Design Document
CS 560 - PA #1
By: Joe Dorris
Drew Masters

Project directory layout

bin - Folder that holds the executable shell

include - Holds headers that specify structs, function definitions, and macros

lib - holds the compiled library

obj - Object files

src - source code

Makefile - Will compile all the c files to object files, create the library, and then compile the shell.

remote - client and server.

Structs

File system - this will contain an array of inodes, an array of chars whose bits represent pages that are free or being used, an array of file descriptors, the index of the inode that is the root directory, and the index of the inode that holds the current directory.

Inode - struct that has a char which is 0 if it is not in use and 1 if it is, a char that is 0 if it is a regular file and a 1 if it is a directory, the current size of the file, an array of the direct links to pages, and

Directory - holds arrays of filenames and inodes associated with them.

File Descriptor - holds a pointer to an inode, holds a char that is 0 if it was opened for read and 1 if it was opened for write, a char that is 0 if it is not in use and 1 if it is, the current offset in the file, and the current byte in the disk file that that offset is associated with.

Constants

MAX_FILE_NAME_LEN - the longest filename allowable

MAX_SIZE_DIRECTORY - the maximum number of files that can be in a directory

NUM_DIRECT_LINKS - the number of direct links to pages available

NUM_INODES - the number of inodes available in the file system

NUM_FREE_LIST_BYTES - the number of bytes

DISK_OFFSET - the offset in the disk file to get to the actual disk

DISK_SIZE - the size of the disk

PAGE_SIZE - the size of a page

NUM_FILE_DESC - the number of file descriptors available

Functions for modifying data structures

void set_page_free(struct file_system *fs, long num)

usage - set the specific bit of the free page to unused

inputs - file_system struct and a long

outputs - nothing

void free_page(struct file_system *fs, long num)
usage - free up page that is no longer needed
inputs - file_system struct and a long
outputs - nothing
int find_first_free_page(struct file_system *fs)
usage - finds first free page in free list
inputs - file system struct
outputs - integer
void set_page_used(struct file_system *fs, long num)
usage - sets specific bit of the page to be used to 1
inputs - file system struct and a long
outputs - nothing
int get_inode(struct file_system * F)
usage - find first free inode, allocate it, and return its identifying number
inputs - file system struct
outputs - integer
int get_fd(struct file_system * F)
usage - find first free file descriptor in file system and return its identifying number
inputs - file system struct
outputs - integer
void free_fd(struct file_system * F, int i)
usage - free specified file descriptor
inputs - file system struct and a integer
outputs - nothing

Commands

void fs_mkfs(FILE * fp, struct file_system * F) - clear all the data in the file_system struct. Set the root and current inode. Write the new file system struct to the disk file and file the rest of the file with DISK_SIZE of some character to set it to the correct size. Write the root directory inode to have "." and ".." and write it back to the disk file.

void fs_mkdir(FILE * fp, struct file_system * F, const char *dirname) - read in the pages associated with the current directory inode. Check that the file name is not already in the directory. Get a free inode and get it a free page and write a directory struct with "." and ".." to it. Set all other files to be called "\0". Add the new directory name and the new inode to the current directory.

void fs_ls(FILE * fp, struct file_system * F) - read in the directory struct from the current directory inode and print all of the file names that are not null.

void fs_cd(struct file_system * F, FILE * fp, const char *dirname_const) - if the path starts with "/" then set current directory index to root directory index. Then find the next occurrence of "/" and make it the "\0" character. Then read in current directory and find the inode index

associated with the filename. Set the current index to that inode and if there is more path after the found “/” then recursively call cd with the rest of the path.

void fs_rmdir(FILE * fp, struct file_system * F, const char *dirname) - read in the current directory. Loop through the filenames and find the inode associated with the directory to be removed

int fs_open(FILE * fp, struct file_system * F, const char *filename, const char *mode) - find first free file descriptor in file system and return the index of it

char *fs_read(FILE * fp, struct file_system *F, int file_d, int size) - read in the specified number of bytes from the given file descriptor and set the offsets.

void fs_write(FILE * fp, struct file_system * F, int file_d, const char * w_string) - write the specified string to the file descriptor and set the offsets.

void fs_seek(FILE * fp, struct file_system * F, int file_d, int offset) - set internal offset of the file descriptor to be the specified offset and the external offset is calculated by dividing the offset by the PAGE_SIZE to get the correct direct page pointer and then add offset % PAGE_SIZE

void fs_close(FILE * fp, struct file_system * F, int fd) - call free_fd on the specified file descriptor to close.

void fs_tree(FILE * fp, struct file_system * F, int tl) - list all files in current directory and all directories below current directory such that all files in the same directory are indented the same amount. If it is a normal file the name and size are printed out at the current indentation level. If it is a directory than directory name is printed out, change to that directory, recursively call tree with incremented indentation level, and change back to parent directory

void fs_cat(FILE * fp, struct file_system * F, const char *filename) - open a the file to read and then fs_read the size of the file and print it. Then close the file descriptor.

void fs_import(FILE * fp, struct file_system * F, const char *srcname, const char *destname) - will get the size of the source file by stat, read it in, open a new file in our file system with fs_open and write the size of it to the new file with fs_write.

void fs_export(FILE * fp, struct file_system * F, const char *srcname, const char *destname) - will open the destination path file, and write the source path file to it by reading in the size of the source path file and printing it to the new file.

void fs_try_exec(FILE * fp, struct file_system * F, const char *filename) - this is called if none of the other commands were specified. It will attempt to find the file specified in the

current directory and if it is not there then it will print that the command is not recognized. If it is there then it will copy it to the host file system, set it to executable, execute it, and then delete the file.

sh implementation

Check if a filename is given (if not disk.disk is used) and read in the file system struct from the beginning of that file. Then print out the "sh" prompt in a while loop and listen for the commands. Then call the function that was specified. If a path is provided that is not the current directory then save the current directory, cd to the directory, execute the command with the new path or filename and then return to the previous current directory.

Remote notes - used sockettome library provided by Dr. Plank and then wrote a server and client in c.

Server implementation

./server port

usage - remote shell server process that is listening for connections on given port
inputs - port
outputs - none

- serve socket that is specified port, wait for a connection, dup2 the stdin and stdout, and then do fork and exec with our shell

Client implementation

./client.sh hostname port

usage - client program to connect to remote shell server on given port
inputs - hostname, port
outputs - none

creates a connection to the server using ncat