PennOS - Group 33

Generated by Doxygen 1.8.20

Chapter 1

CIS380 Group 33 Final Project: PennOS

Daksh Chhokra - File System

Joan Shaho - Kernel

Shreyas Sonbarse - Kernel

Kyven Wu - File System

1.1 Submitted files:

```
INSERT TREE HERE
```

1.2 Compile Instructions

Run `make` to compile both PennFAT and PennOS. Alternatively, `make pennfat` and `make pennos` compiles their respective binaries.

```
The binaries are compiled in the "`bin/"` folder.
## Work Accomplished
### PennFAT: Standalone FAT Filesystem
The standalone filesystem is completely functional and has no memory leaks.
It supports all of the commands specified in the project specification document:
```

mkfs FS_NAME BLOCKS_IN_FAT BLOCK_SIZE_CONFIG mount FS_NAME umount touch FILE ... mv SOURCE DEST rm FILE ... cat FILE ... [-w OUTPUT_FILE] cat FILE ... [-a OUTPUT_FILE] cat -w OUTPUT_FILE cat -a OUTPUT_FILE cp [-h] SOURCE DEST cp SOURCE [-h] DEST ls

```
Additionally, it supports the "`describe"` command which prints out additional information about the
       currently mounted file system.
### PennOS
PennOS is completely functional in terms of creating a kerner, scheduler, and the main shell process.
Additionally, the shell process supports job control as well as redirections. It supports all of the
       commands specified in the project specification document:
```

zombify orphanify man bg [job_id] fg [job_id] jobs logout cat sleep n busy ls touch file ... mv src dest cp src dest rm file ... ps

```
Note that from the commands above the following:
```

zombify orphanify cat sleep n busy touch file ... mv src dest cp src dest rm file ... ps

```
run as independent processes while the remaining ones are shell subroutines.
## Code Layout
At the base level, files are divided in four subdirectories of "`/src/"`, "`/src/include/, /src/fs/,
       /src/pennfat/, /src/pennos"`:
```

/src/include/ general files shared between multiple directories, e.g. macros.h /src/fs/ files directly involved in filesystem implementation /src/pennfat/ files that facilitate the PennFAT standalone program /src/pennos/ files that facilitate PennOS

```
Some additional directories are:
```

/bin all the binary files generated /log/log.txt the log file ```

1.3 Other Comments

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 childTag Struct Reference

`#include <PCB.h>`

Public Attributes

- pid_t **pid**
- struct childTag * **prev**
- struct childTag * **next**

4.1.1 Detailed Description

Process Control Block child object

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

- src/pennos/PCB.h

4.2 directoryEntryNodeType Struct Reference

`#include <fat.h>`

Public Attributes

- directoryEntry * **entry**
- struct directoryEntryNodeType * **next**

4.2.1 Detailed Description

A linked list node containing a directory entry

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

- src/fs/fat.h

4.3 directoryEntryType Struct Reference

`#include <fat.h>`

Public Attributes

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

4.3.1 Detailed Description

A directory entry in PennFAT, encompassing 64 bytes and can be directly written into the FAT file on disk

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

- src/fs/fat.h

4.4 fatType Struct Reference

`#include <fat.h>`

Public Attributes

- char * **fileName**
- uint8_t **numBlocks**
- uint32_t **blockSize**
- uint32_t **numEntries**
- uint32_t **freeBlocks**
- uint32_t **fileCount**
- directoryEntryNode * **firstDirectoryEntryNode**
- directoryEntryNode * **lastDirectoryEntryNode**
- uint16_t * **blocks**

4.4.1 Detailed Description

FAT structure loaded and stored in memory

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

- src/fs/fat.h

4.5 FileDescriptorContainerType Struct Reference

`#include <filedescriptor.h>`

Public Attributes

- fdNode * **firstFdNode**
- fdNode * **lastFdNode**

4.5.1 Detailed Description

Stores file descriptors

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

- src/pennos/filedescriptor.h

4.6 fileDescriptorNodeType Struct Reference

`#include <filedescriptor.h>`

Public Attributes

- int **id**
- directoryEntry * **entry**
- int **mode**
- int **pos**
- struct fileDescriptorNodeType * **next**

4.6.1 Detailed Description

A file descriptor node

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

- src/pennos/filedescriptor.h

4.7 fileType Struct Reference

`#include <file.h>`

Public Attributes

- uint8_t * **bytes**
- unsigned int **len**
- uint8_t **type**
- uint8_t **perm**

4.7.1 Detailed Description

File struct containing an array of bytes and the length of that array

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

- src/fs/file.h

4.8 jobQueue Struct Reference

`#include <jobQueue.h>`

Public Attributes

- int **count**
- job * **front**
- job * **back**

4.8.1 Detailed Description

A FIFO jobQueue the function caller is responsible for freeing the jobQueue pointer after calling jobQueueInit() the function caller is responsible for freeing job pointers from jobQueuePop() and jobQueueRemoveJob()

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

- src/pennos/jobQueue.h

4.9 jobTag Struct Reference

`#include <job.h>`

Public Attributes

- int **jobId**
- int **pgId**
- char * **jobDesc**
- int **commandCount**
- int **processesFinished**
- int * **pids**
- int(* **pipes**)[2]
- int **infile**
- int **outfile**
- bool **isRunning**
- struct jobTag * **prev**
- struct jobTag * **next**

4.9.1 Detailed Description

A job with pointers to next/prev jobs to allow a linked-list of jobs

Each job has a jobId, a pgId (initially -1), a job description, and its input/output file descriptors. Every job is also responsible for mallocing and freeing an array of pids (for each command) and mallocing, closing, and freeing the array of pipes that it uses in its pipeline.

Jobs are passed to handleJob() [commandCount] number of times, in sequence, and pgId is initialized the first time it is passed. Each handleJob() call lazily creates pipes and assigns pids to each command in the job.

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

- src/pennos/job.h

4.10 nodeTag Struct Reference

`#include <node.h>`

Public Attributes

- pcb_t * **pcb**
- pid_t **pid**
- struct nodeTag * **prev**
- struct nodeTag * **next**

4.10.1 Detailed Description

A node with pointers to next/prev nodes to allow a linked-list of nodes

Each node has a pid corresponding to some process

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

- src/pennos/node.h

4.11 pcbType Struct Reference

`#include <PCB.h>`

Public Attributes

- ucontext_t **context**
- int **status**
- int **prevStatus**
- int **priority_level**
- pid_t **pid**
- pid_t **ppid**
- pid_t **pgid**
- int **ticksLeft**
- bool **waitedOn**
- child * **child_pids**
- child * **zombies**
- char * **name**
- int **stdin**
- int **stdout**

4.11.1 Detailed Description

A process control block, which is used by the kernel to context switch to and from a particular process, send signals to a process or process group, etc.

