Companion PDF

Team Members:

Yu Cao (yucao7@seas.upenn.edu)

Chun-Ngai Chan (jamesccn@seas.upenn.edu)

Wanjin Li (wanjinli@seas.upenn.edu)

Larry Zhang (chizhg@seas.upenn.edu)

PennOS

Generated by Doxygen 1.9.1

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:

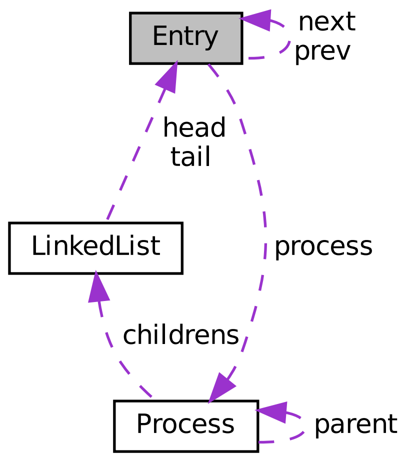
Chapter 3

Class Documentation

3.1 Entry Struct Reference

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

Collaboration diagram for Entry:



Public Attributes

- struct Process ∗ process
- struct Entry ∗ prev
- struct Entry ∗ next

3.1.1 Detailed Description

The Node struct used for our Linkedlist.

3.1.2 Member Data Documentation

3.1.2.1 next

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

the pointer to the next node

3.1.2.2 prev

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

the pointer to the previous node

3.1.2.3 process

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

the Process stored inside this node

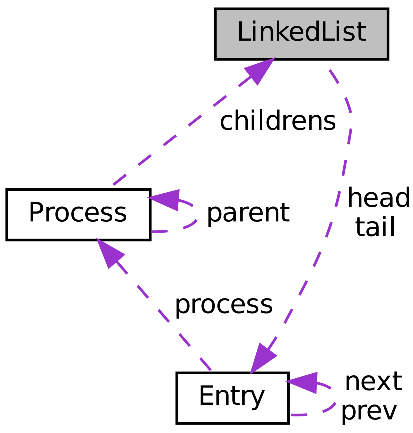
The documentation for this struct was generated from the following file:

- linkedlist-job.h

3.2 LinkedList Struct Reference

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

Collaboration diagram for LinkedList:



Public Attributes

- struct Entry ∗ head
- struct Entry ∗ tail

3.2.1 Detailed Description

The Linkedlist struct.

3.2.2 Member Data Documentation

3.2.2.1 head

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

the pointer to the head node of this linkedlist

3.2.2.2 tail

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

the pointer to the tail node of this linkedlist

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

- linkedlist-job.h

3.3 parsed_command Struct Reference

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

Public Attributes

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

3.3.1 Detailed Description

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

3.3.2 Member Data Documentation

3.3.2.1 commands

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

3.3.2.2 is_background

```
bool parsed_command::is_background
```

3.3.2.3 is_file_append

```
bool parsed_command::is_file_append
```

3.3.2.4 num_commands

```
size_t parsed_command::num_commands
```

3.3.2.5 stdin_file

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

3.3.2.6 stdout_file

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

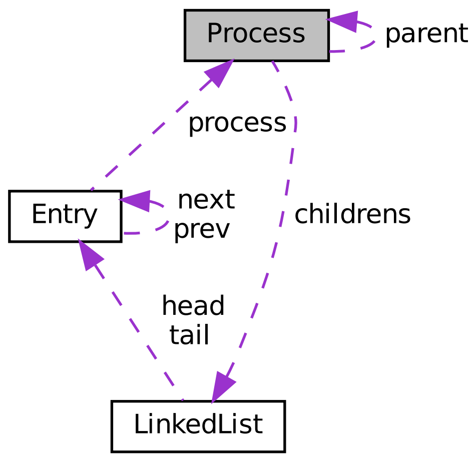
The documentation for this struct was generated from the following file:

- parser.h

3.4 Process Struct Reference

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

Collaboration diagram for Process:



Public Attributes

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

3.4.1 Detailed Description

The PCB struct used for our OS.

3.4.2 Member Data Documentation

3.4.2.1 argv

`char** Process::argv`

The modified arguments we are taking into specific functions

3.4.2.2 awake_time

`int Process::awake_time`

the time this process should awake (only for sleep)

3.4.2.3 bg_time

`int Process::bg_time`

record when was this Process stored in background

3.4.2.4 childrens

`struct LinkedList* Process::childrens`

the Linkedlist of the PCB's children

3.4.2.5 cmd

`char* Process::cmd`

the command for this Process (for logging)

3.4.2.6 context

`ucontext_t* Process::context`

the ucontext of the PCB

3.4.2.7 fg_cont

```
bool Process::fg_cont
```

record if it should be continued at foreground

3.4.2.8 group_id

```
int Process::group_id
```

Process group id (unused)

3.4.2.9 input_descriptor

```
int Process::input_descriptor
```

the input descriptor for this Process

3.4.2.10 is_orphan

```
bool Process::is_orphan
```

record whether it's an orphan

3.4.2.11 num_children

```
int Process::num_children
```

the number of children for this Process

3.4.2.12 output_descriptor

```
int Process::output_descriptor
```

the output descriptor for this Process

3.4.2.13 parent

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

PCB pointer to its parent

3.4.2.14 parent_process_id

```
int Process::parent_process_id
```

the pid of this PCB's parent

3.4.2.15 priority

```
int Process::priority
```

the priority (nice value) of this PCB

3.4.2.16 recorded

```
int Process::recorded
```

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

3.4.2.17 signal_terminated

```
bool Process::signal_terminated
```

record if this process is terminated by signal

3.4.2.18 status

```
int Process::status
```

the status for this PCB

3.4.2.19 stop_time

```
int Process::stop_time
```

record when was this Process stopped

3.4.2.20 thread_process_id

```
int Process::thread_process_id
```

the pid of the PCB

3.4.2.21 to_wait

```
pid_t Process::to_wait
```

the pid this Process is waiting on

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

- linkedlist-job.h

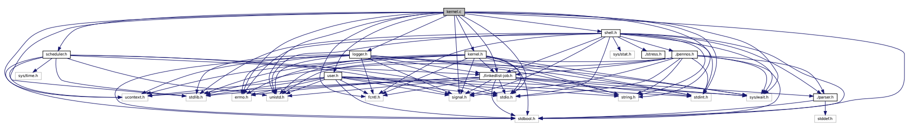
Chapter 4

File Documentation

4.1 kernel.c File Reference

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

Include dependency graph for kernel.c:



Macros

- #define S_SIGSTOP 0
- #define S_SIGCONT 1
- #define S_SIGTERM 2

Functions

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

4.1.1 Macro Definition Documentation

4.1.1.1 S_SIGCONT

```
#define S_SIGCONT 1
```

4.1.1.2 S_SIGSTOP

```
#define S_SIGSTOP 0
```

4.1.1.3 S_SIGTERM

```
#define S_SIGTERM 2
```

4.1.2 Function Documentation

4.1.2.1 k_get_next_pid()

```
int k_get_next_pid ( )
```

Get the current available pid and increment it by 1.

Returns

The next pid we can assign to a new process.

4.1.2.2 k_get_running_status()

```
int k_get_running_status ( )
```

For abstraction sake, get the RUNNING wstatus from outside.

Returns

The RUNNING defined.

4.1.2.3 k_get_sigcont_signal()

```
int k_get_sigcont_signal ( )
```

For abstraction sake, get the S_SIGCONT signal from outside.

Returns

The S_SIGCONT signal defined.

4.1.2.4 k_get_sigcont_str()

```
char* k_get_sigcont_str ( )
```

The helper function we used to get S_SIGCONT_STR for shell use.

Returns

The S_SIGCONT_STR we defined.

4.1.2.5 k_get_sigstop_signal()

```
int k_get_sigstop_signal ( )
```

For abstraction sake, get the S_SIGSTOP signal from outside.

Returns

The S_SIGSTOP signal defined.

4.1.2.6 k_get_sigstop_str()

```
char* k_get_sigstop_str ( )
```

The helper function we used to get S_SIGSTOP_STR for shell use.

Returns

The S_SIGSTOP_STR we defined.

4.1.2.7 k_get_sigterm_signal()

```
int k_get_sigterm_signal ( )
```

For abstraction sake, get the S_SIGTERM signal from outside.

Returns

The S_SIGTERM signal defined.

4.1.2.8 k_get_sigterm_str()

```
char* k_get_sigterm_str ( )
```

The helper function we used to get S_SIGTERM_STR for shell use.

Returns

The S_SIGTERM_STR we defined.

4.1.2.9 k_get_stop_signal_status()

```
int k_get_stop_signal_status ( )
```

For abstraction sake, get the STOP_SIGNAL wstatus from outside.

Returns

The STOP_SIGNAL defined.

4.1.2.10 k_get_terminal_normal_status()

```
int k_get_terminal_normal_status ( )
```

For abstraction sake, get the TERMINAL_NORMAL wstatus from outside.

Returns

The TERMINAL_NORMAL defined.

4.1.2.11 k_get_terminal_signal_status()

```
int k_get_terminal_signal_status ( )
```

For abstraction sake, get the TERMINAL_SIGNAL wstatus from outside.

Returns

The TERMINAL_SIGNAL defined.

4.1.2.12 k_get_to_exit()

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

Get the process stored in to_exit.

Returns

The process stored inside to_exit.

4.1.2.13 k_initiate_to_exit()

```
void k_initiate_to_exit ( )
```

Initiate the to_exit stored for p_exit_process().

4.1.2.14 k_kill_all()

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

Kill all processes with a given signal.

Parameters

signal	The signal we are sending to all processes.

4.1.2.15 k_lookup_process()

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

Look up a process from our job queue.

Parameters

thread←_id	The pid we are looking for.

Returns

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

4.1.2.16 k_process_cleanup()

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

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

Parameters

process	The process to clean up.

4.1.2.17 k_process_create()

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

Create a process with a given process as its parent.

Parameters

parent	The pointer to the parent process.

Returns

The pointer to the Process we created.

4.1.2.18 k_process_create_with_priority()

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

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

Parameters

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

Returns

The process created.

4.1.2.19 k_process_kill()

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

Kill a Process with the given signal.

Parameters

process	The process we want to kill with the signal.
signal	The signal we are sending to the process.

Returns

The status of whether the killing is successful.

4.1.2.20 k_reap_zombie()

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

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

Parameters

pid	The pid of the process to reap.

4.1.2.21 k_set_idle()

```
bool k_set_idle ( )
```

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

4.1.2.22 k_set_to_exit()

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

Set to_exit as a process for p_exit_process().

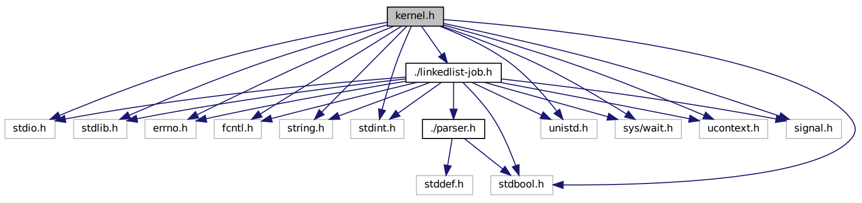
Parameters

process	The process to set as to_exit.

