PennOS

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 file_descriptor Struct Reference

Creates an struct that contains information from a file inside a filesytem.

```
#include <file_descriptor.h>
```

Public Attributes

- int **fd**
- int **start_block**
- int **offset**
- int **file_size**
- int **perm**
- char ∗ **name**

3.1.1 Detailed Description

Creates an struct that contains information from a file inside a filesytem.

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

- src/FAT/file_descriptor.h

3.2 HashNode Struct Reference

Node structure for the hash table.

```
#include <hash.h>
```

Collaboration diagram for HashNode:

Public Attributes

- char ∗ command

  Command string.
- int index

  Index of the command in the hash table.
- command_function ∗ function

  Function pointer to the command's implementation.
- struct HashNode ∗ next

  Pointer to the next node in the chain.

3.2.1 Detailed Description

Node structure for the hash table.

Represents a single entry in the hash table, containing a command, its index, the corresponding function to execute, and a pointer to the next node in the chain for handling collisions.

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

- src/util/hash.h

3.3 LinkedList Struct Reference

Define the doubly linked list structure.

```
#include <linked_list.h>
```

Collaboration diagram for LinkedList:

Public Attributes

- Node ∗ head

  Pointer to the head of the list.
- Node ∗ tail

  Pointer to the tail of the list.
- size_t size

  Number of elements in the list.

3.3.1 Detailed Description

Define the doubly linked list structure.

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

- src/util/linked_list.h

3.4 Node Struct Reference

Define a node in the doubly linked list.

```
#include <linked_list.h>
```

Collaboration diagram for Node:

Public Attributes

- void ∗ data

  Pointer to data held in the node.
- struct Node ∗ next

  Pointer to the next node in the list.
- struct Node ∗ prev

  Pointer to the previous node in the list.
- int index

  Index of the node in the list.

3.4.1 Detailed Description

Define a node in the doubly linked list.

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

- src/util/linked_list.h

3.5 parsed_command Struct Reference

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

Public Attributes

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

3.5.1 Detailed Description

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

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

- src/shell/parser.h

3.6 PCB Struct Reference

Process Control Block (PCB) structure.

```
#include <PCB.h>
```

Collaboration diagram for PCB:

Public Attributes

- ucontext_t ∗ context

  Pointer to the ucontext information.
- int pid

  The thread's process ID.
- int parent_pid

  The parent process ID.
- LinkedList ∗ children

  Pointer to a linked list of children process IDs.
- LinkedList ∗ zombie_q

  Pointer to a linked list of zombie processes.
- LinkedList ∗ wait_q

  Pointer to a linked list of processes waiting on this process.
- int state

  The process state (e.g., RUNNING, STOPPED, BLOCKED).
- int priority

  The priority level of the process.
- int exit_status

  The exit status of the process.
- int waiting_on

  The process ID the current process is waiting on.
- bool state_change_checked

  Flag indicating if the state change has been checked.
- bool in_waitpid_neg_one_hanging

  Flag indicating if the process is in a waitpid(-1, ...) hanging state.
- char ∗ name

  The name or identifier of the process.
- int fd_in

  The file descriptor for stdin.
- int fd_out

  The file descriptor for stdout.

3.6.1 Detailed Description

Process Control Block (PCB) structure.

The PCB contains information about a thread or process, including its context, process ID, parent process ID, children process IDs, open file descriptors, priority level, and more.

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

- src/kernel/PCB.h

3.7 SchedulerContext Struct Reference

Structure for scheduler context.

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

Public Attributes

- ucontext_t ∗ context

  Pointer to the ucontext information.
- int restartFlag

  Flag indicating whether a restart is required.

3.7.1 Detailed Description

Structure for scheduler context.

This structure holds the context information for the scheduler, including the ucontext, restartFlag, and other state variables.

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

- src/kernel/scheduler.h

3.8 signal Struct Reference

Created by Omar Ameen on 11/10/23.

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

Public Attributes

- int sender_pid

  PID of the sender of the signal.
- int receiver_pid

  PID of the receiver of the signal.
- int signal

  Signal number.

3.8.1 Detailed Description

Created by Omar Ameen on 11/10/23.

Structure representing a signal in the system

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

- src/util/signal.h

3.9 timer Struct Reference

Structure for a timer associated with a process.

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

Collaboration diagram for timer:

Public Attributes

- unsigned int end_time

    Time at which the timer ends.
- PCB ∗ process

    Pointer to the process associated with the timer.

3.9.1 Detailed Description

Structure for a timer associated with a process.

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

- src/util/signal.h

3.10 waiting_process Struct Reference

Structure for processes waiting on a particular PID.

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

Collaboration diagram for waiting_process:

Public Attributes

- PCB ∗ pcb

    Pointer to the PCB of the waiting process.
- pid_t pid

    PID of the process being waited on.

3.10.1 Detailed Description

Structure for processes waiting on a particular PID.

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

- src/util/signal.h

Chapter 4

File Documentation

4.1 src/FAT/file_descriptor.h File Reference

Header file for a struct containing a file information and the creation of it.

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

Include dependency graph for file_descriptor.h:

Classes

- struct file_descriptor

  Creates an struct that contains information from a file inside a filesytem.

Typedefs

- typedef struct file_descriptor file_descriptor

  Creates an struct that contains information from a file inside a filesytem.

Functions

- file_descriptor ∗ fd_create (int fd, int start_block, int offset, int perm, char ∗name, int file_size)

  Creates a file_descriptor struct with the necessary information.

4.1.1 Detailed Description

Header file for a struct containing a file information and the creation of it.

4.1.2 Function Documentation

4.1.2.1 fd_create()

```
file_descriptor* fd_create (
            int fd,
            int start_block,
            int offset,
            int perm,
            char * name,
            int file_size )
```

Creates a file_descriptor struct with the necessary information.

Parameters

file_descriptor	int of an open file
start_block	int of a file inside a filesystem
offset	int for the current location of the information in a file inside the filesystem
permissions	int describing the current permission of the file
file_name	char pointer to the name of the file
file_size	int containing the size of the file

Returns

file_descriptor pointer on success with the information that was inputted and NULL on error

4.2 src/FAT/system-calls.h File Reference

Header file for system-related calls for file manipulation.

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

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

Enumerations

- enum mode { F_WRITE , F_READ , F_APPEND }

  Enum for different f_open modes.

