling process to sleep for a specified number of ticks.
- void [sleeps](#) (char *ticks)
Wrapper function for p_sleep that takes a string argument representing the number of ticks to sleep for.
- void [echo](#) (char *argStr)
Print a string argument to the console and terminate the calling process.
- void [kill_as_process](#) (pid_t [pid](#), int sig)
Send a signal to a process with a given PID and terminate the calling process.
- void [print_all_process_info](#) ()
Print information about all processes in the system to the console and terminate the calling process.
- void [dummy](#) (char *str)
Dummy function for testing.
- void [spawn_shell](#) ()
Creates the shell process.
- void [interactive_shell](#) ()
- int [p_tcset](#) (pid_t [pid](#))
Gives terminal control to process specified by pid.
- pid_t [p_getpid](#) ()
Returns active process's PID.
- void [k_block_sigset](#) (sigset_t *mask)
- void [k_unblock_sigset](#) (sigset_t *mask)
- void [p_switch_to_scheduler](#) ()
Saves active context and switches to scheduler context.
- void [p_boot_kernel](#) ()
- void [zombie_child](#) ()
Function that does nothing, used to create a zombie child process.
- void [zombify](#) ()
Create a zombie child process that does nothing and keep the current process running indefinitely.
- void [orphan_child](#) ()
Function that loops indefinitely, used to create an orphan child process.
- void [orphanify](#) ()
Create an orphan child process that loops indefinitely and return immediately.

4.31.1 Enumeration Type Documentation

4.31.1.1 CHANGE

enum [CHANGE](#)

Enumerator

CHANGED	
NO_CHANGE	

4.31.1.2 S_SIGNALS

enum [S_SIGNALS](#)

Enumerator

S_SIGTERM	
S_SIGCONT	
S_SIGSTOP	
S_SIGCHLD	

4.31.2 Function Documentation

4.31.2.1 dummy()

```
void dummy (  
    char * str )
```

Dummy function for testing.

Parameters

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

4.31.2.2 echo()

```
void echo (  
    char * argStr )
```

Print a string argument to the console and terminate the calling process.

Parameters

<i>argStr</i>	The string to print to the console
---------------	------------------------------------

4.31.2.3 interactive_shell()

```
void interactive_shell ( )
```

4.31.2.4 k_block_sigset()

```
void k_block_sigset (
    sigset_t * mask )
```

4.31.2.5 k_unblock_sigset()

```
void k_unblock_sigset (
    sigset_t * mask )
```

4.31.2.6 kill_as_process()

```
void kill_as_process (
    pid_t pid,
    int sig )
```

Send a signal to a process with a given PID and terminate the calling process.

Parameters

<i>pid</i>	The PID of the process to send the signal to
<i>sig</i>	The signal to send

4.31.2.7 orphan_child()

```
void orphan_child ( )
```

Function that loops indefinitely, used to create an orphan child process.

4.31.2.8 orphanify()

```
void orphanify ( )
```

Create an orphan child process that loops indefinitely and return immediately.

4.31.2.9 p_boot_kernel()

```
void p_boot_kernel ( )
```

4.31.2.10 p_exit()

```
void p_exit (
    void )
```

Exits thread unconditionally.

Parameters

None	
------	--

4.31.2.11 p_getpid()

```
pid_t p_getpid ( )
```

Returns active process's PID.

Returns

PID of current process

4.31.2.12 p_kill()

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

sends the signal sig to the thread referenced by pid.

Parameters

<i>pid</i>	pid of process to which we wish to send signal
<i>sig</i>	signal to be sent to process referenced by pid

Returns

It returns 0 on success, -1 on error.

4.31.2.13 p_nice()

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

Change the priority of a process.

Parameters

<i>pid</i>	The process ID of the target process
<i>priority</i>	The new priority of the target process

Returns

0 on success, -1 if the process with the given PID does not exist

4.31.2.14 p_sleep()

```
void p_sleep (
    unsigned int ticks )
```

Put the calling process to sleep for a specified number of ticks.

Parameters

<i>ticks</i>	The number of timer ticks to sleep for
--------------	--

4.31.2.15 p_spawn()

```
pid_t p_spawn (
    void(*) () func,
```

```
char * argv[],
int fd0,
int fd1,
bool fg )
```

4.31.2.16 p_switch_to_scheduler()

```
void p_switch_to_scheduler ( )
```

Saves active context and switches to scheduler context.

4.31.2.17 p_tcset()

```
int p_tcset (
    pid_t pid )
```

Gives terminal control to process specified by pid.

Parameters

<i>pid</i>	Process PID
------------	-------------

4.31.2.18 p_waitpid()

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

4.31.2.19 print_all_process_info()

```
void print_all_process_info ( )
```

Print information about all processes in the system to the console and terminate the calling process.

4.31.2.20 sleeps()

```
void sleeps (
    char * ticks )
```

Wrapper function for p_sleep that takes a string argument representing the number of ticks to sleep for.

Parameters

<i>ticks</i>	A string representing the number of timer ticks to sleep for
--------------	--

4.31.2.21 spawn_shell()

```
void spawn_shell ( )
```

Creates the shell process.