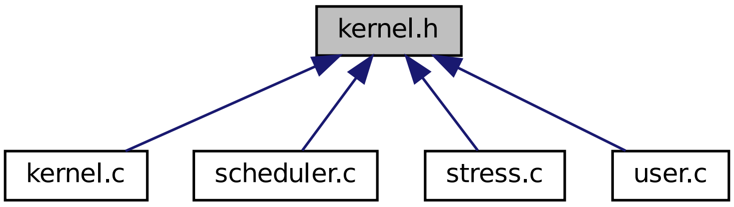
4.2 kernel.h File Reference

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

Include dependency graph for kernel.h:



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



Macros

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

Functions

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

4.2.1 Macro Definition Documentation

4.2.1.1 RUNNING

```
#define RUNNING 3
```

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

4.2.1.2 S_SIGCONT

```
#define S_SIGCONT 1
```

The signal sent to continue a previously stopped process.

4.2.1.3 S_SIGCONT_STR

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

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

4.2.1.4 S_SIGSTOP

```
#define S_SIGSTOP 0
```

The signal sent to stop a process.

4.2.1.5 S_SIGSTOP_STR

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

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

4.2.1.6 S_SIGTERM

```
#define S_SIGTERM 2
```

The signal sent to terminate a process.

4.2.1.7 S_SIGTERM_STR

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

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

4.2.1.8 STOP_SIGNAL

```
#define STOP_SIGNAL 1
```

The wstatus standing for process stopped by a signal.

4.2.1.9 TERMINATE_NORMAL

```
#define TERMINATE_NORMAL 0
```

The wstatus standing for process terminated normally.

4.2.1.10 TERMINATE_SIGNAL

```
#define TERMINATE_SIGNAL 2
```

The wstatus standing for process terminated by a signal.

4.2.2 Function Documentation

4.2.2.1 k_get_next_pid()

```
int k_get_next_pid ( )
```

Get the current available pid and increment it by 1.

Returns

The next pid we can assign to a new process.

4.2.2.2 k_get_running_status()

```
int k_get_running_status ( )
```

For abstraction sake, get the RUNNING wstatus from outside.

Returns

The RUNNING defined.

4.2.2.3 k_get_sigcont_signal()

```
int k_get_sigcont_signal ( )
```

For abstraction sake, get the S_SIGCONT signal from outside.

Returns

The S_SIGCONT signal defined.

4.2.2.4 k_get_sigcont_str()

```
char* k_get_sigcont_str ( )
```

The helper function we used to get S_SIGCONT_STR for shell use.

Returns

The S_SIGCONT_STR we defined.

4.2.2.5 k_get_sigstop_signal()

```
int k_get_sigstop_signal ( )
```

For abstraction sake, get the S_SIGSTOP signal from outside.

Returns

The S_SIGSTOP signal defined.

4.2.2.6 k_get_sigstop_str()

```
char* k_get_sigstop_str ( )
```

The helper function we used to get S_SIGSTOP_STR for shell use.

Returns

The S_SIGSTOP_STR we defined.

4.2.2.7 k_get_sigterm_signal()

```
int k_get_sigterm_signal ( )
```

For abstraction sake, get the S_SIGTERM signal from outside.

Returns

The S_SIGTERM signal defined.

4.2.2.8 k_get_sigterm_str()

```
char* k_get_sigterm_str ( )
```

The helper function we used to get S_SIGTERM_STR for shell use.

Returns

The S_SIGTERM_STR we defined.

4.2.2.9 k_get_stop_signal_status()

```
int k_get_stop_signal_status ( )
```

For abstraction sake, get the STOP_SIGNAL wstatus from outside.

Returns

The STOP_SIGNAL defined.

4.2.2.10 k_get_terminal_normal_status()

```
int k_get_terminal_normal_status ( )
```

For abstraction sake, get the TERMINAL_NORMAL wstatus from outside.

Returns

The TERMINAL_NORMAL defined.

4.2.2.11 k_get_terminal_signal_status()

```
int k_get_terminal_signal_status ( )
```

For abstraction sake, get the TERMINAL_SIGNAL wstatus from outside.

Returns

The TERMINAL_SIGNAL defined.

4.2.2.12 k_get_to_exit()

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

Get the process stored in to_exit.

Returns

The process stored inside to_exit.

4.2.2.13 k_initiate_to_exit()

```
void k_initiate_to_exit ( )
```

Initiate the to_exit stored for p_exit_process().

4.2.2.14 k_kill_all()

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

Kill all processes with a given signal.

Parameters

signal	The signal we are sending to all processes.

4.2.2.15 k_lookup_process()

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

Look up a process from our job queue.

Parameters

thread↩_id	The pid we are looking for.

Returns

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

4.2.2.16 k_process_cleanup()

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

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

Parameters

process	The process to clean up.

4.2.2.17 k_process_create()

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

Create a process with a given process as its parent.

Parameters

parent	The pointer to the parent process.

Returns

The pointer to the Process we created.

4.2.2.18 k_process_create_with_priority()

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

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

Parameters

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

Returns

The process created.

4.2.2.19 k_process_kill()

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

Kill a Process with the given signal.

Parameters

process	The process we want to kill with the signal.
signal	The signal we are sending to the process.

Returns

The status of whether the killing is successful.

4.2.2.20 k_reap_zombie()

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

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

Parameters

pid	The pid of the process to reap.

4.2.2.21 k_set_idle()

```
bool k_set_idle ( )
```

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

4.2.2.22 k_set_to_exit()

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

Set to_exit as a process for p_exit_process().

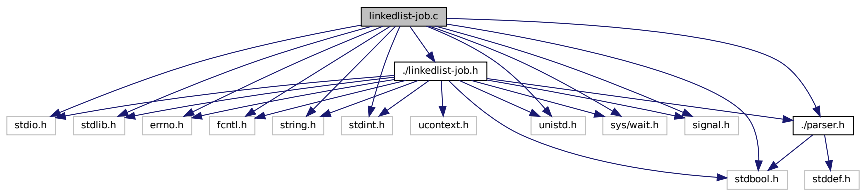
Parameters

process	The process to set as to_exit.

4.3 linkedlist-job.c File Reference

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

Include dependency graph for linkedlist-job.c:



Functions

- void insert_end (struct LinkedList ∗list, struct Process ∗process)
- struct Process ∗ delete_node (struct LinkedList ∗list, int id)
- void free_process (struct Process ∗process)
- struct Entry ∗ search_list (struct LinkedList ∗list, int id)
- struct Process ∗ retrieve_latest (struct LinkedList ∗list)
- struct Process ∗ poll (struct LinkedList ∗list)
- void free_list (struct LinkedList ∗list)
- void set_orphan (struct Entry ∗e, bool orphan)

4.3.1 Function Documentation

4.3.1.1 delete_node()

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

Delete a node containing the process we want from a LinkedList.

Parameters

list	The pointer to the list we want to delete the node from.
id	The pid of the Process we are deleting.

Returns

The node entry containing the process we are looking for.

4.3.1.2 free_list()

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

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

Parameters

list	The pointer to the list we want to free.

4.3.1.3 free_process()

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

Free a process completely from our memory.

Parameters

process	The pointer to process we want to insert.

4.3.1.4 insert_end()

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

Insert a process into the end of the linkedlist.

Parameters

list	The pointer to the linkedlist to insert into.
process	The pointer to process we want to insert.

4.3.1.5 poll()

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

Retrieve the process contained in the first node from a LinkedList.

Parameters

list	The pointer to the list we want to retrieve the node from.

Returns

The process contained in the first node from this LinkedList.

4.3.1.6 retrieve_latest()

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

Retrieve the process contained in the last node from a LinkedList.

Parameters

list	The pointer to the list we want to retrieve the node from.

Returns

The process contained in the last node from this LinkedList.

4.3.1.7 search_list()

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

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

Parameters

list	The pointer to the list we want to search in.
id	The pid of the Process we are searching for.

Returns

The node entry containing the process we are looking for.

4.3.1.8 set_orphan()

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

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

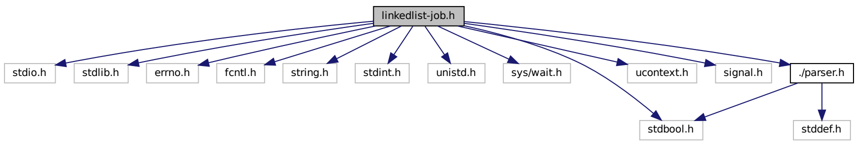
Parameters

e	The node containing the process we are setting to some orphan status.
orphan	The orphan status we are setting.

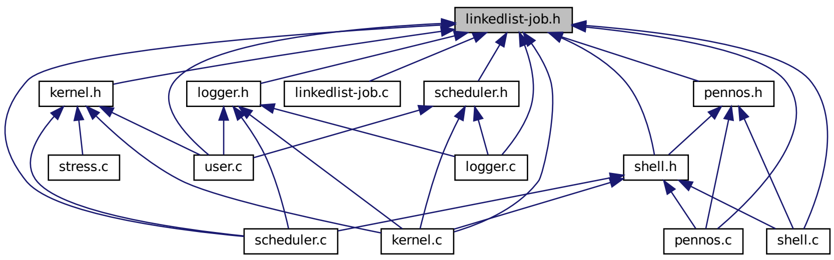
4.4 linkedlist-job.h File Reference

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

Include dependency graph for linkedlist-job.h:



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



Classes

- struct Process
- struct Entry
- struct LinkedList

Macros

- #define ACTIVE_STAT 0
- #define PAUSED_STAT 1
- #define DONE_STAT 2
- #define STOP_STAT 3
- #define PROCESS
- #define ENTRY
- #define LIST

Functions

- void insert_end (struct LinkedList ∗list, struct Process ∗process)
- void free_process (struct Process ∗process)
- struct Entry ∗ search_list (struct LinkedList ∗list, int id)
- struct Process ∗ delete_node (struct LinkedList ∗list, int id)
- void print_list (struct LinkedList ∗list)
- struct Process ∗ retrieve_latest (struct LinkedList ∗list)
- struct Process ∗ poll (struct LinkedList ∗list)
- void free_list (struct LinkedList ∗list)
- void set_orphan (struct Entry ∗e, bool orphan)

4.4.1 Macro Definition Documentation

4.4.1.1 ACTIVE_STAT

```
#define ACTIVE_STAT 0
```

the process is actively running and can be scheduled.

4.4.1.2 DONE_STAT

```
#define DONE_STAT 2
```

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

4.4.1.3 ENTRY

```
#define ENTRY
```

4.4.1.4 LIST

```
#define LIST
```

4.4.1.5 PAUSED_STAT

```
#define PAUSED_STAT 1
```

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

4.4.1.6 PROCESS

```
#define PROCESS
```

4.4.1.7 STOP_STAT

```
#define STOP_STAT 3
```

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

4.4.2 Function Documentation

4.4.2.1 delete_node()

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

Delete a node containing the process we want from a LinkedList.

Parameters

list	The pointer to the list we want to delete the node from.
id	The pid of the Process we are deleting.

Returns

The node entry containing the process we are looking for.

4.4.2.2 free_list()

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

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

Parameters

list	The pointer to the list we want to free.

4.4.2.3 free_process()

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

Free a process completely from our memory.

Parameters

process	The pointer to process we want to insert.

4.4.2.4 insert_end()

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

Insert a process into the end of the linkedlist.

Parameters

list	The pointer to the linkedlist to insert into.
process	The pointer to process we want to insert.

4.4.2.5 poll()

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

Retrieve the process contained in the first node from a LinkedList.

Parameters

list	The pointer to the list we want to retrieve the node from.

Returns

The process contained in the first node from this LinkedList.

4.4.2.6 print_list()

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

Print out a LinkedList.

Parameters

list	The pointer to the list we want to print.

4.4.2.7 retrieve_latest()

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

Retrieve the process contained in the last node from a LinkedList.

