My Project

Generated by Doxygen 1.9.6

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 block Struct Reference

`#include <filesystem.h>`

Public Attributes

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

3.1.1 Member Data Documentation

3.1.1.1 firstBlock

`uint16_t block::firstBlock`

3.1.1.2 mtime

`time_t block::mtime`

3.1.1.3 name

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

3.1.1.4 perm

```
uint8_t block::perm
```

3.1.1.5 size

```
uint32_t block::size
```

3.1.1.6 type

```
uint8_t block::type
```

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

- filesystem.h

3.2 fat_context Struct Reference

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

Public Attributes

- int BLOCK_SIZE
- int LSB
- int MSB
- int NUM_FAT_BLOCKS
- int NUM_DATA_BLOCKS
- int NUM_FAT
- int NUM_FAT_BYTES
- int NUM_DATA_BYTES
- int DIR_SIZE
- int DIRS_PER_BLOCK
- int fd
- char * empty_block
- char * empty_dir

3.2.1 Member Data Documentation

3.2.1.1 BLOCK_SIZE

```
int fat_context::BLOCK_SIZE
```

3.2.1.2 DIR_SIZE

```
int fat_context::DIR_SIZE
```

3.2.1.3 DIRS_PER_BLOCK

```
int fat_context::DIRS_PER_BLOCK
```

3.2.1.4 empty_block

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

3.2.1.5 empty_dir

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

3.2.1.6 fd

```
int fat_context::fd
```

3.2.1.7 LSB

```
int fat_context::LSB
```

3.2.1.8 MSB

```
int fat_context::MSB
```

3.2.1.9 NUM_DATA_BLOCKS

```
int fat_context::NUM_DATA_BLOCKS
```

3.2.1.10 NUM_DATA_BYTES

```
int fat_context::NUM_DATA_BYTES
```

3.2.1.11 NUM_FAT

```
int fat_context::NUM_FAT
```

3.2.1.12 NUM_FAT_BLOCKS

```
int fat_context::NUM_FAT_BLOCKS
```

3.2.1.13 NUM_FAT_BYTES

```
int fat_context::NUM_FAT_BYTES
```

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

- filesystem.h

3.3 fd_entry Struct Reference

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

Public Attributes

- char name [32]
- int fat_pointer
- int curr_data_ptr

3.3.1 Member Data Documentation

3.3.1.1 curr_data_ptr

```
int fd_entry::curr_data_ptr
```

3.3.1.2 fat_pointer

```
int fd_entry::fat_pointer
```

3.3.1.3 name

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

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

- filefunc.h

3.4 Group Struct Reference

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

Public Attributes

- int id
- int * ids
- char * name
- int status
- int size
- bool changed

3.4.1 Member Data Documentation

3.4.1.1 changed

```
bool Group::changed
```

3.4.1.2 id

```
int Group::id
```

3.4.1.3 ids

```
int* Group::ids
```

3.4.1.4 name

```
char* Group::name
```

3.4.1.5 size

```
int Group::size
```

3.4.1.6 status

```
int Group::status
```

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

- shell.h

3.5 List Struct Reference

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

Public Attributes

- Node * head
- Node * tail

3.5.1 Member Data Documentation

3.5.1.1 head

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

3.5.1.2 tail

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

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

- shell.h

3.6 Node Struct Reference

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

Public Attributes

- struct Node * next
- struct Node * prev
- Group * group

3.6.1 Member Data Documentation

3.6.1.1 group

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

3.6.1.2 next

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

3.6.1.3 prev

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

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

- shell.h

3.7 parsed_command Struct Reference

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

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

Public Attributes

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

3.7.1 Detailed Description

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

3.7.2 Member Data Documentation

3.7.2.1 commands

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

3.7.2.2 is_background

```
bool parsed_command::is_background
```

3.7.2.3 is_file_append

```
bool parsed_command::is_file_append
```

3.7.2.4 num_commands

```
size_t parsed_command::num_commands
```

3.7.2.5 stdin_file

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

3.7.2.6 stdout_file

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

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

- parser.h

3.8 pcb Struct Reference

PCB struct.

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

Public Attributes

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

3.8.1 Detailed Description

PCB struct.

3.8.2 Member Data Documentation

3.8.2.1 blocked_until

```
int pcb::blocked_until
```

3.8.2.2 child_pids

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

3.8.2.3 context

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

3.8.2.4 fds

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

3.8.2.5 name

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

3.8.2.6 num_children

```
int pcb::num_children
```

3.8.2.7 num_to_wait_for

```
int pcb::num_to_wait_for
```

3.8.2.8 num_zombies

```
int pcb::num_zombies
```

3.8.2.9 parent

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

3.8.2.10 pid

```
pid_t pcb::pid
```

3.8.2.11 priority

```
int pcb::priority
```

3.8.2.12 sleeping

```
bool pcb::sleeping
```

3.8.2.13 status

```
int pcb::status
```

3.8.2.14 term_status

```
int pcb::term_status
```

3.8.2.15 time_expired

```
bool pcb::time_expired
```

3.8.2.16 to_wait_for

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

3.8.2.17 updated_flag

```
bool pcb::updated_flag
```

3.8.2.18 waiting

```
bool pcb::waiting
```

3.8.2.19 waiting_for

```
pid_t pcb::waiting_for
```

3.8.2.20 woke_up_by

```
pid_t pcb::woke_up_by
```

3.8.2.21 zombies

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

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

- pcb.h

3.9 pcb_list_node Struct Reference

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

Public Attributes

- struct pcb_list_node * next
- struct pcb * pcb

3.9.1 Member Data Documentation

3.9.1.1 next

```
struct pcb_list_node* pcb_list_node::next
```

3.9.1.2 pcb

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

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

- pcb_list.h

Chapter 4

File Documentation

4.1 built_ins.c File Reference

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

Functions

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

4.1.1 Function Documentation

4.1.1.1 busy_process()

```
void busy_process ( )
```

4.1.1.2 cat_wrapper()

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

4.1.1.3 chmod_wrapper()

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

4.1.1.4 cp_wrapper()

