PennOS

Generated by Doxygen 1.13.2

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 block_t Struct Reference

block data type (basically tuple of idx and bytes)

`#include <fs.h>`

Data Fields

- int idx
  index of the block in the fat
- char * data
  raw bytes of the block
- void * mmap_ptr
  pointer to mmap region for page compatabilty

3.1.1 Detailed Description

block data type (basically tuple of idx and bytes)

3.1.2 Field Documentation

3.1.2.1 idx

`int idx`

index of the block in the fat

3.1.2.2 data

`char* data`

raw bytes of the block

3.1.2.3 mmap_ptr

```
void* mmap_ptr
```

pointer to mmap region for page compatabilty

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

- fs.h

3.2 builtin_t Struct Reference

Struct representing a builtin command.

```
#include <pennos.h>
```

Data Fields

- const char * name
- void *(* func)(void *)
- const char * description

3.2.1 Detailed Description

Struct representing a builtin command.

3.2.2 Field Documentation

3.2.2.1 name

```
const char* name
```

3.2.2.2 func

```
void *(* func) (void *)
```

3.2.2.3 description

```
const char* description
```

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

- pennos.h

3.3 dir_t Struct Reference

struct for directory

`#include <dir.h>`

Data Fields

- uint16_t idx
  index of first block of the directory
- Vec vec
  vector of dirent pointers

3.3.1 Detailed Description

struct for directory

3.3.2 Field Documentation

3.3.2.1 idx

`uint16_t idx`

index of first block of the directory

3.3.2.2 vec

`Vec vec`

vector of dirent pointers

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

- dir.h

3.4 dirent_t Struct Reference

struct for directory entry

`#include <dir.h>`

Data Fields

- char name [MAX_FILENAME_LEN]
- uint32_t size

  file size
- uint16_t first_block

  index of first block
- uint8_t type
- uint8_t perm
- time_t mtime

  creation/modification time as returned by time(2)
- char padding [16]

  16 byte padding to make size 64B

3.4.1 Detailed Description

struct for directory entry

3.4.2 Field Documentation

3.4.2.1 name

```
char name[MAX_FILENAME_LEN]
```

name of file, name[0] is a special marker name[0] = 0: end of directory name[0] = 1: deleted entry name[0] = 2: deleted entry, file still in use

3.4.2.2 size

```
uint32_t size
```

file size

3.4.2.3 first_block

```
uint16_t first_block
```

index of first block

3.4.2.4 type

```
uint8_t type
```

type of file 0: unknown 1: regular file 2: directory 3: symlink

3.4.2.5 perm

```
uint8_t perm
```

file permissions 0: none 2: write only 4: read only 5: read and executable (shell scripts) 6: read and write 7: read write and execute

3.4.2.6 mtime

```
time_t mtime
```

creation/modification time as returned by time(2)

3.4.2.7 padding

```
char padding[16]
```

16 byte padding to make size 64B

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

- dir.h

3.5 fd_t Struct Reference

file descriptor struct

```
#include <fd.h>
```

Data Fields

- char filename [MAX_FILENAME_LEN]
  file name
- int first_block
  first data block
- int mode
  mode of current open file
- int offset
  where it reads/writes from
- int in_use
  1 if in use 0 if not in use
- int stdin_out_err
  1 if one of stdin/out/err, 0 if not
- int dir_idx
  dirent index in dir
- int ref_count
- int global_index

3.5.1 Detailed Description

file descriptor struct

3.5.2 Field Documentation

3.5.2.1 filename

```
char filename[MAX_FILENAME_LEN]
```

file name

3.5.2.2 first_block

```
int first_block
```

first data block

3.5.2.3 mode

```
int mode
```

mode of current open file

3.5.2.4 offset

```
int offset
```

where it reads/writes from

3.5.2.5 in_use

```
int in_use
```

1 if in use 0 if not in use

3.5.2.6 stdin_out_err

```
int stdin_out_err
```

1 if one of stdin/out/err, 0 if not

3.5.2.7 dir_idx

```
int dir_idx
```

dirent index in dir

3.5.2.8 ref_count

```
int ref_count
```

3.5.2.9 global_index

```
int global_index
```

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

- fd.h

3.6 fdt_t Struct Reference

file desctriptor table struct

```
#include <fd.h>
```

Data Fields

- int pid
  process id of the table
- Vec vec

3.6.1 Detailed Description

file desctriptor table struct

3.6.2 Field Documentation

3.6.2.1 pid

```
int pid
```

process id of the table

3.6.2.2 vec

```
Vec vec
```

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

- fd.h

3.7 fs_t Struct Reference

structure for file system

```
#include <fs.h>
```

Data Fields

- char * name
  name of file system
- int num_entries
  number of entries in the fat
- int block_size
  size of each block
- int fat_size
  blocks in fat
- int fd
  file descriptor of the file system
- uint16_t * fat
  file allocation table

3.7.1 Detailed Description

structure for file system

3.7.2 Field Documentation

3.7.2.1 name

```
char* name
```

name of file system

3.7.2.2 num_entries

```
int num_entries
```

number of entries in the fat

3.7.2.3 block_size

```
int block_size
```

size of each block

3.7.2.4 fat_size

```
int fat_size
```

blocks in fat

3.7.2.5 fd

```
int fd
```

file descriptor of the file system

3.7.2.6 fat

```
uint16_t* fat
```

file allocation table

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

- fs.h

3.8 job Struct Reference

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

Data Fields

- int job_id
- pid_t pid
- Vec processes
- char * command
- int status
- bool is_background

3.8.1 Field Documentation

3.8.1.1 job_id

```
int job_id
```

3.8.1.2 pid

```
pid_t pid
```

3.8.1.3 processes

```
Vec processes
```

3.8.1.4 command

```
char* command
```

3.8.1.5 status

```
int status
```

3.8.1.6 is_background

```
bool is_background
```

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

- job.h

3.9 parsed_command Struct Reference

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

Data Fields

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

3.9.1 Detailed Description

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

3.9.2 Field Documentation

3.9.2.1 is_background

```
bool is_background
```

3.9.2.2 is_file_append

```
bool is_file_append
```

3.9.2.3 stdin_file

```
const char* stdin_file
```

3.9.2.4 stdout_file

```
const char* stdout_file
```

3.9.2.5 num_commands

```
size_t num_commands
```

3.9.2.6 nice_value

```
size_t nice_value
```

3.9.2.7 commands

```
char** commands[]
```

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

- util/parser.h

3.10 pcb Struct Reference

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

Data Fields

- spthread_t thread
- int pid
- char * name
- int priority
- int state
- struct pcb * parent
- Vec children
- Vec zombie_children
- Vec waitable_children
- int stdin_fd
- int stdout_fd
- int wake
- bool sleeping
- int exit_status
- int job_id
- fdt_t * fdt
- bool signal_handled

3.10.1 Field Documentation

3.10.1.1 thread

```
spthread_t thread
```

3.10.1.2 pid

```
int pid
```

3.10.1.3 name

```
char* name
```

3.10.1.4 priority

```
int priority
```

3.10.1.5 state

```
int state
```

3.10.1.6 parent

```
struct pcb* parent
```

3.10.1.7 children

```
Vec children
```

3.10.1.8 zombie_children

```
Vec zombie_children
```

3.10.1.9 waitable_children

```
Vec waitable_children
```

3.10.1.10 stdin_fd

```
int stdin_fd
```

3.10.1.11 stdout_fd

```
int stdout_fd
```

3.10.1.12 wake

```
int wake
```

3.10.1.13 sleeping

```
bool sleeping
```

3.10.1.14 exit_status

```
int exit_status
```

3.10.1.15 job_id

```
int job_id
```

3.10.1.16 fdt

```
fdt_t* fdt
```

3.10.1.17 signal_handled

```
bool signal_handled
```

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

- scheduler.h

3.11 PriorityQueue Struct Reference

```
#include <queue.h>
```

Data Fields

- Vec * queues
- int levels

3.11.1 Field Documentation

3.11.1.1 queues

```
Vec* queues
```

3.11.1.2 levels

```
int levels
```

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

- queue.h

3.12 proc_info_t Struct Reference

Process information structure for process listing.

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

Data Fields

- pid_t pid
- pid_t ppid
- int priority
- int state
- char name [32]

3.12.1 Detailed Description

Process information structure for process listing.

3.12.2 Field Documentation

3.12.2.1 pid

```
pid_t pid
```

3.12.2.2 ppid

```
pid_t ppid
```

3.12.2.3 priority

```
int priority
```

3.12.2.4 state

```
int state
```

3.12.2.5 name

```
char name[32]
```

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

- kernel.h

3.13 process_queue_t Struct Reference

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

Data Fields

- pcb_t * head
- pcb_t * tail
- int count

3.13.1 Field Documentation

3.13.1.1 head

```
pcb_t* head
```

3.13.1.2 tail

```
pcb_t* tail
```

3.13.1.3 count

```
int count
```

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

- scheduler.h

3.14 spthread_fwd_args_st Struct Reference

Data Fields

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

3.14.1 Field Documentation

3.14.1.1 actual_routine

```
pthread_fn actual_routine
```

3.14.1.2 actual_arg

```
void* actual_arg
```

3.14.1.3 setup_done

```
bool setup_done
```

3.14.1.4 setup_mutex

```
pthread_mutex_t setup_mutex
```

3.14.1.5 setup_cond

```
pthread_cond_t setup_cond
```

3.14.1.6 child_meta

```
spthread_meta_t* child_meta
```

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

- util/spthread.c

3.15 spthread_meta_st Struct Reference

Data Fields

- sigset_t suspend_set
- volatile sig_atomic_t state
- pthread_mutex_t meta_mutex

3.15.1 Field Documentation

3.15.1.1 suspend_set

```
sigset_t suspend_set
```

3.15.1.2 state

```
volatile sig_atomic_t state
```

3.15.1.3 meta_mutex

```
pthread_mutex_t meta_mutex
```

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

- util/spthread.c

3.16 spthread_signal_args_st Struct Reference

Data Fields

- const int signal
- volatile sig_atomic_t ack
- pthread_mutex_t shutup_mutex

3.16.1 Field Documentation

3.16.1.1 signal

```
const int signal
```

3.16.1.2 ack

```
volatile sig_atomic_t ack
```

3.16.1.3 shutup_mutex

```
pthread_mutex_t shutup_mutex
```

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

- util/spthread.c

3.17 spthread_st Struct Reference

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

Data Fields

- pthread_t thread
- spthread_meta_t * meta

3.17.1 Field Documentation

3.17.1.1 thread

```
pthread_t thread
```

3.17.1.2 meta

```
spthread_meta_t* meta
```

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

- util/spthread.h

3.18 standard_fds_t Struct Reference

Data Fields

- int stdin_fd
- int stdout_fd

3.18.1 Field Documentation

3.18.1.1 stdin_fd

```
int stdin_fd
```

3.18.1.2 stdout_fd

```
int stdout_fd
```

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

- system.c

3.19 subroutine_t Struct Reference

Struct representing a subroutine.

```
#include <pennos.h>
```

Data Fields

- const char * name
- const char * description

3.19.1 Detailed Description

Struct representing a subroutine.

3.19.2 Field Documentation

3.19.2.1 name

```
const char* name
```

3.19.2.2 description

```
const char* description
```

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

- pennos.h

3.20 vec_st Struct Reference

```
#include <Vec.h>
```

Data Fields

- ptr_t * data
- size_t length
- size_t capacity
- ptr_dtor_fn ele_dtor_fn

3.20.1 Field Documentation

3.20.1.1 data

```
ptr_t* data
```

3.20.1.2 length

```
size_t length
```

3.20.1.3 capacity

```
size_t capacity
```

3.20.1.4 ele_dtor_fn

```
ptr_dtor_fn ele_dtor_fn
```

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

- util/Vec.h

Chapter 4

File Documentation

4.1 builtins.c File Reference

```
#include "builtins.h"
#include "pennos.h"
#include "system.h"
```

Functions

- void print_system_error (const char *message)

  Helper function to print errors to stderr.
- int count_args (char **argv)

  Helper function to count number of arguments.
- void * cat (void *arg)

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

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

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

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

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

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

  Remove a list of files. Treat each file in the list as a separate transaction. (i.e. if removing file1 fails, still attempt to remove file2, file3, etc.)
- uint8_t str_to_perms (char *str)

  converts a string into permissions integer
- void * chmod (void *arg)

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

4.1.1 Function Documentation

4.1.1.1 print_system_error()

```
void print_system_error (
            const char * message)
```

Helper function to print errors to stderr.

Parameters

message	- error message

4.1.1.2 count_args()

```
int count_args (
            char ** argv)
```

Helper function to count number of arguments.

Parameters

argv	- list of arguments

4.1.1.3 cat()

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

The ususal `cat` program.

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

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

4.1.1.4 busy()

```
void * busy (
            void * arg)
```

Sleep for `n` seconds.

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

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

Example Usage: sleep 10

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

Example Usage: busy

4.1.1.5 echo()

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

Echo back an input string.

Example Usage: echo Hello World

4.1.1.6 ls()

```
void * ls (
            void * arg)
```

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

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

4.1.1.7 touch()

```
void * touch (
            void * arg)
```

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

Example Usage: touch f1 f2 f3 f4 f5

4.1.1.8 mv()

```
void * mv (
            void * arg)
```

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

Print appropriate error message if:

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

Example Usage: mv src_file dst_file

4.1.1.9 cp()

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

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

Print appropriate error message if:

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

Example Usage: cp src_file dst_file

4.1.1.10 rm()

```
void * rm (
            void * arg)
```

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

Print appropriate error message if:

- `file` is not a file that exists

Example Usage: rm f1 f2 f3 f4 f5

4.1.1.11 str_to_perms()

```
uint8_t str_to_perms (
            char * str)
```

converts a string into permissions integer

Parameters

char∗	str string to convert

Returns

unit8_t permissions

4.1.1.12 chmod()

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

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

Print appropriate error message if:

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

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

4.2 builtins.h File Reference

builtin functions for the shell

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

Functions

- void print_system_error (const char *message)
  Helper function to print errors to stderr.
- int count_args (char **argv)
  Helper function to count number of arguments.
- void * cat (void *arg)
  The ususal `cat` program.
- void * busy (void *arg)
  Sleep for `n` seconds.
- void * echo (void *arg)
  Echo back an input string.
- void * ls (void *arg)
  Lists all files in the working directory. For extra credit, it should support relative and absolute file paths.
- void * touch (void *arg)
  For each file, create an empty file if it doesn't exist, else update its timestamp.
- void * mv (void *arg)
  Rename a file. If the `dst_file` file already exists, overwrite it.
- void * cp (void *arg)
- void * rm (void *arg)
  Remove a list of files. Treat each file in the list as a separate transaction. (i.e. if removing file1 fails, still attempt to remove file2, file3, etc.)
- void * chmod (void *arg)
  Change permissions of a file. There's no need to error if a permission being added already exists, or if a permission being removed is already not granted.
- uint8_t str_to_perms (char *str)
  converts a string into permissions integer

Variables

- pcb_t * current_process

4.2.1 Detailed Description

builtin functions for the shell

See also

builtins.c

4.2.2 Function Documentation

4.2.2.1 print_system_error()

```
void print_system_error (
            const char * message)
```

Helper function to print errors to stderr.

Parameters

message	- error message

4.2.2.2 count_args()

```
int count_args (
            char ** argv)
```

Helper function to count number of arguments.

Parameters

argv	- list of arguments

4.2.2.3 cat()

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

The ususal `cat` program.

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

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

4.2.2.4 busy()

```
void * busy (
            void * arg)
```

Sleep for `n` seconds.

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

Example Usage: sleep 10

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

Example Usage: busy

4.2.2.5 echo()

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

Echo back an input string.

Example Usage: echo Hello World

4.2.2.6 ls()

```
void * ls (
            void * arg)
```

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

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

4.2.2.7 touch()

```
void * touch (
            void * arg)
```

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

Example Usage: touch f1 f2 f3 f4 f5

4.2.2.8 mv()

```
void * mv (
            void * arg)
```

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

Print appropriate error message if:

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

Example Usage: mv src_file dst_file

4.2.2.9 cp()

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

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

Print appropriate error message if:

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

Example Usage: cp src_file dst_file

4.2.2.10 rm()

```
void * rm (
            void * arg)
```

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

Print appropriate error message if:

- `file` is not a file that exists

Example Usage: rm f1 f2 f3 f4 f5

4.2.2.11 chmod()

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

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

Print appropriate error message if:

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

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

4.2.2.12 str_to_perms()

```
uint8_t str_to_perms (
            char * str)
```

converts a string into permissions integer

Parameters

char∗	str string to convert

Returns

unit8_t permissions

4.2.3 Variable Documentation

4.2.3.1 current_process

```
pcb_t* current_process  [extern]
```

4.3 builtins.h

Go to the documentation of this file.

```
00001
00006
00007 #ifndef H_BUILTINS
00008 #define H_BUILTINS
00009
00010 #include "user.h"
00011
00012 extern pcb_t* current_process;
00013
00018 void print_system_error(const char* message);
00019
00024 int count_args(char** argv);
00025
00036 void* cat(void* arg);
00037
00046 // void* sleep(void* arg);
00047
00054 void* busy(void* arg);
00055
00061 void* echo(void* arg);
00062
00070 void* ls(void* arg);
00071
00078 void* touch(void* arg);
00079
00090 void* mv(void* arg);
00091
00102 void* cp(void* arg);
00103
00114 void* rm(void* arg);
00115
00130 void* chmod(void* arg);
00131
00138 uint8_t str_to_perms(char* str);
00139
00140 #endif
```

4.4 clock.c File Reference

```
#include "clock.h"
#include <signal.h>
#include <sys/time.h>
```

Functions

- void set_clock (sigset_t scheduler_mask, void(*tick_handler)(int))
  Set up clock ticks.
- void stop_clock ()
  Stop the clock.

4.4.1 Function Documentation

4.4.1.1 set_clock()

```
void set_clock (
            sigset_t scheduler_mask,
            void(* tick_handler )(int))
```

Set up clock ticks.

Parameters

scheduler_mask	The signal mask for the scheduler
tick_handler	The function to handle the clock tick

4.4.1.2 stop_clock()

```
void stop_clock ()
```

Stop the clock.

4.5 clock.h File Reference

```
#include <signal.h>
#include <sys/time.h>
```

Functions

- void set_clock (sigset_t scheduler_mask, void(*tick_handler)(int))
  Set up clock ticks.
- void stop_clock ()
  Stop the clock.

4.5.1 Function Documentation

4.5.1.1 set_clock()

```
void set_clock (
            sigset_t scheduler_mask,
            void(* tick_handler )(int))
```

Set up clock ticks.

Parameters

scheduler_mask	The signal mask for the scheduler
tick_handler	The function to handle the clock tick

4.5.1.2 stop_clock()

```
void stop_clock ()
```

Stop the clock.

4.6 clock.h

Go to the documentation of this file.

```
00001 #ifndef CLOCK_H
00002 #define CLOCK_H
00003
00004 #include <signal.h>
00005 #include <sys/time.h>
00006
00012 void set_clock(sigset_t scheduler_mask, void (*tick_handler)(int));
00013
00017 void stop_clock();
00018
00019 #endif // CLOCK_H
```

4.7 dir.c File Reference

source for directory manipulation

```
#include "dir.h"
#include <math.h>
#include <string.h>
#include "kernel_fs.h"
#include "util/macro.h"
```

Functions

- void print_error (const char *message)
- dir_t * parse_dir (int idx)

  scan a directory block and create a directory object
- void commit_dir (dir_t *dir)

  writes all data on the dir to the fs
- void free_dir (dir_t *dir)

  frees all data associated with directory
- dir_t * mkdir (dir_t *parent)

  create a directory
- int new_dirent (dir_t *parent, dirent_t **ptr_out)

  creates or repurpouses a directory entry in a directory
- int file_in_dir (const char *filename, dir_t *dir)

  scan working directory to find file by name

4.7.1 Detailed Description

source for directory manipulation

Author

ian lamont

See also

dir.h

4.7.2 Function Documentation

4.7.2.1 print_error()

```
void print_error (
            const char * message)  [extern]
```

4.7.2.2 parse_dir()

```
dir_t * parse_dir (
            int idx)
```

scan a directory block and create a directory object

Parameters

int	idx fat index of directory

Returns

dir_t directory

4.7.2.3 commit_dir()

```
void commit_dir (
            dir_t * dir)
```

writes all data on the dir to the fs

Parameters

*dir_t**	dir the directory to commit @BUG when 2 processes use fs changes arent directly mapped, need to fix later @BUG like whole system is wrong because of this

4.7.2.4 free_dir()

```
void free_dir (
            dir_t * dir)
```

frees all data associated with directory

Parameters

*dir_t**	dir directory to free

4.7.2.5 mkdir()

```
dir_t * mkdir (
            dir_t * parent)
```

create a directory

Parameters

*dir_t**	parent the parent of the directory

Returns

dir_t directory struct

4.7.2.6 new_dirent()

```
int new_dirent (
            dir_t * parent,
            dirent_t ** ptr_out)
```

creates or repurpouses a directory entry in a directory

Parameters

	*dir_t**	parent
out	*dirent_t**	a pointer to the new dirent

Returns

int index of new dirent

4.7.2.7 file_in_dir()

```
int file_in_dir (
            const char * filename,
            dir_t * dir)
```

scan working directory to find file by name

Parameters

const	char* filename string of file name

Returns

int -1 on failure, entry index on success

4.8 dir.h File Reference

directory entry header

```
#include <time.h>
#include "fs.h"
#include "util/Vec.h"
```

Data Structures

- struct dirent_t
  struct for directory entry
- struct dir_t
  struct for directory

Macros

- #define MAX_FILENAME_LEN 32

Functions

- int file_in_dir (const char *filename, dir_t *dir)
  scan working directory to find file by name
- dir_t * mkdir (dir_t *parent)
  create a directory
- dir_t * parse_dir (int idx)
  scan a directory block and create a directory object
- void commit_dir (dir_t *dir)
  writes all data on the dir to the fs
- int new_dirent (dir_t *parent, dirent_t **ptr_out)
  creates or repurpouses a directory entry in a directory
- void free_dir (dir_t *dir)
  frees all data associated with directory

4.8.1 Detailed Description

directory entry header

Author

ian lamont

See also

dir.c fs.h

4.8.2 Macro Definition Documentation

4.8.2.1 MAX_FILENAME_LEN

```
#define MAX_FILENAME_LEN 32
```

4.8.3 Function Documentation

4.8.3.1 file_in_dir()

```
int file_in_dir (
            const char * filename,
            dir_t * dir)
```

scan working directory to find file by name

Parameters

const	char* filename string of file name

Returns

int -1 on failure, entry index on success

4.8.3.2 mkdir()

```
dir_t * mkdir (
            dir_t * parent)
```

create a directory

Parameters

*dir_t**	parent the parent of the directory

Returns

dir_t directory struct

4.8.3.3 parse_dir()

```
dir_t * parse_dir (
            int idx)
```

scan a directory block and create a directory object

Parameters

int	idx fat index of directory

Returns

dir_t directory

4.8.3.4 commit_dir()

```
void commit_dir (
            dir_t * dir)
```

writes all data on the dir to the fs

Parameters

*dir_t**	dir the directory to commit
*dir_t**	dir the directory to commit @BUG when 2 processes use fs changes arent directly mapped, need to fix later @BUG like whole system is wrong because of this

4.8.3.5 new_dirent()

```
int new_dirent (
            dir_t * parent,
            dirent_t ** ptr_out)
```

creates or repurpouses a directory entry in a directory

Parameters

	*dir_t**	parent
out	*dirent_t**	a pointer to the new dirent

Returns

int index of new dirent

4.8.3.6 free_dir()

```
void free_dir (
            dir_t * dir)
```

frees all data associated with directory

Parameters

dir↩ _t*	dir directory to free

4.9 dir.h

Go to the documentation of this file.

```
00001
00007
00008 #ifndef DIR_HEADER
00009 #define DIR_HEADER
00010
00011 #ifndef MAX_FILENAME_LEN
00012 #define MAX_FILENAME_LEN 32
00013 #endif
00014
00015 #include <time.h>
00016 #include "fs.h"
00017 #include "util/Vec.h"
00018
00023 typedef struct {
00024   char name[MAX_FILENAME_LEN];
00029   uint32_t size;
00030   uint16_t first_block;
00031   uint8_t type;
00036   uint8_t perm;
00043   time_t mtime;
00044   char padding[16];
00045 } dirent_t;
00046
00051 typedef struct {
00052   uint16_t idx;
00053   Vec vec;
00054 } dir_t;
00055
00062 int file_in_dir(const char* filename, dir_t* dir);
00063
00070 dir_t* mkdir(dir_t* parent);
00071
00078 dir_t* parse_dir(int idx);
00079
00085 void commit_dir(dir_t* dir);
00086
00094 int new_dirent(dir_t* parent, dirent_t** ptr_out);
00095
00101 void free_dir(dir_t* dir);
00102
00109 static inline int dir_len(dir_t* dir) {
00110   return vec_len(&dir->vec);
00111 }
00112
00120 static inline dirent_t* dir_get(dir_t* dir, int idx) {
00121   return (dirent_t*)((dir->vec.data)[idx]);
00122 }
00123
00130 static inline dirent_t** dir_data(dir_t* dir) {
00131   return (dirent_t**)(dir->vec.data);
00132 }
00133
00134 #endif
```

4.10 fd.c File Reference

file descriptor source

```
#include "fd.h"
#include <string.h>
#include "kernel_fs.h"
#include "util/macro.h"
```

Functions

- void print_error (const char *message)
- int file_in_fdt (const char *filename, fdt_t *table)

  checks if a file is in a process' file descriptor table
- fdt_t * new_fdt (int pid)

  creates a new fdt
- void free_fdt (fdt_t *fdt)

  frees a fdt
- int new_fd (fdt_t *table)

  adds a new fd to the fdt
- fd_t * get_fd (fdt_t *table, int fd_index)

  Gets file descriptor object from file descriptor table.
- int close_fd (fdt_t *table, int fd_index)

  Closes the fd by nulling/zeroing out the fields.