Functions

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

  Open a file with the specified mode.
- int f_read (int fd, int n, char ∗buf)

  Read n bytes from the file referenced by fd.
- int f_write (int fd, const char ∗str, int n)

  Write n bytes of the string referenced by str to the file fd and increment the file pointer by n.

- int f_close (int fd)

  Close the file referenced by fd.
- int f_unlink (const char ∗fname)

  Remove a file, waiting if multiple processes are currently using it.
- off_t f_lseek (int fd, int offset, int whence)

  Reposition the file pointer for fd to the offset relative to whence.
- void f_ls (const char ∗filename)

  List the file filename in the current directory. If filename is NULL, list all files in the current directory.
- int f_touch (char ∗file_name)

  Create or update a file's timestamp.
- bool f_move (char ∗∗files)

  Helper function that will copy the name of the source to the destination.
- void f_chmod (char ∗∗file_name)

  Helper function that will change the permissions of a file.
- void mount (char ∗filename)

  Mount the current filesystem.

4.2.1 Detailed Description

Header file for system-related calls for file manipulation.

4.2.2 Enumeration Type Documentation

4.2.2.1 mode

```
enum mode
```

Enum for different f_open modes.

Enumerator

F_WRITE	Write mode
F_READ	Read mode
F_APPEND	Append mode

4.2.3 Function Documentation

4.2.3.1 f_chmod()

```
void f_chmod (
            char ** file_name )
```

Helper function that will change the permissions of a file.

Parameters

file_name	Char double pointer to the new permissions and the file that will be changed.

4.2.3.2 f_close()

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

Close the file referenced by fd.

Parameters

fd	File descriptor of the file to be closed.

Returns

0 on success, or a negative value on failure.

4.2.3.3 f_ls()

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

List the file filename in the current directory. If filename is NULL, list all files in the current directory.

Parameters

filename	Pointer to the name of the filesystem for which f_ls will be performed.

4.2.3.4 f_lseek()

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

Reposition the file pointer for fd to the offset relative to whence.

Parameters

fd	File descriptor of the file.
offset	Number of bytes the file pointer will be offset by.
whence	Whence will be performed (F_SEEK_SET, F_SEEK_CUR, F_SEEK_END).

Returns

Current offset of the file.

4.2.3.5 f_move()

```
bool f_move (
             char ** files )
```

Helper function that will copy the name of the source to the destination.

Parameters

files	Char double pointer to the names of the source file and the destination file.

Returns

True on success and false on failure.

4.2.3.6 f_open()

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

Open a file with the specified mode.

Parameters

fname	Constant char pointer to the filename to be opened.
mode	Mode in which the file will be opened (F_READ, F_WRITE, F_APPEND).

Returns

File descriptor on success, or a negative value on error.

4.2.3.7 f_read()

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

Read n bytes from the file referenced by fd.

Parameters

fd	File descriptor of the file to read from.
n	Number of bytes to be read.
buf	Pointer to a buffer where the read bytes should be stored.

Returns

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

4.2.3.8 f_touch()

```
int f_touch (
            char * file_name )
```

Create or update a file's timestamp.

If the file exists, the timestamp will be updated.

Parameters

file_name	Pointer to the name of the file to be created/updated.

Returns

0 on success, and -1 on failure.

4.2.3.9 f_unlink()

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

Remove a file, waiting if multiple processes are currently using it.

Parameters

fname	Pointer to the filename to be removed.

Returns

0 on success, or a negative value on failure.

4.2.3.10 f_write()

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

Write n bytes of the string referenced by str to the file fd and increment the file pointer by n.

Parameters

fd	File descriptor of the file to write to.
str	Pointer to a buffer containing the bytes to be written.
n	Number of bytes to be written.

Returns

Number of bytes written, or a negative value on error.

4.2.3.11 mount()

```
void mount (
            char * filename )
```

Mount the current filesystem.

Parameters

filename	Pointer to the filename to be mounted.

4.3 src/kernel/kernel.c File Reference

C file for kernel-level functions related to process management: k_process_kill(), k_process_cleanup(), k_process_create(), setStack(), cleanup_context().

```
#include "kernel.h"
#include "PCB.h"
#include <ucontext.h>
#include <valgrind/valgrind.h>
#include <stdlib.h>
#include <signal.h>
#include <math.h>
#include <stdio.h>
#include "../util/linked_list.h"
#include "scheduler.h"
#include "../util/logger.h"
#include <regex.h>
#include "../util/errno.h"
```

Include dependency graph for kernel.c:

Macros

- #define **STACK_SIZE** 819200
- #define **RNG_A** 16807
- #define **RNG_B** 2147483647

Functions

- void cleanup_orphans (LinkedList ∗children)

  Cleans up orphans(children) after a process exits/terminates right before the process is cleaned up.
- void setStack (stack_t ∗stack)

  Sets stack for a process's context.
- PCB ∗ k_process_create (PCB ∗parent)

  Creates a Process/PCB with the given parameters.
- int is_sleep_2 (const char ∗str)

  Checks whether a process is a sleep process based on its name.
- void k_process_kill (PCB ∗process, int signal)

  Sends/kills a process with the given signal.
- void k_process_cleanup (PCB ∗process)

  Cleans up a process.

Variables

- int **curr_pid** = 1

4.3.1 Detailed Description

C file for kernel-level functions related to process management: k_process_kill(), k_process_cleanup(), k_process_create(), setStack(), cleanup_context().

4.3.2 Function Documentation

4.3.2.1 k_process_cleanup()

```
void k_process_cleanup (
            PCB ∗ process )
```

Cleans up a process.

Cleans up resources of a terminated/finished thread.

4.3.2.2 k_process_create()

```
PCB* k_process_create (
            PCB * parent )
```

Creates a Process/PCB with the given parameters.

Creates a new child thread and associated PCB.

4.3.2.3 k_process_kill()

```
void k_process_kill (
            PCB * process,
            int signal )
```

Sends/kills a process with the given signal.

Kills the process referenced by `process` with the specified `signal`.

4.3.2.4 setStack()

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

Sets stack for a process's context.

Sets the stack for a process.

4.4 src/kernel/kernel.h File Reference

Header file for kernel-level functions related to process management.

