

My Project

Generated by Doxygen 1.9.6

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 block Struct Reference	5
3.1.1 Member Data Documentation	5
3.1.1.1 firstBlock	5
3.1.1.2 mtime	5
3.1.1.3 name	6
3.1.1.4 perm	6
3.1.1.5 size	6
3.1.1.6 type	6
3.2 fat_context Struct Reference	6
3.2.1 Member Data Documentation	7
3.2.1.1 BLOCK_SIZE	7
3.2.1.2 DIR_SIZE	7
3.2.1.3 DIRS_PER_BLOCK	7
3.2.1.4 empty_block	7
3.2.1.5 empty_dir	7
3.2.1.6 fd	7
3.2.1.7 LSB	7
3.2.1.8 MSB	8
3.2.1.9 NUM_DATA_BLOCKS	8
3.2.1.10 NUM_DATA_BYTES	8
3.2.1.11 NUM_FAT	8
3.2.1.12 NUM_FAT_BLOCKS	8
3.2.1.13 NUM_FAT_BYTES	8
3.3 fd_entry Struct Reference	8
3.3.1 Member Data Documentation	9
3.3.1.1 curr_data_ptr	9
3.3.1.2 fat_pointer	9
3.3.1.3 name	9
3.4 Group Struct Reference	9
3.4.1 Member Data Documentation	10
3.4.1.1 changed	10
3.4.1.2 id	10
3.4.1.3 ids	10
3.4.1.4 name	10
3.4.1.5 size	10
3.4.1.6 status	10

3.5 List Struct Reference	10
3.5.1 Member Data Documentation	11
3.5.1.1 head	11
3.5.1.2 tail	11
3.6 Node Struct Reference	11
3.6.1 Member Data Documentation	11
3.6.1.1 group	11
3.6.1.2 next	12
3.6.1.3 prev	12
3.7 parsed_command Struct Reference	12
3.7.1 Detailed Description	12
3.7.2 Member Data Documentation	12
3.7.2.1 commands	12
3.7.2.2 is_background	13
3.7.2.3 is_file_append	13
3.7.2.4 num_commands	13
3.7.2.5 stdin_file	13
3.7.2.6 stdout_file	13
3.8 pcb Struct Reference	13
3.8.1 Detailed Description	14
3.8.2 Member Data Documentation	14
3.8.2.1 blocked_until	14
3.8.2.2 child_pids	14
3.8.2.3 context	14
3.8.2.4 fds	15
3.8.2.5 name	15
3.8.2.6 num_children	15
3.8.2.7 num_to_wait_for	15
3.8.2.8 num_zombies	15
3.8.2.9 parent	15
3.8.2.10 pid	15
3.8.2.11 priority	15
3.8.2.12 sleeping	16
3.8.2.13 status	16
3.8.2.14 term_status	16
3.8.2.15 time_expired	16
3.8.2.16 to_wait_for	16
3.8.2.17 updated_flag	16
3.8.2.18 waiting	16
3.8.2.19 waiting_for	16
3.8.2.20 woke_up_by	17
3.8.2.21 zombies	17

3.9 pcb_list_node Struct Reference	17
3.9.1 Member Data Documentation	17
3.9.1.1 next	17
3.9.1.2 pcb	17
4 File Documentation	19
4.1 built_ins.c File Reference	19
4.1.1 Function Documentation	20
4.1.1.1 busy_process()	20
4.1.1.2 cat_wrapper()	20
4.1.1.3 chmod_wrapper()	20
4.1.1.4 cp_wrapper()	20
4.1.1.5 echo_wrapper()	20
4.1.1.6 idle_process()	20
4.1.1.7 invalid_cmd_process()	21
4.1.1.8 kill_process()	21
4.1.1.9 ls_wrapper()	21
4.1.1.10 mv_wrapper()	21
4.1.1.11 orphan_child()	21
4.1.1.12 orphanify()	21
4.1.1.13 parse_script()	21
4.1.1.14 ps_process()	22
4.1.1.15 rm_wrapper()	22
4.1.1.16 sleep_process()	22
4.1.1.17 touch_wrapper()	22
4.1.1.18 zombie_child()	22
4.1.1.19 zombify()	22
4.2 built_ins.h File Reference	23
4.2.1 Function Documentation	23
4.2.1.1 busy_process()	23
4.2.1.2 cat_wrapper()	23
4.2.1.3 chmod_wrapper()	23
4.2.1.4 cp_wrapper()	24
4.2.1.5 echo_wrapper()	24
4.2.1.6 idle_process()	24
4.2.1.7 invalid_cmd_process()	24
4.2.1.8 ls_wrapper()	24
4.2.1.9 mv_wrapper()	24
4.2.1.10 orphanify()	24
4.2.1.11 parse_script()	25
4.2.1.12 ps_process()	25
4.2.1.13 rm_wrapper()	25

4.2.1.14 sleep_process()	25
4.2.1.15 touch_wrapper()	25
4.2.1.16 zombify()	25
4.3 built_ins.h	26
4.4 errno.c File Reference	26
4.4.1 Variable Documentation	26
4.4.1.1 errno	26
4.5 errno.h File Reference	27
4.5.1 Macro Definition Documentation	27
4.5.1.1 E2BIG	27
4.5.1.2 EEOF	27
4.5.1.3 EFBIG	27
4.5.1.4 EFTF	27
4.5.1.5 EINVPERM	28
4.5.1.6 ENAMETOOLONG	28
4.5.1.7 ENOENT	28
4.5.1.8 EPROCESSDOESNOTEXIST	28
4.5.1.9 EUDWV	28
4.5.2 Variable Documentation	28
4.5.2.1 errno	28
4.6 errno.h	28
4.7 filefunc.c File Reference	29
4.7.1 Macro Definition Documentation	30
4.7.1.1 F_SEEK_CUR	30
4.7.1.2 F_SEEK_END	30
4.7.1.3 F_SEEK_SET	30
4.7.1.4 MAXLINELENGTH	31
4.7.2 Function Documentation	31
4.7.2.1 f_cat()	31
4.7.2.2 f_chmod()	31
4.7.2.3 f_close()	31
4.7.2.4 f_cp()	31
4.7.2.5 f_findscript()	32
4.7.2.6 f_ls()	32
4.7.2.7 f_lseek()	32
4.7.2.8 f_mv()	32
4.7.2.9 f_open()	32
4.7.2.10 f_read()	33
4.7.2.11 f_rm()	33
4.7.2.12 f_touch()	33
4.7.2.13 f_unlink()	33
4.7.2.14 f_write()	33

4.7.2.15 filefunc()	34
4.7.2.16 fileFuncConstructor()	34
4.7.2.17 find_file()	34
4.7.2.18 find_next_available_fd()	34
4.7.3 Variable Documentation	34
4.7.3.1 f_eof	34
4.7.3.2 f_fc	34
4.7.3.3 fd_ptr	35
4.7.3.4 files	35
4.7.3.5 NUM_FAT_ENTRIES	35
4.8 filefunc.h File Reference	35
4.8.1 Typedef Documentation	36
4.8.1.1 fd_entry	36
4.8.2 Function Documentation	36
4.8.2.1 f_cat()	36
4.8.2.2 f_chmod()	36
4.8.2.3 f_close()	37
4.8.2.4 f_cp()	37
4.8.2.5 f_findscript()	37
4.8.2.6 f_ls()	37
4.8.2.7 f_lseek()	37
4.8.2.8 f_mv()	38
4.8.2.9 f_open()	38
4.8.2.10 f_read()	38
4.8.2.11 f_rm()	38
4.8.2.12 f_touch()	38
4.8.2.13 f_unlink()	39
4.8.2.14 f_write()	39
4.8.2.15 fileFuncConstructor()	39
4.9 filefunc.h	39
4.10 filesystem.c File Reference	40
4.10.1 Function Documentation	40
4.10.1.1 clear_data_region()	40
4.10.1.2 clear_file()	41
4.10.1.3 constructor()	41
4.10.1.4 create()	41
4.10.1.5 dir_to_struct()	41
4.10.1.6 fatread()	41
4.10.1.7 fatreadblock()	41
4.10.1.8 fatremove()	42
4.10.1.9 fatwrite()	42
4.10.1.10 find_available_block()	42

4.10.1.11 find_dir_entry()	42
4.10.1.12 find_last_block()	42
4.10.1.13 initialize()	42
4.10.1.14 read_dir_entry()	43
4.10.1.15 read_fat_block()	43
4.10.1.16 seek_data_region()	43
4.10.1.17 seek_dir_data_region()	43
4.10.1.18 seek_to_write_data()	43
4.10.1.19 update_dir_entry()	43
4.10.1.20 write_block_fat()	44
4.10.1.21 write_next_dir_entry()	44
4.10.2 Variable Documentation	44
4.10.2.1 root	44
4.10.2.2 zeros	44
4.11 filesystem.h File Reference	44
4.11.1 Typedef Documentation	45
4.11.1.1 block	45
4.11.1.2 fat_context	45
4.11.2 Function Documentation	45
4.11.2.1 clear_file()	46
4.11.2.2 constructor()	46
4.11.2.3 create()	46
4.11.2.4 dir_to_struct()	46
4.11.2.5 fatread()	46
4.11.2.6 fatreadblock()	46
4.11.2.7 fatremove()	47
4.11.2.8 fatwrite()	47
4.11.2.9 find_available_block()	47
4.11.2.10 find_dir_entry()	47
4.11.2.11 find_last_block()	47
4.11.2.12 initialize()	47
4.11.2.13 read_dir_entry()	48
4.11.2.14 read_fat_block()	48
4.11.2.15 seek_data_region()	48
4.11.2.16 seek_dir_data_region()	48
4.11.2.17 seek_to_write_data()	48
4.11.2.18 update_dir_entry()	48
4.11.2.19 write_block_fat()	49
4.11.2.20 write_next_dir_entry()	49
4.12 filesystem.h	49
4.13 kernel.c File Reference	50
4.13.1 Function Documentation	51

4.13.1.1 main()	51
4.13.2 Variable Documentation	51
4.13.2.1 active_pcb	51
4.13.2.2 head	51
4.13.2.3 idle_context	51
4.13.2.4 log_file	52
4.13.2.5 main_context	52
4.13.2.6 max_pid	52
4.13.2.7 scheduler_context	52
4.13.2.8 terminal_control	52
4.13.2.9 ticks	52
4.14 kernel.h File Reference	52
4.14.1 Variable Documentation	53
4.14.1.1 head	53
4.14.1.2 queues	53
4.14.1.3 terminal_control	53
4.14.1.4 ticks	53
4.15 kernel.h	53
4.16 kernel_func.c File Reference	54
4.16.1 Function Documentation	54
4.16.1.1 k_process_cleanup_1()	54
4.16.1.2 k_process_create()	54
4.16.1.3 k_process_kill()	55
4.16.1.4 wake_up_parent()	55
4.17 kernel_func.h File Reference	55
4.17.1 Function Documentation	55
4.17.1.1 k_process_cleanup()	56
4.17.1.2 k_process_cleanup_1()	56
4.17.1.3 k_process_create()	56
4.17.1.4 k_process_kill()	56
4.17.1.5 wake_up_parent()	56
4.17.2 Variable Documentation	56
4.17.2.1 max_pid	57
4.17.2.2 scheduler_context	57
4.18 kernel_func.h	57
4.19 log.c File Reference	57
4.19.1 Function Documentation	58
4.19.1.1 log_command()	58
4.20 log.h File Reference	58
4.20.1 Macro Definition Documentation	58
4.20.1.1 BLOCKED_LOG	59
4.20.1.2 CONTINUED	59

4.20.1.3 CREATE	59
4.20.1.4 EXITED	59
4.20.1.5 NICE	59
4.20.1.6 ORPHAN	59
4.20.1.7 SCHEDULE	59
4.20.1.8 SIGNED	59
4.20.1.9 STOPPED_LOG	60
4.20.1.10 UNBLOCKED	60
4.20.1.11 WAITED	60
4.20.1.12 ZOMBIE	60
4.20.2 Function Documentation	60
4.20.2.1 log_command()	60
4.20.3 Variable Documentation	60
4.20.3.1 log_file	60
4.21 log.h	61
4.22 parser.h File Reference	61
4.22.1 Macro Definition Documentation	62
4.22.1.1 EXPECT_COMMANDS	62
4.22.1.2 EXPECT_INPUT_FILENAME	62
4.22.1.3 EXPECT_OUTPUT_FILENAME	62
4.22.1.4 UNEXPECTED_AMPERSAND	62
4.22.1.5 UNEXPECTED_FILE_INPUT	62
4.22.1.6 UNEXPECTED_FILE_OUTPUT	62
4.22.1.7 UNEXPECTED_PIPELINE	62
4.22.2 Function Documentation	63
4.22.2.1 parse_command()	63
4.22.2.2 print_parsed_command()	63
4.23 parser.h	63
4.24 pcb.h File Reference	64
4.24.1 Macro Definition Documentation	65
4.24.1.1 BLOCKED	65
4.24.1.2 RUNNING	65
4.24.1.3 STOPPED	65
4.24.1.4 TERM_NORMAL	65
4.24.1.5 TERM_SIGNED	65
4.24.1.6 ZOMBIED	65
4.24.2 Typedef Documentation	65
4.24.2.1 pcb	66
4.25 pcb.h	66
4.26 pcb_list.c File Reference	66
4.26.1 Function Documentation	67
4.26.1.1 add_pcb()	67

4.26.1.2 get_pcb_from_pid()	67
4.26.1.3 remove_pcb_from_pid()	67
4.26.1.4 soft_remove()	67
4.27 pcb_list.h File Reference	67
4.27.1 Typedef Documentation	68
4.27.1.1 pcb_list_node	68
4.27.2 Function Documentation	68
4.27.2.1 add_pcb()	68
4.27.2.2 get_pcb_from_pid()	68
4.27.2.3 remove_pcb_from_pid()	68
4.27.2.4 soft_remove()	69
4.28 pcb_list.h	69
4.29 pennfat.c File Reference	69
4.29.1 Macro Definition Documentation	70
4.29.1.1 MAXLINELENGTH	70
4.29.2 Function Documentation	70
4.29.2.1 cat()	70
4.29.2.2 cat_from_terminal()	71
4.29.2.3 chmod()	71
4.29.2.4 count_args()	71
4.29.2.5 cp()	71
4.29.2.6 fake_initialize()	71
4.29.2.7 ls()	71
4.29.2.8 main()	72
4.29.2.9 mkfs()	72
4.29.2.10 mount()	72
4.29.2.11 mv()	72
4.29.2.12 rm()	72
4.29.2.13 touch()	72
4.29.2.14 unmount()	73
4.29.3 Variable Documentation	73
4.29.3.1 eof	73
4.29.3.2 fat	73
4.29.3.3 fc	73
4.30 sched.c File Reference	73
4.30.1 Function Documentation	74
4.30.1.1 handleFinish()	74
4.30.1.2 schedule()	74
4.30.1.3 unblock()	74
4.30.2 Variable Documentation	74
4.30.2.1 queues	74
4.31 sched.h File Reference	74

4.31.1 Function Documentation	75
4.31.1.1 schedule()	75
4.31.2 Variable Documentation	75
4.31.2.1 active_pcb	75
4.31.2.2 idle_context	75
4.31.2.3 main_context	75
4.32 sched.h	75
4.33 shell.c File Reference	76
4.33.1 Function Documentation	76
4.33.1.1 parent_sigint_handler()	76
4.33.1.2 parent_sigstp_handler()	76
4.33.1.3 runOnForeground()	77
4.33.1.4 shell()	77
4.33.2 Variable Documentation	77
4.33.2.1 begin	77
4.33.2.2 lastJob	77
4.33.2.3 stoppedJobs	77
4.34 shell.h File Reference	78
4.34.1 Macro Definition Documentation	79
4.34.1.1 COMMAND_LENGTH	79
4.34.1.2 RETURN_ASCII_CODE	79
4.34.1.3 SHELL_BACKGROUND_TO_FOREGROUND	79
4.34.1.4 SHELL_FINISHED	79
4.34.1.5 SHELL_RESTARTING	79
4.34.1.6 SHELL_RUNNING	79
4.34.1.7 SHELL_STOPPED	80
4.34.2 Typedef Documentation	80
4.34.2.1 Group	80
4.34.2.2 List	80
4.34.2.3 Node	80
4.34.3 Function Documentation	80
4.34.3.1 Add()	80
4.34.3.2 clear()	80
4.34.3.3 execute()	81
4.34.3.4 Init()	82
4.34.3.5 isEmpty()	82
4.34.3.6 Peek()	82
4.34.3.7 printAll()	82
4.34.3.8 printJobs()	82
4.34.3.9 Remove()	83
4.34.3.10 RemoveById()	83
4.34.3.11 report()	83

4.34.3.12 shell()	84
4.34.3.13 updateStatuses()	84
4.34.4 Variable Documentation	84
4.34.4.1 childId	84
4.34.4.2 childStatus	84
4.34.4.3 lastJob	84
4.35 shell.h	85
4.36 shell_execute.c File Reference	85
4.36.1 Function Documentation	86
4.36.1.1 create_child()	86
4.36.1.2 execute()	86
4.37 shell_list.c File Reference	87
4.37.1 Function Documentation	87
4.37.1.1 Add()	87
4.37.1.2 clear()	88
4.37.1.3 Init()	88
4.37.1.4 isEmpty()	88
4.37.1.5 Peek()	88
4.37.1.6 printAll()	88
4.37.1.7 RemoveById()	88
4.38 shell_notif.c File Reference	89
4.38.1 Function Documentation	89
4.38.1.1 printJobs()	89
4.38.1.2 report()	90
4.38.1.3 updateStatuses()	90
4.39 signals.c File Reference	90
4.39.1 Function Documentation	91
4.39.1.1 sigalrm_handler()	91
4.39.1.2 sigint_handler()	91
4.39.1.3 sigstop_handler()	91
4.40 signals.h File Reference	91
4.40.1 Macro Definition Documentation	92
4.40.1.1 S_SIGCONT	92
4.40.1.2 S_SIGSTOP	92
4.40.1.3 S_SIGTERM	92
4.40.2 Function Documentation	92
4.40.2.1 sigalrm_handler()	92
4.40.2.2 sigint_handler()	92
4.40.2.3 sigstop_handler()	93
4.40.3 Variable Documentation	93
4.40.3.1 active_pcb	93
4.40.3.2 scheduler_context	93

