

pennOS

Generated by Doxygen 1.8.17



<b>1 Data Structure Index</b>	<b>1</b>
1.1 Data Structures	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Data Structure Documentation</b>	<b>5</b>
3.1 Define Struct Reference	5
3.1.1 Detailed Description	5
3.2 DIR_NODE Struct Reference	5
3.3 File Struct Reference	6
3.4 FILE_DESCRIPTOR Struct Reference	6
3.5 FILE_NODE Struct Reference	6
3.6 I_Node Struct Reference	7
3.6.1 Detailed Description	7
3.7 I_NODE Struct Reference	7
3.8 job Struct Reference	7
3.9 linked_list Struct Reference	8
3.9.1 Detailed Description	8
3.10 Navigate Struct Reference	8
3.10.1 Detailed Description	8
3.11 node Struct Reference	8
3.11.1 Detailed Description	9
3.12 PARSE_RESULT Struct Reference	9
3.13 pcb Struct Reference	9
3.13.1 Field Documentation	10
3.13.1.1 context	10
3.13.1.2 pq_node	10
3.13.1.3 zombie_children	10
3.14 proc_info Struct Reference	10
3.15 Struct Struct Reference	11
3.15.1 Detailed Description	11
3.16 tokenizer Struct Reference	11
3.16.1 Detailed Description	11
3.17 Tree Struct Reference	11
3.17.1 Detailed Description	11
3.18 trie_node Struct Reference	11
3.19 wait_info Struct Reference	12
<b>4 File Documentation</b>	<b>13</b>
4.1 src/file_system/FAT.h File Reference	13
4.1.1 Detailed Description	14
4.1.2 Function Documentation	14

4.1.2.1	append_FAT_entry()	14
4.1.2.2	clear_FAT()	14
4.1.2.3	create_FAT()	15
4.1.2.4	create_temp_map()	15
4.1.2.5	delete_FAT()	15
4.1.2.6	free_FAT_entry()	15
4.1.2.7	get_free_block()	16
4.1.2.8	list_FAT()	16
4.1.2.9	read_next_FAT_entry()	16
4.1.2.10	register_FAT_entry()	17
4.1.2.11	save_FAT()	17
4.2	src/file_system/FAT_dir.h File Reference	17
4.2.1	Detailed Description	18
4.2.2	Function Documentation	18
4.2.2.1	create_dir()	18
4.2.2.2	free_all_dirs()	19
4.2.2.3	init_top_dir()	19
4.2.2.4	path_parser()	19
4.2.2.5	return_to_top_dir()	20
4.2.2.6	search_dir_by_path()	20
4.2.2.7	visit_dir()	20
4.3	src/file_system/FAT_file.h File Reference	21
4.3.1	Detailed Description	22
4.3.2	Function Documentation	22
4.3.2.1	create_file()	22
4.3.2.2	delete_file_by_block()	22
4.3.2.3	delete_file_by_name()	23
4.3.2.4	delete_file_by_path()	23
4.3.2.5	free_all_files()	23
4.3.2.6	print_file_content()	24
4.3.2.7	read_all_from_dump()	24
4.3.2.8	read_file_content()	24
4.3.2.9	save_all_to_dump()	25
4.3.2.10	search_file_by_block()	25
4.3.2.11	search_file_by_path()	25
4.3.2.12	write_all()	27
4.4	src/file_system/file_system_constant.h File Reference	27
4.4.1	Detailed Description	28
4.5	src/file_system/l_node.h File Reference	28
4.5.1	Detailed Description	29
4.5.2	Function Documentation	29
4.5.2.1	create_inode_list()	29

4.5.2.2 delete_inode_list()	29
4.6 src/file_system/interface.h File Reference	29
4.6.1 Detailed Description	30
4.6.2 Function Documentation	30
4.6.2.1 f_close()	30
4.6.2.2 f_ls()	31
4.6.2.3 f_ls_redirect()	31
4.6.2.4 f_lseek()	31
4.6.2.5 f_open()	32
4.6.2.6 f_read()	32
4.6.2.7 f_unlink()	33
4.6.2.8 f_write()	33
4.6.2.9 init_interface()	33
4.6.2.10 save_interface()	34
4.7 src/kernel/header.h File Reference	34
4.7.1 Detailed Description	35
4.8 src/kernel/kernel.h File Reference	35
4.8.1 Detailed Description	35
4.8.2 Function Documentation	35
4.8.2.1 k_inherit_orphans()	35
4.8.2.2 k_process_cleanup()	36
4.8.2.3 k_process_create()	36
4.8.2.4 k_process_kill()	37
4.8.2.5 k_process_nice()	37
4.8.2.6 k_sleep()	37
4.8.2.7 k_unblock_parent()	38
4.8.2.8 k_wait()	38
4.8.2.9 k_wait_pid()	38
4.9 src/kernel/node.h File Reference	39
4.9.1 Detailed Description	39
4.10 src/kernel/pcb.h File Reference	40
4.10.1 Detailed Description	40
4.10.2 Function Documentation	40
4.10.2.1 create_pcb()	40
4.11 src/kernel/scheduler.h File Reference	41
4.11.1 Detailed Description	41
4.11.2 Function Documentation	42
4.11.2.1 add_node_to_ready_queue()	42
4.11.2.2 change_status()	42
4.11.2.3 mkcontext()	42
4.11.2.4 pick_thread()	43
4.11.2.5 remove_node_from_ready_queue()	43

4.11.2.6 timer_interrupt()	43
4.12 src/kernel/user.h File Reference	43
4.12.1 Detailed Description	44
4.12.2 Function Documentation	44
4.12.2.1 p_exit()	44
4.12.2.2 p_info()	45
4.12.2.3 p_kill()	45
4.12.2.4 p_nice()	45
4.12.2.5 p_sleep()	46
4.12.2.6 p_spawn()	46
4.12.2.7 p_wait()	47
4.12.2.8 print_proc_info()	47
4.12.2.9 W_WIFCONTINUED()	47
4.12.2.10 W_WIFEXITED()	48
4.12.2.11 W_WIFSIGNALED()	48
4.12.2.12 W_WIFSTOPPED()	48
4.13 src/shell/job.h File Reference	50
4.13.1 Detailed Description	51
4.13.2 Function Documentation	51
4.13.2.1 create()	51
4.13.2.2 delete_by_label()	52
4.13.2.3 deleteFirst()	52
4.13.2.4 find_by_label()	52
4.13.2.5 find_by_label_stop()	53
4.13.2.6 find_by_pid()	53
4.13.2.7 find_recent_stop()	54
4.13.2.8 free_job()	54
4.13.2.9 getSize()	54
4.13.2.10 insertFirst()	55
4.13.2.11 printInfo()	55
4.13.2.12 printJobs()	55
4.13.2.13 printList()	55
4.13.2.14 remove_label()	56
4.14 src/shell/shell.h File Reference	56
4.14.1 Detailed Description	57
4.14.2 Function Documentation	57
4.14.2.1 get_minimal_id()	57
4.14.2.2 signal_handler()	57
4.14.3 Variable Documentation	58
4.14.3.1 print_msg	58
4.15 src/shell/user_functions.h File Reference	58
4.15.1 Detailed Description	58

---

4.15.2 Function Documentation . . . . .	58
4.15.2.1 auto_complete() . . . . .	58
4.15.2.2 match_function() . . . . .	59
4.15.2.3 user_cd() . . . . .	59
<b>Index</b>	<b>61</b>





# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

Define	5
DIR_NODE	5
File	6
FILE_DESCRIPTOR	6
FILE_NODE	6
I_Node	7
I_NODE	7
job	7
linked_list	
Create a double linked list with dummy head and dummy tail	8
Navigate	8
node	
Create a single node which could form linked list	8
PARSE_RESULT	9
pcb	9
proc_info	10
Struct	11
tokenizer	11
Tree	11
trie_node	11
wait_info	12



## Chapter 2

# File Index

### 2.1 File List

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

src/file_system/ <b>disk.h</b>	??
src/file_system/ <a href="#">FAT.h</a>	
The running part of the file system	13
src/file_system/ <a href="#">FAT_dir.h</a>	
Define the directory node structure needed and functions needed	17
src/file_system/ <a href="#">FAT_file.h</a>	
Create file node for file system	21
src/file_system/ <a href="#">file_system_constant.h</a>	
Define some common global variables that needed in the file system	27
src/file_system/ <a href="#">l_node.h</a>	
Define some global variables and functions needed	28
src/file_system/ <a href="#">interface.h</a>	
Define struct and functions for file descriptor and implement system calls API	29
src/kernel/ <a href="#">header.h</a>	
Include all libraru and define struct needed to only store information	34
src/kernel/ <b>init.h</b>	??
src/kernel/ <a href="#">kernel.h</a>	
All the kernel functions that may only be called from the kernel side of the operating system	35
src/kernel/ <b>logger.h</b>	??
src/kernel/ <a href="#">node.h</a>	
Create linked list node, and double linked list structure	39
src/kernel/ <a href="#">pcb.h</a>	
Create the process control block for each process to store necessary information	40
src/kernel/ <a href="#">scheduler.h</a>	
Scheduler for all threads to run in round robin policy	41
src/kernel/ <a href="#">user.h</a>	
User level functions	43
src/shell/ <b>error.h</b>	??
src/shell/ <a href="#">job.h</a>	
Each command is a job, and they will be in a single linked list	50
src/shell/ <a href="#">shell.h</a>	
Some helper functions and extern variable declared here	56
src/shell/ <b>tokenizer.h</b>	??
src/shell/ <a href="#">user_functions.h</a>	
All the command will call corresponding functions	58



