Design Document and Test Cases

AJ Sebasitan and Dustin Lapierre

October 15, 2016

1 General design

Structs will be needed to keep track of working directory. Shell will await user input and will execute specific C files based on input. Shell will quit once a specified word is entered (probably quit or exit)

2 Commands

2.1 Cat:

Arguments: file path (absolute or relative)

First will check to make sure that the number of arguments passed is equal to one. If it is, the function will move on to the next step, otherwise it will print an error message and exit.

Next it will check if the file or directory entered actually exists, if it does it will move on to the next step, if not it will print an error message and exit. To do this it will determine whether the path is absolute or relative (by checking if the path starts with /), then check the file at the given location using checkFile(). If the file either doesn't exist, or is a directory, an error message will be printed.

The file will then be opened and read line by line. If the file cannot be opened an error will be displayed and the command will exit. Otherwise the text will be displayed to the terminal using the standard output.

2.2 CD:

One argument (directory name or path)

Arguments: directory name or path First it will check to make sure only one or no arguments was entered, if so it will move on to the next step, otherwise it will print an error and exit.

If one argument was entered it will verify that the given directory is valid, then it will navigate into it. If the directory is not valid an error message will be displayed and the command will exit. If no arguments were given, the command will default to the home directory and do the same thing as if one argument was entered.

2.3 DF:

Ignores all arguments

When the disk is first mounted add up all free clusters on the disk. We could save this as a global variable. Whenever a new FAT table entry is added or deleted we will modify the variable to reflect it.

DF will simply print the variable.

2.4 LS:

Accepts one argument (either directory, file name, or nothing)

Using Dirent.h will be necessary, and we will store the directory data in a struct outlined in the header file.

The LS file will be relatively simple, as long as we keep track of the current directory somehow, maybe a shared variable. This will only be used if there is nothing passed into the function.

If the directory name is given we will print the directory passed in as an argument. This will loop through the directory struct and read and print every file.

If a single file is presented we'll only print the data for said file.

If the directory or file specified does not exist then an error will appear.

2.5 MKDIR:

Accepts one argument (directory name)

1.Checks to see if argument name exists in current directory.

2. Checks to see if there is space on the disk

3. If there is space allocate it to the new directory allocate it and place the file in the target directory.

2.6 PWD

No arguments.

Prints the path to the current directory (using the variable keeping track of current directory).

2.7 RM

Accepts single argument (allows for nothing to be placed there)

If nothing send message, If more than one argument quit and send message.

Check to see if x is file or directory.

If directory send error message and quit

If x is a file you can remove the file from the current directory.

This will free up the data sectors of the target file which will need to be reallocated.

2.8 RMDIR

Accepts single argument.

Checks to see if target directory exists / is empty.

If empty we can delete the directory and reallocate the data sectors it took up prior.

2.9 TOUCH

Accepts single argument of file name

Check to see if argument already exists or if there is no space. (both result in error and quitting)

If there is space on the disk allocate it to this file, and add the file to the target directory.

2.10 Reusable functions

CheckFile() would check whether a file exists and whether or not it is a directory. if the file both exists and is not a directory it will return 1, otherwise returning 0.

3 Tests

To save a lot of space it can be assumed we'll be testing all sorts of arguments issues. We'll make sure the commands only accept the given parameters and will not execute given the wrong parameters. To achieve this the tests should include multiple instances of each command run with multiple variations of input. This would include not only a couple variations on desired input, but also some possible incorrect inputs that should fail. Once we have this test list, we can simply run it against all the commands whenever we make a significant change and it will tell us if all the functions still work correctly and produce the desired result.

We'll also have Tests trying to create files and directories without any space available in the disks, to make sure the program will not try and write to disk space which is not available.

We'll have tests trying to move to directories which don't exist, which should yield no results.