Signals:

We are handling three signals, SIGINT, SIGCONT and SIGTSTP in this assignment. When a SIGINT is received, if the shell is running, prompt again, otherwise it terminates the current running process. When a SIGCONT is received by a certain stopped process, resume the process. When a SIGTSTP is received, if the shell is running, prompt again, otherwise it stops the current running process.

PennOS Team24

Generated by Doxygen 1.9.6

1 PennOS (23sp-pennOS-group-24) 1

1.0.1 Authors . . . 1

1.0.2 Source Files . . . 1

1.0.3 Extra Credit Answers . . . 1

1.0.4 Compilation Instructions . . . 1

1.0.5 Overview of Work Accomplished . . . 2

1.0.6 Description of Code and Code Layout . . . 2

1.0.7 General Comments . . . 2

2 Class Index 3

2.1 Class List . . . 3

3 File Index 5

3.1 File List . . . 5

4 Class Documentation 7

4.1 DirectoryEntry Struct Reference . . . 7

4.1.1 Detailed Description . . . 7

4.2 FATConfig Struct Reference . . . 8

4.2.1 Detailed Description . . . 8

4.3 FdNode Struct Reference . . . 8

4.3.1 Detailed Description . . . 9

4.4 FdTable Struct Reference . . . 9

4.4.1 Detailed Description . . . 9

4.5 Job Struct Reference . . . 9

4.6 JobList Struct Reference . . . 10

4.7 JobListNode Struct Reference . . . 10

4.8 parsed_command Struct Reference . . . 10

4.8.1 Detailed Description . . . 10

4.9 pcb Struct Reference . . . 11

4.9.1 Detailed Description . . . 11

4.10 pcb_node Struct Reference . . . 11

4.10.1 Detailed Description . . . 12

4.11 pcb_queue Struct Reference . . . 12

4.11.1 Detailed Description . . . 12

4.12 priority_queue Struct Reference . . . 12

4.12.1 Detailed Description . . . 12

5 File Documentation 13

5.1 src/kernel/behavior.c File Reference . . . 13

5.1.1 Detailed Description . . . 13

5.1.2 Function Documentation . . . 14

5.1.2.1 executeLine() . . . 14

5.1.2.2 executeProgram() . . . 14

5.1.2.3 executeScript() 15
5.1.2.4 parseLine() 15
5.1.2.5 parseProgramType() 15
5.1.2.6 parseUserInput() 16
5.1.2.7 readAndParseUserInput() 16
5.1.2.8 readUserInput() 16
5.1.2.9 writePrompt() 17
5.2 src/kernel/behavior.h File Reference 17
5.2.1 Detailed Description 18
5.2.2 Enumeration Type Documentation 18
5.2.2.1 LineType 18
5.2.2.2 ProgramType 18
5.2.3 Function Documentation 19
5.2.3.1 executeLine() 19
5.2.3.2 executeProgram() 19
5.2.3.3 executeScript() 19
5.2.3.4 parseLine() 20
5.2.3.5 parseProgramType() 20
5.2.3.6 parseUserInput() 20
5.2.3.7 readAndParseUserInput() 21
5.2.3.8 readUserInput() 21
5.2.3.9 writePrompt() 21
5.3 behavior.h 22
5.4 global.h 22
5.5 global2.h 23
5.6 src/kernel/job.c File Reference 23
5.6.1 Detailed Description 24
5.6.2 Function Documentation 24
5.6.2.1 appendJobList() 24
5.6.2.2 bgBuildinCommand() 25
5.6.2.3 clearJobList() 25
5.6.2.4 createJob() 25
5.6.2.5 executeBuiltinCommand() 26
5.6.2.6 fgBuildinCommand() 26
5.6.2.7 findJobList() 26
5.6.2.8 findJobListByJobId() 27
5.6.2.9 findTheCurrentJob() 27
5.6.2.10 initJobList() 27
5.6.2.11 jobsBuildinCommand() 28
5.6.2.12 killBuildinCommand() 28
5.6.2.13 manBuildinCommand() 28
5.6.2.14 nicePidBuildinCommand() 28

5.6.2.15 parseBuiltinCommandType() 29
5.6.2.16 pollBackgroundProcesses() 29
5.6.2.17 popJobList() 29
5.6.2.18 printCommandLine() 29
5.6.2.19 printJobList() 30
5.6.2.20 removeJobList() 30
5.6.2.21 removeJobListByJobId() 30
5.6.2.22 removeJobListWithoutFreeCmd() 31
5.6.2.23 updateJobList() 31
5.6.2.24 updateJobListByJobId() 32
5.6.2.25 writeJobState() 32
5.6.2.26 writeJobStatePrompt() 32
5.6.2.27 writeNewline() 33

5.7 src/kernel/job.h File Reference 33

5.7.1 Detailed Description 34

5.7.2 Function Documentation 35

5.7.2.1 appendJobList() 35
5.7.2.2 bgBuildinCommand() 35
5.7.2.3 clearJobList() 35
5.7.2.4 createJob() 36
5.7.2.5 executeBuiltinCommand() 36
5.7.2.6 fgBuildinCommand() 36
5.7.2.7 findJobList() 37
5.7.2.8 findJobListByJobId() 37
5.7.2.9 findTheCurrentJob() 37
5.7.2.10 initJobList() 38
5.7.2.11 jobsBuildinCommand() 38
5.7.2.12 killBuildinCommand() 38
5.7.2.13 manBuildinCommand() 38
5.7.2.14 nicePidBuildinCommand() 39
5.7.2.15 parseBuiltinCommandType() 39
5.7.2.16 pollBackgroundProcesses() 39
5.7.2.17 popJobList() 39
5.7.2.18 printCommandLine() 40
5.7.2.19 printJobList() 40
5.7.2.20 removeJobList() 40
5.7.2.21 removeJobListByJobId() 41
5.7.2.22 removeJobListWithoutFreeCmd() 41
5.7.2.23 updateJobList() 41
5.7.2.24 updateJobListByJobId() 43
5.7.2.25 writeJobState() 43
5.7.2.26 writeJobStatePrompt() 44

5.7.2.27 writeNewline() 44
5.8 job.h 44
5.9 kernel.h 45
5.10 log.h 45
5.11 parser.h 45
5.12 src/kernel/perrno.h File Reference 46
5.12.1 Detailed Description 47
5.12.2 Function Documentation 47
5.12.2.1 p_perror() 47
5.12.2.2 p_set_errno() 47
5.13 perrno.h 48
5.14 src/kernel/programs.c File Reference 48
5.14.1 Detailed Description 49
5.14.2 Function Documentation 49
5.14.2.1 s_busy() 49
5.14.2.2 s_cat() 50
5.14.2.3 s_chmod() 50
5.14.2.4 s_cp() 50
5.14.2.5 s_echo() 51
5.14.2.6 s_hang() 51
5.14.2.7 s_kill() 51
5.14.2.8 s_ls() 51
5.14.2.9 s_mv() 52
5.14.2.10 s_nohang() 52
5.14.2.11 s_orphanify() 52
5.14.2.12 s_ps() 53
5.14.2.13 s_recur() 53
5.14.2.14 s_rm() 53
5.14.2.15 s_sleep() 53
5.14.2.16 s_test() 54
5.14.2.17 s_touch() 54
5.14.2.18 s_zombify() 54
5.15 src/kernel/programs.h File Reference 54
5.15.1 Detailed Description 56
5.15.2 Function Documentation 56
5.15.2.1 s_busy() 56
5.15.2.2 s_cat() 56
5.15.2.3 s_chmod() 57
5.15.2.4 s_cp() 57
5.15.2.5 s_echo() 57
5.15.2.6 s_hang() 57
5.15.2.7 s_kill() 58

5.15.2.8 s_ls() 58
5.15.2.9 s_mv() 58
5.15.2.10 s_nohang() 59
5.15.2.11 s_orphanify() 59
5.15.2.12 s_ps() 59
5.15.2.13 s_recur() 59
5.15.2.14 s_rm() 60
5.15.2.15 s_sleep() 60
5.15.2.16 s_test() 60
5.15.2.17 s_touch() 61
5.15.2.18 s_zombify() 61
5.16 programs.h 61
5.17 scheduler.h 62
5.18 src/kernel/shell.c File Reference 62
5.18.1 Detailed Description 63
5.18.2 Function Documentation 63
5.18.2.1 isBuildinCommand() 63
5.18.2.2 isKnownProgram() 64
5.18.2.3 shell_init() 64
5.18.2.4 shell_process() 64
5.19 src/kernel/shell.h File Reference 65
5.19.1 Detailed Description 65
5.19.2 Function Documentation 65
5.19.2.1 isBuildinCommand() 65
5.19.2.2 isKnownProgram() 66
5.19.2.3 shell_init() 66
5.19.2.4 shell_process() 67
5.20 shell.h 67
5.21 stress.h 67
5.22 src/kernel/user.h File Reference 67
5.22.1 Detailed Description 68
5.22.2 Function Documentation 68
5.22.2.1 deconstruct_idle() 69
5.22.2.2 deconstruct_shell() 69
5.22.2.3 get_node_by_pid_all_alive_queues() 69
5.22.2.4 p_exit() 69
5.22.2.5 p_kill() 69
5.22.2.6 p_nice() 70
5.22.2.7 p_sleep() 70
5.22.2.8 p_spawn() 70
5.22.2.9 p_waitpid() 71
5.22.2.10 writePrompt() 71

5.23 user.h 72
5.24 src/PennFAT/utils.c File Reference 72
5.24.1 Detailed Description 72
5.25 utils.h 73
5.26 src/PennFAT/utils.h File Reference 74
5.26.1 Detailed Description 75
5.27 utils.h 75
5.28 src/PennFAT/FAT.c File Reference 75
5.28.1 Detailed Description 76
5.28.2 Function Documentation 77
5.28.2.1 createDirectoryEntry() 77
5.28.2.2 createFAT16InMemory() 77
5.28.2.3 createFATConfig() 78
5.28.2.4 createFATOnDisk() 78
5.28.2.5 createFileDirectoryOnDisk() 78
5.28.2.6 deleteFileDirectory() 79
5.28.2.7 deleteFileDirectoryByName() 79
5.28.2.8 findEmptyFAT16Entry() 80
5.28.2.9 findFileDirectory() 80
5.28.2.10 isDirectoryEntryToDelete() 81
5.28.2.11 readDirectoryEntry() 81
5.28.2.12 readFAT() 82
5.28.2.13 traceBytesFromBeginning() 82
5.28.2.14 traceBytesToEnd() 83
5.28.2.15 traceFileEnd() 83
5.28.2.16 traceOffset() 83
5.28.2.17 writeFAT() 84
5.28.2.18 writeFileDirectory() 84
5.29 src/PennFAT/FAT.h File Reference 85
5.29.1 Detailed Description 87
5.29.2 Typedef Documentation 87
5.29.2.1 DirectoryEntry 87
5.29.3 Function Documentation 87
5.29.3.1 createDirectoryEntry() 87
5.29.3.2 createFAT16InMemory() 88
5.29.3.3 createFATConfig() 88
5.29.3.4 createFATOnDisk() 89
5.29.3.5 createFileDirectoryOnDisk() 89
5.29.3.6 deleteFileDirectory() 90
5.29.3.7 deleteFileDirectoryByName() 90
5.29.3.8 findEmptyFAT16Entry() 90
5.29.3.9 findFileDirectory() 91

5.29.3.10 isDirectoryEntryToDelete() . . . 91
5.29.3.11 readDirectoryEntry() . . . 92
5.29.3.12 readFAT() . . . 92
5.29.3.13 traceBytesFromBeginning() . . . 93
5.29.3.14 traceBytesToEnd() . . . 93
5.29.3.15 traceFileEnd() . . . 94
5.29.3.16 traceOffset() . . . 94
5.29.3.17 writeFAT() . . . 94
5.29.3.18 writeFileDirectory() . . . 95
5.30 FAT.h . . . 95
5.31 src/PennFAT/fd-table.c File Reference . . . 97
5.31.1 Detailed Description . . . 97
5.31.2 Function Documentation . . . 98
5.31.2.1 appendFdTable() . . . 98
5.31.2.2 clearFdTable() . . . 98
5.31.2.3 createFdNode() . . . 98
5.31.2.4 findAvailableFd() . . . 99
5.31.2.5 initFdTable() . . . 99
5.31.2.6 isFileBeingUsed() . . . 100
5.31.2.7 isFileBeingWritten() . . . 100
5.31.2.8 removeFdNode() . . . 100
5.32 src/PennFAT/fd-table.h File Reference . . . 101
5.32.1 Detailed Description . . . 102
5.32.2 Typedef Documentation . . . 102
5.32.2.1 FdTable . . . 103
5.32.3 Function Documentation . . . 103
5.32.3.1 appendFdTable() . . . 103
5.32.3.2 clearFdTable() . . . 103
5.32.3.3 createFdNode() . . . 104
5.32.3.4 findAvailableFd() . . . 104
5.32.3.5 initFdTable() . . . 104
5.32.3.6 isFileBeingUsed() . . . 105
5.32.3.7 isFileBeingWritten() . . . 105
5.32.3.8 removeFdNode() . . . 105
5.33 fd-table.h . . . 106
5.34 src/PennFAT/filesys.c File Reference . . . 106
5.34.1 Detailed Description . . . 107
5.34.2 Function Documentation . . . 108
5.34.2.1 fs_chmod() . . . 108
5.34.2.2 fs_cp() . . . 108
5.34.2.3 fs_mkfs() . . . 108
5.34.2.4 fs_mount() . . . 109

5.34.2.5 fs_mv() 109
5.34.2.6 fs_readFAT() 110
5.34.2.7 fs_rm() 110
5.34.2.8 fs_touch() 110
5.34.2.9 fs_unmount() 111
5.34.2.10 fs_writeFAT() 111
5.34.2.11 isFileSystemMounted() 111
5.34.2.12 isValidFileName() 112
5.35 src/PennFAT/filesys.h File Reference 112
5.35.1 Detailed Description 113
5.35.2 Function Documentation 113
5.35.2.1 fs_chmod() 113
5.35.2.2 fs_cp() 113
5.35.2.3 fs_mkfs() 114
5.35.2.4 fs_mount() 114
5.35.2.5 fs_mv() 115
5.35.2.6 fs_readFAT() 115
5.35.2.7 fs_rm() 115
5.35.2.8 fs_touch() 116
5.35.2.9 fs_unmount() 116
5.35.2.10 fs_writeFAT() 116
5.35.2.11 isFileSystemMounted() 117
5.35.2.12 isValidFileName() 117
5.36 filesys.h 117
5.37 src/PennFAT/interface.c File Reference 118
5.37.1 Detailed Description 119
5.37.2 Function Documentation 119
5.37.2.1 f_close() 119
5.37.2.2 f_find() 119
5.37.2.3 f_isExecutable() 120
5.37.2.4 f_ls() 120
5.37.2.5 f_lseek() 121
5.37.2.6 f_open() 121
5.37.2.7 f_read() 121
5.37.2.8 f_unlink() 122
5.37.2.9 f_write() 122
5.38 src/PennFAT/interface.h File Reference 123
5.38.1 Detailed Description 123
5.38.2 Function Documentation 124
5.38.2.1 f_close() 124
5.38.2.2 f_find() 124
5.38.2.3 f_isExecutable() 124

5.38.2.4 f_ls() 125
5.38.2.5 f_lseek() 125
5.38.2.6 f_open() 126
5.38.2.7 f_read() 126
5.38.2.8 f_unlink() 126
5.38.2.9 f_write() 127
5.39 interface.h 127
5.40 src/PennFAT/pennFAT.c File Reference 128
5.40.1 Detailed Description 128
5.40.2 Function Documentation 129
5.40.2.1 parseInput() 129
5.40.2.2 pf_catFiles() 129
5.40.2.3 pf_chmod() 130
5.40.2.4 pf_isMounted() 130
5.40.2.5 pf_ls() 130
5.40.2.6 pf_mkfs() 130
5.40.2.7 pf_mount() 131
5.40.2.8 pf_mv() 131
5.40.2.9 pf_readFile() 132
5.40.2.10 pf_rm() 132
5.40.2.11 pf_touch() 132
5.40.2.12 pf_umount() 133
5.40.2.13 pf_writeFile() 133
5.40.2.14 readInput() 133
5.40.2.15 SIGINTHandler() 134
5.41 src/PennFAT/pennFAT.h File Reference 134
5.41.1 Detailed Description 135
5.41.2 Enumeration Type Documentation 136
5.41.2.1 PF_WRITEMODE 136
5.41.3 Function Documentation 136
5.41.3.1 parseInput() 136
5.41.3.2 pf_catFiles() 136
5.41.3.3 pf_chmod() 137
5.41.3.4 pf_isMounted() 137
5.41.3.5 pf_ls() 138
5.41.3.6 pf_mkfs() 138
5.41.3.7 pf_mount() 138
5.41.3.8 pf_mv() 139
5.41.3.9 pf_readFile() 139
5.41.3.10 pf_rm() 139
5.41.3.11 pf_touch() 140
5.41.3.12 pf_umount() 140

5.41.3.13 pf_writeFile() 140
5.41.3.14 readInput() 141
5.41.3.15 SIGINTHandler() 141
5.42 pennFAT.h 141
5.43 src/PennFAT/test-playground.c File Reference 142
5.43.1 Detailed Description 142

Index **143**

Chapter 1

PennOS (23sp-pennOS-group-24)

1.0.1 Authors

Name	PennKey
Ruifan Wang	wang321
Jiaqi Xie	jiaqixie
Zhiyuan Liang	liangzhy
Shuo Sun	[PennKey]

1.0.2 Source Files

```
 bin/                     # Compiled binaries
|    | pennFAT
|     pennOS
 src/                     # Source code
|    | kernel/
|     PennFAT/
 doc/                     # Documentation files
|     [Companion Document]
 log/                     # PennOS logs
|     log.txt
 README.md
```

1.0.3 Extra Credit Answers

- [If you completed any extra credit, include your answers here]

1.0.4 Compilation Instructions

- Compile by running (make sure you are in the root directory)
  `make`
- Run the PennOS
  `./bin/pennOS`
- Or the PennFAT
  `./bin/pennFAT`

1.0.5 Overview of Work Accomplished

All requirements of regular credit have been accomplished. We did not implement any feature of extra credit.

1.0.6 Description of Code and Code Layout

[Provide a detailed description of the code, including any algorithms or data structures used. Explain the layout of the code, including the purpose of each file and how they interact with each other.]

1.0.7 General Comments

[Include any general comments or observations about your code, including any challenges that you faced while working on the project. This can help us better understand your thought process and approach to the project.]