4.41 signals.h	93
4.42 stress.c File Reference	93
4.42.1 Function Documentation	94
4.42.1.1 hang()	94
4.42.1.2 nohang()	94
4.42.1.3 recur()	94
4.43 stress.h File Reference	94
4.43.1 Function Documentation	95
4.43.1.1 hang()	95
4.43.1.2 nohang()	95
4.43.1.3 recur()	95
4.44 stress.h	95
4.45 user_func.c File Reference	96
4.45.1 Function Documentation	96
4.45.1.1 check_pid_is_child()	97
4.45.1.2 p_exit()	97
4.45.1.3 p_kill()	97
4.45.1.4 p_nice()	97
4.45.1.5 p_perror()	97
4.45.1.6 p_sleep()	97
4.45.1.7 p_spawn()	98
4.45.1.8 p_waitpid()	98
4.45.1.9 p_waitpid_1()	98
4.45.1.10 W_WIFEXITED()	98
4.45.1.11 W_WIFSIGNALED()	98
4.45.1.12 W_WIFSTOPPED()	99
4.46 user_func.h File Reference	99
4.46.1 Function Documentation	100
4.46.1.1 check_pid_is_child()	100
4.46.1.2 p_exit()	100
4.46.1.3 p_kill()	100
4.46.1.4 p_nice()	100
4.46.1.5 p_perror()	100
4.46.1.6 p_sleep()	101
4.46.1.7 p_spawn()	101
4.46.1.8 p_waitpid()	101
4.46.1.9 W_WIFEXITED()	101
4.46.1.10 W_WIFSIGNALED()	102
4.46.1.11 W_WIFSTOPPED()	102
4.46.2 Variable Documentation	102
4.46.2.1 active_pcb	102
4.46.2.2 head	102

4.46.2.3 queues	102
4.47 user_func.h	103
4.48 utils.c File Reference	103
4.48.1 Function Documentation	103
4.48.1.1 char_to_16()	104
4.48.1.2 char_to_32()	104
4.48.1.3 char_to_8()	104
4.48.1.4 is_zero()	104
4.49 utils.h File Reference	104
4.49.1 Macro Definition Documentation	105
4.49.1.1 MAX	105
4.49.1.2 MIN	105
4.49.2 Function Documentation	105
4.49.2.1 char_to_16()	105
4.49.2.2 char_to_32()	105
4.49.2.3 char_to_8()	106
4.49.2.4 get_msb()	106
4.49.2.5 is_zero()	106
4.50 utils.h	106
Index	107

Chapter 1

Class Index

1.1 Class List

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

block	5
fat_context	6
fd_entry	8
Group	9
List	10
Node	11
parsed_command	
Struct parsed_command stored all necessary information needed for penn-shell	12
pcb	
PCB struct	13
pcb_list_node	17

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

built_ins.c	19
built_ins.h	23
errno.c	26
errno.h	27
filefunc.c	29
filefunc.h	35
filesystem.c	40
filesystem.h	44
kernel.c	50
kernel.h	52
kernel_func.c	54
kernel_func.h	55
log.c	57
log.h	58
parser.h	61
pcb.h	64
pcb_list.c	66
pcb_list.h	67
pennfat.c	69
sched.c	73
sched.h	74
shell.c	76
shell.h	78
shell_execute.c	85
shell_list.c	87
shell_notif.c	89
signals.c	90
signals.h	91
stress.c	93
stress.h	94
user_func.c	96
user_func.h	99
utils.c	103
utils.h	104

Chapter 3

Class Documentation

3.1 block Struct Reference

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

Public Attributes

- char [name](#) [32]
- uint32_t [size](#)
- uint16_t [firstBlock](#)
- uint8_t [type](#)
- uint8_t [perm](#)
- time_t [mtime](#)

3.1.1 Member Data Documentation

3.1.1.1 firstBlock

```
uint16_t block::firstBlock
```

3.1.1.2 mtime

```
time_t block::mtime
```

3.1.1.3 name

```
char block::name[32]
```

3.1.1.4 perm

```
uint8_t block::perm
```

3.1.1.5 size

```
uint32_t block::size
```

3.1.1.6 type

```
uint8_t block::type
```

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

- [filesystem.h](#)

3.2 fat_context Struct Reference

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

Public Attributes

- int [BLOCK_SIZE](#)
- int [LSB](#)
- int [MSB](#)
- int [NUM_FAT_BLOCKS](#)
- int [NUM_DATA_BLOCKS](#)
- int [NUM_FAT](#)
- int [NUM_FAT_BYTES](#)
- int [NUM_DATA_BYTES](#)
- int [DIR_SIZE](#)
- int [DIRS_PER_BLOCK](#)
- int [fd](#)
- char * [empty_block](#)
- char * [empty_dir](#)

3.2.1 Member Data Documentation

3.2.1.1 BLOCK_SIZE

```
int fat_context::BLOCK_SIZE
```

3.2.1.2 DIR_SIZE

```
int fat_context::DIR_SIZE
```

3.2.1.3 DIRS_PER_BLOCK

```
int fat_context::DIRS_PER_BLOCK
```

3.2.1.4 empty_block

```
char* fat_context::empty_block
```

3.2.1.5 empty_dir

```
char* fat_context::empty_dir
```

3.2.1.6 fd

```
int fat_context::fd
```

3.2.1.7 LSB

```
int fat_context::LSB
```

3.2.1.8 MSB

```
int fat_context::MSB
```

3.2.1.9 NUM_DATA_BLOCKS

```
int fat_context::NUM_DATA_BLOCKS
```

3.2.1.10 NUM_DATA_BYTES

```
int fat_context::NUM_DATA_BYTES
```

3.2.1.11 NUM_FAT

```
int fat_context::NUM_FAT
```

3.2.1.12 NUM_FAT_BLOCKS

```
int fat_context::NUM_FAT_BLOCKS
```

3.2.1.13 NUM_FAT_BYTES

```
int fat_context::NUM_FAT_BYTES
```

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

- [filesystem.h](#)

3.3 fd_entry Struct Reference

```
#include <filefunc.h>
```


Public Attributes

- char [name](#) [32]
- int [fat_pointer](#)
- int [curr_data_ptr](#)

3.3.1 Member Data Documentation

3.3.1.1 curr_data_ptr

```
int fd_entry::curr_data_ptr
```

3.3.1.2 fat_pointer

```
int fd_entry::fat_pointer
```

3.3.1.3 name

```
char fd_entry::name[32]
```

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

- [filefunc.h](#)

3.4 Group Struct Reference

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

Public Attributes

- int [id](#)
- int * [ids](#)
- char * [name](#)
- int [status](#)
- int [size](#)
- bool [changed](#)

3.4.1 Member Data Documentation

3.4.1.1 changed

`bool Group::changed`

3.4.1.2 id

`int Group::id`

3.4.1.3 ids

`int* Group::ids`

3.4.1.4 name

`char* Group::name`

3.4.1.5 size

`int Group::size`

3.4.1.6 status

`int Group::status`

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

- [shell.h](#)

3.5 List Struct Reference

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

Public Attributes

- [Node](#) * [head](#)
- [Node](#) * [tail](#)

3.5.1 Member Data Documentation

3.5.1.1 head

```
Node* List::head
```

3.5.1.2 tail

```
Node* List::tail
```

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

- [shell.h](#)

3.6 Node Struct Reference

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

Public Attributes

- struct [Node](#) * [next](#)
- struct [Node](#) * [prev](#)
- [Group](#) * [group](#)

3.6.1 Member Data Documentation

3.6.1.1 group

```
Group* Node::group
```

3.6.1.2 next

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

3.6.1.3 prev

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

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

- [shell.h](#)

3.7 parsed_command Struct Reference

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

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

Public Attributes

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

3.7.1 Detailed Description

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

3.7.2 Member Data Documentation

3.7.2.1 commands

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

3.7.2.2 is_background

```
bool parsed_command::is_background
```

3.7.2.3 is_file_append

```
bool parsed_command::is_file_append
```

3.7.2.4 num_commands

```
size_t parsed_command::num_commands
```

3.7.2.5 stdin_file

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

3.7.2.6 stdout_file

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

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

- [parser.h](#)

3.8 pcb Struct Reference

PCB struct.

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

Public Attributes

- `ucontext_t *` `context`
- `struct pcb *` `parent`
- `pid_t` `pid`
- `int` `num_children`
- `pid_t *` `child_pids`
- `int` `num_zombies`
- `pid_t *` `zombies`
- `int` `fds` [2]
- `int` `priority`
- `int` `status`
- `int` `term_status`
- `bool` `waiting`
- `int` `num_to_wait_for`
- `pid_t *` `to_wait_for`
- `pid_t` `woke_up_by`
- `pid_t` `waiting_for`
- `bool` `updated_flag`
- `bool` `time_expired`
- `char *` `name`
- `int` `blocked_until`
- `bool` `sleeping`

3.8.1 Detailed Description

PCB struct.

3.8.2 Member Data Documentation

3.8.2.1 `blocked_until`

```
int pcb::blocked_until
```

3.8.2.2 `child_pids`

```
pid_t* pcb::child_pids
```

3.8.2.3 `context`

```
ucontext_t* pcb::context
```

3.8.2.4 fds

```
int pcb::fds[2]
```

3.8.2.5 name

```
char* pcb::name
```

3.8.2.6 num_children

```
int pcb::num_children
```

3.8.2.7 num_to_wait_for

```
int pcb::num_to_wait_for
```

3.8.2.8 num_zombies

```
int pcb::num_zombies
```

3.8.2.9 parent

```
struct pcb* pcb::parent
```

3.8.2.10 pid

```
pid_t pcb::pid
```

3.8.2.11 priority

```
int pcb::priority
```

3.8.2.12 sleeping

```
bool pcb::sleeping
```

3.8.2.13 status

```
int pcb::status
```

3.8.2.14 term_status

```
int pcb::term_status
```

3.8.2.15 time_expired

```
bool pcb::time_expired
```

3.8.2.16 to_wait_for

```
pid_t* pcb::to_wait_for
```

3.8.2.17 updated_flag

```
bool pcb::updated_flag
```

3.8.2.18 waiting

```
bool pcb::waiting
```

3.8.2.19 waiting_for

```
pid_t pcb::waiting_for
```


3.8.2.20 woke_up_by

```
pid_t pcb::woke_up_by
```

3.8.2.21 zombies

```
pid_t* pcb::zombies
```

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

- [pcb.h](#)

3.9 pcb_list_node Struct Reference

```
#include <pcb_list.h>
```

Public Attributes

- struct [pcb_list_node](#) * [next](#)
- struct [pcb](#) * [pcb](#)

3.9.1 Member Data Documentation

3.9.1.1 next

```
struct pcb\_list\_node* pcb_list_node::next
```

3.9.1.2 pcb

```
struct pcb* pcb_list_node::pcb
```

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

- [pcb_list.h](#)

Chapter 4

File Documentation

4.1 built_ins.c File Reference

```
#include "built_ins.h"
#include <signal.h>
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include "kernel.h"
#include <unistd.h>
#include "user_func.h"
#include "filefunc.h"
```

Functions

- void [idle_process](#) ()
- void [invalid_cmd_process](#) ()
- void [busy_process](#) ()
- void [sleep_process](#) (char **args)
- void [zombie_child](#) ()
- void [zombify](#) ()
- void [orphan_child](#) ()
- void [orphanify](#) ()
- void [ps_process](#) ()
- void [kill_process](#) ()
- void [echo_wrapper](#) (char **args)
- void [ls_wrapper](#) (char **args)
- void [cat_wrapper](#) (char **args)
- void [touch_wrapper](#) (char **args)
- void [mv_wrapper](#) (char **args)
- void [rm_wrapper](#) (char **args)
- void [cp_wrapper](#) (char **args)
- void [chmod_wrapper](#) (char **args)
- char ** [parse_script](#) (char **args)

4.1.1 Function Documentation

4.1.1.1 busy_process()

```
void busy_process ( )
```

4.1.1.2 cat_wrapper()

```
void cat_wrapper (
    char ** args )
```

4.1.1.3 chmod_wrapper()

```
void chmod_wrapper (
    char ** args )
```

4.1.1.4 cp_wrapper()

```
void cp_wrapper (
    char ** args )
```

4.1.1.5 echo_wrapper()

```
void echo_wrapper (
    char ** args )
```

4.1.1.6 idle_process()

```
void idle_process ( )
```

4.1.1.7 invalid_cmd_process()

```
void invalid_cmd_process ( )
```

4.1.1.8 kill_process()

```
void kill_process ( )
```

4.1.1.9 ls_wrapper()

```
void ls_wrapper (
    char ** args )
```

4.1.1.10 mv_wrapper()

```
void mv_wrapper (
    char ** args )
```

4.1.1.11 orphan_child()

```
void orphan_child ( )
```

4.1.1.12 orphanify()

```
void orphanify ( )
```

4.1.1.13 parse_script()

```
char ** parse_script (
    char ** args )
```

4.1.1.14 ps_process()

```
void ps_process ( )
```

4.1.1.15 rm_wrapper()

```
void rm_wrapper (
    char ** args )
```

4.1.1.16 sleep_process()

```
void sleep_process (
    char ** args )
```

4.1.1.17 touch_wrapper()

```
void touch_wrapper (
    char ** args )
```

4.1.1.18 zombie_child()

```
void zombie_child ( )
```

4.1.1.19 zombify()

```
void zombify ( )
```

4.2 built_ins.h File Reference

Functions

- void [idle_process](#) ()
- void [invalid_cmd_process](#) ()
- void [busy_process](#) ()
- void [sleep_process](#) ()
- void [zombify](#) ()
- void [orphanify](#) ()
- void [ps_process](#) ()
- void [echo_wrapper](#) (char **args)
- void [ls_wrapper](#) (char **args)
- void [cat_wrapper](#) (char **args)
- void [touch_wrapper](#) (char **args)
- void [mv_wrapper](#) (char **args)
- void [rm_wrapper](#) (char **args)
- void [cp_wrapper](#) (char **args)
- void [chmod_wrapper](#) (char **args)
- char ** [parse_script](#) (char **args)

4.2.1 Function Documentation

4.2.1.1 busy_process()

```
void busy_process ( )
```

4.2.1.2 cat_wrapper()

```
void cat_wrapper (
    char ** args )
```

4.2.1.3 chmod_wrapper()

```
void chmod_wrapper (
    char ** args )
```

4.2.1.4 cp_wrapper()

```
void cp_wrapper (
    char ** args )
```

4.2.1.5 echo_wrapper()

```
void echo_wrapper (
    char ** args )
```

4.2.1.6 idle_process()

```
void idle_process ( )
```

4.2.1.7 invalid_cmd_process()

```
void invalid_cmd_process ( )
```

4.2.1.8 ls_wrapper()

```
void ls_wrapper (
    char ** args )
```

4.2.1.9 mv_wrapper()

```
void mv_wrapper (
    char ** args )
```

4.2.1.10 orphanify()

```
void orphanify ( )
```


4.2.1.11 parse_script()

```
char ** parse_script (  
    char ** args )
```

4.2.1.12 ps_process()

```
void ps_process ( )
```

4.2.1.13 rm_wrapper()

```
void rm_wrapper (  
    char ** args )
```

4.2.1.14 sleep_process()

```
void sleep_process ( )
```

4.2.1.15 touch_wrapper()

```
void touch_wrapper (  
    char ** args )
```

4.2.1.16 zombify()

```
void zombify ( )
```

4.3 built_ins.h

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

```
1 #ifndef BUILT_INS_H
2 #define BUILT_INS_H
3
4 void idle_process();
5
6 void invalid_cmd_process();
7
8 void busy_process();
9
10 void sleep_process();
11
12 void zombieify();
13
14 void orphanify();
15
16 void ps_process();
17
18 void echo_wrapper(char **args);
19
20 void ls_wrapper(char **args);
21
22 void cat_wrapper(char **args);
23
24 void touch_wrapper(char **args);
25
26 void mv_wrapper(char **args);
27
28 void rm_wrapper(char **args);
29
30 void cp_wrapper(char **args);
31
32 void chmod_wrapper(char **args);
33
34 char** parse_script(char **args);
35
36 #endif // !BUILT_INS_H
```

4.4 errno.c File Reference

```
#include "errno.h"
```

Variables

- int `errno` = 0

4.4.1 Variable Documentation

4.4.1.1 `errno`

```
int errno = 0
```

4.5 errno.h File Reference

Macros

- `#define E2BIG` 1
- `#define EFBIG` 2
- `#define ENOENT` 3
- `#define ENAMETOOLONG` 4
- `#define EEOF` 5
- `#define EFTF` 6
- `#define EUDWV` 7
- `#define EINVPERM` 8
- `#define EPROCESSDOESNOTEXIST` 9

Variables

- `int errno`

4.5.1 Macro Definition Documentation

4.5.1.1 E2BIG

```
#define E2BIG 1
```

4.5.1.2 EEOF

```
#define EEOF 5
```

4.5.1.3 EFBIG

```
#define EFBIG 2
```

4.5.1.4 EFTF

```
#define EFTF 6
```

4.5.1.5 EINVPERM

```
#define EINVPERM 8
```

4.5.1.6 ENAMETOOLONG

```
#define ENAMETOOLONG 4
```

4.5.1.7 ENOENT

```
#define ENOENT 3
```

4.5.1.8 EPROCESSDOESNOTEXIST

```
#define EPROCESSDOESNOTEXIST 9
```

4.5.1.9 EUDWV

```
#define EUDWV 7
```

4.5.2 Variable Documentation

4.5.2.1 errno

```
int errno [extern]
```

4.6 errno.h

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

```
1 #define E2BIG 1
2 #define EFBIG 2
3 #define ENOENT 3
4 #define ENAMETOOLONG 4
5 #define EEOF 5
6 #define EFTF 6
7 #define EUDWV 7
8 #define EINVPERM 8
9 #define EPROCESSDOESNOTEXIST 9 //PROCESS DOES NOT EXIST
10
11 extern int errno;
```