4.31.2.22 W_WIFEXITED()

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

4.31.2.23 W_WIFSIGNALED()

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

4.31.2.24 W_WIFSTOPPED()

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

4.31.2.25 zombie_child()

```
void zombie_child ( )
```

Function that does nothing, used to create a zombie child process.

4.31.2.26 zombify()

```
void zombify ( )
```

Create a zombie child process that does nothing and keep the current process running indefinitely.

4.32 user.h

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

```

00001 #ifndef USER_H
00002 #define USER_H
00003
00004 #include "pcb.h"
00005 #include "queue.h"
00006 #include "errors.h"
00007 #include <stddef.h>
00008 #include <stdio.h>
00009 #include <stdlib.h>
00010 #include <sys/types.h>
00011 #include <ucontext.h>
00012 #include <unistd.h>
00013 #include <signal.h>
00014
00015 // extern struct Pcb *current_pcb;
00016 // extern struct queue *q;
00017 // extern struct queue *queues[3];
00018
00019 enum S_SIGNALS {
00020     S_SIGTERM,
00021     S_SIGCONT,
00022     S_SIGSTOP,
00023     S_SIGCHLD
00024 };
00025
00026
00027 enum CHANGE {
00028     CHANGED,
00029     NO_CHANGE
00030 };
00031
00032 bool W_WIFEXITED(int status);
00033 bool W_WIFSTOPPED(int status);
00034 bool W_WIFSIGNALED(int status);
00035
00036 pid_t p_spawn(void (*func)(), char *argv[], int fd0, int fd1, bool fg);
00037 pid_t p_waitpid(pid_t pid, int *wstatus, bool nohang);
00038
00045 int p_kill(pid_t pid, int sig);
00046
00049 void p_exit(void);
00050
00057 int p_nice(pid_t pid, int priority);
00058
00063 void p_sleep(unsigned int ticks);
00064
00070 void sleeps(char *ticks);
00071
00076 void echo(char *argStr);
00077
00083 void kill_as_process(pid_t pid, int sig);
00084
00089 void print_all_process_info();
00090
00095 void dummy(char *str);
00096
00100 void spawn_shell();
00101
00102 void interactive_shell();
00103
00104 static bool has_terminal_control = true;
00105 static pid_t tc_pid = 1;
00106
00107
00112 int p_tcset(pid_t pid);
00113
00118 pid_t p_getpid();
00119
00120 static sigset_t intmask;
00121
00122 void k_block_sigset(sigset_t *mask);
00123 void k_unblock_sigset(sigset_t *mask);
00124
00128 void p_switch_to_scheduler();
00129
00130 void p_boot_kernel();
00131
00132 // zombify and orphanify
00136 void zombie_child();
00137
00141 void zombify();
00142

```

```
00146 void orphan_child();
00147
00151 void orphanify();
00152 #endif
```

4.33 utils.c File Reference

```
#include "utils.h"
```

Functions

- void [memsetter](#) (char *buf, char val, unsigned long int len)
- void [print_out](#) (char *str, int len)
- void [debug](#) (char *msg)
- int [extractInt](#) (char str[])
- void [string_copy](#) (const char *from, char *to)
- int [myStrNCpy](#) (char *dest, char *src, int n)

4.33.1 Function Documentation

4.33.1.1 debug()

```
void debug (
    char * msg )
```

4.33.1.2 extractInt()

```
int extractInt (
    char str[] )
```

4.33.1.3 memsetter()

```
void memsetter (
    char * buf,
    char val,
    unsigned long int len )
```

4.33.1.4 myStrNCpy()

```
int myStrNCpy (
    char * dest,
    char * src,
    int n )
```

4.33.1.5 print_out()

```
void print_out (
    char * str,
    int len )
```

4.33.1.6 string_copy()

```
void string_copy (
    const char * from,
    char * to )
```

4.34 utils.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "parser.h"
```

Macros

- `#define DEBUG 0`

Functions

- void [memsetter](#) (char *buf, char val, unsigned long int len)
- void [print_out](#) (char *str, int len)
- void [debug](#) (char *msg)
- void [string_copy](#) (const char *from, char *to)
- int [extractInt](#) (char str[])
- int [myStrNCpy](#) (char *dest, char *src, int n)

4.34.1 Macro Definition Documentation

4.34.1.1 DEBUG

```
#define DEBUG 0
```

4.34.2 Function Documentation

4.34.2.1 debug()

```
void debug (  
    char * msg )
```

4.34.2.2 extractInt()

```
int extractInt (  
    char str[] )
```

4.34.2.3 memsetter()

```
void memsetter (  
    char * buf,  
    char val,  
    unsigned long int len )
```

4.34.2.4 myStrNCpy()

```
int myStrNCpy (  
    char * dest,  
    char * src,  
    int n )
```

4.34.2.5 print_out()

```
void print_out (  
    char * str,  
    int len )
```

4.34.2.6 string_copy()

```
void string_copy (
    const char * from,
    char * to )
```

4.35 utils.h

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