Chapter 2

Class Index

2.1 Class List

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

DirectoryEntry
The structure of the directory entry as stored in the filesystem is as follows: 7

FATConfig
The LSB(rightmost under little endian) of the first entry of the FAT specifies the block size with the mapping as below: {LSB : size in bytes} = {0:256; 1:512; 2:1024; 3:2048; 4:4096} The MSB(leftmost under little endian) of the first entry of the FAT specifies the number of blocks that FAT region occupies. The MSB should be ranged from 1-32 (numbers outside of this range will be considered an error). FAT region size = block size ∗ FAT region block number FAT entry number = FAT region size / FAT entry size (2-byte in FAT16) Data region size = block size ∗ (FAT entry number - 1) 8

FdNode
There are three open mode supported by PennFAT: F_WRITE, F_READ and F_APPEND. According to ed #953, each file can only be read/write exclusivly which means only one instance can open() a file at a time. Under F_APPEND mode, the fileOffset will be set to the end of the file initially and it can only be increased. Under F_WRITE/F_APPEND mode, if the fileOffset is set to the position beyond the file size, the file system will occupy the space for the gap. As a result, the size of the file will increase, however, the gap space may contain uninitialized contents. Under F_READ mode, if the fileOffset is set to the position beyond the file size, f_read() will read nothing 8

FdTable
The file descriptor table is a linked list of FdNode 9

Job 9

JobList 10

JobListNode 10

parsed_command 10

pcb
Attributes of PCB 11

pcb_node
Node in the pcb_queue 11

pcb_queue
A link list of pcb nodes 12

priority_queue
Ready queue including three pcb queues of different priorities 12

Chapter 3

File Index

3.1 File List

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

src/kernel/behavior.c 13
src/kernel/behavior.h 17
src/kernel/global.h 22
src/kernel/global2.h 23
src/kernel/job.c 23
src/kernel/job.h 33
src/kernel/kernel.h 45
src/kernel/log.h 45
src/kernel/parser.h 45
src/kernel/perrno.h
Error codes and messages in PennOS 46
src/kernel/programs.c 48
src/kernel/programs.h 54
src/kernel/scheduler.h 62
src/kernel/shell.c 62
src/kernel/shell.h 65
src/kernel/stress.h 67
src/kernel/user.h
User level functions 67
src/kernel/utils.h 73
src/PennFAT/FAT.c 75
src/PennFAT/FAT.h 85
src/PennFAT/fd-table.c 97
src/PennFAT/fd-table.h 101
src/PennFAT/filesys.c 106
src/PennFAT/filesys.h 112
src/PennFAT/interface.c 118
src/PennFAT/interface.h 123
src/PennFAT/pennFAT.c 128
src/PennFAT/pennFAT.h 134
src/PennFAT/test-playground.c 142
src/PennFAT/utils.c 72
src/PennFAT/utils.h 74

Chapter 4

Class Documentation

4.1 DirectoryEntry Struct Reference

The structure of the directory entry as stored in the filesystem is as follows:

```
#include <FAT.h>
```

Public Attributes

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

4.1.1 Detailed Description

The structure of the directory entry as stored in the filesystem is as follows:

- char name[32]: null-terminated file name name[0] also serves as a special marker: – 0: end of directory – 1: deleted entry; the file is also deleted – 2: deleted entry; the file is still being used
- uint32_t size: number of bytes in file
- uint16_t firstBlock: the first block number of the file (undefined if size is zero) The block index of data region starts from 1. If the firstBlock is 0, it means that this is an empty file which has not occupied any data region block.
- uint8_t type: the type of the file, which will be one of the following: – 0: unknown – 1: a regular file – 2: a directory file – 4: a symbolic link
- uint8_t perm: file permissions, which will be one of the following: – 0: none – 2: write only – 4: read only – 5: read and executable (shell scripts) – 6: read and write – 7: read, write, and executable
- time_t mtime: creation/modification time as returned by time(2) in Linux

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

- src/PennFAT/FAT.h

4.2 FATConfig Struct Reference

The LSB(rightmost under little endian) of the first entry of the FAT specifies the block size with the mapping as below: {LSB : size in bytes} = {0:256; 1:512; 2:1024; 3:2048; 4:4096} The MSB(leftmost under little endian) of the first entry of the FAT specifies the number of blocks that FAT region occupies. The MSB should be ranged from 1-32 (numbers outside of this range will be considered an error). FAT region size = block size ∗ FAT region block number FAT entry number = FAT region size / FAT entry size (2-byte in FAT16) Data region size = block size ∗ (FAT entry number - 1)

```
#include <FAT.h>
```

Public Attributes

- char **name** [MAX_FILE_NAME_LENGTH]
- uint16_t **LSB**
- uint16_t **MSB**
- int **blockSize**
- int **FATRegionBlockNum**
- int **FATRegionSize**
- int **FATEntryNum**
- int **dataRegionSize**
- int **FATSizeInMemory**

4.2.1 Detailed Description

The LSB(rightmost under little endian) of the first entry of the FAT specifies the block size with the mapping as below: {LSB : size in bytes} = {0:256; 1:512; 2:1024; 3:2048; 4:4096} The MSB(leftmost under little endian) of the first entry of the FAT specifies the number of blocks that FAT region occupies. The MSB should be ranged from 1-32 (numbers outside of this range will be considered an error). FAT region size = block size ∗ FAT region block number FAT entry number = FAT region size / FAT entry size (2-byte in FAT16) Data region size = block size ∗ (FAT entry number - 1)

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

- src/PennFAT/FAT.h

4.3 FdNode Struct Reference

There are three open mode supported by PennFAT: F_WRITE, F_READ and F_APPEND. According to ed #953, each file can only be read/write exclusivly which means only one instance can open() a file at a time. Under F_APPEND mode, the fileOffset will be set to the end of the file initially and it can only be increased. Under F_↩
WRITE/F_APPEND mode, if the fileOffset is set to the position beyond the file size, the file system will occupy the space for the gap. As a result, the size of the file will increase, however, the gap space may contain uninitialized contents. Under F_READ mode, if the fileOffset is set to the position beyond the file size, f_read() will read nothing.

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

Public Attributes

- int **openMode**
- int **directoryEntryOffset**
- int **fileOffset**
- struct FdNode ∗ **prev**
- struct FdNode ∗ **next**

4.3.1 Detailed Description

There are three open mode supported by PennFAT: F_WRITE, F_READ and F_APPEND. According to ed #953, each file can only be read/write exclusivly which means only one instance can open() a file at a time. Under F_APPEND mode, the fileOffset will be set to the end of the file initially and it can only be increased. Under F_↩ WRITE/F_APPEND mode, if the fileOffset is set to the position beyond the file size, the file system will occupy the space for the gap. As a result, the size of the file will increase, however, the gap space may contain uninitialized contents. Under F_READ mode, if the fileOffset is set to the position beyond the file size, f_read() will read nothing.

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

- src/PennFAT/fd-table.h

4.4 FdTable Struct Reference

The file descriptor table is a linked list of FdNode.

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

Public Attributes

- FdNode ∗ **head**
- FdNode ∗ **tail**

4.4.1 Detailed Description

The file descriptor table is a linked list of FdNode.

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

- src/PennFAT/fd-table.h

4.5 Job Struct Reference

Public Attributes

- struct parsed_command ∗ **cmd**
- pid_t **pid**
- JobState **state**

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

- src/kernel/utils.h

4.6 JobList Struct Reference

Public Attributes

- JobListNode ∗ **head**
- JobListNode ∗ **tail**
- int **jobCount**

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

- src/kernel/utils.h

4.7 JobListNode Struct Reference

Public Attributes

- Job ∗ **job**
- struct JobListNode ∗ **prev**
- struct JobListNode ∗ **next**
- int **jobId**

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

- src/kernel/utils.h

4.8 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** [ ]

4.8.1 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/kernel/parser.h

4.9 pcb Struct Reference

attributes of PCB

`#include <utils.h>`

Public Attributes

- ucontext_t **ucontext**
- pid_t **pid**
- pid_t **ppid**

  parent pid
- enum process_state **prev_state**
- enum process_state **state**

  state of the process
- int **priority**

  priority of the process
- FdNode ∗ **fds** [MAX_FILE_DESCRIPTOR]

  keep track of open FDs
- int **ticks_left**

  ticks left for sleep to be blocked
- struct pcb_queue ∗ **children**

  processes that have not completed yet
- struct pcb_queue ∗ **zombies**

  processes that are completed but the parent has not waited for it yet
- char ∗ **pname**

  name of the function
- bool **toWait**

  true indicates the parent need to wait for this child

4.9.1 Detailed Description

attributes of PCB

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

- src/kernel/utils.h

4.10 pcb_node Struct Reference

node in the pcb_queue

`#include <utils.h>`

Public Attributes

- pcb ∗ **pcb**
- struct pcb_node ∗ **next**

4.10.1 Detailed Description

node in the pcb_queue

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

- src/kernel/utils.h

4.11 pcb_queue Struct Reference

A link list of pcb nodes.

```
#include <utils.h>
```

Public Attributes

- pcb_node ∗ **head**
- pcb_node ∗ **tail**

4.11.1 Detailed Description

A link list of pcb nodes.

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

- src/kernel/utils.h

4.12 priority_queue Struct Reference

the ready queue including three pcb queues of different priorities

```
#include <utils.h>
```

Public Attributes

- pcb_queue ∗ **high**

  priority -1
- pcb_queue ∗ **mid**

  priority 0
- pcb_queue ∗ **low**

  priority 1

4.12.1 Detailed Description

the ready queue including three pcb queues of different priorities

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

- src/kernel/utils.h

Chapter 5

File Documentation

5.1 src/kernel/behavior.c File Reference

```
#include "behavior.h"
```

Functions

- void writePrompt ()

    Write the prompt to the screen.
- void readUserInput (char ∗∗line)

    Read the user input.
- LineType parseUserInput (char ∗line)

    Parse the user input.
- LineType readAndParseUserInput (char ∗∗line)

    Read and parse the user input.
- int parseLine (char ∗line, struct parsed_command ∗∗cmd)

    Parse the command line.
- ProgramType parseProgramType (struct parsed_command ∗cmd)

    Execute the command line.
- int executeLine (struct parsed_command ∗cmd, int priority)

    Execute the command line.
- pid_t executeProgram (ProgramType programType, char ∗∗argv, int fd_in, int fd_out)

    Execute the user program.
- void executeScript (char ∗argv[ ])

    Execute the user script.

5.1.1 Detailed Description

Author

Shuo Sun (`sunshuo@seas.upenn.edu`)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.1.2 Function Documentation

5.1.2.1 executeLine()

```
int executeLine (
             struct parsed_command * cmd,
             int priority )
```

Execute the command line.

Parameters

cmd	
priority	

Returns

int

5.1.2.2 executeProgram()

```
pid_t executeProgram (
             ProgramType programType,
             char ** argv,
             int fd_in,
             int fd_out )
```

Execute the user program.

Parameters

programType	
argv	
fd_in	
fd_out	

Returns

pid_t

5.1.2.3 executeScript()

```
void executeScript (
             char * argv[] )
```

Execute the user script.

Parameters

argv	

5.1.2.4 parseLine()

```
int parseLine (
             char * line,
             struct parsed_command ** cmd )
```

Parse the command line.

Parameters

line	
cmd	

Returns

int

5.1.2.5 parseProgramType()

```
ProgramType parseProgramType (
             struct parsed_command * cmd )
```

Execute the command line.

Parameters

cmd	

Returns

ProgramType

5.1.2.6 parseUserInput()

```
LineType parseUserInput (
            char * line )
```

Parse the user input.

Parameters

line	

Returns

LineType

5.1.2.7 readAndParseUserInput()

```
LineType readAndParseUserInput (
            char ** line )
```

Read and parse the user input.

Parameters

line	

Returns

LineType

5.1.2.8 readUserInput()

```
void readUserInput (
            char ** line )
```

Read the user input.

Parameters

line	

5.1.2.9 writePrompt()

```
void writePrompt ( )
```

Write the prompt to the screen.

5.2 src/kernel/behavior.h File Reference

```
#include "job.h"
#include "../PennFAT/interface.h"
#include "programs.h"
```

Macros

- #define **S_MAX_BUFFER_SIZE** 4096
- #define **PROMPT** "pennOS> "

Enumerations

- enum LineType { **S_EXIT_SHELL** , **S_EMPTY_LINE** , **S_EXECUTE_COMMAND** }

  The type of the command line.
- enum ProgramType {
  CAT , **SLEEP** , **BUSY** , **ECHO** ,
  LS , **TOUCH** , **MV** , **CP** ,
  RM , **CHMOD** , **PS** , **ZOMBIFY** ,
  ORPHANIFY , **HANG** , **NOHANG** , **RECUR** ,
  TEST , **UNKNOWN** }

  The type of the user program.

Functions

- void writePrompt ()

  Write the prompt to the screen.
- void readUserInput (char **line)

  Read the user input.
- LineType parseUserInput (char *line)

  Parse the user input.
- LineType readAndParseUserInput (char **line)

  Read and parse the user input.
- int parseLine (char *line, struct parsed_command **cmd)

Parse the command line.
- ProgramType parseProgramType (struct parsed_command ∗cmd)

    Execute the command line.
- int executeLine (struct parsed_command ∗cmd, int priority)

    Execute the command line.
- pid_t executeProgram (ProgramType programType, char ∗∗argv, int fd_in, int fd_out)

    Execute the user program.
- void executeScript (char ∗argv[ ])

    Execute the user script.

Variables

- JobList **_jobList**

5.2.1 Detailed Description

Author

Shuo Sun (`sunshuo@seas.upenn.edu`)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.2.2 Enumeration Type Documentation

5.2.2.1 LineType

`enum LineType`

The type of the command line.

5.2.2.2 ProgramType

`enum ProgramType`

The type of the user program.

5.2.3 Function Documentation

5.2.3.1 executeLine()

```
int executeLine (
             struct parsed_command * cmd,
             int priority )
```

Execute the command line.

Parameters

cmd	
priority	

Returns

int

5.2.3.2 executeProgram()

```
pid_t executeProgram (
             ProgramType programType,
             char ** argv,
             int fd_in,
             int fd_out )
```

Execute the user program.

Parameters

programType	
argv	
fd_in	
fd_out	

Returns

pid_t

5.2.3.3 executeScript()

```
void executeScript (
             char * argv[] )
```

Execute the user script.

Parameters

argv	

5.2.3.4 parseLine()

```
int parseLine (
            char * line,
            struct parsed_command ** cmd )
```

Parse the command line.

Parameters

line	
cmd	

Returns

int

5.2.3.5 parseProgramType()

```
ProgramType parseProgramType (
            struct parsed_command * cmd )
```

Execute the command line.

Parameters

cmd	

Returns

ProgramType

5.2.3.6 parseUserInput()

```
LineType parseUserInput (
            char * line )
```

Parse the user input.

Parameters

line	

Returns

LineType

5.2.3.7 readAndParseUserInput()

```
LineType readAndParseUserInput (
             char ** line )
```

Read and parse the user input.

Parameters

line	

Returns

LineType

5.2.3.8 readUserInput()

```
void readUserInput (
             char ** line )
```

Read the user input.

Parameters

line	

5.2.3.9 writePrompt()

```
void writePrompt ( )
```

Write the prompt to the screen.

5.3 behavior.h

Go to the documentation of this file.

```c

#ifndef BEHAVIOR_H
#define BEHAVIOR_H

#define S_MAX_BUFFER_SIZE 4096
#define PROMPT "pennOS> "

#include "job.h"
#include "../PennFAT/interface.h"
#include "programs.h"
typedef enum {
    S_EXIT_SHELL,
    S_EMPTY_LINE,
    S_EXECUTE_COMMAND
} LineType;
typedef enum {
    CAT,
    SLEEP,
    BUSY,
    ECHO,
    LS,
    TOUCH,
    MV,
    CP,
    RM,
    CHMOD,
    PS,
    ZOMBIFY,
    ORPHANIFY,
    HANG,
    NOHANG,
    RECUR,
    TEST,
    UNKNOWN
} ProgramType;

extern JobList _jobList; // store all background job

/* Utility function for writing PROMPT */
void writePrompt();

/* Read and parse utilities */
void readUserInput(char **line);
LineType parseUserInput(char *line);
LineType readAndParseUserInput(char **line);
int parseLine(char *line, struct parsed_command **cmd);


/* Execute a user program */
ProgramType parseProgramType(struct parsed_command *cmd);
int executeLine(struct parsed_command *cmd, int priority);
pid_t executeProgram(ProgramType programType, char **argv, int fd_in, int fd_out);
void executeScript(char *argv[]);



#endif
```

5.4 global.h

```c
#ifndef GLOBAL_H
#define GLOBAL_H

#include "utils.h"

#define HIGH -1
#define MID 0
#define LOW 1

#define SUCCESS 0
#define FAILURE -1

// according to https://chromium.googlesource.com/chromiumos/docs/+/master/constants/signals.md
#define S_SIGSTOP 19
#define S_SIGCONT 18
#define S_SIGTERM 15

#endif
```

5.5 global2.h

```
#ifndef GLOBAL2_H
#define GLOBAL2_H

typedef enum process_state {
    RUNNING,
    READY,
    BLOCKED,
    STOPPED,
    ZOMBIED,
    ORPHANED,
    TERMINATED,     // terminated by signal
    EXITED,         // terminated normally
} process_state;

#endif
```

5.6 src/kernel/job.c File Reference

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

Functions

- void printCommandLine (struct parsed_command ∗cmd)

    The state of the job.
- void writeJobStatePrompt (JobState state)

    Write the job state and prompt to the screen.
- void writeJobState (Job ∗job)

    Write the job state to the screen.
- void writeNewline ()

    Write the a new line to the screen.
- void printJobList (JobList ∗jobList)

    Print the job list.
- CommandType parseBuiltinCommandType (struct parsed_command ∗cmd)

    Parse the built-in command type.
- void killBuildinCommand (struct parsed_command ∗cmd)

    Built-in command: kill.
- CommandType executeBuiltinCommand (struct parsed_command ∗cmd)

    Execute the built-in command.
- void jobsBuildinCommand ()

    Built-in command: jobs.
- void bgBuildinCommand (struct parsed_command ∗cmd)

    Built-in command: bg.
- void fgBuildinCommand (struct parsed_command ∗cmd)

    Built-in command: fg.
- void nicePidBuildinCommand (struct parsed_command ∗cmd)

    Built-in command: nice.
- void manBuildinCommand ()

    Built-in command: man.
- Job ∗ createJob (struct parsed_command ∗cmd, pid_t pid, JobState state)

    Create a Job object.
- void initJobList (JobList ∗jobList)

    Initialize the job list.

- void appendJobList (JobList ∗jobList, Job ∗job)

  Append a job to the job list.
- Job ∗ findJobList (JobList ∗jobList, pid_t pid)

  Find a job in the job list.
- Job ∗ updateJobList (JobList ∗jobList, pid_t pid, JobState state)

  Update the state of a job in the job list.
- int removeJobList (JobList ∗jobList, pid_t pid)

  Remove a job from the job list.
- Job ∗ findJobListByJobId (JobList ∗jobList, int jobId)

  Find a job in the job list by job id.
- Job ∗ updateJobListByJobId (JobList ∗jobList, int jobId, JobState state)

  Update the state of a job in the job list by job id.
- int removeJobListByJobId (JobList ∗jobList, int jobId)

  Remove a job from the job list by job id.
- int removeJobListWithoutFreeCmd (JobList ∗jobList, pid_t pid)

  Remove a job from the job list without free the cmd.
- Job ∗ popJobList (JobList ∗jobList, pid_t pid)

  Pop a job from the job list.
- void clearJobList (JobList ∗jobList)

  Clear the job list.
- void pollBackgroundProcesses ()

  Poll the background processes.
- Job ∗ findTheCurrentJob (JobList ∗jobList)

  Find the current job.

