PennOS API

File System - User Functions

f_open

Synopsis

Open a File Descriptor (User)

int f_open(char* fname, int mode)

Description

Returns a file descriptor (a non-negative integer), which can be used in f_write and f_read calls to write to or read from memory. The file descriptor itself is placed in the pcb file descriptor linked list for the process which is called f_open.

mode can be F_READ, F_WRITE, or F_APPEND

For F_READ the only call that can be made on this File Descriptor is

Return Value

On Success, the file descriptor is returned. On Failure, -1 is returned.

f_close

Synopsis

Close a File Descriptor (User)

int f_close(int fd);

Description

Closes a file corresponding to inputted file descriptor (that is non-standard input or output).

Return Value

On Success, 0 is returned. On Failure, -1 is returned.

f_read

Synopsis

Read from a File (User)

int f_read(int fd, int n);

Description

Takes in a file descriptor and an integer value of *n* bytes. *n* bytes will be read from the file referenced by fd.

Return Value

On success, the number of bytes read is returned (positive if any were read and 0 if the end of the inputted file was read). On failure, -1 is returned.

f_write

Synopsis

Write to a file (User)

int f_write(int fd, const char * str, int numb);

Description

Takes in a file descriptor, a string reference, and an integer value of *numb* bytes. *numb* bytes from the inputted string will be written to the file, and the file pointer of the file corresponding to inputted fd will be offset accordingly by *numb*.

Return Value

On success, the number of bytes written to the file is returned. Otherwise, -1 is returned.

f_lseek

Synopsis

Move the file position

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

Description

Takes in a file descriptor, an integer offset value of *n*, and a constant *whence*. The file pointer of the inputted file is reposition to the offset based on the inputted *whence*.

whence can be F_SEEK_CUR, F_SEEK_SET, or F_SEEK_END:

- F_SEEK_CUR: Moves the position of the file pointer by *offset* from its current position.
- F_SEEK_SET: Sets the position of the file pointer to *offset* (equivalent to moving the pointer by *offset* from the beginning of the file)
- F_SEEK_END: Offsets the position of the file pointer from the end of the file.

Return Value

On success, the position of the file descriptor (as an unsigned int) is returned. On failure, -1 is returned.

f_unlink

Synopsis

Removes the File fname

int f_unlink(const char * fname)

Description

Removes the file with name fname from the directory, and closes all file descriptors associated with that file. If the file can not be located in the directory, no file descriptors are closed, no file is removed, and -1 is returned.

Return Value

Returns 0 if succesful. Returns -1 if file not found.

f_dup

Synopsis

Make a file be STDOUT or STDIN

int f_dup(int oldfd, char* fname, int pid)

Description

Takes and oldfd, which must be 0 (STDIN) or 1 (STDOUT) and makes it now point to a file. Now, writes to STDIN or STDOUT will write or read from files, and not the terminal.

Return Value

Returns 0 for a succesful file descriptor change. Returns -1 for an unsuccesful file descriptor change.

f_ls

Synopsis

List all files in the directory.

char* f_ls()

Description

Traverses through all files in FlatFAT system and displays details on each file in the system (name, size, and block number).

Return Value

This function is a void function, so nothing is returned.

f_logout

Synopsis

Log out of the File System (User land)

void f_logout()

Description

Free all heap-allocated memory used by the file system prior to logout.

Return Value

This function is a void function, so nothing is returned.

f_defragmentSystem

Synopsis

Defragment the File (User land)

void f_defragmentSystem()

Description

Reorder the blocks in filesystem such that files are stored in the fewest contiguous regions (files are not located in separate, non-consecutive blocks). The function first determines an accurate ordering of the blocks, with regions ordered by the ordering of the first occurrence of each file's original first block. Then it moves copy of the file information to its new position determined by the defragmentation ordering.

Return Value

This function is a void function, so nothing is returned.