```
00001 #ifndef UTILS_H
00002 #define UTILS_H
00003 #include <stdio.h>
00004 #include <stdlib.h>
00005 #include <unistd.h>
00006 #include <string.h>
00007 #include "parser.h"
00008
00009 #define DEBUG 0
00010
00011 void memsetter(char *buf, char val, unsigned long int len);
00012 void print_out(char *str, int len);
00013 void debug(char *msg);
00014
00015 void string_copy(const char *from, char *to);
00016 int extractInt(char str[]);
00017 int myStrNCpy(char *dest, char *src, int n);
00018
00019
00020 #endif
```

Index

- `__attribute__`
 - f.h, [27](#)
 - PennFAT.h, [73](#)
- `add_child_to_parent_q`
 - pcb.c, [57](#)
 - pcb.h, [60](#)
- `add_child_to_parent_zombies`
 - pcb.c, [58](#)
 - pcb.h, [60](#)
- `add_to_blocked_list`
 - queue.h, [82](#)
- `back`
 - PennFAT.c, [71](#)
- `BG`
 - Job.h, [36](#)
- `bg`
 - PennFAT.c, [65](#)
- `bgNode, 5`
 - cm, [5](#)
 - groupsize, [5](#)
 - jid, [5](#)
 - next, [5](#)
 - self_pgid, [6](#)
 - self_pid, [6](#)
 - state, [6](#)
- `Block_size`
 - superblock, [16](#)
- `BLOCKED`
 - pcb.h, [60](#)
- `blocked_list`
 - queue.h, [85](#)
- `boot_kernel`
 - kernel.h, [48](#)
- `c`
 - PennFAT.c, [71](#)
- `cat`
 - PennFAT.c, [65](#)
 - PennFAT.h, [74](#)
- `cat_a`
 - PennFAT.c, [65](#)
- `cat_file`
 - PennFAT.c, [65](#)
- `cat_w`
 - PennFAT.c, [65](#)
- `CHANGE`
 - user.h, [99](#)
- `CHANGED`
 - user.h, [100](#)
- `changed_child_pcb`
 - Pcb, [11](#)
- `check_allzero`
 - f.c, [22](#)
 - PennFAT.c, [66](#)
 - PennFAT.h, [74](#)
- `check_current`
 - PennFAT.c, [66](#)
- `check_finished`
 - PennFAT.c, [66](#)
- `check_redirection`
 - PennFAT.c, [66](#)
- `children_pcb`
 - Pcb, [11](#)
- `chmod1`
 - f.c, [22](#)
 - PennFAT.c, [66](#)
 - PennFAT.h, [74](#)
- `cm`
 - bgNode, [5](#)
- `cmd`
 - Job, [7](#)
 - Pcb, [11](#)
- `commands`
 - parsed_command, [10](#)
- `contain_file`
 - f.c, [22](#)
 - PennFAT.c, [66](#)
 - PennFAT.h, [74](#)
- `copy`
 - PennFAT.c, [67](#)
 - PennFAT.h, [74](#)
- `count_finished`
 - Job, [7](#)
- `create_j_queue`
 - j_queue.c, [31](#)
 - j_queue.h, [33](#)
- `create_job`
 - Job.c, [35](#)
 - Job.h, [36](#)
- `create_newfile`
 - f.c, [22](#)
 - PennFAT.c, [67](#)
- `create_pcb`
 - pcb.c, [58](#)
 - pcb.h, [61](#)
- `create_pcb_no_context`
 - pcb.c, [58](#)

- pcb.h, 61
- create_queue
 - queue.c, 78
 - queue.h, 82
- createQueue
 - PennFAT.c, 67
- creatNode
 - PennFAT.c, 67
- currentjob
 - PennFAT.c, 72
- custom_read
 - user.c, 92
- data_write
 - f.c, 23
 - PennFAT.c, 67
 - PennFAT.h, 74
- DBoffset
 - superblock, 16
- DEBUG
 - utils.h, 108
- debug
 - utils.c, 107
 - utils.h, 109
- DEFAULT
 - errors.h, 20
- delete_child_pcb
 - pcb.c, 58
 - pcb.h, 61
- deQueue
 - PennFAT.c, 67
- dummy
 - user.c, 93
 - user.h, 100
- echo
 - user.c, 93
 - user.h, 100
- enQueue
 - PennFAT.c, 68
- ERRORS
 - errors.h, 19
- errors.c, 19
 - set_errno, 19
- errors.h, 19
 - DEFAULT, 20
 - ERRORS, 19
 - p_perror, 20
 - PID_INVALID, 20
 - set_errno, 20
- exited_child
 - Pcb, 12
- EXPECT_COMMANDS
 - parser.h, 55
- EXPECT_INPUT_FILENAME
 - parser.h, 55
- EXPECT_OUTPUT_FILENAME
 - parser.h, 55
- extractInt
 - utils.c, 107
 - utils.h, 109
- f.c, 20
 - check_allzero, 22
 - chmod1, 22
 - contain_file, 22
 - create_newfile, 22
 - data_write, 23
 - f_cat, 23
 - f_chmod, 23
 - f_close, 23
 - f_copy, 23
 - f_ls, 24
 - f_lseek, 24
 - f_mv, 24
 - f_open, 24
 - f_read, 24