5.6.1 Detailed Description

Author

Shuo Sun (`sunshuo@seas.upenn.edu`)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.6.2 Function Documentation

5.6.2.1 appendJobList()

```
void appendJobList (
            JobList ∗ jobList,
            Job ∗ job )
```

Append a job to the job list.

Parameters

jobList	
job	

5.6.2.2 bgBuildinCommand()

```
void bgBuildinCommand (
             struct parsed_command * cmd )
```

Built-in command: bg.

Parameters

cmd	

5.6.2.3 clearJobList()

```
void clearJobList (
             JobList * jobList )
```

Clear the job list.

Parameters

jobList	

5.6.2.4 createJob()

```
Job * createJob (
             struct parsed_command * cmd,
             pid_t pid,
             JobState state )
```

Create a Job object.

Parameters

cmd	
pid	
state	

Returns

Job∗

5.6.2.5 executeBuiltinCommand()

```
CommandType executeBuiltinCommand (
             struct parsed_command ∗ cmd )
```

Execute the built-in command.

Parameters

cmd	

Returns

CommandType

5.6.2.6 fgBuildinCommand()

```
void fgBuildinCommand (
             struct parsed_command ∗ cmd )
```

Built-in command: fg.

Parameters

cmd	

5.6.2.7 findJobList()

```
Job ∗ findJobList (
             JobList ∗ jobList,
             pid_t pid )
```

Find a job in the job list.

Parameters

jobList	
pid	

Returns

Job∗

5.6.2.8 findJobListByJobId()

```
Job * findJobListByJobId (
            JobList * jobList,
            int jobId )
```

Find a job in the job list by job id.

Parameters

jobList	
jobId	

Returns

Job∗

5.6.2.9 findTheCurrentJob()

```
Job * findTheCurrentJob (
            JobList * jobList )
```

Find the current job.

Parameters

jobList	

Returns

Job∗

5.6.2.10 initJobList()

```
void initJobList (
            JobList * jobList )
```

Initialize the job list.

Parameters

jobList	

5.6.2.11 jobsBuildinCommand()

```
void jobsBuildinCommand ( )
```

Built-in command: jobs.

5.6.2.12 killBuildinCommand()

```
void killBuildinCommand (
             struct parsed_command * cmd )
```

Built-in command: kill.

Parameters

cmd	

5.6.2.13 manBuildinCommand()

```
void manBuildinCommand ( )
```

Built-in command: man.

5.6.2.14 nicePidBuildinCommand()

```
void nicePidBuildinCommand (
             struct parsed_command * cmd )
```

Built-in command: nice.

Parameters

cmd	

5.6.2.15 parseBuiltinCommandType()

```
CommandType parseBuiltinCommandType (
             struct parsed_command * cmd )
```

Parse the built-in command type.

Parameters

cmd	

Returns

CommandType

5.6.2.16 pollBackgroundProcesses()

```
void pollBackgroundProcesses ( )
```

Poll the background processes.

5.6.2.17 popJobList()

```
Job * popJobList (
             JobList * jobList,
             pid_t pid )
```

Pop a job from the job list.

Parameters

jobList	
pid	

Returns

Job∗

5.6.2.18 printCommandLine()

```
void printCommandLine (
             struct parsed_command * cmd )
```

The state of the job.

Parameters

cmd	

5.6.2.19 printJobList()

```
void printJobList (
            JobList * jobList )
```

Print the job list.

Parameters

jobList	

5.6.2.20 removeJobList()

```
int removeJobList (
            JobList * jobList,
            pid_t pid )
```

Remove a job from the job list.

Parameters

jobList	
pid	

Returns

int

5.6.2.21 removeJobListByJobId()

```
int removeJobListByJobId (
            JobList * jobList,
            int jobId )
```

Remove a job from the job list by job id.

Parameters

jobList	
jobId	

Returns

int

5.6.2.22 removeJobListWithoutFreeCmd()

```
int removeJobListWithoutFreeCmd (
            JobList * jobList,
            pid_t pid )
```

Remove a job from the job list without free the cmd.

Parameters

jobList	
pid	

Returns

int

5.6.2.23 updateJobList()

```
Job * updateJobList (
            JobList * jobList,
            pid_t pid,
            JobState state )
```

Update the state of a job in the job list.

Parameters

jobList	
pid	
state	

Returns

Job∗

5.6.2.24 updateJobListByJobId()

```
Job * updateJobListByJobId (
            JobList * jobList,
            int jobId,
            JobState state )
```

Update the state of a job in the job list by job id.

Parameters

jobList	
jobId	
state	

Returns

Job∗

5.6.2.25 writeJobState()

```
void writeJobState (
            Job * job )
```

Write the job state to the screen.

Parameters

job	

5.6.2.26 writeJobStatePrompt()

```
void writeJobStatePrompt (
            JobState state )
```

Write the job state and prompt to the screen.

Parameters

state	

5.6.2.27 writeNewline()

```
void writeNewline ( )
```

Write the a new line to the screen.

5.7 src/kernel/job.h File Reference

```
#include "user.h"
#include "utils.h"
#include "parser.h"
#include "unistd.h"
#include "../PennFAT/interface.h"
#include "programs.h"
```

Functions

- void printCommandLine (struct parsed_command ∗cmd)

  The state of the job.
- void writeJobStatePrompt (JobState state)

  Write the job state and prompt to the screen.
- void writeJobState (Job ∗job)

  Write the job state to the screen.
- void writeNewline ()

  Write the a new line to the screen.
- Job ∗ createJob (struct parsed_command ∗cmd, pid_t pid, JobState state)

  Create a Job object.
- void initJobList (JobList ∗jobList)

  Initialize the job list.
- void appendJobList (JobList ∗jobList, Job ∗job)

  Append a job to the job list.
- Job ∗ findJobList (JobList ∗jobList, pid_t pid)

  Find a job in the job list.
- Job ∗ updateJobList (JobList ∗jobList, pid_t pid, JobState state)

  Update the state of a job in the job list.
- int removeJobList (JobList ∗jobList, pid_t pid)

  Remove a job from the job list.
- Job ∗ findJobListByJobId (JobList ∗jobList, int jobId)

  Find a job in the job list by job id.
- Job ∗ updateJobListByJobId (JobList ∗jobList, int jobId, JobState state)

  Update the state of a job in the job list by job id.
- int removeJobListByJobId (JobList ∗jobList, int jobId)

  Remove a job from the job list by job id.
- int removeJobListWithoutFreeCmd (JobList ∗jobList, pid_t pid)

  Remove a job from the job list without free the cmd.
- Job ∗ popJobList (JobList ∗jobList, pid_t pid)

  Pop a job from the job list.
- Job ∗ findTheCurrentJob (JobList ∗jobList)

Find the current job.
- void printJobList (JobList ∗jobList)

  Print the job list.
- CommandType parseBuiltinCommandType (struct parsed_command ∗cmd)

  Parse the built-in command type.
- CommandType executeBuiltinCommand (struct parsed_command ∗cmd)

  Execute the built-in command.
- void clearJobList (JobList ∗jobList)

  Clear the job list.
- void pollBackgroundProcesses ()

  Poll the background processes.
- void bgBuildinCommand (struct parsed_command ∗cmd)

  Built-in command: bg.
- void fgBuildinCommand (struct parsed_command ∗cmd)

  Built-in command: fg.
- void jobsBuildinCommand ()

  Built-in command: jobs.
- void nicePidBuildinCommand (struct parsed_command ∗cmd)

  Built-in command: nice.
- void killBuildinCommand (struct parsed_command ∗cmd)

  Built-in command: kill.
- void manBuildinCommand ()

  Built-in command: man.

Variables

- JobList **_jobList**
- pid_t **fgPid**

5.7.1 Detailed Description

Author

Shuo Sun (sunshuo@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.7.2 Function Documentation

5.7.2.1 appendJobList()

```
void appendJobList (
             JobList * jobList,
             Job * job )
```

Append a job to the job list.

Parameters

jobList	
job	

5.7.2.2 bgBuildinCommand()

```
void bgBuildinCommand (
             struct parsed_command * cmd )
```

Built-in command: bg.

Parameters

cmd	

5.7.2.3 clearJobList()

```
void clearJobList (
             JobList * jobList )
```

Clear the job list.

Parameters

jobList	

5.7.2.4 createJob()

```
Job * createJob (
            struct parsed_command * cmd,
            pid_t pid,
            JobState state )
```

Create a Job object.

Parameters

cmd	
pid	
state	

Returns

Job*

5.7.2.5 executeBuiltinCommand()

```
CommandType executeBuiltinCommand (
            struct parsed_command * cmd )
```

Execute the built-in command.

Parameters

cmd	

Returns

CommandType

5.7.2.6 fgBuildinCommand()

```
void fgBuildinCommand (
            struct parsed_command * cmd )
```

Built-in command: fg.

Parameters

cmd	

5.7.2.7 findJobList()

```
Job * findJobList (
            JobList * jobList,
            pid_t pid )
```

Find a job in the job list.

Parameters

jobList	
pid	

Returns

Job∗

5.7.2.8 findJobListByJobId()

```
Job * findJobListByJobId (
            JobList * jobList,
            int jobId )
```

Find a job in the job list by job id.

Parameters

jobList	
jobId	

Returns

Job∗

5.7.2.9 findTheCurrentJob()

```
Job * findTheCurrentJob (
            JobList * jobList )
```

Find the current job.

Parameters

jobList	

Returns

Job∗

5.7.2.10 initJobList()

```
void initJobList (
             JobList ∗ jobList )
```

Initialize the job list.

Parameters

jobList	

5.7.2.11 jobsBuildinCommand()

```
void jobsBuildinCommand ( )
```

Built-in command: jobs.

5.7.2.12 killBuildinCommand()

```
void killBuildinCommand (
             struct parsed_command ∗ cmd )
```

Built-in command: kill.

Parameters

cmd	

5.7.2.13 manBuildinCommand()

```
void manBuildinCommand ( )
```

Built-in command: man.

5.7.2.14 nicePidBuildinCommand()

```
void nicePidBuildinCommand (
             struct parsed_command * cmd )
```

Built-in command: nice.

Parameters

cmd	

5.7.2.15 parseBuiltinCommandType()

```
CommandType parseBuiltinCommandType (
             struct parsed_command * cmd )
```

Parse the built-in command type.

Parameters

cmd	

Returns

CommandType

5.7.2.16 pollBackgroundProcesses()

```
void pollBackgroundProcesses ( )
```

Poll the background processes.

5.7.2.17 popJobList()

```
Job * popJobList (
             JobList * jobList,
             pid_t pid )
```

Pop a job from the job list.

Parameters

jobList	
pid	

Returns

Job∗

5.7.2.18 printCommandLine()

```
void printCommandLine (
             struct parsed_command ∗ cmd )
```

The state of the job.

Parameters

cmd	

5.7.2.19 printJobList()

```
void printJobList (
             JobList ∗ jobList )
```

Print the job list.

Parameters

jobList	

5.7.2.20 removeJobList()

```
int removeJobList (
             JobList ∗ jobList,
             pid_t pid )
```

Remove a job from the job list.

Parameters

jobList	
pid	

Returns

int

5.7.2.21 removeJobListByJobId()

```
int removeJobListByJobId (
            JobList * jobList,
            int jobId )
```

Remove a job from the job list by job id.

Parameters

jobList	
jobId	

Returns

int

5.7.2.22 removeJobListWithoutFreeCmd()

```
int removeJobListWithoutFreeCmd (
            JobList * jobList,
            pid_t pid )
```

Remove a job from the job list without free the cmd.

Parameters

jobList	
pid	

Returns

int

5.7.2.23 updateJobList()

```
Job * updateJobList (
            JobList * jobList,
            pid_t pid,
            JobState state )
```

Update the state of a job in the job list.

Parameters

jobList	
pid	
state	

Returns

Job∗

5.7.2.24 updateJobListByJobId()

```
Job * updateJobListByJobId (
             JobList * jobList,
             int jobId,
             JobState state )
```

Update the state of a job in the job list by job id.

Parameters

jobList	
jobId	
state	

Returns

Job∗

5.7.2.25 writeJobState()

```
void writeJobState (
             Job * job )
```

Write the job state to the screen.

Parameters

job	

5.7.2.26 writeJobStatePrompt()

```
void writeJobStatePrompt (
            JobState state )
```

Write the job state and prompt to the screen.

Parameters

state	

5.7.2.27 writeNewline()

```
void writeNewline ( )
```

Write the a new line to the screen.

5.8 job.h

Go to the documentation of this file.

```

#ifndef JOB_H
#define JOB_H

#include "user.h"
#include "utils.h"
#include "parser.h"
#include "unistd.h"
#include "../PennFAT/interface.h"
#include "programs.h"

extern JobList _jobList; // store all background job
extern pid_t fgPid; // running foreground process

/* Utility functions for writing job state */
void printCommandLine(struct parsed_command *cmd);
void writeJobStatePrompt(JobState state);
void writeJobState(Job *job);
void writeNewline();

/* Utility functions for job and job list */
Job *createJob(struct parsed_command *cmd, pid_t pid, JobState state);
void initJobList(JobList *jobList);
void appendJobList(JobList *jobList, Job *job);
Job *findJobList(JobList *jobList, pid_t pid);
Job *updateJobList(JobList *jobList, pid_t pid, JobState state);
int removeJobList(JobList *jobList, pid_t pid);

Job *findJobListByJobId(JobList *jobList, int jobId);
Job *updateJobListByJobId(JobList *jobList, int jobId, JobState state);
int removeJobListByJobId(JobList *jobList, int jobId);
int removeJobListWithoutFreeCmd(JobList *jobList, pid_t pid);
Job *popJobList(JobList *jobList, pid_t pid);
Job *findTheCurrentJob(JobList *jobList);

void printJobList(JobList *jobList);

/* Built-in commands */
CommandType parseBuiltinCommandType(struct parsed_command *cmd);
CommandType executeBuiltinCommand(struct parsed_command *cmd);


void clearJobList(JobList *jobList);
void pollBackgroundProcesses();

void bgBuildinCommand(struct parsed_command *cmd);
void fgBuildinCommand(struct parsed_command *cmd);
void jobsBuildinCommand();
void nicePidBuildinCommand(struct parsed_command *cmd);
void killBuildinCommand(struct parsed_command *cmd);
void manBuildinCommand();


#endif
```

5.9 kernel.h

```
#ifndef KERNEL_H
#define KERNEL_H

#include <string.h>
#include <unistd.h>
#include <signal.h>
#include "utils.h"
#include "global.h"
#include "job.h"
#include "log.h"

extern ucontext_t scheduler_context;
extern pid_t lastPID;
extern priority_queue* ready_queue;
extern bool stopped_by_timer;
extern pcb_queue* exited_queue;
extern pcb_queue* stopped_queue;

extern JobList _jobList;
extern pid_t fgPid;

pcb *k_process_create(pcb * parent);

int k_process_kill(pcb *process, int signal);

int k_process_cleanup(pcb *process);

int kernel_init();

void kernel_deconstruct();

int block_process(pid_t pid);

int process_unblock(pid_t pid);

int clean_orphan(pcb * process);

#endif
```

5.10 log.h

```
#ifndef LOG_H
#define LOG_H

#include <stdio.h>
#include "global.h"
#include "utils.h"

extern int tick_tracker;

int log_init(const char *filename);

void log_event(pcb* pcb, char* action);

void log_pnice(pcb* pcb, int new);

void log_cleanup();

#endif
```

5.11 parser.h

```
/* Penn-Shell Parser
```

```
   hanbangw, 21fa    */

#pragma once

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

/* Here defines all possible parser errors */
// parser encountered an unexpected file input token '<'
#define UNEXPECTED_FILE_INPUT 1

// parser encountered an unexpected file output token '>'
#define UNEXPECTED_FILE_OUTPUT 2

// parser encountered an unexpected pipeline token '|'
#define UNEXPECTED_PIPELINE 3

// parser encountered an unexpected ampersand token '&'
#define UNEXPECTED_AMPERSAND 4

// parser didn't find input filename following '<'
#define EXPECT_INPUT_FILENAME 5

// parser didn't find output filename following '>' or '»'
#define EXPECT_OUTPUT_FILENAME 6

// parser didn't find any commands or arguments where it expects one
#define EXPECT_COMMANDS 7

struct parsed_command {
    // indicates the command shall be executed in background
    // (ends with an ampersand '&')
    bool is_background;

    // indicates if the stdout_file shall be opened in append mode
    // ignore this value when stdout_file is NULL
    bool is_file_append;

    // filename for redirecting input from
    const char *stdin_file;

    // filename for redirecting output to
    const char *stdout_file;

    // number of commands (pipeline stages)
    size_t num_commands;

    // an array to a list of arguments
    // size of `commands` is `num_commands`
    char **commands[];
};

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


/* This is a debugging function used for outputting a parsed command line. */
void print_parsed_command(const struct parsed_command *cmd);
```

5.12 src/kernel/perrno.h File Reference

Error codes and messages in PennOS.

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

Macros

- #define **P_SUCCESS** 0

  error codes in PennOS
- #define **P_NO_SUCH_PROCESS** -1
- #define **P_NO_SUCH_FILE** -2
- #define **P_FAIL_TO_INITIALIZE_QUEUE** -3

- #define **P_SHOULD_NOT_KILL_SHELL** -4
- #define **P_NODE_IS_NULL** -5
- #define **P_PARENT_IS_NULL** -6
- #define **P_PROCESS_IS_NULL** -7
- #define **P_PROCESS_NOT_IN_READY_QUEUE** -8
- #define **P_PROCESS_NOT_IN_STOPPED_QUEUE** -9
- #define **P_FAIL_TO_CLEANUP** -10

Functions

- void p_set_errno (int errnum)

  set p_error to specified error number
- void p_perror (const char ∗error_msg)

  log error message and set error number
- void **p_reset_errno** ()

  reset the error number to 0

5.12.1 Detailed Description

Error codes and messages in PennOS.