File System - Kernel Functions

FILESTATUS

Enumerations for Returns in Kernel File System: * FILENOTFOUND * MEMFULL * FILESYSERROR * INVALID_WRITE * INVALID_READ * FILEOK * FILEDELETED * FILEMADE

k_initDirectory

Synopsis

Initializes the Directory Structure

void k_initDirectory()

Description

Initalizes the root linked list which holds the file pointers.

Return Value

Returns FILEOK on Successful Initialization. Return FILESYSERROR on Unsuccessful Initialization.

k_initFdTable

Synopsis

Creates a new file descriptor table

filedescriptor* k_initFdTable()

Description

Initializes a file descriptor table with default values: * The file pointer will be at position 0. * It will indicate that is was opened once.

Return Value

Returns the pointer to the head of the initialized file descriptor table. Returns NULL if there was a malloc error.

k_mkFlatFat

Synopsis

Makes a Flat FAT Host OS File for Memory

FILESTATUS k_mkFlatFat(char* flatfs);

Description

Creates a new file descriptor table. The only open file descriptors on it are STDIN and STDOUT. This file descriptor's head is returned.

Return Value

Returns the pointer to the head of the new file descriptor table. Returns NULL on error.

k_addFile

Synopsis

Creates a new file

struct file* k_addFile(char* fname)

Description

Creates a new file with name *fname* (truncated to the maximum name length of 256) and size 0. It will be assigned the first available block number as its block start.

Return Value

Returns the pointer to the newly created file.

k_setFileSystem

Synopsis

Mount a FlatFAT file system

FILESTATUS k_setFileSystem(char* flatFat)

Description

Map a PennOS file system directly into memory (a file on the host with the name *flatFat*).

Return Value

Returns FILEOK on Successful Initialization. Return FILESYSERROR on Unsuccessful Initialization.

k_determineBlock

Synopsis

Determine the first available block

int k_determineBlock()

Description

Traverses through possible block numbers up to the maximum possible number of block entries, and returns the first number corresponding to a free block.

Return Value

Return the block number of the first free block in the file system, and MEMFULL if no blocks are free.

k_blockFree

Synopsis

Checks if a block is free

int k_blockFree(int n);

Description

Checks if block number *n* is within range of the maximum number of entries. If so, iterates through all files in system and go through all the blocks of each files to search if the block number is in use.

If the block not in use in any of the files, then it is free.

Return Value

Return TRUE if block *n* is free, and FALSE otherwise.

k_open

Synopsis

Open a File Descriptor (Kernel)

file* k_open(char* name, int mode)

Description

First searches for a file with a name matching the inputted *name*. If the name doesn't exist and *mode* is TRUE, a new file titled *name* is created.

mode can be TRUE or FALSE. TRUE corresponds to an f_open call with permissions to write to or append a file. TRUE corresponds to an f_open call with permissions to only read from a file.

Return Value

Returns NULL if no file was created. Otherwise, returns the pointer of the open (or newly created file).

k_read

Synopsis

Read from a File (Kernel)

int k_read(filedescriptor* filed, char* str, int numb)

Description

Read *numb* bytes from file descriptor *filed* starting at *str*.

Return Value

Returns the number of bytes that have been read. Return FILESYSERROR on an unsuccessful write.

Notes

Called from the user file system function f_read().

k_write

Synopsis

Write to a File (Kernel)

int k_write(filedescriptor* filed, char* str, int numb)

Description

If writing to STDOUT, then write *numb* bytes of the string *str* to STDOUT. Otherwise, if writing to a file, write *numb* bytes of the *str* across the blocks of a file.

Return Value

Returns FILEOK on Successful Initialization.

Returns the number of bytes written on a successful write to STDOUT or a file. Return FILESYSERROR on an unsuccessful write.

Notes

Called from the user file system function f_write().

k_ls

Synopsis

List all available Files

void k_ls()

Description

Traverses through all files in directory (top level of FlatFAT system) and displays details on each file in the system (name, size, and block number).

Return Value

This function is a void function, so nothing is returned.

k_loadDirectory

Synopsis

Load Directory from Memory

FILESTATUS k_loadDirectory()

Description

Reads from the flat file on the host and loads the bytes as files on our PennOS file system. Returns a file status based on whether files were able to be loaded from memory.