## Chapter 3

# Data Structure Documentation

### 3.1 Define Struct Reference

#### 3.1.1 Detailed Description

of the top directory

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

- [src/file\\_system/FAT\\_dir.h](#)

### 3.2 DIR\_NODE Struct Reference

#### Data Fields

- struct [DIR\\_NODE](#) \* **parent**
- struct [DIR\\_NODE](#) \* **next\_sibling**
- struct [DIR\\_NODE](#) \* **first\_child**
- char [dir\\_name](#) [MAX\_NAME\_LENGTH]  
*Name of some sub-level directory.*
- char [dir\\_path](#) [MAX\_PATH\_LENGTH]  
*Path of the current directory.*
- int [sub\\_dir\\_count](#)  
*Number of sub-directories under this dir.*
- int [file\\_count](#)  
*Number of files under this dir.*

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

- [src/file\\_system/FAT\\_dir.h](#)

### 3.3 File Struct Reference

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

- `src/file_system/interface.h`

### 3.4 FILE\_DESCRIPTOR Struct Reference

#### Data Fields

- `char file_path` [MAX\_PATH\_LENGTH]
- `char file_name` [MAX\_NAME\_LENGTH]
- `int mode`
- `int pointer`

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

- `src/file_system/interface.h`

### 3.5 FILE\_NODE Struct Reference

#### Data Fields

- `struct FILE_NODE * next`  
*Next file of all files in all dir.*
- `struct DIR_NODE * curr_dir`  
*Directory holding that file.*
- `char file_name` [MAX\_NAME\_LENGTH]  
*Name of the file.*
- `char dir_name` [MAX\_NAME\_LENGTH]  
*Name of the directory holding that file.*
- `char file_path` [MAX\_PATH\_LENGTH]  
*Full path of the file.*
- `char * file_content`  
*Content of the file.*
- `int file_size`  
*Size of current file, in number of blocks.*
- `int file_bytes`  
*Size of current file, in number of bytes.*
- `unsigned char start_block`  
*Starting block number.*

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

- `src/file_system/FAT_file.h`

## 3.6 I\_Node Struct Reference

### 3.6.1 Detailed Description

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

- [src/file\\_system/I\\_node.h](#)

## 3.7 I\_NODE Struct Reference

### Data Fields

- unsigned char **size**
- unsigned char \* **address\_list**

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

- [src/file\\_system/I\\_node.h](#)

## 3.8 job Struct Reference

### Data Fields

- struct [job](#) \* [next](#)  
*next running command*
- int [pid](#)  
*pid*
- int [gpId](#)  
*group id*
- int \* [pids](#)  
*all information*
- bool [bg\\_node](#)  
*whether running in the background*
- char \* [command](#)  
*command name*
- int [status](#)  
*Status to shell.*
- int [label](#)  
*label for jobs information*
- int [num\\_process](#)  
*process running at the same time in one command*

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

- [src/shell/job.h](#)

## 3.9 linked\_list Struct Reference

Create a double linked list with dummy head and dummy tail.

```
#include <node.h>
```

### Data Fields

- [NODE](#) \* **head**
- [NODE](#) \* **tail**
- int **size**

### 3.9.1 Detailed Description

Create a double linked list with dummy head and dummy tail.

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

- [src/kernel/node.h](#)

## 3.10 Navigate Struct Reference

### 3.10.1 Detailed Description

directory

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

- [src/file\\_system/FAT\\_dir.h](#)

## 3.11 node Struct Reference

Create a single node which could form linked list.

```
#include <node.h>
```

### Data Fields

- struct [node](#) \* **prev**
- struct [node](#) \* **next**
- int **pid**



### 3.11.1 Detailed Description

Create a single node which could form linked list.

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

- [src/kernel/node.h](#)

## 3.12 PARSE\_RESULT Struct Reference

### Data Fields

- `char ** result`
- `int count`  
*count of char\* in char\*\**

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

- [src/file\\_system/FAT\\_dir.h](#)

## 3.13 pcb Struct Reference

### Data Fields

- `ucontext_t * context`  
*Context for the thread.*
- `int priority_level`  
*priority level for the process*
- `int pid`  
*Pid of the process.*
- `int parent_pid`  
*parent pid of the process*
- `char ** argument`  
*First argument is process name, others are argument for the function.*
- `int status_to_user`  
*Status for shell.*
- `int status_to_os`  
*Status for os.*
- `int w_status`  
*Status of wait for parent to wait on.*
- `bool sleep`  
*Whether sleep or not, helper for distinguishing with sleep and block.*
- `int block_time`
- `NODE * pq_node`  
*If sleep, block time left.*
- `NODE * child_node`
- `LINKED_LIST * zombie_children`  
*Node store in parents children list.*
- `LINKED_LIST * waitable_children`  
*Children finished running and is waitable by parents. Currently they are zombies.*
- `int fd_in`
- `int fd_out`

### 3.13.1 Field Documentation

#### 3.13.1.1 context

`ucontext_t* context`

Context for the thread.

basic info

#### 3.13.1.2 pq\_node

`NODE* pq_node`

If sleep, block time left.

nodes point to this pcb in its parent's lists Node store in ready queue/ block queue/ stop queue

#### 3.13.1.3 zombie\_children

`LINKED_LIST* zombie_children`

Node store in parents children list.

child lists If the children is still running

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

- [src/kernel/pcb.h](#)

## 3.14 proc\_info Struct Reference

### Data Fields

- int **pid**
- int **ppid**
- int **status**
- char \* **command**
- int **priority**

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

- [src/kernel/header.h](#)

## 3.15 Struct Struct Reference

### 3.15.1 Detailed Description

the result

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

- [src/file\\_system/FAT\\_dir.h](#)

## 3.16 tokenizer Struct Reference

```
#include <tokenizer.h>
```

### Data Fields

- `char * str`
- `char * pos`

### 3.16.1 Detailed Description

Control structure for a string tokenizer. Maintains the tokenizer's state.

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

- [src/shell/tokenizer.h](#)

## 3.17 Tree Struct Reference

### 3.17.1 Detailed Description

dirs of different levels

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

- [src/file\\_system/FAT\\_dir.h](#)

## 3.18 trie\_node Struct Reference

### Data Fields

- `struct trie\_node * children [27]`
- `bool isWord`
- `char * word`

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

- [src/shell/user\\_functions.c](#)

## 3.19 wait\_info Struct Reference

### Data Fields

- int **pid**
- int **status**

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

- src/kernel/[header.h](#)

## Chapter 4

# File Documentation

### 4.1 src/file\_system/FAT.h File Reference

The running part of the file system.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <stdint.h>
#include "file_system_constant.h"
```

#### Functions

- int [create\\_FAT](#) ()  
*Create FAT in the beginning.*
- int [clear\\_FAT](#) ()  
*clear FAT*
- int [delete\\_FAT](#) ()  
*delete FAT*
- int [register\\_FAT\\_entry](#) (int block\_count)  
*Register file into FAT.*
- int [append\\_FAT\\_entry](#) (int end\_block, int block\_count)  
*Add file entry.*
- int [free\\_FAT\\_entry](#) (int start\_block)  
*Free FAT entries after delete some files.*
- int [read\\_next\\_FAT\\_entry](#) (int curr\_block)  
*read next FAT entry given with current FAT entry*
- int [read\\_FAT](#) (char \*FAT\_name, FILE \*file)
- int [save\\_FAT](#) (char \*FAT\_name, FILE \*file)  
*save FAT into a file*
- int [get\\_free\\_block](#) ()

*Get the first free block number.*

- void `list_FAT` ()

*Print the FAT and bitmap.*

- unsigned char \* `create_temp_FAT` ()
- unsigned char \* `create_temp_map` ()

*Create a temp bitmap.*

### 4.1.1 Detailed Description

The running part of the file system.

[Define](#) some global variables and functions needed.

### 4.1.2 Function Documentation

#### 4.1.2.1 `append_FAT_entry()`

