PennOS

1.0

Generated by Doxygen 1.10.0

Chapter 1

PennOS

Welcome to PennOS, a project designed to simulate the functionalities of a UNIX-like operating system. This project was created by Aaron Tsui, Matt Park, Joesph Cho, and Maya Huizar.

1.0.1 Overview

PennOS is a UNIX-like operating system that runs as a guest OS within a single process on a host OS. It includes implementations of a basic priority scheduler, a FAT file system (PennFAT), and user shell interactions.

1.0.2 Documentation

- **Kernel Documentation**: Here is an overview of the Kernel, Scheduler, and Shell, and its related functions and structure.
- **FAT Documentation**: Here is an overview of the structure of the filesystem, and its related functions.

Chapter 2

fat

Chapter 3

kernel

The kernel side of PennOS consists of three main aspects. They are described in more details on their own subpages below.

1. **The Kernel**: The kernel refers to the collection of system calls, as well as the overall datastructures and control mechanisms used by the scheduler.
2. **The Scheduler**: The scheduler is the main function that is in charge of deciding which process to schedule/run based on priorities, blocking and unblocking processes, and idling.
3. **The Shell**: The shell is simply a priority zero process that is instantiated at start, and continuously checks for user input to spawn new processes, or modify existing ones.

Chapter 4

scheduler

The scheduler is the main function in charge of mediating processes. The scheduler works in terms of quanta, of which each quantum lasts 100 ms. The general structure/timeline of a quantum looks like this.

The start of a new quantum is triggered by the scheduler receiving a SIGALRM signal, which it receives every 100 ms as set by a timer.

After receiving the alarm, the scheduler will suspend the currenly running process. It then will check the state and status of all blocked processes, unblocking, or updating them as needed. This may include unblocking a parent whose child has exited, or reducing the number of ticks to sleep for a process that called sleep. These events will be logged if necessary. After that, the process will determine the next process to run by checking the next priority to be run, that has a schedulable (read: running) process available. It will log this, and then unsuspend the processes spthread until the next SIGALRM at the end of the quantum, at which point this cycle will repeat.

Chapter 5

shell

The shell is the main process of pennos. It is declared and definen in shell. It is instatiated at tick 0, or the start of pennos, at priority 0. The shell is then scheduled by the scheduler to take in input from the user, and spawn in additional processes via s_spawn.

The commands available to be typed in the shell can be listed by typing "man" in the shell. These are the shell level commands that are specified in the PennOS assignment.

Chapter 6

system

For the system/kernel of PennOS, the most important notion is that of the process control block (PCB).

This can be seen in more detail here: pcb_t.

The main idea of the process control block it represents a process, or thread, that can be in several states. Processes can be running, stopped, blocked, or zombied.

The transitions between processes are mediated via signals sent via s_kill, s_exit, and s_sleep.

The kernel maintains circular linked lists CircularList's of processes in each state. There are 6 in total, one for each state: ZOMBIED, STOPPED, BLOCKED, and 3 for RUNNING, one for each priority level.

As a rule, user level functions should not need to and should not mediate process state transitions, as these will be handled by the scheduler or by system calls.

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 8

File Index

8.1 File List

Here is a list of all files with brief descriptions:

Chapter 9

Data Structure Documentation

9.1 CircularList Struct Reference

This structure represents a circular linked list for managing processes.

`#include <clinkedlist.h>`

Data Fields

- Node * head
- unsigned int size

9.1.1 Detailed Description

This structure represents a circular linked list for managing processes.

9.1.2 Field Documentation

9.1.2.1 head

```
Node* CircularList::head
```

Pointer to the head (first node) of the list.

9.1.2.2 size

```
unsigned int CircularList::size
```

Number of nodes in the list.

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

- src/util/clinkedlist.h

9.2 directory_entries Struct Reference

`#include <pennfat_kernel.h>`

Data Fields

- char name [32]
- uint32_t size
- uint16_t firstBlock
- uint8_t type
- uint8_t perm
- time_t mtime
- uint8_t reserved [16]

9.2.1 Field Documentation

9.2.1.1 firstBlock

`uint16_t directory_entries::firstBlock`

9.2.1.2 mtime

`time_t directory_entries::mtime`

9.2.1.3 name

`char directory_entries::name[32]`

9.2.1.4 perm

`uint8_t directory_entries::perm`

9.2.1.5 reserved

`uint8_t directory_entries::reserved[16]`

9.2.1.6 size

`uint32_t directory_entries::size`

9.2.1.7 type

```
uint8_t directory_entries::type
```

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

- src/util/pennfat_kernel.h

9.3 DynamicPIDArray Struct Reference

Structure for the dynamic array to store child PIDs.

```
#include <array.h>
```

Data Fields

- pid_t * array
- size_t used
- size_t size

9.3.1 Detailed Description

Structure for the dynamic array to store child PIDs.

9.3.2 Field Documentation

9.3.2.1 array

```
pid_t* DynamicPIDArray::array
```

Pointer to the array of child PIDs.

9.3.2.2 size

```
size_t DynamicPIDArray::size
```

Current allocated size of the array.

9.3.2.3 used

```
size_t DynamicPIDArray::used
```

Number of elements currently used.

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

- src/util/array.h

9.4 FD_Bitmap Struct Reference

Structure for managing open file descriptors using a bitmap.

`#include <bitmap.h>`

Data Fields

- uint8_t bits [FD_BITMAP_BYTES]

9.4.1 Detailed Description

Structure for managing open file descriptors using a bitmap.

9.4.2 Field Documentation

9.4.2.1 bits

`uint8_t FD_Bitmap::bits[FD_BITMAP_BYTES]`

Array of bytes to represent the bitmap.

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

- src/util/bitmap.h

9.5 file_descriptor_st Struct Reference

`#include <pennfat_kernel.h>`

Data Fields

- int fd
- char * fname
- int mode
- int offset
- int ref_cnt

9.5.1 Field Documentation

9.5.1.1 fd

`int file_descriptor_st::fd`

9.5.1.2 fname

```
char* file_descriptor_st::fname
```

9.5.1.3 mode

```
int file_descriptor_st::mode
```

9.5.1.4 offset

```
int file_descriptor_st::offset
```

9.5.1.5 ref_cnt

```
int file_descriptor_st::ref_cnt
```

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

- src/util/pennfat_kernel.h

9.6 Node Struct Reference

This structure represents a node in the circular linked list.

```
#include <clinkedlist.h>
```

Data Fields

- pcb_t * process
- struct Node * next

9.6.1 Detailed Description

This structure represents a node in the circular linked list.

9.6.2 Field Documentation

9.6.2.1 next

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

Pointer to the next node in the list.

9.6.2.2 process

```
pcb_t* Node::process
```

Pointer to the process control block (pcb_t).

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

- src/util/clinkedlist.h

9.7 parsed_command Struct Reference

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

Data Fields

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

9.7.1 Detailed Description

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

9.7.2 Field Documentation

9.7.2.1 commands

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

9.7.2.2 is_background

```
bool parsed_command::is_background
```

9.7.2.3 is_file_append

```
bool parsed_command::is_file_append
```

9.7.2.4 num_commands

```
size_t parsed_command::num_commands
```

9.7.2.5 stdin_file

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

9.7.2.6 stdout_file

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

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

- src/parser.h

9.8 pcb_t Struct Reference

This structure stores all required information about a running process.

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

Data Fields

- spthread_t handle
- pid_t pid

  This stores a handle to the spthread.

- pid_t ppid

  This stores the PID of the process.
- DynamicPIDArray * child_pids

  This stores the PPID of the process.
- unsigned int priority: 2

  This stores a pointer to a dynamically sized array of child pid_t's.
- process_state_t state

  This the priority level of the process. (0, 1, or 2).
- FD_Bitmap * open_fds

  This is an enum storing the process's current state.
- bool statechanged

  This stores a bitmap containg all open file descriptors.
- int exit_status

  This contains a bool that keeps track of whether or not the process state has changed.
- int term_signal

  Exit status of process, 0 if exited , -1 if not exited.
- bool waiting_for_change

  Signal number that caused process to terminate, -1 if not terminated.
- pid_t waiting_on_pid

  Bool describing whether or not the process is currently waiting on a process.
- unsigned int ticks_to_wait

  PID of the child the process is currently waiting on, or -1 if none.
- char * processname

  Ticks remaining to wait, used only for s_sleep calls.

9.8.1 Detailed Description

This structure stores all required information about a running process.

9.8.2 Field Documentation

9.8.2.1 child_pids

```
DynamicPIDArray* pcb_t::child_pids
```

This stores the PPID of the process.

9.8.2.2 exit_status

```
int pcb_t::exit_status
```

This contains a bool that keeps track of whether or not the process state has changed.

9.8.2.3 handle

```
spthread_t pcb_t::handle
```

9.8.2.4 open_fds

```
FD_Bitmap* pcb_t::open_fds
```

This is an enum storing the process's current state.

9.8.2.5 pid

```
pid_t pcb_t::pid
```

This stores a handle to the spthread.

9.8.2.6 ppid

```
pid_t pcb_t::ppid
```

This stores the PID of the process.

9.8.2.7 priority

```
unsigned int pcb_t::priority
```

This stores a pointer to a dynamically sized array of child pid_t's.

9.8.2.8 processname

```
char* pcb_t::processname
```

Ticks remaining to wait, used only for s_sleep calls.

9.8.2.9 state

```
process_state_t pcb_t::state
```

This the priority level of the process. (0, 1, or 2).

9.8.2.10 statechanged

```
bool pcb_t::statechanged
```

This stores a bitmap containg all open file descriptors.

9.8.2.11 term_signal

```
int pcb_t::term_signal
```

Exit status of process, 0 if exited , -1 if not exited.

9.8.2.12 ticks_to_wait

```
unsigned int pcb_t::ticks_to_wait
```

PID of the child the process is currently waiting on, or -1 if none.

9.8.2.13 waiting_for_change

```
bool pcb_t::waiting_for_change
```

Signal number that caused process to terminate, -1 if not terminated.

9.8.2.14 waiting_on_pid

```
pid_t pcb_t::waiting_on_pid
```

Bool describing whether or not the process is currently waiting on a process.

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

- src/util/kernel.h

9.9 PList Struct Reference

`#include <prioritylist.h>`

Data Fields

- PNode * head
- unsigned int size

9.9.1 Field Documentation

9.9.1.1 head

`PNode* PList::head`

Pointer to the head (first node) of the list.

9.9.1.2 size

`unsigned int PList::size`

Number of nodes in the list.

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

- src/util/prioritylist.h

9.10 PNode Struct Reference

`#include <prioritylist.h>`

Data Fields

- unsigned int priority: 2
- struct PNode * next

9.10.1 Field Documentation

9.10.1.1 next

`struct PNode* PNode::next`

Pointer to the next node in the list.

9.10.1.2 priority

```
unsigned int PNode::priority
```

Pointer to the process control block (pcb_t).

