PennOS Team 10 Spring 2025

Generated by Doxygen 1.9.1

Chapter 1

Scheduler Running Instructions

Use the `scheduler.mk` defined in the root directory of the repo using -

```
make -f scheduler.mk
```

Chapter 2

README

Lucky & Aagam Penn Shell

2.1 TODOs

There are a bunch of TODOs in the code. In addition, I've spotted some weird error conditions which i'll put here to investigate:

- [ ] piping in
  - `echo hello > test.txt`
  - `test.txt > cat`
  - see a "text file busy" exception which means that some part of that is an executable that is trying to be modified
  - maybe redirecting into cat does sonething weird?
- [ ] `cat test.txt` and `/bin/cat test.txt` both don't output anything (also `cat test.txt` should error, right?)
- [ ] really weird ctrl-d behavior

  hypothesis: command is hanging homehow but ctrl-c still outputs a print

  - need to add a handler that actually interrupts the child process ``` penn-shell> /bin/echo | cat penn-shell> /bin/echo

  penn-shell> /bin/echo "hello world" | wc -w ^C penn-shell> /bin/echo ^C penn-shell> /bin/echo "hello world" | /bin/wc -w

  ^C penn-shell> ^C penn-shell> ```
- [ ]

2.2 Morning 2/27

Current state:

- can enqueue things onto the list using the `&`
- the commands execute in the BG

Remains to do:

1. ~~figure out why the waitpid(-1,. .., WNOHANG) (the while loop inside of the main loop) isn't detecting the dead background jobs~~
   (a) was missing Wuntraced
2. add function to dequeue a job by pid (what #1 should return to us)
   (a) This requires us to track the PID of the process that wraps the job somewhere. We can probably do this in the job.pids array as it is currently unused, but we could also add a job.leader_pid
   (b) Once we've done that, we can just write a function that finds the node with the right PID and extracts it
3. complete handle_fg and handle_bg functions in jobs.c

Chapter 3

Todo List

Member print_job_list ()

Clarify the purpose of this function.

Member remove_foreground_job (job ∗job)

Rename this function to better reflect its actual usage (removing specific stopped jobs?).

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Class Documentation

6.1 child_process_st Struct Reference

```
#include <scheduler.h>
```

Collaboration diagram for child_process_st:

Public Attributes

- pcb_t ∗ process
- child_process_t ∗ next
- child_process_t ∗ prev

6.1.1 Member Data Documentation

6.1.1.1 next

```
child_process_t* child_process_st::next
```

6.1.1.2 prev

```
child_process_t* child_process_st::prev
```

6.1.1.3 process

```
pcb_t* child_process_st::process
```

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

- src/scheduler/scheduler.h

6.2 command_context Struct Reference

```
#include <commands.h>
```

Public Attributes

- char ** command
- int stdin_fd
- int stdout_fd

6.2.1 Member Data Documentation

6.2.1.1 command

```
char** command_context::command
```

6.2.1.2 stdin_fd

```
int command_context::stdin_fd
```

6.2.1.3 stdout_fd

```
int command_context::stdout_fd
```

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

- src/shell/commands.h

6.3 directory_entry_st Struct Reference

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

Public Attributes

- char name [32]
- uint32_t size
- uint16_t first_block
- uint8_t type
- uint8_t perm
- time_t mtime
- char padding [16]

6.3.1 Member Data Documentation

6.3.1.1 first_block

```
uint16_t directory_entry_st::first_block
```

6.3.1.2 mtime

```
time_t directory_entry_st::mtime
```

6.3.1.3 name

```
char directory_entry_st::name[32]
```

6.3.1.4 padding

```
char directory_entry_st::padding[16]
```

6.3.1.5 perm

```
uint8_t directory_entry_st::perm
```

6.3.1.6 size

```
uint32_t directory_entry_st::size
```

6.3.1.7 type

```
uint8_t directory_entry_st::type
```

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

- src/pennfat/fat.h

6.4 fat16_fs_st Struct Reference

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

Public Attributes

- uint16_t ∗ fat
- size_t fat_size
- uint16_t block_size
- uint16_t blocks_in_fat
- int fd
- void ∗ block_buf

6.4.1 Member Data Documentation

6.4.1.1 block_buf

```
void* fat16_fs_st::block_buf
```

6.4.1.2 block_size

```
uint16_t fat16_fs_st::block_size
```

6.4.1.3 blocks_in_fat

```
uint16_t fat16_fs_st::blocks_in_fat
```

6.4.1.4 fat

```
uint16_t* fat16_fs_st::fat
```

6.4.1.5 fat_size

```
size_t fat16_fs_st::fat_size
```

6.4.1.6 fd

```
int fat16_fs_st::fd
```

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

- src/pennfat/fat.h

6.5 global_fd_entry_st Struct Reference

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

Collaboration diagram for global_fd_entry_st:

Public Attributes

- size_t ref_count
- directory_entry * ptr_to_dir_entry
- uint16_t dir_entry_block_num
- uint8_t dir_entry_idx
- uint8_t write_locked
- uint32_t offset

6.5.1 Member Data Documentation

6.5.1.1 dir_entry_block_num

```
uint16_t global_fd_entry_st::dir_entry_block_num
```

6.5.1.2 dir_entry_idx

```
uint8_t global_fd_entry_st::dir_entry_idx
```

6.5.1.3 offset

```
uint32_t global_fd_entry_st::offset
```

6.5.1.4 ptr_to_dir_entry

```
directory_entry* global_fd_entry_st::ptr_to_dir_entry
```

6.5.1.5 ref_count

```
size_t global_fd_entry_st::ref_count
```

6.5.1.6 write_locked

```
uint8_t global_fd_entry_st::write_locked
```

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

- src/pennfat/fat.h

6.6 job_ll_node_st Struct Reference

```
#include <jobs.h>
```

Collaboration diagram for job_ll_node_st:

Public Attributes

- job ∗ job
- struct job_ll_node_st ∗ next
- struct job_ll_node_st ∗ prev

6.6.1 Member Data Documentation

6.6.1.1 job

```
job* job_ll_node_st::job
```

6.6.1.2 next

```
struct job_ll_node_st* job_ll_node_st::next
```

6.6.1.3 prev

```
struct job_ll_node_st* job_ll_node_st::prev
```

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

- src/shell/jobs.h

6.7 job_st Struct Reference

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

Collaboration diagram for job_st:

Public Attributes

- jid_t id
- pid_t pid
- size_t num_processes
- job_status status
- struct parsed_command ∗ cmd

6.7.1 Member Data Documentation

6.7.1.1 cmd

```
struct parsed_command* job_st::cmd
```

6.7.1.2 id

```
jid_t job_st::id
```

6.7.1.3 num_processes

```
size_t job_st::num_processes
```

6.7.1.4 pid

```
pid_t job_st::pid
```

6.7.1.5 status

```
job_status job_st::status
```

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

- src/shell/Job.h

6.8 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 [ ]

6.8.1 Detailed Description

struct parsed_command stored all necessary information needed for penn-shell.

6.8.2 Member Data Documentation

6.8.2.1 commands

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

6.8.2.2 is_background

```
bool parsed_command::is_background
```

6.8.2.3 is_file_append

```
bool parsed_command::is_file_append
```

6.8.2.4 num_commands

```
size_t parsed_command::num_commands
```

6.8.2.5 stdin_file

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

6.8.2.6 stdout_file

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

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

- src/shell/parser.h

6.9 pcb_st Struct Reference

```
#include <scheduler.h>
```

Collaboration diagram for pcb_st:

Public Attributes

- pid_t pid
- pid_t ppid
- pid_t pgid
- child_process_ll_t children
- process_fd_entry process_fd_table [PROCESS_FD_TABLE_SIZE]
- process_state state
- pid_t waited_child
- bool ignore_sigint
- bool ignore_sigtstp
- int errnumber
- priority_t priority
- double sleep_time
- spthread_t ∗ thread
- void ∗(∗ func)(void ∗)
- char ∗ command
- char ∗∗ argv
- int exit_status
- pcb_t ∗ prev
- pcb_t ∗ next

6.9.1 Member Data Documentation

6.9.1.1 argv

```
char** pcb_st::argv
```

6.9.1.2 children

```
child_process_ll_t pcb_st::children
```

6.9.1.3 command

```
char* pcb_st::command
```

6.9.1.4 errnumber

```
int pcb_st::errnumber
```

6.9.1.5 exit_status

```
int pcb_st::exit_status
```

6.9.1.6 func

```
void*(* pcb_st::func) (void *)
```

6.9.1.7 ignore_sigint

```
bool pcb_st::ignore_sigint
```

6.9.1.8 ignore_sigtstp

```
bool pcb_st::ignore_sigtstp
```

6.9.1.9 next

```
pcb_t* pcb_st::next
```

6.9.1.10 pgid

`pid_t pcb_st::pgid`

6.9.1.11 pid

`pid_t pcb_st::pid`

6.9.1.12 ppid

`pid_t pcb_st::ppid`

6.9.1.13 prev

`pcb_t* pcb_st::prev`

6.9.1.14 priority

`priority_t pcb_st::priority`

6.9.1.15 process_fd_table

`process_fd_entry pcb_st::process_fd_table[PROCESS_FD_TABLE_SIZE]`

6.9.1.16 sleep_time

`double pcb_st::sleep_time`

6.9.1.17 state

`process_state pcb_st::state`

6.9.1.18 thread

```
spthread_t* pcb_st::thread
```

6.9.1.19 waited_child

```
pid_t pcb_st::waited_child
```

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

- src/scheduler/scheduler.h

6.10 process_fd_entry_st Struct Reference

```
#include <scheduler.h>
```

Public Attributes

- uint16_t global_fd
- uint32_t offset
- uint8_t mode
- bool in_use

6.10.1 Member Data Documentation

6.10.1.1 global_fd

```
uint16_t process_fd_entry_st::global_fd
```

6.10.1.2 in_use

```
bool process_fd_entry_st::in_use
```

6.10.1.3 mode

```
uint8_t process_fd_entry_st::mode
```

6.10.1.4 offset

```
uint32_t process_fd_entry_st::offset
```

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

- src/scheduler/scheduler.h

6.11 scheduler Struct Reference

```
#include <scheduler.h>
```

Collaboration diagram for scheduler:

Public Member Functions

- linked_list (pcb_t) ready_queues[3]
- linked_list (pcb_t) blocked_queue
- linked_list (pcb_t) zombie_queue
- linked_list (pcb_t) stopped_queue

Public Attributes

- unsigned int ticks
- pcb_t ∗ init_process
- pcb_t ∗ current_process
- unsigned int process_count
- pid_t terminal_controlling_pid

6.11.1 Member Function Documentation

6.11.1.1 linked_list() [1/4]

```
scheduler::linked_list (
            pcb_t  )
```

6.11.1.2 linked_list() [2/4]

```
scheduler::linked_list (
            pcb_t  )
```

6.11.1.3 linked_list() [3/4]

```
scheduler::linked_list (
            pcb_t  )
```

6.11.1.4 linked_list() [4/4]

```
scheduler::linked_list (
            pcb_t  )
```

6.11.2 Member Data Documentation

6.11.2.1 current_process

```
pcb_t* scheduler::current_process
```

6.11.2.2 init_process

```
pcb_t* scheduler::init_process
```

6.11.2.3 process_count

```
unsigned int scheduler::process_count
```

6.11.2.4 terminal_controlling_pid

```
pid_t scheduler::terminal_controlling_pid
```

6.11.2.5 ticks

```
unsigned int scheduler::ticks
```

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

- src/scheduler/scheduler.h

6.12 spthread_fwd_args_st Struct Reference

Collaboration diagram for spthread_fwd_args_st:

Public Attributes

- pthread_fn actual_routine
- void ∗ actual_arg
- bool setup_done
- pthread_mutex_t setup_mutex
- pthread_cond_t setup_cond
- spthread_meta_t ∗ child_meta

6.12.1 Member Data Documentation

6.12.1.1 actual_arg

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

6.12.1.2 actual_routine

```
pthread_fn spthread_fwd_args_st::actual_routine
```

6.12.1.3 child_meta

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

6.12.1.4 setup_cond

```
pthread_cond_t spthread_fwd_args_st::setup_cond
```

6.12.1.5 setup_done

```
bool spthread_fwd_args_st::setup_done
```

6.12.1.6 setup_mutex

```
pthread_mutex_t spthread_fwd_args_st::setup_mutex
```

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

- src/scheduler/spthread.c

6.13 spthread_meta_st Struct Reference

Public Attributes

- sigset_t suspend_set
- volatile sig_atomic_t state
- pthread_mutex_t meta_mutex

6.13.1 Member Data Documentation

6.13.1.1 meta_mutex

```
pthread_mutex_t spthread_meta_st::meta_mutex
```

6.13.1.2 state

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

6.13.1.3 suspend_set

```
sigset_t spthread_meta_st::suspend_set
```

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

- src/scheduler/spthread.c

6.14 spthread_signal_args_st Struct Reference

Public Attributes

- const int signal
- volatile sig_atomic_t ack
- pthread_mutex_t shutup_mutex

6.14.1 Member Data Documentation

6.14.1.1 ack

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

6.14.1.2 shutup_mutex

```
pthread_mutex_t spthread_signal_args_st::shutup_mutex
```

6.14.1.3 signal

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

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

- src/scheduler/spthread.c

6.15 spthread_st Struct Reference

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

Collaboration diagram for spthread_st:

Public Attributes

- pthread_t thread
- spthread_meta_t ∗ meta

6.15.1 Member Data Documentation

6.15.1.1 meta

```
spthread_meta_t * spthread_st::meta
```

6.15.1.2 thread

```
pthread_t spthread_st::thread
```

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

- src/scheduler/spthread.h

Chapter 7

File Documentation

7.1 src/pennfat/fat.c File Reference

```
#include "src/pennfat/fat.h"
#include "src/pennfat/fat_utils.h"
#include "src/utils/error_codes.h"
#include <stdint.h>
#include <stdbool.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
```

Include dependency graph for fat.c:

Macros

- #define GLOBAL_FD_TABLE_SIZE 4096
- #define GLOBAL_FD_TABLE_ENTRY_NOT_FOUND_SENTINEL 0xFFFF
- #define MIN_FILENAME_SIZE 1
- #define MAX_FILENAME_SIZE 31
- #define FAT_END_OF_FILE 0xFFFF
- #define EGET_BLOCK_BLOCK_NUM_0 1
- #define EGET_BLOCK_BLOCK_NUM_TOO_HIGH 2
- #define EGET_BLOCK_LSEEK_FAILED 4
- #define EGET_BLOCK_READ_FAILED 5
- #define EGET_BLOCK_TOO_FEW_BYTES_READ 6
- #define ENEXT_BLOCK_NUM_BLOCK_NUM_0 1
- #define ENEXT_BLOCK_NUM_BLOCK_NUM_TOO_HIGH 2
- #define EWRITE_BLOCK_BLOCK_NUM_0 1
- #define EWRITE_BLOCK_BLOCK_NUM_TOO_HIGH 2
- #define EWRITE_BLOCK_LSEEK_FAILED 4
- #define EWRITE_BLOCK_WRITE_FAILED 5
- #define EWRITE_BLOCK_TOO_FEW_BYTES_WRITTEN 6
- #define EFIND_FILE_IN_ROOT_DIR_GET_BLOCK_FAILED -1
- #define EFIND_FILE_IN_ROOT_DIR_NEXT_BLOCK_FAILED -2
- #define RFIND_FILE_IN_ROOT_DIR_FILE_NOT_FOUND 1

- #define RFIND_FILE_IN_ROOT_DIR_FILE_FOUND 0
- #define RFIND_FILE_IN_ROOT_DIR_FILE_DELETED 2
- #define RFIND_FILE_IN_GLOBAL_FD_TABLE_NOT_FOUND 1
- #define EFIND_EMPTY_SPOT_IN_ROOT_DIR_GET_BLOCK_FAILED -1
- #define EFIND_EMPTY_SPOT_IN_ROOT_DIR_NEXT_BLOCK_FAILED -2
- #define RFIND_EMPTY_SPOT_IN_ROOT_DIR_DELETED 0
- #define RFIND_EMPTY_SPOT_IN_ROOT_DIR_END_ENTRY 1
- #define EWRITE_ROOT_DIR_ENTRY_GET_BLOCK_FAILED 1
- #define EWRITE_ROOT_DIR_ENTRY_NO_EMPTY_BLOCKS 2
- #define EWRITE_ROOT_DIR_ENTRY_WRITE_BLOCK_FAILED 3
- #define EWRITE_NEW_ROOT_DIR_ENTRY_FIND_EMPTY_SPOT_IN_ROOT_DIR_FAILED 1
- #define EWRITE_NEW_ROOT_DIR_ENTRY_WRITE_ROOT_DIR_ENTRY_FAILED 2

Functions

- int min (int a, int b)
- bool is_mounted (void)

  Check if the pennfat (fat16) filesystem is mounted.
- int mount (char ∗fs_name)

  Mount the pennfat (fat16) filesystem from the file named fs_name.
- int unmount (void)

  Unmount the pennfat (fat16) filesystem from the struct pointed to by ptr_to_fs. This function will 0 out the struct on success (but may not on failure).
- int k_fprintf_short (int fd, const char ∗format,...)

  Like dprintf but using pennfat and limited to 1023 characters.
- void clear_fat_file (uint16_t block)
- uint16_t first_empty_block (void)
- bool check_filename_charset (const char ∗str, uint8_t strlen)
- uint32_t get_blocks_in_data_region (void)
- uint32_t get_byte_offset_of_block (uint16_t block_num)
- int get_block (uint16_t block_num, void ∗data)
- int next_block_num (uint16_t block_num, uint16_t ∗next_block_num)
- int write_block (uint16_t block_num, void ∗data)
- int find_file_in_root_dir (const char ∗fname, directory_entry ∗ptr_to_dir_entry, uint16_t ∗ptr_to_block, uint8↩
_t ∗ptr_to_dir_entry_idx)
- int find_file_in_global_fd_table (const char ∗fname, uint16_t ∗ptr_to_fd_idx)
- int find_empty_spot_in_root_dir (uint16_t ∗ptr_to_block, uint8_t ∗ptr_to_offset)
- int write_root_dir_entry (directory_entry ∗ptr_to_dir_entry, uint16_t block, uint8_t directory_entry_offset)
- int write_new_root_dir_entry (directory_entry ∗ptr_to_dir_entry, uint16_t ∗ptr_to_block, uint8_t ∗ptr_to_dir↩
_entry_idx)
- uint16_t find_empty_fd ()
- bool is_valid_filename (const char ∗fname)
- int k_open (const char ∗fname, int mode)

  Open a file.
- int k_close (int fd)

  Close a file.
- int k_read (int fd, int n, char ∗buf)

  Read from a file.
- int64_t k_lseek (int fd, int offset, int whence)

  Seek to a position in a file.
- int k_write (int fd, const char ∗str, int n)

  Write to a file.

- int k_unlink (const char ∗fname)

  Remove (unlink) a file.
- int ls_dir_entry (directory_entry ∗ptr_to_dir_entry)
- int k_ls (const char ∗filename)

  List the contents of a directory.
- int k_chmod (const char ∗fname, uint8_t perm, int mode)

  Change the permissions of a file.
- int k_mv (const char ∗src, const char ∗dest)

  Move a file from src to dest.
- int k_setmode (int fd, int mode)

  Set the mode the global file descriptor is opened with.
- int k_getmode (int fd)

  Get the mode the global file descriptor is opened with.

Variables

- global_fd_entry global_fd_table [GLOBAL_FD_TABLE_SIZE] = {0}
- fat16_fs fs
- char K_FPRINTF_SHORT_BUF [1024]

7.1.1 Macro Definition Documentation

7.1.1.1 EFIND_EMPTY_SPOT_IN_ROOT_DIR_GET_BLOCK_FAILED

```
#define EFIND_EMPTY_SPOT_IN_ROOT_DIR_GET_BLOCK_FAILED -1
```

7.1.1.2 EFIND_EMPTY_SPOT_IN_ROOT_DIR_NEXT_BLOCK_FAILED

```
#define EFIND_EMPTY_SPOT_IN_ROOT_DIR_NEXT_BLOCK_FAILED -2
```

7.1.1.3 EFIND_FILE_IN_ROOT_DIR_GET_BLOCK_FAILED

```
#define EFIND_FILE_IN_ROOT_DIR_GET_BLOCK_FAILED -1
```

7.1.1.4 EFIND_FILE_IN_ROOT_DIR_NEXT_BLOCK_FAILED

```
#define EFIND_FILE_IN_ROOT_DIR_NEXT_BLOCK_FAILED -2
```

7.1.1.5 EGET_BLOCK_BLOCK_NUM_0

```
#define EGET_BLOCK_BLOCK_NUM_0 1
```

7.1.1.6 EGET_BLOCK_BLOCK_NUM_TOO_HIGH

```
#define EGET_BLOCK_BLOCK_NUM_TOO_HIGH 2
```

7.1.1.7 EGET_BLOCK_LSEEK_FAILED

```
#define EGET_BLOCK_LSEEK_FAILED 4
```

7.1.1.8 EGET_BLOCK_READ_FAILED

```
#define EGET_BLOCK_READ_FAILED 5
```

7.1.1.9 EGET_BLOCK_TOO_FEW_BYTES_READ

```
#define EGET_BLOCK_TOO_FEW_BYTES_READ 6
```

7.1.1.10 ENEXT_BLOCK_NUM_BLOCK_NUM_0

```
#define ENEXT_BLOCK_NUM_BLOCK_NUM_0 1
```

7.1.1.11 ENEXT_BLOCK_NUM_BLOCK_NUM_TOO_HIGH

```
#define ENEXT_BLOCK_NUM_BLOCK_NUM_TOO_HIGH 2
```

7.1.1.12 EWRITE_BLOCK_BLOCK_NUM_0

```
#define EWRITE_BLOCK_BLOCK_NUM_0 1
```

7.1.1.13 EWRITE_BLOCK_BLOCK_NUM_TOO_HIGH

```
#define EWRITE_BLOCK_BLOCK_NUM_TOO_HIGH 2
```

7.1.1.14 EWRITE_BLOCK_LSEEK_FAILED

```
#define EWRITE_BLOCK_LSEEK_FAILED 4
```

7.1.1.15 EWRITE_BLOCK_TOO_FEW_BYTES_WRITTEN

```
#define EWRITE_BLOCK_TOO_FEW_BYTES_WRITTEN 6
```

7.1.1.16 EWRITE_BLOCK_WRITE_FAILED

```
#define EWRITE_BLOCK_WRITE_FAILED 5
```

7.1.1.17 EWRITE_NEW_ROOT_DIR_ENTRY_FIND_EMPTY_SPOT_IN_ROOT_DIR_FAILED

```
#define EWRITE_NEW_ROOT_DIR_ENTRY_FIND_EMPTY_SPOT_IN_ROOT_DIR_FAILED 1
```

7.1.1.18 EWRITE_NEW_ROOT_DIR_ENTRY_WRITE_ROOT_DIR_ENTRY_FAILED

```
#define EWRITE_NEW_ROOT_DIR_ENTRY_WRITE_ROOT_DIR_ENTRY_FAILED 2
```

7.1.1.19 EWRITE_ROOT_DIR_ENTRY_GET_BLOCK_FAILED

```
#define EWRITE_ROOT_DIR_ENTRY_GET_BLOCK_FAILED 1
```

7.1.1.20 EWRITE_ROOT_DIR_ENTRY_NO_EMPTY_BLOCKS

```
#define EWRITE_ROOT_DIR_ENTRY_NO_EMPTY_BLOCKS 2
```

7.1.1.21 EWRITE_ROOT_DIR_ENTRY_WRITE_BLOCK_FAILED

```
#define EWRITE_ROOT_DIR_ENTRY_WRITE_BLOCK_FAILED 3
```

7.1.1.22 FAT_END_OF_FILE

```
#define FAT_END_OF_FILE 0xFFFF
```

7.1.1.23 GLOBAL_FD_TABLE_ENTRY_NOT_FOUND_SENTINEL

```
#define GLOBAL_FD_TABLE_ENTRY_NOT_FOUND_SENTINEL 0xFFFF
```

7.1.1.24 GLOBAL_FD_TABLE_SIZE

```
#define GLOBAL_FD_TABLE_SIZE 4096
```

7.1.1.25 MAX_FILENAME_SIZE

```
#define MAX_FILENAME_SIZE 31
```

7.1.1.26 MIN_FILENAME_SIZE

```
#define MIN_FILENAME_SIZE 1
```

7.1.1.27 RFIND_EMPTY_SPOT_IN_ROOT_DIR_DELETED

```
#define RFIND_EMPTY_SPOT_IN_ROOT_DIR_DELETED 0
```

7.1.1.28 RFIND_EMPTY_SPOT_IN_ROOT_DIR_END_ENTRY

```
#define RFIND_EMPTY_SPOT_IN_ROOT_DIR_END_ENTRY 1
```

7.1.1.29 RFIND_FILE_IN_GLOBAL_FD_TABLE_NOT_FOUND

```
#define RFIND_FILE_IN_GLOBAL_FD_TABLE_NOT_FOUND 1
```

7.1.1.30 RFIND_FILE_IN_ROOT_DIR_FILE_DELETED

```
#define RFIND_FILE_IN_ROOT_DIR_FILE_DELETED 2
```

7.1.1.31 RFIND_FILE_IN_ROOT_DIR_FILE_FOUND

```
#define RFIND_FILE_IN_ROOT_DIR_FILE_FOUND 0
```

7.1.1.32 RFIND_FILE_IN_ROOT_DIR_FILE_NOT_FOUND

```
#define RFIND_FILE_IN_ROOT_DIR_FILE_NOT_FOUND 1
```

7.1.2 Function Documentation

7.1.2.1 check_filename_charset()

```
bool check_filename_charset (
            const char * str,
            uint8_t strlen )
```

Whether the provided string fits thei POSIX filename charset

7.1.2.2 clear_fat_file()

```
void clear_fat_file (
            uint16_t block )
```

Clear a file starting at block. It is expected that block is the first block in the file. If it is not

Note that this function does not validate that the blocks at each stage (including the initially passed block) are in bounds. It assumes the FAT is maintained as valid.