4.7 filefunc.c File Reference

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <stdint.h>
#include <time.h>
#include "filesystem.h"
#include "filefunc.h"
#include "utils.h"
#include "errno.h"
```

Macros

- #define [MAXLINELENGTH](#) 4096
- #define [F_SEEK_SET](#) 0
- #define [F_SEEK_CUR](#) 1
- #define [F_SEEK_END](#) 2

Functions

- void [fileFuncConstructor](#) (char *fatfs)
NAME fileFuncConstructor DESCRIPTION constructor FOUND IN [filefunc.c](#) RETURNS void ERRORS.
- int [find_file](#) (char *filename)
NAME find_file DESCRIPTION finds a file fd based on filename FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- int [find_next_available_fd](#) ()
NAME find_next_available_fd DESCRIPTION finds the next available fd to use FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- int [f_open](#) (const char *fname, int mode)
NAME f_open DESCRIPTION open a file name fname with the mode mode and returns a file descriptor on success and a negative value on error FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- int [f_read](#) (int fd, int n, char *buf)
NAME f_read DESCRIPTION read n bytes from the file referenced by fd.
- int [f_write](#) (int fd, const char *str, int n)
NAME f_write DESCRIPTION write n bytes of the string referenced by str to the file fd and increment the file pointer by n.
- int [f_close](#) (int fd)
NAME f_close DESCRIPTION close the file fd and return 0 on success, or a negative value on failure.
- int [f_unlink](#) (const char *fname)
NAME f_unlink DESCRIPTION remove the file and throw -1 on error FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- int [f_lseek](#) (int fd, int offset, int whence)
NAME f_lseek DESCRIPTION reposition the file pointer for fd to the offset relative to whence.
- int [f_ls](#) (const char *filename, int fd)
NAME f_ls DESCRIPTION list the file filename in the current directory.
- int [f_cat](#) (char **input_files, int num_files, int fd)
NAME f_cat DESCRIPTION Concatenates files together with input files as an input FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

- void `f_touch` (char **input_files, int num_files)
NAME `f_touch` DESCRIPTION Creates the files if they do not exist FOUND IN `filefunc.c` RETURNS void ERRORS.
- int `f_mv` (char *source, char *dest)
NAME `f_mv` DESCRIPTION Renames SOURCE to DEST FOUND IN `filefunc.c` RETURNS int ERRORS -1.
- void `f_rm` (char *file)
NAME `f_rm` DESCRIPTION Removes the files FOUND IN `filefunc.c` RETURNS void ERRORS.
- int `f_cp` (char *source, char *dest)
NAME `f_cp` DESCRIPTION Copies SOURCE to DEST FOUND IN `filefunc.c` RETURNS int ERRORS -1.
- int `f_chmod` (char *filename, bool op, char perm)
NAME `f_chmod` DESCRIPTION Change the permissions based on input FOUND IN `filefunc.c` RETURNS int ERRORS -1.
- int `filefunc` (int argc, char **argv)
- char ** `f_findscript` (char *fname)
NAME `f_findscript` DESCRIPTION Runs a script with commands located in a file FOUND IN `filefunc.c` RETURNS char** ERRORS.

Variables

- int `NUM_FAT_ENTRIES`
- int `fd_ptr` = 2
- const uint16_t `f_eof` = 0xFFFF
- struct `fd_entry` ** `files`
- `fat_context` * `f_fc`

4.7.1 Macro Definition Documentation

4.7.1.1 F_SEEK_CUR

```
#define F_SEEK_CUR 1
```

4.7.1.2 F_SEEK_END

```
#define F_SEEK_END 2
```

4.7.1.3 F_SEEK_SET

```
#define F_SEEK_SET 0
```

4.7.1.4 MAXLINELENGTH

```
#define MAXLINELENGTH 4096
```

4.7.2 Function Documentation

4.7.2.1 f_cat()

```
int f_cat (
    char ** input_files,
    int num_files,
    int fd )
```

NAME `f_cat` DESCRIPTION Concatenates files together with input files as an input FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.7.2.2 f_chmod()

```
int f_chmod (
    char * filename,
    bool op,
    char perm )
```

NAME `f_chmod` DESCRIPTION Change the permissions based on input FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.7.2.3 f_close()

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

NAME `f_close` DESCRIPTION close the file `fd` and return 0 on success, or a negative value on failure.

FOUND IN [filefunc.c](#) RETURNS int ERRORS -1

4.7.2.4 f_cp()

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

NAME `f_cp` DESCRIPTION Copies SOURCE to DEST FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.7.2.5 `f_findscript()`

```
char ** f_findscript (
    char * fname )
```

NAME `f_findscript` DESCRIPTION Runs a script with commands located in a file FOUND IN [filefunc.c](#) RETURNS `char**` ERRORS.

4.7.2.6 `f_ls()`

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

NAME `f_ls` DESCRIPTION list the file `filename` in the current directory.

If `filename` is NULL, list all files in the current directory. FOUND IN [filefunc.c](#) RETURNS `int` ERRORS -1

4.7.2.7 `f_lseek()`

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

NAME `f_lseek` DESCRIPTION reposition the file pointer for `fd` to the offset relative to `whence`.

FOUND IN [filefunc.c](#) RETURNS `int` ERRORS -1

4.7.2.8 `f_mv()`

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

NAME `f_mv` DESCRIPTION Renames `SOURCE` to `DEST` FOUND IN [filefunc.c](#) RETURNS `int` ERRORS -1.

4.7.2.9 `f_open()`

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

NAME `f_open` DESCRIPTION open a file name `fname` with the mode `mode` and returns a file descriptor on success and a negative value on error FOUND IN [filefunc.c](#) RETURNS `int` ERRORS -1.

4.7.2.10 `f_read()`

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

NAME `f_read` DESCRIPTION read *n* bytes from the file referenced by *fd*.

On return, `f_read` returns the number of bytes read, 0 if EOF is reached, or a negative number on error. FOUND IN [filefunc.c](#) RETURNS int ERRORS -1

4.7.2.11 `f_rm()`

```
void f_rm (
    char * file )
```

NAME `f_rm` DESCRIPTION Removes the files FOUND IN [filefunc.c](#) RETURNS void ERRORS.

4.7.2.12 `f_touch()`

```
void f_touch (
    char ** input_files,
    int num_files )
```

NAME `f_touch` DESCRIPTION Creates the files if they do not exist FOUND IN [filefunc.c](#) RETURNS void ERRORS.

4.7.2.13 `f_unlink()`

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

NAME `f_unlink` DESCRIPTION remove the file and throw -1 on error FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.7.2.14 `f_write()`

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

NAME `f_write` DESCRIPTION write *n* bytes of the string referenced by *str* to the file *fd* and increment the file pointer by *n*.

On return, `f_write` returns the number of bytes written, or a negative value on error. FOUND IN [filefunc.c](#) RETURNS int ERRORS -1

4.7.2.15 filefunc()

```
int filefunc (
    int argc,
    char ** argv )
```

4.7.2.16 fileFuncConstructor()

```
void fileFuncConstructor (
    char * fatfs )
```

NAME fileFuncConstructor DESCRIPTION constructor FOUND IN [filefunc.c](#) RETURNS void ERRORS.

4.7.2.17 find_file()

```
int find_file (
    char * filename )
```

NAME find_file DESCRIPTION finds a file fd based on filename FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.7.2.18 find_next_available_fd()

```
int find_next_available_fd ( )
```

NAME find_next_available_fd DESCRIPTION finds the next available fd to use FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.7.3 Variable Documentation

4.7.3.1 f_eof

```
const uint16_t f_eof = 0xFFFF
```

4.7.3.2 f_fc

```
fat\_context* f_fc
```

4.7.3.3 fd_ptr

```
int fd_ptr = 2
```

4.7.3.4 files

```
struct fd_entry** files
```

4.7.3.5 NUM_FAT_ENTRIES

```
int NUM_FAT_ENTRIES
```

4.8 filefunc.h File Reference

Classes

- struct [fd_entry](#)

Typedefs

- typedef struct [fd_entry](#) [fd_entry](#)

Functions

- void [fileFuncConstructor](#) (char *fatfs)
NAME fileFuncConstructor DESCRIPTION constructor FOUND IN [filefunc.c](#) RETURNS void ERRORS.
- int [f_open](#) (const char *fname, int mode)
NAME f_open DESCRIPTION open a file name fname with the mode mode and returns a file descriptor on success and a negative value on error FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- int [f_read](#) (int fd, int n, char *buf)
NAME f_read DESCRIPTION read n bytes from the file referenced by fd.
- int [f_write](#) (int fd, const char *str, int n)
NAME f_write DESCRIPTION write n bytes of the string referenced by str to the file fd and increment the file pointer by n.
- int [f_close](#) (int fd)
NAME f_close DESCRIPTION close the file fd and return 0 on success, or a negative value on failure.
- int [f_unlink](#) (const char *fname)
NAME f_unlink DESCRIPTION remove the file and throw -1 on error FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- int [f_lseek](#) (int fd, int offset, int whence)
NAME f_lseek DESCRIPTION reposition the file pointer for fd to the offset relative to whence.
- int [f_ls](#) (const char *filename, int fd)
NAME f_ls DESCRIPTION list the file filename in the current directory.

- int `f_cat` (char **input_files, int num_files, int fd)
NAME `f_cat` DESCRIPTION Concatenates files together with input files as an input FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- void `f_touch` (char **input_files, int num_files)
NAME `f_touch` DESCRIPTION Creates the files if they do not exist FOUND IN [filefunc.c](#) RETURNS void ERRORS.
- int `f_mv` (char *source, char *dest)
NAME `f_mv` DESCRIPTION Renames SOURCE to DEST FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- void `f_rm` (char *file)
NAME `f_rm` DESCRIPTION Removes the files FOUND IN [filefunc.c](#) RETURNS void ERRORS.
- int `f_cp` (char *source, char *dest)
NAME `f_cp` DESCRIPTION Copies SOURCE to DEST FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- int `f_chmod` (char *filename, bool op, char perm)
NAME `f_chmod` DESCRIPTION Change the permissions based on input FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.
- char ** `f_findscript` (char *fname)
NAME `f_findscript` DESCRIPTION Runs a script with commands located in a file FOUND IN [filefunc.c](#) RETURNS char** ERRORS.

4.8.1 Typedef Documentation

4.8.1.1 fd_entry

```
typedef struct fd_entry fd_entry
```

4.8.2 Function Documentation

4.8.2.1 f_cat()

```
int f_cat (
    char ** input_files,
    int num_files,
    int fd )
```

NAME `f_cat` DESCRIPTION Concatenates files together with input files as an input FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.8.2.2 f_chmod()

```
int f_chmod (
    char * filename,
    bool op,
    char perm )
```

NAME `f_chmod` DESCRIPTION Change the permissions based on input FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.8.2.3 f_close()

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

NAME `f_close` DESCRIPTION close the file `fd` and return 0 on success, or a negative value on failure.

FOUND IN [filefunc.c](#) RETURNS int ERRORS -1

4.8.2.4 f_cp()

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

NAME `f_cp` DESCRIPTION Copies `SOURCE` to `DEST` FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.8.2.5 f_findscript()

```
char ** f_findscript (
    char * fname )
```

NAME `f_findscript` DESCRIPTION Runs a script with commands located in a file FOUND IN [filefunc.c](#) RETURNS `char**` ERRORS.

4.8.2.6 f_ls()

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

NAME `f_ls` DESCRIPTION list the file `filename` in the current directory.

If `filename` is NULL, list all files in the current directory. FOUND IN [filefunc.c](#) RETURNS int ERRORS -1

4.8.2.7 f_lseek()

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

NAME `f_lseek` DESCRIPTION reposition the file pointer for `fd` to the offset relative to `whence`.

FOUND IN [filefunc.c](#) RETURNS int ERRORS -1

4.8.2.8 `f_mv()`

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

NAME `f_mv` DESCRIPTION Renames SOURCE to DEST FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.8.2.9 `f_open()`

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

NAME `f_open` DESCRIPTION open a file name `fname` with the mode `mode` and returns a file descriptor on success and a negative value on error FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.8.2.10 `f_read()`

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

NAME `f_read` DESCRIPTION read `n` bytes from the file referenced by `fd`.

On return, `f_read` returns the number of bytes read, 0 if EOF is reached, or a negative number on error. FOUND IN [filefunc.c](#) RETURNS int ERRORS -1

4.8.2.11 `f_rm()`

```
void f_rm (
    char * file )
```

NAME `f_rm` DESCRIPTION Removes the files FOUND IN [filefunc.c](#) RETURNS void ERRORS.

4.8.2.12 `f_touch()`

```
void f_touch (
    char ** input_files,
    int num_files )
```

NAME `f_touch` DESCRIPTION Creates the files if they do not exist FOUND IN [filefunc.c](#) RETURNS void ERRORS.

4.8.2.13 f_unlink()

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

NAME `f_unlink` DESCRIPTION remove the file and throw -1 on error FOUND IN [filefunc.c](#) RETURNS int ERRORS -1.

4.8.2.14 f_write()

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

NAME `f_write` DESCRIPTION write n bytes of the string referenced by str to the file fd and increment the file pointer by n.

On return, `f_write` returns the number of bytes written, or a negative value on error. FOUND IN [filefunc.c](#) RETURNS int ERRORS -1

4.8.2.15 fileFuncConstructor()

```
void fileFuncConstructor (
    char * fatfs )
```

NAME `fileFuncConstructor` DESCRIPTION constructor FOUND IN [filefunc.c](#) RETURNS void ERRORS.

4.9 filefunc.h

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

```
1 typedef struct fd_entry fd_entry;
2
3 void fileFuncConstructor(char *fatfs);
4
5 struct fd_entry
6 {
7     char name[32];
8     int fat_pointer; // pointing to the start block of the file
9     int curr_data_ptr; // in bytes
10 };
11
12 int f_open(const char *fname, int mode);
13
14 int f_read(int fd, int n, char *buf);
15
16 int f_write(int fd, const char *str, int n);
17
18 int f_close(int fd);
19
20 int f_unlink(const char *fname);
21
22 int f_lseek(int fd, int offset, int whence);
23
24 int f_ls(const char *filename, int fd);
25
26 int f_cat(char **input_files, int num_files, int fd);
27
28 void f_touch(char **input_files, int num_files);
29
30 int f_mv(char *source, char *dest);
31
32 void f_rm(char *file);
33
34 int f_cp(char *source, char *dest);
35
36 int f_chmod(char *filename, bool op, char perm);
37
38 char** f_findscript(char* fname);
```

4.10 filesystem.c File Reference

```
#include "filesystem.h"
```

Functions

- `fat_context * constructor` (int lsb, int num_fat_blocks, char *filename)
- void `initialize` (fat_context *fc)
- void `seek_data_region` (int block_index, fat_context *fc)
- void `seek_dir_data_region` (int block_index, int dir_i, fat_context *fc)
- void `seek_to_write_data` (int block_index, int curr_size, fat_context *fc)
- uint16_t `find_available_block` (fat_context *fc)
- void `write_block_fat` (int block_index, uint16_t value, fat_context *fc)
- uint16_t `read_fat_block` (int block_index, fat_context *fc)
- int `find_last_block` (int first_block, fat_context *fc)
- void `write_next_dir_entry` (block *dir_entry, fat_context *fc)
- int `create` (char *filename, uint8_t perm, fat_context *fc)
- block * `dir_to_struct` (char *dir, fat_context *fc)
- block * `find_dir_entry` (char *filename, fat_context *fc)
- block * `read_dir_entry` (int dir_block, int dir_num, fat_context *fc)
- void `clear_data_region` (int data_block, fat_context *fc)
- void `update_dir_entry` (char *filename, block *dir_entry, fat_context *fc)
- void `clear_file` (char *filename, fat_context *fc)
- void `fatwrite` (char *filename, char *value, int len, fat_context *fc)
- char * `fatreadblock` (int block_index, fat_context *fc)
- void `fatread` (char *filename, fat_context *fc)
- void `fatremove` (char *filename, fat_context *fc)

Variables

- const uint16_t `root` = 0xFFFF
- const uint8_t `zeros` = 0x0000

4.10.1 Function Documentation

4.10.1.1 clear_data_region()

```
void clear_data_region (  
    int data_block,  
    fat_context * fc )
```


4.10.1.2 clear_file()

```
void clear_file (
    char * filename,
    fat_context * fc )
```

4.10.1.3 constructor()

```
fat_context * constructor (
    int lsb,
    int num_fat_blocks,
    char * filename )
```

4.10.1.4 create()

```
int create (
    char * filename,
    uint8_t perm,
    fat_context * fc )
```

4.10.1.5 dir_to_struct()

```
block * dir_to_struct (
    char * dir,
    fat_context * fc )
```

4.10.1.6 fatread()

```
void fatread (
    char * filename,
    fat_context * fc )
```

4.10.1.7 fatreadblock()

```
char * fatreadblock (
    int block_index,
    fat_context * fc )
```

4.10.1.8 fatremove()

```
void fatremove (
    char * filename,
    fat_context * fc )
```

4.10.1.9 fatwrite()

```
void fatwrite (
    char * filename,
    char * value,
    int len,
    fat_context * fc )
```

4.10.1.10 find_available_block()

```
uint16_t find_available_block (
    fat_context * fc )
```

4.10.1.11 find_dir_entry()

```
block * find_dir_entry (
    char * filename,
    fat_context * fc )
```

4.10.1.12 find_last_block()

```
int find_last_block (
    int first_block,
    fat_context * fc )
```

4.10.1.13 initialize()

```
void initialize (
    fat_context * fc )
```

4.10.1.14 read_dir_entry()

```
block * read_dir_entry (
    int dir_block,
    int dir_num,
    fat_context * fc )
```

4.10.1.15 read_fat_block()

```
uint16_t read_fat_block (
    int block_index,
    fat_context * fc )
```

4.10.1.16 seek_data_region()

```
void seek_data_region (
    int block_index,
    fat_context * fc )
```

4.10.1.17 seek_dir_data_region()

```
void seek_dir_data_region (
    int block_index,
    int dir_i,
    fat_context * fc )
```

4.10.1.18 seek_to_write_data()