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

4.1.1.5 echo_wrapper()

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

4.1.1.6 idle_process()

```
void idle_process ( )
```

4.1.1.7 invalid_cmd_process()

```
void invalid_cmd_process ( )
```

4.1.1.8 kill_process()

```
void kill_process ( )
```

4.1.1.9 ls_wrapper()

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

4.1.1.10 mv_wrapper()

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

4.1.1.11 orphan_child()

```
void orphan_child ( )
```

4.1.1.12 orphanify()

```
void orphanify ( )
```

4.1.1.13 parse_script()

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

4.1.1.14 ps_process()

```
void ps_process ( )
```

4.1.1.15 rm_wrapper()

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

4.1.1.16 sleep_process()

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

4.1.1.17 touch_wrapper()

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

4.1.1.18 zombie_child()

```
void zombie_child ( )
```

4.1.1.19 zombify()

```
void zombify ( )
```

4.2 built_ins.h File Reference

Functions

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

4.2.1 Function Documentation

4.2.1.1 busy_process()

```
void busy_process ( )
```

4.2.1.2 cat_wrapper()

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

4.2.1.3 chmod_wrapper()

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

4.2.1.4 cp_wrapper()

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

4.2.1.5 echo_wrapper()

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

4.2.1.6 idle_process()

```
void idle_process ( )
```

4.2.1.7 invalid_cmd_process()

```
void invalid_cmd_process ( )
```

4.2.1.8 ls_wrapper()

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

4.2.1.9 mv_wrapper()

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

4.2.1.10 orphanify()

```
void orphanify ( )
```

4.2.1.11 parse_script()

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

4.2.1.12 ps_process()

```
void ps_process ( )
```

4.2.1.13 rm_wrapper()

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

4.2.1.14 sleep_process()

```
void sleep_process ( )
```

4.2.1.15 touch_wrapper()

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

4.2.1.16 zombify()

```
void zombify ( )
```

4.3 built_ins.h

Go to the documentation of this file.

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

4.4 errno.c File Reference

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

Variables

- int errno = 0

4.4.1 Variable Documentation

4.4.1.1 errno

```
int errno = 0
```

4.5 errno.h File Reference

Macros

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

Variables

- int errno

4.5.1 Macro Definition Documentation

4.5.1.1 E2BIG

```
#define E2BIG 1
```

4.5.1.2 EEOF

```
#define EEOF 5
```

4.5.1.3 EFBIG

```
#define EFBIG 2
```

4.5.1.4 EFTF

```
#define EFTF 6
```

4.5.1.5 EINVPERM

```
#define EINVPERM 8
```

4.5.1.6 ENAMETOOLONG

```
#define ENAMETOOLONG 4
```

4.5.1.7 ENOENT

```
#define ENOENT 3
```

4.5.1.8 EPROCESSDOESNOTEXIST

```
#define EPROCESSDOESNOTEXIST 9
```

4.5.1.9 EUDWV

```
#define EUDWV 7
```

4.5.2 Variable Documentation

4.5.2.1 errno

```
int errno  [extern]
```

4.6 errno.h

Go to the documentation of this file.

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

4.7 filefunc.c File Reference

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

Macros

- #define MAXLINELENGTH 4096
- #define F_SEEK_SET 0
- #define F_SEEK_CUR 1
- #define F_SEEK_END 2

Functions

- void fileFuncConstructor (char *fatfs)

  NAME fileFuncConstructor DESCRIPTION constructor FOUND IN filefunc.c RETURNS void ERRORS.
- int find_file (char *filename)

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

  NAME find_next_available_fd DESCRIPTION finds the next available fd to use FOUND IN filefunc.c RETURNS int ERRORS -1.
- int f_open (const char *fname, int mode)

  NAME f_open DESCRIPTION open a file name fname with the mode mode and returns a file descriptor on success and a negative value on error FOUND IN filefunc.c RETURNS int ERRORS -1.
- int f_read (int fd, int n, char *buf)

  NAME f_read DESCRIPTION read n bytes from the file referenced by fd.
- int f_write (int fd, const char *str, int n)

  NAME f_write DESCRIPTION write n bytes of the string referenced by str to the file fd and increment the file pointer by n.
- int f_close (int fd)

  NAME f_close DESCRIPTION close the file fd and return 0 on success, or a negative value on failure.
- int f_unlink (const char *fname)

  NAME f_unlink DESCRIPTION remove the file and throw -1 on error FOUND IN filefunc.c RETURNS int ERRORS -1.
- int f_lseek (int fd, int offset, int whence)

  NAME f_lseek DESCRIPTION reposition the file pointer for fd to the offset relative to whence.
- int f_ls (const char *filename, int fd)

  NAME f_ls DESCRIPTION list the file filename in the current directory.
- int f_cat (char **input_files, int num_files, int fd)

  NAME f_cat DESCRIPTION Concatenates files together with input files as an input FOUND IN filefunc.c RETURNS int ERRORS -1.

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

  NAME f_touch DESCRIPTION Creates the files if they do not exist FOUND IN filefunc.c RETURNS void ERRORS.
- int f_mv (char *source, char *dest)

  NAME f_mv DESCRIPTION Renames SOURCE to DEST FOUND IN filefunc.c RETURNS int ERRORS -1.
- void f_rm (char *file)

  NAME f_rm DESCRIPTION Removes the files FOUND IN filefunc.c RETURNS void ERRORS.
- int f_cp (char *source, char *dest)

  NAME f_cp DESCRIPTION Copies SOURCE to DEST FOUND IN filefunc.c RETURNS int ERRORS -1.
- int f_chmod (char *filename, bool op, char perm)

  NAME f_chmod DESCRIPTION Change the permissions based on input FOUND IN filefunc.c RETURNS int ERRORS -1.
- int filefunc (int argc, char **argv)
- char ** f_findscript (char *fname)

  *NAME f_findscript DESCRIPTION Runs a script with commands located in a file FOUND IN filefunc.c RETURNS char** ERRORS.*

Variables

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

4.7.1 Macro Definition Documentation

4.7.1.1 F_SEEK_CUR

```
#define F_SEEK_CUR 1
```

4.7.1.2 F_SEEK_END

```
#define F_SEEK_END 2
```

4.7.1.3 F_SEEK_SET

```
#define F_SEEK_SET 0
```

4.7.1.4 MAXLINELENGTH

```
#define MAXLINELENGTH 4096
```

4.7.2 Function Documentation

4.7.2.1 f_cat()

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

NAME f_cat DESCRIPTION Concatenates files together with input files as an input FOUND IN filefunc.c RETURNS int ERRORS -1.