```
int append_FAT_entry (
    int end_block,
    int block_count )
```

Add file entry.

##### Parameters

in	<code>end_block</code>	from end block to append FAT entry
in	<code>block_count</code>	total block need to be append

##### Returns

The end block number of the new entry

#### 4.1.2.2 `clear_FAT()`

```
int clear_FAT ( )
```

clear FAT

##### Returns

int

#### 4.1.2.3 create\_FAT()

```
int create_FAT ( )
```

Create FAT in the beginning.

##### Returns

Status of the function, SUCCESS or FAILURE

#### 4.1.2.4 create\_temp\_map()

```
unsigned char* create_temp_map ( )
```

Create a temp bitmap.

##### Returns

the temp bitmap

#### 4.1.2.5 delete\_FAT()

```
int delete_FAT ( )
```

delete FAT

##### Returns

int

#### 4.1.2.6 free\_FAT\_entry()

```
int free_FAT_entry (
    int start_block )
```

Free FAT entries after delete some files.

##### Parameters

in	<i>start_block</i>	free from start block
----	--------------------	-----------------------

**Returns**

Status of the function, SUCCESS or FAILURE

**4.1.2.7 get\_free\_block()**

```
int get_free_block ( )
```

Get the first free block number.

**Returns**

The free block number or FAILURE

**4.1.2.8 list\_FAT()**

```
void list_FAT ( )
```

Print the FAT and bitmap.

**Returns**

Status of the function, SUCCESS or FAILURE

**4.1.2.9 read\_next\_FAT\_entry()**

```
int read_next_FAT_entry (
    int curr_block )
```

read next FAT entry given with current FAT entry

**Parameters**

in	<i>curr_block</i>	given current block to find next FAT entry
----	-------------------	--

**Returns**

The next FAT entry



#### 4.1.2.10 register\_FAT\_entry()

```
int register_FAT_entry (
    int block_count )
```

Register file into FAT.

##### Parameters

in	<i>block_count</i>	total block count to register in FAT
----	--------------------	--------------------------------------

##### Returns

the first block number of file registered in FAT

#### 4.1.2.11 save\_FAT()

```
int save_FAT (
    char * FAT_name,
    FILE * file )
```

save FAT into a file

##### Parameters

in	<i>FAT_name</i>	get FAT name to save
in	<i>file</i>	save into the given file

##### Returns

int

## 4.2 src/file\_system/FAT\_dir.h File Reference

[Define](#) the directory node structure needed and functions needed.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <stdint.h>
#include "file_system_constant.h"
#include "FAT.h"
#include "FAT_file.h"
```

## Data Structures

- struct [DIR\\_NODE](#)
- struct [PARSE\\_RESULT](#)

## Typedefs

- typedef struct [DIR\\_NODE](#) **DIR\_NODE**
- typedef struct [PARSE\\_RESULT](#) **PARSE\_RESULT**

## Functions

- [PARSE\\_RESULT](#) **path\_parser** (char \*path)  
*parser a path*
- int [init\\_top\\_dir](#) ()  
*initilaze the top level dir for multi-level dir file system*
- int [create\\_dir](#) (char \*dir\_name)  
*Create a dir object with given name.*
- int [visit\\_dir](#) (char \*dir\_name)  
*change path to a given directory*
- int [return\\_to\\_top\\_dir](#) ()  
*go back to the toppest dir*
- int [search\\_dir\\_by\\_path](#) (char \*dir\_path)  
*find directory by given path*
- int [free\\_all\\_dirs](#) ()  
*Free all the directory struct.*

## Variables

- struct [DIR\\_NODE](#) \* **top\_dir**
- struct [DIR\\_NODE](#) \* **dir\_pointer**

### 4.2.1 Detailed Description

[Define](#) the directory node structure needed and functions needed.

### 4.2.2 Function Documentation

#### 4.2.2.1 create\_dir()

```
int create_dir (
    char * dir_name )
```

Create a dir object with given name.

**Parameters**

in	<i>dir_name</i>	given directory name to create
----	-----------------	--------------------------------

**Returns**

int

**4.2.2.2 free\_all\_dirs()**

```
int free_all_dirs ( )
```

Free all the directory struct.

**Returns**

Status in integer type

**4.2.2.3 init\_top\_dir()**

```
int init_top_dir ( )
```

initilaze the top level dir for multi-level dir file system

**Returns**

int

**4.2.2.4 path\_parser()**

```
PARSE_RESULT path_parser (
    char * path )
```

parser a path

**Parameters**

in	<i>path</i>	given path to parser
----	-------------	----------------------

**Returns**

[PARSE\\_RESULT](#)

**4.2.2.5 return\_to\_top\_dir()**

```
int return_to_top_dir ( )
```

go back to the toppest dir

**Returns**

int

**4.2.2.6 search\_dir\_by\_path()**

```
int search_dir_by_path (
    char * dir_path )
```

find directory by given path

**Parameters**

in	<i>dir_path</i>	search by directory path
----	-----------------	--------------------------

**Returns**

int

**4.2.2.7 visit\_dir()**

```
int visit_dir (
    char * dir_name )
```

change path to a given directory

**Parameters**

in	<i>dir_name</i>	change to the given dir name path
----	-----------------	-----------------------------------

**Returns**

int

## 4.3 src/file\_system/FAT\_file.h File Reference

create file node for file system

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <stdint.h>
#include "file_system_constant.h"
#include "FAT.h"
#include "FAT_dir.h"
```

**Data Structures**

- struct [FILE\\_NODE](#)

**Typedefs**

- typedef struct [FILE\\_NODE](#) **FILE\_NODE**

**Functions**

- int **find\_fit\_size** (char \*content)
- int **find\_original\_size** (char \*content)
- int **find\_first\_eof** (char \*content)
- int **find\_block\_size** (char \*content, int max\_size)
- int **create\_file\_with\_size** (char \*file\_name, char \*input, int size)
- int **create\_file** (char \*file\_name, char \*input)
  - Create a file object.*
- int **search\_file\_by\_block** (unsigned char start\_block)
  - find file by given start block*
- int **search\_file\_by\_path** (char \*file\_path)
  - find file by file path*
- int **delete\_file\_by\_block** (unsigned char start\_block)
  - Delete file node by given start block.*
- int **delete\_file\_by\_name** (char \*file\_name)
  - delete file*
- int **delete\_file\_by\_path** (char \*file\_path)
  - delete file*
- char \* **read\_file\_content** (char \*file\_path)
  - read content in file path*

- char \* [print\\_file\\_content](#) (char \*file\_path)  
*print out content in file path*
- int [read\\_all\\_from\\_dump](#) (char \*FAT\_name)  
*read all context with FAT name from dump*
- int [free\\_all\\_files](#) ()  
*free memory*
- int [save\\_all\\_to\\_dump](#) (char \*FAT\_name)  
*save information to dump*
- void [list\\_all](#) ()  
*Print the information of all file nodes.*
- char \* [write\\_all](#) ()  
*write all directory information into a char\* for further write*

### 4.3.1 Detailed Description

create file node for file system

### 4.3.2 Function Documentation

#### 4.3.2.1 create\_file()

```
int create_file (
    char * file_name,
    char * input )
```

Create a file object.

##### Parameters

in	<i>file_name</i>	given file name for new file
in	<i>input</i>	content for the file

##### Returns

int

#### 4.3.2.2 delete\_file\_by\_block()

```
int delete_file_by_block (
    unsigned char start_block )
```

Delete file node by given start block.

**Parameters**

in	<i>start_block</i>	delete from start block
----	--------------------	-------------------------

**Returns**

Status of the function, SUCCESS or FAILURE

**4.3.2.3 delete\_file\_by\_name()**

```
int delete_file_by_name (  
    char * file_name )
```

delete file

**Parameters**

in	<i>file_name</i>	given file name
----	------------------	-----------------

**Returns**

int

**4.3.2.4 delete\_file\_by\_path()**

```
int delete_file_by_path (  
    char * file_path )
```

delete file

**Parameters**

in	<i>file_path</i>	delete the file path
----	------------------	----------------------

**Returns**

int SUCCESS or FAILURE

**4.3.2.5 free\_all\_files()**

```
int free_all_files ( )
```

free memory

**Returns**

int SUCCESS or FAILURE

**4.3.2.6 print\_file\_content()**

```
char* print_file_content (
    char * file_path )
```

print out content in file path

**Parameters**

in	<i>file_path</i>	read from the path
----	------------------	--------------------

**Returns**

char\* all context in char\* format

**4.3.2.7 read\_all\_from\_dump()**

```
int read_all_from_dump (
    char * FAT_name )
```

read all context with FAT name from dump

**Parameters**

in	<i>FAT_name</i>	read from given FAT name
----	-----------------	--------------------------

**Returns**

int

**4.3.2.8 read\_file\_content()**