```
void seek_to_write_data (
    int block_index,
    int curr_size,
    fat_context * fc )
```

4.10.1.19 update_dir_entry()

```
void update_dir_entry (
    char * filename,
    block * dir_entry,
    fat_context * fc )
```

4.10.1.20 write_block_fat()

```
void write_block_fat (
    int block_index,
    uint16_t value,
    fat_context * fc )
```

4.10.1.21 write_next_dir_entry()

```
void write_next_dir_entry (
    block * dir_entry,
    fat_context * fc )
```

4.10.2 Variable Documentation

4.10.2.1 root

```
const uint16_t root = 0xFFFF
```

4.10.2.2 zeros

```
const uint8_t zeros = 0x0000
```

4.11 filesystem.h File Reference

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdint.h>
#include <time.h>
#include "utils.h"
```

Classes

- struct [block](#)
- struct [fat_context](#)

Typedefs

- typedef struct [block](#) [block](#)
- typedef struct [fat_context](#) [fat_context](#)

Functions

- [fat_context](#) * [constructor](#) (int lsb, int num_fat_blocks, char *filename)
- void [initialize](#) ([fat_context](#) *fc)
- void [seek_data_region](#) (int block_index, [fat_context](#) *fc)
- void [seek_dir_data_region](#) (int block_index, int dir_i, [fat_context](#) *fc)
- void [seek_to_write_data](#) (int block_index, int curr_size, [fat_context](#) *fc)
- uint16_t [find_available_block](#) ([fat_context](#) *fc)
- void [write_block_fat](#) (int block_index, uint16_t value, [fat_context](#) *fc)
- uint16_t [read_fat_block](#) (int block_index, [fat_context](#) *fc)
- int [find_last_block](#) (int first_block, [fat_context](#) *fc)
- void [write_next_dir_entry](#) ([block](#) *dir_entry, [fat_context](#) *fc)
- int [create](#) (char *filename, uint8_t perm, [fat_context](#) *fc)
- [block](#) * [dir_to_struct](#) (char *dir, [fat_context](#) *fc)
- [block](#) * [find_dir_entry](#) (char *filename, [fat_context](#) *fc)
- [block](#) * [read_dir_entry](#) (int dir_block, int dir_num, [fat_context](#) *fc)
- void [update_dir_entry](#) (char *filename, [block](#) *dir_entry, [fat_context](#) *fc)
- void [clear_file](#) (char *filename, [fat_context](#) *fc)
- void [fatwrite](#) (char *filename, char *value, int len, [fat_context](#) *fc)
- char * [fatreadblock](#) (int block_index, [fat_context](#) *fc)
- void [fatread](#) (char *filename, [fat_context](#) *fc)
- void [fatremove](#) (char *filename, [fat_context](#) *fc)

4.11.1 Typedef Documentation

4.11.1.1 [block](#)

```
typedef struct block block
```

4.11.1.2 [fat_context](#)

```
typedef struct fat\_context fat\_context
```

4.11.2 Function Documentation

4.11.2.1 clear_file()

```
void clear_file (
    char * filename,
    fat_context * fc )
```

4.11.2.2 constructor()

```
fat_context * constructor (
    int lsb,
    int num_fat_blocks,
    char * filename )
```

4.11.2.3 create()

```
int create (
    char * filename,
    uint8_t perm,
    fat_context * fc )
```

4.11.2.4 dir_to_struct()

```
block * dir_to_struct (
    char * dir,
    fat_context * fc )
```

4.11.2.5 fatread()

```
void fatread (
    char * filename,
    fat_context * fc )
```

4.11.2.6 fatreadblock()

```
char * fatreadblock (
    int block_index,
    fat_context * fc )
```

4.11.2.7 fatremove()

```
void fatremove (
    char * filename,
    fat_context * fc )
```

4.11.2.8 fatwrite()

```
void fatwrite (
    char * filename,
    char * value,
    int len,
    fat_context * fc )
```

4.11.2.9 find_available_block()

```
uint16_t find_available_block (
    fat_context * fc )
```

4.11.2.10 find_dir_entry()

```
block * find_dir_entry (
    char * filename,
    fat_context * fc )
```

4.11.2.11 find_last_block()

```
int find_last_block (
    int first_block,
    fat_context * fc )
```

4.11.2.12 initialize()

```
void initialize (
    fat_context * fc )
```

4.11.2.13 read_dir_entry()

```
block * read_dir_entry (
    int dir_block,
    int dir_num,
    fat_context * fc )
```

4.11.2.14 read_fat_block()

```
uint16_t read_fat_block (
    int block_index,
    fat_context * fc )
```

4.11.2.15 seek_data_region()

```
void seek_data_region (
    int block_index,
    fat_context * fc )
```

4.11.2.16 seek_dir_data_region()

```
void seek_dir_data_region (
    int block_index,
    int dir_i,
    fat_context * fc )
```

4.11.2.17 seek_to_write_data()

```
void seek_to_write_data (
    int block_index,
    int curr_size,
    fat_context * fc )
```

4.11.2.18 update_dir_entry()

```
void update_dir_entry (
    char * filename,
    block * dir_entry,
    fat_context * fc )
```


4.11.2.19 write_block_fat()

```
void write_block_fat (
    int block_index,
    uint16_t value,
    fat_context * fc )
```

4.11.2.20 write_next_dir_entry()

```
void write_next_dir_entry (
    block * dir_entry,
    fat_context * fc )
```

4.12 filesystem.h

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

```
1 /*
2  Stores the directory functions.
3  The directory maps the file information/name to the first block that stores the file in the FAT.
4  */
5 #ifndef FILESYSTEM_H
6 #define FILESYSTEM_H
7
8 #include <stdlib.h>
9 #include <string.h>
10 #include <stdio.h>
11 #include <unistd.h>
12 #include <stdbool.h>
13 #include <sys/types.h>
14 #include <fcntl.h>
15 #include <stdint.h>
16 #include <time.h>
17 #include "utils.h"
18
19 typedef struct block block;
20
21 struct block
22 {
23     char name[32];
24     uint32_t size;
25     uint16_t firstBlock;
26     uint8_t type;
27     uint8_t perm;
28     time_t mtime;
29 };
30
31 typedef struct fat_context fat_context;
32
33 struct fat_context
34 {
35     int BLOCK_SIZE;
36     int LSB;
37     int MSB;
38     int NUM_FAT_BLOCKS;
39     int NUM_DATA_BLOCKS;
40     int NUM_FAT;
41     int NUM_FAT_BYTES;
42     int NUM_DATA_BYTES;
43     int DIR_SIZE;
44     int DIRS_PER_BLOCK;
45     int fd;
46     char *empty_block;
47     char *empty_dir;
48 };
49
50 fat_context *constructor(int lsb, int num_fat_blocks, char *filename);
51
52 void initialize(fat_context *fc);
53
```

```

54 void seek_data_region(int block_index, fat_context *fc);
55
56 void seek_dir_data_region(int block_index, int dir_i, fat_context *fc);
57
58 void seek_to_write_data(int block_index, int curr_size, fat_context *fc);
59
60 uint16_t find_available_block(fat_context *fc);
61
62 void write_block_fat(int block_index, uint16_t value, fat_context *fc);
63
64 uint16_t read_fat_block(int block_index, fat_context *fc);
65
66 int find_last_block(int first_block, fat_context *fc);
67
68 void write_next_dir_entry(block *dir_entry, fat_context *fc);
69
70 int create(char *filename, uint8_t perm, fat_context *fc);
71
72 block *dir_to_struct(char *dir, fat_context *fc);
73
74 block *find_dir_entry(char *filename, fat_context *fc);
75
76 block *read_dir_entry(int dir_block, int dir_num, fat_context *fc);
77
78 void update_dir_entry(char *filename, block *dir_entry, fat_context *fc);
79
80 void clear_file(char *filename, fat_context *fc);
81
82 void fatwrite(char *filename, char *value, int len, fat_context *fc);
83
84 char *fatreadblock(int block_index, fat_context *fc);
85
86 void fatread(char *filename, fat_context *fc);
87
88 void fatremove(char *filename, fat_context *fc);
89
90 #endif // !FILESYSTEM_H

```

4.13 kernel.c File Reference

```

#include <sys/types.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/time.h>
#include "signals.h"
#include "kernel.h"
#include "sched.h"
#include "pcb.h"
#include "pcb_list.h"
#include "user_func.h"
#include "log.h"
#include "built_ins.h"
#include "shell.h"
#include "filefunc.h"
#include <string.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <stdio.h>
#include <ucontext.h>
#include <unistd.h>
#include <valgrind/valgrind.h>

```

Functions

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

Variables

- struct `pcb_list_node` * `head` = NULL
- int `max_pid`
- int `ticks` = 0
- `ucontext_t` `scheduler_context`
- struct `pcb` * `active_pcb`
- `ucontext_t` `main_context`
- FILE * `log_file` = 0
- `ucontext_t` `idle_context`
- `pid_t` `terminal_control` = 0

4.13.1 Function Documentation

4.13.1.1 `main()`

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

4.13.2 Variable Documentation

4.13.2.1 `active_pcb`

```
struct pcb* active_pcb
```

4.13.2.2 `head`

```
struct pcb_list_node* head = NULL
```

4.13.2.3 `idle_context`

```
ucontext_t idle_context
```

4.13.2.4 log_file

```
FILE* log_file = 0
```

4.13.2.5 main_context

```
ucontext_t main_context
```

4.13.2.6 max_pid

```
int max_pid
```

4.13.2.7 scheduler_context

```
ucontext_t scheduler_context
```

4.13.2.8 terminal_control

```
pid_t terminal_control = 0
```

4.13.2.9 ticks

```
int ticks = 0
```

4.14 kernel.h File Reference

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

Variables

- struct [pcb_list_node](#) * [queues](#) [3]
- struct [pcb_list_node](#) * [head](#)
- int [ticks](#)
- pid_t [terminal_control](#)

4.14.1 Variable Documentation

4.14.1.1 head

```
struct pcb_list_node* head [extern]
```

4.14.1.2 queues

```
struct pcb_list_node* queues[3] [extern]
```

4.14.1.3 terminal_control

```
pid_t terminal_control [extern]
```

4.14.1.4 ticks

```
int ticks [extern]
```

4.15 kernel.h

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

```
1 #ifndef KERNEL_H
2 #define KERNEL_H
3
4 #include <ucontext.h>
5 #include "pcb.h"
6
7 extern struct pcb_list_node *queues[3];
8 extern struct pcb_list_node *head;
9 extern int ticks;
10 extern pid_t terminal_control;
11
12 #endif // !KERNEL
```

4.16 kernel_func.c File Reference

```
#include "pcb.h"
#include "pcb_list.h"
#include "signals.h"
#include "kernel_func.h"
#include <signal.h>
#include <stdio.h>
#include "log.h"
#include "kernel.h"
#include <stdlib.h>
#include <sys/types.h>
#include <ucontext.h>
#include <valgrind/valgrind.h>
#include "string.h"
```

Functions

- struct [pcb](#) * [k_process_create](#) (struct [pcb](#) *parent)
NAME *k_process_create* DESCRIPTION Takes in parent pcb and creates and returns child pcb.
- void [k_process_cleanup_1](#) (struct [pcb](#) *process, char *command)
NAME *k_process_cleanup* DESCRIPTION called when a terminated/finished thread's resources needs to be cleaned up.
- void [k_process_kill](#) (struct [pcb](#) *process, int signal)
NAME *k_process_kill* DESCRIPTION kill the process referenced by process with the signal signal.
- void [wake_up_parent](#) ([pcb](#) *process)

4.16.1 Function Documentation

4.16.1.1 k_process_cleanup_1()

```
void k_process_cleanup_1 (
    struct pcb * process,
    char * command )
```

NAME *k_process_cleanup* DESCRIPTION called when a terminated/finished thread's resources needs to be cleaned up.

Such clean-up may include freeing memory, setting the status of the child, etc, FOUND IN [kernel_func.c](#) RETURNS does not return. ERRORS throws no errors.

4.16.1.2 k_process_create()

```
struct pcb * k_process_create (
    struct pcb * parent )
```

NAME *k_process_create* DESCRIPTION Takes in parent pcb and creates and returns child pcb.

Updates parent child pids as well. FOUND IN [kernel_func.c](#) RETURNS child process pcb. ERRORS throws no errors.

4.16.1.3 k_process_kill()

```
void k_process_kill (
    struct pcb * process,
    int signal )
```

NAME k_process_kill DESCRIPTION kill the process referenced by process with the signal signal.

FOUND IN kernel_func.c RETURNS does not return. ERRORS throws no errors.

4.16.1.4 wake_up_parent()

```
void wake_up_parent (
    pcb * process )
```

4.17 kernel_func.h File Reference

```
#include "pcb.h"
#include "pcb_list.h"
#include "signals.h"
#include "kernel.h"
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <ucontext.h>
#include <valgrind/valgrind.h>
```

Functions

- struct pcb * k_process_create (struct pcb *parent)
NAME k_process_create DESCRIPTION Takes in parent pcb and creates and returns child pcb.
- void k_process_kill (struct pcb *process, int signal)
NAME k_process_kill DESCRIPTION kill the process referenced by process with the signal signal.
- void k_process_cleanup (struct pcb *process)
- void k_process_cleanup_1 (struct pcb *process, char *command)
NAME k_process_cleanup DESCRIPTION called when a terminated/finished thread's resources needs to be cleaned up.
- void wake_up_parent (pcb *process)

Variables

- int max_pid
- ucontext_t scheduler_context

4.17.1 Function Documentation

4.17.1.1 k_process_cleanup()

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

4.17.1.2 k_process_cleanup_1()

```
void k_process_cleanup_1 (
    struct pcb * process,
    char * command )
```

NAME `k_process_cleanup` DESCRIPTION called when a terminated/finished thread's resources needs to be cleaned up.

Such clean-up may include freeing memory, setting the status of the child, etc, FOUND IN [kernel_func.c](#) RETURNS does not return. ERRORS throws no errors.

4.17.1.3 k_process_create()

```
struct pcb * k_process_create (
    struct pcb * parent )
```

NAME `k_process_create` DESCRIPTION Takes in parent pcb and creates and returns child pcb.

Updates parent child pids as well. FOUND IN [kernel_func.c](#) RETURNS child process pcb. ERRORS throws no errors.

4.17.1.4 k_process_kill()

```
void k_process_kill (
    struct pcb * process,
    int signal )
```

NAME `k_process_kill` DESCRIPTION kill the process referenced by process with the signal signal.

FOUND IN [kernel_func.c](#) RETURNS does not return. ERRORS throws no errors.

4.17.1.5 wake_up_parent()

```
void wake_up_parent (
    pcb * process )
```

4.17.2 Variable Documentation

4.17.2.1 max_pid

```
int max_pid [extern]
```

4.17.2.2 scheduler_context

```
ucontext_t scheduler_context [extern]
```

4.18 kernel_func.h

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

```
1 #ifndef KERNEL_FUNC_H
2 #define KERNEL_FUNC_H
3
4 #include "pcb.h"
5 #include "pcb_list.h"
6 #include "signals.h"
7 #include "kernel.h"
8 #include <signal.h>
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <sys/types.h>
12 #include <ucontext.h>
13 #include <valgrind/valgrind.h>
14
15 extern int max_pid;
16 extern ucontext_t scheduler_context;
17
18 struct pcb *k_process_create(struct pcb *parent);
19
20 void k_process_kill(struct pcb *process, int signal);
21
22 void k_process_cleanup(struct pcb *process);
23
24 void k_process_cleanup_1(struct pcb *process, char *command);
25
26 void k_process_kill(pcb *process, int signal);
27
28 void wake_up_parent(pcb *process);
29
30 #endif
```

4.19 log.c File Reference

```
#include "pcb.h"
#include "kernel.h"
#include "sched.h"
#include "log.h"
#include "stdio.h"
#include "string.h"
```

Functions

- void [log_command](#) (char *command, [pcb](#) *process, int prev_nice)

4.19.1 Function Documentation

4.19.1.1 log_command()

```
void log_command (
    char * command,
    pcb * process,
    int prev_nice )
```

4.20 log.h File Reference

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

Macros

- #define [SCHEDULE](#) "SCHEDULE"
- #define [CREATE](#) "CREATE"
- #define [SIGNALLED](#) "SIGNALLED"
- #define [EXITED](#) "EXITED"
- #define [ZOMBIE](#) "ZOMBIE"
- #define [NICE](#) "NICE"
- #define [ORPHAN](#) "ORPHAN"
- #define [WAITED](#) "WAITED"
- #define [BLOCKED_LOG](#) "BLOCKED"
- #define [UNBLOCKED](#) "UNBLOCKED"
- #define [STOPPED_LOG](#) "STOPPED"
- #define [CONTINUED](#) "CONTINUED"

Functions

- void [log_command](#) (char *command, [pcb](#) *process, int prev_nice)

Variables

- FILE * [log_file](#)

4.20.1 Macro Definition Documentation

4.20.1.1 BLOCKED_LOG

```
#define BLOCKED_LOG "BLOCKED"
```

4.20.1.2 CONTINUED

```
#define CONTINUED "CONTINUED"
```

4.20.1.3 CREATE

```
#define CREATE "CREATE"
```

4.20.1.4 EXITED

```
#define EXITED "EXITED"
```

4.20.1.5 NICE

```
#define NICE "NICE"
```

4.20.1.6 ORPHAN

```
#define ORPHAN "ORPHAN"
```

4.20.1.7 SCHEDULE

```
#define SCHEDULE "SCHEDULE"
```

4.20.1.8 SIGNED

```
#define SIGNED "SIGNED"
```

4.20.1.9 STOPPED_LOG

```
#define STOPPED_LOG "STOPPED"
```

4.20.1.10 UNBLOCKED

```
#define UNBLOCKED "UNBLOCKED"
```

4.20.1.11 WAITED

```
#define WAITED "WAITED"
```

4.20.1.12 ZOMBIE

```
#define ZOMBIE "ZOMBIE"
```

4.20.2 Function Documentation

4.20.2.1 log_command()

```
void log_command (
    char * command,
    pcb * process,
    int prev_nice )
```

4.20.3 Variable Documentation

4.20.3.1 log_file

```
FILE* log_file [extern]
```

4.21 log.h

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

```
1 #ifndef LOG_H
2 #define LOG_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include "pcb.h"
7
8 extern FILE *log_file;
9
10 void log_command(char *command, pcb *process, int prev_nice);
11
12 #define SCHEDULE "SCHEDULE"
13 #define CREATE "CREATE"
14 #define SIGNED "SIGNED"
15 #define EXITED "EXITED"
16 #define ZOMBIE "ZOMBIE"
17 #define NICE "NICE"
18 #define ORPHAN "ORPHAN"
19 #define WAITED "WAITED"
20 #define BLOCKED_LOG "BLOCKED"
21 #define UNBLOCKED "UNBLOCKED"
22 #define STOPPED_LOG "STOPPED"
23 #define CONTINUED "CONTINUED"
24
25 #endif // !LOG_H
```

4.22 parser.h File Reference

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

Classes

- struct [parsed_command](#)

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

Macros

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

Functions

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

*Arguments: cmd_line: a null-terminated string that is the command line result: a non-null pointer to a struct [parsed_command](#) **

- void [print_parsed_command](#) (const struct [parsed_command](#) *cmd)

4.22.1 Macro Definition Documentation

4.22.1.1 EXPECT_COMMANDS

```
#define EXPECT_COMMANDS 7
```

4.22.1.2 EXPECT_INPUT_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

4.22.1.3 EXPECT_OUTPUT_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

4.22.1.4 UNEXPECTED_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

4.22.1.5 UNEXPECTED_FILE_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

4.22.1.6 UNEXPECTED_FILE_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

4.22.1.7 UNEXPECTED_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

4.22.2 Function Documentation

4.22.2.1 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.