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

- src/util/prioritylist.h

9.11 spthread_fwd_args_st Struct Reference

Data Fields

- 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

9.11.1 Field Documentation

9.11.1.1 actual_arg

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

9.11.1.2 actual_routine

```
pthread_fn spthread_fwd_args_st::actual_routine
```

9.11.1.3 child_meta

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

9.11.1.4 setup_cond

```
pthread_cond_t spthread_fwd_args_st::setup_cond
```

9.11.1.5 setup_done

```
bool spthread_fwd_args_st::setup_done
```

9.11.1.6 setup_mutex

```
pthread_mutex_t spthread_fwd_args_st::setup_mutex
```

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

- src/util/spthread.c

9.12 spthread_meta_st Struct Reference

Data Fields

- sigset_t suspend_set
- volatile sig_atomic_t state
- pthread_mutex_t meta_mutex

9.12.1 Field Documentation

9.12.1.1 meta_mutex

```
pthread_mutex_t spthread_meta_st::meta_mutex
```

9.12.1.2 state

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

9.12.1.3 suspend_set

```
sigset_t spthread_meta_st::suspend_set
```

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

- src/util/spthread.c

9.13 spthread_signal_args_st Struct Reference

Data Fields

- const int signal
- volatile sig_atomic_t ack
- pthread_mutex_t shutup_mutex

9.13.1 Field Documentation

9.13.1.1 ack

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

9.13.1.2 shutup_mutex

```
pthread_mutex_t spthread_signal_args_st::shutup_mutex
```

9.13.1.3 signal

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

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

- src/util/spthread.c

9.14 spthread_st Struct Reference

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

Data Fields

- pthread_t thread
- spthread_meta_t * meta

9.14.1 Field Documentation

9.14.1.1 meta

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

9.14.1.2 thread

```
pthread_t spthread_st::thread
```

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

- src/util/spthread.h

Chapter 10

File Documentation

10.1 doc/fat.md File Reference

10.2 doc/kernel.md File Reference

10.3 doc/README.md File Reference

10.4 doc/scheduler.md File Reference

10.5 doc/shell.md File Reference

10.6 doc/system.md File Reference

10.7 src/parser.h File Reference

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

Data Structures

- struct parsed_command

Macros

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

Functions

- int parse_command (const char *cmd_line, struct parsed_command **result)
- void print_parsed_command (FILE *output, const struct parsed_command *cmd)
- void print_parser_errcode (FILE *output, int err_code)

10.7.1 Macro Definition Documentation

10.7.1.1 EXPECT_COMMANDS

```
#define EXPECT_COMMANDS 7
```

10.7.1.2 EXPECT_INPUT_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

10.7.1.3 EXPECT_OUTPUT_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

10.7.1.4 UNEXPECTED_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

10.7.1.5 UNEXPECTED_FILE_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

10.7.1.6 UNEXPECTED_FILE_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

10.7.1.7 UNEXPECTED_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

10.7.2 Function Documentation

10.7.2.1 parse_command()

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

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

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

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

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

If the function returns a successful value (0), a `struct parsed_command` is 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.

10.7.2.2 print_parsed_command()

```
void print_parsed_command (
            FILE * output,
            const struct parsed_command * cmd )
```

10.7.2.3 print_parser_errcode()

```
void print_parser_errcode (
            FILE * output,
            int err_code )
```

10.8 parser.h

Go to the documentation of this file.

```
00001 /* Penn-Shell Parser
00002    hanbangw, 21fa    */
00003
00004 #pragma once
00005
00006 #include <stdbool.h>
00007 #include <stddef.h>
00008 #include <stdio.h>
00009
00010 /* Here defines all possible parser errors */
00011 // parser encountered an unexpected file input token '<'
00012 #define UNEXPECTED_FILE_INPUT 1
00013
00014 // parser encountered an unexpected file output token '>'
00015 #define UNEXPECTED_FILE_OUTPUT 2
00016
```

```
00017 // parser encountered an unexpected pipeline token '|'
00018 #define UNEXPECTED_PIPELINE 3
00019
00020 // parser encountered an unexpected ampersand token '&'
00021 #define UNEXPECTED_AMPERSAND 4
00022
00023 // parser didn't find input filename following '<'
00024 #define EXPECT_INPUT_FILENAME 5
00025
00026 // parser didn't find output filename following '>' or '»'
00027 #define EXPECT_OUTPUT_FILENAME 6
00028
00029 // parser didn't find any commands or arguments where it expects one
00030 #define EXPECT_COMMANDS 7
00031
00036 struct parsed_command {
00037   // indicates the command shall be executed in background
00038   // (ends with an ampersand '&')
00039   bool is_background;
00040
00041   // indicates if the stdout_file shall be opened in append mode
00042   // ignore this value when stdout_file is NULL
00043   bool is_file_append;
00044
00045   // filename for redirecting input from
00046   const char* stdin_file;
00047
00048   // filename for redirecting output to
00049   const char* stdout_file;
00050
00051   // number of commands (pipeline stages)
00052   size_t num_commands;
00053
00054   // an array to a list of arguments
00055   // size of `commands' is `num_commands'
00056   char** commands[];
00057 };
00058
00086 int parse_command(const char* cmd_line, struct parsed_command** result);
00087
00088 /* This is a debugging function used for outputting a parsed command line. */
00089 void print_parsed_command(FILE* output, const struct parsed_command* cmd);
00090
00091 /* a debugging function for printing out what error was encountered */
00092 void print_parser_errcode(FILE* output, int err_code);
```

10.9 src/pennfat.c File Reference

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

Functions

- void prompt ()
- void read_command (char **cmds)
- void int_handler (int signo)
- void initialize_global_fd_table ()
- void mkfs (const char *fs_name, int blocks_in_fat, int block_size_config)
- int mount (const char *fs_name)
- int unmount ()
- int get_block_size (int block_size_config)
- int get_fat_size (int block_size, int blocks_in_fat)
- int get_num_fat_entries (int block_size, int blocks_in_fat)
- int get_data_size (int block_size, int num_fat_entries)
- int get_offset_size (int block_num, int offset)
- void touch (char **args)
- void rm (char **args)

- void mv (char **args)
- void chmod (char **args)
- void cat_file_wa (char **args)
- void cat_w (char *output)
- void cat_a (char *output)
- void ls ()
- void cp_within_fat (char *source, char *dest)
- void cp_to_host (char *source, char *host_dest)
- void cp_from_host (char *host_source, char *dest)

10.9.1 Function Documentation

10.9.1.1 cat_a()

```
void cat_a (
            char * output )
```

10.9.1.2 cat_file_wa()

```
void cat_file_wa (
            char ** args )
```

10.9.1.3 cat_w()

```
void cat_w (
            char * output )
```

10.9.1.4 chmod()

```
void chmod (
            char ** args )
```

10.9.1.5 cp_from_host()

```
void cp_from_host (
            char * host_source,
            char * dest )
```

10.9.1.6 cp_to_host()

```
void cp_to_host (
            char * source,
            char * host_dest )
```

10.9.1.7 cp_within_fat()

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

10.9.1.8 get_block_size()

```
int get_block_size (
            int block_size_config )
```

10.9.1.9 get_data_size()

```
int get_data_size (
            int block_size,
            int num_fat_entries )
```

10.9.1.10 get_fat_size()

```
int get_fat_size (
            int block_size,
            int blocks_in_fat )
```

10.9.1.11 get_num_fat_entries()

```
int get_num_fat_entries (
            int block_size,
            int blocks_in_fat )
```

10.9.1.12 get_offset_size()

```
int get_offset_size (
            int block_num,
            int offset )
```

10.9.1.13 initialize_global_fd_table()

```
void initialize_global_fd_table ( )
```

10.9.1.14 int_handler()

```
void int_handler (
            int signo )
```

10.9.1.15 ls()

```
void ls ( )
```

10.9.1.16 mkfs()

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

10.9.1.17 mount()

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

10.9.1.18 mv()

```
void mv (
            char ** args )
```

10.9.1.19 prompt()

```
void prompt ( )
```

10.9.1.20 read_command()

```
void read_command (
            char ** cmds )
```

10.9.1.21 rm()

```
void rm (
            char ** args )
```

10.9.1.22 touch()

```
void touch (
            char ** args )
```

10.9.1.23 unmount()

```
int unmount ( )
```

10.10 src/pennfat.h File Reference

```
#include <fcntl.h>
#include <signal.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <time.h>
#include "parser.h"
#include "util/pennfat_kernel.h"
```

Macros

- #define MAX_LEN 4096

Functions

- void mkfs (const char *fs_name, int blocks_in_fat, int block_size_config)
- int mount (const char *fs_name)
- int unmount ()
- void prompt ()
- void read_command ()
- void int_handler (int signo)
- int get_block_size (int block_size_config)
- int get_fat_size (int block_size, int blocks_in_fat)
- int get_num_fat_entries (int block_size, int blocks_in_fat)
- int get_data_size (int block_size, int num_fat_entries)
- int get_offset_size (int block_num, int offset)
- void initialize_global_fd_table ()
- void touch (char **args)
- void rm (char **args)
- void mv (char **args)
- void chmod (char **args)
- void cat_file_wa (char **args)
- void ls ()
- void cp_within_fat (char *source, char *dest)
- void cp_to_host (char *source, char *host_dest)
- void cp_from_host (char *host_source, char *dest)
- void cat_w (char *output)
- void cat_a (char *output)

10.10.1 Macro Definition Documentation

10.10.1.1 MAX_LEN

```
#define MAX_LEN 4096
```

10.10.2 Function Documentation

10.10.2.1 cat_a()

```
void cat_a (
            char * output )
```

10.10.2.2 cat_file_wa()

```
void cat_file_wa (
            char ** args )
```

10.10.2.3 cat_w()

```
void cat_w (
            char * output )
```

10.10.2.4 chmod()

```
void chmod (
            char ** args )
```

10.10.2.5 cp_from_host()

```
void cp_from_host (
            char * host_source,
            char * dest )
```

10.10.2.6 cp_to_host()

```
void cp_to_host (
            char * source,
            char * host_dest )
```

10.10.2.7 cp_within_fat()

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

10.10.2.8 get_block_size()

```
int get_block_size (
            int block_size_config )
```

10.10.2.9 get_data_size()

```
int get_data_size (
            int block_size,
            int num_fat_entries )
```

10.10.2.10 get_fat_size()

```
int get_fat_size (
            int block_size,
            int blocks_in_fat )
```

10.10.2.11 get_num_fat_entries()

```
int get_num_fat_entries (
            int block_size,
            int blocks_in_fat )
```

10.10.2.12 get_offset_size()

```
int get_offset_size (
            int block_num,
            int offset )
```

10.10.2.13 initialize_global_fd_table()

```
void initialize_global_fd_table ( )
```

10.10.2.14 int_handler()

```
void int_handler (
            int signo )
```

10.10.2.15 ls()

```
void ls ( )
```

10.10.2.16 mkfs()

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

10.10.2.17 mount()

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

10.10.2.18 mv()

```
void mv (
            char ** args )
```

10.10.2.19 prompt()

```
void prompt ( )
```

10.10.2.20 read_command()

```
void read_command ( )
```

10.10.2.21 rm()

```
void rm (
            char ** args )
```

10.10.2.22 touch()

```
void touch (
            char ** args )
```

10.10.2.23 unmount()

```
int unmount ( )
```

10.11 pennfat.h

Go to the documentation of this file.

```
00001 #ifndef PENNFAT_H
00002 #define PENNFAT_H
00003
00004 #include <fcntl.h>
00005 #include <signal.h>
00006 #include <stdarg.h>
00007 #include <stdbool.h>
00008 #include <stdint.h>
00009 #include <stdio.h>
00010 #include <stdlib.h>
00011 #include <string.h>
00012 #include <sys/mman.h>
00013 #include <time.h>
00014
00015 #include "parser.h"
00016 #include "util/pennfat_kernel.h"
00017 #ifndef PROMPT
00018 #define PROMPT "penn-fat> "
00019 #endif
00020 #define MAX_LEN 4096
00021
00022 void mkfs(const char* fs_name, int blocks_in_fat, int block_size_config);
00023 int mount(const char* fs_name);
00024 int unmount();
00025
00026 void prompt();
00027 void read_command();
00028 void int_handler(int signo);
00029 int get_block_size(int block_size_config);
00030 int get_fat_size(int block_size, int blocks_in_fat);
00031 int get_num_fat_entries(int block_size, int blocks_in_fat);
00032 int get_data_size(int block_size, int num_fat_entries);
00033 int get_offset_size(int block_num, int offset);
00034
00035 void initialize_global_fd_table();
00036
00037 void touch(char** args);
00038 void rm(char** args);
00039 void mv(char** args);
00040 void chmod(char** args);
00041 void cat_file_wa(char** args);
00042
00043 void ls();
00044 void cp_within_fat(char* source, char* dest);
00045 void cp_to_host(char* source, char* host_dest);
00046 void cp_from_host(char* host_source, char* dest);
00047
00048 void cat_w(char* output);
00049 void cat_a(char* output);
00050
00051 #endif
```

10.12 src/pennos.c File Reference

```
#include "pennos.h"
#include <signal.h>
#include "fcntl.h"
#include "parser.h"
#include "pennfat.h"
#include "unistd.h"
#include "util/kernel.h"
#include "util/prioritylist.h"
```

Functions

- void scheduler (char *logfile)
- void cancel_and_join (spthread_t thread)
- int main (int argc, char **argv)

Variables

- PList * priority

10.12.1 Function Documentation

10.12.1.1 cancel_and_join()

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

10.12.1.2 main()

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

10.12.1.3 scheduler()

```
void scheduler (
            char * logfile )
```

10.12.2 Variable Documentation

10.12.2.1 priority

```
PList* priority
```

10.13 src/pennos.h File Reference

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include "parser.h"
#include "pennfat.h"
#include "util/globals.h"
#include "util/kernel.h"
#include "util/shellbuiltins.h"
#include "util/sys_call.h"
```

Macros

- #define _POSIX_C_SOURCE 200809L
- #define _DEFAULT_SOURCE 1
- #define PROMPT "penn-os> "
- #define _XOPEN_SOURCE 700
- #define MAX_LEN 4096
- #define STDIN_FILENO 0
- #define STDOUT_FILENO 1
- #define STDERR_FILENO 2

Functions

- void * function_from_string (char *program)

10.13.1 Macro Definition Documentation

10.13.1.1 _DEFAULT_SOURCE

```
#define _DEFAULT_SOURCE 1
```

10.13.1.2 _POSIX_C_SOURCE

```
#define _POSIX_C_SOURCE 200809L
```

10.13.1.3 _XOPEN_SOURCE

```
#define _XOPEN_SOURCE 700
```

10.13.1.4 MAX_LEN

```
#define MAX_LEN 4096
```

10.13.1.5 PROMPT

```
#define PROMPT "penn-os> "
```

10.13.1.6 STDERR_FILENO

```
#define STDERR_FILENO 2
```

10.13.1.7 STDIN_FILENO

```
#define STDIN_FILENO 0
```

10.13.1.8 STDOUT_FILENO

```
#define STDOUT_FILENO 1
```

10.13.2 Function Documentation

10.13.2.1 function_from_string()

```
void * function_from_string (
            char * program )
```

10.14 pennos.h

Go to the documentation of this file.