4.7.2.2 f_chmod()

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

NAME f_chmod DESCRIPTION Change the permissions based on input FOUND IN filefunc.c RETURNS int ERRORS -1.

4.7.2.3 f_close()

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

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

FOUND IN filefunc.c RETURNS int ERRORS -1

4.7.2.4 f_cp()

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

NAME f_cp DESCRIPTION Copies SOURCE to DEST FOUND IN filefunc.c RETURNS int ERRORS -1.

4.7.2.5 f_findscript()

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

NAME f_findscript DESCRIPTION Runs a script with commands located in a file FOUND IN filefunc.c RETURNS char** ERRORS.

4.7.2.6 f_ls()

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

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

If filename is NULL, list all files in the current directory. FOUND IN filefunc.c RETURNS int ERRORS -1

4.7.2.7 f_lseek()

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

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

FOUND IN filefunc.c RETURNS int ERRORS -1

4.7.2.8 f_mv()

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

NAME f_mv DESCRIPTION Renames SOURCE to DEST FOUND IN filefunc.c RETURNS int ERRORS -1.

4.7.2.9 f_open()

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

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

4.7.2.10 f_read()

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

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

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

4.7.2.11 f_rm()

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

NAME f_rm DESCRIPTION Removes the files FOUND IN filefunc.c RETURNS void ERRORS.

4.7.2.12 f_touch()

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

NAME f_touch DESCRIPTION Creates the files if they do not exist FOUND IN filefunc.c RETURNS void ERRORS.

4.7.2.13 f_unlink()

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

NAME f_unlink DESCRIPTION remove the file and throw -1 on error FOUND IN filefunc.c RETURNS int ERRORS -1.

4.7.2.14 f_write()

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

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

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

4.7.2.15 filefunc()

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

4.7.2.16 fileFuncConstructor()

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

NAME fileFuncConstructor DESCRIPTION constructor FOUND IN filefunc.c RETURNS void ERRORS.

4.7.2.17 find_file()

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

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

4.7.2.18 find_next_available_fd()

```
int find_next_available_fd ( )
```

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

4.7.3 Variable Documentation

4.7.3.1 f_eof

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

4.7.3.2 f_fc

```
fat_context* f_fc
```

4.7.3.3 fd_ptr

```
int fd_ptr = 2
```

4.7.3.4 files

```
struct fd_entry** files
```

4.7.3.5 NUM_FAT_ENTRIES

```
int NUM_FAT_ENTRIES
```

4.8 filefunc.h File Reference

Classes

- struct fd_entry

Typedefs

- typedef struct fd_entry fd_entry

Functions

- void fileFuncConstructor (char *fatfs)

  NAME fileFuncConstructor DESCRIPTION constructor FOUND IN filefunc.c RETURNS void ERRORS.
- int f_open (const char *fname, int mode)

  NAME f_open DESCRIPTION open a file name fname with the mode mode and returns a file descriptor on success and a negative value on error FOUND IN filefunc.c RETURNS int ERRORS -1.
- int f_read (int fd, int n, char *buf)

  NAME f_read DESCRIPTION read n bytes from the file referenced by fd.
- int f_write (int fd, const char *str, int n)

  NAME f_write DESCRIPTION write n bytes of the string referenced by str to the file fd and increment the file pointer by n.
- int f_close (int fd)

  NAME f_close DESCRIPTION close the file fd and return 0 on success, or a negative value on failure.
- int f_unlink (const char *fname)

  NAME f_unlink DESCRIPTION remove the file and throw -1 on error FOUND IN filefunc.c RETURNS int ERRORS -1.
- int f_lseek (int fd, int offset, int whence)

  NAME f_lseek DESCRIPTION reposition the file pointer for fd to the offset relative to whence.
- int f_ls (const char *filename, int fd)

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

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

  NAME f_cat DESCRIPTION Concatenates files together with input files as an input FOUND IN filefunc.c RETURNS int ERRORS -1.
- void f_touch (char **input_files, int num_files)

  NAME f_touch DESCRIPTION Creates the files if they do not exist FOUND IN filefunc.c RETURNS void ERRORS.
- int f_mv (char *source, char *dest)

  NAME f_mv DESCRIPTION Renames SOURCE to DEST FOUND IN filefunc.c RETURNS int ERRORS -1.
- void f_rm (char *file)

  NAME f_rm DESCRIPTION Removes the files FOUND IN filefunc.c RETURNS void ERRORS.
- int f_cp (char *source, char *dest)

  NAME f_cp DESCRIPTION Copies SOURCE to DEST FOUND IN filefunc.c RETURNS int ERRORS -1.
- int f_chmod (char *filename, bool op, char perm)

  NAME f_chmod DESCRIPTION Change the permissions based on input FOUND IN filefunc.c RETURNS int ERRORS -1.
- char ** f_findscript (char *fname)

  *NAME f_findscript DESCRIPTION Runs a script with commands located in a file FOUND IN filefunc.c RETURNS char** ERRORS.*

4.8.1 Typedef Documentation

4.8.1.1 fd_entry

```
typedef struct fd_entry fd_entry
```

4.8.2 Function Documentation

4.8.2.1 f_cat()

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

NAME f_cat DESCRIPTION Concatenates files together with input files as an input FOUND IN filefunc.c RETURNS int ERRORS -1.

4.8.2.2 f_chmod()

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

NAME f_chmod DESCRIPTION Change the permissions based on input FOUND IN filefunc.c RETURNS int ERRORS -1.