 - f_rm, 24
 - F_SEEK_CUR, 21
 - F_SEEK_END, 21
 - F_SEEK_SET, 22
 - f_touch, 25
 - f_unlink, 25
 - f_unmount, 25
 - f_write, 25
 - FAT_block, 26
 - fatfs_init, 25
 - find_available_block, 25
 - INPUT_SIZE, 22
 - rd, 26
 - rm, 26
 - sb, 27
 - touch, 26
 - update_dir_entry, 26
 - update_fat, 26
- f.h, 27
 - __attribute__, 27
 - f_cat, 28
 - f_close, 28
 - f_copy, 28
 - f_ls, 28
 - f_lseek, 28
 - f_mv, 28
 - f_open, 29
 - f_read, 29
 - f_rm, 29
 - f_touch, 29
 - f_unlink, 29
 - f_unmount, 29
 - f_write, 30
 - fatfs_init, 30
- f_cat
 - f.c, 23
 - f.h, 28
- f_chmod
 - f.c, 23
- f_close
 - f.c, 23

- f.h, 28
- f_copy
 - f.c, 23
 - f.h, 28
- f_ls
 - f.c, 24
 - f.h, 28
- f_lseek
 - f.c, 24
 - f.h, 28
- f_mv
 - f.c, 24
 - f.h, 28
- f_open
 - f.c, 24
 - f.h, 29
- f_read
 - f.c, 24
 - f.h, 29
- f_rm
 - f.c, 24
 - f.h, 29
- F_SEEK_CUR
 - f.c, 21
- F_SEEK_END
 - f.c, 21
- F_SEEK_SET
 - f.c, 22
- f_touch
 - f.c, 25
 - f.h, 29
- f_unlink
 - f.c, 25
 - f.h, 29
- f_unmount
 - f.c, 25
 - f.h, 29
- f_write
 - f.c, 25
 - f.h, 30
- FAT_block
 - f.c, 26
 - PennFAT.c, 72
- FAT_tablesize
 - superblock, 16
- fatfs_init
 - f.c, 25
 - f.h, 30
- fd
 - Pcb, 12
 - superblock, 16
- FG
 - Job.h, 36
- fg
 - Pcb, 12
 - PennFAT.c, 68
- fg_pipeline
 - shell.c, 87
- shell.h, 90
- FIN_RUN
 - kernel.h, 48
- find_available_block
 - f.c, 25
 - PennFAT.c, 68
 - PennFAT.h, 75
- find_job_with_id
 - j_queue.c, 31
 - j_queue.h, 33
- find_last_stopped_job
 - j_queue.c, 31
 - j_queue.h, 33
- FINISHED
 - pcb.h, 60
- free_child_queue
 - queue.c, 78
 - queue.h, 82
- free_j_queue
 - j_queue.c, 32
 - j_queue.h, 33
- free_job
 - Job.c, 35
 - Job.h, 36
- free_pcb
 - pcb.c, 59
 - pcb.h, 62
- free_queue
 - queue.c, 79
 - queue.h, 83
- free_queues
 - kernel.c, 39
- freeq
 - PennFAT.c, 68
- front
 - jobQueue, 9
 - queue.c, 79
 - queue.h, 83
- get_active_context
 - kernel.c, 39
- get_active_pcb
 - kernel.c, 39
 - kernel.h, 48
- get_arg_count
 - shell.c, 87
- get_args_from_cmd
 - shell.c, 87
- get_blocked_queue
 - kernel.c, 39
 - kernel.h, 48
- get_child_pcb_with_pid
 - queue.c, 79
 - queue.h, 83
- get_clock_ticks
 - kernel.c, 39
 - kernel.h, 48
- get_counter
 - kernel.c, 40

- kernel.h, 48
- get_how_ended
 - kernel.c, 40
- get_log_fp
 - kernel.c, 40
 - kernel.h, 49
- get_next_queue
 - kernel.c, 40
- get_next_valid_queue
 - kernel.c, 40
- get_num_of_ready_processes
 - kernel.c, 40
 - kernel.h, 49
- get_pcb
 - kernel.c, 40
 - kernel.h, 49
- get_pcb_with_pid
 - queue.c, 79
 - queue.h, 83
- get_pcb_with_pgid
 - kernel.c, 41
 - kernel.h, 49
- get_pcb_with_pgid_in_queue
 - queue.c, 80
 - queue.h, 84
- get_queue_with_priority
 - kernel.c, 41
 - kernel.h, 49
- get_scheduler_context
 - kernel.c, 41
 - kernel.h, 49
- get_shell_context
 - kernel.c, 41
 - kernel.h, 49
- gq
 - PennFAT.c, 72
- ground
 - Job, 7
- group_id
 - Job, 8
- groupsize
 - bgNode, 5
- handle_command
 - shell.c, 87
- handle_sigint
 - shell.c, 87
- handle_sigstsp
 - shell.c, 88
- has_tc
 - Pcb, 12
- head
 - j_queue, 6
 - queue, 15
- HOW_ENDED
 - kernel.h, 48
- idle_process
 - kernel.c, 41
- kernel.h, 50
- inc_clock
 - kernel.c, 41
- init_scheduler_context
 - kernel.c, 41
- initialise_queues
 - kernel.c, 42
- INPUT_SIZE
 - f.c, 22
 - PennFAT.c, 65
- INTERACTIVE
 - shell.h, 90
- interactive_shell
 - shell.c, 88
 - user.h, 101
- INThandler
 - PennFAT.c, 68
- is_background