Parameters

list	The pointer to the list we want to retrieve the node from.

Returns

The process contained in the last node from this LinkedList.

4.4.2.8 search_list()

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

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

Parameters

list	The pointer to the list we want to search in.
id	The pid of the Process we are searching for.

Returns

The node entry containing the process we are looking for.

4.4.2.9 set_orphan()

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

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

Parameters

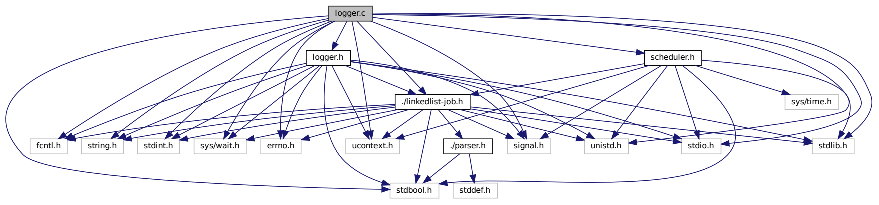
e	The node containing the process we are setting to some orphan status.
orphan	The orphan status we are setting.

4.5 logger.c File Reference

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

`#include "scheduler.h"`

Include dependency graph for logger.c:



Functions

- FILE ∗ open_log_file ()
- void log_event (const char ∗event_type, struct Process ∗process)
- void log_nice_event (int old_nice, struct Process ∗process)

4.5.1 Function Documentation

4.5.1.1 log_event()

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

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

Parameters

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

4.5.1.2 log_nice_event()

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

Special helper function used to log nice-related processes.

Parameters

old_nice	The old nice value.
process	The process we need to print log for.

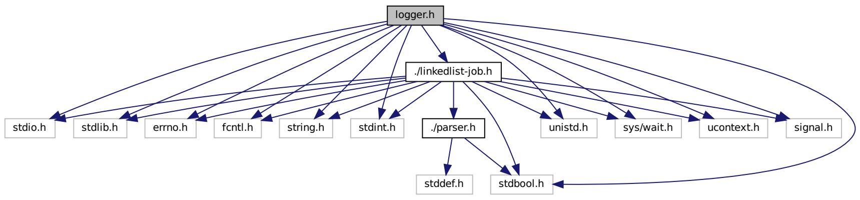
4.5.1.3 open_log_file()

```
FILE* open_log_file ( )
```

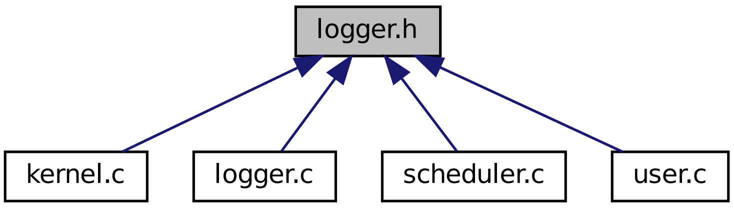
4.6 logger.h File Reference

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

Include dependency graph for logger.h:



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



Functions

- void log_event (const char ∗event_type, struct Process ∗process)
- void log_nice_event (int old_nice, struct Process ∗process)

4.6.1 Function Documentation

4.6.1.1 log_event()

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

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

Parameters

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

4.6.1.2 log_nice_event()

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

Special helper function used to log nice-related processes.

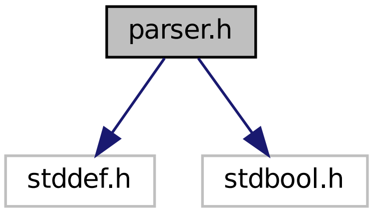
Parameters

old_nice	The old nice value.
process	The process we need to print log for.

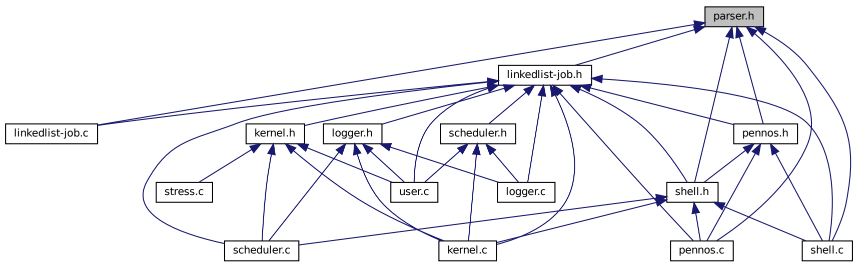
4.7 parser.h File Reference

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

Include dependency graph for parser.h:



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



Classes

- struct parsed_command

Macros

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

Functions

- int parse_command (const char ∗cmd_line, struct parsed_command ∗∗result)
- void print_parsed_command (const struct parsed_command ∗cmd)

4.7.1 Macro Definition Documentation

4.7.1.1 EXPECT_COMMANDS

```
#define EXPECT_COMMANDS 7
```

4.7.1.2 EXPECT_INPUT_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

4.7.1.3 EXPECT_OUTPUT_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

4.7.1.4 UNEXPECTED_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

4.7.1.5 UNEXPECTED_FILE_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

4.7.1.6 UNEXPECTED_FILE_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

4.7.1.7 UNEXPECTED_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

4.7.2 Function Documentation

4.7.2.1 parse_command()

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

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

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

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

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

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

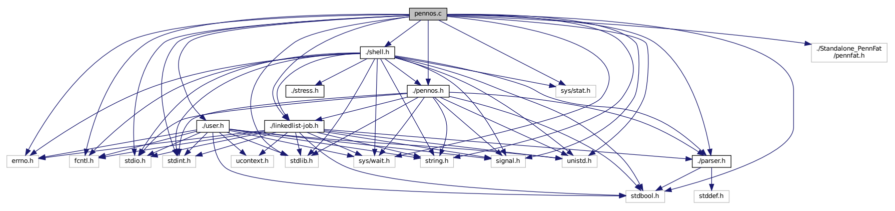
4.7.2.2 print_parsed_command()

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

4.8 pennos.c File Reference

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

```
#include <sys/stat.h>
```
Include dependency graph for pennos.c:



Functions

- int main (int argc, char ∗argv[ ])
- void handler (int signal)

4.8.1 Function Documentation

4.8.1.1 handler()

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

Signal Handler for SIGINT and SIGTSTP.

Parameters

signal	The signal received.

4.8.1.2 main()

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

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

Parameters

argc	The number of arguments passed in from the terminal.
argv	The arguments from terminal.

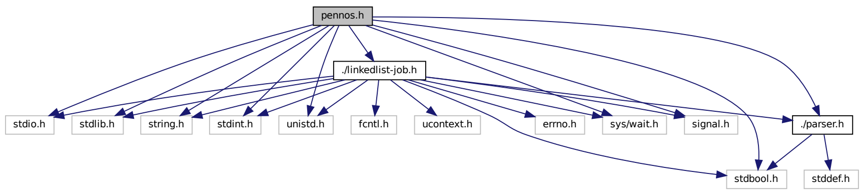
Returns

Anything on exit.

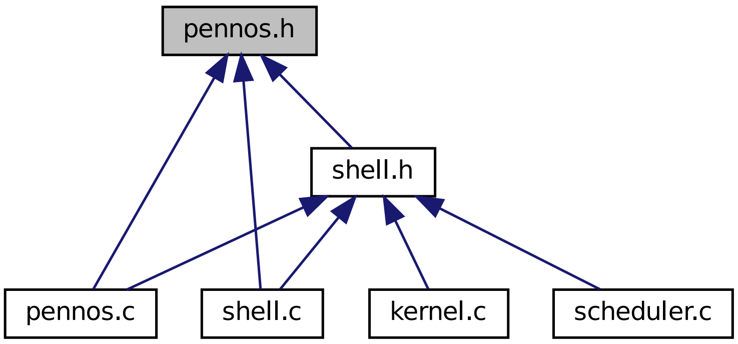
4.9 pennos.h File Reference

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

Include dependency graph for pennos.h:



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



Functions

- int main (int argc, char ∗argv[ ])
- void handler (int signal)

4.9.1 Function Documentation

4.9.1.1 handler()

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

Signal Handler for SIGINT and SIGTSTP.

Parameters

signal	The signal received.

4.9.1.2 main()

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

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

Parameters

argc	The number of arguments passed in from the terminal.
argv	The arguments from terminal.

Returns

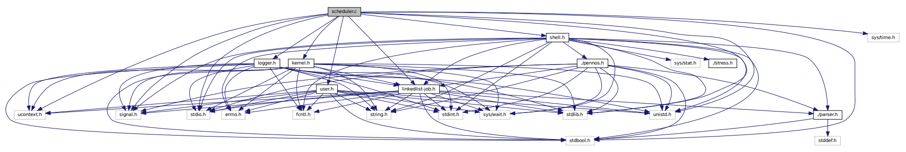
Anything on exit.

4.10 scheduler.c File Reference

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

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

Include dependency graph for scheduler.c:



Macros

- #define THREAD_COUNT 4
- #define NOT_WAITING -2

Functions

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

4.10.1 Macro Definition Documentation

4.10.1.1 NOT_WAITING

```
#define NOT_WAITING -2
```

4.10.1.2 THREAD_COUNT

```
#define THREAD_COUNT 4
```

4.10.2 Function Documentation

4.10.2.1 check_actual()

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

4.10.2.2 freeStacks()

```
void freeStacks (
             void  )
```

The helper function we used to free the current stack.

4.10.2.3 s_check_active()

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

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

Parameters

queue	The pointer to the priority queue we are looking at.

Returns

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

4.10.2.4 s_get_current()

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

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

Returns

The pointer to the currentPCB.

4.10.2.5 s_get_priority()

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

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

Parameters

priority	The priority (nice value) we are trying to get.

Returns

The pointer to the priority job LinkedList we want.

4.10.2.6 s_get_scheduler_context()

```
ucontext_t* s_get_scheduler_context ( )
```

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

Returns

The scheduler context.

4.10.2.7 s_get_time()

```
int s_get_time ( )
```

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

Returns

The current time.

4.10.2.8 s_get_zombie_context()

```
ucontext_t* s_get_zombie_context ( )
```

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

Returns

The zombie context.

4.10.2.9 s_initiate()

```
void s_initiate ( )
```

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

4.10.2.10 s_initiate_priorities()

```
void s_initiate_priorities ( )
```

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

4.10.2.11 s_initiate_shell_context()

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

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

Parameters

argc	The number of arguments passed in.
argv	The arguments passed in.
process	The process PCB for the shell.

4.10.2.12 s_insert()

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

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

Parameters

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

4.10.2.13 s_makeContext()

```
void s_makeContext (
             ucontext_t * ucp,
             void(*)() func,
             int thread )
```

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

Parameters

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

4.10.2.14 s_print_all_jobs()

```
void s_print_all_jobs ( )
```

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

4.10.2.15 s_set()

```
void s_set ( )
```

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

4.10.2.16 s_set_current()

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

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

Parameters

process	The process to set as currentPCB.

4.10.2.17 s_set_idle()

```
void s_set_idle ( )
```

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

4.10.2.18 s_set_status()

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

The helper function used to set the status of the current running Process.

Parameters

status	The status we need to set for the current PCB.

4.10.2.19 s_setup()

```
void s_setup ( )
```

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

4.10.2.20 s_swap()

```
void s_swap ( )
```

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

4.10.2.21 setStack()

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

The helper function we used to set the current stack.