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

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

Functions

- PCB * k_process_create (PCB *parent)

  Creates a new child thread and associated PCB.
- void k_process_kill (PCB *process, int signal)

  Kills the process referenced by `process` with the specified `signal`.
- void k_process_cleanup (PCB *process)

  Cleans up resources of a terminated/finished thread.
- ucontext_t * cleanup_context (PCB *process)

  Retrieves the cleanup context for a process.
- void setStack (stack_t *stack)

  Sets the stack for a process.

4.4.1 Detailed Description

Header file for kernel-level functions related to process management.

4.4.2 Function Documentation

4.4.2.1 cleanup_context()

```
ucontext_t* cleanup_context (
             PCB * process )
```

Retrieves the cleanup context for a process.

Parameters

process	Pointer to the PCB of the process.

Returns

Pointer to the cleanup context.

4.4.2.2 k_process_cleanup()

```
void k_process_cleanup (
             PCB * process )
```

Cleans up resources of a terminated/finished thread.

This function is called when a terminated/finished thread's resources need to be cleaned up. Cleanup may include freeing memory, setting the status of the child, etc.

Parameters

process	Pointer to the PCB of the terminated thread.

Cleans up resources of a terminated/finished thread.

4.4.2.3 k_process_create()

```
PCB* k_process_create (
             PCB * parent )
```

Creates a new child thread and associated PCB.

The new thread retains many properties of the parent. The function returns a reference to the new PCB.

Parameters

parent	Pointer to the parent PCB.

Returns

Pointer to the newly created child PCB.

Creates a new child thread and associated PCB.

4.4.2.4 k_process_kill()

```
void k_process_kill (
            PCB * process,
            int signal )
```

Kills the process referenced by `process` with the specified `signal`.

Parameters

process	Pointer to the PCB of the process to be terminated.
signal	The signal used to terminate the process.

Kills the process referenced by `process` with the specified `signal`.

4.4.2.5 setStack()

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

Sets the stack for a process.

Parameters

stack	Pointer to the stack structure.

Sets the stack for a process.

4.5 src/kernel/PCB.c File Reference

C file for Process Control Block (PCB) related definitions and functions.

```
#include "PCB.h"
#include "../util/linked_list.h"
#include <stdlib.h>
#include <signal.h>
#include <stdio.h>
#include "../util/errno.h"
```

Include dependency graph for PCB.c:

4.6 src/kernel/PCB.h File Reference

Header file for Process Control Block (PCB) related definitions and functions.

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

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

Classes

- struct PCB

  Process Control Block (PCB) structure.

Typedefs

- typedef struct Node Node

  Forward declaration of the Node struct.
- typedef struct LinkedList LinkedList

  Forward declaration of the LinkedList struct.
- typedef struct PCB **PCB**

Enumerations

- enum process_state {
  RUNNING = 1 , STOPPED = 0 , BLOCKED = -1 , ZOMBIE = -2 ,
  EXITED = -3 , TERMINATED = -4 }

  Enumerated type representing process states.

Functions

- PCB ∗ create_PCB (ucontext_t ∗context, int pid, int parent_pid, int state, int priority)

  Creates a new Process Control Block (PCB) with the specified parameters.
- PCB ∗ get_PCB_by_pid (LinkedList ∗list, int pid)

  Retrieves a PCB by its process ID from a linked list of PCBs.
- void free_PCB (PCB ∗pcb)

  Frees the memory occupied by a PCB.
- char ∗ getStateName (int state)

  Gets the name of a process state based on its state code.

4.6.1 Detailed Description

Header file for Process Control Block (PCB) related definitions and functions.

4.6.2 Enumeration Type Documentation

4.6.2.1 process_state

```
enum process_state
```

Enumerated type representing process states.

This enum defines various process states such as RUNNING, STOPPED, BLOCKED, ZOMBIE, and EXITED.

Enumerator

RUNNING	The process is running.
STOPPED	The process is stopped.
BLOCKED	The process is blocked.
ZOMBIE	The process is a zombie.
EXITED	The process has exited.
TERMINATED	The process has been terminated.

4.6.3 Function Documentation

4.6.3.1 create_PCB()

```
PCB* create_PCB (
             ucontext_t * context,
             int pid,
             int parent_pid,
             int state,
             int priority )
```

Creates a new Process Control Block (PCB) with the specified parameters.

Parameters

context	Pointer to the ucontext information.
pid	The process ID.
parent_pid	The parent process ID.
state	The initial state of the process.
priority	The priority level of the process.

Returns

Pointer to the newly created PCB.

4.6.3.2 free_PCB()

```
void free_PCB (
             PCB * pcb )
```

Frees the memory occupied by a PCB.

Parameters

pcb	Pointer to the PCB to be freed.

4.6.3.3 get_PCB_by_pid()

```
PCB* get_PCB_by_pid (
            LinkedList * list,
            int pid )
```

Retrieves a PCB by its process ID from a linked list of PCBs.

Parameters

list	Pointer to the linked list of PCBs.
pid	The process ID to search for.

Returns

Pointer to the PCB with the specified process ID if found, or NULL if not found.

4.6.3.4 getStateName()

```
char* getStateName (
            int state )
```

Gets the name of a process state based on its state code.

Parameters

state	The process state code.

Returns

A string representing the name of the process state.

Gets the name of a process state based on its state code.

4.7 src/kernel/scheduler.h File Reference

Header file for the scheduler module.

```
#include "PCB.h"
#include "../util/linked_list.h"
```

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

Classes

- struct SchedulerContext

  Structure for scheduler context.

Functions

- void scheduleProcess (PCB ∗process)

  Schedule a process for execution.
- PCB ∗ k_current_process ()

  Get a pointer to the currently executing process's PCB.
- int k_get_pq_num ()

  Get the priority queue number for the current process.
- LinkedList ∗ get_process_pq (PCB ∗pcb)

  Get the priority queue of a process.
- void timerService ()

  Timer service routine.