```
00001 #ifndef PENNOS_H
00002 #define PENNOS_H
00003
00004 #ifndef _POSIX_C_SOURCE
00005 #define _POSIX_C_SOURCE 200809L
00006 #endif
00007
00008 #ifndef _DEFAULT_SOURCE
00009 #define _DEFAULT_SOURCE 1
00010 #endif
00011
00012 #undef PROMPT
00013
00014 #ifndef PROMPT
00015 #define PROMPT "penn-os> "
00016 #endif
00017
00018 #define _XOPEN_SOURCE 700
00019
00020 #ifndef MAX_LEN
00021 #define MAX_LEN 4096
00022 #endif
00023
00024 #define STDIN_FILENO 0
00025 #define STDOUT_FILENO 1
00026 #define STDERR_FILENO 2
00027
00028 #include <fcntl.h>
00029 #include <stdio.h>
00030 #include <stdlib.h>
00031 #include "parser.h"
00032 #include "pennfat.h"
00033 #include "util/globals.h"
00034 #include "util/kernel.h"
00035 #include "util/shellbuiltins.h"
00036 #include "util/sys_call.h"
00037
00038 static bool done = false;
00039 void* function_from_string(char* program);
00040
00041 #endif
```

10.15 src/standalonefat.c File Reference

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

Functions

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

10.15.1 Function Documentation

10.15.1.1 main()

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

10.16 src/util/array.c File Reference

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

Functions

- DynamicPIDArray * dynamic_pid_array_create (size_t initial_size)

  Initializes a new dynamic array for PIDs with an initial size.
- void dynamic_pid_array_destroy (DynamicPIDArray *array)

  Destroys a dynamic PID array, freeing its resources.
- bool dynamic_pid_array_add (DynamicPIDArray *array, pid_t pid)

  Adds a PID to the dynamic array, resizing if necessary.
- bool dynamic_pid_array_remove (DynamicPIDArray *array, pid_t pid)

  Removes a PID from the dynamic array.
- bool dynamic_pid_array_contains (const DynamicPIDArray *array, pid_t pid)

  Checks if a PID exists in the dynamic array.

10.16.1 Function Documentation

10.16.1.1 dynamic_pid_array_add()

```
bool dynamic_pid_array_add (
            DynamicPIDArray * array,
            pid_t pid )
```

Adds a PID to the dynamic array, resizing if necessary.

Parameters

array	A pointer to the dynamic PID array.
pid	The PID to add to the array.

Returns

true on success, false on failure (e.g., if memory allocation fails).

10.16.1.2 dynamic_pid_array_contains()

```
bool dynamic_pid_array_contains (
            const DynamicPIDArray * array,
            pid_t pid )
```

Checks if a PID exists in the dynamic array.

Parameters

array	A pointer to the dynamic PID array.
pid	The PID to check for in the array.

Returns

true if the PID exists in the array, false otherwise.

10.16.1.3 dynamic_pid_array_create()

```
DynamicPIDArray * dynamic_pid_array_create (
            size_t initial_size )
```

Initializes a new dynamic array for PIDs with an initial size.

Parameters

initial_size	The initial size of the dynamic array.

Returns

DynamicPIDArray∗ A pointer to the newly created dynamic array structure.

10.16.1.4 dynamic_pid_array_destroy()

```
void dynamic_pid_array_destroy (
            DynamicPIDArray * array )
```

Destroys a dynamic PID array, freeing its resources.

Parameters

array	A pointer to the dynamic PID array to be destroyed.

10.16.1.5 dynamic_pid_array_remove()

```
bool dynamic_pid_array_remove (
```

```
            DynamicPIDArray * array,
            pid_t pid )
```

Removes a PID from the dynamic array.

Parameters

array	A pointer to the dynamic PID array.
pid	The PID to remove from the array.

Returns

true if the PID was successfully removed, false if the PID was not found.

10.17 src/util/array.h File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <sys/types.h>
```

Data Structures

- struct DynamicPIDArray

  Structure for the dynamic array to store child PIDs.

Functions

- DynamicPIDArray * dynamic_pid_array_create (size_t initial_size)

  Initializes a new dynamic array for PIDs with an initial size.
- void dynamic_pid_array_destroy (DynamicPIDArray *array)

  Destroys a dynamic PID array, freeing its resources.
- bool dynamic_pid_array_add (DynamicPIDArray *array, pid_t pid)

  Adds a PID to the dynamic array, resizing if necessary.
- bool dynamic_pid_array_remove (DynamicPIDArray *array, pid_t pid)

  Removes a PID from the dynamic array.
- bool dynamic_pid_array_contains (const DynamicPIDArray *array, pid_t pid)

  Checks if a PID exists in the dynamic array.

10.17.1 Function Documentation

10.17.1.1 dynamic_pid_array_add()

```
bool dynamic_pid_array_add (
            DynamicPIDArray * array,
            pid_t pid )
```

Adds a PID to the dynamic array, resizing if necessary.

Parameters

array	A pointer to the dynamic PID array.
pid	The PID to add to the array.

Returns

true on success, false on failure (e.g., if memory allocation fails).

10.17.1.2 dynamic_pid_array_contains()

```
bool dynamic_pid_array_contains (
            const DynamicPIDArray * array,
            pid_t pid )
```

Checks if a PID exists in the dynamic array.

Parameters

array	A pointer to the dynamic PID array.
pid	The PID to check for in the array.

Returns

true if the PID exists in the array, false otherwise.

10.17.1.3 dynamic_pid_array_create()

```
DynamicPIDArray * dynamic_pid_array_create (
            size_t initial_size )
```

Initializes a new dynamic array for PIDs with an initial size.

Parameters

initial_size	The initial size of the dynamic array.

Returns

DynamicPIDArray* A pointer to the newly created dynamic array structure.

10.17.1.4 dynamic_pid_array_destroy()

```
void dynamic_pid_array_destroy (
            DynamicPIDArray * array )
```

Destroys a dynamic PID array, freeing its resources.

Parameters

array	A pointer to the dynamic PID array to be destroyed.

10.17.1.5 dynamic_pid_array_remove()

```
bool dynamic_pid_array_remove (
            DynamicPIDArray * array,
            pid_t pid )
```

Removes a PID from the dynamic array.

Parameters

array	A pointer to the dynamic PID array.
pid	The PID to remove from the array.

Returns

true if the PID was successfully removed, false if the PID was not found.

10.18 array.h

Go to the documentation of this file.

```
00001 #ifndef ARRAY_H
00002 #define ARRAY_H
00003
00004 #include <stdbool.h>    // For bool type
00005 #include <stddef.h>     // For size_t
00006 #include <sys/types.h>  //needed for ssize_t, if we use ints, can remove
00007
00011 typedef struct {
00012   pid_t* array;
00013   size_t used;
00014   size_t size;
00015 } DynamicPIDArray;
00016
00024 DynamicPIDArray* dynamic_pid_array_create(size_t initial_size);
00025
00031 void dynamic_pid_array_destroy(DynamicPIDArray* array);
00032
00040 bool dynamic_pid_array_add(DynamicPIDArray* array, pid_t pid);
00041
00050 bool dynamic_pid_array_remove(DynamicPIDArray* array, pid_t pid);
00051
00059 bool dynamic_pid_array_contains(const DynamicPIDArray* array, pid_t pid);
00060
00061 #endif
```

10.19 src/util/bitmap.c File Reference

```
#include "bitmap.h"
#include <string.h>
```

Functions

- void fd_bitmap_initialize (FD_Bitmap *bitmap)

  Initializes the bitmap to all zeros, indicating that no file descriptors are in use.
- bool fd_bitmap_set (FD_Bitmap *bitmap, uint32_t fd)

  Sets the bit for a given file descriptor, indicating it is now in use.
- bool fd_bitmap_clear (FD_Bitmap *bitmap, uint32_t fd)

  Clears the bit for a given file descriptor, indicating it is no longer in use.
- bool fd_bitmap_test (const FD_Bitmap *bitmap, uint32_t fd)

  Tests whether the bit for a given file descriptor is set, indicating whether it is in use.

10.19.1 Function Documentation

10.19.1.1 fd_bitmap_clear()

```
bool fd_bitmap_clear (
            FD_Bitmap * bitmap,
            uint32_t fd )
```

Clears the bit for a given file descriptor, indicating it is no longer in use.

Parameters

bitmap	A pointer to the file descriptor bitmap.
fd	The file descriptor to mark as not in use.

Returns

bool True if the operation was successful, false if the fd is out of range.

10.19.1.2 fd_bitmap_initialize()

```
void fd_bitmap_initialize (
            FD_Bitmap * bitmap )
```

Initializes the bitmap to all zeros, indicating that no file descriptors are in use.

Parameters

bitmap	A pointer to the file descriptor bitmap to initialize.

10.19.1.3 fd_bitmap_set()

```
bool fd_bitmap_set (
            FD_Bitmap * bitmap,
            uint32_t fd )
```

Sets the bit for a given file descriptor, indicating it is now in use.

Parameters

bitmap	A pointer to the file descriptor bitmap.
fd	The file descriptor to mark as in use.

Returns

bool True if the operation was successful, false if the fd is out of range.

10.19.1.4 fd_bitmap_test()

```
bool fd_bitmap_test (
            const FD_Bitmap * bitmap,
            uint32_t fd )
```

Tests whether the bit for a given file descriptor is set, indicating whether it is in use.

Parameters

bitmap	A pointer to the file descriptor bitmap.
fd	The file descriptor to check.

Returns

bool True if the file descriptor is in use, false otherwise.

10.20 src/util/bitmap.h File Reference

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

Data Structures

- struct FD_Bitmap

  Structure for managing open file descriptors using a bitmap.

Macros

- #define FD_BITMAP_SIZE 1024
- #define FD_BITMAP_BYTES (FD_BITMAP_SIZE / 8)

Functions

- void fd_bitmap_initialize (FD_Bitmap *bitmap)

  Initializes the bitmap to all zeros, indicating that no file descriptors are in use.
- bool fd_bitmap_set (FD_Bitmap *bitmap, uint32_t fd)

  Sets the bit for a given file descriptor, indicating it is now in use.
- bool fd_bitmap_clear (FD_Bitmap *bitmap, uint32_t fd)

  Clears the bit for a given file descriptor, indicating it is no longer in use.
- bool fd_bitmap_test (const FD_Bitmap *bitmap, uint32_t fd)

  Tests whether the bit for a given file descriptor is set, indicating whether it is in use.

