# CS 415 Operating Systems

## Project 1 Report Collection

Submitted to:
Prof. Allen Malony

Author:
Liam Bouffard
UO ID: lbouffa6
Duck ID: 951811278

# Report

## Introduction:

This first project challenged us with creating a simple shell that can either prompt a user for a command, or read a command from a file. To have the shell read commands from a file, the user simply has to run the program with an "-f" flag along with the input file name, for example "./pseudo-shell -f input.txt". Conversely, to have the shell prompt the user for input you simply run "./pseudo-shell" with no flags or input files.

Within our shell, we were tasked with creating the following commands using only system calls: ls, pwd, mkdir, cd, cp, mv, rm, and cat. The command "ls" writes out the contents of whatever folder the user is in. The command "pwd" writes out the path of the current working directory, "Mkdir" creates a new directory and returns an error if it already exists. "Cd" takes us into the directory given by its arguments. "Mv" command can either move files from one location to the next and has the ability to rename files too. The "rm" command of course deletes the file argument. And finally, "cat" will print out the file's contents.

## Background:

Within this project, one of the far more difficult aspects was string parsing. Our program needs to handle multiple commands in one line as well as a command with arguments. For example, if we received the line:

mkdir test ; cd test ; ls ;

We needed to first separate it by the delimiter ";" and then each sub section by the delimiter " " in order to capture each command and its arguments. This algorithm was of medium complexity and involved using a command line struct and using strtok_r. Using the example from above along with the delimiter ";", the command line struct would hold 4 members: mkdir test, cd test, ls, and Null. We would then need to run our string parsing algorithm again on each one of those elements with the delimiter " " to separate each command and its argument . Since we were challenged with creating these commands using strict system calls, I needed to read quite a few man pages. Since this was my first time reading the man pages, it took a little bit to understand what I was looking at. Some of the system call man pages I read included write, getCwd, remove, close, and a few more. A few more additional complexities include finding out if a destination path was either a file or a directory as well as changing the stdout stream to a file. This case arose specifically in the problem of creating the copy and move functions.

## Implementation

The largest problem I had was with the implementation of the copy function. I wasn't aware open() could take an entire file path and I was under the impression I had to cd to the desired folder, parse the file name out, and create it. Of course this added much more complexity in which I got tangled up in. After talking with the GE Alex, he helped me by explaining open() can take an entire path. With this in mind the solution was much simpler, the only challenge left

was to use the stat() system call along with S_ISDIR to differentiate whether a path was to a file or a directory.

# Performance Results and Discussion

My code runs perfectly except for 2 small details. I have a supposed memory error in main.c and my cat output has an invisible extra character. I looked for a while but couldn't figure out the reason why these errors were occurring. For the sake of time, I decided to turn the code with these problems as the rest run completely fine.

# Conclusion

What I've learned from this is to take time and read the man pages much more thoroughly. I would have saved many hours if I had paid more attention to the arguments open() takes. I learned many new functions/system calls that I didn't know existed, and thus learned how many of the high level commands I already use interact with the operating system (OS). I was also refreshed on tokenizing strings which is always handy.