pennos

Generated by Doxygen 1.9.7

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 directory_entry Struct Reference

```
#include <pennfat.h>
```

Public Attributes

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

3.1.1 Detailed Description

each directory entry stores metadata about another file (including a directory file). PennFAT has a 64-byte fixed directory entry size.

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

- 23sp-pennOS-group-12/src/pennfat/pennfat.h

3.2 FD_TABLE_ENTRY Struct Reference

Public Attributes

- char **file_name** [MAX_NAME_LENGTH]
- int **fd**
- int **status**
- int **mode**
- int **occupied**
- int **file_idx**
- int **dir_idx**
- int **num_read**

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

- 23sp-pennOS-group-12/src/pennfat/syscall.h

3.3 job Struct Reference

A struct to store the information of a job.

```
#include <shellLList.h>
```

Public Attributes

- int processNum
- int pgid
- int jobid
- int status
- int printStatus
- char ∗ cmdContentPtr

3.3.1 Detailed Description

A struct to store the information of a job.

3.3.2 Member Data Documentation

3.3.2.1 cmdContentPtr

```
char∗ job::cmdContentPtr
```

The command content of the job

3.3.2.2 jobid

```
int job::jobid
```

The job id of the job

3.3.2.3 pgid

```
int job::pgid
```

The process group id of the job

3.3.2.4 printStatus

```
int job::printStatus
```

The print status of the job. 0: init, 1: running

3.3.2.5 processNum

```
int job::processNum
```

The number of processes in the job

3.3.2.6 status

```
int job::status
```

The status of the job

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

- 23sp-pennOS-group-12/src/kernel/shellLList.h

3.4 jobList Struct Reference

A struct to store a list of jobs.

```
#include <shellLList.h>
```

Public Attributes

- jobNode ∗ head
- jobNode ∗ tail
- int jobNum

3.4.1 Detailed Description

A struct to store a list of jobs.

3.4.2 Member Data Documentation

3.4.2.1 head

```
jobNode* jobList::head
```

The head jobNode in the jobList

3.4.2.2 jobNum

```
int jobList::jobNum
```

The number of jobs in the jobList

3.4.2.3 tail

```
jobNode* jobList::tail
```

The tail jobNode in the jobList

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

- 23sp-pennOS-group-12/src/kernel/shellLList.h

3.5 jobNode Struct Reference

A struct to store the information of a jobNode.

```
#include <shellLList.h>
```

Public Attributes

- int pgid
- job * jobPtr
- struct jobNode * next

3.5.1 Detailed Description

A struct to store the information of a jobNode.

3.5.2 Member Data Documentation

3.5.2.1 jobPtr

```
job* jobNode::jobPtr
```

Value: the pointer to the job

3.5.2.2 next

```
struct jobNode* jobNode::next
```

The pointer to the next jobNode in the jobList

3.5.2.3 pgid

```
int jobNode::pgid
```

Key (can also use jobPtr->pgid)

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

- 23sp-pennOS-group-12/src/kernel/shellLList.h

3.6 jobSet Struct Reference

A struct to store a set of jobs.

```
#include <shellLList.h>
```

Public Attributes

- struct jobList ∗ jobs
- struct jobList ∗ stoppedJobs
- struct jobList ∗ finishedJobs

3.6.1 Detailed Description

A struct to store a set of jobs.

3.6.2 Member Data Documentation

3.6.2.1 finishedJobs

```
struct jobList* jobSet::finishedJobs
```

The list of finished jobs

3.6.2.2 jobs

```
struct jobList* jobSet::jobs
```

The list of jobs

3.6.2.3 stoppedJobs

```
struct jobList* jobSet::stoppedJobs
```

The list of stopped jobs

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

- 23sp-pennOS-group-12/src/kernel/shellLList.h

3.7 parsed_command Struct Reference

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

Public Attributes

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

3.7.1 Detailed Description

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

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

- 23sp-pennOS-group-12/src/kernel/parser.h

3.8 pcb Struct Reference

The struct for Process Control Block of a process.