 - parsed_command, 10
- is_empty
 - queue.c, 80
 - queue.h, 84
- is_file_append
 - parsed_command, 10
- J_FINISHED
 - Job.h, 36
- j_queue, 6
 - head, 6
 - size, 6
 - tail, 7
- j_queue.c, 31
 - create_j_queue, 31
 - find_job_with_id, 31
 - find_last_stopped_job, 31
 - free_j_queue, 32
 - pop_job, 32
 - print_j_list, 32
 - push_job, 32
 - remove_job, 32
- j_queue.h, 32
 - create_j_queue, 33
 - find_job_with_id, 33
 - find_last_stopped_job, 33
 - free_j_queue, 33
 - pop_job, 33
 - print_j_list, 34
 - push_job, 34
 - remove_job, 34
- J_RUNNING
 - Job.h, 36
- J_STOPPED
 - Job.h, 36
- J_TERMINATED
 - Job.h, 36
- jid
 - bgNode, 5
- Job, 7
 - cmd, 7

- count_finished, 7
- ground, 7
- group_id, 8
- job_id, 8
- next, 8
- prev, 8
- rawCmd, 8
- status, 8
- Job.c, 35
 - create_job, 35
 - free_job, 35
- Job.h, 35
 - BG, 36
 - create_job, 36
 - FG, 36
 - free_job, 36
 - J_FINISHED, 36
 - J_RUNNING, 36
 - J_STOPPED, 36
 - J_TERMINATED, 36
 - jobGround, 36
 - jobStatus, 36
- job_id
 - Job, 8
 - shell.c, 89
- jobGround
 - Job.h, 36
- jobid
 - PennFAT.c, 72
- jobQueue, 9
 - front, 9
 - q_size, 9
 - rear, 9
- jobs
 - PennFAT.c, 68
- jobs_list
 - shell.c, 89
- jobStatus
 - Job.h, 36
- k_block_sigset
 - kernel.c, 42
 - user.h, 101
- k_boot_kernel
 - kernel.c, 42
 - kernel.h, 50
- k_exit
 - kernel.c, 42
 - kernel.h, 50
- k_process_block
 - kernel.c, 42
 - kernel.h, 50
- k_process_cleanup
 - kernel.c, 43
 - kernel.h, 50
- k_process_create
 - kernel.c, 43
 - kernel.h, 51
- k_process_kill
 - kernel.c, 43
 - kernel.h, 51
- k_sleep
 - kernel.c, 43
 - kernel.h, 51
- k_tcset
 - kernel.c, 44
 - kernel.h, 52
- k_unblock_sigset
 - kernel.c, 44
 - user.h, 101
- k_waitpid
 - kernel.c, 44
 - kernel.h, 52
- kernel.c, 37
 - free_queues, 39
 - get_active_context, 39
 - get_active_pcb, 39
 - get_blocked_queue, 39
 - get_clock_ticks, 39
 - get_counter, 40
 - get_how_ended, 40
 - get_log_fp, 40
 - get_next_queue, 40
 - get_next_valid_queue, 40
 - get_num_of_ready_processes, 40
 - get_pcb, 40
 - get_pcb_with_pgid, 41
 - get_queue_with_priority, 41
 - get_scheduler_context, 41
 - get_shell_context, 41
 - idle_process, 41
 - inc_clock, 41
 - init_scheduler_context, 41
 - initialise_queues, 42
 - k_block_sigset, 42
 - k_boot_kernel, 42
 - k_exit, 42
 - k_process_block, 42
 - k_process_cleanup, 43
 - k_process_create, 43
 - k_process_kill, 43
 - k_sleep, 43
 - k_tcset, 44
 - k_unblock_sigset, 44
 - k_waitpid, 44
 - log_fp, 46
 - make_context, 45
 - pid_count, 46
 - print_all_queues_info, 45
 - q, 39
 - scheduler, 45
 - set_active_pcb_and_context, 45
 - set_how_ended, 45
 - setup_idle_process, 46
 - setup_log, 46
 - signal_set, 46
 - switch_to_scheduler, 46

- kernel.h, 47
 - boot_kernel, 48
 - FIN_RUN, 48
 - get_active_pcb, 48
 - get_blocked_queue, 48
 - get_clock_ticks, 48
 - get_counter, 48
 - get_log_fp, 49
 - get_num_of_ready_processes, 49
 - get_pcb, 49
 - get_pcb_with_pgid, 49
 - get_queue_with_priority, 49
 - get_scheduler_context, 49
 - get_shell_context, 49
 - HOW_ENDED, 48
 - idle_process, 50
 - k_boot_kernel, 50
 - k_exit, 50
 - k_process_block, 50
 - k_process_cleanup, 50
 - k_process_create, 51
 - k_process_kill, 51
 - k_sleep, 51
 - k_tcset, 52
 - k_waitpid, 52
 - make_context, 53
 - setup_log, 53
 - switch_to_scheduler, 53
 - TIMED_OUT, 48
 - wait_log, 53
- kill_as_process
 - user.c, 93
 - user.h, 101