```
char* read_file_content (
    char * file_path )
```

read content in file path



**Parameters**

in	<i>file_path</i>	given file path
----	------------------	-----------------

**Returns**

char\* content we want to read

**4.3.2.9 save\_all\_to\_dump()**

```
int save_all_to_dump (
    char * FAT_name )
```

save information to dump

**Parameters**

in	<i>FAT_name</i>	save the FAT name to dump
----	-----------------	---------------------------

**Returns**

int

**4.3.2.10 search\_file\_by\_block()**

```
int search_file_by_block (
    unsigned char start_block )
```

find file by given start block

**Parameters**

in	<i>start_block</i>	get file by the start block
----	--------------------	-----------------------------

**Returns**

int

**4.3.2.11 search\_file\_by\_path()**

```
int search_file_by_path (
    char * file_path )
```

find file by file path

## Parameters

in	<i>file_path</i>	find by the file path
----	------------------	-----------------------

## Returns

int

## 4.3.2.12 write\_all()

```
char* write_all ( )
```

write all directory information into a char\* for further write

## Returns

char\*

## 4.4 src/file\_system/file\_system\_constant.h File Reference

[Define](#) some common global variables that needed in the file system.

## Macros

- `#define MAX_FD_TABLE_SIZE 64`
- `#define MAX_QUEUE_SIZE 100`
- `#define SUCCESS 0`
- `#define FAILURE -1`
- `#define FREE 0`
- `#define OCCUPIED 1`
- `#define FAT_MODE 0`
- `#define INODE_MODE 1`
- `#define ALL_BLOCKS_OCCUPIED_ERROR "All blocks are occupied"`
- `#define UPPER_DIR_NOT_FOUND_ERROR "Upper directory is not found"`
- `#define CREATE_DIR_ERROR "Error in creating directory file"`
- `#define CREATE_FILE_ERROR "Error in creating file node"`
- `#define FILE_SYSTEM_NOT_EXIST_ERROR "Error in finding file system dump"`
- `#define READ_MAP_ERROR "Error in reading occupancy map from dump"`
- `#define SAVE_MAP_ERROR "Error in saving occupancy map"`
- `#define TOO_FEW_ARG_ERROR "Too few argument in input"`
- `#define CREATE_FAT_ERROR "Error in creating FAT"`
- `#define SAVE_FAT_ERROR "Error in saving FAT"`
- `#define READ_FAT_ERROR "Error in reading FAT from dump"`
- `#define MAX_INODE_ADDR_LENGTH 5`
- `#define MAX_INODE_NUM`
- `#define F_NOT_EXIST 0`
- `#define F_WRITE 1`

- `#define F_READ 2`
- `#define F_APPEND 3`
- `#define STD_IN 4`
- `#define STD_OUT 5`
- `#define F_SEEK_SET 1`
- `#define F_SEEK_CUR 2`
- `#define F_SEEK_END 3`
- `#define FILE_NOT_EXIST_ERROR "Error in finding the file from file system"`
- `#define FILE_DESCRIPTOR_CONFLICT_ERROR "Error in redefining the type of existed file descriptor"`
- `#define FILE_DESCRIPTOR_NOT_EXISTED "No file descriptor found"`
- `#define FILE_DESCRIPTOR_DOUBLE_WRITE_ERROR "Error in double writing into the same file"`
- `#define MAX_BLOCK_NUM 256`
- `#define MAX_BLOCK_SIZE 1024`
- `#define MAX_NAME_LENGTH 16`
- `#define MAX_PATH_LENGTH 64`
- `#define PENN_DISK_NAME "disk:"`
- `#define DIR_SLASH "\\"`
- `#define NULL_CHAR '$'`

#### 4.4.1 Detailed Description

[Define](#) some common global variables that needed in the file system.

### 4.5 src/file\_system/I\_node.h File Reference

[Define](#) some global variables and functions needed.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <stdint.h>
#include "file_system_constant.h"
```

#### Data Structures

- struct [I\\_NODE](#)

#### Functions

- int [create\\_inode\\_list](#) ()  
*Create Inode list.*
- int [delete\\_inode\\_list](#) ()  
*Delete Inode list.*

### 4.5.1 Detailed Description

[Define](#) some global variables and functions needed.

### 4.5.2 Function Documentation

#### 4.5.2.1 create\_inode\_list()

```
int create_inode_list ( )
```

Create Inode list.

##### Returns

Status of the function, SUCCESS or FAILURE

#### 4.5.2.2 delete\_inode\_list()

```
int delete_inode_list ( )
```

Delete Inode list.

##### Returns

Status of the function, SUCCESS or FAILURE

## 4.6 src/file\_system/interface.h File Reference

[Define](#) struct and functions for file descriptor and implement system calls API.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <stdint.h>
#include "FAT.h"
#include "FAT_dir.h"
#include "FAT_file.h"
#include "file_system_constant.h"
```

## Data Structures

- struct [FILE\\_DESCRIPTOR](#)

## Macros

- `#define max(a, b) (((a) > (b)) ? (a) : (b))`
- `#define min(a, b) (((a) < (b)) ? (a) : (b))`

## Typedefs

- typedef struct [FILE\\_DESCRIPTOR](#) **FILE\_DESCRIPTOR**

## Functions

- int [init\\_interface](#) (char \*fs\_name)  
*Initialize the file descriptor table.*
- int [save\\_interface](#) (char \*fs\_name)  
*save the interface*
- int [f\\_open](#) (const char \*fname, int mode)  
*Implement open file.*
- int [f\\_read](#) (int fd, int n, char \*buf)  
*read from a file descriptor*
- int [f\\_write](#) (int fd, const char \*str, int n)  
*write content to given file descriptor pointed file*
- int [f\\_close](#) (int fd)  
*Implement close the file.*
- int [f\\_unlink](#) (const char \*fname)  
*Implement unlink the file.*
- int [f\\_lseek](#) (int fd, int offset, int whence)  
*file start from offset*
- int [f\\_ls](#) ()  
*Implement file ls.*
- int [f\\_ls\\_redirect](#) (int fd)  
*ls with redirection*

### 4.6.1 Detailed Description

[Define](#) struct and functions for file descriptor and implement system calls API.

[Define](#) all the Macro needed, and some interface function to initialize.

### 4.6.2 Function Documentation

#### 4.6.2.1 [f\\_close\(\)](#)

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

Implement close the file.

**Parameters**

<i>in</i>	<i>fd</i>	close the given file descriptor
-----------	-----------	---------------------------------

**Returns**

Status of the function, SUCCESS or FAILURE

**4.6.2.2 f\_ls()**

```
int f_ls ( )
```

Implement file ls.

**Returns**

Status of the function, SUCCESS or FAILURE

**4.6.2.3 f\_ls\_redirect()**

```
int f_ls_redirect (
    int fd )
```

ls with redirection

**Parameters**

<i>in</i>	<i>fd</i>	redirect to given file descriptor
-----------	-----------	-----------------------------------

**Returns**

int

**4.6.2.4 f\_lseek()**

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

file start from offset

**Parameters**

in	<i>fd</i>	file descriptor point to file
in	<i>offset</i>	start from given offset
in	<i>whence</i>	from previous place

**Returns**

int

**4.6.2.5 f\_open()**

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

Implement open file.

**Parameters**

in	<i>fname</i>	open current file
in	<i>mode</i>	with read/write/append mode

**Returns**

The fd table or FAILURE

**4.6.2.6 f\_read()**

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

read from a file descriptor

**Parameters**

in	<i>fd</i>	given file descriptor to read from
in	<i>n</i>	would like to read n length
in	<i>buf</i>	read into buf



**Returns**

int read byte number

**4.6.2.7 f\_unlink()**

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

Implement unlink the file.

**Parameters**

in	<i>fname</i>	remove the given file name
----	--------------	----------------------------

**Returns**

Status of the function

**4.6.2.8 f\_write()**

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

write content to given file descriptor pointed file

**Parameters**

in	<i>fd</i>	file descriptor pointed to file
in	<i>str</i>	write str content to file
in	<i>n</i>	write the given length

**Returns**

int write bytes

**4.6.2.9 init\_interface()**

```
int init_interface (
    char * fs_name )
```

Initialize the file descirptor table.

**Parameters**

in	<i>fs_name</i>	given the first file system name
----	----------------	----------------------------------

**Returns**

Status of the function, SUCCESS or FAILURE

**4.6.2.10 save\_interface()**

```
int save_interface (
    char * fs_name )
```

save the interface

**Parameters**

in	<i>fs_name</i>	given file system name
----	----------------	------------------------

**Returns**

Status of the function, SUCCESS or FAILURE

**4.7 src/kernel/header.h File Reference**

Include all libraru and define struct needed to only store information.

