Design Document and Test Cases

AJ Sebasitan and Dustin Lapierre

November 3, 2016

1 General design

The shell will store both the current working directory path and the First Logical Cluster for the current directory in shared memory. The shell will await user input and will execute specific C files based on input by forking a new process. It will then wait for the process to finish running before starting it's main loop over again. the shell loops infinitely until the user enters "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:

Arguments: a single directory name, relative path, or absolute 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.

First it checks the number of arguements. If two arguments were given it prints an error and exits. If no arguments were given, the command will simply move you to the home directory and exit. When one argument is given it verifies whether the requested path is absolute or relative. If it is absolute it will begin in root, otherwise it will begin from the current directory stored in shared memory. Using the FLC of the next folder in the path CD will then loop through one piece at a time to the desired location. During this traversal a temporary FLC variable will be modified, changing to -1 on a loop if it encounters an invalid spot in the path. The current path in shared memory will be modified with each loop to reflect the new position (but the original path will be saved in case of error). After the traversal has finished if the FLC is -1 then the path is invalid and the shared memory variables will return to their original values. Otherwise the FLC in shared memory will be set to the temporary FLC.

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 shared memory 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)

If nothing is input there are two possible functions to use. PrintRoot and PrintOther. If the FLC we passed through shared memory is 0 then we'll use printRoot and if it's anything we'll call printOther. These just print the filename, extension, filetype, size, and FLC.

When an arugment is passed things get more complicated. We will break down the arguments and remove the final part of the path. From there we'll concatenate the path back together and then call CD on the path minus the last part. Once we're in that directory we will Determine whether it's a file or directory and take the appropriate action.

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 path saved in shared memory

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

parsePath(file path) accepts a string as an input and tokenizes it using / as a delimiter. Returns an pointer to each char[] pointer created. Used to separate each element in the path into individual strings. fatsupport.c functions are used repeatedly throughout.

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.