4.10.1 Detailed Description

file descriptor source

Author

ashely francisco, ian lamont

See also

fd.c, kernel.h

4.10.2 Function Documentation

4.10.2.1 print_error()

```
void print_error (
            const char * message)  [extern]
```

4.10.2.2 file_in_fdt()

```
int file_in_fdt (
            const char * filename,
            fdt_t * table)
```

checks if a file is in a process' file descriptor table

Parameters

const	char* filename the name of the file to check for
*fd_table_t**	table the table to check

Returns

int index of the file in the fd table, -1 on failure

Note

will have to revist for multi directory (path not filename?)

4.10.2.3 new_fdt()

```
fdt_t * new_fdt (
            int pid)
```

creates a new fdt

Parameters

int	pid the process to associate this table with

Returns

fdt_t* fdt pointer

4.10.2.4 free_fdt()

```
void free_fdt (
            fdt_t * fdt)
```

frees a fdt

Parameters

fdt	to free

4.10.2.5 new_fd()

```
int new_fd (
            fdt_t * table)
```

adds a new fd to the fdt

Parameters

*fdt_t**	fdt to add to

Returns

int the new fd index in the fdt

4.10.2.6 get_fd()

```
fd_t * get_fd (
            fdt_t * table,
            int fd_index)
```

Gets file descriptor object from file descriptor table.

Parameters

fdt_t∗	fdt that holds the fd
fd_index	index of the fd

4.10.2.7 close_fd()

```
int close_fd (
            fdt_t * table,
            int fd_index)
```

Closes the fd by nulling/zeroing out the fields.

Parameters

fdt_t∗	fdt that holds the fd
fd_index	index of the fd

4.11 fd.h File Reference

file descriptor header

```
#include "util/Vec.h"
```

Data Structures

- struct fd_t
  file descriptor struct
- struct fdt_t
  file desctriptor table struct

Macros

- #define MAX_FILENAME_LEN 32

Functions

- int file_in_fdt (const char ∗filename, fdt_t ∗table)
  checks if a file is in a process' file descriptor table
- fdt_t ∗ new_fdt (int pid)
  creates a new fdt
- void free_fdt (fdt_t ∗fdt)
  frees a fdt
- int new_fd (fdt_t ∗table)
  adds a new fd to the fdt
- fd_t ∗ get_fd (fdt_t ∗table, int fd_index)
  Gets file descriptor object from file descriptor table.
- int close_fd (fdt_t ∗table, int fd_index)
  Closes the fd by nulling/zeroing out the fields.

4.11.1 Detailed Description

file descriptor header

Author

ashely francisco, ian lamont

See also

fd.c, kernel.h

4.11.2 Macro Definition Documentation

4.11.2.1 MAX_FILENAME_LEN

```
#define MAX_FILENAME_LEN 32
```

4.11.3 Function Documentation

4.11.3.1 file_in_fdt()

```
int file_in_fdt (
            const char * filename,
            fdt_t * table)
```

checks if a file is in a process' file descriptor table

Parameters

const	char* filename the name of the file to check for
*fd_table↩ _t**	table the table to check

Returns

int index of the file in the fd table, -1 on failure

Note

will have to revist for multi directory (path not filename?)

4.11.3.2 new_fdt()

```
fdt_t * new_fdt (
            int pid)
```

creates a new fdt

Parameters

int	pid the process to associate this table with

Returns

fdt_t* fdt pointer

4.11.3.3 free_fdt()

```
void free_fdt (
            fdt_t * fdt)
```

frees a fdt

Parameters

fdt	to free

4.11.3.4 new_fd()

```
int new_fd (
            fdt_t * table)
```

adds a new fd to the fdt

Parameters

*fdt_t**	fdt to add to

Returns

int the new fd index in the fdt

4.11.3.5 get_fd()

```
fd_t * get_fd (
            fdt_t * table,
            int fd_index)
```

Gets file descriptor object from file descriptor table.

Parameters

*fdt_t**	fdt that holds the fd
fd_index	index of the fd

4.11.3.6 close_fd()

```
int close_fd (
            fdt_t * table,
            int fd_index)
```

Closes the fd by nulling/zeroing out the fields.

Parameters

*fdt_t**	fdt that holds the fd
fd_index	index of the fd

4.12 fd.h

Go to the documentation of this file.

```
00001
00007
00008 #ifndef HEADER_FILEDESC
00009 #define HEADER_FILEDESC
00010
00011 #include "util/Vec.h"
00012
00013 #ifndef MAX_FILENAME_LEN
00014 #define MAX_FILENAME_LEN 32
00015 #endif
00016
00021 typedef struct {
00022   char filename[MAX_FILENAME_LEN];
00023   int first_block;
00024   int mode;
00025   int offset;
00026   int in_use;
00027   int stdin_out_err;
00028   int dir_idx;
00029   int ref_count;  // Used to keep track of active refs to open file
00030   int global_index;
00031 } fd_t;
00032
00037 typedef struct {
00038   int pid;
00039   Vec vec;  //< undelying table of file_desc_t*
00040 } fdt_t;
00041
00050 int file_in_fdt(const char* filename, fdt_t* table);
00051
00058 fdt_t* new_fdt(int pid);
00059
00065 void free_fdt(fdt_t* fdt);
00066
00073 int new_fd(fdt_t* table);
00074
00082 fd_t* get_fd(fdt_t* table, int fd_index);
00083
00090 int close_fd(fdt_t* table, int fd_index);
00091
00098 static inline int fdt_len(fdt_t* fdt) {
00099   return vec_len(&fdt->vec);
00100 }
00101
00108 static inline fd_t** fdt_data(fdt_t* fdt) {
00109   return (fd_t**)(fdt->vec.data);
00110 }
00111
00112 #endif
```

4.13 fs.c File Reference

source for make file system

```
#include "fs.h"
#include <fcntl.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <sys/mman.h>
```

```
#include <unistd.h>
#include "dir.h"
#include "util/macro.h"
```

Functions

- fs_t * new_fs (char *name, int num_blocks, int block_size_config)
  create a new file system object
- void free_fs (fs_t *fs)
  deletes and unmaps file system
- block_t * get_block (int idx)
  gets a block of data from the mounted fs
- void free_block (block_t *blk)
  frees data associated with a block
- int fat_next (int idx)
  get the next value of a fat entry
- uint16_t fat_alloc_block ()
  finds a free fat block and uses it
- void fat_free_block (int idx)
  clears and frees up fat block
- int fat_expand_file (uint16_t *first_block, int num_to_add)
  adds some number of new blocks to a file
- void fat_free_file (uint16_t *first_block)
  removes all blocks associated with a file in the fat

4.13.1 Detailed Description

source for make file system

Author

ian lamont

See also

fs.h

4.13.2 Function Documentation

4.13.2.1 new_fs()

```
fs_t * new_fs (
            char * name,
            int num_blocks,
            int block_size)
```

create a new file system object

Parameters

char∗	name name of the file system
int	blocks_in_fat number of blocks in the fat
int	block_size_config size of each block in the fat in config notation

4.13.2.2 free_fs()

```
void free_fs (
            fs_t * fs)
```

deletes and unmaps file system

Parameters

fs↩ _t∗	fs files system to delete

4.13.2.3 get_block()

```
block_t * get_block (
            int idx)
```

gets a block of data from the mounted fs

Parameters

int	idx index of the block to be retrieved

Returns

block_t∗ block that was retrieved

4.13.2.4 free_block()

```
void free_block (
            block_t * blk)
```

frees data associated with a block

Parameters

block↩ _t∗	blk block to free

4.13.2.5 fat_next()

```
int fat_next (
            int idx)
```

get the next value of a fat entry

Parameters

int	idx fat table index @retruns int next block index

4.13.2.6 fat_alloc_block()

```
uint16_t fat_alloc_block ()
```

finds a free fat block and uses it

Returns

uint16_t the index of the fat block, FREE_MARKER on failure

Note

Only used within this .c file

4.13.2.7 fat_free_block()

```
void fat_free_block (
            int idx)
```

clears and frees up fat block

Parameters

int	idx fat index of block to free

Note

Only used within this .c file

4.13.2.8 fat_expand_file()

```
int fat_expand_file (
            uint16_t * first_block,
            int num_to_add)
```

adds some number of new blocks to a file

Parameters

in,out	*uint16_t**	first_block first block of the file to add to, will update to new block if file started empty
	int	num_to_add number of blocks to add after the first block

Returns

-1 on failure, 0 on success

4.13.2.9 fat_free_file()

```
void fat_free_file (
            uint16_t * first_block)
```

removes all blocks associated with a file in the fat

Parameters

in,out	*uint16_t**	first block of the file to free

Note

cannot fail

4.14 fs.h File Reference

header for file system; covers fs, fat, and blocks

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

Data Structures

- struct fs_t
  structure for file system
- struct block_t
  block data type (basically tuple of idx and bytes)

Macros

- #define MAX_FS_NAME_LEN 100
- #define ROOT_IDX 1
- #define META_IDX 0
- #define LAST_MARKER 0xffff
- #define FREE_MARKER 0x0000

Functions

- fs_t * new_fs (char *name, int num_blocks, int block_size)
  create a new file system object
- void free_fs (fs_t *fs)
  deletes and unmaps file system
- int fat_next (int idx)
  get the next value of a fat entry
- int fat_expand_file (uint16_t *first_block, int num_to_add)
  adds some number of new blocks to a file
- void fat_free_file (uint16_t *first_block)
  removes all blocks associated with a file in the fat
- block_t * get_block (int idx)
  gets a block of data from the mounted fs
- void free_block (block_t *blk)
  frees data associated with a block

Variables

- fs_t * MOUNTED_FS
  global mounted file system

4.14.1 Detailed Description

header for file system; covers fs, fat, and blocks

Author

ian lamont

See also

fs.c

4.14.2 Macro Definition Documentation

4.14.2.1 MAX_FS_NAME_LEN

```
#define MAX_FS_NAME_LEN 100
```

4.14.2.2 ROOT_IDX

```
#define ROOT_IDX 1
```

4.14.2.3 META_IDX

```
#define META_IDX 0
```

4.14.2.4 LAST_MARKER

```
#define LAST_MARKER 0xffff
```

4.14.2.5 FREE_MARKER

```
#define FREE_MARKER 0x0000
```

4.14.3 Function Documentation

4.14.3.1 new_fs()

```
fs_t * new_fs (
            char * name,
            int num_blocks,
            int block_size)
```

create a new file system object

Parameters

char∗	name name of the file system
int	blocks_in_fat number of blocks in the fat
int	block_size_config size of each block in the fat in config notation

4.14.3.2 free_fs()

```
void free_fs (
            fs_t * fs)
```

deletes and unmaps file system

Parameters

fs↩ _t∗	fs files system to delete

4.14.3.3 fat_next()

```
int fat_next (
            int idx)
```

get the next value of a fat entry

Parameters

int	idx fat table index @retruns int next block index

4.14.3.4 fat_expand_file()

```
int fat_expand_file (
            uint16_t * first_block,
            int num_to_add)
```

adds some number of new blocks to a file

Parameters

in,out	*uint16↩ _t∗*	first_block first block of the file to add to, will update to new block if file started empty
	int	num_to_add number of blocks to add after the first block

Returns

-1 on failure, 0 on success

4.14.3.5 fat_free_file()

```
void fat_free_file (
            uint16_t * first_block)
```

removes all blocks associated with a file in the fat

Parameters

in,out	*uint16_t**	first block of the file to free

Note

cannot fail

4.14.3.6 get_block()

```
block_t * get_block (
            int idx)
```

gets a block of data from the mounted fs

Parameters

int	idx index of the block to be retrieved

Returns

block_t* block that was retrieved

4.14.3.7 free_block()

```
void free_block (
            block_t * blk)
```

frees data associated with a block

Parameters

*block_t**	blk block to free

4.14.4 Variable Documentation

4.14.4.1 MOUNTED_FS

```
fs_t* MOUNTED_FS  [extern]
```

global mounted file system

4.15 fs.h

Go to the documentation of this file.

```
00001
00007
00008 #ifndef FS_HEADER
00009 #define FS_HEADER
00010
00011 #include <stdint.h>
00012 #include <stdlib.h>
00013
00014 #define MAX_FS_NAME_LEN 100
00015 #define ROOT_IDX 1
00016 #define META_IDX 0
00017 #define LAST_MARKER 0xffff
00018 #define FREE_MARKER 0x0000
00019
00024 typedef struct {
00025   char* name;
00026   int num_entries;
00027   int block_size;
00028   int fat_size;
00029   int fd;
00030   uint16_t* fat;
00031 } fs_t;
00032
00037 typedef struct {
00038   int idx;
00039   char* data;
00040   void* mmap_ptr;
00041 } block_t;
00042
00043 extern fs_t* MOUNTED_FS;
00044
00052 fs_t* new_fs(char* name, int num_blocks, int block_size);
00053
00059 void free_fs(fs_t* fs);
00060
00067 int fat_next(int idx);
00068
00077 int fat_expand_file(uint16_t* first_block, int num_to_add);
00078
00085 void fat_free_file(uint16_t* first_block);
00086
00093 block_t* get_block(int idx);
00094
00100 void free_block(block_t* blk);
00101
00102 #endif
```

4.16 job.c File Reference

```
#include "job.h"
#include "system.h"
#include "pennos.h"
#include "queue.h"
#include "util/Vec.h"
```

Macros

- #define MAX_BUFFER_LEN 1024
- #define STDIN_FILENO 0
- #define STDOUT_FILENO 1

Functions

- void init_job_system (void)

  Initialize the job system start the sizes at 4 and vec stores int values of job_id.
- job_t * add_job (int job_id, const char *command, bool is_background)

  Add a new job to the jobs list.
- job_t * find_job_by_id (int job_id)

  Find a job by its job ID.
- job_t * find_job_by_pid (pid_t pid)

  Find a job by its process ID.
- int update_job_status (int job_id)

  Update job status and move it to the appropriate list.
- void cleanup_jobs (void)

  Clean up finished jobs from the finished_jobs list.
- void print_jobs (Vec *job_list, const char *status)

  Print jobs with a specific status.
- void print_job_processes (Vec *processes)

  Print the processes in a job.

Variables

- Vec job_table
- Vec running_jobs
- Vec stopped_jobs
- Vec finished_jobs
- int next_job_id = 1
- int current_job_id = 0
- int foreground_job_id = 0
- int most_recent_job_id = 0

4.16.1 Macro Definition Documentation

4.16.1.1 MAX_BUFFER_LEN

```
#define MAX_BUFFER_LEN 1024
```

4.16.1.2 STDIN_FILENO

```
#define STDIN_FILENO 0
```

4.16.1.3 STDOUT_FILENO

```
#define STDOUT_FILENO 1
```

4.16.2 Function Documentation

4.16.2.1 init_job_system()

```
void init_job_system (
            void )
```

Initialize the job system start the sizes at 4 and vec stores int values of job_id.

4.16.2.2 add_job()

```
job_t * add_job (
            int job_id,
            const char * command,
            bool is_background)
```

Add a new job to the jobs list.

Parameters

job_id	Job ID to assign to the new job
command	Command string that created the job
is_background	Whether the job is running in the background

Returns

job_t* Pointer to the new job or NULL if allocation failed

4.16.2.3 find_job_by_id()

```
job_t * find_job_by_id (
            int job_id)
```

Find a job by its job ID.

Parameters

job_id	Job ID to search for

Returns

job_t* Pointer to the job or NULL if not found

4.16.2.4 find_job_by_pid()

```
job_t * find_job_by_pid (
            pid_t pid)
```

Find a job by its process ID.

Parameters

pid	Process ID to search for

Returns

job_t* Pointer to the job or NULL if not found

4.16.2.5 update_job_status()

```
int update_job_status (
            int job_id)
```

Update job status and move it to the appropriate list.

Parameters

job_id	Job ID of the job
new_status	New status for the job

Returns

int 0 on success, -1 if job not found

4.16.2.6 cleanup_jobs()

```
void cleanup_jobs (
            void )
```

Clean up finished jobs from the finished_jobs list.

4.16.2.7 print_jobs()

```
void print_jobs (
            Vec * job_list,
            const char * status)
```

Print jobs with a specific status.

Parameters

job_list	Vector of job IDs
status	Status string ("Running", "Stopped", "Done")

4.16.2.8 print_job_processes()

```
void print_job_processes (
            Vec * processes)
```

Print the processes in a job.

Parameters

processes	Vector of processes

4.16.3 Variable Documentation

4.16.3.1 job_table

```
Vec job_table
```

4.16.3.2 running_jobs

```
Vec running_jobs
```

4.16.3.3 stopped_jobs

```
Vec stopped_jobs
```

4.16.3.4 finished_jobs

```
Vec finished_jobs
```

4.16.3.5 next_job_id

```
int next_job_id = 1
```

4.16.3.6 current_job_id

```
int current_job_id = 0
```

4.16.3.7 foreground_job_id

```
int foreground_job_id = 0
```

4.16.3.8 most_recent_job_id

```
int most_recent_job_id = 0
```

4.17 job.h File Reference

job declarations

```
#include <sys/types.h>
#include "util/Vec.h"
```

Data Structures

- struct job

Macros

- #define JOB_RUNNING 0
- #define JOB_STOPPED 1
- #define JOB_DONE 2

Typedefs

- typedef struct job job_t

Functions

- void print_error (const char *message)
- void init_job_system (void)

  Initialize the job system start the sizes at 4 and vec stores int values of job_id.
- job_t * add_job (int job_id, const char *command, bool is_background)

  Add a new job to the jobs list.
- job_t * find_job_by_id (int job_id)

  Find a job by its job ID.
- job_t * find_job_by_pid (pid_t pid)

  Find a job by its process ID.
- int update_job_status (int job_id)

  Update job status and move it to the appropriate list.
- void cleanup_jobs (void)

  Clean up finished jobs from the finished_jobs list.
- void print_jobs (Vec *job_list, const char *status)

  Print jobs with a specific status.
- void print_job_processes (Vec *processes)

  Print the processes in a job.

Variables

- Vec job_table
- Vec running_jobs
- Vec stopped_jobs
- Vec finished_jobs
- int next_job_id
- int current_job_id

4.17.1 Detailed Description

job declarations

See also

job.c

4.17.2 Macro Definition Documentation

4.17.2.1 JOB_RUNNING

```
#define JOB_RUNNING 0
```

4.17.2.2 JOB_STOPPED

```
#define JOB_STOPPED 1
```

4.17.2.3 JOB_DONE

```
#define JOB_DONE 2
```

4.17.3 Typedef Documentation

4.17.3.1 job_t

```
typedef struct job job_t
```

4.17.4 Function Documentation

4.17.4.1 print_error()

```
void print_error (
            const char * message)  [extern]
```

4.17.4.2 init_job_system()

```
void init_job_system (
            void )
```

Initialize the job system start the sizes at 4 and vec stores int values of job_id.

4.17.4.3 add_job()

```
job_t * add_job (
            int job_id,
            const char * command,
            bool is_background)
```

Add a new job to the jobs list.

Parameters

job_id	Job ID to assign to the new job
command	Command string that created the job
is_background	Whether the job is running in the background

Returns

job_t* Pointer to the new job or NULL if allocation failed

4.17.4.4 find_job_by_id()

```
job_t * find_job_by_id (
            int job_id)
```

Find a job by its job ID.

Parameters

job_id	Job ID to search for

Returns

job_t* Pointer to the job or NULL if not found

4.17.4.5 find_job_by_pid()

```
job_t * find_job_by_pid (
            pid_t pid)
```

Find a job by its process ID.

Parameters

pid	Process ID to search for

Returns

job_t* Pointer to the job or NULL if not found

4.17.4.6 update_job_status()

```
int update_job_status (
            int job_id)
```

Update job status and move it to the appropriate list.

Parameters

job_id	Job ID of the job
new_status	New status for the job

Returns

int 0 on success, -1 if job not found

4.17.4.7 cleanup_jobs()

```
void cleanup_jobs (
            void )
```

Clean up finished jobs from the finished_jobs list.

4.17.4.8 print_jobs()

```
void print_jobs (
            Vec * job_list,
            const char * status)
```

Print jobs with a specific status.

Parameters

job_list	Vector of job IDs
status	Status string ("Running", "Stopped", "Done")

4.17.4.9 print_job_processes()

```
void print_job_processes (
            Vec * processes)
```

Print the processes in a job.

Parameters

processes	Vector of processes

4.17.5 Variable Documentation

4.17.5.1 job_table

```
Vec job_table  [extern]
```

4.17.5.2 running_jobs

```
Vec running_jobs  [extern]
```

4.17.5.3 stopped_jobs

```
Vec stopped_jobs  [extern]
```

4.17.5.4 finished_jobs

```
Vec finished_jobs  [extern]
```

4.17.5.5 next_job_id

```
int next_job_id  [extern]
```

4.17.5.6 current_job_id

```
int current_job_id  [extern]
```

4.18 job.h

Go to the documentation of this file.

```
00001
00006
00007 #ifndef JOB_H
00008 #define JOB_H
00009
00010 #include <sys/types.h>
00011 #include "util/Vec.h"
00012
00013 extern void print_error(const char* message);
00014
00015 /* Job status constants */
00016 #define JOB_RUNNING 0
00017 #define JOB_STOPPED 1
00018 #define JOB_DONE 2
00019
00020 /* Job structure */
00021 typedef struct job {
00022     int job_id;             // Job ID will be incremented by 1 each time a job is created
00023     pid_t pid;              // Process ID of the job's main process
00024     Vec processes;          // Processes in the job
00025     char *command;          // Command string that created this job
00026     int status;             // Current job status (running, stopped, etc.)
00027     bool is_background;     // Whether the job was started as a background job
00028 } job_t;
00029
00030 /* Global job vectors */
00031 extern Vec job_table;       // Stores all jobs indexed by job_id
00032 extern Vec running_jobs;    // Stores job_ids of running jobs
00033 extern Vec stopped_jobs;    // Stores job_ids of stopped jobs
00034 extern Vec finished_jobs;   // Stores job_ids of finished jobs
00035 extern int next_job_id;     // Next job ID to assign
00036 extern int current_job_id;  // Current active job ID
00037
00038 /* Function declarations */
00039 void init_job_system(void);
00040 job_t* add_job(int job_id, const char *command, bool is_background);
00041 job_t* find_job_by_id(int job_id);
00042 job_t* find_job_by_pid(pid_t pid);
00043 int update_job_status(int job_id);
00044 void cleanup_jobs(void);
00045 void print_jobs(Vec *job_list, const char *status);
00046 void print_job_processes(Vec *processes);
00047
00048 #endif /* JOB_H */
```

4.19 kernel.c File Reference

```
#include "kernel.h"
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "logger.h"
#include "p_errno.h"
#include "scheduler.h"
#include "system.h"
```

Functions

- pcb_t * k_proc_create (pcb_t *parent, void *(*func)(void *), char *arg[ ], int nice_value)

  Create a new child process, inheriting applicable properties from the parent.
- void k_thread_cleanup (spthread_t thread)

  Clean up an spthread's resources.
- void k_proc_cleanup (pcb_t *proc)

  Clean up a terminated/finished thread's resources. This may include freeing the PCB, handling children, etc.
- void k_proc_block (int pid)

  Block a process.
- void k_proc_kill (int pid)

  Terminate a process.
- void k_proc_unblock (int pid)

  Unblock a process.
- pid_t k_init (void)

  Initialize the first process (init) in the system.
- pid_t k_shell (void)

  Initialize the shell process.
- int k_proc_info (Vec *processes)

  Populate a vector with information about all processes.
- void k_orphan_children (pcb_t *process)

  Orphan all children of a process.
- bool k_proc_has_children (pcb_t *process)

  Check if a process has any zombie or alive children.

Variables

- const int PRIORITY_HIGH
- const int PRIORITY_MEDIUM
- const int PRIORITY_LOW
- const int NUM_PRIORITY_LEVELS
- const int PROCESS_RUNNING
- const int PROCESS_BLOCKED
- const int PROCESS_ZOMBIE
- PriorityQueue process_queue
- pcb_t * current_process

4.19.1 Function Documentation

4.19.1.1 k_proc_create()

```
pcb_t * k_proc_create (
            pcb_t * parent,
            void *(* func )(void *),
            char * arg[],
            int nice_value)
```

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

Parameters

parent	The parent PCB or NULL for no parent.
func	Function to be executed by the process thread.
arg	Argument to pass to the thread function.
nice_value	Priority for the process

Returns

Reference to the child PCB.

4.19.1.2 k_thread_cleanup()

```
void k_thread_cleanup (
            spthread_t thread)
```

Clean up an spthread's resources.

Parameters

thread	The spthread to clean up

4.19.1.3 k_proc_cleanup()

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

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

Parameters

proc	The PCB to clean up

4.19.1.4 k_proc_block()

```
void k_proc_block (
            int pid)
```

Block a process.

Parameters

pid	The PID of the process to block

4.19.1.5 k_proc_kill()

```
void k_proc_kill (
            int pid)
```

Terminate a process.

Parameters

pid	The PID of the process to terminate

4.19.1.6 k_proc_unblock()

```
void k_proc_unblock (
            int pid)
```

Unblock a process.

Parameters

pid	The PID of the process to unblock

4.19.1.7 k_init()

```
pid_t k_init (
            void )
```

Initialize the first process (init) in the system.

This creates the init process (PID 1) which serves as the ancestor of all other processes in the system, similar to Unix's init.

Returns

pid_t The process ID of the init process (should be 1).

4.19.1.8 k_shell()

```
pid_t k_shell (
            void )
```

Initialize the shell process.

This creates the shell process (PID 2) which serves as the shell of the system.

Returns

pid_t The process ID of the shell process

4.19.1.9 k_proc_info()

```
int k_proc_info (
            Vec * processes)
```

Populate a vector with information about all processes.

