

pennos

Generated by Doxygen 1.9.7

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 directory_entry Struct Reference	5
3.1.1 Detailed Description	5
3.2 FD_TABLE_ENTRY Struct Reference	5
3.3 job Struct Reference	6
3.3.1 Detailed Description	6
3.3.2 Member Data Documentation	6
3.3.2.1 cmdContentPtr	6
3.3.2.2 jobid	6
3.3.2.3 pgid	6
3.3.2.4 printStatus	6
3.3.2.5 processNum	7
3.3.2.6 status	7
3.4 jobList Struct Reference	7
3.4.1 Detailed Description	7
3.4.2 Member Data Documentation	7
3.4.2.1 head	7
3.4.2.2 jobNum	7
3.4.2.3 tail	8
3.5 jobNode Struct Reference	8
3.5.1 Detailed Description	8
3.5.2 Member Data Documentation	8
3.5.2.1 jobPtr	8
3.5.2.2 next	8
3.5.2.3 pgid	8
3.6 jobSet Struct Reference	9
3.6.1 Detailed Description	9
3.6.2 Member Data Documentation	9
3.6.2.1 finishedJobs	9
3.6.2.2 jobs	9
3.6.2.3 stoppedJobs	9
3.7 parsed_command Struct Reference	9
3.7.1 Detailed Description	10
3.8 pcb Struct Reference	10
3.8.1 Detailed Description	10
3.8.2 Member Data Documentation	10
3.8.2.1 children	10

3.8.2.2 p_level	11
3.8.2.3 parent	11
3.8.2.4 pid	11
3.8.2.5 process_name	11
3.8.2.6 sleep_stop_tick	11
3.8.2.7 thread_state	11
3.8.2.8 ucontext	11
3.8.2.9 waitPid	11
3.8.2.10 zombie_children	12
3.9 proc Struct Reference	12
3.9.1 Detailed Description	12
3.9.2 Member Data Documentation	12
3.9.2.1 pid	12
3.9.2.2 status	12
3.10 procList Struct Reference	13
3.10.1 Detailed Description	13
3.10.2 Member Data Documentation	13
3.10.2.1 head	13
3.10.2.2 procNum	13
3.10.2.3 tail	13
3.11 procNode Struct Reference	13
3.11.1 Detailed Description	14
3.11.2 Member Data Documentation	14
3.11.2.1 next	14
3.11.2.2 pcbPtr	14
3.11.2.3 pid	14
3.12 Scheduler Struct Reference	14
3.12.1 Detailed Description	15
3.12.2 Member Data Documentation	15
3.12.2.1 all_procList	15
3.12.2.2 blocked_procList	15
3.12.2.3 blockPid	15
3.12.2.4 idle	15
3.12.2.5 idlePcb	16
3.12.2.6 minus1_procList	16
3.12.2.7 plus1_procList	16
3.12.2.8 runningLock	16
3.12.2.9 schedulerContext	16
3.12.2.10 stopped_procList	16
3.12.2.11 terminate	16
3.12.2.12 totalticks	16
3.12.2.13 zero_procList	17

3.12.2.14 zombie_procList	17
4 File Documentation	19
4.1 helperfunc.h	19
4.2 kernel.h	19
4.3 logger.h	20
4.4 parser.h	20
4.5 pcb.h	21
4.6 scheduler.h	22
4.7 shell.h	23
4.8 shellLList.h	23
4.9 ufuncs.h	27
4.10 uprocess.h	28
4.11 usignal.h	28
4.12 global.h	28
4.13 interface.h	29
4.14 pennfat.h	29
4.15 pennfat_utils.h	30
4.16 syscall.h	30
4.17 syscall_utils.h	30
Index	33

Chapter 1

Class Index

1.1 Class List

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

directory_entry	5
FD_TABLE_ENTRY	5
job	
A struct to store the information of a job	6
jobList	
A struct to store a list of jobs	7
jobNode	
A struct to store the information of a jobNode	8
jobSet	
A struct to store a set of jobs	9
parsed_command	9
pcb	
The struct for Process Control Block of a process	10
proc	
A struct to store the information of a process	12
procList	
The struct for a list of process nodes	13
procNode	
The struct for the process node inside a procList	13
Scheduler	
The struct for the scheduler	14

Chapter 2

File Index

2.1 File List

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

23sp-pennOS-group-12/src/kernel/ helperfunc.h	19
23sp-pennOS-group-12/src/kernel/ kernel.h	19
23sp-pennOS-group-12/src/kernel/ logger.h	20
23sp-pennOS-group-12/src/kernel/ parser.h	20
23sp-pennOS-group-12/src/kernel/ pcb.h	21
23sp-pennOS-group-12/src/kernel/ scheduler.h	22
23sp-pennOS-group-12/src/kernel/ shell.h	23
23sp-pennOS-group-12/src/kernel/ shellList.h	23
23sp-pennOS-group-12/src/kernel/ ufuncs.h	27
23sp-pennOS-group-12/src/kernel/ uprocess.h	28
23sp-pennOS-group-12/src/kernel/ usignal.h	28
23sp-pennOS-group-12/src/pennfat/ global.h	28
23sp-pennOS-group-12/src/pennfat/ interface.h	29
23sp-pennOS-group-12/src/pennfat/ pennfat.h	29
23sp-pennOS-group-12/src/pennfat/ pennfat_utils.h	30
23sp-pennOS-group-12/src/pennfat/ syscall.h	30
23sp-pennOS-group-12/src/pennfat/ syscall_utils.h	30