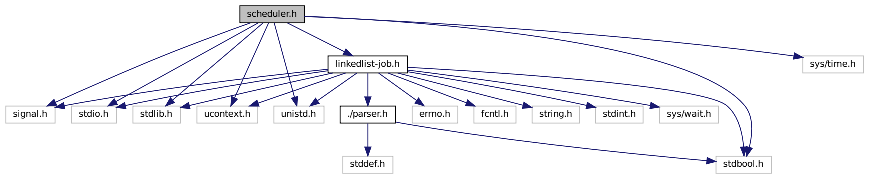
Parameters

The	pointer to the stack we are setting.

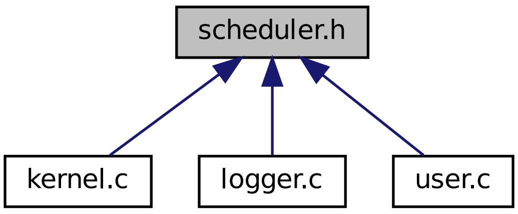
4.11 scheduler.h File Reference

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

Include dependency graph for scheduler.h:



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



Macros

- #define NOT_WAITING -2

Functions

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

4.11.1 Macro Definition Documentation

4.11.1.1 NOT_WAITING

```
#define NOT_WAITING -2
```

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

4.11.2 Function Documentation

4.11.2.1 freeStacks()

```
void freeStacks (
             void  )
```

The helper function we used to free the current stack.

4.11.2.2 s_check_active()

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

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

Parameters

queue	The pointer to the priority queue we are looking at.

Returns

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

4.11.2.3 s_get_current()

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

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

Returns

The pointer to the currentPCB.

4.11.2.4 s_get_priority()

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

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

Parameters

priority	The priority (nice value) we are trying to get.

Returns

The pointer to the priority job LinkedList we want.

4.11.2.5 s_get_scheduler_context()

```
ucontext_t* s_get_scheduler_context ( )
```

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

Returns

The scheduler context.

4.11.2.6 s_get_time()

```
int s_get_time ( )
```

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

Returns

The current time.

4.11.2.7 s_get_zombie_context()

```
ucontext_t* s_get_zombie_context ( )
```

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

Returns

The zombie context.

4.11.2.8 s_initiate()

```
void s_initiate ( )
```

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

4.11.2.9 s_initiate_priorities()

```
void s_initiate_priorities ( )
```

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

4.11.2.10 s_initiate_shell_context()

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

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

Parameters

argc	The number of arguments passed in.
argv	The arguments passed in.
process	The process PCB for the shell.

4.11.2.11 s_insert()

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

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

Parameters

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

4.11.2.12 s_makeContext()

```
void s_makeContext (
            ucontext_t * ucp,
            void(*)() func,
            int thread )
```

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

Parameters

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

4.11.2.13 s_print_all_jobs()

```
void s_print_all_jobs ( )
```

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

4.11.2.14 s_set()

```
void s_set ( )
```

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

4.11.2.15 s_set_current()

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

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

Parameters

process	The process to set as currentPCB.

4.11.2.16 s_set_idle()

```
void s_set_idle ( )
```

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

4.11.2.17 s_set_status()

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

The helper function used to set the status of the current running Process.

Parameters

status	The status we need to set for the current PCB.

4.11.2.18 s_setup()

```
void s_setup ( )
```

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

4.11.2.19 s_swap()

```
void s_swap ( )
```

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

4.11.2.20 setStack()

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

The helper function we used to set the current stack.

Parameters

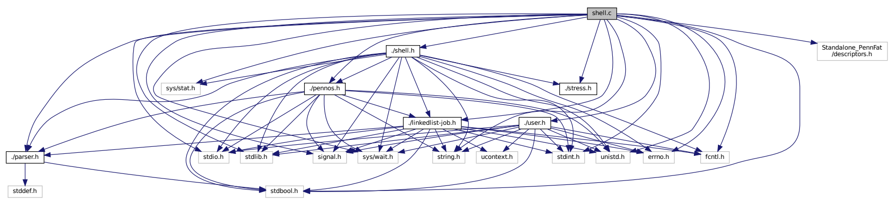
The	pointer to the stack we are setting.

4.12 shell.c File Reference

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

```
#include "./user.h"
#include "stress.h"
#include "Standalone_PennFat/descriptors.h"
```

Include dependency graph for shell.c:



Functions

- int get_fg_pid ()
- void set_fg_pid (int pid)
- void sigint_handler ()
- void sigstp_handler ()
- void shell ()

4.12.1 Function Documentation

4.12.1.1 get_fg_pid()

```
int get_fg_pid ( )
```

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

Returns

The current foreground job pid.

4.12.1.2 set_fg_pid()

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

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

Parameters

pid	The pid we set as the foreground job pid.

4.12.1.3 shell()

```
void shell ( )
```

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

4.12.1.4 sigint_handler()

```
void sigint_handler ( )
```

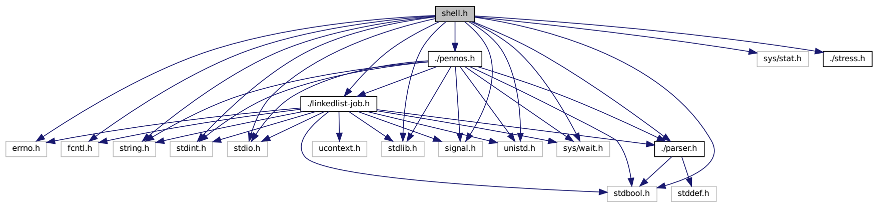
4.12.1.5 sigstp_handler()

```
void sigstp_handler ( )
```

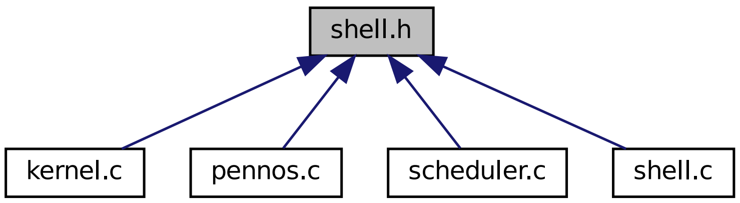
4.13 shell.h File Reference

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

Include dependency graph for shell.h:



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



Functions

- int get_fg_pid ()
- void shell ()
- void set_fg_pid (int pid)

4.13.1 Function Documentation

4.13.1.1 get_fg_pid()

```
int get_fg_pid ( )
```

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

Returns

The current foreground job pid.

4.13.1.2 set_fg_pid()

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

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

Parameters

pid	The pid we set as the foreground job pid.

4.13.1.3 shell()

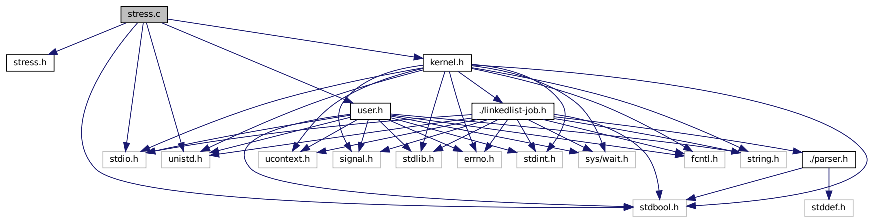
```
void shell ( )
```

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

4.14 stress.c File Reference

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

Include dependency graph for stress.c:



Functions

- void hang (void)
- void nohang (void)
- void recur (void)

4.14.1 Function Documentation

4.14.1.1 hang()

```
void hang (
             void  )
```

4.14.1.2 nohang()

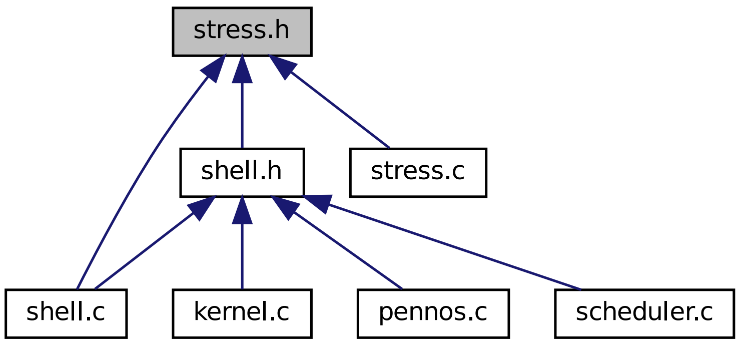
```
void nohang (
             void )
```

4.14.1.3 recur()

```
void recur (
             void )
```

4.15 stress.h File Reference

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



Functions

- void hang (void)
- void nohang (void)
- void recur (void)

4.15.1 Function Documentation

4.15.1.1 hang()

```
void hang (
             void )
```

4.15.1.2 nohang()

```
void nohang (
             void  )
```

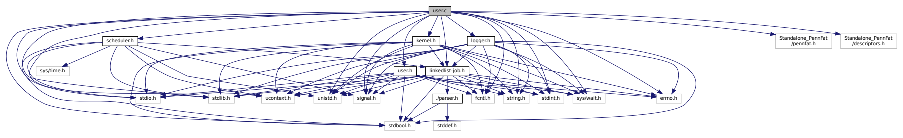
4.15.1.3 recur()

```
void recur (
             void  )
```

4.16 user.c File Reference

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

Include dependency graph for user.c:



Functions

- int getSignal (char ∗sig)
- pid_t p_spawn (void(∗func)(), char ∗argv[ ], int fd0, int fd1)
- pid_t p_spawn_with_priority (void(∗func)(), char ∗argv[ ], int fd0, int fd1, int priority)
- pid_t p_spawn_with_input (void(∗func)(), char ∗argv[ ], int fd0, int fd1, char ∗∗actual_input)
- pid_t p_initiate_shell (void(∗func)(), int argc, char ∗argv[ ])
- pid_t p_waitpid (pid_t pid, int ∗wstatus, bool nohang)
- int p_kill (pid_t pid, int sig)
- void p_exit (void)

- void p_exit_process ()
- bool W_WIFEXITED (int ∗status)
- bool W_WIFSTOPPED (int ∗status)
- bool W_WIFSIGNALED (int ∗status)
- int p_nice (pid_t pid, int priority)
- void p_sleep (unsigned int ticks)
- void p_busy_wait ()
- void p_zombie_child ()
- void p_orphan_child ()
- void p_print_all_jobs ()
- void p_add_background_job (int pid)
- void p_add_stop_job (int pid)
- void p_remove_background_job (int pid)
- void p_remove_stop_job (int pid)
- void p_search_and_remove (int pid)
- int p_search_most_recent ()
- int p_search_most_recent_stop ()
- struct Process ∗ p_search_bg (int pid)
- int p_get_sigstop_signal ()
- int p_get_sigcont_signal ()
- int p_get_sigterm_signal ()
- void p_initiate_to_exit ()
- struct Process ∗ p_get_current ()
- void p_initiate_priorities ()
- void p_setup ()
- void p_initiate ()
- struct Process ∗ p_lookup_process (pid_t pid)
- void p_zombify ()
- void p_orphanify ()
- void p_run_kill (int sig, pid_t pid)
- void p_run_mv_fs (char ∗source, char ∗dest)
- void p_run_chmod_fs (char ∗filename, char ∗perm)
- void p_run_cp_fs (char ∗source, char ∗des, int from_host)
- void p_run_touch_fs ()
- void p_run_f_ls_list ()
- void p_run_f_ls_null ()
- void p_run_rm_fs ()
- void p_run_echo ()
- void p_run_cat_fs ()
- char ∗ p_get_sigstop_str ()
- char ∗ p_get_sigcont_str ()
- char ∗ p_get_sigterm_str ()

4.16.1 Function Documentation

4.16.1.1 getSignal()

```
int getSignal (
            char ∗ sig )
```

4.16.1.2 p_add_background_job()

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

The helper function to add a thread into the background.