Return Value

Returns FILEOK on Successful Load of Directory from Memory. Returns FILESYSERROR on Unsuccessful Load of Directory from Memory.

k_cleanFat

Synopsis

Clear Up a File from A FAT file system

void k_cleanFat(int blockStart)

Description

Starting at *blockStart*, iterate through subsequent blocks of a file and clearing them of any links to other blocks (assign them all a link value of -1).

Return Value

This function is a void function, so nothing is returned.

k_storeDirectory

Synopsis

Stores into the Memory

FILESTATUS k_storeDirectory()

Description

Writes bytes into memory (the flat file on the host). Returns a file status based on whether the file was able to be stored in the directory.

Return Value

Returns FILEOK on Successful Initialization. Returns FILESYSERROR on Unsuccessful Initialization. Returns MEMFULL if all blocks are being used in file system.

k_fileSysLogout

Synopsis

Log out of the File System

void k_fileSysLogout()

Description

Free all heap-allocated memory used by the file system prior to logout.

Return Value

This function is a void function, so nothing is returned.

k_defragmentSystem

Synopsis

Defragment the File system (Kernel Land)

void k_defragmentSystem()

Description

Reorder the blocks in filesystem such that files are stored in the fewest contiguous regions (files are not located in separate, non-consecutive blocks). The function first determines an accurate ordering of the blocks, with regions ordered by the ordering of the first occurrence of each file's original first block. Then it moves copy of the file information to its new position determined by the defragmentation ordering.

Return Value

This function is a void function, so nothing is returned.

Standalone Executable File System Functions

These functions interact with the file system without having to be called from the shell.

k_mkFlatFat

Synopsis

Makes a Flat FAT Host OS File for Memory

FILESTATUS k_mkFlatFat(char* flatfs);

Description

Creates a file called flatfs which serves as the Memory of the OS. The root directory is initialized to only contain the root file, which starts in block 0 with size 0.

Return Value

Returns FILESYSERROR if Unsuccessful. Returns FILEOK if Succesful.

Notes

Used only by mkFlatFat. File must be formatted by begining of OS operations.

k_initKERNELFdTable

Synopsis

Initialize the file descriptor table

void k_initKERNELFdTable()

Description Return Value

Notes

Used only by catFlatFat. File must be formatted by begining of OS operations.

k_filesLogout()

Synopsis

Free heap memory associated to files

void k_filesLogout()

Description

Iterate through all files and free all memory allocated to structure elements of each file.

Return Value

This function is a void function, so nothing is returned.

k_fileDescriptorLogout

Synopsis

Free heap memory associated to file descriptors

void k_fileDescriptorLogout()

Description

Iterate through all file descriptors on the FD table and free all memory allocated to structure elements of each file descriptor.

Return Value

This function is a void function, so nothing is returned.

b_fileSystemLogout

Synopsis

Log out from the file system

void b_fileSystemLogout() (Binary File System)

Description

Frees heap memory allocated to the file descriptors on the FD table and the files themselves.

Return Value

This function is a void function, so nothing is returned.

Notes

Used only by catFlatFat. File must be formatted by begining of OS operations.

b_open(char* name, int mode)

Synopsis

Open a file by name (Binary File System)

int b_open(char* name, int mode)

Description

Opens a file *fname* with the mode *mode* and return a file descriptor. The *mode* can attain the following constants: * F_WRITE - writes onto the file and truncates if the file exists, or creates it if it does not exist * F_READ - open the file for reading only, return an error if the file does not exist * F_APPEND - open the file for reading and writing if it already exist but does not truncate the file; the file pointer also references the end of the file.

Return Value

Returns a file descriptor on success and -1 on error.

Notes

Used only by catFlatFat. File must be formatted by begining of OS operations.

b_write(int fd, char* str, int numb)

Synopsis

Write bytes to a file (Binary File System)

int b_write(int fd, char* str, int numb)

Description

Checks that file *fd* exists and that the F_WRITE/F_APPEND flag is inputted. Writes *numb* bytes of the string referenced by *str* to the file *fd*; increments the file pointer by *numb*.