- log_fp
 - kernel.c, 46
- ls
 - PennFAT.c, 69
 - PennFAT.h, 75
- main
 - PennFAT.c, 69
 - shell.c, 88
- make_context
 - kernel.c, 45
 - kernel.h, 53
- MAX_LEN
 - shell.c, 87
- memsetter
 - utils.c, 107
 - utils.h, 109
- mkfs
 - PennFAT.c, 69
 - PennFAT.h, 75
- mode
 - shell.c, 89
- mount
 - PennFAT.c, 69
 - PennFAT.h, 75
- mv
 - PennFAT.c, 69
 - PennFAT.h, 76
- myStrNCpy
 - utils.c, 107
 - utils.h, 109
- name
 - Pcb, 12
- newline
 - shell.c, 88
- next
 - bgNode, 5
 - Job, 8
 - Pcb, 12
- next_child
 - Pcb, 12
- NO_CHANGE
 - user.h, 100
- non_interactive
 - shell.c, 88
- num_commands
 - parsed_command, 10
- num_data_Blocks
 - superblock, 17
- num_FATBlocks
 - superblock, 17
- number_of_children
 - Pcb, 12
- ORPHAN
 - pcb.h, 60
- orphan_child
 - user.c, 94
 - user.h, 101
- orphanify
 - user.c, 94
 - user.h, 101
- p_boot_kernel
 - user.c, 94
 - user.h, 102
- p_exit
 - user.c, 94
 - user.h, 102
- p_getpid
 - user.c, 94
 - user.h, 102
- p_kill
 - user.c, 94
 - user.h, 102
- p_nice
 - user.c, 95
 - user.h, 103
- p_perror
 - errors.h, 20
- p_sleep
 - user.c, 95
 - user.h, 103

- p_spawn
 - user.c, 96
 - user.h, 103
- p_switch_to_scheduler
 - user.c, 96
 - user.h, 104
- p_tcset
 - user.c, 96
 - user.h, 104
- p_waitpid
 - user.c, 96
 - user.h, 104
- p_waitpid_old
 - user.c, 96
- parent_pcb
 - Pcb, 13
- parse
 - shell.c, 88
 - shell.h, 90
- parse_command
 - parser.h, 56
- parsed_command, 9
 - commands, 10
 - is_background, 10
 - is_file_append, 10
 - num_commands, 10
 - stdin_file, 10
 - stdout_file, 10
- parser.h, 54
 - EXPECT_COMMANDS, 55
 - EXPECT_INPUT_FILENAME, 55
 - EXPECT_OUTPUT_FILENAME, 55
 - parse_command, 56
 - print_parsed_command, 56
 - UNEXPECTED_AMPERSAND, 55
 - UNEXPECTED_FILE_INPUT, 55
 - UNEXPECTED_FILE_OUTPUT, 55
 - UNEXPECTED_PIPELINE, 55
- Pcb, 11
 - changed_child_pcb, 11
 - children_pcb, 11
 - cmd, 11
 - exited_child, 12
 - fd, 12
 - fg, 12
 - has_tc, 12
 - name, 12
 - next, 12
 - next_child, 12
 - number_of_children, 12
 - parent_pcb, 13
 - pgid, 13
 - pid, 13
 - ppid, 13
 - prev, 13
 - prev_child, 13
 - priority, 13
 - reading, 13
 - sleep_time, 14
 - sleeping, 14
 - state_change, 14
 - stateChangeType, 14
 - status, 14
 - uc, 14
 - waiting, 14
 - waiting_on, 14
 - zombies, 15
- pcb.c, 57
 - add_child_to_parent_q, 57
 - add_child_to_parent_zombies, 58
 - create_pcb, 58
 - create_pcb_no_context, 58
 - delete_child_pcb, 58
 - free_pcb, 59
 - remove_child_pcb, 59
- pcb.h, 59
 - add_child_to_parent_q, 60
 - add_child_to_parent_zombies, 60
 - BLOCKED, 60
 - create_pcb, 61
 - create_pcb_no_context, 61
 - delete_child_pcb, 61
 - FINISHED, 60
 - free_pcb, 62
 - ORPHAN, 60
 - READY, 60
 - remove_child_pcb, 62
 - RUNNING, 60
 - status, 60
 - STOPPED, 60
 - TERMINATED, 60
 - ZOMBIE, 60
- PennFAT.c, 63
 - back, 71
 - bg, 65
 - c, 71
 - cat, 65
 - cat_a, 65
 - cat_file, 65
 - cat_w, 65
 - check_allzero, 66
 - check_current, 66
 - check_finished, 66
 - check_redirection, 66
 - chmod1, 66
 - contain_file, 66
 - copy, 67
 - create_newfile, 67
 - createQueue, 67
 - creatNode, 67
 - currentjob, 72
 - data_write, 67
 - deQueue, 67
 - enQueue, 68
 - FAT_block, 72
 - fg, 68

- find_available_block, 68
- freeq, 68
- gq, 72
- INPUT_SIZE, 65
- INThandler, 68
- jobid, 72
- jobs, 68
- ls, 69
- main, 69
- mkfs, 69
- mount, 69
- mv, 69
- pid, 72
- pid1, 72
- print_status, 69
- rd, 72
