

# Companion PDF

PennOS: *Page 2*

Standalone PennFAT: *Page 117*

Team Members:

Yu Cao ([yucao7@seas.upenn.edu](mailto:yucao7@seas.upenn.edu))

Chun-Ngai Chan ([jamesccn@seas.upenn.edu](mailto:jamesccn@seas.upenn.edu))

Wanjin Li ([wanjinli@seas.upenn.edu](mailto:wanjinli@seas.upenn.edu))

Larry Zhang ([chizhg@seas.upenn.edu](mailto:chizhg@seas.upenn.edu))

PennOS

Generated by Doxygen 1.9.1



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 Entry Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	6
3.1.2.1 next	6
3.1.2.2 prev	6
3.1.2.3 process	6
3.2 LinkedList Struct Reference	6
3.2.1 Detailed Description	7
3.2.2 Member Data Documentation	7
3.2.2.1 head	7
3.2.2.2 tail	7
3.3 parsed_command Struct Reference	7
3.3.1 Detailed Description	7
3.3.2 Member Data Documentation	8
3.3.2.1 commands	8
3.3.2.2 is_background	8
3.3.2.3 is_file_append	8
3.3.2.4 num_commands	8
3.3.2.5 stdin_file	8
3.3.2.6 stdout_file	8
3.4 Process Struct Reference	9
3.4.1 Detailed Description	10
3.4.2 Member Data Documentation	10
3.4.2.1 argv	10
3.4.2.2 awake_time	10
3.4.2.3 bg_time	10
3.4.2.4 childrens	10
3.4.2.5 cmd	10
3.4.2.6 context	10
3.4.2.7 fg_cont	11
3.4.2.8 group_id	11
3.4.2.9 input_descriptor	11
3.4.2.10 is_orphan	11
3.4.2.11 num_children	11
3.4.2.12 output_descriptor	11
3.4.2.13 parent	11

3.4.2.14 parent_process_id . . . . .	11
3.4.2.15 priority . . . . .	12
3.4.2.16 recorded . . . . .	12
3.4.2.17 signal_terminated . . . . .	12
3.4.2.18 status . . . . .	12
3.4.2.19 stop_time . . . . .	12
3.4.2.20 thread_process_id . . . . .	12
3.4.2.21 to_wait . . . . .	12
<b>4 File Documentation</b>	<b>13</b>
4.1 kernel.c File Reference . . . . .	13
4.1.1 Macro Definition Documentation . . . . .	14
4.1.1.1 S_SIGCONT . . . . .	14
4.1.1.2 S_SIGSTOP . . . . .	14
4.1.1.3 S_SIGTERM . . . . .	14
4.1.2 Function Documentation . . . . .	14
4.1.2.1 k_get_next_pid() . . . . .	15
4.1.2.2 k_get_running_status() . . . . .	15
4.1.2.3 k_get_sigcont_signal() . . . . .	15
4.1.2.4 k_get_sigcont_str() . . . . .	15
4.1.2.5 k_get_sigstop_signal() . . . . .	16
4.1.2.6 k_get_sigstop_str() . . . . .	16
4.1.2.7 k_get_sigterm_signal() . . . . .	16
4.1.2.8 k_get_sigterm_str() . . . . .	16
4.1.2.9 k_get_stop_signal_status() . . . . .	17
4.1.2.10 k_get_terminal_normal_status() . . . . .	17
4.1.2.11 k_get_terminal_signal_status() . . . . .	17
4.1.2.12 k_get_to_exit() . . . . .	17
4.1.2.13 k_initiate_to_exit() . . . . .	17
4.1.2.14 k_kill_all() . . . . .	17
4.1.2.15 k_lookup_process() . . . . .	18
4.1.2.16 k_process_cleanup() . . . . .	18
4.1.2.17 k_process_create() . . . . .	18
4.1.2.18 k_process_create_with_priority() . . . . .	19
4.1.2.19 k_process_kill() . . . . .	19
4.1.2.20 k_reap_zombie() . . . . .	19
4.1.2.21 k_set_idle() . . . . .	20
4.1.2.22 k_set_to_exit() . . . . .	20
4.2 kernel.h File Reference . . . . .	20
4.2.1 Macro Definition Documentation . . . . .	22
4.2.1.1 RUNNING . . . . .	22
4.2.1.2 S_SIGCONT . . . . .	22

4.2.1.3 S_SIGCONT_STR . . . . .	22
4.2.1.4 S_SIGSTOP . . . . .	22
4.2.1.5 S_SIGSTOP_STR . . . . .	22
4.2.1.6 S_SIGTERM . . . . .	22
4.2.1.7 S_SIGTERM_STR . . . . .	22
4.2.1.8 STOP_SIGNAL . . . . .	23
4.2.1.9 TERMINATE_NORMAL . . . . .	23
4.2.1.10 TERMINATE_SIGNAL . . . . .	23
4.2.2 Function Documentation . . . . .	23
4.2.2.1 k_get_next_pid() . . . . .	23
4.2.2.2 k_get_running_status() . . . . .	23
4.2.2.3 k_get_sigcont_signal() . . . . .	24
4.2.2.4 k_get_sigcont_str() . . . . .	24
4.2.2.5 k_get_sigstop_signal() . . . . .	24
4.2.2.6 k_get_sigstop_str() . . . . .	24
4.2.2.7 k_get_sigterm_signal() . . . . .	25
4.2.2.8 k_get_sigterm_str() . . . . .	25
4.2.2.9 k_get_stop_signal_status() . . . . .	25
4.2.2.10 k_get_terminal_normal_status() . . . . .	25
4.2.2.11 k_get_terminal_signal_status() . . . . .	26
4.2.2.12 k_get_to_exit() . . . . .	26
4.2.2.13 k_initiate_to_exit() . . . . .	26
4.2.2.14 k_kill_all() . . . . .	26
4.2.2.15 k_lookup_process() . . . . .	26
4.2.2.16 k_process_cleanup() . . . . .	28
4.2.2.17 k_process_create() . . . . .	28
4.2.2.18 k_process_create_with_priority() . . . . .	28
4.2.2.19 k_process_kill() . . . . .	29
4.2.2.20 k_reap_zombie() . . . . .	29
4.2.2.21 k_set_idle() . . . . .	29
4.2.2.22 k_set_to_exit() . . . . .	30
4.3 linkedlist-job.c File Reference . . . . .	30
4.3.1 Function Documentation . . . . .	30
4.3.1.1 delete_node() . . . . .	31
4.3.1.2 free_list() . . . . .	31
4.3.1.3 free_process() . . . . .	31
4.3.1.4 insert_end() . . . . .	31
4.3.1.5 poll() . . . . .	32
4.3.1.6 retrieve_latest() . . . . .	32
4.3.1.7 search_list() . . . . .	33
4.3.1.8 set_orphan() . . . . .	33
4.4 linkedlist-job.h File Reference . . . . .	33

4.4.1 Macro Definition Documentation	35
4.4.1.1 ACTIVE_STAT	35
4.4.1.2 DONE_STAT	35
4.4.1.3 ENTRY	35
4.4.1.4 LIST	35
4.4.1.5 PAUSED_STAT	35
4.4.1.6 PROCESS	35
4.4.1.7 STOP_STAT	35
4.4.2 Function Documentation	35
4.4.2.1 delete_node()	35
4.4.2.2 free_list()	36
4.4.2.3 free_process()	36
4.4.2.4 insert_end()	36
4.4.2.5 poll()	37
4.4.2.6 print_list()	37
4.4.2.7 retrieve_latest()	37
4.4.2.8 search_list()	38
4.4.2.9 set_orphan()	38
4.5 logger.c File Reference	38
4.5.1 Function Documentation	39
4.5.1.1 log_event()	39
4.5.1.2 log_nice_event()	39
4.5.1.3 open_log_file()	40
4.6 logger.h File Reference	40
4.6.1 Function Documentation	41
4.6.1.1 log_event()	41
4.6.1.2 log_nice_event()	41
4.7 parser.h File Reference	41
4.7.1 Macro Definition Documentation	43
4.7.1.1 EXPECT_COMMANDS	43
4.7.1.2 EXPECT_INPUT_FILENAME	43
4.7.1.3 EXPECT_OUTPUT_FILENAME	43
4.7.1.4 UNEXPECTED_AMPERSAND	43
4.7.1.5 UNEXPECTED_FILE_INPUT	43
4.7.1.6 UNEXPECTED_FILE_OUTPUT	43
4.7.1.7 UNEXPECTED_PIPELINE	43
4.7.2 Function Documentation	44
4.7.2.1 parse_command()	44
4.7.2.2 print_parsed_command()	44
4.8 pennos.c File Reference	44
4.8.1 Function Documentation	45
4.8.1.1 handler()	45

4.8.1.2 main()	45
4.9 pennos.h File Reference	46
4.9.1 Function Documentation	47
4.9.1.1 handler()	47
4.9.1.2 main()	47
4.10 scheduler.c File Reference	47
4.10.1 Macro Definition Documentation	48
4.10.1.1 NOT_WAITING	48
4.10.1.2 THREAD_COUNT	49
4.10.2 Function Documentation	49
4.10.2.1 check_actual()	49
4.10.2.2 freeStacks()	49
4.10.2.3 s_check_active()	49
4.10.2.4 s_get_current()	49
4.10.2.5 s_get_priority()	50
4.10.2.6 s_get_scheduler_context()	50
4.10.2.7 s_get_time()	50
4.10.2.8 s_get_zombie_context()	50
4.10.2.9 s_initiate()	51
4.10.2.10 s_initiate_priorities()	51
4.10.2.11 s_initiate_shell_context()	51
4.10.2.12 s_insert()	51
4.10.2.13 s_makeContext()	51
4.10.2.14 s_print_all_jobs()	52
4.10.2.15 s_set()	52
4.10.2.16 s_set_current()	52
4.10.2.17 s_set_idle()	52
4.10.2.18 s_set_status()	53
4.10.2.19 s_setup()	54
4.10.2.20 s_swap()	54
4.10.2.21 setStack()	54
4.11 scheduler.h File Reference	54
4.11.1 Macro Definition Documentation	56
4.11.1.1 NOT_WAITING	56
4.11.2 Function Documentation	56
4.11.2.1 freeStacks()	56
4.11.2.2 s_check_active()	56
4.11.2.3 s_get_current()	56
4.11.2.4 s_get_priority()	57
4.11.2.5 s_get_scheduler_context()	57
4.11.2.6 s_get_time()	57
4.11.2.7 s_get_zombie_context()	57



4.11.2.8 s_initiate()	58
4.11.2.9 s_initiate_priorities()	58
4.11.2.10 s_initiate_shell_context()	58
4.11.2.11 s_insert()	58
4.11.2.12 s_makeContext()	58
4.11.2.13 s_print_all_jobs()	59
4.11.2.14 s_set()	59
4.11.2.15 s_set_current()	59
4.11.2.16 s_set_idle()	59
4.11.2.17 s_set_status()	60
4.11.2.18 s_setup()	61
4.11.2.19 s_swap()	61
4.11.2.20 setStack()	61
4.12 shell.c File Reference	61
4.12.1 Function Documentation	62
4.12.1.1 get_fg_pid()	62
4.12.1.2 set_fg_pid()	62
4.12.1.3 shell()	63
4.12.1.4 sigint_handler()	63
4.12.1.5 sigstp_handler()	63
4.13 shell.h File Reference	63
4.13.1 Function Documentation	64
4.13.1.1 get_fg_pid()	64
4.13.1.2 set_fg_pid()	64
4.13.1.3 shell()	65
4.14 stress.c File Reference	65
4.14.1 Function Documentation	65
4.14.1.1 hang()	65
4.14.1.2 nohang()	66
4.14.1.3 recur()	66
4.15 stress.h File Reference	66
4.15.1 Function Documentation	66
4.15.1.1 hang()	66
4.15.1.2 nohang()	67
4.15.1.3 recur()	67
4.16 user.c File Reference	67
4.16.1 Function Documentation	68
4.16.1.1 getSignal()	68
4.16.1.2 p_add_background_job()	69
4.16.1.3 p_add_stop_job()	69
4.16.1.4 p_busy_wait()	69
4.16.1.5 p_exit()	69

4.16.1.6 p_exit_process()	69
4.16.1.7 p_get_current()	70
4.16.1.8 p_get_sigcont_signal()	70
4.16.1.9 p_get_sigcont_str()	70
4.16.1.10 p_get_sigstop_signal()	70
4.16.1.11 p_get_sigstop_str()	71
4.16.1.12 p_get_sigterm_signal()	71
4.16.1.13 p_get_sigterm_str()	71
4.16.1.14 p_initiate()	71
4.16.1.15 p_initiate_priorities()	71
4.16.1.16 p_initiate_shell()	71
4.16.1.17 p_initiate_to_exit()	72
4.16.1.18 p_kill()	72
4.16.1.19 p_lookup_process()	72
4.16.1.20 p_nice()	73
4.16.1.21 p_orphan_child()	73
4.16.1.22 p_orphanify()	73
4.16.1.23 p_print_all_jobs()	73
4.16.1.24 p_remove_background_job()	73
4.16.1.25 p_remove_stop_job()	74
4.16.1.26 p_run_cat_fs()	74
4.16.1.27 p_run_chmod_fs()	74
4.16.1.28 p_run_cp_fs()	74
4.16.1.29 p_run_echo()	75
4.16.1.30 p_run_f_ls_list()	75
4.16.1.31 p_run_f_ls_null()	75
4.16.1.32 p_run_kill()	75
4.16.1.33 p_run_mv_fs()	76
4.16.1.34 p_run_rm_fs()	76
4.16.1.35 p_run_touch_fs()	76
4.16.1.36 p_search_and_remove()	76
4.16.1.37 p_search_bg()	76
4.16.1.38 p_search_most_recent()	77
4.16.1.39 p_search_most_recent_stop()	77
4.16.1.40 p_setup()	77
4.16.1.41 p_sleep()	77
4.16.1.42 p_spawn()	78
4.16.1.43 p_spawn_with_input()	78
4.16.1.44 p_spawn_with_priority()	79
4.16.1.45 p_waitpid()	79
4.16.1.46 p_zombie_child()	80
4.16.1.47 p_zombify()	80

4.16.1.48 W_WIFEXITED()	80
4.16.1.49 W_WIFSIGNALED()	80
4.16.1.50 W_WIFSTOPPED()	81
4.17 user.h File Reference	81
4.17.1 Function Documentation	82
4.17.1.1 p_add_background_job()	83
4.17.1.2 p_add_stop_job()	83
4.17.1.3 p_busy_wait()	83
4.17.1.4 p_exit()	83
4.17.1.5 p_exit_process()	83
4.17.1.6 p_get_current()	84
4.17.1.7 p_get_sigcont_signal()	84
4.17.1.8 p_get_sigcont_str()	84
4.17.1.9 p_get_sigstop_signal()	84
4.17.1.10 p_get_sigstop_str()	85
4.17.1.11 p_get_sigterm_signal()	85
4.17.1.12 p_get_sigterm_str()	85
4.17.1.13 p_initiate()	85
4.17.1.14 p_initiate_priorities()	85
4.17.1.15 p_initiate_shell()	85
4.17.1.16 p_initiate_to_exit()	86
4.17.1.17 p_kill()	86
4.17.1.18 p_lookup_process()	86
4.17.1.19 p_nice()	87
4.17.1.20 p_orphan_child()	87
4.17.1.21 p_orphanify()	87
4.17.1.22 p_print_all_jobs()	87
4.17.1.23 p_remove_background_job()	87
4.17.1.24 p_remove_stop_job()	88
4.17.1.25 p_run_cat_fs()	88
4.17.1.26 p_run_chmod_fs()	88
4.17.1.27 p_run_cp_fs()	88
4.17.1.28 p_run_echo()	89
4.17.1.29 p_run_f_ls_list()	89
4.17.1.30 p_run_f_ls_null()	89
4.17.1.31 p_run_kill()	89
4.17.1.32 p_run_mv_fs()	90
4.17.1.33 p_run_rm_fs()	90
4.17.1.34 p_run_touch_fs()	90
4.17.1.35 p_search_and_remove()	90
4.17.1.36 p_search_bg()	90
4.17.1.37 p_search_most_recent()	91

4.17.1.38 <code>p_search_most_recent_stop()</code> . . . . .	91
4.17.1.39 <code>p_setup()</code> . . . . .	91
4.17.1.40 <code>p_sleep()</code> . . . . .	91
4.17.1.41 <code>p_spawn()</code> . . . . .	92
4.17.1.42 <code>p_spawn_with_input()</code> . . . . .	92
4.17.1.43 <code>p_spawn_with_priority()</code> . . . . .	93
4.17.1.44 <code>p_waitpid()</code> . . . . .	93
4.17.1.45 <code>p_zombie_child()</code> . . . . .	94
4.17.1.46 <code>p_zombify()</code> . . . . .	94
4.17.1.47 <code>W_WIFEXITED()</code> . . . . .	94
4.17.1.48 <code>W_WIFSIGNALED()</code> . . . . .	94
4.17.1.49 <code>W_WIFSTOPPED()</code> . . . . .	95

<b>Index</b>	<b>97</b>
--------------	-----------



# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">Entry</a>	5
<a href="#">LinkedList</a>	6
<a href="#">parsed_command</a>	7
<a href="#">Process</a>	9



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">kernel.c</a>	13
<a href="#">kernel.h</a>	20
<a href="#">linkedList-job.c</a>	30
<a href="#">linkedList-job.h</a>	33
<a href="#">logger.c</a>	38
<a href="#">logger.h</a>	40
<a href="#">parser.h</a>	41
<a href="#">pennos.c</a>	44
<a href="#">pennos.h</a>	46
<a href="#">scheduler.c</a>	47
<a href="#">scheduler.h</a>	54
<a href="#">shell.c</a>	61
<a href="#">shell.h</a>	63
<a href="#">stress.c</a>	65
<a href="#">stress.h</a>	66
<a href="#">user.c</a>	67
<a href="#">user.h</a>	81





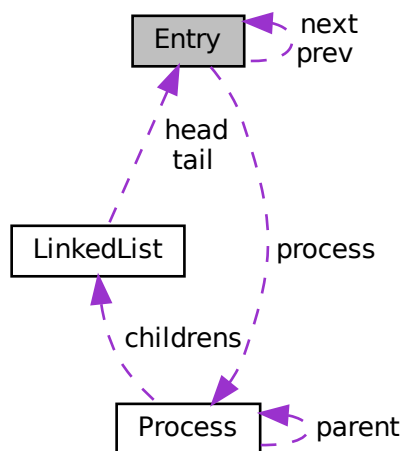
## Chapter 3

# Class Documentation

### 3.1 Entry Struct Reference

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

Collaboration diagram for Entry:



#### Public Attributes

- struct [Process](#) \* [process](#)
- struct [Entry](#) \* [prev](#)
- struct [Entry](#) \* [next](#)

#### 3.1.1 Detailed Description

The Node struct used for our LinkedList.

### 3.1.2 Member Data Documentation

#### 3.1.2.1 next

```
struct Entry* Entry::next
```

the pointer to the next node

#### 3.1.2.2 prev

```
struct Entry* Entry::prev
```

the pointer to the previous node

#### 3.1.2.3 process

```
struct Process* Entry::process
```

the [Process](#) stored inside this node

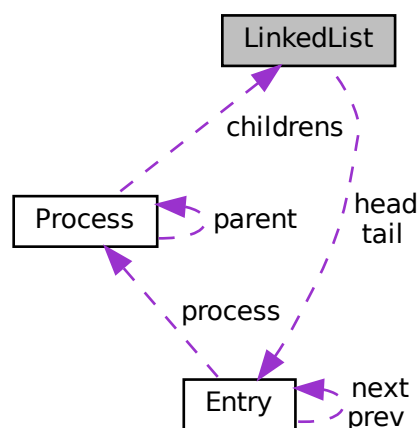
The documentation for this struct was generated from the following file:

- [linkedList-job.h](#)

## 3.2 LinkedList Struct Reference

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

Collaboration diagram for LinkedList:



## Public Attributes

- struct [Entry](#) \* [head](#)
- struct [Entry](#) \* [tail](#)

### 3.2.1 Detailed Description

The LinkedList struct.