4.22.2.2 print_parsed_command()

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

4.23 parser.h

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

```
1 /* Penn-Shell Parser
2    hanbangw, 21fa */
3
4 #pragma once
5
6 #include <stddef.h>
7 #include <stdbool.h>
8
9 /* Here defines all possible parser errors */
10 // parser encountered an unexpected file input token '<'
11 #define UNEXPECTED_FILE_INPUT 1
12
13 // parser encountered an unexpected file output token '>'
14 #define UNEXPECTED_FILE_OUTPUT 2
15
16 // parser encountered an unexpected pipeline token '|'
17 #define UNEXPECTED_PIPELINE 3
18
19 // parser encountered an unexpected ampersand token '&'
20 #define UNEXPECTED_AMPERSAND 4
21
22 // parser didn't find input filename following '<'
23 #define EXPECT_INPUT_FILENAME 5
24
```

```

25 // parser didn't find output filename following '>' or '>>'
26 #define EXPECT_OUTPUT_FILENAME 6
27
28 // parser didn't find any commands or arguments where it expects one
29 #define EXPECT_COMMANDS 7
30
31 struct parsed_command {
32     // if the command shall be executed in background
33     // (ends with an ampersand '&')
34     bool is_background;
35
36     // if the stdout_file shall be opened in append mode
37     // ignore this value when stdout_file is NULL
38     bool is_file_append;
39
40     // filename for redirecting input from
41     const char *stdin_file;
42
43     // filename for redirecting output to
44     const char *stdout_file;
45
46     // number of commands (pipeline stages)
47     size_t num_commands;
48
49     // an array to a list of arguments
50     // size of 'commands' is 'num_commands'
51     char **commands[];
52 };
53
54 int parse_command(const char *cmd_line, struct parsed_command **result);
55
56 /* This is a debugging function used for outputting a parsed command line. */
57 void print_parsed_command(const struct parsed_command *cmd);

```

4.24 pcb.h File Reference

```

#include <stdbool.h>
#include <sys/types.h>
#include <ucontext.h>

```

Classes

- struct [pcb](#)
PCB struct.

Macros

- #define [TERM_NORMAL](#) 10
- #define [TERM_SIGNALED](#) 20
- #define [RUNNING](#) 3
- #define [STOPPED](#) 4
- #define [BLOCKED](#) 5
- #define [ZOMBIED](#) 6

Typedefs

- typedef struct [pcb](#) [pcb](#)
PCB struct.

4.24.1 Macro Definition Documentation

4.24.1.1 BLOCKED

```
#define BLOCKED 5
```

4.24.1.2 RUNNING

```
#define RUNNING 3
```

4.24.1.3 STOPPED

```
#define STOPPED 4
```

4.24.1.4 TERM_NORMAL

```
#define TERM_NORMAL 10
```

4.24.1.5 TERM_SIGNALED

```
#define TERM_SIGNALED 20
```

4.24.1.6 ZOMBIED

```
#define ZOMBIED 6
```

4.24.2 Typedef Documentation

4.24.2.1 pcb

typedef struct pcb pcb

PCB struct.

4.25 pcb.h

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

```

1 #ifndef PCB_H
2 #define PCB_H
3
4 #include <stdbool.h>
5 #include <sys/types.h>
6 #include <ucontext.h>
7
8 #define TERM_NORMAL 10
9 #define TERM_SIGNALED 20
10 #define RUNNING 3
11 #define STOPPED 4
12 #define BLOCKED 5
13 #define ZOMBIED 6
14
15 typedef struct pcb
16 {
17     ucontext_t *context;
18     struct pcb *parent;
19     pid_t pid;
20     int num_children;
21     pid_t *child_pids;
22     int num_zombies;
23     pid_t *zombies;
24     int fds[2]; // fd set to -1 as default
25     int priority;
26     int status;
27     int term_status;
28     bool waiting;
29     int num_to_wait_for;
30     pid_t *to_wait_for;
31     pid_t woke_up_by;
32     pid_t waiting_for;
33     bool updated_flag; // flag set to 1 if pcb status was updated
34     bool time_expired;
35     char *name;
36     int blocked_until;
37     bool sleeping;
38     // add more later
39 } pcb;
40 #endif

```

4.26 pcb_list.c File Reference

```

#include "pcb.h"
#include "log.h"
#include "pcb_list.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <ucontext.h>

```

Functions

- struct [pcb_list_node](#) * [add_pcb](#) (struct [pcb_list_node](#) *[head](#), struct [pcb](#) *[pcb](#))
- struct [pcb](#) * [get_pcb_from_pid](#) (struct [pcb_list_node](#) *[head](#), int [pid](#))
- struct [pcb_list_node](#) * [remove_pcb_from_pid](#) (struct [pcb_list_node](#) *[head](#), int [pid](#))
- [pcb_list_node](#) * [soft_remove](#) ([pcb_list_node](#) *[head](#), int [pid](#))

4.26.1 Function Documentation

4.26.1.1 add_pcb()

```
struct pcb_list_node * add_pcb (
    struct pcb_list_node * head,
    struct pcb * pcb )
```

4.26.1.2 get_pcb_from_pid()

```
struct pcb * get_pcb_from_pid (
    struct pcb_list_node * head,
    int pid )
```

4.26.1.3 remove_pcb_from_pid()

```
struct pcb_list_node * remove_pcb_from_pid (
    struct pcb_list_node * head,
    int pid )
```

4.26.1.4 soft_remove()

```
pcb_list_node * soft_remove (
    pcb_list_node * head,
    int pid )
```

4.27 pcb_list.h File Reference

```
#include "pcb.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <ucontext.h>
```

Classes

- struct `pcb_list_node`

Typedefs

- typedef struct [pcb_list_node](#) [pcb_list_node](#)

Functions

- struct [pcb_list_node](#) * [add_pcb](#) (struct [pcb_list_node](#) **head*, struct [pcb](#) **pcb*)
- struct [pcb](#) * [get_pcb_from_pid](#) (struct [pcb_list_node](#) **head*, int *pid*)
- struct [pcb_list_node](#) * [remove_pcb_from_pid](#) (struct [pcb_list_node](#) **head*, int *pid*)
- [pcb_list_node](#) * [soft_remove](#) ([pcb_list_node](#) **head*, int *pid*)

4.27.1 Typedef Documentation

4.27.1.1 [pcb_list_node](#)

```
typedef struct pcb\_list\_node pcb\_list\_node
```

4.27.2 Function Documentation

4.27.2.1 [add_pcb\(\)](#)

```
struct pcb\_list\_node * add\_pcb (  
    struct pcb\_list\_node * head,  
    struct pcb * pcb )
```

4.27.2.2 [get_pcb_from_pid\(\)](#)

```
struct pcb * get\_pcb\_from\_pid (  
    struct pcb\_list\_node * head,  
    int pid )
```

4.27.2.3 [remove_pcb_from_pid\(\)](#)

```
struct pcb\_list\_node * remove\_pcb\_from\_pid (  
    struct pcb\_list\_node * head,  
    int pid )
```

4.27.2.4 soft_remove()

```
pcb_list_node * soft_remove (
    pcb_list_node * head,
    int pid )
```

4.28 pcb_list.h

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

```
1 #ifndef PCB_LIST_H
2 #define PCB_LIST_H
3
4 #include "pcb.h"
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <sys/types.h>
8 #include <sys/types.h>
9 #include <ucontext.h>
10
11 typedef struct pcb_list_node
12 {
13     struct pcb_list_node *next;
14     struct pcb *pcb;
15 } pcb_list_node;
16
17 struct pcb_list_node *add_pcb(struct pcb_list_node *head, struct pcb *pcb);
18
19 struct pcb *get_pcb_from_pid(struct pcb_list_node *head, int pid);
20
21 struct pcb_list_node *remove_pcb_from_pid(struct pcb_list_node *head, int pid);
22
23 pcb_list_node *soft_remove(pcb_list_node *head, int pid);
24
25 #endif
```

4.29 pennfat.c File Reference

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

Macros

- #define [MAXLINELENGTH](#) 4096

Functions

- void `mkfs` (char *fs_name, int blocks_in_fat, int block_size_config)
- void `fake_initialize` ()
- void `mount` (char *fs_name)
- void `unmount` ()
- void `touch` (char **files, int num_files)
- void `mv` (char *source, char *dest)
- void `rm` (char *file)
- void `cp` (char *source, char *dest, bool source_os, bool dest_os)
- void `cat_from_terminal` (char *to_write, char *output, bool overwrite)
- void `cat` (char **files, int num_files, char *output, bool overwrite)
- void `ls` ()
- void `chmod` (char *filename, int perm)
- int `count_args` (char *str, int length)
- int `main` (int argc, char **argv)

Variables

- const uint16_t `eof` = 0xFFFF
- uint16_t * `fat`
- fat_context * `fc`

4.29.1 Macro Definition Documentation

4.29.1.1 MAXLINELENGTH

```
#define MAXLINELENGTH 4096
```

4.29.2 Function Documentation

4.29.2.1 cat()

```
void cat (  
    char ** files,  
    int num_files,  
    char * output,  
    bool overwrite )
```

4.29.2.2 cat_from_terminal()

```
void cat_from_terminal (
    char * to_write,
    char * output,
    bool overwrite )
```

4.29.2.3 chmod()

```
void chmod (
    char * filename,
    int perm )
```

4.29.2.4 count_args()

```
int count_args (
    char * str,
    int length )
```

4.29.2.5 cp()

```
void cp (
    char * source,
    char * dest,
    bool source_os,
    bool dest_os )
```

4.29.2.6 fake_initialize()

```
void fake_initialize ( )
```

4.29.2.7 ls()

```
void ls ( )
```

4.29.2.8 main()

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

4.29.2.9 mkfs()

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

4.29.2.10 mount()

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

4.29.2.11 mv()

```
void mv (
    char * source,
    char * dest )
```

4.29.2.12 rm()

```
void rm (
    char * file )
```

4.29.2.13 touch()

```
void touch (
    char ** files,
    int num_files )
```


4.29.2.14 unmount()

```
void unmount ( )
```

4.29.3 Variable Documentation

4.29.3.1 eof

```
const uint16_t eof = 0xFFFF
```

4.29.3.2 fat

```
uint16_t* fat
```

4.29.3.3 fc

```
fat_context* fc
```

4.30 sched.c File Reference

```
#include <stdio.h>
#include "sched.h"
#include <time.h>
#include <stdlib.h>
#include "kernel.h"
#include "pcb_list.h"
#include "user_func.h"
#include "kernel_func.h"
#include <string.h>
#include "log.h"
```

Functions

- void [handleFinish](#) ()
- void [unblock](#) ()
- void [schedule](#) (void)

Variables

- struct `pcb_list_node` * `queues` [] = {NULL, NULL, NULL}

4.30.1 Function Documentation

4.30.1.1 `handleFinish()`

```
void handleFinish ( )
```

4.30.1.2 `schedule()`

```
void schedule (
    void )
```

4.30.1.3 `unblock()`

```
void unblock ( )
```

4.30.2 Variable Documentation

4.30.2.1 `queues`

```
struct pcb_list_node* queues[] = {NULL, NULL, NULL}
```

4.31 `sched.h` File Reference

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

Functions

- void `schedule` (void)

Variables

- struct `pcb` * `active_pcb`
- struct `ucontext_t` `main_context`
- struct `ucontext_t` `idle_context`

4.31.1 Function Documentation

4.31.1.1 `schedule()`

```
void schedule (  
    void )
```

4.31.2 Variable Documentation

4.31.2.1 `active_pcb`

```
struct pcb* active_pcb [extern]
```

4.31.2.2 `idle_context`

```
struct ucontext_t idle_context [extern]
```

4.31.2.3 `main_context`

```
struct ucontext_t main_context [extern]
```

4.32 sched.h

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

```
1 #ifndef SCHED_H  
2 #define SCHED_H  
3  
4 #include "pcb.h"  
5 #include <ucontext.h>  
6  
7 extern struct pcb *active_pcb;  
8 extern struct ucontext_t main_context;  
9 extern struct ucontext_t idle_context;  
10  
11 void schedule(void);  
12  
13 #endif // !SCHED
```

4.33 shell.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include "shell.h"
#include "kernel.h"
#include "user_func.h"
#include "kernel_func.h"
#include "parser.h"
#include "signals.h"
#include "log.h"
#include "filefunc.h"
```

Functions

- void [parent_sigstsp_handler](#) (int signo)
- void [parent_sigint_handler](#) (int signo)
- void [runOnForeground](#) ([Group](#) *group)

This function takes a group, runs it in the foreground, waits for it to complete and then returns terminal control to the parent.

- void [shell](#) (int argc, char *arg[])

Variables

- int [lastJob](#) = -1
- bool [stoppedJobs](#) = false
- int [begin](#) = 1

4.33.1 Function Documentation

4.33.1.1 [parent_sigint_handler\(\)](#)

```
void parent_sigint_handler (
    int signo )
```

4.33.1.2 [parent_sigstsp_handler\(\)](#)

```
void parent_sigstsp_handler (
    int signo )
```

4.33.1.3 runOnForeground()

```
void runOnForeground (
    Group * group )
```

This function takes a group, runs it in the foreground, waits for it to complete and then returns terminal control to the parent.

This is called if a job is run on the foreground or if fg is called.

This called if the user runs a job normally in the foreground or when the fg command is called

Parameters

<i>group</i>	the group we want to move to the foreground
--------------	---

Returns

void.

4.33.1.4 shell()

```
void shell (
    int argc,
    char * arg[] )
```

4.33.2 Variable Documentation

4.33.2.1 begin

```
int begin = 1
```

4.33.2.2 lastJob

```
int lastJob = -1
```

4.33.2.3 stoppedJobs

```
bool stoppedJobs = false
```

4.34 shell.h File Reference

```
#include "stdbool.h"
```

Classes

- struct [Group](#)
- struct [Node](#)
- struct [List](#)

Macros

- #define [COMMAND_LENGTH](#) 4096
- #define [RETURN_ASCII_CODE](#) 10
- #define [SHELL_STOPPED](#) 101
- #define [SHELL_RUNNING](#) 102
- #define [SHELL_RESTARTING](#) 103
- #define [SHELL_BACKGROUND_TO_FOREGROUND](#) 104
- #define [SHELL_FINISHED](#) 105

Typedefs

- typedef struct [Group](#) [Group](#)
- typedef struct [Node](#) [Node](#)
- typedef struct [List](#) [List](#)

Functions

- void [shell](#) (int argc, char *arg[])
- void [execute](#) (char **command, int outputFile, int inputFile, int readFd[], int writeFd[], bool writing, bool reading, [Group](#) *group, int index)

Creates a new process for the given command, sets the file descriptors and modifies the group struct as necessary.

- void [Init](#) ([List](#) *s)
- void [RemoveById](#) (int id, [List](#) *s)
- void [Remove](#) ([Node](#) *node, [List](#) *s)
- void [Add](#) ([List](#) *s, [Group](#) *g)
- bool [isEmpty](#) ([List](#) *s)
- [Node](#) * [Peek](#) ([List](#) *s)
- void [printAll](#) ([List](#) *list)
- void [clear](#) ([List](#) *list)
- void [report](#) ([List](#) *list)

Print updates about all the jobs before prompting the user.

- void [updateStatuses](#) ([List](#) *list)

This will update the status of the jobs.

- void [printJobs](#) ([List](#) *list)

This prints all the jobs in the linked list with their indices and status.

Variables

- int [childId](#)
- int [childStatus](#)
- int [lastJob](#)

4.34.1 Macro Definition Documentation

4.34.1.1 COMMAND_LENGTH

```
#define COMMAND_LENGTH 4096
```

4.34.1.2 RETURN_ASCII_CODE

```
#define RETURN_ASCII_CODE 10
```

4.34.1.3 SHELL_BACKGROUND_TO_FOREGROUND

```
#define SHELL_BACKGROUND_TO_FOREGROUND 104
```

4.34.1.4 SHELL_FINISHED

```
#define SHELL_FINISHED 105
```

4.34.1.5 SHELL_RESTARTING

```
#define SHELL_RESTARTING 103
```

4.34.1.6 SHELL_RUNNING

```
#define SHELL_RUNNING 102
```

4.34.1.7 SHELL_STOPPED

```
#define SHELL_STOPPED 101
```

4.34.2 Typedef Documentation

4.34.2.1 Group

```
typedef struct Group Group
```

4.34.2.2 List

```
typedef struct List List
```

4.34.2.3 Node

```
typedef struct Node Node
```

4.34.3 Function Documentation

4.34.3.1 Add()