Parameters

processes	Vector to store process information

Returns

int Number of processes retrieved, -1 on error

4.19.1.10 k_orphan_children()

```
void k_orphan_children (
            pcb_t * process)
```

Orphan all children of a process.

This includes both alive and zombie children.

Parameters

process	The parent process of the children to orphan

4.19.1.11 k_proc_has_children()

```
bool k_proc_has_children (
            pcb_t * process)
```

Check if a process has any zombie or alive children.

Parameters

process	The parent process to check

Returns

True if the process has any children, False if not

4.19.2 Variable Documentation

4.19.2.1 PRIORITY_HIGH

```
const int PRIORITY_HIGH  [extern]
```

4.19.2.2 PRIORITY_MEDIUM

```
const int PRIORITY_MEDIUM  [extern]
```

4.19.2.3 PRIORITY_LOW

```
const int PRIORITY_LOW  [extern]
```

4.19.2.4 NUM_PRIORITY_LEVELS

```
const int NUM_PRIORITY_LEVELS  [extern]
```

4.19.2.5 PROCESS_RUNNING

```
const int PROCESS_RUNNING  [extern]
```

4.19.2.6 PROCESS_BLOCKED

```
const int PROCESS_BLOCKED  [extern]
```

4.19.2.7 PROCESS_ZOMBIE

```
const int PROCESS_ZOMBIE  [extern]
```

4.19.2.8 process_queue

```
PriorityQueue process_queue  [extern]
```

4.19.2.9 current_process

```
pcb_t* current_process  [extern]
```

4.20 kernel.h File Reference

```
#include "./util/Vec.h"
#include "scheduler.h"
```

Data Structures

- struct proc_info_t

  Process information structure for process listing.

Functions

- pcb_t * k_proc_create (pcb_t *parent, void *(*func)(void *), char *arg[], int nice_value)

  Create a new child process, inheriting applicable properties from the parent.
- void k_thread_cleanup (spthread_t thread)

  Clean up an spthread's resources.
- void k_proc_cleanup (pcb_t *proc)

  Clean up a terminated/finished thread's resources. This may include freeing the PCB, handling children, etc.
- void k_proc_block (int pid)

  Block a process.
- void k_proc_kill (int pid)

  Terminate a process.
- void k_proc_unblock (int pid)

  Unblock a process.
- pid_t k_init (void)

  Initialize the first process (init) in the system.
- pid_t k_shell (void)

  Initialize the shell process.
- int k_proc_info (Vec *processes)

  Populate a vector with information about all processes.
- void k_orphan_children (pcb_t *process)

  Orphan all children of a process.
- bool k_proc_has_children (pcb_t *process)

  Check if a process has any zombie or alive children.

4.20.1 Function Documentation

4.20.1.1 k_proc_create()

```
pcb_t * k_proc_create (
            pcb_t * parent,
            void *(* func )(void *),
            char * arg[],
            int nice_value)
```

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

Parameters

parent	The parent PCB or NULL for no parent.
func	Function to be executed by the process thread.
arg	Argument to pass to the thread function.
nice_value	Priority for the process

Returns

Reference to the child PCB.

4.20.1.2 k_thread_cleanup()

```
void k_thread_cleanup (
            spthread_t thread)
```

Clean up an spthread's resources.

Parameters

thread	The spthread to clean up

4.20.1.3 k_proc_cleanup()

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

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

Parameters

proc	The PCB to clean up

4.20.1.4 k_proc_block()

```
void k_proc_block (
            int pid)
```

Block a process.

Parameters

pid	The PID of the process to block

4.20.1.5 k_proc_kill()

```
void k_proc_kill (
            int pid)
```

Terminate a process.

Parameters

pid	The PID of the process to terminate

4.20.1.6 k_proc_unblock()

```
void k_proc_unblock (
            int pid)
```

Unblock a process.

Parameters

pid	The PID of the process to unblock

4.20.1.7 k_init()

```
pid_t k_init (
            void )
```

Initialize the first process (init) in the system.

This creates the init process (PID 1) which serves as the ancestor of all other processes in the system, similar to Unix's init.

Returns

pid_t The process ID of the init process (should be 1).

4.20.1.8 k_shell()

```
pid_t k_shell (
            void )
```

Initialize the shell process.

This creates the shell process (PID 2) which serves as the shell of the system.

Returns

pid_t The process ID of the shell process

4.20.1.9 k_proc_info()

```
int k_proc_info (
            Vec * processes)
```

Populate a vector with information about all processes.

Parameters

processes	Vector to store process information

Returns

int Number of processes retrieved, -1 on error

4.20.1.10 k_orphan_children()

```
void k_orphan_children (
            pcb_t * process)
```

Orphan all children of a process.

This includes both alive and zombie children.

Parameters

process	The parent process of the children to orphan

4.20.1.11 k_proc_has_children()

```
bool k_proc_has_children (
            pcb_t * process)
```

Check if a process has any zombie or alive children.

Parameters

process	The parent process to check

Returns

True if the process has any children, False if not

4.21 kernel.h

Go to the documentation of this file.

```
00001 #ifndef KERNEL_H
00002 #define KERNEL_H
00003
00004 #include "./util/Vec.h"
00005 #include "scheduler.h"
00006
00010 typedef struct {
00011   pid_t pid;
00012   pid_t ppid;
00013   int priority;
00014   int state;
00015   char name[32];
00016 } proc_info_t;
00017
00027 pcb_t* k_proc_create(pcb_t* parent,
00028                     void* (*func)(void*),
00029                     char* arg[],
00030                     int nice_value);
00031
00036 void k_thread_cleanup(spthread_t thread);
00037
00043 void k_proc_cleanup(pcb_t* proc);
00044
00049 void k_proc_block(int pid);
00050
00055 void k_proc_kill(int pid);
00056
00061 void k_proc_unblock(int pid);
00062
00071 pid_t k_init(void);
00072
00081 pid_t k_shell(void);
00082
00089 int k_proc_info(Vec* processes);
00090
00098 void k_orphan_children(pcb_t* process);
00099
00107 bool k_proc_has_children(pcb_t* process);
00108
00109 #endif  // KERNEL
```

4.22 kernel_fs.c File Reference

kernel function defines for file system

```
#include "./kernel_fs.h"
#include <math.h>
#include <unistd.h>
#include "fs.h"
#include "util/macro.h"
```

Functions

- void print_error (const char *message)
  prints error message to stderr using k_write
- int check_file_name (const char *str)
  checks if a string can be a valid file name
- int file_in_use (int global_idx)
  checks if a file is open
- int truncate_file (dirent_t *file)
  truncates a file
- int k_open (const char *fname, int mode)
  opens a file in a given mode
- int manipulate_fd (fd_t *fd, int n, char *buf, int mode)
  manipulates data in a file
- int k_read (int fd, int n, char *buf)
  reads a set number of bytes from a file into a buffer
- int k_write (int fd, const char *str, int n)
  writes bytes to a file
- int k_close (int fd)
  closes a open file
- int k_unlink (const char *fname)
  remove the file from the fs
- int k_lseek (int fd, int offset, int whence)
  repositon offset of fd to the offset relative to whence
- int k_chmod (const char *fname, uint8_t new_perm)
  change the permissions on a file
- int k_ls (const char *filename)
  list data on files
- uint8_t k_get_perms (const char *fname)
  Get specified file's permissions.
- int k_rename_file (const char *old_name, const char *new_name)
  renames a file

4.22.1 Detailed Description

kernel function defines for file system

Author

ashely francisco, ian lamont

See also

kernel.h

4.22.2 Function Documentation

4.22.2.1 print_error()

```
void print_error (
            const char * message)
```

prints error message to stderr using k_write

Parameters

const	char* message

4.22.2.2 check_file_name()

```
int check_file_name (
            const char * str)
```

checks if a string can be a valid file name

Parameters

const	char* str string to check

Returns

-1 on failure, 0 on success

4.22.2.3 file_in_use()

```
int file_in_use (
            int idx)
```

checks if a file is open

Parameters

int	global_idx global fdt idx to check

Returns

1 on open 0 on not open, -1 on dne

4.22.2.4 truncate_file()

```
int truncate_file (
            dirent_t * file)
```

truncates a file

Parameters

int	dir_index the index of the file in a directory

Returns

-1 on failure, 0 on success

4.22.2.5 k_open()

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

opens a file in a given mode

Parameters

const	char* fname file name
int	mode mode of file open (F_READ, F_WRITE, or F_APPEND)

Returns

-1 on failure, index in file table on success

4.22.2.6 manipulate_fd()

```
int manipulate_fd (
            fd_t * fd,
            int n,
            char * buf,
            int mode)
```

manipulates data in a file

Parameters

*fd_t**	fd file to manipulate
int	n bytes to manipulate
*char**	buf buffer to store to or write from
mode	F_READ for read, F_WRITE for write

Returns

number of bytes manipulated

4.22.2.7 k_read()

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

reads a set number of bytes from a file into a buffer

Parameters

int	fd file descriptor number
int	n number of bytes to read
char∗	buf buffer to write into

Returns

int number of bytes read

4.22.2.8 k_write()

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

writes bytes to a file

Parameters

int	fd file descriptor to write to
const	char∗ str string of bytes to write (null terminated)
int	n number of bytes to write

Returns

bytes written on success, -1 on failure

4.22.2.9 k_close()

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

closes a open file

Parameters

int	fd file descriptor index

Returns

0 on succes, -1 on failure

4.22.2.10 k_unlink()

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

remove the file from the fs

Parameters

const	char* fname name of file to remove

Returns

0 on success, -1 on failure

4.22.2.11 k_lseek()

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

repositon offset of fd to the offset relative to whence

Parameters

int	fd fdt index
int	offset how much to displace pointer
int	whence where to displace pointer from

Returns

-1 on failure, 0 on success

4.22.2.12 k_chmod()

```
int k_chmod (
             const char * fname,
             uint8_t new_perm)
```

change the permissions on a file

Parameters

const	char* fname file name to modify
uint8_t	change to permissions

Returns

-1 on failure 0 on success

4.22.2.13 k_ls()

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

list data on files

Parameters

const	char* filename file to list display data on

Note

if filename is NULL, lists all files in working dir

Returns

int -1 on failure, 0 on success

4.22.2.14 k_get_perms()

```
uint8_t k_get_perms (
            const char * fname)
```

Get specified file's permissions.

Parameters

const	char* fname filename of interest

Returns

-1 on failure, return permission bits on success

4.22.2.15 k_rename_file()

```
int k_rename_file (
            const char * old_name,
            const char * new_name)
```

renames a file

Parameters

const	char* old_name old file name
const	char* new_name new file name

Returns

-1 on failure, 0 on success

4.23 kernel_fs.h File Reference

header for kernel fucntions relating to file system

```
#include <ctype.h>
#include <fcntl.h>
#include <signal.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "dir.h"
#include "fd.h"
#include "fs.h"
```

Macros

- #define MAX_FILE_DESCRIPTORS 1024
- #define MAX_DIRECTORIES 128
- #define F_READ 0
- #define F_WRITE 1
- #define F_APPEND 2
- #define END_MARKER 0xFFFF
- #define FAT_EOF END_MARKER
- #define FAT_FREE_BLOCK FREE_MARKER
- #define F_SEEK_SET 0
- #define F_SEEK_CUR 1
- #define F_SEEK_END 2
- #define K_STDIN 0
- #define K_STDOUT 1
- #define K_STDERR 2

Functions

- void print_error (const char *message)

  prints error message to stderr using k_write
- int check_file_name (const char *str)

  checks if a string can be a valid file name
- int file_in_use (int idx)

  checks if a file is open
- int truncate_file (dirent_t *file)

  truncates a file
- int manipulate_fd (fd_t *fd, int n, char *buf, int mode)

  manipulates data in a file
- int k_open (const char *fname, int mode)

  opens a file in a given mode
- int k_read (int fd, int n, char *buf)

  reads a set number of bytes from a file into a buffer
- int k_write (int fd, const char *str, int n)

writes bytes to a file

- int k_close (int fd)

  closes a open file
- int k_unlink (const char *fname)

  remove the file from the fs
- int k_lseek (int fd, int offset, int whence)

  repositon offset of fd to the offset relative to whence
- int k_chmod (const char *fname, uint8_t new_perm)

  change the permissions on a file
- int k_ls (const char *filename)

  list data on files
- int k_rename_file (const char *old_name, const char *new_name)

  renames a file
- uint8_t k_get_perms (const char *fname)

  Get specified file's permissions.

Variables

- fs_t * MOUNTED_FS

  global mounted file system
- dir_t * WORKING_DIR

  global current working directory
- fdt_t * GLOBAL_FDT

  global file descriptor table

4.23.1 Detailed Description

header for kernel fucntions relating to file system

Author

ashley francisco, ian lamont

See also

kernel.c dir.h

4.23.2 Macro Definition Documentation

4.23.2.1 MAX_FILE_DESCRIPTORS

```
#define MAX_FILE_DESCRIPTORS 1024
```

4.23.2.2 MAX_DIRECTORIES

```
#define MAX_DIRECTORIES 128
```

4.23.2.3 F_READ

```
#define F_READ 0
```

4.23.2.4 F_WRITE

```
#define F_WRITE 1
```

4.23.2.5 F_APPEND

```
#define F_APPEND 2
```

4.23.2.6 END_MARKER

```
#define END_MARKER 0xFFFF
```

4.23.2.7 FAT_EOF

```
#define FAT_EOF END_MARKER
```

4.23.2.8 FAT_FREE_BLOCK

```
#define FAT_FREE_BLOCK FREE_MARKER
```

4.23.2.9 F_SEEK_SET

```
#define F_SEEK_SET 0
```

4.23.2.10 F_SEEK_CUR

```
#define F_SEEK_CUR 1
```

4.23.2.11 F_SEEK_END

```
#define F_SEEK_END 2
```

4.23.2.12 K_STDIN

```
#define K_STDIN 0
```

4.23.2.13 K_STDOUT

```
#define K_STDOUT 1
```

4.23.2.14 K_STDERR

```
#define K_STDERR 2
```

4.23.3 Function Documentation

4.23.3.1 print_error()

```
void print_error (
            const char * message)
```

prints error message to stderr using k_write

Parameters

const	char* message

4.23.3.2 check_file_name()

```
int check_file_name (
            const char * str)
```

checks if a string can be a valid file name

Parameters

const	char* str string to check

Returns

-1 on failure, 0 on success

4.23.3.3 file_in_use()

```
int file_in_use (
            int idx)
```

checks if a file is open

Parameters

int	global_idx global fdt idx to check

Returns

1 on open 0 on not open, -1 on dne

4.23.3.4 truncate_file()

```
int truncate_file (
            dirent_t * file)
```

truncates a file

Parameters

int	dir_index the index of the file in a directory

Returns

-1 on failure, 0 on success

4.23.3.5 manipulate_fd()

```
int manipulate_fd (
            fd_t * fd,
            int n,
            char * buf,
            int mode)
```

manipulates data in a file

Parameters

fd_t∗	fd file to manipulate
int	n bytes to manipulate
char∗	buf buffer to store to or write from
mode	F_READ for read, F_WRITE for write

Returns

number of bytes manipulated

4.23.3.6 k_open()

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

opens a file in a given mode

Parameters

const	char∗ fname file name
int	mode mode of file open (F_READ, F_WRITE, or F_APPEND)

Returns

-1 on failure, index in file table on success

4.23.3.7 k_read()

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

reads a set number of bytes from a file into a buffer

Parameters

int	fd file descriptor number
int	n number of bytes to read
char∗	buf buffer to write into

Returns

int number of bytes read

4.23.3.8 k_write()

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

writes bytes to a file

Parameters

int	fd file descriptor to write to
const	char∗ str string of bytes to write (null terminated)
int	n number of bytes to write

Returns

bytes written on success, -1 on failure

4.23.3.9 k_close()

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

closes a open file

Parameters

int	fd file descriptor index

Returns

0 on succes, -1 on failure

4.23.3.10 k_unlink()

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

remove the file from the fs

Parameters

const	char* fname name of file to remove

Returns

0 on success, -1 on failure

4.23.3.11 k_lseek()

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

repositon offset of fd to the offset relative to whence

Parameters

int	fd fdt index
int	offset how much to displace pointer
int	whence where to displace pointer from

Returns

-1 on failure, 0 on success

4.23.3.12 k_chmod()

```
int k_chmod (
             const char * fname,
             uint8_t new_perm)
```

change the permissions on a file

Parameters

const	char* fname file name to modify
uint8_t	change to permissions

Returns

-1 on failure 0 on success

4.23.3.13 k_ls()

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

list data on files

Parameters

const	char* filename file to list display data on

Note

if filename is NULL, lists all files in working dir

Returns

int -1 on failure, 0 on success

4.23.3.14 k_rename_file()

```
int k_rename_file (
            const char * old_name,
            const char * new_name)
```

renames a file

Parameters

const	char* old_name old file name
const	char* new_name new file name

Returns

-1 on failure, 0 on success

4.23.3.15 k_get_perms()

```
uint8_t k_get_perms (
            const char * fname)
```

Get specified file's permissions.

Parameters

const	char* fname filename of interest

Returns

-1 on failure, return permission bits on success

4.23.4 Variable Documentation

4.23.4.1 MOUNTED_FS

```
fs_t* MOUNTED_FS  [extern]
```

global mounted file system

4.23.4.2 WORKING_DIR

```
dir_t* WORKING_DIR  [extern]
```

global current working directory

4.23.4.3 GLOBAL_FDT

```
fdt_t* GLOBAL_FDT  [extern]
```

global file descriptor table

4.24 kernel_fs.h

Go to the documentation of this file.

```
00001
00007
00008 #include <ctype.h>
00009 #include <fcntl.h>
00010 #include <signal.h>
00011 #include <stdbool.h>
00012 #include <stdint.h>
00013 #include <stdio.h>
00014 #include <stdlib.h>
00015 #include <string.h>
00016 #include <time.h>
00017 #include "dir.h"
00018 #include "fd.h"
00019 #include "fs.h"
00020
00021 #define MAX_FILE_DESCRIPTORS 1024
00022 #define MAX_DIRECTORIES 128
00023 #define F_READ 0
00024 #define F_WRITE 1
00025 #define F_APPEND 2
00026 #define END_MARKER 0xFFFF
00027 #define FAT_EOF END_MARKER
00028 #define FAT_FREE_BLOCK FREE_MARKER
00029 #define F_SEEK_SET 0
00030 #define F_SEEK_CUR 1
00031 #define F_SEEK_END 2
00032 #define K_STDIN 0
00033 #define K_STDOUT 1
00034 #define K_STDERR 2
00035
00036 extern fs_t* MOUNTED_FS;
00037 extern dir_t* WORKING_DIR;
00038 extern fdt_t* GLOBAL_FDT;
00039
00045 void print_error(const char* message);
00046
00053 int check_file_name(const char* str);
00054
00061 int file_in_use(int idx);
00062
00069 int truncate_file(dirent_t* file);
00070
00080 int manipulate_fd(fd_t* fd, int n, char* buf, int mode);
00081
00089 int k_open(const char* fname, int mode);
00090
00099 int k_read(int fd, int n, char* buf);
00100
00109 int k_write(int fd, const char* str, int n);
00110
00117 int k_close(int fd);
00118
00125 int k_unlink(const char* fname);
00126
00135 int k_lseek(int fd, int offset, int whence);
00136
00144 int k_chmod(const char* fname, uint8_t new_perm);
00145
00153 int k_ls(const char* filename);
00154
00162 int k_rename_file(const char* old_name, const char* new_name);
00163
00170 uint8_t k_get_perms(const char* fname);
```

4.25 logger.c File Reference

```
#include "logger.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
```

Functions

- void logger_init ()

  Initialize the logger.
- void logger_cleanup ()

  Clean up logger resources.
- void log_schedule (int ticks, int pid, int queue, const char *process_name)

  Log the scheduling of a process.
- void log_create (int ticks, int pid, int nice_value, const char *process_name)

  Log the creation of a process.
- void log_signaled (int ticks, int pid, int nice_value, const char *process_name)

  Log the signaling of a process.
- void log_exited (int ticks, int pid, int nice_value, const char *process_name)

  Log the exiting of a process.
- void log_zombie (int ticks, int pid, int nice_value, const char *process_name)

  Log the zombification of a process.
- void log_orphan (int ticks, int pid, int nice_value, const char *process_name)

  Log the orphanifying of a process.
- void log_waited (int ticks, int pid, int nice_value, const char *process_name, const char *parent_name)

  Log the waiting of a process.
- void log_nice (int ticks, int pid, int old_nice, int new_nice, const char *process_name)

  Log the nice value adjustment of a process.
- void log_blocked (int ticks, int pid, int nice_value, const char *process_name)

  Log the blocking of a process.
- void log_unblocked (int ticks, int pid, int nice_value, const char *process_name)

  Log the unblocking of a process.
- void log_stopped (int ticks, int pid, int nice_value, const char *process_name)

  Log the stopping of a process.
- void log_continued (int ticks, int pid, int nice_value, const char *process_name)

  Log the continuing of a process.
- void log_message (const char *message)

  Log a message (for internal use)
- void log_timestamp (const char *message)

  Log a message with a timestamp (for internal use)

Variables

- int ticks

4.25.1 Function Documentation

4.25.1.1 logger_init()

```
void logger_init ()
```

Initialize the logger.

4.25.1.2 logger_cleanup()

```
void logger_cleanup ()
```

Clean up logger resources.

4.25.1.3 log_schedule()

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

Log the scheduling of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
queue	The queue of the process
process_name	The name of the process

4.25.1.4 log_create()

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

Log the creation of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.5 log_signaled()

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

Log the signaling of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.6 log_exited()

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

Log the exiting of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.7 log_zombie()

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

Log the zombification of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.8 log_orphan()

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

Log the orphanifying of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.9 log_waited()

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

Log the waiting of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.10 log_nice()

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

Log the nice value adjustment of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
old_nice	The old nice value of the process
new_nice	The new nice value of the process
process_name	The name of the process

4.25.1.11 log_blocked()

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

Log the blocking of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.12 log_unblocked()

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

Log the unblocking of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.13 log_stopped()

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

Log the stopping of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.14 log_continued()

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

Log the continuing of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.25.1.15 log_message()

```
void log_message (
            const char * message)
```

Log a message (for internal use)

Parameters

message	The message to log

4.25.1.16 log_timestamp()

```
void log_timestamp (
            const char * message)
```

Log a message with a timestamp (for internal use)

Parameters

message	The message to log

4.25.2 Variable Documentation

4.25.2.1 ticks

```
int ticks  [extern]
```

4.26 logger.h File Reference

```
#include <stdio.h>
#include <stdbool.h>
#include "scheduler.h"
```

Functions

- void logger_init ()

  Initialize the logger.
- void logger_cleanup ()

  Clean up logger resources.
- void log_schedule (int tick_count, int pid, int queue, const char *process_name)

  Log the scheduling of a process.
- void log_create (int tick_count, int pid, int nice_value, const char *process_name)

  Log the creation of a process.
- void log_signaled (int tick_count, int pid, int nice_value, const char *process_name)

  Log the signaling of a process.
- void log_exited (int tick_count, int pid, int nice_value, const char *process_name)

  Log the exiting of a process.
- void log_zombie (int tick_count, int pid, int nice_value, const char *process_name)

  Log the zombification of a process.
- void log_orphan (int tick_count, int pid, int nice_value, const char *process_name)

  Log the orphanifying of a process.
- void log_waited (int tick_count, int pid, int nice_value, const char *process_name, const char *parent_name)

  Log the waiting of a process.
- void log_nice (int tick_count, int pid, int old_nice, int new_nice, const char *process_name)

  Log the nice value adjustment of a process.
- void log_blocked (int tick_count, int pid, int nice_value, const char *process_name)

  Log the blocking of a process.
- void log_unblocked (int tick_count, int pid, int nice_value, const char *process_name)

  Log the unblocking of a process.
- void log_stopped (int tick_count, int pid, int nice_value, const char *process_name)

  Log the stopping of a process.
- void log_continued (int tick_count, int pid, int nice_value, const char *process_name)

  Log the continuing of a process.
- void log_message (const char *message)

  Log a message (for internal use)
- void log_timestamp (const char *message)

  Log a message with a timestamp (for internal use)

4.26.1 Function Documentation

4.26.1.1 logger_init()

```
void logger_init ()
```

Initialize the logger.

4.26.1.2 logger_cleanup()

```
void logger_cleanup ()
```

Clean up logger resources.

4.26.1.3 log_schedule()

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

Log the scheduling of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
queue	The queue of the process
process_name	The name of the process

4.26.1.4 log_create()

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

Log the creation of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.5 log_signaled()

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

Log the signaling of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.6 log_exited()

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

Log the exiting of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.7 log_zombie()

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

Log the zombification of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.8 log_orphan()

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

Log the orphanifying of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.9 log_waited()

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

Log the waiting of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.10 log_nice()

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

Log the nice value adjustment of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
old_nice	The old nice value of the process
new_nice	The new nice value of the process
process_name	The name of the process

4.26.1.11 log_blocked()

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

Log the blocking of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.12 log_unblocked()

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

Log the unblocking of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.13 log_stopped()

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

Log the stopping of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.14 log_continued()

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

Log the continuing of a process.