7.1.2.3 find_empty_fd()

```
uint16_t find_empty_fd ( )
```

7.1.2.4 find_empty_spot_in_root_dir()

```
int find_empty_spot_in_root_dir (
            uint16_t * ptr_to_block,
            uint8_t * ptr_to_offset )
```

7.1.2.5 find_file_in_global_fd_table()

```
int find_file_in_global_fd_table (
            const char * fname,
            uint16_t * ptr_to_fd_idx )
```

Looks for a file in the global file table matching fname, returning 0 and setting the memory address at ptr_to_fd_idx to the index in the file table if found and returning RFIND_FILE_IN_GLOBAL_FD_TABLE_NOT_FOUND (1) if not found.

Note that this function skips files that have 0,1 or 2 as the first byte of their name because these entries are not valid entries (in fact, they've have their name mangled, so they should not match any but an adversarial fname). It also skips files with a ref_count of 0

7.1.2.6 find_file_in_root_dir()

```
int find_file_in_root_dir (
            const char * fname,
            directory_entry * ptr_to_dir_entry,
            uint16_t * ptr_to_block,
            uint8_t * ptr_to_dir_entry_idx )
```

Find the file with name fname in the root directory of the filesystem. If it is found, mallocs new directory_entry and returns a pointer to it via the ptr_to_dir_entry_buf

Returns >= 0 on success (see RFIND_FILE_IN_ROOT_DIR_* return codes) and < 0 on error (see EFIND_FILE↩
_IN_ROOT_DIR_* error codes)

7.1.2.7 first_empty_block()

```
uint16_t first_empty_block (
            void )
```

Finds the first empty block by walking the fat from index 1. Returns the block index if such a block exists and 0 if there is no empty block

7.1.2.8 get_block()

```
int get_block (
             uint16_t block_num,
             void * data )
```

Get the data inside of a block and store it into the buffer pointed to by data. It is assumed that the memory pointed to by data has sufficient capacity to store a full block (fs.block_size).

Returns 0 on success and an error code on error. See the EGET_BLOCK_∗ error code.

7.1.2.9 get_blocks_in_data_region()

```
uint32_t get_blocks_in_data_region (
             void )
```

7.1.2.10 get_byte_offset_of_block()

```
uint32_t get_byte_offset_of_block (
             uint16_t block_num )
```

7.1.2.11 is_mounted()

```
bool is_mounted (
             void )
```

Check if the pennfat (fat16) filesystem is mounted.

Returns

bool true if the pennfat (fat16) filesystem is mounted, false otherwise

7.1.2.12 is_valid_filename()

```
bool is_valid_filename (
             const char * fname )
```

7.1.2.13 k_chmod()

```
int k_chmod (
             const char * fname,
             uint8_t perm,
             int mode )
```

Change the permissions of a file.

Parameters

fname	file name
perm	permissions to set the file to
mode	mode to set the file to (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int 0 on success, or negative error code

7.1.2.14 k_close()

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

Close a file.

Parameters

fd	global file descriptor to close

Returns

int 0 on success, or negative error code

7.1.2.15 k_fprintf_short()

```
int k_fprintf_short (
             int fd,
             const char * format,
              ... )
```

Like dprintf but using pennfat and limited to 1023 characters.

Parameters

fd	global file descriptor to write to
format	format string
...	arguments to format string

Returns

int number of characters written, or negative error code

7.1.2.16 k_getmode()

```
int k_getmode (
            int fd )
```

Get the mode the global file descriptor is opened with.

Parameters

fd	global file descriptor to get the mode of

Returns

int mode of the global file descriptor, or negative error code

7.1.2.17 k_ls()

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

List the contents of a directory.

Parameters

filename	directory to list the contents of

Returns

int 0 on success, or negative error code

7.1.2.18 k_lseek()

```
int64_t k_lseek (
            int fd,
            int offset,
            int whence )
```

Seek to a position in a file.

Parameters

fd	global file descriptor to seek in
offset	offset to seek to
whence	whence to seek from (i.e., F_SEEK_SET, F_SEEK_CUR, F_SEEK_END)

Returns

int64_t new offset, or negative error code

Note

IMPORTANT: this function does not have the same signature as lseek(2) because it returns a negative error code on error. On success, it returns the offset which is guaranteed to fit in a uint32_t

7.1.2.19 k_mv()

```
int k_mv (
            const char * src,
            const char * dest )
```

Move a file from src to dest.

Parameters

src	source file name
dest	destination file name

Returns

int 0 on success, or negative error code

7.1.2.20 k_open()

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

Open a file.

Parameters

fname	file name
mode	mode to open the file with (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int global file descriptor, or negative error code

7.1.2.21 k_read()

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

Read from a file.

Parameters

fd	global file descriptor to read from
n	number of bytes to read from the file
buf	buffer to read the bytes into

Returns

int number of bytes read, or negative error code

7.1.2.22 k_setmode()

```
int k_setmode (
            int fd,
            int mode )
```

Set the mode the global file descriptor is opened with.

Parameters

fd	global file descriptor to set the mode of
mode	mode to set the global file descriptor to (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int 0 on success, or negative error code

7.1.2.23 k_unlink()

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

Remove (unlink) a file.

Parameters

fname	file name

Returns

int 0 on success, or negative error code

7.1.2.24 k_write()

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

Write to a file.

Parameters

fd	global file descriptor to write to
str	bytes to write to the file
n	number of bytes to write to the file

Returns

int number of bytes written, or negative error code

7.1.2.25 ls_dir_entry()

```
int ls_dir_entry (
            directory_entry * ptr_to_dir_entry )
```

7.1.2.26 min()

```
int min (
            int a,
            int b )
```

7.1.2.27 mount()

```
int mount (
            char * fs_name )
```

Mount the pennfat (fat16) filesystem from the file named fs_name.

Parameters

fs_name	file name of the FAT in the host filesystem

Returns

int 0 on success, and an error code on error

7.1.2.28 next_block_num()

```
int next_block_num (
            uint16_t block_num,
            uint16_t * next_block_num )
```

Get the next block number from the FAT table.

7.1.2.29 unmount()

```
int unmount (
            void )
```

Unmount the pennfat (fat16) filesystem from the struct pointed to by ptr_to_fs. This function will 0 out the struct on success (but may not on failure).

Returns

int 0 on success, and an error code on error

7.1.2.30 write_block()

```
int write_block (
            uint16_t block_num,
            void * data )
```

A thin wrapper around wrapper that makes it easier to write exactly 1 block of data

7.1.2.31 write_new_root_dir_entry()

```
int write_new_root_dir_entry (
            directory_entry * ptr_to_dir_entry,
            uint16_t * ptr_to_block,
            uint8_t * ptr_to_dir_entry_idx )
```

7.1.2.32 write_root_dir_entry()

```
int write_root_dir_entry (
            directory_entry * ptr_to_dir_entry,
            uint16_t block,
            uint8_t directory_entry_offset )
```

7.1.3 Variable Documentation

7.1.3.1 fs

```
fat16_fs fs
```

Initial value:

```
= {
    .fat = NULL,
    .fat_size = 0,
    .block_size = 0,
    .blocks_in_fat = 0,
    .fd = -1,
    .block_buf = NULL}
```

7.1.3.2 global_fd_table

```
global_fd_entry global_fd_table[GLOBAL_FD_TABLE_SIZE] = {0}
```

7.1.3.3 K_FPRINTF_SHORT_BUF

```
char K_FPRINTF_SHORT_BUF[1024]
```

7.2 src/pennfat/fat.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>
#include <time.h>
#include "src/pennfat/fat_constants.h"
```

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

Classes

- struct fat16_fs_st
- struct directory_entry_st
- struct global_fd_entry_st

Macros

- #define EFS_NOT_MOUNTED 99
- #define EMOUNT_BAD_FAT_FIRST_ENTRY 1
- #define EMOUNT_ALREADY_MOUNTED 2
- #define EMOUNT_MALLOC_FAILED 4
- #define EMOUNT_OPEN_FAILED 5
- #define EMOUNT_MMAP_FAILED 6
- #define EMOUNT_READ_FAILED 8
- #define EUNMOUNT_MUNMAP_FAILED 1
- #define EUNMOUNT_CLOSE_FAILED 2
- #define F_SEEK_SET 1
- #define F_SEEK_CUR 2
- #define F_SEEK_END 3
- #define STDIN_FD 0
- #define STDOUT_FD 1
- #define STDERR_FD 2
- #define P_NO_FILE_PERMISSION 0
- #define P_WRITE_ONLY_FILE_PERMISSION 2
- #define P_READ_ONLY_FILE_PERMISSION 4
- #define P_READ_AND_EXECUTABLE_FILE_PERMISSION 5
- #define P_READ_WRITE_FILE_PERMISSION 6
- #define P_READ_WRITE_AND_EXECUTABLE_FILE_PERMISSION 5

Typedefs

- typedef struct fat16_fs_st fat16_fs
- typedef struct directory_entry_st directory_entry
- typedef struct global_fd_entry_st global_fd_entry

Functions

- _Static_assert (sizeof(directory_entry)==64, "directory_entry must be 64 byes")
- int mount (char ∗fs_name)

  Mount the pennfat (fat16) filesystem from the file named fs_name.
- int unmount (void)

  Unmount the pennfat (fat16) filesystem from the struct pointed to by ptr_to_fs. This function will 0 out the struct on success (but may not on failure).
- bool is_mounted (void)

  Check if the pennfat (fat16) filesystem is mounted.
- int k_open (const char ∗fname, int mode)

  Open a file.
- int k_close (int fd)

  Close a file.
- int k_read (int fd, int n, char ∗buf)

  Read from a file.
- int64_t k_lseek (int fd, int offset, int whence)

  Seek to a position in a file.
- int k_write (int fd, const char ∗str, int n)

  Write to a file.
- int k_unlink (const char ∗fname)

Remove (unlink) a file.
- int k_ls (const char ∗filename)

  List the contents of a directory.
- int k_chmod (const char ∗fname, uint8_t perm, int mode)

  Change the permissions of a file.
- int k_mv (const char ∗src, const char ∗dest)

  Move a file from src to dest.
- int k_setmode (int fd, int mode)

  Set the mode the global file descriptor is opened with.
- int k_getmode (int fd)

  Get the mode the global file descriptor is opened with.
- int k_fprintf_short (int fd, const char ∗format,...)

  Like dprintf but using pennfat and limited to 1023 characters.

7.2.1 Macro Definition Documentation

7.2.1.1 EFS_NOT_MOUNTED

```
#define EFS_NOT_MOUNTED 99
```

7.2.1.2 EMOUNT_ALREADY_MOUNTED

```
#define EMOUNT_ALREADY_MOUNTED 2
```

7.2.1.3 EMOUNT_BAD_FAT_FIRST_ENTRY

```
#define EMOUNT_BAD_FAT_FIRST_ENTRY 1
```

7.2.1.4 EMOUNT_MALLOC_FAILED

```
#define EMOUNT_MALLOC_FAILED 4
```

7.2.1.5 EMOUNT_MMAP_FAILED

```
#define EMOUNT_MMAP_FAILED 6
```

7.2.1.6 EMOUNT_OPEN_FAILED

```
#define EMOUNT_OPEN_FAILED 5
```

7.2.1.7 EMOUNT_READ_FAILED

```
#define EMOUNT_READ_FAILED 8
```

7.2.1.8 EUNMOUNT_CLOSE_FAILED

```
#define EUNMOUNT_CLOSE_FAILED 2
```

7.2.1.9 EUNMOUNT_MUNMAP_FAILED

```
#define EUNMOUNT_MUNMAP_FAILED 1
```

7.2.1.10 F_SEEK_CUR

```
#define F_SEEK_CUR 2
```

7.2.1.11 F_SEEK_END

```
#define F_SEEK_END 3
```

7.2.1.12 F_SEEK_SET

```
#define F_SEEK_SET 1
```

7.2.1.13 P_NO_FILE_PERMISSION

```
#define P_NO_FILE_PERMISSION 0
```

7.2.1.14 P_READ_AND_EXECUTABLE_FILE_PERMISSION

```
#define P_READ_AND_EXECUTABLE_FILE_PERMISSION 5
```

7.2.1.15 P_READ_ONLY_FILE_PERMISSION

```
#define P_READ_ONLY_FILE_PERMISSION 4
```

7.2.1.16 P_READ_WRITE_AND_EXECUTABLE_FILE_PERMISSION

```
#define P_READ_WRITE_AND_EXECUTABLE_FILE_PERMISSION 5
```

7.2.1.17 P_READ_WRITE_FILE_PERMISSION

```
#define P_READ_WRITE_FILE_PERMISSION 6
```

7.2.1.18 P_WRITE_ONLY_FILE_PERMISSION

```
#define P_WRITE_ONLY_FILE_PERMISSION 2
```

7.2.1.19 STDERR_FD

```
#define STDERR_FD 2
```

7.2.1.20 STDIN_FD

```
#define STDIN_FD 0
```

7.2.1.21 STDOUT_FD

```
#define STDOUT_FD 1
```

7.2.2 Typedef Documentation

7.2.2.1 directory_entry

```
typedef struct directory_entry_st directory_entry
```

7.2.2.2 fat16_fs

```
typedef struct fat16_fs_st fat16_fs
```

7.2.2.3 global_fd_entry

```
typedef struct global_fd_entry_st global_fd_entry
```

7.2.3 Function Documentation

7.2.3.1 _Static_assert()

```
_Static_assert (
            sizeof(directory_entry)  = =64,
            "directory_entry must be 64 byes"  )
```

7.2.3.2 is_mounted()

```
bool is_mounted (
            void  )
```

Check if the pennfat (fat16) filesystem is mounted.

Returns

bool true if the pennfat (fat16) filesystem is mounted, false otherwise

7.2.3.3 k_chmod()

```
int k_chmod (
            const char * fname,
            uint8_t perm,
            int mode )
```

Change the permissions of a file.

Parameters

fname	file name
perm	permissions to set the file to
mode	mode to set the file to (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int 0 on success, or negative error code

7.2.3.4 k_close()

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

Close a file.

Parameters

fd	global file descriptor to close

Returns

int 0 on success, or negative error code

7.2.3.5 k_fprintf_short()

```
int k_fprintf_short (
            int fd,
            const char * format,
             ... )
```

Like dprintf but using pennfat and limited to 1023 characters.

Parameters

fd	global file descriptor to write to
format	format string
...	arguments to format string

Returns

int number of characters written, or negative error code

7.2.3.6 k_getmode()

```
int k_getmode (
            int fd )
```

Get the mode the global file descriptor is opened with.

Parameters

fd	global file descriptor to get the mode of

Returns

int mode of the global file descriptor, or negative error code

7.2.3.7 k_ls()

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

List the contents of a directory.

Parameters

filename	directory to list the contents of

Returns

int 0 on success, or negative error code

7.2.3.8 k_lseek()

```
int64_t k_lseek (
            int fd,
            int offset,
            int whence )
```

Seek to a position in a file.

Parameters

fd	global file descriptor to seek in
offset	offset to seek to
whence	whence to seek from (i.e., F_SEEK_SET, F_SEEK_CUR, F_SEEK_END)

Returns

int64_t new offset, or negative error code

Note

IMPORTANT: this function does not have the same signature as lseek(2) because it returns a negative error code on error. On success, it returns the offset which is guaranteed to fit in a uint32_t

7.2.3.9 k_mv()

```
int k_mv (
            const char * src,
            const char * dest )
```

Move a file from src to dest.

Parameters

src	source file name
dest	destination file name

Returns

int 0 on success, or negative error code

7.2.3.10 k_open()

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

Open a file.

Parameters

fname	file name
mode	mode to open the file with (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int global file descriptor, or negative error code

7.2.3.11 k_read()

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

Read from a file.

Parameters

fd	global file descriptor to read from
n	number of bytes to read from the file
buf	buffer to read the bytes into

Returns

int number of bytes read, or negative error code

7.2.3.12 k_setmode()

```
int k_setmode (
            int fd,
            int mode )
```

Set the mode the global file descriptor is opened with.

Parameters

fd	global file descriptor to set the mode of
mode	mode to set the global file descriptor to (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int 0 on success, or negative error code

7.2.3.13 k_unlink()

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

Remove (unlink) a file.

Parameters

fname	file name

Returns

int 0 on success, or negative error code

7.2.3.14 k_write()

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

Write to a file.

Parameters

fd	global file descriptor to write to
str	bytes to write to the file
n	number of bytes to write to the file

Returns

int number of bytes written, or negative error code

7.2.3.15 mount()

```
int mount (
            char * fs_name )
```

Mount the pennfat (fat16) filesystem from the file named fs_name.

Parameters

fs_name	file name of the FAT in the host filesystem

Returns

int 0 on success, and an error code on error

7.2.3.16 unmount()

```
int unmount (
            void )
```

Unmount the pennfat (fat16) filesystem from the struct pointed to by ptr_to_fs. This function will 0 out the struct on success (but may not on failure).

Returns

int 0 on success, and an error code on error

7.3 src/pennfat/fat_constants.h File Reference

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

Macros

- #define F_WRITE 1
- #define F_READ 0
- #define F_APPEND 2
- #define F_SEEK_SET 1
- #define F_SEEK_CUR 2
- #define F_SEEK_END 3
- #define STDIN_FD 0
- #define STDOUT_FD 1
- #define STDERR_FD 2
- #define F_CHMOD_SET 0
- #define F_CHMOD_ADD 1
- #define F_CHMOD_REMOVE 2
- #define F_CHMOD_R 4
- #define F_CHMOD_W 2
- #define F_CHMOD_X 1

7.3.1 Macro Definition Documentation

7.3.1.1 F_APPEND

```
#define F_APPEND 2
```

7.3.1.2 F_CHMOD_ADD

```
#define F_CHMOD_ADD 1
```

7.3.1.3 F_CHMOD_R

```
#define F_CHMOD_R 4
```

7.3.1.4 F_CHMOD_REMOVE

```
#define F_CHMOD_REMOVE 2
```

7.3.1.5 F_CHMOD_SET

```
#define F_CHMOD_SET 0
```

7.3.1.6 F_CHMOD_W

```
#define F_CHMOD_W 2
```

7.3.1.7 F_CHMOD_X

```
#define F_CHMOD_X 1
```

7.3.1.8 F_READ

```
#define F_READ 0
```

7.3.1.9 F_SEEK_CUR

```
#define F_SEEK_CUR 2
```

7.3.1.10 F_SEEK_END

```
#define F_SEEK_END 3
```

7.3.1.11 F_SEEK_SET

```
#define F_SEEK_SET 1
```

7.3.1.12 F_WRITE

```
#define F_WRITE 1
```

7.3.1.13 STDERR_FD

```
#define STDERR_FD 2
```

7.3.1.14 STDIN_FD

```
#define STDIN_FD 0
```

7.3.1.15 STDOUT_FD

```
#define STDOUT_FD 1
```

7.4 src/pennfat/fat_utils.c File Reference

```
#include "src/pennfat/fat_utils.h"
#include <stdint.h>
```
Include dependency graph for fat_utils.c:

Functions

- uint16_t block_size_of_config (uint8_t block_size_config)
- int parse_first_fat_entry (uint16_t first_entry, uint16_t ∗block_size_ptr, uint8_t ∗blocks_in_fat_ptr)

7.4.1 Function Documentation

7.4.1.1 block_size_of_config()

```
uint16_t block_size_of_config (
            uint8_t block_size_config )
```

Maps 0,1,2,3,4 to 256,512,1024,2048,4096 bytes

0 return indicates an invalid block_size_config was passed

7.4.1.2 parse_first_fat_entry()

```
int parse_first_fat_entry (
            uint16_t first_entry,
            uint16_t * block_size_ptr,
            uint8_t * blocks_in_fat_ptr )
```

Parse the first entry of the fat into block_size and blocks_in_fat

7.5 src/pennfat/fat_utils.h File Reference

```
#include <stdint.h>
```

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

Functions

- uint16_t block_size_of_config (uint8_t block_size_config)
- int parse_first_fat_entry (uint16_t first_entry, uint16_t ∗block_size_ptr, uint8_t ∗blocks_in_fat_ptr)

7.5.1 Function Documentation

7.5.1.1 block_size_of_config()