5.12.2 Function Documentation

5.12.2.1 p_perror()

```
void p_perror (
             const char * error_msg )
```

log error message and set error number

Parameters

msg	error message that user wants to display

5.12.2.2 p_set_errno()

```
void p_set_errno (
             int errnum )
```

set p_error to specified error number

Parameters

err_num	error number

5.13 perrno.h

Go to the documentation of this file.

```
#ifndef PERRNO_H
#define PERRNO_H

#include "stdio.h"
#include "errno.h"

#define P_SUCCESS 0
#define P_NO_SUCH_PROCESS -1
#define P_NO_SUCH_FILE -2
#define P_FAIL_TO_INITIALIZE_QUEUE -3
#define P_SHOULD_NOT_KILL_SHELL -4
#define P_NODE_IS_NULL -5
#define P_PARENT_IS_NULL -6
#define P_PROCESS_IS_NULL -7
#define P_PROCESS_NOT_IN_READY_QUEUE -8
#define P_PROCESS_NOT_IN_STOPPED_QUEUE -9
#define P_FAIL_TO_CLEANUP -10

void p_set_errno(int errnum);

void p_perror(const char* error_msg);

void p_reset_errno();

#endif
```

5.14 src/kernel/programs.c File Reference

```
#include "programs.h"
```

Functions

- int **argc** (char ∗argv[ ])
- void s_cat (char ∗argv[ ])

  Shell cat command.
- void s_sleep (char ∗argv[ ])

  Shell sleep command.
- void s_busy (char ∗argv[ ])

  Shell busy command.
- void s_echo (char ∗argv[ ])

  Shell echo command.
- void s_ls (char ∗argv[ ])

  Shell ls command.
- void s_touch (char ∗argv[ ])

  Shell touch command.
- void s_mv (char ∗argv[ ])

  Shell mv command.
- void s_cp (char ∗argv[ ])

Shell cp command.
- void s_rm (char ∗argv[ ])

  Shell rm command.
- void s_chmod (char ∗argv[ ])

  Shell chmod command.
- void s_ps (char ∗argv[ ])

  Shell ps command.
- void s_kill (char ∗argv[ ])

  Shell kill command.
- void **zombie_child** ()
- void **orphan_child** ()
- void s_zombify (char ∗argv[ ])

  Shell zombify command.
- void s_orphanify (char ∗argv[ ])

  Shell orphanify command.
- void s_hang (char ∗argv[ ])

  Shell hang command.
- void s_nohang (char ∗argv[ ])

  Shell nohang command.
- void s_recur (char ∗argv[ ])

  Shell recur command.
- void s_test (char ∗argv[ ])

  Our shell test command.

5.14.1 Detailed Description

Author

Shuo Sun (sunshuo@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.14.2 Function Documentation

5.14.2.1 s_busy()

```
void s_busy (
            char * argv[ ] )
```

Shell busy command.

Parameters

argv	

5.14.2.2 s_cat()

```
void s_cat (
             char * argv[] )
```

Shell cat command.

Parameters

argv	

5.14.2.3 s_chmod()

```
void s_chmod (
             char * argv[] )
```

Shell chmod command.

Parameters

argv	

5.14.2.4 s_cp()

```
void s_cp (
             char * argv[] )
```

Shell cp command.

Parameters

argv	

5.14.2.5 s_echo()

```
void s_echo (
             char * argv[] )
```

Shell echo command.

Parameters

argv	

5.14.2.6 s_hang()

```
void s_hang (
             char * argv[] )
```

Shell hang command.

Parameters

argv	

5.14.2.7 s_kill()

```
void s_kill (
             char * argv[] )
```

Shell kill command.

Parameters

argv	

5.14.2.8 s_ls()

```
void s_ls (
             char * argv[] )
```

Shell ls command.

Parameters

argv	

5.14.2.9 s_mv()

```
void s_mv (
            char * argv[] )
```

Shell mv command.

Parameters

argv	

5.14.2.10 s_nohang()

```
void s_nohang (
            char * argv[] )
```

Shell nohang command.

Parameters

argv	

5.14.2.11 s_orphanify()

```
void s_orphanify (
            char * argv[] )
```

Shell orphanify command.

Parameters

argv	

5.14.2.12 s_ps()

```
void s_ps (
            char * argv[] )
```

Shell ps command.

Parameters

argv	

5.14.2.13 s_recur()

```
void s_recur (
            char * argv[] )
```

Shell recur command.

Parameters

argv	

5.14.2.14 s_rm()

```
void s_rm (
            char * argv[] )
```

Shell rm command.

Parameters

argv	

5.14.2.15 s_sleep()

```
void s_sleep (
            char * argv[] )
```

Shell sleep command.

Parameters

argv	

5.14.2.16 s_test()

```
void s_test (
            char * argv[] )
```

Our shell test command.

Parameters

argv	

5.14.2.17 s_touch()

```
void s_touch (
            char * argv[] )
```

Shell touch command.

Parameters

argv	

5.14.2.18 s_zombify()

```
void s_zombify (
            char * argv[] )
```

Shell zombify command.

Parameters

argv	

5.15 src/kernel/programs.h File Reference

```
#include "behavior.h"
```

```
#include "stress.h"
#include "../PennFAT/pennFAT.h"
```

Macros

- #define **READ_BUFFER_SIZE** 4096

Functions

- void s_cat (char ∗argv[ ])

  Shell cat command.
- void s_sleep (char ∗argv[ ])

  Shell sleep command.
- void s_busy (char ∗argv[ ])

  Shell busy command.
- void s_echo (char ∗argv[ ])

  Shell echo command.
- void s_ls (char ∗argv[ ])

  Shell ls command.
- void s_touch (char ∗argv[ ])

  Shell touch command.
- void s_mv (char ∗argv[ ])

  Shell mv command.
- void s_cp (char ∗argv[ ])

  Shell cp command.
- void s_rm (char ∗argv[ ])

  Shell rm command.
- void s_chmod (char ∗argv[ ])

  Shell chmod command.
- void s_ps (char ∗argv[ ])

  Shell ps command.
- void s_kill (char ∗argv[ ])

  Shell kill command.
- void s_zombify (char ∗argv[ ])

  Shell zombify command.
- void s_orphanify (char ∗argv[ ])

  Shell orphanify command.
- void s_hang (char ∗argv[ ])

  Shell hang command.
- void s_nohang (char ∗argv[ ])

  Shell nohang command.
- void s_recur (char ∗argv[ ])

  Shell recur command.
- void s_test (char ∗argv[ ])

  Our shell test command.

5.15.1 Detailed Description

Author

Shuo Sun (sunshuo@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.15.2 Function Documentation

5.15.2.1 s_busy()

```
void s_busy (
            char * argv[] )
```

Shell busy command.

Parameters

argv	

5.15.2.2 s_cat()

```
void s_cat (
            char * argv[] )
```

Shell cat command.

Parameters

argv	

5.15.2.3 s_chmod()

```
void s_chmod (
             char * argv[] )
```

Shell chmod command.

Parameters

argv	

5.15.2.4 s_cp()

```
void s_cp (
             char * argv[] )
```

Shell cp command.

Parameters

argv	

5.15.2.5 s_echo()

```
void s_echo (
             char * argv[] )
```

Shell echo command.

Parameters

argv	

5.15.2.6 s_hang()

```
void s_hang (
             char * argv[] )
```

Shell hang command.

Parameters

argv	

5.15.2.7 s_kill()

```
void s_kill (
             char * argv[] )
```

Shell kill command.

Parameters

argv	

5.15.2.8 s_ls()

```
void s_ls (
             char * argv[] )
```

Shell ls command.

Parameters

argv	

5.15.2.9 s_mv()

```
void s_mv (
             char * argv[] )
```

Shell mv command.

Parameters

argv	

5.15.2.10 s_nohang()

```
void s_nohang (
            char * argv[] )
```

Shell nohang command.

Parameters

argv	

5.15.2.11 s_orphanify()

```
void s_orphanify (
            char * argv[] )
```

Shell orphanify command.

Parameters

argv	

5.15.2.12 s_ps()

```
void s_ps (
            char * argv[] )
```

Shell ps command.

Parameters

argv	

5.15.2.13 s_recur()

```
void s_recur (
            char * argv[] )
```

Shell recur command.

Parameters

argv	

5.15.2.14 s_rm()

```
void s_rm (
             char * argv[] )
```

Shell rm command.

Parameters

argv	

5.15.2.15 s_sleep()

```
void s_sleep (
             char * argv[] )
```

Shell sleep command.

Parameters

argv	

5.15.2.16 s_test()

```
void s_test (
             char * argv[] )
```

Our shell test command.

Parameters

argv	

5.15.2.17 s_touch()

```
void s_touch (
             char * argv[ ] )
```

Shell touch command.

Parameters

argv	

5.15.2.18 s_zombify()

```
void s_zombify (
             char * argv[ ] )
```

Shell zombify command.

Parameters

argv	

5.16 programs.h

Go to the documentation of this file.

```
#ifndef PROGRAMS_H
#define PROGRAMS_H

#include "behavior.h"
#include "stress.h"
#include "../PennFAT/pennFAT.h"

#define READ_BUFFER_SIZE 4096

/* Known user programs */
void s_cat(char *argv[]);
void s_sleep(char *argv[]);
void s_busy(char *argv[]);
void s_echo(char *argv[]);
void s_ls(char *argv[]);
void s_touch(char *argv[]);
void s_mv(char *argv[]);
void s_cp(char *argv[]);
void s_rm(char *argv[]);
void s_chmod(char *argv[]);
void s_ps(char *argv[]);
void s_kill(char *argv[]);
void s_zombify(char *argv[]);
void s_orphanify(char *argv[]);
void s_hang(char *argv[]);
void s_nohang(char *argv[]);
void s_recur(char *argv[]);
void s_test(char *argv[]);

#endif
```

5.17 scheduler.h

```
#ifndef SCHEDULER_H
#define SCHEDULER_H

#include <signal.h>    // sigaction, sigemptyset, sigfillset, signal
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>  // setitimer
#include <ucontext.h>  // getcontext, makecontext, setcontext, swapcontext
#include <valgrind/valgrind.h>

#include "utils.h"
#include "global.h"
#include "kernel.h"
#include "log.h"

extern pcb* active_process;
extern priority_queue* ready_queue;
extern ucontext_t main_context;
extern ucontext_t scheduler_context;
extern ucontext_t* p_active_context;
extern bool stopped_by_timer;
extern pcb_queue* exited_queue;
extern pcb_queue* stopped_queue;
extern int tick_tracker;

#define TICK 100000     // 1 tick = 0.1s

int set_alarm_handler();

void alarm_handler();

int set_timer();

pcb* next_process();

pcb_node* get_node_by_pid_all_queues(pid_t pid);

void scheduler();

void idle_func();

int scheduler_init();

int idle_process_init();    // initialize the idle process

int haveChildrenToWait(pcb *process);

void deconstruct_idle();

#endif
```

5.18 src/kernel/shell.c File Reference

```
#include "shell.h"
#include "scheduler.h"
```

Functions

- void shell_process ()

  The main function of shell.
- bool isBuildinCommand (struct parsed_command ∗cmd)

  Check if the command is a buildin command.
- bool isKnownProgram (struct parsed_command ∗cmd)

  Check if the command is a known program.
- int shell_init (int argc, const char ∗∗argv)

  Initialize the shell.

Variables

- JobList **_jobList**
- pid_t **fgPid** = 1

5.18.1 Detailed Description

Author

Shuo Sun (sunshuo@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.18.2 Function Documentation

5.18.2.1 isBuildinCommand()

```
bool isBuildinCommand (
              struct parsed_command * cmd )
```

Check if the command is a buildin command.

Parameters

cmd	

Returns

true

false

5.18.2.2 isKnownProgram()

```
bool isKnownProgram (
            struct parsed_command * cmd )
```

Check if the command is a known program.

Parameters

cmd	

Returns

true

false

5.18.2.3 shell_init()

```
int shell_init (
            int argc,
            const char ** argv )
```

Initialize the shell.

Parameters

argc	
argv	

Returns

int

5.18.2.4 shell_process()

```
void shell_process ( )
```

The main function of shell.

5.19 src/kernel/shell.h File Reference

```
#include "utils.h"
#include "job.h"
#include "user.h"
#include "behavior.h"
#include "log.h"
#include "../PennFAT/filesys.h"
#include "../PennFAT/interface.h"
```

Functions

- void shell_process ()
    The main function of shell.
- int shell_init (int argc, const char ∗∗argv)
    Initialize the shell.
- bool isBuildinCommand (struct parsed_command ∗cmd)
    Check if the command is a buildin command.
- bool isKnownProgram (struct parsed_command ∗cmd)
    Check if the command is a known program.

5.19.1 Detailed Description

Author

Shuo Sun (sunshuo@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.19.2 Function Documentation

5.19.2.1 isBuildinCommand()

```
bool isBuildinCommand (
          struct parsed_command * cmd )
```

Check if the command is a buildin command.

Parameters

cmd	

Returns

true

false

5.19.2.2 isKnownProgram()

```
bool isKnownProgram (
            struct parsed_command * cmd )
```

Check if the command is a known program.

Parameters

cmd	

Returns

true

false

5.19.2.3 shell_init()

```
int shell_init (
            int argc,
            const char ** argv )
```

Initialize the shell.

Parameters

argc	
argv	

Returns

int

5.19.2.4 shell_process()

```
void shell_process ( )
```

The main function of shell.

5.20 shell.h

Go to the documentation of this file.

```
#ifndef SHELL_H
#define SHELL_H

#include "utils.h"
#include "job.h"
#include "user.h"
#include "behavior.h"
#include "log.h"
#include "../PennFAT/filesys.h"
#include "../PennFAT/interface.h"

void shell_process();
int shell_init(int argc, const char **argv);
bool isBuildinCommand(struct parsed_command *cmd);
bool isKnownProgram(struct parsed_command *cmd);


#endif
```

5.21 stress.h

```
#ifndef STRESS_H
#define STRESS_H

void hang(void);
void nohang(void);
void recur(void);

#endif
```

5.22 src/kernel/user.h File Reference

user level functions

```
#include "kernel.h"
#include "utils.h"
#include "global.h"
#include "job.h"
#include "log.h"
```

Functions

- void writePrompt ()

    Write the prompt to the screen.
- void deconstruct_idle ()

    deconstruct the idle process
- bool **W_WIFEXITED** (int status)
- bool **W_WIFSTOPPED** (int status)
- bool **W_WIFSIGNALED** (int status)
- void **signal_handler** (int signal)
- int **register_signals** ()
- pid_t p_spawn (void(∗func)(), char ∗argv[ ], int fd0, int fd1)

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

    sets the calling thread as blocked (if nohang is false) until a child of the calling thread changes state
- pcb_node ∗ get_node_by_pid_all_alive_queues (pid_t pid)

    Get the node by pid all alive queues object.
- int p_kill (pid_t pid, int sig)

    sends the signal sig to the thread referenced by pid.
- int p_exit (void)

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

    sets the priority of the thread pid to priority
- void p_sleep (unsigned int ticks)

    sleeps the current thread for ticks of the system clock
- void deconstruct_shell ()

    deconstructs the shell

Variables

- priority_queue ∗ **ready_queue**
- pcb ∗ **active_process**
- ucontext_t **scheduler_context**
- int **tick_tracker**
- ucontext_t ∗ **p_active_context**
- ucontext_t **main_context**
- JobList **_jobList**
- pid_t **fgPid**

5.22.1 Detailed Description

user level functions

5.22.2 Function Documentation

5.22.2.1 deconstruct_idle()

```
void deconstruct_idle ( )
```

deconstruct the idle process

5.22.2.2 deconstruct_shell()

```
void deconstruct_shell ( )
```

deconstructs the shell

5.22.2.3 get_node_by_pid_all_alive_queues()

```
pcb_node * get_node_by_pid_all_alive_queues (
             pid_t pid )
```

Get the node by pid all alive queues object.

Parameters

pid	

Returns

pcb_node∗

5.22.2.4 p_exit()

```
int p_exit (
             void   )
```

exits the current thread unconditionally

Returns

int

5.22.2.5 p_kill()

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

sends the signal sig to the thread referenced by pid.

Parameters

pid	pid of the thread to send the signal to
sig	signal to be sent

Returns

returns 0 on success, or -1 on error.

5.22.2.6 p_nice()

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

sets the priority of the thread pid to priority

Parameters

pid	
priority	

Returns

int

5.22.2.7 p_sleep()

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

sleeps the current thread for ticks of the system clock

Parameters

ticks	

5.22.2.8 p_spawn()

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

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

Parameters

func	the function to be executed by the new thread
argv	the arguments to be passed to the function
fd0	the file descriptor to be used as stdin for the new thread
fd1	the file descriptor to be used as stdout for the new thread

Returns

the pid of the new thread on success, or -1 on error

5.22.2.9 p_waitpid()

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

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

Parameters

pid	pid of the child thread to wait for
wstatus	status of the child thread
nohang	whether to block the calling thread or not

Returns

pid_t

5.22.2.10 writePrompt()

```
void writePrompt ( )
```

Write the prompt to the screen.

5.23 user.h

Go to the documentation of this file.

```
#include "kernel.h"
#include "utils.h"
#include "global.h"
#include "job.h"
#include "log.h"

extern priority_queue* ready_queue;
extern pcb* active_process;
extern ucontext_t scheduler_context;
extern int tick_tracker;
extern ucontext_t* p_active_context;
extern ucontext_t main_context;

extern JobList _jobList;
extern pid_t fgPid;

extern void writePrompt();
extern void deconstruct_idle();

bool W_WIFEXITED(int status);
bool W_WIFSTOPPED(int status);
bool W_WIFSIGNALED(int status);

void signal_handler(int signal);

int register_signals();

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

// helper functions for waitpid
pid_t wait_for_one(pid_t pid, int *wstatus);
pid_t wait_for_anyone(int *wstatus);

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

pcb_node* get_node_by_pid_all_alive_queues(pid_t pid);

int p_kill(pid_t pid, int sig);

int p_exit(void);

int p_nice(pid_t pid, int priority);

void p_sleep(unsigned int ticks);

void deconstruct_shell();
```

5.24 src/PennFAT/utils.c File Reference

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

5.24.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.25 utils.h

```
#ifndef UTILS_H
#define UTILS_H

#include <signal.h>    // sigaction, sigemptyset, sigfillset, signal
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/time.h>  // setitimer
#include <ucontext.h>  // getcontext, makecontext, setcontext, swapcontext
#include <time.h>
#include <valgrind/valgrind.h>
#include "global2.h"
#include "perrno.h"
#include "../PennFAT/fd-table.h"

typedef struct pcb {
    ucontext_t ucontext;
    pid_t pid;
    pid_t ppid;
    enum process_state prev_state;
    enum process_state state;
    int priority;
    FdNode *fds[MAX_FILE_DESCRIPTOR];
    int ticks_left;
    struct pcb_queue* children;
    struct pcb_queue* zombies;
    char* pname;
    bool toWait;
} pcb;

typedef struct pcb_node {
    pcb* pcb;
    struct pcb_node* next;
} pcb_node;

typedef struct pcb_queue {
    pcb_node* head;
    pcb_node* tail;
} pcb_queue;

typedef struct priority_queue {
    pcb_queue* high;
    pcb_queue* mid;
    pcb_queue* low;
} priority_queue;

pcb* new_pcb(ucontext_t* ucontext, pid_t pid);

pcb_node* new_pcb_node(pcb* pcb);

pcb_queue* new_pcb_queue();

priority_queue* new_priority_queue();

bool is_empty(pcb_queue* queue);

bool is_priority_queue_empty(priority_queue* ready_queue);

pcb_queue* get_pcb_queue_by_priority(priority_queue* ready_queue, int priority);

void enqueue(pcb_queue* queue, pcb_node* node);

void enqueue_by_priority(priority_queue* ready_queue, int priority, pcb_node* node);

pcb_node* dequeue_by_pid(pcb_queue* queue, pid_t pid);

pcb_node *dequeue_front(pcb_queue* queue);

pcb_node *dequeue_front_by_priority(priority_queue* ready_queue, int priority);

pcb_node* get_node_by_pid(pcb_queue* queue, pid_t pid);

pcb_node* get_node_by_pid_from_priority_queue(priority_queue* ready_queue, pid_t pid);

void deconstruct_queue(pcb_queue* queue);

void deconstruct_priority_queue(priority_queue* ready_queue);

pcb_node* get_node_from_ready_queue(priority_queue* ready_queue, pid_t pid);

int pick_priority();

void set_stack(stack_t *stack);