Return Value

Returns the number of bytes written, or -1 on error.

Notes

Used only by catFlatFat. File must be formatted by begining of OS operations.

b_close(int fd)

Synopsis

Close a file

int b_close(int fd) (Binary File System)

Description

Free all heap allocated memory for the a file noted by *fd* in our file system before closing it. Return -1 if a file corresponding to the fd does not exist or if an invalid call to close STDIN or STDOUT was made.

Return Value

Returns 0 on success, or -1 on failure.

Notes

Used only by catFlatFat. File must be formatted by begining of OS operations.

b_read(int fd, char* str, int n)

Synopsis

Read bytes from a file (Binary File System)

int b_read(int fd, char* str, int n)

Description

Checks that file *fd* exists and that the F_READ flag is inputted. Reads *n* bytes from the file referenced by *fd* onto a buffer *str*.

Return Value

Returns the number of bytes read, 0 if EOF is reached, or a -1 on error.

Notes

Used only by catFlatFat. File must be formatted by begining of OS operations.

b_lseek(int fd, int offset, int whence)

Synopsis

Move the file position (Binary File System)

int b_lseek(int fd, int offset, int whence)

Description

Takes in a file descriptor, an integer offset value of *n*, and a constant *whence*. The file pointer of the inputted file is reposition to the offset based on the inputted *whence*.

whence can be F_SEEK_CUR, F_SEEK_SET, or F_SEEK_END:

- F_SEEK_CUR: Moves the position of the file pointer by *offset* from its current position.
- F_SEEK_SET: Sets the position of the file pointer to *offset* (equivalent to moving the pointer by *offset* from the beginning of the file)
- F_SEEK_END: Offsets the position of the file pointer from the end of the file.

Return Value

On success, the position of the file descriptor (as an unsigned int) is returned. On failure, -1 is returned.

Scheduler and Process Queue Documentation

Queue Functions

The process queue(s) structure is discussed in depth in the README.

set_quanta

Synopsis

Set time quanta for Processes in a queue

void set_quanta(queue *head, int quanta)

Description

Sets how many runs the processes a queue can undergo within each of their PCB's.

Return Value

This function is a void function, so nothing is returned.

add_to_active

Synopsis

Set inactive processes to active

void add_to_active()

Description

Iterates through all the queue nodes of inactive processes and set them to active by moving them to the queue of active processes.

Return Value

This function is a void function, so nothing is returned.

add_to_inactive

Synopsis

Add a process to the Inactive queue

void add_to_inactive(pcb_t *pcb)

Description

Takes in a PCB *pcb* and adds it onto queue node to be placed at the end of the queue of inactive processes.

Return Value

This function is a void function, so nothing is returned.

set_time

Synopsis

Set Time for Queues Based on Priority

void set_time()

Description

Depending on how many queues are present, set_time() determines the time quanta for the process such that each subsequent process

Return Value

This function is a void function, so nothing is returned.

next_to_schedule

Synopsis

Get the next PCB to Schedule

pcb_t *next_to_schedule()

Description

Checks for any inactive process to be scheduled and renders them active if so. Searched for the first process in the highest-priority active queue. If its run count has not yet equaled its quanta, then it gets moved to the end of that active queue. Otherwise, the process gets moved to the the inactive queue. The PCB of this process gets returned to be scheduled.

Return Value

Returns the PCB of the process that is ready to be scheduled. Returns NULL if there are no inactive process queues, no processes in the queues, or if the process is zombied.

scheduler_logout

Synopsis

Handles the Scheduler Logout

void scheduler_logout()

Description

Frees all heap-allocated memory associated to the scheduler's queue of active and inactive queues.

Return Value

This function is a void function, so nothing is returned.