```
uint16_t block_size_of_config (
            uint8_t block_size_config )
```

Maps 0,1,2,3,4 to 256,512,1024,2048,4096 bytes

0 return indicates an invalid block_size_config was passed

7.5.1.2 parse_first_fat_entry()

```
int parse_first_fat_entry (
            uint16_t first_entry,
            uint16_t * block_size_ptr,
            uint8_t * blocks_in_fat_ptr )
```

Parse the first entry of the fat into block_size and blocks_in_fat

7.6 src/pennfat/mkfs.c File Reference

```
#include "src/pennfat/mkfs.h"
#include "src/pennfat/fat_utils.h"
#include <fcntl.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>
```

Include dependency graph for mkfs.c:

Functions

- int mkfs (char ∗fs_name, uint8_t blocks_in_fat, uint8_t block_size_config)

7.6.1 Function Documentation

7.6.1.1 mkfs()

```
int mkfs (
            char * fs_name,
            uint8_t blocks_in_fat,
            uint8_t block_size_config )
```

Create a new filesystem

Returns 0 on success and a non-zero error code on error

7.7 src/pennfat/mkfs.h File Reference

```
#include <stdint.h>
```

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

Macros

- #define BAD_BLOCKS_IN_FAT_VAL 1
- #define BAD_BLOCK_SIZE_CONFIG_VAL 2
- #define BAD_FS_NAME_VAL 3
- #define MKFS_UNKNOWN_ERROR 4
- #define EMKFS_OPEN_FAILED 5
- #define EMKFS_WRITE_FAILED 6
- #define EMKFS_WRITE_LESS 7
- #define EMKFS_LSEEK_FAILED 8
- #define EMKFS_CALLOC_FAILED 9
- #define EMKFS_CLOSE_FAILED 10

Functions

- int mkfs (char ∗fs_name, uint8_t blocks_in_fat, uint8_t block_size_config)

7.7.1 Macro Definition Documentation

7.7.1.1 BAD_BLOCK_SIZE_CONFIG_VAL

```
#define BAD_BLOCK_SIZE_CONFIG_VAL 2
```

7.7.1.2 BAD_BLOCKS_IN_FAT_VAL

```
#define BAD_BLOCKS_IN_FAT_VAL 1
```

7.7.1.3 BAD_FS_NAME_VAL

```
#define BAD_FS_NAME_VAL 3
```

7.7.1.4 EMKFS_CALLOC_FAILED

```
#define EMKFS_CALLOC_FAILED 9
```

7.7.1.5 EMKFS_CLOSE_FAILED

```
#define EMKFS_CLOSE_FAILED 10
```

7.7.1.6 EMKFS_LSEEK_FAILED

```
#define EMKFS_LSEEK_FAILED 8
```

7.7.1.7 EMKFS_OPEN_FAILED

```
#define EMKFS_OPEN_FAILED 5
```

7.7.1.8 EMKFS_WRITE_FAILED

```
#define EMKFS_WRITE_FAILED 6
```

7.7.1.9 EMKFS_WRITE_LESS

```
#define EMKFS_WRITE_LESS 7
```

7.7.1.10 MKFS_UNKNOWN_ERROR

```
#define MKFS_UNKNOWN_ERROR 4
```

7.7.2 Function Documentation

7.7.2.1 mkfs()

```
int mkfs (
            char * fs_name,
            uint8_t blocks_in_fat,
            uint8_t block_size_config )
```

Create a new filesystem

Returns 0 on success and a non-zero error code on error

7.8 src/pennfat/pennfat.c File Reference

```
#include "src/pennfat/mkfs.h"
#include "src/pennfat/fat.h"
#include "src/pennfat/fat_constants.h"
#include <string.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
```

Include dependency graph for pennfat.c:

Macros

- #define MAX_LINE_SIZE 1024

Functions

- char ∗∗ whitespace_tokenize (char ∗string, size_t ∗n_tokens)
- long int safe_strtol (char ∗string, char ∗prefix, bool ∗ok)
- int main (void)

Variables

- char line [MAX_LINE_SIZE]

7.8.1 Macro Definition Documentation

7.8.1.1 MAX_LINE_SIZE

```
#define MAX_LINE_SIZE 1024
```

7.8.2 Function Documentation

7.8.2.1 main()

```
int main (
             void  )
```

7.8.2.2 safe_strtol()

```
long int safe_strtol (
             char ∗ string,
             char ∗ prefix,
             bool ∗ ok )
```

7.8.2.3 whitespace_tokenize()

```
char** whitespace_tokenize (
            char * string,
            size_t * n_tokens )
```

7.8.3 Variable Documentation

7.8.3.1 line

```
char line[MAX_LINE_SIZE]
```

7.9 src/scheduler/fat_syscalls.c File Reference

```
#include <stdint.h>
#include <stdarg.h>
#include <string.h>
#include <stdio.h>
#include "src/pennfat/fat.h"
#include "src/scheduler/kernel.h"
#include "src/utils/error_codes.h"
#include "src/scheduler/sys.h"
```

Include dependency graph for fat_syscalls.c:

Macros

- #define PROCESS_FD_TABLE_ENTRY_NOT_FOUND_SENTINEL 0xFFFF
- #define ES_PROCESS_FILE_TABLE_FULL -100
- #define ES_READ_UNKNOWN_FD -101
- #define ES_READ_NO_TERMINAL_CONTROL -106

Functions

- int find_empty_spot_in_process_fd_table (void)
- int s_open (const char ∗fname, int mode)

  Open a file.
- int s_read (int fd, int n, char ∗buf)

  Read from a file.
- int s_write (int fd, const char ∗str, int n)

  Write to a file.
- int s_close (int fd)

  Close a file.
- int s_unlink (const char ∗fname)

  Remove (unlink) a file.
- int s_ls (const char ∗filename)

List the contents of a directory

- int s_chmod (const char ∗fname, uint8_t perm, int mode)

  Change the permissions of a file.
- int s_mv (const char ∗src, const char ∗dest)

  Move a file from src to dest.
- int s_fprintf_short (int fd, const char ∗format,...)

  Like dprintf but using pennfat and limited to 1023 characters.

Variables

- char S_FPRINTF_SHORT_BUF [1024]

7.9.1 Macro Definition Documentation

7.9.1.1 ES_PROCESS_FILE_TABLE_FULL

```
#define ES_PROCESS_FILE_TABLE_FULL -100
```

7.9.1.2 ES_READ_NO_TERMINAL_CONTROL

```
#define ES_READ_NO_TERMINAL_CONTROL -106
```

7.9.1.3 ES_READ_UNKNOWN_FD

```
#define ES_READ_UNKNOWN_FD -101
```

7.9.1.4 PROCESS_FD_TABLE_ENTRY_NOT_FOUND_SENTINEL

```
#define PROCESS_FD_TABLE_ENTRY_NOT_FOUND_SENTINEL 0xFFFF
```

7.9.2 Function Documentation

7.9.2.1 find_empty_spot_in_process_fd_table()

```
int find_empty_spot_in_process_fd_table (
            void  )
```

7.9.2.2 s_chmod()

```
int s_chmod (
            const char * fname,
            uint8_t perm,
            int mode )
```

Change the permissions of a file.

Parameters

fname	file name
perm	permissions to set the file to
mode	mode to set the file to (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int 0 on success, or negative error code

7.9.2.3 s_close()

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

Close a file.

Parameters

fd	process-level file descriptor to close

Returns

int 0 on success, or negative error code

7.9.2.4 s_fprintf_short()

```
int s_fprintf_short (
            int fd,
            const char * format,
             ... )
```

Like dprintf but using pennfat and limited to 1023 characters.

Parameters

fd	process-level file descriptor to write to
format	format string
...	arguments to format string

Returns

int number of characters written, or negative error code

7.9.2.5 s_ls()

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

List the contents of a directory

Parameters

filename	directory to list the contents of

Returns

int 0 on success, or negative error code

7.9.2.6 s_mv()

```
int s_mv (
            const char * src,
            const char * dest )
```

Move a file from src to dest.

Parameters

src	source file name
dest	destination file name

Returns

int 0 on success, or negative error code

7.9.2.7 s_open()

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

Open a file.

Parameters

fname	file name
mode	mode to open the file with (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int process-level file descriptor, or negative error code

7.9.2.8 s_read()

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

Read from a file.

Parameters

fd	process-level file descriptor to read from
n	number of bytes to read from the file
buf	buffer to read the bytes into

Returns

int number of bytes read, or negative error code

7.9.2.9 s_unlink()

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

Remove (unlink) a file.

Parameters

fname	file name

Returns

int 0 on success, or negative error code

7.9.2.10 s_write()

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

Write to a file.

Parameters

fd	process-level file descriptor to write to
str	string to write to the file
n	number of bytes to write to the file

Returns

int number of bytes written, or negative error code

7.9.3 Variable Documentation

7.9.3.1 S_FPRINTF_SHORT_BUF

```
char S_FPRINTF_SHORT_BUF[1024]
```

7.10 src/scheduler/fat_syscalls.h File Reference

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

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

Functions

- int s_open (const char ∗fname, int mode)

  Open a file.
- int s_read (int fd, int n, char ∗buf)

  Read from a file.
- int s_write (int fd, const char ∗str, int n)

  Write to a file.
- int s_close (int fd)

  Close a file.
- int s_unlink (const char ∗fname)

  Remove (unlink) a file.
- int s_ls (const char ∗filename)

  List the contents of a directory
- int s_chmod (const char ∗fname, uint8_t perm, int mode)

  Change the permissions of a file.
- int s_mv (const char ∗src, const char ∗dest)

  Move a file from src to dest.
- int s_fprintf_short (int fd, const char ∗format,...)

  Like dprintf but using pennfat and limited to 1023 characters.

7.10.1 Function Documentation

7.10.1.1 s_chmod()

```
int s_chmod (
            const char * fname,
            uint8_t perm,
            int mode )
```

Change the permissions of a file.

Parameters

fname	file name
perm	permissions to set the file to
mode	mode to set the file to (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int 0 on success, or negative error code

7.10.1.2 s_close()

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

Close a file.

Parameters

fd	process-level file descriptor to close

Returns

int 0 on success, or negative error code

7.10.1.3 s_fprintf_short()

```
int s_fprintf_short (
            int fd,
            const char * format,
             ... )
```

Like dprintf but using pennfat and limited to 1023 characters.

Parameters

fd	process-level file descriptor to write to
format	format string
...	arguments to format string

Returns

int number of characters written, or negative error code

7.10.1.4 s_ls()

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

List the contents of a directory

Parameters

filename	directory to list the contents of

Returns

int 0 on success, or negative error code

7.10.1.5 s_mv()

```
int s_mv (
            const char * src,
            const char * dest )
```

Move a file from src to dest.

Parameters

src	source file name
dest	destination file name

Returns

int 0 on success, or negative error code

7.10.1.6 s_open()

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

Open a file.

Parameters

fname	file name
mode	mode to open the file with (i.e., F_READ, F_WRITE, F_APPEND)

Returns

int process-level file descriptor, or negative error code

7.10.1.7 s_read()

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

Read from a file.

Parameters

fd	process-level file descriptor to read from
n	number of bytes to read from the file
buf	buffer to read the bytes into

Returns

int number of bytes read, or negative error code

7.10.1.8 s_unlink()

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

Remove (unlink) a file.

Parameters

fname	file name

Returns

int 0 on success, or negative error code

7.10.1.9 s_write()

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

Write to a file.

Parameters

fd	process-level file descriptor to write to
str	string to write to the file
n	number of bytes to write to the file

Returns

int number of bytes written, or negative error code

7.11 src/scheduler/kernel.c File Reference

```
#include "src/scheduler/kernel.h"
#include "scheduler.h"
#include "logger.h"
#include "../../lib/linked_list.h"
#include "spthread.h"
#include <string.h>
#include "src/shell/commands.h"
#include <stdlib.h>
#include "src/utils/error_codes.h"
```

Include dependency graph for kernel.c:

Functions

- pid_t k_proc_create (pcb_t ∗parent, void ∗(∗func)(void ∗), char ∗const argv[ ], priority_t priority)

  Creates a new process control block (PCB) as a child of the given parent, initializes its thread, and adds it to the scheduler's ready queue.
- int k_proc_cleanup (pcb_t ∗proc)

  Cleans up resources associated with a terminated or finished process.

7.11.1 Function Documentation

7.11.1.1 k_proc_cleanup()

```
int k_proc_cleanup (
            pcb_t * proc )
```

Cleans up resources associated with a terminated or finished process.

Clean up a terminated/finished process's resources. Called internally by the kernel, typically after reaping.

This function performs the necessary cleanup steps when a process is destroyed:

1. Reparents any orphaned children of the process to the init process (PID 1). It assumes `scheduler_state` and `scheduler_state->init_process` are accessible.
2. Removes the process from its parent's list of children.
3. Frees allocated resources within the PCB, including the thread structure (if any), command string, argv array, and the children list structure.
4. Frees the PCB structure itself.

Note: This function does *not* handle removing the process from scheduler queues (ready, blocked, etc.) or joining the underlying thread; those actions should occur before calling cleanup.

Parameters

proc	Pointer to the PCB of the process to clean up.

7.11.1.2 k_proc_create()

```
pid_t k_proc_create (
            pcb_t * parent,
            void *(*)(void *) func,
            char *const argv[],
            priority_t priority )
```

Creates a new process control block (PCB) as a child of the given parent, initializes its thread, and adds it to the scheduler's ready queue.

Create a new child process.

This function handles the complete setup of a new process:

1. Allocates and initializes the PCB.
2. Assigns a unique PID.
3. Sets up parent-child relationship.
4. Allocates and initializes the children list.
5. Sets file descriptors, function pointer.
6. Duplicates the command and arguments (`argv`).
7. Allocates the spthread structure.
8. Creates the underlying spthread using `spthread_create`.
9. Sets the initial state and priority.
10. Adds the process to the parent's children list.
11. Adds the process to the appropriate scheduler ready queue using `k_add_to_ready_queue`.

Parameters

parent	Pointer to the parent process's PCB. If NULL, the process is assumed to be an initial process (like init) with PPID 0.
func	The function the new process should execute.
argv	Null-terminated argument vector for the new process. The kernel copies this.
priority	The priority of the new process.

Returns

pid_t The PID of the newly created process, or -1 if any allocation or thread creation fails.

7.12 src/scheduler/kernel.h File Reference

```
#include "scheduler.h"
#include "src/pennfat/fat.h"
#include "src/pennfat/fat_constants.h"
```

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

Macros

- #define P_SIGTERM 1
- #define P_SIGSTOP 2
- #define P_SIGCONT 3

Functions

- pid_t k_proc_create (pcb_t ∗parent, void ∗(∗func)(void ∗), char ∗const argv[ ], priority_t priority)

  Create a new child process.
- int k_proc_cleanup (pcb_t ∗proc)

  Clean up a terminated/finished process's resources. Called internally by the kernel, typically after reaping.
- int k_add_to_ready_queue (pcb_t ∗proc)

  Adds a process to the appropriate scheduler ready queue based on its priority.
- pcb_t ∗ k_get_current_process (void)

  Retrieves the Process Control Block (PCB) of the currently running process.
- pcb_t ∗ k_get_process_by_pid (pid_t pid)

  Finds a process by its PID across all scheduler queues (except zombie).
- int k_block_process (pcb_t ∗process)

  Moves a process from its current ready/running queue to the blocked queue. Does not change the process state field directly, assumes caller manages state.
- int k_unblock_process (pcb_t ∗process)

  Moves a process from the blocked queue to the appropriate ready queue. Does not change the process state field directly, assumes caller manages state.
- int k_proc_exit (pcb_t ∗process, int exit_status)

  Marks a process as terminated (zombie), sets its exit status, moves it to the zombie queue, and potentially unblocks a waiting parent.
- void k_yield (void)

  Voluntarily yields the CPU to the scheduler.
- int k_stop_process (pcb_t ∗process)

  Stops a process, moving it to the stopped queue.
- int k_continue_process (pcb_t ∗process)

  Continues a stopped process, moving it back to the ready queue.
- int k_set_priority (pcb_t ∗process, int priority)

  Sets the priority of a process. If the process is currently ready, it will be moved to the correct ready queue. If blocked or stopped, only the priority field is updated.
- int k_sleep (pcb_t ∗process, unsigned int ticks)

  Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set.
- void k_logout ()

  Sets a flag to indicate that the logout command has been issued and the scheduler should exit. Since this only signals the scheduler to exit, it is not guaranteed to exit immediately.

7.12.1 Macro Definition Documentation

7.12.1.1 P_SIGCONT

```
#define P_SIGCONT 3
```

7.12.1.2 P_SIGSTOP

```
#define P_SIGSTOP 2
```

7.12.1.3 P_SIGTERM

```
#define P_SIGTERM 1
```

7.12.2 Function Documentation

7.12.2.1 k_add_to_ready_queue()

```
int k_add_to_ready_queue (
            pcb_t * process )
```

Adds a process to the appropriate scheduler ready queue based on its priority.

Parameters

proc	The process control block to add.

Returns

0 on success, or an error code, which is a negative integer.

Adds a process to the appropriate scheduler ready queue based on its priority.

Parameters

process	The process PCB to add.

7.12.2.2 k_block_process()

```
int k_block_process (
            pcb_t * process )
```

Moves a process from its current ready/running queue to the blocked queue. Does not change the process state field directly, assumes caller manages state.

Parameters

process	The process to block.

Returns

0 on success, and a negative error code on error

7.12.2.3 k_continue_process()

```
int k_continue_process (
            pcb_t * process )
```

Continues a stopped process, moving it back to the ready queue.

Parameters

process	The process to continue.

Returns

0 on success, and a negative error code on error

Continues a stopped process, moving it back to the ready queue.

Parameters

process	The process to continue.

Returns

0 on success, and a negative error code on error

7.12.2.4 k_get_current_process()

```
pcb_t* k_get_current_process (
            void  )
```

Retrieves the Process Control Block (PCB) of the currently running process.

Returns

pcb_t* Pointer to the current process's PCB, or NULL if no process is running.

pcb_t* Pointer to the current process's PCB, or NULL if the scheduler state is not initialized or no process is currently assigned.

7.12.2.5 k_get_process_by_pid()

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

Finds a process by its PID across all scheduler queues (except zombie).

Parameters

pid	The PID of the process to find.

Returns

pcb_t* Pointer to the PCB if found, NULL otherwise.

Finds a process by its PID across all scheduler queues (except zombie).

Parameters

pid	The PID of the process to find.

Returns

pcb_t* Pointer to the PCB if found, NULL otherwise.

7.12.2.6 k_logout()

```
void k_logout ( )
```

Sets a flag to indicate that the logout command has been issued and the scheduler should exit. Since this only signals the scheduler to exit, it is not guaranteed to exit immediately.

7.12.2.7 k_proc_cleanup()

```
int k_proc_cleanup (
            pcb_t * proc )
```

Clean up a terminated/finished process's resources. Called internally by the kernel, typically after reaping.

Parameters

proc	Pointer to the PCB of the process to clean up.

Returns

0 on success, or an error code, which is a negative integer.

Clean up a terminated/finished process's resources. Called internally by the kernel, typically after reaping.

This function performs the necessary cleanup steps when a process is destroyed:

1. Reparents any orphaned children of the process to the init process (PID 1). It assumes `scheduler_state` and `scheduler_state->init_process` are accessible.
2. Removes the process from its parent's list of children.
3. Frees allocated resources within the PCB, including the thread structure (if any), command string, argv array, and the children list structure.
4. Frees the PCB structure itself.

Note: This function does *not* handle removing the process from scheduler queues (ready, blocked, etc.) or joining the underlying thread; those actions should occur before calling cleanup.

Parameters

proc	Pointer to the PCB of the process to clean up.

7.12.2.8 k_proc_create()

```
pid_t k_proc_create (
            pcb_t * parent,
            void *(*)(void *) func,
            char *const argv[],
            priority_t priority )
```

Create a new child process.

Parameters

parent	The parent process PCB (can be NULL for initial processes).
func	The function the new process should execute.
argv	Null-terminated argument vector for the new process. The kernel will copy this.

Returns

pid_t The PID of the newly created process, or an error code, which is a negative integer.

Create a new child process.

This function handles the complete setup of a new process:

1. Allocates and initializes the PCB.
2. Assigns a unique PID.
3. Sets up parent-child relationship.
4. Allocates and initializes the children list.
5. Sets file descriptors, function pointer.
6. Duplicates the command and arguments (`argv`).
7. Allocates the spthread structure.
8. Creates the underlying spthread using `spthread_create`.
9. Sets the initial state and priority.
10. Adds the process to the parent's children list.
11. Adds the process to the appropriate scheduler ready queue using `k_add_to_ready_queue`.

Parameters

parent	Pointer to the parent process's PCB. If NULL, the process is assumed to be an initial process (like init) with PPID 0.
func	The function the new process should execute.
argv	Null-terminated argument vector for the new process. The kernel copies this.
priority	The priority of the new process.

Returns

pid_t The PID of the newly created process, or -1 if any allocation or thread creation fails.

7.12.2.9 k_proc_exit()

```
int k_proc_exit (
            pcb_t * process,
            int exit_status )
```

Marks a process as terminated (zombie), sets its exit status, moves it to the zombie queue, and potentially unblocks a waiting parent.

Parameters

process	The process that is exiting.
exit_status	The exit status code.

Returns

0 if successful, and a negative error code on failure.

Marks a process as terminated (zombie), sets its exit status, moves it to the zombie queue, and potentially unblocks a waiting parent.

This is the first stage of process termination. The actual cleanup (joining thread, freeing PCB) happens later when the process is reaped by its parent via k_waitpid/k_reap_child. It assumes the process's thread function has already completed or is about to terminate.

Parameters

process	The PCB of the process that is exiting.
exit_status	The exit status code for the process.

Returns

0 on success, and a negative error code on failure.

7.12.2.10 k_set_priority()

```
int k_set_priority (
            pcb_t * process,
            int priority )
```

Sets the priority of a process. If the process is currently ready, it will be moved to the correct ready queue. If blocked or stopped, only the priority field is updated.

Parameters

process	The process to modify.
priority	The new priority (PRIORITY_HIGH, PRIORITY_MEDIUM, PRIORITY_LOW).

Returns

0 on success, and a negative error code on error

7.12.2.11 k_sleep()

```
int k_sleep (
            pcb_t * process,
            unsigned int ticks )
```

Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set.

Parameters

process	The process to put to sleep.
ticks	The number of ticks to sleep (must be $>$ 0).

Returns

0 on success, and a negative error code on error

Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set.

Parameters

process	The process to put to sleep.
ticks	The number of ticks to sleep (must be $>$ 0).

Returns

0 on success, and a negative error code on error

7.12.2.12 k_stop_process()

```
int k_stop_process (
            pcb_t * process )
```

Stops a process, moving it to the stopped queue.

Parameters

process	The process to stop.

Returns

0 on success, and a negative error code on error

Stops a process, moving it to the stopped queue.

Parameters

process	The process to stop.

Returns

0 on success, and a negative error code on error

7.12.2.13 k_unblock_process()

```
int k_unblock_process (
            pcb_t * process )
```

Moves a process from the blocked queue to the appropriate ready queue. Does not change the process state field directly, assumes caller manages state.

Parameters

process	The process to unblock.

Returns

0 on success, and a negative error code on error

7.12.2.14 k_yield()

```
void k_yield (
            void )
```

Voluntarily yields the CPU to the scheduler.

Allows the scheduler to run other processes. The calling process will be paused and resumed later according to the scheduling policy. This is a placeholder implementation relying on signal suspension.

Allows the scheduler to run other processes. The calling process will be paused and resumed later according to the scheduling policy. Placeholder implementation: Suspends the calling thread until the next scheduler timer interrupt (SIGALRM).

7.13 src/scheduler/logger.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include "logger.h"
#include "scheduler.h"
```

Include dependency graph for logger.c:

Macros

- #define LOG_BUF_SIZE 256

Functions

- void init_logger (const char ∗file_path)
- void log_schedule (pid_t pid, int queue, const char ∗process_name)
- void log_create (pid_t pid, int nice_value, const char ∗process_name)
- void log_signaled (pid_t pid, int nice_value, const char ∗process_name)
- void log_exited (pid_t pid, int nice_value, const char ∗process_name)
- void log_zombie (pid_t pid, int nice_value, const char ∗process_name)
- void log_orphan (pid_t pid, int nice_value, const char ∗process_name)
- void log_waited (pid_t pid, int nice_value, const char ∗process_name)

    Log a process that has been waited on.
- void log_nice (pid_t pid, int old_nice, int new_nice, const char ∗process_name)
- void log_blocked (pid_t pid, int nice_value, const char ∗process_name)
- void log_unblocked (pid_t pid, int nice_value, const char ∗process_name)
- void log_sleep (pid_t pid, int nice_value, const char ∗process_name)
- void log_stopped (pid_t pid, int nice_value, const char ∗process_name)
- void log_continued (pid_t pid, int nice_value, const char ∗process_name)
- void log_custom (const char ∗operation, const char ∗format,...)
- void log_message (log_level_t level, const char ∗format,...)