### 3.2.2 Member Data Documentation

#### 3.2.2.1 head

```
struct Entry* LinkedList::head
```

the pointer to the head node of this linkedlist

#### 3.2.2.2 tail

```
struct Entry* LinkedList::tail
```

the pointer to the tail node of this linkedlist

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

- [linkedlist-job.h](#)

## 3.3 parsed\_command Struct Reference

```
#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](#) []

### 3.3.1 Detailed Description

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

## 3.3.2 Member Data Documentation

### 3.3.2.1 commands

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

### 3.3.2.2 is\_background

```
bool parsed_command::is_background
```

### 3.3.2.3 is\_file\_append

```
bool parsed_command::is_file_append
```

### 3.3.2.4 num\_commands

```
size_t parsed_command::num_commands
```

### 3.3.2.5 stdin\_file

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

### 3.3.2.6 stdout\_file

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

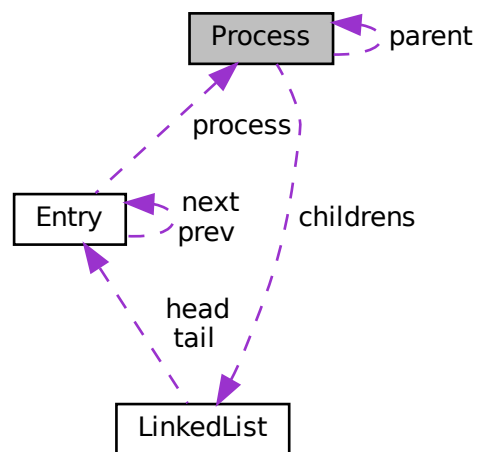
The documentation for this struct was generated from the following file:

- [parser.h](#)

## 3.4 Process Struct Reference

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

Collaboration diagram for Process:



### Public Attributes

- `ucontext_t *` `context`
- `int` `thread_process_id`
- `int` `parent_process_id`
- `struct Process *` `parent`
- `struct LinkedList *` `childrens`
- `int` `priority`
- `int` `input_descriptor`
- `int` `output_descriptor`
- `int` `num_children`
- `int` `status`
- `bool` `signal_terminated`
- `int` `group_id`
- `int` `awake_time`
- `char *` `cmd`
- `pid_t` `to_wait`
- `int` `recorded`
- `bool` `is_orphan`
- `int` `bg_time`
- `int` `stop_time`
- `bool` `fg_cont`
- `char **` `argv`

### 3.4.1 Detailed Description

The PCB struct used for our OS.

### 3.4.2 Member Data Documentation

#### 3.4.2.1 argv

```
char** Process::argv
```

The modified arguments we are taking into specific functions

#### 3.4.2.2 awake\_time

```
int Process::awake_time
```

the time this process should awake (only for sleep)

#### 3.4.2.3 bg\_time

```
int Process::bg_time
```

record when was this [Process](#) stored in background

#### 3.4.2.4 childrens

```
struct LinkedList* Process::childrens
```

the Linkedlist of the PCB's children

#### 3.4.2.5 cmd

```
char* Process::cmd
```

the command for this [Process](#) (for logging)

#### 3.4.2.6 context

```
ucontext_t* Process::context
```

the ucontext of the PCB

#### 3.4.2.7 fg\_cont

```
bool Process::fg_cont
```

record if it should be continued at foreground

#### 3.4.2.8 group\_id

```
int Process::group_id
```

[Process](#) group id (unused)

#### 3.4.2.9 input\_descriptor

```
int Process::input_descriptor
```

the input descriptor for this [Process](#)

#### 3.4.2.10 is\_orphan

```
bool Process::is_orphan
```

record whether it's an orphan

#### 3.4.2.11 num\_children

```
int Process::num_children
```

the number of children for this [Process](#)

#### 3.4.2.12 output\_descriptor

```
int Process::output_descriptor
```

the output descriptor for this [Process](#)

#### 3.4.2.13 parent

```
struct Process* Process::parent
```

PCB pointer to its parent

#### 3.4.2.14 parent\_process\_id

```
int Process::parent_process_id
```

the pid of this PCB's parent



**3.4.2.15 priority**

```
int Process::priority
```

the priority (nice value) of this PCB

**3.4.2.16 recorded**

```
int Process::recorded
```

the recorded status (modified when it's parent called waitpid with hang)

**3.4.2.17 signal\_terminated**

```
bool Process::signal_terminated
```

record if this process is terminated by signal

**3.4.2.18 status**

```
int Process::status
```

the status for this PCB

**3.4.2.19 stop\_time**

```
int Process::stop_time
```

record when was this [Process](#) stopped

**3.4.2.20 thread\_process\_id**

```
int Process::thread_process_id
```

the pid of the PCB

**3.4.2.21 to\_wait**

```
pid_t Process::to_wait
```

the pid this [Process](#) is waiting on

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

- [linkedlist-job.h](#)

## Chapter 4

# File Documentation

### 4.1 kernel.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <ucontext.h>
#include <signal.h>
#include "kernel.h"
#include "scheduler.h"
#include "linkedlist-job.h"
#include "user.h"
#include "logger.h"
#include "shell.h"
```

Include dependency graph for kernel.c:



### Macros

- #define [S\\_SIGSTOP](#) 0
- #define [S\\_SIGCONT](#) 1
- #define [S\\_SIGTERM](#) 2

## Functions

- struct [Process](#) \* [k\\_process\\_create](#) (struct [Process](#) \*parent)
- struct [Process](#) \* [k\\_process\\_create\\_with\\_priority](#) (struct [Process](#) \*parent, int priority)
- int [k\\_get\\_next\\_pid](#) ()
- int [k\\_process\\_kill](#) (struct [Process](#) \*process, int signal)
- struct [Process](#) \* [k\\_lookup\\_process](#) (int thread\_id)
- bool [k\\_set\\_idle](#) ()
- void [k\\_kill\\_all](#) (int signal)
- void [k\\_process\\_cleanup](#) (struct [Process](#) \*process)
- void [k\\_reap\\_zombie](#) (int pid)
- int [k\\_get\\_terminal\\_normal\\_status](#) ()
- int [k\\_get\\_stop\\_signal\\_status](#) ()
- int [k\\_get\\_terminal\\_signal\\_status](#) ()
- int [k\\_get\\_running\\_status](#) ()
- int [k\\_get\\_sigstop\\_signal](#) ()
- int [k\\_get\\_sigcont\\_signal](#) ()
- int [k\\_get\\_sigterm\\_signal](#) ()
- void [k\\_initiate\\_to\\_exit](#) ()
- void [k\\_set\\_to\\_exit](#) (struct [Process](#) \*process)
- struct [Process](#) \* [k\\_get\\_to\\_exit](#) ()
- char \* [k\\_get\\_sigstop\\_str](#) ()
- char \* [k\\_get\\_sigcont\\_str](#) ()
- char \* [k\\_get\\_sigterm\\_str](#) ()

## 4.1.1 Macro Definition Documentation

### 4.1.1.1 S\_SIGCONT

```
#define S_SIGCONT 1
```

### 4.1.1.2 S\_SIGSTOP

```
#define S_SIGSTOP 0
```

### 4.1.1.3 S\_SIGTERM

```
#define S_SIGTERM 2
```

## 4.1.2 Function Documentation

#### 4.1.2.1 k\_get\_next\_pid()

```
int k_get_next_pid ( )
```

Get the current available pid and increment it by 1.

##### Returns

The next pid we can assign to a new process.

#### 4.1.2.2 k\_get\_running\_status()

```
int k_get_running_status ( )
```

For abstraction sake, get the RUNNING wstatus from outside.

##### Returns

The RUNNING defined.

#### 4.1.2.3 k\_get\_sigcont\_signal()

```
int k_get_sigcont_signal ( )
```

For abstraction sake, get the S\_SIGCONT signal from outside.

##### Returns

The S\_SIGCONT signal defined.

#### 4.1.2.4 k\_get\_sigcont\_str()

```
char* k_get_sigcont_str ( )
```

The helper function we used to get S\_SIGCONT\_STR for shell use.

##### Returns

The S\_SIGCONT\_STR we defined.

#### 4.1.2.5 k\_get\_sigstop\_signal()

```
int k_get_sigstop_signal ( )
```

For abstraction sake, get the S\_SIGSTOP signal from outside.

##### Returns

The S\_SIGSTOP signal defined.

#### 4.1.2.6 k\_get\_sigstop\_str()

```
char* k_get_sigstop_str ( )
```

The helper function we used to get S\_SIGSTOP\_STR for shell use.

##### Returns

The S\_SIGSTOP\_STR we defined.

#### 4.1.2.7 k\_get\_sigterm\_signal()

```
int k_get_sigterm_signal ( )
```

For abstraction sake, get the S\_SIGTERM signal from outside.

##### Returns

The S\_SIGTERM signal defined.

#### 4.1.2.8 k\_get\_sigterm\_str()

```
char* k_get_sigterm_str ( )
```

The helper function we used to get S\_SIGTERM\_STR for shell use.

##### Returns

The S\_SIGTERM\_STR we defined.

#### 4.1.2.9 k\_get\_stop\_signal\_status()

```
int k_get_stop_signal_status ( )
```

For abstraction sake, get the STOP\_SIGNAL wstatus from outside.

##### Returns

The STOP\_SIGNAL defined.

#### 4.1.2.10 k\_get\_terminal\_normal\_status()

```
int k_get_terminal_normal_status ( )
```

For abstraction sake, get the TERMINAL\_NORMAL wstatus from outside.

##### Returns

The TERMINAL\_NORMAL defined.

#### 4.1.2.11 k\_get\_terminal\_signal\_status()

```
int k_get_terminal_signal_status ( )
```

For abstraction sake, get the TERMINAL\_SIGNAL wstatus from outside.

##### Returns

The TERMINAL\_SIGNAL defined.

#### 4.1.2.12 k\_get\_to\_exit()

```
struct Process* k_get_to_exit ( )
```

Get the process stored in to\_exit.

##### Returns

The process stored inside to\_exit.

#### 4.1.2.13 k\_initiate\_to\_exit()

```
void k_initiate_to_exit ( )
```

Initiate the to\_exit stored for [p\\_exit\\_process\(\)](#).

#### 4.1.2.14 k\_kill\_all()

```
void k_kill_all (
    int signal )
```

Kill all processes with a given signal.

## Parameters

<i>signal</i>	The signal we are sending to all processes.
---------------	---

**4.1.2.15 k\_lookup\_process()**

```
struct Process* k_lookup_process (
    int thread_id )
```

Look up a process from our job queue.

## Parameters

<i>thread_id</i>	The pid we are looking for.
------------------	-----------------------------

## Returns

The process we found with this pid, NULL on failed.

**4.1.2.16 k\_process\_cleanup()**

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

Clean up a process from our job queue and free it.

## Parameters

<i>process</i>	The process to clean up.
----------------	--------------------------

**4.1.2.17 k\_process\_create()**

```
struct Process* k_process_create (
    struct Process * parent )
```

Create a process with a given process as its parent.

## Parameters

<i>parent</i>	The pointer to the parent process.
---------------	------------------------------------

**Returns**

The pointer to the [Process](#) we created.

**4.1.2.18 k\_process\_create\_with\_priority()**

```
struct Process* k_process_create_with_priority (
    struct Process * parent,
    int priority )
```

Create a process with a certain priority (nice value) and a process as its parent.

**Parameters**

<i>parent</i>	The process we need to set as this process's parent.
<i>priority</i>	The nice value for this process.

**Returns**

The process created.

**4.1.2.19 k\_process\_kill()**

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

Kill a [Process](#) with the given signal.

**Parameters**

<i>process</i>	The process we want to kill with the signal.
<i>signal</i>	The signal we are sending to the process.

**Returns**

The status of whether the killing is successful.

**4.1.2.20 k\_reap\_zombie()**

```
void k_reap_zombie (
    int pid )
```

Reap a zombie process from our job queue with its pid.



## Parameters

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

**4.1.2.21 k\_set\_idle()**

```
bool k_set_idle ( )
```

Set the idle param to false, i.e. we are leaving the idle context.

**4.1.2.22 k\_set\_to\_exit()**

```
void k_set_to_exit (
    struct Process * process )
```

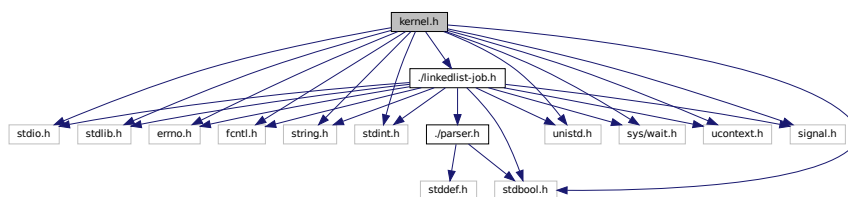
Set to\_exit as a process for [p\\_exit\\_process\(\)](#).

## Parameters

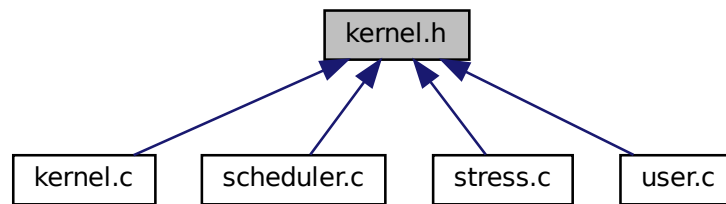
<i>process</i>	The process to set as to_exit.
----------------	--------------------------------

**4.2 kernel.h File Reference**

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <ucontext.h>
#include <signal.h>
#include "../linkedlist-job.h"
Include dependency graph for kernel.h:
```



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



## Macros

- `#define TERMINATE_NORMAL 0`
- `#define STOP_SIGNAL 1`
- `#define TERMINATE_SIGNAL 2`
- `#define RUNNING 3`
- `#define S_SIGSTOP_STR "0"`
- `#define S_SIGCONT_STR "1"`
- `#define S_SIGTERM_STR "2"`
- `#define S_SIGSTOP 0`
- `#define S_SIGCONT 1`
- `#define S_SIGTERM 2`

## Functions

- `struct Process * k_process_create (struct Process *parent)`
- `int k_process_kill (struct Process *process, int signal)`
- `void k_kill_all (int signal)`
- `bool k_set_idle ()`
- `int k_get_next_pid ()`
- `struct Process * k_lookup_process (int thread_id)`
- `struct Process * k_process_create_with_priority (struct Process *parent, int priority)`
- `void k_process_cleanup (struct Process *process)`
- `void k_reap_zombie (int pid)`
- `int k_get_terminal_normal_status ()`
- `int k_get_stop_signal_status ()`
- `int k_get_terminal_signal_status ()`
- `int k_get_running_status ()`
- `int k_get_sigstop_signal ()`
- `int k_get_sigcont_signal ()`
- `int k_get_sigterm_signal ()`
- `void k_initiate_to_exit ()`
- `void k_set_to_exit (struct Process *process)`
- `struct Process * k_get_to_exit ()`
- `char * k_get_sigstop_str ()`
- `char * k_get_sigcont_str ()`
- `char * k_get_sigterm_str ()`

## 4.2.1 Macro Definition Documentation

### 4.2.1.1 RUNNING

```
#define RUNNING 3
```

The wstatus standing for process still running (for nohang).

### 4.2.1.2 S\_SIGCONT

```
#define S_SIGCONT 1
```

The signal sent to continue a previously stopped process.

### 4.2.1.3 S\_SIGCONT\_STR

```
#define S_SIGCONT_STR "1"
```

The string of the signal sent to continue a previously stopped process. (used for shell)

### 4.2.1.4 S\_SIGSTOP

```
#define S_SIGSTOP 0
```

The signal sent to stop a process.

### 4.2.1.5 S\_SIGSTOP\_STR

```
#define S_SIGSTOP_STR "0"
```

The string of the signal sent to stop a process. (used for shell)

### 4.2.1.6 S\_SIGTERM

```
#define S_SIGTERM 2
```

The signal sent to terminate a process.

### 4.2.1.7 S\_SIGTERM\_STR

```
#define S_SIGTERM_STR "2"
```

The string of the signal sent to terminate a process. (used for shell)

#### 4.2.1.8 STOP\_SIGNAL

```
#define STOP_SIGNAL 1
```

The wstatus standing for process stopped by a signal.

#### 4.2.1.9 TERMINATE\_NORMAL

```
#define TERMINATE_NORMAL 0
```

The wstatus standing for process terminated normally.

#### 4.2.1.10 TERMINATE\_SIGNAL

```
#define TERMINATE_SIGNAL 2
```

The wstatus standing for process terminated by a signal.

### 4.2.2 Function Documentation

#### 4.2.2.1 k\_get\_next\_pid()

```
int k_get_next_pid ( )
```

Get the current available pid and increment it by 1.

##### Returns

The next pid we can assign to a new process.

#### 4.2.2.2 k\_get\_running\_status()

```
int k_get_running_status ( )
```

For abstraction sake, get the RUNNING wstatus from outside.

##### Returns

The RUNNING defined.

#### 4.2.2.3 k\_get\_sigcont\_signal()

```
int k_get_sigcont_signal ( )
```

For abstraction sake, get the S\_SIGCONT signal from outside.

##### Returns

The S\_SIGCONT signal defined.

#### 4.2.2.4 k\_get\_sigcont\_str()

```
char* k_get_sigcont_str ( )
```

The helper function we used to get S\_SIGCONT\_STR for shell use.

##### Returns

The S\_SIGCONT\_STR we defined.

#### 4.2.2.5 k\_get\_sigstop\_signal()

```
int k_get_sigstop_signal ( )
```

For abstraction sake, get the S\_SIGSTOP signal from outside.

##### Returns

The S\_SIGSTOP signal defined.

#### 4.2.2.6 k\_get\_sigstop\_str()

```
char* k_get_sigstop_str ( )
```

The helper function we used to get S\_SIGSTOP\_STR for shell use.

##### Returns

The S\_SIGSTOP\_STR we defined.

#### 4.2.2.7 k\_get\_sigterm\_signal()

```
int k_get_sigterm_signal ( )
```

For abstraction sake, get the S\_SIGTERM signal from outside.

##### Returns

The S\_SIGTERM signal defined.

#### 4.2.2.8 k\_get\_sigterm\_str()

```
char* k_get_sigterm_str ( )
```

The helper function we used to get S\_SIGTERM\_STR for shell use.

##### Returns

The S\_SIGTERM\_STR we defined.

#### 4.2.2.9 k\_get\_stop\_signal\_status()

```
int k_get_stop_signal_status ( )
```

For abstraction sake, get the STOP\_SIGNAL wstatus from outside.

##### Returns

The STOP\_SIGNAL defined.

#### 4.2.2.10 k\_get\_terminal\_normal\_status()

```
int k_get_terminal_normal_status ( )
```

For abstraction sake, get the TERMINAL\_NORMAL wstatus from outside.

##### Returns

The TERMINAL\_NORMAL defined.

#### 4.2.2.11 k\_get\_terminal\_signal\_status()

```
int k_get_terminal_signal_status ( )
```

For abstraction sake, get the `TERMINAL_SIGNAL` wstatus from outside.

##### Returns

The `TERMINAL_SIGNAL` defined.

#### 4.2.2.12 k\_get\_to\_exit()

```
struct Process* k_get_to_exit ( )
```

Get the process stored in `to_exit`.

##### Returns

The process stored inside `to_exit`.

#### 4.2.2.13 k\_initiate\_to\_exit()

```
void k_initiate_to_exit ( )
```

Initiate the `to_exit` stored for `p_exit_process()`.

#### 4.2.2.14 k\_kill\_all()

```
void k_kill_all (
    int signal )
```

Kill all processes with a given signal.

##### Parameters

<i>signal</i>	The signal we are sending to all processes.
---------------	---

#### 4.2.2.15 k\_lookup\_process()

```
struct Process* k_lookup_process (
    int thread_id )
```

Look up a process from our job queue.



## Parameters

<i>thread</i> ↵ _id	The pid we are looking for.
------------------------	-----------------------------

## Returns

The process we found with this pid, NULL on failed.

**4.2.2.16 k\_process\_cleanup()**

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

Clean up a process from our job queue and free it.

## Parameters

<i>process</i>	The process to clean up.
----------------	--------------------------

**4.2.2.17 k\_process\_create()**

```
struct Process* k_process_create (
    struct Process * parent )
```

Create a process with a given process as its parent.

## Parameters

<i>parent</i>	The pointer to the parent process.
---------------	------------------------------------

## Returns

The pointer to the *Process* we created.

**4.2.2.18 k\_process\_create\_with\_priority()**

```
struct Process* k_process_create_with_priority (
    struct Process * parent,
    int priority )
```

Create a process with a certain priority (nice value) and a process as its parent.

## Parameters

<i>parent</i>	The process we need to set as this process's parent.
<i>priority</i>	The nice value for this process.

## Returns

The process created.

**4.2.2.19 k\_process\_kill()**

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

Kill a [Process](#) with the given signal.

## Parameters

<i>process</i>	The process we want to kill with the signal.
<i>signal</i>	The signal we are sending to the process.

## Returns

The status of whether the killing is successful.

**4.2.2.20 k\_reap\_zombie()**

```
void k_reap_zombie (
    int pid )
```

Reap a zombie process from our job queue with its pid.

## Parameters

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

**4.2.2.21 k\_set\_idle()**

```
bool k_set_idle ( )
```

Set the idle param to false, i.e. we are leaving the idle context.

#### 4.2.2.22 k\_set\_to\_exit()

```
void k_set_to_exit (
    struct Process * process )
```

Set to\_exit as a process for [p\\_exit\\_process\(\)](#).

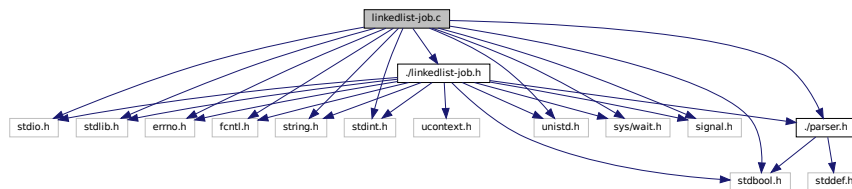
##### Parameters

<i>process</i>	The process to set as to_exit.
----------------	--------------------------------

## 4.3 linkedlist-job.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <signal.h>
#include "../linkedlist-job.h"
#include "../parser.h"
```

Include dependency graph for linkedlist-job.c:



## Functions

- void [insert\\_end](#) (struct [LinkedList](#) \*list, struct [Process](#) \*process)
- struct [Process](#) \* [delete\\_node](#) (struct [LinkedList](#) \*list, int id)
- void [free\\_process](#) (struct [Process](#) \*process)
- struct [Entry](#) \* [search\\_list](#) (struct [LinkedList](#) \*list, int id)
- struct [Process](#) \* [retrieve\\_latest](#) (struct [LinkedList](#) \*list)
- struct [Process](#) \* [poll](#) (struct [LinkedList](#) \*list)
- void [free\\_list](#) (struct [LinkedList](#) \*list)
- void [set\\_orphan](#) (struct [Entry](#) \*e, bool orphan)

### 4.3.1 Function Documentation

#### 4.3.1.1 delete\_node()

```
struct Process* delete_node (
    struct LinkedList * list,
    int id )
```

Delete a node containing the process we want from a [LinkedList](#).

##### Parameters

<i>list</i>	The pointer to the list we want to delete the node from.
<i>id</i>	The pid of the <a href="#">Process</a> we are deleting.

##### Returns

The node entry containing the process we are looking for.

#### 4.3.1.2 free\_list()

```
void free_list (
    struct LinkedList * list )
```

Completely free all entries inside a list and the Linkedlist itself.

##### Parameters

<i>list</i>	The pointer to the list we want to free.
-------------	--

#### 4.3.1.3 free\_process()

```
void free_process (
    struct Process * process )
```

Free a process completely from our memory.

##### Parameters

<i>process</i>	The pointer to process we want to insert.
----------------	---

#### 4.3.1.4 insert\_end()

```
void insert_end (
```

```
struct LinkedList * list,  
struct Process * process )
```

Insert a process into the end of the linkedlist.

#### Parameters

<i>list</i>	The pointer to the linkedlist to insert into.
<i>process</i>	The pointer to process we want to insert.

#### 4.3.1.5 poll()

```
struct Process* poll (  
    struct LinkedList * list )
```

Retrieve the process contained in the first node from a [LinkedList](#).

#### Parameters

<i>list</i>	The pointer to the list we want to retrieve the node from.
-------------	--

#### Returns

The process contained in the first node from this [LinkedList](#).

#### 4.3.1.6 retrieve\_latest()

```
struct Process* retrieve_latest (  
    struct LinkedList * list )
```

Retrieve the process contained in the last node from a [LinkedList](#).

#### Parameters

<i>list</i>	The pointer to the list we want to retrieve the node from.
-------------	--

#### Returns

The process contained in the last node from this [LinkedList](#).

#### 4.3.1.7 search\_list()

```
struct Entry* search_list (
    struct LinkedList * list,
    int id )
```

Search a linkedlist for the node containing the process we want.

##### Parameters

<i>list</i>	The pointer to the list we want to search in.
<i>id</i>	The pid of the <a href="#">Process</a> we are searching for.

##### Returns

The node entry containing the process we are looking for.

#### 4.3.1.8 set\_orphan()

```
void set_orphan (
    struct Entry * e,
    bool orphan )
```

Set the process containing in a node entry as some orphan status.

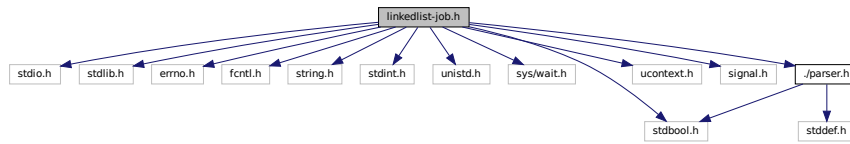
##### Parameters

<i>e</i>	The node containing the process we are setting to some orphan status.
<i>orphan</i>	The orphan status we are setting.