4.8.2.3 f_close()

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

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

FOUND IN filefunc.c RETURNS int ERRORS -1

4.8.2.4 f_cp()

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

NAME f_cp DESCRIPTION Copies SOURCE to DEST FOUND IN filefunc.c RETURNS int ERRORS -1.

4.8.2.5 f_findscript()

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

NAME f_findscript DESCRIPTION Runs a script with commands located in a file FOUND IN filefunc.c RETURNS char∗∗ ERRORS.

4.8.2.6 f_ls()

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

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

If filename is NULL, list all files in the current directory. FOUND IN filefunc.c RETURNS int ERRORS -1

4.8.2.7 f_lseek()

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

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

FOUND IN filefunc.c RETURNS int ERRORS -1

4.8.2.8 f_mv()

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

NAME f_mv DESCRIPTION Renames SOURCE to DEST FOUND IN filefunc.c RETURNS int ERRORS -1.

4.8.2.9 f_open()

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

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

4.8.2.10 f_read()

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

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

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

4.8.2.11 f_rm()

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

NAME f_rm DESCRIPTION Removes the files FOUND IN filefunc.c RETURNS void ERRORS.

4.8.2.12 f_touch()

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

NAME f_touch DESCRIPTION Creates the files if they do not exist FOUND IN filefunc.c RETURNS void ERRORS.

4.8.2.13 f_unlink()

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

NAME f_unlink DESCRIPTION remove the file and throw -1 on error FOUND IN filefunc.c RETURNS int ERRORS -1.

4.8.2.14 f_write()

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

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

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

4.8.2.15 fileFuncConstructor()

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

NAME fileFuncConstructor DESCRIPTION constructor FOUND IN filefunc.c RETURNS void ERRORS.

4.9 filefunc.h

Go to the documentation of this file.

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

4.10 filesystem.c File Reference

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

Functions

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

Variables

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

4.10.1 Function Documentation

4.10.1.1 clear_data_region()

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

4.10.1.2 clear_file()

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

4.10.1.3 constructor()

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

4.10.1.4 create()

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

4.10.1.5 dir_to_struct()

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

4.10.1.6 fatread()

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

4.10.1.7 fatreadblock()

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

4.10.1.8 fatremove()

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

4.10.1.9 fatwrite()

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

4.10.1.10 find_available_block()

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

4.10.1.11 find_dir_entry()

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

4.10.1.12 find_last_block()

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

4.10.1.13 initialize()

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

4.10.1.14 read_dir_entry()

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

4.10.1.15 read_fat_block()

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

4.10.1.16 seek_data_region()

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

4.10.1.17 seek_dir_data_region()

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

4.10.1.18 seek_to_write_data()

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

4.10.1.19 update_dir_entry()

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

4.10.1.20 write_block_fat()

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

4.10.1.21 write_next_dir_entry()

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

4.10.2 Variable Documentation

4.10.2.1 root

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

4.10.2.2 zeros

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

4.11 filesystem.h File Reference

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

Classes

- struct block
- struct fat_context

Typedefs

- typedef struct block block
- typedef struct fat_context fat_context

Functions

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

4.11.1 Typedef Documentation

4.11.1.1 block

```
typedef struct block block
```

4.11.1.2 fat_context

```
typedef struct fat_context fat_context
```

4.11.2 Function Documentation

4.11.2.1 clear_file()

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

4.11.2.2 constructor()

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

4.11.2.3 create()

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

4.11.2.4 dir_to_struct()

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

4.11.2.5 fatread()

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

4.11.2.6 fatreadblock()

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

4.11.2.7 fatremove()

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

4.11.2.8 fatwrite()

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

4.11.2.9 find_available_block()

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

4.11.2.10 find_dir_entry()

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

4.11.2.11 find_last_block()

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

4.11.2.12 initialize()

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

4.11.2.13 read_dir_entry()

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

4.11.2.14 read_fat_block()

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

4.11.2.15 seek_data_region()

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

4.11.2.16 seek_dir_data_region()

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

4.11.2.17 seek_to_write_data()

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

4.11.2.18 update_dir_entry()

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

4.11.2.19 write_block_fat()

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

4.11.2.20 write_next_dir_entry()

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

4.12 filesystem.h

Go to the documentation of this file.

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

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

4.13 kernel.c File Reference

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

Functions

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

Variables

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

4.13.1 Function Documentation

4.13.1.1 main()

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

4.13.2 Variable Documentation

4.13.2.1 active_pcb

```
struct pcb* active_pcb
```

4.13.2.2 head

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

4.13.2.3 idle_context

```
ucontext_t idle_context
```

4.13.2.4 log_file

```
FILE* log_file = 0
```

4.13.2.5 main_context

```
ucontext_t main_context
```

4.13.2.6 max_pid

```
int max_pid
```

4.13.2.7 scheduler_context

```
ucontext_t scheduler_context
```

4.13.2.8 terminal_control

```
pid_t terminal_control = 0
```

4.13.2.9 ticks

```
int ticks = 0
```

4.14 kernel.h File Reference

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

Variables

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

4.14.1 Variable Documentation

4.14.1.1 head

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

4.14.1.2 queues

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

4.14.1.3 terminal_control

```
pid_t terminal_control  [extern]
```

4.14.1.4 ticks

```
int ticks  [extern]
```

4.15 kernel.h

Go to the documentation of this file.

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

4.16 kernel_func.c File Reference

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

Functions

- struct pcb * k_process_create (struct pcb *parent)

  NAME k_process_create DESCRIPTION Takes in parent pcb and creates and returns child pcb.
- void k_process_cleanup_1 (struct pcb *process, char *command)

  NAME k_process_cleanup DESCRIPTION called when a terminated/finished thread's resources needs to be cleaned up.
- void k_process_kill (struct pcb *process, int signal)

  NAME k_process_kill DESCRIPTION kill the process referenced by process with the signal signal.
- void wake_up_parent (pcb *process)

4.16.1 Function Documentation

4.16.1.1 k_process_cleanup_1()

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

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

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

4.16.1.2 k_process_create()

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

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

Updates parent child pids as well. FOUND IN kernel_func.c RETURNS child process pcb. ERRORS throws no errors.