```
#include <ucontext.h>
#include <sys/types.h>
#include <sys/time.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <poll.h>
#include <stdbool.h>
#include <string.h>
#include <valgrind/valgrind.h>
```

**Data Structures**

- struct [proc\\_info](#)
- struct [wait\\_info](#)

## Macros

- `#define STACKSIZE 4096`

### 4.7.1 Detailed Description

Include all library and define struct needed to only store information.

## 4.8 src/kernel/kernel.h File Reference

All the kernel functions that may only be called from the kernel side of the operating system.

```
#include "init.h"
#include "pcb.h"
#include "../file_system/interface.h"
```

## Functions

- `int k_process_create` (int priority, `pcb_t` \*parent, void \*function, char \*\*argument)  
*Create a child process from its parents.*
- `int k_process_kill` (`pcb_t` \*process, int signal)  
*Receive STOP/KILL/CONTINUE signals.*
- `int k_process_nice` (`pcb_t` \*process, int priority)  
*Change the priority of a process.*
- `void k_process_cleanup` (`pcb_t` \*pcb)  
*When a process is waited by its parent, cleanup the whole process.*
- `int k_sleep` (int ticks)  
*When the current running process would like to sleep.*
- `wait_info * k_wait` (int mode)  
*When the current running process would like to wait for a child to be stopped/terminated.*
- `wait_info * k_wait_pid` (int pid, int mode)  
*When the current running process would like to wait for a child with specific pid to be stopped/terminated.*
- `void k_unblock_parent` (`pcb_t` \*process)  
*When the child has stopped/terminated and need to tell its parent if the parent is waiting.*
- `void k_inherit_orphans` (`pcb_t` \*parent)  
*Function to let the calling process inherit all the orphan and zombie process of a process.*

### 4.8.1 Detailed Description

All the kernel functions that may only be called from the kernel side of the operating system.

### 4.8.2 Function Documentation

#### 4.8.2.1 k\_inherit\_orphans()

```
void k_inherit_orphans (
    pcb_t * parent )
```

Function to let the calling process inherit all the orphan and zombie process of a process.

## Parameters

in	<i>parent</i>	the target process whose children and zombie process would be inherited by the calling process
----	---------------	--

**4.8.2.2 k\_process\_cleanup()**

```
void k_process_cleanup (
    pcb_t * pcb )
```

When a process is waited by its parent, cleanup the whole process.

## Parameters

in	<i>process</i>	process you would like to cleanup
----	----------------	-----------------------------------

## Returns

SUCCESS or FAILURE

**4.8.2.3 k\_process\_create()**

```
int k_process_create (
    int priority,
    pcb_t * parent,
    void * function,
    char ** argument )
```

Create a child process from its parents.

## Parameters

in	<i>priority</i>	The process should set the priority.
in	<i>parent</i>	The process's parent process.
in	<i>function</i>	Which function will the process triggered.
in	<i>argument</i>	Function name as the first argument, then are the argument we would like to put in the triggered function

## Returns

Success or failure

#### 4.8.2.4 k\_process\_kill()

```
int k_process_kill (
    pcb_t * process,
    int signal )
```

Receive STOP/KILL/CONTINUE signals.

When it receive one of the signals, it will check if the signal is valid, then it will immediately execute.

##### Parameters

in	<i>process</i>	process you would like deliver the signal to
in	<i>signal</i>	S_SIGSTOP to stop, S_SIGCONT to continue stoppped process, S_SIGTERM to terminate the signal

##### Returns

SUCCESS or FAILURE

#### 4.8.2.5 k\_process\_nice()

```
int k_process_nice (
    pcb_t * process,
    int priority )
```

Change the priority of a process.

##### Parameters

in	<i>process</i>	process you would like to chenge the priority
in	<i>priority</i>	the priority level you would like to set

##### Returns

SUCCESS or FAILURE

#### 4.8.2.6 k\_sleep()

```
int k_sleep (
    int ticks )
```

When the current running process would like to sleep.

**Parameters**

in	<i>ticks</i>	number of ticks to sleep
----	--------------	--------------------------

**Returns**

SUCCESS or FAILURE

**4.8.2.7 k\_unblock\_parent()**

```
void k_unblock_parent (
    pcb_t * process )
```

When the child has stopped/terminated and need to tell its parent if the parent is waiting.

**Parameters**

in	<i>process</i>	process has been in a waitable status for its parent
----	----------------	--

**4.8.2.8 k\_wait()**

```
wait_info* k_wait (
    int mode )
```

When the current running process would like to wait for a child to be stopped/terminated.

**Parameters**

in	<i>mode</i>	block the process itself to wait or just keep running no matter whether wait on the child or not
----	-------------	--

**Returns**

status and pid information of waited child

**4.8.2.9 k\_wait\_pid()**

```
wait_info* k_wait_pid (
    int pid,
    int mode )
```

When the current running process would like to wait for a child with specific pid to be stopped/terminated.

## Parameters

in	<i>mode</i>	block the process itself to wait or just keep running no matter whether wait on the child or not
in	<i>pid</i>	the pid of the child process to be waited on

## Returns

status and pid information of waited child

## 4.9 src/kernel/node.h File Reference

Create linked list node, and double linked list structure.

```
#include "header.h"
```

### Data Structures

- struct [node](#)  
*Create a single node which could form linked list.*
- struct [linked\\_list](#)  
*Create a double linked list with dummy head and dummy tail.*

### Typedefs

- typedef struct [node](#) [NODE](#)  
*Create a single node which could form linked list.*
- typedef struct [linked\\_list](#) [LINKED\\_LIST](#)  
*Create a double linked list with dummy head and dummy tail.*

### Functions

- [LINKED\\_LIST](#) \* [init\\_lst](#) ()
- [NODE](#) \* [init\\_node](#) (int pid)
- void [insert\\_in\\_front](#) ([NODE](#) \*[node](#), [LINKED\\_LIST](#) \*[lst](#))
- void [push\\_back](#) ([NODE](#) \*[node](#), [LINKED\\_LIST](#) \*[lst](#))
- int [remove\\_node\\_from\\_list](#) ([NODE](#) \*[node](#), [LINKED\\_LIST](#) \*[lst](#))
- void [free\\_node](#) ([NODE](#) \*[node](#))
- void [print\\_node](#) ([NODE](#) \*[node](#))
- void [print\\_lst](#) ([LINKED\\_LIST](#) \*[lst](#))
- void [free\\_list](#) ([LINKED\\_LIST](#) \*[lst](#))
- [LINKED\\_LIST](#) \* [get\\_ready\\_q](#) (int priority)
- bool [check\\_node\\_in\\_lst](#) ([NODE](#) \*[node](#), [LINKED\\_LIST](#) \*[lst](#))

#### 4.9.1 Detailed Description

Create linked list node, and double linked list structure.

## 4.10 src/kernel/pcb.h File Reference

Create the process control block for each process to store necessary information.

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

### Data Structures

- struct [pcb](#)

### Typedefs

- typedef struct [pcb](#) [pcb\\_t](#)

### Functions

- [pcb\\_t](#) \* [create\\_pcb](#) (ucontext\_t \*context, int pid, int ppid, int prioity, char \*\*argument)  
*Create a pcb for a process/thread.*
- int [free\\_pcb](#) ([pcb\\_t](#) \*pcb)

#### 4.10.1 Detailed Description

Create the process control block for each process to store necessary information.

#### 4.10.2 Function Documentation

##### 4.10.2.1 create\_pcb()

```
pcb\_t* create_pcb (
    ucontext_t * context,
    int pid,
    int ppid,
    int prioity,
    char ** argument )
```

Create a pcb for a process/thread.

##### Parameters

in	<i>context</i>	a pointer points to the ucontext_t of the process context
in	<i>pid</i>	process id of the process
in	<i>ppid</i>	process id of the parent process
in	<i>priority</i>	priority of the process
in	<i>argument</i>	the first is the name of process aka the command, the second following argument will be the argument needed in functions



**Returns**

the pcb pointer of the process

## 4.11 src/kernel/scheduler.h File Reference

Scheduler for all threads to run in round robin policy.