Parameters

tick_count	The current tick count
pid	The PID of the process
nice_value	The nice value of the process
process_name	The name of the process

4.26.1.15 log_message()

```
void log_message (
            const char * message)
```

Log a message (for internal use)

Parameters

message	The message to log

4.26.1.16 log_timestamp()

```
void log_timestamp (
            const char * message)
```

Log a message with a timestamp (for internal use)

Parameters

message	The message to log

4.27 logger.h

Go to the documentation of this file.

```
00001 #ifndef LOGGER_H
00002 #define LOGGER_H
00003
00004 #include <stdio.h>
00005 #include <stdbool.h>
00006 #include "scheduler.h"
00007
00011 void logger_init();
00012
00016 void logger_cleanup();
00017
00025 void log_schedule(int tick_count, int pid, int queue, const char *process_name);
00026
00034 void log_create(int tick_count, int pid, int nice_value, const char *process_name);
00035
00043 void log_signaled(int tick_count, int pid, int nice_value, const char *process_name);
00044
00052 void log_exited(int tick_count, int pid, int nice_value, const char *process_name);
00053
00061 void log_zombie(int tick_count, int pid, int nice_value, const char *process_name);
00062
00070 void log_orphan(int tick_count, int pid, int nice_value, const char *process_name);
00071
00079 void log_waited(int tick_count, int pid, int nice_value, const char *process_name, const char
      *parent_name);
00080
00089 void log_nice(int tick_count, int pid, int old_nice, int new_nice, const char *process_name);
00090
00098 void log_blocked(int tick_count, int pid, int nice_value, const char *process_name);
00099
00107 void log_unblocked(int tick_count, int pid, int nice_value, const char *process_name);
00108
00116 void log_stopped(int tick_count, int pid, int nice_value, const char *process_name);
00117
00125 void log_continued(int tick_count, int pid, int nice_value, const char *process_name);
00126
00131 void log_message(const char *message);
00132
00137 void log_timestamp(const char *message);
00138
00139 #endif // LOGGER_H
```

4.28 p_errno.c File Reference

```
#include "p_errno.h"
#include "system.h"
#include <stdio.h>
#include <string.h>
```

Macros

- #define MAX_BUFFER_LEN 1024
- #define STDIN_FILENO 0
- #define STDOUT_FILENO 1

Functions

- const char * p_strerror (int errnum)
  Get string description of error number.
- void u_perror (const char *s)
  Print error message based on P_ERRNO.

Variables

- int P_ERRNO = 0
  Global error number.

4.28.1 Macro Definition Documentation

4.28.1.1 MAX_BUFFER_LEN

```
#define MAX_BUFFER_LEN 1024
```

4.28.1.2 STDIN_FILENO

```
#define STDIN_FILENO 0
```

4.28.1.3 STDOUT_FILENO

```
#define STDOUT_FILENO 1
```

4.28.2 Function Documentation

4.28.2.1 p_strerror()

```
const char * p_strerror (
            int errnum)
```

Get string description of error number.

Parameters

errnum	Error number

Returns

Pointer to error message string

4.28.2.2 u_perror()

```
void u_perror (
            const char * s)
```

Print error message based on P_ERRNO.

Parameters

s	Prefix string for error message

4.28.3 Variable Documentation

4.28.3.1 P_ERRNO

```
int P_ERRNO = 0
```

Global error number.

Global error number set by PennOS system calls.

4.29 p_errno.h File Reference

Macros

- #define P_EINVAL 1 /* Invalid argument */
- #define P_ESRCH 2 /* No such process */
- #define P_ENOMEM 3 /* Out of memory */
- #define P_EAGAIN 4 /* Resource temporarily unavailable */
- #define P_ECHILD 6 /* No child processes */
- #define P_ENOSPC 7 /* No space left on device */
- #define P_EINTR 8 /* Interrupted function call */

Functions

- const char * p_strerror (int errnum)

  Get string description of error number.

Variables

- int P_ERRNO

  Global error number set by PennOS system calls.

4.29.1 Macro Definition Documentation

4.29.1.1 P_EINVAL

```
#define P_EINVAL 1 /* Invalid argument */
```

4.29.1.2 P_ESRCH

```
#define P_ESRCH 2 /* No such process */
```

4.29.1.3 P_ENOMEM

```
#define P_ENOMEM 3 /* Out of memory */
```

4.29.1.4 P_EAGAIN

```
#define P_EAGAIN 4 /* Resource temporarily unavailable */
```

4.29.1.5 P_ECHILD

```
#define P_ECHILD 6 /* No child processes */
```

4.29.1.6 P_ENOSPC

```
#define P_ENOSPC 7 /* No space left on device */
```

4.29.1.7 P_EINTR

```
#define P_EINTR 8 /* Interrupted function call */
```

4.29.2 Function Documentation

4.29.2.1 p_strerror()

```
const char * p_strerror (
            int errnum)  [extern]
```

Get string description of error number.

Parameters

errnum	Error number

Returns

Pointer to error message string

4.29.3 Variable Documentation

4.29.3.1 P_ERRNO

```
int P_ERRNO  [extern]
```

Global error number set by PennOS system calls.

Global error number set by PennOS system calls.

4.30 p_errno.h

Go to the documentation of this file.

```
00001 #ifndef P_ERRNO_H
00002 #define P_ERRNO_H
00003
00007 extern int P_ERRNO;
00008
00009 #define P_EINVAL    1      /* Invalid argument */
00010 #define P_ESRCH     2      /* No such process */
00011 #define P_ENOMEM    3      /* Out of memory */
00012 #define P_EAGAIN    4      /* Resource temporarily unavailable */
00013 #define P_ECHILD    6      /* No child processes */
00014 #define P_ENOSPC    7      /* No space left on device */
00015 #define P_EINTR     8      /* Interrupted function call */
00016
00022 extern const char* p_strerror(int errnum);
00023
00024 #endif /* P_ERRNO_H */
```

4.31 pennfat.c File Reference

entrypoint for pennfat standalone shell

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

Functions

- void write_newline ()
  Function to write newline to stderr.
- void sigint_handler3 (int signo)
  signal handler to exit command and reprompt when Ctrl-C
- void sigtstp_handler (int signo)
  signal handler to reprompt when Ctrl-Z
- char ** string_to_argv (char *input, int *argc_out)
  helper to convert parser command to argv style
- void free_argv (int argc, char **argv)
  frees data of an argv
- int start_env ()
  starts the global enviornment on a new fs
- void stop_env ()
  closes and frees environment
- int main (int argc, char *argv[ ])
  pennfat shell

Variables

- fs_t * MOUNTED_FS = NULL
  global mounted file system
- fdt_t * GLOBAL_FDT = NULL
  global file descriptor table
- dir_t * WORKING_DIR = NULL
  global current working directory
- int ENV = 0
  global flag for intialized environment
- volatile sig_atomic_t interrupted = 0

4.31.1 Detailed Description

entrypoint for pennfat standalone shell

Author

ashely francisco, ian lamont

See also

fs.h, kernel.h, routines.h

4.31.2 Function Documentation

4.31.2.1 write_newline()

```
void write_newline ()
```

Function to write newline to stderr.

4.31.2.2 sigint_handler3()

```
void sigint_handler3 (
            int signo)
```

signal handler to exit command and reprompt when Ctrl-C

Parameters

signo	signal number

4.31.2.3 sigtstp_handler()

```
void sigtstp_handler (
            int signo)
```

signal handler to reprompt when Ctrl-Z

Parameters

signo	signal number

4.31.2.4 string_to_argv()

```
char ** string_to_argv (
            char * input,
            int * argc_out)
```

helper to convert parser command to argv style

Parameters

char∗∗	cmd[] command
int	argc number of commands

Returns

char∗∗ argv

4.31.2.5 free_argv()

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

frees data of an argv

Parameters

char∗∗	argv argv to free

4.31.2.6 start_env()

```
int start_env ()
```

starts the global enviornment on a new fs

Returns

-1 on failure 0 on success

4.31.2.7 stop_env()

```
void stop_env ()
```

closes and frees environment

4.31.2.8 main()

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

pennfat shell

Note

usage: pennfat

4.31.3 Variable Documentation

4.31.3.1 MOUNTED_FS

```
fs_t* MOUNTED_FS = NULL
```

global mounted file system

4.31.3.2 GLOBAL_FDT

```
fdt_t* GLOBAL_FDT = NULL
```

global file descriptor table

4.31.3.3 WORKING_DIR

```
dir_t* WORKING_DIR = NULL
```

global current working directory

4.31.3.4 ENV

```
int ENV = 0
```

global flag for intialized environment

4.31.3.5 interrupted

```
volatile sig_atomic_t interrupted = 0
```

4.32 pennfat.h File Reference

header for pennfat standalone entry

```
#include <signal.h>
#include "dir.h"
#include "kernel_fs.h"
#include "routines.h"
#include "util/parser.h"
```

Macros

- #define PROMPT "pennfat# "

Functions

- void print_error (const char *message)
  prints error message to stderr using k_write
- void write_newline ()
  Function to write newline to stderr.
- void sigint_handler3 (int signo)
  signal handler to exit command and reprompt when Ctrl-C
- void sigtstp_handler (int signo)
  signal handler to reprompt when Ctrl-Z
- char ** string_to_argv (char *input, int *argc_out)
  helper to convert parser command to argv style
- void free_argv (int argc, char **argv)
  frees data of an argv
- int start_env ()
  starts the global enviornment on a new fs
- void stop_env ()
  closes and frees environment

4.32.1 Detailed Description

header for pennfat standalone entry

See also

pennfat.c

4.32.2 Macro Definition Documentation

4.32.2.1 PROMPT

```
#define PROMPT "pennfat# "
```

4.32.3 Function Documentation

4.32.3.1 print_error()

```
void print_error (
            const char * message)  [extern]
```

prints error message to stderr using k_write

Parameters

const	char* message

4.32.3.2 write_newline()

```
void write_newline ()
```

Function to write newline to stderr.

4.32.3.3 sigint_handler3()

```
void sigint_handler3 (
            int signo)
```

signal handler to exit command and reprompt when Ctrl-C

Parameters

signo	signal number

4.32.3.4 sigtstp_handler()

```
void sigtstp_handler (
            int signo)
```

signal handler to reprompt when Ctrl-Z

Parameters

signo	signal number

4.32.3.5 string_to_argv()

```
char ** string_to_argv (
            char * input,
            int * argc_out)
```

helper to convert parser command to argv style

Parameters

char∗∗	cmd[] command
int	argc number of commands

Returns

char∗∗ argv

4.32.3.6 free_argv()

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

frees data of an argv

Parameters

char∗∗	argv argv to free

4.32.3.7 start_env()

```
int start_env ()
```

starts the global enviornment on a new fs

Returns

-1 on failure 0 on success

4.32.3.8 stop_env()

```
void stop_env ()
```

closes and frees environment

4.33 pennfat.h

Go to the documentation of this file.

```
00001
00006
00007 #ifndef H_PENNFAT
00008 #define H_PENNFAT
00009
00010 #include <signal.h>
00011 #include "dir.h"
00012 #include "kernel_fs.h"
00013 #include "routines.h"
00014 #include "util/parser.h"
00015
00016 #ifndef PROMPT
00017 #define PROMPT "pennfat# "
00018 #endif
00019
00020 extern void print_error(const char* message);
00021
00026 void write_newline();
00027
00033 void sigint_handler3(int signo);
00034
00040 void sigtstp_handler(int signo);
00041
00049 char** string_to_argv(char* input, int* argc_out);
00050
00056 void free_argv(int argc, char** argv);
00057
00063 int start_env();
00064
00069 void stop_env();
00070
00071 #endif // H_PENNFAT
```

4.34 pennos.c File Reference

```
#include "pennos.h"
#include <errno.h>
#include "fd.h"
#include "fs.h"
#include "job.h"
#include "kernel_fs.h"
#include "builtins.h"
#include "logger.h"
#include "p_errno.h"
#include "stress.h"
#include "system.h"
#include "user.h"
#include "util/macro.h"
#include "util/parser.h"
```

Macros

- #define MAX_CMD_LEN 1024
- #define MAX_BUFFER_LEN 1024
- #define MAX_ARGS 32

Functions

- builtin_t * lookup_builtin (const char *name)

  Lookup a builtin command.
- int in_subroutine (const char *name)

  Check if a command is a subroutine.
- int unmount (int argc, char *argv[])

  command to unmount active file system
- int mount (int argc, char *argv[])

  command to mount a file system
- int start_env ()

  starts the global enviornment on a new fs
- void stop_env ()

  closes and frees environment
- subroutine_t * lookup_subroutine (const char *name)

  Lookup a subroutine.
- void jobs ()

  Display all jobs in the system.
- void man ()

  Lists all available commands.
- void logout ()

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

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

  Resumes the most recently stopped job in the background, or the job specified by job_id.
- int execute_command (struct parsed_command *pcmd)

  Execute a command.
- void * shell (void *arg)

  The main shell process function implementation.
- int main (int argc, char *argv[])

Variables

- int next_job_id
- int current_job_id
- int foreground_job_id
- int most_recent_job_id
- bool running
- fs_t * MOUNTED_FS = NULL

  global mounted file system
- fdt_t * GLOBAL_FDT = NULL

  global file descriptor table
- dir_t * WORKING_DIR = NULL

  global current working directory
- int ENV = 0

  global flag for intialized environment
- volatile sig_atomic_t interrupted = 0
- int stdin_fd = 0
- int stdout_fd = 1
- builtin_t builtin_table [ ]
- subroutine_t subroutine_table [ ]

4.34.1 Macro Definition Documentation

4.34.1.1 MAX_CMD_LEN

```
#define MAX_CMD_LEN 1024
```

4.34.1.2 MAX_BUFFER_LEN

```
#define MAX_BUFFER_LEN 1024
```

4.34.1.3 MAX_ARGS

```
#define MAX_ARGS 32
```

4.34.2 Function Documentation

4.34.2.1 lookup_builtin()

```
builtin_t * lookup_builtin (
            const char * name)
```

Lookup a builtin command.

Parameters

name	The name of the command to lookup

Returns

The builtin command if found, NULL otherwise

4.34.2.2 in_subroutine()

```
int in_subroutine (
            const char * name)
```

Check if a command is a subroutine.

Parameters

name	The name of the command to check

Returns

1 if the command is a subroutine, -1 otherwise

4.34.2.3 unmount()

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

command to unmount active file system

Note

usage: numount

Returns

-1 on failure 0 on success

4.34.2.4 mount()

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

command to mount a file system

Returns

-1 on faulure 0 on success

Note

usage: mount NAME

name does not include .pennfat extension

4.34.2.5 start_env()

```
int start_env ()
```

starts the global enviornment on a new fs

Returns

-1 on failure 0 on success

4.34.2.6 stop_env()

```
void stop_env ()
```

closes and frees environment

4.34.2.7 lookup_subroutine()

```
subroutine_t * lookup_subroutine (
            const char * name)
```

Lookup a subroutine.

Parameters

name	The name of the subroutine to lookup

Returns

The subroutine if found, NULL otherwise

4.34.2.8 jobs()

```
void jobs (
            void )
```

Display all jobs in the system.

4.34.2.9 man()

```
void man (
            void )
```

Lists all available commands.

Example Usage: man

4.34.2.10 logout()

```
void logout (
            void )
```

Exits the shell and shutsdown PennOS.

Example Usage: logout

4.34.2.11 fg()

```
void * fg (
            void * arg)
```

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

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

4.34.2.12 bg()

```
void * bg (
            void * arg)
```

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

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

4.34.2.13 execute_command()

```
int execute_command (
            struct parsed_command * pcmd)
```

Execute a command.

Parameters

pcmd	Parsed command structure

Returns

int Process ID of the spawned command or -1 on error

4.34.2.14 shell()

```
void * shell (
            void * arg)
```

The main shell process function implementation.

Parameters

arg	Command line arguments

Returns

void* NULL

4.34.2.15 main()

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

4.34.3 Variable Documentation

4.34.3.1 next_job_id

```
int next_job_id  [extern]
```

4.34.3.2 current_job_id

```
int current_job_id  [extern]
```

4.34.3.3 foreground_job_id

```
int foreground_job_id  [extern]
```

4.34.3.4 most_recent_job_id

```
int most_recent_job_id  [extern]
```

4.34.3.5 running

```
bool running  [extern]
```

4.34.3.6 MOUNTED_FS

```
fs_t* MOUNTED_FS = NULL
```

global mounted file system

4.34.3.7 GLOBAL_FDT

```
fdt_t* GLOBAL_FDT = NULL
```

global file descriptor table

4.34.3.8 WORKING_DIR

```
dir_t* WORKING_DIR = NULL
```

global current working directory

4.34.3.9 ENV

```
int ENV = 0
```

global flag for intialized environment

4.34.3.10 interrupted

```
volatile sig_atomic_t interrupted = 0
```

4.34.3.11 stdin_fd

```
int stdin_fd = 0
```

4.34.3.12 stdout_fd

```
int stdout_fd = 1
```

4.34.3.13 builtin_table

```
builtin_t builtin_table[]
```

Initial value:

```
= {
    {"sleep", u_sleep, "Sleep for `n` seconds."},
    {"ps", ps,
     "List all processes on PennOS, displaying PID, PPID, priority, status, "
     "and command name."},
    {"zombify", zombify,
     "Used to test zombifying functionality of your kernel."},
    {"orphanify", orphanify,
     "Used to test orphanifying functionality of your kernel."},
    {"busy", busy,
     "Busy wait indefinitely. It can only be interrupted via signals."},
    {"echo", echo, "Echo back an input string."},
    {"hang", hang, "Tests the hang functionality of kernel."},
    {"nohang", nohang, "Tests the nohang functionality of kernel."},
    {"recur", recur, "Tests the recur functionality of kernel."},
    {"cat", cat, "Concatenates files and prints to stdout"},
    {"ls", ls, "Lists all files in directory"},
    {"touch", touch, "Make files"},
    {"mv", mv, "Renames files"},
    {"cp", cp, "Copy contents of one file into another"},
    {"rm", rm, "Remove files"},
    {"chmod", chmod, "Change file permissions"},
    {"crash", crash, "crash demo test"},
    {NULL, NULL, NULL}
}
```

4.34.3.14 subroutine_table

```
subroutine_t subroutine_table[]
```

Initial value:

```
= {
    {"nice",
     "Spawn a new process for `command` and set its priority to `priority`."},
    {"nice_pid", "Adjust the priority level of an existing process."},
    {"man", "Lists all available commands."},
    {"bg",
     "Resumes the most recently stopped job in the background, or the job "
     "specified by job_id."},
    {"fg",
     "Brings the most recently stopped or background job to the foreground, or "
     "the job specified by job_id."},
    {"jobs", "Lists all jobs."},
    {"logout", "Exits the shell and shutsdown PennOS."},
    {"kill", "Sends a signal to a process."},
    {NULL, NULL}
}
```

4.35 pennos.h File Reference

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

Data Structures

- struct builtin_t

  Struct representing a builtin command.
- struct subroutine_t

  Struct representing a subroutine.

Functions

- builtin_t * lookup_builtin (const char *name)
  Lookup a builtin command.
- int in_subroutine (const char *name)
  Check if a command is a subroutine.
- subroutine_t * lookup_subroutine (const char *name)
  Lookup a subroutine.
- int unmount (int argc, char *argv[])
  command to unmount active file system
- int mount (int argc, char *argv[])
  command to mount a file system
- int start_env ()
  starts the global enviornment on a new fs
- void stop_env ()
  closes and frees environment
- int execute_command (struct parsed_command *pcmd)
  Execute a command.
- void * shell (void *arg)
  The main shell process function implementation.
- void * bg (void *arg)
  Resumes the most recently stopped job in the background, or the job specified by job_id.
- void * fg (void *arg)
  Brings the most recently stopped or background job to the foreground, or the job specified by job_id.
- void jobs (void)
  Display all jobs in the system.
- void logout (void)
  Exits the shell and shutsdown PennOS.
- void man (void)
  Lists all available commands.

Variables

- int next_job_id
- int current_job_id
- int stdin_fd
- int stdout_fd

4.35.1 Function Documentation

4.35.1.1 lookup_builtin()

```
builtin_t * lookup_builtin (
            const char * name)
```

Lookup a builtin command.

Parameters

name	The name of the command to lookup

Returns

The builtin command if found, NULL otherwise

4.35.1.2 in_subroutine()

```
int in_subroutine (
            const char * name)
```

Check if a command is a subroutine.

Parameters

name	The name of the command to check

Returns

1 if the command is a subroutine, -1 otherwise

4.35.1.3 lookup_subroutine()

```
subroutine_t * lookup_subroutine (
            const char * name)
```

Lookup a subroutine.

Parameters

name	The name of the subroutine to lookup

Returns

The subroutine if found, NULL otherwise

4.35.1.4 unmount()

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

command to unmount active file system

Note

usage: numount

Returns

-1 on failure 0 on success

4.35.1.5 mount()

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

command to mount a file system

Returns

-1 on faulure 0 on success

Note

usage: mount NAME

name does not include .pennfat extension

4.35.1.6 start_env()

```
int start_env ()
```

starts the global enviornment on a new fs

Returns

-1 on failure 0 on success

4.35.1.7 stop_env()

```
void stop_env ()
```

closes and frees environment

4.35.1.8 execute_command()

```
int execute_command (
            struct parsed_command * pcmd)
```

Execute a command.

Parameters

pcmd	Parsed command structure

Returns

int Process ID of the spawned command or -1 on error

4.35.1.9 shell()

```
void * shell (
            void * arg)
```

The main shell process function implementation.

Parameters

arg	Command line arguments

Returns

void∗ NULL

4.35.1.10 bg()

```
void * bg (
            void * arg)
```

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

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

4.35.1.11 fg()

```
void * fg (
            void * arg)
```

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

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

4.35.1.12 jobs()

```
void jobs (
            void )
```

Display all jobs in the system.

4.35.1.13 logout()

```
void logout (
            void )
```

Exits the shell and shutsdown PennOS.

Example Usage: logout

4.35.1.14 man()

```
void man (
            void )
```

Lists all available commands.

Example Usage: man

4.35.2 Variable Documentation

4.35.2.1 next_job_id

```
int next_job_id  [extern]
```

4.35.2.2 current_job_id

```
int current_job_id  [extern]
```

4.35.2.3 stdin_fd

```
int stdin_fd  [extern]
```

4.35.2.4 stdout_fd

```
int stdout_fd  [extern]
```

4.36 pennos.h

Go to the documentation of this file.

```
00001 #ifndef PENNOS_H
00002 #define PENNOS_H
00003
00004 #include "util/parser.h"
00005
00009 typedef struct {
00010   const char* name;
00011   void* (*func)(void*);
00012   const char* description;
00013 } builtin_t;
00014
00018 typedef struct {
00019   const char* name;
00020   const char* description;
00021 } subroutine_t;
00022
00023 extern int next_job_id;
00024 extern int current_job_id;
00025 extern int stdin_fd;
00026 extern int stdout_fd;
00027
00033 builtin_t* lookup_builtin(const char* name);
00034
00040 int in_subroutine(const char* name);
00041
00047 subroutine_t* lookup_subroutine(const char* name);
00048
00049 /*============================================================================
00050  * Filesystem Setup
00051  *============================================================================*/
00052
00059 int unmount(int argc, char* argv[]);
00060
00068 int mount(int argc, char* argv[]) ;
00069
00075 int start_env();
00076
00081 void stop_env();
00082
00083 /*============================================================================
00084  * Execute Command
00085  *============================================================================*/
```

```
00086
00093 int execute_command(struct parsed_command* pcmd);
00094
00095 /*==========================================================================
00096  * Shell
00097  *==========================================================================*/
00098
00105 void* shell(void* arg);
00106
00114 void* bg(void* arg);
00115
00123 void* fg(void* arg);
00124
00128 void jobs(void);
00129
00135 void logout(void);
00136
00142 void man(void);
00143
00144 #endif  // PENNOS_H
```

4.37 queue.c File Reference

```
#include "./queue.h"
#include <stdlib.h>
#include "../src/util/panic.h"
#include "../src/util/Vec.h"
#include "scheduler.h"
```

Functions

- void queue_destroy (Vec *queue)
- bool queue_enqueue (Vec *queue, pcb_t *child)
- bool queue_dequeue (Vec *queue, pcb_t *child)
- bool queue_is_empty (Vec *queue)
- bool in_queue (Vec *queue, int pid)
- PriorityQueue pq_create (int levels, ptr_dtor_fn ele_dtor_fn)
- void pq_destroy (PriorityQueue *pq)
- bool pq_enqueue (PriorityQueue *pq, ptr_t element, int level)
- ptr_t pq_dequeue (PriorityQueue *pq, int level_out)
- bool pq_is_empty (PriorityQueue *pq)
- size_t pq_size (PriorityQueue *pq)
- bool pq_remove (PriorityQueue *pq, int pid)

4.37.1 Function Documentation

4.37.1.1 queue_destroy()

```
void queue_destroy (
            Vec * queue)
```

4.37.1.2 queue_enqueue()

```
bool queue_enqueue (
            Vec * queue,
            pcb_t * child)
```

4.37.1.3 queue_dequeue()

```
bool queue_dequeue (
            Vec * queue,
            pcb_t * child)
```

4.37.1.4 queue_is_empty()

```
bool queue_is_empty (
            Vec * queue)
```

4.37.1.5 in_queue()

```
bool in_queue (
            Vec * queue,
            int pid)
```

4.37.1.6 pq_create()

```
PriorityQueue pq_create (
            int levels,
            ptr_dtor_fn ele_dtor_fn)
```

Create a new priority queue with the specified number of priority levels

Parameters

levels	The number of priority levels
ele_dtor_fn	Function to clean up elements

Returns

A newly created priority queue

4.37.1.7 pq_destroy()

```
void pq_destroy (
            PriorityQueue * pq)
```

Destroy a priority queue and free all resources

Parameters

pq	Pointer to the priority queue to destroy

4.37.1.8 pq_enqueue()

```
bool pq_enqueue (
            PriorityQueue * pq,
            ptr_t element,
            int level)
```

Add an element to the priority queue with the specified priority

Parameters

pq	Pointer to the priority queue
element	Element to add
level	Level (0 is highest)

Returns

true if operation was successful

4.37.1.9 pq_dequeue()

```
ptr_t pq_dequeue (
            PriorityQueue * pq,
            int level_out)
```

Remove and return the next element from the priority queue based on priority and round-robin selection within each priority level

Parameters

pq	Pointer to the priority queue
level_out	Pointer to store the level of the dequeued element (can be NULL)

Returns

The dequeued element, or NULL if queue is empty

4.37.1.10 pq_is_empty()

```
bool pq_is_empty (
            PriorityQueue * pq)
```