4.16.1.3 k_process_kill()

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

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

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

4.16.1.4 wake_up_parent()

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

4.17 kernel_func.h File Reference

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

Functions

- struct pcb * k_process_create (struct pcb *parent)

  NAME k_process_create DESCRIPTION Takes in parent pcb and creates and returns child pcb.
- void k_process_kill (struct pcb *process, int signal)

  NAME k_process_kill DESCRIPTION kill the process referenced by process with the signal signal.
- void k_process_cleanup (struct pcb *process)
- void k_process_cleanup_1 (struct pcb *process, char *command)

  NAME k_process_cleanup DESCRIPTION called when a terminated/finished thread's resources needs to be cleaned up.
- void wake_up_parent (pcb *process)

Variables

- int max_pid
- ucontext_t scheduler_context

4.17.1 Function Documentation

4.17.1.1 k_process_cleanup()

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

4.17.1.2 k_process_cleanup_1()

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

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

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

4.17.1.3 k_process_create()

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

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

Updates parent child pids as well. FOUND IN kernel_func.c RETURNS child process pcb. ERRORS throws no errors.

4.17.1.4 k_process_kill()

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

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

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

4.17.1.5 wake_up_parent()

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

4.17.2 Variable Documentation

4.17.2.1 max_pid

```
int max_pid  [extern]
```

4.17.2.2 scheduler_context

```
ucontext_t scheduler_context  [extern]
```

4.18 kernel_func.h

Go to the documentation of this file.

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

4.19 log.c File Reference

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

Functions

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

4.19.1 Function Documentation

4.19.1.1 log_command()

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

4.20 log.h File Reference

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

Macros

- #define SCHEDULE "SCHEDULE"
- #define CREATE "CREATE"
- #define SIGNALED "SIGNALED"
- #define EXITED "EXITED"
- #define ZOMBIE "ZOMBIE"
- #define NICE "NICE"
- #define ORPHAN "ORPHAN"
- #define WAITED "WAITED"
- #define BLOCKED_LOG "BLOCKED"
- #define UNBLOCKED "UNBLOCKED"
- #define STOPPED_LOG "STOPPED"
- #define CONTINUED "CONTINUED"

Functions

- void log_command (char ∗command, pcb ∗process, int prev_nice)

Variables

- FILE ∗ log_file

4.20.1 Macro Definition Documentation

4.20.1.1 BLOCKED_LOG

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

4.20.1.2 CONTINUED

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

4.20.1.3 CREATE

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

4.20.1.4 EXITED

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

4.20.1.5 NICE

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

4.20.1.6 ORPHAN

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

4.20.1.7 SCHEDULE

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

4.20.1.8 SIGNALED

```
#define SIGNALED "SIGNALED"
```

4.20.1.9 STOPPED_LOG

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

4.20.1.10 UNBLOCKED

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

4.20.1.11 WAITED

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

4.20.1.12 ZOMBIE

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

4.20.2 Function Documentation

4.20.2.1 log_command()

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

4.20.3 Variable Documentation

4.20.3.1 log_file

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

4.21 log.h

Go to the documentation of this file.

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

4.22 parser.h File Reference

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

Classes

- struct parsed_command

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

Macros

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

Functions

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

  Arguments: cmd_line: a null-terminated string that is the command line result: a non-null pointer to a `struct parsed_command *`
- void print_parsed_command (const struct parsed_command *cmd)

4.22.1 Macro Definition Documentation

4.22.1.1 EXPECT_COMMANDS

```
#define EXPECT_COMMANDS 7
```

4.22.1.2 EXPECT_INPUT_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

4.22.1.3 EXPECT_OUTPUT_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

4.22.1.4 UNEXPECTED_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

4.22.1.5 UNEXPECTED_FILE_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

4.22.1.6 UNEXPECTED_FILE_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

4.22.1.7 UNEXPECTED_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

4.22.2 Function Documentation

4.22.2.1 parse_command()

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

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

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

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

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

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

4.22.2.2 print_parsed_command()

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

4.23 parser.h

Go to the documentation of this file.

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

```
25 // parser didn't find output filename following '>' or '»'
26 #define EXPECT_OUTPUT_FILENAME 6
27
28 // parser didn't find any commands or arguments where it expects one
29 #define EXPECT_COMMANDS 7
30
35 struct parsed_command {
36     // if the command shall be executed in background
37     // (ends with an ampersand '&')
38     bool is_background;
39
40     // if the stdout_file shall be opened in append mode
41     // ignore this value when stdout_file is NULL
42     bool is_file_append;
43
44     // filename for redirecting input from
45     const char *stdin_file;
46
47     // filename for redirecting output to
48     const char *stdout_file;
49
50     // number of commands (pipeline stages)
51     size_t num_commands;
52
53     // an array to a list of arguments
54     // size of `commands` is `num_commands`
55     char **commands[];
56 };
57
83 int parse_command(const char *cmd_line, struct parsed_command **result);
84
85
86 /* This is a debugging function used for outputting a parsed command line. */
87 void print_parsed_command(const struct parsed_command *cmd);
```

4.24 pcb.h File Reference

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

Classes

- struct pcb

  PCB struct.

Macros

- #define TERM_NORMAL 10
- #define TERM_SIGNALED 20
- #define RUNNING 3
- #define STOPPED 4
- #define BLOCKED 5
- #define ZOMBIED 6

Typedefs

- typedef struct pcb pcb

  PCB struct.

4.24.1 Macro Definition Documentation

4.24.1.1 BLOCKED

```
#define BLOCKED 5
```

4.24.1.2 RUNNING

```
#define RUNNING 3
```

4.24.1.3 STOPPED

```
#define STOPPED 4
```

4.24.1.4 TERM_NORMAL

```
#define TERM_NORMAL 10
```

4.24.1.5 TERM_SIGNALED

```
#define TERM_SIGNALED 20
```

4.24.1.6 ZOMBIED

```
#define ZOMBIED 6
```

4.24.2 Typedef Documentation

4.24.2.1 pcb

```
typedef struct pcb pcb
```

PCB struct.