Process Queue Functions

enqueue_q

Synopsis

Add a queue node to the end of a Queue

void enqueue_q(queue **head, queue *temp)

Description

Places the queue node indicated by *temp* onto the end of the queue indicated by *head*.

Return Value

This function is a void function, so nothing is returned.

enqueue

Synopsis

Add a process to the end of a Queue

void enqueue(queue **head, pcb_t *pcb)

Description

Creates a queue node that references the PCB of the process to be enqueued and places this node at the end of the queue indicated by *head*.

Return Value

This function is a void function, so nothing is returned.

dequeue

Synopsis

Take a process off the Queue

queue *dequeue(queue **head)

Description

Removes the queue node at the front of the queue off the queue.

Return Value

Returns the queue node of the recently popped node. Returns NULL if the queue is empty.

remove_by_pid

Synopsis

Remove a process from Queue by PID

void remove_by_pid(queue **head, int pid);

Description

Given a process denoted by PID *pid*, remove this process from the queue indicated by *head*.

Return Value

This function is a void function, so nothing is returned.

size

Synopsis

Get the size of the queue

int size(queue *head)

Description

Iterates through the nodes of the queue indicated by *head* and counts the number of nodes in the inputted queue.

Return Value

Return an non-negative number corresponding to the length of the queue.

Kernel and Shell Documentation

List of Shell Functions

Required: * cat [fname] (S*) cat a file to stdout * nice priority command [arg]\ (S*) set the priority level of the command to priority and execute the command * sleep n (S*) sleep for n seconds * busy (S*) busy wait indefinitely * ls (S*) list all files in the file system and relevant information * touch file (S*) create a zero size file file if it does not exist * rm file (S*) remove a file file * ps (S*) list all running processes on PennOS * nice pid priority pid (S) adjust the nice level of process pid to priority priority * man (S) list all available commands * bg [pid] (S) continue the last stopped thread or the thread pid * fg [pid] (S) bring the last stopped or backgrounded thread to the foreground or the thread specified by pid * jobs (S*) list current running processes in the shell * logout (S) exit the shell, and shutdown PennOS

Extra: * Extra 1 - mv - move/rename a file * Extra 2 - cp - make a copy of file * Extra 3 - df - display space left on file system (in blocks)

Kernel - User Functions

p_spawn

Synopsis

Fork a new child thread.

pid_t p_spawn(void *func, char *cmd, int argc, char *argv[], int state)

Description

p_spawn() forks a new thread that retains most of the attributes of the parent thread. Once the thread is spawned, it will execute the function referenced by *func*. The function will have name *cmd* and *argc* number of arguments which are stored in the array *argv*. The new thread will be specified to run in the foreground or background by *state*.

Return Value

On success, returns a new PID. Exits on error.

p_kill

Synopsis

Kills a thread.

p_kill(int pid, int signal)

Description

p_kill() kills the thread referenced by *pid* with the signal *signal*.

Return Value

p_wait

Synopsis

Wait until a child of calling thread changes state.

wait_t* p_wait(int mode);

Description

p_wait() sets the calling thread as blocked (and does not return) until a child of the calling thread changes state. The mode argument should be used to indicate a NOHANG condition. In this case, p_wait() should not block and should return NULL immediately if there are no child threads to wait on. If the calling thread has no children, p_wait() should return NULL immediately.

Return Value

p_wait() returns a structure, *wait_t*, with two fields: pid_t *pid*, the process id of the child thread that changed state, and status_e (an enum) *status*, indicating the state of the child.

p_exit

Synopsis

Exits the current thread.

p_exit();

Description

Exits the current thread unconditionally.

Return Value

Returns 1 on success, -1 on error.

p_nice

Synopsis

Set the priority level of a thread.

p_nice(int pid, int priority);

Description

p_nice() sets the level of the thread *pid* to *priority*.

Return Value

Returns 1 on success, -1 on error.

p_info

Synopsis

Returns information about a thread.

p_info(int pid);

Description

p_info() returns information about the thread referenced by *pid*.