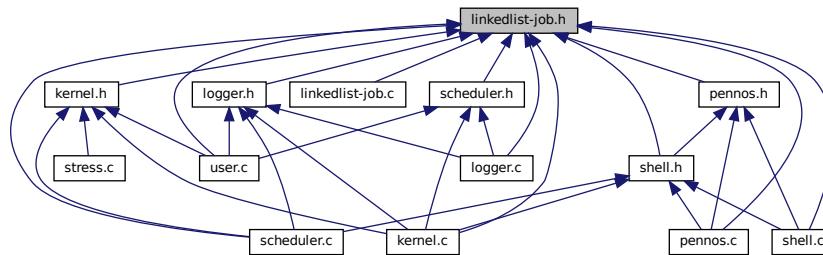
## 4.4 linkedlist-job.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <ucontext.h>
#include <signal.h>
#include "../parser.h"
```

Include dependency graph for `linkedList-job.h`:



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



## Classes

- struct [Process](#)
- struct [Entry](#)
- struct [LinkedList](#)

## Macros

- `#define` [ACTIVE\\_STAT](#) 0
- `#define` [PAUSED\\_STAT](#) 1
- `#define` [DONE\\_STAT](#) 2
- `#define` [STOP\\_STAT](#) 3
- `#define` [PROCESS](#)
- `#define` [ENTRY](#)
- `#define` [LIST](#)

## Functions

- void [insert\\_end](#) (struct [LinkedList](#) \*list, struct [Process](#) \*process)
- void [free\\_process](#) (struct [Process](#) \*process)
- struct [Entry](#) \* [search\\_list](#) (struct [LinkedList](#) \*list, int id)
- struct [Process](#) \* [delete\\_node](#) (struct [LinkedList](#) \*list, int id)
- void [print\\_list](#) (struct [LinkedList](#) \*list)
- struct [Process](#) \* [retrieve\\_latest](#) (struct [LinkedList](#) \*list)
- struct [Process](#) \* [poll](#) (struct [LinkedList](#) \*list)
- void [free\\_list](#) (struct [LinkedList](#) \*list)
- void [set\\_orphan](#) (struct [Entry](#) \*e, bool orphan)

## 4.4.1 Macro Definition Documentation

### 4.4.1.1 ACTIVE\_STAT

```
#define ACTIVE_STAT 0
```

the process is actively running and can be scheduled.

### 4.4.1.2 DONE\_STAT

```
#define DONE_STAT 2
```

the process is already terminated and didn't get reaped as a zombie.

### 4.4.1.3 ENTRY

```
#define ENTRY
```

### 4.4.1.4 LIST

```
#define LIST
```

### 4.4.1.5 PAUSED\_STAT

```
#define PAUSED_STAT 1
```

the process is paused for waiting on children and cannot be scheduled.

### 4.4.1.6 PROCESS

```
#define PROCESS
```

### 4.4.1.7 STOP\_STAT

```
#define STOP_STAT 3
```

the process is stopped by signal and can be continued by SIGCONT.

## 4.4.2 Function Documentation

### 4.4.2.1 delete\_node()

```
struct Process* delete_node (  
    struct LinkedList * list,  
    int id )
```

Delete a node containing the process we want from a [LinkedList](#).



**Parameters**

<i>list</i>	The pointer to the list we want to delete the node from.
<i>id</i>	The pid of the <a href="#">Process</a> we are deleting.

**Returns**

The node entry containing the process we are looking for.

**4.4.2.2 free\_list()**

```
void free_list (
    struct LinkedList * list )
```

Completely free all entries inside a list and the LinkedList itself.

**Parameters**

<i>list</i>	The pointer to the list we want to free.
-------------	--

**4.4.2.3 free\_process()**

```
void free_process (
    struct Process * process )
```

Free a process completely from our memory.

**Parameters**

<i>process</i>	The pointer to process we want to insert.
----------------	---

**4.4.2.4 insert\_end()**

```
void insert_end (
    struct LinkedList * list,
    struct Process * process )
```

Insert a process into the end of the linkedlist.

**Parameters**

<i>list</i>	The pointer to the linkedlist to insert into.
<i>process</i>	The pointer to process we want to insert.

#### 4.4.2.5 poll()

```
struct Process* poll (
    struct LinkedList * list )
```

Retrieve the process contained in the first node from a [LinkedList](#).

##### Parameters

<i>list</i>	The pointer to the list we want to retrieve the node from.
-------------	--

##### Returns

The process contained in the first node from this [LinkedList](#).

#### 4.4.2.6 print\_list()

```
void print_list (
    struct LinkedList * list )
```

Print out a [LinkedList](#).

##### Parameters

<i>list</i>	The pointer to the list we want to print.
-------------	---

#### 4.4.2.7 retrieve\_latest()

```
struct Process* retrieve_latest (
    struct LinkedList * list )
```

Retrieve the process contained in the last node from a [LinkedList](#).

##### Parameters

<i>list</i>	The pointer to the list we want to retrieve the node from.
-------------	--

##### Returns

The process contained in the last node from this [LinkedList](#).

#### 4.4.2.8 search\_list()

```
struct Entry* search_list (
    struct LinkedList * list,
    int id )
```

Search a linkedlist for the node containing the process we want.

##### Parameters

<i>list</i>	The pointer to the list we want to search in.
<i>id</i>	The pid of the <a href="#">Process</a> we are searching for.

##### Returns

The node entry containing the process we are looking for.

#### 4.4.2.9 set\_orphan()

```
void set_orphan (
    struct Entry * e,
    bool orphan )
```

Set the process containing in a node entry as some orphan status.

##### Parameters

<i>e</i>	The node containing the process we are setting to some orphan status.
<i>orphan</i>	The orphan status we are setting.

## 4.5 logger.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <ucontext.h>
#include <signal.h>
#include "logger.h"
#include "linkedlist-job.h"
```

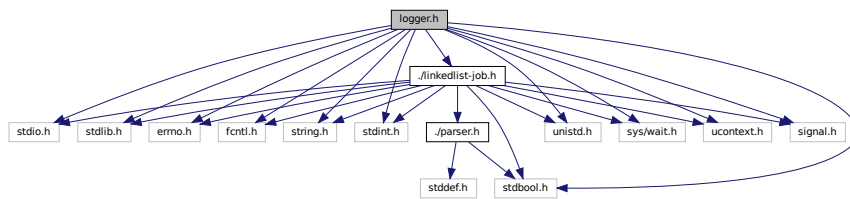


#### 4.5.1.3 open\_log\_file()

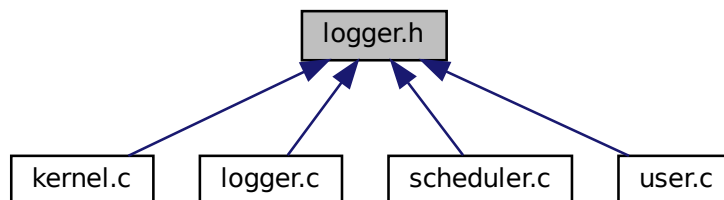
```
FILE* open_log_file ( )
```

### 4.6 logger.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <ucontext.h>
#include <signal.h>
#include "../linkedlist-job.h"
Include dependency graph for logger.h:
```



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



### Functions

- void `log_event` (const char \*event\_type, struct `Process` \*process)
- void `log_nice_event` (int old\_nice, struct `Process` \*process)

## 4.6.1 Function Documentation

### 4.6.1.1 log\_event()

```
void log_event (
    const char * event_type,
    struct Process * process )
```

Helper function used to open the log file and print logs into the log file.

#### Parameters

<i>event_type</i>	The event status to print as shown in demo.
<i>process</i>	The process we need to print log for.

### 4.6.1.2 log\_nice\_event()

```
void log_nice_event (
    int old_nice,
    struct Process * process )
```

Special helper function used to log nice-related processes.

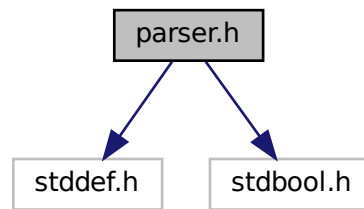
#### Parameters

<i>old_nice</i>	The old nice value.
<i>process</i>	The process we need to print log for.

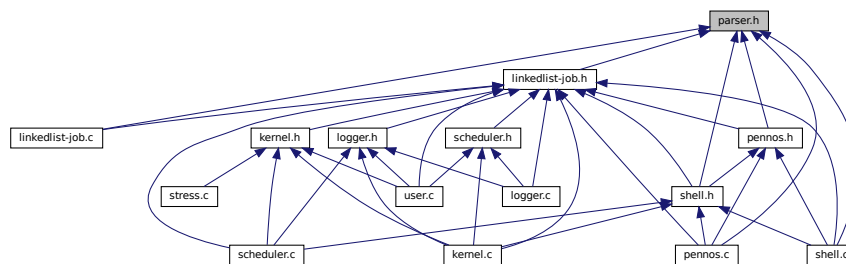
## 4.7 parser.h File Reference

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

Include dependency graph for parser.h:



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



## Classes

- struct [parsed\\_command](#)

## Macros

- #define [UNEXPECTED\\_FILE\\_INPUT](#) 1
- #define [UNEXPECTED\\_FILE\\_OUTPUT](#) 2
- #define [UNEXPECTED\\_PIPELINE](#) 3
- #define [UNEXPECTED\\_AMPERSAND](#) 4
- #define [EXPECT\\_INPUT\\_FILENAME](#) 5
- #define [EXPECT\\_OUTPUT\\_FILENAME](#) 6
- #define [EXPECT\\_COMMANDS](#) 7

## Functions

- int [parse\\_command](#) (const char \*cmd\_line, struct [parsed\\_command](#) \*\*result)
- void [print\\_parsed\\_command](#) (const struct [parsed\\_command](#) \*cmd)

## 4.7.1 Macro Definition Documentation

### 4.7.1.1 EXPECT\_COMMANDS

```
#define EXPECT_COMMANDS 7
```

### 4.7.1.2 EXPECT\_INPUT\_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

### 4.7.1.3 EXPECT\_OUTPUT\_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

### 4.7.1.4 UNEXPECTED\_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

### 4.7.1.5 UNEXPECTED\_FILE\_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

### 4.7.1.6 UNEXPECTED\_FILE\_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

### 4.7.1.7 UNEXPECTED\_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```



## 4.7.2 Function Documentation

### 4.7.2.1 `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.

### 4.7.2.2 `print_parsed_command()`

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

## 4.8 pennos.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <signal.h>
#include "../pennos.h"
#include "../user.h"
#include "../parser.h"
#include "../linkedlist-job.h"
#include "../shell.h"
#include "../Standalone_PennFat/pennfat.h"
```



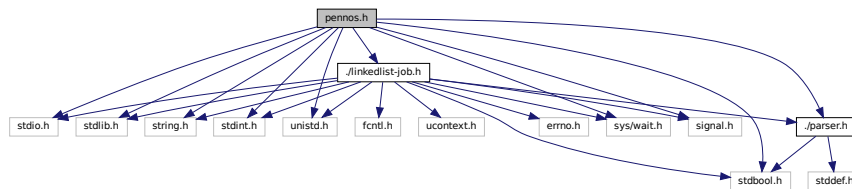
## Returns

Anything on exit.

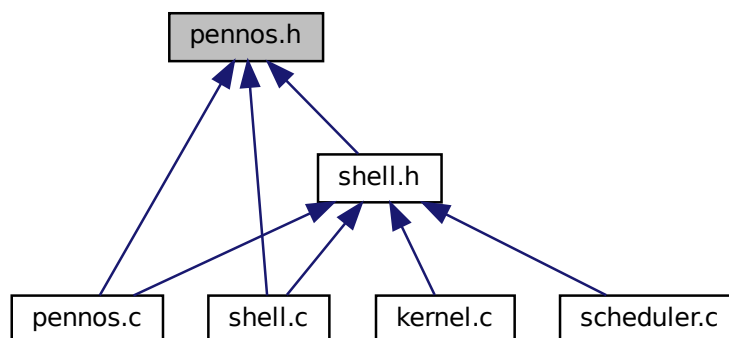
## 4.9 pennos.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <signal.h>
#include "../parser.h"
#include "../linkedlist-job.h"
```

Include dependency graph for pennos.h:



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



## Functions

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

## 4.9.1 Function Documentation

### 4.9.1.1 handler()

```
void handler (
    int signal )
```

Signal Handler for SIGINT and SIGTSTP.

#### Parameters

<i>signal</i>	The signal received.
---------------	----------------------

### 4.9.1.2 main()

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

The main function of our pennos, used to initiate everything and spawn the shell.

#### Parameters

<i>argc</i>	The number of arguments passed in from the terminal.
<i>argv</i>	The arguments from terminal.

#### Returns

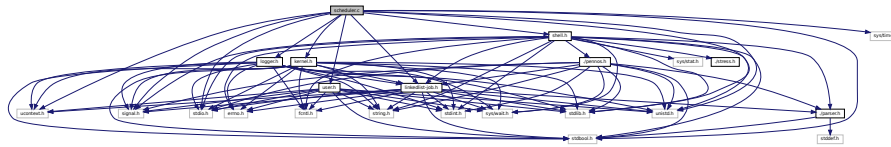
Anything on exit.

## 4.10 scheduler.c File Reference

```
#include <signal.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <ucontext.h>
#include <unistd.h>
#include "linkedlist-job.h"
#include "shell.h"
#include "logger.h"
#include "kernel.h"
```

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

Include dependency graph for scheduler.c:



## Macros

- `#define` [THREAD\\_COUNT](#) 4
- `#define` [NOT\\_WAITING](#) -2

## Functions

- void [s\\_set](#) ()
- void [s\\_swap](#) ()
- bool [s\\_check\\_active](#) (struct [LinkedList](#) \*queue)
- bool [check\\_actual](#) (struct [LinkedList](#) \*queue)
- void [setStack](#) (stack\_t \*stack)
- void [s\\_makeContext](#) (ucontext\_t \*ucp, void(\*func)(), int thread)
- void [freeStacks](#) ()
- void [s\\_setup](#) ()
- void [s\\_initiate\\_priorities](#) ()
- void [s\\_initiate](#) ()
- void [s\\_insert](#) (int priority, struct [Process](#) \*pcb)
- void [s\\_initiate\\_shell\\_context](#) (int argc, char \*argv[], struct [Process](#) \*process)
- void [s\\_set\\_current](#) (struct [Process](#) \*process)
- struct [Process](#) \* [s\\_get\\_current](#) ()
- int [s\\_get\\_time](#) ()
- struct [LinkedList](#) \* [s\\_get\\_priority](#) (int priority)
- ucontext\_t \* [s\\_get\\_zombie\\_context](#) ()
- ucontext\_t \* [s\\_get\\_scheduler\\_context](#) ()
- void [s\\_set\\_status](#) (int status)
- void [s\\_print\\_all\\_jobs](#) ()
- void [s\\_set\\_idle](#) ()

### 4.10.1 Macro Definition Documentation

#### 4.10.1.1 NOT\_WAITING

```
#define NOT_WAITING -2
```

### 4.10.1.2 THREAD\_COUNT

```
#define THREAD_COUNT 4
```

## 4.10.2 Function Documentation

### 4.10.2.1 check\_actual()

```
bool check_actual (
    struct LinkedList * queue )
```

### 4.10.2.2 freeStacks()

```
void freeStacks (
    void )
```

The helper function we used to free the current stack.

### 4.10.2.3 s\_check\_active()

```
bool s_check_active (
    struct LinkedList * queue )
```

The function we used to check if there is active job in the priority queue while checking if any of sleep jobs should be awoken.

#### Parameters

<i>queue</i>	The pointer to the priority queue we are looking at.
--------------	--

#### Returns

True if the current priority queue has an active job or we wake up some process in any of the priority queues.

### 4.10.2.4 s\_get\_current()

```
struct Process* s_get_current ( )
```

The helper function used to get the current PCB. (for Abstraction)

#### Returns

The pointer to the currentPCB.

#### 4.10.2.5 `s_get_priority()`

```
struct LinkedList* s_get_priority (
    int priority )
```

The helper function used to get a certain priority job queue.

##### Parameters

<i>priority</i>	The priority (nice value) we are trying to get.
-----------------	---

##### Returns

The pointer to the priority job [LinkedList](#) we want.

#### 4.10.2.6 `s_get_scheduler_context()`

```
ucontext_t* s_get_scheduler_context ( )
```

The helper function we used to get the Scheduler Context for abstraction.

##### Returns

The scheduler context.

#### 4.10.2.7 `s_get_time()`

```
int s_get_time ( )
```

The helper function we used to get the current time(ticks) for abstraction.

##### Returns

The current time.

#### 4.10.2.8 `s_get_zombie_context()`

```
ucontext_t* s_get_zombie_context ( )
```

The helper function we used to get the Zombie Context (context after a process finished) for abstraction.

##### Returns

The zombie context.

#### 4.10.2.9 s\_initiate()

```
void s_initiate ( )
```

The helper function we used to officially swap to the shell context.

#### 4.10.2.10 s\_initiate\_priorities()

```
void s_initiate_priorities ( )
```

The helper function we used to initialize our three priority job queues.

#### 4.10.2.11 s\_initiate\_shell\_context()

```
void s_initiate_shell_context (
    int argc,
    char * argv[],
    struct Process * process )
```

The helper function to initiate the shell context and stored inside its PCB.

##### Parameters

<i>argc</i>	The number of arguments passed in.
<i>argv</i>	The arguments passed in.
<i>process</i>	The process PCB for the shell.

#### 4.10.2.12 s\_insert()

```
void s_insert (
    int priority,
    struct Process * pcb )
```

The function we used to insert a job into a certain priority queue.

##### Parameters

<i>priority</i>	The priority (nice value) we are trying to insert.
<i>pcb</i>	The pointer to the PCB of the process we are trying to insert.

#### 4.10.2.13 s\_makeContext()

```
void s_makeContext (
    ucontext_t * ucp,
```



```
void(*)() func,
int thread )
```

The helper function we used to set the relevant properties of a context.

#### Parameters

<i>ucp</i>	The pointer to the context we are setting.
<i>func</i>	The function run on the context ucp.
<i>thread</i>	indicating whether this is shell make context.

#### 4.10.2.14 s\_print\_all\_jobs()

```
void s_print_all_jobs ( )
```

The helper function we used to print all the jobs in the order of nice value. (used for 'ps' and 'jobs')

#### 4.10.2.15 s\_set()

```
void s_set ( )
```

The helper function we used to set the current context as scheduler.

#### 4.10.2.16 s\_set\_current()

```
void s_set_current (
    struct Process * process )
```

The helper function used to set the current PCB. (for Abstraction)

#### Parameters

<i>process</i>	The process to set as currentPCB.
----------------	-----------------------------------

#### 4.10.2.17 s\_set\_idle()

```
void s_set_idle ( )
```

The helper function we used to set the current status of if we are in the idleContext.

#### 4.10.2.18 `s_set_status()`

```
void s_set_status (
    int status )
```

The helper function used to set the status of the current running [Process](#).

**Parameters**

<i>status</i>	The status we need to set for the current PCB.
---------------	--

**4.10.2.19 s\_setup()**

```
void s_setup ( )
```

The helper function we used to set up signal handler for shell and zombieContext (run after a process finished) and idleContext (for sleep).

**4.10.2.20 s\_swap()**

```
void s_swap ( )
```

The helper function we used to swap the current context with scheduler.

**4.10.2.21 setStack()**

```
void setStack (
    stack_t * stack )
```

The helper function we used to set the current stack.

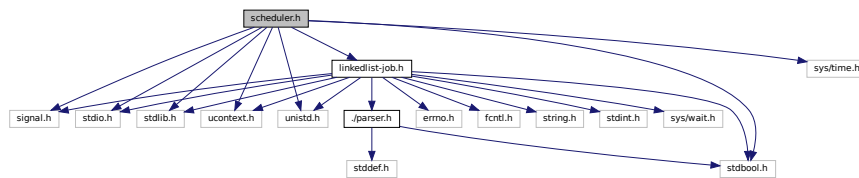
**Parameters**

<i>The</i>	pointer to the stack we are setting.
------------	--------------------------------------

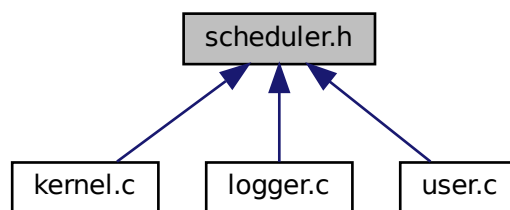
**4.11 scheduler.h File Reference**

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <ucontext.h>
#include <unistd.h>
#include "linkedlist-job.h"
#include <stdbool.h>
```

Include dependency graph for scheduler.h:



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



## Macros

- `#define NOT_WAITING -2`

## Functions

- void `setStack` (stack\_t \*stack)
- void `s_makeContext` (ucontext\_t \*ucp, void(\*func)(), int thread)
- void `freeStacks` (void)
- bool `s_check_active` (struct `LinkedList` \*queue)
- void `s_setup` ()
- void `s_initiate` ()
- void `s_initiate_priorities` ()
- void `s_insert` (int priority, struct `Process` \*pcb)
- void `s_initiate_shell_context` (int argc, char \*argv[], struct `Process` \*process)
- void `s_set_current` (struct `Process` \*process)
- struct `Process` \* `s_get_current` ()
- struct `LinkedList` \* `s_get_priority` (int priority)
- void `s_swap` ()
- void `s_set` ()
- ucontext\_t \* `s_get_zombie_context` ()
- void `s_set_status` (int status)
- void `s_print_all_jobs` ()
- ucontext\_t \* `s_get_scheduler_context` ()
- int `s_get_time` ()
- void `s_set_idle` ()

## 4.11.1 Macro Definition Documentation

### 4.11.1.1 NOT\_WAITING

```
#define NOT_WAITING -2
```

the indicator we used to indicate the parent is not waiting for it's child (stored inside to\_wait).

## 4.11.2 Function Documentation

### 4.11.2.1 freeStacks()

```
void freeStacks (
    void )
```

The helper function we used to free the current stack.

### 4.11.2.2 s\_check\_active()

```
bool s_check_active (
    struct LinkedList * queue )
```

The function we used to check if there is active job in the priority queue while checking if any of sleep jobs should be awoken.

#### Parameters

<i>queue</i>	The pointer to the priority queue we are looking at.
--------------	--

#### Returns

True if the current priority queue has an active job or we wake up some process in any of the priority queues.

### 4.11.2.3 s\_get\_current()

```
struct Process* s_get_current ( )
```

The helper function used to get the current PCB. (for Abstraction)

#### Returns

The pointer to the currentPCB.

#### 4.11.2.4 s\_get\_priority()

```
struct LinkedList* s_get_priority (
    int priority )
```

The helper function used to get a certain priority job queue.

##### Parameters

<i>priority</i>	The priority (nice value) we are trying to get.
-----------------	---

##### Returns

The pointer to the priority job [LinkedList](#) we want.

#### 4.11.2.5 s\_get\_scheduler\_context()

```
ucontext_t* s_get_scheduler_context ( )
```

The helper function we used to get the Scheduler Context for abstraction.

##### Returns

The scheduler context.

#### 4.11.2.6 s\_get\_time()

```
int s_get_time ( )
```

The helper function we used to get the current time(ticks) for abstraction.

##### Returns

The current time.

#### 4.11.2.7 s\_get\_zombie\_context()

```
ucontext_t* s_get_zombie_context ( )
```

The helper function we used to get the Zombie Context (context after a process finished) for abstraction.

##### Returns

The zombie context.

#### 4.11.2.8 s\_initiate()

```
void s_initiate ( )
```

The helper function we used to officially swap to the shell context.

#### 4.11.2.9 s\_initiate\_priorities()

```
void s_initiate_priorities ( )
```

The helper function we used to initialize our three priority job queues.

#### 4.11.2.10 s\_initiate\_shell\_context()

```
void s_initiate_shell_context (
    int argc,
    char * argv[],
    struct Process * process )
```

The helper function to initiate the shell context and stored inside its PCB.

