PennOS

Generated by Doxygen 1.9.6

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 files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 bgNode Struct Reference

Data Fields

- int self_pid
- int self_pgid
- int jid
- int state
- int groupsize
- char ∗ cm
- struct bgNode ∗ next

3.1.1 Field Documentation

3.1.1.1 cm

```
char* cm
```

3.1.1.2 groupsize

```
int groupsize
```

3.1.1.3 jid

```
int jid
```

3.1.1.4 next

```
struct bgNode* next
```

3.1.1.5 self_pgid

```
int self_pgid
```

3.1.1.6 self_pid

```
int self_pid
```

3.1.1.7 state

```
int state
```

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

- PennFAT.c

3.2 j_queue Struct Reference

```
#include <j_queue.h>
```

Data Fields

- struct Job ∗ head
- struct Job ∗ tail
- int size

3.2.1 Field Documentation

3.2.1.1 head

```
struct Job* head
```

3.2.1.2 size

```
int size
```

3.2.1.3 tail

```
struct Job* tail
```

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

- j_queue.h

3.3 Job Struct Reference

```
#include <Job.h>
```

Data Fields

- struct parsed_command * cmd
- char * rawCmd
- int group_id
- struct Job * next
- struct Job * prev
- int job_id
- enum jobStatus status
- enum jobGround ground
- int count_finished

3.3.1 Field Documentation

3.3.1.1 cmd

```
struct parsed_command* cmd
```

3.3.1.2 count_finished

```
int count_finished
```

3.3.1.3 ground

```
enum jobGround ground
```

3.3.1.4 group_id

```
int group_id
```

3.3.1.5 job_id

```
int job_id
```

3.3.1.6 next

```
struct Job* next
```

3.3.1.7 prev

```
struct Job* prev
```

3.3.1.8 rawCmd

```
char* rawCmd
```

3.3.1.9 status

```
enum jobStatus status
```

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

- Job.h

3.4 jobQueue Struct Reference

Data Fields

- struct bgNode ∗ front
- struct bgNode ∗ rear
- int q_size

3.4.1 Field Documentation

3.4.1.1 front

```
struct bgNode* front
```

3.4.1.2 q_size

```
int q_size
```

3.4.1.3 rear

```
struct bgNode * rear
```

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

- PennFAT.c

3.5 parsed_command Struct Reference

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

Data Fields

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

3.5.1 Detailed Description

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

3.5.2 Field Documentation

3.5.2.1 commands

```
char** commands[ ]
```

3.5.2.2 is_background

```
bool is_background
```

3.5.2.3 is_file_append

```
bool is_file_append
```

3.5.2.4 num_commands

```
size_t num_commands
```

3.5.2.5 stdin_file

```
const char* stdin_file
```

3.5.2.6 stdout_file

```
const char* stdout_file
```

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

- parser.h

3.6 Pcb Struct Reference

```
#include <pcb.h>
```

Data Fields

- ucontext_t ∗ uc
- pid_t pid
- pid_t ppid
- pid_t pgid
- int priority
- int status
- int stateChangeType
- int fd [2]
- struct Pcb ∗ next
- struct Pcb ∗ prev
- struct Pcb ∗ next_child
- struct Pcb ∗ prev_child
- struct queue ∗ children_pcb
- struct queue ∗ zombies
- struct Pcb ∗ parent_pcb
- int number_of_children
- int exited_child
- char ∗ name
- bool sleeping
- unsigned int sleep_time
- bool waiting
- pid_t waiting_on
- struct parsed_command ∗ cmd
- bool state_change
- struct Pcb ∗ changed_child_pcb
- bool has_tc
- bool fg
- bool reading

3.6.1 Field Documentation

3.6.1.1 changed_child_pcb

```
struct Pcb* changed_child_pcb
```

3.6.1.2 children_pcb

```
struct queue* children_pcb
```

3.6.1.3 cmd

```
struct parsed_command* cmd
```

3.6.1.4 exited_child

```
int exited_child
```

3.6.1.5 fd

```
int fd[2]
```

3.6.1.6 fg

```
bool fg
```

3.6.1.7 has_tc

```
bool has_tc
```

3.6.1.8 name

```
char* name
```

3.6.1.9 next

```
struct Pcb* next
```

3.6.1.10 next_child

```
struct Pcb* next_child
```

3.6.1.11 number_of_children

```
int number_of_children
```

3.6.1.12 parent_pcb

```
struct Pcb* parent_pcb
```

3.6.1.13 pgid

```
pid_t pgid
```

3.6.1.14 pid

```
pid_t pid
```

3.6.1.15 ppid

```
pid_t ppid
```

3.6.1.16 prev

```
struct Pcb* prev
```

3.6.1.17 prev_child

```
struct Pcb* prev_child
```

3.6.1.18 priority

```
int priority
```

3.6.1.19 reading

```
bool reading
```

3.6.1.20 sleep_time

```
unsigned int sleep_time
```

3.6.1.21 sleeping

```
bool sleeping
```

3.6.1.22 state_change

```
bool state_change
```

3.6.1.23 stateChangeType

```
int stateChangeType
```

3.6.1.24 status

```
int status
```

3.6.1.25 uc

```
ucontext_t* uc
```

3.6.1.26 waiting

```
bool waiting
```

3.6.1.27 waiting_on

```
pid_t waiting_on
```

3.6.1.28 zombies

```
struct queue* zombies
```

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

- pcb.h

3.7 queue Struct Reference

```
#include <queue.h>
```

Data Fields

- struct Pcb ∗ head
- struct Pcb ∗ tail
- int size

3.7.1 Field Documentation

3.7.1.1 head

```
struct Pcb* head
```

3.7.1.2 size

```
int size
```

3.7.1.3 tail

```
struct Pcb* tail
```

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

- queue.h

3.8 superblock Struct Reference

```
#include <f.h>
```

Data Fields

- int fd
- int Block_size
- int DBoffset
- int num_data_Blocks
- int num_FATBlocks
- int FAT_tablesize
- int rdsize

3.8.1 Field Documentation

3.8.1.1 Block_size

```
int Block_size
```

3.8.1.2 DBoffset

```
int DBoffset
```

3.8.1.3 FAT_tablesize

```
int FAT_tablesize
```

3.8.1.4 fd

```
int fd
```

3.8.1.5 num_data_Blocks

```
int num_data_Blocks
```

3.8.1.6 num_FATBlocks

```
int num_FATBlocks
```

3.8.1.7 rdsize

```
int rdsize
```

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

- f.h
- PennFAT.h

Chapter 4

File Documentation

4.1 errors.c File Reference

```
#include "errors.h"
```

Functions

- void set_errno (enum ERRORS num)

4.1.1 Function Documentation

4.1.1.1 set_errno()

```
void set_errno (
            enum ERRORS num )
```

4.2 errors.h File Reference

Enumerations

- enum ERRORS { DEFAULT , PID_INVALID }

Functions

- void p_perror (char ∗str)
- void set_errno (enum ERRORS num)

4.2.1 Enumeration Type Documentation

4.2.1.1 ERRORS

```
enum ERRORS
```

Enumerator

DEFAULT	
PID_INVALID	

4.2.2 Function Documentation

4.2.2.1 p_perror()

```
void p_perror (
             char * str )
```

4.2.2.2 set_errno()

```
void set_errno (
             enum ERRORS num )
```

4.3 errors.h

Go to the documentation of this file.

```
#ifndef ERR_H
#define ERR_H

enum ERRORS
{
    DEFAULT,
    PID_INVALID
};

static int error_number = 0;

void p_perror(char *str);
void set_errno(enum ERRORS num);

#endif
```

4.4 f.c File Reference

```
#include "f.h"
#include <fcntl.h>
#include <sys/mman.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include "parser.h"
#include <errno.h>
#include <time.h>
#include <sys/stat.h>
```

Macros

- #define INPUT_SIZE 10000
- #define F_SEEK_SET 0
- #define F_SEEK_CUR 1
- #define F_SEEK_END 2

Functions

- bool check_allzero (char ∗entry)
- int find_available_block ()
- void update_fat (int cur, int block)
- void update_dir_entry (struct Root_directory_entry ∗entry, int delete)
- void create_newfile (char ∗fn)
- int contain_file (char ∗filename)
- void touch (char ∗∗file_name, int fsize)
- void rm (char ∗fn, int delete)
- void data_write (char ∗fn, char ∗data, int write_size)
- void chmod1 (char ∗filename, char ∗perm)
- int f_cat (char ∗∗cmd, int filenum)
- int fatfs_init (char ∗fatfs)
- int f_open (const char ∗fname, int mode)
- int f_read (int fd, int n, char ∗buf)
- int f_write (int fd, const char ∗str, int n)
- int f_close (int fd)
- int f_lseek (int fd, int offset, int whence)
- int f_unlink (const char ∗fname)
- int f_ls (const char ∗filename)
- int f_rm (char ∗file)
- int f_mv (char ∗source, char ∗dest)
- int f_copy (char ∗source, char ∗dest1)
- int f_touch (char ∗∗file_name, int fsize)
- int f_unmount ()
- int f_chmod ()

Variables

- struct superblock sb
- struct Root_directory_entry ∗ rd
- int16_t ∗ FAT_block

4.4.1 Macro Definition Documentation

4.4.1.1 F_SEEK_CUR

```
#define F_SEEK_CUR 1
```

4.4.1.2 F_SEEK_END

```
#define F_SEEK_END 2
```

4.4.1.3 F_SEEK_SET

```
#define F_SEEK_SET 0
```

4.4.1.4 INPUT_SIZE

```
#define INPUT_SIZE 10000
```

4.4.2 Function Documentation

4.4.2.1 check_allzero()

```
bool check_allzero (
             char * entry )
```

4.4.2.2 chmod1()

```
void chmod1 (
             char * filename,
             char * perm )
```

4.4.2.3 contain_file()

```
int contain_file (
             char * filename )
```

4.4.2.4 create_newfile()

```
void create_newfile (
             char * fn )
```

f1 f2 f3 f4

4.4.2.5 data_write()

```
void data_write (
             char * fn,
             char * data,
             int write_size )
```

1 root dir->filename, firstblock first block->data:first data, FAT_table[firstblock]=nextblock use size to write.

FAT_block update

update fat block in file

appif

cur is the last block

write to data

4.4.2.6 f_cat()

```
int f_cat (
             char ** cmd,
             int filenum )
```

4.4.2.7 f_chmod()

```
int f_chmod ( )
```

4.4.2.8 f_close()

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

4.4.2.9 f_copy()

```
int f_copy (
            char * source,
            char * dest1 )
```

4.4.2.10 f_ls()

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

4.4.2.11 f_lseek()

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

4.4.2.12 f_mv()

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

4.4.2.13 f_open()

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

F_WRITE

F_APPEND

4.4.2.14 f_read()

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

4.4.2.15 f_rm()

```
int f_rm (
             char * file )
```

4.4.2.16 f_touch()

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

check if the file already exist(check rd)

4.4.2.17 f_unlink()

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

4.4.2.18 f_unmount()

```
int f_unmount ( )
```

4.4.2.19 f_write()

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

FAT_block update

update fat block in file

write to data

4.4.2.20 fatfs_init()

```
int fatfs_init (
             char * fatfs )
```

rdsize wrong!!