Variables

- SchedulerContext ∗ schedulerContext

  Pointer to the scheduler context.
- LinkedList ∗ pcb_list

  Pointer to the PCB list.
- LinkedList ∗ pq_neg_one

  Pointer to priority queue with priority -1.
- LinkedList ∗ pq_zero

  Pointer to priority queue with priority 0.
- LinkedList ∗ pq_one

  Pointer to priority queue with priority 1.
- LinkedList ∗ signals

  Pointer to the list of signals.
- ucontext_t ∗ shellContext

  Pointer to the shell context.
- PCB ∗ shellProcess

  Pointer to the shell process.
- PCB ∗ foregroundProcess

  Pointer to the foreground process.
- PCB ∗ currentProcess

  Pointer to the current process.
- const int centisecond

  Centisecond constant.
- LinkedList ∗ waiting_processes

  Pointer to the list of waiting processes.
- unsigned int curr_ticks

  Current ticks count.
- LinkedList ∗ timerQueue

  Pointer to the timer queue.

4.7.1 Detailed Description

Header file for the scheduler module.

4.7.2 Function Documentation

4.7.2.1 get_process_pq()

```
LinkedList* get_process_pq (
              PCB * pcb )
```

Get the priority queue of a process.

Parameters

pcb	Pointer to the PCB of the process.

Returns

Pointer to the priority queue of the process.

4.7.2.2 k_current_process()

```
PCB* k_current_process ( )
```

Get a pointer to the currently executing process's PCB.

Returns

Pointer to the PCB of the currently executing process.

4.7.2.3 k_get_pq_num()

```
int k_get_pq_num ( )
```

Get the priority queue number for the current process.

Returns

The priority queue number.

4.7.2.4 scheduleProcess()

```
void scheduleProcess (
              PCB * process )
```

Schedule a process for execution.

Parameters

process	Pointer to the PCB of the process to be scheduled.

4.8 src/kernel/user.c File Reference

C file for user-level functions and system calls.

```
#include "user.h"
#include <stdlib.h>
#include <ucontext.h>
#include "PCB.h"
#include "kernel.h"
#include "scheduler.h"
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include "../util/errno.h"
#include "../shell/shell-commands.h"
#include "../util/linked_list.h"
#include "../util/logger.h"
```

Include dependency graph for user.c:

Functions

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

    Forks a new thread that retains most attributes of the parent thread and executes a function with arguments.
- pid_t p_spawn_shell (void(∗func)(), int fd0, int fd1)

    Forks a new thread that retains most attributes of the parent thread and executes a function.
- bool W_WIFEXITED (int ∗wstatus)

    Check if the child process has exited.
- bool W_WIFSTOPPED (int ∗wstatus)

    Check if the child process has been stopped.
- bool W_WIFSIGNALED (int ∗wstatus)

    Check if the child process has been signaled.
- void **collectProcess** (PCB ∗process, int ∗wstatus)
- pid_t p_waitpid (pid_t pid, int ∗wstatus, bool nohang)

    Wait for a child process to change state.
- int p_kill (pid_t pid, int sig)

    Send a signal to a thread.
- void p_exit (void)

    Exit the current thread unconditionally.
- int p_nice (pid_t pid, int priority)

    Change the priority of a process.
- void p_sleep (unsigned int ticks)

    Sleep for a specified number of ticks.

Variables

- const int **MAX_CHAR_SIZE** = 10000

4.8.1 Detailed Description

C file for user-level functions and system calls.

4.8.2 Function Documentation

4.8.2.1 p_kill()

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

Send a signal to a thread.

This function sends the signal `sig` to the thread referenced by `pid`.

Parameters

pid	PID of the thread to send the signal to.
sig	Signal to send.

Returns

0 on success, or -1 on error.

4.8.2.2 p_nice()

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

Change the priority of a process.

Parameters

pid	PID of the process to change priority.
priority	New priority level.

Returns

0 on success, or -1 on error.

4.8.2.3 p_sleep()

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

Sleep for a specified number of ticks.

Parameters

ticks	Number of ticks to sleep.

4.8.2.4 p_spawn()

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

Forks a new thread that retains most attributes of the parent thread and executes a function with arguments.

This function creates a new thread that executes the function referenced by `func` with the argument array `argv`. `fd0` is the file descriptor for the input file, and `fd1` is the file descriptor for the output file.

Parameters

func	Pointer to the function to execute.
argv	Argument array for the function.
fd0	File descriptor for input.
fd1	File descriptor for output.

Returns

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

4.8.2.5 p_spawn_shell()

```
pid_t p_spawn_shell (
            void(*)() func,
            int fd0,
            int fd1 )
```

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

This function creates a new thread that executes the function referenced by `func`. `fd0` is the file descriptor for the input file, and `fd1` is the file descriptor for the output file.

Parameters

func	Pointer to the function to execute.
fd0	File descriptor for input.
fd1	File descriptor for output.

Returns

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

4.8.2.6 p_waitpid()

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

Wait for a child process to change state.

This function sets the calling thread as blocked (if `nohang` is false) until a child of the calling thread changes state. It is similar to Linux `waitpid(2)`. If `nohang` is true, `p_waitpid` does not block but returns immediately.

Parameters

pid	PID of the child process to wait for.
wstatus	Pointer to the status of the child process.
nohang	If true, do not block and return immediately.

Returns

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

4.8.2.7 W_WIFEXITED()

```
bool W_WIFEXITED (
             int * wstatus )
```

Check if the child process has exited.

Parameters

wstatus	Pointer to the status of the child process.

Returns

True if the child process has exited, false otherwise.

4.8.2.8 W_WIFSIGNALED()

```
bool W_WIFSIGNALED (
            int * wstatus )
```

Check if the child process has been signaled.

Parameters

wstatus	Pointer to the status of the child process.

Returns

True if the child process has been signaled, false otherwise.

4.8.2.9 W_WIFSTOPPED()

```
bool W_WIFSTOPPED (
            int * wstatus )
```

Check if the child process has been stopped.

Parameters

wstatus	Pointer to the status of the child process.

Returns

True if the child process has been stopped, false otherwise.

4.9 src/kernel/user.h File Reference

Header file for user-level functions and system calls.