##### Parameters

<i>argc</i>	The number of arguments passed in.
<i>argv</i>	The arguments passed in.
<i>process</i>	The process PCB for the shell.

#### 4.11.2.11 s\_insert()

```
void s_insert (
    int priority,
    struct Process * pcb )
```

The function we used to insert a job into a certain priority queue.

##### Parameters

<i>priority</i>	The priority (nice value) we are trying to insert.
<i>pcb</i>	The pointer to the PCB of the process we are trying to insert.

#### 4.11.2.12 s\_makeContext()

```
void s_makeContext (
    ucontext_t * ucp,
```

```
void(*)() func,  
int thread )
```

The helper function we used to set the relevant properties of a context.

#### Parameters

<i>ucp</i>	The pointer to the context we are setting.
<i>func</i>	The function run on the context ucp.
<i>thread</i>	indicating whether this is shell make context.

#### 4.11.2.13 s\_print\_all\_jobs()

```
void s_print_all_jobs ( )
```

The helper function we used to print all the jobs in the order of nice value. (used for 'ps' and 'jobs')

#### 4.11.2.14 s\_set()

```
void s_set ( )
```

The helper function we used to set the current context as scheduler.

#### 4.11.2.15 s\_set\_current()

```
void s_set_current (   
    struct Process * process )
```

The helper function used to set the current PCB. (for Abstraction)

#### Parameters

<i>process</i>	The process to set as currentPCB.
----------------	-----------------------------------

#### 4.11.2.16 s\_set\_idle()

```
void s_set_idle ( )
```

The helper function we used to set the current status of if we are in the idleContext.



#### 4.11.2.17 `s_set_status()`

```
void s_set_status (
    int status )
```

The helper function used to set the status of the current running [Process](#).

## Parameters

<i>status</i>	The status we need to set for the current PCB.
---------------	--

**4.11.2.18 s\_setup()**

```
void s_setup ( )
```

The helper function we used to set up signal handler for shell and zombieContext (run after a process finished) and idleContext (for sleep).

**4.11.2.19 s\_swap()**

```
void s_swap ( )
```

The helper function we used to swap the current context with scheduler.

**4.11.2.20 setStack()**

```
void setStack (
    stack_t * stack )
```

The helper function we used to set the current stack.

## Parameters

<i>The</i>	pointer to the stack we are setting.
------------	--------------------------------------

**4.12 shell.c File Reference**

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <signal.h>
#include <sys/stat.h>
#include "./pennos.h"
#include "./parser.h"
#include "./linkedlist-job.h"
#include "./shell.h"
```



#### 4.12.1.3 shell()

```
void shell ( )
```

The function we used to create an interactive shell. With this we can spawn a process with shell.

#### 4.12.1.4 sigint\_handler()

```
void sigint_handler ( )
```

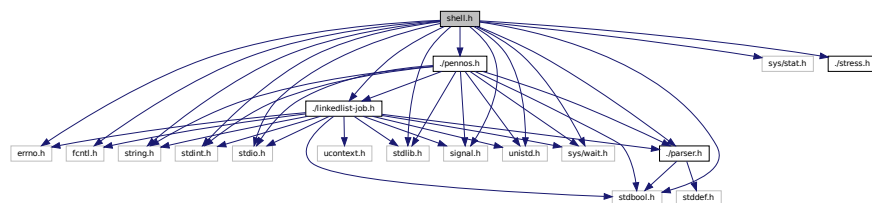
#### 4.12.1.5 sigstp\_handler()

```
void sigstp_handler ( )
```

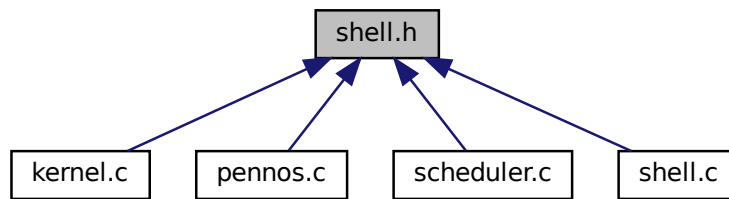
## 4.13 shell.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <signal.h>
#include "../pennos.h"
#include "../parser.h"
#include "../linkedlist-job.h"
#include <sys/stat.h>
#include "../stress.h"
```

Include dependency graph for shell.h:



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



## Functions

- int [get\\_fg\\_pid](#) ()
- void [shell](#) ()
- void [set\\_fg\\_pid](#) (int pid)

### 4.13.1 Function Documentation

#### 4.13.1.1 [get\\_fg\\_pid\(\)](#)

```
int get_fg_pid ( )
```

The helper function we used to get the current foreground job pid.

##### Returns

The current foreground job pid.

#### 4.13.1.2 [set\\_fg\\_pid\(\)](#)

```
void set_fg_pid (  
    int pid )
```

The helper function we used to set the foreground job pid.

##### Parameters

<i>pid</i>	The pid we set as the foreground job pid.
------------	---

### 4.13.1.3 shell()

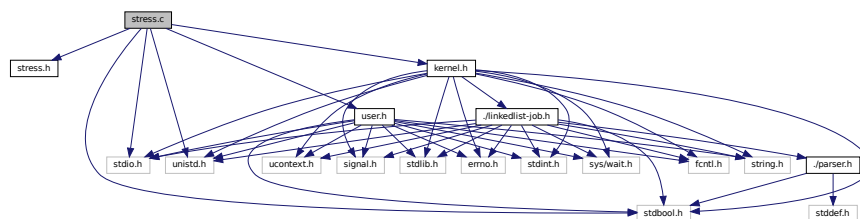
```
void shell ( )
```

The function we used to create an interactive shell. With this we can spawn a process with shell.

## 4.14 stress.c File Reference

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

Include dependency graph for stress.c:



## Functions

- void [hang](#) (void)
- void [nohang](#) (void)
- void [recur](#) (void)

### 4.14.1 Function Documentation

#### 4.14.1.1 hang()

```
void hang (
    void )
```

#### 4.14.1.2 nohang()

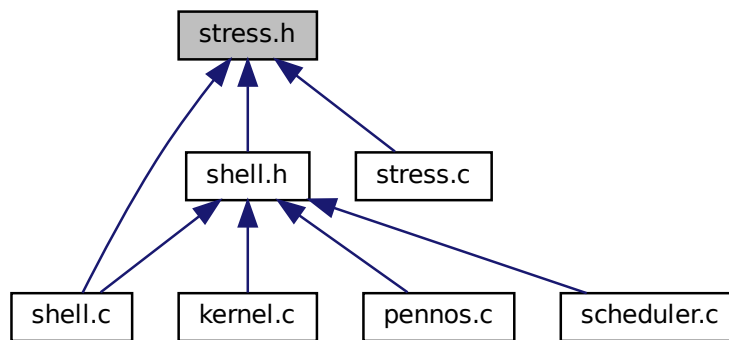
```
void nohang (
    void )
```

#### 4.14.1.3 recur()

```
void recur (
    void )
```

### 4.15 stress.h File Reference

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



#### Functions

- void [hang](#) (void)
- void [nohang](#) (void)
- void [recur](#) (void)

#### 4.15.1 Function Documentation

##### 4.15.1.1 hang()

```
void hang (
    void )
```

## 4.15.1.2 nohang()

```
void nohang (
    void )
```

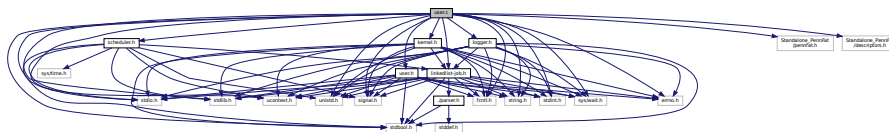
## 4.15.1.3 recur()

```
void recur (
    void )
```

## 4.16 user.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <ucontext.h>
#include <signal.h>
#include "user.h"
#include "scheduler.h"
#include "linkedlist-job.h"
#include "logger.h"
#include "kernel.h"
#include "Standalone_PennFat/pennfat.h"
#include "Standalone_PennFat/descriptors.h"
```

Include dependency graph for user.c:



## Functions

- int [getSignal](#) (char \*sig)
- pid\_t [p\\_spawn](#) (void(\*func)(), char \*argv[], int fd0, int fd1)
- pid\_t [p\\_spawn\\_with\\_priority](#) (void(\*func)(), char \*argv[], int fd0, int fd1, int priority)
- pid\_t [p\\_spawn\\_with\\_input](#) (void(\*func)(), char \*argv[], int fd0, int fd1, char \*\*actual\_input)
- pid\_t [p\\_initiate\\_shell](#) (void(\*func)(), int argc, char \*argv[])
- pid\_t [p\\_waitpid](#) (pid\_t pid, int \*wstatus, bool [nohang](#))
- int [p\\_kill](#) (pid\_t pid, int sig)
- void [p\\_exit](#) (void)



- void [p\\_exit\\_process](#) ()
- bool [W\\_WIFEXITED](#) (int \*status)
- bool [W\\_WIFSTOPPED](#) (int \*status)
- bool [W\\_WIFSIGNALED](#) (int \*status)
- int [p\\_nice](#) (pid\_t pid, int priority)
- void [p\\_sleep](#) (unsigned int ticks)
- void [p\\_busy\\_wait](#) ()
- void [p\\_zombie\\_child](#) ()
- void [p\\_orphan\\_child](#) ()
- void [p\\_print\\_all\\_jobs](#) ()
- void [p\\_add\\_background\\_job](#) (int pid)
- void [p\\_add\\_stop\\_job](#) (int pid)
- void [p\\_remove\\_background\\_job](#) (int pid)
- void [p\\_remove\\_stop\\_job](#) (int pid)
- void [p\\_search\\_and\\_remove](#) (int pid)
- int [p\\_search\\_most\\_recent](#) ()
- int [p\\_search\\_most\\_recent\\_stop](#) ()
- struct [Process](#) \* [p\\_search\\_bg](#) (int pid)
- int [p\\_get\\_sigstop\\_signal](#) ()
- int [p\\_get\\_sigcont\\_signal](#) ()
- int [p\\_get\\_sigterm\\_signal](#) ()
- void [p\\_initiate\\_to\\_exit](#) ()
- struct [Process](#) \* [p\\_get\\_current](#) ()
- void [p\\_initiate\\_priorities](#) ()
- void [p\\_setup](#) ()
- void [p\\_initiate](#) ()
- struct [Process](#) \* [p\\_lookup\\_process](#) (pid\_t pid)
- void [p\\_zombify](#) ()
- void [p\\_orphanify](#) ()
- void [p\\_run\\_kill](#) (int sig, pid\_t pid)
- void [p\\_run\\_mv\\_fs](#) (char \*source, char \*dest)
- void [p\\_run\\_chmod\\_fs](#) (char \*filename, char \*perm)
- void [p\\_run\\_cp\\_fs](#) (char \*source, char \*des, int from\_host)
- void [p\\_run\\_touch\\_fs](#) ()
- void [p\\_run\\_f\\_ls\\_list](#) ()
- void [p\\_run\\_f\\_ls\\_null](#) ()
- void [p\\_run\\_rm\\_fs](#) ()
- void [p\\_run\\_echo](#) ()
- void [p\\_run\\_cat\\_fs](#) ()
- char \* [p\\_get\\_sigstop\\_str](#) ()
- char \* [p\\_get\\_sigcont\\_str](#) ()
- char \* [p\\_get\\_sigterm\\_str](#) ()

## 4.16.1 Function Documentation

### 4.16.1.1 getSignal()

```
int getSignal (
    char * sig )
```

#### 4.16.1.2 p\_add\_background\_job()

```
void p_add_background_job (
    int pid )
```

The helper function to add a thread into the background.

##### Parameters

<i>pid</i>	The pid for the thread to add into the background.
------------	--

#### 4.16.1.3 p\_add\_stop\_job()

```
void p_add_stop_job (
    int pid )
```

The helper function to add a thread into the stopped queue.

##### Parameters

<i>pid</i>	The pid for the thread to add into the stopped queue.
------------	---

#### 4.16.1.4 p\_busy\_wait()

```
void p_busy_wait ( )
```

The function to busy wait (used for 'busy') command.

#### 4.16.1.5 p\_exit()

```
void p_exit (
    void )
```

The function we used to exit the current thread unconditionally and check/log for zombie/orphan status.

#### 4.16.1.6 p\_exit\_process()

```
void p_exit_process ( )
```

The function we used to exit a certain thread (pid stored in `to_exit`) unconditionally and check/log for zombie/orphan status.

#### 4.16.1.7 p\_get\_current()

```
struct Process* p_get_current ( )
```

A helper function to get the current PCB for abstraction sake.

##### Returns

The PCB for the currently running thread.

#### 4.16.1.8 p\_get\_sigcont\_signal()

```
int p_get_sigcont_signal ( )
```

A helper function to get the S\_SIGCONT we defined for abstraction.

##### Returns

The S\_SIGCONT we defined.

#### 4.16.1.9 p\_get\_sigcont\_str()

```
char* p_get_sigcont_str ( )
```

The helper function we used to redirect into kernel and get S\_SIGCONT\_STR we defined inside of kernel for shell use.

##### Returns

The S\_SIGSTOP\_STR we defined inside of kernel.

#### 4.16.1.10 p\_get\_sigstop\_signal()

```
int p_get_sigstop_signal ( )
```

A helper function to get the S\_SIGSTOP we defined for abstraction.

##### Returns

The S\_SIGSTOP we defined.

#### 4.16.1.11 p\_get\_sigstop\_str()

```
char* p_get_sigstop_str ( )
```

The helper function we used to redirect into kernel and get S\_SIGSTOP\_STR we defined inside of kernel for shell use.

##### Returns

The S\_SIGSTOP\_STR we defined inside of kernel.

#### 4.16.1.12 p\_get\_sigterm\_signal()

```
int p_get_sigterm_signal ( )
```

A helper function to get the S\_SIGTERM we defined for abstraction.

##### Returns

The S\_SIGTERM we defined.

#### 4.16.1.13 p\_get\_sigterm\_str()

```
char* p_get_sigterm_str ( )
```

The helper function we used to redirect into kernel and get S\_SIGTERM\_STR we defined inside of kernel for shell use.

##### Returns

The S\_SIGSTOP\_STR we defined inside of kernel.

#### 4.16.1.14 p\_initiate()

```
void p_initiate ( )
```

A helper function to invoke initialization inside kernel/scheduler for abstraction.

#### 4.16.1.15 p\_initiate\_priorities()

```
void p_initiate_priorities ( )
```

A helper function to invoke scheduler initialization inside kernel/scheduler for abstraction.

#### 4.16.1.16 p\_initiate\_shell()

```
pid_t p_initiate_shell (
    void(*)() func,
    int argc,
    char * argv[] )
```

The helper function we used to invoke the kernel shell initiation and initiate the shell.

**Parameters**

<i>func</i>	The function to run (which is just shell).
<i>argc</i>	The number of arguments passed in.
<i>argv</i>	The arguments passed in.

**Returns**

The pid of the thread created.

**4.16.1.17 p\_initiate\_to\_exit()**

```
void p_initiate_to_exit ( )
```

A helper function to initiate the to\_exit PCB stored for [p\\_exit\\_process\(\)](#) as NULL.

**4.16.1.18 p\_kill()**

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

Function used to send a thread to a running thread.

**Parameters**

<i>pid</i>	The pid of the thread we are trying to send a signal.
<i>sig</i>	The signal we are trying to send.

**Returns**

0 on success, -1 on error.

**4.16.1.19 p\_lookup\_process()**

```
struct Process* p_lookup_process (
    pid_t pid )
```

A helper function to redirect k\_lookup\_process for abstraction sake.

**Parameters**

<i>pid</i>	The pid of the thread we are searching for.
------------	---

**Returns**

The pointer to the PCB of the thread we want, -1 on error.

**4.16.1.20 p\_nice()**

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

The function used to set the priority of the thread with pid to some priority.

**Parameters**

<i>pid</i>	The pid of the thread we want to set its priority.
<i>priority</i>	The new priority for this thread.

**Returns**

0 on success, -1 on error.

**4.16.1.21 p\_orphan\_child()**

```
void p_orphan_child ( )
```

The function used to spawn an orphan child.

**4.16.1.22 p\_orphanify()**

```
void p_orphanify ( )
```

The function we used to deal with 'orphanify' command, which spawns an orphan child.

**4.16.1.23 p\_print\_all\_jobs()**

```
void p_print_all_jobs ( )
```

The function to print out all the jobs by the order of priority.

**4.16.1.24 p\_remove\_background\_job()**

```
void p_remove_background_job (
    int pid )
```

The helper function to remove a thread from the background.

## Parameters

<i>pid</i>	The pid for the thread to remove from the background queue.
------------	---

**4.16.1.25 p\_remove\_stop\_job()**

```
void p_remove_stop_job (
    int pid )
```

The helper function to remove a thread from the stopped queue.

## Parameters

<i>pid</i>	The pid for the thread to remove from the stopped queue.
------------	--

**4.16.1.26 p\_run\_cat\_fs()**

```
void p_run_cat_fs ( )
```

The function used to p\_spawn processes for 'cat' command. In which we does basically the 'cat' behavior as in bash.

**4.16.1.27 p\_run\_chmod\_fs()**

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

The function used to p\_spawn processes for 'chmod' command. In which we modify the access permission to a file in our filesystem.

## Parameters

<i>filename</i>	The name of the file descriptor.
<i>perm</i>	The target permission of the file descriptor.

**4.16.1.28 p\_run\_cp\_fs()**

```
void p_run_cp_fs (
    char * source,
```

```
char * des,  
int from_host )
```

The function used to `p_spawn` processes for 'cp' command. In which we copy a file into a new file.

#### Parameters

<i>source</i>	The name of the file descriptor to copy from.
<i>des</i>	The name of the file descriptor to copy into.
<i>from_host</i>	Indicator for whether the file is in the host or our own filesystem.

#### 4.16.1.29 `p_run_echo()`

```
void p_run_echo ( )
```

The function used to `p_spawn` processes for 'echo' command. In which we does basically the `echo(1)` behavior as in VM. The inputs will be passed from our struct.

#### 4.16.1.30 `p_run_f_ls_list()`

```
void p_run_f_ls_list ( )
```

The function used to `p_spawn` processes for 'ls' command. In which we list out details of all files in our filesystem. This is the special version when we take in inputs for 'ls', and we only list out the details for the files specified.

#### 4.16.1.31 `p_run_f_ls_null()`

```
void p_run_f_ls_null ( )
```

The function used to `p_spawn` processes for 'ls' command. In which we list out details of all files in our filesystem. This is the special version when we don't take in input, and we just list out the details for all the files.

#### 4.16.1.32 `p_run_kill()`

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

The function used to `p_spawn` processes for 'kill' command. In which we kill processes with some signal.

#### Parameters

<i>sig</i>	The signal we are sending.
<i>pid</i>	The process we are sending the signal from.



#### 4.16.1.33 p\_run\_mv\_fs()

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

The function used to p\_spawn processes for 'mv' command. In which we rename a file in our filesystem.

##### Parameters

<i>source</i>	The original name of the file descriptor.
<i>dest</i>	The target name of the file descriptor.

#### 4.16.1.34 p\_run\_rm\_fs()

```
void p_run_rm_fs ( )
```

The function used to p\_spawn processes for 'rm' command. In which we remove a file from our filesystem. The inputs will be passed from our struct.

#### 4.16.1.35 p\_run\_touch\_fs()

```
void p_run_touch_fs ( )
```

The function used to p\_spawn processes for 'touch' command. In which we touch a file in our filesystem. It creates the file if it does not exist, and it doesn't do anything otherwise.

#### 4.16.1.36 p\_search\_and\_remove()

```
void p_search_and_remove (
    int pid )
```

The helper function to search for a thread and remove this thread.

##### Parameters

<i>pid</i>	The pid for the thread to search and remove.
------------	--

#### 4.16.1.37 p\_search\_bg()

```
struct Process* p_search_bg (
```

```
int pid )
```

The helper function to search if the thread with pid is in the background.

#### Parameters

<i>pid</i>	The pid to search in the background.
------------	--------------------------------------

#### Returns

The PCB of the thread if this pid exists in the background queue, NULL otherwise.

#### 4.16.1.38 p\_search\_most\_recent()

```
int p_search_most_recent ( )
```

The helper function to search for the most recent background/stopped job.

#### Returns

The pid of the most recent background/stopped job.

#### 4.16.1.39 p\_search\_most\_recent\_stop()

```
int p_search_most_recent_stop ( )
```

The helper function to search for the most recent stopped job.

#### Returns

The pid of the most recent stopped job.

#### 4.16.1.40 p\_setup()

```
void p_setup ( )
```

A helper function to invoke setup inside kernel/scheduler for abstraction.

#### 4.16.1.41 p\_sleep()

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

The function used to set the calling process to blocked until ticks of the system clock elapse, and then sets the thread to running.

## Parameters

<i>ticks</i>	The number of ticks to sleep.
--------------	-------------------------------

**4.16.1.42 p\_spawn()**

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

Forks a new thread that retains most of the attributes of the parent thread.

## Parameters

<i>func</i>	The function to execute inside this PCB.
<i>argv</i>	The arguments passed in when executing func.
<i>fd0</i>	The file descriptor for the input file.
<i>fd1</i>	The file descriptor for the output file.

## Returns

The pid of the child thread on success, or -1 on error.

**4.16.1.43 p\_spawn\_with\_input()**

```
pid_t p_spawn_with_input (
    void(*)() func,
    char * argv[],
    int fd0,
    int fd1,
    char ** actual_input )
```

A modified version of p\_spawn to take in input from the terminal. Forks a new thread that retains most of the attributes of the parent thread.

## Parameters

<i>func</i>	The function to execute inside this PCB.
<i>argv</i>	The arguments passed in when executing func.
<i>fd0</i>	The file descriptor for the input file.
<i>fd1</i>	The file descriptor for the output file. (-1 if not specified)
<i>priority</i>	The priority of the thread created.
<i>actual_input</i>	The pointer to the modified input from the terminal.

**Returns**

The pid of the child thread on success, or -1 on error.

**4.16.1.44 p\_spawn\_with\_priority()**

```
pid_t p_spawn_with_priority (
    void(*)() func,
    char * argv[],
    int fd0,
    int fd1,
    int priority )
```

Forks a new thread that retains most of the attributes of the parent thread with a certain priority (nice value).

**Parameters**

<i>func</i>	The function to execute inside this PCB.
<i>argv</i>	The arguments passed in when executing func.
<i>fd0</i>	The file descriptor for the input file.
<i>fd1</i>	The file descriptor for the output file.
<i>priority</i>	The priority of the thread created.

**Returns**

The pid of the child thread on success, or -1 on error.

**4.16.1.45 p\_waitpid()**

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

Set the calling thread as blocked (if nohang is false) until a child of the calling thread changes state.

**Parameters**

<i>pid</i>	The pid the calling thread is trying to wait on.
<i>wstatus</i>	The status pointer to store the wstatus.
<i>nohang</i>	Indicates if the calling thread should be block-waiting on the child.

**Returns**

The pid of the child which has changed state on success, or -1 on error.

#### 4.16.1.46 p\_zombie\_child()

```
void p_zombie_child ( )
```

The function used to spawn a Zombie child.

#### 4.16.1.47 p\_zombify()