Check if the priority queue is empty

Parameters

pq	Pointer to the priority queue

Returns

true if the queue is empty, false otherwise

4.37.1.11 pq_size()

```
size_t pq_size (
            PriorityQueue * pq)
```

Get the number of elements in the priority queue

Parameters

pq	Pointer to the priority queue

Returns

The total number of elements across all priority levels

4.37.1.12 pq_remove()

```
bool pq_remove (
            PriorityQueue * pq,
            int pid)
```

Remove all instances of an element from the priority queue by PID

Parameters

pq	Pointer to the priority queue
pid	The PID of the element to remove

Returns

true if any elements were removed, false otherwise

4.38 queue.h File Reference

```
#include <stdbool.h>
#include "util/Vec.h"
```

Data Structures

- struct PriorityQueue

Typedefs

- typedef struct pcb pcb_t

Functions

- void queue_destroy (Vec *queue)
- bool queue_enqueue (Vec *queue, pcb_t *child)
- bool queue_dequeue (Vec *queue, pcb_t *child)
- bool queue_is_empty (Vec *queue)
- void queue_print (Vec *queue)
- bool in_queue (Vec *queue, int pid)
- PriorityQueue pq_create (int levels, ptr_dtor_fn ele_dtor_fn)
- void pq_destroy (PriorityQueue *pq)
- bool pq_enqueue (PriorityQueue *pq, ptr_t element, int level)
- ptr_t pq_dequeue (PriorityQueue *pq, int level_out)
- bool pq_is_empty (PriorityQueue *pq)
- size_t pq_size (PriorityQueue *pq)
- bool pq_remove (PriorityQueue *pq, int pid)
- void print_priority_queue (PriorityQueue *pq)

4.38.1 Typedef Documentation

4.38.1.1 pcb_t

```
typedef struct pcb pcb_t
```

4.38.2 Function Documentation

4.38.2.1 queue_destroy()

```
void queue_destroy (
            Vec * queue)
```

4.38.2.2 queue_enqueue()

```
bool queue_enqueue (
            Vec * queue,
            pcb_t * child)
```

4.38.2.3 queue_dequeue()

```
bool queue_dequeue (
            Vec * queue,
            pcb_t * child)
```

4.38.2.4 queue_is_empty()

```
bool queue_is_empty (
            Vec * queue)
```

4.38.2.5 queue_print()

```
void queue_print (
            Vec * queue)
```

4.38.2.6 in_queue()

```
bool in_queue (
            Vec * queue,
            int pid)
```

4.38.2.7 pq_create()

```
PriorityQueue pq_create (
            int levels,
            ptr_dtor_fn ele_dtor_fn)
```

Create a new priority queue with the specified number of priority levels

Parameters

levels	The number of priority levels
ele_dtor↩ _fn	Function to clean up elements

Returns

A newly created priority queue

4.38.2.8 pq_destroy()

```
void pq_destroy (
            PriorityQueue * pq)
```

Destroy a priority queue and free all resources

Parameters

pq	Pointer to the priority queue to destroy

4.38.2.9 pq_enqueue()

```
bool pq_enqueue (
            PriorityQueue * pq,
            ptr_t element,
            int level)
```

Add an element to the priority queue with the specified priority

Parameters

pq	Pointer to the priority queue
element	Element to add
level	Level (0 is highest)

Returns

true if operation was successful

4.38.2.10 pq_dequeue()

```
ptr_t pq_dequeue (
            PriorityQueue * pq,
            int level_out)
```

Remove and return the next element from the priority queue based on priority and round-robin selection within each priority level

Parameters

pq	Pointer to the priority queue
level_out	Pointer to store the level of the dequeued element (can be NULL)

Returns

The dequeued element, or NULL if queue is empty

4.38.2.11 pq_is_empty()

```
bool pq_is_empty (
            PriorityQueue * pq)
```

Check if the priority queue is empty

Parameters

pq	Pointer to the priority queue

Returns

true if the queue is empty, false otherwise

4.38.2.12 pq_size()

```
size_t pq_size (
            PriorityQueue * pq)
```

Get the number of elements in the priority queue

Parameters

pq	Pointer to the priority queue

Returns

The total number of elements across all priority levels

4.38.2.13 pq_remove()

```
bool pq_remove (
            PriorityQueue * pq,
            int pid)
```

Remove all instances of an element from the priority queue by PID

Parameters

pq	Pointer to the priority queue
pid	The PID of the element to remove

Returns

true if any elements were removed, false otherwise

4.38.2.14 print_priority_queue()

```
void print_priority_queue (
            PriorityQueue * pq)
```

Print the priority queue

Parameters

pq	Pointer to the priority queue

4.39 queue.h

Go to the documentation of this file.

```
00001 #ifndef QUEUE_H_
00002 #define QUEUE_H_
00003
00004 #include <stdbool.h>
00005 #include "util/Vec.h"
00006
00007 // Forward declare pcb_t to avoid circular dependency
00008 typedef struct pcb pcb_t;
00009
00010 typedef struct {
00011     Vec *queues; // Array of Vecs, one for each priority level
00012     int levels;  // Number of priority levels
```

```
00013 } PriorityQueue;
00014
00015 void queue_destroy(Vec *queue);
00016
00017 bool queue_enqueue(Vec *queue, pcb_t *child);
00018
00019 bool queue_dequeue(Vec *queue, pcb_t *child);
00020
00021 bool queue_is_empty(Vec *queue);
00022
00023 void queue_print(Vec *queue);
00024
00025 bool in_queue(Vec *queue, int pid);
00026
00034 PriorityQueue pq_create(int levels, ptr_dtor_fn ele_dtor_fn);
00035
00041 void pq_destroy(PriorityQueue *pq);
00042
00051 bool pq_enqueue(PriorityQueue *pq, ptr_t element, int level);
00052
00061 ptr_t pq_dequeue(PriorityQueue *pq, int level_out);
00062
00069 bool pq_is_empty(PriorityQueue *pq);
00070
00077 size_t pq_size(PriorityQueue *pq);
00078
00086 bool pq_remove(PriorityQueue *pq, int pid);
00087
00093 void print_priority_queue(PriorityQueue *pq);
00094
00095 #endif // QUEUE_H_
```

4.40 routines.c File Reference

routines for pennfat

```
#include "routines.h"
#include <stdio.h>
#include "util/macro.h"
```

Functions

- int unmount (int argc, char *argv[])
  command to unmount active file system
- int mount (int argc, char *argv[])
  command to mount a file system
- int mkfs (int argc, char *argv[])
  command to create a file system
- int touch (int num_args, char *argv[])
  creates files
- int mv (int argc, char *argv[])
  renames a file from source to desitination
- int rm (int num_args, char *argv[])
  removes files
- int cp (int num_args, char *argv[])
  copies files
- void sigint_handler2 (int signo)
- int cat_terminal_append_or_write (int fd)
  helper for cat when reading from terminal
- int cat (int num_args, char *argv[])
  concatinates files and sends output to a file

- uint8_t str_to_perms (char *str)
  converts a string into permissions integer
- int chmod (int argc, char *argv[])
  changes a files permissions
- int ls (int argc, char *argv[])
  lists file info

4.40.1 Detailed Description

routines for pennfat

Author

ashely francisco, ian lamont

See also

routines.h

4.40.2 Function Documentation

4.40.2.1 unmount()

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

command to unmount active file system

Note

usage: numount

Returns

-1 on failure 0 on success

4.40.2.2 mount()

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

command to mount a file system

Returns

-1 on faulure 0 on success

Note

usage: mount NAME

name does not include .pennfat extension

4.40.2.3 mkfs()

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

command to create a file system

Returns

-1 on failure 0 on success

Note

usage: mkfs NAME BLOCKS_IN_FAT BLOCK_SIZE_CONFIG

BLOCKS_IN_FAT in range 1-32 and BLOCK_SIZE_CONFIG following 2^{8+x} where x in range 0-4

4.40.2.4 touch()

```
int touch (
            int num_args,
            char * argv[])
```

creates files

Parameters

int	num_args argc
char∗	argv[0] argv

Note

touch FILE ...

creates each file in arguments if it does not already exist

Returns

-1 on error 0 on success

4.40.2.5 mv()

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

renames a file from source to desitination

Parameters

int	argc argc
char∗	argv[] argv

Note

mv SOURCE DEST

renames SOURCE to DEST, if DEST exists overwrite it

Returns

-1 on error 0 on success

4.40.2.6 rm()

```
int rm (
            int num_args,
            char * argv[])
```

removes files

Parameters

int	num_args argc
char∗	argv[] argv

Note

rm FILE ...

Returns

-1 on failure 0 on success

4.40.2.7 cp()

```
int cp (
            int num_args,
            char * argv[])
```

copies files

Parameters

int	num_args argc
char∗	argv[] argv

Note

cp [-h] SOURCE DEST

copies file SOURCE into DEST on file system, -h flag specifies host file

cp SOURCE [-h] DEST

copies file SOURCE from file system into DEST, -h flag specifies host system file

Returns

-1 on failure 0 on success

4.40.2.8 sigint_handler2()

```
void sigint_handler2 (
            int signo)
```

4.40.2.9 cat_terminal_append_or_write()

```
int cat_terminal_append_or_write (
            int fd)
```

helper for cat when reading from terminal

Parameters

int	fd output file

Returns

-1 on failure 0 on success

Note

Function only used within this .c file

4.40.2.10 cat()

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

concatinates files and sends output to a file

Parameters

int	num_args argc
char∗	argv[] argv

Note

cat FILE... [((-w | -a) OUTPUT_FILE)]

concatinates the contents of FILE... and writes (-w) or appends (-a) to OUTPUT_FILE or to sdtout

cat (-w | -a) OUTPUT_FILE

reads sdtin and writes (-w) or appends (-a) to OUTPUT_FILE

Returns

-1 on failure 0 on success

4.40.2.11 str_to_perms()

```
uint8_t str_to_perms (
            char * str)
```

converts a string into permissions integer

Parameters

char∗	str string to convert

Returns

unit8_t permissions

4.40.2.12 chmod()

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

changes a files permissions

Parameters

int	argc argc
char∗	argv[] argv

Returns

-1 on failure 0 on success

Note

chmod OPTION FILE...

changes the permision of FILE... based on OPTION which is a regex

OPTION follows +/- to add or remove permissions and rwx to specifiy read write and execute

4.40.2.13 ls()

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

lists file info

Parameters

int	argc argc
char∗	argv[] argv

Returns

-1 on failure 0 on success

Note

ls

list all files in directory

4.41 routines.h File Reference

routines for pennfat

```
#include "fs.h"
#include "kernel_fs.h"
```

Functions

- void print_error (const char *message)
  prints error message to stderr using k_write
- int mkfs (int argc, char *argv[ ])
  command to create a file system
- int mount (int argc, char *argv[ ])
  command to mount a file system
- int unmount (int argc, char *argv[ ])
  command to unmount active file system
- int touch (int num_args, char *argv[ ])
  creates files
- int mv (int argc, char *argv[ ])
  renames a file from source to desitination
- int rm (int num_args, char *argv[ ])
  removes files
- int cp (int num_args, char *argv[ ])
  copies files
- int cat (int argc, char *argv[ ])
  concatinates files and sends output to a file
- int chmod (int argc, char *argv[ ])
  changes a files permissions
- int ls (int argc, char *argv[ ])
  lists file info
- uint8_t str_to_perms (char *str)
  converts a string into permissions integer

Variables

- volatile sig_atomic_t interrupted

4.41.1 Detailed Description

routines for pennfat

Author

ashely francisco, ian lamont

See also

routines.c

4.41.2 Function Documentation

4.41.2.1 print_error()

```
void print_error (
            const char * message)  [extern]
```

prints error message to stderr using k_write

Parameters

const	char* message

4.41.2.2 mkfs()

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

command to create a file system

Returns

-1 on failure 0 on success

Note

usage: mkfs NAME BLOCKS_IN_FAT BLOCK_SIZE_CONFIG

BLOCKS_IN_FAT in range 1-32 and BLOCK_SIZE_CONFIG following 2^{8+x} where x in range 0-4

4.41.2.3 mount()

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

command to mount a file system

Returns

-1 on faulure 0 on success

Note

usage: mount NAME

name does not include .pennfat extension

4.41.2.4 unmount()

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

command to unmount active file system

Note

usage: numount

Returns

-1 on failure 0 on success

4.41.2.5 touch()

```
int touch (
            int num_args,
            char * argv[])
```

creates files

Parameters

int	num_args argc
char∗	argv[0] argv

Note

touch FILE ...

creates each file in arguments if it does not already exist

Returns

-1 on error 0 on success

4.41.2.6 mv()

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

renames a file from source to desitination

Parameters

int	argc argc
char∗	argv[] argv

Note

mv SOURCE DEST

renames SOURCE to DEST, if DEST exists overwrite it

Returns

-1 on error 0 on success

4.41.2.7 rm()

```
int rm (
            int num_args,
            char * argv[])
```

removes files

Parameters

int	num_args argc
char∗	argv[] argv

Note

rm FILE ...

Returns

-1 on failure 0 on success

4.41.2.8 cp()

```
int cp (
            int num_args,
            char * argv[])
```

copies files

Parameters

int	num_args argc
char∗	argv[] argv

Note

cp [-h] SOURCE DEST

copies file SOURCE into DEST on file system, -h flag specifies host file

cp SOURCE [-h] DEST

copies file SOURCE from file system into DEST, -h flag specifies host system file

Returns

-1 on failure 0 on success

4.41.2.9 cat()

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

concatinates files and sends output to a file

Parameters

int	num_args argc
char∗	argv[] argv

Note

cat FILE... [((-w | -a) OUTPUT_FILE)]

concatinates the contents of FILE... and writes (-w) or appends (-a) to OUTPUT_FILE or to sdtout

cat (-w | -a) OUTPUT_FILE

reads sdtin and writes (-w) or appends (-a) to OUTPUT_FILE

Returns

-1 on failure 0 on success

4.41.2.10 chmod()

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

changes a files permissions

Parameters

int	argc argc
char∗	argv[] argv

Returns

-1 on failure 0 on success

Note

chmod OPTION FILE...

changes the permision of FILE... based on OPTION which is a regex

OPTION follows +/- to add or remove permissions and rwx to specifiy read write and execute

4.41.2.11 ls()

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

lists file info

Parameters

int	argc argc
char∗	argv[] argv

Returns

-1 on failure 0 on success

Note

ls

list all files in directory

4.41.2.12 str_to_perms()

```
uint8_t str_to_perms (
            char * str)
```

converts a string into permissions integer

Parameters

char∗	str string to convert

Returns

unit8_t permissions

4.41.3 Variable Documentation

4.41.3.1 interrupted

```
volatile sig_atomic_t interrupted  [extern]
```

4.42 routines.h

Go to the documentation of this file.

```
00001
00007
00008 #ifndef ROUTINES_HEADER
00009 #define ROUTINES_HEADER
00010
00011 #include "fs.h"
00012 #include "kernel_fs.h"
00013 extern volatile sig_atomic_t interrupted;
00014 extern void print_error(const char* message);
00015
00024 int mkfs(int argc, char* argv[]);
00025
00033 int mount(int argc, char* argv[]);
00034
```

```
00041 int unmount(int argc, char* argv[]);
00042
00052 int touch(int num_args, char* argv[]);
00053
00063 int mv(int argc, char* argv[]);
00064
00073 int rm(int num_args, char* argv[]);
00074
00088 int cp(int num_args, char* argv[]);
00089
00102 int cat(int argc, char* argv[]);
00103
00115 int chmod(int argc, char* argv[]);
00116
00126 int ls(int argc, char* argv[]);
00127
00134 uint8_t str_to_perms(char* str);
00135
00136 #endif
```

4.43 scheduler.c File Reference

```
#include "scheduler.h"
#include "job.h"
#include "kernel.h"
#include "clock.h"
#include "logger.h"
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include "system.h"
```

Functions

- void scheduler_init ()

  Initialize the scheduler.
- int get_priority_by_tick (int tick_count)

  Get priority level based on tick count.
- void check_sleeping_processes ()

  Check if any sleeping processes should be unblocked Looks through all the processes and unblocks the sleeping processes and adds them back to the priority queue if needed.
- void clock_tick_handler (int signum)

  Clock tick handler Called at every time quantum, increments tick.
- void signal_handler ()

  Handle processes that are signaled to be stopped or killed before the next scheduling cycle.
- void job_status_update ()

  Update the status of running and stopped jobs.
- void reap_shell_zombies ()

  Reap shell zombies to be done before scheduling.
- pcb_t * find_process (int pid)

  Find a process by pid.
- void schedule ()

  Schedule a process to run.
- void scheduler_idle ()

  Idle the scheduler when no processes are ready.
- void scheduler_cleanup ()

  Clean up scheduler resources.
- void scheduler_run ()

  Run the scheduler until stopped.

Variables

- const int PRIORITY_HIGH = 0
- const int PRIORITY_MEDIUM = 1
- const int PRIORITY_LOW = 2
- const int PROCESS_RUNNING = 0
- const int PROCESS_BLOCKED = 1
- const int PROCESS_STOPPED = 2
- const int PROCESS_ZOMBIE = 3
- const int P_STATUS_NONE = 0
- const int P_STATUS_EXITED = 1
- const int P_STATUS_STOPPED = 2
- const int P_STATUS_SIGNALED = 3
- const int NUM_PRIORITY_LEVELS = 3
- int ticks = 0
- int next_pid = 1
- Vec all_processes
- bool running = false
- sigset_t scheduler_mask
- PriorityQueue process_queue
- pcb_t * current_process = NULL
- pcb_t * terminal_controller = NULL
- Vec running_jobs
- Vec stopped_jobs

4.43.1 Function Documentation

4.43.1.1 scheduler_init()

```
void scheduler_init ()
```

Initialize the scheduler.

1. Initialize priority queue w/ 3 levels
2. Initialize storage for all processes
3. Signal mask for scheduler to only receive SIGALRM
4. Set up clock and logger and running state to true

4.43.1.2 get_priority_by_tick()

```
int get_priority_by_tick (
            int tick_count)
```

Get priority level based on tick count.

Parameters

tick_count	The current tick count

Returns

The priority level

4.43.1.3 check_sleeping_processes()

```
void check_sleeping_processes ()
```

Check if any sleeping processes should be unblocked Looks through all the processes and unblocks the sleeping processes and adds them back to the priority queue if needed.

4.43.1.4 clock_tick_handler()

```
void clock_tick_handler (
            int signum)
```

Clock tick handler Called at every time quantum, increments tick.

1. Critical section - protect with sigprocmask
2. Increment the tick counter

Parameters

signum	The signal number

4.43.1.5 signal_handler()

```
void signal_handler ()
```

Handle processes that are signaled to be stopped or killed before the next scheduling cycle.

4.43.1.6 job_status_update()

```
void job_status_update ()
```

Update the status of running and stopped jobs.

4.43.1.7 reap_shell_zombies()

```
void reap_shell_zombies ()
```

Reap shell zombies to be done before scheduling.

4.43.1.8 find_process()

```
pcb_t * find_process (
            int pid)
```

Find a process by pid.

Parameters

pid	The pid of the process to find

Returns

The process if found, NULL otherwise

4.43.1.9 schedule()

```
void schedule ()
```

Schedule a process to run.

4.43.1.10 scheduler_idle()

```
void scheduler_idle ()
```

Idle the scheduler when no processes are ready.

4.43.1.11 scheduler_cleanup()

```
void scheduler_cleanup ()
```

Clean up scheduler resources.

1. Stop the clock
2. Cleanup and destroy process table
3. Cleanup current process
4. Destroy the priority queue
5. Cleanup the logger

4.43.1.12 scheduler_run()

```
void scheduler_run ()
```

Run the scheduler until stopped.

4.43.2 Variable Documentation

4.43.2.1 PRIORITY_HIGH

```
const int PRIORITY_HIGH = 0
```

4.43.2.2 PRIORITY_MEDIUM

```
const int PRIORITY_MEDIUM = 1
```

4.43.2.3 PRIORITY_LOW

```
const int PRIORITY_LOW = 2
```

4.43.2.4 PROCESS_RUNNING

```
const int PROCESS_RUNNING = 0
```

4.43.2.5 PROCESS_BLOCKED

```
const int PROCESS_BLOCKED = 1
```

4.43.2.6 PROCESS_STOPPED

```
const int PROCESS_STOPPED = 2
```

4.43.2.7 PROCESS_ZOMBIE

```
const int PROCESS_ZOMBIE = 3
```

4.43.2.8 P_STATUS_NONE

```
const int P_STATUS_NONE = 0
```

4.43.2.9 P_STATUS_EXITED

```
const int P_STATUS_EXITED = 1
```

4.43.2.10 P_STATUS_STOPPED

```
const int P_STATUS_STOPPED = 2
```

4.43.2.11 P_STATUS_SIGNALED

```
const int P_STATUS_SIGNALED = 3
```

4.43.2.12 NUM_PRIORITY_LEVELS

```
const int NUM_PRIORITY_LEVELS = 3
```

4.43.2.13 ticks

```
int ticks = 0
```

4.43.2.14 next_pid

```
int next_pid = 1
```

4.43.2.15 all_processes

```
Vec all_processes
```

4.43.2.16 running

```
bool running = false
```

4.43.2.17 scheduler_mask

```
sigset_t scheduler_mask
```

4.43.2.18 process_queue

```
PriorityQueue process_queue
```

4.43.2.19 current_process

```
pcb_t* current_process = NULL
```

4.43.2.20 terminal_controller

```
pcb_t* terminal_controller = NULL
```

4.43.2.21 running_jobs

```
Vec running_jobs  [extern]
```

4.43.2.22 stopped_jobs

```
Vec stopped_jobs  [extern]
```

4.44 scheduler.h File Reference

```
#include <signal.h>
#include <stdbool.h>
#include <sys/time.h>
#include "../src/util/Vec.h"
#include "../src/util/spthread.h"
#include "./queue.h"
#include "fd.h"
```

Data Structures

- struct pcb
- struct process_queue_t

Typedefs

- typedef struct pcb pcb_t

Functions

- void scheduler_init ()

  Initialize the scheduler.
- int get_priority_by_tick (int tick_count)

  Get priority level based on tick count.
- void check_sleeping_processes ()

  Check if any sleeping processes should be unblocked Looks through all the processes and unblocks the sleeping processes and adds them back to the priority queue if needed.
- void clock_tick_handler (int signum)

  Clock tick handler Called at every time quantum, increments tick.
- void signal_handler ()

  Handle processes that are signaled to be stopped or killed before the next scheduling cycle.
- void job_status_update ()

  Update the status of running and stopped jobs.
- void reap_shell_zombies ()

  Reap shell zombies to be done before scheduling.
- pcb_t * find_process (int pid)

  Find a process by pid.
- void schedule ()

  Schedule a process to run.
- void scheduler_idle ()

  Idle the scheduler when no processes are ready.
- void scheduler_cleanup ()

  Clean up scheduler resources.
- void scheduler_run ()

  Run the scheduler until stopped.

Variables

- const int PRIORITY_HIGH
- const int PRIORITY_MEDIUM
- const int PRIORITY_LOW
- const int NUM_PRIORITY_LEVELS
- const int PROCESS_RUNNING
- const int PROCESS_BLOCKED
- const int PROCESS_STOPPED
- const int PROCESS_ZOMBIE
- const int P_STATUS_NONE
- const int P_STATUS_EXITED
- const int P_STATUS_STOPPED
- const int P_STATUS_SIGNALED
- PriorityQueue process_queue
- pcb_t * current_process
- int next_pid
- bool running
- int ticks
- Vec all_processes
- sigset_t scheduler_mask
- pcb_t * terminal_controller

4.44.1 Typedef Documentation

4.44.1.1 pcb_t

```
typedef struct pcb pcb_t
```

4.44.2 Function Documentation

4.44.2.1 scheduler_init()

```
void scheduler_init ()
```

Initialize the scheduler.

1. Initialize priority queue w/ 3 levels
2. Initialize storage for all processes
3. Signal mask for scheduler to only receive SIGALRM
4. Set up clock and logger and running state to true

4.44.2.2 get_priority_by_tick()

```
int get_priority_by_tick (
            int tick_count)
```

Get priority level based on tick count.

Parameters

tick_count	The current tick count

Returns

The priority level

4.44.2.3 check_sleeping_processes()

```
void check_sleeping_processes ()
```

Check if any sleeping processes should be unblocked Looks through all the processes and unblocks the sleeping processes and adds them back to the priority queue if needed.

4.44.2.4 clock_tick_handler()

```
void clock_tick_handler (
            int signum)
```

Clock tick handler Called at every time quantum, increments tick.

1. Critical section - protect with sigprocmask
2. Increment the tick counter

Parameters

signum	The signal number

4.44.2.5 signal_handler()

```
void signal_handler ()
```

Handle processes that are signaled to be stopped or killed before the next scheduling cycle.

4.44.2.6 job_status_update()

```
void job_status_update ()
```

Update the status of running and stopped jobs.

4.44.2.7 reap_shell_zombies()

```
void reap_shell_zombies ()
```

Reap shell zombies to be done before scheduling.

4.44.2.8 find_process()

```
pcb_t * find_process (
            int pid)
```

Find a process by pid.

Parameters

pid	The pid of the process to find

Returns

The process if found, NULL otherwise

4.44.2.9 schedule()

```
void schedule ()
```

Schedule a process to run.

4.44.2.10 scheduler_idle()

```
void scheduler_idle ()
```

Idle the scheduler when no processes are ready.

4.44.2.11 scheduler_cleanup()

```
void scheduler_cleanup ()
```

Clean up scheduler resources.

1. Stop the clock
2. Cleanup and destroy process table
3. Cleanup current process
4. Destroy the priority queue
5. Cleanup the logger

4.44.2.12 scheduler_run()

```
void scheduler_run ()
```

Run the scheduler until stopped.