7.13.1 Macro Definition Documentation

7.13.1.1 LOG_BUF_SIZE

```
#define LOG_BUF_SIZE 256
```

7.13.2 Function Documentation

7.13.2.1 init_logger()

```
void init_logger (
             const char * log_file )
```

Initialize the logger

Parameters

log_file	Path to the log file, or NULL to use stderr

7.13.2.2 log_blocked()

```
void log_blocked (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process blocked event Format: [ticks] BLOCKED PID NICE_VALUE PROCESS_NAME

7.13.2.3 log_continued()

```
void log_continued (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process continued event Format: [ticks] CONTINUED PID NICE_VALUE PROCESS_NAME

7.13.2.4 log_create()

```
void log_create (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process creation event Format: [ticks] CREATE PID NICE_VALUE PROCESS_NAME

7.13.2.5 log_custom()

```
void log_custom (
            const char * operation,
            const char * format,
             ... )
```

Log a custom event Format: [ticks] OPERATION args...

Parameters

operation	The operation name to log
format	Format string for the arguments

7.13.2.6 log_exited()

```
void log_exited (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process exit event Format: [ticks] EXITED PID NICE_VALUE PROCESS_NAME

7.13.2.7 log_message()

```
void log_message (
            log_level_t level,
            const char * format,
             ... )
```

7.13.2.8 log_nice()

```
void log_nice (
            pid_t pid,
            int old_nice,
            int new_nice,
            const char * process_name )
```

Log a nice value change event Format: [ticks] NICE PID OLD_NICE_VALUE NEW_NICE_VALUE PROCESS_↩ NAME

7.13.2.9 log_orphan()

```
void log_orphan (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process becoming orphan event Format: [ticks] ORPHAN PID NICE_VALUE PROCESS_NAME

7.13.2.10 log_schedule()

```
void log_schedule (
            pid_t pid,
            int queue,
            const char * process_name )
```

Log a schedule event Format: [ticks] SCHEDULE PID QUEUE PROCESS_NAME

7.13.2.11 log_signaled()

```
void log_signaled (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process signaled event Format: [ticks] SIGNALED PID NICE_VALUE PROCESS_NAME

7.13.2.12 log_sleep()

```
void log_sleep (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a sleep event Format: [ticks] SLEEPING PID NICE_VALUE PROCESS_NAME

7.13.2.13 log_stopped()

```
void log_stopped (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process stopped event Format: [ticks] STOPPED PID NICE_VALUE PROCESS_NAME

7.13.2.14 log_unblocked()

```
void log_unblocked (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process unblocked event Format: [ticks] UNBLOCKED PID NICE_VALUE PROCESS_NAME

7.13.2.15 log_waited()

```
void log_waited (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process that has been waited on.

Parameters

pid	The PID of the process that has been waited on
nice_value	The nice value of the process that has been waited on
process_name	The name of the process that has been waited on

7.13.2.16 log_zombie()

```
void log_zombie (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process becoming zombie event Format: [ticks] ZOMBIE PID NICE_VALUE PROCESS_NAME

7.14 src/scheduler/logger.h File Reference

```
#include <stdlib.h>
#include <unistd.h>
#include <stdarg.h>
```

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

Macros

- #define LOG_DEBUG(...) log_message(LOG_DEBUG, __VA_ARGS__)
- #define LOG_INFO(...) log_message(LOG_INFO, __VA_ARGS__)
- #define LOG_WARN(...) log_message(LOG_WARN, __VA_ARGS__)
- #define LOG_ERROR(...) log_message(LOG_ERROR, __VA_ARGS__)

Enumerations

- enum log_level_t { LOG_DEBUG , LOG_INFO , LOG_WARN , LOG_ERROR }

Functions

- void init_logger (const char ∗log_file)
- void update_ticks (unsigned long ticks)
- void log_schedule (pid_t pid, int queue, const char ∗process_name)
- void log_sleep (pid_t pid, int nice_value, const char ∗process_name)
- void log_create (pid_t pid, int nice_value, const char ∗process_name)
- void log_signaled (pid_t pid, int nice_value, const char ∗process_name)
- void log_exited (pid_t pid, int nice_value, const char ∗process_name)
- void log_zombie (pid_t pid, int nice_value, const char ∗process_name)
- void log_orphan (pid_t pid, int nice_value, const char ∗process_name)
- void log_waited (pid_t pid, int nice_value, const char ∗process_name)

    Log a process that has been waited on.
- void log_nice (pid_t pid, int old_nice, int new_nice, const char ∗process_name)
- void log_blocked (pid_t pid, int nice_value, const char ∗process_name)
- void log_unblocked (pid_t pid, int nice_value, const char ∗process_name)
- void log_stopped (pid_t pid, int nice_value, const char ∗process_name)
- void log_continued (pid_t pid, int nice_value, const char ∗process_name)
- void log_custom (const char ∗operation, const char ∗format,...)
- void log_message (log_level_t level, const char ∗format,...)

7.14.1 Macro Definition Documentation

7.14.1.1 LOG_DEBUG

```
#define LOG_DEBUG(
              ... ) log_message(LOG_DEBUG, __VA_ARGS__)
```

7.14.1.2 LOG_ERROR

```
#define LOG_ERROR(
              ... ) log_message(LOG_ERROR, __VA_ARGS__)
```

7.14.1.3 LOG_INFO

```
#define LOG_INFO(
              ... ) log_message(LOG_INFO, __VA_ARGS__)
```

7.14.1.4 LOG_WARN

```
#define LOG_WARN(
              ... ) log_message(LOG_WARN, __VA_ARGS__)
```

7.14.2 Enumeration Type Documentation

7.14.2.1 log_level_t

```
enum log_level_t
```

Enumerator

LOG_DEBUG	
LOG_INFO	
LOG_WARN	
LOG_ERROR	

7.14.3 Function Documentation

7.14.3.1 init_logger()

```
void init_logger (
            const char * log_file )
```

Initialize the logger

Parameters

log_file	Path to the log file, or NULL to use stderr

7.14.3.2 log_blocked()

```
void log_blocked (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process blocked event Format: [ticks] BLOCKED PID NICE_VALUE PROCESS_NAME

7.14.3.3 log_continued()

```
void log_continued (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process continued event Format: [ticks] CONTINUED PID NICE_VALUE PROCESS_NAME

7.14.3.4 log_create()

```
void log_create (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process creation event Format: [ticks] CREATE PID NICE_VALUE PROCESS_NAME

7.14.3.5 log_custom()

```
void log_custom (
            const char * operation,
            const char * format,
             ... )
```

Log a custom event Format: [ticks] OPERATION args...

Parameters

operation	The operation name to log
format	Format string for the arguments

7.14.3.6 log_exited()

```
void log_exited (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process exit event Format: [ticks] EXITED PID NICE_VALUE PROCESS_NAME

7.14.3.7 log_message()

```
void log_message (
            log_level_t level,
            const char * format,
             ... )
```

7.14.3.8 log_nice()

```
void log_nice (
            pid_t pid,
            int old_nice,
            int new_nice,
            const char * process_name )
```

Log a nice value change event Format: [ticks] NICE PID OLD_NICE_VALUE NEW_NICE_VALUE PROCESS_↩
NAME

7.14.3.9 log_orphan()

```
void log_orphan (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process becoming orphan event Format: [ticks] ORPHAN PID NICE_VALUE PROCESS_NAME

7.14.3.10 log_schedule()

```
void log_schedule (
            pid_t pid,
            int queue,
            const char * process_name )
```

Log a schedule event Format: [ticks] SCHEDULE PID QUEUE PROCESS_NAME

7.14.3.11 log_signaled()

```
void log_signaled (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process signaled event Format: [ticks] SIGNALED PID NICE_VALUE PROCESS_NAME

7.14.3.12 log_sleep()

```
void log_sleep (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a sleep event Format: [ticks] SLEEPING PID NICE_VALUE PROCESS_NAME

7.14.3.13 log_stopped()

```
void log_stopped (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process stopped event Format: [ticks] STOPPED PID NICE_VALUE PROCESS_NAME

7.14.3.14 log_unblocked()

```
void log_unblocked (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process unblocked event Format: [ticks] UNBLOCKED PID NICE_VALUE PROCESS_NAME

7.14.3.15 log_waited()

```
void log_waited (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process that has been waited on.

Log a process being waited on event Format: [ticks] WAITED PID NICE_VALUE PROCESS_NAME

Parameters

pid	The PID of the process that has been waited on
nice_value	The nice value of the process that has been waited on
process_name	The name of the process that has been waited on

7.14.3.16 log_zombie()

```
void log_zombie (
            pid_t pid,
            int nice_value,
            const char * process_name )
```

Log a process becoming zombie event Format: [ticks] ZOMBIE PID NICE_VALUE PROCESS_NAME

7.14.3.17 update_ticks()

```
void update_ticks (
            unsigned long ticks )
```

Update the current tick count

Parameters

ticks	The current tick count

7.15 src/scheduler/pennos.c File Reference

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

Include dependency graph for pennos.c:

Functions

- int main (void)

7.15.1 Function Documentation

7.15.1.1 main()

```
int main (
            void  )
```

7.16 src/scheduler/README.md File Reference

7.17 src/shell/README.md File Reference

7.18 src/scheduler/scheduler.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>
#include "scheduler.h"
#include "logger.h"
#include "../../lib/linked_list.h"
#include "spthread.h"
#include <sys/time.h>
#include "kernel.h"
#include <bits/sigaction.h>
#include <asm-generic/signal-defs.h>
#include <stdbool.h>
#include "src/utils/error_codes.h"
#include <stdio.h>
```

Include dependency graph for scheduler.c:

Macros

- #define W_EXITED 1
- #define W_STOPPED 2

Functions

- void k_log (const char ∗format,...)

  General kernel logger function. Prints messages to stderr only if extra_logging_enabled is true. Uses printf-style formatting.
- void pcb_destructor (void ∗pcb)

  PCB destructor function for linked lists.
- int init_scheduler ()

  Initialize the scheduler.
- int k_add_to_ready_queue (pcb_t ∗process)

  Adds a process to the appropriate ready queue based on its priority. This is a kernel-level function.
- int _select_next_queue (scheduler_t ∗scheduler_state)

  Select the next queue to run a process from.
- void _update_blocked_processes ()

  Update the blocked processes.
- void unblock_parents (pcb_t ∗process)

- void reparent_children (pcb_t ∗process)
- void _run_next_process ()

  Run the next process.
- void run_scheduler ()

  Run the scheduler.
- void block_process (pcb_t ∗process)

  Block a process.
- void unblock_process (pcb_t ∗process)

  Unblock a process.
- pcb_t ∗ k_get_process_by_pid (pid_t pid)

  Finds a process by its PID across active scheduler queues (ready, blocked, stopped) and checks the currently running process. Does NOT search the zombie queue.
- void block_and_wait (scheduler_t ∗scheduler_state, pcb_t ∗process, pcb_t ∗child, int ∗wstatus)

  Block and wait for a child process to finish.
- void remove_from_children_list (pcb_t ∗process, pcb_t ∗child)

  Remove a child from the children list of a process.
- pid_t k_waitpid (pid_t pid, int ∗wstatus, bool nohang)

  Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.
- int k_proc_exit (pcb_t ∗process, int exit_status)

  Marks a process as terminated (zombie), sets its exit status, removes it from active queues, moves it to the zombie queue, and potentially unblocks a waiting parent.
- pcb_t ∗ k_get_current_process (void)

  Retrieves the Process Control Block (PCB) of the currently running process.
- void k_yield (void)

  Voluntarily yields the CPU to the scheduler.
- int k_stop_process (pcb_t ∗process)

  Stops a process, moving it to the stopped queue. Removes the process from active queues and sets its state.
- int k_continue_process (pcb_t ∗process)

  Continues a stopped process, moving it back to the ready queue. Removes the process from the stopped queue and sets its state.
- int k_set_priority (pcb_t ∗process, int priority)

  Sets the priority of a process. If the process is currently ready, it will be moved to the correct ready queue. If blocked or stopped, only the priority field is updated.
- int k_sleep (pcb_t ∗process, unsigned int ticks)

  Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set. Caller should likely call k_yield() after this.
- bool k_resume_sleep (pcb_t ∗process)

  Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set. Caller should likely call k_yield() after this.
- void k_get_processes_from_queue (pcb_ll_t queue)
- int k_print_processes_from_queue (pcb_ll_t queue, char state_char)

  Helper function to print processes from a queue in ps format.
- void k_get_all_process_info ()

  List all processes on PennOS, displaying PID, PPID, priority, status, and command name, similar to `ps`.
- int k_get_quantum ()
- void k_toggle_logging ()

  Toggles the extra logging feature on or off.
- void k_tcsetpid (pid_t pid)
- pid_t k_tcgetpid ()
- void k_logout ()

  Sets a flag to indicate that the logout command has been issued and the scheduler should exit. Since this only signals the scheduler to exit, it is not guaranteed to exit immediately.

Variables

- scheduler_t * scheduler_state = NULL
- bool logout_issued = false

7.18.1 Macro Definition Documentation

7.18.1.1 W_EXITED

```
#define W_EXITED 1
```

7.18.1.2 W_STOPPED

```
#define W_STOPPED 2
```

7.18.2 Function Documentation

7.18.2.1 _run_next_process()

```
void _run_next_process ( )
```

Run the next process.

This function runs the next process from the queue that was selected.

Parameters

scheduler_state	The scheduler state

7.18.2.2 _select_next_queue()

```
int _select_next_queue (
             scheduler_t * scheduler_state )
```

Select the next queue to run a process from.

This function selects the next queue to run a process from.

Parameters

scheduler_state	The scheduler state

7.18.2.3 _update_blocked_processes()

```
void _update_blocked_processes ( )
```

Update the blocked processes.

This function updates the blocked processes by decrementing their sleep time.

Parameters

scheduler_state	The scheduler state

7.18.2.4 block_and_wait()

```
void block_and_wait (
             scheduler_t * scheduler_state,
             pcb_t * process,
             pcb_t * child,
             int * wstatus )
```

Block and wait for a child process to finish.

This function blocks the parent process and waits for the child process to finish.

Parameters

scheduler_state	The scheduler state
process	The parent process
child	The child process
wstatus	The status of the child process

7.18.2.5 block_process()

```
void block_process (
             pcb_t * process )
```

Block a process.

This function blocks a process by removing it from the queue it is currently on and adding it to the blocked queue.

Parameters

process	The process to block

7.18.2.6 init_scheduler()

```
int init_scheduler ( )
```

Initialize the scheduler.

7.18.2.7 k_add_to_ready_queue()

```
int k_add_to_ready_queue (
            pcb_t * process )
```

Adds a process to the appropriate ready queue based on its priority. This is a kernel-level function.

Adds a process to the appropriate scheduler ready queue based on its priority.

Parameters

process	The process PCB to add.

7.18.2.8 k_continue_process()

```
int k_continue_process (
            pcb_t * process )
```

Continues a stopped process, moving it back to the ready queue. Removes the process from the stopped queue and sets its state.

Continues a stopped process, moving it back to the ready queue.

Parameters

process	The process to continue.

Returns

0 on success, and a negative error code on error

7.18.2.9 k_get_all_process_info()

```
void k_get_all_process_info ( )
```

List all processes on PennOS, displaying PID, PPID, priority, status, and command name, similar to `ps`.

Prints only if extra logging i s enabled via k_toggle_logging(). Status codes: R (Running), B (Blocked), S (Stopped), Z (Zombie).

7.18.2.10 k_get_current_process()

```
pcb_t* k_get_current_process (
            void )
```

Retrieves the Process Control Block (PCB) of the currently running process.

Returns

pcb_t* Pointer to the current process's PCB, or NULL if the scheduler state is not initialized or no process is currently assigned.

7.18.2.11 k_get_process_by_pid()

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

Finds a process by its PID across active scheduler queues (ready, blocked, stopped) and checks the currently running process. Does NOT search the zombie queue.

Finds a process by its PID across all scheduler queues (except zombie).

Parameters

pid	The PID of the process to find.

Returns

pcb_t* Pointer to the PCB if found, NULL otherwise.

7.18.2.12 k_get_processes_from_queue()

```
void k_get_processes_from_queue (
            pcb_ll_t queue )
```

7.18.2.13 k_get_quantum()

```
int k_get_quantum ( )
```

7.18.2.14 k_log()

```
void k_log (
            const char * format,
             ... )
```

General kernel logger function. Prints messages to stderr only if extra_logging_enabled is true. Uses printf-style formatting.

Parameters

format	The format string.
...	Variable arguments for the format string.

7.18.2.15 k_logout()

```
void k_logout ( )
```

Sets a flag to indicate that the logout command has been issued and the scheduler should exit. Since this only signals the scheduler to exit, it is not guaranteed to exit immediately.

7.18.2.16 k_print_processes_from_queue()

```
int k_print_processes_from_queue (
            pcb_ll_t queue,
            char state_char )
```

Helper function to print processes from a queue in ps format.

Parameters

queue	The queue to iterate over.
state_char	The character representing the process state ('R', 'B', 'S', 'Z').

7.18.2.17 k_proc_exit()

```
int k_proc_exit (
            pcb_t * process,
            int exit_status )
```

Marks a process as terminated (zombie), sets its exit status, removes it from active queues, moves it to the zombie queue, and potentially unblocks a waiting parent.

Marks a process as terminated (zombie), sets its exit status, moves it to the zombie queue, and potentially unblocks a waiting parent.

This is the first stage of process termination. The actual cleanup (joining thread, freeing PCB) happens later when the process is reaped by its parent via k_waitpid/k_reap_child. It assumes the process's thread function has already completed or is about to terminate.

Parameters

process	The PCB of the process that is exiting.
exit_status	The exit status code for the process.

Returns

0 on success, and a negative error code on failure.

7.18.2.18 k_resume_sleep()

```
bool k_resume_sleep (
            pcb_t * process )
```

Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set. Caller should likely call k_yield() after this.

Parameters

process	The process to put to sleep.
ticks	The number of ticks to sleep (must be $>$ 0).

Returns

true on success, false if process is NULL or sleep time left is 0

7.18.2.19 k_set_priority()

```
int k_set_priority (
            pcb_t * process,
            int priority )
```

Sets the priority of a process. If the process is currently ready, it will be moved to the correct ready queue. If blocked or stopped, only the priority field is updated.

Parameters

process	The process to modify.
priority	The new priority (PRIORITY_HIGH, PRIORITY_MEDIUM, PRIORITY_LOW).

Returns

0 on success, and a negative error code on error

7.18.2.20 k_sleep()

```
int k_sleep (
            pcb_t * process,
            unsigned int ticks )
```

Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set. Caller should likely call k_yield() after this.

Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set.

Parameters

process	The process to put to sleep.
ticks	The number of ticks to sleep (must be $>$ 0).

Returns

0 on success, and a negative error code on error

7.18.2.21 k_stop_process()

```
int k_stop_process (
            pcb_t * process )
```

Stops a process, moving it to the stopped queue. Removes the process from active queues and sets its state.

Stops a process, moving it to the stopped queue.

Parameters

process	The process to stop.

Returns

0 on success, and a negative error code on error

7.18.2.22 k_tcgetpid()

```
pid_t k_tcgetpid ( )
```

7.18.2.23 k_tcsetpid()

```
void k_tcsetpid (
             pid_t pid )
```

7.18.2.24 k_toggle_logging()

```
void k_toggle_logging ( )
```

Toggles the extra logging feature on or off.

7.18.2.25 k_waitpid()

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

Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.

First clean up zombies, then check nohang status, then block and wait if required.

Parameters

pid	Process ID of the child to wait for.
wstatus	Pointer to an integer variable where the status will be stored.
nohang	If true, return immediately if no child has exited.

Returns

pid_t The process ID of the child which has changed state on success, -1 on error.

7.18.2.26 k_yield()

```
void k_yield (
            void )
```

Voluntarily yields the CPU to the scheduler.

Allows the scheduler to run other processes. The calling process will be paused and resumed later according to the scheduling policy. Placeholder implementation: Suspends the calling thread until the next scheduler timer interrupt (SIGALRM).

7.18.2.27 pcb_destructor()

```
void pcb_destructor (
            void * pcb )
```

PCB destructor function for linked lists.

This function is used as a destructor for PCBs in linked lists. It frees the memory allocated for the PCB and its associated resources.

Parameters

pcb	Pointer to the PCB to destroy

7.18.2.28 remove_from_children_list()

```
void remove_from_children_list (
            pcb_t * process,
            pcb_t * child )
```

Remove a child from the children list of a process.

This function searches through the children list of a process and removes a child process from it. It compares the process pointer of each child in the list to find the child process to remove.

Parameters

process	The process to remove the child from
child	The child process to remove

7.18.2.29 reparent_children()

```
void reparent_children (
            pcb_t * process )
```

7.18.2.30 run_scheduler()

```
void run_scheduler ( )
```

Run the scheduler.

This function runs the scheduler.

