pennOS

Generated by Doxygen 1.8.17

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

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_DESCIRPTOR 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 distingushing 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	*end_block*	from end block to append FAT entry
in	*block_count*	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	*start_block*	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	*curr_block*	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	*block_count*	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	*FAT_name*	get FAT name to save
in	*file*	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 ()

  initiliaze 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	*dir_name*	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 ( )
```

initiliaze 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	*path*	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	*dir_path*	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	*dir_name*	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	*file_name*	given file name for new file
in	*input*	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	*start_block*	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	*file_name*	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	*file_path*	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	*file_path*	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	*FAT_name*	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	*file_path*	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	*FAT_name*	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	*start_block*	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	*file_path*	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 descripter 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_DESCIRPTOR

Macros

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

Typedefs

- typedef struct FILE_DESCIRPTOR **FILE_DESCIRPTOR**

Functions

- int init_interface (char *fs_name)
  Initialize the file descirptor 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 descripter 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

in	*fd*	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

in	*fd*	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	*fd*	file descriptor point to file
in	*offset*	start from given offset
in	*whence*	from previous place

Returns

int

4.6.2.5 f_open()

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

Implement open file.

Parameters

in	*fname*	open current file
in	*mode*	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	*fd*	given file desciptor to read from
in	*n*	would like to read n length
in	*buf*	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	*fname*	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	*fd*	file descriptor pointed to file
in	*str*	write str content to file
in	*n*	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	*fs_name*	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	*fs_name*	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 libraru 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/teminated.
- 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/teminated.
- 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	*parent*	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	*process*	process you would like to cleanup

Returns

SUCESSS 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	*priority*	The process should set the priority.
in	*parent*	The process's parent process.
in	*function*	Which function will the process triggered.
in	*argument*	Function name as the first argument, then are the aregument 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	*process*	process you would like deliver the signal to
in	*signal*	S_SIGSTOP to stop, S_SIGCONT to continue stoppped process, S_SIGTERM to terminate the signal

Returns

SUCESSS 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	*process*	process you would like to chenge the priority
in	*priority*	the priority level you would like to set

Returns

SUCESSS or FAILURE

4.8.2.6 k_sleep()

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

When the current running process would like to sleep.

Parameters

in	*ticks*	number of ticks to sleep

Returns

SUCESSS 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	*process*	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/teminated.

Parameters

in	*mode*	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/teminated.

Parameters

in	*mode*	block the process itself to wait or just keep running no matter whether wait on the child or not
in	*pid*	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	*context*	a pointer points to the ucontext_t of the process context
in	*pid*	process id of the process
in	*ppid*	process id of the parent process
in	*priority*	priority of the process
in	*argument*	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	*node*	the process node to be added
in	*priority*	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	*uc*	current context
in	*function*	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	*j*	
in	*si*	
in	*old_context*	

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 sleeped 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 siganl
- int print_proc_info (proc_info *pinfo)

  helper function to print out the content of proc_info structure
- void p_ps ()

  helper funtion 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	*pid*	the pid of the process whose infomation 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	*pid*	the pid of the process to be killed
in	*signal*	the signal to be dealed 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	*pid*	the pid of the process we are dealing with
in	*priority*	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 sleeped or blocked for a specific period of time, then resume the process

See also

k_sleep()

Parameters

in	*ticks*	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	*func*	the function to be executed in the process created
in	*argv*	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

in	*status*	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

in	*status*	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 siganl

Parameters

in	*status*	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)

  funtion 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 infomation 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	*bg_node*	boolean value indicateing if the job is in background or not
in	*command*	the command the job is executing
in	*label*	label for the information of the job
in	*status*	status of the job to shell
in	*num_process*	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	*label*	the label of the node to be deleted
in	*node*	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 )
```

funtion to delete the first job node link

Parameters

in	*node*	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	*label*	the label of the job node to be searched
in	*ptr*	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	*label*	the label of the job node to be searched
in	*ptr*	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	*pid*	the pid of the job node to be searched
in	*ptr*	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	*ptr*	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, termimate its running process

Parameters

in	*ptr*	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	*ptr*	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	*new_node*	the job about to be inserted to the front of job node prev
in	*prev*	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 infomation of the node

Parameters

in	*node*	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	*ptr*	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	*node*	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	*ptr*	the node pointer used for iterating the job list
in	*label*	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	*unused_identifier*	

Returns

int

4.14.2.2 signal_handler()

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

handle signals

Parameters

in	*signal*	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	*string*	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	*arguments*	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	*arguments*	parameters of functions
in	*argc*	argument number for the function cd.

Index