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 <shellLLList.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 <shellLLList.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/shellLLList.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 proclList* 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 <shellLLList.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/shellLLList.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

```
00001 #ifndef HELPERFUNC
00002 #define HELPERFUNC
00003
00004 #include <stdio.h>
00005 #include <string.h>
00006 #include <stdlib.h>
00007 #include <stdbool.h>
00008
00009 char** copyArgv(char** argv, bool addNewLine);
00010 void free_charArr(char*** arr);
00011 int num_arguments(char** args);
00012 char** parseInputFromScript(char* buf);
00013
00014
00015 #endif
```

4.2 kernel.h

```
00001 #ifndef KERNEL
00002 #define KERNEL
00003
00004 #include <unistd.h>
00005 #include <signal.h>
00006 #include <sys/time.h> // setitimer
00007 #include <stdlib.h>
00008 #include <fcntl.h>
00009 #include "logger.h"
00010 #include "scheduler.h"
00011 #include "pcb.h"
00012 #include "usignal.h"
00013 #include "uprocess.h"
00014
00018 extern Scheduler* global_scheduler_ptr;
00019
00023 extern pcb* active_pcb_ptr;
00024
00030 extern int process_num;
00031
00035 static const int centisecond = 100000; // 100 ms
00036
00042 void alarmHandler(int signum);
00043
00048 void setAlarmHandler(void);
00049
00054 void setTimer();
00055
00061 pcb* k_process_create(pcb* parent, int p_level, void (*func)(), int argc, char *argv[], char*
    process_name, int fd0, int fd1);
00062
00069 bool k_process_kill(pcb* proc, int sig);
00070
00074 void k_process_exit();
00075
00082 void k_process_cleanup(pcb* proc);
```

```

00083 void k_process_setChildOrphan(pcb* proc);
00084
00089 void shell();
00090
00094 void sigSTPHandler(int sig);
00095
00099 void sigIntHandler(int sig);
00100
00101 #endif

```

4.3 logger.h

```

00001 #ifndef LOGGER
00002 #define LOGGER
00003
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006 #include <string.h>
00007
00008 extern char *logfile_path;
00009
00018 void log_schedule_event(int ticks, int pid, int priority, char* process_name);
00019
00028 void log_create_event(int ticks, int pid, int priority, char* process_name);
00029
00038 void log_signaled_event(int ticks, int pid, int priority, char* process_name);
00039
00048 void log_exited_event(int ticks, int pid, int priority, char* process_name);
00049
00058 void log_zombie_event(int ticks, int pid, int priority, char* process_name);
00059
00068 void log_orphan_event(int ticks, int pid, int priority, char* process_name);
00069
00078 void log_waited_event(int ticks, int pid, int priority, char* process_name);
00079
00089 void log_nice_event(int ticks, int pid, int priority_old, int priority_new, char* process_name);
00090
00099 void log_blocked_event(int ticks, int pid, int priority, char* process_name);
00100
00109 void log_unblocked_event(int ticks, int pid, int priority, char* process_name);
00110
00119 void log_stopped_event(int ticks, int pid, int priority, char* process_name);
00120
00129 void log_continued_event(int ticks, int pid, int priority, char* process_name);
00130
00131 #endif

```

4.4 parser.h

```

00001 /* Penn-Shell Parser
00002     hanbangw, 21fa */
00003
00004 #pragma once
00005
00006 #include <stddef.h>
00007 #include <stdbool.h>
00008
00009 /* Here defines all possible parser errors */
00010 // parser encountered an unexpected file input token '<'
00011 #define UNEXPECTED_FILE_INPUT 1
00012
00013 // parser encountered an unexpected file output token '>'
00014 #define UNEXPECTED_FILE_OUTPUT 2
00015
00016 // parser encountered an unexpected pipeline token '|'
00017 #define UNEXPECTED_PIPELINE 3
00018
00019 // parser encountered an unexpected ampersand token '&'
00020 #define UNEXPECTED_AMPERSAND 4
00021
00022 // parser didn't find input filename following '<'
00023 #define EXPECT_INPUT_FILENAME 5
00024
00025 // parser didn't find output filename following '>' or '>'
00026 #define EXPECT_OUTPUT_FILENAME 6
00027
00028 // parser didn't find any commands or arguments where it expects one
00029 #define EXPECT_COMMANDS 7
00030

```

```

00035 struct parsed_command {
00036     // indicates the command shall be executed in background
00037     // (ends with an ampersand '&')
00038     bool is_background;
00039
00040     // indicates if the stdout_file shall be opened in append mode
00041     // ignore this value when stdout_file is NULL
00042     bool is_file_append;
00043
00044     // filename for redirecting input from
00045     const char *stdin_file;
00046
00047     // filename for redirecting output to
00048     const char *stdout_file;
00049
00050     // number of commands (pipeline stages)
00051     size_t num_commands;
00052
00053     // an array to a list of arguments
00054     // size of 'commands' is 'num_commands'
00055     char **commands[];
00056 };
00057
00083 int parse_command(const char *cmd_line, struct parsed_command **result);
00084
00085
00086 /* This is a debugging function used for outputting a parsed command line. */
00087 void print_parsed_command(const struct parsed_command *cmd);