Parameters

scheduler_state	The scheduler state

7.18.2.31 unblock_parents()

```
void unblock_parents (
            pcb_t * process )
```

7.18.2.32 unblock_process()

```
void unblock_process (
            pcb_t * process )
```

Unblock a process.

This function unblocks a process by removing it from the blocked queue and adding it to the appropriate queue based on its priority.

Parameters

process	The process to unblock

7.18.3 Variable Documentation

7.18.3.1 logout_issued

```
bool logout_issued = false
```

7.18.3.2 scheduler_state

```
scheduler_t* scheduler_state = NULL
```

7.19 src/scheduler/scheduler.h File Reference

```
#include <signal.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <sys/time.h>
#include <sys/types.h>
#include "../../lib/linked_list.h"
#include "./spthread.h"
```

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

Classes

- struct child_process_st
- struct process_fd_entry_st
- struct pcb_st
- struct scheduler

Macros

- #define PROCESS_FD_TABLE_SIZE 128

Typedefs

- typedef struct pcb_st pcb_t
- typedef struct child_process_st child_process_t
- typedef struct process_fd_entry_st process_fd_entry
- typedef struct scheduler scheduler_t

Enumerations

- enum process_state { PROCESS_RUNNING , PROCESS_BLOCKED , PROCESS_STOPPED , PROCESS_ZOMBIED }
- enum priority_t { PRIORITY_HIGH , PRIORITY_MEDIUM , PRIORITY_LOW }

Functions

- typedef linked_list (pcb_t) ∗pcb_ll_t
- typedef linked_list (child_process_t) ∗child_process_ll_t
- int init_scheduler ()

  Initialize the scheduler.
- void pcb_destructor (void ∗pcb)

  PCB destructor function for linked lists.
- void unblock_process (pcb_t ∗process)

  Unblock a process.
- void block_process (pcb_t ∗process)

  Block a process.
- void kill_process (pcb_t ∗process)
- void continue_process (pcb_t ∗process)
- void put_process_to_sleep (pcb_t ∗process, unsigned int ticks)
- void cleanup_zombie_children (pcb_t ∗parent)
- void run_scheduler ()

  Run the scheduler.
- int k_add_to_ready_queue (pcb_t ∗process)

  Adds a process to the appropriate ready queue based on its priority. This is a kernel-level function.
- int k_block_process (pcb_t ∗process)
- int k_unblock_process (pcb_t ∗process)
- int k_proc_exit (pcb_t ∗process, int exit_status)

  Marks a process as terminated (zombie), sets its exit status, removes it from active queues, moves it to the zombie queue, and potentially unblocks a waiting parent.
- void k_yield (void)

  Voluntarily yields the CPU to the scheduler.
- int k_stop_process (pcb_t ∗process)

  Stops a process, moving it to the stopped queue. Removes the process from active queues and sets its state.
- int k_continue_process (pcb_t ∗process)

  Continues a stopped process, moving it back to the ready queue. Removes the process from the stopped queue and sets its state.
- int k_set_priority (pcb_t ∗process, int priority)

  Sets the priority of a process. If the process is currently ready, it will be moved to the correct ready queue. If blocked or stopped, only the priority field is updated.
- int k_sleep (pcb_t ∗process, unsigned int ticks)

  Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set. Caller should likely call k_yield() after this.
- void k_get_processes_from_queue (pcb_ll_t queue)
- void k_get_all_process_info ()

  List all processes on PennOS, displaying PID, PPID, priority, status, and command name, similar to `ps`.
- pcb_t ∗ k_get_current_process (void)

  Retrieves the Process Control Block (PCB) of the currently running process.
- pid_t k_waitpid (pid_t pid, int ∗wstatus, bool nohang)

  Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.
- pcb_t ∗ get_process_by_pid (pid_t pid)
- int k_get_quantum ()
- void k_toggle_logging ()

  Toggles the extra logging feature on or off.
- void k_print_ps_output ()
- void k_log (const char ∗format,...)

General kernel logger function. Prints messages to stderr only if extra_logging_enabled is true. Uses printf-style formatting.

- void k_tcsetpid (pid_t pid)
- pid_t k_tcgetpid ()
- bool k_resume_sleep (pcb_t ∗process)

  Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set. Caller should likely call k_yield() after this.

Variables

- scheduler_t ∗ scheduler_state

7.19.1 Macro Definition Documentation

7.19.1.1 PROCESS_FD_TABLE_SIZE

```
#define PROCESS_FD_TABLE_SIZE 128
```

Process level file descriptor table

7.19.2 Typedef Documentation

7.19.2.1 child_process_t

```
typedef struct child_process_st child_process_t
```

7.19.2.2 pcb_t

```
typedef struct pcb_st pcb_t
```

7.19.2.3 process_fd_entry

```
typedef struct process_fd_entry_st process_fd_entry
```

7.19.2.4 scheduler_t

```
typedef struct scheduler scheduler_t
```

7.19.3 Enumeration Type Documentation

7.19.3.1 priority_t

```
enum priority_t
```

Enumerator

PRIORITY_HIGH	
PRIORITY_MEDIUM	
PRIORITY_LOW	

7.19.3.2 process_state

```
enum process_state
```

Enumerator

PROCESS_RUNNING	
PROCESS_BLOCKED	
PROCESS_STOPPED	
PROCESS_ZOMBIED	

7.19.4 Function Documentation

7.19.4.1 block_process()

```
void block_process (
            pcb_t * process )
```

Block a process.

This function blocks a process by removing it from the queue it is currently on and adding it to the blocked queue.

Parameters

process	The process to block

7.19.4.2 cleanup_zombie_children()

```
void cleanup_zombie_children (
            pcb_t * parent )
```

7.19.4.3 continue_process()

```
void continue_process (
            pcb_t * process )
```

7.19.4.4 get_process_by_pid()

```
pcb_t* get_process_by_pid (
            pid_t pid )
```

7.19.4.5 init_scheduler()

```
int init_scheduler ( )
```

Initialize the scheduler.

7.19.4.6 k_add_to_ready_queue()

```
int k_add_to_ready_queue (
            pcb_t * process )
```

Adds a process to the appropriate ready queue based on its priority. This is a kernel-level function.

Adds a process to the appropriate scheduler ready queue based on its priority.

Parameters

process	The process PCB to add.

7.19.4.7 k_block_process()

```
int k_block_process (
            pcb_t * process )
```

7.19.4.8 k_continue_process()

```
int k_continue_process (
            pcb_t * process )
```

Continues a stopped process, moving it back to the ready queue. Removes the process from the stopped queue and sets its state.

Continues a stopped process, moving it back to the ready queue.

Parameters

process	The process to continue.

Returns

0 on success, and a negative error code on error

7.19.4.9 k_get_all_process_info()

```
void k_get_all_process_info ( )
```

List all processes on PennOS, displaying PID, PPID, priority, status, and command name, similar to `ps`.

Prints only if extra logging i s enabled via k_toggle_logging(). Status codes: R (Running), B (Blocked), S (Stopped), Z (Zombie).

7.19.4.10 k_get_current_process()

```
pcb_t* k_get_current_process (
            void )
```

Retrieves the Process Control Block (PCB) of the currently running process.

Returns

pcb_t* Pointer to the current process's PCB, or NULL if the scheduler state is not initialized or no process is currently assigned.

7.19.4.11 k_get_processes_from_queue()

```
void k_get_processes_from_queue (
            pcb_ll_t queue )
```

7.19.4.12 k_get_quantum()

```
int k_get_quantum ( )
```

7.19.4.13 k_log()

```
void k_log (
            const char * format,
             ...  )
```

General kernel logger function. Prints messages to stderr only if extra_logging_enabled is true. Uses printf-style formatting.

Parameters

format	The format string.
...	Variable arguments for the format string.

7.19.4.14 k_print_ps_output()

```
void k_print_ps_output ( )
```

7.19.4.15 k_proc_exit()

```
int k_proc_exit (
            pcb_t * process,
            int exit_status )
```

Marks a process as terminated (zombie), sets its exit status, removes it from active queues, moves it to the zombie queue, and potentially unblocks a waiting parent.

Marks a process as terminated (zombie), sets its exit status, moves it to the zombie queue, and potentially unblocks a waiting parent.

This is the first stage of process termination. The actual cleanup (joining thread, freeing PCB) happens later when the process is reaped by its parent via k_waitpid/k_reap_child. It assumes the process's thread function has already completed or is about to terminate.

Parameters

process	The PCB of the process that is exiting.
exit_status	The exit status code for the process.

Returns

0 on success, and a negative error code on failure.

7.19.4.16 k_resume_sleep()

```
bool k_resume_sleep (
            pcb_t * process )
```

Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set. Caller should likely call k_yield() after this.

Parameters

process	The process to put to sleep.
ticks	The number of ticks to sleep (must be $>$ 0).

Returns

true on success, false if process is NULL or sleep time left is 0

7.19.4.17 k_set_priority()

```
int k_set_priority (
            pcb_t * process,
            int priority )
```

Sets the priority of a process. If the process is currently ready, it will be moved to the correct ready queue. If blocked or stopped, only the priority field is updated.

Parameters

process	The process to modify.
priority	The new priority (PRIORITY_HIGH, PRIORITY_MEDIUM, PRIORITY_LOW).

Returns

0 on success, and a negative error code on error

7.19.4.18 k_sleep()

```
int k_sleep (
            pcb_t * process,
            unsigned int ticks )
```

Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set. Caller should likely call k_yield() after this.

Puts the calling process to sleep for a specified number of ticks. The process is blocked, and sleep_time is set.

Parameters

process	The process to put to sleep.
ticks	The number of ticks to sleep (must be $>$ 0).

Returns

0 on success, and a negative error code on error

7.19.4.19 k_stop_process()

```
int k_stop_process (
            pcb_t * process )
```

Stops a process, moving it to the stopped queue. Removes the process from active queues and sets its state.

Stops a process, moving it to the stopped queue.

Parameters

process	The process to stop.

Returns

0 on success, and a negative error code on error

7.19.4.20 k_tcgetpid()

```
pid_t k_tcgetpid ( )
```

7.19.4.21 k_tcsetpid()

```
void k_tcsetpid (
            pid_t pid )
```

7.19.4.22 k_toggle_logging()

```
void k_toggle_logging ( )
```

Toggles the extra logging feature on or off.

7.19.4.23 k_unblock_process()

```
int k_unblock_process (
            pcb_t * process )
```

7.19.4.24 k_waitpid()

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

Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.

First clean up zombies, then check nohang status, then block and wait if required.

Parameters

pid	Process ID of the child to wait for.
wstatus	Pointer to an integer variable where the status will be stored.
nohang	If true, return immediately if no child has exited.

Returns

pid_t The process ID of the child which has changed state on success, -1 on error.

7.19.4.25 k_yield()

```
void k_yield (
            void )
```

Voluntarily yields the CPU to the scheduler.

Allows the scheduler to run other processes. The calling process will be paused and resumed later according to the scheduling policy. Placeholder implementation: Suspends the calling thread until the next scheduler timer interrupt (SIGALRM).

7.19.4.26 kill_process()

```
void kill_process (
            pcb_t * process )
```

7.19.4.27 linked_list() [1/2]

```
typedef linked_list (
            child_process_t  )
```

7.19.4.28 linked_list() [2/2]

```
typedef linked_list (
            pcb_t  )
```

7.19.4.29 pcb_destructor()

```
void pcb_destructor (
            void * pcb )
```

PCB destructor function for linked lists.

This function is used as a destructor for PCBs in linked lists. It frees the memory allocated for the PCB and its associated resources.

Parameters

pcb	Pointer to the PCB to destroy

7.19.4.30 put_process_to_sleep()

```
void put_process_to_sleep (
            pcb_t * process,
            unsigned int ticks )
```

7.19.4.31 run_scheduler()

```
void run_scheduler ( )
```

Run the scheduler.

This function runs the scheduler.

Parameters

scheduler_state	The scheduler state

7.19.4.32 unblock_process()

```
void unblock_process (
            pcb_t * process )
```

Unblock a process.

This function unblocks a process by removing it from the blocked queue and adding it to the appropriate queue based on its priority.

Parameters

process	The process to unblock

7.19.5 Variable Documentation

7.19.5.1 scheduler_state

```
scheduler_t* scheduler_state  [extern]
```

7.20 src/scheduler/spthread.c File Reference

```
#include <errno.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>
#include <stdlib.h>
#include "./spthread.h"
#include <string.h>
#include <stdio.h>
```

Include dependency graph for spthread.c:

Classes

- struct spthread_fwd_args_st
- struct spthread_signal_args_st
- struct spthread_meta_st

Macros

- #define _GNU_SOURCE
- #define MILISEC_IN_NANO 100000
- #define SPTHREAD_RUNNING_STATE 0
- #define SPTHREAD_SUSPENDED_STATE 1
- #define SPTHREAD_TERMINATED_STATE 2
- #define P_WIFEXITED(status) ((status) & 0xFF)
- #define P_WIFSIGNALED(status) (((status) & 0x7F) != 0)
- #define P_WIFSTOPPED(status) (((status) & 0xFF) == 0x7F)
- #define SPTHREAD_SIG_SUSPEND -1
- #define SPTHREAD_SIG_CONTINUE -2

Typedefs

- typedef void ∗(∗ pthread_fn) (void ∗)
- typedef struct spthread_fwd_args_st spthread_fwd_args
- typedef struct spthread_signal_args_st spthread_signal_args
- typedef struct spthread_meta_st spthread_meta_t

Functions

- int spthread_create (spthread_t ∗thread, const pthread_attr_t ∗attr, pthread_fn start_routine, void ∗arg)
- int spthread_suspend (spthread_t thread)
- int spthread_suspend_self ()
- int spthread_continue (spthread_t thread)
- int spthread_cancel (spthread_t thread)
- bool spthread_self (spthread_t ∗thread)
- int spthread_join (spthread_t thread, void ∗∗retval)
- void spthread_exit (void ∗status)
- bool spthread_equal (spthread_t first, spthread_t second)
- int spthread_disable_interrupts_self ()
- int spthread_enable_interrupts_self ()

7.20.1 Macro Definition Documentation

7.20.1.1 _GNU_SOURCE

```
#define _GNU_SOURCE
```

7.20.1.2 MILISEC_IN_NANO

```
#define MILISEC_IN_NANO 100000
```

7.20.1.3 P_WIFEXITED

```
#define P_WIFEXITED(
              status ) ((status) & 0xFF)
```

7.20.1.4 P_WIFSIGNALED

```
#define P_WIFSIGNALED(
              status ) (((status) & 0x7F) != 0)
```

7.20.1.5 P_WIFSTOPPED

```
#define P_WIFSTOPPED(
              status ) (((status) & 0xFF) == 0x7F)
```

7.20.1.6 SPTHREAD_RUNNING_STATE

```
#define SPTHREAD_RUNNING_STATE 0
```

7.20.1.7 SPTHREAD_SIG_CONTINUE

```
#define SPTHREAD_SIG_CONTINUE -2
```

7.20.1.8 SPTHREAD_SIG_SUSPEND

```
#define SPTHREAD_SIG_SUSPEND -1
```

7.20.1.9 SPTHREAD_SUSPENDED_STATE

```
#define SPTHREAD_SUSPENDED_STATE 1
```

7.20.1.10 SPTHREAD_TERMINATED_STATE

```
#define SPTHREAD_TERMINATED_STATE 2
```

7.20.2 Typedef Documentation

7.20.2.1 pthread_fn

```
typedef void*(* pthread_fn) (void *)
```

7.20.2.2 spthread_fwd_args

```
typedef struct spthread_fwd_args_st spthread_fwd_args
```

7.20.2.3 spthread_meta_t

```
typedef struct spthread_meta_st spthread_meta_t
```

7.20.2.4 spthread_signal_args

```
typedef struct spthread_signal_args_st spthread_signal_args
```

7.20.3 Function Documentation

7.20.3.1 spthread_cancel()

```
int spthread_cancel (
            spthread_t thread )
```

7.20.3.2 spthread_continue()

```
int spthread_continue (
            spthread_t thread )
```

7.20.3.3 spthread_create()

```
int spthread_create (
            spthread_t * thread,
            const pthread_attr_t * attr,
            pthread_fn start_routine,
            void * arg )
```

7.20.3.4 spthread_disable_interrupts_self()

```
int spthread_disable_interrupts_self ( )
```

7.20.3.5 spthread_enable_interrupts_self()

```
int spthread_enable_interrupts_self ( )
```

7.20.3.6 spthread_equal()

```
bool spthread_equal (
            spthread_t first,
            spthread_t second )
```

7.20.3.7 spthread_exit()

```
void spthread_exit (
            void * status )
```

7.20.3.8 spthread_join()

```
int spthread_join (
            spthread_t thread,
            void ** retval )
```

7.20.3.9 spthread_self()

```
bool spthread_self (
            spthread_t * thread )
```

7.20.3.10 spthread_suspend()

```
int spthread_suspend (
            spthread_t thread )
```

7.20.3.11 spthread_suspend_self()

```
int spthread_suspend_self ( )
```

7.21 src/utils/spthread.c File Reference

```
#include <errno.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>
#include <stdlib.h>
#include "./spthread.h"
#include <string.h>
#include <stdio.h>
```

Include dependency graph for spthread.c:

Classes

- struct spthread_fwd_args_st
- struct spthread_signal_args_st
- struct spthread_meta_st

Macros

- #define _GNU_SOURCE
- #define MILISEC_IN_NANO 100000
- #define SPTHREAD_RUNNING_STATE 0
- #define SPTHREAD_SUSPENDED_STATE 1
- #define SPTHREAD_TERMINATED_STATE 2
- #define SPTHREAD_SIG_SUSPEND -1
- #define SPTHREAD_SIG_CONTINUE -2

Typedefs

- typedef void *(* pthread_fn) (void *)
- typedef struct spthread_fwd_args_st spthread_fwd_args
- typedef struct spthread_signal_args_st spthread_signal_args
- typedef struct spthread_meta_st spthread_meta_t

Functions

- int spthread_create (spthread_t *thread, const pthread_attr_t *attr, pthread_fn start_routine, void *arg)
- int spthread_suspend (spthread_t thread)
- int spthread_suspend_self ()
- int spthread_continue (spthread_t thread)
- int spthread_cancel (spthread_t thread)
- bool spthread_self (spthread_t *thread)
- int spthread_join (spthread_t thread, void **retval)
- void spthread_exit (void *status)
- bool spthread_equal (spthread_t first, spthread_t second)
- int spthread_disable_interrupts_self ()
- int spthread_enable_interrupts_self ()

7.21.1 Macro Definition Documentation

7.21.1.1 _GNU_SOURCE

```
#define _GNU_SOURCE
```

7.21.1.2 MILISEC_IN_NANO

```
#define MILISEC_IN_NANO 100000
```

7.21.1.3 SPTHREAD_RUNNING_STATE

```
#define SPTHREAD_RUNNING_STATE 0
```

7.21.1.4 SPTHREAD_SIG_CONTINUE

```
#define SPTHREAD_SIG_CONTINUE -2
```

7.21.1.5 SPTHREAD_SIG_SUSPEND

```
#define SPTHREAD_SIG_SUSPEND -1
```

7.21.1.6 SPTHREAD_SUSPENDED_STATE

```
#define SPTHREAD_SUSPENDED_STATE 1
```

7.21.1.7 SPTHREAD_TERMINATED_STATE

```
#define SPTHREAD_TERMINATED_STATE 2
```

7.21.2 Typedef Documentation

7.21.2.1 pthread_fn

```
typedef void*(* pthread_fn) (void *)
```

7.21.2.2 spthread_fwd_args

```
typedef struct spthread_fwd_args_st spthread_fwd_args
```

7.21.2.3 spthread_meta_t

```
typedef struct spthread_meta_st spthread_meta_t
```

7.21.2.4 spthread_signal_args

```
typedef struct spthread_signal_args_st spthread_signal_args
```

7.21.3 Function Documentation

7.21.3.1 spthread_cancel()

```
int spthread_cancel (
            spthread_t thread )
```

7.21.3.2 spthread_continue()

```
int spthread_continue (
            spthread_t thread )
```

7.21.3.3 spthread_create()

```
int spthread_create (
            spthread_t * thread,
            const pthread_attr_t * attr,
            pthread_fn start_routine,
            void * arg )
```

7.21.3.4 spthread_disable_interrupts_self()

```
int spthread_disable_interrupts_self ( )
```

7.21.3.5 spthread_enable_interrupts_self()

```
int spthread_enable_interrupts_self ( )
```

7.21.3.6 spthread_equal()

```
bool spthread_equal (
             spthread_t first,
             spthread_t second )
```

7.21.3.7 spthread_exit()

```
void spthread_exit (
             void * status )
```

7.21.3.8 spthread_join()

```
int spthread_join (
             spthread_t thread,
             void ** retval )
```

7.21.3.9 spthread_self()

```
bool spthread_self (
             spthread_t * thread )
```

7.21.3.10 spthread_suspend()

```
int spthread_suspend (
             spthread_t thread )
```

7.21.3.11 spthread_suspend_self()

```
int spthread_suspend_self ( )
```

7.22 src/scheduler/spthread.h File Reference

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

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

Classes

- struct spthread_st

Macros

- #define SIGPTHD SIGUSR1

Typedefs

- typedef struct spthread_meta_st spthread_meta_t
- typedef struct spthread_st spthread_t

Functions

- int spthread_create (spthread_t ∗thread, const pthread_attr_t ∗attr, void ∗(∗start_routine)(void ∗), void ∗arg)
- int spthread_suspend (spthread_t thread)
- int spthread_suspend_self ()
- int spthread_continue (spthread_t thread)
- int spthread_cancel (spthread_t thread)
- bool spthread_self (spthread_t ∗thread)
- int spthread_join (spthread_t thread, void ∗∗retval)
- void spthread_exit (void ∗status)
- bool spthread_equal (spthread_t first, spthread_t second)
- int spthread_disable_interrupts_self ()
- int spthread_enable_interrupts_self ()

7.22.1 Macro Definition Documentation

7.22.1.1 SIGPTHD

```
#define SIGPTHD SIGUSR1
```

7.22.2 Typedef Documentation

7.22.2.1 spthread_meta_t

```
typedef struct spthread_meta_st spthread_meta_t
```

7.22.2.2 spthread_t

```
typedef struct spthread_st spthread_t
```

7.22.3 Function Documentation

7.22.3.1 spthread_cancel()

```
int spthread_cancel (
             spthread_t thread )
```

7.22.3.2 spthread_continue()

```
int spthread_continue (
             spthread_t thread )
```

7.22.3.3 spthread_create()

```
int spthread_create (
             spthread_t * thread,
             const pthread_attr_t * attr,
             void *(*)(void *) start_routine,
             void * arg )
```

7.22.3.4 spthread_disable_interrupts_self()

```
int spthread_disable_interrupts_self ( )
```

7.22.3.5 spthread_enable_interrupts_self()

```
int spthread_enable_interrupts_self ( )
```

7.22.3.6 spthread_equal()

```
bool spthread_equal (
             spthread_t first,
             spthread_t second )
```

7.22.3.7 spthread_exit()

```
void spthread_exit (
             void * status )
```

7.22.3.8 spthread_join()

```
int spthread_join (
             spthread_t thread,
             void ** retval )
```