```
void Add (  
    List * s,  
    Group * g )
```

4.34.3.2 clear()

```
void clear (  
    List * list )
```


4.34.3.3 execute()

```
void execute (
    char ** command_,
    int outputFile,
    int inputFile,
    int readFd[],
    int writeFd[],
    bool writing,
    bool reading,
    Group * group,
    int index )
```

Creates a new process for the given command, sets the file descriptors and modifies the group struct as necessary.

Parameters

<i>command</i>	the command to execute
<i>outputFile</i>	The redirection file it is reading from. -1 if there is no redirection input.
<i>inputFile</i>	The redirection file it is writing to. -1 if there is no redirection output.
<i>readFd</i>	The descriptors of the file it is reading to if there is a pipeline.
<i>readFd</i>	The descriptors of the file it is writing to if there is a pipeline.
<i>reading</i>	Wether or not we are going to read from the pipe file or not (true for all process in a pipe except the first)
<i>writing</i>	Wether or not we are going to write to the pipe file or not (true for all process in a pipe except the last)
<i>group</i>	The group struct of the pipeline
<i>index</i>	The index of the process within the struct

4.34.3.4 Init()

```
void Init (
    List * s )
```

4.34.3.5 isEmpty()

```
bool isEmpty (
    List * s )
```

4.34.3.6 Peek()

```
Node * Peek (
    List * s )
```

4.34.3.7 printAll()

```
void printAll (
    List * list )
```

4.34.3.8 printJobs()

```
void printJobs (
    List * list )
```

This prints all the jobs in the linked list with their indices and status.

It should be called when the user enters the command jobs

Parameters

<i>list</i>	the linked list with all the jobs
-------------	-----------------------------------

Returns

void.

4.34.3.9 Remove()

```
void Remove (
    Node * node,
    List * s )
```

4.34.3.10 RemoveById()

```
void RemoveById (
    int id,
    List * s )
```

4.34.3.11 report()

```
void report (
    List * list )
```

Print updates about all the jobs before prompting the user.

Will only print the jobs whose status has changed

Parameters

<i>list</i>	the linked list with all the jobs
-------------	-----------------------------------

Returns

void.

4.34.3.12 shell()

```
void shell (
    int argc,
    char * arg[] )
```

4.34.3.13 updateStatuses()

```
void updateStatuses (
    List * list )
```

This will update the status of the jobs.

It will loop over the struct and kill all zombies and will update their status accordingly so that when will report statuses the next time, we correctly tell the user

Parameters

<i>list</i>	the linked list with all the jobs
-------------	-----------------------------------

Returns

void.

4.34.4 Variable Documentation

4.34.4.1 childId

```
int childId [extern]
```

4.34.4.2 childStatus

```
int childStatus [extern]
```

4.34.4.3 lastJob

```
int lastJob [extern]
```

4.35 shell.h

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

```

1 #ifndef SHELL_H
2 #define SHELL_H
3
4 #define COMMAND_LENGTH 4096
5 #define RETURN_ASCII_CODE 10
6
7 #define SHELL_STOPPED 101
8 #define SHELL_RUNNING 102
9 #define SHELL_RESTARTING 103
10 #define SHELL_BACKGROUND_TO_FOREGROUND 104
11 #define SHELL_FINISHED 105
12
13 #include "stdbool.h"
14
15 // Queue is a queue that stores integers that is
16 // implemented with a singly linked list
17 // a user can create a queue, add integers to the end,
18 // remove integers from the front, clear the queue,
19 // and print all values in it.
20
21 void shell(int argc, char *arg[]);
22
23 typedef struct Group
24 {
25     int id;
26     int *ids;
27     char *name;
28     int status;
29     int size;
30     bool changed;
31 } Group;
32
33
34 typedef struct Node
35 {
36     struct Node *next;
37     struct Node *prev;
38     Group *group;
39 } Node;
40
41 typedef struct List
42 {
43     Node *head;
44     Node *tail;
45 } List;
46
47 void execute(char **command, int outputFile, int inputFile, int readFd[], int writeFd[], bool writing,
48             bool reading, Group *group, int index);
49
50 // Normally it is good practice to have comments
51 // in here to explain the behaviour of each function
52 void Init(List *s);
53 void RemoveById(int id, List *s);
54 void Remove(Node *node, List *s);
55 void Add(List *s, Group *g);
56 bool isEmpty(List *s);
57 Node *Peek(List *s);
58 void printAll(List *list);
59 void clear(List *list);
60 void report(List *list);
61
62 void updateStatuses(List *list);
63
64 void printJobs(List *list);
65
66 #endif
67
68 extern int childId;
69 extern int childStatus;
70 extern int lastJob;

```

4.36 shell_execute.c File Reference

```

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

```

```
#include <sys/types.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>
#include "shell.h"
#include "parser.h"
#include <fcntl.h>
#include "user_func.h"
#include "filefunc.h"
#include "stress.h"
#include "built_ins.h"
```

Functions

- int [create_child](#) (char **command, char **args, int inputFile, int outputFile)
- void [execute](#) (char **command_, int outputFile, int inputFile, int readFd[], int writeFd[], bool writing, bool reading, [Group](#) *group, int index)

Creates a new process for the given command, sets the file descriptors and modifies the group struct as necessary.

4.36.1 Function Documentation

4.36.1.1 [create_child\(\)](#)

```
int create_child (
    char ** command,
    char ** args,
    int inputFile,
    int outputFile )
```

4.36.1.2 [execute\(\)](#)

```
void execute (
    char ** command_,
    int outputFile,
    int inputFile,
    int readFd[],
    int writeFd[],
    bool writing,
    bool reading,
    Group * group,
    int index )
```

Creates a new process for the given command, sets the file descriptors and modifies the group struct as necessary.

Parameters

<i>command</i>	the command to execute
<i>outputFile</i>	The redirection file it is reading from. -1 if there is no redirection input.
<i>inputFile</i>	The redirection file it is writing to. -1 if there is no redirection output.
<i>readFd</i>	The descriptors of the file it is reading to if there is a pipeline.
<i>readFd</i>	The descriptors of the file it is writing to if there is a pipeline.
<i>reading</i>	Wether or not we are going to read from the pipe file or not (true for all process in a pipe except the first)
<i>writing</i>	Wether or not we are going to write to the pipe file or not (true for all process in a pipe except the last)
<i>group</i>	The group struct of the pipeline
<i>index</i>	The index of the process within the struct

4.37 shell_list.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "shell.h"
```

Functions

- void [Init](#) ([List](#) *s)
- void [Add](#) ([List](#) *s, [Group](#) *g)
- [Node](#) * [Peek](#) ([List](#) *s)
- void [RemoveByld](#) (int id, [List](#) *list)
- bool [isEmpty](#) ([List](#) *s)
- void [printAll](#) ([List](#) *list)
- void [clear](#) ([List](#) *list)

4.37.1 Function Documentation

4.37.1.1 Add()

```
void Add (
    List * s,
    Group * g )
```

4.37.1.2 clear()

```
void clear (  
    List * list )
```

4.37.1.3 Init()

```
void Init (  
    List * s )
```

4.37.1.4 isEmpty()

```
bool isEmpty (  
    List * s )
```

4.37.1.5 Peek()

```
Node * Peek (  
    List * s )
```

4.37.1.6 printAll()

```
void printAll (  
    List * list )
```

4.37.1.7 RemoveById()

```
void RemoveById (  
    int id,  
    List * list )
```


4.38 shell_notif.c File Reference

```
#include <string.h>
#include <stdio.h>
#include <stdbool.h>
#include <sys/types.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>
#include "shell.h"
#include "log.h"
#include "user_func.h"
#include "kernel_func.h"
```

Functions

- void `report` (`List *list`)
Print updates about all the jobs before prompting the user.
- void `updateStatuses` (`List *list`)
This will update the status of the jobs.
- void `printJobs` (`List *list`)
This prints all the jobs in the linked list with their indices and status.

4.38.1 Function Documentation

4.38.1.1 `printJobs()`

```
void printJobs (
    List * list )
```

This prints all the jobs in the linked list with their indices and status.

It should be called when the user enters the command jobs

Parameters

<i>list</i>	the linked list with all the jobs
-------------	-----------------------------------

Returns

void.

4.38.1.2 report()

```
void report (
    List * list )
```

Print updates about all the jobs before prompting the user.

Will only print the jobs whose status has changed

Parameters

<i>list</i>	the linked list with all the jobs
-------------	-----------------------------------

Returns

void.

4.38.1.3 updateStatuses()

```
void updateStatuses (
    List * list )
```

This will update the status of the jobs.

It will loop over the struct and kill all zombies and will update their status accordingly so that when will report statuses the next time, we correctly tell the user

Parameters

<i>list</i>	the linked list with all the jobs
-------------	-----------------------------------

Returns

void.

4.39 signals.c File Reference

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include "signals.h"
#include "user_func.h"
#include "sched.h"
#include "log.h"
```

Functions

- void [sigalrm_handler](#) (int signo)
- void [sigstop_handler](#) (int signo)
- void [sigint_handler](#) (int signo)

4.39.1 Function Documentation

4.39.1.1 sigalrm_handler()

```
void sigalrm_handler (  
    int signo )
```

4.39.1.2 sigint_handler()

```
void sigint_handler (  
    int signo )
```

4.39.1.3 sigstop_handler()

```
void sigstop_handler (  
    int signo )
```

4.40 signals.h File Reference

```
#include "pcb.h"  
#include <ucontext.h>  
#include "kernel.h"
```

Macros

- #define [S_SIGSTOP](#) 1
- #define [S_SIGCONT](#) 2
- #define [S_SIGTERM](#) 3

Functions

- void [sigalrm_handler](#) (int signo)
- void [sigstop_handler](#) (int signo)
- void [sigint_handler](#) (int signo)

Variables

- struct `pcb` * `active_pcb`
- `ucontext_t` `scheduler_context`

4.40.1 Macro Definition Documentation

4.40.1.1 S_SIGCONT

```
#define S_SIGCONT 2
```

4.40.1.2 S_SIGSTOP

```
#define S_SIGSTOP 1
```

4.40.1.3 S_SIGTERM

```
#define S_SIGTERM 3
```

4.40.2 Function Documentation

4.40.2.1 sigalrm_handler()

```
void sigalrm_handler (  
    int signo )
```

4.40.2.2 sigint_handler()

```
void sigint_handler (  
    int signo )
```

4.40.2.3 sigstop_handler()

```
void sigstop_handler (
    int signo )
```

4.40.3 Variable Documentation

4.40.3.1 active_pcb

```
struct pcb* active_pcb [extern]
```

4.40.3.2 scheduler_context

```
ucontext_t scheduler_context [extern]
```

4.41 signals.h

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

```
1 #ifndef SIGNALS_H
2 #define SIGNALS_H
3
4 #include "pcb.h"
5 #include <ucontext.h>
6 #include "kernel.h"
7
8 #define S_SIGSTOP 1
9 #define S_SIGCONT 2
10 #define S_SIGTERM 3
11
12 extern struct pcb *active_pcb;
13 extern ucontext_t scheduler_context;
14
15 void sigalrm_handler(int signo);
16 void sigstop_handler(int signo);
17 void sigint_handler(int signo);
18
19 #endif
```

4.42 stress.c File Reference

```
#include "stress.h"
#include <stdbool.h>
#include <stdio.h>
#include <unistd.h>
#include "kernel.h"
#include "user_func.h"
```

Functions

- void `hang` (void)
 - *Add commands `hang`, `nohang`, and `recur` to the shell as built-in subroutines * which call the following functions, respectively.*
- void `nohang` (void)
- void `recur` (void)

4.42.1 Function Documentation

4.42.1.1 `hang()`

```
void hang (  
    void )
```

- Add commands `hang`, `nohang`, and `recur` to the shell as built-in subroutines * which call the following functions, respectively.
- —

4.42.1.2 `nohang()`

```
void nohang (  
    void )
```

4.42.1.3 `recur()`

```
void recur (  
    void )
```

4.43 `stress.h` File Reference

Functions

- void `hang` (void)
 - *Add commands `hang`, `nohang`, and `recur` to the shell as built-in subroutines * which call the following functions, respectively.*
- void `nohang` (void)
- void `recur` (void)

4.43.1 Function Documentation

4.43.1.1 hang()

```
void hang (
    void )
```

- Add commands hang, nohang, and recur to the shell as built-in subroutines * which call the following functions, respectively.

- —

4.43.1.2 nohang()

```
void nohang (
    void )
```

4.43.1.3 recur()

```
void recur (
    void )
```

4.44 stress.h

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

```
1 #ifndef STRESS_H
2 #define STRESS_H
3
4 void hang(void);
5 void nohang(void);
6 void recur(void);
7
8 #endif
```

4.45 user_func.c File Reference

```
#include "kernel_func.h"
#include "pcb.h"
#include "log.h"
#include "pcb_list.h"
#include "signals.h"
#include "user_func.h"
#include <signal.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <ucontext.h>
#include <unistd.h>
#include <valgrind/valgrind.h>
#include <string.h>
#include "errno.h"
```

Functions

- void [p_perror](#) (char *toPrint)
- bool [check_pid_is_child](#) (struct [pcb](#) *parent, int pid)
- pid_t [p_spawn](#) (void(*func)(), char *argv[], int fd0, int fd1, char *name)
NAME p_spawn DESCRIPTION forks a new thread that retains most of the attributes of the parent thread (see [k_process_create](#)).
- pid_t [p_waitpid_1](#) (pid_t pid, int *wstatus, bool [nohang](#))
- pid_t [p_waitpid](#) (pid_t pid, int *wstatus, bool [nohang](#))
NAME p_waitpid DESCRIPTION sets the calling thread as blocked (if nohang is false) until a child of the calling thread changes state.
- bool [W_WIFEXITED](#) (int status)
- bool [W_WIFSTOPPED](#) (int status)
- bool [W_WIFSIGNALED](#) (int status)
- int [p_kill](#) (pid_t pid, int sig)
NAME p_kill DESCRIPTION kills a process with specified pid and signal.
- void [p_exit](#) ()
NAME p_exit DESCRIPTION exits the current thread unconditionally.
- void [p_nice](#) (pid_t pid, int priority)
NAME p_nice DESCRIPTION sets the priority of the thread pid to priority.
- void [p_sleep](#) (unsigned int len)
NAME p_sleep DESCRIPTION sets the calling process to blocked until ticks of the system clock elapse, and then sets the thread to running.

4.45.1 Function Documentation

4.45.1.1 check_pid_is_child()

```
bool check_pid_is_child (
    struct pcb * parent,
    int pid )
```

4.45.1.2 p_exit()

```
void p_exit ( )
```

NAME `p_exit` DESCRIPTION exits the current thread unconditionally.

FOUND IN [user_func.c](#) RETURNS successfully exits the terminal. ERRORS active process does not exist.

4.45.1.3 p_kill()

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

NAME `p_kill` DESCRIPTION kills a process with specified pid and signal.

FOUND IN [user_func.c](#) RETURNS 0 upon success and -1 upon error ERRORS pid is invalid

4.45.1.4 p_nice()

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

NAME `p_nice` DESCRIPTION sets the priority of the thread pid to priority.

FOUND IN [user_func.c](#) RETURNS returns upon successful completion. ERRORS fail to retrieve the process pcb.

4.45.1.5 p_perror()

```
void p_perror (
    char * toPrint )
```

4.45.1.6 p_sleep()

```
void p_sleep (
    unsigned int len )
```

NAME `p_sleep` DESCRIPTION sets the calling process to blocked until ticks of the system clock elapse, and then sets the thread to running.

`p_sleep` does not return until the thread resumes running; however, it can be interrupted by a `S_SIGTERM` signal. Like `sleep(3)` in Linux, the clock keeps ticking even when `p_sleep` is interrupted. FOUND IN [user_func.c](#) RETURNS it does not return. ERRORS throws no errors.

4.45.1.7 p_spawn()

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

NAME p_spawn DESCRIPTION forks a new thread that retains most of the attributes of the parent thread (see k_process_create).

Once the thread is spawned, it executes the function referenced by func with its argument array argv. fd0 is the file descriptor for the input file, and fd1 is the file descriptor for the output file. FOUND IN [user_func.c](#) RETURNS child pid on success ERRORS failed to fork and allocate necessary kernal structures because memory is tight.

4.45.1.8 p_waitpid()

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

NAME p_waitpid DESCRIPTION sets the calling thread as blocked (if nohang is false) until a child of the calling thread changes state.

It is similar to Linux waitpid(2). If nohang is true, p_waitpid does not block but returns immediately. FOUND IN [user_func.c](#) RETURNS child pid which has changed state on success, or nohang is true and there is no block. ERRORS child has not changed state on success because child does not have

4.45.1.9 p_waitpid_1()

```
pid_t p_waitpid_1 (
    pid_t pid,
    int * wstatus,
    bool nohang )
```

4.45.1.10 W_WIFEXITED()

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

4.45.1.11 W_WIFSIGNALED()

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

4.45.1.12 W_WIFSTOPPED()

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

4.46 user_func.h File Reference

```
#include "pcb.h"
#include "pcb_list.h"
#include <signal.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <ucontext.h>
#include <unistd.h>
#include <valgrind/valgrind.h>
```

Functions

- void [p_perror](#) (char *toPrint)
- bool [check_pid_is_child](#) (struct [pcb](#) *parent, int pid)
- pid_t [p_spawn](#) (void(*func)(), char *argv[], int fd0, int fd1, char *name)
NAME p_spawn DESCRIPTION forks a new thread that retains most of the attributes of the parent thread (see k_process_create).
- pid_t [p_waitpid](#) (pid_t pid, int *wstatus, bool [nohang](#))
NAME p_waitpid DESCRIPTION sets the calling thread as blocked (if nohang is false) until a child of the calling thread changes state.
- int [p_kill](#) (pid_t pid, int sig)
NAME p_kill DESCRIPTION kills a process with specified pid and signal.
- void [p_nice](#) (pid_t pid, int priority)
NAME p_nice DESCRIPTION sets the priority of the thread pid to priority.
- void [p_exit](#) ()
NAME p_exit DESCRIPTION exits the current thread unconditionally.
- void [p_sleep](#) (unsigned int len)
NAME p_sleep DESCRIPTION sets the calling process to blocked until ticks of the system clock elapse, and then sets the thread to running.
- bool [W_WIFEXITED](#) (int status)
- bool [W_WIFSTOPPED](#) (int status)
- bool [W_WIFSIGNALED](#) (int status)

Variables

- struct [pcb_list_node](#) * [head](#)
- struct [pcb](#) * [active_pcb](#)
- struct [pcb_list_node](#) * [queues](#) [3]

4.46.1 Function Documentation

4.46.1.1 `check_pid_is_child()`