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

Public Attributes

- int pid
- int p_level
- int thread_state
- ucontext_t ∗ ucontext
- struct pcb ∗ parent
- struct procList ∗ children
- struct procList ∗ zombie_children
- int waitPid
- int sleep_stop_tick
- char ∗ process_name

3.8.1 Detailed Description

The struct for Process Control Block of a process.

3.8.2 Member Data Documentation

3.8.2.1 children

```
struct procList* pcb::children
```

The pointer to the list of all the children processes of the current process

3.8.2.2 p_level

```
int pcb::p_level
```

The priority level of the process

3.8.2.3 parent

```
struct pcb* pcb::parent
```

The pointer to the pcb of the parent process of the current process

3.8.2.4 pid

```
int pcb::pid
```

The process id of the process

3.8.2.5 process_name

```
char* pcb::process_name
```

The name of the process

3.8.2.6 sleep_stop_tick

```
int pcb::sleep_stop_tick
```

The tick when the process is going to be woken up

3.8.2.7 thread_state

```
int pcb::thread_state
```

The state of the process

3.8.2.8 ucontext

```
ucontext_t* pcb::ucontext
```

The ucontext of the process

3.8.2.9 waitPid

```
int pcb::waitPid
```

The pid of the process that the current process is waiting for

3.8.2.10 zombie_children

```
struct procList* pcb::zombie_children
```

The pointer to the list of all the zombie children processes of the current process

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

- 23sp-pennOS-group-12/src/kernel/pcb.h

3.9 proc Struct Reference

A struct to store the information of a process.

```
#include <shellLList.h>
```

Public Attributes

- int status
- int pid

3.9.1 Detailed Description

A struct to store the information of a process.

3.9.2 Member Data Documentation

3.9.2.1 pid

```
int proc::pid
```

The process id of the process

3.9.2.2 status

```
int proc::status
```

The status of the process

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

- 23sp-pennOS-group-12/src/kernel/shellLList.h

3.10 procList Struct Reference

The struct for a list of process nodes.

`#include <pcb.h>`

Public Attributes

- procNode ∗ head
- procNode ∗ tail
- int procNum

3.10.1 Detailed Description

The struct for a list of process nodes.

3.10.2 Member Data Documentation

3.10.2.1 head

`procNode* procList::head`

The head process node in the process list.

3.10.2.2 procNum

`int procList::procNum`

Number of process nodes in the process list.

3.10.2.3 tail

`procNode* procList::tail`

The tail process node in the process list.

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

- 23sp-pennOS-group-12/src/kernel/pcb.h

3.11 procNode Struct Reference

The struct for the process node inside a procList.

`#include <pcb.h>`

Public Attributes

- int pid
- pcb ∗ pcbPtr
- struct procNode ∗ next

3.11.1 Detailed Description

The struct for the process node inside a procList.

3.11.2 Member Data Documentation

3.11.2.1 next

```
struct procNode* procNode::next
```

The pointer to the next process node in the procList.

3.11.2.2 pcbPtr

```
pcb* procNode::pcbPtr
```

The pointer to the pcb of the process node. Use as value.

3.11.2.3 pid

```
int procNode::pid
```

The pid of the process node. Use as key (can also use procPtr->pgid).

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

- 23sp-pennOS-group-12/src/kernel/pcb.h

3.12 Scheduler Struct Reference

The struct for the scheduler.

```
#include <scheduler.h>
```

Public Attributes

- procList ∗ minus1_procList
- procList ∗ zero_procList
- procList ∗ plus1_procList
- procList ∗ blocked_procList
- procList ∗ zombie_procList
- procList ∗ stopped_procList
- procList ∗ all_procList
- int totalticks
- bool runningLock
- bool terminate
- bool idle
- int blockPid
- ucontext_t ∗ schedulerContext
- pcb ∗ idlePcb

3.12.1 Detailed Description

The struct for the scheduler.

3.12.2 Member Data Documentation

3.12.2.1 all_procList

```
procList* Scheduler::all_procList
```

The list of all the processes.

3.12.2.2 blocked_procList

```
procList* Scheduler::blocked_procList
```

The list of all the processes that are blocked.

3.12.2.3 blockPid

```
int Scheduler::blockPid
```

The pid of the process that is blocked.