7.22.3.9 spthread_self()

```
bool spthread_self (
             spthread_t * thread )
```

7.22.3.10 spthread_suspend()

```
int spthread_suspend (
             spthread_t thread )
```

7.22.3.11 spthread_suspend_self()

```
int spthread_suspend_self ( )
```

7.23 src/utils/spthread.h File Reference

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

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

Classes

- struct spthread_st

Macros

- #define SIGPTHD SIGUSR1

Typedefs

- typedef struct spthread_meta_st spthread_meta_t
- typedef struct spthread_st spthread_t

Functions

- int spthread_create (spthread_t ∗thread, const pthread_attr_t ∗attr, void ∗(∗start_routine)(void ∗), void ∗arg)
- int spthread_suspend (spthread_t thread)
- int spthread_suspend_self ()
- int spthread_continue (spthread_t thread)
- int spthread_cancel (spthread_t thread)
- bool spthread_self (spthread_t ∗thread)
- int spthread_join (spthread_t thread, void ∗∗retval)
- void spthread_exit (void ∗status)
- bool spthread_equal (spthread_t first, spthread_t second)
- int spthread_disable_interrupts_self ()
- int spthread_enable_interrupts_self ()

7.23.1 Macro Definition Documentation

7.23.1.1 SIGPTHD

```
#define SIGPTHD SIGUSR1
```

7.23.2 Typedef Documentation

7.23.2.1 spthread_meta_t

```
typedef struct spthread_meta_st spthread_meta_t
```

7.23.2.2 spthread_t

```
typedef struct spthread_st spthread_t
```

7.23.3 Function Documentation

7.23.3.1 spthread_cancel()

```
int spthread_cancel (
            spthread_t thread )
```

7.23.3.2 spthread_continue()

```
int spthread_continue (
            spthread_t thread )
```

7.23.3.3 spthread_create()

```
int spthread_create (
            spthread_t * thread,
            const pthread_attr_t * attr,
            void *(*)(void *) start_routine,
            void * arg )
```

7.23.3.4 spthread_disable_interrupts_self()

```
int spthread_disable_interrupts_self ( )
```

7.23.3.5 spthread_enable_interrupts_self()

```
int spthread_enable_interrupts_self ( )
```

7.23.3.6 spthread_equal()

```
bool spthread_equal (
             spthread_t first,
             spthread_t second )
```

7.23.3.7 spthread_exit()

```
void spthread_exit (
             void * status )
```

7.23.3.8 spthread_join()

```
int spthread_join (
             spthread_t thread,
             void ** retval )
```

7.23.3.9 spthread_self()

```
bool spthread_self (
             spthread_t * thread )
```

7.23.3.10 spthread_suspend()

```
int spthread_suspend (
             spthread_t thread )
```

7.23.3.11 spthread_suspend_self()

```
int spthread_suspend_self ( )
```

7.24 src/scheduler/spthread_demo.c File Reference

```
#include <signal.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>
#include "./spthread.h"
```

Include dependency graph for spthread_demo.c:

Macros

- #define NUM_THREADS 4
- #define BUF_SIZE 4096

Functions

- void cancel_and_join (spthread_t thread)
- int main (void)

7.24.1 Macro Definition Documentation

7.24.1.1 BUF_SIZE

```
#define BUF_SIZE 4096
```

7.24.1.2 NUM_THREADS

```
#define NUM_THREADS 4
```

7.24.2 Function Documentation

7.24.2.1 cancel_and_join()

```
void cancel_and_join (
          spthread_t thread )
```

7.24.2.2 main()

```
int main (
            void  )
```

7.25 src/scheduler/sys.c File Reference

```
#include "src/scheduler/sys.h"
#include "scheduler.h"
#include "kernel.h"
#include "logger.h"
#include "../../lib/exiting_alloc.h"
#include "../../lib/linked_list.h"
#include "spthread.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include "src/utils/error_codes.h"
```

Include dependency graph for sys.c:

Functions

- void run_scheduler ()

  Run the scheduler.
- pid_t s_spawn (void ∗(∗func)(void ∗), char ∗argv[ ], int fd0, int fd1, priority_t priority)

  Create a child process that executes the function `func`. The child will retain some attributes of the parent.
- bool P_WIFEXITED (int wstatus)
- bool P_WIFSTOPPED (int wstatus)
- bool P_WIFSIGNALED (int wstatus)
- pid_t s_waitpid (pid_t pid, int ∗wstatus, bool nohang)

  Wait on a child of the calling process, until it changes state (zombies). If `nohang` is true, this will not block the calling process and return immediately.
- int s_kill (pid_t pid, int signal)

  Send a signal to a particular process. Current implementation supports SIGTERM, SIGSTOP, SIGCONT.
- void s_exit (int status)

  Unconditionally exit the calling process with the given status. This function does not return.
- int s_nice (pid_t pid, int priority)

  Set the priority of the specified process.
- int s_sleep (unsigned int ticks)

  Suspends execution of the calling process for a specified number of clock ticks.
- void s_get_process_info ()

  Get information about all processes. Implements `ps`.
- int s_tcsetpid (pid_t pid)

  Set the terminal controlling process.
- int s_ignore_sigint (bool ignore)

  Ignore or unignore SIGINT signals for the current process.
- int s_ignore_sigtstp (bool ignore)

  Ignore or unignore SIGTSTP signals for the current process.

- void s_logout ()

  Sets a flag to indicate that the logout command has been issued and the scheduler should exit. Since this only signals the scheduler to exit, it is not guaranteed to exit immediately.
- pcb_t ∗ s_get_current_process ()

  Get the current process.
- void s_set_errno (int errno)

  Set the errno of the current process.
- int s_get_errno ()

  Get the error number for the current process.

7.25.1 Function Documentation

7.25.1.1 P_WIFEXITED()

```
bool P_WIFEXITED (
             int wstatus )
```

7.25.1.2 P_WIFSIGNALED()

```
bool P_WIFSIGNALED (
             int wstatus )
```

7.25.1.3 P_WIFSTOPPED()

```
bool P_WIFSTOPPED (
             int wstatus )
```

7.25.1.4 run_scheduler()

```
void run_scheduler ( )
```

Run the scheduler.

This function runs the scheduler.

Parameters

scheduler_state	The scheduler state

7.25.1.5 s_exit()

```
void s_exit (
            int status )
```

Unconditionally exit the calling process with the given status. This function does not return.

Parameters

status	The exit status code.

7.25.1.6 s_get_current_process()

```
pcb_t* s_get_current_process ( )
```

Get the current process.

Returns

pcb_t∗ The current process.

7.25.1.7 s_get_errno()

```
int s_get_errno ( )
```

Get the error number for the current process.

Returns

int The error number.

7.25.1.8 s_get_process_info()

```
void s_get_process_info ( )
```

Get information about all processes. Implements `ps`.

Get the process info of the current process and print it to stderr.

This function retrieves and prints detailed information about all processes, including their PID, PPID, priority, and state.

7.25.1.9 s_ignore_sigint()

```
int s_ignore_sigint (
            bool ignore )
```

Ignore or unignore SIGINT signals for the current process.

Parameters

ignore	true to ignore, false to unignore

Returns

int 0 on success, or negative error code

7.25.1.10 s_ignore_sigtstp()

```
int s_ignore_sigtstp (
            bool ignore )
```

Ignore or unignore SIGTSTP signals for the current process.

Parameters

ignore	true to ignore, false to unignore

Returns

int 0 on success, or negative error code

7.25.1.11 s_kill()

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

Send a signal to a particular process. Current implementation supports SIGTERM, SIGSTOP, SIGCONT.

Send a signal to a particular process.

Parameters

pid	Process ID of the target proces.
signal	Signal number to be sent.

Returns

0 on success, -1 on error (e.g., process not found, invalid signal).

7.25.1.12 s_logout()

```
void s_logout ( )
```

Sets a flag to indicate that the logout command has been issued and the scheduler should exit. Since this only signals the scheduler to exit, it is not guaranteed to exit immediately.

7.25.1.13 s_nice()

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

Set the priority of the specified process.

Set the priority of the specified thread.

Parameters

pid	Process ID of the target process.
priority	The new priority value of the process (0, 1, or 2).

Returns

0 on success, -1 on failure (e.g., process not found, invalid priority).

7.25.1.14 s_set_errno()

```
void s_set_errno (
            int errno )
```

Set the errno of the current process.

Parameters

errno	The errno to set.

7.25.1.15 s_sleep()

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

Suspends execution of the calling process for a specified number of clock ticks.

Suspends execution of the calling proces for a specified number of clock ticks.

Parameters

ticks	Duration of the sleep in system clock ticks. Must be greater than 0.

Returns

0 on success, -1 on error (e.g., invalid ticks, no current process).

7.25.1.16 s_spawn()

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

Create a child process that executes the function `func`. The child will retain some attributes of the parent.

Parameters

func	Function to be executed by the child process.
arg	Argument to be passed to the function.

Returns

pid_t The process ID of the created child process, or -1 on error.

7.25.1.17 s_tcsetpid()

```
int s_tcsetpid (
            pid_t pid )
```

Set the terminal controlling process.

Parameters

pid	process id of the new terminal controlling process

Returns

int 0 on success, or negative error code

Note

If the current process is not the terminal controlling process, this will fail. Note also that the s_waitpid function automatically passes terminal control to the child process when it is waited on with nohang set to false and pid set to the pid of the child process. Thus it is expected that this function will rarely need to be called explicitly. Terminaly control is also returned to the parent process on stop or termination.

7.25.1.18 s_waitpid()

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

Wait on a child of the calling process, until it changes state (zombies). If `nohang` is true, this will not block the calling process and return immediately.

Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.

First clean up zombies, then check nohang status, then block and wait if required.

Parameters

pid	Process ID of the child to wait for (-1 for any child).
wstatus	Pointer to an integer variable where the exit status will be stored.
nohang	If true, return immediately if no child has zombied.

Returns

pid_t The process ID of the zombied child on success, 0 if nohang and no child zombied, -1 on error.

7.26 src/scheduler/sys.h File Reference

```
#include "scheduler.h"
#include "logger.h"
#include "../../lib/linked_list.h"
#include "spthread.h"
#include "src/scheduler/fat_syscalls.h"
```
Include dependency graph for sys.h: This graph shows which files directly or indirectly include this file:

Macros

- #define P_SIGTERM 1
- #define P_SIGSTOP 2
- #define P_SIGCONT 3
- #define P_SIGINT 4
- #define P_SIGTSTP 5
- #define S_SPAWN_INVALID_FD_ERROR -100
- #define W_EXITED 1
- #define W_STOPPED 2

Functions

- pid_t s_spawn (void ∗(∗func)(void ∗), char ∗argv[ ], int fd0, int fd1, priority_t priority)

  Create a child process that executes the function `func`. The child will retain some attributes of the parent.
- bool P_WIFEXITED (int wstatus)
- bool P_WIFSTOPPED (int wstatus)
- bool P_WIFSIGNALED (int wstatus)
- pid_t s_waitpid (pid_t pid, int ∗wstatus, bool nohang)

  Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.
- int s_kill (pid_t pid, int signal)

  Send a signal to a particular process.
- void s_exit (int status)

  Unconditionally exit the calling process with the given status. This function does not return.
- int s_nice (pid_t pid, int priority)

  Set the priority of the specified thread.
- int s_sleep (unsigned int ticks)

  Suspends execution of the calling proces for a specified number of clock ticks.
- void s_get_process_info ()

  Get the process info of the current process and print it to stderr.
- int s_tcsetpid (pid_t pid)

  Set the terminal controlling process.
- int s_ignore_sigint (bool ignore)

  Ignore or unignore SIGINT signals for the current process.
- int s_ignore_sigtstp (bool ignore)

  Ignore or unignore SIGTSTP signals for the current process.
- void s_logout ()

  Sets a flag to indicate that the logout command has been issued and the scheduler should exit. Since this only signals the scheduler to exit, it is not guaranteed to exit immediately.
- pcb_t ∗ s_get_current_process ()

  Get the current process.
- void s_set_errno (int errno)

  Set the errno of the current process.
- int s_get_errno ()

  Get the error number for the current process.

7.26.1 Macro Definition Documentation

7.26.1.1 P_SIGCONT

```
#define P_SIGCONT 3
```

7.26.1.2 P_SIGINT

```
#define P_SIGINT 4
```

7.26.1.3 P_SIGSTOP

```
#define P_SIGSTOP 2
```

7.26.1.4 P_SIGTERM

```
#define P_SIGTERM 1
```

7.26.1.5 P_SIGTSTP

```
#define P_SIGTSTP 5
```

7.26.1.6 S_SPAWN_INVALID_FD_ERROR

```
#define S_SPAWN_INVALID_FD_ERROR -100
```

7.26.1.7 W_EXITED

```
#define W_EXITED 1
```

7.26.1.8 W_STOPPED

```
#define W_STOPPED 2
```

7.26.2 Function Documentation

7.26.2.1 P_WIFEXITED()

```
bool P_WIFEXITED (
             int wstatus )
```

7.26.2.2 P_WIFSIGNALED()

```
bool P_WIFSIGNALED (
             int wstatus )
```

7.26.2.3 P_WIFSTOPPED()

```
bool P_WIFSTOPPED (
             int wstatus )
```

7.26.2.4 s_exit()

```
void s_exit (
             int status )
```

Unconditionally exit the calling process with the given status. This function does not return.

Parameters

status	The exit status code.

7.26.2.5 s_get_current_process()

```
pcb_t* s_get_current_process ( )
```

Get the current process.

Returns

pcb_t∗ The current process.

7.26.2.6 s_get_errno()

```
int s_get_errno ( )
```

Get the error number for the current process.

Returns

int The error number.

7.26.2.7 s_get_process_info()

```
void s_get_process_info ( )
```

Get the process info of the current process and print it to stderr.

Get the process info of the current process and print it to stderr.

This function retrieves and prints detailed information about all processes, including their PID, PPID, priority, and state.

7.26.2.8 s_ignore_sigint()

```
int s_ignore_sigint (
             bool ignore )
```

Ignore or unignore SIGINT signals for the current process.

Parameters

ignore	true to ignore, false to unignore

Returns

int 0 on success, or negative error code

7.26.2.9 s_ignore_sigtstp()

```
int s_ignore_sigtstp (
             bool ignore )
```

Ignore or unignore SIGTSTP signals for the current process.

Parameters

ignore	true to ignore, false to unignore

Returns

int 0 on success, or negative error code

7.26.2.10 s_kill()

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

Send a signal to a particular process.

Parameters

pid	Process ID of the target proces.
signal	Signal number to be sent.

Returns

0 on success, -1 on error.

Send a signal to a particular process.

Parameters

pid	Process ID of the target proces.
signal	Signal number to be sent.

Returns

0 on success, -1 on error (e.g., process not found, invalid signal).

7.26.2.11 s_logout()

```
void s_logout ( )
```

Sets a flag to indicate that the logout command has been issued and the scheduler should exit. Since this only signals the scheduler to exit, it is not guaranteed to exit immediately.

7.26.2.12 s_nice()

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

Set the priority of the specified thread.

Parameters

pid	Process ID of the target thread.
priority	The new priorty value of the thread (0, 1, or 2)

Returns

0 on success, -1 on failure.

Set the priority of the specified thread.

Parameters

pid	Process ID of the target process.
priority	The new priority value of the process (0, 1, or 2).

Returns

0 on success, -1 on failure (e.g., process not found, invalid priority).

7.26.2.13 s_set_errno()

```
void s_set_errno (
             int errno )
```

Set the errno of the current process.

Parameters

errno	The errno to set.

7.26.2.14 s_sleep()

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

Suspends execution of the calling proces for a specified number of clock ticks.

This function is analogous to `sleep(3)` in Linux, with the behavior that the system clock continues to tick even if the call is interrupted. The sleep can be interrupted by a P_SIGTERM signal, after which the function will return prematurely.

Parameters

ticks	Duration of the sleep in system clock ticks. Must be greater than 0.

Suspends execution of the calling proces for a specified number of clock ticks.

Parameters

ticks	Duration of the sleep in system clock ticks. Must be greater than 0.

Returns

0 on success, -1 on error (e.g., invalid ticks, no current process).

7.26.2.15 s_spawn()

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

Create a child process that executes the function `func`. The child will retain some attributes of the parent.

Parameters

func	Function to be executed by the child process.
argv	Null-terminated array of args, including the command name as argv[0].
fd0	Input file descriptor.
fd1	Output file descriptor.

Returns

pid_t The process ID of the created child process.

Parameters

func	Function to be executed by the child process.
arg	Argument to be passed to the function.

Returns

pid_t The process ID of the created child process, or -1 on error.

7.26.2.16 s_tcsetpid()

```
int s_tcsetpid (
            pid_t pid )
```

Set the terminal controlling process.

Parameters

pid	process id of the new terminal controlling process

Returns

int 0 on success, or negative error code

Note

If the current process is not the terminal controlling process, this will fail. Note also that the s_waitpid function automatically passes terminal control to the child process when it is waited on with nohang set to false and pid set to the pid of the child process. Thus it is expected that this function will rarely need to be called explicitly. Terminaly control is also returned to the parent process on stop or termination.

7.26.2.17 s_waitpid()

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

Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.

Parameters

pid	Process ID of the child to wait for.
wstatus	Pointer to an integer variable where the status will be stored.
nohang	If true, return immediately if no child has exited.

Returns

pid_t The process ID of the child which has changed state on success, -1 on error.

Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.

First clean up zombies, then check nohang status, then block and wait if required.

Parameters

pid	Process ID of the child to wait for (-1 for any child).
wstatus	Pointer to an integer variable where the exit status will be stored.
nohang	If true, return immediately if no child has zombied.

Returns

pid_t The process ID of the zombied child on success, 0 if nohang and no child zombied, -1 on error.

7.27 src/shell/command_execution.c File Reference

```
#include "./command_execution.h"
#include <errno.h>
#include <signal.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <wait.h>
#include "./exiting_signal.h"
#include <sys/stat.h>
#include <fcntl.h>
#include "./Job.h"
#include <assert.h>
#include "./valid_input.h"
#include "./signals.h"
#include "./jobs.h"
#include "commands.h"
#include "../scheduler/sys.h"
#include "../../lib/exiting_alloc.h"
#include <string.h>
```

Include dependency graph for command_execution.c:

Macros

- #define DEFAULT_FILE_PERMISSIONS 0644
- #define FORK_SETUP_DELAY_USEC 500

Functions

- void execute_job (job ∗job)

  A higher level function that manages the execution of the stages of a command.

Variables

- pid_t current_pid

7.27.1 Macro Definition Documentation

7.27.1.1 DEFAULT_FILE_PERMISSIONS

```
#define DEFAULT_FILE_PERMISSIONS 0644
```

7.27.1.2 FORK_SETUP_DELAY_USEC

```
#define FORK_SETUP_DELAY_USEC 500
```

7.27.2 Function Documentation

7.27.2.1 execute_job()

```
void execute_job (
            job * job )
```

A higher level function that manages the execution of the stages of a command.

This function expects that it is only called if the caller thinks there is something to execute (not checking for cases like the num_commands being 0).

7.27.3 Variable Documentation

7.27.3.1 current_pid

```
pid_t current_pid
```

Execute the "lead" child, which executes all the other commands.

7.28 src/shell/command_execution.h File Reference

```
#include "./parser.h"
#include "./Job.h"
```

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

Functions

- void execute_job (job *job)

  A higher level function that manages the execution of the stages of a command.

Variables

- pid_t current_pid

7.28.1 Function Documentation

7.28.1.1 execute_job()

```
void execute_job (
            job * job )
```

A higher level function that manages the execution of the stages of a command.

This function expects that it is only called if the caller thinks there is something to execute (not checking for cases like the num_commands being 0).

7.28.2 Variable Documentation

7.28.2.1 current_pid

```
pid_t current_pid [extern]
```

Execute the "lead" child, which executes all the other commands.

7.29 src/shell/commands.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include "commands.h"
#include "../scheduler/scheduler.h"
#include "../scheduler/logger.h"
#include "../scheduler/sys.h"
#include "../../lib/exiting_alloc.h"
#include "../scheduler/spthread.h"
#include "src/pennfat/fat_constants.h"
#include "jobs.h"
#include "stress.h"
#include "src/utils/errno.h"
```

Include dependency graph for commands.c:

Macros

- #define BUFFER_SIZE 256

Functions

- void ∗ ps (void ∗arg)
- void ∗ zombie_child (void ∗arg)
- void ∗ zombify (void ∗arg)
- void ∗ orphan_child (void ∗arg)
- void ∗ orphanify (void ∗arg)
- void ∗ busy (void ∗arg, char ∗priority)

    This function is used to busy wait a process.
- void ∗ nice_pid_command (void ∗arg, char ∗pid, char ∗priority)
- void ∗ nice_command (void ∗arg, char ∗priority)
- void ∗ sleep_command (void ∗arg, char ∗time)
- void ∗ kill_process_shell (void ∗arg, char ∗first_term)
- void ∗ man (void ∗arg)
- void ∗ jobs_command (void ∗arg)
- void ∗ ls (void ∗arg)
- void ∗ echo (void ∗arg)
- void ∗ touch (void ∗arg)
- void ∗ rm (void ∗arg)
- void ∗ cp (void ∗arg)
- void ∗ cat (void ∗arg)
- void ∗ chmod (void ∗arg)
- void ∗ mv (void ∗arg)
- void ∗ hang_helper (void ∗arg)
- void ∗ logout (void ∗arg)
- void ∗ execute_command (void ∗arg)

7.29.1 Macro Definition Documentation

7.29.1.1 BUFFER_SIZE

```
#define BUFFER_SIZE 256
```

7.29.2 Function Documentation

7.29.2.1 busy()

```
void* busy (
             void * arg,
             char * priority )
```

This function is used to busy wait a process.

It removes the current process from the ready queue and adds it to the ready queue with the new priority. It then busy waits.

Parameters

arg	The command context.
priority	The priority to set the process to.

Returns

NULL.

7.29.2.2 cat()

```
void* cat (
            void * arg )
```

7.29.2.3 chmod()

```
void* chmod (
            void * arg )
```

7.29.2.4 cp()

```
void* cp (
            void * arg )
```

7.29.2.5 echo()

```
void* echo (
            void * arg )
```

7.29.2.6 execute_command()

```
void* execute_command (
            void * arg )
```

7.29.2.7 hang_helper()

```
void* hang_helper (
            void * arg )
```

7.29.2.8 jobs_command()

```
void* jobs_command (
            void * arg )
```

7.29.2.9 kill_process_shell()

```
void* kill_process_shell (
            void * arg,
            char * first_term )
```

7.29.2.10 logout()

```
void* logout (
            void * arg )
```

7.29.2.11 ls()

```
void* ls (
            void * arg )
```

7.29.2.12 man()

```
void* man (
            void * arg )
```

7.29.2.13 mv()

```
void* mv (
            void * arg )
```

7.29.2.14 nice_command()

```
void* nice_command (
            void * arg,
            char * priority )
```

7.29.2.15 nice_pid_command()

```
void* nice_pid_command (
            void * arg,
            char * pid,
            char * priority )
```

7.29.2.16 orphan_child()

```
void* orphan_child (
            void * arg )
```

7.29.2.17 orphanify()

```
void* orphanify (
            void * arg )
```

7.29.2.18 ps()

```
void* ps (
            void * arg )
```

7.29.2.19 rm()

```
void* rm (
            void * arg )
```

7.29.2.20 sleep_command()

```
void* sleep_command (
             void * arg,
             char * time )
```

7.29.2.21 touch()

```
void* touch (
             void * arg )
```

7.29.2.22 zombie_child()

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

7.29.2.23 zombify()

```
void* zombify (
             void * arg )
```

7.30 src/shell/commands.h File Reference

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

Classes

- struct command_context

Functions

- void ∗ execute_command (void ∗arg)

7.30.1 Function Documentation

7.30.1.1 execute_command()

```
void* execute_command (
            void * arg )
```

7.31 src/shell/exiting_signal.c File Reference

```
#include "./exiting_signal.h"
#include <signal.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
```

Include dependency graph for exiting_signal.c:

Functions

- void exiting_set_signal_handler (int sig, void(∗handler)(int))

7.31.1 Function Documentation

7.31.1.1 exiting_set_signal_handler()

```
void exiting_set_signal_handler (
            int sig,
            void(*)(int) handler )
```

7.32 src/shell/exiting_signal.h File Reference

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

Functions

- void exiting_set_signal_handler (int sig, void(∗handler)(int))

7.32.1 Function Documentation

7.32.1.1 exiting_set_signal_handler()

```
void exiting_set_signal_handler (
            int sig,
            void(*)(int) handler )
```

7.33 src/shell/Job.c File Reference

```
#include "./Job.h"
#include "./parser.h"
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for Job.c:

7.34 src/shell/Job.h File Reference

```
#include <stdint.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "./parser.h"
```

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

Classes

- struct job_st

Typedefs

- typedef uint64_t jid_t
- typedef enum job_status_enum job_status
- typedef struct job_st job

Enumerations

- enum job_status_enum { J_RUNNING_FG = 0 , J_RUNNING_BG = 1 , J_STOPPED = 2 }

Functions

- void destroy_job (job ∗job)

  Frees the job, also destroying its fields as needed (ie, the `cmd` field). Note that it is assumed that the job contains the only reference to the fields within it (specifically, to the `cmd` field of type `parsed_command∗`), thus this function will free the fields in the job struct as necessary.