4.44.3 Variable Documentation

4.44.3.1 PRIORITY_HIGH

```
const int PRIORITY_HIGH  [extern]
```

4.44.3.2 PRIORITY_MEDIUM

```
const int PRIORITY_MEDIUM  [extern]
```

4.44.3.3 PRIORITY_LOW

```
const int PRIORITY_LOW  [extern]
```

4.44.3.4 NUM_PRIORITY_LEVELS

```
const int NUM_PRIORITY_LEVELS  [extern]
```

4.44.3.5 PROCESS_RUNNING

```
const int PROCESS_RUNNING  [extern]
```

4.44.3.6 PROCESS_BLOCKED

```
const int PROCESS_BLOCKED  [extern]
```

4.44.3.7 PROCESS_STOPPED

```
const int PROCESS_STOPPED  [extern]
```

4.44.3.8 PROCESS_ZOMBIE

```
const int PROCESS_ZOMBIE  [extern]
```

4.44.3.9 P_STATUS_NONE

```
const int P_STATUS_NONE  [extern]
```

4.44.3.10 P_STATUS_EXITED

```
const int P_STATUS_EXITED  [extern]
```

4.44.3.11 P_STATUS_STOPPED

```
const int P_STATUS_STOPPED  [extern]
```

4.44.3.12 P_STATUS_SIGNALED

```
const int P_STATUS_SIGNALED  [extern]
```

4.44.3.13 process_queue

```
PriorityQueue process_queue  [extern]
```

4.44.3.14 current_process

```
pcb_t* current_process  [extern]
```

4.44.3.15 next_pid

```
int next_pid  [extern]
```

4.44.3.16 running

```
bool running  [extern]
```

4.44.3.17 ticks

```
int ticks  [extern]
```

4.44.3.18 all_processes

```
Vec all_processes  [extern]
```

4.44.3.19 scheduler_mask

```
sigset_t scheduler_mask  [extern]
```

4.44.3.20 terminal_controller

```
pcb_t* terminal_controller  [extern]
```

4.45 scheduler.h

Go to the documentation of this file.

```
00001 #ifndef SCHEDULER_H
00002 #define SCHEDULER_H
00003
00004 #include <signal.h>
00005 #include <stdbool.h>
00006 #include <sys/time.h>
00007 #include "../src/util/Vec.h"
00008 #include "../src/util/spthread.h"
00009 #include "./queue.h"
00010 #include "fd.h"
00011 #include "util/spthread.h"
00012
00013 // Process structure
00014 typedef struct pcb {
00015   spthread_t thread;       // Thread handle
00016   int pid;                 // Process ID for identification
00017   char* name;              // Process name for logging
00018   int priority;            // Priority level (0, 1, 2)
00019   int state;               // Process state
00020   struct pcb* parent;      // Parent process
00021   Vec children;            // Currently alive children
00022   Vec zombie_children;     // Zombie children
00023   Vec waitable_children;   // Waitable children
00024   int stdin_fd;            // Standard input file descriptor
00025   int stdout_fd;           // Standard output file descriptor
00026   int wake;                // Wake-up time for the process
00027   bool sleeping;           // Whether the process is sleeping
00028   int exit_status;         // Exit status of the process
00029   int job_id;              // Job ID of the process
00030   fdt_t* fdt;              // Fd table to hold info from global table
00031   bool signal_handled;     // Whether the signal has been handled
00032 } pcb_t;
00033
00034 // Priority levels
00035 extern const int PRIORITY_HIGH;
00036 extern const int PRIORITY_MEDIUM;
00037 extern const int PRIORITY_LOW;
00038
00039 // Number of priority levels, CAN ONLY BE 3 FOR PennOS
00040 extern const int NUM_PRIORITY_LEVELS;
00041
00042 // Process states
00043 extern const int PROCESS_RUNNING;
00044 extern const int PROCESS_BLOCKED;
00045 extern const int PROCESS_STOPPED;
00046 extern const int PROCESS_ZOMBIE;
00047
00048 // Process exit status
00049 extern const int P_STATUS_NONE;
00050 extern const int P_STATUS_EXITED;
00051 extern const int P_STATUS_STOPPED;
00052 extern const int P_STATUS_SIGNALED;
00053
00054 // Global variables
00055 extern PriorityQueue process_queue;
00056
00057 // Currently running process
00058 extern pcb_t* current_process;
00059
00060 // Next process ID to assign
00061 extern int next_pid;
00062
00063 // Indicator for whether scheduler is running or not
00064 extern bool running;
00065
00066 // Number of ticks counted by the clock so far
00067 extern int ticks;
00068
00069 // For tracking process states and finding processes by pid
00070 extern Vec all_processes;
00071
00072 // Mask for scheduler operations
00073 extern sigset_t scheduler_mask;
00074
00075 // Process that currently has terminal control
00076 extern pcb_t* terminal_controller;
00077
00078 // Queue structure for each priority level
00079 typedef struct {
00080   pcb_t* head;  // Head of the queue
00081   pcb_t* tail;  // Tail of the queue
00082   int count;    // Number of processes in the queue
```

```
00083 } process_queue_t;
00084
00092 void scheduler_init();
00093
00099 int get_priority_by_tick(int tick_count);
00100
00106 void check_sleeping_processes();
00107
00116 void clock_tick_handler(int signum);
00117
00122 void signal_handler();
00123
00127 void job_status_update();
00128
00132 void reap_shell_zombies();
00133
00139 pcb_t* find_process(int pid);
00140
00144 void schedule();
00145
00149 void scheduler_idle();
00150
00159 void scheduler_cleanup();
00160
00164 void scheduler_run();
00165
00166 #endif  // SCHEDULER_H
```

4.46 stress.c File Reference

```
#include "stress.h"
#include "kernel.h"
#include "system.h"
#include <signal.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
```

Functions

- void * hang (void *arg)
- void * nohang (void *arg)
- void * recur (void *arg)
- void * crash (void *arg)

4.46.1 Function Documentation

4.46.1.1 hang()

```
void * hang (
            void * arg)
```

4.46.1.2 nohang()

```
void * nohang (
            void * arg)
```

4.46.1.3 recur()

```
void * recur (
            void * arg)
```

4.46.1.4 crash()

```
void * crash (
            void * arg)
```

4.47 stress.h File Reference

Functions

- void * hang (void *)
- void * nohang (void *)
- void * recur (void *)
- void * crash (void *)

4.47.1 Function Documentation

4.47.1.1 hang()

```
void * hang (
            void * arg)
```

4.47.1.2 nohang()

```
void * nohang (
            void * arg)
```

4.47.1.3 recur()

```
void * recur (
            void * arg)
```

4.47.1.4 crash()

```
void * crash (
            void * arg)
```

4.48 stress.h

Go to the documentation of this file.

```
00001 #ifndef STRESS_H_
00002 #define STRESS_H_
00003
00004 void* hang(void*);
00005 void* nohang(void*);
00006 void* recur(void*);
00007
00008 // this one requires the fs to hold at least 5480 bytes for a file.
00009 void* crash(void*);
00010
00011 #endif
```

4.49 system.c File Reference

```
#include "system.h"
#include "../src/util/Vec.h"
#include "./queue.h"
#include "job.h"
#include "kernel.h"
#include "kernel_fs.h"
#include "logger.h"
#include "p_errno.h"
#include "pennos.h"
#include "scheduler.h"
#include "user.h"
```

Data Structures

- struct standard_fds_t

Macros

- #define MAX_NAME_LEN 32
- #define STDIN_FILENO 0
- #define STDOUT_FILENO 1

Functions

- fdt_t * get_current_process_fdt ()

  returns the current process' file descriptor table
- fdt_t * get_shell_fdt ()

  returns the shell's file descriptor table
- pid_t s_spawn (void *(*func)(void *), char *argv[], int fd0, int fd1, int job_id, int nice_value)

  Create a child process that executes the function `func`. The child will retain some attributes of the parent.
- pid_t s_reap (pcb_t *parent, pcb_t *child)

  Reap a child process.
- pid_t s_update_wstatus (pcb_t *child, pcb_t *parent, int *wstatus)

  Update wait status of a child process and reap it.
- pid_t s_waitpid (pid_t pid, int *wstatus, bool nohang)

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

- pid_t s_waitpid_any (pcb_t *parent, int *wstatus, bool nohang)

  Wait on any child of the calling process, until it changes state. If `nohang` is true, return immediately if no child has exited.
- int s_kill (pid_t pid, int signal)

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

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

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

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

  Get the process ID of the calling process.
- void * init_process (void *arg)

  The init process function that creates the shell process.
- void * s_shell (void *arg)

  The shell process function.
- void * s_init (void *arg)

  Initialize the init process and set it as the current process, then set the shell as the terminal controller then set the current process to the init process.
- int s_proc_info (Vec *processes)

  Get information about all processes in the system.
- bool s_check_job_status_change (int job_id)

  Check if all of a job's processes have changed state.
- int s_open (const char *fname, int mode)

  opens a file in a given mode
- int s_read (int fd, int n, char *buf)

  reads bytes into a buffer from a file
- int s_write (int fd, const char *str, int n)

  writes a given number of bytes to a file
- int s_close (int fd)

  closes a file
- int s_unlink (const char *fname)

  removes a file from directory
- int s_lseek (int fd, int offset, int whence)

  sets the read/write head of a file
- int s_ls (const char *filename)

  lists files in a directory
- int s_chmod (char **argv, int argc, uint8_t perms)

  changes the permissions of files
- uint8_t s_get_permissions (const char *filename)

  return the permissions on a file
- int set_redirection (struct parsed_command *pcmd, int *stdin_fd, int *stdout_fd)

  sets the commands redirected files
- void sigint_handler (int sig)

  Signal handler for SIGINT (Ctrl+C) should be sent to all processes in the foreground process group.
- void sigstop_handler (int sig)

  Signal handler for SIGSTOP should be sent to all processes in the foreground process group.

Variables

- int ticks
- Vec all_processes
- PriorityQueue process_queue
- pcb_t * current_process
- pcb_t * terminal_controller
- int foreground_job_id
- const int P_STATUS_NONE
- const int P_STATUS_EXITED
- const int P_STATUS_STOPPED
- const int P_STATUS_SIGNALED

4.49.1 Macro Definition Documentation

4.49.1.1 MAX_NAME_LEN

```
#define MAX_NAME_LEN 32
```

4.49.1.2 STDIN_FILENO

```
#define STDIN_FILENO 0
```

4.49.1.3 STDOUT_FILENO

```
#define STDOUT_FILENO 1
```

4.49.2 Function Documentation

4.49.2.1 get_current_process_fdt()

```
fdt_t * get_current_process_fdt ()
```

returns the current process' file descriptor table

Returns

fdt_t* curent process file descriptor table

4.49.2.2 get_shell_fdt()

```
fdt_t * get_shell_fdt ()
```

returns the shell's file descriptor table

Returns

fdt_t* shell file descriptor table

4.49.2.3 s_spawn()

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

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

Parameters

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

Returns

pid_t The process ID of the created child process.

4.49.2.4 s_reap()

```
pid_t s_reap (
            pcb_t * parent,
            pcb_t * child)
```

Reap a child process.

Parameters

parent	The parent process
child	The child process

Returns

The pid of the child process

4.49.2.5 s_update_wstatus()

```
pid_t s_update_wstatus (
            pcb_t * child,
            pcb_t * parent,
            int * wstatus)
```

Update wait status of a child process and reap it.

Parameters

child	Child process
parent	Parent process
wstatus	Pointer to integer status variable

Returns

Int pid of child process

4.49.2.6 s_waitpid()

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

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

Parameters

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

Returns

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

4.49.2.7 s_waitpid_any()

```
pid_t s_waitpid_any (
            pcb_t * parent,
            int * wstatus,
            bool nohang)
```

Wait on any child of the calling process, until it changes state. If `nohang` is true, return immediately if no child has exited.

Parameters

parent	The parent process.
wstatus	Pointer to an integer variable where the status will be stored.
nohang	If true, return immediately if no child has exited.

4.49.2.8 s_kill()

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

Send a signal to a particular process.

Parameters

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

Returns

0 on success, -1 on error.

4.49.2.9 s_exit()

```
void s_exit (
             void )
```

Unconditionally exit the calling process.

4.49.2.10 s_nice()

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

Set the priority of the specified thread.

1. Remove the process from the current priority queue
2. Set the new priority and log the change
3. Add the process back to the priority queue with the new priority

Parameters

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

Returns

0 on success, -1 on failure.

4.49.2.11 s_sleep()

```
void s_sleep (
             unsigned int sleep_ticks)
```

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

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

Parameters

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

4.49.2.12 s_getpid()

```
pid_t s_getpid (
            void )
```

Get the process ID of the calling process.

Returns

pid_t The process ID of the calling process.

4.49.2.13 init_process()

```
void * init_process (
            void * arg)
```

The init process function that creates the shell process.

4.49.2.14 s_shell()

```
void * s_shell (
            void * arg)
```

The shell process function.

4.49.2.15 s_init()

```
void * s_init (
            void * arg)
```

Initialize the init process and set it as the current process, then set the shell as the terminal controller then set the current process to the init process.

4.49.2.16 s_proc_info()

```
int s_proc_info (
            Vec * processes)
```

Get information about all processes in the system.

Parameters

pointer	to a vector of processes

Returns

int Number of processes retrieved, -1 on error

4.49.2.17 s_check_job_status_change()

```
bool s_check_job_status_change (
            int job_id)
```

Check if all of a job's processes have changed state.

Parameters

job↩ _id	The job's job_id

Returns

bool True if all processes in the job have changed state, false otherwise

4.49.2.18 s_open()

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

opens a file in a given mode

Parameters

const	char* fname file name to open
int	mode mode to open file in

Returns

int file descriptor int, -1 on error

4.49.2.19 s_read()

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

reads bytes into a buffer from a file

Parameters

int	fd file to read from
int	n number of bytes to read
char∗	buf byte array to read to

Returns

int number of bytes read, -1 on error

4.49.2.20 s_write()

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

writes a given number of bytes to a file

Parameters

int	fd file to write to
const	char∗ str byte string to write to file
int	n number of bytes to write

Returns

int number of bytes written, -1 on error

4.49.2.21 s_close()

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

closes a file

Parameters

int	fd file to close

Returns

int0 on success, -1 on error

4.49.2.22 s_unlink()

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

removes a file from directory

Parameters

const	char* fname file to remove

Returns

int 0 on success, -1 on error

4.49.2.23 s_lseek()

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

sets the read/write head of a file

Parameters

int	fd file
int	offset how far to move the head
int	whence takes constant mode

Returns

int new offset on success, -1 on error

4.49.2.24 s_ls()

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

lists files in a directory

Parameters

const	char* filename file(s) to list or whole directory if none listed

Returns

int 0 on success, -1 on error

4.49.2.25 s_chmod()

```
int s_chmod (
            char ** argv,
            int argc,
            uint8_t perms)
```

changes the permissions of files

Parameters

char∗∗	argv command argv with files to change
int	argc command argc
uint8_t	perms permissions to update for files

Returns

int 0 on success, -1 on error

4.49.2.26 s_get_permissions()

```
uint8_t s_get_permissions (
            const char * filename)
```

return the permissions on a file

Parameters

const	char∗ filename file

Returns

uint8_t permissions on that file

4.49.2.27 set_redirection()

```
int set_redirection (
            struct parsed_command * pcmd,
            int * stdin_fd,
            int * stdout_fd)
```

sets the commands redirected files

Parameters

	struct	parsed_command∗ pcmd command to redirect
out	*int∗*	stdin_fd new stdin file
out	*int∗*	stdout_fd new stdout file

Returns

int 0 on success, -1 on error

4.49.2.28 sigint_handler()

```
void sigint_handler (
            int sig)
```

Signal handler for SIGINT (Ctrl+C) should be sent to all processes in the foreground process group.

4.49.2.29 sigstop_handler()

```
void sigstop_handler (
            int sig)
```

Signal handler for SIGSTOP should be sent to all processes in the foreground process group.

4.49.3 Variable Documentation

4.49.3.1 ticks

```
int ticks  [extern]
```

4.49.3.2 all_processes

```
Vec all_processes  [extern]
```

4.49.3.3 process_queue

```
PriorityQueue process_queue  [extern]
```

4.49.3.4 current_process

```
pcb_t* current_process  [extern]
```

4.49.3.5 terminal_controller

```
pcb_t* terminal_controller  [extern]
```

4.49.3.6 foreground_job_id

```
int foreground_job_id  [extern]
```

4.49.3.7 P_STATUS_NONE

```
const int P_STATUS_NONE  [extern]
```

4.49.3.8 P_STATUS_EXITED

```
const int P_STATUS_EXITED  [extern]
```

4.49.3.9 P_STATUS_STOPPED

```
const int P_STATUS_STOPPED  [extern]
```

4.49.3.10 P_STATUS_SIGNALED

```
const int P_STATUS_SIGNALED  [extern]
```

4.50 system.h File Reference

system functions for pennos

```
#include "./util/Vec.h"
#include "kernel.h"
#include "kernel_fs.h"
#include "util/macro.h"
#include "util/parser.h"
```

Macros

- #define P_SIGSTOP 17
- #define P_SIGCONT 19
- #define P_SIGTERM 1

Functions

- void print_error (const char *message)

  prints error message to stderr using k_write
- pid_t s_spawn (void *(*func)(void *), char *argv[], int fd0, int fd1, int job_id, int nice_value)

  Create a child process that executes the function `func`. The child will retain some attributes of the parent.
- pid_t s_reap (pcb_t *parent, pcb_t *child)

  Reap a child process.
- pid_t s_update_wstatus (pcb_t *child, pcb_t *parent, int *wstatus)

  Update wait status of a child process and reap it.
- pid_t s_waitpid (pid_t pid, int *wstatus, bool nohang)

  Wait on a child of the calling process, until it changes state. If `nohang` is true, this will not block the calling process and return immediately.
- pid_t s_waitpid_any (pcb_t *parent, int *wstatus, bool nohang)

  Wait on any child of the calling process, until it changes state. If `nohang` is true, return immediately if no child has exited.
- int s_kill (pid_t pid, int signal)

  Send a signal to a particular process.

- void s_exit (void)

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

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

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

  Get the process ID of the calling process.
- void * init_process (void *arg)

  The init process function that creates the shell process.
- void * s_shell (void *arg)

  The shell process function.
- void * s_init (void *arg)

  Initialize the init process and set it as the current process, then set the shell as the terminal controller then set the current process to the init process.
- int s_proc_info (Vec *processes)

  Get information about all processes in the system.
- bool s_check_job_status_change (int job_id)

  Check if all of a job's processes have changed state.
- void sigint_handler (int sig)

  Signal handler for SIGINT (Ctrl+C) should be sent to all processes in the foreground process group.
- void sigstop_handler (int sig)

  Signal handler for SIGSTOP should be sent to all processes in the foreground process group.
- int s_open (const char *fname, int mode)

  opens a file in a given mode
- int s_read (int fd, int n, char *buf)

  reads bytes into a buffer from a file
- int s_write (int fd, const char *str, int n)

  writes a given number of bytes to a file
- int s_close (int fd)

  closes a file
- int s_unlink (const char *fname)

  removes a file from directory
- int s_lseek (int fd, int offset, int whence)

  sets the read/write head of a file
- int s_ls (const char *filename)

  lists files in a directory
- int s_chmod (char **argv, int argc, uint8_t perms)

  changes the permissions of files
- int set_redirection (struct parsed_command *pcmd, int *stdin_fd, int *stdout_fd)

  sets the commands redirected files
- uint8_t s_get_permissions (const char *filename)

  return the permissions on a file
- fdt_t * get_current_process_fdt ()

  returns the current process' file descriptor table
- fdt_t * get_shell_fdt ()

  returns the shell's file descriptor table

4.50.1 Detailed Description

system functions for pennos

See also

system.c

4.50.2 Macro Definition Documentation

4.50.2.1 P_SIGSTOP

```
#define P_SIGSTOP 17
```

4.50.2.2 P_SIGCONT

```
#define P_SIGCONT 19
```

4.50.2.3 P_SIGTERM

```
#define P_SIGTERM 1
```

4.50.3 Function Documentation

4.50.3.1 print_error()

```
void print_error (
            const char * message)  [extern]
```

prints error message to stderr using k_write

Parameters

const	char* message

4.50.3.2 s_spawn()

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

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

Parameters

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

Returns

pid_t The process ID of the created child process.

4.50.3.3 s_reap()

```
pid_t s_reap (
            pcb_t * parent,
            pcb_t * child)
```

Reap a child process.

Parameters

parent	The parent process
child	The child process

Returns

The pid of the child process

4.50.3.4 s_update_wstatus()

```
pid_t s_update_wstatus (
            pcb_t * child,
            pcb_t * parent,
            int * wstatus)
```

Update wait status of a child process and reap it.

Parameters

child	Child process
parent	Parent process
wstatus	Pointer to integer status variable

Returns

Int pid of child process

4.50.3.5 s_waitpid()

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

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

Parameters

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

Returns

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

4.50.3.6 s_waitpid_any()

```
pid_t s_waitpid_any (
            pcb_t * parent,
            int * wstatus,
            bool nohang)
```

Wait on any child of the calling process, until it changes state. If `nohang` is true, return immediately if no child has exited.

Parameters

parent	The parent process.
wstatus	Pointer to an integer variable where the status will be stored.
nohang	If true, return immediately if no child has exited.

4.50.3.7 s_kill()

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

Send a signal to a particular process.

Parameters

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

Returns

0 on success, -1 on error.

4.50.3.8 s_exit()

```
void s_exit (
             void )
```

Unconditionally exit the calling process.

4.50.3.9 s_nice()

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

Set the priority of the specified thread.

1. Remove the process from the current priority queue
2. Set the new priority and log the change
3. Add the process back to the priority queue with the new priority

Parameters

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

Returns

0 on success, -1 on failure.

4.50.3.10 s_sleep()

```
void s_sleep (
             unsigned int sleep_ticks)
```

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

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

Parameters

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

4.50.3.11 s_getpid()

```
pid_t s_getpid (
            void )
```

Get the process ID of the calling process.

Returns

pid_t The process ID of the calling process.

4.50.3.12 init_process()

```
void * init_process (
            void * arg)
```

The init process function that creates the shell process.

4.50.3.13 s_shell()

```
void * s_shell (
            void * arg)
```

The shell process function.

4.50.3.14 s_init()

```
void * s_init (
            void * arg)
```

Initialize the init process and set it as the current process, then set the shell as the terminal controller then set the current process to the init process.

4.50.3.15 s_proc_info()

```
int s_proc_info (
            Vec * processes)
```

Get information about all processes in the system.

Parameters

pointer	to a vector of processes

Returns

int Number of processes retrieved, -1 on error

4.50.3.16 s_check_job_status_change()

```
bool s_check_job_status_change (
            int job_id)
```

Check if all of a job's processes have changed state.

Parameters

job_id	The job's job_id

Returns

bool True if all processes in the job have changed state, false otherwise

4.50.3.17 sigint_handler()

```
void sigint_handler (
            int sig)
```

Signal handler for SIGINT (Ctrl+C) should be sent to all processes in the foreground process group.

4.50.3.18 sigstop_handler()

```
void sigstop_handler (
            int sig)
```

Signal handler for SIGSTOP should be sent to all processes in the foreground process group.