```
#include "init.h"
```

**Functions**

- int **setup\_scheduler** ()
- void **setup\_signals** (void)
  - setup the SIGALRM signal handler*
- void **timer\_interrupt** (int j, siginfo\_t \*si, void \*old\_context)
  - Creates a new context to run the scheduler in, masks signals, then swaps contexts saving the previously executing thread and jumping to the scheduler.*
- void **scheduler** ()
  - Selects the next context to run, then starts running it.*
- void **mkcontext** (ucontext\_t \*uc, void \*function)
  - Create a context, initialize the context from the current context, setup the newstack, signal mask, and tell it which function to call.*
- int **pick\_thread** ()
  - Pick a ready queue for scheduling the next process.*
- void **context\_switch** ()
  - when need to context switch to another ready process*
- void **clean\_up\_terminated** ()
  - helper functions for process which has been terminated normally but haven't reach round robin time slot*
- void **init\_scheduler\_pcb** ()
  - Initialize two process, first one is scheduler used for context switch, second one is idle, used when there is no running thread.*
- void **add\_up\_time\_interval** ()
- void **block\_timer** ()
  - Stop the penn\_os timer temporarily.*
- void **unblock\_timer** ()
  - Resume the penn\_os timer.*
- int **remove\_node\_from\_ready\_queue** ()
  - Remove a process node from its ready queue.*
- int **add\_node\_to\_ready\_queue** (NODE \*node, int priority)
  - Add a process node to the corresponding ready queue.*
- int **change\_status** ()
  - Change the status of the process, and put the process into running queue with corresponding priority.*

### 4.11.1 Detailed Description

Scheduler for all threads to run in round robin policy.

## 4.11.2 Function Documentation

### 4.11.2.1 add\_node\_to\_ready\_queue()

```
int add_node_to_ready_queue (
    NODE * node,
    int priority )
```

Add a process node to the corresponding ready queue.

#### Parameters

in	<i>node</i>	the process node to be added
in	<i>priority</i>	the priority of the node

#### Returns

SUCCESS or FAILURE

### 4.11.2.2 change\_status()

```
int change_status ( )
```

Change the status of the process, and put the process into running queue with corresponding priority.

#### Returns

SUCCESS or FAILURE

### 4.11.2.3 mkcontext()

```
void mkcontext (
    ucontext_t * uc,
    void * function )
```

Create a context, initialize the context from the current context, setup the newstack, signal mask, and tell it which function to call.

#### Parameters

in	<i>uc</i>	current context
in	<i>function</i>	function to call

#### 4.11.2.4 pick\_thread()

```
int pick_thread ( )
```

Pick a ready queue for scheduling the next process.

##### Returns

SUCCESS or FAILURE

#### 4.11.2.5 remove\_node\_from\_ready\_queue()

```
int remove_node_from_ready_queue ( )
```

Remove a process node from its ready queue.

##### Returns

SUCCESS or FAILURE

#### 4.11.2.6 timer\_interrupt()

```
void timer_interrupt (
    int j,
    siginfo_t * si,
    void * old_context )
```

Creates a new context to run the scheduler in, masks signals, then swaps contexts saving the previously executing thread and jumping to the scheduler.

##### Parameters

in	<i>j</i>	
in	<i>si</i>	
in	<i>old_context</i>	

## 4.12 src/kernel/user.h File Reference

user level functions

```
#include "header.h"
```

## Functions

- int `p_spawn` (void \*func, char \*\*argv)  
*fork a new thread that retains most of the attributes of the parent thread*
- int `p_kill` (int pid, int signal)  
*kill the process referenced by pid with the signal signal*
- int `p_sleep` (int ticks)  
*set a process to be slept or blocked for a specific period of time, then resume the process*
- int `p_exit` ()  
*exit the current process unconditionally*
- int `p_nice` (int pid, int priority)  
*set the priority level of the thread*
- `proc_info` \* `p_info` (int pid)  
*return the structure representing standard information about the thread pid*
- `wait_info` \* `p_wait` (int mode)  
*set the calling process to be blocked*
- `wait_info` \* `p_wait_pid` (int pid, int mode)
- bool `W_WIFEXITED` (int status)  
*return true if the child terminated normally*
- bool `W_WIFSTOPPED` (int status)  
*return true if the child was stopped by a signal*
- bool `W_WIFCONTINUED` (int status)  
*return true if the child was continued by the signal*
- bool `W_WIFSIGNALED` (int status)  
*return true if the child terminated by p\_kill with the S\_SIGTERM signal*
- int `print_proc_info` (`proc_info` \*pinfo)  
*helper function to print out the content of proc\_info structure*
- void `p_ps` ()  
*helper function to print out the process info of all threads in the ready queues*
- int `get_fd` (int mode, int pid)
- void `set_fd` (int mode, int pid, int fd)

### 4.12.1 Detailed Description

user level functions

### 4.12.2 Function Documentation

#### 4.12.2.1 p\_exit()

```
int p_exit ( )
```

exit the current process unconditionally

#### Returns

success or failure

#### 4.12.2.2 p\_info()

```
proc_info* p_info (
    int pid )
```

return the structure representing standard information about the thread pid

##### Parameters

in	<i>pid</i>	the pid of the process whose information we are about to return
----	------------	---

##### Returns

the information structure of the process, including its status, pid, command and priority level

#### 4.12.2.3 p\_kill()

```
int p_kill (
    int pid,
    int signal )
```

kill the process referenced by pid with the signal signal

##### See also

[k\\_process\\_kill\(\)](#)

##### Parameters

in	<i>pid</i>	the pid of the process to be killed
in	<i>signal</i>	the signal to be dealt with when killing the process

##### Returns

success or failure

#### 4.12.2.4 p\_nice()

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

set the priority level of the thread

**Parameters**

in	<i>pid</i>	the pid of the process we are dealing with
in	<i>priority</i>	the new priority level of the process

**Returns**

success or failure

**4.12.2.5 p\_sleep()**

```
int p_sleep (
    int ticks )
```

set a process to be slept or blocked for a specific period of time, then resume the process

**See also**

[k\\_sleep\(\)](#)

**Parameters**

in	<i>ticks</i>	the amount of time the process to sleep
----	--------------	---

**Returns**

success or failure

**4.12.2.6 p\_spawn()**

```
int p_spawn (
    void * func,
    char ** argv )
```

fork a new thread that retains most of the attributes of the parent thread

**See also**

[k\\_process\\_create\(\)](#)

**Parameters**

in	<i>func</i>	the function to be executed in the process created
in	<i>argv</i>	the argument of the process, including the name and job

**Returns**

pid of the new process

**4.12.2.7 p\_wait()**

```
wait_info* p_wait (
    int mode )
```

set the calling process to be blocked

**Parameters**

in	mode	NOHANG condition
----	------	------------------

**Returns**

[wait\\_info](#) structure

**4.12.2.8 print\_proc\_info()**

```
int print_proc_info (
    proc_info * pinfo )
```

helper function to print out the content of [proc\\_info](#) structure

**Parameters**

in	pinfo	the process info structure we are about to print
----	-------	--

**Returns**

success or failure

**4.12.2.9 W\_WIFCONTINUED()**

```
bool W_WIFCONTINUED (
    int status )
```

return true if the child was continued by the signal

**Parameters**

<i>in</i>	<i>status</i>	the status returned by p_wait
-----------	---------------	-------------------------------

**Returns**

true or false

**4.12.2.10 W\_WIFEXITED()**

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

return true if the child terminated normally

**Parameters**

<i>in</i>	<i>status</i>	the status returned by p_wait
-----------	---------------	-------------------------------

**Returns**

true or false

**4.12.2.11 W\_WIFSIGNALED()**

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

return true if the child terminated by p\_kill with the S\_SIGTERM signal

**Parameters**

<i>in</i>	<i>status</i>	the status returned by p_wait
-----------	---------------	-------------------------------

**Returns**

true or false

**4.12.2.12 W\_WIFSTOPPED()**

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



return true if the child was stopped by a signal

**Parameters**

in	status	the status returned by p_wait
----	--------	-------------------------------

**Returns**

true or false

**4.13 src/shell/job.h File Reference**

Each command is a job, and they will be in a single linked list.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
#include <limits.h>
#include <signal.h>
#include <stdbool.h>
#include <fcntl.h>
#include "tokenizer.h"
#include "shell.h"
#include "../file_system/interface.h"
```

**Data Structures**

- struct [job](#)

**Typedefs**

- typedef struct [job](#) **JOB**

**Functions**

- **JOB \* create** (bool bg\_node, char \*command, int label, int status, int num\_process)  
*function to create a new job*
- **JOB \* insertFirst** (JOB \*new\_node, JOB \*prev)  
*function to insert a new job to the front of another job*
- **JOB \* deleteFirst** (JOB \*node)  
*function to delete the first job node link*
- **JOB \* delete\_by\_label** (int label, JOB \*node)  
*delete the node with the specific label*
- void **printInfo** (JOB \*node)  
*print the information of the node*
- void **printList** (JOB \*node)  
*print the whole job list, started by head*

- `JOB * find_by_label` (int label, `JOB *ptr`)  
*find the job node by its label, return and change the linkedlist*
- `JOB * find_by_pid` (int pid, `JOB *ptr`)  
*find the job node by its pid*
- int `getSize` (`JOB *ptr`)  
*helper function to get the size of the job list*
- void `printJobs` (`JOB *ptr`)  
*print out the label, commad and status of each job in the list*
- `JOB * find_recent_stop` (`JOB *ptr`)  
*find the most recently stopped job from the list*
- void `free_job` (`JOB *ptr`)  
*free a job, termimate its running process*
- `JOB * find_by_label_stop` (int label, `JOB *ptr`)  
*find the most recently stopped job node by its label*
- `JOB * remove_label` (`JOB *ptr`, int label)  
*function to remove a job node by label*

## Variables

- int \* `unused_identifier`
- `JOB * head`

### 4.13.1 Detailed Description

Each command is a job, and they will be in a single linked list.