- removeQueue, 70
- rm, 70
- sb, 72
- search_byjid, 70
- sig_handler, 70
- sigchld_handler, 70
- sigstp_handler, 70
- stringcpy, 70
- touch, 71
- unmount, 71
- update_dir_entry, 71
- update_fat, 71
- PennFAT.h, 73
 - __attribute__, 73
 - cat, 74
 - check_allzero, 74
 - chmod1, 74
 - contain_file, 74
 - copy, 74
 - data_write, 74
 - find_available_block, 75
 - ls, 75
 - mkfs, 75
 - mount, 75
 - mv, 76
 - print_status, 76
 - rm, 76
 - touch, 76
 - unmount, 76
 - update_dir_entry, 76
 - update_fat, 77
- pgid
 - Pcb, 13
- pid
 - Pcb, 13
 - PennFAT.c, 72
- pid1
 - PennFAT.c, 72
- pid_count
 - kernel.c, 46
- PID_INVALID
 - errors.h, 20
- poll_background_jobs
 - shell.c, 88
 - shell.h, 91
- pop
 - queue.c, 80
 - queue.h, 84
- pop_child
 - queue.c, 80
- pop_job
 - j_queue.c, 32
 - j_queue.h, 33
- ppid
 - Pcb, 13
- prev
 - Job, 8
 - Pcb, 13
- prev_child
 - Pcb, 13
- print_all_process_info
 - user.c, 96
 - user.h, 104
- print_all_queues_info
 - kernel.c, 45
- print_finished
 - shell.c, 89
 - shell.h, 91
- print_j_list
 - j_queue.c, 32
 - j_queue.h, 34
- print_out
 - utils.c, 108
 - utils.h, 109
- print_parsed_command
 - parser.h, 56
- print_queue
 - queue.c, 80
 - queue.h, 84
- print_queue_info
 - queue.c, 80
 - queue.h, 84
- print_status
 - PennFAT.c, 69
 - PennFAT.h, 76
- priority
 - Pcb, 13
- prompt
 - shell.c, 89
- push
 - queue.c, 81
 - queue.h, 84
- push_job
 - j_queue.c, 32
 - j_queue.h, 34
- push_to_front
 - queue.c, 81
- q
 - kernel.c, 39
- q_size

- jobQueue, 9
- queue, 15
 - head, 15
 - size, 15
 - tail, 15
- queue.c, 78
 - create_queue, 78
 - free_child_queue, 78
 - free_queue, 79
 - front, 79
 - get_child_pcb_with_pid, 79
 - get_pcb_with_pid, 79
 - get_pcb_with_pgid_in_queue, 80
 - is_empty, 80
 - pop, 80
 - pop_child, 80
 - print_queue, 80
 - print_queue_info, 80
 - push, 81
 - push_to_front, 81
 - remove_pcb, 81
- queue.h, 81
 - add_to_blocked_list, 82
 - blocked_list, 85
 - create_queue, 82
 - free_child_queue, 82
 - free_queue, 83
 - front, 83
 - get_child_pcb_with_pid, 83
 - get_pcb_with_pid, 83
 - get_pcb_with_pgid_in_queue, 84
 - is_empty, 84
 - pop, 84
 - print_queue, 84
 - print_queue_info, 84
 - push, 84
 - remove_pcb, 85
- rawCmd
 - Job, 8
- rd
 - f.c, 26
 - PennFAT.c, 72
- rdsize
 - superblock, 17
- reading
 - Pcb, 13
- READY
 - pcb.h, 60
- rear
 - jobQueue, 9
- remove_child_pcb
 - pcb.c, 59
 - pcb.h, 62
- remove_job
 - j_queue.c, 32
 - j_queue.h, 34
- remove_pcb
 - queue.c, 81
 - queue.h, 85
- removeQueue
 - PennFAT.c, 70
- rm
 - f.c, 26
 - PennFAT.c, 70
 - PennFAT.h, 76
- RUNNING
 - pcb.h, 60
- S_SIGCHLD
 - user.h, 100
- S_SIGCONT
 - user.h, 100
- S_SIGNALS
 - user.h, 100
- S_SIGSTOP
 - user.h, 100
- S_SIGTERM
 - user.h, 100
- sb
 - f.c, 27
 - PennFAT.c, 72
- scheduler
 - kernel.c, 45
- search_byjid
 - PennFAT.c, 70
- self_pgid
 - bgNode, 6
- self_pid
 - bgNode, 6
- set_active_pcb_and_context
 - kernel.c, 45
- set_errno
 - errors.c, 19
 - errors.h, 20
- set_how_ended
 - kernel.c, 45
- setup_idle_process
 - kernel.c, 46
- setup_log
 - kernel.c, 46
 - kernel.h, 53
- shell.c, 86
 - fg_pipeline, 87
 - get_arg_count, 87
 - get_args_from_cmd, 87
 - handle_command, 87
 - handle_sigint, 87
 - handle_sigstp, 88
 - interactive_shell, 88
 - job_id, 89
 - jobs_list, 89
 - main, 88
 - MAX_LEN, 87
 - mode, 89
 - newline, 88
 - non_interactive, 88
 - parse, 88