10.20.1 Macro Definition Documentation

10.20.1.1 FD_BITMAP_BYTES

```
#define FD_BITMAP_BYTES (FD_BITMAP_SIZE / 8)
```

10.20.1.2 FD_BITMAP_SIZE

```
#define FD_BITMAP_SIZE 1024
```

10.20.2 Function Documentation

10.20.2.1 fd_bitmap_clear()

```
bool fd_bitmap_clear (
            FD_Bitmap * bitmap,
            uint32_t fd )
```

Clears the bit for a given file descriptor, indicating it is no longer in use.

Parameters

bitmap	A pointer to the file descriptor bitmap.
fd	The file descriptor to mark as not in use.

Returns

bool True if the operation was successful, false if the fd is out of range.

10.20.2.2 fd_bitmap_initialize()

```
void fd_bitmap_initialize (
            FD_Bitmap * bitmap )
```

Initializes the bitmap to all zeros, indicating that no file descriptors are in use.

Parameters

bitmap	A pointer to the file descriptor bitmap to initialize.

10.20.2.3 fd_bitmap_set()

```
bool fd_bitmap_set (
            FD_Bitmap * bitmap,
            uint32_t fd )
```

Sets the bit for a given file descriptor, indicating it is now in use.

Parameters

bitmap	A pointer to the file descriptor bitmap.
fd	The file descriptor to mark as in use.

Returns

bool True if the operation was successful, false if the fd is out of range.

10.20.2.4 fd_bitmap_test()

```
bool fd_bitmap_test (
            const FD_Bitmap * bitmap,
            uint32_t fd )
```

Tests whether the bit for a given file descriptor is set, indicating whether it is in use.

Parameters

bitmap	A pointer to the file descriptor bitmap.
fd	The file descriptor to check.

Returns

bool True if the file descriptor is in use, false otherwise.

10.21 bitmap.h

Go to the documentation of this file.

```
00001 #ifndef OPENFD_BITMAP_H
00002 #define OPENFD_BITMAP_H
00003
00004 #include <stdbool.h>  // For bool type
00005 #include <stdint.h>   // For uint8_t, uint32_t types
00006
00007 #define FD_BITMAP_SIZE 1024  // Maximum number of file descriptors
00008 #define FD_BITMAP_BYTES (FD_BITMAP_SIZE / 8)  // Number of bytes needed
```

```
00009
00013 typedef struct {
00014   uint8_t bits[FD_BITMAP_BYTES];
00015 } FD_Bitmap;
00016
00023 void fd_bitmap_initialize(FD_Bitmap* bitmap);
00024
00033 bool fd_bitmap_set(FD_Bitmap* bitmap, uint32_t fd);
00034
00044 bool fd_bitmap_clear(FD_Bitmap* bitmap, uint32_t fd);
00045
00054 bool fd_bitmap_test(const FD_Bitmap* bitmap, uint32_t fd);
00055
00056 #endif
```

10.22 src/util/clinkedlist.c File Reference

```
#include "clinkedlist.h"
#include <stdlib.h>
#include "kernel.h"
```

Functions

- CircularList * init_list (void)
- void add_process (CircularList *list, pcb_t *process)
- bool remove_process (CircularList *list, pid_t pid)
- pcb_t * find_process (CircularList *list, pid_t pid)

10.22.1 Function Documentation

10.22.1.1 add_process()

```
void add_process (
            CircularList * list,
            pcb_t * process )
```

Adds a new process to the circular linked list.

Parameters

list	Pointer to the circular linked list.
process	Pointer to the process control block (pcb_t) to add.

10.22.1.2 find_process()

```
pcb_t * find_process (
            CircularList * list,
            pid_t pid )
```

Finds a process in the circular linked list by its PID.

Parameters

list	Pointer to the circular linked list.
pid	PID of the process to find.

Returns

pcb_t* Pointer to the found process control block, or NULL if not found.

10.22.1.3 init_list()

```
CircularList * init_list (
            void  )
```

Initializes a circular linked list.

Returns

CircularList* Pointer to the newly initialized list.

10.22.1.4 remove_process()

```
bool remove_process (
            CircularList * list,
            pid_t pid )
```

Removes a process from the circular linked list by its PID.

Parameters

list	Pointer to the circular linked list.
pid	PID of the process to remove.

Returns

bool true if the process was successfully removed, false otherwise.

10.23 src/util/clinkedlist.h File Reference

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

Data Structures

- struct Node

  This structure represents a node in the circular linked list.
- struct CircularList

  This structure represents a circular linked list for managing processes.

Typedefs

- typedef struct pcb_t pcb_t
- typedef struct Node Node

Functions

- CircularList * init_list (void)
- void add_process (CircularList *list, pcb_t *process)
- bool remove_process (CircularList *list, pid_t pid)
- pcb_t * find_process (CircularList *list, pid_t pid)

10.23.1 Typedef Documentation

10.23.1.1 Node

```
typedef struct Node Node
```

10.23.1.2 pcb_t

```
typedef struct pcb_t pcb_t
```

10.23.2 Function Documentation

10.23.2.1 add_process()

```
void add_process (
            CircularList * list,
            pcb_t * process )
```

Adds a new process to the circular linked list.

Parameters

list	Pointer to the circular linked list.
process	Pointer to the process control block (pcb_t) to add.

10.23.2.2 find_process()

```
pcb_t * find_process (
            CircularList * list,
            pid_t pid )
```

Finds a process in the circular linked list by its PID.

Parameters

list	Pointer to the circular linked list.
pid	PID of the process to find.

Returns

pcb_t* Pointer to the found process control block, or NULL if not found.

10.23.2.3 init_list()

```
CircularList * init_list (
            void  )
```

Initializes a circular linked list.

Returns

CircularList* Pointer to the newly initialized list.

10.23.2.4 remove_process()

```
bool remove_process (
            CircularList * list,
            pid_t pid )
```

Removes a process from the circular linked list by its PID.

Parameters

list	Pointer to the circular linked list.
pid	PID of the process to remove.

Returns

bool true if the process was successfully removed, false otherwise.

10.24 clinkedlist.h

Go to the documentation of this file.

```
00001 #ifndef SCHEDULER_LIST_H
00002 #define SCHEDULER_LIST_H
00003
00004 #include <stdbool.h>
00005 #include <sys/types.h>
00006
00007 typedef struct pcb_t pcb_t;
00008
00013 typedef struct Node {
00014   pcb_t* process;
00015   struct Node* next;
```

```
00016 } Node;
00017
00023 typedef struct {
00024   Node* head;
00025   unsigned int size;
00026 } CircularList;
00027
00032 CircularList* init_list(void);
00033
00039 void add_process(CircularList* list, pcb_t* process);
00040
00047 bool remove_process(CircularList* list, pid_t pid);
00048
00056 pcb_t* find_process(CircularList* list, pid_t pid);
00057
00058 #endif  // SCHEDULER_LIST_H
```

10.25 src/util/error.h File Reference

Functions

- void u_perror (char ∗message)

  This is an analogue of the perror(3) function. This allows the shell or other function to pass in a message describing what error occured, which the function then concatenates to the default error message based on your errno value. The filesystem and kernel will both set errno and return -1 on system calls if they fail, after which u_perror() can be called.

Variables

- int errno

10.25.1 Function Documentation

10.25.1.1 u_perror()

```
void u_perror (
            char * message )
```

This is an analogue of the perror(3) function. This allows the shell or other function to pass in a message describing what error occured, which the function then concatenates to the default error message based on your errno value. The filesystem and kernel will both set errno and return -1 on system calls if they fail, after which u_perror() can be called.

Parameters

message	This is the message that will be concatenated to the default error message.

10.25.2 Variable Documentation

10.25.2.1 errno

```
int errno  [extern]
```

This is an integer that can be set by various functions to denote the kind of error that occurred. This is similar to that defined by errno.h, except that it is a custom definition.

10.26 error.h

Go to the documentation of this file.

```
00001 #ifndef ERROR
00002 #define ERROR
00009 extern int errno;
00010
00021 void u_perror(char* message);
00022
00023 #endif
```

10.27 src/util/globals.c File Reference

```
#include "globals.h"
```

Variables

- CircularList * processes [3]
- CircularList * blocked
- CircularList * stopped
- CircularList * zombied
- pcb_t * current = NULL
- pid_t next_pid = 1
- int logfiledescriptor = 0
- unsigned int tick = 0

10.27.1 Variable Documentation

10.27.1.1 blocked

```
CircularList* blocked
```

A global pointer to the process list of blocked processes. Processes enter this list via s_waitpid() or s_sleep().

10.27.1.2 current

```
pcb_t* current = NULL
```

This is the currently scheduled process. It can be accessed by any method to easily access the current method.

10.27.1.3 logfiledescriptor

```
int logfiledescriptor = 0
```

This is the int representing the file descriptor of the log file, to be used for writing purposes for the logging of events.

10.27.1.4 next_pid

```
pid_t next_pid = 1
```

This the next pid to be used, by k_proc_create(), to ensure that PIDs are not duplicated. This may be rewritten later to reuse/reallocate old processes that have been exited/terminated. I have no strong desire to do so, but do so if you wish.

10.27.1.5 processes

```
CircularList* processes[3]
```

A global array of pointers to the process lists. Each priority level can be accessed via processes[priority]. Processes enter this list after creation or via s_kill after receiving P_SIGCONT when stopped.

10.27.1.6 stopped

```
CircularList* stopped
```

A global pointer to the process list of stopped processes. Processes enter this list via s_kill() after receiving a P_SIGTERM signal.

10.27.1.7 tick

```
unsigned int tick = 0
```

This is an int representing the current tick of pennos, to be used for logging purposes.

10.27.1.8 zombied

```
CircularList* zombied
```

A global pointer to the process list of zombied/terminated processes. These processes enter this list via s_exit() or s_kill(), with the P_SIGTERM signal.

10.28 src/util/globals.h File Reference

```
#include "clinkedlist.h"
#include "kernel.h"
#include "prioritylist.h"
```

Variables

- CircularList * processes [3]
- CircularList * blocked
- CircularList * stopped
- CircularList * zombied
- pcb_t * current
- pid_t next_pid
- int logfiledescriptor
- unsigned int tick

10.28.1 Variable Documentation

10.28.1.1 blocked

```
CircularList* blocked  [extern]
```

A global pointer to the process list of blocked processes. Processes enter this list via s_waitpid() or s_sleep().

10.28.1.2 current

```
pcb_t* current  [extern]
```

This is the currently scheduled process. It can be accessed by any method to easily access the current method.

10.28.1.3 logfiledescriptor

```
int logfiledescriptor  [extern]
```

This is the int representing the file descriptor of the log file, to be used for writing purposes for the logging of events.

10.28.1.4 next_pid

```
pid_t next_pid  [extern]
```

This the next pid to be used, by k_proc_create(), to ensure that PIDs are not duplicated. This may be rewritten later to reuse/reallocate old processes that have been exited/terminated. I have no strong desire to do so, but do so if you wish.

10.28.1.5 processes

```
CircularList* processes[3]  [extern]
```

A global array of pointers to the process lists. Each priority level can be accessed via processes[priority]. Processes enter this list after creation or via s_kill after receiving P_SIGCONT when stopped.

10.28.1.6 stopped

```
CircularList* stopped  [extern]
```

A global pointer to the process list of stopped processes. Processes enter this list via s_kill() after receiving a P_SIGTERM signal.

10.28.1.7 tick

```
unsigned int tick  [extern]
```

This is an int representing the current tick of pennos, to be used for logging purposes.

10.28.1.8 zombied

```
CircularList* zombied  [extern]
```

A global pointer to the process list of zombied/terminated processes. These processes enter this list via s_exit() or s_kill(), with the P_SIGTERM signal.