4.25 pcb.h

Go to the documentation of this file.

```
1 #ifndef PCB_H
2 #define PCB_H
3
4 #include <stdbool.h>
5 #include <sys/types.h>
6 #include <ucontext.h>
7
8 #define TERM_NORMAL 10
9 #define TERM_SIGNALED 20
10 #define RUNNING 3
11 #define STOPPED 4
12 #define BLOCKED 5
13 #define ZOMBIED 6
14
18 typedef struct pcb
19 {
20     ucontext_t *context;
21     struct pcb *parent;
22     pid_t pid;
23     int num_children;
24     pid_t *child_pids;
25     int num_zombies;
26     pid_t *zombies;
27     int fds[2]; // fd set to -1 as default
28     int priority;
29     int status;
30     int term_status;
31     bool waiting;
32     int num_to_wait_for;
33     pid_t *to_wait_for;
34     pid_t woke_up_by;
35     pid_t waiting_for;
36     bool updated_flag; // flag set to 1 if pcb status was updated
37     bool time_expired;
38     char *name;
39     int blocked_until;
40     bool sleeping;
41     // add more later
42 } pcb;
43
44 #endif
```

4.26 pcb_list.c File Reference

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

Functions

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

4.26.1 Function Documentation

4.26.1.1 add_pcb()

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

4.26.1.2 get_pcb_from_pid()

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

4.26.1.3 remove_pcb_from_pid()

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

4.26.1.4 soft_remove()

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

4.27 pcb_list.h File Reference

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

Classes

- struct pcb_list_node

Typedefs

- typedef struct pcb_list_node pcb_list_node

Functions

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

4.27.1 Typedef Documentation

4.27.1.1 pcb_list_node

```
typedef struct pcb_list_node pcb_list_node
```

4.27.2 Function Documentation

4.27.2.1 add_pcb()

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

4.27.2.2 get_pcb_from_pid()

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

4.27.2.3 remove_pcb_from_pid()

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

4.27.2.4 soft_remove()

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

4.28 pcb_list.h

Go to the documentation of this file.

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

4.29 pennfat.c File Reference

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

Macros

- #define MAXLINELENGTH 4096

Functions

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

Variables

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

4.29.1 Macro Definition Documentation

4.29.1.1 MAXLINELENGTH

```
#define MAXLINELENGTH 4096
```

4.29.2 Function Documentation

4.29.2.1 cat()

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

4.29.2.2 cat_from_terminal()

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

4.29.2.3 chmod()

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

4.29.2.4 count_args()

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

4.29.2.5 cp()

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

4.29.2.6 fake_initialize()

```
void fake_initialize ( )
```

4.29.2.7 ls()

```
void ls ( )
```

4.29.2.8 main()

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

4.29.2.9 mkfs()

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

4.29.2.10 mount()

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

4.29.2.11 mv()

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

4.29.2.12 rm()

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

4.29.2.13 touch()

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

4.29.2.14 unmount()

```
void unmount ( )
```

4.29.3 Variable Documentation

4.29.3.1 eof

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

4.29.3.2 fat

```
uint16_t* fat
```

4.29.3.3 fc

```
fat_context* fc
```

4.30 sched.c File Reference

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

Functions

- void handleFinish ()
- void unblock ()
- void schedule (void)

Variables

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

4.30.1 Function Documentation

4.30.1.1 handleFinish()

```
void handleFinish ( )
```

4.30.1.2 schedule()

```
void schedule (
            void  )
```

4.30.1.3 unblock()

```
void unblock ( )
```

4.30.2 Variable Documentation

4.30.2.1 queues

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

4.31 sched.h File Reference

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

Functions

- void schedule (void)

Variables

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

4.31.1 Function Documentation

4.31.1.1 schedule()

```
void schedule (
            void  )
```

4.31.2 Variable Documentation

4.31.2.1 active_pcb

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

4.31.2.2 idle_context

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

4.31.2.3 main_context

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

4.32 sched.h

Go to the documentation of this file.

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

4.33 shell.c File Reference

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

Functions

- void parent_sigtstp_handler (int signo)
- void parent_sigint_handler (int signo)
- void runOnForeground (Group *group)

  This function takes a group, runs it in the foreground, waits for it to complete and then returns terminal control to the parent.
- void shell (int argc, char *arg[ ])

Variables

- int lastJob = -1
- bool stoppedJobs = false
- int begin = 1

4.33.1 Function Documentation

4.33.1.1 parent_sigint_handler()

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

4.33.1.2 parent_sigtstp_handler()

```
void parent_sigtstp_handler (
            int signo )
```

4.33.1.3 runOnForeground()

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

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

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

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

Parameters

group	the group we want to move to the foreground

Returns

void.

4.33.1.4 shell()

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

4.33.2 Variable Documentation

4.33.2.1 begin

```
int begin = 1
```

4.33.2.2 lastJob

```
int lastJob = -1
```

4.33.2.3 stoppedJobs

```
bool stoppedJobs = false
```

4.34 shell.h File Reference

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

Classes

- struct Group
- struct Node
- struct List

Macros

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

Typedefs

- typedef struct Group Group
- typedef struct Node Node
- typedef struct List List

Functions

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

  Creates a new process for the given command, sets the file descriptors and modifies the group struct as necessary.
- void Init (List ∗s)
- void RemoveById (int id, List ∗s)
- void Remove (Node ∗node, List ∗s)
- void Add (List ∗s, Group ∗g)
- bool isEmpty (List ∗s)
- Node ∗ Peek (List ∗s)
- void printAll (List ∗list)
- void clear (List ∗list)
- void report (List ∗list)

  Print updates about all the jobs before prompting the user.
- void updateStatuses (List ∗list)

  This will update the status of the jobs.
- void printJobs (List ∗list)