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

- src/pennos/PCB.h

4.12 queue Struct Reference

`#include <queue.h>`

Public Attributes

- int **count**
- node * **front**
- node * **back**

4.12.1 Detailed Description

a FIFO node queue the function caller is responsible for freeing the queue pointer after calling queueInit() the function caller is responsible for freeing node pointers from queuePop() and queueRemovenode()

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

- src/pennos/queue.h

4.13 scheduler Struct Reference

Public Attributes

- int **quantaCount**
- queue ∗ **high**
- queue ∗ **med**
- queue ∗ **low**

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

- src/pennos/scheduler.h

Chapter 5

File Documentation

5.1 src/fs/fat.h File Reference

Alongside file.h, contains a FAT filesystem implementation along with functions for creating, saving and loading a FAT.

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

Classes

- struct directoryEntryType
- struct directoryEntryNodeType
- struct fatType

Typedefs

- typedef struct directoryEntryType directoryEntry
- typedef struct directoryEntryNodeType directoryEntryNode
- typedef struct fatType fat

Functions

- directoryEntryNode * newDirectoryEntryNode (char *fileName, uint32_t size, uint16_t firstBlock, uint8_t type, uint8_t perm, time_t time)

  Creates a new directory entry node.
- void freeDirectoryEntryNode (directoryEntryNode *node)

  Frees a directory entry node.
- fat * getFat (char *fileName, uint8_t numBlocks, uint8_t blockSizeIndicator, bool creating)

  Makes a PennFAT filesystem.
- fat * loadFat (char *fileName)

  Loads a PennFAT filesystem from disk.
- int saveFat (fat *fat)

  Saves a PennFAT filesystem to disk.
- void freeFat (fat **fat)

  Frees the FAT from memory and ensures the handler is NULL.

5.1.1 Detailed Description

Alongside file.h, contains a FAT filesystem implementation along with functions for creating, saving and loading a FAT.

5.1.2 Typedef Documentation

5.1.2.1 directoryEntry

```
typedef struct directoryEntryType directoryEntry
```

A directory entry in PennFAT, encompassing 64 bytes and can be directly written into the FAT file on disk

5.1.2.2 directoryEntryNode

```
typedef struct directoryEntryNodeType directoryEntryNode
```

A linked list node containing a directory entry

5.1.2.3 fat

```
typedef struct fatType fat
```

FAT structure loaded and stored in memory

5.1.3 Function Documentation

5.1.3.1 freeDirectoryEntryNode()

```
void freeDirectoryEntryNode (
            directoryEntryNode * node )
```

Frees a directory entry node.

Parameters

node	The directory entry node to free

5.1.3.2 freeFat()

```
void freeFat (
            fat ** fat )
```

Frees the FAT from memory and ensures the handler is NULL.

Parameters

fat	Pointer to the FAT pointer

5.1.3.3 getFat()

```
fat* getFat (
            char * fileName,
            uint8_t numBlocks,
            uint8_t blockSizeIndicator,
            bool creating )
```

Makes a PennFAT filesystem.

Parameters

in	*fileName*	The filename
in	*numBlocks*	The number of blocks in this FAT (1 - 32)
in	*blockSizeIndicator*	The size of each block (1 -> 512 bytes, 2 -> 1024 bytes, 3 -> 2048 bytes, 4 -> 4096 bytes)
in	*creating*	Whether or not we are creating a new FAT / overwriting existing FAT on disk

Returns

A pointer to the FAT stored in memory

5.1.3.4 loadFat()

```
fat* loadFat (
            char * fileName )
```

Loads a PennFAT filesystem from disk.

Parameters

fileName	The filename

Returns

A pointer to the loaded FAT stored in memory

5.1.3.5 newDirectoryEntryNode()

```
directoryEntryNode* newDirectoryEntryNode (
            char * fileName,
            uint32_t size,
            uint16_t firstBlock,
            uint8_t type,
            uint8_t perm,
            time_t time )
```

Creates a new directory entry node.

Parameters

	fileName	The file name
in	*size*	The size
in	*firstBlock*	The first block
in	*type*	The type
in	*perm*	The permission
in	*time*	The time

Returns

Pointer to a new directory entry node

5.1.3.6 saveFat()

```
int saveFat (
            fat * fat )
```

Saves a PennFAT filesystem to disk.

Parameters

fat	The FAT

Returns

SUCCESS on successful save, FAILURE when syscalls fail or unable to write to disk

5.2 src/fs/file.h File Reference

Contains a file struct and functions that allow users to read, write, and modify files in a FAT filesystem and additional functions for interacting with a PennFAT filesystem.

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

Classes

- struct fileType

Typedefs

- typedef struct fileType file

Functions

- void freeFile (file *file)

  Frees a file struct pointer and its bytes.
- void getEntryNodeAndPrev (directoryEntryNode **prev, directoryEntryNode **found, char *fileName, fat *fat)

  Find a directory entry node for a certain filename.
- file * getDirectoryFile (fat *fat)

  Helper function to get the root directory file in a FAT.
- uint8_t * getBytes (uint16_t startIndex, uint32_t length, fat *fat)

  Helper function to get the bytes of a file starting at some index.
- file * readFileFromFAT (char *fileName, fat *fat)

  Reads a file as byte pointers.
- int deleteFileFromFAT (char *fileName, fat *fat, bool syscall)

  Delete a file from a FAT filesystem.
- int renameFile (char *oldFileName, char *newFileName, fat *fat)

  Rename a file and update the last modified time.
- int writeFileToFAT (char *fileName, uint8_t *bytes, uint32_t offset, uint32_t length, uint8_t type, uint8_t perm, fat *fat, bool appending, bool syscall, bool writeDir)

  Writes (or overwrites) a file in a FAT filesystem.
- int appendToFileInFAT (char *fileName, uint8_t *bytes, uint32_t length, fat *fat, bool syscall)

  Appends to file in a FAT filesystem.
- int writeDirectoryFile (fat *fat)

  Writes the directory file.
- int chmodFile (fat *fat, char *fileName, int newPerms)

  Changes the access permissions of the file.