Parameters

pid	The pid for the thread to add into the background.

4.16.1.3 p_add_stop_job()

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

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

Parameters

pid	The pid for the thread to add into the stopped queue.

4.16.1.4 p_busy_wait()

```
void p_busy_wait ( )
```

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

4.16.1.5 p_exit()

```
void p_exit (
             void  )
```

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

4.16.1.6 p_exit_process()

```
void p_exit_process ( )
```

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

4.16.1.7 p_get_current()

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

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

Returns

The PCB for the currently running thread.

4.16.1.8 p_get_sigcont_signal()

```
int p_get_sigcont_signal ( )
```

A helper function to get the S_SIGCONT we defined for abstraction.

Returns

The S_SIGCONT we defined.

4.16.1.9 p_get_sigcont_str()

```
char* p_get_sigcont_str ( )
```

The helper function we used to redirect into kernel and get S_SIGCONT_STR we defined inside of kernel for shell use.

Returns

The S_SIGSTOP_STR we defined inside of kernel.

4.16.1.10 p_get_sigstop_signal()

```
int p_get_sigstop_signal ( )
```

A helper function to get the S_SIGSTOP we defined for abstraction.

Returns

The S_SIGSTOP we defined.

4.16.1.11 p_get_sigstop_str()

```
char* p_get_sigstop_str ( )
```

The helper function we used to redirect into kernel and get S_SIGSTOP_STR we defined inside of kernel for shell use.

Returns

The S_SIGSTOP_STR we defined inside of kernel.

4.16.1.12 p_get_sigterm_signal()

```
int p_get_sigterm_signal ( )
```

A helper function to get the S_SIGTERM we defined for abstraction.

Returns

The S_SIGTERM we defined.

4.16.1.13 p_get_sigterm_str()

```
char* p_get_sigterm_str ( )
```

The helper function we used to redirect into kernel and get S_SIGTERM_STR we defined inside of kernel for shell use.

Returns

The S_SIGSTOP_STR we defined inside of kernel.

4.16.1.14 p_initiate()

```
void p_initiate ( )
```

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

4.16.1.15 p_initiate_priorities()

```
void p_initiate_priorities ( )
```

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

4.16.1.16 p_initiate_shell()

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

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

Parameters

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

Returns

The pid of the thread created.

4.16.1.17 p_initiate_to_exit()

```
void p_initiate_to_exit ( )
```

A helper function to initiate the to_exit PCB stored for p_exit_process() as NULL.

4.16.1.18 p_kill()

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

Function used to send a thread to a running thread.

Parameters

pid	The pid of the thread we are trying to send a signal.
sig	The signal we are trying to send.

Returns

0 on success, -1 on error.

4.16.1.19 p_lookup_process()

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

A helper function to redirect k_lookup_process for abstraction sake.

Parameters

pid	The pid of the thread we are searching for.

Returns

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

4.16.1.20 p_nice()

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

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

Parameters

pid	The pid of the thread we want to set its priority.
priority	The new priority for this thread.

Returns

0 on success, -1 on error.

4.16.1.21 p_orphan_child()

```
void p_orphan_child ( )
```

The function used to spawn an orphan child.

4.16.1.22 p_orphanify()

```
void p_orphanify ( )
```

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

4.16.1.23 p_print_all_jobs()

```
void p_print_all_jobs ( )
```

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

4.16.1.24 p_remove_background_job()

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

The helper function to remove a thread from the background.

Parameters

pid	The pid for the thread to remove from the background queue.

4.16.1.25 p_remove_stop_job()

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

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

Parameters

pid	The pid for the thread to remove from the stopped queue.

4.16.1.26 p_run_cat_fs()

```
void p_run_cat_fs ( )
```

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

4.16.1.27 p_run_chmod_fs()

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

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

Parameters

filename	The name of the file descriptor.
perm	The target permission of the file descriptor.

4.16.1.28 p_run_cp_fs()

```
void p_run_cp_fs (
            char * source,
            char * des,
            int from_host )
```

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

Parameters

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

4.16.1.29 p_run_echo()

```
void p_run_echo ( )
```

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

4.16.1.30 p_run_f_ls_list()

```
void p_run_f_ls_list ( )
```

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

4.16.1.31 p_run_f_ls_null()

```
void p_run_f_ls_null ( )
```

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

4.16.1.32 p_run_kill()

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

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

Parameters

sig	The signal we are sending.
pid	The process we are sending the signal from.

4.16.1.33 p_run_mv_fs()

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

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

Parameters

source	The original name of the file descriptor.
dest	The target name of the file descriptor.

4.16.1.34 p_run_rm_fs()

```
void p_run_rm_fs ( )
```

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

4.16.1.35 p_run_touch_fs()

```
void p_run_touch_fs ( )
```

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

4.16.1.36 p_search_and_remove()

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

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

Parameters

pid	The pid for the thread to search and remove.

4.16.1.37 p_search_bg()

```
struct Process* p_search_bg (
            int pid )
```

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

Parameters

pid	The pid to search in the background.

Returns

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

4.16.1.38 p_search_most_recent()

```
int p_search_most_recent ( )
```

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

Returns

The pid of the most recent background/stopped job.

4.16.1.39 p_search_most_recent_stop()

```
int p_search_most_recent_stop ( )
```

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

Returns

The pid of the most recent stopped job.

4.16.1.40 p_setup()

```
void p_setup ( )
```

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

4.16.1.41 p_sleep()

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

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

Parameters

ticks	The number of ticks to sleep.

4.16.1.42 p_spawn()

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

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

Parameters

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

Returns

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

4.16.1.43 p_spawn_with_input()

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

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

Parameters

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

Returns

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

4.16.1.44 p_spawn_with_priority()

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

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

Parameters

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

Returns

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

4.16.1.45 p_waitpid()

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

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

Parameters

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

Returns

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

4.16.1.46 p_zombie_child()

```
void p_zombie_child ( )
```

The function used to spawn a Zombie child.

4.16.1.47 p_zombify()

```
void p_zombify ( )
```

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

4.16.1.48 W_WIFEXITED()

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

A helper function to check if the child terminates normally(calling p_exit).

Parameters

status	The status pointer we are looking at.

Returns

True if the child terminates normally, False otherwise.

4.16.1.49 W_WIFSIGNALED()

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

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

Parameters

status	The status pointer we are looking at.

Returns

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

4.16.1.50 W_WIFSTOPPED()

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

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

Parameters

status	The status pointer we are looking at.

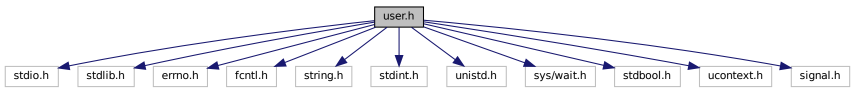
Returns

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

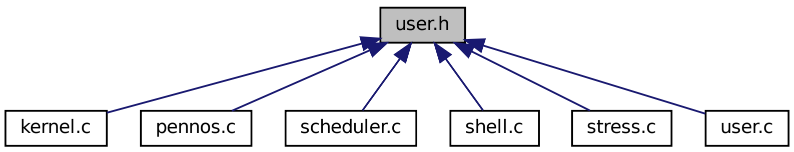
4.17 user.h File Reference

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

Include dependency graph for user.h:



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



Functions

- pid_t p_spawn (void(∗func)(), char ∗argv[ ], int fd0, int fd1)
- pid_t p_spawn_with_priority (void(∗func)(), char ∗argv[ ], int fd0, int fd1, int priority)
- pid_t p_waitpid (pid_t pid, int ∗wstatus, bool nohang)
- pid_t p_initiate_shell (void(∗func)(), int argc, char ∗argv[ ])
- int p_kill (pid_t pid, int sig)
- void p_exit (void)
- void p_exit_process ()
- bool W_WIFEXITED (int ∗status)
- bool W_WIFSTOPPED (int ∗status)
- bool W_WIFSIGNALED (int ∗status)
- int p_nice (pid_t pid, int priority)
- void p_sleep (unsigned int ticks)
- void p_busy_wait ()
- void p_zombie_child ()
- void p_orphan_child ()
- void p_print_all_jobs ()
- void p_add_background_job (int pid)
- void p_add_stop_job (int pid)
- void p_remove_background_job (int pid)
- void p_remove_stop_job (int pid)
- void p_search_and_remove (int pid)
- int p_search_most_recent ()
- int p_search_most_recent_stop ()
- struct Process ∗ p_search_bg (int pid)
- int p_get_sigstop_signal ()
- int p_get_sigcont_signal ()
- int p_get_sigterm_signal ()
- void p_initiate_to_exit ()
- struct Process ∗ p_get_current ()
- void p_initiate_priorities ()
- void p_setup ()
- void p_initiate ()
- pid_t p_spawn_with_input (void(∗func)(), char ∗argv[ ], int fd0, int fd1, char ∗∗actual_input)
- struct Process ∗ p_lookup_process (pid_t pid)
- void p_zombify ()
- void p_orphanify ()
- void p_run_kill (int sig, pid_t pid)
- void p_run_mv_fs (char ∗source, char ∗dest)
- void p_run_chmod_fs (char ∗filename, char ∗perm)
- void p_run_cp_fs (char ∗source, char ∗des, int from_host)
- void p_run_touch_fs ()
- void p_run_f_ls_list ()
- void p_run_f_ls_null ()
- void p_run_rm_fs ()
- void p_run_echo ()
- void p_run_cat_fs ()
- char ∗ p_get_sigstop_str ()
- char ∗ p_get_sigcont_str ()
- char ∗ p_get_sigterm_str ()

4.17.1 Function Documentation

4.17.1.1 p_add_background_job()

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

The helper function to add a thread into the background.

Parameters

pid	The pid for the thread to add into the background.

4.17.1.2 p_add_stop_job()

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

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

Parameters

pid	The pid for the thread to add into the stopped queue.

4.17.1.3 p_busy_wait()

```
void p_busy_wait ( )
```

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

4.17.1.4 p_exit()

```
void p_exit (
            void )
```

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

4.17.1.5 p_exit_process()

```
void p_exit_process ( )
```

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

4.17.1.6 p_get_current()

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

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

Returns

The PCB for the currently running thread.

4.17.1.7 p_get_sigcont_signal()

```
int p_get_sigcont_signal ( )
```

A helper function to get the S_SIGCONT we defined for abstraction.

Returns

The S_SIGCONT we defined.

4.17.1.8 p_get_sigcont_str()

```
char* p_get_sigcont_str ( )
```

The helper function we used to redirect into kernel and get S_SIGCONT_STR we defined inside of kernel for shell use.

Returns

The S_SIGSTOP_STR we defined inside of kernel.

4.17.1.9 p_get_sigstop_signal()

```
int p_get_sigstop_signal ( )
```

A helper function to get the S_SIGSTOP we defined for abstraction.

Returns

The S_SIGSTOP we defined.

4.17.1.10 p_get_sigstop_str()

```
char* p_get_sigstop_str ( )
```

The helper function we used to redirect into kernel and get S_SIGSTOP_STR we defined inside of kernel for shell use.

Returns

The S_SIGSTOP_STR we defined inside of kernel.

4.17.1.11 p_get_sigterm_signal()

```
int p_get_sigterm_signal ( )
```

A helper function to get the S_SIGTERM we defined for abstraction.

Returns

The S_SIGTERM we defined.

4.17.1.12 p_get_sigterm_str()

```
char* p_get_sigterm_str ( )
```

The helper function we used to redirect into kernel and get S_SIGTERM_STR we defined inside of kernel for shell use.