```
void p_zombify ( )
```

The function we used to deal with 'zombify' command, which spawns a zombie child.

#### 4.16.1.48 W\_WIFEXITED()

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

A helper function to check if the child terminates normally(calling p\_exit).

##### Parameters

<i>status</i>	The status pointer we are looking at.
---------------	---------------------------------------

##### Returns

True if the child terminates normally, False otherwise.

#### 4.16.1.49 W\_WIFSIGNALED()

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

A helper function to check if the child is terminated by a signal.

##### Parameters

<i>status</i>	The status pointer we are looking at.
---------------	---------------------------------------

##### Returns

True if the child is terminated by a signal, False otherwise.

## 4.16.1.50 W\_WIFSTOPPED()

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

A helper function to check if the child is stopped by a signal.

## Parameters

<i>status</i>	The status pointer we are looking at.
---------------	---------------------------------------

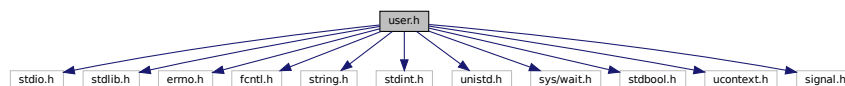
## Returns

True if the child is stopped by a signal, False otherwise.

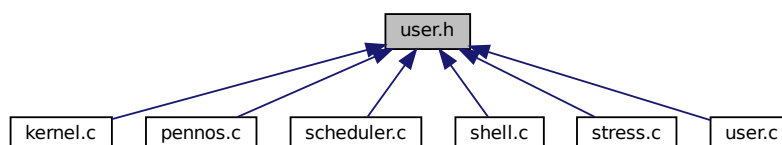
## 4.17 user.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <ucontext.h>
#include <signal.h>
```

Include dependency graph for user.h:



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



## Functions

- pid\_t [p\\_spawn](#) (void(\*func)(), char \*argv[], int fd0, int fd1)
- pid\_t [p\\_spawn\\_with\\_priority](#) (void(\*func)(), char \*argv[], int fd0, int fd1, int priority)
- pid\_t [p\\_waitpid](#) (pid\_t pid, int \*wstatus, bool [nohang](#))
- pid\_t [p\\_initiate\\_shell](#) (void(\*func)(), int argc, char \*argv[])
- int [p\\_kill](#) (pid\_t pid, int sig)
- void [p\\_exit](#) (void)
- void [p\\_exit\\_process](#) ()
- bool [W\\_WIFEXITED](#) (int \*status)
- bool [W\\_WIFSTOPPED](#) (int \*status)
- bool [W\\_WIFSIGNALED](#) (int \*status)
- int [p\\_nice](#) (pid\_t pid, int priority)
- void [p\\_sleep](#) (unsigned int ticks)
- void [p\\_busy\\_wait](#) ()
- void [p\\_zombie\\_child](#) ()
- void [p\\_orphan\\_child](#) ()
- void [p\\_print\\_all\\_jobs](#) ()
- void [p\\_add\\_background\\_job](#) (int pid)
- void [p\\_add\\_stop\\_job](#) (int pid)
- void [p\\_remove\\_background\\_job](#) (int pid)
- void [p\\_remove\\_stop\\_job](#) (int pid)
- void [p\\_search\\_and\\_remove](#) (int pid)
- int [p\\_search\\_most\\_recent](#) ()
- int [p\\_search\\_most\\_recent\\_stop](#) ()
- struct [Process](#) \* [p\\_search\\_bg](#) (int pid)
- int [p\\_get\\_sigstop\\_signal](#) ()
- int [p\\_get\\_sigcont\\_signal](#) ()
- int [p\\_get\\_sigterm\\_signal](#) ()
- void [p\\_initiate\\_to\\_exit](#) ()
- struct [Process](#) \* [p\\_get\\_current](#) ()
- void [p\\_initiate\\_priorities](#) ()
- void [p\\_setup](#) ()
- void [p\\_initiate](#) ()
- pid\_t [p\\_spawn\\_with\\_input](#) (void(\*func)(), char \*argv[], int fd0, int fd1, char \*\*actual\_input)
- struct [Process](#) \* [p\\_lookup\\_process](#) (pid\_t pid)
- void [p\\_zombify](#) ()
- void [p\\_orphanify](#) ()
- void [p\\_run\\_kill](#) (int sig, pid\_t pid)
- void [p\\_run\\_mv\\_fs](#) (char \*source, char \*dest)
- void [p\\_run\\_chmod\\_fs](#) (char \*filename, char \*perm)
- void [p\\_run\\_cp\\_fs](#) (char \*source, char \*des, int from\_host)
- void [p\\_run\\_touch\\_fs](#) ()
- void [p\\_run\\_f\\_ls\\_list](#) ()
- void [p\\_run\\_f\\_ls\\_null](#) ()
- void [p\\_run\\_rm\\_fs](#) ()
- void [p\\_run\\_echo](#) ()
- void [p\\_run\\_cat\\_fs](#) ()
- char \* [p\\_get\\_sigstop\\_str](#) ()
- char \* [p\\_get\\_sigcont\\_str](#) ()
- char \* [p\\_get\\_sigterm\\_str](#) ()

### 4.17.1 Function Documentation

#### 4.17.1.1 `p_add_background_job()`

```
void p_add_background_job (
    int pid )
```

The helper function to add a thread into the background.

##### Parameters

<i>pid</i>	The pid for the thread to add into the background.
------------	--

#### 4.17.1.2 `p_add_stop_job()`

```
void p_add_stop_job (
    int pid )
```

The helper function to add a thread into the stopped queue.

##### Parameters

<i>pid</i>	The pid for the thread to add into the stopped queue.
------------	---

#### 4.17.1.3 `p_busy_wait()`

```
void p_busy_wait ( )
```

The function to busy wait (used for 'busy') command.

#### 4.17.1.4 `p_exit()`

```
void p_exit (
    void )
```

The function we used to exit the current thread unconditionally and check/log for zombie/orphan status.

#### 4.17.1.5 `p_exit_process()`

```
void p_exit_process ( )
```

The function we used to exit a certain thread (pid stored in `to_exit`) unconditionally and check/log for zombie/orphan status.



#### 4.17.1.6 p\_get\_current()

```
struct Process* p_get_current ( )
```

A helper function to get the current PCB for abstraction sake.

##### Returns

The PCB for the currently running thread.

#### 4.17.1.7 p\_get\_sigcont\_signal()

```
int p_get_sigcont_signal ( )
```

A helper function to get the S\_SIGCONT we defined for abstraction.

##### Returns

The S\_SIGCONT we defined.

#### 4.17.1.8 p\_get\_sigcont\_str()

```
char* p_get_sigcont_str ( )
```

The helper function we used to redirect into kernel and get S\_SIGCONT\_STR we defined inside of kernel for shell use.

##### Returns

The S\_SIGSTOP\_STR we defined inside of kernel.

#### 4.17.1.9 p\_get\_sigstop\_signal()

```
int p_get_sigstop_signal ( )
```

A helper function to get the S\_SIGSTOP we defined for abstraction.

##### Returns

The S\_SIGSTOP we defined.

#### 4.17.1.10 p\_get\_sigstop\_str()

```
char* p_get_sigstop_str ( )
```

The helper function we used to redirect into kernel and get S\_SIGSTOP\_STR we defined inside of kernel for shell use.

##### Returns

The S\_SIGSTOP\_STR we defined inside of kernel.

#### 4.17.1.11 p\_get\_sigterm\_signal()

```
int p_get_sigterm_signal ( )
```

A helper function to get the S\_SIGTERM we defined for abstraction.

##### Returns

The S\_SIGTERM we defined.

#### 4.17.1.12 p\_get\_sigterm\_str()

```
char* p_get_sigterm_str ( )
```

The helper function we used to redirect into kernel and get S\_SIGTERM\_STR we defined inside of kernel for shell use.

##### Returns

The S\_SIGSTOP\_STR we defined inside of kernel.

#### 4.17.1.13 p\_initiate()

```
void p_initiate ( )
```

A helper function to invoke initialization inside kernel/scheduler for abstraction.

#### 4.17.1.14 p\_initiate\_priorities()

```
void p_initiate_priorities ( )
```

A helper function to invoke scheduler initialization inside kernel/scheduler for abstraction.

#### 4.17.1.15 p\_initiate\_shell()

```
pid_t p_initiate_shell (
    void(*)() func,
    int argc,
    char * argv[] )
```

The helper function we used to invoke the kernel shell initiation and initiate the shell.

**Parameters**

<i>func</i>	The function to run (which is just shell).
<i>argc</i>	The number of arguments passed in.
<i>argv</i>	The arguments passed in.

**Returns**

The pid of the thread created.

**4.17.1.16 p\_initiate\_to\_exit()**

```
void p_initiate_to_exit ( )
```

A helper function to initiate the to\_exit PCB stored for [p\\_exit\\_process\(\)](#) as NULL.

**4.17.1.17 p\_kill()**

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

Function used to send a thread to a running thread.

**Parameters**

<i>pid</i>	The pid of the thread we are trying to send a signal.
<i>sig</i>	The signal we are trying to send.

**Returns**

0 on success, -1 on error.

**4.17.1.18 p\_lookup\_process()**

```
struct Process* p_lookup_process (
    pid_t pid )
```

A helper function to redirect k\_lookup\_process for abstraction sake.

**Parameters**

<i>pid</i>	The pid of the thread we are searching for.
------------	---

**Returns**

The pointer to the PCB of the thread we want, -1 on error.

**4.17.1.19 p\_nice()**

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

The function used to set the priority of the thread with pid to some priority.

**Parameters**

<i>pid</i>	The pid of the thread we want to set its priority.
<i>priority</i>	The new priority for this thread.

**Returns**

0 on success, -1 on error.

**4.17.1.20 p\_orphan\_child()**

```
void p_orphan_child ( )
```

The function used to spawn an orphan child.

**4.17.1.21 p\_orphanify()**

```
void p_orphanify ( )
```

The function we used to deal with 'orphanify' command, which spawns an orphan child.

**4.17.1.22 p\_print\_all\_jobs()**

```
void p_print_all_jobs ( )
```

The function to print out all the jobs by the order of priority.

**4.17.1.23 p\_remove\_background\_job()**

```
void p_remove_background_job (
    int pid )
```

The helper function to remove a thread from the background.

## Parameters

<i>pid</i>	The pid for the thread to remove from the background queue.
------------	---

**4.17.1.24 p\_remove\_stop\_job()**

```
void p_remove_stop_job (
    int pid )
```

The helper function to remove a thread from the stopped queue.

## Parameters

<i>pid</i>	The pid for the thread to remove from the stopped queue.
------------	--

**4.17.1.25 p\_run\_cat\_fs()**

```
void p_run_cat_fs ( )
```

The function used to p\_spawn processes for 'cat' command. In which we does basically the 'cat' behavior as in bash.

**4.17.1.26 p\_run\_chmod\_fs()**

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

The function used to p\_spawn processes for 'chmod' command. In which we modify the access permission to a file in our filesystem.

## Parameters

<i>filename</i>	The name of the file descriptor.
<i>perm</i>	The target permission of the file descriptor.

**4.17.1.27 p\_run\_cp\_fs()**

```
void p_run_cp_fs (
    char * source,
```

```
char * des,  
int from_host )
```

The function used to `p_spawn` processes for 'cp' command. In which we copy a file into a new file.

#### Parameters

<i>source</i>	The name of the file descriptor to copy from.
<i>des</i>	The name of the file descriptor to copy into.
<i>from_host</i>	Indicator for whether the file is in the host or our own filesystem.

#### 4.17.1.28 `p_run_echo()`

```
void p_run_echo ( )
```

The function used to `p_spawn` processes for 'echo' command. In which we does basically the `echo(1)` behavior as in VM. The inputs will be passed from our struct.

#### 4.17.1.29 `p_run_f_ls_list()`

```
void p_run_f_ls_list ( )
```

The function used to `p_spawn` processes for 'ls' command. In which we list out details of all files in our filesystem. This is the special version when we take in inputs for 'ls', and we only list out the details for the files specified.

#### 4.17.1.30 `p_run_f_ls_null()`

```
void p_run_f_ls_null ( )
```

The function used to `p_spawn` processes for 'ls' command. In which we list out details of all files in our filesystem. This is the special version when we don't take in input, and we just list out the details for all the files.

#### 4.17.1.31 `p_run_kill()`

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

The function used to `p_spawn` processes for 'kill' command. In which we kill processes with some signal.

#### Parameters

<i>sig</i>	The signal we are sending.
<i>pid</i>	The process we are sending the signal from.

#### 4.17.1.32 p\_run\_mv\_fs()

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

The function used to p\_spawn processes for 'mv' command. In which we rename a file in our filesystem.

##### Parameters

<i>source</i>	The original name of the file descriptor.
<i>dest</i>	The target name of the file descriptor.

#### 4.17.1.33 p\_run\_rm\_fs()

```
void p_run_rm_fs ( )
```

The function used to p\_spawn processes for 'rm' command. In which we remove a file from our filesystem. The inputs will be passed from our struct.

#### 4.17.1.34 p\_run\_touch\_fs()

```
void p_run_touch_fs ( )
```

The function used to p\_spawn processes for 'touch' command. In which we touch a file in our filesystem. It creates the file if it does not exist, and it doesn't do anything otherwise.

#### 4.17.1.35 p\_search\_and\_remove()

```
void p_search_and_remove (
    int pid )
```

The helper function to search for a thread and remove this thread.

##### Parameters

<i>pid</i>	The pid for the thread to search and remove.
------------	--

#### 4.17.1.36 p\_search\_bg()

```
struct Process* p_search_bg (
```

```
int pid )
```

The helper function to search if the thread with pid is in the background.

#### Parameters

<i>pid</i>	The pid to search in the background.
------------	--------------------------------------

#### Returns

The PCB of the thread if this pid exists in the background queue, NULL otherwise.

#### 4.17.1.37 p\_search\_most\_recent()

```
int p_search_most_recent ( )
```

The helper function to search for the most recent background/stopped job.

#### Returns

The pid of the most recent background/stopped job.

#### 4.17.1.38 p\_search\_most\_recent\_stop()

```
int p_search_most_recent_stop ( )
```

The helper function to search for the most recent stopped job.

#### Returns

The pid of the most recent stopped job.

#### 4.17.1.39 p\_setup()

```
void p_setup ( )
```

A helper function to invoke setup inside kernel/scheduler for abstraction.

#### 4.17.1.40 p\_sleep()

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

The function used to set the calling process to blocked until ticks of the system clock elapse, and then sets the thread to running.



## Parameters

<i>ticks</i>	The number of ticks to sleep.
--------------	-------------------------------

**4.17.1.41 p\_spawn()**

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

Forks a new thread that retains most of the attributes of the parent thread.

## Parameters

<i>func</i>	The function to execute inside this PCB.
<i>argv</i>	The arguments passed in when executing func.
<i>fd0</i>	The file descriptor for the input file.
<i>fd1</i>	The file descriptor for the output file.

## Returns

The pid of the child thread on success, or -1 on error.

**4.17.1.42 p\_spawn\_with\_input()**

```
pid_t p_spawn_with_input (
    void(*)() func,
    char * argv[],
    int fd0,
    int fd1,
    char ** actual_input )
```

A modified version of p\_spawn to take in input from the terminal. Forks a new thread that retains most of the attributes of the parent thread.

## Parameters

<i>func</i>	The function to execute inside this PCB.
<i>argv</i>	The arguments passed in when executing func.
<i>fd0</i>	The file descriptor for the input file.
<i>fd1</i>	The file descriptor for the output file. (-1 if not specified)
<i>priority</i>	The priority of the thread created.
<i>actual_input</i>	The pointer to the modified input from the terminal.

**Returns**

The pid of the child thread on success, or -1 on error.

**4.17.1.43 p\_spawn\_with\_priority()**

```
pid_t p_spawn_with_priority (
    void(*)() func,
    char * argv[],
    int fd0,
    int fd1,
    int priority )
```

Forks a new thread that retains most of the attributes of the parent thread with a certain priority (nice value).

**Parameters**

<i>func</i>	The function to execute inside this PCB.
<i>argv</i>	The arguments passed in when executing func.
<i>fd0</i>	The file descriptor for the input file.
<i>fd1</i>	The file descriptor for the output file.
<i>priority</i>	The priority of the thread created.

**Returns**

The pid of the child thread on success, or -1 on error.

**4.17.1.44 p\_waitpid()**

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

Set the calling thread as blocked (if nohang is false) until a child of the calling thread changes state.

**Parameters**

<i>pid</i>	The pid the calling thread is trying to wait on.
<i>wstatus</i>	The status pointer to store the wstatus.
<i>nohang</i>	Indicates if the calling thread should be block-waiting on the child.

**Returns**

The pid of the child which has changed state on success, or -1 on error.

#### 4.17.1.45 p\_zombie\_child()

```
void p_zombie_child ( )
```

The function used to spawn a Zombie child.

#### 4.17.1.46 p\_zombify()

```
void p_zombify ( )
```

The function we used to deal with 'zombify' command, which spawns a zombie child.

#### 4.17.1.47 W\_WIFEXITED()

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

A helper function to check if the child terminates normally(calling p\_exit).

##### Parameters

<i>status</i>	The status pointer we are looking at.
---------------	---------------------------------------

##### Returns

True if the child terminates normally, False otherwise.

#### 4.17.1.48 W\_WIFSIGNALED()

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

A helper function to check if the child is terminated by a signal.

##### Parameters

<i>status</i>	The status pointer we are looking at.
---------------	---------------------------------------

##### Returns

True if the child is terminated by a signal, False otherwise.

#### 4.17.1.49 W\_WIFSTOPPED()

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

A helper function to check if the child is stopped by a signal.

##### Parameters

<i>status</i>	The status pointer we are looking at.
---------------	---------------------------------------

##### Returns

True if the child is stopped by a signal, False otherwise.



# Index

ACTIVE\_STAT  
    [linkedlist-job.h, 35](#)

argv  
    Process, [10](#)

awake\_time  
    Process, [10](#)

bg\_time  
    Process, [10](#)

check\_actual  
    [scheduler.c, 49](#)

childrens  
    Process, [10](#)

cmd  
    Process, [10](#)

commands  
    [parsed\\_command, 8](#)

context  
    Process, [10](#)

delete\_node  
    [linkedlist-job.c, 30](#)  
    [linkedlist-job.h, 35](#)

DONE\_STAT  
    [linkedlist-job.h, 35](#)

ENTRY  
    [linkedlist-job.h, 35](#)

Entry, [5](#)  
    [next, 6](#)  
    [prev, 6](#)  
    [process, 6](#)

EXPECT\_COMMANDS  
    [parser.h, 43](#)

EXPECT\_INPUT\_FILENAME  
    [parser.h, 43](#)

EXPECT\_OUTPUT\_FILENAME  
    [parser.h, 43](#)

fg\_cont  
    Process, [10](#)

free\_list  
    [linkedlist-job.c, 31](#)  
    [linkedlist-job.h, 36](#)

free\_process  
    [linkedlist-job.c, 31](#)  
    [linkedlist-job.h, 36](#)

freeStacks  
    [scheduler.c, 49](#)  
    [scheduler.h, 56](#)

get\_fg\_pid  
    [shell.c, 62](#)  
    [shell.h, 64](#)

getSignal  
    [user.c, 68](#)

group\_id  
    Process, [11](#)

handler  
    [pennos.c, 45](#)  
    [pennos.h, 47](#)

hang  
    [stress.c, 65](#)  
    [stress.h, 66](#)

head  
    [LinkedList, 7](#)

input\_descriptor  
    Process, [11](#)

insert\_end  
    [linkedlist-job.c, 31](#)  
    [linkedlist-job.h, 36](#)

is\_background  
    [parsed\\_command, 8](#)

is\_file\_append  
    [parsed\\_command, 8](#)

is\_orphan  
    Process, [11](#)

k\_get\_next\_pid  
    [kernel.c, 14](#)  
    [kernel.h, 23](#)

k\_get\_running\_status  
    [kernel.c, 15](#)  
    [kernel.h, 23](#)

k\_get\_sigcont\_signal  
    [kernel.c, 15](#)  
    [kernel.h, 23](#)

k\_get\_sigcont\_str  
    [kernel.c, 15](#)  
    [kernel.h, 24](#)

k\_get\_sigstop\_signal  
    [kernel.c, 15](#)  
    [kernel.h, 24](#)

k\_get\_sigstop\_str  
    [kernel.c, 16](#)  
    [kernel.h, 24](#)

k\_get\_sigterm\_signal  
    [kernel.c, 16](#)  
    [kernel.h, 24](#)

- k\_get\_sigterm\_str
  - kernel.c, 16
  - kernel.h, 25
- k\_get\_stop\_signal\_status
  - kernel.c, 16
  - kernel.h, 25
- k\_get\_terminal\_normal\_status
  - kernel.c, 17
  - kernel.h, 25
- k\_get\_terminal\_signal\_status
  - kernel.c, 17
  - kernel.h, 25
- k\_get\_to\_exit
  - kernel.c, 17
  - kernel.h, 26
- k\_initiate\_to\_exit
  - kernel.c, 17
  - kernel.h, 26
- k\_kill\_all
  - kernel.c, 17
  - kernel.h, 26
- k\_lookup\_process
  - kernel.c, 18
  - kernel.h, 26
- k\_process\_cleanup
  - kernel.c, 18
  - kernel.h, 28
- k\_process\_create
  - kernel.c, 18
  - kernel.h, 28
- k\_process\_create\_with\_priority
  - kernel.c, 19
  - kernel.h, 28
- k\_process\_kill
  - kernel.c, 19
  - kernel.h, 29
- k\_reap\_zombie
  - kernel.c, 19
  - kernel.h, 29
- k\_set\_idle
  - kernel.c, 20
  - kernel.h, 29
- k\_set\_to\_exit
  - kernel.c, 20
  - kernel.h, 29
- kernel.c, 13
  - k\_get\_next\_pid, 14
  - k\_get\_running\_status, 15
  - k\_get\_sigcont\_signal, 15
  - k\_get\_sigcont\_str, 15
  - k\_get\_sigstop\_signal, 15
  - k\_get\_sigstop\_str, 16
  - k\_get\_sigterm\_signal, 16
  - k\_get\_sigterm\_str, 16
  - k\_get\_stop\_signal\_status, 16
  - k\_get\_terminal\_normal\_status, 17
  - k\_get\_terminal\_signal\_status, 17
  - k\_get\_to\_exit, 17
  - k\_initiate\_to\_exit, 17
  - k\_kill\_all, 17
  - k\_lookup\_process, 18
  - k\_process\_cleanup, 18
  - k\_process\_create, 18
  - k\_process\_create\_with\_priority, 19
  - k\_process\_kill, 19
  - k\_reap\_zombie, 19
  - k\_set\_idle, 20
  - k\_set\_to\_exit, 20
  - S\_SIGCONT, 14
  - S\_SIGSTOP, 14
  - S\_SIGTERM, 14
- kernel.h, 20
  - k\_get\_next\_pid, 23
  - k\_get\_running\_status, 23
  - k\_get\_sigcont\_signal, 23
  - k\_get\_sigcont\_str, 24
  - k\_get\_sigstop\_signal, 24
  - k\_get\_sigstop\_str, 24
  - k\_get\_sigterm\_signal, 24
  - k\_get\_sigterm\_str, 25
  - k\_get\_stop\_signal\_status, 25
  - k\_get\_terminal\_normal\_status, 25
  - k\_get\_terminal\_signal\_status, 25
  - k\_get\_to\_exit, 26
  - k\_initiate\_to\_exit, 26
  - k\_kill\_all, 26
  - k\_lookup\_process, 26
  - k\_process\_cleanup, 28
  - k\_process\_create, 28
  - k\_process\_create\_with\_priority, 28
  - k\_process\_kill, 29
  - k\_reap\_zombie, 29
  - k\_set\_idle, 29
  - k\_set\_to\_exit, 29
  - RUNNING, 22
  - S\_SIGCONT, 22
  - S\_SIGCONT\_STR, 22
  - S\_SIGSTOP, 22
  - S\_SIGSTOP\_STR, 22
  - S\_SIGTERM, 22
  - S\_SIGTERM\_STR, 22
  - STOP\_SIGNAL, 22
  - TERMINATE\_NORMAL, 23
  - TERMINATE\_SIGNAL, 23
- LinkedList, 6
  - head, 7
  - tail, 7
- linkedlist-job.c, 30
  - delete\_node, 30
  - free\_list, 31
  - free\_process, 31
  - insert\_end, 31
  - poll, 32
  - retrieve\_latest, 32
  - search\_list, 32
  - set\_orphan, 33