3.12.2.4 idle

```
bool Scheduler::idle
```

The flag to indicate if the scheduler is running the idle process.

3.12.2.5 idlePcb

`pcb* Scheduler::idlePcb`

The pcb of the idle process.

3.12.2.6 minus1_procList

`procList* Scheduler::minus1_procList`

The list of all the processes with priority -1.

3.12.2.7 plus1_procList

`procList* Scheduler::plus1_procList`

The list of all the processes with priority 1.

3.12.2.8 runningLock

`bool Scheduler::runningLock`

The lock to prevent the scheduler from running.

3.12.2.9 schedulerContext

`ucontext_t* Scheduler::schedulerContext`

The context of the scheduler.

3.12.2.10 stopped_procList

`procList* Scheduler::stopped_procList`

The list of all the processes that are stopped.

3.12.2.11 terminate

`bool Scheduler::terminate`

The flag to indicate if the scheduler should terminate.

3.12.2.12 totalticks

`int Scheduler::totalticks`

The total number of ticks that the scheduler has run.

3.12.2.13 zero_procList

```
procList* Scheduler::zero_procList
```

The list of all the processes with priority 0.

3.12.2.14 zombie_procList

```
procList* Scheduler::zombie_procList
```

The list of all the processes that are zombie.

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

- 23sp-pennOS-group-12/src/kernel/scheduler.h

Chapter 4

File Documentation

4.1 helperfunc.h

```c
#ifndef HELPERFUNC
#define HELPERFUNC

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

char** copyArgv(char** argv, bool addNewLine);
void free_charArr(char*** arr);
int num_arguments(char** args);
char** parseInputFromScript(char* buf);


#endif
```

4.2 kernel.h

```c
#ifndef KERNEL
#define KERNEL

#include <unistd.h>
#include <signal.h>
#include <sys/time.h> // setitimer
#include <stdlib.h>
#include <fcntl.h>
#include "logger.h"
#include "scheduler.h"
#include "pcb.h"
#include "usignal.h"
#include "uprocess.h"

extern Scheduler* global_scheduler_ptr;

extern pcb* active_pcb_ptr;

extern int process_num;

static const int centisecond = 100000; // 100 ms

void alarmHandler(int signum);

void setAlarmHandler(void);

void setTimer();

pcb* k_process_create(pcb* parent, int p_level, void (*func)(), int argc, char *argv[], char* process_name, int fd0, int fd1);

bool k_process_kill(pcb* proc, int sig);

void k_process_exit();

void k_process_cleanup(pcb* proc);
void k_process_setChildOrphan(pcb* proc);

void shell();

void sigSTPHandler(int sig);

void sigIntHandler(int sig);

#endif
```

4.3 logger.h

```
#ifndef LOGGER
#define LOGGER

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern char *logfile_path;

void log_schedule_event(int ticks, int pid, int priority, char* process_name);

void log_create_event(int ticks, int pid, int priority, char* process_name);

void log_signaled_event(int ticks, int pid, int priority, char* process_name);

void log_exited_event(int ticks, int pid, int priority, char* process_name);

void log_zombie_event(int ticks, int pid, int priority, char* process_name);

void log_orphan_event(int ticks, int pid, int priority, char* process_name);

void log_waited_event(int ticks, int pid, int priority, char* process_name);

void log_nice_event(int ticks, int pid, int priority_old, int priority_new, char* process_name);

void log_blocked_event(int ticks, int pid, int priority, char* process_name);

void log_unblocked_event(int ticks, int pid, int priority, char* process_name);

void log_stopped_event(int ticks, int pid, int priority, char* process_name);

void log_continued_event(int ticks, int pid, int priority, char* process_name);

#endif
```

4.4 parser.h

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

#pragma once

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

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

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

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

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

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

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

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

```

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

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

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

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

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

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

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


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

4.5 pcb.h

```
#ifndef PCB
#define PCB

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <signal.h>
#include <ucontext.h>
#include <threads.h>
#include <unistd.h>
#include <valgrind/valgrind.h>