5.2.1 Detailed Description

Contains a file struct and functions that allow users to read, write, and modify files in a FAT filesystem and additional functions for interacting with a PennFAT filesystem.

5.2.2 Typedef Documentation

5.2.2.1 file

```
typedef struct fileType file
```

File struct containing an array of bytes and the length of that array

5.2.3 Function Documentation

5.2.3.1 appendToFileInFAT()

```
int appendToFileInFAT (
            char * fileName,
            uint8_t * bytes,
            uint32_t length,
            fat * fat,
            bool syscall )
```

Appends to file in a FAT filesystem.

Parameters

	fileName	The file name of the file to append to
	bytes	The bytes to append
in	*length*	The number of bytes to append
	fat	The FAT filesystem
	syscall	Whether or not this call is allowed to modify files no matter the permissions

Returns

-1 (FAILURE) on failure, 0 (SUCCESS) on success

5.2.3.2 chmodFile()

```
int chmodFile (
            fat * fat,
            char * fileName,
            int newPerms )
```

Changes the access permissions of the file.

Parameters

	fat	The fat
	fileName	The file name
in	*newPerms*	The new permissions

Returns

-1 (FAILURE) on failure, 0 (SUCCESS) on success

5.2.3.3 deleteFileFromFAT()

```
int deleteFileFromFAT (
            char * fileName,
            fat * fat,
            bool syscall )
```

Delete a file from a FAT filesystem.

Parameters

fileName	The file name
fat	The FAT
syscall	Whether or not this call is allowed to modify files no matter the permissions

Returns

-1 (FAILURE) on failure, 0 (SUCCESS) on success

5.2.3.4 freeFile()

```
void freeFile (
            file * file )
```

Frees a file struct pointer and its bytes.

Parameters

file	The file

5.2.3.5 getBytes()

```
uint8_t* getBytes (
            uint16_t startIndex,
            uint32_t length,
            fat * fat )
```

Helper function to get the bytes of a file starting at some index.

Parameters

in	*startIndex*	The start index of the file
	length	The length of the file
	fat	The FAT

Returns

A null-terminated byte array representing the file

5.2.3.6 getDirectoryFile()

```
file* getDirectoryFile (
            fat * fat )
```

Helper function to get the root directory file in a FAT.

Parameters

fat	The fat

Returns

The root directory file

5.2.3.7 getEntryNodeAndPrev()

```
void getEntryNodeAndPrev (
            directoryEntryNode ** prev,
```

```
            directoryEntryNode ** found,
            char * fileName,
            fat * fat )
```

Find a directory entry node for a certain filename.

Parameters

prev	Pointer to the pointer that will store the node before the found node, pass NULL if unused
found	Pointer to the pointer that will store the found node, pass NULL if unused
fileName	The file name to search for
fat	The FAT

5.2.3.8 readFileFromFAT()

```
file* readFileFromFAT (
            char * fileName,
            fat * fat )
```

Reads a file as byte pointers.

Parameters

fileName	The file name to write to
fat	The FAT
readDir	Whether or not we are reading the directory file – only called by the system

Returns

Returns a pointer to the null-terminated array of bytes

5.2.3.9 renameFile()

```
int renameFile (
            char * oldFileName,
            char * newFileName,
            fat * fat )
```

Rename a file and update the last modified time.

Parameters

oldFileName	The old file name
newFileName	The new file name
fat	The FAT

Returns

SUCCESS on successful rename, FAILURE when no file is found or newFileName already exists

5.2.3.10 writeDirectoryFile()

```
int writeDirectoryFile (
            fat * fat )
```

Writes the directory file.

Parameters

fat	The FAT filesystem

Returns

-1 (FAILURE) on failure, 0 (SUCCESS) on success

5.2.3.11 writeFileToFAT()

```
int writeFileToFAT (
            char * fileName,
            uint8_t * bytes,
            uint32_t offset,
            uint32_t length,
            uint8_t type,
            uint8_t perm,
            fat * fat,
            bool appending,
            bool syscall,
            bool writeDir )
```

Writes (or overwrites) a file in a FAT filesystem.

Parameters

	fileName	The file name of the file to write to
	bytes	The bytes to write
	offset	The byte offset to start writing at if not in append mode
in	*length*	The number of bytes to write
	type	The type of the file
	perm	The permissions of this file
	fat	The FAT filesystem
	appending	Whether or not to append to the file
	syscall	Whether or not this call is allowed to modify files no matter the permissions
	writeDir	Only used with system calls – allows writing to the directory file

Returns

-1 (FAILURE) on failure, 0 (SUCCESS) on success

5.3 src/include/macros.h File Reference

Useful macros for code readability.

```
#include "p_errno.h"
```

Macros

- #define **SUCCESS** 0
- #define **FAILURE** -1
- #define **RESET_ERRNO** p_errno = 0;
- #define **PENNOS_PROMPT** "penn-os> "
- #define **PENNOS_PROMPT_LENGTH** 9
- #define **PENNFAT_PROMPT** "pennfat> "
- #define **PENNFAT_PROMPT_LENGTH** 9
- #define **DIRECTORY_FILENAME** "/"
- #define **UNKNOWN_FILETYPE** 0
- #define **REGULAR_FILETYPE** 1
- #define **DIRECTORY_FILETYPE** 2
- #define **SYMLINK_FILETYPE** 4
- #define **NONE_PERMS** 0
- #define **WRITE_PERMS** 2
- #define **READ_PERMS** 4
- #define **READWRITE_PERMS** 6
- #define **S_SIGSTOP** 0
- #define **S_SIGCONT** 1
- #define **S_SIGTERM** 2
- #define **W_WIFEXITED**(status) (status == EXITED)
- #define **W_WIFSTOPPED**(status) (status == STOPPED)
- #define **W_WIFSIGNALED**(status) (status == SIGNALED)
- #define **F_WRITE** 0
- #define **F_READ** 1
- #define **F_APPEND** 2
- #define **F_SEEK_SET** 0
- #define **F_SEEK_CUR** 1
- #define **F_SEEK_END** 2
- #define **READY** 0
- #define **BLOCKED** 1
- #define **STOPPED** 2
- #define **SIGNALED** 3
- #define **EXITED** 4
- #define **QUANTUM** 100

5.3.1 Detailed Description

Useful macros for code readability.

5.4 src/include/parsejob.h File Reference

Library for parsing inputs, provided by the instructors.