```

```
int makeContext(ucontext_t *ucp,  void (*func)(), int argc, ucontext_t *next_context, char *argv[]);

int printQueue(pcb_queue *queue);

typedef enum {
    JOB_RUNNING,
    JOB_STOPPED,
    JOB_FINISHED,
    JOB_TERMINATED
} JobState;


typedef enum {
    NICE, // Syntax: nice priority command [args]
    NICE_PID, // Syntax: nice_pid priority pid
    MAN, // Syntax: man
    BG, // Syntax: bg [job_id]
    FG, // Syntax: fg [job_id]
    JOBS, // Syntax: jobs
    LOGOUT, // Syntax: logout
    KILL,
    OTHERS // non-builtin command
} CommandType;

typedef struct Job {
    struct parsed_command *cmd;
    pid_t pid;
    JobState state;
} Job;

typedef struct JobListNode {
    Job *job;
    struct JobListNode *prev;
    struct JobListNode *next;
    int jobId;
} JobListNode;

typedef struct JobList {
    JobListNode *head;
    JobListNode *tail;
    int jobCount;
} JobList;

pcb_queue* sortQueue(pcb_queue* queue);

pcb_queue* merge_two_queues(pcb_queue* queue1, pcb_queue* queue2);

extern priority_queue* ready_queue;
extern pcb_queue* exited_queue;
extern pcb_queue* stopped_queue;

extern pcb_node* get_node_by_pid_all_queues(pid_t pid);

#endif
```

5.26 src/PennFAT/utils.h File Reference

```
#include <stdlib.h>
#include <inttypes.h>
#include <time.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdio.h>
```

Macros

- #define **FS_SUCCESS** 0
- #define **FS_FAILURE** -1
- #define **FS_NOT_FOUND** -1

5.26.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.27 utils.h

Go to the documentation of this file.

```
#ifndef FS_UTILS_H
#define FS_UTILS_H

#include <stdlib.h>
#include <inttypes.h>
#include <time.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>

#include <stdio.h>

// #define FS_DEBUG_INFO 1

#define FS_SUCCESS 0
#define FS_FAILURE -1
#define FS_NOT_FOUND -1

#endif

```

5.28 src/PennFAT/FAT.c File Reference

```
#include "FAT.h"
```

Functions

- FATConfig ∗ createFATConfig (const char ∗name, uint16_t LSB, uint16_t MSB)

  Create a FATConfig object.
- uint16_t ∗ createFAT16InMemory (FATConfig ∗config)

  Create a FAT16 object in memory.
- int createFATOnDisk (FATConfig ∗config)

  Create a FAT16 object on disk.
- int findEmptyFAT16Entry (FATConfig ∗config, uint16_t ∗FAT16)

  Return the index of next empty FAT entry.
- DirectoryEntry ∗ createDirectoryEntry (const char ∗name, uint32_t size, uint16_t firstBlock, uint8_t type, uint8_t perm)

  Create a Directory Entry object.
- int createFileDirectoryOnDisk (FATConfig ∗config, uint16_t ∗FAT16, const char ∗fileName, uint8_t fileType, uint8_t filePerm)

  Create a File Directory On Disk object.
- int findFileDirectory (FATConfig ∗config, uint16_t ∗FAT16, const char ∗fileName)

  Return the offset of the file directory entry in data region.
- int readDirectoryEntry (FATConfig ∗config, int offset, DirectoryEntry ∗dir)

  Read the directory entry from FAT and set it to dir.
- int writeFileDirectory (FATConfig ∗config, int offset, DirectoryEntry ∗dir)

  Write the directory entry to the offset.
- int deleteFileDirectory (FATConfig ∗config, uint16_t ∗FAT16, int offset)

  Delete the file directory entry.
- int deleteFileDirectoryByName (FATConfig ∗config, uint16_t ∗FAT16, const char ∗fileName)

  Delete the file directory entry by file name.
- bool isDirectoryEntryToDelete (FATConfig ∗config, int directoryEntryOffset)

  Judge whether the directory entry is to be deleted.
- int readFAT (FATConfig ∗config, uint16_t ∗FAT16, int startBlock, int startBlockOffset, int size, char ∗buffer)

  Read the file data from FAT and set it to buffer.
- int writeFAT (FATConfig ∗config, uint16_t ∗FAT16, int startBlock, int startBlockOffset, int size, const char ∗buffer)

  Write the buffer to FAT.
- int traceFileEnd (FATConfig ∗config, uint16_t ∗FAT16, const char ∗fileName)

  Return the offset of the file end.
- int traceBytesFromBeginning (FATConfig ∗config, uint16_t ∗FAT16, int directoryEntryOffset, int fileOffset)

  Return the byte number from the file beginning to the file offset.
- int traceBytesToEnd (FATConfig ∗config, uint16_t ∗FAT16, int directoryEntryOffset, int fileOffset)

  Return the byte number from the file offset to the file end.
- int traceOffset (FATConfig ∗config, uint16_t ∗FAT16, int directoryEntryOffset, int fileOffset, int n)

  Return the offset of the n bytes after the given file offset.

5.28.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.28.2 Function Documentation

5.28.2.1 createDirectoryEntry()

```
DirectoryEntry * createDirectoryEntry (
            const char * name,
            uint32_t size,
            uint16_t firstBlock,
            uint8_t type,
            uint8_t perm )
```

Create a Directory Entry object.

Parameters

name	
size	
firstBlock	
type	
perm	

Returns

DirectoryEntry*

5.28.2.2 createFAT16InMemory()

```
uint16_t * createFAT16InMemory (
            FATConfig * config )
```

Create a FAT16 object in memory.

Parameters

config	

Returns

uint16_t∗

5.28.2.3 createFATConfig()

```
FATConfig ∗ createFATConfig (
             const char ∗ name,
             uint16_t LSB,
             uint16_t MSB )
```

Create a FATConfig object.

Parameters

name	
LSB	
MSB	

Returns

FATConfig∗

5.28.2.4 createFATOnDisk()

```
int createFATOnDisk (
             FATConfig ∗ config )
```

Create a FAT16 object on disk.

Parameters

config	

Returns

int

5.28.2.5 createFileDirectoryOnDisk()

```
int createFileDirectoryOnDisk (
             FATConfig ∗ config,
             uint16_t * FAT16,
             const char * fileName,
             uint8_t fileType,
             uint8_t filePerm )
```

Create a File Directory On Disk object.

Parameters

config	
FAT16	
fileName	
fileType	
filePerm	

Returns

int

5.28.2.6 deleteFileDirectory()

```
int deleteFileDirectory (
             FATConfig * config,
             uint16_t * FAT16,
             int directoryEntryOffset )
```

Delete the file directory entry.

Parameters

config	
FAT16	
directoryEntryOffset	

Returns

int

5.28.2.7 deleteFileDirectoryByName()

```
int deleteFileDirectoryByName (
             FATConfig * config,
             uint16_t * FAT16,
             const char * fileName )
```

Delete the file directory entry by file name.

Parameters

config	
FAT16	
fileName	

Returns

int

5.28.2.8 findEmptyFAT16Entry()

```
int findEmptyFAT16Entry (
            FATConfig * config,
            uint16_t * FAT16 )
```

Return the index of next empty FAT entry.

Parameters

config	
FAT16	

Returns

int

5.28.2.9 findFileDirectory()

```
int findFileDirectory (
            FATConfig * config,
            uint16_t * FAT16,
            const char * fileName )
```

Return the offset of the file directory entry in data region.

Parameters

config	
FAT16	
fileName	

Returns

int

5.28.2.10 isDirectoryEntryToDelete()

```
bool isDirectoryEntryToDelete (
            FATConfig * config,
            int offset )
```

Judge whether the directory entry is to be deleted.

Parameters

config	
offset	

Returns

true

false

5.28.2.11 readDirectoryEntry()

```
int readDirectoryEntry (
            FATConfig * config,
            int offset,
            DirectoryEntry * dir )
```

Read the directory entry from FAT and set it to dir.

Parameters

config	
offset	
dir	

Returns

int

5.28.2.12 readFAT()

```
int readFAT (
            FATConfig * config,
            uint16_t * FAT16,
            int startBlock,
            int startBlockOffset,
            int size,
            char * buffer )
```

Read the file data from FAT and set it to buffer.

Parameters

config	
FAT16	
startBlock	
startBlockOffset	
size	
buffer	

Returns

int

5.28.2.13 traceBytesFromBeginning()

```
int traceBytesFromBeginning (
            FATConfig * config,
            uint16_t * FAT16,
            int directoryEntryOffset,
            int fileOffset )
```

Return the byte number from the file beginning to the file offset.

Parameters

config	
FAT16	
directoryEntryOffset	
fileOffset	

Returns

int

5.28.2.14 traceBytesToEnd()

```
int traceBytesToEnd (
            FATConfig * config,
            uint16_t * FAT16,
            int directoryEntryOffset,
            int fileOffset )
```

Return the byte number from the file offset to the file end.

Parameters

config	
FAT16	
directoryEntryOffset	
fileOffset	

Returns

int

5.28.2.15 traceFileEnd()

```
int traceFileEnd (
            FATConfig * config,
            uint16_t * FAT16,
            const char * fileName )
```

Return the offset of the file end.

Parameters

config	
FAT16	
fileName	

Returns

int

5.28.2.16 traceOffset()

```
int traceOffset (
            FATConfig * config,
            uint16_t * FAT16,
            int directoryEntryOffset,
            int fileOffset,
            int n )
```

Return the offset of the n bytes after the given file offset.

Parameters

config	
FAT16	
directoryEntryOffset	
fileOffset	
n	

Returns

int

5.28.2.17 writeFAT()

```
int writeFAT (
            FATConfig * config,
            uint16_t * FAT16,
            int startBlock,
            int startBlockOffset,
            int size,
            const char * buffer )
```

Write the buffer to FAT.

Parameters

config	
FAT16	
startBlock	
startBlockOffset	
size	
buffer	

Returns

int

5.28.2.18 writeFileDirectory()

```
int writeFileDirectory (
            FATConfig * config,
            int offset,
            DirectoryEntry * dir )
```

Write the directory entry to the offset.

Parameters

config	
offset	
dir	

Returns

int

5.29 src/PennFAT/FAT.h File Reference

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

Classes

- struct FATConfig

  The LSB(rightmost under little endian) of the first entry of the FAT specifies the block size with the mapping as below: {LSB : size in bytes} = {0:256; 1:512; 2:1024; 3:2048; 4:4096} The MSB(leftmost under little endian) of the first entry of the FAT specifies the number of blocks that FAT region occupies. The MSB should be ranged from 1-32 (numbers outside of this range will be considered an error). FAT region size = block size ∗ FAT region block number FAT entry number = FAT region size / FAT entry size (2-byte in FAT16) Data region size = block size ∗ (FAT entry number - 1)
- struct DirectoryEntry

  The structure of the directory entry as stored in the filesystem is as follows:

Macros

- #define **MIN_BLOCK_SIZE** 256
- #define **MAX_FAT_BLOCK_NUM** 32
- #define **MAX_BLOCK_SCALE** 4
- #define **FAT_ENTRY_SIZE** 2
- #define **EMPTY_FAT_ENTRY** 0x0000
- #define **NO_SUCC_FAT_ENTRY** 0xFFFF
- #define **MAX_FILE_NAME_LENGTH** 32
- #define **DIRECTORY_ENTRY_SIZE** 64
- #define **NO_FIRST_BLOCK** 0x0000
- #define **DIRECTORY_END** "0"
- #define **DELETED_DIRECTORY** "1"
- #define **DELETED_DIRECTORY_IN_USE** "2"
- #define **FILE_TYPE_UNKNOWN** 0
- #define **FILE_TYPE_REGULAR** 1
- #define **FILE_TYPE_DIRECTORY** 2
- #define **FILE_TYPE_SYMBOLIC_LINK** 4
- #define **FILE_PERM_NONE** 0
- #define **FILE_PERM_WRITE** 2
- #define **FILE_PERM_READ** 4
- #define **FILE_PERM_READ_EXEC** 5
- #define **FILE_PERM_READ_WRITE** 6
- #define **FILE_PERM_READ_WRITE_EXEC** 7
- #define **RESERVED_BYTES** 16

Typedefs

- typedef struct FATConfig **FATConfig**

    The LSB(rightmost under little endian) of the first entry of the FAT specifies the block size with the mapping as below: {LSB : size in bytes} = {0:256; 1:512; 2:1024; 3:2048; 4:4096} The MSB(leftmost under little endian) of the first entry of the FAT specifies the number of blocks that FAT region occupies. The MSB should be ranged from 1-32 (numbers outside of this range will be considered an error). FAT region size = block size ∗ FAT region block number FAT entry number = FAT region size / FAT entry size (2-byte in FAT16) Data region size = block size ∗ (FAT entry number - 1)
- typedef struct DirectoryEntry DirectoryEntry

    The structure of the directory entry as stored in the filesystem is as follows:

Functions

- FATConfig ∗ createFATConfig (const char ∗name, uint16_t LSB, uint16_t MSB)

    Create a FATConfig object.
- uint16_t ∗ createFAT16InMemory (FATConfig ∗config)

    Create a FAT16 object in memory.
- int createFATOnDisk (FATConfig ∗config)

    Create a FAT16 object on disk.
- int findEmptyFAT16Entry (FATConfig ∗config, uint16_t ∗FAT16)

    Return the index of next empty FAT entry.
- DirectoryEntry ∗ createDirectoryEntry (const char ∗name, uint32_t size, uint16_t firstBlock, uint8_t type, uint8_t perm)

    Create a Directory Entry object.
- int createFileDirectoryOnDisk (FATConfig ∗config, uint16_t ∗FAT16, const char ∗fileName, uint8_t fileType, uint8_t filePerm)

    Create a File Directory On Disk object.
- int findFileDirectory (FATConfig ∗config, uint16_t ∗FAT16, const char ∗fileName)

    Return the offset of the file directory entry in data region.
- int readDirectoryEntry (FATConfig ∗config, int offset, DirectoryEntry ∗dir)

    Read the directory entry from FAT and set it to dir.
- int writeFileDirectory (FATConfig ∗config, int offset, DirectoryEntry ∗dir)

    Write the directory entry to the offset.
- int deleteFileDirectory (FATConfig ∗config, uint16_t ∗FAT16, int directoryEntryOffset)

    Delete the file directory entry.
- int deleteFileDirectoryByName (FATConfig ∗config, uint16_t ∗FAT16, const char ∗fileName)

    Delete the file directory entry by file name.
- bool isDirectoryEntryToDelete (FATConfig ∗config, int offset)

    Judge whether the directory entry is to be deleted.
- int readFAT (FATConfig ∗config, uint16_t ∗FAT16, int startBlock, int startBlockOffset, int size, char ∗buffer)

    Read the file data from FAT and set it to buffer.
- int writeFAT (FATConfig ∗config, uint16_t ∗FAT16, int startBlock, int startBlockOffset, int size, const char ∗buffer)

    Write the buffer to FAT.
- int traceFileEnd (FATConfig ∗config, uint16_t ∗FAT16, const char ∗fileName)

    Return the offset of the file end.
- int traceBytesFromBeginning (FATConfig ∗config, uint16_t ∗FAT16, int directoryEntryOffset, int fileOffset)

    Return the byte number from the file beginning to the file offset.
- int traceBytesToEnd (FATConfig ∗config, uint16_t ∗FAT16, int directoryEntryOffset, int fileOffset)

    Return the byte number from the file offset to the file end.
- int traceOffset (FATConfig ∗config, uint16_t ∗FAT16, int directoryEntryOffset, int fileOffset, int n)

    Return the offset of the n bytes after the given file offset.

5.29.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.29.2 Typedef Documentation

5.29.2.1 DirectoryEntry

```
typedef struct DirectoryEntry DirectoryEntry
```

The structure of the directory entry as stored in the filesystem is as follows:

- char name[32]: null-terminated file name name[0] also serves as a special marker: – 0: end of directory – 1: deleted entry; the file is also deleted – 2: deleted entry; the file is still being used
- uint32_t size: number of bytes in file
- uint16_t firstBlock: the first block number of the file (undefined if size is zero) The block index of data region starts from 1. If the firstBlock is 0, it means that this is an empty file which has not occupied any data region block.
- uint8_t type: the type of the file, which will be one of the following: – 0: unknown – 1: a regular file – 2: a directory file – 4: a symbolic link
- uint8_t perm: file permissions, which will be one of the following: – 0: none – 2: write only – 4: read only – 5: read and executable (shell scripts) – 6: read and write – 7: read, write, and executable
- time_t mtime: creation/modification time as returned by time(2) in Linux

5.29.3 Function Documentation

5.29.3.1 createDirectoryEntry()

```
DirectoryEntry * createDirectoryEntry (
            const char * name,
            uint32_t size,
            uint16_t firstBlock,
            uint8_t type,
            uint8_t perm )
```

Create a Directory Entry object.

Parameters

name	
size	
firstBlock	
type	
perm	

Returns

DirectoryEntry∗

5.29.3.2 createFAT16InMemory()

```
uint16_t * createFAT16InMemory (
             FATConfig * config )
```

Create a FAT16 object in memory.

Parameters

config	

Returns

uint16_t∗

5.29.3.3 createFATConfig()

```
FATConfig * createFATConfig (
             const char * name,
             uint16_t LSB,
             uint16_t MSB )
```

Create a FATConfig object.

Parameters

name	
LSB	
MSB	

Returns

FATConfig∗

5.29.3.4 createFATOnDisk()

```
int createFATOnDisk (
             FATConfig ∗ config )
```

Create a FAT16 object on disk.

Parameters

config	

Returns

int

5.29.3.5 createFileDirectoryOnDisk()

```
int createFileDirectoryOnDisk (
             FATConfig ∗ config,
             uint16_t ∗ FAT16,
             const char ∗ fileName,
             uint8_t fileType,
             uint8_t filePerm )
```

Create a File Directory On Disk object.

Parameters

config	
FAT16	
fileName	
fileType	
filePerm	

Returns

int

5.29.3.6 deleteFileDirectory()

```
int deleteFileDirectory (
            FATConfig * config,
            uint16_t * FAT16,
            int directoryEntryOffset )
```

Delete the file directory entry.

Parameters

config	
FAT16	
directoryEntryOffset	

Returns

int

5.29.3.7 deleteFileDirectoryByName()

```
int deleteFileDirectoryByName (
            FATConfig * config,
            uint16_t * FAT16,
            const char * fileName )
```

Delete the file directory entry by file name.

Parameters

config	
FAT16	
fileName	

Returns

int

5.29.3.8 findEmptyFAT16Entry()

```
int findEmptyFAT16Entry (
            FATConfig * config,
            uint16_t * FAT16 )
```

Return the index of next empty FAT entry.

Parameters

config	
FAT16	

Returns

int

5.29.3.9 findFileDirectory()

```
int findFileDirectory (
            FATConfig * config,
            uint16_t * FAT16,
            const char * fileName )
```

Return the offset of the file directory entry in data region.