4.4.2.21 find_available_block()

```
int find_available_block ( )
```

4.4.2.22 rm()

```
void rm (
            char * fn,
            int delete )
```

4.4.2.23 touch()

```
void touch (
            char ** file_name,
            int fsize )
```

check if the file already exist(check rd)

4.4.2.24 update_dir_entry()

```
void update_dir_entry (
            struct Root_directory_entry * entry,
            int delete )
```

4.4.2.25 update_fat()

```
void update_fat (
            int cur,
            int block )
```

4.4.3 Variable Documentation

4.4.3.1 FAT_block

```
int16_t* FAT_block
```

4.4.3.2 rd

```
struct Root_directory_entry* rd
```

4.4.3.3 sb

```
struct superblock sb
```

4.5 f.h File Reference

```
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
```

Data Structures

- struct superblock

Functions

- struct __attribute__ ((__packed__)) Root_directory_entry
- int f_open (const char ∗fname, int mode)
- int f_read (int fd, int n, char ∗buf)
- int f_unlink (const char ∗fname)
- int f_write (int fd, const char ∗str, int n)
- int f_close (int fd)
- int f_ls (const char ∗filename)
- int f_lseek (int fd, int offset, int whence)
- int f_touch (char ∗∗file_name, int fsize)
- int f_copy (char ∗source, char ∗dest1)
- int f_unmount ()
- int f_cat (char ∗∗cmd, int filenum)
- int f_rm (char ∗file)
- int f_mv (char ∗source, char ∗dest)
- int fatfs_init (char ∗fatfs)

4.5.1 Function Documentation

4.5.1.1 __attribute__()

```
struct __attribute__ (
             (__packed__)  )
```

4.5.1.2 f_cat()

```
int f_cat (
             char ** cmd,
             int filenum )
```

4.5.1.3 f_close()

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

4.5.1.4 f_copy()

```
int f_copy (
             char * source,
             char * dest1 )
```

4.5.1.5 f_ls()

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

4.5.1.6 f_lseek()

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

4.5.1.7 f_mv()

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

4.5.1.8 f_open()

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

F_WRITE

F_APPEND

4.5.1.9 f_read()

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

4.5.1.10 f_rm()

```
int f_rm (
             char * file )
```

4.5.1.11 f_touch()

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

check if the file already exist(check rd)

4.5.1.12 f_unlink()

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

4.5.1.13 f_unmount()

```
int f_unmount ( )
```

4.5.1.14 f_write()

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

FAT_block update

update fat block in file

write to data

4.5.1.15 fatfs_init()

```
int fatfs_init (
             char * fatfs )
```

rdsize wrong!!

4.6 f.h

Go to the documentation of this file.

```
#ifndef F_H
#define F_H
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
struct superblock {
    int fd;
    int Block_size; //total amount of blocks of virtual disk
    //int16_t RdirIndex; // root directory block index
    int DBoffset; // data block start index
    int num_data_Blocks; // amount of data blocks
    int num_FATBlocks; // number of blocks for FAT
    int FAT_tablesize;
    int rdsize;

};
struct __attribute__((__packed__)) Root_directory_entry {
    char name[32];
    uint32_t size;
    uint16_t firstBlock;
    uint8_t type;
    uint8_t perm;
    time_t mtime;
    char remain[16];
    int cursor;

};

int f_open(const char *fname, int mode);
int f_read(int fd, int n, char *buf);
int f_unlink(const char *fname);
int f_write(int fd, const char *str, int n);
int f_close(int fd);
int f_ls(const char *filename);
int f_lseek(int fd, int offset, int whence);
int f_touch(char**file_name,int fsize);
int f_copy(char *source, char *dest1);
int f_unmount();
int f_cat(char **cmd, int filenum);
int f_rm(char *file);
int f_mv(char *source, char *dest);
int f_copy(char *source, char *dest1);
int f_touch(char**file_name,int fsize);
int fatfs_init(char *fatfs);


#endif
```

4.7 j_queue.c File Reference

```
#include "j_queue.h"
#include <stdio.h>
```

Functions

- struct j_queue * create_j_queue ()
- void push_job (struct j_queue *q, struct Job *j)
- struct Job * pop_job (struct j_queue *q)
- void free_j_queue (struct j_queue *q)
- struct Job * find_job_with_id (struct j_queue *q, int job_id)
- struct Job * find_last_stopped_job (struct j_queue *jobs_list)
- void remove_job (struct j_queue *q, struct Job *p)
- void print_j_list (struct j_queue *ll)

4.7.1 Function Documentation

4.7.1.1 create_j_queue()

```
struct j_queue * create_j_queue ( )
```

4.7.1.2 find_job_with_id()

```
struct Job * find_job_with_id (
            struct j_queue * q,
            int job_id )
```

4.7.1.3 find_last_stopped_job()

```
struct Job * find_last_stopped_job (
             struct j_queue * jobs_list )
```

4.7.1.4 free_j_queue()

```
void free_j_queue (
             struct j_queue * q )
```

4.7.1.5 pop_job()

```
struct Job * pop_job (
             struct j_queue * q )
```

4.7.1.6 print_j_list()

```
void print_j_list (
             struct j_queue * ll )
```

4.7.1.7 push_job()

```
void push_job (
             struct j_queue * q,
             struct Job * j )
```

4.7.1.8 remove_job()

```
void remove_job (
             struct j_queue * q,
             struct Job * p )
```

4.8 j_queue.h File Reference

```
#include "Job.h"
```

Data Structures

- struct j_queue

Functions

- struct j_queue ∗ create_j_queue ()
- void push_job (struct j_queue ∗q, struct Job ∗j)
- struct Job ∗ pop_job (struct j_queue ∗q)
- void free_j_queue (struct j_queue ∗q)
- struct Job ∗ find_job_with_id (struct j_queue ∗q, int job_id)
- struct Job ∗ find_last_stopped_job (struct j_queue ∗jobs_list)
- void remove_job (struct j_queue ∗q, struct Job ∗p)
- void print_j_list (struct j_queue ∗ll)

4.8.1 Function Documentation

4.8.1.1 create_j_queue()

```
struct j_queue * create_j_queue ( )
```

4.8.1.2 find_job_with_id()

```
struct Job * find_job_with_id (
             struct j_queue * q,
             int job_id )
```

4.8.1.3 find_last_stopped_job()

```
struct Job * find_last_stopped_job (
             struct j_queue * jobs_list )
```

4.8.1.4 free_j_queue()

```
void free_j_queue (
             struct j_queue * q )
```

4.8.1.5 pop_job()

```
struct Job * pop_job (
            struct j_queue * q )
```

4.8.1.6 print_j_list()

```
void print_j_list (
            struct j_queue * ll )
```

4.8.1.7 push_job()

```
void push_job (
            struct j_queue * q,
            struct Job * j )
```

4.8.1.8 remove_job()

```
void remove_job (
            struct j_queue * q,
            struct Job * p )
```

4.9 j_queue.h

Go to the documentation of this file.

```
#ifndef JQ_H
#define JQ_H

#include "Job.h"

struct j_queue
{
    struct Job *head;
    struct Job *tail;
    int size;
};

struct j_queue *create_j_queue();
void push_job(struct j_queue *q, struct Job *j);
struct Job *pop_job(struct j_queue *q);
void free_j_queue(struct j_queue *q);
struct Job *find_job_with_id(struct j_queue *q, int job_id);
struct Job *find_last_stopped_job(struct j_queue *jobs_list);
void remove_job(struct j_queue *q, struct Job *p);
void print_j_list(struct j_queue *ll);
#endif
```

4.10 Job.c File Reference

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

Functions

- struct Job ∗ create_job (struct parsed_command ∗cmd, char ∗rawCmd, int group_id, enum jobStatus status, enum jobGround ground)
- void free_job (struct Job ∗job)

4.10.1 Function Documentation

4.10.1.1 create_job()

```
struct Job * create_job (
            struct parsed_command * cmd,
            char * rawCmd,
            int group_id,
            enum jobStatus status,
            enum jobGround ground )
```

4.10.1.2 free_job()

```
void free_job (
            struct Job * job )
```

4.11 Job.h File Reference

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

Data Structures

- struct Job

Enumerations

- enum jobStatus { J_RUNNING , J_STOPPED , J_FINISHED , J_TERMINATED }
- enum jobGround { BG , FG }

Functions

- struct Job ∗ create_job (struct parsed_command ∗cmd, char ∗rawCmd, int group_id, enum jobStatus status, enum jobGround ground)
- void free_job (struct Job ∗job)

4.11.1 Enumeration Type Documentation

4.11.1.1 jobGround

```
enum jobGround
```

Enumerator

BG	
FG	

4.11.1.2 jobStatus

```
enum jobStatus
```

Enumerator

J_RUNNING	
J_STOPPED	
J_FINISHED	
J_TERMINATED	

4.11.2 Function Documentation

4.11.2.1 create_job()

```
struct Job * create_job (
            struct parsed_command * cmd,
            char * rawCmd,
            int group_id,
            enum jobStatus status,
            enum jobGround ground )
```

4.11.2.2 free_job()

```
void free_job (
              struct Job * job )
```

4.12 Job.h

Go to the documentation of this file.

```
#ifndef JOB_H
#define JOB_H

#include "parser.h"
#include <stdlib.h>

static int JOBCOUNT = 1;

enum jobStatus
{
    J_RUNNING,
    J_STOPPED,
    J_FINISHED,
    J_TERMINATED
};
enum jobGround
{
    BG,
    FG
};

struct Job
{
    struct parsed_command *cmd;
    char *rawCmd; // the unparsed command as a single string
    int group_id;
    struct Job *next;
    struct Job *prev;
    int job_id;
    enum jobStatus status;
    enum jobGround ground;
    int count_finished;
};

struct Job *create_job(struct parsed_command *cmd,
                       char *rawCmd,
                       int group_id,
                       enum jobStatus status,
                       enum jobGround ground);
void free_job(struct Job *job);
#endif
```

4.13 kernel.c File Reference

```
#include "kernel.h"
#include <valgrind/valgrind.h>
```

Typedefs

- typedef struct queue q

Functions

- struct queue ∗ get_next_queue ()
- int get_clock_ticks ()

    Getter for clock ticks.
- int get_how_ended ()
- void set_how_ended (int how)
- void make_context (ucontext_t ∗ucp, void(∗func)(), int argc, char ∗argv[ ], bool link_to_scheduler)
- int get_counter ()
- void initialise_queues ()
- struct Pcb ∗ get_pcb (pid_t pi)
- struct queue ∗ get_pcbs_with_pgid (pid_t pgid)
- void print_all_queues_info ()
- void inc_clock ()
- struct queue ∗ get_queue_with_priority (int priority)
- void free_queues ()
- struct Pcb ∗ get_active_pcb ()
- ucontext_t ∗ get_active_context ()
- void set_active_pcb_and_context (struct Pcb ∗p)
- int get_num_of_ready_processes ()
- void setup_log ()
- FILE ∗ get_log_fp ()
- void setup_idle_process ()
- void idle_process ()
- struct queue ∗ get_blocked_queue ()
- struct queue ∗ get_next_valid_queue ()
- void scheduler ()
- struct Pcb ∗ k_process_create (struct Pcb ∗parent)
- void k_process_block (struct Pcb ∗p, struct Pcb ∗cp)

    Blocks the process.
- pid_t k_waitpid (struct Pcb ∗calling_pcb, struct Pcb ∗child_pcb, int ∗wstatus, pid_t pid, bool nohang)

    Kernel side function used for waiting on a child process to terminate or stop.
- void k_exit (struct Pcb ∗calling_pcb)

    Kernel side assist function for p_exit.
- int k_process_kill (struct Pcb ∗process, int signal)

    Delivers signal to specified process.
- void k_process_cleanup (struct Pcb ∗process)

    Recursively cleanup a process's child tree when the process is terminated.
- void k_sleep (unsigned int ticks)

    Put the currently executing process to sleep for a specified number of timer ticks.
- ucontext_t ∗ get_scheduler_context (void)
- ucontext_t ∗ get_shell_context (void)
- void init_scheduler_context ()
- void switch_to_scheduler ()

    Saves active context and switches to scheduler context.
- void k_boot_kernel ()
- void k_block_sigset (sigset_t ∗mask)
- void k_unblock_sigset (sigset_t ∗mask)
- int k_tcset (pid_t pid, pid_t ∗tc_pid)

    Kernel side function used for setting terminal control.