### 4.13.2 Function Documentation

#### 4.13.2.1 create()

```
JOB* create (
    bool bg_node,
    char * command,
    int label,
    int status,
    int num_process )
```

function to create a new job

#### Parameters

in	<code>bg_node</code>	boolean value indicateing if the job is in background or not
in	<code>command</code>	the command the job is executing
in	<code>label</code>	label for the information of the job
in	<code>status</code>	status of the job to shell
in	<code>num_process</code>	number fo processes that are runnning concurrently

**Returns**

the job structure created

**4.13.2.2 delete\_by\_label()**

```
JOB* delete_by_label (
    int label,
    JOB * node )
```

delete the node with the specific label

**Parameters**

in	<i>label</i>	the label of the node to be deleted
in	<i>node</i>	the dummy node to begin the search

**Returns**

the deleted the node, or NULL if nothing has been done

**4.13.2.3 deleteFirst()**

```
JOB* deleteFirst (
    JOB * node )
```

function to delete the first job node link

**Parameters**

in	<i>node</i>	the node to be deleted
----	-------------	------------------------

**Returns**

the deleted link

**4.13.2.4 find\_by\_label()**

```
JOB* find_by_label (
    int label,
    JOB * ptr )
```

find the job node by its label, return and change the linkedlist

**Parameters**

in	<i>label</i>	the label of the job node to be searched
in	<i>ptr</i>	the node pointer used for iterating the job list

**Returns**

the node of the label, or NULL if not found

**4.13.2.5 find\_by\_label\_stop()**

```
JOB* find_by_label_stop (  
    int label,  
    JOB * ptr )
```

find the most recently stopped job node by its label

**Parameters**

in	<i>label</i>	the label of the job node to be searched
in	<i>ptr</i>	the node pointer used for iterating the job list

**Returns**

the node of the label, with its status to be stopped, or NULL if no node is matched

**4.13.2.6 find\_by\_pid()**

```
JOB* find_by_pid (  
    int pid,  
    JOB * ptr )
```

find the job node by its pid

**Parameters**

in	<i>pid</i>	the pid of the job node to be searched
in	<i>ptr</i>	the node pointer used for iterating the job list

**Returns**

the node with the exact pid, or NULL if not found

#### 4.13.2.7 find\_recent\_stop()

```
JOB* find_recent_stop (
    JOB * ptr )
```

find the most recently stopped job from the list

##### Parameters

in	<i>ptr</i>	the node pointer used for iterating the job list
----	------------	--

##### Returns

the node that has been stopped most recently

#### 4.13.2.8 free\_job()

```
void free_job (
    JOB * ptr )
```

free a job, terminate its running process

##### Parameters

in	<i>ptr</i>	the job to be freed
----	------------	---------------------

#### 4.13.2.9 getSize()

```
int getSize (
    JOB * ptr )
```

helper function to get the size of the job list

##### Parameters

in	<i>ptr</i>	the node pointer used for iterating the job list
----	------------	--

##### Returns

the size of the job list

#### 4.13.2.10 insertFirst()

```
JOB* insertFirst (
    JOB * new_node,
    JOB * prev )
```

function to insert a new job to the front of another job

##### Parameters

in	<i>new_node</i>	the job about to be inserted to the front of job node prev
in	<i>prev</i>	the to be next job node of new_node

##### Returns

the new job node after the insersion

#### 4.13.2.11 printInfo()

```
void printInfo (
    JOB * node )
```

print the information of the node

##### Parameters

in	<i>node</i>	the node whose information is about to be printed
----	-------------	---

#### 4.13.2.12 printJobs()

```
void printJobs (
    JOB * ptr )
```

print out the label, commad and status of each job in the list

##### Parameters

in	<i>ptr</i>	the node pointer used for iterating the job list
----	------------	--

#### 4.13.2.13 printList()

```
void printList (
```

```
JOB * node )
```

print the whole job list, started by head

#### Parameters

in	<i>node</i>	the head node of the list we begin to iterate and print
----	-------------	---

#### 4.13.2.14 remove\_label()

```
JOB* remove_label (
    JOB * ptr,
    int label )
```

function to remove a job node by label

#### Parameters

in	<i>ptr</i>	the node pointer used for iterating the job list
in	<i>label</i>	the label of the job node to be searched and deleted

#### Returns

the head or the new head of the job list after deletion, or NULL if nothing has been done

## 4.14 src/shell/shell.h File Reference

Some helper functions and extern variable declared here.

```
#include "../file_system/interface.h"
```

### Macros

- #define **BUFFER\_SIZE** 4096

### Functions

- void [signal\\_handler](#) (int signal)  
*handle signals*
- void [io\\_loop](#) ()  
*i/o loop to execute keyboard commnads*
- int [get\\_minimal\\_id](#) (int \*\*unused\_identifier)  
*Get the minimal id object.*



## Variables

- int **shell\_in**
- int **shell\_out**
- char **print\_msg** [BUFFER\_SIZE]

### 4.14.1 Detailed Description

Some helper functions and extern variable declared here.

Shell file to work like a shell.

### 4.14.2 Function Documentation

#### 4.14.2.1 get\_minimal\_id()

```
int get_minimal_id (
    int ** unused_identifier )
```

Get the minimal id object.

##### Parameters

in	<i>unused_identifier</i>	
----	--------------------------	--

##### Returns

int

#### 4.14.2.2 signal\_handler()

```
void signal_handler (
    int signal )
```

handle signals

##### Parameters

in	<i>signal</i>	to deal with ^C and ^Z
----	---------------	------------------------

### 4.14.3 Variable Documentation

#### 4.14.3.1 print\_msg

```
char print_msg[BUFFER_SIZE]
```

print message buffer

## 4.15 src/shell/user\_functions.h File Reference

All the command will call corresponding functions.

```
#include "../kernel/init.h"
#include "../kernel/user.h"
#include "../file_system/FAT.h"
#include "../file_system/FAT_dir.h"
#include "../file_system/FAT_file.h"
#include "../file_system/file_system_constant.h"
#include "../file_system/interface.h"
#include "shell.h"
```

### Functions

- void [match\\_function](#) (char \*\*arguments)  
*match command and real functions to call*
- void [user\\_man](#) ()  
*print out all the valid command*
- void [auto\\_complete](#) (char \*string)  
*Search the string in constructed trie.*
- void [init\\_auto\\_complete](#) ()  
*build the trie and insert all user level command to the trie for auto-completion functionality*
- void [user\\_cd](#) (char \*\*arguments, int argc)  
*change directory*

### 4.15.1 Detailed Description

All the command will call corresponding functions.

### 4.15.2 Function Documentation

#### 4.15.2.1 auto\_complete()

```
void auto_complete (  
    char * string )
```

Search the string in constructed trie.

## Parameters

in	<i>string</i>	to be searched in the trie for auto-completion
----	---------------	--

### 4.15.2.2 match\_function()

```
void match_function (
    char ** arguments )
```

match command and real functions to call

## Parameters

in	<i>arguments</i>	function name in the 0 position, others are arguments for the function
----	------------------	--

### 4.15.2.3 user\_cd()

```
void user_cd (
    char ** arguments,
    int argc )
```

change directory

## Parameters

in	<i>arguments</i>	parameters of functions
in	<i>argc</i>	argument number for the function cd.