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

Variables

- int childId
- int childStatus
- int lastJob

4.34.1 Macro Definition Documentation

4.34.1.1 COMMAND_LENGTH

```
#define COMMAND_LENGTH 4096
```

4.34.1.2 RETURN_ASCII_CODE

```
#define RETURN_ASCII_CODE 10
```

4.34.1.3 SHELL_BACKGROUND_TO_FOREGROUND

```
#define SHELL_BACKGROUND_TO_FOREGROUND 104
```

4.34.1.4 SHELL_FINISHED

```
#define SHELL_FINISHED 105
```

4.34.1.5 SHELL_RESTARTING

```
#define SHELL_RESTARTING 103
```

4.34.1.6 SHELL_RUNNING

```
#define SHELL_RUNNING 102
```

4.34.1.7 SHELL_STOPPED

```
#define SHELL_STOPPED 101
```

4.34.2 Typedef Documentation

4.34.2.1 Group

```
typedef struct Group Group
```

4.34.2.2 List

```
typedef struct List List
```

4.34.2.3 Node

```
typedef struct Node Node
```

4.34.3 Function Documentation

4.34.3.1 Add()

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

4.34.3.2 clear()

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

4.34.3.3 execute()

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

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

Parameters

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

4.34.3.4 Init()

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

4.34.3.5 isEmpty()

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

4.34.3.6 Peek()

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

4.34.3.7 printAll()

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

4.34.3.8 printJobs()

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

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

It should be called when the user enters the command jobs

Parameters

list	the linked list with all the jobs

Returns

void.

4.34.3.9 Remove()

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

4.34.3.10 RemoveById()

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

4.34.3.11 report()

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

Print updates about all the jobs before prompting the user.

Will only print the jobs whose status has changed

Parameters

list	the linked list with all the jobs

Returns

void.

4.34.3.12 shell()

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

4.34.3.13 updateStatuses()

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

This will update the status of the jobs.

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

Parameters

list	the linked list with all the jobs

Returns

void.

4.34.4 Variable Documentation

4.34.4.1 childId

```
int childId  [extern]
```

4.34.4.2 childStatus

```
int childStatus  [extern]
```

4.34.4.3 lastJob

```
int lastJob  [extern]
```

4.35 shell.h

Go to the documentation of this file.

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

4.36 shell_execute.c File Reference

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

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

Functions

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

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

4.36.1 Function Documentation

4.36.1.1 create_child()

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

4.36.1.2 execute()

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

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

Parameters

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

4.37 shell_list.c File Reference

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

Functions

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

4.37.1 Function Documentation

4.37.1.1 Add()

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

4.37.1.2 clear()

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

4.37.1.3 Init()

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

4.37.1.4 isEmpty()

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

4.37.1.5 Peek()

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

4.37.1.6 printAll()

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

4.37.1.7 RemoveById()

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

4.38 shell_notif.c File Reference

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

Functions

- void report (List *list)

  Print updates about all the jobs before prompting the user.
- void updateStatuses (List *list)

  This will update the status of the jobs.
- void printJobs (List *list)

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

4.38.1 Function Documentation

4.38.1.1 printJobs()

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

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

It should be called when the user enters the command jobs

Parameters

list	the linked list with all the jobs

Returns

void.

4.38.1.2 report()

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

Print updates about all the jobs before prompting the user.

Will only print the jobs whose status has changed

Parameters

list	the linked list with all the jobs

Returns

void.

4.38.1.3 updateStatuses()

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

This will update the status of the jobs.

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

Parameters

list	the linked list with all the jobs

Returns

void.

4.39 signals.c File Reference

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

Functions

- void sigalrm_handler (int signo)
- void sigstop_handler (int signo)
- void sigint_handler (int signo)

4.39.1 Function Documentation

4.39.1.1 sigalrm_handler()

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

4.39.1.2 sigint_handler()

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

4.39.1.3 sigstop_handler()

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

4.40 signals.h File Reference

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

Macros

- #define S_SIGSTOP 1
- #define S_SIGCONT 2
- #define S_SIGTERM 3

Functions

- void sigalrm_handler (int signo)
- void sigstop_handler (int signo)
- void sigint_handler (int signo)

Variables

- struct pcb * active_pcb
- ucontext_t scheduler_context

4.40.1 Macro Definition Documentation

4.40.1.1 S_SIGCONT

```
#define S_SIGCONT 2
```

4.40.1.2 S_SIGSTOP

```
#define S_SIGSTOP 1
```

4.40.1.3 S_SIGTERM

```
#define S_SIGTERM 3
```

4.40.2 Function Documentation

4.40.2.1 sigalrm_handler()

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

4.40.2.2 sigint_handler()

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

4.40.2.3 sigstop_handler()

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

4.40.3 Variable Documentation

4.40.3.1 active_pcb

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

4.40.3.2 scheduler_context

```
ucontext_t scheduler_context  [extern]
```

4.41 signals.h

Go to the documentation of this file.

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

4.42 stress.c File Reference

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

Functions

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

4.42.1 Function Documentation

4.42.1.1 hang()

```
void hang (
            void  )
```

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

- –

4.42.1.2 nohang()

```
void nohang (
            void  )
```

4.42.1.3 recur()

```
void recur (
            void  )
```

4.43 stress.h File Reference

Functions

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

4.43.1 Function Documentation

4.43.1.1 hang()

```
void hang (
            void  )
```

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

- –

4.43.1.2 nohang()

```
void nohang (
            void  )
```

4.43.1.3 recur()

```
void recur (
            void  )
```

4.44 stress.h

Go to the documentation of this file.