- poll_background_jobs, 88
- print_finished, 89
- prompt, 89
- start, 89
- shell.h, 90
 - fg_pipeline, 90
 - INTERACTIVE, 90
 - parse, 90
 - poll_background_jobs, 91
 - print_finished, 91
 - SHELL_MODE, 90
 - SHELL_SCRIPT, 90
- SHELL_MODE
 - shell.h, 90
- SHELL_SCRIPT
 - shell.h, 90
- sig_handler
 - PennFAT.c, 70
- sigchld_handler
 - PennFAT.c, 70
- signal_set
 - kernel.c, 46
- sigstp_handler
 - PennFAT.c, 70
- size
 - j_queue, 6
 - queue, 15
- sleep_time
 - Pcb, 14
- sleeping
 - Pcb, 14
- sleeps
 - user.c, 97
 - user.h, 104
- spawn_shell
 - user.c, 97
 - user.h, 105
- start
 - shell.c, 89
- state
 - bgNode, 6
- state_change
 - Pcb, 14
- stateChangeType
 - Pcb, 14
- status
 - Job, 8
 - Pcb, 14
 - pcb.h, 60
- stdin_file
 - parsed_command, 10
- stdout_file
 - parsed_command, 10
- STOPPED
 - pcb.h, 60
- string_copy
 - utils.c, 108
 - utils.h, 109
- stringcpy
 - PennFAT.c, 70
- superblock, 16
 - Block_size, 16
 - DBoffset, 16
 - FAT_tablesize, 16
 - fd, 16
 - num_data_Blocks, 17
 - num_FATBlocks, 17
 - rdsiz, 17
- switch_to_scheduler
 - kernel.c, 46
 - kernel.h, 53
- tail
 - j_queue, 7
 - queue, 15
- TERMINATED
 - pcb.h, 60
- TIMED_OUT
 - kernel.h, 48
- touch
 - f.c, 26
 - PennFAT.c, 71
 - PennFAT.h, 76
- uc
 - Pcb, 14
- UNEXPECTED_AMPERSAND
 - parser.h, 55
- UNEXPECTED_FILE_INPUT
 - parser.h, 55
- UNEXPECTED_FILE_OUTPUT
 - parser.h, 55
- UNEXPECTED_PIPELINE
 - parser.h, 55
- unmount
 - PennFAT.c, 71
 - PennFAT.h, 76
- update_dir_entry
 - f.c, 26
 - PennFAT.c, 71
 - PennFAT.h, 76
- update_fat
 - f.c, 26
 - PennFAT.c, 71
 - PennFAT.h, 77
- user.c, 91
 - custom_read, 92
 - dummy, 93
 - echo, 93
 - kill_as_process, 93
 - orphan_child, 94
 - orphanify, 94
 - p_boot_kernel, 94
 - p_exit, 94
 - p_getpid, 94
 - p_kill, 94
 - p_nice, 95

- p_sleep, 95
- p_spawn, 96
- p_switch_to_scheduler, 96
- p_tcset, 96
- p_waitpid, 96
- p_waitpid_old, 96
- print_all_process_info, 96
- sleeps, 97
- spawn_shell, 97
- W_WIFCONTINUED, 97
- W_WIFEXITED, 97
- W_WIFSIGNALED, 97
- W_WIFSTOPPED, 97
- wait_log, 98
- zombie_child, 98
- zombify, 98
- user.h, 98
 - CHANGE, 99
 - CHANGED, 100
 - dummy, 100
 - echo, 100
 - interactive_shell, 101
 - k_block_sigset, 101
 - k_unblock_sigset, 101
 - kill_as_process, 101
 - NO_CHANGE, 100
 - orphan_child, 101
 - orphanify, 101
 - p_boot_kernel, 102
 - p_exit, 102
 - p_getpid, 102
 - p_kill, 102
 - p_nice, 103
 - p_sleep, 103
 - p_spawn, 103
 - p_switch_to_scheduler, 104
 - p_tcset, 104
 - p_waitpid, 104
 - print_all_process_info, 104
 - S_SIGCHLD, 100
 - S_SIGCONT, 100
 - S_SIGNALS, 100
 - S_SIGSTOP, 100
 - S_SIGTERM, 100
 - sleeps, 104
 - spawn_shell, 105
 - W_WIFEXITED, 105
 - W_WIFSIGNALED, 105
 - W_WIFSTOPPED, 105
 - zombie_child, 105
 - zombify, 105
- utils.c, 107
 - debug, 107
 - extractInt, 107
 - memsetter, 107
 - myStrNCpy, 107
 - print_out, 108
 - string_copy, 108
- utils.h, 108
 - DEBUG, 108
 - debug, 109
 - extractInt, 109
 - memsetter, 109
 - myStrNCpy, 109
 - print_out, 109
 - string_copy, 109
- W_WIFCONTINUED
 - user.c, 97
- W_WIFEXITED
 - user.c, 97
 - user.h, 105
- W_WIFSIGNALED
 - user.c, 97
 - user.h, 105
- W_WIFSTOPPED
 - user.c, 97
 - user.h, 105
- wait_log
 - kernel.h, 53
 - user.c, 98
- waiting
 - Pcb, 14
- waiting_on
 - Pcb, 14
- ZOMBIE
 - pcb.h, 60
- zombie_child
 - user.c, 98
 - user.h, 105
- zombies
 - Pcb, 15
- zombify
 - user.c, 98
 - user.h, 105