INFO1910 Week 9 Tutorial

In this tutorial you will write a program simulating a simplified Linux terminal and a virtual filesystem for it to access. The files and directories of your virtual filesystem will be stored using files in your computer's real filesystem. You may find some of the string functions you write in the Week 8 tutorial useful.

Question 1

Write a program, `terminal.c`, which accepts and performs commands in a similar way to the terminal. Implement the following commands:

`touch <filename> <filecontents>` : creates a virtual file with given contents. If the second argument is empty, create an empty file.
`cat <filename>` : prints the contents of the given virtual file to standard out.
`rm <filename>` : delete the named virtual file.
`mv <filename1> <filename2>` : rename file with filename1 to have filename2.
`exit` : end the program. All the real files the program created for data storage should be deleted; use C's `remove()` function.

You will need to create several buffers to contain strings for the program to work with. Define some large maximum size for a string in the program, such as 1024.
When a file is created in the virtual filesystem, create a real file called <name>.file to hold its contents. Can you implement error checking to determine if the file to be read or changed actually exists?

Question 2

Expand your terminal program to handle directories also. All the virtual files and directories will each be represented by a single real file in your machine's filesystem.When your program is told to create a virtual directory <name>, it should create a real file <name>.dir which contains the directory's immediate parent (on line 1) and a list of all virtual files and directories inside it (on the subsequent lines). When the program starts up, the user will initially be placed in the root directory (/). Implement the following commands:
`mkdir <name>` : creates a directory on the virtual filesystem.

`rmdir <name>` : deletes a directory on the virtual filesystem.
`ls <name>` : lists all the files inside the indicated directory. If no name is provided, list all files inside the current working directory.
`cd <name>` : change current working directory to the one indicated. If the provided name is "..", move to the parent of the current working directory.
`pwd` : print out the full path (from root) to the current working directory.
`mv` : maintain previous functionality; but now can also be used to move files from one place to another.

Question 3

Can you get these commands to work with paths as well as filenames? You may store the full path to the current working directory; but it should also be possible to store only the name of the current working directory, not the full path, and get the path only when necessary.

Question 4

The system as it stands will not be able to have duplicate file or directory names, even if they exist inside different virtual directories, since they will be represented by real files with the same name. Can you implement error checking to warn the user when a file or directory they create would overwrite another? Can you devise a way to solve this problem and allow virtual files and directories in different locations to have the same name?

Hints

Building a terminal program with all of the above functionality is a large and complex task. You may want to continue to work on this in future weeks; completing the whole program will be excellent practise at using C.
Because the program has many interrelated features, make sure to write lots of tests. In particular, the input/output tests using diff outlined at the end of week 8's INFO1110 tutorial may be very useful, as you can create a script to run them all automatically, and thus easily re-check all your testcases every time you finish a new feature for your terminal program.