7.34.1 Typedef Documentation

7.34.1.1 jid_t

```
typedef uint64_t jid_t
```

7.34.1.2 job

```
typedef struct job_st job
```

7.34.1.3 job_status

```
typedef enum job_status_enum job_status
```

7.34.2 Enumeration Type Documentation

7.34.2.1 job_status_enum

```
enum job_status_enum
```

Enumerator

J_RUNNING_FG	
J_RUNNING_BG	
J_STOPPED	

7.34.3 Function Documentation

7.34.3.1 destroy_job()

```
void destroy_job (
            job * job )
```

Frees the job, also destroying its fields as needed (ie, the `cmd` field). Note that it is assumed that the job contains the only reference to the fields within it (specifically, to the `cmd` field of type `parsed_command*`), thus this function will free the fields in the job struct as necessary.

7.35 src/shell/jobs.c File Reference

```
#include "./Job.h"
#include "./jobs.h"
#include "../../lib/linked_list.h"
#include <stdlib.h>
#include <string.h>
#include "./command_execution.h"
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include "../scheduler/sys.h"
#include "../scheduler/logger.h"
#include "../scheduler/fat_syscalls.h"
```

Include dependency graph for jobs.c:

Functions

- typedef linked_list (job_ll_node)

  The global linked list storing background and stopped jobs.
- void print_job_command (job ∗job)

  Prints the command line representation of a job to stderr.
- void handle_jobs ()

  Handles the "jobs" built-in command.
- void print_all_jobs ()

  Prints all jobs in the list along with their first PID to stderr.
- void handle_fg (struct parsed_command ∗cmd)

  Handles the "fg" built-in command.
- void handle_bg (struct parsed_command ∗cmd)

  Handles the "bg" built-in command.
- bool handle_jobs_commands (struct parsed_command ∗cmd)

  Determines if a parsed command is a job control built-in (jobs, fg, bg) and handles it if it is.
- void enqueue_job (job ∗job)

  Adds a job to the end of the background/stopped jobs list.
- job ∗ find_job_by_id (jid_t id)

  Finds a job in the list by its job ID.
- void continue_job (job ∗job)

  Sets a job's status to J_RUNNING_FG and executes it.
- job ∗ find_job_by_pid (pid_t pid)

  Finds a job in the list by its process group ID (or first PID).
- void print_job_list ()

  Prints the command for every job in the list.
- void remove_job_by_pid (pid_t pid)

  Removes a job from the list based on its process group ID (or first PID) and cleans up its resources.
- void add_foreground_job (job ∗job)

  Adds a job node representing the current foreground job to the head of the list.
- void remove_foreground_job (job ∗job)

  Removes a specific job pointer from the jobs list.
- job ∗ get_jobs_head ()

  Gets the job struct from the head of the jobs list.

7.35.1 Function Documentation

7.35.1.1 add_foreground_job()

```
void add_foreground_job (
            job * job )
```

Adds a job node representing the current foreground job to the head of the list.

This seems to be used to temporarily track the foreground job, potentially for signal handling purposes or easy access via get_jobs_head().

Parameters

job	The job currently running in the foreground.

7.35.1.2 continue_job()

```
void continue_job (
            job * job )
```

Sets a job's status to J_RUNNING_FG and executes it.

Note: This function seems intended to continue a stopped job in the foreground, but might need integration with terminal control transfer.

Parameters

job	The job to continue.

7.35.1.3 enqueue_job()

```
void enqueue_job (
            job * job )
```

Adds a job to the end of the background/stopped jobs list.

Adds a (background/stopped) job to the queue.

This function is typically called for jobs launched in the background or for foreground jobs that have been stopped. It performs error checking to ensure only background or stopped jobs are enqueued.

Parameters

job	The job to add to the list.

7.35.1.4 find_job_by_id()

```
job* find_job_by_id (
            jid_t id )
```

Finds a job in the list by its job ID.

Parameters

id	The job ID to search for.

Returns

A pointer to the job if found, NULL otherwise.

7.35.1.5 find_job_by_pid()

```
job* find_job_by_pid (
            pid_t pid )
```

Finds a job in the list by its process group ID (or first PID).

Iterates through the list, comparing the given PID with the first PID stored in each job.

Parameters

pid	The process ID (expected to be the process group ID) to search for.

Returns

A pointer to the job if found, NULL otherwise.

7.35.1.6 get_jobs_head()

```
job* get_jobs_head ( )
```

Gets the job struct from the head of the jobs list.

Useful for accessing the most recently added job (e.g., the current foreground job if add_foreground_job was just called).

Returns

A pointer to the job at the head of the list, or NULL if the list is empty.

7.35.1.7 handle_bg()

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

Handles the "bg" built-in command.

Resumes the specified stopped job (by ID) or the most recently stopped job in the background. It finds the job, changes its status to J_RUNNING_BG, and sends SIGCONT. The job remains in the jobs list.

Parameters

cmd	The parsed command structure, potentially containing the job ID.

7.35.1.8 handle_fg()

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

Handles the "fg" built-in command.

Brings the specified job (by ID) or the most recently added job from the background/stopped list to the foreground. It removes the job from the list, gives it terminal control, sends SIGCONT if stopped, and waits for it.

Parameters

cmd	The parsed command structure, potentially containing the job ID.

7.35.1.9 handle_jobs()

```
void handle_jobs ( )
```

Handles the "jobs" built-in command.

Iterates through the global jobs list and prints information (job ID and command) for each job to stderr using s_↩
fprintf_short.

7.35.1.10 handle_jobs_commands()

```
bool handle_jobs_commands (
            struct parsed_command * cmd )
```

Determines if a parsed command is a job control built-in (jobs, fg, bg) and handles it if it is.

Check if `cmd` matches any of `bg`, `fg`, `jobs`, and if so handle it.

Parameters

cmd	The parsed command structure.

Returns

true if the command was a job control built-in and was handled, false otherwise.

7.35.1.11 linked_list()

```
typedef linked_list (
            job_ll_node )
```

The global linked list storing background and stopped jobs.

7.35.1.12 print_all_jobs()

```
void print_all_jobs ( )
```

Prints all jobs in the list along with their first PID to stderr.

Note: This function seems primarily for debugging and uses fprintf directly.

7.35.1.13 print_job_command()

```
void print_job_command (
            job * job )
```

Prints the command line representation of a job to stderr.

Iterates through the parsed command structure within the job and prints each command and argument, separated by pipes if necessary.

Parameters

job	A pointer to the job whose command should be printed.

7.35.1.14 print_job_list()

```
void print_job_list ( )
```

Prints the command for every job in the list.

Note: Marked with TODO. Its intended use compared to handle_jobs() is unclear.

Todo Clarify the purpose of this function.

7.35.1.15 remove_foreground_job()

```
void remove_foreground_job (
            job * job )
```

Removes a specific job pointer from the jobs list.

Searches the list for a node containing the exact job pointer provided and removes that node. Note: The comment suggests this is used for removing stopped jobs, implying a potential misnomer.

Parameters

job	The pointer to the job struct whose corresponding node should be removed.

Todo Rename this function to better reflect its actual usage (removing specific stopped jobs?).

7.35.1.16 remove_job_by_pid()

```
void remove_job_by_pid (
            pid_t pid )
```

Removes a job from the list based on its process group ID (or first PID) and cleans up its resources.

Finds the job node by PID, removes it from the linked list, prints a completion message, destroys the job struct, and frees the list node.

Parameters

pid	The process ID (expected to be the process group ID) of the job to remove.

7.36 src/shell/jobs.h File Reference

```
#include "./Job.h"
#include <stdlib.h>
#include <stdbool.h>
```
Include dependency graph for jobs.h: This graph shows which files directly or indirectly include this file:

Classes

- struct job_ll_node_st

Typedefs

- typedef struct job_ll_node_st job_ll_node

Functions

- bool handle_jobs_commands (struct parsed_command ∗cmd)

  Check if `cmd` matches any of `bg`, `fg`, `jobs`, and if so handle it.
- void handle_jobs ()

  Handles the "jobs" built-in command.
- void enqueue_job (job ∗job)

  Adds a (background/stopped) job to the queue.
- void print_all_jobs ()

  Prints all jobs in the list along with their first PID to stderr.
- job ∗ find_job_by_id (jid_t id)

  Finds a job in the list by its job ID.
- void continue_job (job ∗job)

  Sets a job's status to J_RUNNING_FG and executes it.
- job ∗ find_job_by_pid (pid_t pid)

  Finds a job in the list by its process group ID (or first PID).
- void print_job_list ()

  Prints the command for every job in the list.
- void print_job_command (job ∗job)

  Prints the command line representation of a job to stderr.
- void remove_job_by_pid (pid_t pid)

  Removes a job from the list based on its process group ID (or first PID) and cleans up its resources.
- void add_foreground_job (job ∗job)

  Adds a job node representing the current foreground job to the head of the list.
- void remove_foreground_job (job ∗job)

  Removes a specific job pointer from the jobs list.
- job ∗ get_jobs_head ()

  Gets the job struct from the head of the jobs list.

7.36.1 Typedef Documentation

7.36.1.1 job_ll_node

```
typedef struct job_ll_node_st job_ll_node
```

7.36.2 Function Documentation

7.36.2.1 add_foreground_job()

```
void add_foreground_job (
             job * job )
```

Adds a job node representing the current foreground job to the head of the list.

This seems to be used to temporarily track the foreground job, potentially for signal handling purposes or easy access via get_jobs_head().

Parameters

job	The job currently running in the foreground.

7.36.2.2 continue_job()

```
void continue_job (
             job * job )
```

Sets a job's status to J_RUNNING_FG and executes it.

Note: This function seems intended to continue a stopped job in the foreground, but might need integration with terminal control transfer.

Parameters

job	The job to continue.

7.36.2.3 enqueue_job()

```
void enqueue_job (
             job * job )
```

Adds a (background/stopped) job to the queue.

Note that if the job is not background/stopped, this function will throw an error.

Adds a (background/stopped) job to the queue.

This function is typically called for jobs launched in the background or for foreground jobs that have been stopped. It performs error checking to ensure only background or stopped jobs are enqueued.

Parameters

job	The job to add to the list.

7.36.2.4 find_job_by_id()

```
job* find_job_by_id (
            jid_t id )
```

Finds a job in the list by its job ID.

Parameters

id	The job ID to search for.

Returns

A pointer to the job if found, NULL otherwise.

7.36.2.5 find_job_by_pid()

```
job* find_job_by_pid (
            pid_t pid )
```

Finds a job in the list by its process group ID (or first PID).

Iterates through the list, comparing the given PID with the first PID stored in each job.

Parameters

pid	The process ID (expected to be the process group ID) to search for.

Returns

A pointer to the job if found, NULL otherwise.

7.36.2.6 get_jobs_head()

```
job* get_jobs_head ( )
```

Gets the job struct from the head of the jobs list.

Useful for accessing the most recently added job (e.g., the current foreground job if add_foreground_job was just called).

Returns

A pointer to the job at the head of the list, or NULL if the list is empty.

7.36.2.7 handle_jobs()

```
void handle_jobs ( )
```

Handles the "jobs" built-in command.

Iterates through the global jobs list and prints information (job ID and command) for each job to stderr using s_↩
fprintf_short.

7.36.2.8 handle_jobs_commands()

```
bool handle_jobs_commands (
             struct parsed_command * cmd )
```

Check if `cmd` matches any of `bg`, `fg`, `jobs`, and if so handle it.

Returns

A boolean that indicates whether the cmd matched bg, fg, or jobs and therefore if this function actually handled the passed command.

It is generally expected that if this function returns true, the caller will not have to do further work on the given `cmd`.

Check if `cmd` matches any of `bg`, `fg`, `jobs`, and if so handle it.

Parameters

cmd	The parsed command structure.

Returns

true if the command was a job control built-in and was handled, false otherwise.

7.36.2.9 print_all_jobs()

```
void print_all_jobs ( )
```

Prints all jobs in the list along with their first PID to stderr.

Note: This function seems primarily for debugging and uses fprintf directly.

7.36.2.10 print_job_command()

```
void print_job_command (
            job * job )
```

Prints the command line representation of a job to stderr.

Iterates through the parsed command structure within the job and prints each command and argument, separated by pipes if necessary.

Parameters

job	A pointer to the job whose command should be printed.

7.36.2.11 print_job_list()

```
void print_job_list ( )
```

Prints the command for every job in the list.

Note: Marked with TODO. Its intended use compared to handle_jobs() is unclear.

Todo Clarify the purpose of this function.

7.36.2.12 remove_foreground_job()

```
void remove_foreground_job (
            job * job )
```

Removes a specific job pointer from the jobs list.

Searches the list for a node containing the exact job pointer provided and removes that node. Note: The comment suggests this is used for removing stopped jobs, implying a potential misnomer.

Parameters

job	The pointer to the job struct whose corresponding node should be removed.

Todo Rename this function to better reflect its actual usage (removing specific stopped jobs?).

7.36.2.13 remove_job_by_pid()

```
void remove_job_by_pid (
            pid_t pid )
```

Removes a job from the list based on its process group ID (or first PID) and cleans up its resources.

Finds the job node by PID, removes it from the linked list, prints a completion message, destroys the job struct, and frees the list node.

Parameters

pid	The process ID (expected to be the process group ID) of the job to remove.

7.37 src/shell/parser.c File Reference

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

Include dependency graph for parser.c:

Macros

- #define JUMP_OUT(code) do {ret_code = code; goto PROCESS_ERROR;} while (0)

Functions

- int parse_command (const char ∗const cmd_line, struct parsed_command ∗∗const result)
- void print_parsed_command (const struct parsed_command ∗const cmd)
- void print_parser_errcode (FILE ∗output, int err_code)

7.37.1 Macro Definition Documentation

7.37.1.1 JUMP_OUT

```
#define JUMP_OUT(
              code ) do {ret_code = code; goto PROCESS_ERROR;} while (0)
```

7.37.2 Function Documentation

7.37.2.1 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 guareenteed 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. layout of memory for `struct parsed_command` bool is_background; bool is_file_append;

const char *stdin_file; const char *stdout_file;

size_t num_commands;

commands are pointers to `arguments` char **commands[num_commands];

below are hidden in memory **

arguments are pointers to `original_string + num_commands` because all argv are null-terminated char *arguments[total_strings + num_commands];

original_string is a copy of the cmdline but with each token null-terminated char *original_string;

7.37.2.2 print_parsed_command()

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

7.37.2.3 print_parser_errcode()

```
void print_parser_errcode (
            FILE * output,
            int err_code )
```

7.38 src/shell/parser.h File Reference

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

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

Classes

- struct parsed_command

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)
- void print_parser_errcode (FILE ∗output, int err_code)

7.38.1 Macro Definition Documentation

7.38.1.1 EXPECT_COMMANDS

```
#define EXPECT_COMMANDS 7
```

7.38.1.2 EXPECT_INPUT_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

7.38.1.3 EXPECT_OUTPUT_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

7.38.1.4 UNEXPECTED_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

7.38.1.5 UNEXPECTED_FILE_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

7.38.1.6 UNEXPECTED_FILE_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

7.38.1.7 UNEXPECTED_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

7.38.2 Function Documentation

7.38.2.1 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 guareenteed 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. layout of memory for `struct parsed_command` bool is_background; bool is_file_append;

const char *stdin_file; const char *stdout_file;

size_t num_commands;

commands are pointers to `arguments` char **commands[num_commands];

below are hidden in memory **

arguments are pointers to `original_string + num_commands` because all argv are null-terminated char *arguments[total_strings + num_commands];

original_string is a copy of the cmdline but with each token null-terminated char *original_string;

7.38.2.2 print_parsed_command()

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

7.38.2.3 print_parser_errcode()

```
void print_parser_errcode (
            FILE * output,
            int err_code )
```

7.39 src/shell/penn-shell.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include "./parser.h"
#include "../scheduler/logger.h"
#include "./command_execution.h"
#include "./shell_porcelain.h"
#include "./Job.h"
#include "./jobs.h"
#include "./signals.h"
#include "../scheduler/scheduler.h"
#include "../scheduler/sys.h"
#include "commands.h"
#include "src/pennfat/fat.h"
#include <string.h>
#include "src/scheduler/fat_syscalls.h"
```

Include dependency graph for penn-shell.c:

Functions

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

Variables

- jid_t job_id = 0

7.39.1 Function Documentation

7.39.1.1 main()

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

7.39.2 Variable Documentation

7.39.2.1 job_id

```
jid_t job_id = 0
```

7.40 src/shell/shell_porcelain.c File Reference

```
#include "./shell_porcelain.h"
#include <assert.h>
#include <signal.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include "src/utils/errno.h"
#include "../../lib/exiting_alloc.h"
#include "./exiting_signal.h"
#include "./command_execution.h"
#include "./parser.h"
#include "src/scheduler/sys.h"
```

Include dependency graph for shell_porcelain.c:

Macros

- #define PROMPT "$ "
- #define CATCHPHRASE "Welcome to Penn Shell!"

Enumerations

- enum { MAX_LINE_LENGTH = 4096 }

Functions

- int read_command (struct parsed_command **parsed_command)
- void display_prompt ()

7.40.1 Macro Definition Documentation

7.40.1.1 CATCHPHRASE

```
#define CATCHPHRASE "Welcome to Penn Shell!"
```

7.40.1.2 PROMPT

```
#define PROMPT "$ "
```

7.40.2 Enumeration Type Documentation

7.40.2.1 anonymous enum

```
anonymous enum
```

Enumerator

MAX_LINE_LENGTH	

7.40.3 Function Documentation

7.40.3.1 display_prompt()

```
void display_prompt ( )
```

7.40.3.2 read_command()

```
int read_command (
             struct parsed_command ** parsed_command )
```

7.41 src/shell/shell_porcelain.h File Reference

```
#include "./parser.h"
```

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

Functions

- int read_command (struct parsed_command **parsed_command)
- void display_prompt ()
- void display_catchphrase ()

7.41.1 Function Documentation

7.41.1.1 display_catchphrase()

```
void display_catchphrase ( )
```

7.41.1.2 display_prompt()

```
void display_prompt ( )
```

7.41.1.3 read_command()

```
int read_command (
            struct parsed_command ** parsed_command )
```

7.42 src/shell/signals.c File Reference

```
#include "./signals.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <stdbool.h>
#include <signal.h>
#include "./Job.h"
#include "./jobs.h"
#include "../../lib/linked_list.h"
#include "./command_execution.h"
#include "../scheduler/sys.h"
#include <bits/sigaction.h>
#include <asm-generic/signal-defs.h>
```

Include dependency graph for signals.c:

Functions

- void pennos_signal_handler (int sig)

  Handler for job control signals (SIGINT and SIGTSTP) When a signal is received and there's a foreground job, stops the job and adds it to the background jobs list.
- int setup_job_control_handlers (void)

  Sets up handlers for SIGINT (Ctrl-C) and SIGTSTP (Ctrl-Z)
- void ignore_signals (void)

  Ignores signals in the shell.
- void ignore_sigint (void)

  Ignores SIGINT and SIGTSTP in the shell process.

7.42.1 Function Documentation

7.42.1.1 ignore_sigint()

```
void ignore_sigint (
             void  )
```

Ignores SIGINT and SIGTSTP in the shell process.

7.42.1.2 ignore_signals()

```
void ignore_signals (
             void  )
```

Ignores signals in the shell.

7.42.1.3 pennos_signal_handler()

```
void pennos_signal_handler (
             int sig )
```

Handler for job control signals (SIGINT and SIGTSTP) When a signal is received and there's a foreground job, stops the job and adds it to the background jobs list.

Signal handler for PennOS to accept host OS signals and forward them appropriately.

7.42.1.4 setup_job_control_handlers()

```
int setup_job_control_handlers (
             void  )
```

Sets up handlers for SIGINT (Ctrl-C) and SIGTSTP (Ctrl-Z)

Returns

0 on success, -1 on error

7.43 src/shell/signals.h File Reference

```
#include <signal.h>
#include <sys/types.h>
```

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

Macros

- #define CHILD_STOPPED_EXIT_STATUS 99

Functions

- void stop_handler (int sig)

    Handler for SIGSTOP signals When a SIGALRM is received and there's a child process, sends SIGSTOP to that child process.
- int setup_alarm_handler (void)

    Sets up the SIGALRM handler.
- void pennos_signal_handler (int sig)

    Handler for job control signals (SIGINT and SIGTSTP) When a signal is received and there's a foreground job, stops the job and adds it to the background jobs list.
- int setup_job_control_handlers (void)

    Sets up handlers for SIGINT (Ctrl-C) and SIGTSTP (Ctrl-Z)
- void ignore_signals (void)

    Ignores signals in the shell.
- void ignore_sigint (void)

    Ignores SIGINT and SIGTSTP in the shell process.

Variables

- pid_t shell_pgid

7.43.1 Macro Definition Documentation

7.43.1.1 CHILD_STOPPED_EXIT_STATUS