Variables

- pid_t pid_count = 1
- FILE ∗ log_fp
- sigset_t signal_set

4.13.1 Typedef Documentation

4.13.1.1 q

```
typedef struct queue q
```

4.13.2 Function Documentation

4.13.2.1 free_queues()

```
void free_queues ( )
```

4.13.2.2 get_active_context()

```
ucontext_t ∗ get_active_context ( )
```

4.13.2.3 get_active_pcb()

```
struct Pcb ∗ get_active_pcb ( )
```

4.13.2.4 get_blocked_queue()

```
struct queue ∗ get_blocked_queue ( )
```

4.13.2.5 get_clock_ticks()

```
int get_clock_ticks ( )
```

Getter for clock ticks.

4.13.2.6 get_counter()

```
int get_counter ( )
```

4.13.2.7 get_how_ended()

```
int get_how_ended ( )
```

4.13.2.8 get_log_fp()

```
FILE * get_log_fp ( )
```

4.13.2.9 get_next_queue()

```
struct queue * get_next_queue ( )
```

4.13.2.10 get_next_valid_queue()

```
struct queue * get_next_valid_queue ( )
```

4.13.2.11 get_num_of_ready_processes()

```
int get_num_of_ready_processes ( )
```

4.13.2.12 get_pcb()

```
struct Pcb * get_pcb (
            pid_t pi )
```

4.13.2.13 get_pcbs_with_pgid()

```
struct queue * get_pcbs_with_pgid (
            pid_t pgid )
```

4.13.2.14 get_queue_with_priority()

```
struct queue * get_queue_with_priority (
            int priority )
```

4.13.2.15 get_scheduler_context()

```
ucontext_t * get_scheduler_context (
            void  )
```

4.13.2.16 get_shell_context()

```
ucontext_t * get_shell_context (
            void  )
```

4.13.2.17 idle_process()

```
void idle_process ( )
```

4.13.2.18 inc_clock()

```
void inc_clock ( )
```

4.13.2.19 init_scheduler_context()

```
void init_scheduler_context ( )
```

4.13.2.20 initialise_queues()

```
void initialise_queues ( )
```

4.13.2.21 k_block_sigset()

```
void k_block_sigset (
            sigset_t * mask )
```

4.13.2.22 k_boot_kernel()

```
void k_boot_kernel ( )
```

4.13.2.23 k_exit()

```
void k_exit (
            struct Pcb * calling_pcb )
```

Kernel side assist function for p_exit.

Parameters

calling_pcb	PCB of the process calling p_exit

4.13.2.24 k_process_block()

```
void k_process_block (
            struct Pcb * p,
            struct Pcb * cp )
```

Blocks the process.

Parameters

p	Refers to the PCB of the process to be blocked
cp	Optional parameter referring to child process being waited on leading to parent getting blocked

4.13.2.25 k_process_cleanup()

```
void k_process_cleanup (
             struct Pcb * process )
```

Recursively cleanup a process's child tree when the process is terminated.

Parameters

process	Pointer to the process's PCB

4.13.2.26 k_process_create()

```
struct Pcb * k_process_create (
             struct Pcb * parent )
```

4.13.2.27 k_process_kill()

```
int k_process_kill (
             struct Pcb * process,
             int signal )
```

Delivers signal to specified process.

Parameters

process	The process being signaled
signal	The kind of signal being delivered

Returns

CHANGE 0, NO CHANGE 1 where CHANGE indicates that a process changed state as a result of the signal

4.13.2.28 k_sleep()

```
void k_sleep (
            unsigned int ticks )
```

Put the currently executing process to sleep for a specified number of timer ticks.

Parameters

ticks	The number of timer ticks to sleep for

4.13.2.29 k_tcset()

```
int k_tcset (
            pid_t pid,
            pid_t * tc_pid )
```

Kernel side function used for setting terminal control.

Parameters

pid	PID of the process being given terminal control
tc_pid	Used for tracking which process (with this PID) has terminal control

Returns

0 on success and -1 on error

4.13.2.30 k_unblock_sigset()

```
void k_unblock_sigset (
            sigset_t * mask )
```

4.13.2.31 k_waitpid()

```
pid_t k_waitpid (
            struct Pcb * calling_pcb,
            struct Pcb * child_pcb,
            int * wstatus,
            pid_t pid,
            bool nohang )
```

Kernel side function used for waiting on a child process to terminate or stop.

Parameters

calling_pcb	Pointer to the PCB of the process that is waiting on the child process
child_pcb	Pointer to the PCB of the child process being waited on
wstatus	Pointer to an integer variable used to store the status of the child process
pid	PID of the child process being waited on
nohang	If true, return immediately if the child process is not terminated or stopped

Returns

0 if the child process is not terminated or stopped and nohang is true, or the PID of the child process if it is terminated or stopped

4.13.2.32 make_context()

```
void make_context (
             ucontext_t * ucp,
             void(*)() func,
             int argc,
             char * argv[],
             bool link_to_scheduler )
```

4.13.2.33 print_all_queues_info()

```
void print_all_queues_info ( )
```

4.13.2.34 scheduler()

```
void scheduler ( )
```

4.13.2.35 set_active_pcb_and_context()

```
void set_active_pcb_and_context (
             struct Pcb * p )
```

4.13.2.36 set_how_ended()

```
void set_how_ended (
             int how )
```

4.13.2.37 setup_idle_process()

```
void setup_idle_process ( )
```

4.13.2.38 setup_log()

```
void setup_log ( )
```

4.13.2.39 switch_to_scheduler()

```
void switch_to_scheduler ( )
```

Saves active context and switches to scheduler context.

4.13.3 Variable Documentation

4.13.3.1 log_fp

```
FILE* log_fp
```

4.13.3.2 pid_count

```
pid_t pid_count = 1
```

4.13.3.3 signal_set

```
sigset_t signal_set
```

4.14 kernel.h File Reference

```
#include "queue.h"
#include "user.h"
#include <stdio.h>
```

Enumerations

- enum HOW_ENDED { TIMED_OUT , FIN_RUN }

Functions

- void idle_process ()
- struct Pcb ∗ get_active_pcb ()
- int get_counter ()
- void boot_kernel ()
- void setup_log ()
- FILE ∗ get_log_fp ()
- int get_num_of_ready_processes ()
- struct Pcb ∗ k_process_create (struct Pcb ∗parent)
- int k_process_kill (struct Pcb ∗process, int signal)

    Delivers signal to specified process.
- void k_process_cleanup (struct Pcb ∗process)

    Recursively cleanup a process's child tree when the process is terminated.
- void k_sleep (unsigned int ticks)

    Put the currently executing process to sleep for a specified number of timer ticks.
- struct queue ∗ get_queue_with_priority (int priority)
- struct queue ∗ get_blocked_queue ()
- struct Pcb ∗ get_pcb (pid_t pi)
- ucontext_t ∗ get_scheduler_context (void)
- ucontext_t ∗ get_shell_context (void)
- void make_context (ucontext_t ∗ucp, void(∗func)(), int argc, char ∗argv[ ], bool link_to_scheduler)
- struct queue ∗ get_pcbs_with_pgid (pid_t pgid)
- int get_clock_ticks ()

    Getter for clock ticks.
- void switch_to_scheduler ()

    Saves active context and switches to scheduler context.
- void k_process_block (struct Pcb ∗p, struct Pcb ∗cp)

    Blocks the process.
- void k_exit (struct Pcb ∗calling_pcb)

    Kernel side assist function for p_exit.
- int k_tcset (pid_t pid, pid_t ∗tc_pid)

    Kernel side function used for setting terminal control.
- pid_t k_waitpid (struct Pcb ∗calling_pcb, struct Pcb ∗child_pcb, int ∗wstatus, pid_t pid, bool nohang)

    Kernel side function used for waiting on a child process to terminate or stop.
- void wait_log (struct Pcb ∗calling_pcb)
- void k_boot_kernel ()

4.14.1 Enumeration Type Documentation

4.14.1.1 HOW_ENDED

```
enum HOW_ENDED
```

Enumerator

TIMED_OUT	
FIN_RUN	

4.14.2 Function Documentation

4.14.2.1 boot_kernel()

```
void boot_kernel ( )
```

4.14.2.2 get_active_pcb()

```
struct Pcb * get_active_pcb ( )
```

4.14.2.3 get_blocked_queue()

```
struct queue * get_blocked_queue ( )
```

4.14.2.4 get_clock_ticks()

```
int get_clock_ticks ( )
```

Getter for clock ticks.

4.14.2.5 get_counter()

```
int get_counter ( )
```

4.14.2.6 get_log_fp()

```
FILE * get_log_fp ( )
```

4.14.2.7 get_num_of_ready_processes()

```
int get_num_of_ready_processes ( )
```

4.14.2.8 get_pcb()

```
struct Pcb * get_pcb (
             pid_t pi )
```

4.14.2.9 get_pcbs_with_pgid()

```
struct queue * get_pcbs_with_pgid (
             pid_t pgid )
```

4.14.2.10 get_queue_with_priority()

```
struct queue * get_queue_with_priority (
             int priority )
```

4.14.2.11 get_scheduler_context()

```
ucontext_t * get_scheduler_context (
             void  )
```

4.14.2.12 get_shell_context()

```
ucontext_t * get_shell_context (
             void )
```

4.14.2.13 idle_process()

```
void idle_process ( )
```

4.14.2.14 k_boot_kernel()

```
void k_boot_kernel ( )
```

4.14.2.15 k_exit()

```
void k_exit (
             struct Pcb * calling_pcb )
```

Kernel side assist function for p_exit.