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

Functions

- bool **parseJob** (char ∗cmdLine, bool tty)
- char ∗ **getJobStdin** (void)
- char ∗ **getJobStdout** (void)
- char ∗∗∗ **getJobCommands** (void)
- int **getCommandCount** (void)
- bool **isAppendingStdout** (void)
- bool **isBackgroundJob** (void)
- void **freeJobCommands** (char ∗∗commands[ ])
- void **printJob** (void)

5.4.1 Detailed Description

Library for parsing inputs, provided by the instructors.

5.5 src/pennfat/pennfat.h File Reference

Contains the interface for the standalone file system.

```
#include "../fs/fat.h"
```

5.5.1 Detailed Description

Contains the interface for the standalone file system.

5.6 src/pennfat/pennfathandler.h File Reference

Contains functions that handle pennFat commands in the standalone shell.

```
#include "../fs/fat.h"
```

Functions

- int handlePennFatCommand (char ***commands, int commandCount, fat **fat)

  Handles pennfat commands.
- int handleMakeFsCommand (char *fileName, uint8_t numBlocks, uint8_t blockSizeIndicator, fat **fat)

  Handles mkfs command.
- int handleMountCommand (char *fileName, fat **fat)

  Handles mount command.
- int handleUnmountCommand (fat **fat)

  Handles unmount command.
- int handleTouchCommand (char **files, fat *fat)

  Handles touch command which creates a file if it doesn't exist, and updates the timestamp.
- int handleMoveCommand (char *oldFileName, char *newFileName, fat *fat)

  Handle move command which renames a file.
- int handleRemoveCommand (char **files, fat *fat)

  Handle remove command which removes a file.
- int handleCatCommand (char **commands, fat *fat)

  Handle cat command.
- int handleCopyCommand (char **commands, fat *fat)

  Handle copy command.
- int handleLsCommand (fat *fat)

  Prints all of the files in the directory.
- int handleChmodCommand (char **commands, fat *fat)

  Changes the permissions of a file.

5.6.1 Detailed Description

Contains functions that handle pennFat commands in the standalone shell.

5.6.2 Function Documentation

5.6.2.1 handleCatCommand()

```
int handleCatCommand (
            char ** commands,
            fat * fat )
```

Handle cat command.

Parameters

commands	The commands to parse and define behavior on
fat	The FAT pointer

Returns

SUCCESS on success, FAILURE on failure

5.6.2.2 handleChmodCommand()

```
int handleChmodCommand (
            char ** commands,
            fat * fat )
```

Changes the permissions of a file.

Parameters

commands	The commands
fat	The fat

Returns

SUCCESS on success, FAILURE if no FAT is mounted

5.6.2.3 handleCopyCommand()

```
int handleCopyCommand (
            char ** commands,
            fat * fat )
```

Handle copy command.

Parameters

commands	The commands to parse and define behavior on
fat	The FAT pointer

Returns

SUCCESS on success, FAILURE on failure

5.6.2.4 handleLsCommand()

```
int handleLsCommand (
            fat * fat )
```

Prints all of the files in the directory.

Parameters

fat	The FAT pointer

Returns

SUCCESS on success, FAILURE if no FAT is mounted

5.6.2.5 handleMakeFsCommand()

```
int handleMakeFsCommand (
            char * fileName,
            uint8_t numBlocks,
            uint8_t blockSizeIndicator,
            fat ** fat )
```

Handles mkfs command.

Parameters

	fileName	The file name
in	*numBlocks*	The number blocks
in	*blockSizeIndicator*	The block size indicator
	fat	Pointer to the FAT pointer

Returns

SUCCESS on success, FAILURE on invalid inputs or failure to allocate space

5.6.2.6 handleMountCommand()

```
int handleMountCommand (
            char * fileName,
            fat ** fat )
```

Handles mount command.

Parameters

fileName	The file name on disk of the FAT
fat	Pointer to the FAT pointer

Returns

SUCCESS on success, FAILURE on failure to allocate or any failed library calls

5.6.2.7 handleMoveCommand()

```
int handleMoveCommand (
            char * oldFileName,
            char * newFileName,
            fat * fat )
```

Handle move command which renames a file.

Parameters

oldFileName	The old file name
newFileName	The new file name
fat	The FAT pointer

Returns

SUCCESS on success, FAILURE on failure

5.6.2.8 handlePennFatCommand()

```
int handlePennFatCommand (
            char *** commands,
            int commandCount,
            fat ** fat )
```

Handles pennfat commands.

Parameters

	commands	The commands
in	*commandCount*	The command count
	fat	Pointer to the FAT pointer

Returns

SUCCESS on success, FAILURE on failure to process command

5.6.2.9 handleRemoveCommand()

```
int handleRemoveCommand (
            char ** files,
            fat * fat )
```

Handle remove command which removes a file.

Parameters

files	The files to touch
fat	The FAT pointer

Returns

SUCCESS on success, FAILURE on failure

5.6.2.10 handleTouchCommand()

```
int handleTouchCommand (
            char ** files,
            fat * fat )
```

Handles touch command which creates a file if it doesn't exist, and updates the timestamp.

Parameters

files	The files to touch
fat	The FAT pointer

Returns

SUCCESS on success, FAILURE on failure

5.6.2.11 handleUnmountCommand()

```
int handleUnmountCommand (
            fat ** fat )
```

Handles unmount command.

Parameters

fat	Pointer to the FAT pointer

Returns

SUCCESS on success, FAILURE if no FAT mounted

5.7 src/pennos/filedescriptor.h File Reference

Contains file descriptor abstraction and user-facing functions for file system interaction.

```
#include "../fs/fat.h"
```

Classes

- struct fileDescriptorNodeType
- struct FileDescriptorContainerType

Typedefs

- typedef struct fileDescriptorNodeType fdNode
- typedef struct FileDescriptorContainerType fdContainer

Functions

- fdContainer * newContainer ()

  Instantiate a new file descriptor container.
- fdNode * newFileDescriptorNode (char *fileName, int mode, directoryEntry *entry)

  Creates and adds a new file descriptor with specified filename and node and assigns an id.
- int f_open (char *fname, int mode)

  System calls using file descriptor abstractions.
- int f_read (int fd, int n, uint8_t *buf)

  Read n bytes from fd into buf.
- int f_write (int fd, uint8_t *buf, int n)

  Write n bytes from buf into fd.
- int f_close (int fd)

  Close the file descriptor.
- int f_mv (char *src, char *dest)

  Rename the src file to dest.
- int f_unlink (char *fileName)

  Remove a file with the specified filename.
- int f_lseek (int fd, int offset, int whence)

  Reposition the file position pointer for the specified file descriptor.
- void f_ls ()

  Lists all the files in the system.
- int f_chmod (char *fileName, int permission)

  Changes the permission of a file.