#define STACK_SIZE 819200

#define READY 0

#define STOPPED 1

#define BLOCKED 2

#define EXITED 3

#define ZOMBIED EXITED

#define ORPHAN 5

#define CLEAN 6

#define SIGKILLED 7

struct procList;

typedef struct pcb {
    int pid;
    int p_level;
    int thread_state;
    ucontext_t* ucontext;
    struct pcb* parent;
    struct procList* children;
    struct procList* zombie_children;
    int waitPid;
    int sleep_stop_tick;
    char* process_name;
} pcb;


typedef struct procNode{
    int pid;
    pcb* pcbPtr;
    struct procNode* next;
} procNode;


typedef struct procList{
    procNode *head;
    procNode *tail;
    int procNum;
} procList;

void setStack(stack_t *stack);

pcb* init_pcb(int pid, pcb* parent, int p_level, void (*func)(), int argc, char *argv[], ucontext_t*
nextContext, char* process_name, int fd0, int fd1, int scale);

void free_pcb(pcb* pcbPtr);

void print_pcb(pcb* pcbPtr);

procList* initprocList();

void freeprocPtr(procNode* procPtr, bool freePCB);

void freeprocList(procList* listPtr, bool freePCB);

void print_procList(procList* procListPtr);

void addNode2procList(procList* listPtr, pcb* pcbPtr);

bool deleteNodeFromList(procList* listPtr, int pid);

procNode* removeNodeFromList(procList* listPtr, int pid);

pcb* find_pcb_in_procList(procList* listPtr, int pid);

void roundRobinUpdate(procList* listPtr);

#endif
```

4.6 scheduler.h

```
#ifndef SCHEDULER
#define SCHEDULER

#include <stdlib.h>   // malloc, free
#include <sys/types.h>
#include "pcb.h"

#define SHELL_PID 1

#define IDLE_PID -1

typedef struct Scheduler {
    procList* minus1_procList;
    procList* zero_procList;
    procList* plus1_procList;
    procList* blocked_procList;
    procList* zombie_procList;
    procList* stopped_procList;
    procList* all_procList;
    int totalticks;
    bool runningLock;
    bool terminate;
    bool idle;
    int blockPid;
    ucontext_t* schedulerContext;
    pcb* idlePcb;
} Scheduler;

Scheduler* init_scheduler(ucontext_t* mainContext);

pcb* init_idlePcb();

void init_schedulerContext(Scheduler* schedulerPtr, ucontext_t* mainContext);

void free_scheduler(Scheduler* schedulerPtr);

void addProc2ReadyQueue(Scheduler* schedulerPtr, pcb* pcbPtr);

pcb* chooseproctoRun_minus1(Scheduler* schedulerPtr);

pcb* chooseproctoRun_zero(Scheduler* schedulerPtr);

pcb* chooseproctoRun_plus1(Scheduler* schedulerPtr);
```

```c

void deleteFromReadyQueue(pcb* pcbPtr);

void checkReadyQueue(procList* procListPtr);

bool checkPcb(pcb* pcbPtr, char* logPrefix, int excludeState, procList* procListPtr);

void checkAllReadyQueue(Scheduler* schedulerPtr);

void checkActivePcb();

void checkZombieQueue(Scheduler* schedulerPtr);

void checkBlockedQueue(Scheduler* schedulerPtr);

bool unblockWaitingParent(pcb* pcbPtr);

bool scheduleTick();

pcb* find_pcb_with_pid(int pid);

void idle_func();

void idle();

#endif
```

4.7 shell.h

```c
#ifndef SHELL
#define SHELL

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <signal.h>
#include "parser.h"
#include "kernel.h"
#include "ufuncs.h"
#include "usignal.h"
#include "shellLList.h"
#include "helperfunc.h"

void shell();

void handleOneLineCmd(char* line, struct parsed_command** cmd, jobSet** bgJobs, job** fgjob);

bool isBuildInCmd(struct parsed_command** cmd);

void handleBuildInCmds(struct parsed_command** cmd, jobSet** bgJobs, job** fgjob, int fd_0, int fd_1);

int handleUserCmds(char** argv, int p_level, int fd_0, int fd_1);

void queryfgJob(jobSet** bgJobs, job** fgJob);

void querybgJobs(jobSet** bgJobs);

void handleBuildInCmdJobs(jobSet** bgJobs);

bool handleBuildInCmdBg(struct parsed_command** cmd, jobSet** bgJobs);

bool handleBuildInCmdFg(struct parsed_command** cmd, jobSet** bgJobs, job** fgJob);

void handleBuildInCmdPrint(jobSet** bgJobs, job** fgjob);

void handleBuildInCmdNice(struct parsed_command** cmd, int fd_0, int fd_1);

void handleBuildInCmdMan();

int num_arguments(char** args);