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

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

Functions

- bool W_WIFEXITED (int ∗wstatus)

  Check if the child process has exited.
- bool W_WIFSTOPPED (int ∗wstatus)

  Check if the child process has been stopped.
- bool W_WIFSIGNALED (int ∗wstatus)

  Check if the child process has been signaled.
- pid_t p_spawn (void(∗func)(), char ∗argv[ ], int fd0, int fd1)

  Forks a new thread that retains most attributes of the parent thread and executes a function with arguments.
- pid_t p_spawn_shell (void(∗func)(), int fd0, int fd1)

  Forks a new thread that retains most attributes of the parent thread and executes a function.
- pid_t p_waitpid (pid_t pid, int ∗wstatus, bool nohang)

  Wait for a child process to change state.
- int p_kill (pid_t pid, int sig)

  Send a signal to a thread.
- void collectChild (PCB ∗child, int ∗wstatus)

  Collect information about a child process.
- void p_exit (void)

  Exit the current thread unconditionally.
- int p_nice (pid_t pid, int priority)

  Change the priority of a process.
- void p_sleep (unsigned int ticks)

  Sleep for a specified number of ticks.

4.9.1 Detailed Description

Header file for user-level functions and system calls.

4.9.2 Function Documentation

4.9.2.1 collectChild()

```
void collectChild (
            PCB ∗ child,
            int ∗ wstatus )
```

Collect information about a child process.

Parameters

child	Pointer to the PCB of the child process.
wstatus	Pointer to the status of the child process.

4.9.2.2 p_kill()

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

Send a signal to a thread.

This function sends the signal `sig` to the thread referenced by `pid`.

Parameters

pid	PID of the thread to send the signal to.
sig	Signal to send.

Returns

0 on success, or -1 on error.

4.9.2.3 p_nice()

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

Change the priority of a process.

Parameters

pid	PID of the process to change priority.
priority	New priority level.

Returns

0 on success, or -1 on error.

4.9.2.4 p_sleep()

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

Sleep for a specified number of ticks.

Parameters

ticks	Number of ticks to sleep.

4.9.2.5 p_spawn()

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

Forks a new thread that retains most attributes of the parent thread and executes a function with arguments.

This function creates a new thread that executes the function referenced by `func` with the argument array `argv`. `fd0` is the file descriptor for the input file, and `fd1` is the file descriptor for the output file.

Parameters

func	Pointer to the function to execute.
argv	Argument array for the function.
fd0	File descriptor for input.
fd1	File descriptor for output.

Returns

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

4.9.2.6 p_spawn_shell()

```
pid_t p_spawn_shell (
            void(*)() func,
            int fd0,
            int fd1 )
```

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

This function creates a new thread that executes the function referenced by `func`. `fd0` is the file descriptor for the input file, and `fd1` is the file descriptor for the output file.

Parameters

func	Pointer to the function to execute.
fd0	File descriptor for input.
fd1	File descriptor for output.

Returns

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

4.9.2.7 p_waitpid()

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

Wait for a child process to change state.

This function sets the calling thread as blocked (if `nohang` is false) until a child of the calling thread changes state. It is similar to Linux `waitpid(2)`. If `nohang` is true, `p_waitpid` does not block but returns immediately.

Parameters

pid	PID of the child process to wait for.
wstatus	Pointer to the status of the child process.
nohang	If true, do not block and return immediately.

Returns

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

4.9.2.8 W_WIFEXITED()

```
bool W_WIFEXITED (
            int * wstatus )
```

Check if the child process has exited.

Parameters

wstatus	Pointer to the status of the child process.

Returns

True if the child process has exited, false otherwise.

4.9.2.9 W_WIFSIGNALED()

```
bool W_WIFSIGNALED (
            int * wstatus )
```

Check if the child process has been signaled.

Parameters

wstatus	Pointer to the status of the child process.

Returns

True if the child process has been signaled, false otherwise.

4.9.2.10 W_WIFSTOPPED()

```
bool W_WIFSTOPPED (
            int * wstatus )
```

Check if the child process has been stopped.

Parameters

wstatus	Pointer to the status of the child process.

Returns

True if the child process has been stopped, false otherwise.

4.10 src/shell/shell-commands.c File Reference

This file implements all shell commands and built-in functions for the PennOS project. It includes functions like 'cat', 'sleep', 'echo', as well as unique commands like 'zombify' and 'orphanify'. The file also contains signal handlers that send signals to the foreground process.

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

```
#include <dirent.h>
#include <utime.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/stat.h>
#include "shell-commands.h"
#include "../kernel/scheduler.h"
#include "../kernel/user.h"
#include "../kernel/kernel.h"
#include "shell-parser.h"
#include "../util/logger.h"
#include "../util/errno.h"
#include "../FAT/system-calls.h"
```

Include dependency graph for shell-commands.c:

Functions

- void **sig_handler** (int signum)
- void register_sig_handlers ()

  Registers signal handlers that will be used by the shell.
- void s_cat (char ∗∗args)

  Concatenates and prints arguments.
- void s_sleep (char ∗∗args)

  Spawns a sleep process that will sleep for the specified duration.
- void s_busy (char ∗∗args)

  Spawns a busy process that will busy wait indefinitely.
- void s_echo (char ∗∗args)

  Prints arguments to the console.
- void list_directory (const char ∗path)

  Helper function: Lists the contents of the specified directory.
- void s_ls (char ∗∗args)

  Lists the contents of the specified directory.
- void touch_file (char ∗filename)

  Helper function: Updates the access and modification times of the specified file.
- void s_touch (char ∗∗args)

  Creates an empty file or updates timestamp of existing files, similar to 'touch' command in bash.
- void s_mv (char ∗∗args)

  Renames a file from source to destination, similar to 'mv' command in bash.
- void s_cp (char ∗∗args)

  Copies a file from source to destination, similar to 'cp' command in bash.
- void s_rm (char ∗∗args)

  Removes files, similar to 'rm' command in bash.
- void change_mode (mode_t ∗curr_mode, const char ∗changes)

  Helper function: Changes the mode of the specified file.
- void s_chmod (char ∗∗args)

  Changes file modes, similar to 'chmod' command in bash.
- void s_ps (char ∗∗args)

  Lists all processes on PennOS, displaying pid, ppid, and priority.
- void s_kill (char ∗∗args)

  Sends a signal to a process.
- void s_nice (char ∗∗args)

  Runs a command with a modified priority.