```

4.5 pcb.h

```

00001 #ifndef PCB
00002 #define PCB
00003
00004 #include <stdio.h>
00005 #include <string.h>
00006 #include <stdlib.h>
00007 #include <stdbool.h>
00008 #include <signal.h>
00009 #include <ucontext.h>
00010 #include <threads.h>
00011 #include <unistd.h>
00012 #include <valgrind/valgrind.h>
00013
00014
00020 #define STACK_SIZE 819200
00021
00027 #define READY 0
00028
00034 #define STOPPED 1
00035
00041 #define BLOCKED 2
00042
00048 #define EXITED 3
00049
00055 #define ZOMBIED EXITED
00056
00062 #define ORPHAN 5
00063
00069 #define CLEAN 6
00070
00076 #define SIGKILLED 7
00077
00078 struct procList;
00079
00085 typedef struct pcb {
00086     int pid;
00087     int p_level;
00088     int thread_state;
00089     ucontext_t *ucontext;
00090     struct pcb *parent;
00091     struct procList *children;
00092     struct procList *zombie_children;
00093     int waitPid;
00094     int sleep_stop_tick;
00095     char *process_name;
00096 } pcb;
00097
00098
00104 typedef struct procNode{
00105     int pid;
00106     pcb *pcbPtr;
00107     struct procNode *next;

```

```

00108 } procNode;
00109
00110
00116 typedef struct procList{
00117     procNode *head;
00118     procNode *tail;
00119     int procNum;
00120 } procList;
00121
00126 void setStack(stack_t *stack);
00127
00145 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);
00146
00151 void free_pcb(pcb* pcbPtr);
00152
00157 void print_pcb(pcb* pcbPtr);
00158
00163 procList* initprocList();
00164
00171 void freeprocPtr(procNode* procPtr, bool freePCB);
00172
00179 void freeprocList(procList* listPtr, bool freePCB);
00180
00185 void print_procList(procList* procListPtr);
00186
00193 void addNode2procList(procList* listPtr, pcb* pcbPtr);
00194
00203 bool deleteNodeFromList(procList* listPtr, int pid);
00204
00213 procNode* removeNodeFromList(procList* listPtr, int pid);
00214
00223 pcb* find_pcb_in_procList(procList* listPtr, int pid);
00224
00229 void roundRobinUpdate(procList* listPtr);
00230
00231 #endif

```

4.6 scheduler.h

```

00001 #ifndef SCHEDULER
00002 #define SCHEDULER
00003
00004 #include <stdlib.h> // malloc, free
00005 #include <sys/types.h>
00006 #include "pcb.h"
00007
00012 #define SHELL_PID 1
00013
00018 #define IDLE_PID -1
00019
00025 typedef struct Scheduler {
00026     procList* minus1_procList;
00027     procList* zero_procList;
00028     procList* plus1_procList;
00029     procList* blocked_procList;
00030     procList* zombie_procList;
00031     procList* stopped_procList;
00032     procList* all_procList;
00033     int totalticks;
00034     bool runningLock;
00035     bool terminate;
00036     bool idle;
00037     int blockPid;
00038     ucontext_t* schedulerContext;
00039     pcb* idlePcb;
00040 } Scheduler;
00041
00049 Scheduler* init_scheduler(ucontext_t* mainContext);
00050
00056 pcb* init_idlePcb();
00057
00064 void init_schedulerContext(Scheduler* schedulerPtr, ucontext_t* mainContext);
00065
00071 void free_scheduler(Scheduler* schedulerPtr);
00072
00076 void addProc2ReadyQueue(Scheduler* schedulerPtr, pcb* pcbPtr);
00077
00084 pcb* chooseproctoRun_minus1(Scheduler* schedulerPtr);
00085
00092 pcb* chooseproctoRun_zero(Scheduler* schedulerPtr);
00093
00100 pcb* chooseproctoRun_plus1(Scheduler* schedulerPtr);

```

```

00101
00107 void deleteFromReadyQueue(pcb* pcbPtr);
00108
00114 void checkReadyQueue(procList* procListPtr);
00115
00126 bool checkPcb(pcb* pcbPtr, char* logPrefix, int excludeState, procList* procListPtr);
00127
00133 void checkAllReadyQueue(Scheduler* schedulerPtr);
00134
00138 void checkActivePcb();
00139
00148 void checkZombieQueue(Scheduler* schedulerPtr);
00149
00155 void checkBlockedQueue(Scheduler* schedulerPtr);
00156
00164 bool unblockWaitingParent(pcb* pcbPtr);
00165
00171 bool scheduleTick();
00172
00180 pcb* find_pcb_with_pid(int pid);
00181
00186 void idle_func();
00187
00192 void idle();
00193
00194 #endif