Returns

The S_SIGSTOP_STR we defined inside of kernel.

4.17.1.13 p_initiate()

```
void p_initiate ( )
```

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

4.17.1.14 p_initiate_priorities()

```
void p_initiate_priorities ( )
```

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

4.17.1.15 p_initiate_shell()

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

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

Parameters

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

Returns

The pid of the thread created.

4.17.1.16 p_initiate_to_exit()

```
void p_initiate_to_exit ( )
```

A helper function to initiate the to_exit PCB stored for p_exit_process() as NULL.

4.17.1.17 p_kill()

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

Function used to send a thread to a running thread.

Parameters

pid	The pid of the thread we are trying to send a signal.
sig	The signal we are trying to send.

Returns

0 on success, -1 on error.

4.17.1.18 p_lookup_process()

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

A helper function to redirect k_lookup_process for abstraction sake.

Parameters

pid	The pid of the thread we are searching for.

Returns

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

4.17.1.19 p_nice()

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

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

Parameters

pid	The pid of the thread we want to set its priority.
priority	The new priority for this thread.

Returns

0 on success, -1 on error.

4.17.1.20 p_orphan_child()

```
void p_orphan_child ( )
```

The function used to spawn an orphan child.

4.17.1.21 p_orphanify()

```
void p_orphanify ( )
```

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

4.17.1.22 p_print_all_jobs()

```
void p_print_all_jobs ( )
```

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

4.17.1.23 p_remove_background_job()

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

The helper function to remove a thread from the background.

Parameters

pid	The pid for the thread to remove from the background queue.

4.17.1.24 p_remove_stop_job()

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

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

Parameters

pid	The pid for the thread to remove from the stopped queue.

4.17.1.25 p_run_cat_fs()

```
void p_run_cat_fs ( )
```

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

4.17.1.26 p_run_chmod_fs()

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

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

Parameters

filename	The name of the file descriptor.
perm	The target permission of the file descriptor.

4.17.1.27 p_run_cp_fs()

```
void p_run_cp_fs (
            char * source,
            char * des,
            int from_host )
```

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

Parameters

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

4.17.1.28 p_run_echo()

```
void p_run_echo ( )
```

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

4.17.1.29 p_run_f_ls_list()

```
void p_run_f_ls_list ( )
```

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

4.17.1.30 p_run_f_ls_null()

```
void p_run_f_ls_null ( )
```

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

4.17.1.31 p_run_kill()

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

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

Parameters

sig	The signal we are sending.
pid	The process we are sending the signal from.

4.17.1.32 p_run_mv_fs()

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

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

Parameters

source	The original name of the file descriptor.
dest	The target name of the file descriptor.

4.17.1.33 p_run_rm_fs()

```
void p_run_rm_fs ( )
```

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

4.17.1.34 p_run_touch_fs()

```
void p_run_touch_fs ( )
```

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

4.17.1.35 p_search_and_remove()

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

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

Parameters

pid	The pid for the thread to search and remove.

4.17.1.36 p_search_bg()

```
struct Process* p_search_bg (
            int pid )
```

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

Parameters

pid	The pid to search in the background.

Returns

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

4.17.1.37 p_search_most_recent()

```
int p_search_most_recent ( )
```

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

Returns

The pid of the most recent background/stopped job.

4.17.1.38 p_search_most_recent_stop()

```
int p_search_most_recent_stop ( )
```

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

Returns

The pid of the most recent stopped job.

4.17.1.39 p_setup()

```
void p_setup ( )
```

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

4.17.1.40 p_sleep()

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

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

Parameters

ticks	The number of ticks to sleep.

4.17.1.41 p_spawn()

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

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

Parameters

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

Returns

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

4.17.1.42 p_spawn_with_input()

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

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

Parameters

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

Returns

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

4.17.1.43 p_spawn_with_priority()

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

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

Parameters

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

Returns

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

4.17.1.44 p_waitpid()

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

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

Parameters

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

Returns

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

4.17.1.45 p_zombie_child()

```
void p_zombie_child ( )
```

The function used to spawn a Zombie child.

4.17.1.46 p_zombify()

```
void p_zombify ( )
```

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

4.17.1.47 W_WIFEXITED()

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

A helper function to check if the child terminates normally(calling p_exit).

Parameters

status	The status pointer we are looking at.

Returns

True if the child terminates normally, False otherwise.

4.17.1.48 W_WIFSIGNALED()

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

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

Parameters

status	The status pointer we are looking at.

Returns

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

4.17.1.49 W_WIFSTOPPED()

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

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

Parameters

status	The status pointer we are looking at.

Returns

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

Index

Standalone PennFAT

Generated by Doxygen 1.9.1

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 DirectoryEntry Struct Reference

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

Public Attributes

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

3.1.1 Detailed Description

The DirectoryEntry struct we use to store the information for each file.

3.1.2 Member Data Documentation

3.1.2.1 firstBlock

```
uint16_t DirectoryEntry::firstBlock
```

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

3.1.2.2 mtime

```
time_t DirectoryEntry::mtime
```

time of last update for this file

3.1.2.3 name

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

filename

3.1.2.4 perm

```
uint8_t DirectoryEntry::perm
```

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

3.1.2.5 reserved

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

reserved bits for this file, unused

3.1.2.6 size

```
uint32_t DirectoryEntry::size
```

size of bytes written in the file

3.1.2.7 type

```
uint8_t DirectoryEntry::type
```

type of file: 0, 1, 2, 4

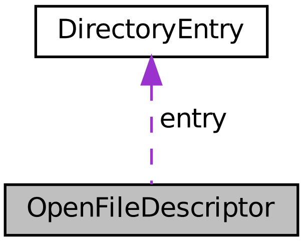
The documentation for this struct was generated from the following file:

- pennfat.h

3.2 OpenFileDescriptor Struct Reference

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

Collaboration diagram for OpenFileDescriptor:



Public Attributes

- int used

  Flag indicating if this descriptor is in use.
- DirectoryEntry ∗ entry

  Pointer to the associated directory entry.
- int mode

  Mode in which the file was opened.
- unsigned int cursor

  Current position in the file.

3.2.1 Detailed Description

Structure representing an open file descriptor.

3.2.2 Member Data Documentation

3.2.2.1 cursor

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

Current position in the file.

3.2.2.2 entry

```
DirectoryEntry* OpenFileDescriptor::entry
```

Pointer to the associated directory entry.

3.2.2.3 mode

```
int OpenFileDescriptor::mode
```

Mode in which the file was opened.

3.2.2.4 used

```
int OpenFileDescriptor::used
```

Flag indicating if this descriptor is in use.

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

- descriptors.h

3.3 PennFAT Struct Reference

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

Public Attributes

- uint16_t ∗ fat
- size_t fat_size
- size_t block_size
- int fs_fd
- int num_directories

3.3.1 Detailed Description

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

3.3.2 Member Data Documentation

3.3.2.1 block_size

```
size_t PennFAT::block_size
```

size of one data block

3.3.2.2 fat

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

pointer to the file allocation table (fat)

3.3.2.3 fat_size

```
size_t PennFAT::fat_size
```

size of the fat entry

3.3.2.4 fs_fd

```
int PennFAT::fs_fd
```

file descriptor of the file system

3.3.2.5 num_directories

```
int PennFAT::num_directories
```

number of directories in the file system, unused

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

- pennfat.h

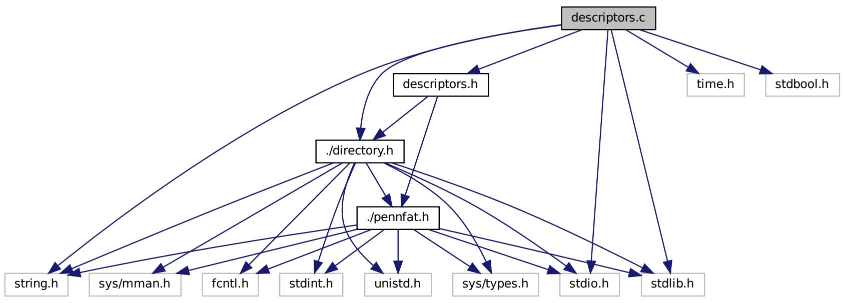
Chapter 4

File Documentation

4.1 descriptors.c File Reference

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

Include dependency graph for descriptors.c:



Functions

- int initialize_file_descriptor (DirectoryEntry ∗entry, int mode)
- int f_open (const char ∗fname, int mode)
- int f_read (int fd, int n, char ∗buf)
- int f_write (int fd, const char ∗str, int n)
- void clear_file_content (DirectoryEntry ∗entry)
- int f_close (int fd)
- int f_unlink (const char ∗fname)
- int f_lseek (int fd, int offset, int whence)
- void f_ls (const char ∗filename)
- int f_dup2 (int fd_curr, int fd_new)
- int find_descriptor_by_name (const char ∗fname)

    Finds the file descriptor for a given file name.

Variables

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

4.1.1 Function Documentation

4.1.1.1 clear_file_content()

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

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

Parameters

entry	Directory entry of the file to clear.

4.1.1.2 f_close()

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

Closes an open file descriptor.

Parameters

fd	File descriptor to close.

Returns

0 on success, -1 on error.

4.1.1.3 f_dup2()

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

Duplicates a file descriptor.

Parameters

fd_curr	Current file descriptor.
fd_new	New file descriptor.

Returns

New file descriptor on success, -1 on error.

4.1.1.4 f_ls()

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

Lists files in a directory.

Parameters

filename	Name of the file or directory to list.

4.1.1.5 f_lseek()

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

Sets the file cursor of an open file descriptor.

Modes: F_SEEK_SET - The file offset is set to offset bytes. F_SEEK_CUR - The file offset is set to its current location plus offset bytes. F_SEEK_END - The file offset is set to the size of the file plus offset bytes.

Parameters

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

Returns

New cursor position on success, -1 on error.

4.1.1.6 f_open()

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

Opens a file named fname in the specified mode.

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

Parameters

fname	Name of the file to open. Filename should follow POSIX standards.
mode	Mode to open the file.

Returns

File descriptor on success, negative value on error.

4.1.1.7 f_read()

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

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

Parameters

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

Returns

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

4.1.1.8 f_unlink()

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

Deletes a file with the specified name.

Parameters

fname	Name of the file to delete.

Returns

0 on success, -1 on error.

4.1.1.9 f_write()

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

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

Parameters

fd	File descriptor to write to.
str	String containing the data to write.
n	Number of bytes to write.

Returns

Number of bytes written, negative value on error.

4.1.1.10 find_descriptor_by_name()

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

Finds the file descriptor for a given file name.