Parameters

calling_pcb	PCB of the process calling p_exit

4.14.2.16 k_process_block()

```
void k_process_block (
             struct Pcb * p,
             struct Pcb * cp )
```

Blocks the process.

Parameters

p	Refers to the PCB of the process to be blocked
cp	Optional parameter referring to child process being waited on leading to parent getting blocked

4.14.2.17 k_process_cleanup()

```
void k_process_cleanup (
            struct Pcb * process )
```

Recursively cleanup a process's child tree when the process is terminated.

Parameters

process	Pointer to the process's PCB

4.14.2.18 k_process_create()

```
struct Pcb * k_process_create (
            struct Pcb * parent )
```

4.14.2.19 k_process_kill()

```
int k_process_kill (
            struct Pcb * process,
            int signal )
```

Delivers signal to specified process.

Parameters

process	The process being signaled
signal	The kind of signal being delivered

Returns

CHANGE 0, NO CHANGE 1 where CHANGE indicates that a process changed state as a result of the signal

4.14.2.20 k_sleep()

```
void k_sleep (
            unsigned int ticks )
```

Put the currently executing process to sleep for a specified number of timer ticks.

Parameters

ticks	The number of timer ticks to sleep for

4.14.2.21 k_tcset()

```
int k_tcset (
            pid_t pid,
            pid_t * tc_pid )
```

Kernel side function used for setting terminal control.

Parameters

pid	PID of the process being given terminal control
tc_pid	Used for tracking which process (with this PID) has terminal control

Returns

0 on success and -1 on error

4.14.2.22 k_waitpid()

```
pid_t k_waitpid (
            struct Pcb * calling_pcb,
            struct Pcb * child_pcb,
            int * wstatus,
            pid_t pid,
            bool nohang )
```

Kernel side function used for waiting on a child process to terminate or stop.

Parameters

calling_pcb	Pointer to the PCB of the process that is waiting on the child process
child_pcb	Pointer to the PCB of the child process being waited on
wstatus	Pointer to an integer variable used to store the status of the child process
pid	PID of the child process being waited on
nohang	If true, return immediately if the child process is not terminated or stopped

Returns

0 if the child process is not terminated or stopped and nohang is true, or the PID of the child process if it is terminated or stopped

4.14.2.23 make_context()

```
void make_context (
            ucontext_t * ucp,
            void(*)() func,
            int argc,
            char * argv[],
            bool link_to_scheduler )
```

4.14.2.24 setup_log()

```
void setup_log ( )
```

4.14.2.25 switch_to_scheduler()

```
void switch_to_scheduler ( )
```

Saves active context and switches to scheduler context.

4.14.2.26 wait_log()

```
void wait_log (
            struct Pcb * calling_pcb )
```

4.15 kernel.h

Go to the documentation of this file.

```
#ifndef KER_H
#define KER_H
#include "queue.h"
#include "user.h"
#include <stdio.h>
enum HOW_ENDED {
    TIMED_OUT,
    FIN_RUN
};


static sigset_t intmask;
void idle_process();
struct Pcb *get_active_pcb();
int get_counter();
void boot_kernel();
void setup_log();
FILE *get_log_fp();
int get_num_of_ready_processes();
struct Pcb *k_process_create(struct Pcb *parent);

```

```
int k_process_kill(struct Pcb *process, int signal);

void k_process_cleanup(struct Pcb *process);

void k_sleep(unsigned int ticks);
struct queue *get_queue_with_priority(int priority);
struct queue *get_blocked_queue();
struct Pcb *get_pcb(pid_t pi); // Check all queues for the pcb with given pid

ucontext_t *get_scheduler_context(void);

ucontext_t *get_shell_context(void);
// void initialise_queues();
// void setup_idle_process();
// void init_scheduler_context();


void make_context(ucontext_t *ucp, void (*func)(), int argc, char *argv[], bool link_to_scheduler);
struct queue *get_pcbs_with_pgid(pid_t pgid);

int get_clock_ticks();

void switch_to_scheduler();




void k_process_block(struct Pcb *p, struct Pcb *cp);

void k_exit(struct Pcb *calling_pcb);

int k_tcset(pid_t pid, pid_t *tc_pid);

pid_t k_waitpid(struct Pcb *calling_pcb, struct Pcb *child_pcb, int *wstatus, pid_t pid, bool nohang);


void wait_log(struct Pcb *calling_pcb);
static pid_t current_pid = 0;

void k_boot_kernel();
#endif
```

4.16 parser.h File Reference

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

Data Structures

- struct parsed_command

Macros

- #define UNEXPECTED_FILE_INPUT 1
- #define UNEXPECTED_FILE_OUTPUT 2
- #define UNEXPECTED_PIPELINE 3
- #define UNEXPECTED_AMPERSAND 4
- #define EXPECT_INPUT_FILENAME 5
- #define EXPECT_OUTPUT_FILENAME 6
- #define EXPECT_COMMANDS 7

Functions

- int parse_command (const char *cmd_line, struct parsed_command **result)
- void print_parsed_command (const struct parsed_command *cmd)

4.16.1 Macro Definition Documentation

4.16.1.1 EXPECT_COMMANDS

```
#define EXPECT_COMMANDS 7
```

4.16.1.2 EXPECT_INPUT_FILENAME

```
#define EXPECT_INPUT_FILENAME 5
```

4.16.1.3 EXPECT_OUTPUT_FILENAME

```
#define EXPECT_OUTPUT_FILENAME 6
```

4.16.1.4 UNEXPECTED_AMPERSAND

```
#define UNEXPECTED_AMPERSAND 4
```

4.16.1.5 UNEXPECTED_FILE_INPUT

```
#define UNEXPECTED_FILE_INPUT 1
```

4.16.1.6 UNEXPECTED_FILE_OUTPUT

```
#define UNEXPECTED_FILE_OUTPUT 2
```

4.16.1.7 UNEXPECTED_PIPELINE

```
#define UNEXPECTED_PIPELINE 3
```

4.16.2 Function Documentation

4.16.2.1 parse_command()

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

Arguments: cmd_line: a null-terminated string that is the command line result: a non-null pointer to a `struct parsed_command *`

Return value (int): an error code which can be, 0: parser finished succesfully -1: parser encountered a system call error 1-7: parser specific error, see error type above

This function will parse the given `cmd_line` and store the parsed information into a `struct parsed_command`. The memory needed for the struct will be allocated by this function, and the pointer to the memory will be stored into the given `*result`.

You can directly use the result in system calls. See demo for more information.

If the function returns a successful value (0), a `struct parsed_command` is guareenteed to be allocated and stored in the given `*result`. It is the caller's responsibility to free the given pointer using `free(3)`.

Otherwise, no `struct parsed_command` is allocated and `*result` is unchanged. If a system call error (-1) is returned, the caller can use `errno(3)` or `perror(3)` to gain more information about the error.

4.16.2.2 print_parsed_command()

```
void print_parsed_command (
            const struct parsed_command * cmd )
```

4.17 parser.h

Go to the documentation of this file.

```
/* Penn-Shell Parser
   hanbangw, 21fa    */

#pragma once

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

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

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

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

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

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

```

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

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

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

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

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

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

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

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

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


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

4.18 pcb.c File Reference

```
#include "pcb.h"
#include <stdio.h>
```

Functions

- struct Pcb ∗ create_pcb (ucontext_t ∗uc, pid_t pid, pid_t ppid, pid_t pgid, int priority, int status, struct parsed_command ∗cmd)
- struct Pcb ∗ create_pcb_no_context (pid_t pid, pid_t ppid, pid_t pgid, int priority, int status)
- void free_pcb (struct Pcb ∗p)
- void add_child_to_parent_q (struct Pcb ∗parent_pcb, struct Pcb ∗child_pcb)
- void remove_child_pcb (struct Pcb ∗parent_pcb, struct Pcb ∗child_pcb)

  Removes a child process from the list of children of a parent process but does not free the memory.
- void delete_child_pcb (struct Pcb ∗parent_pcb, struct Pcb ∗child_pcb)

  Removes a child process from the list of children of a parent process and frees the associated memory.
- void add_child_to_parent_zombies (struct Pcb ∗parent_pcb, struct Pcb ∗child_pcb)

  Adds a child process's PCB to a parent process's zombie queue.

4.18.1 Function Documentation

4.18.1.1 add_child_to_parent_q()

```
void add_child_to_parent_q (
             struct Pcb * parent_pcb,
             struct Pcb * child_pcb )
```

4.18.1.2 add_child_to_parent_zombies()

```
void add_child_to_parent_zombies (
             struct Pcb * parent_pcb,
             struct Pcb * child_pcb )
```

Adds a child process's PCB to a parent process's zombie queue.

Parameters

parent_pcb	Pointer to the parent process's PCB
child_pcb	Pointer to the child process's PCB

4.18.1.3 create_pcb()

```
struct Pcb * create_pcb (
             ucontext_t * uc,
             pid_t pid,
             pid_t ppid,
             pid_t pgid,
             int priority,
             int status,
             struct parsed_command * cmd )
```

4.18.1.4 create_pcb_no_context()

```
struct Pcb * create_pcb_no_context (
             pid_t pid,
             pid_t ppid,
             pid_t pgid,
             int priority,
             int status )
```

4.18.1.5 delete_child_pcb()

```
void delete_child_pcb (
             struct Pcb * parent_pcb,
             struct Pcb * child_pcb )
```

Removes a child process from the list of children of a parent process and frees the associated memory.

Parameters

parent_pcb	Pointer to the parent process's PCB
child_pcb	Pointer to the child process's PCB

4.18.1.6 free_pcb()

```
void free_pcb (
            struct Pcb * p )
```

4.18.1.7 remove_child_pcb()

```
void remove_child_pcb (
            struct Pcb * parent_pcb,
            struct Pcb * child_pcb )
```

Removes a child process from the list of children of a parent process but does not free the memory.