```
bool check_pid_is_child (
    struct pcb * parent,
    int pid )
```

4.46.1.2 `p_exit()`

```
void p_exit ( )
```

NAME `p_exit` DESCRIPTION exits the current thread unconditionally.

FOUND IN [user_func.c](#) RETURNS successfully exits the terminal. ERRORS active process does not exist.

4.46.1.3 `p_kill()`

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

NAME `p_kill` DESCRIPTION kills a process with specified pid and signal.

FOUND IN [user_func.c](#) RETURNS 0 upon success and -1 upon error ERRORS pid is invalid

4.46.1.4 `p_nice()`

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

NAME `p_nice` DESCRIPTION sets the priority of the thread pid to priority.

FOUND IN [user_func.c](#) RETURNS returns upon successful completion. ERRORS fail to retrieve the process pcb.

4.46.1.5 `p_perror()`

```
void p_perror (
    char * toPrint )
```

4.46.1.6 p_sleep()

```
void p_sleep (
    unsigned int len )
```

NAME `p_sleep` DESCRIPTION sets the calling process to blocked until ticks of the system clock elapse, and then sets the thread to running.

`p_sleep` does not return until the thread resumes running; however, it can be interrupted by a `S_SIGTERM` signal. Like `sleep(3)` in Linux, the clock keeps ticking even when `p_sleep` is interrupted. FOUND IN [user_func.c](#) RETURNS it does not return. ERRORS throws no errors.

4.46.1.7 p_spawn()

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

NAME `p_spawn` DESCRIPTION forks a new thread that retains most of the attributes of the parent thread (see `k_process_create`).

Once the thread is spawned, it executes the function referenced by `func` with its argument array `argv`. `fd0` is the file descriptor for the input file, and `fd1` is the file descriptor for the output file. FOUND IN [user_func.c](#) RETURNS child pid on success ERRORS failed to fork and allocate necessary kernal structures because memory is tight.

4.46.1.8 p_waitpid()

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

NAME `p_waitpid` DESCRIPTION sets the calling thread as blocked (if `nohang` is false) until a child of the calling thread changes state.

It is similar to Linux `waitpid(2)`. If `nohang` is true, `p_waitpid` does not block but returns immediately. FOUND IN [user_func.c](#) RETURNS child pid which has changed state on success, or `nohang` is true and there is no block. ERRORS child has not changed state on success because child does not have

4.46.1.9 W_WIFEXITED()

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

4.46.1.10 W_WIFSIGNALED()

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

4.46.1.11 W_WIFSTOPPED()

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

4.46.2 Variable Documentation

4.46.2.1 active_pcb

```
struct pcb* active_pcb [extern]
```

4.46.2.2 head

```
struct pcb\_list\_node* head [extern]
```

4.46.2.3 queues

```
struct pcb\_list\_node* queues[3] [extern]
```

4.47 user_func.h

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

```
1 #ifndef USER_FUNC_H
2 #define USER_FUNC_H
3
4 #include "pcb.h"
5 #include "pcb_list.h"
6 #include <signal.h>
7 #include <stdbool.h>
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <sys/types.h>
11 #include <ucontext.h>
12 #include <unistd.h>
13 #include <valgrind/valgrind.h>
14
15 extern struct pcb_list_node *head;
16 extern struct pcb *active_pcb;
17 extern struct pcb_list_node *queues[3];
18
19 void p_perror(char* toPrint);
20
21 bool check_pid_is_child(struct pcb *parent, int pid);
22
23 pid_t p_spawn(void (*func)(), char *argv[], int fd0, int fd1, char *name);
24
25 pid_t p_waitpid(pid_t pid, int *wstatus, bool nohang);
26
27 int p_kill(pid_t pid, int sig);
28
29 void p_nice(pid_t pid, int priority);
30
31 void p_exit();
32
33 void p_sleep(unsigned int len);
34
35 bool W_WIFEXITED(int status);
36
37 bool W_WIFSTOPPED(int status);
38
39 bool W_WIFSIGNALED(int status);
40
41 #endif
```

4.48 utils.c File Reference

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

Functions

- uint32_t [char_to_32](#) (char a, char b, char c, char d)
- uint16_t [char_to_16](#) (char a, char b)
- uint8_t [char_to_8](#) (char a)
- bool [is_zero](#) (char *buf)

4.48.1 Function Documentation

4.48.1.1 char_to_16()

```
uint16_t char_to_16 (  
    char a,  
    char b )
```

4.48.1.2 char_to_32()

```
uint32_t char_to_32 (  
    char a,  
    char b,  
    char c,  
    char d )
```

4.48.1.3 char_to_8()

```
uint8_t char_to_8 (  
    char a )
```

4.48.1.4 is_zero()

```
bool is_zero (  
    char * buf )
```

4.49 utils.h File Reference

```
#include <stdio.h>  
#include <sys/types.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <string.h>  
#include <fcntl.h>  
#include <stdint.h>  
#include <stdbool.h>
```

Macros

- `#define MIN(x, y) (((x) < (y)) ? (x) : (y))`
- `#define MAX(x, y) (((x) > (y)) ? (x) : (y))`

Functions

- uint32_t [char_to_32](#) (char a, char b, char c, char d)
- uint16_t [char_to_16](#) (char a, char b)
- uint8_t [char_to_8](#) (char a)
- bool [is_zero](#) (char *buf)
- uint8_t [get_msb](#) (uint16_t val)

4.49.1 Macro Definition Documentation

4.49.1.1 MAX

```
#define MAX(  
    x,  
    y ) ((x) > (y)) ? (x) : (y)
```

4.49.1.2 MIN

```
#define MIN(  
    x,  
    y ) ((x) < (y)) ? (x) : (y)
```

4.49.2 Function Documentation

4.49.2.1 char_to_16()

```
uint16_t char_to_16 (  
    char a,  
    char b )
```

4.49.2.2 char_to_32()

```
uint32_t char_to_32 (  
    char a,  
    char b,  
    char c,  
    char d )
```

4.49.2.3 char_to_8()

```
uint8_t char_to_8 (  
    char a )
```

4.49.2.4 get_msb()

```
uint8_t get_msb (  
    uint16_t val )
```

4.49.2.5 is_zero()

```
bool is_zero (  
    char * buf )
```

4.50 utils.h

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