Parameters

config	
FAT16	
fileName	

Returns

int

5.29.3.10 isDirectoryEntryToDelete()

```
bool isDirectoryEntryToDelete (
            FATConfig * config,
            int offset )
```

Judge whether the directory entry is to be deleted.

Parameters

config	
offset	

Returns

true

false

5.29.3.11 readDirectoryEntry()

```
int readDirectoryEntry (
            FATConfig * config,
            int offset,
            DirectoryEntry * dir )
```

Read the directory entry from FAT and set it to dir.

Parameters

config	
offset	
dir	

Returns

int

5.29.3.12 readFAT()

```
int readFAT (
            FATConfig * config,
            uint16_t * FAT16,
            int startBlock,
            int startBlockOffset,
            int size,
            char * buffer )
```

Read the file data from FAT and set it to buffer.

Parameters

config	
FAT16	
startBlock	
startBlockOffset	
size	
buffer	

Returns

int

5.29.3.13 traceBytesFromBeginning()

```
int traceBytesFromBeginning (
            FATConfig * config,
            uint16_t * FAT16,
            int directoryEntryOffset,
            int fileOffset )
```

Return the byte number from the file beginning to the file offset.

Parameters

config	
FAT16	
directoryEntryOffset	
fileOffset	

Returns

int

5.29.3.14 traceBytesToEnd()

```
int traceBytesToEnd (
            FATConfig * config,
            uint16_t * FAT16,
            int directoryEntryOffset,
            int fileOffset )
```

Return the byte number from the file offset to the file end.

Parameters

config	
FAT16	
directoryEntryOffset	
fileOffset	

Returns

int

5.29.3.15 traceFileEnd()

```
int traceFileEnd (
            FATConfig * config,
            uint16_t * FAT16,
            const char * fileName )
```

Return the offset of the file end.

Parameters

config	
FAT16	
fileName	

Returns

int

5.29.3.16 traceOffset()

```
int traceOffset (
            FATConfig * config,
            uint16_t * FAT16,
            int directoryEntryOffset,
            int fileOffset,
            int n )
```

Return the offset of the n bytes after the given file offset.

Parameters

config	
FAT16	
directoryEntryOffset	
fileOffset	
n	

Returns

int

5.29.3.17 writeFAT()

```
int writeFAT (
            FATConfig * config,
            uint16_t * FAT16,
            int startBlock,
            int startBlockOffset,
            int size,
            const char * buffer )
```

Write the buffer to FAT.

Parameters

config	
FAT16	
startBlock	
startBlockOffset	
size	
buffer	

Returns

int

5.29.3.18 writeFileDirectory()

```
int writeFileDirectory (
            FATConfig * config,
            int offset,
            DirectoryEntry * dir )
```

Write the directory entry to the offset.

Parameters

config	
offset	
dir	

Returns

int

5.30 FAT.h

Go to the documentation of this file.

```
#ifndef FAT_H
#define FAT_H

#include "utils.h"

/* PennFAT is based on FAT16 */
```

```
#define MIN_BLOCK_SIZE 256
#define MAX_FAT_BLOCK_NUM 32
#define MAX_BLOCK_SCALE 4

#define FAT_ENTRY_SIZE 2
#define EMPTY_FAT_ENTRY 0x0000
#define NO_SUCC_FAT_ENTRY 0xFFFF

#define MAX_FILE_NAME_LENGTH 32

typedef struct FATConfig {
    char name[MAX_FILE_NAME_LENGTH];

    uint16_t LSB;
    uint16_t MSB;

    int blockSize;
    int FATRegionBlockNum;
    int FATRegionSize;
    int FATEntryNum;
    int dataRegionSize;

    /*
    The FAT region will be mapped to the memory through mmap().
    The size of the mapping area must be a multiple of the memory page size.
    */
    int FATSizeInMemory;
} FATConfig;

/* PennFAT has a 64-byte fixed directory entry size (32 + 4 + 2 + 1 + 1 + 8 + 16 = 64 bytes)*/
#define DIRECTORY_ENTRY_SIZE 64

#define NO_FIRST_BLOCK 0x0000

#define DIRECTORY_END "0"
#define DELETED_DIRECTORY "1"
/*
Process A and Process B opened the same file.
Process A unlinked the file but Process B was still using the file.
The file should be deleted after Process B closes the file.
*/
#define DELETED_DIRECTORY_IN_USE "2"

#define FILE_TYPE_UNKNOWN 0
#define FILE_TYPE_REGULAR 1
#define FILE_TYPE_DIRECTORY 2
#define FILE_TYPE_SYMBOLIC_LINK 4

#define FILE_PERM_NONE 0  // 000
#define FILE_PERM_WRITE 2 // 010
#define FILE_PERM_READ 4  // 100
#define FILE_PERM_READ_EXEC 5 // 101
#define FILE_PERM_READ_WRITE 6 // 110
#define FILE_PERM_READ_WRITE_EXEC 7 // 111

#define RESERVED_BYTES 16

typedef struct DirectoryEntry {
    char name[MAX_FILE_NAME_LENGTH]; // 32-byte
    uint32_t size; // 4-byte
    uint16_t firstBlock; // 2-byte
    uint8_t type; // 1-byte
    uint8_t perm; // 1-byte
    time_t mtime; // 8-byte
    char reserved[RESERVED_BYTES];   // 16-byte reserved space for extension
} DirectoryEntry;
FATConfig *createFATConfig(const char *name, uint16_t LSB, uint16_t MSB);
uint16_t *createFAT16InMemory(FATConfig *config);
int createFATOnDisk(FATConfig *config);

/* ##### TODO: Thread Safety ##### */

int findEmptyFAT16Entry(FATConfig *config, uint16_t *FAT16);
DirectoryEntry *createDirectoryEntry(const char *name, uint32_t size, uint16_t firstBlock, uint8_t
      type, uint8_t perm);
int createFileDirectoryOnDisk(FATConfig *config, uint16_t *FAT16, const char *fileName, uint8_t
      fileType, uint8_t filePerm);
int findFileDirectory(FATConfig *config, uint16_t *FAT16, const char *fileName);
int readDirectoryEntry(FATConfig *config, int offset, DirectoryEntry *dir);
int writeFileDirectory(FATConfig *config, int offset, DirectoryEntry *dir);
int deleteFileDirectory(FATConfig *config, uint16_t *FAT16, int directoryEntryOffset);
int deleteFileDirectoryByName(FATConfig *config, uint16_t *FAT16, const char *fileName);
bool isDirectoryEntryToDelete(FATConfig *config, int offset);
int readFAT(FATConfig *config, uint16_t *FAT16, int startBlock, int startBlockOffset, int size, char
      *buffer);
int writeFAT(FATConfig *config, uint16_t *FAT16, int startBlock, int startBlockOffset, int size, const
      char *buffer);
```

```
int traceFileEnd(FATConfig *config, uint16_t *FAT16, const char *fileName);
int traceBytesFromBeginning(FATConfig *config, uint16_t *FAT16, int directoryEntryOffset, int
     fileOffset);
int traceBytesToEnd(FATConfig *config, uint16_t *FAT16, int directoryEntryOffset, int fileOffset);
int traceOffset(FATConfig *config, uint16_t *FAT16, int directoryEntryOffset, int fileOffset, int n);

#endif
```

5.31 src/PennFAT/fd-table.c File Reference

```
#include "fd-table.h"
```

Functions

- FdNode ∗ createFdNode (int openMode, int directoryEntryOffset, int fileOffset)
  Create a Fd Node object.
- int initFdTable (FdTable ∗fdTable)
  Initialize the file descriptor table.
- int clearFdTable (FdTable ∗fdTable)
  Clear the file descriptor table.
- int appendFdTable (FdTable ∗fdTable, FdNode ∗newNode)
  Append a new FdNode to the file descriptor table.
- int removeFdNode (FdNode ∗fdNode)
  Remove a FdNode from the file descriptor table.
- bool isFileBeingUsed (FdTable ∗fdTable, int directoryEntryOffset)
  Check if the file is being used by any file descriptor.
- bool isFileBeingWritten (FdTable ∗fdTable, int directoryEntryOffset)
  Check if the file is being written by any file descriptor.
- int findAvailableFd (FdNode ∗∗fds)
  Find the first available file descriptor.

5.31.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.31.2 Function Documentation

5.31.2.1 appendFdTable()

```
int appendFdTable (
            FdTable * fdTable,
            FdNode * newNode )
```

Append a new FdNode to the file descriptor table.

Parameters

fdTable	
newNode	

Returns

int

5.31.2.2 clearFdTable()

```
int clearFdTable (
            FdTable * fdTable )
```

Clear the file descriptor table.

Parameters

fdTable	

Returns

int

5.31.2.3 createFdNode()

```
FdNode * createFdNode (
            int openMode,
            int directoryEntryOffset,
            int fileOffset )
```

Create a Fd Node object.

Parameters

openMode	
directoryEntryOffset	
fileOffset	

Returns

FdNode∗

5.31.2.4 findAvailableFd()

```
int findAvailableFd (
            FdNode ** fds )
```

Find the first available file descriptor.

Parameters

fds	

Returns

int

5.31.2.5 initFdTable()

```
int initFdTable (
            FdTable * fdTable )
```

Initialize the file descriptor table.

Parameters

fdTable	

Returns

int

5.31.2.6 isFileBeingUsed()

```
bool isFileBeingUsed (
            FdTable * fdTable,
            int directoryEntryOffset )
```

Check if the file is being used by any file descriptor.

Parameters

fdTable	
directoryEntryOffset	

Returns

true

false

5.31.2.7 isFileBeingWritten()

```
bool isFileBeingWritten (
            FdTable * fdTable,
            int directoryEntryOffset )
```

Check if the file is being written by any file descriptor.

Parameters

fdTable	
directoryEntryOffset	

Returns

true

false

5.31.2.8 removeFdNode()

```
int removeFdNode (
            FdNode * fdNode )
```

Remove a FdNode from the file descriptor table.

Parameters

fdNode	

Returns

int

5.32 src/PennFAT/fd-table.h File Reference

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

Classes

- struct FdNode

  There are three open mode supported by PennFAT: F_WRITE, F_READ and F_APPEND. According to ed #953, each file can only be read/write exclusivly which means only one instance can open() a file at a time. Under F_APPEND mode, the fileOffset will be set to the end of the file initially and it can only be increased. Under F_WRITE/F_APPEND mode, if the fileOffset is set to the position beyond the file size, the file system will occupy the space for the gap. As a result, the size of the file will increase, however, the gap space may contain uninitialized contents. Under F_READ mode, if the fileOffset is set to the position beyond the file size, f_read() will read nothing.
- struct FdTable

  The file descriptor table is a linked list of FdNode.

Macros

- #define **MAX_FILE_DESCRIPTOR** 8
- #define **F_STDIN_FD** 0
- #define **F_STDOUT_FD** 1
- #define **F_ERROR** 2
- #define **F_MIN_FD** 3
- #define **F_WRITE** 0
- #define **F_READ** 1
- #define **F_APPEND** 2

Typedefs

- typedef struct FdNode **FdNode**

  There are three open mode supported by PennFAT: F_WRITE, F_READ and F_APPEND. According to ed #953, each file can only be read/write exclusivly which means only one instance can open() a file at a time. Under F_APPEND mode, the fileOffset will be set to the end of the file initially and it can only be increased. Under F_WRITE/F_APPEND mode, if the fileOffset is set to the position beyond the file size, the file system will occupy the space for the gap. As a result, the size of the file will increase, however, the gap space may contain uninitialized contents. Under F_READ mode, if the fileOffset is set to the position beyond the file size, f_read() will read nothing.
- typedef struct FdTable FdTable

  The file descriptor table is a linked list of FdNode.

Functions

- FdNode ∗ createFdNode (int openMode, int directoryEntryOffset, int fileOffset)

  Create a Fd Node object.
- int initFdTable (FdTable ∗fdTable)

  Initialize the file descriptor table.
- int clearFdTable (FdTable ∗fdTable)

  Clear the file descriptor table.
- int appendFdTable (FdTable ∗fdTable, FdNode ∗newNode)

  Append a new FdNode to the file descriptor table.
- int removeFdNode (FdNode ∗fdNode)

  Remove a FdNode from the file descriptor table.
- bool isFileBeingUsed (FdTable ∗fdTable, int directoryEntryOffset)

  Check if the file is being used by any file descriptor.
- bool isFileBeingWritten (FdTable ∗fdTable, int directoryEntryOffset)

  Check if the file is being written by any file descriptor.
- int findAvailableFd (FdNode ∗∗fds)

  Find the first available file descriptor.

Variables

- FATConfig ∗ **fs_FATConfig**
- uint16_t ∗ **fs_FAT16InMemory**

5.32.1 Detailed Description

Author

Zhiyuan Liang (`liangzhy@seas.upenn.edu`)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.32.2 Typedef Documentation

5.32.2.1 FdTable

```
typedef struct FdTable FdTable
```

The file descriptor table is a linked list of FdNode.

5.32.3 Function Documentation

5.32.3.1 appendFdTable()

```
int appendFdTable (
            FdTable * fdTable,
            FdNode * newNode )
```

Append a new FdNode to the file descriptor table.

Parameters

fdTable	
newNode	

Returns

int

5.32.3.2 clearFdTable()

```
int clearFdTable (
            FdTable * fdTable )
```

Clear the file descriptor table.

Parameters

fdTable	

Returns

int

5.32.3.3 createFdNode()

```
FdNode * createFdNode (
            int openMode,
            int directoryEntryOffset,
            int fileOffset )
```

Create a Fd Node object.

Parameters

openMode	
directoryEntryOffset	
fileOffset	

Returns

FdNode∗

5.32.3.4 findAvailableFd()

```
int findAvailableFd (
            FdNode ** fds )
```

Find the first available file descriptor.

Parameters

fds	

Returns

int

5.32.3.5 initFdTable()

```
int initFdTable (
            FdTable * fdTable )
```

Initialize the file descriptor table.

Parameters

fdTable	

Returns

int

5.32.3.6 isFileBeingUsed()

```
bool isFileBeingUsed (
            FdTable * fdTable,
            int directoryEntryOffset )
```

Check if the file is being used by any file descriptor.

Parameters

fdTable	
directoryEntryOffset	

Returns

true

false

5.32.3.7 isFileBeingWritten()

```
bool isFileBeingWritten (
            FdTable * fdTable,
            int directoryEntryOffset )
```

Check if the file is being written by any file descriptor.

Parameters

fdTable	
directoryEntryOffset	

Returns

true

false

5.32.3.8 removeFdNode()

```
int removeFdNode (
            FdNode * fdNode )
```

Remove a FdNode from the file descriptor table.

Parameters

fdNode	

Returns

int

5.33 fd-table.h

Go to the documentation of this file.

```
#ifndef FD_TABLE_H
#define FD_TABLE_H

#include "utils.h"
#include "FAT.h"

extern FATConfig *fs_FATConfig;
extern uint16_t *fs_FAT16InMemory;

#define MAX_FILE_DESCRIPTOR 8

/* file descriptor 0,1,2 were reserved */
#define F_STDIN_FD 0
#define F_STDOUT_FD 1
#define F_ERROR 2
#define F_MIN_FD 3

#define F_WRITE 0
#define F_READ 1
#define F_APPEND 2

/* We use linked list to implement the file descriptor table. */


typedef struct FdNode {
    int openMode;
    int directoryEntryOffset; // directory entry location
    /* If a file is an empty file, the fileOffset will be set to 0. */
    int fileOffset;

    struct FdNode* prev;
    struct FdNode* next;
} FdNode;

typedef struct FdTable {
    FdNode *head;
    FdNode *tail;
} FdTable;

FdNode *createFdNode(int openMode, int directoryEntryOffset, int fileOffset);
int initFdTable(FdTable *fdTable);
int clearFdTable(FdTable *fdTable);
int appendFdTable(FdTable *fdTable, FdNode *newNode);
int removeFdNode(FdNode *fdNode);
bool isFileBeingUsed(FdTable *fdTable, int directoryEntryOffset);
bool isFileBeingWritten(FdTable *fdTable, int directoryEntryOffset);
int findAvailableFd(FdNode **fds);

#endif
```

5.34 src/PennFAT/filesys.c File Reference

```
#include "filesys.h"
```

Functions

- bool isFileSystemMounted ()

  Check if the file system is mounted.
- bool isValidFileName (const char ∗fileName)

  Check if the file name is valid.
- int fs_mkfs (const char ∗fsName, uint16_t blockSizeConfig, uint16_t FATRegionBlockNum)

  Create a file system. Return 0 if success.
- int fs_mount (const char ∗fsName)

  Mount the file system. Return 0 if success.
- int fs_unmount ()

  Unmount the file system. Return 0 if success.
- int fs_touch (const char ∗fileName)

  Creates the file if it does not exist, otherwise update its timestamp. Return the offset of the file directory entry.
- int fs_rm (const char ∗fileName)

  Remove the file from FAT. Return 0 if success.
- int fs_mv (const char ∗src, const char ∗dst)

  Rename src to dst. If dst exists, remove it from FAT. Return 0 if success.
- int fs_cp (const char ∗src, const char ∗dst)

  Duplicate src to dst. If dst exists, replace it. Return 0 if success.
- int fs_readFAT (int startBlock, int startBlockOffset, int size, char ∗buffer)

  Encapsulation of readFAT(). Return 0 if success.
- int fs_writeFAT (int startBlock, int startBlockOffset, int size, const char ∗buffer)

  Encapsulation of writeFAT(). Return 0 if success.
- int fs_chmod (char ∗fileName, uint8_t perm)

  Change the permission of the file. Return 0 if success.

Variables

- FATConfig ∗ **fs_FATConfig** = NULL
- uint16_t ∗ **fs_FAT16InMemory** = NULL
- FdTable **fs_fdTable**

5.34.1 Detailed Description

Author

Zhiyuan Liang (`liangzhy@seas.upenn.edu`)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.34.2 Function Documentation

5.34.2.1 fs_chmod()

```
int fs_chmod (
            char * fileName,
            uint8_t perm )
```

Change the permission of the file. Return 0 if success.

Parameters

fileName	
perm	

Returns

int

5.34.2.2 fs_cp()

```
int fs_cp (
            const char * src,
            const char * dst )
```

Duplicate src to dst. If dst exists, replace it. Return 0 if success.

Parameters

src	
dst	

Returns

int

5.34.2.3 fs_mkfs()

```
int fs_mkfs (
            const char * fsName,
            uint16_t blockSizeConfig,
            uint16_t FATRegionBlockNum )
```

Create a file system. Return 0 if success.

Parameters

fsName	
blockSizeConfig	
FATRegionBlockNum	

Returns

int

5.34.2.4 fs_mount()

```
int fs_mount (
            const char * fsName )
```

Mount the file system. Return 0 if success.

Parameters

fsName	

Returns

int

5.34.2.5 fs_mv()

```
int fs_mv (
            const char * src,
            const char * dst )
```

Rename src to dst. If dst exists, remove it from FAT. Return 0 if success.

Parameters

src	
dst	

Returns

int

5.34.2.6 fs_readFAT()

```
int fs_readFAT (
            int startBlock,
            int startBlockOffset,
            int size,
            char * buffer )
```

Encapsulation of readFAT(). Return 0 if success.