```

4.7 shell.h

```

00001 #ifndef SHELL
00002 #define SHELL
00003
00004 #include <stdio.h>
00005 #include <string.h>
00006 #include <stdlib.h>
00007 #include <stdbool.h>
00008 #include <signal.h>
00009 #include "parser.h"
00010 #include "kernel.h"
00011 #include "ufuncs.h"
00012 #include "usignal.h"
00013 #include "shellLLList.h"
00014 #include "helperfunc.h"
00015
00020 void shell();
00021
00030 void handleOneLineCmd(char* line, struct parsed_command** cmd, jobSet** bgJobs, job** fgjob);
00031
00038 bool isBuildInCmd(struct parsed_command** cmd);
00039
00049 void handleBuildInCmds(struct parsed_command** cmd, jobSet** bgJobs, job** fgjob, int fd_0, int fd_1);
00050
00059 int handleUserCmds(char** argv, int p_level, int fd_0, int fd_1);
00060
00067 void queryfgJob(jobSet** bgJobs, job** fgJob);
00068
00074 void querybgJobs(jobSet** bgJobs);
00075
00082 void handleBuildInCmdJobs(jobSet** bgJobs);
00083
00090 bool handleBuildInCmdBg(struct parsed_command** cmd, jobSet** bgJobs);
00091
00099 bool handleBuildInCmdFg(struct parsed_command** cmd, jobSet** bgJobs, job** fgJob);
00100
00107 void handleBuildInCmdPrint(jobSet** bgJobs, job** fgjob);
00108
00116 void handleBuildInCmdNice(struct parsed_command** cmd, int fd_0, int fd_1);
00117
00121 void handleBuildInCmdMan();
00122
00130 int num_arguments(char** args);
00131
00132 #endif

```

4.8 shellLLList.h

```

00001 #include <stdio.h>
00002 #include <string.h>
00003 #include <stdlib.h>

```

```

00004 #include <stdbool.h>
00005 #include <signal.h>
00006
00007 #define JOBINITSTAT -1
00008
00009 /*
00010 LinkList structure:
00011
00012 proc: a struct stored
00013 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
00014
00015 jobNode: just like a linkList Node, except the value is a pointer to a job
00016 jobList: just like a linkList
00017
00018 jobSet: A set of jobs. It has severla different behaviors compared to a simple linklist, like
    changing the jobid, so we define separate function for them
00019
00020 When you remove a node from a linkList, the nodePtr will not be freed, but be returned;
00021 When you delete a job from a jobset, both the job and the nodePtr will be free.
00022
00023 printStatus of job:
00024 0: init
00025 1: running
00026 */
00027
00033 typedef struct proc {
00034     int status;
00035     int pid;
00036 }proc;
00037
00042 typedef struct job {
00043     int processNum;
00044     int pgid;
00045     int jobid;
00046     int status;
00047     int printStatus;
00048     char* cmdContentPtr;
00049 } job;
00050
00056 typedef struct jobNode{
00057     int pgid;
00058     job* jobPtr;
00059     struct jobNode* next;
00060 }jobNode;
00061
00067 typedef struct jobList{
00068     jobNode *head;
00069     jobNode *tail;
00070     int jobNum;
00071 }jobList;
00072
00078 typedef struct jobSet{
00079     struct jobList *jobs;
00080     struct jobList *stoppedJobs;
00081     struct jobList *finishedJobs;
00082 }jobSet;
00083
00084
00085 jobList* initJobList();
00086 jobSet* initJobSet();
00087 jobNode* initJobNode();
00088 void strcpyForJobContent(char* dest, char* src);
00089 job* createJob(int processNum, int pgid, char* cmdContent);
00090 void addNode2jobSet(jobSet* setPtr, job* jobPtr);
00091 void addNode2jobList(jobList* listPtr, job* jobPtr);
00092 void freeJob(job* jobPtr);
00093 void freeJobList(jobList* listPtr);
00094 void freeJobSet(jobSet* setPtr);
00095 jobNode* getCurrJob(jobSet* setPtr);
00096 void deleteJobFromSet(jobSet* setPtr, int pgid);
00097 jobNode* removeJobNodeFromList(jobList* listPtr, int pgid);
00098 void deleteJobNodeFromList(jobList* listPtr, int pgid);
00099 bool nodeInList(jobList* listPtr, int pgid);
00100 void printList(jobList* listPtr);
00101
00102
00103 jobList* initJobList()
00104 {
00105     jobList* jobListPtr = malloc(sizeof(jobList));
00106     jobListPtr->head = NULL;
00107     jobListPtr->tail = NULL;
00108     jobListPtr->jobNum = 0;
00109     return jobListPtr;
00110 }
00111
00112 jobSet* initJobSet()