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

Parameters

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

Returns

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

4.1.1.11 initialize_file_descriptor()

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

4.1.2 Variable Documentation

4.1.2.1 curr_stdin

```
int curr_stdin
```

4.1.2.2 curr_stdout

```
int curr_stdout
```

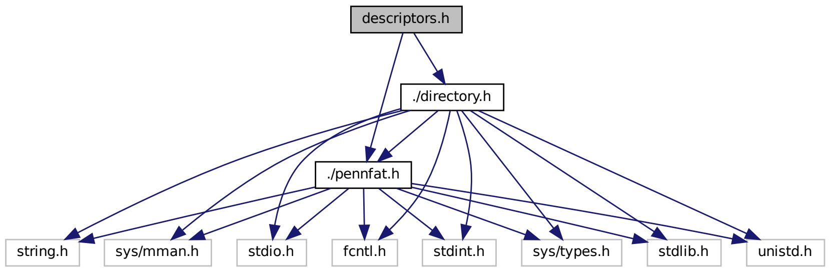
4.1.2.3 openFileDescriptors

```
OpenFileDescriptor openFileDescriptors[MAX_OPEN_FILES]
```

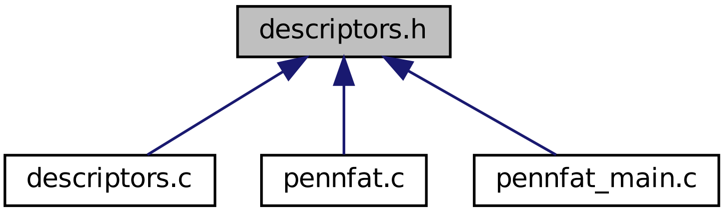
4.2 descriptors.h File Reference

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

Include dependency graph for descriptors.h:



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



Classes

- struct OpenFileDescriptor

Macros

- #define MAX_OPEN_FILES 100
- #define F_WRITE 1

    Writing mode, truncates file if exists, creates if not. Only one instance can be open.
- #define F_READ 2

    Read-only mode, error if file does not exist.
- #define F_APPEND 3

    Append mode, for reading and writing. Does not truncate, file pointer at end.
- #define F_SEEK_SET 0

    Seek from the beginning of the file.
- #define F_SEEK_CUR 1

    Seek from the current file position.
- #define F_SEEK_END 2

    Seek from the end of the file.

Functions

- int f_open (const char ∗fname, int mode)
- int f_read (int fd, int n, char ∗buf)
- int f_write (int fd, const char ∗str, int n)
- void clear_file_content (DirectoryEntry ∗entry)
- int f_unlink (const char ∗fname)
- int f_lseek (int fd, int offset, int whence)
- int f_close (int fd)
- void f_ls (const char ∗filename)
- int f_dup2 (int fd_curr, int fd_new)
- int find_descriptor_by_name (const char ∗fname)

    Finds the file descriptor for a given file name.

4.2.1 Macro Definition Documentation

4.2.1.1 F_APPEND

```
#define F_APPEND 3
```

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

4.2.1.2 F_READ

```
#define F_READ 2
```

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

4.2.1.3 F_SEEK_CUR

```
#define F_SEEK_CUR 1
```

Seek from the current file position.

4.2.1.4 F_SEEK_END

```
#define F_SEEK_END 2
```

Seek from the end of the file.

4.2.1.5 F_SEEK_SET

```
#define F_SEEK_SET 0
```

Seek from the beginning of the file.

4.2.1.6 F_WRITE

```
#define F_WRITE 1
```

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

4.2.1.7 MAX_OPEN_FILES

```
#define MAX_OPEN_FILES 100
```

4.2.2 Function Documentation

4.2.2.1 clear_file_content()

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

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

Parameters

entry	Directory entry of the file to clear.

4.2.2.2 f_close()

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

Closes an open file descriptor.

Parameters

fd	File descriptor to close.

Returns

0 on success, -1 on error.

4.2.2.3 f_dup2()

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

Duplicates a file descriptor.

Parameters

fd_curr	Current file descriptor.
fd_new	New file descriptor.

Returns

New file descriptor on success, -1 on error.

4.2.2.4 f_ls()

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

Lists files in a directory.

Parameters

filename	Name of the file or directory to list.

4.2.2.5 f_lseek()

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

Sets the file cursor of an open file descriptor.

Modes: F_SEEK_SET - The file offset is set to offset bytes. F_SEEK_CUR - The file offset is set to its current location plus offset bytes. F_SEEK_END - The file offset is set to the size of the file plus offset bytes.

Parameters

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

Returns

New cursor position on success, -1 on error.

4.2.2.6 f_open()

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

Opens a file named fname in the specified mode.

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

Parameters

fname	Name of the file to open. Filename should follow POSIX standards.
mode	Mode to open the file.

Returns

File descriptor on success, negative value on error.

4.2.2.7 f_read()

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

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

Parameters

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

Returns

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

4.2.2.8 f_unlink()

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

Deletes a file with the specified name.

Parameters

fname	Name of the file to delete.

Returns

0 on success, -1 on error.

4.2.2.9 f_write()

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

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

Parameters

fd	File descriptor to write to.
str	String containing the data to write.
n	Number of bytes to write.

Returns

Number of bytes written, negative value on error.

4.2.2.10 find_descriptor_by_name()

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

Finds the file descriptor for a given file name.

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

Parameters

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

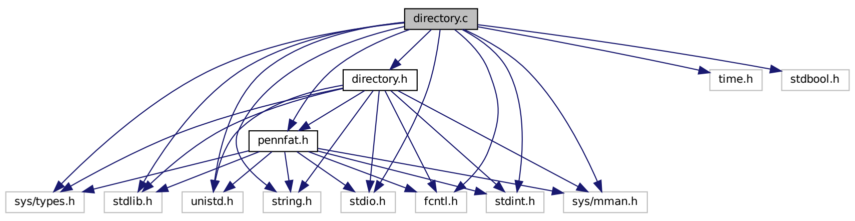
Returns

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

4.3 directory.c File Reference

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

Include dependency graph for directory.c:



Functions

- unsigned int find_free_block ()
- unsigned int fetch_block_number (uint16_t start_block, unsigned int block_offset)
- unsigned int allocate_new_block ()
- void update_fat_entry (int fs_fd, uint16_t current_block, uint16_t new_block)
- void update_directory_entry (int fs_fd, DirectoryEntry ∗entry)

4.3.1 Function Documentation

4.3.1.1 allocate_new_block()

```
unsigned int allocate_new_block ( )
```

Allocates a new block in the file system.

Returns

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

4.3.1.2 fetch_block_number()

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

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

Parameters

start_block	The starting block number.
block_offset	The block offset from the start block.

Returns

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

4.3.1.3 find_free_block()

```
unsigned int find_free_block ( )
```

Finds a free block in the file system.

Returns

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

4.3.1.4 update_directory_entry()

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

Updates a directory entry in the file system.

Parameters

fs←↩ _fd	File descriptor for the file system.
entry	Pointer to the directory entry to update.

4.3.1.5 update_fat_entry()

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

Updates a FAT entry in the file system.

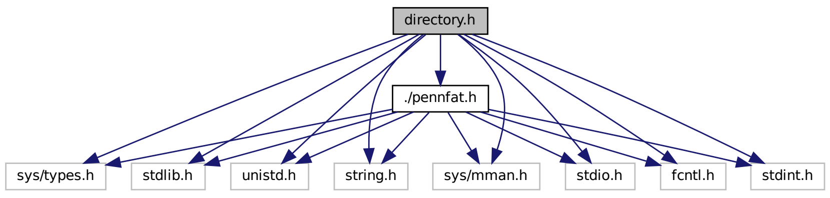
Parameters

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

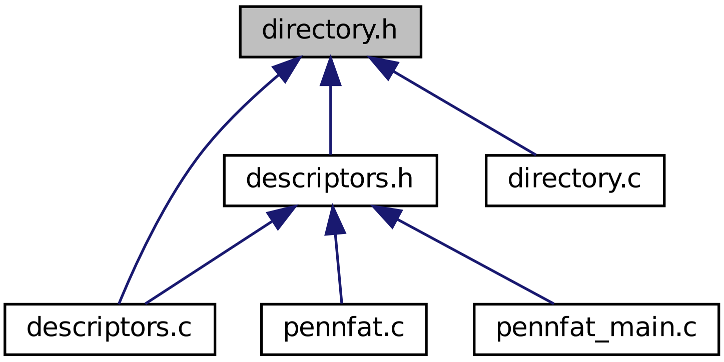
4.4 directory.h File Reference

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

Include dependency graph for directory.h:



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



Macros

- #define F_WRITE 1

    Writing mode.
- #define F_READ 2

    Read-only mode.
- #define F_APPEND 3

    Append mode.

Functions

- unsigned int find_free_block ()
- unsigned int fetch_block_number (uint16_t start_block, unsigned int block_offset)
- unsigned int allocate_new_block ()
- void update_fat_entry (int fs_fd, uint16_t current_block, uint16_t new_block)
- void update_directory_entry (int fs_fd, DirectoryEntry ∗entry)

4.4.1 Macro Definition Documentation

4.4.1.1 F_APPEND

```
#define F_APPEND 3
```

Append mode.

4.4.1.2 F_READ

```
#define F_READ 2
```

Read-only mode.

4.4.1.3 F_WRITE

```
#define F_WRITE 1
```

Writing mode.

4.4.2 Function Documentation

4.4.2.1 allocate_new_block()

```
unsigned int allocate_new_block ( )
```

Allocates a new block in the file system.

Returns

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

4.4.2.2 fetch_block_number()

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

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

Parameters

start_block	The starting block number.
block_offset	The block offset from the start block.

Returns

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

4.4.2.3 find_free_block()

```
unsigned int find_free_block ( )
```

Finds a free block in the file system.

Returns

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

4.4.2.4 update_directory_entry()

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

Updates a directory entry in the file system.

Parameters

fs↩ _fd	File descriptor for the file system.
entry	Pointer to the directory entry to update.

4.4.2.5 update_fat_entry()

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

Updates a FAT entry in the file system.

Parameters

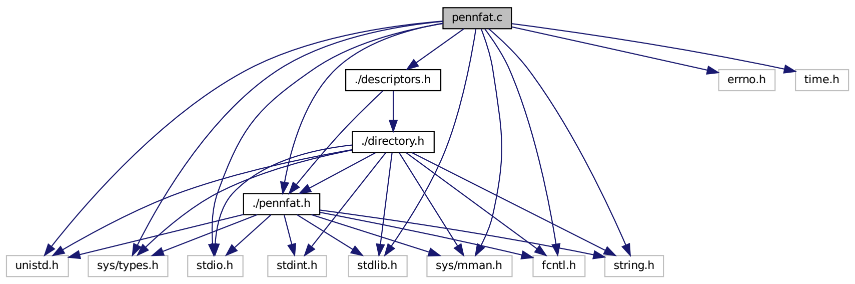
fs_fd	File descriptor for the file system.
current_block	The current block to be updated.
new_block	The new block number to update to.

4.5 pennfat.c File Reference

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

```
#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <time.h>
#include "./pennfat.h"
#include "./descriptors.h"
```

Include dependency graph for pennfat.c:



Functions

- void mkfs (char ∗fs_name, int blocks_in_fat, int block_size_config)
- void mount_fs (char ∗fs_name)
- void unmount_fs ()
- uint16_t touch_fs (char ∗filename)
- void ls_fs ()
- DirectoryEntry ∗ find_file (int ∗directory_block_offset, int ∗directory_entry_offset, const char ∗filename)
- void mv_fs (const char ∗source, const char ∗dest)
- void trim_newline (char ∗str)
- void cp_fs (char ∗source, char ∗dest, int from_host)
- void rm_fs (const char ∗filename)
- void cat_fs (int argc, char ∗∗argv)
- void chmod_fs (char ∗filename, char ∗perm)

Variables

- struct PennFAT ∗ pf = NULL

4.5.1 Function Documentation

4.5.1.1 cat_fs()

```
void cat_fs (
            int argc,
            char ** argv )
```

Reads in the `cat` command from the command line, parse argv according to the different formats of `cat`. Specifically, a `cat` command can:

1. `cat FILE... -w/a OUTPUT_FILE`: concatenates the content in FILE and write/append it to OUTPUT_FILE.
2. `cat -w/a OUTPUT_FILE`: reads content from stdin and writes/appends it to OUTPUT_FILE.
3. `cat FILE...`: concatenates all content in FILE and write to stdout.

Parameters

argc	number of parameters read from the `cat` command
argv	array of parameters

4.5.1.2 chmod_fs()

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

Changes the permission for the file named filename according to the specifications in perm string.