- linkedList-job.h, 33
  - ACTIVE\_STAT, 35
  - delete\_node, 35
  - DONE\_STAT, 35
  - ENTRY, 35
  - free\_list, 36
  - free\_process, 36
  - insert\_end, 36
  - LIST, 35
  - PAUSED\_STAT, 35
  - poll, 37
  - print\_list, 37
  - PROCESS, 35
  - retrieve\_latest, 37
  - search\_list, 37
  - set\_orphan, 38
  - STOP\_STAT, 35
- LIST
  - linkedList-job.h, 35
- log\_event
  - logger.c, 39
  - logger.h, 41
- log\_nice\_event
  - logger.c, 39
  - logger.h, 41
- logger.c, 38
  - log\_event, 39
  - log\_nice\_event, 39
  - open\_log\_file, 40
- logger.h, 40
  - log\_event, 41
  - log\_nice\_event, 41
- main
  - pennos.c, 45
  - pennos.h, 47
- next
  - Entry, 6
- nohang
  - stress.c, 65
  - stress.h, 66
- NOT\_WAITING
  - scheduler.c, 48
  - scheduler.h, 56
- num\_children
  - Process, 11
- num\_commands
  - parsed\_command, 8
- open\_log\_file
  - logger.c, 40
- output\_descriptor
  - Process, 11
- p\_add\_background\_job
  - user.c, 68
  - user.h, 82
- p\_add\_stop\_job
  - user.c, 69
  - user.h, 83
- p\_busy\_wait
  - user.c, 69
  - user.h, 83
- p\_exit
  - user.c, 69
  - user.h, 83
- p\_exit\_process
  - user.c, 69
  - user.h, 83
- p\_get\_current
  - user.c, 69
  - user.h, 83
- p\_get\_sigcont\_signal
  - user.c, 70
  - user.h, 84
- p\_get\_sigcont\_str
  - user.c, 70
  - user.h, 84
- p\_get\_sigstop\_signal
  - user.c, 70
  - user.h, 84
- p\_get\_sigstop\_str
  - user.c, 70
  - user.h, 84
- p\_get\_sigterm\_signal
  - user.c, 71
  - user.h, 85
- p\_get\_sigterm\_str
  - user.c, 71
  - user.h, 85
- p\_initiate
  - user.c, 71
  - user.h, 85
- p\_initiate\_priorities
  - user.c, 71
  - user.h, 85
- p\_initiate\_shell
  - user.c, 71
  - user.h, 85
- p\_initiate\_to\_exit
  - user.c, 72
  - user.h, 86
- p\_kill
  - user.c, 72
  - user.h, 86
- p\_lookup\_process
  - user.c, 72
  - user.h, 86
- p\_nice
  - user.c, 73
  - user.h, 87
- p\_orphan\_child
  - user.c, 73
  - user.h, 87
- p\_orphanify
  - user.c, 73



- user.h, [87](#)
- p\_print\_all\_jobs
  - user.c, [73](#)
  - user.h, [87](#)
- p\_remove\_background\_job
  - user.c, [73](#)
  - user.h, [87](#)
- p\_remove\_stop\_job
  - user.c, [74](#)
  - user.h, [88](#)
- p\_run\_cat\_fs
  - user.c, [74](#)
  - user.h, [88](#)
- p\_run\_chmod\_fs
  - user.c, [74](#)
  - user.h, [88](#)
- p\_run\_cp\_fs
  - user.c, [74](#)
  - user.h, [88](#)
- p\_run\_echo
  - user.c, [75](#)
  - user.h, [89](#)
- p\_run\_f\_ls\_list
  - user.c, [75](#)
  - user.h, [89](#)
- p\_run\_f\_ls\_null
  - user.c, [75](#)
  - user.h, [89](#)
- p\_run\_kill
  - user.c, [75](#)
  - user.h, [89](#)
- p\_run\_mv\_fs
  - user.c, [76](#)
  - user.h, [90](#)
- p\_run\_rm\_fs
  - user.c, [76](#)
  - user.h, [90](#)
- p\_run\_touch\_fs
  - user.c, [76](#)
  - user.h, [90](#)
- p\_search\_and\_remove
  - user.c, [76](#)
  - user.h, [90](#)
- p\_search\_bg
  - user.c, [76](#)
  - user.h, [90](#)
- p\_search\_most\_recent
  - user.c, [77](#)
  - user.h, [91](#)
- p\_search\_most\_recent\_stop
  - user.c, [77](#)
  - user.h, [91](#)
- p\_setup
  - user.c, [77](#)
  - user.h, [91](#)
- p\_sleep
  - user.c, [77](#)
  - user.h, [91](#)
- p\_spawn
  - user.c, [78](#)
  - user.h, [92](#)
- p\_spawn\_with\_input
  - user.c, [78](#)
  - user.h, [92](#)
- p\_spawn\_with\_priority
  - user.c, [79](#)
  - user.h, [93](#)
- p\_waitpid
  - user.c, [79](#)
  - user.h, [93](#)
- p\_zombie\_child
  - user.c, [79](#)
  - user.h, [93](#)
- p\_zombify
  - user.c, [80](#)
  - user.h, [94](#)
- parent
  - Process, [11](#)
- parent\_process\_id
  - Process, [11](#)
- parse\_command
  - parser.h, [44](#)
- parsed\_command, [7](#)
  - commands, [8](#)
  - is\_background, [8](#)
  - is\_file\_append, [8](#)
  - num\_commands, [8](#)
  - stdin\_file, [8](#)
  - stdout\_file, [8](#)
- parser.h, [41](#)
  - EXPECT\_COMMANDS, [43](#)
  - EXPECT\_INPUT\_FILENAME, [43](#)
  - EXPECT\_OUTPUT\_FILENAME, [43](#)
  - parse\_command, [44](#)
  - print\_parsed\_command, [44](#)
  - UNEXPECTED\_AMPERSAND, [43](#)
  - UNEXPECTED\_FILE\_INPUT, [43](#)
  - UNEXPECTED\_FILE\_OUTPUT, [43](#)
  - UNEXPECTED\_PIPELINE, [43](#)
- PAUSED\_STAT
  - linkedlist-job.h, [35](#)
- pennos.c, [44](#)
  - handler, [45](#)
  - main, [45](#)
- pennos.h, [46](#)
  - handler, [47](#)
  - main, [47](#)
- poll
  - linkedlist-job.c, [32](#)
  - linkedlist-job.h, [37](#)
- prev
  - Entry, [6](#)
- print\_list
  - linkedlist-job.h, [37](#)
- print\_parsed\_command
  - parser.h, [44](#)

- priority
  - Process, [11](#)
- PROCESS
  - linkedlist-job.h, [35](#)
- Process, [9](#)
  - argv, [10](#)
  - awake\_time, [10](#)
  - bg\_time, [10](#)
  - childrens, [10](#)
  - cmd, [10](#)
  - context, [10](#)
  - fg\_cont, [10](#)
  - group\_id, [11](#)
  - input\_descriptor, [11](#)
  - is\_orphan, [11](#)
  - num\_children, [11](#)
  - output\_descriptor, [11](#)
  - parent, [11](#)
  - parent\_process\_id, [11](#)
  - priority, [11](#)
  - recorded, [12](#)
  - signal\_terminated, [12](#)
  - status, [12](#)
  - stop\_time, [12](#)
  - thread\_process\_id, [12](#)
  - to\_wait, [12](#)
- process
  - Entry, [6](#)
- recorded
  - Process, [12](#)
- recur
  - stress.c, [66](#)
  - stress.h, [67](#)
- retrieve\_latest
  - linkedlist-job.c, [32](#)
  - linkedlist-job.h, [37](#)
- RUNNING
  - kernel.h, [22](#)
- s\_check\_active
  - scheduler.c, [49](#)
  - scheduler.h, [56](#)
- s\_get\_current
  - scheduler.c, [49](#)
  - scheduler.h, [56](#)
- s\_get\_priority
  - scheduler.c, [49](#)
  - scheduler.h, [56](#)
- s\_get\_scheduler\_context
  - scheduler.c, [50](#)
  - scheduler.h, [57](#)
- s\_get\_time
  - scheduler.c, [50](#)
  - scheduler.h, [57](#)
- s\_get\_zombie\_context
  - scheduler.c, [50](#)
  - scheduler.h, [57](#)
- s\_initiate
  - scheduler.c, [50](#)
  - scheduler.h, [57](#)
- s\_initiate\_priorities
  - scheduler.c, [51](#)
  - scheduler.h, [58](#)
- s\_initiate\_shell\_context
  - scheduler.c, [51](#)
  - scheduler.h, [58](#)
- s\_insert
  - scheduler.c, [51](#)
  - scheduler.h, [58](#)
- s\_makeContext
  - scheduler.c, [51](#)
  - scheduler.h, [58](#)
- s\_print\_all\_jobs
  - scheduler.c, [52](#)
  - scheduler.h, [59](#)
- s\_set
  - scheduler.c, [52](#)
  - scheduler.h, [59](#)
- s\_set\_current
  - scheduler.c, [52](#)
  - scheduler.h, [59](#)
- s\_set\_idle
  - scheduler.c, [52](#)
  - scheduler.h, [59](#)
- s\_set\_status
  - scheduler.c, [52](#)
  - scheduler.h, [59](#)
- s\_setup
  - scheduler.c, [54](#)
  - scheduler.h, [61](#)
- S\_SIGCONT
  - kernel.c, [14](#)
  - kernel.h, [22](#)
- S\_SIGCONT\_STR
  - kernel.h, [22](#)
- S\_SIGSTOP
  - kernel.c, [14](#)
  - kernel.h, [22](#)
- S\_SIGSTOP\_STR
  - kernel.h, [22](#)
- S\_SIGTERM
  - kernel.c, [14](#)
  - kernel.h, [22](#)
- S\_SIGTERM\_STR
  - kernel.h, [22](#)
- s\_swap
  - scheduler.c, [54](#)
  - scheduler.h, [61](#)
- scheduler.c, [47](#)
  - check\_actual, [49](#)
  - freeStacks, [49](#)
  - NOT\_WAITING, [48](#)
  - s\_check\_active, [49](#)
  - s\_get\_current, [49](#)
  - s\_get\_priority, [49](#)
  - s\_get\_scheduler\_context, [50](#)

- s\_get\_time, 50
- s\_get\_zombie\_context, 50
- s\_initiate, 50
- s\_initiate\_priorities, 51
- s\_initiate\_shell\_context, 51
- s\_insert, 51
- s\_makeContext, 51
- s\_print\_all\_jobs, 52
- s\_set, 52
- s\_set\_current, 52
- s\_set\_idle, 52
- s\_set\_status, 52
- s\_setup, 54
- s\_swap, 54
- setStack, 54
- THREAD\_COUNT, 48
- scheduler.h, 54
  - freeStacks, 56
  - NOT\_WAITING, 56
  - s\_check\_active, 56
  - s\_get\_current, 56
  - s\_get\_priority, 56
  - s\_get\_scheduler\_context, 57
  - s\_get\_time, 57
  - s\_get\_zombie\_context, 57
  - s\_initiate, 57
  - s\_initiate\_priorities, 58
  - s\_initiate\_shell\_context, 58
  - s\_insert, 58
  - s\_makeContext, 58
  - s\_print\_all\_jobs, 59
  - s\_set, 59
  - s\_set\_current, 59
  - s\_set\_idle, 59
  - s\_set\_status, 59
  - s\_setup, 61
  - s\_swap, 61
  - setStack, 61
- search\_list
  - linkedlist-job.c, 32
  - linkedlist-job.h, 37
- set\_fg\_pid
  - shell.c, 62
  - shell.h, 64
- set\_orphan
  - linkedlist-job.c, 33
  - linkedlist-job.h, 38
- setStack
  - scheduler.c, 54
  - scheduler.h, 61
- shell
  - shell.c, 63
  - shell.h, 65
- shell.c, 61
  - get\_fg\_pid, 62
  - set\_fg\_pid, 62
  - shell, 63
  - sigstp\_handler, 63
- sigstp\_handler, 63
- shell.h, 63
  - get\_fg\_pid, 64
  - set\_fg\_pid, 64
  - shell, 65
- sigint\_handler
  - shell.c, 63
- signal\_terminated
  - Process, 12
- sigstp\_handler
  - shell.c, 63
- status
  - Process, 12
- stdin\_file
  - parsed\_command, 8
- stdout\_file
  - parsed\_command, 8
- STOP\_SIGNAL
  - kernel.h, 22
- STOP\_STAT
  - linkedlist-job.h, 35
- stop\_time
  - Process, 12
- stress.c, 65
  - hang, 65
  - nohang, 65
  - recur, 66
- stress.h, 66
  - hang, 66
  - nohang, 66
  - recur, 67
- tail
  - LinkedList, 7
- TERMINATE\_NORMAL
  - kernel.h, 23
- TERMINATE\_SIGNAL
  - kernel.h, 23
- THREAD\_COUNT
  - scheduler.c, 48
- thread\_process\_id
  - Process, 12
- to\_wait
  - Process, 12
- UNEXPECTED\_AMPERSAND
  - parser.h, 43
- UNEXPECTED\_FILE\_INPUT
  - parser.h, 43
- UNEXPECTED\_FILE\_OUTPUT
  - parser.h, 43
- UNEXPECTED\_PIPELINE
  - parser.h, 43
- user.c, 67
  - getSignal, 68
  - p\_add\_background\_job, 68
  - p\_add\_stop\_job, 69
  - p\_busy\_wait, 69
  - p\_exit, 69

- p\_exit\_process, 69
- p\_get\_current, 69
- p\_get\_sigcont\_signal, 70
- p\_get\_sigcont\_str, 70
- p\_get\_sigstop\_signal, 70
- p\_get\_sigstop\_str, 70
- p\_get\_sigterm\_signal, 71
- p\_get\_sigterm\_str, 71
- p\_initiate, 71
- p\_initiate\_priorities, 71
- p\_initiate\_shell, 71
- p\_initiate\_to\_exit, 72
- p\_kill, 72
- p\_lookup\_process, 72
- p\_nice, 73
- p\_orphan\_child, 73
- p\_orphanify, 73
- p\_print\_all\_jobs, 73
- p\_remove\_background\_job, 73
- p\_remove\_stop\_job, 74
- p\_run\_cat\_fs, 74
- p\_run\_chmod\_fs, 74
- p\_run\_cp\_fs, 74
- p\_run\_echo, 75
- p\_run\_f\_ls\_list, 75
- p\_run\_f\_ls\_null, 75
- p\_run\_kill, 75
- p\_run\_mv\_fs, 76
- p\_run\_rm\_fs, 76
- p\_run\_touch\_fs, 76
- p\_search\_and\_remove, 76
- p\_search\_bg, 76
- p\_search\_most\_recent, 77
- p\_search\_most\_recent\_stop, 77
- p\_setup, 77
- p\_sleep, 77
- p\_spawn, 78
- p\_spawn\_with\_input, 78
- p\_spawn\_with\_priority, 79
- p\_waitpid, 79
- p\_zombie\_child, 79
- p\_zombify, 80
- W\_WIFEXITED, 80
- W\_WIFSIGNALED, 80
- W\_WIFSTOPPED, 80
- user.h, 81
  - p\_add\_background\_job, 82
  - p\_add\_stop\_job, 83
  - p\_busy\_wait, 83
  - p\_exit, 83
  - p\_exit\_process, 83
  - p\_get\_current, 83
  - p\_get\_sigcont\_signal, 84
  - p\_get\_sigcont\_str, 84
  - p\_get\_sigstop\_signal, 84
  - p\_get\_sigstop\_str, 84
  - p\_get\_sigterm\_signal, 85
  - p\_get\_sigterm\_str, 85
  - p\_initiate, 85
  - p\_initiate\_priorities, 85
  - p\_initiate\_shell, 85
  - p\_initiate\_to\_exit, 86
  - p\_kill, 86
  - p\_lookup\_process, 86
  - p\_nice, 87
  - p\_orphan\_child, 87
  - p\_orphanify, 87
  - p\_print\_all\_jobs, 87
  - p\_remove\_background\_job, 87
  - p\_remove\_stop\_job, 88
  - p\_run\_cat\_fs, 88
  - p\_run\_chmod\_fs, 88
  - p\_run\_cp\_fs, 88
  - p\_run\_echo, 89
  - p\_run\_f\_ls\_list, 89
  - p\_run\_f\_ls\_null, 89
  - p\_run\_kill, 89
  - p\_run\_mv\_fs, 90
  - p\_run\_rm\_fs, 90
  - p\_run\_touch\_fs, 90
  - p\_search\_and\_remove, 90
  - p\_search\_bg, 90
  - p\_search\_most\_recent, 91
  - p\_search\_most\_recent\_stop, 91
  - p\_setup, 91
  - p\_sleep, 91
  - p\_spawn, 92
  - p\_spawn\_with\_input, 92
  - p\_spawn\_with\_priority, 93
  - p\_waitpid, 93
  - p\_zombie\_child, 93
  - p\_zombify, 94
  - W\_WIFEXITED, 94
  - W\_WIFSIGNALED, 94
  - W\_WIFSTOPPED, 94

Standalone PennFAT

Generated by Doxygen 1.9.1



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 DirectoryEntry Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 firstBlock	5
3.1.2.2 mtime	6
3.1.2.3 name	6
3.1.2.4 perm	6
3.1.2.5 reserved	6
3.1.2.6 size	6
3.1.2.7 type	6
3.2 OpenFileDescriptor Struct Reference	7
3.2.1 Detailed Description	7
3.2.2 Member Data Documentation	7
3.2.2.1 cursor	7
3.2.2.2 entry	8
3.2.2.3 mode	8
3.2.2.4 used	8
3.3 PennFAT Struct Reference	8
3.3.1 Detailed Description	8
3.3.2 Member Data Documentation	8
3.3.2.1 block_size	9
3.3.2.2 fat	9
3.3.2.3 fat_size	9
3.3.2.4 fs_fd	9
3.3.2.5 num_directories	9
<b>4 File Documentation</b>	<b>11</b>
4.1 descriptors.c File Reference	11
4.1.1 Function Documentation	12
4.1.1.1 clear_file_content()	12
4.1.1.2 f_close()	12
4.1.1.3 f_dup2()	12
4.1.1.4 f_ls()	13
4.1.1.5 f_lseek()	13
4.1.1.6 f_open()	14
4.1.1.7 f_read()	14

4.1.1.8 f_unlink()	14
4.1.1.9 f_write()	15
4.1.1.10 find_descriptor_by_name()	15
4.1.1.11 initialize_file_descriptor()	16
4.1.2 Variable Documentation	16
4.1.2.1 curr_stdin	16
4.1.2.2 curr_stdout	16
4.1.2.3 openFileDescriptors	16
4.2 descriptors.h File Reference	17
4.2.1 Macro Definition Documentation	18
4.2.1.1 F_APPEND	18
4.2.1.2 F_READ	18
4.2.1.3 F_SEEK_CUR	18
4.2.1.4 F_SEEK_END	18
4.2.1.5 F_SEEK_SET	19
4.2.1.6 F_WRITE	19
4.2.1.7 MAX_OPEN_FILES	19
4.2.2 Function Documentation	19
4.2.2.1 clear_file_content()	19
4.2.2.2 f_close()	19
4.2.2.3 f_dup2()	20
4.2.2.4 f_ls()	20
4.2.2.5 f_lseek()	20
4.2.2.6 f_open()	21
4.2.2.7 f_read()	21
4.2.2.8 f_unlink()	22
4.2.2.9 f_write()	22
4.2.2.10 find_descriptor_by_name()	23
4.3 directory.c File Reference	23
4.3.1 Function Documentation	24
4.3.1.1 allocate_new_block()	24
4.3.1.2 fetch_block_number()	24
4.3.1.3 find_free_block()	24
4.3.1.4 update_directory_entry()	25
4.3.1.5 update_fat_entry()	26
4.4 directory.h File Reference	26
4.4.1 Macro Definition Documentation	27
4.4.1.1 F_APPEND	27
4.4.1.2 F_READ	28
4.4.1.3 F_WRITE	28
4.4.2 Function Documentation	28
4.4.2.1 allocate_new_block()	28



4.4.2.2 fetch_block_number()	28
4.4.2.3 find_free_block()	29
4.4.2.4 update_directory_entry()	29
4.4.2.5 update_fat_entry()	29
4.5 pennfat.c File Reference	29
4.5.1 Function Documentation	30
4.5.1.1 cat_fs()	31
4.5.1.2 chmod_fs()	31
4.5.1.3 cp_fs()	31
4.5.1.4 find_file()	32
4.5.1.5 ls_fs()	32
4.5.1.6 mkfs()	32
4.5.1.7 mount_fs()	33
4.5.1.8 mv_fs()	33
4.5.1.9 rm_fs()	33
4.5.1.10 touch_fs()	33
4.5.1.11 trim_newline()	35
4.5.1.12 unmount_fs()	35
4.5.2 Variable Documentation	35
4.5.2.1 pf	35
4.6 pennfat.h File Reference	36
4.6.1 Macro Definition Documentation	37
4.6.1.1 CAT_BUFFER_SIZE	37
4.6.2 Typedef Documentation	37
4.6.2.1 DirectoryEntry	37
4.6.2.2 PennFAT	38
4.6.3 Function Documentation	38
4.6.3.1 cat_fs()	38
4.6.3.2 chmod_fs()	38
4.6.3.3 cp_fs()	39
4.6.3.4 find_file()	39
4.6.3.5 ls_fs()	39
4.6.3.6 mkfs()	39
4.6.3.7 mount_fs()	40
4.6.3.8 mv_fs()	40
4.6.3.9 rm_fs()	40
4.6.3.10 touch_fs()	41
4.6.3.11 trim_newline()	41
4.6.3.12 unmount_fs()	41
4.6.4 Variable Documentation	41
4.6.4.1 pf	41
4.7 pennfat_main.c File Reference	42

4.7.1 Function Documentation . . . . .	42
4.7.1.1 main() . . . . .	42
4.8 pennfat_main.h File Reference . . . . .	43
4.8.1 Function Documentation . . . . .	43
4.8.1.1 main() . . . . .	43
<b>Index</b>	<b>45</b>

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">DirectoryEntry</a>	5
<a href="#">OpenFileDescriptor</a>	7
<a href="#">PennFAT</a>	8



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">descriptors.c</a>	11
<a href="#">descriptors.h</a>	17
<a href="#">directory.c</a>	23
<a href="#">directory.h</a>	26
<a href="#">pennfat.c</a>	29
<a href="#">pennfat.h</a>	36
<a href="#">pennfat_main.c</a>	42
<a href="#">pennfat_main.h</a>	43



## Chapter 3

# Class Documentation

### 3.1 DirectoryEntry Struct Reference

```
#include <pennfat.h>
```

#### Public Attributes

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

#### 3.1.1 Detailed Description

The [DirectoryEntry](#) struct we use to store the information for each file.