#endif
```

4.8 shellLList.h

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

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

#define JOBINITSTAT -1

/*
LinkList structure:

proc: a struct stored
job: a struct stored all the information for a job, like the command, the pgid, the jobid, the status,
printstatus, an array for all the process

jobNode: just like a linkList Node, except the value is a pointer to a job
jobList: just like a linkList

jobSet: A set of jobs. It has serveral different behaviors compared to a simple linklist, like
changing the jobid,  so we define separate function for them

When you remove a node from a linkList, the nodePtr will not be freed, but be returned;
When you delete a job from a jobset, both the job and the nodePtr will be free.

printStatus of job:
0: init
1: running
*/

typedef struct proc {
    int status;
    int pid;
}proc;

typedef struct job {
    int processNum;
    int pgid;
    int jobid;
    int status;
    int printStatus;
    char* cmdContentPtr;
} job;

typedef struct jobNode{
    int pgid;
    job* jobPtr;
    struct jobNode* next;
}jobNode;

typedef struct jobList{
    jobNode *head;
    jobNode *tail;
    int jobNum;
}jobList;

typedef struct jobSet{
    struct jobList *jobs;
    struct jobList *stoppedJobs;
    struct jobList *finishedJobs;
}jobSet;


jobList* initJobList();
jobSet* initJobSet();
jobNode* initJobNode();
void strcpyForJobContent(char* dest, char* src);
job* createJob(int processNum, int pgid, char* cmdContent);
void addNode2jobSet(jobSet* setPtr, job* jobPtr);
void addNode2jobList(jobList* listPtr, job* jobPtr);
void freeJob(job* jobPtr);
void freeJobList(jobList* listPtr);
void freeJobSet(jobSet* setPtr);
jobNode* getCurrJob(jobSet* setPtr);
void deleteJobFromSet(jobSet* setPtr, int pgid);
jobNode* removeJobNodeFromList(jobList* listPtr, int pgid);
void deleteJobNodeFromList(jobList* listPtr, int pgid);
bool nodeInList(jobList* listPtr, int pgid);
void printList(jobList* listPtr);


jobList* initJobList()
{
    jobList* jobListPtr = malloc(sizeof(jobList));
    jobListPtr->head = NULL;
    jobListPtr->tail = NULL;
    jobListPtr->jobNum = 0;
    return jobListPtr;
}

jobSet* initJobSet()
```

```
{
    jobSet* jobSetPtr = malloc(sizeof(jobSet));

    jobSetPtr->jobs = initJobList();
    jobSetPtr->stoppedJobs = initJobList();
    jobSetPtr->finishedJobs = initJobList();

    return jobSetPtr;
}

jobNode* initJobNode()
{
    jobNode* jobNodePtr = malloc(sizeof(jobNode));
    jobNodePtr->jobPtr = NULL;
    jobNodePtr->next = NULL;
    jobNodePtr->pgid = 0;
    return jobNodePtr;
}

void strcpyForJobContent(char* dest, char* src)
{
    int i=0;
    int n=strlen(src);
    while(i<n)
    {
        if(src[i]=='&') break;
        dest[i] = src[i];
        i++;
    }
    dest[i] = '\0';
}

job* createJob(int processNum, int pgid, char* cmdContent)
{
    job* jobPtr = NULL;
    jobPtr=malloc(sizeof(job));
    jobPtr->cmdContentPtr = malloc(strlen(cmdContent)+1);
    // strcpy(jobPtr->cmdContentPtr, cmdContent);
    strcpyForJobContent(jobPtr->cmdContentPtr, cmdContent);
    jobPtr->processNum = processNum;
    jobPtr->pgid = pgid;
    jobPtr->jobid = 0;
    jobPtr->status = JOBINITSTAT;
    jobPtr->printStatus = 0;

    return jobPtr;
}

void addNode2jobSet(jobSet* setPtr, job* jobPtr)
{
    jobList* listPtr = setPtr->jobs;
    jobNode* nodePtr = initJobNode();
    nodePtr->jobPtr = jobPtr;
    nodePtr->pgid = jobPtr->pgid;

    if(listPtr->head==NULL)
    {
        listPtr->head = nodePtr;
        listPtr->tail = nodePtr;
        nodePtr->jobPtr->jobid = 1;
    }
    else
    {
        listPtr->tail->next = nodePtr;
        nodePtr->jobPtr->jobid = listPtr->tail->jobPtr->jobid+1;
        listPtr->tail = nodePtr;
    }
    if(nodePtr!=NULL) listPtr->jobNum++;
}

void addNode2jobList(jobList* listPtr, job* jobPtr)
{
    jobNode* nodePtr = initJobNode();
    nodePtr->jobPtr = jobPtr;
    nodePtr->pgid = jobPtr->pgid;
    if(listPtr->head==NULL)
    {
        listPtr->head = nodePtr;
        listPtr->tail = nodePtr;
    }
    else
    {
        listPtr->tail->next = nodePtr;
        listPtr->tail = nodePtr;
    }
    if(nodePtr!=NULL) listPtr->jobNum++;
}
```

```

void freeJob(job* jobPtr)
{
    if(!jobPtr) return;
    if(jobPtr->cmdContentPtr) free(jobPtr->cmdContentPtr);
    free(jobPtr);
}

void freeJobList(jobList* listPtr)
{
    if(!listPtr) return;
    jobNode* curr = listPtr->head;
    jobNode* prev=NULL;
    while(curr!=NULL)
    {
        prev = curr;
        curr = curr->next;
        free(prev);
    }
    free(listPtr);
}

void freeJobSet(jobSet* setPtr)
{
    if(!setPtr) return;

    freeJobList(setPtr->stoppedJobs);
    freeJobList(setPtr->finishedJobs);
    setPtr->stoppedJobs = NULL;
    setPtr->finishedJobs = NULL;

    // free all jobs in the job set
    jobList* listPtr = setPtr->jobs;
    jobNode* curr = listPtr->head;
    jobNode* prev=NULL;
    while(curr!=NULL)
    {
        prev = curr;
        curr = curr->next;
        freeJob(prev->jobPtr);
        free(prev);
    }
    free(setPtr->jobs);
    free(setPtr);
}

jobNode* getCurrJob(jobSet* setPtr)
{
    if(!setPtr) return NULL;
    if(!setPtr->stoppedJobs->tail) return setPtr->jobs->tail;
    return setPtr->stoppedJobs->tail;
}

void deleteJobFromSet(jobSet* setPtr, int pgid)
{
    // need free jobPtr
    jobNode* deletedJobPtr=NULL;
    deletedJobPtr = removeJobNodeFromList(setPtr->jobs, pgid);
    if(deletedJobPtr) freeJob(deletedJobPtr->jobPtr);
    free(deletedJobPtr);
}

jobNode* removeJobNodeFromList(jobList* listPtr, int pgid)
{
    jobNode *prev = NULL;
    jobNode *curr = listPtr->head;
    jobNode* deleted = NULL;

    while(curr!=NULL && curr->pgid!=pgid)
    {
        prev = curr;
        curr = curr->next;
    }

    if(curr!=NULL && curr->pgid==pgid)
    {
        deleted = curr;
        // curr is not head
        if(prev!=NULL)
        {
            prev->next = curr->next;
            // curr is tail, update tail
            if(curr->next==NULL) listPtr->tail = prev;
        }
        // curr is head
        else
        {
            listPtr->head = curr->next;
            // curr is tail, update tail
            if(curr->next==NULL) listPtr->tail = NULL;
        }
        listPtr->jobNum--;
    }
    return deleted;
}

void deleteJobNodeFromList(jobList* listPtr, int pgid)
{
    jobNode* deleted = removeJobNodeFromList(listPtr, pgid);
    if(deleted) free(deleted);
}

bool nodeInList(jobList* listPtr, int pgid)
{
    if(!listPtr || !listPtr->head) return false;
    jobNode* curr = listPtr->head;
    while(curr)
    {
        if(curr->pgid==pgid) return true;
        curr = curr->next;
    }
    return false;
}

void printList(jobList* listPtr) {
   jobNode* ptr = listPtr->head;
   while(ptr != NULL) {
    job* jobPtr = ptr->jobPtr;
    dprintf(STDERR_FILENO, "([pointer Addr: %p, pgid:%d, jobName:%s] ) ", jobPtr, jobPtr->pgid,
jobPtr->cmdContentPtr);
    ptr = ptr->next;
   }
   dprintf(STDERR_FILENO, "======End print list=======\n");
}
```