```

```

00113 {
00114     jobSet* jobSetPtr = malloc(sizeof(jobSet));
00115
00116     jobSetPtr->jobs = initJobList();
00117     jobSetPtr->stoppedJobs = initJobList();
00118     jobSetPtr->finishedJobs = initJobList();
00119
00120     return jobSetPtr;
00121 }
00122
00123 jobNode* initJobNode()
00124 {
00125     jobNode* jobNodePtr = malloc(sizeof(jobNode));
00126     jobNodePtr->jobPtr = NULL;
00127     jobNodePtr->next = NULL;
00128     jobNodePtr->pgid = 0;
00129     return jobNodePtr;
00130 }
00131
00132 void strcpyForJobContent(char* dest, char* src)
00133 {
00134     int i=0;
00135     int n=strlen(src);
00136     while(i<n)
00137     {
00138         if(src[i]=='&') break;
00139         dest[i] = src[i];
00140         i++;
00141     }
00142     dest[i] = '\0';
00143 }
00144
00145 job* createJob(int processNum, int pgid, char* cmdContent)
00146 {
00147     job* jobPtr = NULL;
00148     jobPtr=malloc(sizeof(job));
00149     jobPtr->cmdContentPtr = malloc(strlen(cmdContent)+1);
00150     // strcpy(jobPtr->cmdContentPtr, cmdContent);
00151     strcpyForJobContent(jobPtr->cmdContentPtr, cmdContent);
00152     jobPtr->processNum = processNum;
00153     jobPtr->pgid = pgid;
00154     jobPtr->jobid = 0;
00155     jobPtr->status = JOBINITSTAT;
00156     jobPtr->printStatus = 0;
00157
00158     return jobPtr;
00159 }
00160
00161 void addNode2jobSet(jobSet* setPtr, job* jobPtr)
00162 {
00163     jobList* listPtr = setPtr->jobs;
00164     jobNode* nodePtr = initJobNode();
00165     nodePtr->jobPtr = jobPtr;
00166     nodePtr->pgid = jobPtr->pgid;
00167
00168     if(listPtr->head==NULL)
00169     {
00170         listPtr->head = nodePtr;
00171         listPtr->tail = nodePtr;
00172         nodePtr->jobPtr->jobid = 1;
00173     }
00174     else
00175     {
00176         listPtr->tail->next = nodePtr;
00177         nodePtr->jobPtr->jobid = listPtr->tail->jobPtr->jobid+1;
00178         listPtr->tail = nodePtr;
00179     }
00180     if(nodePtr!=NULL) listPtr->jobNum++;
00181 }
00182
00183 void addNode2jobList(jobList* listPtr, job* jobPtr)
00184 {
00185     jobNode* nodePtr = initJobNode();
00186     nodePtr->jobPtr = jobPtr;
00187     nodePtr->pgid = jobPtr->pgid;
00188     if(listPtr->head==NULL)
00189     {
00190         listPtr->head = nodePtr;
00191         listPtr->tail = nodePtr;
00192     }
00193     else
00194     {
00195         listPtr->tail->next = nodePtr;
00196         listPtr->tail = nodePtr;
00197     }
00198     if(nodePtr!=NULL) listPtr->jobNum++;
00199 }

```

```

00200
00201 void freeJob(job* jobPtr)
00202 {
00203     if(!jobPtr) return;
00204     if(jobPtr->cmdContentPtr) free(jobPtr->cmdContentPtr);
00205     free(jobPtr);
00206 }
00207
00208 void freeJobList(jobList* listPtr)
00209 {
00210     if(!listPtr) return;
00211     jobNode* curr = listPtr->head;
00212     jobNode* prev=NULL;
00213     while(curr!=NULL)
00214     {
00215         prev = curr;
00216         curr = curr->next;
00217         free(prev);
00218     }
00219     free(listPtr);
00220 }
00221
00222 void freeJobSet(jobSet* setPtr)
00223 {
00224     if(!setPtr) return;
00225
00226     freeJobList(setPtr->stoppedJobs);
00227     freeJobList(setPtr->finishedJobs);
00228     setPtr->stoppedJobs = NULL;
00229     setPtr->finishedJobs = NULL;
00230
00231     // free all jobs in the job set
00232     jobList* listPtr = setPtr->jobs;
00233     jobNode* curr = listPtr->head;
00234     jobNode* prev=NULL;
00235     while(curr!=NULL)
00236     {
00237         prev = curr;
00238         curr = curr->next;
00239         freeJob(prev->jobPtr);
00240         free(prev);
00241     }
00242     free(setPtr->jobs);
00243     free(setPtr);
00244 }
00245
00246 jobNode* getCurrJob(jobSet* setPtr)
00247 {
00248     if(!setPtr) return NULL;
00249     if(!setPtr->stoppedJobs->tail) return setPtr->jobs->tail;
00250     return setPtr->stoppedJobs->tail;
00251 }
00252
00253 void deleteJobFromSet(jobSet* setPtr, int pgid)
00254 {
00255     // need free jobPtr
00256     jobNode* deletedJobPtr=NULL;
00257     deletedJobPtr = removeJobNodeFromList(setPtr->jobs, pgid);
00258     if(deletedJobPtr) freeJob(deletedJobPtr->jobPtr);
00259     free(deletedJobPtr);
00260 }
00261
00262 jobNode* removeJobNodeFromList(jobList* listPtr, int pgid)
00263 {
00264     jobNode *prev = NULL;
00265     jobNode *curr = listPtr->head;
00266     jobNode* deleted = NULL;
00267
00268     while(curr!=NULL && curr->pgid!=pgid)
00269     {
00270         prev = curr;
00271         curr = curr->next;
00272     }
00273
00274     if(curr!=NULL && curr->pgid==pgid)
00275     {
00276         deleted = curr;
00277         // curr is not head
00278         if(prev!=NULL)
00279         {
00280             prev->next = curr->next;
00281             // curr is tail, update tail
00282             if(curr->next==NULL) listPtr->tail = prev;
00283         }
00284         // curr is head
00285         else
00286         {