- void s_nice_pid (char ∗∗args)

  Adjusts the priority of a process.
- void s_man (char ∗∗args)

  Lists all available commands.
- void s_bg (char ∗∗args)

  Continues the last stopped job, or the job specified by job_id.
- void s_fg (char ∗∗args)

  Brings the last stopped or backgrounded job to the foreground, or the job specified by job_id.
- void s_jobs (char ∗∗args)

  Lists all jobs.
- void s_logout (char ∗∗args)

  Exits the shell.
- void zombie_child (char ∗∗args)

  A process is spawned with this ucontext function in s_zombify()
- void s_zombify (char ∗∗args)

  Spawns a child process that will become a zombie.
- void orphan_child (char ∗∗args)

  A process is spawned with this ucontext function in s_orphanify()
- void s_orphanify (char ∗∗args)

  Spawns a child process that will become an orphan.
- void idle ()

  Idle function for Idle Context.

4.10.1 Detailed Description

This file implements all shell commands and built-in functions for the PennOS project. It includes functions like 'cat', 'sleep', 'echo', as well as unique commands like 'zombify' and 'orphanify'. The file also contains signal handlers that send signals to the foreground process.

4.10.2 Function Documentation

4.10.2.1 idle()

```
void idle ( )
```

Idle function for Idle Context.

Spawns a child process that idle waits by calling sigsuspend until a signal is received.

4.10.2.2 register_sig_handlers()

```
void register_sig_handlers ( )
```

Registers signal handlers that will be used by the shell.

Registers signal handlers.

4.10.2.3 s_bg()

```
void s_bg (
            char ** args )
```

Continues the last stopped job, or the job specified by job_id.

Parameters

args	Optional job_id to be continued; if not provided, continues the last stopped job.

4.10.2.4 s_busy()

```
void s_busy (
            char ** args )
```

Spawns a busy process that will busy wait indefinitely.

Busy waits indefinitely.

4.10.2.5 s_cat()

```
void s_cat (
            char ** args )
```

Concatenates and prints arguments.

Parameters

args	Arguments to be concatenated.

4.10.2.6 s_chmod()

```
void s_chmod (
            char ** args )
```

Changes file modes, similar to 'chmod' command in bash.

Parameters

args	Arguments specifying file mode changes.

4.10.2.7 s_cp()

```
void s_cp (
            char ** args )
```

Copies a file from source to destination, similar to 'cp' command in bash.

Parameters

args	Array containing source and destination file names.

4.10.2.8 s_echo()

```
void s_echo (
            char ** args )
```

Prints arguments to the console.

Echoes arguments to the output, similar to 'echo' command in bash.

4.10.2.9 s_fg()

```
void s_fg (
            char ** args )
```

Brings the last stopped or backgrounded job to the foreground, or the job specified by job_id.

Parameters

args	Optional job_id to be brought to the foreground; if not provided, brings the last job to the foreground.

4.10.2.10 s_jobs()

```
void s_jobs (
            char ** args )
```

Lists all jobs.

Parameters

args	Arguments to be ignored.

4.10.2.11 s_kill()

```
void s_kill (
            char ** args )
```

Sends a signal to a process.

Sends a specified signal to specified processes, similar to '/bin/kill' in bash.

4.10.2.12 s_logout()

```
void s_logout (
            char ** args )
```

Exits the shell.

Exits the shell and shuts down PennOS.

4.10.2.13 s_ls()

```
void s_ls (
            char ** args )
```

Lists the contents of the specified directory.

Lists all files in the working directory, similar to 'ls -il' command in bash.

4.10.2.14 s_man()

```
void s_man (
            char ** args )
```

Lists all available commands.

Parameters

args	Arguments to be ignored.

4.10.2.15 s_mv()

```
void s_mv (
            char ** args )
```

Renames a file from source to destination, similar to 'mv' command in bash.

Parameters

args	Array containing source and destination file names.

4.10.2.16 s_nice()

```
void s_nice (
            char ** args )
```

Runs a command with a modified priority.

Sets the priority of the command and executes it.

4.10.2.17 s_nice_pid()

```
void s_nice_pid (
             char ** args )
```

Adjusts the priority of a process.

Adjusts the nice level of a process to a specified priority.

4.10.2.18 s_orphanify()

```
void s_orphanify (
             char ** args )
```

Spawns a child process that will become an orphan.

Parameters

args	Arguments to be ignored.

4.10.2.19 s_rm()

```
void s_rm (
             char ** args )
```

Removes files, similar to 'rm' command in bash.

Parameters

args	File names to be removed.

4.10.2.20 s_sleep()

```
void s_sleep (
             char ** args )
```

Spawns a sleep process that will sleep for the specified duration.

Spawns a sleep process.

4.10.2.21 s_touch()

```
void s_touch (
            char ** args )
```

Creates an empty file or updates timestamp of existing files, similar to 'touch' command in bash.

Parameters

args	File names to be created or updated.

4.10.2.22 s_zombify()

```
void s_zombify (
            char ** args )
```

Spawns a child process that will become a zombie.

Parameters

args	Arguments to be ignored.

4.11 src/shell/shell-parser.c File Reference

Implements the parsing and execution of shell commands for the PennOS project. This file contains the implementation of functions to read and parse shell commands, convert strings to uppercase, initialize shell commands, and identify commands.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "shell-parser.h"
#include "stress.h"
#include "../util/errno.h"
```

Include dependency graph for shell-parser.c:

Macros

- #define **SHELL_TOK_BUFSIZE** 64
- #define **SHELL_TOK_DELIM** " \t\r\n\a"

Functions

- char ∗ read_line (void)

  Reads a line of input from stdin and returns it.
- char ∗ to_upper (char ∗string)

  Converts a string to uppercase.
- void init_shell_commands (void)

  Initializes shell commands.

Variables

- command_function commands [ ]

  Array of function pointers to shell command functions.

4.11.1 Detailed Description

Implements the parsing and execution of shell commands for the PennOS project. This file contains the implementation of functions to read and parse shell commands, convert strings to uppercase, initialize shell commands, and identify commands.

4.11.2 Function Documentation

4.11.2.1 read_line()

```
char* read_line (
             void  )
```

Reads a line of input from stdin and returns it.

Reads a line from the standard input.

4.11.2.2 to_upper()

```
char* to_upper (
             char * string )
```

Converts a string to uppercase.

Parameters

string	The string to be converted.

Returns

A pointer to the uppercase string.

4.11.3 Variable Documentation

4.11.3.1 commands

```
commands
```

Initial value:

```
= {
        s_cat,
        s_sleep,
        s_busy,
        s_echo,
        s_ls,
        s_touch,
        s_mv,
        s_cp,
        s_rm,
        s_chmod,
        s_ps,
        s_kill,
        s_nice,
        s_nice_pid,
        s_man,
        s_bg,
        s_fg,
        s_jobs,
        s_logout,
        s_zombify,
        s_orphanify,
        (void (*)(char **))hang,
        (void (*)(char **))nohang,
        (void (*)(char **))recur
}
```

Array of function pointers to shell command functions.

4.12 src/shell/shell-parser.h File Reference

Defines the interface for shell command parsing in the PennOS project. It includes function prototypes for reading and processing shell commands, as well as the enumeration of supported shell commands.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../util/hash.h"
```

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

Typedefs

- typedef void(∗ **command_function**) (char ∗∗)
- typedef enum shell_command **shell_command**

Enumerations

- enum shell_command {
  CAT = 0 , **SLEEP** = 1 , **BUSY** = 2 , **ECHO** = 3 ,
  LS = 4 , **TOUCH** = 5 , **MV** = 6 , **CP** = 7 ,
  RM = 8 , **CHMOD** = 9 , **PS** = 10 , **KILL** = 11 ,
  NICE = 12 , **NICE_PID** = 13 , **MAN** = 14 , **BG** = 15 ,
  FG = 16 , **JOBS** = 17 , **LOGOUT** = 18 , **ZOMBIFY** = 19 ,
  ORPHANIFY = 20 , **HANG** = 21 , **NOHANG** = 22 , **RECUR** = 23 }

  Enumerates the available shell commands.

Functions

- char ∗ read_line (void)

  Reads a line from the standard input.
- void init_shell_commands (void)

  Initializes shell commands.
- int identify_command (char ∗command)

  Identifies a command from a given string.
- char ∗ to_upper (char ∗string)

  Converts a string to uppercase.

Variables

- command_function commands [ ]

  Array of function pointers to shell command functions.

4.12.1 Detailed Description

Defines the interface for shell command parsing in the PennOS project. It includes function prototypes for reading and processing shell commands, as well as the enumeration of supported shell commands.

4.12.2 Function Documentation

4.12.2.1 identify_command()

```
int identify_command (
             char ∗ command )
```

Identifies a command from a given string.

Parameters

command	The command string.

Returns

Integer representing the command.

4.12.2.2 read_line()

```
char* read_line (
            void  )
```

Reads a line from the standard input.

Returns

A pointer to the read line.

Reads a line from the standard input.

4.12.2.3 to_upper()

```
char* to_upper (
            char * string )
```

Converts a string to uppercase.

Parameters

string	The string to be converted.

Returns

A pointer to the uppercase string.

4.13 src/shell/shell.c File Reference

C file for the PennOS shell, containing functions for the shell.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
#include "shell.h"
#include "shell-parser.h"
#include "parser.h"
#include "../kernel/user.h"
#include "../kernel/scheduler.h"
#include "../util/logger.h"
#include "../util/errno.h"
#include "../FAT/system-calls.h"
```

Include dependency graph for shell.c:

Functions

- int **processLines** (char ∗buffer, long length)
- int execute_script (char ∗filename)

  Spawns processes by reading a script file line by line and executing each line as a command.
- void shell_loop ()

  Runs the shell loop that spawns commands and waits for them if the command is in the foreground.

4.13.1 Detailed Description

C file for the PennOS shell, containing functions for the shell.

4.13.2 Function Documentation

4.13.2.1 shell_loop()

```
void shell_loop (
             void  )
```

Runs the shell loop that spawns commands and waits for them if the command is in the foreground.

Runs the shell loop.

4.14 src/shell/shell.h File Reference

Header file for the PennOS shell, containing function declarations for the shell.

```
#include "parser.h"
```
Include dependency graph for shell.h: This graph shows which files directly or indirectly include this file:

Functions

- void shell_loop (void)

  Runs the shell loop.

4.14.1 Detailed Description

Header file for the PennOS shell, containing function declarations for the shell.

4.14.2 Function Documentation

4.14.2.1 shell_loop()

```
void shell_loop (
             void  )
```

Runs the shell loop.

Runs the shell loop.

4.15 src/shell/stress.h File Reference

Header file for stress test commands, including hang, nohang, and recur.

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

Functions

- void hang (void)

  Hang command that spawns 10 children and block-waits for any of them until all are reaped.
- void nohang (void)

  Nohang command that spawns 10 children and nonblocking waits for any of them until all are reaped.
- void recur (void)

  Recur command that recursively spawns itself 26 times, naming the spawned processes Gen_A through Gen_Z.

4.15.1 Detailed Description

Header file for stress test commands, including hang, nohang, and recur.

4.15.2 Function Documentation

4.15.2.1 hang()

```
void hang (
             void  )
```

Hang command that spawns 10 children and block-waits for any of them until all are reaped.

This command spawns 10 children and block-waits for any of them in a loop until all of them are reaped. Note that the order in which the children get reaped is non-deterministic.

4.15.2.2 nohang()

```
void nohang (
             void )
```

Nohang command that spawns 10 children and nonblocking waits for any of them until all are reaped.

This command spawns 10 children and nonblocking waits for any of them in a loop until all of them are reaped. Note that the order in which the children get reaped is non-deterministic.

4.15.2.3 recur()

```
void recur (
             void )
```

Recur command that recursively spawns itself 26 times, naming the spawned processes Gen_A through Gen_Z.

Each process is block-waited and reaped by its parent.

4.16 src/util/hash.c File Reference

Implementation of hash table functions for command processing.