```
1 #ifndef UTIL_H  
2 #define UTIL_H  
3  
4 #include <stdio.h>  
5 #include <sys/types.h>  
6 #include <unistd.h>  
7 #include <stdlib.h>  
8 #include <string.h>  
9 #include <fcntl.h>  
10 #include <stdint.h>  
11 #include <stdbool.h>  
12 #define MIN(x, y) (((x) < (y)) ? (x) : (y))  
13 #define MAX(x, y) (((x) > (y)) ? (x) : (y))  
14  
15 uint32_t char_to_32(char a, char b, char c, char d);  
16  
17 uint16_t char_to_16(char a, char b);  
18  
19 uint8_t char_to_8(char a);  
20  
21 bool is_zero(char *buf);  
22  
23 uint8_t get_msb(uint16_t val);  
24  
25 #endif // !UTIL_H
```

Index

- active_pcb
 - kernel.c, [51](#)
 - sched.h, [75](#)
 - signals.h, [93](#)
 - user_func.h, [102](#)
- Add
 - shell.h, [80](#)
 - shell_list.c, [87](#)
- add_pcb
 - pcb_list.c, [67](#)
 - pcb_list.h, [68](#)
- begin
 - shell.c, [77](#)
- block, [5](#)
 - filesystem.h, [45](#)
 - firstBlock, [5](#)
 - mtime, [5](#)
 - name, [5](#)
 - perm, [6](#)
 - size, [6](#)
 - type, [6](#)
- BLOCK_SIZE
 - fat_context, [7](#)
- BLOCKED
 - pcb.h, [65](#)
- BLOCKED_LOG
 - log.h, [58](#)
- blocked_until
 - pcb, [14](#)
- built_ins.c, [19](#)
 - busy_process, [20](#)
 - cat_wrapper, [20](#)
 - chmod_wrapper, [20](#)
 - cp_wrapper, [20](#)
 - echo_wrapper, [20](#)
 - idle_process, [20](#)
 - invalid_cmd_process, [20](#)
 - kill_process, [21](#)
 - ls_wrapper, [21](#)
 - mv_wrapper, [21](#)
 - orphan_child, [21](#)
 - orphanify, [21](#)
 - parse_script, [21](#)
 - ps_process, [21](#)
 - rm_wrapper, [22](#)
 - sleep_process, [22](#)
 - touch_wrapper, [22](#)
 - zombie_child, [22](#)
 - zombify, [22](#)
- built_ins.h, [23](#)
 - busy_process, [23](#)
 - cat_wrapper, [23](#)
 - chmod_wrapper, [23](#)
 - cp_wrapper, [23](#)
 - echo_wrapper, [24](#)
 - idle_process, [24](#)
 - invalid_cmd_process, [24](#)
 - ls_wrapper, [24](#)
 - mv_wrapper, [24](#)
 - orphanify, [24](#)
 - parse_script, [24](#)
 - ps_process, [25](#)
 - rm_wrapper, [25](#)
 - sleep_process, [25](#)
 - touch_wrapper, [25](#)
 - zombify, [25](#)
- busy_process
 - built_ins.c, [20](#)
 - built_ins.h, [23](#)
- cat
 - pennfat.c, [70](#)
- cat_from_terminal
 - pennfat.c, [70](#)
- cat_wrapper
 - built_ins.c, [20](#)
 - built_ins.h, [23](#)
- changed
 - Group, [10](#)
- char_to_16
 - utils.c, [103](#)
 - utils.h, [105](#)
- char_to_32
 - utils.c, [104](#)
 - utils.h, [105](#)
- char_to_8
 - utils.c, [104](#)
 - utils.h, [105](#)
- check_pid_is_child
 - user_func.c, [96](#)
 - user_func.h, [100](#)
- child_pids
 - pcb, [14](#)
- childId
 - shell.h, [84](#)
- childStatus
 - shell.h, [84](#)
- chmod
 - pennfat.c, [71](#)

- chmod_wrapper
 - built_ins.c, [20](#)
 - built_ins.h, [23](#)
- clear
 - shell.h, [80](#)
 - shell_list.c, [87](#)
- clear_data_region
 - filesystem.c, [40](#)
- clear_file
 - filesystem.c, [40](#)
 - filesystem.h, [45](#)
- COMMAND_LENGTH
 - shell.h, [79](#)
- commands
 - parsed_command, [12](#)
- constructor
 - filesystem.c, [41](#)
 - filesystem.h, [46](#)
- context
 - pcb, [14](#)
- CONTINUED
 - log.h, [59](#)
- count_args
 - pennfat.c, [71](#)
- cp
 - pennfat.c, [71](#)
- cp_wrapper
 - built_ins.c, [20](#)
 - built_ins.h, [23](#)
- CREATE
 - log.h, [59](#)
- create
 - filesystem.c, [41](#)
 - filesystem.h, [46](#)
- create_child
 - shell_execute.c, [86](#)
- curr_data_ptr
 - fd_entry, [9](#)
- DIR_SIZE
 - fat_context, [7](#)
- dir_to_struct
 - filesystem.c, [41](#)
 - filesystem.h, [46](#)
- DIRS_PER_BLOCK
 - fat_context, [7](#)
- E2BIG
 - errno.h, [27](#)
- echo_wrapper
 - built_ins.c, [20](#)
 - built_ins.h, [24](#)
- EEOF
 - errno.h, [27](#)
- EFBIG
 - errno.h, [27](#)
- EFTF
 - errno.h, [27](#)
- EINVPERM
- errno.h, [27](#)
- empty_block
 - fat_context, [7](#)
- empty_dir
 - fat_context, [7](#)
- ENAMETOOLONG
 - errno.h, [28](#)
- ENOENT
 - errno.h, [28](#)
- eof
 - pennfat.c, [73](#)
- EPROCESSDOESNOTEXIST
 - errno.h, [28](#)
- errno
 - errno.c, [26](#)
 - errno.h, [28](#)
- errno.c, [26](#)
- errno, [26](#)
- errno.h, [27](#)
- E2BIG, [27](#)
- EEOF, [27](#)
- EFBIG, [27](#)
- EFTF, [27](#)
- EINVPERM, [27](#)
- ENAMETOOLONG, [28](#)
- ENOENT, [28](#)
- EPROCESSDOESNOTEXIST, [28](#)
- errno, [28](#)
- EUDWV, [28](#)
- EUDWV
 - errno.h, [28](#)
- execute
 - shell.h, [80](#)
 - shell_execute.c, [86](#)
- EXITED
 - log.h, [59](#)
- EXPECT_COMMANDS
 - parser.h, [62](#)
- EXPECT_INPUT_FILENAME
 - parser.h, [62](#)
- EXPECT_OUTPUT_FILENAME
 - parser.h, [62](#)
- f_cat
 - filefunc.c, [31](#)
 - filefunc.h, [36](#)
- f_chmod
 - filefunc.c, [31](#)
 - filefunc.h, [36](#)
- f_close
 - filefunc.c, [31](#)
 - filefunc.h, [36](#)
- f_cp
 - filefunc.c, [31](#)
 - filefunc.h, [37](#)
- f_eof
 - filefunc.c, [34](#)
- f_fc
 - filefunc.c, [34](#)

- f_findscript
 - filefunc.c, 31
 - filefunc.h, 37
- f_ls
 - filefunc.c, 32
 - filefunc.h, 37
- f_lseek
 - filefunc.c, 32
 - filefunc.h, 37
- f_mv
 - filefunc.c, 32
 - filefunc.h, 37
- f_open
 - filefunc.c, 32
 - filefunc.h, 38
- f_read
 - filefunc.c, 32
 - filefunc.h, 38
- f_rm
 - filefunc.c, 33
 - filefunc.h, 38
- F_SEEK_CUR
 - filefunc.c, 30
- F_SEEK_END
 - filefunc.c, 30
- F_SEEK_SET
 - filefunc.c, 30
- f_touch
 - filefunc.c, 33
 - filefunc.h, 38
- f_unlink
 - filefunc.c, 33
 - filefunc.h, 38
- f_write
 - filefunc.c, 33
 - filefunc.h, 39
- fake_initialize
 - pennfat.c, 71
- fat
 - pennfat.c, 73
- fat_context, 6
 - BLOCK_SIZE, 7
 - DIR_SIZE, 7
 - DIRS_PER_BLOCK, 7
 - empty_block, 7
 - empty_dir, 7
 - fd, 7
 - filesystem.h, 45
 - LSB, 7
 - MSB, 7
 - NUM_DATA_BLOCKS, 8
 - NUM_DATA_BYTES, 8
 - NUM_FAT, 8
 - NUM_FAT_BLOCKS, 8
 - NUM_FAT_BYTES, 8
- fat_pointer
 - fd_entry, 9
- fatread
 - filesystem.c, 41
 - filesystem.h, 46
- fatreadblock
 - filesystem.c, 41
 - filesystem.h, 46
- fatremove
 - filesystem.c, 41
 - filesystem.h, 46
- fatwrite
 - filesystem.c, 42
 - filesystem.h, 47
- fc
 - pennfat.c, 73
- fd
 - fat_context, 7
- fd_entry, 8
 - curr_data_ptr, 9
 - fat_pointer, 9
 - filefunc.h, 36
 - name, 9
- fd_ptr
 - filefunc.c, 34
- fds
 - pcb, 14
- filefunc
 - filefunc.c, 33
- filefunc.c, 29
 - f_cat, 31
 - f_chmod, 31
 - f_close, 31
 - f_cp, 31
 - f_eof, 34
 - f_fc, 34
 - f_findscript, 31
 - f_ls, 32
 - f_lseek, 32
 - f_mv, 32
 - f_open, 32
 - f_read, 32
 - f_rm, 33
 - F_SEEK_CUR, 30
 - F_SEEK_END, 30
 - F_SEEK_SET, 30
 - f_touch, 33
 - f_unlink, 33
 - f_write, 33
 - fd_ptr, 34
 - filefunc, 33
 - fileFuncConstructor, 34
 - files, 35
 - find_file, 34
 - find_next_available_fd, 34
 - MAXLINELENGTH, 30
 - NUM_FAT_ENTRIES, 35
- filefunc.h, 35
 - f_cat, 36
 - f_chmod, 36
 - f_close, 36

- [f_cp](#), 37
- [f_findscript](#), 37
- [f_ls](#), 37
- [f_lseek](#), 37
- [f_mv](#), 37
- [f_open](#), 38
- [f_read](#), 38
- [f_rm](#), 38
- [f_touch](#), 38
- [f_unlink](#), 38
- [f_write](#), 39
- [fd_entry](#), 36
- [fileFuncConstructor](#), 39
- [fileFuncConstructor](#)
 - [filefunc.c](#), 34
 - [filefunc.h](#), 39
- [files](#)
 - [filefunc.c](#), 35
- [filesystem.c](#), 40
 - [clear_data_region](#), 40
 - [clear_file](#), 40
 - [constructor](#), 41
 - [create](#), 41
 - [dir_to_struct](#), 41
 - [fatread](#), 41
 - [fatreadblock](#), 41
 - [fatremove](#), 41
 - [fatwrite](#), 42
 - [find_available_block](#), 42
 - [find_dir_entry](#), 42
 - [find_last_block](#), 42
 - [initialize](#), 42
 - [read_dir_entry](#), 42
 - [read_fat_block](#), 43
 - [root](#), 44
 - [seek_data_region](#), 43
 - [seek_dir_data_region](#), 43
 - [seek_to_write_data](#), 43
 - [update_dir_entry](#), 43
 - [write_block_fat](#), 43
 - [write_next_dir_entry](#), 44
 - [zeros](#), 44
- [filesystem.h](#), 44
 - [block](#), 45
 - [clear_file](#), 45
 - [constructor](#), 46
 - [create](#), 46
 - [dir_to_struct](#), 46
 - [fat_context](#), 45
 - [fatread](#), 46
 - [fatreadblock](#), 46
 - [fatremove](#), 46
 - [fatwrite](#), 47
 - [find_available_block](#), 47
 - [find_dir_entry](#), 47
 - [find_last_block](#), 47
 - [initialize](#), 47
 - [read_dir_entry](#), 47
 - [read_fat_block](#), 48
 - [seek_data_region](#), 48
 - [seek_dir_data_region](#), 48
 - [seek_to_write_data](#), 48
 - [update_dir_entry](#), 48
 - [write_block_fat](#), 48
 - [write_next_dir_entry](#), 49
- [find_available_block](#)
 - [filesystem.c](#), 42
 - [filesystem.h](#), 47
- [find_dir_entry](#)
 - [filesystem.c](#), 42
 - [filesystem.h](#), 47
- [find_file](#)
 - [filefunc.c](#), 34
- [find_last_block](#)
 - [filesystem.c](#), 42
 - [filesystem.h](#), 47
- [find_next_available_fd](#)
 - [filefunc.c](#), 34
- [firstBlock](#)
 - [block](#), 5
- [get_msb](#)
 - [utils.h](#), 106
- [get_pcb_from_pid](#)
 - [pcb_list.c](#), 67
 - [pcb_list.h](#), 68
- [Group](#), 9
 - [changed](#), 10
 - [id](#), 10
 - [ids](#), 10
 - [name](#), 10
 - [shell.h](#), 80
 - [size](#), 10
 - [status](#), 10
- [group](#)
 - [Node](#), 11
- [handleFinish](#)
 - [sched.c](#), 74
- [hang](#)
 - [stress.c](#), 94
 - [stress.h](#), 95
- [head](#)
 - [kernel.c](#), 51
 - [kernel.h](#), 53
 - [List](#), 11
 - [user_func.h](#), 102
- [id](#)
 - [Group](#), 10
- [idle_context](#)
 - [kernel.c](#), 51
 - [sched.h](#), 75
- [idle_process](#)
 - [built_ins.c](#), 20
 - [built_ins.h](#), 24
- [ids](#)

- Group, 10
- Init
 - shell.h, 82
 - shell_list.c, 88
- initialize
 - filesystem.c, 42
 - filesystem.h, 47
- invalid_cmd_process
 - built_ins.c, 20
 - built_ins.h, 24
- is_background
 - parsed_command, 12
- is_file_append
 - parsed_command, 13
- is_zero
 - utils.c, 104
 - utils.h, 106
- isEmpty
 - shell.h, 82
 - shell_list.c, 88
- k_process_cleanup
 - kernel_func.h, 55
- k_process_cleanup_1
 - kernel_func.c, 54
 - kernel_func.h, 56
- k_process_create
 - kernel_func.c, 54
 - kernel_func.h, 56
- k_process_kill
 - kernel_func.c, 54
 - kernel_func.h, 56
- kernel.c, 50
 - active_pcb, 51
 - head, 51
 - idle_context, 51
 - log_file, 51
 - main, 51
 - main_context, 52
 - max_pid, 52
 - scheduler_context, 52
 - terminal_control, 52
 - ticks, 52
- kernel.h, 52
 - head, 53
 - queues, 53
 - terminal_control, 53
 - ticks, 53
- kernel_func.c, 54
 - k_process_cleanup_1, 54
 - k_process_create, 54
 - k_process_kill, 54
 - wake_up_parent, 55
- kernel_func.h, 55
 - k_process_cleanup, 55
 - k_process_cleanup_1, 56
 - k_process_create, 56
 - k_process_kill, 56
 - max_pid, 56
 - scheduler_context, 57
 - wake_up_parent, 56
- kill_process
 - built_ins.c, 21
- lastJob
 - shell.c, 77
 - shell.h, 84
- List, 10
 - head, 11
 - shell.h, 80
 - tail, 11
- log.c, 57
 - log_command, 58
- log.h, 58
 - BLOCKED_LOG, 58
 - CONTINUED, 59
 - CREATE, 59
 - EXITED, 59
 - log_command, 60
 - log_file, 60
 - NICE, 59
 - ORPHAN, 59
 - SCHEDULE, 59
 - SIGNALED, 59
 - STOPPED_LOG, 59
 - UNBLOCKED, 60
 - WAITED, 60
 - ZOMBIE, 60
- log_command
 - log.c, 58
 - log.h, 60
- log_file
 - kernel.c, 51
 - log.h, 60
- ls
 - pennfat.c, 71
- ls_wrapper
 - built_ins.c, 21
 - built_ins.h, 24
- LSB
 - fat_context, 7
- main
 - kernel.c, 51
 - pennfat.c, 71
- main_context
 - kernel.c, 52
 - sched.h, 75
- MAX
 - utils.h, 105
- max_pid
 - kernel.c, 52
 - kernel_func.h, 56
- MAXLINELENGTH
 - filefunc.c, 30
 - pennfat.c, 70
- MIN
 - utils.h, 105

- mkfs
 - pennfat.c, 72
- mount
 - pennfat.c, 72
- MSB
 - fat_context, 7
- mtime
 - block, 5
- mv
 - pennfat.c, 72
- mv_wrapper
 - built_ins.c, 21
 - built_ins.h, 24
- name
 - block, 5
 - fd_entry, 9
 - Group, 10
 - pcb, 15
- next
 - Node, 11
 - pcb_list_node, 17
- NICE
 - log.h, 59
- Node, 11
 - group, 11
 - next, 11
 - prev, 12
 - shell.h, 80
- nohang
 - stress.c, 94
 - stress.h, 95
- num_children
 - pcb, 15
- num_commands
 - parsed_command, 13
- NUM_DATA_BLOCKS
 - fat_context, 8
- NUM_DATA_BYTES
 - fat_context, 8
- NUM_FAT
 - fat_context, 8
- NUM_FAT_BLOCKS
 - fat_context, 8
- NUM_FAT_BYTES
 - fat_context, 8
- NUM_FAT_ENTRIES
 - filefunc.c, 35
- num_to_wait_for
 - pcb, 15
- num_zombies
 - pcb, 15
- ORPHAN
 - log.h, 59
- orphan_child
 - built_ins.c, 21
- orphanify
 - built_ins.c, 21
- built_ins.h, 24
- p_exit
 - user_func.c, 97
 - user_func.h, 100
- p_kill
 - user_func.c, 97
 - user_func.h, 100
- p_nice
 - user_func.c, 97
 - user_func.h, 100
- p_perror
 - user_func.c, 97
 - user_func.h, 100
- p_sleep
 - user_func.c, 97
 - user_func.h, 100
- p_spawn
 - user_func.c, 97
 - user_func.h, 101
- p_waitpid
 - user_func.c, 98
 - user_func.h, 101
- p_waitpid_1
 - user_func.c, 98
- parent
 - pcb, 15
- parent_sigint_handler
 - shell.c, 76
- parent_sigstsp_handler
 - shell.c, 76
- parse_command
 - parser.h, 63
- parse_script
 - built_ins.c, 21
 - built_ins.h, 24
- parsed_command, 12
 - commands, 12
 - is_background, 12
 - is_file_append, 13
 - num_commands, 13
 - stdin_file, 13
 - stdout_file, 13
- parser.h, 61
 - EXPECT_COMMANDS, 62
 - EXPECT_INPUT_FILENAME, 62
 - EXPECT_OUTPUT_FILENAME, 62
 - parse_command, 63
 - print_parsed_command, 63
 - UNEXPECTED_AMPERSAND, 62
 - UNEXPECTED_FILE_INPUT, 62
 - UNEXPECTED_FILE_OUTPUT, 62
 - UNEXPECTED_PIPELINE, 62
- pcb, 13
 - blocked_until, 14
 - child_pids, 14
 - context, 14
 - fds, 14
 - name, 15

- num_children, 15
- num_to_wait_for, 15
- num_zombies, 15
- parent, 15
- pcb.h, 65
- pcb_list_node, 17
- pid, 15
- priority, 15
- sleeping, 15
- status, 16
- term_status, 16
- time_expired, 16
- to_wait_for, 16
- updated_flag, 16
- waiting, 16
- waiting_for, 16
- woke_up_by, 16
- zombies, 17
- pcb.h, 64
 - BLOCKED, 65
 - pcb, 65
 - RUNNING, 65
 - STOPPED, 65
 - TERM_NORMAL, 65
 - TERM_SIGNALED, 65
 - ZOMBIED, 65
- pcb_list.c, 66
 - add_pcb, 67
 - get_pcb_from_pid, 67
 - remove_pcb_from_pid, 67
 - soft_remove, 67
- pcb_list.h, 67
 - add_pcb, 68
 - get_pcb_from_pid, 68
 - pcb_list_node, 68
 - remove_pcb_from_pid, 68
 - soft_remove, 68
- pcb_list_node, 17
 - next, 17
 - pcb, 17
 - pcb_list.h, 68
- Peek
 - shell.h, 82
 - shell_list.c, 88
- pennfat.c, 69
 - cat, 70
 - cat_from_terminal, 70
 - chmod, 71
 - count_args, 71
 - cp, 71
 - eof, 73
 - fake_initialize, 71
 - fat, 73
 - fc, 73
 - ls, 71
 - main, 71
 - MAXLINELENGTH, 70
 - mkfs, 72
 - mount, 72
 - mv, 72
 - rm, 72
 - touch, 72
 - unmount, 72
- perm
 - block, 6
- pid
 - pcb, 15
- prev
 - Node, 12
- print_parsed_command
 - parser.h, 63
- printAll
 - shell.h, 82
 - shell_list.c, 88
- printJobs
 - shell.h, 82
 - shell_notif.c, 89
- priority
 - pcb, 15
- ps_process
 - built_ins.c, 21
 - built_ins.h, 25
- queues
 - kernel.h, 53
 - sched.c, 74
 - user_func.h, 102
- read_dir_entry
 - filesystem.c, 42
 - filesystem.h, 47
- read_fat_block
 - filesystem.c, 43
 - filesystem.h, 48
- recur
 - stress.c, 94
 - stress.h, 95
- Remove
 - shell.h, 83
- remove_pcb_from_pid
 - pcb_list.c, 67
 - pcb_list.h, 68
- RemoveById
 - shell.h, 83
 - shell_list.c, 88
- report
 - shell.h, 83
 - shell_notif.c, 89
- RETURN_ASCII_CODE
 - shell.h, 79
- rm
 - pennfat.c, 72
- rm_wrapper
 - built_ins.c, 22
 - built_ins.h, 25
- root
 - filesystem.c, 44

- RUNNING
 - pcb.h, 65
- runOnForeground
 - shell.c, 76
- S_SIGCONT
 - signals.h, 92
- S_SIGSTOP
 - signals.h, 92
- S_SIGTERM
 - signals.h, 92
- sched.c, 73
 - handleFinish, 74
 - queues, 74
 - schedule, 74
 - unblock, 74
- sched.h, 74
 - active_pcb, 75
 - idle_context, 75
 - main_context, 75
 - schedule, 75
- SCHEDULE
 - log.h, 59
- schedule
 - sched.c, 74
 - sched.h, 75
- scheduler_context
 - kernel.c, 52
 - kernel_func.h, 57
 - signals.h, 93
- seek_data_region
 - filesystem.c, 43
 - filesystem.h, 48
- seek_dir_data_region
 - filesystem.c, 43
 - filesystem.h, 48
- seek_to_write_data
 - filesystem.c, 43
 - filesystem.h, 48
- shell
 - shell.c, 77
 - shell.h, 83
- shell.c, 76
 - begin, 77
 - lastJob, 77
 - parent_sigint_handler, 76
 - parent_sigstp_handler, 76
 - runOnForeground, 76
 - shell, 77
 - stoppedJobs, 77
- shell.h, 78
 - Add, 80
 - childId, 84
 - childStatus, 84
 - clear, 80
 - COMMAND_LENGTH, 79
 - execute, 80
 - Group, 80
 - Init, 82
 - isEmpty, 82
 - lastJob, 84
 - List, 80
 - Node, 80
 - Peek, 82
 - printAll, 82
 - printJobs, 82
 - Remove, 83
 - RemoveById, 83
 - report, 83
 - RETURN_ASCII_CODE, 79
 - shell, 83
 - SHELL_BACKGROUND_TO_FOREGROUND, 79
 - SHELL_FINISHED, 79
 - SHELL_RESTARTING, 79
 - SHELL_RUNNING, 79
 - SHELL_STOPPED, 79
 - updateStatuses, 84
- SHELL_BACKGROUND_TO_FOREGROUND
 - shell.h, 79
- shell_execute.c, 85
 - create_child, 86
 - execute, 86
- SHELL_FINISHED
 - shell.h, 79
- shell_list.c, 87
 - Add, 87
 - clear, 87
 - Init, 88
 - isEmpty, 88
 - Peek, 88
 - printAll, 88
 - RemoveById, 88
- shell_notif.c, 89
 - printJobs, 89
 - report, 89
 - updateStatuses, 90
- SHELL_RESTARTING
 - shell.h, 79
- SHELL_RUNNING
 - shell.h, 79
- SHELL_STOPPED
 - shell.h, 79
- sigalrm_handler
 - signals.c, 91
 - signals.h, 92
- sigint_handler
 - signals.c, 91
 - signals.h, 92
- SIGNALED
 - log.h, 59
- signals.c, 90
 - sigalrm_handler, 91
 - sigint_handler, 91
 - sigstop_handler, 91
- signals.h, 91
 - active_pcb, 93
 - S_SIGCONT, 92

- S_SIGSTOP, 92
- S_SIGTERM, 92
- scheduler_context, 93
- sigarm_handler, 92
- sigint_handler, 92
- sigstop_handler, 92
- sigstop_handler
 - signals.c, 91
 - signals.h, 92
- size
 - block, 6
 - Group, 10
- sleep_process
 - built_ins.c, 22
 - built_ins.h, 25
- sleeping
 - pcb, 15
- soft_remove
 - pcb_list.c, 67
 - pcb_list.h, 68
- status
 - Group, 10
 - pcb, 16
- stdin_file
 - parsed_command, 13
- stdout_file
 - parsed_command, 13
- STOPPED
 - pcb.h, 65
- STOPPED_LOG
 - log.h, 59
- stoppedJobs
 - shell.c, 77
- stress.c, 93
 - hang, 94
 - nohang, 94
 - recur, 94
- stress.h, 94
 - hang, 95
 - nohang, 95
 - recur, 95
- tail
 - List, 11
- TERM_NORMAL
 - pcb.h, 65
- TERM_SIGNALED
 - pcb.h, 65
- term_status
 - pcb, 16
- terminal_control
 - kernel.c, 52
 - kernel.h, 53
- ticks
 - kernel.c, 52
 - kernel.h, 53
- time_expired
 - pcb, 16
- to_wait_for
 - pcb, 16
- touch
 - pennfat.c, 72
- touch_wrapper
 - built_ins.c, 22
 - built_ins.h, 25
- type
 - block, 6
- unblock
 - sched.c, 74
- UNBLOCKED
 - log.h, 60
- UNEXPECTED_AMPERSAND
 - parser.h, 62
- UNEXPECTED_FILE_INPUT
 - parser.h, 62
- UNEXPECTED_FILE_OUTPUT
 - parser.h, 62
- UNEXPECTED_PIPELINE
 - parser.h, 62
- unmount
 - pennfat.c, 72
- update_dir_entry
 - filesystem.c, 43
 - filesystem.h, 48
- updated_flag
 - pcb, 16
- updateStatuses
 - shell.h, 84
 - shell_notif.c, 90
- user_func.c, 96
 - check_pid_is_child, 96
 - p_exit, 97
 - p_kill, 97
 - p_nice, 97
 - p_perror, 97
 - p_sleep, 97
 - p_spawn, 97
 - p_waitpid, 98
 - p_waitpid_1, 98
 - W_WIFEXITED, 98
 - W_WIFSIGNALED, 98
 - W_WIFSTOPPED, 98
- user_func.h, 99
 - active_pcb, 102
 - check_pid_is_child, 100
 - head, 102
 - p_exit, 100
 - p_kill, 100
 - p_nice, 100
 - p_perror, 100
 - p_sleep, 100
 - p_spawn, 101
 - p_waitpid, 101
 - queues, 102
 - W_WIFEXITED, 101
 - W_WIFSIGNALED, 101
 - W_WIFSTOPPED, 102

- utils.c, [103](#)
 - char_to_16, [103](#)
 - char_to_32, [104](#)
 - char_to_8, [104](#)
 - is_zero, [104](#)
- utils.h, [104](#)
 - char_to_16, [105](#)
 - char_to_32, [105](#)
 - char_to_8, [105](#)
 - get_msb, [106](#)
 - is_zero, [106](#)
 - MAX, [105](#)
 - MIN, [105](#)
- W_WIFEXITED
 - user_func.c, [98](#)
 - user_func.h, [101](#)
- W_WIFSIGNALED
 - user_func.c, [98](#)
 - user_func.h, [101](#)
- W_WIFSTOPPED
 - user_func.c, [98](#)
 - user_func.h, [102](#)
- WAITED
 - log.h, [60](#)
- waiting
 - pcb, [16](#)
- waiting_for
 - pcb, [16](#)
- wake_up_parent
 - kernel_func.c, [55](#)
 - kernel_func.h, [56](#)
- woke_up_by
 - pcb, [16](#)
- write_block_fat
 - filesystem.c, [43](#)
 - filesystem.h, [48](#)
- write_next_dir_entry
 - filesystem.c, [44](#)
 - filesystem.h, [49](#)
- zeros
 - filesystem.c, [44](#)
- ZOMBIE
 - log.h, [60](#)
- zombie_child
 - built_ins.c, [22](#)
- ZOMBIED
 - pcb.h, [65](#)
- zombies
 - pcb, [17](#)
- zombify
 - built_ins.c, [22](#)
 - built_ins.h, [25](#)