Return Value

Returns a structure, info_t, which contains the following fields: status_e (an enum) *status*, indicating the state of the child, char* *command*, the name of the command executed by the thread (e.g., cat), and int *priority*, indicating the priority level of the thread *pid*.

p_sleep

Synopsis

Blocks the thread for a certain number of ticks.

Description

p_sleep(int ticks);

p_sleep() sets the thread pid to blocked until *ticks* of the clock occur, and then sets the thread to running.

Return Value

Void.

Status Functions

Synopsis

Checks the status to see if a thread is exited, stopped, continued, or terminated.

Description

int W_WIFEXITED(status_e status);

int W_WIFSTOPPED(status_e status);

int W_WIFCONTINUED(status_e status);

int W_WIFSIGNALED(status_e status);

W_WIFEXITED() returns true if the child terminated normally, that is, by a call to p exit or by returning. W_WIFSTOPPED() returns true if the child was stopped by a signal. W_WIFCONTINUED() returns true if the child was continued by a signal. W_WIFSIGNALED() returns true if the child was terminated by a signal, that is, by a call to p kill with the S SIGTERM signal.

Return Value

Returns 1 when *status* corresponds to the function's status, 0 otherwise.

Kernel - Kernel Functions

k_process_create

Synopsis

Creates a new child thread and PCB.

pid_t k_process_create(void *func, char *cmd, int num_args, char *args[], int fg_bg)

Description

Create a new child thread and associated PCB. The new thread retains most properties of the parents, but will execute the function referenced by *func*. The function will have name *cmd* and *argc* number of arguments which are stored in the array *argv*. The new thread will be specified to run in the foreground or background by *state*.

Return Value

Returns a pid_t that is the pid of the new child thread, with which you can access the PCB through the PCB table.

k_process_kill

Synopsis

Kill a process

k_process_kill(pcb t * process, int signal)

Description Kill the process referenced by *process* with the signal *signal*.

Return Value

k_process_terminate

Synopsis

Handle a terminated/returned process

k_process_terminate(pcb t * process)

Description

(K) called when a thread returns or is terminated. This will perform any necessary clean up, such as, but not limited to: freeing memory, setting the status of the child, etc.

Return Value

k_process_wait

Synopsis

Description wait_t *k_process_wait(int mode);

Return Value

Helper Functions - Kernel Level

void sig_alrm_handler();

void start_timer();

void pause_timer();

void go_to_scheduler();

void init_kern();

Signals and Signal Queue Documentation

Signals

S_SIGSTOP

Stops the thread that receives this signal.

S_SIGCONT

Continues the thread that receives this signal.

S_SIGTERM

Terminates the thread that receives this signal.

Signal Queue Functions

sigEnqueue

Synopsis

Add a signal to the end of the signal Queue

int sigEnqueue(signals_e sig, pid_t pid)

Description

Creates a node that takes in the signal *sig* and PID *pid*. sigEnqueue() then adds this node to the end of the the signal Queue.

Return Value

Returns 1 on a successful enqueue, and -1 if a calloc error arises

sigPop

Synopsis

signalNode* sigPop()

Description

Checks if the signal queue is already empty. If not, the signal node at the front of the queue is popped off and returned.

Return Value

Returns the recently popped off signal node. Returns 0 if the signal queue was empty.

Log Documentation

Logging Functions

init_logging

Synopsis

Initialize logging for PennOS

void init_logging(char *log_name);

Description

Creates a directory for logging if none exist by the name "log" yet. If a logfile of *log_name* has not yet been created, create a file that can be written onto.

Return Value

This function is a void function, so nothing is returned.

log_event

Synopsis

Log an Event onto the Logfile

void log_event(unsigned long ticks, char *func, pid_t pid, int nice, char *cmd);

Description

Print critical details about the event onto the logfile: * Tick count (time) * Invoked Function Name * PID * Nice Value * Process Name

Return Value

This function is a void function, so nothing is returned.

logging_logout

Synopsis

Handles Logger logout

void logging_logout();

Description

Close the logfile prior to logging out and quitting.

Return Value

This function is a void function, so nothing is returned.