4.50.3.19 s_open()

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

opens a file in a given mode

Parameters

const	char* fname file name to open
int	mode mode to open file in

Returns

int file descriptor int, -1 on error

4.50.3.20 s_read()

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

reads bytes into a buffer from a file

Parameters

int	fd file to read from
int	n number of bytes to read
char∗	buf byte array to read to

Returns

int number of bytes read, -1 on error

4.50.3.21 s_write()

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

writes a given number of bytes to a file

Parameters

int	fd file to write to
const	char∗ str byte string to write to file
int	n number of bytes to write

Returns

int number of bytes written, -1 on error

4.50.3.22 s_close()

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

closes a file

Parameters

int	fd file to close

Returns

int0 on success, -1 on error

4.50.3.23 s_unlink()

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

removes a file from directory

Parameters

const	char* fname file to remove

Returns

int 0 on success, -1 on error

4.50.3.24 s_lseek()

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

sets the read/write head of a file

Parameters

int	fd file
int	offset how far to move the head
int	whence takes constant mode

Returns

int new offset on success, -1 on error

4.50.3.25 s_ls()

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

lists files in a directory

Parameters

const	char* filename file(s) to list or whole directory if none listed

Returns

int 0 on success, -1 on error

4.50.3.26 s_chmod()

```
int s_chmod (
            char ** argv,
            int argc,
            uint8_t perms)
```

changes the permissions of files

Parameters

char∗∗	argv command argv with files to change
int	argc command argc
uint8_t	perms permissions to update for files

Returns

int 0 on success, -1 on error

4.50.3.27 set_redirection()

```
int set_redirection (
            struct parsed_command * pcmd,
            int * stdin_fd,
            int * stdout_fd)
```

sets the commands redirected files

Parameters

	struct	parsed_command∗ pcmd command to redirect
out	*int∗*	stdin_fd new stdin file
out	*int∗*	stdout_fd new stdout file

Returns

int 0 on success, -1 on error

4.50.3.28 s_get_permissions()

```
uint8_t s_get_permissions (
            const char * filename)
```

return the permissions on a file

Parameters

const	char∗ filename file

Returns

uint8_t permissions on that file

4.50.3.29 get_current_process_fdt()

```
fdt_t * get_current_process_fdt ()
```

returns the current process' file descriptor table

Returns

fdt_t* curent process file descriptor table

4.50.3.30 get_shell_fdt()

```
fdt_t * get_shell_fdt ()
```

returns the shell's file descriptor table

Returns

fdt_t* shell file descriptor table

4.51 system.h

Go to the documentation of this file.

```
00001
00006
00007 #ifndef SYSTEM_H
00008 #define SYSTEM_H
00009
00010 #include "./util/Vec.h"
00011 #include "./util/Vec.h"  // Include Vec.h
00012 #include "kernel.h"
00013 #include "kernel.h"  // Include kernel.h for proc_info_t
00014 #include "kernel_fs.h"
00015 #include "util/macro.h"
00016 #include "util/parser.h"
00017
00018 extern void print_error(const char* message);
00019
00020 // 1.2 Signals in PennOS
00021 // Your PennOS will not have traditional signals and
00022 // signal handlers (however, the host operating system may
00023 // deliver signals that you must handle). Instead, signaling
00024 // a PennOS process indicates to PennOS that it should take
00025 // some action related to the signaled thread, such as change
00026 // the state of a thread to stopped or running. Your operating
00027 // system will minimally define the following signals:
00028
00029 #define P_SIGSTOP 17  // a thread receiving this signal should be stopped
00030 #define P_SIGCONT 19  // a thread receiving this signal should be continued
00031 #define P_SIGTERM 1   // a thread receiving this signal should be terminated
00032
00033 // If you would like to add additional signals, be sure to document
00034 // them and their functionality in your companion document file.
00035 // To be clear, you should not need to call the signal function,
00036 // pthread_kill or other such functions to deliver these signals
00037 // through the host operating system. Your code should instead handle
00038 // these "signals" their own way, even if they are not used like how
00039 // we have used signals in past assignments.
00040
00041 // Forward declaration of PCB structure
00042 typedef struct pcb pcb_t;
00043
00044 // Your operating system must provide the following system
00045 // call functions for interacting with PennOS process creation:
00046
00060 pid_t s_spawn(void* (*func)(void*),
00061               char* argv[],
```

```
00062                int fd0,
00063                int fd1,
00064                int job_id,
00065                int nice_value);
00066
00073 pid_t s_reap(pcb_t* parent, pcb_t* child);
00074
00082 pid_t s_update_wstatus(pcb_t* child, pcb_t* parent, int* wstatus);
00083
00096 pid_t s_waitpid(pid_t pid, int* wstatus, bool nohang);
00097
00107 pid_t s_waitpid_any(pcb_t* parent, int* wstatus, bool nohang);
00108
00116 int s_kill(pid_t pid, int signal);
00117
00121 void s_exit(void);
00122
00133 int s_nice(pid_t pid, int priority);
00134
00147 void s_sleep(unsigned int sleep_ticks);
00148
00154 pid_t s_getpid(void);
00155
00159 void* init_process(void* arg);
00160
00164 void* s_shell(void* arg);
00165
00172 void* s_init(void* arg);
00173
00180 int s_proc_info(Vec* processes);
00181
00189 bool s_check_job_status_change(int job_id);
00190
00195 void sigint_handler(int sig);
00196
00201 void sigstop_handler(int sig);
00202
00210 int s_open(const char* fname, int mode);
00211
00220 int s_read(int fd, int n, char* buf);
00221
00230 int s_write(int fd, const char* str, int n);
00231
00238 int s_close(int fd);
00239
00246 int s_unlink(const char* fname);
00247
00256 int s_lseek(int fd, int offset, int whence);
00257
00264 int s_ls(const char* filename);
00265
00274 int s_chmod(char** argv, int argc, uint8_t perms);
00275
00284 int set_redirection(struct parsed_command* pcmd, int* stdin_fd, int* stdout_fd);
00285
00292 uint8_t s_get_permissions(const char* filename);
00293
00299 fdt_t* get_current_process_fdt();
00300
00306 fdt_t* get_shell_fdt();
00307
00308 #endif  // SYSTEM_H
```

4.52 user.c File Reference

```
#include "user.h"
#include "./util/Vec.h"
#include "p_errno.h"
#include "system.h"
```

Macros

- #define STDIN_FILENO 0
- #define STDOUT_FILENO 1

Functions

- void proc_info_destroy (void *ptr)

  Destructor for proc_info_t.
- void * ps (void *arg)

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

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

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

  Helper for orphanify.
- void * orphanify (void *arg)

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

  Sleep for `n` seconds.

Variables

- const int P_STATUS_EXITED
- const int P_STATUS_STOPPED
- const int P_STATUS_SIGNALED
- const int PROCESS_RUNNING
- const int PROCESS_BLOCKED
- const int PROCESS_STOPPED
- const int PROCESS_ZOMBIE
- int next_job_id

4.52.1 Macro Definition Documentation

4.52.1.1 STDIN_FILENO

```
#define STDIN_FILENO 0
```

4.52.1.2 STDOUT_FILENO

```
#define STDOUT_FILENO 1
```

4.52.2 Function Documentation

4.52.2.1 proc_info_destroy()

```
void proc_info_destroy (
            void * ptr)
```

Destructor for proc_info_t.

Parameters

ptr	Pointer to process info

4.52.2.2 ps()

```
void * ps (
            void * arg)
```

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

Example Usage: ps

4.52.2.3 zombie_child()

```
void * zombie_child (
            void * arg)
```

Helper for zombify.

4.52.2.4 zombify()

```
void * zombify (
            void * arg)
```

Used to test zombifying functionality of your kernel.

Example Usage: zombify

4.52.2.5 orphan_child()

```
void * orphan_child (
            void * arg)
```

Helper for orphanify.

4.52.2.6 orphanify()

```
void * orphanify (
            void * arg)
```

Used to test orphanifying functionality of your kernel.

Example Usage: orphanify

4.52.2.7 u_sleep()

```
void * u_sleep (
            void * arg)
```

Sleep for n seconds.

Note that you'll have to convert the number of seconds to the correct number of ticks. Note that a tick is 100 milliseconds. So there are 10 ticks in a second.

Example Usage: sleep 10

4.52.3 Variable Documentation

4.52.3.1 P_STATUS_EXITED

```
const int P_STATUS_EXITED  [extern]
```

4.52.3.2 P_STATUS_STOPPED

```
const int P_STATUS_STOPPED  [extern]
```

4.52.3.3 P_STATUS_SIGNALED

```
const int P_STATUS_SIGNALED  [extern]
```

4.52.3.4 PROCESS_RUNNING

```
const int PROCESS_RUNNING  [extern]
```

4.52.3.5 PROCESS_BLOCKED

```
const int PROCESS_BLOCKED  [extern]
```

4.52.3.6 PROCESS_STOPPED

```
const int PROCESS_STOPPED  [extern]
```

4.52.3.7 PROCESS_ZOMBIE

```
const int PROCESS_ZOMBIE  [extern]
```

4.52.3.8 next_job_id

```
int next_job_id  [extern]
```

4.53 user.h File Reference

```
#include "system.h"
#include "p_errno.h"
```

Macros

- #define P_WIFEXITED(status)
- #define P_WIFSTOPPED(status)
- #define P_WIFSIGNALED(status)

Functions

- void u_perror (const char *s)

  Print error message based on P_ERRNO.
- void * ps (void *arg)

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

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

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

  Helper for orphanify.
- void * orphanify (void *arg)

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

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

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

  Echo back an input string.
- void proc_info_destroy (void *ptr)

  Destructor for proc_info_t.

4.53.1 Macro Definition Documentation

4.53.1.1 P_WIFEXITED

```
#define P_WIFEXITED(
              status)
```

Value:

```
(status == P_STATUS_EXITED)
```

4.53.1.2 P_WIFSTOPPED

```
#define P_WIFSTOPPED(
              status)
```

Value:

```
(status == P_STATUS_STOPPED)
```

4.53.1.3 P_WIFSIGNALED

```
#define P_WIFSIGNALED(
              status)
```

Value:

```
(status == P_STATUS_SIGNALED)
```

4.53.2 Function Documentation

4.53.2.1 u_perror()

```
void u_perror (
             const char * s)
```

Print error message based on P_ERRNO.

Parameters

s	Prefix string for error message

4.53.2.2 ps()

```
void * ps (
             void * arg)
```

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

Example Usage: ps

4.53.2.3 zombie_child()

```
void * zombie_child (
             void * arg)
```

Helper for zombify.

4.53.2.4 zombify()

```
void * zombify (
            void * arg)
```

Used to test zombifying functionality of your kernel.

Example Usage: zombify

4.53.2.5 orphan_child()

```
void * orphan_child (
            void * arg)
```

Helper for orphanify.

4.53.2.6 orphanify()

```
void * orphanify (
            void * arg)
```

Used to test orphanifying functionality of your kernel.

Example Usage: orphanify

4.53.2.7 busy()

```
void * busy (
            void * arg)
```

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

Example Usage: busy

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

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

Example Usage: sleep 10

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

Example Usage: busy

4.53.2.8 u_sleep()

```
void * u_sleep (
            void * arg)
```

Sleep for `n` seconds.

Note that you'll have to convert the number of seconds to the correct number of ticks. Note that a tick is 100 milliseconds. So there are 10 ticks in a second.

Example Usage: sleep 10

4.53.2.9 echo()

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

Echo back an input string.

Example Usage: echo Hello World

4.53.2.10 proc_info_destroy()

```
void proc_info_destroy (
            void * ptr)
```

Destructor for proc_info_t.

Parameters

ptr	Pointer to process info

4.54 user.h

Go to the documentation of this file.

```
00001 #include "system.h"
00002 #include "p_errno.h"
00003
00004 #define P_WIFEXITED(status)   (status == P_STATUS_EXITED)
00005 #define P_WIFSTOPPED(status)  (status == P_STATUS_STOPPED)
00006 #define P_WIFSIGNALED(status) (status == P_STATUS_SIGNALED)
00007
00012 void u_perror(const char* s);
00013
00020 void* ps(void *arg);
00021
00025 void* zombie_child(void* arg);
00026
00032 void* zombify(void* arg);
00033
00037 void* orphan_child(void* arg);
00038
00044 void* orphanify(void* arg);
00045
00052 void* busy(void *arg);
00053
00062 void* u_sleep(void *arg);
00063
00069 void* echo(void *arg);
00070
00075 void proc_info_destroy(void* ptr);
```

4.55 util/macro.h File Reference

macros for pennos

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "kernel_fs.h"
```

Macros

- #define K_FPRINTF_BUFFER 100
- #define safe_alloc(fn, ...)
- #define safe_fn(fn, ...)
- #define k_fprintf(file, str, ...)
- #define k_printf(file, str)
- #define host_fprintf(file, str, ...)
- #define host_printf(file, str)
- #define k_std_fprintf(str, ...)
- #define k_std_printf(str)
- #define host_std_fprintf(str, ...)
- #define host_std_printf(str)

4.55.1 Detailed Description

macros for pennos

Author

ian lamont

4.55.2 Macro Definition Documentation

4.55.2.1 K_FPRINTF_BUFFER

```
#define K_FPRINTF_BUFFER 100
```

4.55.2.2 safe_alloc

```
#define safe_alloc(
              fn,
              ...)
```

Value:

```
({                                                      \
  void* sm_ptr = fn(__VA_ARGS__);                       \
  if (sm_ptr == NULL) {                                 \
    fprintf(stderr, "Function failed: %s:%d (%s): %s \n", \
      __FILE__, __LINE__, __func__, #fn);               \
    perror("Error");                                    \
    exit(EXIT_FAILURE);                                 \
  }                                                     \
  (sm_ptr);                                             \
})
```

4.55.2.3 safe_fn

```
#define safe_fn(
              fn,
              ...)
```

Value:

```
({                                                      \
  pid_t sf_ret = fn(__VA_ARGS__);                       \
  if (sf_ret < 0) {                                     \
    fprintf(stderr, "Function failed: %s:%d (%s): %s \n", \
      __FILE__, __LINE__, __func__, #fn);               \
    perror("Error");                                    \
    exit(EXIT_FAILURE);                                 \
  }                                                     \
  (sf_ret);                                             \
})
```

4.55.2.4 k_fprintf

```
#define k_fprintf(
               file,
               str,
               ...)
```

Value:

```
({\
   int k_fprintf_len = strlen(str);\
   int k_fprintf_written = 0;\
   char k_fprintf_buf[K_FPRINTF_BUFFER];\
   while (k_fprintf_written < k_fprintf_len) {\
      snprintf(K_FPRINTF_BUFFER, k_fprintf_buf, str, __VA_ARGS__);\
      int k_fprintf_len_buf = strlen(k_fprintf_buf);\
      k_fprintf_written += k_fprintf_len_buf;\
      if (k_write(file, k_fprintf_buf, K_FPRINTF_BUFFER) != k_fprintf_len_buf) {\
        write(STDERR_FILENO, "Failed to k_write error message: \0", 34);\
        write(STDERR_FILENO, k_fprintf_buf, k_fprintf_len_buf);\
        exit(EXIT_FAILURE);                                    \
      }\
   }\
})
```

4.55.2.5 k_printf

```
#define k_printf(
               file,
               str)
```

Value:

```
({\
   int k_fprintf_len = strlen(str);\
   int k_fprintf_written = 0;\
   char k_fprintf_buf[K_FPRINTF_BUFFER];\
   while (k_fprintf_written < k_fprintf_len) {\
      strncpy(str, k_fprintf_buf, K_FPRINTF_BUFFER);\
      int k_fprintf_len_buf = strlen(k_fprintf_buf);\
      k_fprintf_written += k_fprintf_len_buf;\
      if (k_write(file, k_fprintf_buf, K_FPRINTF_BUFFER) != k_fprintf_len_buf) {\
        write(STDERR_FILENO, "Failed to k_write in k_fprint message: \0", 34);\
        write(STDERR_FILENO, k_fprintf_buf, k_fprintf_len_buf);\
        exit(EXIT_FAILURE);                                    \
      }\
   }\
})
```

4.55.2.6 host_fprintf

```
#define host_fprintf(
               file,
               str,
               ...)
```

Value:

```
({\
   int k_fprintf_len = strlen(str);\
   int k_fprintf_written = 0;\
   char k_fprintf_buf[K_FPRINTF_BUFFER];\
   while (k_fprintf_written < k_fprintf_len) {\
      snprintf(K_FPRINTF_BUFFER, k_fprintf_buf, str, __VA_ARGS__);\
      int k_fprintf_len_buf = strlen(k_fprintf_buf);\
      k_fprintf_written += k_fprintf_len_buf;\
      write(STDERR_FILENO, k_fprintf_buf, k_fprintf_len_buf);\
      exit(EXIT_FAILURE);                                    \
   }\
})
```

4.55.2.7 host_printf

```
#define host_printf(
                file,
                str)
```

Value:

```
({\
   int k_fprintf_len = strlen(str);\
   int k_fprintf_written = 0;\
   char k_fprintf_buf[K_FPRINTF_BUFFER];\
   while (k_fprintf_written < k_fprintf_len) {\
      strncpy(str, k_fprintf_buf, K_FPRINTF_BUFFER);\
      int k_fprintf_len_buf = strlen(k_fprintf_buf);\
      k_fprintf_written += k_fprintf_len_buf;\
      write(file, k_fprintf_buf, k_fprintf_len_buf);\
   }\
})
```

4.55.2.8 k_std_fprintf

```
#define k_std_fprintf(
                str,
                ...)
```

Value:

```
k_fprintf(K_STDERR, str, ...)
```

4.55.2.9 k_std_printf

```
#define k_std_printf(
                str)
```

Value:

```
k_printf(K_STDERR, str)
```

4.55.2.10 host_std_fprintf

```
#define host_std_fprintf(
                str,
                ...)
```

Value:

```
host_fprintf(STDERR_FILENO, str, ...)
```

4.55.2.11 host_std_printf

```
#define host_std_printf(
                str)
```

Value:

```
host_printf(STDERR_FILENO, str)
```

4.56 macro.h

Go to the documentation of this file.

```
00001
00006
00007 #ifndef PENNOS_MACROS
00008 #define PENNOS_MACROS
00009
00010 #define K_FPRINTF_BUFFER 100
00011
00012 #include <stdio.h>
00013 #include <stdlib.h>
00014 #include <unistd.h>
00015 #include "kernel_fs.h"
00016
00017 #define safe_alloc(fn, ...)                                    \
00018   ({                                                           \
00019     void* sm_ptr = fn(__VA_ARGS__);                            \
00020     if (sm_ptr == NULL) {                                      \
00021       fprintf(stderr, "Function failed: %s:%d (%s): %s \n", \
00022         __FILE__, __LINE__, __func__, #fn);                    \
00023       perror("Error");                                         \
00024       exit(EXIT_FAILURE);                                      \
00025     }                                                          \
00026     (sm_ptr);                                                  \
00027   })
00028
00029 #define safe_fn(fn, ...)                                       \
00030   ({                                                           \
00031     pid_t sf_ret = fn(__VA_ARGS__);                            \
00032     if (sf_ret < 0) {                                          \
00033       fprintf(stderr, "Function failed: %s:%d (%s): %s \n", \
00034         __FILE__, __LINE__, __func__, #fn);                    \
00035       perror("Error");                                         \
00036       exit(EXIT_FAILURE);                                      \
00037     }                                                          \
00038     (sf_ret);                                                  \
00039   })
00040
00041 #define k_fprintf(file, str, ...)\
00042 ({\
00043    int k_fprintf_len = strlen(str);\
00044    int k_fprintf_written = 0;\
00045    char k_fprintf_buf[K_FPRINTF_BUFFER];\
00046    while (k_fprintf_written < k_fprintf_len) {\
00047       snprintf(K_FPRINTF_BUFFER, k_fprintf_buf, str, __VA_ARGS__);\
00048       int k_fprintf_len_buf = strlen(k_fprintf_buf);\
00049       k_fprintf_written += k_fprintf_len_buf;\
00050       if (k_write(file, k_fprintf_buf, K_FPRINTF_BUFFER) != k_fprintf_len_buf) {\
00051         write(STDERR_FILENO, "Failed to k_write error message: \0", 34);\
00052         write(STDERR_FILENO, k_fprintf_buf, k_fprintf_len_buf);\
00053         exit(EXIT_FAILURE);                                    \
00054       }\
00055    }\
00056 })
00057
00058 #define k_printf(file, str)\
00059 ({\
00060    int k_fprintf_len = strlen(str);\
00061    int k_fprintf_written = 0;\
00062    char k_fprintf_buf[K_FPRINTF_BUFFER];\
00063    while (k_fprintf_written < k_fprintf_len) {\
00064       strncpy(str, k_fprintf_buf, K_FPRINTF_BUFFER);\
00065       int k_fprintf_len_buf = strlen(k_fprintf_buf);\
00066       k_fprintf_written += k_fprintf_len_buf;\
00067       if (k_write(file, k_fprintf_buf, K_FPRINTF_BUFFER) != k_fprintf_len_buf) {\
00068         write(STDERR_FILENO, "Failed to k_write in k_fprint message: \0", 34);\
00069         write(STDERR_FILENO, k_fprintf_buf, k_fprintf_len_buf);\
00070         exit(EXIT_FAILURE);                                    \
00071       }\
00072    }\
00073 })
00074
00075 #define host_fprintf(file, str, ...)\
00076 ({\
00077    int k_fprintf_len = strlen(str);\
00078    int k_fprintf_written = 0;\
00079    char k_fprintf_buf[K_FPRINTF_BUFFER];\
00080    while (k_fprintf_written < k_fprintf_len) {\
00081       snprintf(K_FPRINTF_BUFFER, k_fprintf_buf, str, __VA_ARGS__);\
00082       int k_fprintf_len_buf = strlen(k_fprintf_buf);\
00083       k_fprintf_written += k_fprintf_len_buf;\
00084       write(STDERR_FILENO, k_fprintf_buf, k_fprintf_len_buf);\
00085       exit(EXIT_FAILURE);                                      \
00086    }\
```

```
00087 })
00088
00089 #define host_printf(file, str)\
00090 ({\
00091    int k_fprintf_len = strlen(str);\
00092    int k_fprintf_written = 0;\
00093    char k_fprintf_buf[K_FPRINTF_BUFFER];\
00094    while (k_fprintf_written < k_fprintf_len) {\
00095       strncpy(str, k_fprintf_buf, K_FPRINTF_BUFFER);\
00096       int k_fprintf_len_buf = strlen(k_fprintf_buf);\
00097       k_fprintf_written += k_fprintf_len_buf;\
00098       write(file, k_fprintf_buf, k_fprintf_len_buf);\
00099    }\
00100 })
00101
00102 #define k_std_fprintf(str, ...) k_fprintf(K_STDERR, str, ...)
00103 #define k_std_printf(str) k_printf(K_STDERR, str)
00104 #define host_std_fprintf(str, ...) host_fprintf(STDERR_FILENO, str, ...)
00105 #define host_std_printf(str) host_printf(STDERR_FILENO, str)
00106
00107 #endif
```

4.57 util/panic.c File Reference

```
#include "./panic.h"
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Functions

- void print_and_abort (const char *error_message)

4.57.1 Function Documentation

4.57.1.1 print_and_abort()

```
void print_and_abort (
            const char * error_message)
```

4.58 util/panic.h File Reference

Macros

- #define panic(error_message)

Functions

- void print_and_abort (const char *error_message)

4.58.1 Macro Definition Documentation

4.58.1.1 panic

```
#define panic(
              error_message)
```

Value:

```
print_and_abort(error_message)
```

4.58.2 Function Documentation

4.58.2.1 print_and_abort()

```
void print_and_abort (
             const char * error_message)
```

4.59 panic.h

Go to the documentation of this file.

```
00001 #ifndef PANIC_H_
00002 #define PANIC_H_
00003
00004 #ifdef DISABLE_PANIC
00005
00006 // If panics are disabled, do nothing
00007 #define panic(error_message) ((void)0)
00008
00009 #else
00010
00011 // if panices are enabled (whith they are by default) then
00012 // call the function that handles panicing
00013 #define panic(error_message) print_and_abort(error_message)
00014
00015 #endif
00016
00017 [[noreturn]] void print_and_abort(const char* error_message);
00018
00019 #endif  // PANIC_H_
```

4.60 util/parser.c File Reference

```
#include "parser.h"
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

Macros

- #define JUMP_OUT(code)

Functions

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

4.60.1 Macro Definition Documentation

4.60.1.1 JUMP_OUT

```
#define JUMP_OUT(
                code)
```

Value:

```
do {                    \
  ret_code = code;      \
  goto PROCESS_ERROR; \
} while (0)
```

4.60.2 Function Documentation

4.60.2.1 parse_command()

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

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

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

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

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

If the function returns a successful value (0), a `struct parsed_command` is guareenteed to be allocated and stored in the given `*result`. It is the caller's responsibility to free the given pointer using `free(3)`.

Otherwise, no `struct parsed_command` is allocated and `*result` is unchanged. If a system call error (-1) is returned, the caller can use `errno(3)` or `perror(3)` to gain more information about the error. layout of memory for `struct parsed_command` bool is_background; bool is_file_append;

const char *stdin_file; const char *stdout_file;

size_t num_commands;

commands are pointers to `arguments` char **commands[num_commands];

below are hidden in memory **

arguments are pointers to `original_string + num_commands` because all argv are null-terminated char *arguments[total_strings + num_commands];

original_string is a copy of the cmdline but with each token null-terminated char *original_string;

4.60.2.2 print_parsed_command()

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

4.60.2.3 print_parser_errcode()

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

4.61 util/parser.h File Reference

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

Data Structures

- struct parsed_command

Macros

- #define UNEXPECTED_FILE_INPUT 1
- #define UNEXPECTED_FILE_OUTPUT 2
- #define UNEXPECTED_PIPELINE 3
- #define UNEXPECTED_AMPERSAND 4
- #define UNEXPECTED_NICE_VALUE 5
- #define EXPECT_INPUT_FILENAME 6
- #define EXPECT_OUTPUT_FILENAME 7
- #define EXPECT_COMMANDS 8

Functions

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

4.61.1 Macro Definition Documentation

4.61.1.1 UNEXPECTED_FILE_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

4.61.1.2 UNEXPECTED_FILE_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

4.61.1.3 UNEXPECTED_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

4.61.1.4 UNEXPECTED_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

4.61.1.5 UNEXPECTED_NICE_VALUE

```
#define UNEXPECTED_NICE_VALUE 5
```

4.61.1.6 EXPECT_INPUT_FILENAME

```
#define EXPECT_INPUT_FILENAME 6
```

4.61.1.7 EXPECT_OUTPUT_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 7
```

4.61.1.8 EXPECT_COMMANDS

```
#define EXPECT_COMMANDS 8
```

4.61.2 Function Documentation

4.61.2.1 parse_command()

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

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

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

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

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

If the function returns a successful value (0), a `struct parsed_command` is guareenteed to be allocated and stored in the given `*result`. It is the caller's responsibility to free the given pointer using `free(3)`.