```
#define CHILD_STOPPED_EXIT_STATUS 99
```

7.43.2 Function Documentation

7.43.2.1 ignore_sigint()

```
void ignore_sigint (
             void  )
```

Ignores SIGINT and SIGTSTP in the shell process.

7.43.2.2 ignore_signals()

```
void ignore_signals (
             void  )
```

Ignores signals in the shell.

7.43.2.3 pennos_signal_handler()

```
void pennos_signal_handler (
             int sig )
```

Handler for job control signals (SIGINT and SIGTSTP) When a signal is received and there's a foreground job, stops the job and adds it to the background jobs list.

Signal handler for PennOS to accept host OS signals and forward them appropriately.

7.43.2.4 setup_alarm_handler()

```
int setup_alarm_handler (
             void  )
```

Sets up the SIGALRM handler.

Returns

0 on success, -1 on error

7.43.2.5 setup_job_control_handlers()

```
int setup_job_control_handlers (
             void  )
```

Sets up handlers for SIGINT (Ctrl-C) and SIGTSTP (Ctrl-Z)

Returns

0 on success, -1 on error

7.43.2.6 stop_handler()

```
void stop_handler (
             int sig )
```

Handler for SIGSTOP signals When a SIGALRM is received and there's a child process, sends SIGSTOP to that child process.

7.43.3 Variable Documentation

7.43.3.1 shell_pgid

```
pid_t shell_pgid  [extern]
```

7.44 src/shell/stress.c File Reference

```
#include "stress.h"
#include <stdbool.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <time.h>
#include "src/scheduler/sys.h"
```

Include dependency graph for stress.c:

Functions

- void ∗ hang (void ∗arg)
- void ∗ nohang (void ∗arg)
- void ∗ recur (void ∗arg)
- void ∗ crash (void ∗arg)

7.44.1 Function Documentation

7.44.1.1 crash()

```
void* crash (
            void * arg )
```

7.44.1.2 hang()

```
void* hang (
            void * arg )
```

7.44.1.3 nohang()

```
void* nohang (
            void * arg )
```

7.44.1.4 recur()

```
void* recur (
             void * arg )
```

7.45 src/shell/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 ∗)
- void ∗ crash (void ∗)

7.45.1 Function Documentation

7.45.1.1 crash()

```
void* crash (
             void * arg )
```

7.45.1.2 hang()

```
void* hang (
             void * arg )
```

7.45.1.3 nohang()

```
void* nohang (
             void * arg )
```

7.45.1.4 recur()

```
void* recur (
             void * arg )
```

7.46 src/shell/valid_input.c File Reference

```
#include "./command_execution.h"
#include <errno.h>
#include <signal.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for valid_input.c:

Functions

- void validate_command (struct parsed_command *parsed_command)

7.46.1 Function Documentation

7.46.1.1 validate_command()

```
void validate_command (
            struct parsed_command * parsed_command )
```

7.47 src/shell/valid_input.h File Reference

```
#include "./parser.h"
```

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

Functions

- void validate_command (struct parsed_command *parsed_command)

7.47.1 Function Documentation

7.47.1.1 validate_command()

```
void validate_command (
            struct parsed_command * parsed_command )
```

7.48 src/utils/errno.c File Reference

```
#include "src/utils/errno.h"
#include "src/scheduler/sys.h"
#include <string.h>
```

Include dependency graph for errno.c:

Functions

- char ∗ u_strerror (int errno)

    Convert an error code to a string.
- void u_perror (const char ∗s)

    Print an error message to stderr

Variables

- char err_message [101]

7.48.1 Function Documentation

7.48.1.1 u_perror()

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

Print an error message to stderr

Parameters

s	error message

7.48.1.2 u_strerror()

```
char* u_strerror (
             int errno )
```

Convert an error code to a string.

Parameters

errno	error code

Returns

char∗ string representation of the error code. Note that this is a static string allocated to an internal buffer. It should not be freed nor modified, and should be copied if the caller needs to keep it.

7.48.2 Variable Documentation

7.48.2.1 err_message

```
char err_message[101]
```

7.49 src/utils/errno.h File Reference

```
#include "src/utils/error_codes.h"
```

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

Functions

- char ∗ u_strerror (int errno)
  Convert an error code to a string.
- void u_perror (const char ∗s)
  Print an error message to stderr

7.49.1 Function Documentation

7.49.1.1 u_perror()

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

Print an error message to stderr

Parameters

s	error message

7.49.1.2 u_strerror()

```
char* u_strerror (
            int errno )
```

Convert an error code to a string.

Parameters

errno	error code

Returns

char∗ string representation of the error code. Note that this is a static string allocated to an internal buffer. It should not be freed nor modified, and should be copied if the caller needs to keep it.

7.50 src/utils/error_codes.h File Reference

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

Macros

- #define E_NO_INIT_PROCESS -1
- #define E_INIT_ALREADY_EXISTS -2
- #define E_FAILED_TO_ALLOCATE -3
- #define E_BAD_ARGV -4
- #define E_STR_FORMAT_FAILED -5
- #define E_STR_TOO_LONG_FOR_FPRINTF_BUF -6
- #define E_INVALID_PCB -7
- #define E_INVALID_SCHEDULER_STATE -8
- #define E_NO_SUCH_PROCESS -9
- #define E_INVALID_ARGUMENT -10
- #define E_TRIED_TO_KILL_INIT -11
- #define E_TCSET_NO_TERMINAL_CONTROL -12
- #define E_CONTINUE_NON_STOPPED_PROCESS -13
- #define E_STOP_STOPPED_PROCESS -14
- #define E_STOP_NON_ACTIVE_QUEUE_PROCESS -15
- #define E_PID_NOT_FOUND -16
- #define E_NO_CURRENT_PROCESS -17
- #define E_RUNNING_PROCESS_NOT_IN_READY_QUEUE -18
- #define EK_OPEN_INVALID_FILENAME -19
- #define EK_OPEN_FIND_FILE_IN_ROOT_DIR_FAILED -20
- #define EK_OPEN_GLOBAL_FD_TABLE_FULL -21
- #define EK_OPEN_FILE_DOES_NOT_EXIST -22
- #define EK_OPEN_MALLOC_FAILED -23
- #define EK_OPEN_TIME_FAILED -24
- #define EK_OPEN_ALREADY_WRITE_LOCKED -25
- #define EK_OPEN_WRITE_NEW_ROOT_DIR_ENTRY_FAILED -26
- #define EK_OPEN_NO_EMPTY_BLOCKS -27
- #define EK_OPEN_WRONG_PERMISSIONS -28
- #define EK_OPEN_WRITE_ROOT_DIR_ENTRY_FAILED -29

- #define EK_CLOSE_FD_OUT_OF_RANGE -30
- #define EK_CLOSE_SPECIAL_FD -31
- #define EK_CLOSE_WRITE_ROOT_DIR_ENTRY_FAILED -32
- #define EK_READ_FD_OUT_OF_RANGE -33
- #define EK_READ_FD_NOT_IN_TABLE -34
- #define EK_READ_COULD_NOT_JUMP_TO_BLOCK_FOR_OFFSET -35
- #define EK_READ_WRONG_PERMISSIONS -36
- #define EK_READ_READ_FAILED -37
- #define EK_LSEEK_BAD_WHENCE -38
- #define EK_LSEEK_NEGATIVE_OFFSET -39
- #define EK_LSEEK_OFFSET_OVERFLOW -40
- #define EK_LSEEK_FD_OUT_OF_RANGE -41
- #define EK_LSEEK_FD_NOT_IN_TABLE -42
- #define EK_LSEEK_WRONG_PERMISSIONS -43
- #define EK_LSEEK_SPECIAL_FD -44
- #define EK_WRITE_WRONG_PERMISSIONS -45
- #define EK_WRITE_FD_NOT_IN_TABLE -46
- #define EK_WRITE_FD_OUT_OF_RANGE -47
- #define EK_WRITE_GET_BLOCK_FAILED -48
- #define EK_WRITE_NEXT_BLOCK_NUM_FAILED -49
- #define EK_WRITE_WRITE_BLOCK_FAILED -50
- #define EK_WRITE_WRITE_ROOT_DIR_ENTRY_FAILED -51
- #define EK_WRITE_NO_EMPTY_BLOCKS -52
- #define EK_WRITE_TIME_FAILED -53
- #define EK_WRITE_WRITE_FAILED -54
- #define EK_UNLINK_FILE_NOT_FOUND -55
- #define EK_UNLINK_FIND_FILE_IN_ROOT_DIR_FAILED -56
- #define EK_UNLINK_WRITE_ROOT_DIR_ENTRY_FAILED -57
- #define EK_UNLINK_INVALID_FILENAME -58
- #define EK_LS_WRITE_FAILED -59
- #define EK_LS_FIND_FILE_IN_ROOT_DIR_FAILED -60
- #define EK_LS_NOT_IMPLEMENTED -61
- #define EK_LS_MALLOC_FAILED -62
- #define EK_LS_GET_BLOCK_FAILED -63
- #define EK_LS_NEXT_BLOCK_NUM_FAILED -64
- #define EK_CHMOD_FILE_NOT_FOUND -65
- #define EK_CHMOD_WRITE_ROOT_DIR_ENTRY_FAILED -66
- #define EK_CHMOD_WRONG_PERMISSIONS -67
- #define EK_CHMOD_INVALID_FILENAME -68
- #define EK_CHMOD_INVALID_MODE -69
- #define EK_MV_FILE_NOT_FOUND -70
- #define EK_MV_WRONG_PERMISSIONS -71
- #define EK_MV_UNLINK_FAILED -72
- #define EK_MV_INVALID_FILENAME -73
- #define EK_MV_OPEN_FAILED -74
- #define EK_MV_WRITE_ROOT_DIR_ENTRY_FAILED -75
- #define EK_MV_CLOSE_FAILED -76
- #define EK_SETMODE_FD_OUT_OF_RANGE -77
- #define EK_SETMODE_BAD_MODE -78
- #define EK_SETMODE_FD_NOT_IN_USE -79
- #define EK_GETMODE_FD_OUT_OF_RANGE -80
- #define EK_GETMODE_FD_NOT_IN_USE -81
- #define E_UNKNOWN_FD -103
- #define E_PROCESS_FILE_TABLE_FULL -100
- #define E_READ_UNKNOWN_FD -101
- #define E_STRING_FORMAT_FAILED -104
- #define E_STRING_TOO_LONG_FOR_PRINTF_BUF -105

7.50.1 Macro Definition Documentation

7.50.1.1 E_BAD_ARGV

```
#define E_BAD_ARGV -4
```

7.50.1.2 E_CONTINUE_NON_STOPPED_PROCESS

```
#define E_CONTINUE_NON_STOPPED_PROCESS -13
```

7.50.1.3 E_FAILED_TO_ALLOCATE

```
#define E_FAILED_TO_ALLOCATE -3
```

7.50.1.4 E_INIT_ALREADY_EXISTS

```
#define E_INIT_ALREADY_EXISTS -2
```

7.50.1.5 E_INVALID_ARGUMENT

```
#define E_INVALID_ARGUMENT -10
```

7.50.1.6 E_INVALID_PCB

```
#define E_INVALID_PCB -7
```

7.50.1.7 E_INVALID_SCHEDULER_STATE

```
#define E_INVALID_SCHEDULER_STATE -8
```

7.50.1.8 E_NO_CURRENT_PROCESS

```
#define E_NO_CURRENT_PROCESS -17
```

7.50.1.9 E_NO_INIT_PROCESS

```
#define E_NO_INIT_PROCESS -1
```

7.50.1.10 E_NO_SUCH_PROCESS

```
#define E_NO_SUCH_PROCESS -9
```

7.50.1.11 E_PID_NOT_FOUND

```
#define E_PID_NOT_FOUND -16
```

7.50.1.12 E_PROCESS_FILE_TABLE_FULL

```
#define E_PROCESS_FILE_TABLE_FULL -100
```

7.50.1.13 E_READ_UNKNOWN_FD

```
#define E_READ_UNKNOWN_FD -101
```

7.50.1.14 E_RUNNING_PROCESS_NOT_IN_READY_QUEUE

```
#define E_RUNNING_PROCESS_NOT_IN_READY_QUEUE -18
```

7.50.1.15 E_STOP_NON_ACTIVE_QUEUE_PROCESS

```
#define E_STOP_NON_ACTIVE_QUEUE_PROCESS -15
```

7.50.1.16 E_STOP_STOPPED_PROCESS

```
#define E_STOP_STOPPED_PROCESS -14
```

7.50.1.17 E_STR_FORMAT_FAILED

```
#define E_STR_FORMAT_FAILED -5
```

7.50.1.18 E_STR_TOO_LONG_FOR_FPRINTF_BUF

```
#define E_STR_TOO_LONG_FOR_FPRINTF_BUF -6
```

7.50.1.19 E_STRING_FORMAT_FAILED

```
#define E_STRING_FORMAT_FAILED -104
```

7.50.1.20 E_STRING_TOO_LONG_FOR_PRINTF_BUF

```
#define E_STRING_TOO_LONG_FOR_PRINTF_BUF -105
```

7.50.1.21 E_TCSET_NO_TERMINAL_CONTROL

```
#define E_TCSET_NO_TERMINAL_CONTROL -12
```

7.50.1.22 E_TRIED_TO_KILL_INIT

```
#define E_TRIED_TO_KILL_INIT -11
```

7.50.1.23 E_UNKNOWN_FD

```
#define E_UNKNOWN_FD -103
```

7.50.1.24 EK_CHMOD_FILE_NOT_FOUND

```
#define EK_CHMOD_FILE_NOT_FOUND -65
```

7.50.1.25 EK_CHMOD_INVALID_FILENAME

```
#define EK_CHMOD_INVALID_FILENAME -68
```

7.50.1.26 EK_CHMOD_INVALID_MODE

```
#define EK_CHMOD_INVALID_MODE -69
```

7.50.1.27 EK_CHMOD_WRITE_ROOT_DIR_ENTRY_FAILED

```
#define EK_CHMOD_WRITE_ROOT_DIR_ENTRY_FAILED -66
```

7.50.1.28 EK_CHMOD_WRONG_PERMISSIONS

```
#define EK_CHMOD_WRONG_PERMISSIONS -67
```

7.50.1.29 EK_CLOSE_FD_OUT_OF_RANGE

```
#define EK_CLOSE_FD_OUT_OF_RANGE -30
```

7.50.1.30 EK_CLOSE_SPECIAL_FD

```
#define EK_CLOSE_SPECIAL_FD -31
```

7.50.1.31 EK_CLOSE_WRITE_ROOT_DIR_ENTRY_FAILED

```
#define EK_CLOSE_WRITE_ROOT_DIR_ENTRY_FAILED -32
```

7.50.1.32 EK_GETMODE_FD_NOT_IN_USE

```
#define EK_GETMODE_FD_NOT_IN_USE -81
```

7.50.1.33 EK_GETMODE_FD_OUT_OF_RANGE

```
#define EK_GETMODE_FD_OUT_OF_RANGE -80
```

7.50.1.34 EK_LS_FIND_FILE_IN_ROOT_DIR_FAILED

```
#define EK_LS_FIND_FILE_IN_ROOT_DIR_FAILED -60
```

7.50.1.35 EK_LS_GET_BLOCK_FAILED

```
#define EK_LS_GET_BLOCK_FAILED -63
```

7.50.1.36 EK_LS_MALLOC_FAILED

```
#define EK_LS_MALLOC_FAILED -62
```

7.50.1.37 EK_LS_NEXT_BLOCK_NUM_FAILED

```
#define EK_LS_NEXT_BLOCK_NUM_FAILED -64
```

7.50.1.38 EK_LS_NOT_IMPLEMENTED

```
#define EK_LS_NOT_IMPLEMENTED -61
```

7.50.1.39 EK_LS_WRITE_FAILED

```
#define EK_LS_WRITE_FAILED -59
```

7.50.1.40 EK_LSEEK_BAD_WHENCE

```
#define EK_LSEEK_BAD_WHENCE -38
```

7.50.1.41 EK_LSEEK_FD_NOT_IN_TABLE

```
#define EK_LSEEK_FD_NOT_IN_TABLE -42
```

7.50.1.42 EK_LSEEK_FD_OUT_OF_RANGE

```
#define EK_LSEEK_FD_OUT_OF_RANGE -41
```

7.50.1.43 EK_LSEEK_NEGATIVE_OFFSET

```
#define EK_LSEEK_NEGATIVE_OFFSET -39
```

7.50.1.44 EK_LSEEK_OFFSET_OVERFLOW

```
#define EK_LSEEK_OFFSET_OVERFLOW -40
```

7.50.1.45 EK_LSEEK_SPECIAL_FD

```
#define EK_LSEEK_SPECIAL_FD -44
```

7.50.1.46 EK_LSEEK_WRONG_PERMISSIONS

```
#define EK_LSEEK_WRONG_PERMISSIONS -43
```

7.50.1.47 EK_MV_CLOSE_FAILED

```
#define EK_MV_CLOSE_FAILED -76
```

7.50.1.48 EK_MV_FILE_NOT_FOUND

```
#define EK_MV_FILE_NOT_FOUND -70
```

7.50.1.49 EK_MV_INVALID_FILENAME

```
#define EK_MV_INVALID_FILENAME -73
```

7.50.1.50 EK_MV_OPEN_FAILED

```
#define EK_MV_OPEN_FAILED -74
```

7.50.1.51 EK_MV_UNLINK_FAILED

```
#define EK_MV_UNLINK_FAILED -72
```

7.50.1.52 EK_MV_WRITE_ROOT_DIR_ENTRY_FAILED

```
#define EK_MV_WRITE_ROOT_DIR_ENTRY_FAILED -75
```

7.50.1.53 EK_MV_WRONG_PERMISSIONS

```
#define EK_MV_WRONG_PERMISSIONS -71
```

7.50.1.54 EK_OPEN_ALREADY_WRITE_LOCKED

```
#define EK_OPEN_ALREADY_WRITE_LOCKED -25
```

7.50.1.55 EK_OPEN_FILE_DOES_NOT_EXIST

```
#define EK_OPEN_FILE_DOES_NOT_EXIST -22
```

7.50.1.56 EK_OPEN_FIND_FILE_IN_ROOT_DIR_FAILED

```
#define EK_OPEN_FIND_FILE_IN_ROOT_DIR_FAILED -20
```

7.50.1.57 EK_OPEN_GLOBAL_FD_TABLE_FULL

```
#define EK_OPEN_GLOBAL_FD_TABLE_FULL -21
```

7.50.1.58 EK_OPEN_INVALID_FILENAME

```
#define EK_OPEN_INVALID_FILENAME -19
```

7.50.1.59 EK_OPEN_MALLOC_FAILED

```
#define EK_OPEN_MALLOC_FAILED -23
```

7.50.1.60 EK_OPEN_NO_EMPTY_BLOCKS

```
#define EK_OPEN_NO_EMPTY_BLOCKS -27
```

7.50.1.61 EK_OPEN_TIME_FAILED

```
#define EK_OPEN_TIME_FAILED -24
```

7.50.1.62 EK_OPEN_WRITE_NEW_ROOT_DIR_ENTRY_FAILED

```
#define EK_OPEN_WRITE_NEW_ROOT_DIR_ENTRY_FAILED -26
```

7.50.1.63 EK_OPEN_WRITE_ROOT_DIR_ENTRY_FAILED

```
#define EK_OPEN_WRITE_ROOT_DIR_ENTRY_FAILED -29
```

7.50.1.64 EK_OPEN_WRONG_PERMISSIONS

```
#define EK_OPEN_WRONG_PERMISSIONS -28
```

7.50.1.65 EK_READ_COULD_NOT_JUMP_TO_BLOCK_FOR_OFFSET

```
#define EK_READ_COULD_NOT_JUMP_TO_BLOCK_FOR_OFFSET -35
```

7.50.1.66 EK_READ_FD_NOT_IN_TABLE

```
#define EK_READ_FD_NOT_IN_TABLE -34
```

7.50.1.67 EK_READ_FD_OUT_OF_RANGE

```
#define EK_READ_FD_OUT_OF_RANGE -33
```

7.50.1.68 EK_READ_READ_FAILED

```
#define EK_READ_READ_FAILED -37
```

7.50.1.69 EK_READ_WRONG_PERMISSIONS

```
#define EK_READ_WRONG_PERMISSIONS -36
```

7.50.1.70 EK_SETMODE_BAD_MODE

```
#define EK_SETMODE_BAD_MODE -78
```

7.50.1.71 EK_SETMODE_FD_NOT_IN_USE

```
#define EK_SETMODE_FD_NOT_IN_USE -79
```

7.50.1.72 EK_SETMODE_FD_OUT_OF_RANGE

```
#define EK_SETMODE_FD_OUT_OF_RANGE -77
```

7.50.1.73 EK_UNLINK_FILE_NOT_FOUND

```
#define EK_UNLINK_FILE_NOT_FOUND -55
```

7.50.1.74 EK_UNLINK_FIND_FILE_IN_ROOT_DIR_FAILED

```
#define EK_UNLINK_FIND_FILE_IN_ROOT_DIR_FAILED -56
```

7.50.1.75 EK_UNLINK_INVALID_FILENAME

```
#define EK_UNLINK_INVALID_FILENAME -58
```

7.50.1.76 EK_UNLINK_WRITE_ROOT_DIR_ENTRY_FAILED

```
#define EK_UNLINK_WRITE_ROOT_DIR_ENTRY_FAILED -57
```

7.50.1.77 EK_WRITE_FD_NOT_IN_TABLE

```
#define EK_WRITE_FD_NOT_IN_TABLE -46
```

7.50.1.78 EK_WRITE_FD_OUT_OF_RANGE

```
#define EK_WRITE_FD_OUT_OF_RANGE -47
```

7.50.1.79 EK_WRITE_GET_BLOCK_FAILED

```
#define EK_WRITE_GET_BLOCK_FAILED -48
```

7.50.1.80 EK_WRITE_NEXT_BLOCK_NUM_FAILED

```
#define EK_WRITE_NEXT_BLOCK_NUM_FAILED -49
```

7.50.1.81 EK_WRITE_NO_EMPTY_BLOCKS

```
#define EK_WRITE_NO_EMPTY_BLOCKS -52
```

7.50.1.82 EK_WRITE_TIME_FAILED

```
#define EK_WRITE_TIME_FAILED -53
```

7.50.1.83 EK_WRITE_WRITE_BLOCK_FAILED

```
#define EK_WRITE_WRITE_BLOCK_FAILED -50
```

7.50.1.84 EK_WRITE_WRITE_FAILED

```
#define EK_WRITE_WRITE_FAILED -54
```

7.50.1.85 EK_WRITE_WRITE_ROOT_DIR_ENTRY_FAILED

```
#define EK_WRITE_WRITE_ROOT_DIR_ENTRY_FAILED -51
```

7.50.1.86 EK_WRITE_WRONG_PERMISSIONS

```
#define EK_WRITE_WRONG_PERMISSIONS -45
```

Index