10.29 globals.h

Go to the documentation of this file.

```
00001 #ifndef GLOBALS_H
00002 #define GLOBALS_H
00003
00004 #include "clinkedlist.h"
00005 #include "kernel.h"
00006 #include "prioritylist.h"
00007
00014 extern CircularList* processes[3];
00015
00020 extern CircularList* blocked;
00021
00026 extern CircularList* stopped;
00027
00033 extern CircularList* zombied;
00034
00040 extern pcb_t* current;
00041
00049 extern pid_t next_pid;
00050
00056 extern int logfiledescriptor;
00057
00063 extern unsigned int tick;
00064
00065 #endif
```

10.30 src/util/kernel.c File Reference

```
#include "kernel.h"
#include "stdio.h"
```

Functions

- pcb_t * k_proc_create (pcb_t *parent)

  Create a new child process, inheriting applicable properties from the parent.
- void k_proc_cleanup (pcb_t *proc)

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

10.30.1 Function Documentation

10.30.1.1 k_proc_cleanup()

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

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

Parameters

proc	This is a pointer to the process control block of a process that has terminated.

10.30.1.2 k_proc_create()

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

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

Parameters

parent	This is a pointer to the process control block of the parent, from which it inherits.

Returns

Reference to the child PCB.

10.31 src/util/kernel.h File Reference

```
#include <sys/types.h>
#include "array.h"
#include "bitmap.h"
#include "clinkedlist.h"
#include "globals.h"
#include "spthread.h"
#include "stdlib.h"
```

Data Structures

- struct pcb_t

  This structure stores all required information about a running process.

Macros

- #define P_SIGSTOP 0

  This is the STOP signal definition to be used by s_kill(). Running processes (ONLY) that receive the P_SIGSTOP signal will become stopped and have their state and process list adjusted accordingly. Note that statechanged will NOT be changed, as this state transition does NOT cause s_waitpid() to return/unblock.
- #define P_SIGCONT 1

  This is the CONTINUE signal definition to be used by s_kill(). Stopped processes (ONLY) that receive the P_SIGCONT signal will become running and have their state and process list adjusted accordingly. Note that statechanged will NOT be changed, as this state transition does NOT cause s_waitpid() to return/unblock.
- #define P_SIGTER 2

  This is the TERMINATE signal definition to be used by s_kill(). Any process that receives the P_SIGTER signal will become zombied and have their state and process list adjusted accordingly. Note that statechanged WILL be changed, as this state transition DOES cause s_waitpid() to return/unblock.

Typedefs

- typedef struct pcb_t pcb_t

Enumerations

- enum process_state_t { RUNNING , STOPPED , BLOCKED , ZOMBIED }

  Defines the possible states of a process in the system.

Functions

- pcb_t * k_proc_create (pcb_t *parent)

  Create a new child process, inheriting applicable properties from the parent.
- void k_proc_cleanup (pcb_t *proc)

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

10.31.1 Macro Definition Documentation

10.31.1.1 P_SIGCONT

```
#define P_SIGCONT 1
```

This is the CONTINUE signal definition to be used by s_kill(). Stopped processes (ONLY) that receive the P_SIGCONT signal will become running and have their state and process list adjusted accordingly. Note that statechanged will NOT be changed, as this state transition does NOT cause s_waitpid() to return/unblock.

10.31.1.2 P_SIGSTOP

```
#define P_SIGSTOP 0
```

This is the STOP signal definition to be used by s_kill(). Running processes (ONLY) that receive the P_SIGSTOP signal will become stopped and have their state and process list adjusted accordingly. Note that statechanged will NOT be changed, as this state transition does NOT cause s_waitpid() to return/unblock.

10.31.1.3 P_SIGTER

```
#define P_SIGTER 2
```

This is the TERMINATE signal definition to be used by s_kill(). Any process that receives the P_SIGTER signal will become zombied and have their state and process list adjusted accordingly. Note that statechanged WILL be changed, as this state transition DOES cause s_waitpid() to return/unblock.

10.31.2 Typedef Documentation

10.31.2.1 pcb_t

```
typedef struct pcb_t pcb_t
```

10.31.3 Enumeration Type Documentation

10.31.3.1 process_state_t

```
enum process_state_t
```

Defines the possible states of a process in the system.

This enumeration lists all the possible states that a process could be in at any given time. It is used within the pcb_t structure to track the current state of each process.

Enumerator

RUNNING	Process is currently executing. A process enters the RUNNING state when the scheduler selects it for execution, typically from the READY state.
STOPPED	Process is not executing, but can be resumed. A process should only become STOPPED if signaled by s_kill, recieving the P_SIGSTOP signal.
BLOCKED	Process is not executing, waiting for an event to occur. A process should only be blocked if it made a call to either s_waitpid or s_sleep.
ZOMBIED	Process has finished execution but awaits resource cleanup. A process enters the ZOMBIED state after it has finished its execution and is waiting for the parent process to read its exit status. If the parent process ever exits prior to reading exit status, this process should immediately cleaned up.

10.31.4 Function Documentation

10.31.4.1 k_proc_cleanup()

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

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

Parameters

proc	This is a pointer to the process control block of a process that has terminated.

10.31.4.2 k_proc_create()

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

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

Parameters

parent	This is a pointer to the process control block of the parent, from which it inherits.

Returns

Reference to the child PCB.

10.32 kernel.h

Go to the documentation of this file.

```
00001 #ifndef KERNEL_H
00002 #define KERNEL_H
00003
00004 #include <sys/types.h>  //needed for ssize_t, if we use ints, can remove
00005 #include "array.h"
00006 #include "bitmap.h"
00007 #include "clinkedlist.h"
00008 #include "globals.h"
00009 #include "spthread.h"
00010 #include "stdlib.h"
00011
00020 typedef enum {
00021   RUNNING,
00025   STOPPED,
00030   BLOCKED,
00035   ZOMBIED
00042 } process_state_t;
00043
00052 #define P_SIGSTOP 0
00053
00061 #define P_SIGCONT 1
00062
00070 #define P_SIGTER 2
00071
00077 typedef struct pcb_t {
00078   spthread_t handle;
00079   pid_t pid;
00080   pid_t ppid;
```

```
00081   DynamicPIDArray* child_pids;
00083   unsigned int priority : 2;
00085   process_state_t
00086       state;
00087   FD_Bitmap* open_fds;
00089   bool statechanged;
00092   int exit_status;
00094   int term_signal;
00096   bool waiting_for_change;
00098   pid_t waiting_on_pid;
00100   unsigned int ticks_to_wait;
00103   char* processname;
00104 } pcb_t;
00105
00113 pcb_t* k_proc_create(pcb_t* parent);
00114
00121 void k_proc_cleanup(pcb_t* proc);
00122
00123 #endif
```

10.33 src/util/pennfat_kernel.c File Reference

```
#include "pennfat_kernel.h"
#include "unistd.h"
```

Functions

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

  Open file name `fname` with the mode `mode`, and return a file descriptor to that file.
- struct directory_entries * does_file_exist (const char *fname)
- void move_to_open_de (bool found)
- off_t does_file_exist2 (const char *fname)
- int get_first_empty_fat_index ()
- void lseek_to_root_directory ()
- struct file_descriptor_st * get_file_descriptor (int fd)
- struct file_descriptor_st * create_file_descriptor (int fd, char *fname, int mode, int offset)
- struct directory_entries * create_directory_entry (const char *name, uint32_t size, uint16_t firstBlock, uint8_t type, uint8_t perm, time_t mtime)
- ssize_t k_read (int fd, int n, char *buf)
- void extend_fat (int start_index, int empty_fat_index)
- void write_one_byte_in_while (int bytes_left, int size, int *size_increment, int *bytes_written, int *current_offset, const char *str, uint16_t firstBlock)
- void update_directory_entry_after_write (struct directory_entries *curr_de, char *fname, int bytes_written)
- ssize_t k_write (int fd, const char *str, int n)
- int k_close (int fd)
- int k_unlink (const char *fname)
- off_t k_lseek (int fd, int offset, int whence)
- void generate_permission (uint8_t perm, char **permissions)
- char * formatTime (time_t t)
- void k_ls (const char *filename)
- void k_rename (const char *source, const char *dest)
- void k_change_mode (const char *change, const char *filename)
- char * k_read_all (const char *filename, int *read_num)

Variables

- uint16_t * fat = NULL
- struct file_descriptor_st * global_fd_table = NULL
- int fs_fd = -1
- int block_size = 0
- int fat_size = 0
- int num_fat_entries = 0
- int data_size = 0
- int fd_counter = 3

10.33.1 Function Documentation

10.33.1.1 create_directory_entry()

```
struct directory_entries * create_directory_entry (
            const char * name,
            uint32_t size,
            uint16_t firstBlock,
            uint8_t type,
            uint8_t perm,
            time_t mtime )
```

10.33.1.2 create_file_descriptor()

```
struct file_descriptor_st * create_file_descriptor (
            int fd,
            char * fname,
            int mode,
            int offset )
```

10.33.1.3 does_file_exist()

```
struct directory_entries * does_file_exist (
            const char * fname )
```

10.33.1.4 does_file_exist2()

```
off_t does_file_exist2 (
            const char * fname )
```

10.33.1.5 extend_fat()

```
void extend_fat (
            int start_index,
            int empty_fat_index )
```

10.33.1.6 formatTime()

```
char * formatTime (
            time_t t )
```

10.33.1.7 generate_permission()

```
void generate_permission (
            uint8_t perm,
            char ** permissions )
```

10.33.1.8 get_file_descriptor()

```
struct file_descriptor_st * get_file_descriptor (
            int fd )
```

10.33.1.9 get_first_empty_fat_index()

```
int get_first_empty_fat_index ( )
```

10.33.1.10 k_change_mode()

```
void k_change_mode (
            const char * change,
            const char * filename )
```

10.33.1.11 k_close()

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

Parameters

fd	

Returns

int

10.33.1.12 k_ls()

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

Parameters

filename	

10.33.1.13 k_lseek()

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

Parameters

fd	
offset	
whence	

Returns

off_t

10.33.1.14 k_open()

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

Open file name `fname` with the mode `mode`, and return a file descriptor to that file.

This function opens a file specified by the file name `fname` in the mode specified by `mode` and returns a file descriptor associated with the open file that can be used for subsequent file operations.

Parameters

fname	The name of the file to open. See POSIX standard for allowed names.
mode	The mode with which to open the file. This should specify the access mode (e.g., read, write) and other flags as defined by the operating system. Allowed modes are: write (`F_WRITE`), read (`F_READ`), and append (`F_APPEND`).

Returns

int A non-negative file descriptor on success, or -1 on error and `errno` set.

Note

The `mode` parameter may only be `F_WRITE`, `F_READ`, or `F_APPEND`. Note that despite their names, write and append support both reading and writing. `F_APPEND`'s file pointer will point to the end of the file rather than the beginning. Both `F_WRITE` and `F_APPEND` will create the named file if it does not already exist.