Variables

- fdContainer * container

5.7.1 Detailed Description

Contains file descriptor abstraction and user-facing functions for file system interaction.

5.7.2 Typedef Documentation

5.7.2.1 fdContainer

```
typedef struct FileDescriptorContainerType fdContainer
```

Stores file descriptors

5.7.2.2 fdNode

```
typedef struct fileDescriptorNodeType fdNode
```

A file descriptor node

5.7.3 Function Documentation

5.7.3.1 f_chmod()

```
int f_chmod (
            char * fileName,
            int permission )
```

Changes the permission of a file.

Parameters

	fileName	The file name
in	*permission*	The new permission

Returns

SUCCESS (0) on success, FAILURE (-1) on failure

5.7.3.2 f_close()

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

Close the file descriptor.

Parameters

in	*fd*	The id of the file descriptor to close

Returns

SUCCESS (0) on success, FAILURE (-1) on failure

5.7.3.3 f_lseek()

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

Reposition the file position pointer for the specified file descriptor.

Parameters

in	*fd*	The file descriptor
in	*offset*	The offset
in	*whence*	The whence

Returns

The new location of the file pointer on success, FAILURE (-1) on failure

5.7.3.4 f_mv()

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

Rename the src file to dest.

Parameters

src	The source
dest	The destination

Returns

SUCCESS (0) on success, FAILURE (-1) on failure

5.7.3.5 f_open()

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

System calls using file descriptor abstractions.

Opens a file descriptor

Parameters

in	*fname*	The filename
in	*mode*	The mode

Returns

The ID of the file descriptor on success or FAILURE (-1) if it failed

5.7.3.6 f_read()

```
int f_read (
            int fd,
            int n,
            uint8_t * buf )
```

Read n bytes from fd into buf.

Parameters

in	*fd*	The file descriptor
in	*n*	The number of bytes to read
	buf	The buffer to read into

Returns

The number of bytes read, 0 if EOF reached, or FAILURE (-1) on error

5.7.3.7 f_unlink()

```
int f_unlink (
            char * fileName )
```

Remove a file with the specified filename.

Parameters

fileName	The file name

Returns

SUCCESS (0) on success, FAILURE (-1) on failure

5.7.3.8 f_write()

```
int f_write (
            int fd,
            uint8_t * buf,
            int n )
```

Write n bytes from buf into fd.

Parameters

in	*fd*	The file descriptor
	buf	The buffer to read from
in	*n*	The number of bytes to write

Returns

The number of bytes written, or FAILURE (-1) on error

5.7.3.9 newContainer()

```
fdContainer* newContainer ( )
```

Instantiate a new file descriptor container.

Returns

Pointer to the new container

5.7.3.10 newFileDescriptorNode()

```
fdNode* newFileDescriptorNode (
            char * fileName,
            int mode,
            directoryEntry * entry )
```

Creates and adds a new file descriptor with specified filename and node and assigns an id.

Parameters

	fileName	The file name for the descriptor
in	*mode*	The mode for the descriptor

Returns

Pointer to the new file descriptor node, or NULL if it failed

5.7.4 Variable Documentation

5.7.4.1 container

```
fdContainer* container
```

Global container for file descriptors

5.8 src/pennos/handlejob.h File Reference

Contains a handler to handle jobs inputted into the shell.

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

Functions

- void terCtrlSighandler (int signum)

  Handles the terminal control signal.
- int handleJob (char ∗∗∗commands, int commandCount, int index, job ∗job, int ∗currentPgId)

5.8.1 Detailed Description

Contains a handler to handle jobs inputted into the shell.

5.8.2 Function Documentation

5.8.2.1 handleJob()

```
int handleJob (
            char *** commands,
            int commandCount,
            int index,
            job * job,
            int * currentPgId )
```

[handleJob description]

Parameters

commands	the array (NULL-terminated) of parsed commands
commandCount	the number of commands in the parsed input
index	the index of the current command of the job
job	this job pointer for this job
currentPgId	a pointer to the currentPgId in the parent

Returns

currentPgId if job completed successfully, -1 otherwise

5.8.2.2 terCtrlSighandler()

```
void terCtrlSighandler (
            int signum )
```

Handles the terminal control signal.

Parameters

in	*signum*	The signum

5.9 src/pennos/iter.h File Reference

Contains the core logic of a shell loop iteration.

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

Functions

- void iter (char ∗line, bool interactive, int ∗currentPgId, jobQueue ∗q)

  Processes user input into the shell.
- void exitGracefully (char ∗∗∗jobCommands, char ∗jobStdin, char ∗jobStdout, jobQueue ∗queue, char ∗line, char ∗errMsg)

  Frees the passed parameters and then exits with EXIT_FAILURE.

5.9.1 Detailed Description

Contains the core logic of a shell loop iteration.

5.9.2 Function Documentation

5.9.2.1 exitGracefully()

```
void exitGracefully (
            char *** jobCommands,
            char * jobStdin,
            char * jobStdout,
            jobQueue * queue,
            char * line,
            char * errMsg )
```

Frees the passed parameters and then exits with EXIT_FAILURE.

Parameters

jobCommands	the parsed commands
jobStdin	the redirected stdin, or null
jobStdout	the redirected stdout, or null
queue	the job queue to destroy
line	the buffer line used by getline in main
errMsg	the message to display with perror

5.9.2.2 iter()

```
void iter (
            char * line,
            bool interactive,
            int * currentPgId,
            jobQueue * q )
```

Processes user input into the shell.

Parameters

line	the user input string
interactive	whether this is interactive mode or not
currentPgId	a pointer to currentPgId in the parent
q	a pointer to the job queue

5.10 src/pennos/job.h File Reference

Contains a job abstraction for the shell.

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

Classes

- struct jobTag

Typedefs

- typedef struct jobTag job

Functions

- job * newJob (char ***jobCommands, int commandCount, int infile, int outfile)

  Initialize a new job, with no process group id until first passed into handleJob()
- void freeJob (job *j)

  Frees a job and its job description.
- void printJobDetails (job *j)

  Prints a job in a human-readable format.
- void printRunningJob (job *j)

  Prints the "Running: <command>" message for background jobs.
- void printFinishedJob (job *j)

  Prints the "Finished: <command>" message for background jobs.