#### 3.1.2 Member Data Documentation

##### 3.1.2.1 firstBlock

```
uint16_t DirectoryEntry::firstBlock
```

index of the first block of the file, 0 if no memory allocated

### 3.1.2.2 mtime

```
time_t DirectoryEntry::mtime
```

time of last update for this file

### 3.1.2.3 name

```
char DirectoryEntry::name[32]
```

filename

### 3.1.2.4 perm

```
uint8_t DirectoryEntry::perm
```

permission of file: 0, 2, 4, 5, 6, 7

### 3.1.2.5 reserved

```
char DirectoryEntry::reserved[16]
```

reserved bits for this file, unused

### 3.1.2.6 size

```
uint32_t DirectoryEntry::size
```

size of bytes written in the file

### 3.1.2.7 type

```
uint8_t DirectoryEntry::type
```

type of file: 0, 1, 2, 4

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

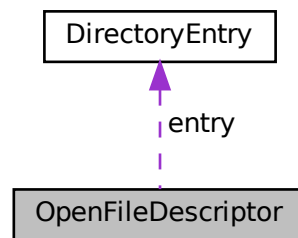
- [pennfat.h](#)



## 3.2 OpenFileDescriptor Struct Reference

```
#include <descriptors.h>
```

Collaboration diagram for OpenFileDescriptor:



### Public Attributes

- int [used](#)  
*Flag indicating if this descriptor is in use.*
- [DirectoryEntry](#) \* [entry](#)  
*Pointer to the associated directory entry.*
- int [mode](#)  
*Mode in which the file was opened.*
- unsigned int [cursor](#)  
*Current position in the file.*

### 3.2.1 Detailed Description

Structure representing an open file descriptor.

### 3.2.2 Member Data Documentation

#### 3.2.2.1 cursor

```
unsigned int OpenFileDescriptor::cursor
```

Current position in the file.

### 3.2.2.2 entry

`DirectoryEntry* OpenFileDescriptor::entry`

Pointer to the associated directory entry.

### 3.2.2.3 mode

`int OpenFileDescriptor::mode`

Mode in which the file was opened.

### 3.2.2.4 used

`int OpenFileDescriptor::used`

Flag indicating if this descriptor is in use.

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

- [descriptors.h](#)

## 3.3 PennFAT Struct Reference

```
#include <pennfat.h>
```

### Public Attributes

- `uint16_t * fat`
- `size_t fat_size`
- `size_t block_size`
- `int fs_fd`
- `int num_directories`

### 3.3.1 Detailed Description

The [PennFAT](#) struct we use to store metadata for the file system, and a pointer to the file allocation table (FAT)

### 3.3.2 Member Data Documentation

### 3.3.2.1 block\_size

```
size_t PennFAT::block_size
```

size of one data block

### 3.3.2.2 fat

```
uint16_t* PennFAT::fat
```

pointer to the file allocation table (fat)

### 3.3.2.3 fat\_size

```
size_t PennFAT::fat_size
```

size of the fat entry

### 3.3.2.4 fs\_fd

```
int PennFAT::fs_fd
```

file descriptor of the file system

### 3.3.2.5 num\_directories

```
int PennFAT::num_directories
```

number of directories in the file system, unused

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

- [pennfat.h](#)



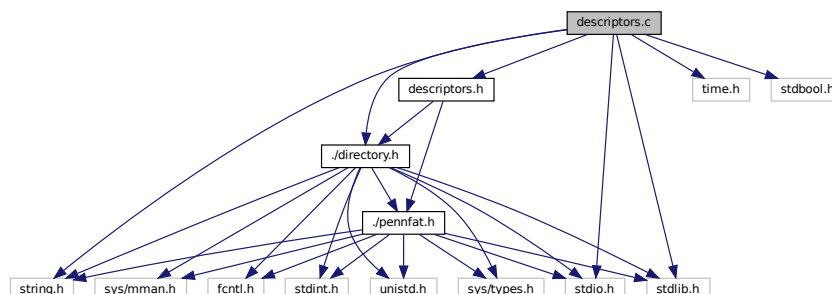
## Chapter 4

# File Documentation

### 4.1 descriptors.c File Reference

```
#include "descriptors.h"
#include "directory.h"
#include <time.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for descriptors.c:



### Functions

- int [initialize\\_file\\_descriptor](#) ([DirectoryEntry](#) \*entry, int mode)
- int [f\\_open](#) (const char \*fname, int mode)
- int [f\\_read](#) (int fd, int n, char \*buf)
- int [f\\_write](#) (int fd, const char \*str, int n)
- void [clear\\_file\\_content](#) ([DirectoryEntry](#) \*entry)
- int [f\\_close](#) (int fd)
- int [f\\_unlink](#) (const char \*fname)
- int [f\\_lseek](#) (int fd, int offset, int whence)
- void [f\\_ls](#) (const char \*filename)
- int [f\\_dup2](#) (int fd\_curr, int fd\_new)
- int [find\\_descriptor\\_by\\_name](#) (const char \*fname)

*Finds the file descriptor for a given file name.*

## Variables

- int `curr_stdin`
- int `curr_stdout`
- `OpenFileDescriptor` `openFileDescriptors` [`MAX_OPEN_FILES`]

## 4.1.1 Function Documentation

### 4.1.1.1 `clear_file_content()`

```
void clear_file_content (
    DirectoryEntry * entry )
```

Clears the content of a file associated with a directory entry. Used in `F_WRITE` mode to overwrite the current directory entry.

#### Parameters

<i>entry</i>	Directory entry of the file to clear.
--------------	---------------------------------------

### 4.1.1.2 `f_close()`

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

Closes an open file descriptor.

#### Parameters

<i>fd</i>	File descriptor to close.
-----------	---------------------------

#### Returns

0 on success, -1 on error.

### 4.1.1.3 `f_dup2()`

```
int f_dup2 (
    int fd_curr,
    int fd_new )
```

Duplicates a file descriptor.

## Parameters

<i>fd_curr</i>	Current file descriptor.
<i>fd_new</i>	New file descriptor.

## Returns

New file descriptor on success, -1 on error.

**4.1.1.4 f\_ls()**

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

Lists files in a directory.

## Parameters

<i>filename</i>	Name of the file or directory to list.
-----------------	--

**4.1.1.5 f\_lseek()**

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

Sets the file cursor of an open file descriptor.

Modes: F\_SEEK\_SET - The file offset is set to offset bytes. F\_SEEK\_CUR - The file offset is set to its current location plus offset bytes. F\_SEEK\_END - The file offset is set to the size of the file plus offset bytes.

## Parameters

<i>fd</i>	File descriptor to seek.
<i>offset</i>	Offset to set the cursor.
<i>whence</i>	Position from where offset is applied.

## Returns

New cursor position on success, -1 on error.

#### 4.1.1.6 `f_open()`

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

Opens a file named *fname* in the specified mode.

Modes: `F_WRITE` - Opens for writing and reading; truncates if exists, creates if not. Only one instance can be open in `F_WRITE` mode. `F_READ` - Opens for reading only; errors if file does not exist. `F_APPEND` - Opens for reading and writing; does not truncate, sets pointer at end.

##### Parameters

<i>fname</i>	Name of the file to open. Filename should follow POSIX standards.
<i>mode</i>	Mode to open the file.

##### Returns

File descriptor on success, negative value on error.

#### 4.1.1.7 `f_read()`

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

Reads *n* bytes from the file referenced by *fd* into *buf*.

##### Parameters

<i>fd</i>	File descriptor to read from.
<i>n</i>	Number of bytes to read.
<i>buf</i>	Buffer to store the read data.

##### Returns

Number of bytes read, 0 if EOF, negative number on error.

#### 4.1.1.8 `f_unlink()`

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

Deletes a file with the specified name.



## Parameters

<i>fname</i>	Name of the file to delete.
--------------	-----------------------------

## Returns

0 on success, -1 on error.

**4.1.1.9 f\_write()**

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

Writes *n* bytes from the string *str* to the file referenced by *fd*.

## Parameters

<i>fd</i>	File descriptor to write to.
<i>str</i>	String containing the data to write.
<i>n</i>	Number of bytes to write.

## Returns

Number of bytes written, negative value on error.

**4.1.1.10 find\_descriptor\_by\_name()**

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

Finds the file descriptor for a given file name.

This function searches through the open file descriptors to find one that is associated with the specified file name. If a matching file descriptor is found, it is returned. If the file is not currently open or if the file name is not found, the function returns -1.

## Parameters

<i>fname</i>	The name of the file for which the file descriptor is sought. It is a null-terminated string. The function performs a case-sensitive search for this file name.
--------------	---

**Returns**

int Returns the file descriptor (an integer) if a matching file is found. If the file is not open or the file name is not found, the function returns -1.

**4.1.1.11 initialize\_file\_descriptor()**

```
int initialize_file_descriptor (
    DirectoryEntry * entry,
    int mode )
```

**4.1.2 Variable Documentation****4.1.2.1 curr\_stdin**

```
int curr_stdin
```

**4.1.2.2 curr\_stdout**

```
int curr_stdout
```

**4.1.2.3 openFileDescriptors**

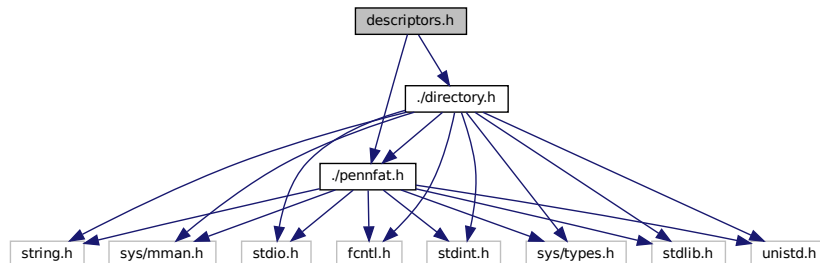
```
OpenFileDescriptor openFileDescriptors[MAX_OPEN_FILES]
```

## 4.2 descriptors.h File Reference

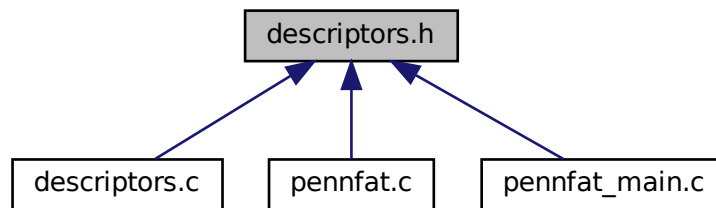
```
#include "../directory.h"
```

```
#include "../pennfat.h"
```

Include dependency graph for descriptors.h:



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



### Classes

- struct [OpenFileDescriptor](#)

### Macros

- #define [MAX\\_OPEN\\_FILES](#) 100
- #define [F\\_WRITE](#) 1  
*Writing mode, truncates file if exists, creates if not. Only one instance can be open.*
- #define [F\\_READ](#) 2  
*Read-only mode, error if file does not exist.*
- #define [F\\_APPEND](#) 3  
*Append mode, for reading and writing. Does not truncate, file pointer at end.*
- #define [F\\_SEEK\\_SET](#) 0  
*Seek from the beginning of the file.*
- #define [F\\_SEEK\\_CUR](#) 1  
*Seek from the current file position.*
- #define [F\\_SEEK\\_END](#) 2  
*Seek from the end of the file.*

## Functions

- int [f\\_open](#) (const char \*fname, int mode)
- int [f\\_read](#) (int fd, int n, char \*buf)
- int [f\\_write](#) (int fd, const char \*str, int n)
- void [clear\\_file\\_content](#) ([DirectoryEntry](#) \*entry)
- int [f\\_unlink](#) (const char \*fname)
- int [f\\_lseek](#) (int fd, int offset, int whence)
- int [f\\_close](#) (int fd)
- void [f\\_ls](#) (const char \*filename)
- int [f\\_dup2](#) (int fd\_curr, int fd\_new)
- int [find\\_descriptor\\_by\\_name](#) (const char \*fname)

*Finds the file descriptor for a given file name.*

## 4.2.1 Macro Definition Documentation

### 4.2.1.1 F\_APPEND

```
#define F_APPEND 3
```

Append mode, for reading and writing. Does not truncate, file pointer at end.

### 4.2.1.2 F\_READ

```
#define F_READ 2
```

Read-only mode, error if file does not exist.

### 4.2.1.3 F\_SEEK\_CUR

```
#define F_SEEK_CUR 1
```

Seek from the current file position.

### 4.2.1.4 F\_SEEK\_END

```
#define F_SEEK_END 2
```

Seek from the end of the file.

#### 4.2.1.5 F\_SEEK\_SET

```
#define F_SEEK_SET 0
```

Seek from the beginning of the file.

#### 4.2.1.6 F\_WRITE

```
#define F_WRITE 1
```

Writing mode, truncates file if exists, creates if not. Only one instance can be open.

#### 4.2.1.7 MAX\_OPEN\_FILES

```
#define MAX_OPEN_FILES 100
```

### 4.2.2 Function Documentation

#### 4.2.2.1 clear\_file\_content()

```
void clear_file_content (
    DirectoryEntry * entry )
```

Clears the content of a file associated with a directory entry. Used in F\_WRITE mode to overwrite the current directory entry.

##### Parameters

<i>entry</i>	Directory entry of the file to clear.
--------------	---------------------------------------

#### 4.2.2.2 f\_close()

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

Closes an open file descriptor.

**Parameters**

<i>fd</i>	File descriptor to close.
-----------	---------------------------

**Returns**

0 on success, -1 on error.

**4.2.2.3 f\_dup2()**

```
int f_dup2 (
    int fd_curr,
    int fd_new )
```

Duplicates a file descriptor.

**Parameters**

<i>fd_curr</i>	Current file descriptor.
<i>fd_new</i>	New file descriptor.

**Returns**

New file descriptor on success, -1 on error.

**4.2.2.4 f\_ls()**

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

Lists files in a directory.

**Parameters**

<i>filename</i>	Name of the file or directory to list.
-----------------	--

**4.2.2.5 f\_lseek()**

```
int f_lseek (
    int fd,
```

```
int offset,  
int whence )
```

Sets the file cursor of an open file descriptor.

Modes: `F SEEK SET` - The file offset is set to offset bytes. `F SEEK CUR` - The file offset is set to its current location plus offset bytes. `F SEEK END` - The file offset is set to the size of the file plus offset bytes.

#### Parameters

<i>fd</i>	File descriptor to seek.
<i>offset</i>	Offset to set the cursor.
<i>whence</i>	Position from where offset is applied.

#### Returns

New cursor position on success, -1 on error.

### 4.2.2.6 `f_open()`

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

Opens a file named *fname* in the specified mode.

Modes: `F WRITE` - Opens for writing and reading; truncates if exists, creates if not. Only one instance can be open in `F WRITE` mode. `F READ` - Opens for reading only; errors if file does not exist. `F APPEND` - Opens for reading and writing; does not truncate, sets pointer at end.

#### Parameters

<i>fname</i>	Name of the file to open. Filename should follow POSIX standards.
<i>mode</i>	Mode to open the file.

#### Returns

File descriptor on success, negative value on error.

### 4.2.2.7 `f_read()`

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

Reads *n* bytes from the file referenced by *fd* into *buf*.

**Parameters**

<i>fd</i>	File descriptor to read from.
<i>n</i>	Number of bytes to read.
<i>buf</i>	Buffer to store the read data.

**Returns**

Number of bytes read, 0 if EOF, negative number on error.

**4.2.2.8 f\_unlink()**

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

Deletes a file with the specified name.

**Parameters**

<i>fname</i>	Name of the file to delete.
--------------	-----------------------------

**Returns**

0 on success, -1 on error.

**4.2.2.9 f\_write()**

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

Writes *n* bytes from the string *str* to the file referenced by *fd*.

**Parameters**

<i>fd</i>	File descriptor to write to.
<i>str</i>	String containing the data to write.
<i>n</i>	Number of bytes to write.

**Returns**

Number of bytes written, negative value on error.



## 4.2.2.10 find\_descriptor\_by\_name()

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

Finds the file descriptor for a given file name.

This function searches through the open file descriptors to find one that is associated with the specified file name. If a matching file descriptor is found, it is returned. If the file is not currently open or if the file name is not found, the function returns -1.

## Parameters

<i>fname</i>	The name of the file for which the file descriptor is sought. It is a null-terminated string. The function performs a case-sensitive search for this file name.
--------------	---

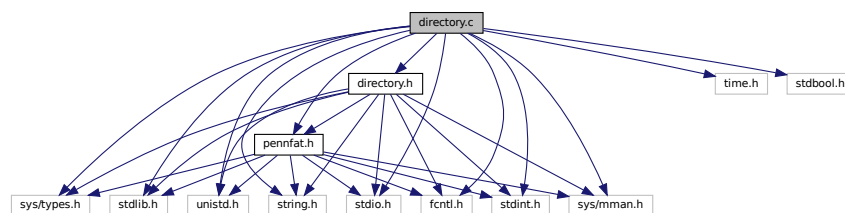
## Returns

int Returns the file descriptor (an integer) if a matching file is found. If the file is not open or the file name is not found, the function returns -1.

## 4.3 directory.c File Reference

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

Include dependency graph for directory.c:



## Functions

- unsigned int [find\\_free\\_block](#) ()
- unsigned int [fetch\\_block\\_number](#) (uint16\_t start\_block, unsigned int block\_offset)
- unsigned int [allocate\\_new\\_block](#) ()
- void [update\\_fat\\_entry](#) (int fs\_fd, uint16\_t current\_block, uint16\_t new\_block)
- void [update\\_directory\\_entry](#) (int fs\_fd, [DirectoryEntry](#) \*entry)

### 4.3.1 Function Documentation

#### 4.3.1.1 `allocate_new_block()`

```
unsigned int allocate_new_block ( )
```

Allocates a new block in the file system.

##### Returns

The block number of the newly allocated block, or 0 if no block is available.

#### 4.3.1.2 `fetch_block_number()`

```
unsigned int fetch_block_number (
    uint16_t start_block,
    unsigned int block_offset )
```

Fetches the block number at a given offset from the start block.

##### Parameters

<i>start_block</i>	The starting block number.
<i>block_offset</i>	The block offset from the start block.

##### Returns

The block number at the offset, or 0xFFFF if the end of the file is reached or an error occurs.

#### 4.3.1.3 `find_free_block()`

```
unsigned int find_free_block ( )
```

Finds a free block in the file system.

##### Returns

The index of the free block, or -1 if no block is available or an error occurs.

#### 4.3.1.4 update\_directory\_entry()

```
void update_directory_entry (  
    int fs_fd,  
    DirectoryEntry * entry )
```

Updates a directory entry in the file system.

## Parameters

<i>fs_fd</i>	File descriptor for the file system.
<i>_fd</i>	
<i>entry</i>	Pointer to the directory entry to update.

## 4.3.1.5 update\_fat\_entry()

```
void update_fat_entry (
    int fs_fd,
    uint16_t current_block,
    uint16_t new_block )
```

Updates a FAT entry in the file system.

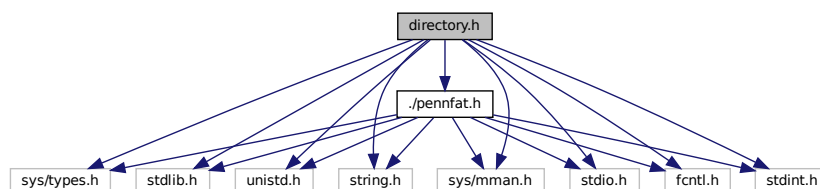
## Parameters

<i>fs_fd</i>	File descriptor for the file system.
<i>current_block</i>	The current block to be updated.
<i>new_block</i>	The new block number to update to.

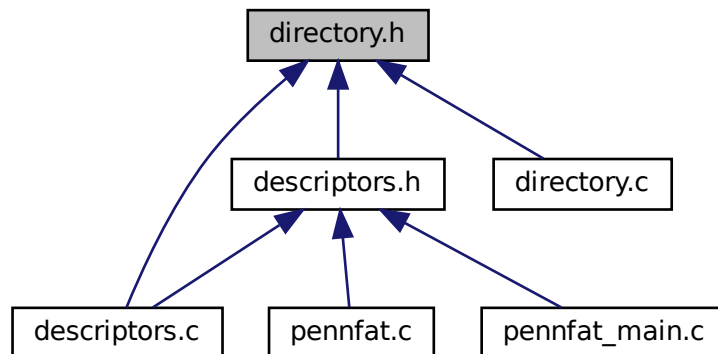
## 4.4 directory.h File Reference

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

Include dependency graph for directory.h:



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



## Macros

- `#define F_WRITE 1`  
*Writing mode.*
- `#define F_READ 2`  
*Read-only mode.*
- `#define F_APPEND 3`  
*Append mode.*

## Functions

- unsigned int `find_free_block` ()
- unsigned int `fetch_block_number` (uint16\_t start\_block, unsigned int block\_offset)
- unsigned int `allocate_new_block` ()
- void `update_fat_entry` (int fs\_fd, uint16\_t current\_block, uint16\_t new\_block)
- void `update_directory_entry` (int fs\_fd, `DirectoryEntry` \*entry)

### 4.4.1 Macro Definition Documentation

#### 4.4.1.1 F\_APPEND

```
#define F_APPEND 3
```

Append mode.

#### 4.4.1.2 F\_READ

```
#define F_READ 2
```

Read-only mode.

#### 4.4.1.3 F\_WRITE

```
#define F_WRITE 1
```

Writing mode.

### 4.4.2 Function Documentation

#### 4.4.2.1 allocate\_new\_block()

```
unsigned int allocate_new_block ( )
```

Allocates a new block in the file system.

##### Returns

The block number of the newly allocated block, or 0 if no block is available.

#### 4.4.2.2 fetch\_block\_number()

```
unsigned int fetch_block_number (
    uint16_t start_block,
    unsigned int block_offset )
```

Fetches the block number at a given offset from the start block.

##### Parameters

<i>start_block</i>	The starting block number.
<i>block_offset</i>	The block offset from the start block.

##### Returns

The block number at the offset, or 0xFFFF if the end of the file is reached or an error occurs.

#### 4.4.2.3 find\_free\_block()

```
unsigned int find_free_block ( )
```

Finds a free block in the file system.

##### Returns

The index of the free block, or -1 if no block is available or an error occurs.

#### 4.4.2.4 update\_directory\_entry()

```
void update_directory_entry (
    int fs_fd,
    DirectoryEntry * entry )
```

Updates a directory entry in the file system.

##### Parameters

<i>fs_fd</i>	File descriptor for the file system.
<i>entry</i>	Pointer to the directory entry to update.

#### 4.4.2.5 update\_fat\_entry()