See also

https://www.ibm.com/docs/en/zos/3.1.0?topic=locales-posix-portable-file-name-charac

Possible values of `errno` are:

- `EACCES` : // need to fill these in, will expand as further progress
- `ENAMETOOLONG` :

10.33.1.15 k_read()

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

Parameters

fd	A
n	
buf	

Returns

ssize_t

10.33.1.16 k_read_all()

```
char * k_read_all (
            const char * filename,
            int * read_num )
```

10.33.1.17 k_rename()

```
void k_rename (
            const char * source,
            const char * dest )
```

10.33.1.18 k_unlink()

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

Parameters

fname	

Returns

int

10.33.1.19 k_write()

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

Parameters

fd	
str	
n	

Returns

ssize_t

10.33.1.20 lseek_to_root_directory()

```
void lseek_to_root_directory ( )
```

10.33.1.21 move_to_open_de()

```
void move_to_open_de (
            bool found )
```

10.33.1.22 update_directory_entry_after_write()

```
void update_directory_entry_after_write (
            struct directory_entries * curr_de,
            char * fname,
            int bytes_written )
```

10.33.1.23 write_one_byte_in_while()

```
void write_one_byte_in_while (
            int bytes_left,
            int size,
            int * size_increment,
            int * bytes_written,
            int * current_offset,
            const char * str,
            uint16_t firstBlock )
```

10.33.2 Variable Documentation

10.33.2.1 block_size

```
int block_size = 0
```

10.33.2.2 data_size

```
int data_size = 0
```

10.33.2.3 fat

```
uint16_t* fat = NULL
```

10.33.2.4 fat_size

```
int fat_size = 0
```

10.33.2.5 fd_counter

```
int fd_counter = 3
```

10.33.2.6 fs_fd

```
int fs_fd = -1
```

10.33.2.7 global_fd_table

```
struct file_descriptor_st* global_fd_table = NULL
```

10.33.2.8 num_fat_entries

```
int num_fat_entries = 0
```

10.34 src/util/pennfat_kernel.h File Reference

```
#include <stdint.h>
#include <sys/types.h>
#include "../pennfat.h"
#include "spthread.h"
```

Data Structures

- struct directory_entries
- struct file_descriptor_st

Macros

- #define READ_WRITE 0
- #define READ 1
- #define WRITE 2
- #define APPEND 3
- #define MAX_FD_NUM 1024

Enumerations

- enum Whence { F_SEEK_SET , F_SEEK_CUR , F_SEEK_END }

Functions

- struct file_descriptor_st * create_file_descriptor (int fd, char *fname, int mode, int offset)
- struct directory_entries * create_directory_entry (const char *name, uint32_t size, uint16_t firstBlock, uint8_t type, uint8_t perm, time_t mtime)
- void lseek_to_root_directory ()
- void extend_fat (int start_index, int empty_fat_index)
- int get_first_empty_fat_index ()
- void move_to_open_de (bool found)
- struct directory_entries * does_file_exist (const char *fname)
- off_t does_file_exist2 (const char *fname)
- struct file_descriptor_st * get_file_descriptor (int fd)
- int k_open (const char *fname, int mode)

  Open file name `fname` with the mode `mode`, and return a file descriptor to that file.
- ssize_t k_read (int fd, int n, char *buf)
- ssize_t k_write (int fd, const char *str, int n)
- int k_close (int fd)
- int k_unlink (const char *fname)
- off_t k_lseek (int fd, int offset, int whence)
- void k_ls (const char *filename)
- void k_rename (const char *source, const char *dest)
- void k_change_mode (const char *change, const char *filename)
- char * k_read_all (const char *filename, int *read_num)

Variables

- uint16_t * fat
- struct file_descriptor_st * global_fd_table
- int fs_fd
- int block_size
- int fat_size
- int num_fat_entries
- int data_size

10.34.1 Macro Definition Documentation

10.34.1.1 APPEND

```
#define APPEND 3
```

10.34.1.2 MAX_FD_NUM

```
#define MAX_FD_NUM 1024
```

10.34.1.3 READ

```
#define READ 1
```

10.34.1.4 READ_WRITE

```
#define READ_WRITE 0
```

10.34.1.5 WRITE

```
#define WRITE 2
```

10.34.2 Enumeration Type Documentation

10.34.2.1 Whence

```
enum Whence
```

Enumerator

F_SEEK_SET	
F_SEEK_CUR	
F_SEEK_END	

10.34.3 Function Documentation

10.34.3.1 create_directory_entry()

```
struct directory_entries * create_directory_entry (
            const char * name,
            uint32_t size,
            uint16_t firstBlock,
```

```
            uint8_t type,
            uint8_t perm,
            time_t mtime )
```

10.34.3.2 create_file_descriptor()

```
struct file_descriptor_st * create_file_descriptor (
            int fd,
            char * fname,
            int mode,
            int offset )
```

10.34.3.3 does_file_exist()

```
struct directory_entries * does_file_exist (
            const char * fname )
```

10.34.3.4 does_file_exist2()

```
off_t does_file_exist2 (
            const char * fname )
```

10.34.3.5 extend_fat()

```
void extend_fat (
            int start_index,
            int empty_fat_index )
```

10.34.3.6 get_file_descriptor()

```
struct file_descriptor_st * get_file_descriptor (
            int fd )
```

10.34.3.7 get_first_empty_fat_index()

```
int get_first_empty_fat_index ( )
```

10.34.3.8 k_change_mode()

```
void k_change_mode (
            const char * change,
            const char * filename )
```

10.34.3.9 k_close()

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

Parameters

fd	

Returns

int

10.34.3.10 k_ls()

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

Parameters

filename	

10.34.3.11 k_lseek()

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

Parameters

fd	
offset	
whence	

Returns

off_t

10.34.3.12 k_open()

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

Open file name `fname` with the mode `mode`, and return a file descriptor to that file.

This function opens a file specified by the file name `fname` in the mode specified by `mode` and returns a file descriptor associated with the open file that can be used for subsequent file operations.

Parameters

fname	The name of the file to open. See POSIX standard for allowed names.
mode	The mode with which to open the file. This should specify the access mode (e.g., read, write) and other flags as defined by the operating system. Allowed modes are: write (`F_WRITE`), read (`F_READ`), and append (`F_APPEND`).

Returns

int A non-negative file descriptor on success, or -1 on error and `errno` set.

Note

The `mode` parameter may only be `F_WRITE`, `F_READ`, or `F_APPEND`. Note that despite their names, write and append support both reading and writing. `F_APPEND`'s file pointer will point to the end of the file rather than the beginning. Both `F_WRITE` and `F_APPEND` will create the named file if it does not already exist.

See also

https://www.ibm.com/docs/en/zos/3.1.0?topic=locales-posix-portable-file-name-charac

Possible values of `errno` are:

- `EACCES` : // need to fill these in, will expand as further progress
- `ENAMETOOLONG` :

10.34.3.13 k_read()

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

Parameters

fd	A
n	
buf	

Returns

ssize_t

10.34.3.14 k_read_all()

```
char * k_read_all (
            const char * filename,
            int * read_num )
```

10.34.3.15 k_rename()

```
void k_rename (
            const char * source,
            const char * dest )
```

10.34.3.16 k_unlink()

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

Parameters

fname	

Returns

int

10.34.3.17 k_write()

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

Parameters

fd	
str	
n	

Returns

ssize_t

10.34.3.18 lseek_to_root_directory()

```
void lseek_to_root_directory ( )
```

10.34.3.19 move_to_open_de()

```
void move_to_open_de (
            bool found )
```

10.34.4 Variable Documentation

10.34.4.1 block_size

```
int block_size  [extern]
```

10.34.4.2 data_size

```
int data_size  [extern]
```

10.34.4.3 fat

```
uint16_t* fat  [extern]
```

10.34.4.4 fat_size

```
int fat_size  [extern]
```

10.34.4.5 fs_fd

```
int fs_fd  [extern]
```

10.34.4.6 global_fd_table

```
struct file_descriptor_st* global_fd_table  [extern]
```

10.34.4.7 num_fat_entries

```
int num_fat_entries  [extern]
```

10.35 pennfat_kernel.h

Go to the documentation of this file.

```
00001 #ifndef PENNFAT_KERNEL_H
00002 #define PENNFAT_KERNEL_H
00003
00004 #include <stdint.h>
00005 #include <sys/types.h>  //needed for ssize_t, if we use ints, can remove
00006 #include "../pennfat.h"
00007 #include "spthread.h"
00008
00009 #define READ_WRITE 0
00010 #define READ 1
00011 #define WRITE 2
00012 #define APPEND 3
00013
00014 #define MAX_FD_NUM 1024
00015
00016 enum Whence { F_SEEK_SET, F_SEEK_CUR, F_SEEK_END };
00017
00018 extern uint16_t* fat;
00019 extern struct file_descriptor_st* global_fd_table;
00020 extern int fs_fd;
00021 extern int block_size;
00022 extern int fat_size;
00023 extern int num_fat_entries;
00024 extern int data_size;
00025
00026 struct directory_entries {
00027   char name[32];
00028   uint32_t size;
00029   uint16_t firstBlock;
00030   uint8_t type;
00031   uint8_t perm;
00032   time_t mtime;
00033   uint8_t reserved[16];
00034 };
00035
00036 struct file_descriptor_st {
00037   int fd;
00038   char* fname;
00039   int mode;
00040   int offset;
00041   int ref_cnt;
00042 };
00043
00044 // helper functions
00045 struct file_descriptor_st* create_file_descriptor(int fd,
00046                                                   char* fname,
00047                                                   int mode,
00048                                                   int offset);
00049 struct directory_entries* create_directory_entry(const char* name,
00050                                                  uint32_t size,
00051                                                  uint16_t firstBlock,
00052                                                  uint8_t type,
00053                                                  uint8_t perm,
00054                                                  time_t mtime);
00055 void lseek_to_root_directory();
00056 void extend_fat(int start_index, int empty_fat_index);
00057 int get_first_empty_fat_index();
00058 void move_to_open_de(bool found);
00059 struct directory_entries* does_file_exist(const char* fname);
00060 off_t does_file_exist2(const char* fname);
00061 struct file_descriptor_st* get_file_descriptor(int fd);
00062
00063 /*********************************************
00064  *  PENNFAT KERNEL LEVEL FUNCTIONS
00065  *********************************************/
00066
00102 int k_open(const char* fname, int mode);
00103
00112 ssize_t k_read(int fd, int n, char* buf);
00113
00122 ssize_t k_write(int fd, const char* str, int n);
00123
00130 int k_close(int fd);
00131
00138 int k_unlink(const char* fname);
00139
00148 off_t k_lseek(int fd, int offset, int whence);
00149
00155 void k_ls(const char* filename);
00156
00157 void k_rename(const char* source, const char* dest);
00158
```

```
00159 void k_change_mode(const char* change, const char* filename);
00160
00161 char* k_read_all(const char* filename, int* read_num);
00162
00163 #endif
```

10.36 src/util/prioritylist.c File Reference

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

Functions

- PList * init_priority (void)
- void add_priority (PList *list, unsigned int priority)
- bool remove_priority (PList *list, unsigned int priority)

10.36.1 Function Documentation

10.36.1.1 add_priority()

```
void add_priority (
            PList * list,
            unsigned int priority )
```

Adds a new process to the circular linked list.

Parameters

list	Pointer to the circular linked list.
process	Pointer to the process control block (pcb_t) to add.

10.36.1.2 init_priority()

```
PList * init_priority (
            void  )
```

Initializes a circular linked list.

Returns

CircularList* Pointer to the newly initialized list.

10.36.1.3 remove_priority()

```
bool remove_priority (
            PList * list,
            unsigned int priority )
```

Removes a process from the circular linked list by its PID.

Parameters

list	Pointer to the circular linked list.
pid	PID of the process to remove.

Returns

bool true if the process was successfully removed, false otherwise.

10.37 src/util/prioritylist.h File Reference

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

Data Structures

- struct PNode
- struct PList

Typedefs

- typedef struct PNode PNode

Functions

- PList * init_priority (void)
- void add_priority (PList *list, unsigned int priority)
- bool remove_priority (PList *list, unsigned int priority)

10.37.1 Typedef Documentation

10.37.1.1 PNode

```
typedef struct PNode PNode
```

10.37.2 Function Documentation

10.37.2.1 add_priority()

```
void add_priority (
            PList * list,
            unsigned int priority )
```

Adds a new process to the circular linked list.

Parameters

list	Pointer to the circular linked list.
process	Pointer to the process control block (pcb_t) to add.

10.37.2.2 init_priority()

```
PList * init_priority (
            void  )
```

Initializes a circular linked list.

Returns

CircularList* Pointer to the newly initialized list.

10.37.2.3 remove_priority()

```
bool remove_priority (
            PList * list,
            unsigned int priority )
```

Removes a process from the circular linked list by its PID.

Parameters

list	Pointer to the circular linked list.
pid	PID of the process to remove.

Returns

bool true if the process was successfully removed, false otherwise.

10.38 prioritylist.h

Go to the documentation of this file.

```
00001 #ifndef PLIST_H
00002 #define PLIST_H
00003
00004 #include <stdbool.h>
00005
00010 typedef struct PNode {
00011   unsigned int
00012       priority : 2;
00013   struct PNode* next;
00014 } PNode;
00015
00021 typedef struct {
00022   PNode* head;
00023   unsigned int size;
00024 } PList;
00025
00030 PList* init_priority(void);
00031
```

```
00037 void add_priority(PList* list, unsigned int priority);
00038
00045 bool remove_priority(PList* list, unsigned int priority);
00046
00047 #endif  // SCHEDULER_LIST_H
```

10.39 src/util/shellbuiltins.c File Reference

```
#include "shellbuiltins.h"
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include "errno.h"
#include "pennos.h"
#include "sys_call.h"
```

Functions

- void * b_sleep (void *arg)

  Sleep for n seconds.
- void * b_kill (void *arg)

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

  Adjust the priority level of an existing process.
- void * b_logout (void *arg)

  Exits the shell and shutsdown PennOS.