5.10.1 Detailed Description

Contains a job abstraction for the shell.

5.10.2 Typedef Documentation

5.10.2.1 job

```
typedef struct jobTag job
```

A job with pointers to next/prev jobs to allow a linked-list of jobs

Each job has a jobId, a pgId (initially -1), a job description, and its input/output file descriptors. Every job is also responsible for mallocing and freeing an array of pids (for each command) and mallocing, closing, and freeing the array of pipes that it uses in its pipeline.

Jobs are passed to handleJob() [commandCount] number of times, in sequence, and pgId is initialized the first time it is passed. Each handleJob() call lazily creates pipes and assigns pids to each command in the job.

5.10.3 Function Documentation

5.10.3.1 freeJob()

```
void freeJob (
            job * j )
```

Frees a job and its job description.

Parameters

j	a job pointer to free

5.10.3.2 newJob()

```
job* newJob (
            char *** jobCommands,
            int commandCount,
            int infile,
            int outfile )
```

Initialize a new job, with no process group id until first passed into handleJob()

Parameters

jobCommands	the array of commands from parsejob
commandCount	the number of commands in this job
infile	the input file descriptor number for this job
outfile	the output file descriptor number for this job

Returns

a pointer to the job, or NULL if it failed to initialize

5.10.3.3 printFinishedJob()

```
void printFinishedJob (
            job * j )
```

Prints the "Finished: <command>" message for background jobs.

Parameters

j	a job pointer to print the message for

5.10.3.4 printJobDetails()

```
void printJobDetails (
            job * j )
```

Prints a job in a human-readable format.

Parameters

j	a job pointer to print

5.10.3.5 printRunningJob()

```
void printRunningJob (
            job * j )
```

Prints the "Running: <command>" message for background jobs.

Parameters

j	a job pointer to print the message for

5.11 src/pennos/jobcontrol.h File Reference

Contains logic for facilitating job control in the shell.

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

Functions

- bool isJobControlCommand (char **command)
- void handleJobControlCommand (char **command, jobQueue *q, int *currentPgId)
- int handleForegroundCommand (jobQueue *q, job *job, int *currentPgId)
- int handleBackgroundCommand (jobQueue *q, job *job)
- int handleJobsCommand (jobQueue *q)
- int putJobInForeground (jobQueue *q, job *job, bool interactive)
- job ** pollJobChanges (jobQueue *q)

5.11.1 Detailed Description

Contains logic for facilitating job control in the shell.

5.11.2 Function Documentation

5.11.2.1 handleBackgroundCommand()

```
int handleBackgroundCommand (
            jobQueue * q,
            job * job )
```

Runs the chosen stopped job in the job jobQueue in the background

Parameters

q	the job jobQueue to handle commands on
job	the job to resume in the background if a valid command, or NULL

Returns

0 on success, or -1 on failure

5.11.2.2 handleForegroundCommand()

```
int handleForegroundCommand (
            jobQueue * q,
            job * job,
            int * currentPgId )
```

Runs the chosen job in the job jobQueue in the foreground

Parameters

q	the job jobQueue to handle commands on
job	the job to resume in the foreground if a valid command, or NULL
current↩ PgId	a pointer to the currentPgId in the parent

Returns

0 on success, or -1 on failure

5.11.2.3 handleJobControlCommand()

```
void handleJobControlCommand (
            char ** command,
            jobQueue * q,
            int * currentPgId )
```

Only called when isJobControlCommand returns true and passes the command to the correct handler

Parameters

command	the array of strings from parsejob
q	the job jobQueue to handle commands on
current↩ PgId	a pointer to the currentPgId in the parent

5.11.2.4 handleJobsCommand()

```
int handleJobsCommand (
            jobQueue * q )
```

Lists the jobs in the job jobQueue

Parameters

q	the job jobQueue to handle commands on

Returns

0 on success, or -1 on failure

5.11.2.5 isJobControlCommand()

```
bool isJobControlCommand (
            char ** command )
```

Takes in an array of strings from parsejob and returns true if the command is a job control command

Parameters

command	the array of strings from parsejob

Returns

true only if the commands represent a job control command

5.11.2.6 pollJobChanges()

```
job** pollJobChanges (
            jobQueue * q )
```

Polls for any state changes and if so, looks for completed jobs and returns an array of those jobs

Parameters

q	The jobQueue to search for jobs from

Returns

a null-terminated array of finished job pointers, or NULL on failure

5.11.2.7 putJobInForeground()

```
int putJobInForeground (
            jobQueue * q,
            job * job,
            bool interactive )
```

Puts a given job in the foreground

Parameters

q	the job jobQueue to handle commands on
job	the job to put in the foreground
interactive	whether the shell is in interactive mode

Returns

0 on success, or -1 on failure

5.12 src/pennos/jobQueue.h File Reference

Contains a linked-list style job queue implementation.

```
#include "job.h"
```

Classes

- struct jobQueue

Functions

- jobQueue * jobQueueInit ()
- int jobQueueCount (jobQueue *q)
- job * jobQueueFront (jobQueue *q)
- job * jobQueuePush (jobQueue *q, job *j)
- job * jobQueuePop (jobQueue *q)
- job * jobQueueRemoveJob (jobQueue *q, job *j)
- void jobQueueClear (jobQueue *q)
- void jobQueueDestroy (jobQueue *q)
- void jobQueuePrint (jobQueue *q)

5.12.1 Detailed Description

Contains a linked-list style job queue implementation.