# Index

add\_node\_to\_ready\_queue  
    scheduler.h, [42](#)  
append\_FAT\_entry  
    FAT.h, [14](#)  
auto\_complete  
    user\_functions.h, [58](#)

change\_status  
    scheduler.h, [42](#)  
clear\_FAT  
    FAT.h, [14](#)  
context  
    pcb, [10](#)  
create  
    job.h, [51](#)  
create\_dir  
    FAT\_dir.h, [18](#)  
create\_FAT  
    FAT.h, [14](#)  
create\_file  
    FAT\_file.h, [22](#)  
create\_inode\_list  
    I\_node.h, [29](#)  
create\_pcb  
    pcb.h, [40](#)  
create\_temp\_map  
    FAT.h, [15](#)

Define, [5](#)  
delete\_by\_label  
    job.h, [52](#)  
delete\_FAT  
    FAT.h, [15](#)  
delete\_file\_by\_block  
    FAT\_file.h, [22](#)  
delete\_file\_by\_name  
    FAT\_file.h, [23](#)  
delete\_file\_by\_path  
    FAT\_file.h, [23](#)  
delete\_inode\_list  
    I\_node.h, [29](#)  
deleteFirst  
    job.h, [52](#)  
DIR\_NODE, [5](#)

f\_close  
    interface.h, [30](#)  
f\_ls  
    interface.h, [31](#)  
f\_ls\_redirect

    interface.h, [31](#)  
f\_lseek  
    interface.h, [31](#)  
f\_open  
    interface.h, [32](#)  
f\_read  
    interface.h, [32](#)  
f\_unlink  
    interface.h, [33](#)  
f\_write  
    interface.h, [33](#)  
FAT.h  
    append\_FAT\_entry, [14](#)  
    clear\_FAT, [14](#)  
    create\_FAT, [14](#)  
    create\_temp\_map, [15](#)  
    delete\_FAT, [15](#)  
    free\_FAT\_entry, [15](#)  
    get\_free\_block, [16](#)  
    list\_FAT, [16](#)  
    read\_next\_FAT\_entry, [16](#)  
    register\_FAT\_entry, [16](#)  
    save\_FAT, [17](#)  
FAT\_dir.h  
    create\_dir, [18](#)  
    free\_all\_dirs, [19](#)  
    init\_top\_dir, [19](#)  
    path\_parser, [19](#)  
    return\_to\_top\_dir, [20](#)  
    search\_dir\_by\_path, [20](#)  
    visit\_dir, [20](#)  
FAT\_file.h  
    create\_file, [22](#)  
    delete\_file\_by\_block, [22](#)  
    delete\_file\_by\_name, [23](#)  
    delete\_file\_by\_path, [23](#)  
    free\_all\_files, [23](#)  
    print\_file\_content, [24](#)  
    read\_all\_from\_dump, [24](#)  
    read\_file\_content, [24](#)  
    save\_all\_to\_dump, [25](#)  
    search\_file\_by\_block, [25](#)  
    search\_file\_by\_path, [25](#)  
    write\_all, [27](#)  
File, [6](#)  
FILE\_DESCRIPTOR, [6](#)  
FILE\_NODE, [6](#)  
find\_by\_label  
    job.h, [52](#)

- find\_by\_label\_stop
  - job.h, [53](#)
- find\_by\_pid
  - job.h, [53](#)
- find\_recent\_stop
  - job.h, [53](#)
- free\_all\_dirs
  - FAT\_dir.h, [19](#)
- free\_all\_files
  - FAT\_file.h, [23](#)
- free\_FAT\_entry
  - FAT.h, [15](#)
- free\_job
  - job.h, [54](#)
- get\_free\_block
  - FAT.h, [16](#)
- get\_minimal\_id
  - shell.h, [57](#)
- getSize
  - job.h, [54](#)
- I\_NODE, [7](#)
- I\_Node, [7](#)
- I\_node.h
  - create\_inode\_list, [29](#)
  - delete\_inode\_list, [29](#)
- init\_interface
  - interface.h, [33](#)
- init\_top\_dir
  - FAT\_dir.h, [19](#)
- insertFirst
  - job.h, [54](#)
- interface.h
  - f\_close, [30](#)
  - f\_ls, [31](#)
  - f\_ls\_redirect, [31](#)
  - f\_lseek, [31](#)
  - f\_open, [32](#)
  - f\_read, [32](#)
  - f\_unlink, [33](#)
  - f\_write, [33](#)
  - init\_interface, [33](#)
  - save\_interface, [34](#)
- job, [7](#)
- job.h
  - create, [51](#)
  - delete\_by\_label, [52](#)
  - deleteFirst, [52](#)
  - find\_by\_label, [52](#)
  - find\_by\_label\_stop, [53](#)
  - find\_by\_pid, [53](#)
  - find\_recent\_stop, [53](#)
  - free\_job, [54](#)
  - getSize, [54](#)
  - insertFirst, [54](#)
  - printInfo, [55](#)
  - printJobs, [55](#)
  - printList, [55](#)
  - remove\_label, [56](#)
- k\_inherit\_orphans
  - kernel.h, [35](#)
- k\_process\_cleanup
  - kernel.h, [36](#)
- k\_process\_create
  - kernel.h, [36](#)
- k\_process\_kill
  - kernel.h, [36](#)
- k\_process\_nice
  - kernel.h, [37](#)
- k\_sleep
  - kernel.h, [37](#)
- k\_unblock\_parent
  - kernel.h, [38](#)
- k\_wait
  - kernel.h, [38](#)
- k\_wait\_pid
  - kernel.h, [38](#)
- kernel.h
  - k\_inherit\_orphans, [35](#)
  - k\_process\_cleanup, [36](#)
  - k\_process\_create, [36](#)
  - k\_process\_kill, [36](#)
  - k\_process\_nice, [37](#)
  - k\_sleep, [37](#)
  - k\_unblock\_parent, [38](#)
  - k\_wait, [38](#)
  - k\_wait\_pid, [38](#)
- linked\_list, [8](#)
- list\_FAT
  - FAT.h, [16](#)
- match\_function
  - user\_functions.h, [59](#)
- mkcontext
  - scheduler.h, [42](#)
- Navigate, [8](#)
- node, [8](#)
- p\_exit
  - user.h, [44](#)
- p\_info
  - user.h, [44](#)
- p\_kill
  - user.h, [45](#)
- p\_nice
  - user.h, [45](#)
- p\_sleep
  - user.h, [46](#)
- p\_spawn
  - user.h, [46](#)
- p\_wait
  - user.h, [47](#)
- PARSE\_RESULT, [9](#)

- path\_parser
  - FAT\_dir.h, 19
- pcb, 9
  - context, 10
  - pq\_node, 10
  - zombie\_children, 10
- pcb.h
  - create\_pcb, 40
- pick\_thread
  - scheduler.h, 43
- pq\_node
  - pcb, 10
- print\_file\_content
  - FAT\_file.h, 24
- print\_msg
  - shell.h, 58
- print\_proc\_info
  - user.h, 47
- printInfo
  - job.h, 55
- printJobs
  - job.h, 55
- printList
  - job.h, 55
- proc\_info, 10
- read\_all\_from\_dump
  - FAT\_file.h, 24
- read\_file\_content
  - FAT\_file.h, 24
- read\_next\_FAT\_entry
  - FAT.h, 16
- register\_FAT\_entry
  - FAT.h, 16
- remove\_label
  - job.h, 56
- remove\_node\_from\_ready\_queue
  - scheduler.h, 43
- return\_to\_top\_dir
  - FAT\_dir.h, 20
- save\_all\_to\_dump
  - FAT\_file.h, 25
- save\_FAT
  - FAT.h, 17
- save\_interface
  - interface.h, 34
- scheduler.h
  - add\_node\_to\_ready\_queue, 42
  - change\_status, 42
  - mkcontext, 42
  - pick\_thread, 43
  - remove\_node\_from\_ready\_queue, 43
  - timer\_interrupt, 43
- search\_dir\_by\_path
  - FAT\_dir.h, 20
- search\_file\_by\_block
  - FAT\_file.h, 25
- search\_file\_by\_path
  - FAT\_file.h, 25
- shell.h
  - get\_minimal\_id, 57
  - print\_msg, 58
  - signal\_handler, 57
- signal\_handler
  - shell.h, 57
- src/file\_system/FAT.h, 13
- src/file\_system/FAT\_dir.h, 17
- src/file\_system/FAT\_file.h, 21
- src/file\_system/file\_system\_constant.h, 27
- src/file\_system/l\_node.h, 28
- src/file\_system/interface.h, 29
- src/kernel/header.h, 34
- src/kernel/kernel.h, 35
- src/kernel/node.h, 39
- src/kernel/pcb.h, 40
- src/kernel/scheduler.h, 41
- src/kernel/user.h, 43
- src/shell/job.h, 50
- src/shell/shell.h, 56
- src/shell/user\_functions.h, 58
- Struct, 11
- timer\_interrupt
  - scheduler.h, 43
- tokenizer, 11
- Tree, 11
- trie\_node, 11
- user.h
  - p\_exit, 44
  - p\_info, 44
  - p\_kill, 45
  - p\_nice, 45
  - p\_sleep, 46
  - p\_spawn, 46
  - p\_wait, 47
  - print\_proc\_info, 47
  - W\_WIFCONTINUED, 47
  - W\_WIFEXITED, 48
  - W\_WIFSIGNALED, 48
  - W\_WIFSTOPPED, 48
- user\_cd
  - user\_functions.h, 59
- user\_functions.h
  - auto\_complete, 58
  - match\_function, 59
  - user\_cd, 59
- visit\_dir
  - FAT\_dir.h, 20
- W\_WIFCONTINUED
  - user.h, 47
- W\_WIFEXITED
  - user.h, 48
- W\_WIFSIGNALED
  - user.h, 48

W\_WIFSTOPPED

user.h, [48](#)

wait\_info, [12](#)

write\_all

FAT\_file.h, [27](#)

zombie\_children

pcb, [10](#)