Otherwise, no `struct parsed_command` is allocated and `*result` is unchanged. If a system call error (-1) is returned, the caller can use `errno(3)` or `perror(3)` to gain more information about the error. layout of memory for `struct parsed_command` bool is_background; bool is_file_append;

const char *stdin_file; const char *stdout_file;

size_t num_commands;

commands are pointers to `arguments` char **commands[num_commands];

below are hidden in memory **

arguments are pointers to `original_string + num_commands` because all argv are null-terminated char *arguments[total_strings + num_commands];

original_string is a copy of the cmdline but with each token null-terminated char *original_string;

4.61.2.2 print_parsed_command()

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

4.61.2.3 print_parser_errcode()

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

4.62 parser.h

Go to the documentation of this file.

```
00001 /* Penn-Shell Parser
00002    hanbangw, 21fa    */
00003
00004 #pragma once
00005
00006 #include <stdbool.h>
00007 #include <stddef.h>
00008 #include <stdio.h>
00009
00010 /* Here defines all possible parser errors */
00011 // parser encountered an unexpected file input token '<'
00012 #define UNEXPECTED_FILE_INPUT 1
00013
00014 // parser encountered an unexpected file output token '>'
00015 #define UNEXPECTED_FILE_OUTPUT 2
00016
00017 // parser encountered an unexpected pipeline token '|'
00018 #define UNEXPECTED_PIPELINE 3
00019
00020 // parser encountered an unexpected ampersand token '&'
00021 #define UNEXPECTED_AMPERSAND 4
00022
00023 // parser encountered an unexpected ampersand token '&'
00024 #define UNEXPECTED_NICE_VALUE 5
```

```
00025
00026 // parser didn't find input filename following '<'
00027 #define EXPECT_INPUT_FILENAME 6
00028
00029 // parser didn't find output filename following '>' or '»'
00030 #define EXPECT_OUTPUT_FILENAME 7
00031
00032 // parser didn't find any commands or arguments where it expects one
00033 #define EXPECT_COMMANDS 8
00034
00039 struct parsed_command {
00040   // indicates the command shall be executed in background
00041   // (ends with an ampersand '&')
00042   bool is_background;
00043
00044   // indicates if the stdout_file shall be opened in append mode
00045   // ignore this value when stdout_file is NULL
00046   bool is_file_append;
00047
00048   // filename for redirecting input from
00049   const char* stdin_file;
00050
00051   // filename for redirecting output to
00052   const char* stdout_file;
00053
00054   // number of commands (pipeline stages)
00055   size_t num_commands;
00056
00057   // nice value (priority)
00058   size_t nice_value;
00059
00060   // an array to a list of arguments
00061   // size of `commands' is `num_commands'
00062   char** commands[];
00063 };
00064
00092 int parse_command(const char* cmd_line, struct parsed_command** result);
00093
00094 /* This is a debugging function used for outputting a parsed command line. */
00095 void print_parsed_command(const struct parsed_command* cmd);
00096
00097 /* a debugging function for printing out what error was encountered */
00098 void print_parser_errcode(FILE* output, int err_code);
```

4.63 util/spthread.c File Reference

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

Data Structures

- struct spthread_fwd_args_st
- struct spthread_signal_args_st
- struct spthread_meta_st

Macros

- #define _GNU_SOURCE
- #define MILISEC_IN_NANO 100000

- #define SPTHREAD_RUNNING_STATE 0
- #define SPTHREAD_SUSPENDED_STATE 1
- #define SPTHREAD_TERMINATED_STATE 2
- #define SPTHREAD_SIG_SUSPEND -1
- #define SPTHREAD_SIG_CONTINUE -2

Typedefs

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

Functions

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

4.63.1 Macro Definition Documentation

4.63.1.1 _GNU_SOURCE

```
#define _GNU_SOURCE
```

4.63.1.2 MILISEC_IN_NANO

```
#define MILISEC_IN_NANO 100000
```

4.63.1.3 SPTHREAD_RUNNING_STATE

```
#define SPTHREAD_RUNNING_STATE 0
```

4.63.1.4 SPTHREAD_SUSPENDED_STATE

```
#define SPTHREAD_SUSPENDED_STATE 1
```

4.63.1.5 SPTHREAD_TERMINATED_STATE

```
#define SPTHREAD_TERMINATED_STATE 2
```

4.63.1.6 SPTHREAD_SIG_SUSPEND

```
#define SPTHREAD_SIG_SUSPEND -1
```

4.63.1.7 SPTHREAD_SIG_CONTINUE

```
#define SPTHREAD_SIG_CONTINUE -2
```

4.63.2 Typedef Documentation

4.63.2.1 pthread_fn

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

4.63.2.2 spthread_fwd_args

```
typedef struct spthread_fwd_args_st spthread_fwd_args
```

4.63.2.3 spthread_signal_args

```
typedef struct spthread_signal_args_st spthread_signal_args
```

4.63.2.4 spthread_meta_t

```
typedef struct spthread_meta_st spthread_meta_t
```

4.63.3 Function Documentation

4.63.3.1 spthread_create()

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

4.63.3.2 spthread_suspend()

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

4.63.3.3 spthread_suspend_self()

```
int spthread_suspend_self ()
```

4.63.3.4 spthread_continue()

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

4.63.3.5 spthread_cancel()

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

4.63.3.6 spthread_self()

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

4.63.3.7 spthread_join()

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

4.63.3.8 spthread_exit()

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

4.63.3.9 spthread_equal()

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

4.63.3.10 spthread_disable_interrupts_self()

```
int spthread_disable_interrupts_self ()
```

4.63.3.11 spthread_enable_interrupts_self()

```
int spthread_enable_interrupts_self ()
```

4.64 util/spthread.h File Reference

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

Data Structures

- struct spthread_st

Macros

- #define SIGPTHD SIGUSR1

Typedefs

- typedef struct spthread_st spthread_t

Functions

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

4.64.1 Macro Definition Documentation

4.64.1.1 SIGPTHD

```
#define SIGPTHD SIGUSR1
```

4.64.2 Typedef Documentation

4.64.2.1 spthread_t

```
typedef struct spthread_st spthread_t
```

4.64.3 Function Documentation

4.64.3.1 spthread_create()

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

4.64.3.2 spthread_suspend()

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

4.64.3.3 spthread_suspend_self()

```
int spthread_suspend_self ()
```

4.64.3.4 spthread_continue()

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

4.64.3.5 spthread_cancel()

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

4.64.3.6 spthread_self()

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

4.64.3.7 spthread_join()

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

4.64.3.8 spthread_exit()

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

4.64.3.9 spthread_equal()

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

4.64.3.10 spthread_disable_interrupts_self()

```
int spthread_disable_interrupts_self ()
```

4.64.3.11 spthread_enable_interrupts_self()

```
int spthread_enable_interrupts_self ()
```

4.65 spthread.h

Go to the documentation of this file.

```
00001 #ifndef SPTHREAD_H_
00002 #define SPTHREAD_H_
00003
00004 #include <pthread.h>
00005 #include <stdbool.h>
00006
00007 // CAUTION: according to `man 7 pthread`:
00008 //
00009 //   On older Linux kernels, SIGUSR1 and SIGUSR2
00010 //   are used.  Applications must avoid the use of whichever set of
00011 //   signals is employed by the implementation.
00012 //
00013 // This may not work on other linux versions
00014
00015 // SIGNAL PTHREAD
00016 // NOTE: if within a created spthread you change
00017 // the behaviour of SIGUSR1, then you will not be able
00018 // to suspend and continue a spthread
00019 #define SIGPTHD SIGUSR1
00020
00021 // declares a struct, but the internals of the
00022 // struct cannot be seen by functions outside of spthread.c
00023 typedef struct spthread_meta_st spthread_meta_t;
00024
00025 // The spthread wrapper struct.
00026 // Sometimes you may have to access the inner pthread member
00027 // but you shouldn't need to do that
00028 typedef struct spthread_st {
00029   pthread_t thread;
00030   spthread_meta_t* meta;
00031 } spthread_t;
00032
00033 // NOTE:
00034 // None of these are signal safe
00035 // Also note that most of these functions are not safe to suspension,
00036 // meaning that if the thread calling these is an spthread and is suspended
00037 // in the middle of spthread_continue or spthread_suspend, then it may not work.
00038 //
00039 // Make sure that the calling thread cannot be suspended before calling these functions.
00040 // Exceptions to this are spthread_exit(), spthread_self() and if a thread is
00041 // continuing or suspending itself.
00042
00043 // spthread_create:
00044 // this function works similar to pthread_create, except for two differences.
00045 // 1) the created pthread is able to be asychronously suspended, and continued
00046 //    using the functions:
00047 //      - spthread_suspend
00048 //      - spthread_continue
00049 // 2) The created pthread will be suspended before it executes the specified
00050 //    routine. It must first be continued with `spthread_continue` before
00051 //    it will start executing.
00052 //
```

```
00053 // It is worth noting that this function is not signal safe.
00054 // In other words, it should not be called from a signal handler.
00055 //
00056 // to avoid repetition, see pthread_create(3) for details
00057 // on arguments and return values as they are the same here.
00058 int spthread_create(spthread_t* thread,
00059                     const pthread_attr_t* attr,
00060                     void* (*start_routine)(void*),
00061                     void* arg);
00062
00063 // The spthread_suspend function will signal to the
00064 // specified thread to suspend execution.
00065 //
00066 // Calling spthread_suspend on an already suspended
00067 // thread does not do anything.
00068 //
00069 // It is worth noting that this function is not signal safe.
00070 // In other words, it should not be called from a signal handler.
00071 //
00072 // args:
00073 // - pthread_t thread: the thread we want to suspend
00074 //   This thread must be created using the spthread_create() function,
00075 //   if created by some other function, the behaviour is undefined.
00076 //
00077 // returns:
00078 // - 0 on success
00079 // - EAGAIN if the thread could not be signaled
00080 // - ENOSYS if not supported on this system
00081 // - ESRCH if the thread specified is not a valid pthread
00082 int spthread_suspend(spthread_t thread);
00083
00084 // The spthread_suspend_self function will cause the calling
00085 // thread (which should be created by spthread_create) to suspend
00086 // itself.
00087 //
00088 // returns:
00089 // - 0 on success
00090 // - EAGAIN if the thread could not be signaled
00091 // - ENOSYS if not supported on this system
00092 // - ESRCH if the calling thread is not an spthread
00093 int spthread_suspend_self();
00094
00095 // The spthread_continue function will signal to the
00096 // specified thread to resume execution if suspended.
00097 //
00098 // Calling spthread_continue on an already non-suspended
00099 // thread does not do anything.
00100 //
00101 // It is worth noting that this function is not signal safe.
00102 // In other words, it should not be called from a signal handler.
00103 //
00104 // args:
00105 // - spthread_t thread: the thread we want to continue
00106 //   This thread must be created using the spthread_create() function,
00107 //   if created by some other function, the behaviour is undefined.
00108 //
00109 // returns:
00110 // - 0 on success
00111 // - EAGAIN if the thread could not be signaled
00112 // - ENOSYS if not supported on this system
00113 // - ESRCH if the thread specified is not a valid pthread
00114 int spthread_continue(spthread_t thread);
00115
00116 // The spthread_cancel function will send a
00117 // cancellation request to the specified thread.
00118 //
00119 // as of now, this function is identical to pthread_cancel(3)
00120 // so to avoid repitition, you should look there.
00121 //
00122 // Here are a few things that are worth highlighting:
00123 // - it is worth noting that it is a cancellation __request__
00124 //   the thread may not terminate immediately, instead the
00125 //   thread is checked whenever it calls a function that is
00126 //   marked as a cancellation point. At those points, it will
00127 //   start the cancellation procedure
00128 // - to make sure all things are de-allocated properly on
00129 //   normal exiting of the thread and when it is cancelled,
00130 //   you should mark a defered de-allocation with
00131 //   pthread_cleanup_push(3).
00132 //   consider the following example:
00133 //
00134 //     void* thread_routine(void* arg) {
00135 //        int* num = malloc(sizeof(int));
00136 //        pthread_cleanup_push(&free, num);
00137 //        return NULL;
00138 //     }
00139 //
```

```
00140 //    this program will allocate an integer on the heap
00141 //    and mark that data to be de-allocated on cleanup.
00142 //    This means that when the thread returns from the
00143 //    routine specified in spthread_create, free will
00144 //    be called on num. This will also happen if the thread
00145 //    is cancelled and not able to be exited normally.
00146 //
00147 //    Another function that should be used in conjunction
00148 //    is pthread_cleanup_pop(3). I will leave that
00149 //    to you to read more on.
00150 //
00151 // It is worth noting that this function is not signal safe.
00152 // In other words, it should not be called from a signal handler.
00153 //
00154 // args:
00155 // - spthread_t thread: the thread we want to cancel.
00156 //   This thread must be created using the spthread_create() function,
00157 //   if created by some other function, the behaviour is undefined.
00158 //
00159 // returns:
00160 // - 0 on success
00161 // - ESRCH if the thread specified is not a valid pthread
00162 int spthread_cancel(spthread_t thread);
00163
00164 // Can be called by a thread to get two peices of information:
00165 // 1. Whether or not the calling thread is an spthread (true or false)
00166 // 2. The spthread_t of the calling thread, if it is an spthread_t
00167 //
00168 // almost always the function will be called like this:
00169 // spthread_t self;
00170 // bool i_am_spthread = spthread_self(&self);
00171 //
00172 // args:
00173 // - spthread_t* thread: the output parameter to get the spthread_t
00174 //   representing the calling thread, if it is an spthread
00175 //
00176 // returns:
00177 // - true if the calling thread is an spthread_t
00178 // - false otherwise.
00179 bool spthread_self(spthread_t* thread);
00180
00181 // The equivalent of pthread_join but for spthread
00182 // To make sure all resources are cleaned up appropriately
00183 // spthreads that are created must at some ppoint have spthread_join
00184 // called on them. Do not use pthread_join on an spthread.
00185 //
00186 // to avoid repetition, see pthread_join(3) for details
00187 // on arguments and return values as they are the same as this function.
00188 int spthread_join(spthread_t thread, void** retval);
00189
00190 // The equivalent of pthread_exit but for spthread
00191 // spthread_exit must be used by spthreads instead of pthread_exit.
00192 // Otherwise, calls to spthread_join or other functions (like spthread_suspend)
00193 // may not work as intended.
00194 //
00195 // to avoid repetition, see pthread_exit(3) for details
00196 // on arguments and return values as they are the same as this function.
00197 void spthread_exit(void* status);
00198
00199 // The equivalent of pthread_equal but for spthread.
00200 // It two spthread_t's describe the same thread, returns a
00201 // non-zero value; otherwise it returns 0.
00202 bool spthread_equal(spthread_t first, spthread_t second);
00203
00204 // Calling this function from an spthread prevents it from
00205 // being suspended until re-enabled by the sibling function
00206 // "spthread_enable_interrupts_self".
00207 //
00208 // This is done by blocking the SIG_PTHD signal
00209 //
00210 // returns 0 on success, or -1 on error
00211 int spthread_disable_interrupts_self();
00212
00213 // Calling this function from an spthread re-enables it to
00214 // being suspendable. Should be called after it's sibling function
00215 // "spthread_disable_interrupts_self".
00216 //
00217 // This is done by unblocking the SIG_PTHD signal
00218 //
00219 // returns 0 on success, or -1 on error
00220 int spthread_enable_interrupts_self();
00221
00222 #endif  // SPTHREAD_H_
```

4.66 util/Vec.c File Reference

```
#include "./Vec.h"
#include <stdlib.h>
#include "./panic.h"
```

Functions

- Vec vec_new (size_t initial_capacity, ptr_dtor_fn ele_dtor_fn)
- void grteq_length_panic_check (Vec *self, size_t index)
- void grt_length_panic_check (Vec *self, size_t index)
- bool vec_at_capacity (Vec *self)
- bool vec_cleanup (Vec *self)
- void vec_expand (Vec *self)
- ptr_t vec_get (Vec *self, size_t index)
- void vec_set (Vec *self, size_t index, ptr_t new_ele)
- void vec_push_back (Vec *self, ptr_t new_ele)
- bool vec_pop_back (Vec *self)
- void vec_insert (Vec *self, size_t index, ptr_t new_ele)
- void vec_erase (Vec *self, size_t index)
- void vec_resize (Vec *self, size_t new_capacity)
- void vec_clear (Vec *self)
- void vec_destroy (Vec *self)

4.66.1 Function Documentation

4.66.1.1 vec_new()

```
Vec vec_new (
            size_t initial_capacity,
            ptr_dtor_fn ele_dtor_fn)
```

Creates a new empty Vec(tor) with the specified initial_capacity and specified function to clean up elements in the vector.

Parameters

initial_capacity	the initial capacity of the newly created vector, non negative
ele_dtor_fn	a function pointer to a function that takes in a ptr_t (a vector element) and cleans it up. This is commonly just `free` but custom functions can be passed in. NULL can also be passed in to specify that there is no cleanup function that needs to be called on each element.

Returns

a newly created vector with specified capacity, 0 length and the specified element destructor (cleanup) function.

Postcondition

if memory allocation fails, the function will panic.

4.66.1.2 grteq_length_panic_check()

```
void grteq_length_panic_check (
            Vec * self,
            size_t index)
```

4.66.1.3 grt_length_panic_check()

```
void grt_length_panic_check (
            Vec * self,
            size_t index)
```

4.66.1.4 vec_at_capacity()

```
bool vec_at_capacity (
            Vec * self)
```

4.66.1.5 vec_cleanup()

```
bool vec_cleanup (
            Vec * self)
```

4.66.1.6 vec_expand()

```
void vec_expand (
            Vec * self)
```

4.66.1.7 vec_get()

```
ptr_t vec_get (
            Vec * self,
            size_t index)
```

4.66.1.8 vec_set()

```
void vec_set (
            Vec * self,
            size_t index,
            ptr_t new_ele)
```

4.66.1.9 vec_push_back()

```
void vec_push_back (
            Vec * self,
            ptr_t new_ele)
```

4.66.1.10 vec_pop_back()

```
bool vec_pop_back (
            Vec * self)
```

4.66.1.11 vec_insert()

```
void vec_insert (
            Vec * self,
            size_t index,
            ptr_t new_ele)
```

4.66.1.12 vec_erase()

```
void vec_erase (
            Vec * self,
            size_t index)
```

4.66.1.13 vec_resize()

```
void vec_resize (
            Vec * self,
            size_t new_capacity)
```

4.66.1.14 vec_clear()

```
void vec_clear (
            Vec * self)
```

4.66.1.15 vec_destroy()

```
void vec_destroy (
            Vec * self)
```

4.67 util/Vec.h File Reference

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

Data Structures

- struct vec_st

Macros

- #define vec_capacity(vec)
- #define vec_len(vec)
- #define vec_is_empty(vec)

Typedefs

- typedef void * ptr_t
- typedef void(* ptr_dtor_fn) (ptr_t)
- typedef struct vec_st Vec

Functions

- Vec vec_new (size_t initial_capacity, ptr_dtor_fn ele_dtor_fn)
- ptr_t vec_get (Vec *self, size_t index)
- void vec_set (Vec *self, size_t index, ptr_t new_ele)
- void vec_push_back (Vec *self, ptr_t new_ele)
- bool vec_pop_back (Vec *self)
- void vec_insert (Vec *self, size_t index, ptr_t new_ele)
- void vec_erase (Vec *self, size_t index)
- void vec_resize (Vec *self, size_t new_capacity)
- void vec_clear (Vec *self)
- void vec_destroy (Vec *self)

4.67.1 Macro Definition Documentation

4.67.1.1 vec_capacity

```
#define vec_capacity(
              vec)
```

Value:

```
((vec)->capacity)
```

4.67.1.2 vec_len

```
#define vec_len(
              vec)
```

Value:

```
((vec)->length)
```

4.67.1.3 vec_is_empty

```
#define vec_is_empty(
              vec)
```

Value:

```
((vec)->length == 0)
```

4.67.2 Typedef Documentation

4.67.2.1 ptr_t

```
typedef void* ptr_t
```

4.67.2.2 ptr_dtor_fn

```
typedef void(* ptr_dtor_fn) (ptr_t)
```

4.67.2.3 Vec

```
typedef struct vec_st Vec
```

4.67.3 Function Documentation

4.67.3.1 vec_new()

```
Vec vec_new (
            size_t initial_capacity,
            ptr_dtor_fn ele_dtor_fn)
```

Creates a new empty Vec(tor) with the specified initial_capacity and specified function to clean up elements in the vector.

Parameters

initial_capacity	the initial capacity of the newly created vector, non negative
ele_dtor_fn	a function pointer to a function that takes in a ptr_t (a vector element) and cleans it up. This is commonly just `free` but custom functions can be passed in. NULL can also be passed in to specify that there is no cleanup function that needs to be called on each element.

Returns

a newly created vector with specified capacity, 0 length and the specified element destructor (cleanup) function.

Postcondition

if memory allocation fails, the function will panic.

4.67.3.2 vec_get()

```
ptr_t vec_get (
            Vec * self,
            size_t index)
```

4.67.3.3 vec_set()

```
void vec_set (
            Vec * self,
            size_t index,
            ptr_t new_ele)
```

4.67.3.4 vec_push_back()

```
void vec_push_back (
            Vec * self,
            ptr_t new_ele)
```

4.67.3.5 vec_pop_back()

```
bool vec_pop_back (
            Vec * self)
```

4.67.3.6 vec_insert()

```
void vec_insert (
            Vec * self,
            size_t index,
            ptr_t new_ele)
```

4.67.3.7 vec_erase()

```
void vec_erase (
            Vec * self,
            size_t index)
```

4.67.3.8 vec_resize()

```
void vec_resize (
            Vec * self,
            size_t new_capacity)
```

4.67.3.9 vec_clear()

```
void vec_clear (
            Vec * self)
```

4.67.3.10 vec_destroy()

```
void vec_destroy (
            Vec * self)
```

4.68 Vec.h

Go to the documentation of this file.

```
00001 #ifndef VEC_H_
00002 #define VEC_H_
00003
00004 #include <stdbool.h>
00005 #include <stddef.h>  // for size_t
00006
00007 typedef void* ptr_t;
00008 typedef void (*ptr_dtor_fn)(ptr_t);
00009
00010 typedef struct vec_st {
00011   ptr_t* data;
00012   size_t length;
00013   size_t capacity;
00014   ptr_dtor_fn ele_dtor_fn;
00015 } Vec;
00016
00033 Vec vec_new(size_t initial_capacity, ptr_dtor_fn ele_dtor_fn);
00034
00035 /* Returns the current capacity of the Vec
00036  * Written as a function-like macro
00037  *
00038  * @param vec, a pointer to the vector we want to grab the capacity of.
00039  */
00040 // TODO: finish this macro
00041 #define vec_capacity(vec) ((vec)->capacity)
00042
00043 /* Returns the current length of the Vec
00044  * written as a function-like macro
00045  *
00046  * @param vec, a pointer to the vector we want to grab the len of.
00047  */
00048 #define vec_len(vec) ((vec)->length)
00049
00050 /* Checks if the Vec is empty
00051  * written as a function-like macro
00052  *
00053  * @param vec, a pointer to the vector we want to check emptiness of.
00054  */
00055 #define vec_is_empty(vec) ((vec)->length == 0)
00056
00057 /* Gets the specified element of the Vec
00058  *
00059  * @param self a pointer to the vector who's element we want to get.
00060  * @param index the index of the element to get.
00061  * @returns the element at the specified index.
00062  * @pre Assumes self points to a valid vector. If the index is >= self->length
00063  * then this function will panic()
00064  */
00065 ptr_t vec_get(Vec* self, size_t index);
00066
00067 /* Sets the specified element of the Vec to the specified value
00068  *
00069  * @param self    a pointer to the vector who's element we want to set.
00070  * @param index   the index of the element to set.
00071  * @param new_ele the value we want to set the element at that index to
00072  * @returns the element at the specified index.
00073  * @pre Assumes self points to a valid vector. If the index is >= self->length
00074  * then this function will panic()
00075  */
00076 void vec_set(Vec* self, size_t index, ptr_t new_ele);
00077
00078 /* Appends the given element to the end of the Vec
00079  *
00080  * @param self      a pointer to the vector we are pushing onto
00081  * @param new_ele   the value we want to add to the end of the container
00082  * @pre Assumes self points to a valid vector.
00083  * @post If a resize is needed and it fails, then this function will panic()
00084  * @post If after the operation the new length is greater than the old capacity
00085  * then a reallocation takes place and all elements are copied over.
00086  * Capacity is doubled. If initial capacity is zero, it is resized to
00087  * capacity 1. Any pointers to elements prior to this reallocation are
00088  * invalidated.
00089  */
00090 void vec_push_back(Vec* self, ptr_t new_ele);
00091
00092 /* Removes and destroys the last element of the Vec
00093  *
00094  * @param self a pointer to the vector we are popping.
00095  * @returns true iff an element was removed.
00096  * @pre Assumes self points to a valid vector.
00097  * @post The capacity of self stays the same. The removed element is
00098  * destructed (cleaned up) as specified by the dtor_fn provided in vec_new.
```

```
00099  */
00100 bool vec_pop_back(Vec* self);
00101
00102 /* Inserts an element at the specified location in the container
00103  *
00104  * @param self    a pointer to the vector we want to insert into.
00105  * @param index   the index of the element we want to insert at.
00106  *                Elements at this index and after it are "shifted" up
00107  *                one position. If index is equal to the length, then we insert
00108  *                at the end of the vector.
00109  * @param new_ele the value we want to insert
00110  * @pre Assumes self points to a valid vector. If the index is > self->length
00111  * then this function will panic().
00112  * @post If after the operation the new length is greater than the old capacity
00113  * then a reallocation takes place and all elements are copied over. Capacity is
00114  * doubled. Any pointers to elements prior to this reallocation are invalidated.
00115  */
00116 void vec_insert(Vec* self, size_t index, ptr_t new_ele);
00117
00118 /* Erases an element at the specified valid location in the container
00119  *
00120  * @param self    a pointer to the vector we want to erase from.
00121  * @param index   the index of the element we want to erase at. Elements
00122  *                after this index are "shifted" down one position.
00123  * @pre Assumes self points to a valid vector. If the index is >= self->length
00124  * then this function will panic().
00125  */
00126 void vec_erase(Vec* self, size_t index);
00127
00128 /* Resizes the container to a new specified capacity.
00129  * Does nothing if new_capacity <= self->length
00130  *
00131  * @param self         a pointer to the vector we want to resize.
00132  * @param new_capacity the new capacity of the vector.
00133  * @pre Assumes self points to a valid vector.
00134  * @post If a resize takes place, then a reallocation takes place and all
00135  * elements are copied over. Any pointers to elements prior to this
00136  * reallocation are invalidated.
00137  * @post The removed elements are destructed (cleaned up).
00138  */
00139 void vec_resize(Vec* self, size_t new_capacity);
00140
00141 /* Erases all elements from the container.
00142  * After this, the length of the vector is zero.
00143  * Capacity of the vector is unchanged.
00144  *
00145  * @param self a pointer to the vector we want to clear.
00146  * @pre Assumes self points to a valid vector.
00147  * @post The removed elements are destructed (cleaned up).
00148  */
00149 void vec_clear(Vec* self);
00150
00151 /* Destruct the vector.
00152  * All elements are destructed and storage is deallocated.
00153  * Must set capacity and length to zero. Data is set to NULL.
00154  *
00155  * @param self a pointer to the vector we want to destruct.
00156  * @pre Assumes self points to a valid vector.
00157  * @post The removed elements are destructed (cleaned up)
00158  * and data storage deallocated.
00159  */
00160 void vec_destroy(Vec* self);
00161
00162 #endif  // VEC_H_
```

Index