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

4.45 user_func.c File Reference

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

Functions

- void p_perror (char *toPrint)
- bool check_pid_is_child (struct pcb *parent, int pid)
- pid_t p_spawn (void(*func)(), char *argv[], int fd0, int fd1, char *name)

  NAME p_spawn DESCRIPTION forks a new thread that retains most of the attributes of the parent thread (see k_process_create).
- pid_t p_waitpid_1 (pid_t pid, int *wstatus, bool nohang)
- pid_t p_waitpid (pid_t pid, int *wstatus, bool nohang)

  NAME p_waitpid DESCRIPTION sets the calling thread as blocked (if nohang is false) until a child of the calling thread changes state.
- bool W_WIFEXITED (int status)
- bool W_WIFSTOPPED (int status)
- bool W_WIFSIGNALED (int status)
- int p_kill (pid_t pid, int sig)

  NAME p_kill DESCRIPTION kills a process with specified pid and signal.
- void p_exit ()

  NAME p_exit DESCRIPTION exits the current thread unconditionally.
- void p_nice (pid_t pid, int priority)

  NAME p_nice DESCRIPTION sets the priority of the thread pid to priority.
- void p_sleep (unsigned int len)

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

4.45.1 Function Documentation

4.45.1.1 check_pid_is_child()

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

4.45.1.2 p_exit()

```
void p_exit ( )
```

NAME p_exit DESCRIPTION exits the current thread unconditionally.

FOUND IN user_func.c RETURNS successfully exits the terminal. ERRORS active process does not exist.

4.45.1.3 p_kill()

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

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

FOUND IN user_func.c RETURNS 0 upon success and -1 upon error ERRORS pid is invalid

4.45.1.4 p_nice()

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

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

FOUND IN user_func.c RETURNS returns upon successful completion. ERRORS fail to retrieve the process pcb.

4.45.1.5 p_perror()

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

4.45.1.6 p_sleep()

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

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

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

4.45.1.7 p_spawn()

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

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

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

4.45.1.8 p_waitpid()

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

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

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

4.45.1.9 p_waitpid_1()

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

4.45.1.10 W_WIFEXITED()

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

4.45.1.11 W_WIFSIGNALED()

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

4.45.1.12 W_WIFSTOPPED()

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

4.46 user_func.h File Reference

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

Functions

- void p_perror (char *toPrint)
- bool check_pid_is_child (struct pcb *parent, int pid)
- pid_t p_spawn (void(*func)(), char *argv[], int fd0, int fd1, char *name)

  NAME p_spawn DESCRIPTION forks a new thread that retains most of the attributes of the parent thread (see k_process_create).
- pid_t p_waitpid (pid_t pid, int *wstatus, bool nohang)

  NAME p_waitpid DESCRIPTION sets the calling thread as blocked (if nohang is false) until a child of the calling thread changes state.
- int p_kill (pid_t pid, int sig)

  NAME p_kill DESCRIPTION kills a process with specified pid and signal.
- void p_nice (pid_t pid, int priority)

  NAME p_nice DESCRIPTION sets the priority of the thread pid to priority.
- void p_exit ()

  NAME p_exit DESCRIPTION exits the current thread unconditionally.
- void p_sleep (unsigned int len)

  NAME p_sleep DESCRIPTION sets the calling process to blocked until ticks of the system clock elapse, and then sets the thread to running.
- bool W_WIFEXITED (int status)
- bool W_WIFSTOPPED (int status)
- bool W_WIFSIGNALED (int status)

Variables

- struct pcb_list_node * head
- struct pcb * active_pcb
- struct pcb_list_node * queues [3]

4.46.1 Function Documentation

4.46.1.1 check_pid_is_child()

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

4.46.1.2 p_exit()

```
void p_exit ( )
```

NAME p_exit DESCRIPTION exits the current thread unconditionally.

FOUND IN user_func.c RETURNS successfully exits the terminal. ERRORS active process does not exist.

4.46.1.3 p_kill()

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

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

FOUND IN user_func.c RETURNS 0 upon success and -1 upon error ERRORS pid is invalid

4.46.1.4 p_nice()

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

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

FOUND IN user_func.c RETURNS returns upon successful completion. ERRORS fail to retrieve the process pcb.

4.46.1.5 p_perror()

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

4.46.1.6 p_sleep()

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

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

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

4.46.1.7 p_spawn()

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

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

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

4.46.1.8 p_waitpid()

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

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

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

4.46.1.9 W_WIFEXITED()

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

4.46.1.10 W_WIFSIGNALED()

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

4.46.1.11 W_WIFSTOPPED()

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

4.46.2 Variable Documentation

4.46.2.1 active_pcb

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

4.46.2.2 head

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

4.46.2.3 queues

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

4.47 user_func.h

Go to the documentation of this file.

```
1 #ifndef USER_FUNC_H
2 #define USER_FUNC_H
3
4 #include "pcb.h"
5 #include "pcb_list.h"
6 #include <signal.h>
7 #include <stdbool.h>
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <sys/types.h>
11 #include <ucontext.h>
12 #include <unistd.h>
13 #include <valgrind/valgrind.h>
14
15 extern struct pcb_list_node *head;
16 extern struct pcb *active_pcb;
17 extern struct pcb_list_node *queues[3];
18
19 void p_perror(char* toPrint);
20
21 bool check_pid_is_child(struct pcb *parent, int pid);
22
23 pid_t p_spawn(void (*func)(), char *argv[], int fd0, int fd1, char *name);
24
25 pid_t p_waitpid(pid_t pid, int *wstatus, bool nohang);
26
34 int p_kill(pid_t pid, int sig);
35
43 void p_nice(pid_t pid, int priority);
44
52 void p_exit();
53
61 void p_sleep(unsigned int len);
62
63 bool W_WIFEXITED(int status);
64
65 bool W_WIFSTOPPED(int status);
66
67 bool W_WIFSIGNALED(int status);
68
69 #endif
```

4.48 utils.c File Reference

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

Functions

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

4.48.1 Function Documentation

4.48.1.1 char_to_16()

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

4.48.1.2 char_to_32()

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

4.48.1.3 char_to_8()

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

4.48.1.4 is_zero()

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

4.49 utils.h File Reference

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

Macros

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

Functions

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

4.49.1 Macro Definition Documentation

4.49.1.1 MAX

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

4.49.1.2 MIN

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

4.49.2 Function Documentation

4.49.2.1 char_to_16()

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

4.49.2.2 char_to_32()

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

4.49.2.3 char_to_8()

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

4.49.2.4 get_msb()

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

4.49.2.5 is_zero()

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

4.50 utils.h

Go to the documentation of this file.

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

Index