5.12.2 Function Documentation

5.12.2.1 jobQueueClear()

```
void jobQueueClear (
            jobQueue * q )
```

Frees and removes all jobs in the jobQueue

Parameters

q	a pointer to the jobQueue

5.12.2.2 jobQueueCount()

```
int jobQueueCount (
            jobQueue * q )
```

Returns the number of items in the jobQueue

Parameters

q	a pointer to the jobQueue

Returns

the number of jobs in the jobQueue of -1 if the jobQueue is invalid

5.12.2.3 jobQueueDestroy()

```
void jobQueueDestroy (
            jobQueue * q )
```

Frees and removes all jobs in the jobQueue and frees the jobQueue itself

Parameters

q	a pointer to the jobQueue

5.12.2.4 jobQueueFront()

```
job* jobQueueFront (
            jobQueue * q )
```

Returns the job at the front of the jobQueue

Parameters

q	a pointer to the jobQueue

Returns

a pointer to the job at the front of the jobQueue, or null if the jobQueue is uninitialized/empty

5.12.2.5 jobQueueInit()

```
jobQueue* jobQueueInit ( )
```

Allocates and initializes an empty jobQueue

Returns

a pointer to the newly created jobQueue, or null if failed to allocate memory

5.12.2.6 jobQueuePop()

```
job* jobQueuePop (
            jobQueue * q )
```

Removes a job from the front of the jobQueue

Parameters

q	a pointer to the jobQueue

Returns

a pointer to the job just removed, or null if the jobQueue is uninitialized/empty

5.12.2.7 jobQueuePrint()

```
void jobQueuePrint (
            jobQueue * q )
```

Prints out the jobs of this jobQueue in order to stdout

Parameters

q	a pointer to the jobQueue

5.12.2.8 jobQueuePush()

```
job* jobQueuePush (
            jobQueue * q,
            job * j )
```

Adds a job to the end of the jobQueue

Parameters

q	a pointer to the jobQueue
j	a pointer to the job to add

Returns

a pointer to the job just added, or null if the jobQueue is uninitialized

5.12.2.9 jobQueueRemoveJob()

```
job* jobQueueRemoveJob (
            jobQueue * q,
            job * j )
```

Removes a specific particular job from a particular jobQueue

Parameters

q	a pointer to the jobQueue
j	the pointer to the job in the jobQueue to remove

5.13 src/pennos/kernel.h File Reference

Contains kernel level functions.

```
#include "PCB.h"
#include "queue.h"
#include "../fs/fat.h"
#include "scheduler.h"
```

Functions

- pcb_t * **k_process_create** (pcb_t *parent)
- void **dealWithUnwaitedProcess** (pcb_t *process)
- void **k_process_kill** (pcb_t *process, int signal)
- void **k_process_cleanup** (pcb_t *process)
- void **clearZombiesAndChildren** (pcb_t *process)
- void **schedule** ()
- pcb_t * **getForegroundProcess** ()
- pcb_t * **getCurrProcess** ()
- queue * **getProcessTable** ()
- int **getNumTicks** ()
- scheduler * **getScheduler** ()

- void **makeContext** (ucontext_t *ucp, void(*func)(), char *argv[])
- void **switchContext** (int signum)
- void **addToAsleep** (node *n)
- void **setForeground** (pid_t pid)
- void **unblockParent** (pid_t ppid)
- FILE * **getLogfile** ()
- pid_t **traverseChild** (pcb_t *process, int *wstatus)

Variables

- fat * **mountedFat**

5.13.1 Detailed Description

Contains kernel level functions.

5.14 src/pennos/node.h File Reference

Defines a linked process node that contains a PCB and a PID.

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

Classes

- struct nodeTag

Typedefs

- typedef struct nodeTag node

Functions

- node * newNode (pid_t pid, pcb_t *pcb)
- void freenode (node *n)
- void printnodeDetails (node *n)

5.14.1 Detailed Description

Defines a linked process node that contains a PCB and a PID.

5.14.2 Typedef Documentation

5.14.2.1 node

```
typedef struct nodeTag node
```

A node with pointers to next/prev nodes to allow a linked-list of nodes

Each node has a pid corresponding to some process

5.14.3 Function Documentation

5.14.3.1 freenode()

```
void freenode (
            node * n )
```

Frees a node

Parameters

n	a node pointer to free

5.14.3.2 newNode()

```
node* newNode (
            pid_t pid,
            pcb_t * pcb )
```

Initialize a new node, with no process group id until first passed into handlenode()

Parameters

pid	pid of the process

Returns

a pointer to the node

5.14.3.3 printnodeDetails()

```
void printnodeDetails (
            node * n )
```

Prints a node

Parameters

n	a node pointer to print

5.15 src/pennos/PCB.h File Reference

Defines a PCB type.

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

Classes

- struct childTag
- struct pcbType

Typedefs

- typedef struct childTag child
- typedef struct pcbType pcb_t

5.15.1 Detailed Description

Defines a PCB type.

5.15.2 Typedef Documentation

5.15.2.1 child

```
typedef struct childTag child
```

Process Control Block child object

5.15.2.2 pcb_t

```
typedef struct pcbType pcb_t
```

A process control block, which is used by the kernel to context switch to and from a particular process, send signals to a process or process group, etc.

5.16 src/pennos/queue.h File Reference

Defines a FIFO doubly-linked queue and provides functions for interacting with queues.

```
#include "node.h"
```

Classes

- struct queue

Functions

- queue ∗ queueInit ()
- int queueCount (queue ∗q)
- node ∗ queueFront (queue ∗q)
- node ∗ queuePush (queue ∗q, node ∗n)
- node ∗ queuePop (queue ∗q)
- node ∗ queueSearch (queue ∗q, node ∗n)
- node ∗ queueRemoveNode (queue ∗q, node ∗n)
- void queueClear (queue ∗q)
- void queueDestroy (queue ∗q)
- void queuePrint (queue ∗q)

5.16.1 Detailed Description

Defines a FIFO doubly-linked queue and provides functions for interacting with queues.

5.16.2 Function Documentation

5.16.2.1 queueClear()

```
void queueClear (
            queue * q )
```

Frees and removes all nodes in the queue

Parameters

q	a pointer to the queue

5.16.2.2 queueCount()

```
int queueCount (
            queue * q )
```

Returns the number of items in the queue

Parameters

q	a pointer to the node queue

Returns

the number of nodes in the queue of -1 if the queue is invalid

5.16.2.3 queueDestroy()

```
void queueDestroy (
            queue * q )
```

Frees and removes all nodes in the node queue and frees the queue itself

Parameters

q	a pointer to the queue

5.16.2.4 queueFront()

```
node* queueFront (
            queue * q )
```

Returns the node at the front of the queue

Parameters

q	a pointer to the queue

Returns

a pointer to the node at the front of the node queue, or null if the queue is uninitialized/empty

5.16.2.5 queueInit()

```
queue* queueInit ( )
```

Allocates and initializes an empty queue

Returns

a pointer to the newly created queue, or null if failed to allocate memory

5.16.2.6 queuePop()

```
node* queuePop (
            queue * q )
```

Removes a node from the front of the queue

Parameters

q	a pointer to the queue

Returns

a pointer to the node just removed, or null if the queue is uninitialized/empty

5.16.2.7 queuePrint()

```
void queuePrint (
            queue * q )
```

Prints out the nodes of this queue in order to stdout

Parameters

q	a pointer to the queue

5.16.2.8 queuePush()

```
node* queuePush (
            queue * q,
            node * n )
```