Parameters

parent_pcb	Pointer to the parent process's PCB
child_pcb	Pointer to the child process's PCB

4.19 pcb.h File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <ucontext.h>
#include <string.h>
#include <stdbool.h>
#include "parser.h"
#include "queue.h"
```

Data Structures

- struct Pcb

Enumerations

- enum status {
  READY , RUNNING , BLOCKED , STOPPED ,
  TERMINATED , FINISHED , ZOMBIE , ORPHAN }

Functions

- struct Pcb ∗ create_pcb (ucontext_t ∗uc, pid_t pid, pid_t ppid, pid_t pgid, int priority, int status, struct parsed_command ∗cmd)
- struct Pcb ∗ create_pcb_no_context (pid_t pid, pid_t ppid, pid_t pgid, int priority, int status)
- void free_pcb (struct Pcb ∗pcb)
- void add_child_to_parent_q (struct Pcb ∗parent_pcb, struct Pcb ∗child_pcb)
- void remove_child_pcb (struct Pcb ∗parent_pcb, struct Pcb ∗child_pcb)

  Removes a child process from the list of children of a parent process but does not free the memory.
- void delete_child_pcb (struct Pcb ∗parent_pcb, struct Pcb ∗child_pcb)

  Removes a child process from the list of children of a parent process and frees the associated memory.
- void add_child_to_parent_zombies (struct Pcb ∗parent_pcb, struct Pcb ∗child_pcb)

  Adds a child process's PCB to a parent process's zombie queue.

4.19.1 Enumeration Type Documentation

4.19.1.1 status

```
enum status
```

Enumerator

READY	
RUNNING	
BLOCKED	
STOPPED	
TERMINATED	
FINISHED	
ZOMBIE	
ORPHAN	

4.19.2 Function Documentation

4.19.2.1 add_child_to_parent_q()

```
void add_child_to_parent_q (
            struct Pcb * parent_pcb,
            struct Pcb * child_pcb )
```

4.19.2.2 add_child_to_parent_zombies()

```
void add_child_to_parent_zombies (
             struct Pcb * parent_pcb,
             struct Pcb * child_pcb )
```

Adds a child process's PCB to a parent process's zombie queue.

Parameters

parent_pcb	Pointer to the parent process's PCB
child_pcb	Pointer to the child process's PCB

4.19.2.3 create_pcb()

```
struct Pcb * create_pcb (
             ucontext_t * uc,
             pid_t pid,
             pid_t ppid,
             pid_t pgid,
             int priority,
             int status,
             struct parsed_command * cmd )
```

4.19.2.4 create_pcb_no_context()

```
struct Pcb * create_pcb_no_context (
             pid_t pid,
             pid_t ppid,
             pid_t pgid,
             int priority,
             int status )
```

4.19.2.5 delete_child_pcb()

```
void delete_child_pcb (
             struct Pcb * parent_pcb,
             struct Pcb * child_pcb )
```

Removes a child process from the list of children of a parent process and frees the associated memory.

Parameters

parent_pcb	Pointer to the parent process's PCB
child_pcb	Pointer to the child process's PCB

4.19.2.6 free_pcb()

```
void free_pcb (
            struct Pcb * pcb )
```

4.19.2.7 remove_child_pcb()

```
void remove_child_pcb (
            struct Pcb * parent_pcb,
            struct Pcb * child_pcb )
```

Removes a child process from the list of children of a parent process but does not free the memory.

Parameters

parent_pcb	Pointer to the parent process's PCB
child_pcb	Pointer to the child process's PCB

4.20 pcb.h

Go to the documentation of this file.

```
#ifndef PCB_H
#define PCB_H

#include <unistd.h>
#include <stdlib.h>
#include <ucontext.h>
#include <string.h>
#include <stdbool.h>
#include "parser.h"
#include "queue.h"

enum status {
    READY,
    RUNNING,
    BLOCKED,
    STOPPED,
    TERMINATED,
    FINISHED,
    ZOMBIE,
    ORPHAN
};

struct Pcb
{
    ucontext_t *uc;
    pid_t pid;
    pid_t ppid;
    pid_t pgid;
    int priority;
    int status;
    int stateChangeType; // 0 if no change; 1 if terminated normally; 2 if stopped; 3 if terminated by
a signal; 4 if continued
    int fd[2];
    struct Pcb *next;
    struct Pcb *prev;
    struct Pcb *next_child;
    struct Pcb *prev_child;
    struct queue *children_pcb;
    struct queue *zombies;
    struct Pcb *parent_pcb;
    int number_of_children;
    int exited_child;
    char *name;
    bool sleeping;
    unsigned int sleep_time;
    bool waiting;
    pid_t waiting_on;
    struct parsed_command *cmd;
    bool state_change;
    struct Pcb *changed_child_pcb;
    bool has_tc;
    bool fg;
    bool reading;
};

struct Pcb *create_pcb(ucontext_t *uc, pid_t pid, pid_t ppid, pid_t pgid, int priority, int status,
struct parsed_command *cmd);
struct Pcb *create_pcb_no_context(pid_t pid, pid_t ppid, pid_t pgid, int priority, int status);
void free_pcb(struct Pcb *pcb);
void add_child_to_parent_q(struct Pcb *parent_pcb, struct Pcb *child_pcb);

void remove_child_pcb(struct Pcb *parent_pcb, struct Pcb *child_pcb);

void delete_child_pcb(struct Pcb *parent_pcb, struct Pcb *child_pcb);

void add_child_to_parent_zombies(struct Pcb *parent_pcb, struct Pcb *child_pcb);


#endif
```

4.21 PennFAT.c File Reference

```
#include "PennFAT.h"
#include <fcntl.h>
#include <sys/mman.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include "parser.h"
#include <errno.h>
#include <time.h>
#include <sys/stat.h>
```

Data Structures

- struct bgNode
- struct jobQueue

Macros

- #define INPUT_SIZE 10000

Functions

- void mkfs (char ∗fs_name, int blocks_in_fat, int block_size_config)
- bool check_allzero (char ∗entry)
- void mount (char ∗fs_name)
- void unmount ()
- int contain_file (char ∗filename)
- void create_newfile (char ∗fn)
- void touch (char ∗∗file_name, int fsize)
- void update_dir_entry (struct Root_directory_entry ∗entry, int delete)
- void print_status ()
- int find_available_block ()
- void rm (char ∗fn, int delete)
- void update_fat (int cur, int block)
- void data_write (char ∗fn, char ∗data, int write_size)
- void cat_a (char ∗∗cmd, int filenum, int out)
- void cat_file (char ∗∗cmd, int filenum, int out)
- void cat_w (char ∗∗cmd, int filenum, int out)
- void copy (char ∗source, char ∗dest1, bool source_os, bool dest_os)
- void cat (char ∗∗cm, int filenum)
- void ls ()
- void chmod1 (char ∗filename, char ∗perm)
- void mv (char ∗filename, char ∗name)
- char ∗ stringcpy (char ∗destination, const char ∗source)
- struct bgNode ∗ creatNode (int p, int pg, char ∗c, int a, int state)
- struct jobQueue ∗ createQueue ()
- void enQueue (struct jobQueue ∗q, int p, int pg, char ∗c, int a, int state)
- void deQueue (struct jobQueue ∗q)
- void removeQueue (struct jobQueue ∗q, int pg)
- void freeq (struct jobQueue ∗q)
- void sig_handler (int signo)
- void INThandler (int signo)
- void sigtstp_handler (int sig)
- int check_finished (struct jobQueue ∗q, int asyn)
- void check_redirection (struct parsed_command ∗cmd)
- struct bgNode ∗ search_byjid (int j, struct jobQueue ∗q)
- void bg (char ∗∗comd, struct jobQueue ∗q)
- void fg (char ∗∗comd, struct jobQueue ∗q)
- void jobs (struct jobQueue ∗q)

    jobs no complete!!!!
- void check_current (struct jobQueue ∗q)
- void sigchld_handler (int sig)
- int main (int argc, char ∗argv[ ])

Variables

- struct superblock sb
- struct Root_directory_entry ∗ rd
- int16_t ∗ FAT_block
- int c = 0
- int pid
- int pid1
- int jobid = 0
- struct bgNode ∗ currentjob
- int back
- struct jobQueue ∗ gq

4.21.1 Macro Definition Documentation

4.21.1.1 INPUT_SIZE

```
#define INPUT_SIZE 10000
```

4.21.2 Function Documentation

4.21.2.1 bg()

```
void bg (
            char ** comd,
            struct jobQueue * q )
```

4.21.2.2 cat()

```
void cat (
            char ** cm,
            int filenum )
```

4.21.2.3 cat_a()

```
void cat_a (
            char ** cmd,
            int filenum,
            int out )
```

4.21.2.4 cat_file()

```
void cat_file (
            char ** cmd,
            int filenum,
            int out )
```

start to print

4.21.2.5 cat_w()

```
void cat_w (
             char ** cmd,
             int filenum,
             int out )
```

4.21.2.6 check_allzero()

```
bool check_allzero (
             char * entry )
```

4.21.2.7 check_current()

```
void check_current (
             struct jobQueue * q )
```

4.21.2.8 check_finished()

```
int check_finished (
             struct jobQueue * q,
             int asyn )
```

4.21.2.9 check_redirection()

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

4.21.2.10 chmod1()

```
void chmod1 (
             char * filename,
             char * perm )
```

4.21.2.11 contain_file()

```
int contain_file (
             char * filename )
```

4.21.2.12 copy()

```
void copy (
             char * source,
             char * dest1,
             bool source_os,
             bool dest_os )
```

4.21.2.13 create_newfile()

```
void create_newfile (
             char * fn )
```

f1 f2 f3 f4

4.21.2.14 createQueue()

```
struct jobQueue * createQueue ( )
```

4.21.2.15 creatNode()

```
struct bgNode * creatNode (
             int p,
             int pg,
             char * c,
             int a,
             int state )
```

4.21.2.16 data_write()

```
void data_write (
             char * fn,
             char * data,
             int write_size )
```

1 root dir->filename, firstblock first block->data:first data, FAT_table[firstblock]=nextblock use size to write.

FAT_block update

update fat block in file

appif

cur is the last block

write to data

4.21.2.17 deQueue()

```
void deQueue (
            struct jobQueue * q )
```

4.21.2.18 enQueue()

```
void enQueue (
            struct jobQueue * q,
            int p,
            int pg,
            char * c,
            int a,
            int state )
```

4.21.2.19 fg()

```
void fg (
            char ** comd,
            struct jobQueue * q )
```

4.21.2.20 find_available_block()

```
int find_available_block ( )
```

4.21.2.21 freeq()

```
void freeq (
            struct jobQueue * q )
```

4.21.2.22 INThandler()

```
void INThandler (
            int signo )
```

4.21.2.23 jobs()

```
void jobs (
             struct jobQueue * q )
```

jobs no complete!!!!

4.21.2.24 ls()

```
void ls ( )
```

4.21.2.25 main()

```
int main (
             int argc,
             char * argv[] )
```

not first time

not last time

4.21.2.26 mkfs()

```
void mkfs (
             char * fs_name,
             int blocks_in_fat,
             int block_size_config )
```

4.21.2.27 mount()

```
void mount (
             char * fs_name )
```

rdsize wrong!!

4.21.2.28 mv()

```
void mv (
             char * filename,
             char * name )
```

4.21.2.29 print_status()

```
void print_status ( )
```

4.21.2.30 removeQueue()

```
void removeQueue (
             struct jobQueue * q,
             int pg )
```

4.21.2.31 rm()

```
void rm (
             char * fn,
             int delete )
```

4.21.2.32 search_byjid()

```
struct bgNode * search_byjid (
             int j,
             struct jobQueue * q )
```

4.21.2.33 sig_handler()

```
void sig_handler (
             int signo )
```

4.21.2.34 sigchld_handler()

```
void sigchld_handler (
             int sig )
```

4.21.2.35 sigtstp_handler()

```
void sigtstp_handler (
             int sig )
```

4.21.2.36 stringcpy()

```
char * stringcpy (
             char * destination,
             const char * source )
```

4.21.2.37 touch()