```



```

00287         listPtr->head = curr->next;
00288         // curr is tail, update tail
00289         if(curr->next==NULL) listPtr->tail = NULL;
00290     }
00291     listPtr->jobNum--;
00292 }
00293 return deleted;
00294 }
00295
00296 void deleteJobNodeFromList(jobList* listPtr, int pgid)
00297 {
00298     jobNode* deleted = removeJobNodeFromList(listPtr, pgid);
00299     if(deleted) free(deleted);
00300 }
00301
00302 bool nodeInList(jobList* listPtr, int pgid)
00303 {
00304     if(!listPtr || !listPtr->head) return false;
00305     jobNode* curr = listPtr->head;
00306     while(curr)
00307     {
00308         if(curr->pgid==pgid) return true;
00309         curr = curr->next;
00310     }
00311     return false;
00312 }
00313
00314 void printList(jobList* listPtr) {
00315     jobNode* ptr = listPtr->head;
00316     while(ptr != NULL) {
00317         job* jobPtr = ptr->jobPtr;
00318         dprintf(STDERR_FILENO, "([pointer Addr: %p, pgid:%d, jobName:%s] ) ", jobPtr, jobPtr->pgid,
00319             jobPtr->cmdContentPtr);
00320         ptr = ptr->next;
00321     }
00322     dprintf(STDERR_FILENO, "====End print list====\n");
00323 }

```

4.9 ufuncs.h

```

00001 #ifndef UFUNCS
00002 #define UFUNCS
00003
00004 #include <signal.h> // sigaction, sigemptyset, sigfillset, signal
00005 #include <stdio.h> // dprintf, fputs, perror
00006 #include <stdlib.h> // malloc, free
00007 #include <sys/time.h> // setitimer
00008 #include <ucontext.h> // getcontext, makecontext, setcontext, swapcontext
00009 #include <unistd.h> // read, usleep, write
00010 #include <valgrind/valgrind.h>
00011 #include <ctype.h>
00012 #include "uprocess.h"
00013 #include "helperfunc.h"
00014 #include "../pennfat/interface.h"
00015
00016 void nap();
00017 // pcb* k_init_booting_process();
00018 void printArgs(char** argv, int fd0, int fd1);
00019 int u_kill_getPid(char* pidArr);
00020 void u_kill(char** argv, int fd0, int fd1);
00021 void u_busy();
00022 void u_sleep(char** argv, int fd0, int fd1);
00023 void u_ps();
00024 void u_zombify();
00025 void u_zombie_child();
00026 void u_orphanify();
00027 void u_orphan_child();
00028 void u_hang();
00029 void u_nohang();
00030 void u_recur();
00031
00032 void u_mkfs(char** argv, int fd0, int fd1);
00033 void u_mount(char** argv, int fd0, int fd1);
00034 void u_umount(char** argv, int fd0, int fd1);
00035 void u_touch(char** argv, int fd0, int fd1);
00036 void u_mv(char** argv, int fd0, int fd1);
00037 void u_rm(char** argv, int fd0, int fd1);
00038 void u_cat(char** argv, int fd0, int fd1);
00039 void u_cp(char** argv, int fd0, int fd1);
00040 void u_ls(char** argv, int fd0, int fd1);
00041 void u_chmod(char** argv, int fd0, int fd1);
00042 void u_echo(char** argv, int fd0, int fd1);
00043 void u_getmod(char** argv, int fd0, int fd1);

```

```
00044
00045 #endif
```

4.10 uprocess.h

```
00001 #ifndef USERPROCESS
00002 #define USERPROCESS
00003
00004 #include "pcb.h"
00005 #include "kernel.h"
00006 #include "usignal.h"
00007 #include <sys/types.h>
00008 #include <stdbool.h>
00009
00022 pid_t p_spawn(int p_level, void (*func)(), int argc, char *argv[], int fd0, int fd1, char*
    process_name);
00023
00033 pid_t p_waitpid(pid_t pid, int *wstatus, bool nohang);
00034
00043 int p_kill(pid_t pid, int sig);
00044
00048 void p_exit(void);
00049
00058 int p_nice(pid_t pid, int priority);
00059
00065 void p_sleep(int ticks);
00066
00071 void p_ps();
00072
00080 pid_t p_wait(int *wstatus, bool nohang);
00081
00087 void p_tcsetgroup(int pid);
00088
00089 #endif
```

4.11 usignal.h

```
00001 #ifndef SIGNAL
00002 #define SIGNAL
00003
00004 #include "pcb.h"
00005
00011 #define S_SIGSTOP 0
00012
00018 #define S_SIGCONT 1
00019
00025 #define S_SIGTERM 2
00026
00027 /* [DEBUG] if use macro, should be ((int)W_IFEXITED(status))!=1, too complicated */
00028
00033 #define W_WIFEXITED(status) status==EXITED? true:false
00034
00039 #define W_WIFSTOPPED(status) status==STOPPED? true:false
00040
00045 #define W_WIFSIGNALED(status) status==SIGKILLED? true:false
00046
00051 #define W_WIFREADY(status) status==READY? true:false
00052
00053 #endif
```

4.12 global.h

```
00001 #ifndef GLOBAL_H
00002 #define GLOBAL_H
00003
00004 #include <stdint.h>
00005 #include <stdbool.h>
00006
00007 extern int num_blocks_in_fat;
00008 extern int num_entries_in_fat;
00009 extern int block_size;
00010 extern int fat_size;
00011 extern int data_size;
00012 extern bool has_mounted;
00013
```

```

00014 extern int fs_fd; // file descriptor of the current file system
00015 extern uint16_t* fat; // pointer to the current file system as an array
00016
00017 #endif