```
void update_fat_entry (
    int fs_fd,
    uint16_t current_block,
    uint16_t new_block )
```

Updates a FAT entry in the file system.

##### Parameters

<i>fs_fd</i>	File descriptor for the file system.
<i>current_block</i>	The current block to be updated.
<i>new_block</i>	The new block number to update to.

## 4.5 pennfat.c File Reference

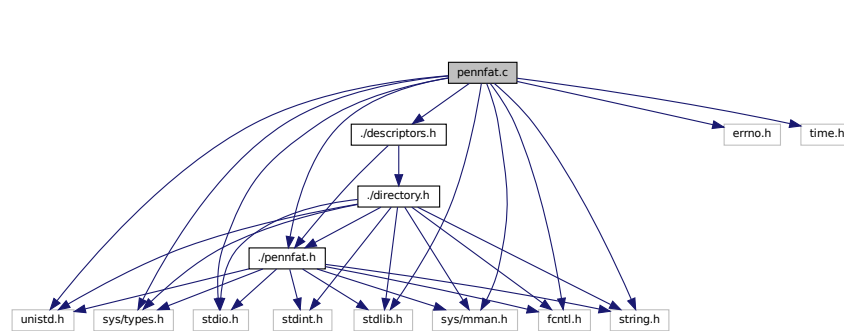
```
#include <sys/types.h>
#include <stdlib.h>
```

```

#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <time.h>
#include "../pennfat.h"
#include "../descriptors.h"

```

Include dependency graph for pennfat.c:



## Functions

- void [mkfs](#) (char \*fs\_name, int blocks\_in\_fat, int block\_size\_config)
- void [mount\\_fs](#) (char \*fs\_name)
- void [unmount\\_fs](#) ()
- uint16\_t [touch\\_fs](#) (char \*filename)
- void [ls\\_fs](#) ()
- [DirectoryEntry](#) \* [find\\_file](#) (int \*directory\_block\_offset, int \*directory\_entry\_offset, const char \*filename)
- void [mv\\_fs](#) (const char \*source, const char \*dest)
- void [trim\\_newline](#) (char \*str)
- void [cp\\_fs](#) (char \*source, char \*dest, int from\_host)
- void [rm\\_fs](#) (const char \*filename)
- void [cat\\_fs](#) (int argc, char \*\*argv)
- void [chmod\\_fs](#) (char \*filename, char \*perm)

## Variables

- struct [PennFAT](#) \* [pf](#) = NULL

### 4.5.1 Function Documentation



#### 4.5.1.1 cat\_fs()

```
void cat_fs (
    int argc,
    char ** argv )
```

Reads in the `cat` command from the command line, parse `argv` according to the different formats of `cat`. Specifically, a `cat` command can:

1. `cat FILE... -w/a OUTPUT_FILE`: concatenates the content in `FILE` and write/append it to `OUTPUT_FILE`.
2. `cat -w/a OUTPUT_FILE`: reads content from stdin and writes/appends it to `OUTPUT_FILE`.
3. `cat FILE...`: concatenates all content in `FILE` and write to stdout.

##### Parameters

<i>argc</i>	number of parameters read from the <code>cat</code> command
<i>argv</i>	array of parameters

#### 4.5.1.2 chmod\_fs()

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

Changes the permission for the file named `filename` according to the specifications in `perm` string.

##### Parameters

<i>filename</i>	name of the file whose permission we are changing
<i>perm</i>	string that specifies modification for permission

#### 4.5.1.3 cp\_fs()

```
void cp_fs (
    char * source,
    char * dest,
    int from_host )
```

Copies the content from a file named `source` to the file named `dest`. If `source` doesn't exist, an error is given. If `dest` doesn't exist, creates the new file named `dest`.

## Parameters

<i>source</i>	name of source file
<i>dest</i>	name of dest file
<i>from_host</i>	0: both files are in directory, 1: source is from host, 2: dest is from host

**4.5.1.4 find\_file()**

```
DirectoryEntry* find_file (
    int * directory_block_offset,
    int * directory_entry_offset,
    const char * filename )
```

Iterates through all the directory blocks and attempts to find the file named filename. If found, `directory_block_offset` is the index of the block that this directory entry is in, and `directory_entry_offset` is the number of entry this specific file is in this block.

## Parameters

<i>directory_block_offset</i>	index of data block that stores this directory entry
<i>directory_entry_offset</i>	index of directory entry within this data block
<i>filename</i>	name of file we are attempting to find.

**4.5.1.5 ls\_fs()**

```
void ls_fs ( )
```

Lists all files in the directory.

**4.5.1.6 mkfs()**

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

Creates a [PennFAT](#) filesystem in the file named FS\_NAME. The number of blocks in the FAT region is BLOCKS\_IN\_FAT (ranging from 1 through 32), and the block size is 256, 512, 1024, 2048, or 4096 bytes corresponding to the value (0 through 4) of BLOCK\_SIZE\_CONFIG fed into BLOCK\_SIZES[].

## Parameters

<i>fs_name</i>	filename
<i>blocks_in_fat</i>	number of blocks in the FAT region (1-32)
<i>block_size_config</i>	a number 0-4 that corresponds to block size

#### 4.5.1.7 mount\_fs()

```
void mount_fs (
    char * fs_name )
```

Mounts the filesystem named FS\_NAME by loading the FAT into memory.

##### Parameters

<i>fs_name</i>	name of the file we are mounting
----------------	----------------------------------

#### 4.5.1.8 mv\_fs()

```
void mv_fs (
    const char * source,
    const char * dest )
```

Renames the file named source to dest

##### Parameters

<i>source</i>	name of the file wanting to rename
<i>dest</i>	name of file renamed to

#### 4.5.1.9 rm\_fs()

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

Removes the file named filename from the directory

##### Parameters

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

#### 4.5.1.10 touch\_fs()

```
uint16_t touch_fs (
    char * filename )
```

Creates a [DirectoryEntry](#) for a new file called filename and stores the [DirectoryEntry](#) in a directory block.

## Parameters

<i>filename</i>	name of the file we want to create/touch
-----------------	--

**4.5.1.11 trim\_newline()**

```
void trim_newline (
    char * str )
```

Helper function to trim the last '  
' character from *str*.

## Parameters

<i>str</i>	string to trim ' ' from
------------	----------------------------

**4.5.1.12 unmount\_fs()**

```
void unmount_fs ( )
```

Unmount the current filesystem in memory.

**4.5.2 Variable Documentation****4.5.2.1 pf**

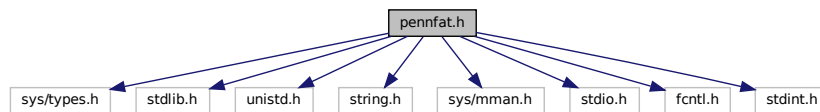
```
struct PennFAT* pf = NULL
```

Pointer to the [PennFAT](#) struct *pf*, which is accessible all throughout the program after the file system is mounted.

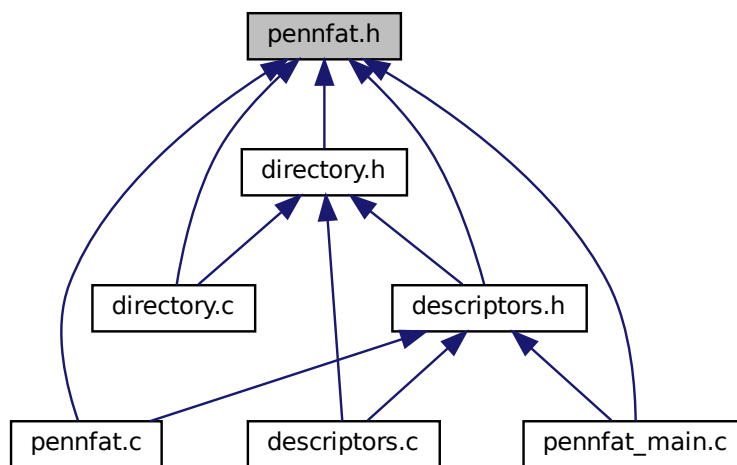
## 4.6 pennfat.h File Reference

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

Include dependency graph for pennfat.h:



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



### Classes

- struct [DirectoryEntry](#)
- struct [PennFAT](#)

### Macros

- #define [CAT\\_BUFFER\\_SIZE](#) 4096

## Typedefs

- typedef struct [DirectoryEntry](#) [DirectoryEntry](#)
- typedef struct [PennFAT](#) [PennFAT](#)

## Functions

- void [mkfs](#) (char \*fs\_name, int blocks\_in\_fat, int block\_size\_config)
- void [mount\\_fs](#) (char \*fs\_name)
- void [unmount\\_fs](#) ()
- uint16\_t [touch\\_fs](#) (char \*filename)
- void [cat\\_fs](#) (int argc, char \*\*argv)
- void [ls\\_fs](#) ()
- void [mv\\_fs](#) (const char \*source, const char \*dest)
- void [rm\\_fs](#) (const char \*filename)
- void [trim\\_newline](#) (char \*str)
- void [cp\\_fs](#) (char \*source, char \*dest, int from\_host)
- void [chmod\\_fs](#) (char \*filename, char \*perm)
- [DirectoryEntry](#) \* [find\\_file](#) (int \*directory\_block\_offset, int \*directory\_entry\_offset, const char \*filename)

## Variables

- struct [PennFAT](#) \* [pf](#)

### 4.6.1 Macro Definition Documentation

#### 4.6.1.1 CAT\_BUFFER\_SIZE

```
#define CAT_BUFFER_SIZE 4096
```

buffer for writing

### 4.6.2 Typedef Documentation

#### 4.6.2.1 DirectoryEntry

```
typedef struct DirectoryEntry DirectoryEntry
```

The [DirectoryEntry](#) struct we use to store the information for each file.

#### 4.6.2.2 PennFAT

```
typedef struct PennFAT PennFAT
```

The `PennFAT` struct we use to store metadata for the file system, and a pointer to the file allocation table (FAT)

### 4.6.3 Function Documentation

#### 4.6.3.1 `cat_fs()`

```
void cat_fs (
    int argc,
    char ** argv )
```

Reads in the `cat` command from the command line, parse `argv` according to the different formats of `cat`. Specifically, a `cat` command can:

1. `cat FILE... -w/a OUTPUT_FILE`: concatenates the content in `FILE` and write/append it to `OUTPUT_FILE`.
2. `cat -w/a OUTPUT_FILE`: reads content from `stdin` and writes/appends it to `OUTPUT_FILE`.
3. `cat FILE...`: concatenates all content in `FILE` and write to `stdout`.

##### Parameters

<i>argc</i>	number of parameters read from the <code>cat</code> command
<i>argv</i>	array of parameters

#### 4.6.3.2 `chmod_fs()`

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

Changes the permission for the file named `filename` according to the specifications in `perm` string.

##### Parameters

<i>filename</i>	name of the file whose permission we are changing
<i>perm</i>	string that specifies modification for permission



#### 4.6.3.3 cp\_fs()

```
void cp_fs (
    char * source,
    char * dest,
    int from_host )
```

Copies the content from a file named *source* to the file named *dest*. If *source* doesn't exist, an error is given. If *dest* doesn't exist, creates the new file named *dest*.

##### Parameters

<i>source</i>	name of source file
<i>dest</i>	name of dest file
<i>from_host</i>	0: both files are in directory, 1: source is from host, 2: dest is from host

#### 4.6.3.4 find\_file()

```
DirectoryEntry* find_file (
    int * directory_block_offset,
    int * directory_entry_offset,
    const char * filename )
```

Iterates through all the directory blocks and attempts to find the file named *filename*. If found, *directory\_block\_offset* is the index of the block that this directory entry is in, and *directory\_entry\_offset* is the number of entry this specific file is in this block.

##### Parameters

<i>directory_block_offset</i>	index of data block that stores this directory entry
<i>directory_entry_offset</i>	index of directory entry within this data block
<i>filename</i>	name of file we are attempting to find.

#### 4.6.3.5 ls\_fs()

```
void ls_fs ( )
```

Lists all files in the directory.

#### 4.6.3.6 mkfs()

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

Creates a [PennFAT](#) filesystem in the file named *FS\_NAME*. The number of blocks in the FAT region is *BLOCKS\_IN\_FAT* (ranging from 1 through 32), and the block size is 256, 512, 1024, 2048, or 4096 bytes corresponding to the value (0 through 4) of *BLOCK\_SIZE\_CONFIG* fed into *BLOCK\_SIZES[]*.

## Parameters

<i>fs_name</i>	filename
<i>blocks_in_fat</i>	number of blocks in the FAT region (1-32)
<i>block_size_config</i>	a number 0-4 that corresponds to block size

**4.6.3.7 mount\_fs()**

```
void mount_fs (
    char * fs_name )
```

Mounts the filesystem named FS\_NAME by loading the FAT into memory.

## Parameters

<i>fs_name</i>	name of the file we are mounting
----------------	----------------------------------

**4.6.3.8 mv\_fs()**

```
void mv_fs (
    const char * source,
    const char * dest )
```

Renames the file named source to dest

## Parameters

<i>source</i>	name of the file wanting to rename
<i>dest</i>	name of file renamed to

**4.6.3.9 rm\_fs()**

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

Removes the file named filename from the directory

## Parameters

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

#### 4.6.3.10 touch\_fs()

```
uint16_t touch_fs (
    char * filename )
```

Creates a [DirectoryEntry](#) for a new file called filename and stores the [DirectoryEntry](#) in a directory block.

##### Parameters

<i>filename</i>	name of the file we want to create/touch
-----------------	--

#### 4.6.3.11 trim\_newline()

```
void trim_newline (
    char * str )
```

Helper function to trim the last '  
' character from str.

##### Parameters

<i>str</i>	string to trim ' ' from
------------	----------------------------

#### 4.6.3.12 unmount\_fs()

```
void unmount_fs ( )
```

Unmount the current filesystem in memory.

### 4.6.4 Variable Documentation

#### 4.6.4.1 pf

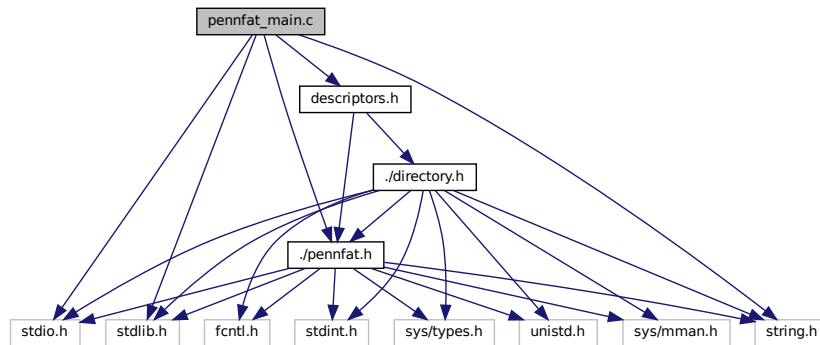
```
struct PennFAT* pf [extern]
```

Pointer to the [PennFAT](#) struct pf, which is accessible all throughout the program after the file system is mounted.

## 4.7 pennfat\_main.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "../pennfat.h"
#include "descriptors.h"
```

Include dependency graph for pennfat\_main.c:



### Functions

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

### 4.7.1 Function Documentation

#### 4.7.1.1 main()

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

The main function of our standalone PennFaT, used to initiate everything and allow PennFat to execute the commands required.

#### Parameters

<i>argc</i>	The number of arguments passed in from the terminal.
<i>argv</i>	The arguments from terminal.

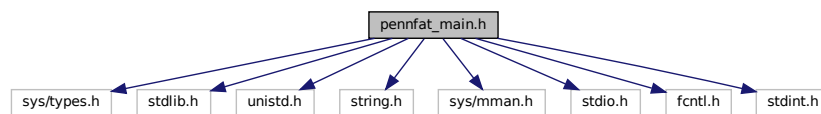
#### Returns

Anything on exit.

## 4.8 pennfat\_main.h File Reference

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

Include dependency graph for pennfat\_main.h:



## Functions

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

### 4.8.1 Function Documentation

#### 4.8.1.1 main()

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

The main function of our standalone PennFaT, used to initiate everything and allow PennFat to execute the commands required.

#### Parameters

<i>argc</i>	The number of arguments passed in from the terminal.
<i>argv</i>	The arguments from terminal.

#### Returns

Anything on exit.



# Index

- allocate\_new\_block
  - directory.c, [24](#)
  - directory.h, [28](#)
- block\_size
  - PennFAT, [8](#)
- CAT\_BUFFER\_SIZE
  - pennfat.h, [37](#)
- cat\_fs
  - pennfat.c, [30](#)
  - pennfat.h, [38](#)
- chmod\_fs
  - pennfat.c, [31](#)
  - pennfat.h, [38](#)
- clear\_file\_content
  - descriptors.c, [12](#)
  - descriptors.h, [19](#)
- cp\_fs
  - pennfat.c, [31](#)
  - pennfat.h, [38](#)
- curr\_stdin
  - descriptors.c, [16](#)
- curr\_stdout
  - descriptors.c, [16](#)
- cursor
  - OpenFileDescriptor, [7](#)
- descriptors.c, [11](#)
  - clear\_file\_content, [12](#)
  - curr\_stdin, [16](#)
  - curr\_stdout, [16](#)
  - f\_close, [12](#)
  - f\_dup2, [12](#)
  - f\_ls, [13](#)
  - f\_lseek, [13](#)
  - f\_open, [13](#)
  - f\_read, [14](#)
  - f\_unlink, [14](#)
  - f\_write, [15](#)
  - find\_descriptor\_by\_name, [15](#)
  - initialize\_file\_descriptor, [16](#)
  - openFileDescriptors, [16](#)
- descriptors.h, [17](#)
  - clear\_file\_content, [19](#)
  - F\_APPEND, [18](#)
  - f\_close, [19](#)
  - f\_dup2, [20](#)
  - f\_ls, [20](#)
  - f\_lseek, [20](#)
  - f\_open, [21](#)
  - F\_READ, [18](#)
  - f\_read, [21](#)
  - F\_SEEK\_CUR, [18](#)
  - F\_SEEK\_END, [18](#)
  - F\_SEEK\_SET, [18](#)
  - f\_unlink, [22](#)
  - F\_WRITE, [19](#)
  - f\_write, [22](#)
  - find\_descriptor\_by\_name, [22](#)
  - MAX\_OPEN\_FILES, [19](#)
- directory.c, [23](#)
  - allocate\_new\_block, [24](#)
  - fetch\_block\_number, [24](#)
  - find\_free\_block, [24](#)
  - update\_directory\_entry, [24](#)
  - update\_fat\_entry, [26](#)
- directory.h, [26](#)
  - allocate\_new\_block, [28](#)
  - F\_APPEND, [27](#)
  - F\_READ, [27](#)
  - F\_WRITE, [28](#)
  - fetch\_block\_number, [28](#)
  - find\_free\_block, [28](#)
  - update\_directory\_entry, [29](#)
  - update\_fat\_entry, [29](#)
- DirectoryEntry, [5](#)
  - firstBlock, [5](#)
  - mtime, [5](#)
  - name, [6](#)
  - pennfat.h, [37](#)
  - perm, [6](#)
  - reserved, [6](#)
  - size, [6](#)
  - type, [6](#)
- entry
  - OpenFileDescriptor, [7](#)
- F\_APPEND
  - descriptors.h, [18](#)
  - directory.h, [27](#)
- f\_close
  - descriptors.c, [12](#)
  - descriptors.h, [19](#)
- f\_dup2
  - descriptors.c, [12](#)
  - descriptors.h, [20](#)
- f\_ls
  - descriptors.c, [13](#)

- descriptors.h, 20
- f\_lseek
  - descriptors.c, 13
  - descriptors.h, 20
- f\_open
  - descriptors.c, 13
  - descriptors.h, 21
- F\_READ
  - descriptors.h, 18
  - directory.h, 27
- f\_read
  - descriptors.c, 14
  - descriptors.h, 21
- F\_SEEK\_CUR
  - descriptors.h, 18
- F\_SEEK\_END
  - descriptors.h, 18
- F\_SEEK\_SET
  - descriptors.h, 18
- f\_unlink
  - descriptors.c, 14
  - descriptors.h, 22
- F\_WRITE
  - descriptors.h, 19
  - directory.h, 28
- f\_write
  - descriptors.c, 15
  - descriptors.h, 22
- fat
  - PennFAT, 9
- fat\_size
  - PennFAT, 9
- fetch\_block\_number
  - directory.c, 24
  - directory.h, 28
- find\_descriptor\_by\_name
  - descriptors.c, 15
  - descriptors.h, 22
- find\_file
  - pennfat.c, 32
  - pennfat.h, 39
- find\_free\_block
  - directory.c, 24
  - directory.h, 28
- firstBlock
  - DirectoryEntry, 5
- fs\_fd
  - PennFAT, 9
- initialize\_file\_descriptor
  - descriptors.c, 16
- ls\_fs
  - pennfat.c, 32
  - pennfat.h, 39
- main
  - pennfat\_main.c, 42
  - pennfat\_main.h, 43
- MAX\_OPEN\_FILES
  - descriptors.h, 19
- mkfs
  - pennfat.c, 32
  - pennfat.h, 39
- mode
  - OpenFileDescriptor, 8
- mount\_fs
  - pennfat.c, 33
  - pennfat.h, 40
- mtime
  - DirectoryEntry, 5
- mv\_fs
  - pennfat.c, 33
  - pennfat.h, 40
- name
  - DirectoryEntry, 6
- num\_directories
  - PennFAT, 9
- OpenFileDescriptor, 7
  - cursor, 7
  - entry, 7
  - mode, 8
  - used, 8
- openFileDescriptors
  - descriptors.c, 16
- PennFAT, 8
  - block\_size, 8
  - fat, 9
  - fat\_size, 9
  - fs\_fd, 9
  - num\_directories, 9
  - pennfat.h, 37
- pennfat.c, 29
  - cat\_fs, 30
  - chmod\_fs, 31
  - cp\_fs, 31
  - find\_file, 32
  - ls\_fs, 32
  - mkfs, 32
  - mount\_fs, 33
  - mv\_fs, 33
  - pf, 35
  - rm\_fs, 33
  - touch\_fs, 33
  - trim\_newline, 35
  - unmount\_fs, 35
- pennfat.h, 36
  - CAT\_BUFFER\_SIZE, 37
  - cat\_fs, 38
  - chmod\_fs, 38
  - cp\_fs, 38
  - DirectoryEntry, 37
  - find\_file, 39
  - ls\_fs, 39
  - mkfs, 39



- mount\_fs, [40](#)
- mv\_fs, [40](#)
- PennFAT, [37](#)
- pf, [41](#)
- rm\_fs, [40](#)
- touch\_fs, [41](#)
- trim\_newline, [41](#)
- unmount\_fs, [41](#)
- pennfat\_main.c, [42](#)
  - main, [42](#)
- pennfat\_main.h, [43](#)
  - main, [43](#)
- perm
  - DirectoryEntry, [6](#)
- pf
  - pennfat.c, [35](#)
  - pennfat.h, [41](#)
- reserved
  - DirectoryEntry, [6](#)
- rm\_fs
  - pennfat.c, [33](#)
  - pennfat.h, [40](#)
- size
  - DirectoryEntry, [6](#)
- touch\_fs
  - pennfat.c, [33](#)
  - pennfat.h, [41](#)
- trim\_newline
  - pennfat.c, [35](#)
  - pennfat.h, [41](#)
- type
  - DirectoryEntry, [6](#)
- unmount\_fs
  - pennfat.c, [35](#)
  - pennfat.h, [41](#)
- update\_directory\_entry
  - directory.c, [24](#)
  - directory.h, [29](#)
- update\_fat\_entry
  - directory.c, [26](#)
  - directory.h, [29](#)
- used
  - OpenFileDescriptor, [8](#)