```
void touch (
             char ** file_name,
             int fsize )
```

check if the file already exist(check rd)

4.21.2.38 unmount()

```
void unmount ( )
```

4.21.2.39 update_dir_entry()

```
void update_dir_entry (
             struct Root_directory_entry * entry,
             int delete )
```

3 FAT[1]=2 FAT[2]=3 FAT[3]=-1

4.21.2.40 update_fat()

```
void update_fat (
             int cur,
             int block )
```

4.21.3 Variable Documentation

4.21.3.1 back

```
int back
```

4.21.3.2 c

```
int c = 0
```

4.21.3.3 currentjob

```
struct bgNode* currentjob
```

4.21.3.4 FAT_block

```
int16_t* FAT_block
```

4.21.3.5 gq

```
struct jobQueue* gq
```

4.21.3.6 jobid

```
int jobid = 0
```

4.21.3.7 pid

```
int pid
```

4.21.3.8 pid1

```
int pid1
```

4.21.3.9 rd

```
struct Root_directory_entry* rd
```

4.21.3.10 sb

```
struct superblock sb
```

4.22 PennFAT.h File Reference

```
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
```

Data Structures

- struct superblock

Functions

- struct __attribute__ ((__packed__)) Root_directory_entry
- void mkfs (char ∗fs_name, int blocks_in_fat, int block_size_config)
- void update_dir_entry (struct Root_directory_entry ∗entry, int delete)
- void update_fat (int cur, int block)
- void mount (char ∗fs_name)
- void unmount ()
- void touch (char ∗∗file_name, int fsize)
- void data_write (char ∗fn, char ∗data, int write_size)
- void copy (char ∗source, char ∗dest1, bool source_os, bool dest_os)
- void cat (char ∗∗cm, int filenum)
- void ls ()
- void chmod1 (char ∗filename, char ∗perm)
- void mv (char ∗filename, char ∗name)
- void print_status ()
- void rm (char ∗fn, int delete)
- int contain_file (char ∗filename)
- bool check_allzero (char ∗entry)
- int find_available_block ()

4.22.1 Function Documentation

4.22.1.1 __attribute__()

```
struct __attribute__ (
             (__packed__)  )
```

4.22.1.2 cat()

```
void cat (
             char ** cm,
             int filenum )
```

4.22.1.3 check_allzero()

```
bool check_allzero (
             char * entry )
```

4.22.1.4 chmod1()

```
void chmod1 (
             char * filename,
             char * perm )
```

4.22.1.5 contain_file()

```
int contain_file (
             char * filename )
```

4.22.1.6 copy()

```
void copy (
             char * source,
             char * dest1,
             bool source_os,
             bool dest_os )
```

4.22.1.7 data_write()

```
void data_write (
            char * fn,
            char * data,
            int write_size )
```

1 root dir->filename, firstblock first block->data:first data, FAT_table[firstblock]=nextblock use size to write.

FAT_block update

update fat block in file

appif

cur is the last block

write to data

1 root dir->filename, firstblock first block->data:first data, FAT_table[firstblock]=nextblock use size to write.

FAT_block update

update fat block in file

appif

cur is the last block

write to data

4.22.1.8 find_available_block()

```
int find_available_block ( )
```

4.22.1.9 ls()

```
void ls ( )
```

4.22.1.10 mkfs()

```
void mkfs (
            char * fs_name,
            int blocks_in_fat,
            int block_size_config )
```

4.22.1.11 mount()

```
void mount (
             char * fs_name )
```

rdsize wrong!!

4.22.1.12 mv()

```
void mv (
             char * filename,
             char * name )
```

4.22.1.13 print_status()

```
void print_status ( )
```

4.22.1.14 rm()

```
void rm (
             char * fn,
             int delete )
```

4.22.1.15 touch()

```
void touch (
             char ** file_name,
             int fsize )
```

check if the file already exist(check rd)

check if the file already exist(check rd)

4.22.1.16 unmount()

```
void unmount ( )
```

4.22.1.17 update_dir_entry()

```
void update_dir_entry (
            struct Root_directory_entry * entry,
            int delete )
```

3 FAT[1]=2 FAT[2]=3 FAT[3]=-1

4.22.1.18 update_fat()

```
void update_fat (
            int cur,
            int block )
```

4.23 PennFAT.h

Go to the documentation of this file.

```
#ifndef PENNFAT_H
#define PENNFAT_H

#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>


struct superblock {
    int fd;
    int Block_size; //total amount of blocks of virtual disk
    //int16_t RdirIndex; // root directory block index
    int DBoffset; // data block start index
    int num_data_Blocks; // amount of data blocks
    int num_FATBlocks; // number of blocks for FAT
    int FAT_tablesize;
    int rdsize;

};
struct __attribute__((__packed__)) Root_directory_entry {
    char name[32];
    uint32_t size;
    uint16_t firstBlock;
    uint8_t type;
    uint8_t perm;
    time_t mtime;
    char remain[16];
    int cursor;

};

void mkfs(char *fs_name, int blocks_in_fat,int block_size_config);
void update_dir_entry(struct Root_directory_entry *entry, int delete);
void update_fat(int cur,int block);
void mount(char *fs_name);
void unmount();
void touch(char **file_name, int fsize);
void data_write(char *fn, char *data, int write_size);
void copy(char *source, char *dest1, bool source_os, bool dest_os);
void cat(char **cm, int filenum);
void ls();
void chmod1(char *filename, char *perm);
void mv(char *filename, char *name);
void print_status();
void rm(char *fn, int delete);
int contain_file(char *filename);
bool check_allzero(char *entry);
int find_available_block();


#endif
```

4.24 queue.c File Reference

```
#include "queue.h"
#include <stdio.h>
#include "kernel.h"
```

Functions

- struct queue ∗ create_queue ()

  Allocates memory for a queue of PCBs.
- void push (struct queue ∗q, struct Pcb ∗new_pcb)
- void push_to_front (struct queue ∗q, struct Pcb ∗new_pcb)
- struct Pcb ∗ pop (struct queue ∗q)

  Pops and returns the Pcb struct pointer of the process at the front of the queue.
- struct Pcb ∗ pop_child (struct queue ∗q)
- void print_queue (struct queue ∗q)
- void print_queue_info (struct queue ∗q)
- void free_queue (struct queue ∗q)

  Frees all PCBs in the queue.
- void free_child_queue (struct queue ∗q)

  Frees all PCBs in the children queue.
- struct Pcb ∗ front (struct queue ∗q)

  Returns pointer to front element.
- bool is_empty (struct queue ∗q)
- struct Pcb ∗ get_pcb_with_pid (struct queue ∗q, pid_t pid)
- void get_pcbs_with_pgid_in_queue (struct queue ∗q, pid_t pgid, struct queue ∗result)
- void remove_pcb (struct queue ∗q, struct Pcb ∗p)

  Removes process from the queue but does not free the memory.
- struct Pcb ∗ get_child_pcb_with_pid (struct queue ∗q, pid_t pid)

  Looks for the PCB in children queue of parent PCB.

4.24.1 Function Documentation

4.24.1.1 create_queue()

```
struct queue * create_queue ( )
```

Allocates memory for a queue of PCBs.

Returns

Returns the pointer to the queue struct on success and NULL on failure

4.24.1.2 free_child_queue()

```
void free_child_queue (
            struct queue * q )
```

Frees all PCBs in the children queue.

Parameters

q	Reference to queue

4.24.1.3 free_queue()

```
void free_queue (
            struct queue * q )
```

Frees all PCBs in the queue.

Parameters

q	Reference to queue

4.24.1.4 front()

```
struct Pcb * front (
            struct queue * q )
```

Returns pointer to front element.

Parameters

q	Reference to queue

4.24.1.5 get_child_pcb_with_pid()

```
struct Pcb * get_child_pcb_with_pid (
            struct queue * q,
            pid_t pid )
```

Looks for the PCB in children queue of parent PCB.

Parameters

q	Children queue
pid	PID of child process

4.24.1.6 get_pcb_with_pid()

```
struct Pcb * get_pcb_with_pid (
            struct queue * q,
            pid_t pid )
```

4.24.1.7 get_pcbs_with_pgid_in_queue()

```
void get_pcbs_with_pgid_in_queue (
            struct queue * q,
            pid_t pgid,
            struct queue * result )
```

4.24.1.8 is_empty()

```
bool is_empty (
            struct queue * q )
```

4.24.1.9 pop()

```
struct Pcb * pop (
            struct queue * q )
```

Pops and returns the Pcb struct pointer of the process at the front of the queue.

Parameters

q	Queue of PCBs

4.24.1.10 pop_child()

```
struct Pcb * pop_child (
            struct queue * q )
```

4.24.1.11 print_queue()

```
void print_queue (
            struct queue * q )
```

4.24.1.12 print_queue_info()

```
void print_queue_info (
            struct queue * q )
```

4.24.1.13 push()

```
void push (
            struct queue * q,
            struct Pcb * new_pcb )
```

4.24.1.14 push_to_front()

```
void push_to_front (
            struct queue * q,
            struct Pcb * new_pcb )
```

4.24.1.15 remove_pcb()

```
void remove_pcb (
            struct queue * q,
            struct Pcb * p )
```

Removes process from the queue but does not free the memory.

Parameters

q	Queue from which process is to be removed
p	Pointer to the PCB of the process which is to be removed from queue q

4.25 queue.h File Reference

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

Data Structures

- struct queue

Functions

- struct queue ∗ create_queue ()

  Allocates memory for a queue of PCBs.
- void push (struct queue ∗q, struct Pcb ∗new_pcb)
- struct Pcb ∗ pop (struct queue ∗q)

  Pops and returns the Pcb struct pointer of the process at the front of the queue.
- void print_queue (struct queue ∗q)
- void print_queue_info (struct queue ∗q)
- void free_queue (struct queue ∗q)

  Frees all PCBs in the queue.
- void free_child_queue (struct queue ∗q)

  Frees all PCBs in the children queue.
- struct Pcb ∗ front (struct queue ∗q)

  Returns pointer to front element.
- bool is_empty (struct queue ∗q)
- struct Pcb ∗ get_pcb_with_pid (struct queue ∗q, pid_t pid)
- struct Pcb ∗ get_child_pcb_with_pid (struct queue ∗q, pid_t pid)

  Looks for the PCB in children queue of parent PCB.
- void get_pcbs_with_pgid_in_queue (struct queue ∗q, pid_t pgid, struct queue ∗result)
- void add_to_blocked_list (struct Pcb ∗p)
- void remove_pcb (struct queue ∗q, struct Pcb ∗p)

  Removes process from the queue but does not free the memory.

Variables

- struct Pcb ∗ blocked_list

4.25.1 Function Documentation

4.25.1.1 add_to_blocked_list()

```
void add_to_blocked_list (
            struct Pcb * p )
```

4.25.1.2 create_queue()

```
struct queue * create_queue ( )
```

Allocates memory for a queue of PCBs.

Returns

Returns the pointer to the queue struct on success and NULL on failure

4.25.1.3 free_child_queue()