```

4.13 interface.h

```

00001 #ifndef INTERFACE_H
00002 #define INTERFACE_H
00003
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006 #include <unistd.h>
00007 #include <string.h>
00008 #include <stdbool.h>
00009 #include <fcntl.h>
00010 #include <errno.h>
00011 #include <stdint.h>
00012 #include <sys/mman.h>
00013 #include <time.h>
00014 #include "interface.h"
00015 #include "pennfat_utils.h"
00016 #include "global.h"
00017 #include "syscall.h"
00018
00026 int mkfs(int num_args, char** args);
00027
00032 int mount(int num_args, char** args);
00033
00037 int umount(int num_args);
00038
00043 int touch(int num_args, char** args);
00044
00049 int mv(int num_args, char** args);
00050
00055 int rm(int num_args, char** args);
00056
00067 int cat(int num_args, char** args, int source_fd, int dest_fd);
00068
00075 int cp(int num_args, char** args);
00076
00081 int ls(int num_args);
00082
00093 int chmod(int num_args, char** args);
00094
00105 int getmod(int num_args, char** args);
00106 int getmod_nameOnly(int num_args, char** args);
00107
00112 int echo(int num_args, char** args, int dest_fd);
00113
00114 #endif

```

4.14 pennfat.h

```

00001 #ifndef PENNFAT_H
00002 #define PENNFAT_H
00003
00004 #include <stdint.h>
00005 #include <time.h>
00006
00007 #define MAX_LINE_LENGTH 4096
00008 #define DIR_ENTRY_SIZE 64
00009 #define FAT_ENTRY_SIZE 2
00010 #define FAT_PROMPT "pennfat# "
00011
00016 typedef struct directory_entry {
00017     char name[32];
00018     uint32_t size;
00019     uint16_t firstBlock;
00020     uint8_t type;
00021     uint8_t perm;
00022     time_t mtime;
00023     // The remaining 16 bytes are reserved
00024 } DirectoryEntry;
00025
00026 int main(int argc, char *argv[]);
00027
00028 #endif

```

4.15 pennfat_utils.h

```

00001 #ifndef PENNFAT_UTILS_H
00002 #define PENNFAT_UTILS_H
00003
00004 int map_block_size(int block_size_id);
00005 int map_perm(int perm);
00006
00007 #endif

```

4.16 syscall.h

```

00001 #ifndef SYSCALL_H
00002 #define SYSCALL_H
00003
00004 #include "global.h"
00005 #include "pennfat.h"
00006
00007 #define F_NONEXIST 0
00008 #define F_WRITE 1
00009 #define F_READ 2
00010 #define F_APPEND 3
00011 #define F_RM 4
00012
00013 #define S_NONE 0
00014 #define S_WRONLY 2
00015 #define S_RDONLY 4
00016 #define S_REONLY 5
00017 #define S_RDWR_TRUN 6
00018 #define S_RWE 7
00019 #define S_RDWR_NOTRUN 9
00020
00021 #define F_SEEK_SET 0
00022 #define F_SEEK_CUR 1
00023 #define F_SEEK_END 2
00024
00025 #define MAX_NAME_LENGTH 32
00026 #define FDTABLE_SIZE block_size/DIR_ENTRY_SIZE
00027
00028 typedef struct{
00029     char file_name[MAX_NAME_LENGTH];
00030     int fd;
00031     int status;
00032     int mode;
00033     int occupied;
00034     int file_idx; // use this to access file pointer in pennFAT
00035     int dir_idx;
00036     int num_read;
00037     // int file_size;
00038 } FD_TABLE_ENTRY;
00039
00040 void init_fd_table(char* fs_name);
00041 void close_fd_table();
00042 int f_open(const char *fname, int mode);
00043 int f_read(int fd, int n, char *buf);
00044 int f_write(int fd, const char *str, int n);
00045 int f_close(int fd);
00046 int f_unlink(const char *fname);
00047 int f_lseek(int fd, int offset, int whence);
00048 int f_ls(const char *filename);
00049 int f_chmod(const char *filename, int perm);
00050 int f_getmod(const char *filename);
00051
00052 #endif

```

4.17 syscall_utils.h

```

00001 #ifndef SYSCALL_UTILS_H
00002 #define SYSCALL_UTILS_H
00003
00004 #include <stdint.h>
00005 #include "syscall.h"
00006
00007 int min2(int a, int b);
00008 int min3(int a, int b, int c);
00009
00010 int find_avail_entry_in_root();
00011 int create_new_file_pennfat(const char *fname, int mode);
00012 DirectoryEntry* find_curr_dir_entry(FD_TABLE_ENTRY *fd_entry);

```

```
00013
00014 void create_fat_entry(DirectoryEntry* dir_entry);
00015 int read_from_pennfat(int num, char* buf, FD_TABLE_ENTRY *fd_entry);
00016 int write_to_pennfat(int num, const char* buf, FD_TABLE_ENTRY *fd_entry, int status);
00017 void remove_from_pennfat(FD_TABLE_ENTRY *fd_entry);
00018
00019 int look_through_directory_blocks(const char *filename);
00020 int change_permission(const char *filename, int perm);
00021 int get_permission(const char *filename);
00022
00023 int get_begin_file_idx(FD_TABLE_ENTRY *fd_entry);
00024 int get_curr_file_idx(FD_TABLE_ENTRY *fd_entry);
00025 int get_end_file_idx(FD_TABLE_ENTRY *fd_entry);
00026
00027 #endif
```