4.9 ufuncs.h

```
#ifndef UFUNCS
#define UFUNCS

#include <signal.h>   // sigaction, sigemptyset, sigfillset, signal
#include <stdio.h>    // dprintf, fputs, perror
#include <stdlib.h>   // malloc, free
#include <sys/time.h> // setitimer
#include <ucontext.h> // getcontext, makecontext, setcontext, swapcontext
#include <unistd.h>   // read, usleep, write
#include <valgrind/valgrind.h>
#include <ctype.h>
#include "uprocess.h"
#include "helperfunc.h"
#include "../pennfat/interface.h"

void nap();
// pcb* k_init_booting_process();
void printArgs(char** argv, int fd0, int fd1);
int u_kill_getPid(char* pidArr);
void u_kill(char** argv, int fd0, int fd1);
void u_busy();
void u_sleep(char** argv, int fd0, int fd1);
void u_ps();
void u_zombify();
void zombie_child();
void u_orphanify();
void orphan_child();
void u_hang();
void u_nohang();
void u_recur();

void u_mkfs(char** argv, int fd0, int fd1);
void u_mount(char** argv, int fd0, int fd1);
void u_umount(char** argv, int fd0, int fd1);
void u_touch (char** argv, int fd0, int fd1);
void u_mv(char** argv, int fd0, int fd1);
void u_rm(char** argv, int fd0, int fd1);
void u_cat(char** argv, int fd0, int fd1);
void u_cp(char** argv, int fd0, int fd1);
void u_ls(char** argv, int fd0, int fd1);
void u_chmod(char** argv, int fd0, int fd1);
void u_echo(char** argv, int fd0, int fd1);
void u_getmod(char** argv, int fd0, int fd1);
```

```

#endif
```