```
void free_child_queue (
            struct queue * q )
```

Frees all PCBs in the children queue.

Parameters

q	Reference to queue

4.25.1.4 free_queue()

```
void free_queue (
            struct queue * q )
```

Frees all PCBs in the queue.

Parameters

q	Reference to queue

4.25.1.5 front()

```
struct Pcb * front (
            struct queue * q )
```

Returns pointer to front element.

Parameters

q	Reference to queue

4.25.1.6 get_child_pcb_with_pid()

```
struct Pcb * get_child_pcb_with_pid (
            struct queue * q,
            pid_t pid )
```

Looks for the PCB in children queue of parent PCB.

Parameters

q	Children queue
pid	PID of child process

4.25.1.7 get_pcb_with_pid()

```
struct Pcb * get_pcb_with_pid (
             struct queue * q,
             pid_t pid )
```

4.25.1.8 get_pcbs_with_pgid_in_queue()

```
void get_pcbs_with_pgid_in_queue (
             struct queue * q,
             pid_t pgid,
             struct queue * result )
```

4.25.1.9 is_empty()

```
bool is_empty (
             struct queue * q )
```

4.25.1.10 pop()

```
struct Pcb * pop (
             struct queue * q )
```

Pops and returns the Pcb struct pointer of the process at the front of the queue.

Parameters

q	Queue of PCBs

4.25.1.11 print_queue()

```
void print_queue (
             struct queue * q )
```

4.25.1.12 print_queue_info()

```
void print_queue_info (
             struct queue * q )
```

4.25.1.13 push()

```
void push (
            struct queue * q,
            struct Pcb * new_pcb )
```

4.25.1.14 remove_pcb()

```
void remove_pcb (
            struct queue * q,
            struct Pcb * p )
```

Removes process from the queue but does not free the memory.

Parameters

q	Queue from which process is to be removed
p	Pointer to the PCB of the process which is to be removed from queue q

4.25.2 Variable Documentation

4.25.2.1 blocked_list

```
struct Pcb* blocked_list [extern]
```

4.26 queue.h

Go to the documentation of this file.

```
#ifndef Q_H
#define Q_H

#include "pcb.h"
#include <signal.h>
struct queue
{
    struct Pcb *head;
    struct Pcb *tail;
    int size;
};

struct queue *create_queue();

extern struct Pcb *blocked_list;

void push(struct queue *q, struct Pcb *new_pcb);

struct Pcb *pop(struct queue *q); // Removes and returns front element
void print_queue(struct queue *q);
void print_queue_info(struct queue *q);

void free_queue(struct queue *q);

void free_child_queue(struct queue *q);


struct Pcb *front(struct queue *q);


bool is_empty(struct queue *q);

struct Pcb *get_pcb_with_pid(struct queue *q, pid_t pid);

struct Pcb *get_child_pcb_with_pid(struct queue *q, pid_t pid);


void get_pcbs_with_pgid_in_queue(struct queue *q, pid_t pgid, struct queue *result);
// void push_child(struct Pcb *parent_pcb, struct Pcb *child_pcb);

void add_to_blocked_list(struct Pcb *p);

void remove_pcb(struct queue *q, struct Pcb *p);
#endif
```

4.27 shell.c File Reference

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

Macros

- #define MAX_LEN 512

Functions

- void prompt ()
- void newline ()
- void handle_sigint (int signal)
- void handle_sigtstp (int signal)
- int get_arg_count (struct parsed_command ∗cmd)
- void get_args_from_cmd (struct parsed_command ∗cmd, int arg_count, char ∗args[ ])
- void non_interactive ()
- int handle_command (struct parsed_command ∗cmd, char ∗raw_cmd, void(∗func)(), char ∗args[ ])
- int parse (char ∗str)
- void interactive_shell ()
- void fg_pipeline (struct Job ∗j)
- void print_finished ()
- void poll_background_jobs ()
- int main (int argc, char ∗argv[ ])

Variables

- int mode = INTERACTIVE
- int job_id = 1
- bool start = true
- struct j_queue ∗ jobs_list

4.27.1 Macro Definition Documentation

4.27.1.1 MAX_LEN

```
#define MAX_LEN 512
```

4.27.2 Function Documentation

4.27.2.1 fg_pipeline()

```
void fg_pipeline (
            struct Job * j )
```

4.27.2.2 get_arg_count()

```
int get_arg_count (
            struct parsed_command * cmd )
```

4.27.2.3 get_args_from_cmd()

```
void get_args_from_cmd (
            struct parsed_command * cmd,
            int arg_count,
            char * args[] )
```

4.27.2.4 handle_command()

```
int handle_command (
            struct parsed_command * cmd,
            char * raw_cmd,
            void(*)() func,
            char * args[] )
```

4.27.2.5 handle_sigint()

```
void handle_sigint (
             int signal )
```

4.27.2.6 handle_sigtstp()

```
void handle_sigtstp (
             int signal )
```

4.27.2.7 interactive_shell()

```
void interactive_shell ( )
```

4.27.2.8 main()

```
int main (
             int argc,
             char * argv[] )
```

4.27.2.9 newline()

```
void newline ( )
```

4.27.2.10 non_interactive()

```
void non_interactive ( )
```

4.27.2.11 parse()

```
int parse (
             char * str )
```

4.27.2.12 poll_background_jobs()

```
void poll_background_jobs ( )
```

4.27.2.13 print_finished()

```
void print_finished ( )
```

4.27.2.14 prompt()

```
void prompt ( )
```

4.27.3 Variable Documentation

4.27.3.1 job_id

```
int job_id = 1
```

4.27.3.2 jobs_list

```
struct j_queue* jobs_list
```

4.27.3.3 mode

```
int mode = INTERACTIVE
```

4.27.3.4 start

```
bool start = true
```

4.28 shell.h File Reference

```
#include "user.h"
#include "Job.h"
#include "j_queue.h"
#include "f.h"
#include "utils.h"
```

Enumerations

- enum SHELL_MODE { INTERACTIVE , SHELL_SCRIPT }

Functions

- void print_finished ()
- void poll_background_jobs ()
- void fg_pipeline (struct Job ∗j)
- int parse (char ∗str)

4.28.1 Enumeration Type Documentation

4.28.1.1 SHELL_MODE

```
enum SHELL_MODE
```

Enumerator

INTERACTIVE	
SHELL_SCRIPT	

4.28.2 Function Documentation

4.28.2.1 fg_pipeline()

```
void fg_pipeline (
            struct Job ∗ j )
```

4.28.2.2 parse()

```
int parse (
             char * str )
```

4.28.2.3 poll_background_jobs()

```
void poll_background_jobs ( )
```

4.28.2.4 print_finished()

```
void print_finished ( )
```

4.29 shell.h

Go to the documentation of this file.

```
#ifndef SHELL_H
#define SHELL_H

#include "user.h"
#include "Job.h"
#include "j_queue.h"
#include "f.h"
#include "utils.h"

enum SHELL_MODE {
    INTERACTIVE,
    SHELL_SCRIPT
};

void print_finished();
void poll_background_jobs();
void fg_pipeline(struct Job *j);
int parse(char *str);

#endif
```

4.30 user.c File Reference

```
#include "user.h"
#include "kernel.h"
```

Functions

- bool W_WIFEXITED (int status)
- bool W_WIFSTOPPED (int status)
- bool W_WIFSIGNALED (int status)
- bool W_WIFCONTINUED (int status)
- int p_nice (pid_t pid, int priority)

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

  Put the calling process to sleep for a specified number of ticks.
- void sleeps (char ∗ticks)

  Wrapper function for p_sleep that takes a string argument representing the number of ticks to sleep for.
- void echo (char ∗argStr)

  Print a string argument to the console and terminate the calling process.
- void kill_as_process (pid_t pid, int sig)

  Send a signal to a process with a given PID and terminate the calling process.
- void print_all_process_info ()

  Print information about all processes in the system to the console and terminate the calling process.
- pid_t p_spawn (void(∗func)(), char ∗argv[ ], int fd0, int fd1, bool fg)
- void wait_log (struct Pcb ∗calling_pcb)
- pid_t p_waitpid (pid_t pid, int ∗wstatus, bool nohang)
- pid_t p_waitpid_old (pid_t pid, int ∗wstatus, bool nohang)
- int p_kill (pid_t pid, int sig)

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

  Exits thread unconditionally.
- int p_tcset (pid_t pid)

  Gives terminal control to process specified by pid.
- void custom_read (int fd, char ∗buf, ssize_t n)
- void dummy (char ∗str)

  Dummy function for testing.
- void spawn_shell ()

  Creates the shell process.
- pid_t p_getpid ()

  Returns active process's PID.
- void p_switch_to_scheduler ()

  Saves active context and switches to scheduler context.
- void p_boot_kernel ()
- void zombie_child ()

  Function that does nothing, used to create a zombie child process.
- void zombify ()

  Create a zombie child process that does nothing and keep the current process running indefinitely.
- void orphan_child ()

  Function that loops indefinitely, used to create an orphan child process.
- void orphanify ()

  Create an orphan child process that loops indefinitely and return immediately.

4.30.1 Function Documentation

4.30.1.1 custom_read()

```
void custom_read (
             int fd,
             char * buf,
             ssize_t n )
```

4.30.1.2 dummy()

```
void dummy (
             char * str )
```

Dummy function for testing.

Parameters

str	

4.30.1.3 echo()

```
void echo (
             char * argStr )
```

Print a string argument to the console and terminate the calling process.

Parameters

argStr	The string to print to the console

4.30.1.4 kill_as_process()

```
void kill_as_process (
             pid_t pid,
             int sig )
```

Send a signal to a process with a given PID and terminate the calling process.

Parameters

pid	The PID of the process to send the signal to
sig	The signal to send

4.30.1.5 orphan_child()

```
void orphan_child ( )
```

Function that loops indefinitely, used to create an orphan child process.

4.30.1.6 orphanify()

```
void orphanify ( )
```

Create an orphan child process that loops indefinitely and return immediately.

4.30.1.7 p_boot_kernel()

```
void p_boot_kernel ( )
```

4.30.1.8 p_exit()

```
void p_exit (
             void )
```

Exits thread unconditionally.

Parameters

None	

4.30.1.9 p_getpid()

```
pid_t p_getpid ( )
```

Returns active process's PID.

Returns

PID of current process

4.30.1.10 p_kill()

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

sends the signal sig to the thread referenced by pid.

Parameters

pid	pid of process to which we wish to send signal
sig	signal to be sent to process referenced by pid

Returns

It returns 0 on success, -1 on error.

4.30.1.11 p_nice()

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

Change the priority of a process.

Parameters

pid	The process ID of the target process
priority	The new priority of the target process

Returns

0 on success, -1 if the process with the given PID does not exist

4.30.1.12 p_sleep()

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

Put the calling process to sleep for a specified number of ticks.

Parameters

ticks	The number of timer ticks to sleep for

4.30.1.13 p_spawn()

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

4.30.1.14 p_switch_to_scheduler()

```
void p_switch_to_scheduler ( )
```

Saves active context and switches to scheduler context.

4.30.1.15 p_tcset()