10.39.1 Function Documentation

10.39.1.1 b_kill()

```
void * b_kill (
            void * arg )
```

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

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

10.39.1.2 b_logout()

```
void * b_logout (
            void * arg )
```

Exits the shell and shutsdown PennOS.

Example Usage: logout

10.39.1.3 b_nice_pid()

```
void * b_nice_pid (
            void * arg )
```

Adjust the priority level of an existing process.

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

10.39.1.4 b_sleep()

```
void * b_sleep (
            void * arg )
```

Sleep for `n` seconds.

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

Example Usage: sleep 10

10.40 src/util/shellbuiltins.h File Reference

Functions

- void * b_cat (void *arg)

  The ususal `cat` program.
- void * b_sleep (void *arg)

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

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

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

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

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

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

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

  Change permissions of a file. There's no need to error if a permission being added already exists, or if a permission being removed is already not granted.
- void * b_ps (void *arg)

  List all processes on PennOS, displaying PID, PPID, priority, status, and command name.
- void * b_kill (void *arg)

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

- void * b_nice (void *arg)

  Spawn a new process for `command` and set its priority to `priority`.
- void * b_nice_pid (void *arg)

  Adjust the priority level of an existing process.
- void * b_man (void *arg)

  Lists all available commands.
- void * b_bg (void *arg)

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

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

  Lists all jobs.
- void * b_logout (void *arg)

  Exits the shell and shutsdown PennOS.
- void * b_zombify (void *arg)

  Used to test zombifying functionality of your kernel.
- void * b_zombie_child (void *arg)

  Helper for zombify.
- void * b_orphanify (void *arg)

  Used to test orphanifying functionality of your kernel.
- void * b_orphan_child (void *arg)

  Helper for orphanify.

10.40.1 Function Documentation

10.40.1.1 b_bg()

```
void * b_bg (
            void * arg )
```

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

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

10.40.1.2 b_busy()

```
void * b_busy (
            void * arg )
```

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

Example Usage: busy

10.40.1.3 b_cat()

```
void * b_cat (
            void * arg )
```

The ususal `cat` program.

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

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

10.40.1.4 b_chmod()

```
void * b_chmod (
            void * arg )
```

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

Print appropriate error message if:

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

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

10.40.1.5 b_cp()

```
void * b_cp (
            void * arg )
```

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

Print appropriate error message if:

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

Example Usage: cp src_file dst_file

10.40.1.6 b_echo()

```
void * b_echo (
            void * arg )
```

Echo back an input string.

Example Usage: echo Hello World

10.40.1.7 b_fg()

```
void * b_fg (
            void * arg )
```

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

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

10.40.1.8 b_jobs()

```
void * b_jobs (
            void * arg )
```

Lists all jobs.

Example Usage: jobs

10.40.1.9 b_kill()

```
void * b_kill (
            void * arg )
```

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

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

10.40.1.10 b_logout()

```
void * b_logout (
            void * arg )
```

Exits the shell and shutsdown PennOS.

Example Usage: logout

10.40.1.11 b_ls()

```
void * b_ls (
            void * arg )
```

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

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

10.40.1.12 b_man()

```
void * b_man (
            void * arg )
```

Lists all available commands.

Example Usage: man

10.40.1.13 b_mv()

```
void * b_mv (
            void * arg )
```

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

Print appropriate error message if:

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

Example Usage: mv src_file dst_file

10.40.1.14 b_nice()

```
void * b_nice (
            void * arg )
```

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

1. Adjust the priority level of an existing process.

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

10.40.1.15 b_nice_pid()

```
void * b_nice_pid (
            void * arg )
```

Adjust the priority level of an existing process.

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

10.40.1.16 b_orphan_child()

```
void * b_orphan_child (
            void * arg )
```

Helper for orphanify.

10.40.1.17 b_orphanify()

```
void * b_orphanify (
            void * arg )
```

Used to test orphanifying functionality of your kernel.

Example Usage: orphanify

10.40.1.18 b_ps()

```
void * b_ps (
            void * arg )
```

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

Example Usage: ps

10.40.1.19 b_rm()

```
void * b_rm (
            void * arg )
```

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

Print appropriate error message if:

- `file` is not a file that exists

Example Usage: rm f1 f2 f3 f4 f5

10.40.1.20 b_sleep()

```
void * b_sleep (
            void * arg )
```

Sleep for `n` seconds.

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

Example Usage: sleep 10

10.40.1.21 b_touch()

```
void * b_touch (
            void * arg )
```

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

Example Usage: touch f1 f2 f3 f4 f5

10.40.1.22 b_zombie_child()

```
void * b_zombie_child (
            void * arg )
```

Helper for zombify.

10.40.1.23 b_zombify()

```
void * b_zombify (
            void * arg )
```

Used to test zombifying functionality of your kernel.

Example Usage: zombify

10.41 shellbuiltins.h

Go to the documentation of this file.

```
00001 #ifndef SHELL_BUILTINS
00002 #define SHELL_BUILTINS
00003
00004 // SHELL BUILTINS: Implemented using user and system level functions only!
00005
00016 void* b_cat(void* arg);
00017
00026 void* b_sleep(void* arg);
00027
00034 void* b_busy(void* arg);
00035
00041 void* b_echo(void* arg);
00042
00050 void* b_ls(void* arg);
00051
00058 void* b_touch(void* arg);
00059
00070 void* b_mv(void* arg);
00071
00082 void* b_cp(void* arg);
00083
00094 void* b_rm(void* arg);
00095
00110 void* b_chmod(void* arg);
00111
00118 void* b_ps(void* arg);
00119
00130 void* b_kill(void* arg);
00131
00132 // SHELL BUILTINS THAT DON'T SPAWN PROCESSES
00133
00141 void* b_nice(void* arg);
00142
00148 void* b_nice_pid(void* arg);
00149
00155 void* b_man(void* arg);
00156
00164 void* b_bg(void* arg);
00165
00173 void* b_fg(void* arg);
00174
00180 void* b_jobs(void* arg);
00181
00187 void* b_logout(void* arg);
00188
00189 // SHELL BUILTINS TO TEST ZOMBIE + ORPHANS
00195 void* b_zombify(void* arg);
00196
00200 void* b_zombie_child(void* arg);
00201
00207 void* b_orphanify(void* arg);
00208
00212 void* b_orphan_child(void* arg);
00213
00214 #endif
```

10.42 src/util/spthread.c File Reference

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

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

Data Structures

- struct spthread_fwd_args_st
- struct spthread_signal_args_st
- struct spthread_meta_st

Macros

- #define _GNU_SOURCE
- #define _XOPEN_SOURCE 700
- #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)

10.42.1 Macro Definition Documentation

10.42.1.1 _GNU_SOURCE

```
#define _GNU_SOURCE
```

10.42.1.2 _XOPEN_SOURCE

```
#define _XOPEN_SOURCE 700
```

10.42.1.3 MILISEC_IN_NANO

```
#define MILISEC_IN_NANO 100000
```

10.42.1.4 SPTHREAD_RUNNING_STATE

```
#define SPTHREAD_RUNNING_STATE 0
```

10.42.1.5 SPTHREAD_SIG_CONTINUE

```
#define SPTHREAD_SIG_CONTINUE -2
```

10.42.1.6 SPTHREAD_SIG_SUSPEND

```
#define SPTHREAD_SIG_SUSPEND -1
```

10.42.1.7 SPTHREAD_SUSPENDED_STATE

```
#define SPTHREAD_SUSPENDED_STATE 1
```

10.42.1.8 SPTHREAD_TERMINATED_STATE

```
#define SPTHREAD_TERMINATED_STATE 2
```

10.42.2 Typedef Documentation

10.42.2.1 pthread_fn

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

10.42.2.2 spthread_fwd_args

```
typedef struct spthread_fwd_args_st spthread_fwd_args
```

10.42.2.3 spthread_meta_t

```
typedef struct spthread_meta_st spthread_meta_t
```

10.42.2.4 spthread_signal_args

```
typedef struct spthread_signal_args_st spthread_signal_args
```

10.42.3 Function Documentation

10.42.3.1 spthread_cancel()

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

10.42.3.2 spthread_continue()

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

10.42.3.3 spthread_create()

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

10.42.3.4 spthread_exit()

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

10.42.3.5 spthread_join()

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

10.42.3.6 spthread_self()

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

10.42.3.7 spthread_suspend()

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

10.42.3.8 spthread_suspend_self()

```
int spthread_suspend_self ( )
```

10.43 src/util/spthread.h File Reference

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

Data Structures

- 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)

10.43.1 Macro Definition Documentation

10.43.1.1 SIGPTHD

```
#define SIGPTHD SIGUSR1
```

10.43.2 Typedef Documentation

10.43.2.1 spthread_meta_t

```
typedef struct spthread_meta_st spthread_meta_t
```

10.43.2.2 spthread_t

```
typedef struct spthread_st spthread_t
```

10.43.3 Function Documentation

10.43.3.1 spthread_cancel()

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

10.43.3.2 spthread_continue()

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

10.43.3.3 spthread_create()

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

10.43.3.4 spthread_exit()

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

10.43.3.5 spthread_join()

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

10.43.3.6 spthread_self()

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

10.43.3.7 spthread_suspend()