4.10 uprocess.h

```
#ifndef USERPROCESS
#define USERPROCESS

#include "pcb.h"
#include "kernel.h"
#include "usignal.h"
#include <sys/types.h>
#include <stdbool.h>

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

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(int ticks);

void p_ps();

pid_t p_wait(int *wstatus, bool nohang);

void p_tcsetgroup(int pid);

#endif
```

4.11 usignal.h

```
#ifndef SIGNAL
#define SIGNAL

#include "pcb.h"

#define S_SIGSTOP 0

#define S_SIGCONT 1

#define S_SIGTERM 2

/* [DEBUG] if use macro, should be ((int)W_IFEXITED(status))!=1, too complicated */

#define W_WIFEXITED(status) status==EXITED? true:false

#define W_WIFSTOPPED(status) status==STOPPED? true:false

#define W_WIFSIGNALED(status) status==SIGKILLED? true:false

#define W_WIFREADY(status) status==READY? true:false

#endif
```

4.12 global.h

```
#ifndef GLOBAL_H
#define GLOBAL_H

#include <stdint.h>
#include <stdbool.h>

extern int num_blocks_in_fat;
extern int num_entries_in_fat;
extern int block_size;
extern int fat_size;
extern int data_size;
extern bool has_mounted;

extern int fs_fd;  // file descriptor of the current file system
extern uint16_t* fat;  // pointer to the current file system as an array

#endif
```