Parameters

filename	name of the file whose permission we are changing
perm	string that specifies modification for permission

4.5.1.3 cp_fs()

```
void cp_fs (
            char * source,
            char * dest,
            int from_host )
```

Copies the content from a file named source to the file named dest. If source doesn't exist, an error is given. If dest doesn't exist, creates the new file named dest.

Parameters

source	name of source file
dest	name of dest file
from_host	0: both files are in directory, 1: source is from host, 2: dest is from host

4.5.1.4 find_file()

```
DirectoryEntry* find_file (
             int * directory_block_offset,
             int * directory_entry_offset,
             const char * filename )
```

Iterates through all the directory blocks and attempts to find the file named filename. If found, directory_block_offset is the index of the block that this directory entry is in, and directory_entry_offset is the number of entry this specific file is in this block.

Parameters

directory_block_offset	index of data block that stores this directory entry
directory_entry_offset	index of directory entry within this data block
filename	name of file we are attempting to find.

4.5.1.5 ls_fs()

```
void ls_fs ( )
```

Lists all files in the directory.

4.5.1.6 mkfs()

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

Creates a PennFAT filesystem in the file named FS_NAME. The number of blocks in the FAT region is BLOCKS↩_IN_FAT (ranging from 1 through 32), and the block size is 256, 512, 1024, 2048, or 4096 bytes corresponding to the value (0 through 4) of BLOCK_SIZE_CONFIG fed into BLOCK_SIZES[].

Parameters

fs_name	filename
blocks_in_fat	number of blocks in the FAT region (1-32)
block_size_config	a number 0-4 that corresponds to block size

4.5.1.7 mount_fs()

```
void mount_fs (
             char * fs_name )
```

Mounts the filesystem named FS_NAME by loading the FAT into memory.

Parameters

fs_name	name of the file we are mounting

4.5.1.8 mv_fs()

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

Renames the file named source to dest

Parameters

source	name of the file wanting to rename
dest	name of file renamed to

4.5.1.9 rm_fs()

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

Removes the file named filename from the directory

Parameters

filename	name of the file to remove

4.5.1.10 touch_fs()

```
uint16_t touch_fs (
             char * filename )
```

Creates a DirectoryEntry for a new file called filename and stores the DirectoryEntry in a directory block.

Parameters

filename	name of the file we want to create/touch

4.5.1.11 trim_newline()

```
void trim_newline (
             char * str )
```

Helper function to trim the last '
' character from str.

Parameters

str	string to trim ' ' from

4.5.1.12 unmount_fs()

```
void unmount_fs ( )
```

Unmount the current filesystem in memory.

4.5.2 Variable Documentation

4.5.2.1 pf

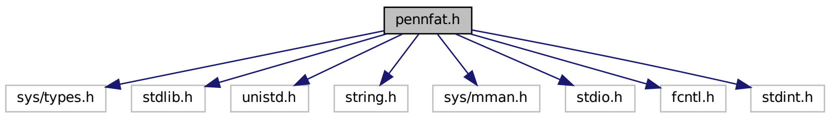
```
struct PennFAT* pf = NULL
```

Pointer to the PennFAT struct pf, which is accessible all throughout the program after the file system is mounted.

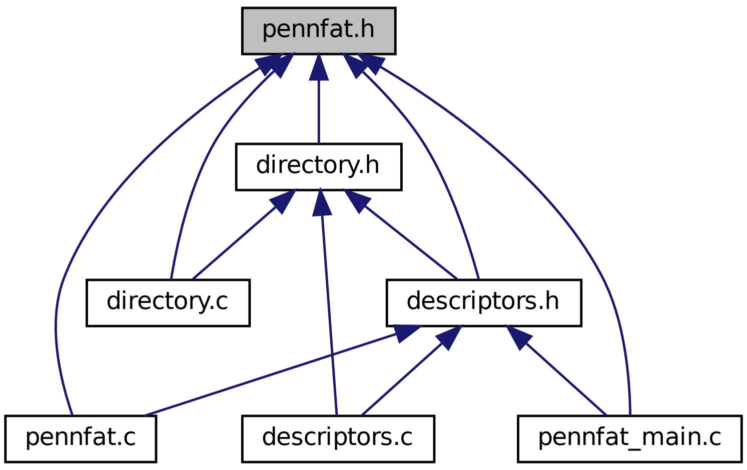
4.6 pennfat.h File Reference

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

Include dependency graph for pennfat.h:



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



Classes

- struct DirectoryEntry
- struct PennFAT

Macros

- #define CAT_BUFFER_SIZE 4096

Typedefs

- typedef struct DirectoryEntry DirectoryEntry
- typedef struct PennFAT PennFAT

Functions

- void mkfs (char ∗fs_name, int blocks_in_fat, int block_size_config)
- void mount_fs (char ∗fs_name)
- void unmount_fs ()
- uint16_t touch_fs (char ∗filename)
- void cat_fs (int argc, char ∗∗argv)
- void ls_fs ()
- void mv_fs (const char ∗source, const char ∗dest)
- void rm_fs (const char ∗filename)
- void trim_newline (char ∗str)
- void cp_fs (char ∗source, char ∗dest, int from_host)
- void chmod_fs (char ∗filename, char ∗perm)
- DirectoryEntry ∗ find_file (int ∗directory_block_offset, int ∗directory_entry_offset, const char ∗filename)

Variables

- struct PennFAT ∗ pf

4.6.1 Macro Definition Documentation

4.6.1.1 CAT_BUFFER_SIZE

```
#define CAT_BUFFER_SIZE 4096
```

buffer for writing

4.6.2 Typedef Documentation

4.6.2.1 DirectoryEntry

```
typedef struct DirectoryEntry DirectoryEntry
```

The DirectoryEntry struct we use to store the information for each file.

4.6.2.2 PennFAT

```
typedef struct PennFAT PennFAT
```

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

4.6.3 Function Documentation

4.6.3.1 cat_fs()

```
void cat_fs (
             int argc,
             char ** argv )
```

Reads in the `cat` command from the command line, parse argv according to the different formats of `cat`. Specifically, a `cat` command can:

1. `cat FILE... -w/a OUTPUT_FILE`: concatenates the content in FILE and write/append it to OUTPUT_FILE.
2. `cat -w/a OUTPUT_FILE`: reads content from stdin and writes/appends it to OUTPUT_FILE.
3. `cat FILE...`: concatenates all content in FILE and write to stdout.

Parameters

argc	number of parameters read from the `cat` command
argv	array of parameters

4.6.3.2 chmod_fs()

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

Changes the permission for the file named filename according to the specifications in perm string.

Parameters

filename	name of the file whose permission we are changing
perm	string that specifies modification for permission

4.6.3.3 cp_fs()

```
void cp_fs (
            char * source,
            char * dest,
            int from_host )
```

Copies the content from a file named source to the file named dest. If source doesn't exist, an error is given. If dest doesn't exist, creates the new file named dest.

Parameters

source	name of source file
dest	name of dest file
from_host	0: both files are in directory, 1: source is from host, 2: dest is from host

4.6.3.4 find_file()

```
DirectoryEntry* find_file (
            int * directory_block_offset,
            int * directory_entry_offset,
            const char * filename )
```

Iterates through all the directory blocks and attempts to find the file named filename. If found, directory_block_offset is the index of the block that this directory entry is in, and directory_entry_offset is the number of entry this specific file is in this block.

Parameters

directory_block_offset	index of data block that stores this directory entry
directory_entry_offset	index of directory entry within this data block
filename	name of file we are attempting to find.

4.6.3.5 ls_fs()

```
void ls_fs ( )
```

Lists all files in the directory.

4.6.3.6 mkfs()

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

Creates a PennFAT filesystem in the file named FS_NAME. The number of blocks in the FAT region is BLOCKS↩
_IN_FAT (ranging from 1 through 32), and the block size is 256, 512, 1024, 2048, or 4096 bytes corresponding to the value (0 through 4) of BLOCK_SIZE_CONFIG fed into BLOCK_SIZES[].

Parameters

fs_name	filename
blocks_in_fat	number of blocks in the FAT region (1-32)
block_size_config	a number 0-4 that corresponds to block size

4.6.3.7 mount_fs()

```
void mount_fs (
            char * fs_name )
```

Mounts the filesystem named FS_NAME by loading the FAT into memory.

Parameters

fs_name	name of the file we are mounting

4.6.3.8 mv_fs()

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

Renames the file named source to dest

Parameters

source	name of the file wanting to rename
dest	name of file renamed to

4.6.3.9 rm_fs()

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

Removes the file named filename from the directory

Parameters

filename	name of the file to remove

4.6.3.10 touch_fs()

```
uint16_t touch_fs (
             char * filename )
```

Creates a DirectoryEntry for a new file called filename and stores the DirectoryEntry in a directory block.

Parameters

filename	name of the file we want to create/touch

4.6.3.11 trim_newline()

```
void trim_newline (
             char * str )
```

Helper function to trim the last '
' character from str.

Parameters

str	string to trim ' ' from

4.6.3.12 unmount_fs()

```
void unmount_fs ( )
```

Unmount the current filesystem in memory.

4.6.4 Variable Documentation

4.6.4.1 pf

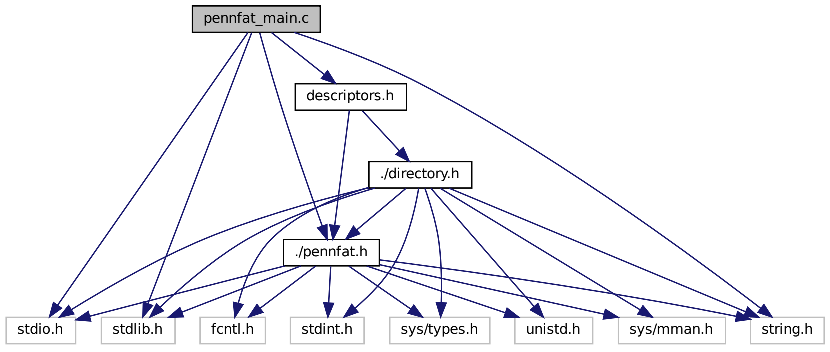
```
struct PennFAT* pf  [extern]
```

Pointer to the PennFAT struct pf, which is accessible all throughout the program after the file system is mounted.

4.7 pennfat_main.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "./pennfat.h"
#include "descriptors.h"
```

Include dependency graph for pennfat_main.c:



Functions

- int main (int argc, char ∗argv[ ])

4.7.1 Function Documentation

4.7.1.1 main()

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

The main function of our standalone PennFaT, used to initiate everything and allow PennFat to execute the commands required.

Parameters

argc	The number of arguments passed in from the terminal.
argv	The arguments from terminal.

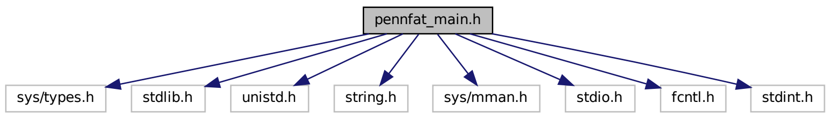
Returns

Anything on exit.

4.8 pennfat_main.h File Reference

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

Include dependency graph for pennfat_main.h:



Functions

- int main (int argc, char ∗argv[ ])

4.8.1 Function Documentation

4.8.1.1 main()

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

The main function of our standalone PennFaT, used to initiate everything and allow PennFat to execute the commands required.

Parameters

argc	The number of arguments passed in from the terminal.
argv	The arguments from terminal.

Returns

Anything on exit.

Index