```
#include "hash.h"
#include <ctype.h>
#include "../shell/shell-parser.h"
#include "../util/errno.h"
```

Include dependency graph for hash.c:

Functions

- unsigned int hash (char ∗str)

  Hash function for command strings.
- void insert (char ∗command, int function_index)

  Inserts a command into the hash table.
- int lookup (char ∗command)

  Looks up a command in the hash table.

Variables

- HashNode ∗ hashTable [TABLE_SIZE]

  Global hash table array.

4.16.1 Detailed Description

Implementation of hash table functions for command processing.

This file contains the implementations of the functions defined in hash.h. It includes the creation and manipulation of a hash table used for storing and retrieving command-related data efficiently. The functions provide capabilities to hash strings, insert commands into the hash table, and look up commands based on their string representation.

4.16.2 Function Documentation

4.16.2.1 hash()

```
unsigned int hash (
             char * str )
```

Hash function for command strings.

Computes the hash value for a given command string.

Parameters

str	The command string to hash.

Returns

The computed hash value.

4.16.2.2 insert()

```
void insert (
             char * command,
             int index )
```

Inserts a command into the hash table.

Adds a new command and its associated function to the hash table.

Parameters

command	The command string.
index	The index in the hash table.

4.16.2.3 lookup()

```
int lookup (
             char * command )
```

Looks up a command in the hash table.

Searches for a command in the hash table and returns its index if found.

Parameters

command	The command string to look up.

Returns

The index of the command in the hash table or -1 if not found.

4.17 src/util/hash.h File Reference

Header file for hash table implementation in PennOS.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../shell/shell-commands.h"
```

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

Classes

- struct HashNode

  Node structure for the hash table.

Macros

- #define TABLE_SIZE 100

  Size of the hash table.

Typedefs

- typedef void(∗ command_function) (char ∗∗)

  Type definition for a command function.
- typedef struct HashNode HashNode

  Node structure for the hash table.

Functions

- unsigned int hash (char ∗str)

  Hash function for command strings.
- void insert (char ∗command, int index)

  Inserts a command into the hash table.
- int lookup (char ∗command)

  Looks up a command in the hash table.

Variables

- HashNode ∗ hashTable [TABLE_SIZE]

    Global hash table array.

4.17.1 Detailed Description

Header file for hash table implementation in PennOS.

Defines structures and functions for managing a hash table used in PennOS, primarily for command processing and lookup. Created by Omar Ameen on 11/14/23.

4.17.2 Typedef Documentation

4.17.2.1 HashNode

```
typedef struct HashNode HashNode
```

Node structure for the hash table.

Represents a single entry in the hash table, containing a command, its index, the corresponding function to execute, and a pointer to the next node in the chain for handling collisions.

4.17.3 Function Documentation

4.17.3.1 hash()

```
unsigned int hash (
             char * str )
```

Hash function for command strings.

Computes the hash value for a given command string.

Parameters

str	The command string to hash.

Returns

The computed hash value.

4.17.3.2 insert()

```
void insert (
            char * command,
            int index )
```

Inserts a command into the hash table.

Adds a new command and its associated function to the hash table.

Parameters

command	The command string.
index	The index in the hash table.

4.17.3.3 lookup()

```
int lookup (
            char * command )
```

Looks up a command in the hash table.

Searches for a command in the hash table and returns its index if found.

Parameters

command	The command string to look up.

Returns

The index of the command in the hash table or -1 if not found.

4.18 src/util/linked_list.c File Reference

Implementation of linked list functions.

```
#include "linked_list.h"
#include <stdlib.h>
#include <stdbool.h>
#include "errno.h"
#include <stdio.h>
```

Include dependency graph for linked_list.c:

Functions

- LinkedList * create_list ()

  Initialize the doubly linked list.

- Node ∗ create_node (void ∗data)

  Create a new node with the given data.
- void insert_front (LinkedList ∗list, void ∗data)

  Insert a node at the front of the list.
- void insert_end (LinkedList ∗list, void ∗data)

  Insert a node at the end of the list.
- void insert_before (LinkedList ∗list, Node ∗referenceNode, Node ∗newNode)

  Insert a node before the referenceNode.
- void delete_node (LinkedList ∗list, Node ∗node)

  Delete a node from the list.
- Node ∗ **get_fd_node** (LinkedList ∗list, int data)
- Node ∗ get_node (LinkedList ∗list, void ∗data)

  Find a node in the list with the given data.
- Node ∗ get_node_int (LinkedList ∗list, int pid)

  Find a node in the list with given pid int.
- Node ∗ get_node_by_index (LinkedList ∗list, int index)

  Get the node at the given index.
- void free_list (LinkedList ∗list)

  Free the list and all its nodes.
- bool move_head_to_end (LinkedList ∗list)

  Move the head to the end of the list.
- int get_index (LinkedList ∗list, void ∗data)

  Find a node in the list with the given data and return its index.

4.18.1 Detailed Description

Implementation of linked list functions.

This file contains the implementation of functions declared in linked_list.h. It includes functions for creating and manipulating a doubly linked list, such as adding, removing, and searching for nodes, as well as managing the list itself.

4.19 src/util/logger.c File Reference

Implementation of logging functions for PennOS.

```
#include <stdio.h>
#include "logger.h"
#include "../kernel/scheduler.h"
#include "../util/errno.h"
```
Include dependency graph for logger.c:

Functions

- void write_log_nice (logtype log_type, pid_t pid, int oldprio, int newprio, char ∗process_name)

  Writes a 'nice' log entry, logging priority changes.
- FILE ∗ log_file ()

  Opens the log file for appending.
- void write_log (logtype log_type, pid_t pid, int priority, char ∗process_name)

  Writes a general log entry based on the specified log type.

4.19.1 Detailed Description

Implementation of logging functions for PennOS.

Created by Omar Ameen on 11/19/23. This file contains the implementation of logging functions defined in logger.h. It includes functions to write different types of log entries to a log file.

4.19.2 Function Documentation

4.19.2.1 log_file()

```
FILE* log_file ( )
```

Opens the log file for appending.

Opens the log file.

4.19.2.2 write_log()

```
void write_log (
             logtype log_type,
             pid_t pid,
             int priority,
             char * process_name )
```

Writes a general log entry based on the specified log type.

Writes a log entry.

4.20 src/util/signal.c File Reference

Implementation of signal handling functions for PennOS.

```
#include <stdlib.h>
#include "signal.h"
#include "linked_list.h"
#include "../util/errno.h"
#include <stdio.h>
```

Include dependency graph for signal.c:

Functions

- signal_t ∗ add_signal (int sender_pid, int receiver_pid, int signal)
  Adds a signal to the signal queue.
- void add_timer (PCB ∗pcb, unsigned int ticks)
  Adds a timer for a process to expire after a certain number of ticks.

4.20.1 Detailed Description

Implementation of signal handling functions for PennOS.

Created by Omar Ameen on 11/10/23. This file contains the implementations of the functions defined in signal.h. It includes functions for adding signals, waiters, and timers, as well as retrieving waiting processes based on PID or PCB pointers.