Parameters

startBlock	
startBlockOffset	
size	
buffer	

Returns

int

5.34.2.7 fs_rm()

```
int fs_rm (
            const char * fileName )
```

Remove the file from FAT. Return 0 if success.

Parameters

fileName	

Returns

int

5.34.2.8 fs_touch()

```
int fs_touch (
            const char * fileName )
```

Creates the file if it does not exist, otherwise update its timestamp. Return the offset of the file directory entry.

Parameters

fileName	

Returns

int

5.34.2.9 fs_unmount()

```
int fs_unmount ( )
```

Unmount the file system. Return 0 if success.

Returns

int

5.34.2.10 fs_writeFAT()

```
int fs_writeFAT (
            int startBlock,
            int startBlockOffset,
            int size,
            const char * buffer )
```

Encapsulation of writeFAT(). Return 0 if success.

Parameters

startBlock	
startBlockOffset	
size	
buffer	

Returns

int

5.34.2.11 isFileSystemMounted()

```
bool isFileSystemMounted ( )
```

Check if the file system is mounted.

Returns

true

false

5.34.2.12 isValidFileName()

```
bool isValidFileName (
            const char * fileName )
```

Check if the file name is valid.

Parameters

fileName	

Returns

true

false

5.35 src/PennFAT/filesys.h File Reference

```
#include "utils.h"
#include "FAT.h"
#include "fd-table.h"
```

Functions

- bool isFileSystemMounted ()

  Check if the file system is mounted.
- bool isValidFileName (const char ∗fileName)

  Check if the file name is valid.
- int fs_mkfs (const char ∗fsName, uint16_t blockSizeConfig, uint16_t FATRegionBlockNum)

  Create a file system. Return 0 if success.
- int fs_mount (const char ∗fsName)

  Mount the file system. Return 0 if success.
- int fs_unmount ()

  Unmount the file system. Return 0 if success.
- int fs_touch (const char ∗fileName)

  Creates the file if it does not exist, otherwise update its timestamp. Return the offset of the file directory entry.
- int fs_rm (const char ∗fileName)

  Remove the file from FAT. Return 0 if success.
- int fs_mv (const char ∗src, const char ∗dst)

  Rename src to dst. If dst exists, remove it from FAT. Return 0 if success.
- int fs_cp (const char ∗src, const char ∗dst)

  Duplicate src to dst. If dst exists, replace it. Return 0 if success.
- int fs_readFAT (int startBlock, int startBlockOffset, int size, char ∗buffer)

  Encapsulation of readFAT(). Return 0 if success.
- int fs_writeFAT (int startBlock, int startBlockOffset, int size, const char ∗buffer)

  Encapsulation of writeFAT(). Return 0 if success.
- int fs_chmod (char ∗fileName, uint8_t perm)

  Change the permission of the file. Return 0 if success.

5.35.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.35.2 Function Documentation

5.35.2.1 fs_chmod()

```
int fs_chmod (
            char * fileName,
            uint8_t perm )
```

Change the permission of the file. Return 0 if success.

Parameters

fileName	
perm	

Returns

int

5.35.2.2 fs_cp()

```
int fs_cp (
            const char * src,
            const char * dst )
```

Duplicate src to dst. If dst exists, replace it. Return 0 if success.

Parameters

src	
dst	

Returns

int

5.35.2.3 fs_mkfs()

```
int fs_mkfs (
             const char * fsName,
             uint16_t blockSizeConfig,
             uint16_t FATRegionBlockNum )
```

Create a file system. Return 0 if success.

Parameters

fsName	
blockSizeConfig	
FATRegionBlockNum	

Returns

int

5.35.2.4 fs_mount()

```
int fs_mount (
             const char * fsName )
```

Mount the file system. Return 0 if success.

Parameters

fsName	

Returns

int

5.35.2.5 fs_mv()

```
int fs_mv (
            const char * src,
            const char * dst )
```

Rename src to dst. If dst exists, remove it from FAT. Return 0 if success.

Parameters

src	
dst	

Returns

int

5.35.2.6 fs_readFAT()

```
int fs_readFAT (
            int startBlock,
            int startBlockOffset,
            int size,
            char * buffer )
```

Encapsulation of readFAT(). Return 0 if success.

Parameters

startBlock	
startBlockOffset	
size	
buffer	

Returns

int

5.35.2.7 fs_rm()

```
int fs_rm (
            const char * fileName )
```

Remove the file from FAT. Return 0 if success.

Parameters

fileName	

Returns

int

5.35.2.8 fs_touch()

```
int fs_touch (
            const char * fileName )
```

Creates the file if it does not exist, otherwise update its timestamp. Return the offset of the file directory entry.

Parameters

fileName	

Returns

int

5.35.2.9 fs_unmount()

```
int fs_unmount ( )
```

Unmount the file system. Return 0 if success.

Returns

int

5.35.2.10 fs_writeFAT()

```
int fs_writeFAT (
            int startBlock,
            int startBlockOffset,
            int size,
            const char * buffer )
```

Encapsulation of writeFAT(). Return 0 if success.

Parameters

startBlock	
startBlockOffset	
size	
buffer	

Returns

int

5.35.2.11 isFileSystemMounted()

```
bool isFileSystemMounted ( )
```

Check if the file system is mounted.

Returns

true

false

5.35.2.12 isValidFileName()

```
bool isValidFileName (
             const char * fileName )
```

Check if the file name is valid.

Parameters

fileName	

Returns

true

false

5.36 filesys.h

Go to the documentation of this file.

```
00001
00011 #ifndef FILESYS_H
```

```
#define FILESYS_H

#include "utils.h"
#include "FAT.h"
#include "fd-table.h"

bool isFileSystemMounted();
bool isValidFileName(const char *fileName);

int fs_mkfs(const char *fsName, uint16_t blockSizeConfig, uint16_t FATRegionBlockNum);
int fs_mount(const char *fsName);
int fs_unmount();
int fs_touch(const char *fileName);
int fs_rm(const char *fileName);
int fs_mv(const char *src, const char *dst);
int fs_cp(const char *src, const char *dst);
int fs_readFAT(int startBlock, int startBlockOffset, int size, char *buffer);

int fs_writeFAT(int startBlock, int startBlockOffset, int size, const char *buffer);
int fs_chmod(char *fileName, uint8_t perm);


#endif
```

5.37 src/PennFAT/interface.c File Reference

```
#include "interface.h"
#include "filesys.h"
```

Functions

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

  Open a file with the given mode.
- int f_close (int fd)

  Close a file.
- int f_read (int fd, int n, char ∗buf)

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

  Write n bytes to the file.
- int f_lseek (int fd, int offset, int whence)

  Seek to a position in the file.
- int f_unlink (const char ∗fname)

  Unlink a file.
- int f_ls (const char ∗filename)

  List all files in the current directory.
- bool f_find (const char ∗filename)

  Check if a file exists.
- bool f_isExecutable (const char ∗filename)

  Check if a file is executable.

Variables

- FATConfig ∗ **fs_FATConfig**
- uint16_t ∗ **fs_FAT16InMemory**
- FdTable **fs_fdTable**
- pcb ∗ **active_process**

5.37.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.37.2 Function Documentation

5.37.2.1 f_close()

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

Close a file.

Parameters

fd	

Returns

int

5.37.2.2 f_find()

```
bool f_find (
             const char * filename )
```

Check if a file exists.

Parameters

filename	

Returns

true

false

5.37.2.3 f_isExecutable()

```
bool f_isExecutable (
            const char * filename )
```

Check if a file is executable.

Parameters

filename	

Returns

true

false

5.37.2.4 f_ls()

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

List all files in the current directory.

Parameters

filename	

Returns

int

5.37.2.5 f_lseek()

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

Seek to a position in the file.

Parameters

fd	
offset	
whence	

Returns

int

5.37.2.6 f_open()

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

Open a file with the given mode.

Parameters

fname	
mode	

Returns

int

5.37.2.7 f_read()

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

Read n bytes from the file.

Parameters

fd	
n	
buf	

Returns

int

5.37.2.8 f_unlink()

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

Unlink a file.

Parameters

fname	

Returns

int

5.37.2.9 f_write()

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

Write n bytes to the file.

Parameters

fd	
str	
n	

Returns

int

5.38 src/PennFAT/interface.h File Reference

```
#include "../kernel/utils.h"
```

Macros

- #define **F_STDIN_FD** 0
- #define **F_STDOUT_FD** 1
- #define **F_ERROR** 2
- #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 **F_SUCCESS** 0
- #define **F_FAILURE** -1

Functions

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

  Open a file with the given mode.
- int f_close (int fd)

  Close a file.
- int f_read (int fd, int n, char ∗buf)

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

  Write n bytes to the file.
- int f_lseek (int fd, int offset, int whence)

  Seek to a position in the file.
- int f_unlink (const char ∗fname)

  Unlink a file.
- int f_ls (const char ∗filename)

  List all files in the current directory.
- bool f_find (const char ∗filename)

  Check if a file exists.
- bool f_isExecutable (const char ∗filename)

  Check if a file is executable.

5.38.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.38.2 Function Documentation

5.38.2.1 f_close()

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

Close a file.

Parameters

fd	

Returns

int

5.38.2.2 f_find()

```
bool f_find (
             const char * filename )
```

Check if a file exists.

Parameters

filename	

Returns

true

false

5.38.2.3 f_isExecutable()

```
bool f_isExecutable (
             const char * filename )
```

Check if a file is executable.

Parameters

filename	

Returns

true

false

5.38.2.4 f_ls()

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

List all files in the current directory.

Parameters

filename	

Returns

int

5.38.2.5 f_lseek()

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

Seek to a position in the file.

Parameters

fd	
offset	
whence	

Returns

int

5.38.2.6 f_open()

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

Open a file with the given mode.

Parameters

fname	
mode	

Returns

int

5.38.2.7 f_read()

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

Read n bytes from the file.

Parameters

fd	
n	
buf	

Returns

int

5.38.2.8 f_unlink()

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

Unlink a file.

Parameters

fname	

Returns

int

5.38.2.9 f_write()

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

Write n bytes to the file.

Parameters

fd	
str	
n	

Returns

int

5.39 interface.h

Go to the documentation of this file.

```

#ifndef FILESYS_INTERFACE_H
#define FILESYS_INTERFACE_H

#define F_STDIN_FD 0
#define F_STDOUT_FD 1
#define F_ERROR 2

#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 F_SUCCESS 0
#define F_FAILURE -1

#include "../kernel/utils.h"

int f_open(const char *fname, int mode);
int f_close(int fd);
int f_read(int fd, int n, char *buf);
int f_write(int fd, const char *str, int n);
int f_lseek(int fd, int offset, int whence);
int f_unlink(const char *fname);
int f_ls(const char *filename);
bool f_find(const char *filename);
bool f_isExecutable(const char *filename);

#endif
```

5.40 src/PennFAT/pennFAT.c File Reference

```
#include "pennFAT.h"
```

Functions

- bool pf_isMounted ()
  Check if the PennFAT is mounted.
- int pf_readFile (const char ∗fileName, int size, char ∗buffer)
  Read a file.
- int pf_writeFile (const char ∗fileName, int size, const char ∗buffer, PF_WRITEMODE mode)
  Write a file.
- int pf_mkfs (const char ∗fsName, int BLOCKS_IN_FAT, int BLOCK_SIZE_CONFIG)
  Make a PennFAT file system.
- int pf_mount (const char ∗fsName)
  Mount a PennFAT file system.
- int pf_umount ()
  Unmount a PennFAT file system.
- int pf_touch (const char ∗fileName)
  Create a file.
- int pf_rm (const char ∗fileName)
  Remove a file.
- int pf_mv (const char ∗src, const char ∗dst)
  Rename a file.
- int pf_ls ()
  List all files.
- int pf_chmod (const char ∗fileName, uint8_t perm)
  Change the permission of a file.
- int pf_catFiles (char ∗∗fileNames, int fileNum, int ∗size, char ∗buffer)
  Cat files.
- void SIGINTHandler (int sig)
  Signal handler for SIGINT.
- char ∗ readInput (char ∗inputBuffer)
  Read input utility function.
- int parseInput (char ∗userInput, char ∗∗argsBuffer, int ∗argNum)
  Parse input utility function.
- int **main** (int argc, char ∗∗argv)

5.40.1 Detailed Description

Author

Zhiyuan Liang (`liangzhy@seas.upenn.edu`)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.40.2 Function Documentation

5.40.2.1 parseInput()

```
int parseInput (
            char * userInput,
            char ** argsBuffer,
            int * argNum )
```

Parse input utility function.

Parameters

userInput	
argsBuffer	
argNum	

Returns

int

5.40.2.2 pf_catFiles()

```
int pf_catFiles (
            char ** fileNames,
            int fileNum,
            int * size,
            char * buffer )
```

Cat files.

Parameters

fileNames	
fileNum	
size	
buffer	

Returns

int

5.40.2.3 pf_chmod()

```
int pf_chmod (
            const char * fileName,
            uint8_t perm )
```

Change the permission of a file.

Parameters

fileName	
perm	

Returns

int

5.40.2.4 pf_isMounted()

```
bool pf_isMounted ( )
```

Check if the PennFAT is mounted.

Returns

true

false

5.40.2.5 pf_ls()

```
int pf_ls ( )
```

List all files.

Returns

int

5.40.2.6 pf_mkfs()

```
int pf_mkfs (
            const char * fsName,
            int BLOCKS_IN_FAT,
            int BLOCK_SIZE_CONFIG )
```

Make a PennFAT file system.

Parameters

fsName	
BLOCKS_IN_FAT	
BLOCK_SIZE_CONFIG	

Returns

int

5.40.2.7 pf_mount()

```
int pf_mount (
            const char * fsName )
```

Mount a PennFAT file system.

Parameters

fsName	

Returns

int

5.40.2.8 pf_mv()

```
int pf_mv (
            const char * src,
            const char * dst )
```

Rename a file.

Parameters

src	
dst	

Returns

int

5.40.2.9 pf_readFile()

```
int pf_readFile (
            const char * fileName,
            int size,
            char * buffer )
```

Read a file.

Parameters

fileName	
size	
buffer	

Returns

int

5.40.2.10 pf_rm()

```
int pf_rm (
            const char * fileName )
```

Remove a file.

Parameters

fileName	

Returns

int

5.40.2.11 pf_touch()

```
int pf_touch (
            const char * fileName )
```

Create a file.

Parameters

fileName	

Returns

int

5.40.2.12 pf_umount()

```
int pf_umount ( )
```

Unmount a PennFAT file system.

Returns

int

5.40.2.13 pf_writeFile()

```
int pf_writeFile (
             const char * fileName,
             int size,
             const char * buffer,
             PF_WRITEMODE mode )
```

Write a file.

Parameters

fileName	
size	
buffer	
mode	

Returns

int

5.40.2.14 readInput()

```
char * readInput (
             char * inputBuffer )
```

Read input utility function.

Parameters

inputBuffer	

Returns

char∗

5.40.2.15 SIGINTHandler()

```
void SIGINTHandler (
            int sig )
```

Signal handler for SIGINT.

Parameters

sig	

5.41 src/PennFAT/pennFAT.h File Reference

```
#include "filesys.h"
#include "signal.h"
```

Macros

- #define **PF_MAX_BUFFER_SIZE** 32512
- #define **PF_MAX_FILE_NUM** 4
- #define **MAX_LINE_LENGTH** 4096
- #define **MAX_ARGS_NUM** 8
- #define **EXIT_SHREDDER** -1
- #define **EMPTY_LINE** 0
- #define **EXECUTE_COMMAND** 1

Enumerations

- enum PF_WRITEMODE { **PF_OVERWRITE** , **PF_APPEND** , **PF_STDOUT** }

  PennFAT write mode.

Functions

- bool pf_isMounted ()

    Check if the PennFAT is mounted.
- int pf_readFile (const char ∗fileName, int size, char ∗buffer)

    Read a file.
- int pf_writeFile (const char ∗fileName, int size, const char ∗buffer, PF_WRITEMODE mode)

    Write a file.
- int pf_mkfs (const char ∗fsName, int BLOCKS_IN_FAT, int BLOCK_SIZE_CONFIG)

    Make a PennFAT file system.
- int pf_mount (const char ∗fsName)

    Mount a PennFAT file system.
- int pf_umount ()

    Unmount a PennFAT file system.
- int pf_touch (const char ∗fileName)

    Create a file.
- int pf_rm (const char ∗fileName)

    Remove a file.
- int pf_mv (const char ∗src, const char ∗dst)

    Rename a file.
- int pf_ls ()

    List all files.
- int pf_chmod (const char ∗fileName, uint8_t perm)

    Change the permission of a file.
- int pf_catFiles (char ∗∗fileNames, int fileNum, int ∗size, char ∗buffer)

    Cat files.
- void SIGINTHandler (int sig)

    Signal handler for SIGINT.
- char ∗ readInput (char ∗inputBuffer)

    Read input utility function.
- int parseInput (char ∗userInput, char ∗∗argsBuffer, int ∗argNum)

    Parse input utility function.

Variables

- FATConfig ∗ **fs_FATConfig**
- uint16_t ∗ **fs_FAT16InMemory**

5.41.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

5.41.2 Enumeration Type Documentation

5.41.2.1 PF_WRITEMODE

```
enum PF_WRITEMODE
```

PennFAT write mode.

5.41.3 Function Documentation

5.41.3.1 parseInput()

```
int parseInput (
            char * userInput,
            char ** argsBuffer,
            int * argNum )
```

Parse input utility function.

Parameters

userInput	
argsBuffer	
argNum	

Returns

int

5.41.3.2 pf_catFiles()

```
int pf_catFiles (
            char ** fileNames,
            int fileNum,
            int * size,
            char * buffer )
```

Cat files.

Parameters

fileNames	
fileNum	
size	
buffer	

Returns

int

5.41.3.3 pf_chmod()

```
int pf_chmod (
            const char * fileName,
            uint8_t perm )
```

Change the permission of a file.

Parameters

fileName	
perm	

Returns

int

5.41.3.4 pf_isMounted()

```
bool pf_isMounted ( )
```

Check if the PennFAT is mounted.

Returns

true

false

5.41.3.5 pf_ls()

```
int pf_ls ( )
```

List all files.

Returns

int

5.41.3.6 pf_mkfs()

```
int pf_mkfs (
             const char * fsName,
             int BLOCKS_IN_FAT,
             int BLOCK_SIZE_CONFIG )
```

Make a PennFAT file system.

Parameters

fsName	
BLOCKS_IN_FAT	
BLOCK_SIZE_CONFIG	

Returns

int

5.41.3.7 pf_mount()

```
int pf_mount (
             const char * fsName )
```

Mount a PennFAT file system.

Parameters

fsName	

Returns

int

5.41.3.8 pf_mv()

```
int pf_mv (
            const char * src,
            const char * dst )
```

Rename a file.

Parameters

src	
dst	

Returns

int

5.41.3.9 pf_readFile()

```
int pf_readFile (
            const char * fileName,
            int size,
            char * buffer )
```

Read a file.

Parameters

fileName	
size	
buffer	

Returns

int

5.41.3.10 pf_rm()

```
int pf_rm (
            const char * fileName )
```

Remove a file.

Parameters

fileName	

Returns

int

5.41.3.11 pf_touch()

```
int pf_touch (
             const char * fileName )
```

Create a file.