```
int p_tcset (
            pid_t pid )
```

Gives terminal control to process specified by pid.

Parameters

pid	Process PID

4.30.1.16 p_waitpid()

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

4.30.1.17 p_waitpid_old()

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

4.30.1.18 print_all_process_info()

```
void print_all_process_info ( )
```

Print information about all processes in the system to the console and terminate the calling process.

4.30.1.19 sleeps()

```
void sleeps (
             char * ticks )
```

Wrapper function for p_sleep that takes a string argument representing the number of ticks to sleep for.

Parameters

ticks	A string representing the number of timer ticks to sleep for

4.30.1.20 spawn_shell()

```
void spawn_shell ( )
```

Creates the shell process.

4.30.1.21 W_WIFCONTINUED()

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

4.30.1.22 W_WIFEXITED()

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

4.30.1.23 W_WIFSIGNALED()

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

4.30.1.24 W_WIFSTOPPED()

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

4.30.1.25 wait_log()

```
void wait_log (
            struct Pcb * calling_pcb )
```

4.30.1.26 zombie_child()

```
void zombie_child ( )
```

Function that does nothing, used to create a zombie child process.

4.30.1.27 zombify()

```
void zombify ( )
```

Create a zombie child process that does nothing and keep the current process running indefinitely.

4.31 user.h File Reference

```
#include "pcb.h"
#include "queue.h"
#include "errors.h"
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <ucontext.h>
#include <unistd.h>
#include <signal.h>
```

Enumerations

- enum S_SIGNALS { S_SIGTERM , S_SIGCONT , S_SIGSTOP , S_SIGCHLD }
- enum CHANGE { CHANGED , NO_CHANGE }

Functions

- bool W_WIFEXITED (int status)
- bool W_WIFSTOPPED (int status)
- bool W_WIFSIGNALED (int status)
- pid_t p_spawn (void(∗func)(), char ∗argv[ ], int fd0, int fd1, bool fg)
- pid_t p_waitpid (pid_t pid, int ∗wstatus, bool nohang)
- int p_kill (pid_t pid, int sig)

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

    Exits thread unconditionally.
- int p_nice (pid_t pid, int priority)

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

    Put the calling process to sleep for a specified number of ticks.
- void sleeps (char ∗ticks)

    Wrapper function for p_sleep that takes a string argument representing the number of ticks to sleep for.
- void echo (char ∗argStr)

    Print a string argument to the console and terminate the calling process.
- void kill_as_process (pid_t pid, int sig)

    Send a signal to a process with a given PID and terminate the calling process.
- void print_all_process_info ()

    Print information about all processes in the system to the console and terminate the calling process.
- void dummy (char ∗str)

    Dummy function for testing.
- void spawn_shell ()

    Creates the shell process.
- void interactive_shell ()
- int p_tcset (pid_t pid)

    Gives terminal control to process specified by pid.
- pid_t p_getpid ()

    Returns active process's PID.
- void k_block_sigset (sigset_t ∗mask)
- void k_unblock_sigset (sigset_t ∗mask)
- void p_switch_to_scheduler ()

    Saves active context and switches to scheduler context.
- void p_boot_kernel ()
- void zombie_child ()

    Function that does nothing, used to create a zombie child process.
- void zombify ()

    Create a zombie child process that does nothing and keep the current process running indefinitely.
- void orphan_child ()

    Function that loops indefinitely, used to create an orphan child process.
- void orphanify ()

    Create an orphan child process that loops indefinitely and return immediately.

4.31.1 Enumeration Type Documentation

4.31.1.1 CHANGE

```
enum CHANGE
```

Enumerator

CHANGED	
NO_CHANGE	

4.31.1.2 S_SIGNALS

```
enum S_SIGNALS
```

Enumerator

S_SIGTERM	
S_SIGCONT	
S_SIGSTOP	
S_SIGCHLD	

4.31.2 Function Documentation

4.31.2.1 dummy()

```
void dummy (
             char * str )
```

Dummy function for testing.

Parameters

str	

4.31.2.2 echo()

```
void echo (
             char * argStr )
```

Print a string argument to the console and terminate the calling process.

Parameters

argStr	The string to print to the console

4.31.2.3 interactive_shell()

```
void interactive_shell ( )
```

4.31.2.4 k_block_sigset()

```
void k_block_sigset (
             sigset_t * mask )
```

4.31.2.5 k_unblock_sigset()

```
void k_unblock_sigset (
             sigset_t * mask )
```

4.31.2.6 kill_as_process()

```
void kill_as_process (
             pid_t pid,
             int sig )
```

Send a signal to a process with a given PID and terminate the calling process.

Parameters

pid	The PID of the process to send the signal to
sig	The signal to send

4.31.2.7 orphan_child()

```
void orphan_child ( )
```

Function that loops indefinitely, used to create an orphan child process.

4.31.2.8 orphanify()

```
void orphanify ( )
```

Create an orphan child process that loops indefinitely and return immediately.

4.31.2.9 p_boot_kernel()

```
void p_boot_kernel ( )
```

4.31.2.10 p_exit()

```
void p_exit (
             void )
```

Exits thread unconditionally.

Parameters

None	

4.31.2.11 p_getpid()

```
pid_t p_getpid ( )
```

Returns active process's PID.

Returns

PID of current process

4.31.2.12 p_kill()

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

sends the signal sig to the thread referenced by pid.

Parameters

pid	pid of process to which we wish to send signal
sig	signal to be sent to process referenced by pid

Returns

It returns 0 on success, -1 on error.

4.31.2.13 p_nice()

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

Change the priority of a process.

Parameters

pid	The process ID of the target process
priority	The new priority of the target process

Returns

0 on success, -1 if the process with the given PID does not exist

4.31.2.14 p_sleep()

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

Put the calling process to sleep for a specified number of ticks.

Parameters

ticks	The number of timer ticks to sleep for

4.31.2.15 p_spawn()

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

4.31.2.16 p_switch_to_scheduler()

```
void p_switch_to_scheduler ( )
```

Saves active context and switches to scheduler context.

4.31.2.17 p_tcset()

```
int p_tcset (
            pid_t pid )
```

Gives terminal control to process specified by pid.

Parameters

pid	Process PID

4.31.2.18 p_waitpid()

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

4.31.2.19 print_all_process_info()

```
void print_all_process_info ( )
```

Print information about all processes in the system to the console and terminate the calling process.

4.31.2.20 sleeps()

```
void sleeps (
            char * ticks )
```

Wrapper function for p_sleep that takes a string argument representing the number of ticks to sleep for.

Parameters

ticks	A string representing the number of timer ticks to sleep for

4.31.2.21 spawn_shell()

```
void spawn_shell ( )
```

Creates the shell process.

4.31.2.22 W_WIFEXITED()

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

4.31.2.23 W_WIFSIGNALED()

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

4.31.2.24 W_WIFSTOPPED()

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

4.31.2.25 zombie_child()

```
void zombie_child ( )
```

Function that does nothing, used to create a zombie child process.

4.31.2.26 zombify()

```
void zombify ( )
```

Create a zombie child process that does nothing and keep the current process running indefinitely.

4.32 user.h

Go to the documentation of this file.

```
#ifndef USER_H
#define USER_H

#include "pcb.h"
#include "queue.h"
#include "errors.h"
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <ucontext.h>
#include <unistd.h>
#include <signal.h>

// extern struct Pcb *current_pcb;
// extern struct queue *q;
// extern struct queue *queues[3];

enum S_SIGNALS {
    S_SIGTERM,
    S_SIGCONT,
    S_SIGSTOP,
    S_SIGCHLD
};


enum CHANGE {
    CHANGED,
    NO_CHANGE
};

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

pid_t p_spawn(void (*func)(), char *argv[], int fd0, int fd1, bool fg);
pid_t p_waitpid(pid_t pid, int *wstatus, bool nohang);

int p_kill(pid_t pid, int sig);

void p_exit(void);

int p_nice(pid_t pid, int priority);

void p_sleep(unsigned int ticks);

void sleeps(char *ticks);

void echo(char *argStr);

void kill_as_process(pid_t pid, int sig);

void print_all_process_info();

void dummy(char *str);

void spawn_shell();

void interactive_shell();

static bool has_terminal_control = true;
static pid_t tc_pid = 1;


int p_tcset(pid_t pid);

pid_t p_getpid();

static sigset_t intmask;

void k_block_sigset(sigset_t *mask);
void k_unblock_sigset(sigset_t *mask);

void p_switch_to_scheduler();

void p_boot_kernel();

// zombify and orphanify
void zombie_child();

void zombify();

void orphan_child();

void orphanify();
#endif
```

4.33 utils.c File Reference

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

Functions

- void memsetter (char ∗buf, char val, unsigned long int len)
- void print_out (char ∗str, int len)
- void debug (char ∗msg)
- int extractInt (char str[ ])
- void string_copy (const char ∗from, char ∗to)
- int myStrNCpy (char ∗dest, char ∗src, int n)

4.33.1 Function Documentation

4.33.1.1 debug()

```
void debug (
             char * msg )
```

4.33.1.2 extractInt()

```
int extractInt (
             char str[] )
```

4.33.1.3 memsetter()

```
void memsetter (
             char * buf,
             char val,
             unsigned long int len )
```

4.33.1.4 myStrNCpy()

```
int myStrNCpy (
             char * dest,
             char * src,
             int n )
```

4.33.1.5 print_out()

```
void print_out (
             char * str,
             int len )
```

4.33.1.6 string_copy()

```
void string_copy (
             const char * from,
             char * to )
```

4.34 utils.h File Reference

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

Macros

- #define DEBUG 0

Functions

- void memsetter (char ∗buf, char val, unsigned long int len)
- void print_out (char ∗str, int len)
- void debug (char ∗msg)
- void string_copy (const char ∗from, char ∗to)
- int extractInt (char str[ ])
- int myStrNCpy (char ∗dest, char ∗src, int n)

4.34.1 Macro Definition Documentation

4.34.1.1 DEBUG

```
#define DEBUG 0
```

4.34.2 Function Documentation

4.34.2.1 debug()

```
void debug (
            char * msg )
```

4.34.2.2 extractInt()

```
int extractInt (
            char str[] )
```

4.34.2.3 memsetter()

```
void memsetter (
            char * buf,
            char val,
            unsigned long int len )
```

4.34.2.4 myStrNCpy()

```
int myStrNCpy (
            char * dest,
            char * src,
            int n )
```

4.34.2.5 print_out()

```
void print_out (
            char * str,
            int len )
```

4.34.2.6 string_copy()

```
void string_copy (
              const char * from,
              char * to )
```

4.35 utils.h

Go to the documentation of this file.

```
#ifndef UTILS_H
#define UTILS_H
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "parser.h"

#define DEBUG 0

void memsetter(char *buf, char val, unsigned long int len);
void print_out(char *str, int len);
void debug(char *msg);

void string_copy(const char *from, char *to);
int extractInt(char str[]);
int myStrNCpy(char *dest, char *src, int n);


#endif
```

Index