Index

- 23sp-pennOS-group-12/src/kernel/helperfunc.h, [19](#)
- 23sp-pennOS-group-12/src/kernel/kernel.h, [19](#)
- 23sp-pennOS-group-12/src/kernel/logger.h, [20](#)
- 23sp-pennOS-group-12/src/kernel/parser.h, [20](#)
- 23sp-pennOS-group-12/src/kernel/pcb.h, [21](#)
- 23sp-pennOS-group-12/src/kernel/scheduler.h, [22](#)
- 23sp-pennOS-group-12/src/kernel/shell.h, [23](#)
- 23sp-pennOS-group-12/src/kernel/shellLList.h, [23](#)
- 23sp-pennOS-group-12/src/kernel/ufuncs.h, [27](#)
- 23sp-pennOS-group-12/src/kernel/uprocess.h, [28](#)
- 23sp-pennOS-group-12/src/kernel/usignal.h, [28](#)
- 23sp-pennOS-group-12/src/pennfat/global.h, [28](#)
- 23sp-pennOS-group-12/src/pennfat/interface.h, [29](#)
- 23sp-pennOS-group-12/src/pennfat/pennfat.h, [29](#)
- 23sp-pennOS-group-12/src/pennfat/pennfat_utils.h, [30](#)
- 23sp-pennOS-group-12/src/pennfat/syscall.h, [30](#)
- 23sp-pennOS-group-12/src/pennfat/syscall_utils.h, [30](#)

- all_procList
 - Scheduler, [15](#)

- blocked_procList
 - Scheduler, [15](#)

- blockPid
 - Scheduler, [15](#)

- children
 - pcb, [10](#)
- cmdContentPtr
 - job, [6](#)

- directory_entry, [5](#)

- FD_TABLE_ENTRY, [5](#)
- finishedJobs
 - jobSet, [9](#)

- head
 - jobList, [7](#)
 - procList, [13](#)

- idle
 - Scheduler, [15](#)
- idlePcb
 - Scheduler, [15](#)

- job, [6](#)
 - cmdContentPtr, [6](#)
 - jobid, [6](#)
 - pgid, [6](#)
 - printStatus, [6](#)
 - processNum, [6](#)
 - status, [7](#)
- jobid
 - job, [6](#)
- jobList, [7](#)
 - head, [7](#)
 - jobNum, [7](#)
 - tail, [7](#)
- jobNode, [8](#)
 - jobPtr, [8](#)
 - next, [8](#)
 - pgid, [8](#)
- jobNum
 - jobList, [7](#)
- jobPtr
 - jobNode, [8](#)
- jobs
 - jobSet, [9](#)
- jobSet, [9](#)
 - finishedJobs, [9](#)
 - jobs, [9](#)
 - stoppedJobs, [9](#)

- minus1_procList
 - Scheduler, [16](#)

- next
 - jobNode, [8](#)
 - procNode, [14](#)

- p_level
 - pcb, [10](#)
- parent
 - pcb, [11](#)
- parsed_command, [9](#)
- pcb, [10](#)
 - children, [10](#)
 - p_level, [10](#)
 - parent, [11](#)
 - pid, [11](#)
 - process_name, [11](#)
 - sleep_stop_tick, [11](#)
 - thread_state, [11](#)
 - ucontext, [11](#)
 - waitPid, [11](#)
 - zombie_children, [11](#)
- pcbPtr
 - procNode, [14](#)
- pgid
 - job, [6](#)

- jobNode, 8
- pid
 - pcb, 11
 - proc, 12
 - procNode, 14
- plus1_procList
 - Scheduler, 16
- printStatus
 - job, 6
- proc, 12
 - pid, 12
 - status, 12
- process_name
 - pcb, 11
- processNum
 - job, 6
- procList, 13
 - head, 13
 - procNum, 13
 - tail, 13
- procNode, 13
 - next, 14
 - pcbPtr, 14
 - pid, 14
- procNum
 - procList, 13
- runningLock
 - Scheduler, 16
- Scheduler, 14
 - all_procList, 15
 - blocked_procList, 15
 - blockPid, 15
 - idle, 15
 - idlePcb, 15
 - minus1_procList, 16
 - plus1_procList, 16
 - runningLock, 16
 - schedulerContext, 16
 - stopped_procList, 16
 - terminate, 16
 - totalticks, 16
 - zero_procList, 16
 - zombie_procList, 17
- schedulerContext
 - Scheduler, 16
- sleep_stop_tick
 - pcb, 11
- status
 - job, 7
 - proc, 12
- stopped_procList
 - Scheduler, 16
- stoppedJobs
 - jobSet, 9
- tail
 - jobList, 7
 - procList, 13
- terminate
 - Scheduler, 16
- thread_state
 - pcb, 11
- totalticks
 - Scheduler, 16
- ucontext
 - pcb, 11
- waitPid
 - pcb, 11
- zero_procList
 - Scheduler, 16
- zombie_children
 - pcb, 11
- zombie_procList
 - Scheduler, 17