Adds a node to the end of the queue

Parameters

q	a pointer to the queue
n	a pointer to the node to add

Returns

a pointer to the node just added, or null if the queue is uninitialized

5.16.2.9 queueRemoveNode()

```
node* queueRemoveNode (
              queue * q,
              node * n )
```

Removes a specific particular node from a particular queue

Parameters

q	a pointer to the queue
n	the pointer to the node in the queue to remove

Returns

pointer to the removed node, or NULL if it doesn't exist

5.16.2.10 queueSearch()

```
node* queueSearch (
              queue * q,
              node * n )
```

Finds a specific node with a given pid

Parameters

n	the node to find

Returns

the node with the input node's pid, or NULL if it doesn't exist

5.17 src/pennos/scheduler.h File Reference

Contains the scheduler runqueues as well as functions to interact with a scheduler.

```
#include "queue.h"
#include "node.h"
```

Classes

- struct scheduler

Functions

- scheduler * schedulerInit ()

  Initialize a scheduler.
- void addToScheduler (node *process, scheduler *s)

  Adds a process to the scheduler.
- void removeFromScheduler (node *process, scheduler *s)

  Removes a process from the scheduler.
- node * getNextProcess (scheduler *s)

  Gets the next process the scheduler wants to run.

5.17.1 Detailed Description

Contains the scheduler runqueues as well as functions to interact with a scheduler.

5.17.2 Function Documentation

5.17.2.1 addToScheduler()

```
void addToScheduler (
            node * process,
            scheduler * s )
```

Adds a process to the scheduler.

Parameters

process	Pointer to the process
s	Pointer to the scheduler

5.17.2.2 getNextProcess()

```
node* getNextProcess (
            scheduler * s )
```

Gets the next process the scheduler wants to run.

Parameters

s	Pointer to the scheduler

Returns

The next process to run, returns NULL if idle

5.17.2.3 removeFromScheduler()

```
void removeFromScheduler (
            node * process,
            scheduler * s )
```

Removes a process from the scheduler.

Parameters

process	Pointer to the process
s	Pointer to the scheduler

5.17.2.4 schedulerInit()

```
scheduler* schedulerInit ( )
```

Initialize a scheduler.

Returns

Pointer to a new scheduler in memory, or NULL if it failed to initialize

5.18 src/pennos/shell.h File Reference

Contains shell built-ins and the shell loop.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <string.h>
#include "../include/parsejob.h"
#include "user_level_funcs.h"
```

Functions

- void **shell** ()
- void **busy** ()
- void head (char **argv)

  Function for printing first ten lines of specified files to console.
- void ps ()
- void **killer** (char *argv[])
- void **zombify** ()
- void **orphanify** ()
- void **man** ()
- void **createSleep** (char **argv)
- void ls ()

  Lists all the files in the mounted filesystem.
- void touch (char **argv)

  Creates empty files if they do not exist or update timestamp otherwise.
- void mv (char **argv)

  Rename src to dest.
- void cp (char **argv)

  Copy src to a new file dest.
- void rm (char **argv)

  Removes files.
- void cat (char **argv)

  Concatenates files to a file descriptor or reads input and spits it back to user.
- void chmod (char **argv)

  Changes the permission of a file.

5.18.1 Detailed Description

Contains shell built-ins and the shell loop.

5.18.2 Function Documentation

5.18.2.1 cat()

```
void cat (
            char ** argv )
```

Concatenates files to a file descriptor or reads input and spits it back to user.

Parameters

argv	The arguments array

5.18.2.2 chmod()

```
void chmod (
            char ** argv )
```

Changes the permission of a file.

Parameters

argv	The arguments array

5.18.2.3 cp()

```
void cp (
            char ** argv )
```

Copy src to a new file dest.

Parameters

argv	The arguments to parse

5.18.2.4 head()

```
void head (
            char ** argv )
```

Function for printing first ten lines of specified files to console.

Parameters

argv	list of files to call head on

5.18.2.5 mv()

```
void mv (
            char ** argv )
```

Rename src to dest.

Parameters

argv	The arguments to parse

5.18.2.6 ps()

```
void ps ( )
```

build output string with pid, ppid, and priority

5.18.2.7 rm()

```
void rm (
            char ** argv )
```

Removes files.

Parameters

argv	The arguments to parse

5.18.2.8 touch()

```
void touch (
            char ** argv )
```

Creates empty files if they do not exist or update timestamp otherwise.

Parameters

argv	The arguments to parse

5.19 src/pennos/signal.h File Reference

Contains macros to define signals.

Macros

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

5.19.1 Detailed Description

Contains macros to define signals.

5.20 src/pennos/token.h File Reference

Contains a function reads the parsed job commands and concatenates them to form a string.

Functions

- char * getCommandStringFromTokens (char ***jobCommands, int commandCount)
- char *** **getCopyOfCommands** (char ***jobCommands, int commandCount)

5.20.1 Detailed Description

Contains a function reads the parsed job commands and concatenates them to form a string.

5.20.2 Function Documentation

5.20.2.1 getCommandStringFromTokens()

```
char* getCommandStringFromTokens (
            char *** jobCommands,
            int commandCount )
```

Returns a pointer to a string representing the job

Parameters

jobCommands	a pointer to the array of commands for the job
commandCount	the number of commands in this job

Returns

a pointer to the string representing the job

5.21 src/pennos/user_level_funcs.h File Reference

Contains user level functions for dealing with processes.

Functions

- pid_t **p_spawn** (void(∗func)(), char ∗argv[ ], int fd0, int fd1)
- pid_t **p_waitpid** (pid_t pid, int ∗wstatus, bool nohang)
- int **p_kill** (pid_t pid, int sig)
- void **p_sleep** (unsigned int ticks)
- void **p_exit** (void)
- void **setForegroundProcess** (pid_t pid)
- void **checkForTerminalControl** ()
- int p_nice (pid_t pid, int priority)

  Sets priority of process.

5.21.1 Detailed Description

Contains user level functions for dealing with processes.

5.21.2 Function Documentation

5.21.2.1 p_nice()

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

Sets priority of process.

Parameters

pid	Pid of process whose priority will be set
priority	priority to be set

Returns

0 if successful else -1

Index