CS 415 Operating Systems

Project 1 Report

Submitted to:

Prof. Allen Malony

Author:

Nathan Gong

951892158

ngong

Report

Introduction

For this project, I was tasked with constructing a pseudo shell in which I would mimic some common Unix commands like ls (which lists all contents of a directory) and rm (which deletes a specified file, but is unable to delete a directory). This pseudo terminal is extremely simplified and unable to imitate a number of common commands that couldn't be replicated due to time/workload purposes.

The pseudo shell has two possible modes. One where the shell imitates an actual terminal that takes user input for the commands, and another where a file full of commands is read and the output is written to an "output.txt" file. The code is decent but not great at detecting errors, and the implementation is very barebones, but overall it does a good job at replicating the simple Unix commands it intends to imitate.

Background

There were many choices to be had during the project because while the goal is straightforward, the directions are abstract and the number of pathways that I could've taken was numerous. For instance, there were many cases where I could've used pointers instead of character arrays, but it made much more sense to utilize static memory instead of dynamic simply because there was no need to overcomplicate instances where buffer overflow was extremely unlikely and mallocing/freeing continuously didn't make sense.

There weren't any complicated algorithms/methods to be implemented or chosen in this project because the implementation was relatively straightforward in the direction, but another decision was the choice between switch case versus if/else if statements for identifying certain commands. Perhaps there was another way that was more efficient to implement the command identification part of my code, but I ended up with if/else if statements.

Implementation

Something that was a little tricky was getting the implementation of file names and directory paths correct. Because the project was unclear on how specific file names and directories would be when inputted for commands like mv, cp, and so on, I had to implement multiple cases for different formats. Similarly, for the ls command, I found that I had to check if the directory was essentially empty (only having the "." and ".." objects), and then not write anything if those were the only objects detected.

It was interesting how different the mv and cp commands ended up being in their implementation as well. Even though in theory they're extremely similar in their function, one required me to individually copy and write lines to another file while the other had a simple system call function (rename()) built in to move a file to another location, all because of the simple difference of deleting the original file or not.

The other commands implemented wound up being relatively easy compared to the mv, cp, and rm commands, but they still all had their complications I had to work through.

Performance Results and Discussion

I felt like my project performed pretty well in comparison to the example output and test script. The main issue that I wasn't able to fix before the deadline was the rm command, but other than that my commands did well when tested against the provided test script. I couldn't find the issue with the rm command, but I suspect it had something to do with remove() vs. unlink() and the ability (or disability) to remove files vs. directories.

Conclusion

This was a fun project that gave me a broader understanding about how terminals and shells work as well as how some commands might be implemented if I desired to expand this project in the future.