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

10.43.3.8 spthread_suspend_self()

```
int spthread_suspend_self ( )
```

10.44 spthread.h

Go to the documentation of this file.

```
00001 #ifndef SPTHREAD_H_
00002 #define SPTHREAD_H_
00003
00004 #include <pthread.h>
00005 #include <stdbool.h>
00006
00007 // CAUTION: according to `man 7 pthread`:
00008 //
00009 //   On older Linux kernels, SIGUSR1 and SIGUSR2
00010 //   are used.  Applications must avoid the use of whichever set of
00011 //   signals is employed by the implementation.
00012 //
00013 // This may not work on other linux versions
00014
00015 // SIGNAL PTHREAD
00016 // NOTE: if within a created spthread you change
00017 // the behaviour of SIGUSR1, then you will not be able
00018 // to suspend and continue a spthread
00019 #define SIGPTHD SIGUSR1
00020
00021 // declares a struct, but the internals of the
00022 // struct cannot be seen by functions outside of spthread.c
00023 typedef struct spthread_meta_st spthread_meta_t;
00024
00025 // The spthread wrapper struct.
00026 // Sometimes you may have to access the inner pthread member
00027 // but you shouldn't need to do that
00028 typedef struct spthread_st {
00029   pthread_t thread;
00030   spthread_meta_t* meta;
00031 } spthread_t;
00032
00033 // NOTE:
00034 // None of these are signal safe
00035 // Also note that most of these functions are not safe to suspension,
00036 // meaning that if the thread calling these is an spthread and is suspended
00037 // in the middle of spthread_continue or spthread_suspend, then it may not work.
00038 //
00039 // Make sure that the calling thread cannot be suspended before calling these
00040 // functions. Exceptions to this are spthread_exit(), spthread_self() and if a
00041 // thread is continuing or suspending itself.
00042 // spthread_create:
00043 // this function works similar to pthread_create, except for two differences.
00044 // 1) the created pthread is able to be asychronously suspended, and continued
00045 //    using the functions:
00046 //      - spthread_suspend
00047 //      - spthread_continue
00048 // 2) The created pthread will be suspended before it executes the specified
00049 //    routine. It must first be continued with `spthread_continue` before
00050 //    it will start executing.
00051 //
00052 // It is worth noting that this function is not signal safe.
00053 // In other words, it should not be called from a signal handler.
00054 //
00055 // to avoid repetition, see pthread_create(3) for details
00056 // on arguments and return values as they are the same here.
00057 int spthread_create(spthread_t* thread,
00058                     const pthread_attr_t* attr,
00059                     void* (*start_routine)(void*),
00060                     void* arg);
00061
00062 // The spthread_suspend function will signal to the
00063 // specified thread to suspend execution.
00064 //
00065 // Calling spthread_suspend on an already suspended
00066 // thread does not do anything.
00067 //
00068 // It is worth noting that this function is not signal safe.
00069 // In other words, it should not be called from a signal handler.
00070 //
00071 // args:
00072 // - pthread_t thread: the thread we want to suspend
```

```
00073 //   This thread must be created using the spthread_create() function,
00074 //   if created by some other function, the behaviour is undefined.
00075 //
00076
00077 // returns:
00078 // - 0 on success
00079 // - EAGAIN if the thread could not be signaled
00080 // - ENOSYS if not supported on this system
00081 // - ESRCH if the thread specified is not a valid pthread
00082 int spthread_suspend(spthread_t thread);
00083
00084 // The spthread_suspend_self function will cause the calling
00085 // thread (which should be created by spthread_create) to suspend
00086 // itself.
00087 //
00088 // returns:
00089 // - 0 on success
00090 // - EAGAIN if the thread could not be signaled
00091 // - ENOSYS if not supported on this system
00092 // - ESRCH if the calling thread is not an spthread
00093 int spthread_suspend_self();
00094
00095 // The spthread_continue function will signal to the
00096 // specified thread to resume execution if suspended.
00097 //
00098 // Calling spthread_continue on an already non-suspended
00099 // thread does not do anything.
00100 //
00101 // It is worth noting that this function is not signal safe.
00102 // In other words, it should not be called from a signal handler.
00103 //
00104 // args:
00105 // - spthread_t thread: the thread we want to continue
00106 //   This thread must be created using the spthread_create() function,
00107 //   if created by some other function, the behaviour is undefined.
00108 //
00109 // returns:
00110 // - 0 on success
00111 // - EAGAIN if the thread could not be signaled
00112 // - ENOSYS if not supported on this system
00113 // - ESRCH if the thread specified is not a valid pthread
00114 int spthread_continue(spthread_t thread);
00115
00116 // The spthread_cancel function will send a
00117 // cancellation request to the specified thread.
00118 //
00119 // as of now, this function is identical to pthread_cancel(3)
00120 // so to avoid repitition, you should look there.
00121 //
00122 // Here are a few things that are worth highlighting:
00123 // - it is worth noting that it is a cancellation __request__
00124 //   the thread may not terminate immediately, instead the
00125 //   thread is checked whenever it calls a function that is
00126 //   marked as a cancellation point. At those points, it will
00127 //   start the cancellation procedure
00128 // - to make sure all things are de-allocated properly on
00129 //   normal exiting of the thread and when it is cancelled,
00130 //   you should mark a defered de-allocation with
00131 //   pthread_cleanup_push(3).
00132 //   consider the following example:
00133 //
00134 //     void* thread_routine(void* arg) {
00135 //        int* num = malloc(sizeof(int));
00136 //        pthread_cleanup_push(&free, num);
00137 //        return NULL;
00138 //     }
00139 //
00140 //    this program will allocate an integer on the heap
00141 //    and mark that data to be de-allocated on cleanup.
00142 //    This means that when the thread returns from the
00143 //    routine specified in spthread_create, free will
00144 //    be called on num. This will also happen if the thread
00145 //    is cancelled and not able to be exited normally.
00146 //
00147 //    Another function that should be used in conjunction
00148 //    is pthread_cleanup_pop(3). I will leave that
00149 //    to you to read more on.
00150 //
00151 // It is worth noting that this function is not signal safe.
00152 // In other words, it should not be called from a signal handler.
00153 //
00154 // args:
00155 // - spthread_t thread: the thread we want to cancel.
00156 //   This thread must be created using the spthread_create() function,
00157 //   if created by some other function, the behaviour is undefined.
00158 //
00159 // returns:
```

```
00160 // - 0 on success
00161 // - ESRCH if the thread specified is not a valid pthread
00162 int spthread_cancel(spthread_t thread);
00163
00164 // Can be called by a thread to get two peices of information:
00165 // 1. Whether or not the calling thread is an spthread (true or false)
00166 // 2. The spthread_t of the calling thread, if it is an spthread_t
00167 //
00168 // almost always the function will be called like this:
00169 // spthread_t self;
00170 // bool i_am_spthread = spthread_self(&self);
00171 //
00172 // args:
00173 // - spthread_t* thread: the output parameter to get the spthread_t
00174 //   representing the calling thread, if it is an spthread
00175 //
00176 // returns:
00177 // - true if the calling thread is an spthread_t
00178 // - false otherwise.
00179 bool spthread_self(spthread_t* thread);
00180
00181 // The equivalent of pthread_join but for spthread
00182 // To make sure all resources are cleaned up appropriately
00183 // spthreads that are created must at some ppoint have spthread_join
00184 // called on them. Do not use pthread_join on an spthread.
00185 //
00186 // to avoid repetition, see pthread_join(3) for details
00187 // on arguments and return values as they are the same as this function.
00188 int spthread_join(spthread_t thread, void** retval);
00189
00190 // The equivalent of pthread_exit but for spthread
00191 // spthread_exit must be used by spthreads instead of pthread_exit.
00192 // Otherwise, calls to spthread_join or other functions (like spthread_suspend)
00193 // may not work as intended.
00194 //
00195 // to avoid repetition, see pthread_exit(3) for details
00196 // on arguments and return values as they are the same as this function.
00197 void spthread_exit(void* status);
00198
00199 #endif  // SPTHREAD_H_
```

10.45 src/util/sys_call.c File Reference

```
#include "sys_call.h"
#include <unistd.h>
#include "stdio.h"
```

Functions

- char ** duplicate_argv (char *argv[])
- void free_argv (char *argv[])
- pid_t s_spawn (void *(*func)(void *), char *argv[], int fd0, int fd1)

  Create a child process that executes the function `func`. The child will retain some attributes of the parent.
- pid_t s_spawn_nice (void *(*func)(void *), char *argv[], int fd0, int fd1, unsigned int priority)
- 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 (void)

  Unconditionally exit the calling process.
- int s_nice (pid_t pid, int priority)

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

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

10.45.1 Function Documentation

10.45.1.1 duplicate_argv()

```
char ** duplicate_argv (
            char * argv[] )
```

10.45.1.2 free_argv()

```
void free_argv (
            char * argv[] )
```

10.45.1.3 s_exit()

```
void s_exit (
            void  )
```

Unconditionally exit the calling process.

10.45.1.4 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.

10.45.1.5 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.

10.45.1.6 s_sleep()

```
void 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.

10.45.1.7 s_spawn()

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

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. // need to define error output?

10.45.1.8 s_spawn_nice()

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

10.45.1.9 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.

10.46 src/util/sys_call.h File Reference

```
#include <stdbool.h>
#include <string.h>
#include "globals.h"
#include "kernel.h"
```

Macros

- #define STATUS_EXITED 0x00
- #define STATUS_STOPPED 0x01
- #define STATUS_SIGNALED 0x02
- #define P_WIFEXITED(status) (((status) & 0xFF) == STATUS_EXITED)
- #define P_WIFSTOPPED(status) (((status) & 0xFF) == STATUS_STOPPED)
- #define P_WIFSIGNALED(status) (((status) & 0xFF) == STATUS_SIGNALED)

Functions

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

  Create a child process that executes the function `func`. The child will retain some attributes of the parent.
- pid_t s_spawn_nice (void ∗(∗func)(void ∗), char ∗argv[ ], int fd0, int fd1, unsigned int priority)
- 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 (void)

  Unconditionally exit the calling process.

- int s_nice (pid_t pid, int priority)

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

  Suspends execution of the calling proces for a specified number of clock ticks.
- int s_open (const char *fname, int mode)
- ssize_t s_read (int fd, int n, char *buf)
- ssize_t s_write (int fd, const char *str, int n)
- int s_close (int fd)
- int s_unlink (const char *fname)
- off_t s_lseek (int fd, int offset, int whence)

  Construct a new s lseek object.
- int s_ls (const char *filename)

  Construct a new s ls object.

10.46.1 Macro Definition Documentation

10.46.1.1 P_WIFEXITED

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

10.46.1.2 P_WIFSIGNALED

```
#define P_WIFSIGNALED(
              status ) (((status) & 0xFF) == STATUS_SIGNALED)
```

10.46.1.3 P_WIFSTOPPED

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

10.46.1.4 STATUS_EXITED

```
#define STATUS_EXITED 0x00
```

10.46.1.5 STATUS_SIGNALED

```
#define STATUS_SIGNALED 0x02
```

10.46.1.6 STATUS_STOPPED

```
#define STATUS_STOPPED 0x01
```

10.46.2 Function Documentation

10.46.2.1 s_close()

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

Parameters

fd	

Returns

int

10.46.2.2 s_exit()

```
void s_exit (
            void  )
```

Unconditionally exit the calling process.

10.46.2.3 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.

10.46.2.4 s_ls()

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

Construct a new s ls object.

Parameters

filename	

10.46.2.5 s_lseek()

```
off_t s_lseek (
```

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

Construct a new s lseek object.

Parameters

fd	
offset	
whence	

10.46.2.6 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.

10.46.2.7 s_open()

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

Parameters

fname	
mode	

Returns

int

10.46.2.8 s_read()

```
ssize_t s_read (
            int fd,
```

```
            int n,
            char * buf )
```

Parameters

fd	
n	
buf	

Returns

ssize_t

10.46.2.9 s_sleep()

```
void 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.

10.46.2.10 s_spawn()

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

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. // need to define error output?

10.46.2.11 s_spawn_nice()

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

10.46.2.12 s_unlink()

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

Parameters

fname	

Returns

int

10.46.2.13 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.

10.46.2.14 s_write()

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

Parameters

fd	
str	
n	

Returns

ssize_t

10.47 sys_call.h

Go to the documentation of this file.

```
00001 #ifndef SYSCALL_H
00002 #define SYSCALL_H
00003
00004 #include <stdbool.h>
00005 #include <string.h>
00006 #include "globals.h"
00007 #include "kernel.h"
00008
00009 #define STATUS_EXITED 0x00
00010 #define STATUS_STOPPED 0x01
00011 #define STATUS_SIGNALED 0x02
00012
00013 #define P_WIFEXITED(status) (((status) & 0xFF) == STATUS_EXITED)
00014 #define P_WIFSTOPPED(status) (((status) & 0xFF) == STATUS_STOPPED)
00015 #define P_WIFSIGNALED(status) (((status) & 0xFF) == STATUS_SIGNALED)
00016
00017 /*== system call functions for interacting with PennOS process creation==*/
00030 pid_t s_spawn(void* (*func)(void*), char* argv[], int fd0, int fd1);
00031
00032 pid_t s_spawn_nice(void* (*func)(void*),
00033                    char* argv[],
00034                    int fd0,
00035                    int fd1,
00036                    unsigned int priority);
00037
00050 pid_t s_waitpid(pid_t pid, int* wstatus, bool nohang);
00051
00059 int s_kill(pid_t pid, int signal);
00060
00064 void s_exit(void);
00065
00066 /*============================ system calls for interacting with the scheduler
00067  * ==================================*/
00068
00076 int s_nice(pid_t pid, int priority);
00077
00090 void s_sleep(unsigned int ticks);
00091
00092 /*== system call functions for interacting with PennOS filesystem ==*/
00100 int s_open(const char* fname, int mode);
00101
00110 ssize_t s_read(int fd, int n, char* buf);
00111
00120 ssize_t s_write(int fd, const char* str, int n);
00121
00128 int s_close(int fd);
00129
00136 int s_unlink(const char* fname);
00137
00145 off_t s_lseek(int fd, int offset, int whence);
00146
00152 int s_ls(const char* filename);
00153
00154 #endif
```

10.48 test/sched-demo.c File Reference

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

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

Macros

- #define _POSIX_C_SOURCE 200809L
- #define _DEFAULT_SOURCE 1
- #define _XOPEN_SOURCE 700
- #define NUM_THREADS 4
- #define BUF_SIZE 4096

Functions

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

10.48.1 Macro Definition Documentation

10.48.1.1 _DEFAULT_SOURCE

```
#define _DEFAULT_SOURCE 1
```

10.48.1.2 _POSIX_C_SOURCE

```
#define _POSIX_C_SOURCE 200809L
```

10.48.1.3 _XOPEN_SOURCE

```
#define _XOPEN_SOURCE 700
```

10.48.1.4 BUF_SIZE

```
#define BUF_SIZE 4096
```

10.48.1.5 NUM_THREADS

```
#define NUM_THREADS 4
```

10.48.2 Function Documentation

10.48.2.1 cancel_and_join()

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

10.48.2.2 main()

```
int main (
            void  )
```

Index