4.13 interface.h

```
#ifndef INTERFACE_H
#define INTERFACE_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdbool.h>
#include <fcntl.h>
#include <errno.h>
#include <stdint.h>
#include <sys/mman.h>
#include <time.h>
#include "interface.h"
#include "pennfat_utils.h"
#include "global.h"
#include "syscall.h"

int mkfs(int num_args, char** args);

int mount(int num_args, char** args);

int umount(int num_args);

int touch(int num_args, char** args);

int mv(int num_args, char** args);

int rm(int num_args, char** args);

int cat(int num_args, char** args, int source_fd, int dest_fd);

int cp(int num_args, char** args);

int ls(int num_args);

int chmod(int num_args, char** args);

int getmod(int num_args, char** args);
int getmod_nameOnly(int num_args, char** args);

int echo(int num_args, char** args, int dest_fd);

#endif
```

4.14 pennfat.h

```
#ifndef PENNFAT_H
#define PENNFAT_H

#include <stdint.h>
#include <time.h>

#define MAX_LINE_LENGTH 4096
#define DIR_ENTRY_SIZE 64
#define FAT_ENTRY_SIZE 2
#define FAT_PROMPT "pennfat# "

typedef struct directory_entry {
    char name[32];
    uint32_t size;
    uint16_t firstBlock;
    uint8_t type;
    uint8_t perm;
    time_t mtime;
    // The remaining 16 bytes are reserved
} DirectoryEntry;

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

#endif
```

4.15 pennfat_utils.h

```
#ifndef PENNFAT_UTILS_H
#define PENNFAT_UTILS_H

int map_block_size(int block_size_id);
int map_perm(int perm);

#endif
```

4.16 syscall.h

```
#ifndef SYSCALL_H
#define SYSCALL_H

#include "global.h"
#include "pennfat.h"

#define F_NONEXIST 0
#define F_WRITE 1
#define F_READ 2
#define F_APPEND 3
#define F_RM 4

#define S_NONE 0
#define S_WRONLY 2
#define S_RDONLY 4
#define S_REONLY 5
#define S_RDWR_TRUN 6
#define S_RWE 7
#define S_RDWR_NOTRUN 9

#define F_SEEK_SET 0
#define F_SEEK_CUR 1
#define F_SEEK_END 2

#define MAX_NAME_LENGTH 32
#define FDTABLE_SIZE block_size/DIR_ENTRY_SIZE

typedef struct{
    char file_name[MAX_NAME_LENGTH];
    int fd;
    int status;
    int mode;
    int occupied;
    int file_idx; // use this to access file pointer in pennFAT
    int dir_idx;
    int num_read;
    // int file_size;
} FD_TABLE_ENTRY;

void init_fd_table(char* fs_name);
void close_fd_table();
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_unlink(const char *fname);
int f_lseek(int fd, int offset, int whence);
int f_ls(const char *filename);
int f_chmod(const char *filename, int perm);
int f_getmod(const char *filename);

#endif
```

4.17 syscall_utils.h

```
#ifndef SYSCALL_UTILS_H
#define SYSCALL_UTILS_H

#include <stdint.h>
#include "syscall.h"

int min2(int a, int b);
int min3(int a, int b, int c);

int find_avail_entry_in_root();
int create_new_file_pennfat(const char *fname, int mode);
DirectoryEntry* find_curr_dir_entry(FD_TABLE_ENTRY *fd_entry);
```

```c

void create_fat_entry(DirectoryEntry* dir_entry);
int read_from_pennfat(int num, char* buf, FD_TABLE_ENTRY *fd_entry);
int write_to_pennfat(int num, const char* buf, FD_TABLE_ENTRY *fd_entry, int status);
void remove_from_pennfat(FD_TABLE_ENTRY *fd_entry);

int look_through_directory_blocks(const char *filename);
int change_permission(const char *filename, int perm);
int get_permission(const char *filename);

int get_begin_file_idx(FD_TABLE_ENTRY *fd_entry);
int get_curr_file_idx(FD_TABLE_ENTRY *fd_entry);
int get_end_file_idx(FD_TABLE_ENTRY *fd_entry);

#endif
```

Index