Parameters

fileName	

Returns

int

5.41.3.12 pf_umount()

```
int pf_umount ( )
```

Unmount a PennFAT file system.

Returns

int

5.41.3.13 pf_writeFile()

```
int pf_writeFile (
             const char * fileName,
             int size,
             const char * buffer,
             PF_WRITEMODE mode )
```

Write a file.

Parameters

fileName	
size	
buffer	
mode	

Returns

int

5.41.3.14 readInput()

```
char * readInput (
             char * inputBuffer )
```

Read input utility function.

Parameters

inputBuffer	

Returns

char∗

5.41.3.15 SIGINTHandler()

```
void SIGINTHandler (
             int sig )
```

Signal handler for SIGINT.

Parameters

sig	

5.42 pennFAT.h

Go to the documentation of this file.

```

#ifndef PENNFAT_H
#define PENNFAT_H

#include "filesys.h"
#include "signal.h"

#define PF_MAX_BUFFER_SIZE 32512
#define PF_MAX_FILE_NUM 4

extern FATConfig *fs_FATConfig;
extern uint16_t *fs_FAT16InMemory;

typedef enum {
    PF_OVERWRITE,
    PF_APPEND,
    PF_STDOUT
} PF_WRITEMODE;

bool pf_isMounted();
int pf_readFile(const char *fileName, int size, char *buffer);
int pf_writeFile(const char *fileName, int size, const char *buffer, PF_WRITEMODE mode);
;
int pf_mkfs(const char *fsName, int BLOCKS_IN_FAT, int BLOCK_SIZE_CONFIG);
int pf_mount(const char *fsName);
int pf_umount();
int pf_touch(const char *fileName);
int pf_rm(const char *fileName);
int pf_mv(const char *src, const char *dst);
int pf_ls();
int pf_chmod(const char *fileName, uint8_t perm);
int pf_catFiles(char **fileNames, int fileNum, int *size, char *buffer);


#define MAX_LINE_LENGTH 4096
#define MAX_ARGS_NUM 8

#define EXIT_SHREDDER -1
#define EMPTY_LINE 0
#define EXECUTE_COMMAND 1
void SIGINTHandler(int sig);
char *readInput(char *inputBuffer);
int parseInput(char *userInput, char **argsBuffer, int *argNum);

#endif
```

5.43 src/PennFAT/test-playground.c File Reference

```
#include "interface.h"
#include "pennFAT.h"
```

Functions

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

Variables

- pcb * **active_process**

5.43.1 Detailed Description

Author

Zhiyuan Liang (liangzhy@seas.upenn.edu)

Version

0.1

Date

2023-04-16

Copyright

Copyright (c) 2023

Index

appendFdTable
  fd-table.c, 98
  fd-table.h, 103
appendJobList
  job.c, 24
  job.h, 35

behavior.c
  executeLine, 14
  executeProgram, 14
  executeScript, 15
  parseLine, 15
  parseProgramType, 15
  parseUserInput, 16
  readAndParseUserInput, 16
  readUserInput, 16
  writePrompt, 17
behavior.h
  executeLine, 19
  executeProgram, 19
  executeScript, 19
  LineType, 18
  parseLine, 20
  parseProgramType, 20
  parseUserInput, 20
  ProgramType, 18
  readAndParseUserInput, 21
  readUserInput, 21
  writePrompt, 21
bgBuildinCommand
  job.c, 25
  job.h, 35

clearFdTable
  fd-table.c, 98
  fd-table.h, 103
clearJobList
  job.c, 25
  job.h, 35
createDirectoryEntry
  FAT.c, 77
  FAT.h, 87
createFAT16InMemory
  FAT.c, 77
  FAT.h, 88
createFATConfig
  FAT.c, 78
  FAT.h, 88
createFATOnDisk
  FAT.c, 78
  FAT.h, 89
createFdNode
  fd-table.c, 98
  fd-table.h, 103
createFileDirectoryOnDisk
  FAT.c, 78
  FAT.h, 89
createJob
  job.c, 25
  job.h, 35

deconstruct_idle
  user.h, 68
deconstruct_shell
  user.h, 69
deleteFileDirectory
  FAT.c, 79
  FAT.h, 89
deleteFileDirectoryByName
  FAT.c, 79
  FAT.h, 90
DirectoryEntry, 7
  FAT.h, 87

executeBuiltinCommand
  job.c, 26
  job.h, 36
executeLine
  behavior.c, 14
  behavior.h, 19
executeProgram
  behavior.c, 14
  behavior.h, 19
executeScript
  behavior.c, 15
  behavior.h, 19

f_close
  interface.c, 119
  interface.h, 124
f_find
  interface.c, 119
  interface.h, 124
f_isExecutable
  interface.c, 120
  interface.h, 124
f_ls
  interface.c, 120
  interface.h, 125
f_lseek

interface.c, 120
interface.h, 125
f_open
interface.c, 121
interface.h, 125
f_read
interface.c, 121
interface.h, 126
f_unlink
interface.c, 122
interface.h, 126
f_write
interface.c, 122
interface.h, 127
FAT.c
createDirectoryEntry, 77
createFAT16InMemory, 77
createFATConfig, 78
createFATOnDisk, 78
createFileDirectoryOnDisk, 78
deleteFileDirectory, 79
deleteFileDirectoryByName, 79
findEmptyFAT16Entry, 80
findFileDirectory, 80
isDirectoryEntryToDelete, 81
readDirectoryEntry, 81
readFAT, 81
traceBytesFromBeginning, 82
traceBytesToEnd, 82
traceFileEnd, 83
traceOffset, 83
writeFAT, 84
writeFileDirectory, 84
FAT.h
createDirectoryEntry, 87
createFAT16InMemory, 88
createFATConfig, 88
createFATOnDisk, 89
createFileDirectoryOnDisk, 89
deleteFileDirectory, 89
deleteFileDirectoryByName, 90
DirectoryEntry, 87
findEmptyFAT16Entry, 90
findFileDirectory, 91
isDirectoryEntryToDelete, 91
readDirectoryEntry, 92
readFAT, 92
traceBytesFromBeginning, 93
traceBytesToEnd, 93
traceFileEnd, 93
traceOffset, 94
writeFAT, 94
writeFileDirectory, 95
FATConfig, 8
fd-table.c
appendFdTable, 98
clearFdTable, 98
createFdNode, 98
findAvailableFd, 99
initFdTable, 99
isFileBeingUsed, 99
isFileBeingWritten, 100
removeFdNode, 100
fd-table.h
appendFdTable, 103
clearFdTable, 103
createFdNode, 103
FdTable, 102
findAvailableFd, 104
initFdTable, 104
isFileBeingUsed, 105
isFileBeingWritten, 105
removeFdNode, 105
FdNode, 8
FdTable, 9
fd-table.h, 102
fgBuildinCommand
job.c, 26
job.h, 36
filesys.c
fs_chmod, 108
fs_cp, 108
fs_mkfs, 108
fs_mount, 109
fs_mv, 109
fs_readFAT, 109
fs_rm, 110
fs_touch, 110
fs_unmount, 111
fs_writeFAT, 111
isFileSystemMounted, 111
isValidFileName, 111
filesys.h
fs_chmod, 113
fs_cp, 113
fs_mkfs, 114
fs_mount, 114
fs_mv, 114
fs_readFAT, 115
fs_rm, 115
fs_touch, 116
fs_unmount, 116
fs_writeFAT, 116
isFileSystemMounted, 117
isValidFileName, 117
findAvailableFd
fd-table.c, 99
fd-table.h, 104
findEmptyFAT16Entry
FAT.c, 80
FAT.h, 90
findFileDirectory
FAT.c, 80
FAT.h, 91
findJobList
job.c, 26

job.h, 37
findJobListByJobId
job.c, 27
job.h, 37
findTheCurrentJob
job.c, 27
job.h, 37
fs_chmod
filesys.c, 108
filesys.h, 113
fs_cp
filesys.c, 108
filesys.h, 113
fs_mkfs
filesys.c, 108
filesys.h, 114
fs_mount
filesys.c, 109
filesys.h, 114
fs_mv
filesys.c, 109
filesys.h, 114
fs_readFAT
filesys.c, 109
filesys.h, 115
fs_rm
filesys.c, 110
filesys.h, 115
fs_touch
filesys.c, 110
filesys.h, 116
fs_unmount
filesys.c, 111
filesys.h, 116
fs_writeFAT
filesys.c, 111
filesys.h, 116

get_node_by_pid_all_alive_queues
user.h, 69

initFdTable
fd-table.c, 99
fd-table.h, 104
initJobList
job.c, 27
job.h, 38
interface.c
f_close, 119
f_find, 119
f_isExecutable, 120
f_ls, 120
f_lseek, 120
f_open, 121
f_read, 121
f_unlink, 122
f_write, 122
interface.h
f_close, 124
f_find, 124
f_isExecutable, 124
f_ls, 125
f_lseek, 125
f_open, 125
f_read, 126
f_unlink, 126
f_write, 127
isBuildinCommand
shell.c, 63
shell.h, 65
isDirectoryEntryToDelete
FAT.c, 81
FAT.h, 91
isFileBeingUsed
fd-table.c, 99
fd-table.h, 105
isFileBeingWritten
fd-table.c, 100
fd-table.h, 105
isFileSystemMounted
filesys.c, 111
filesys.h, 117
isKnownProgram
shell.c, 63
shell.h, 66
isValidFileName
filesys.c, 111
filesys.h, 117

Job, 9
job.c
appendJobList, 24
bgBuildinCommand, 25
clearJobList, 25
createJob, 25
executeBuiltinCommand, 26
fgBuildinCommand, 26
findJobList, 26
findJobListByJobId, 27
findTheCurrentJob, 27
initJobList, 27
jobsBuildinCommand, 28
killBuildinCommand, 28
manBuildinCommand, 28
nicePidBuildinCommand, 28
parseBuiltinCommandType, 28
pollBackgroundProcesses, 29
popJobList, 29
printCommandLine, 29
printJobList, 30
removeJobList, 30
removeJobListByJobId, 30
removeJobListWithoutFreeCmd, 31
updateJobList, 31
updateJobListByJobId, 32
writeJobState, 32
writeJobStatePrompt, 32
writeNewline, 32

job.h
    appendJobList, 35
    bgBuildinCommand, 35
    clearJobList, 35
    createJob, 35
    executeBuiltinCommand, 36
    fgBuildinCommand, 36
    findJobList, 37
    findJobListByJobId, 37
    findTheCurrentJob, 37
    initJobList, 38
    jobsBuildinCommand, 38
    killBuildinCommand, 38
    manBuildinCommand, 38
    nicePidBuildinCommand, 39
    parseBuiltinCommandType, 39
    pollBackgroundProcesses, 39
    popJobList, 39
    printCommandLine, 40
    printJobList, 40
    removeJobList, 40
    removeJobListByJobId, 41
    removeJobListWithoutFreeCmd, 41
    updateJobList, 41
    updateJobListByJobId, 43
    writeJobState, 43
    writeJobStatePrompt, 43
    writeNewline, 44
JobList, 10
JobListNode, 10
jobsBuildinCommand
    job.c, 28
    job.h, 38

killBuildinCommand
    job.c, 28
    job.h, 38

LineType
    behavior.h, 18

manBuildinCommand
    job.c, 28
    job.h, 38

nicePidBuildinCommand
    job.c, 28
    job.h, 39

p_exit
    user.h, 69
p_kill
    user.h, 69
p_nice
    user.h, 70
p_perror
    perrno.h, 47
p_set_errno
    perrno.h, 47
p_sleep
    user.h, 70
p_spawn
    user.h, 70
p_waitpid
    user.h, 71
parseBuiltinCommandType
    job.c, 28
    job.h, 39
parsed_command, 10
parseInput
    pennFAT.c, 129
    pennFAT.h, 136
parseLine
    behavior.c, 15
    behavior.h, 20
parseProgramType
    behavior.c, 15
    behavior.h, 20
parseUserInput
    behavior.c, 16
    behavior.h, 20
pcb, 11
pcb_node, 11
pcb_queue, 12
pennFAT.c
    parseInput, 129
    pf_catFiles, 129
    pf_chmod, 129
    pf_isMounted, 130
    pf_ls, 130
    pf_mkfs, 130
    pf_mount, 131
    pf_mv, 131
    pf_readFile, 131
    pf_rm, 132
    pf_touch, 132
    pf_umount, 133
    pf_writeFile, 133
    readInput, 133
    SIGINTHandler, 134
pennFAT.h
    parseInput, 136
    pf_catFiles, 136
    pf_chmod, 137
    pf_isMounted, 137
    pf_ls, 137
    pf_mkfs, 138
    pf_mount, 138
    pf_mv, 138
    pf_readFile, 139
    pf_rm, 139
    pf_touch, 140
    pf_umount, 140
    pf_writeFile, 140
    PF_WRITEMODE, 136
    readInput, 141
    SIGINTHandler, 141

perrno.h
    p_perror, 47
    p_set_errno, 47
pf_catFiles
    pennFAT.c, 129
    pennFAT.h, 136
pf_chmod
    pennFAT.c, 129
    pennFAT.h, 137
pf_isMounted
    pennFAT.c, 130
    pennFAT.h, 137
pf_ls
    pennFAT.c, 130
    pennFAT.h, 137
pf_mkfs
    pennFAT.c, 130
    pennFAT.h, 138
pf_mount
    pennFAT.c, 131
    pennFAT.h, 138
pf_mv
    pennFAT.c, 131
    pennFAT.h, 138
pf_readFile
    pennFAT.c, 131
    pennFAT.h, 139
pf_rm
    pennFAT.c, 132
    pennFAT.h, 139
pf_touch
    pennFAT.c, 132
    pennFAT.h, 140
pf_umount
    pennFAT.c, 133
    pennFAT.h, 140
pf_writeFile
    pennFAT.c, 133
    pennFAT.h, 140
PF_WRITEMODE
    pennFAT.h, 136
pollBackgroundProcesses
    job.c, 29
    job.h, 39
popJobList
    job.c, 29
    job.h, 39
printCommandLine
    job.c, 29
    job.h, 40
printJobList
    job.c, 30
    job.h, 40
priority_queue, 12
programs.c
    s_busy, 49
    s_cat, 50
    s_chmod, 50
    s_cp, 50
    s_echo, 50
    s_hang, 51
    s_kill, 51
    s_ls, 51
    s_mv, 52
    s_nohang, 52
    s_orphanify, 52
    s_ps, 52
    s_recur, 53
    s_rm, 53
    s_sleep, 53
    s_test, 54
    s_touch, 54
    s_zombify, 54
programs.h
    s_busy, 56
    s_cat, 56
    s_chmod, 56
    s_cp, 57
    s_echo, 57
    s_hang, 57
    s_kill, 58
    s_ls, 58
    s_mv, 58
    s_nohang, 58
    s_orphanify, 59
    s_ps, 59
    s_recur, 59
    s_rm, 60
    s_sleep, 60
    s_test, 60
    s_touch, 60
    s_zombify, 61
ProgramType
    behavior.h, 18

readAndParseUserInput
    behavior.c, 16
    behavior.h, 21
readDirectoryEntry
    FAT.c, 81
    FAT.h, 92
readFAT
    FAT.c, 81
    FAT.h, 92
readInput
    pennFAT.c, 133
    pennFAT.h, 141
readUserInput
    behavior.c, 16
    behavior.h, 21
removeFdNode
    fd-table.c, 100
    fd-table.h, 105
removeJobList
    job.c, 30
    job.h, 40
removeJobListByJobId

job.c, 30
job.h, 41
removeJobListWithoutFreeCmd
job.c, 31
job.h, 41

s_busy
programs.c, 49
programs.h, 56
s_cat
programs.c, 50
programs.h, 56
s_chmod
programs.c, 50
programs.h, 56
s_cp
programs.c, 50
programs.h, 57
s_echo
programs.c, 50
programs.h, 57
s_hang
programs.c, 51
programs.h, 57
s_kill
programs.c, 51
programs.h, 58
s_ls
programs.c, 51
programs.h, 58
s_mv
programs.c, 52
programs.h, 58
s_nohang
programs.c, 52
programs.h, 58
s_orphanify
programs.c, 52
programs.h, 59
s_ps
programs.c, 52
programs.h, 59
s_recur
programs.c, 53
programs.h, 59
s_rm
programs.c, 53
programs.h, 60
s_sleep
programs.c, 53
programs.h, 60
s_test
programs.c, 54
programs.h, 60
s_touch
programs.c, 54
programs.h, 60
s_zombify
programs.c, 54
programs.h, 61
shell.c
isBuildinCommand, 63
isKnownProgram, 63
shell_init, 64
shell_process, 64
shell.h
isBuildinCommand, 65
isKnownProgram, 66
shell_init, 66
shell_process, 66
shell_init
shell.c, 64
shell.h, 66
shell_process
shell.c, 64
shell.h, 66
SIGINTHandler
pennFAT.c, 134
pennFAT.h, 141
src/kernel/behavior.c, 13
src/kernel/behavior.h, 17, 22
src/kernel/global.h, 22
src/kernel/global2.h, 23
src/kernel/job.c, 23
src/kernel/job.h, 33, 44
src/kernel/kernel.h, 45
src/kernel/log.h, 45
src/kernel/parser.h, 45
src/kernel/perrno.h, 46, 48
src/kernel/programs.c, 48
src/kernel/programs.h, 54, 61
src/kernel/scheduler.h, 62
src/kernel/shell.c, 62
src/kernel/shell.h, 65, 67
src/kernel/stress.h, 67
src/kernel/user.h, 67, 72
src/kernel/utils.h, 73
src/PennFAT/FAT.c, 75
src/PennFAT/FAT.h, 85, 95
src/PennFAT/fd-table.c, 97
src/PennFAT/fd-table.h, 101, 106
src/PennFAT/filesys.c, 106
src/PennFAT/filesys.h, 112, 117
src/PennFAT/interface.c, 118
src/PennFAT/interface.h, 123, 127
src/PennFAT/pennFAT.c, 128
src/PennFAT/pennFAT.h, 134, 141
src/PennFAT/test-playground.c, 142
src/PennFAT/utils.c, 72
src/PennFAT/utils.h, 74, 75

traceBytesFromBeginning
FAT.c, 82
FAT.h, 93
traceBytesToEnd
FAT.c, 82
FAT.h, 93
traceFileEnd

FAT.c, 83
FAT.h, 93
traceOffset
FAT.c, 83
FAT.h, 94

updateJobList
job.c, 31
job.h, 41
updateJobListByJobId
job.c, 32
job.h, 43
user.h
deconstruct_idle, 68
deconstruct_shell, 69
get_node_by_pid_all_alive_queues, 69
p_exit, 69
p_kill, 69
p_nice, 70
p_sleep, 70
p_spawn, 70
p_waitpid, 71
writePrompt, 71

writeFAT
FAT.c, 84
FAT.h, 94
writeFileDirectory
FAT.c, 84
FAT.h, 95
writeJobState
job.c, 32
job.h, 43
writeJobStatePrompt
job.c, 32
job.h, 43
writeNewline
job.c, 32
job.h, 44
writePrompt
behavior.c, 17
behavior.h, 21
user.h, 71