

CS 415 Operating Systems

Project 1 Report Collection

Submitted to:
Prof. Allen Malony

Author:
Abie Safdie
asafdie
951912125

Report

Introduction

For our first project in CS 415 we implemented a shell-like program that utilizes system calls to achieve its functionality. System calls are a more direct way of communicating with the kernel. They are more efficient as they talk directly with the OS. Ultimately, our program has the ability to call a variety of shell commands, giving the user a way of giving the computer commands to manipulate their directories and files.

Background

This project gave me the ability to dive deep into system calls and the C programming language. To start, I utilized some built-in system calls to call simple shell commands like `pwd`, `change dir`, and `make dir`. But more complex methods to implement commands like `move`, `copy` and `delete`. I extensively used the linux man pages to familiarize myself with these system calls and their functionality. All together, the study of system calls helped me comprehend the topic of this project and create a program that followed specifications.

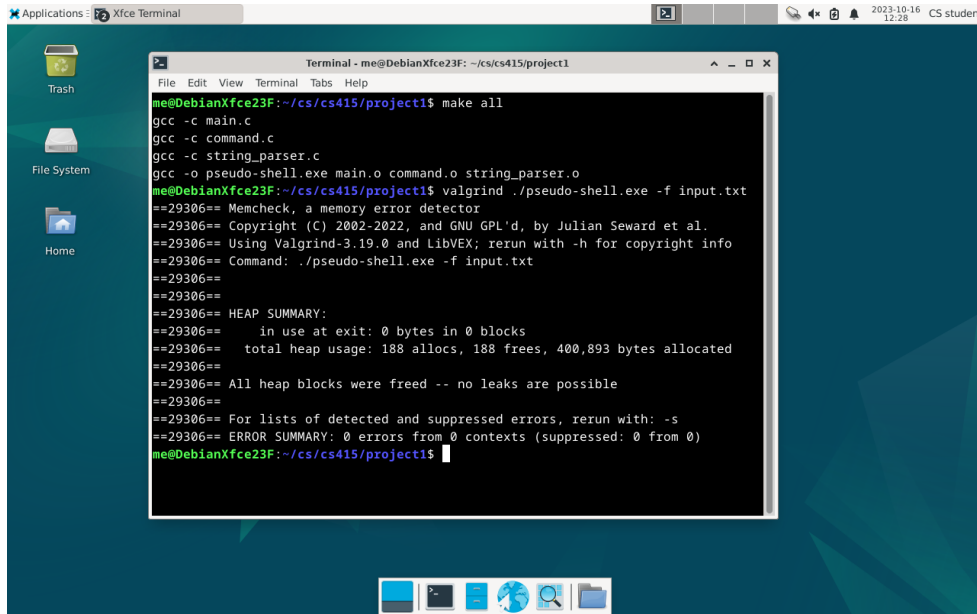
Implementation

To implement this program I created two distinct modes: file mode and interactive mode. To do this, I utilized a conditional checking whether the `-f` flag was specified. Then I found the file we intended to read from, split the input into readable tokens, then called the requested commands. In interactive mode, I split the input from `stdin` into readable tokens, then called the requested commands. In `command.c`, I implemented their functionality using system calls. For instance, for the command `moveFile`, I found the file we intended to read from, read and copied the file to the destination (using `read` and `write` system calls), then deleted the original file using the system call `unlink()`. This similar logic and use of system calls continued for the commands needed to be implemented.

Performance Results and Discussion

My program runs to the desired output performance. I have stress-tested my program and I believe that it runs error-free with the correct output in every scenario. Additionally, it leaks no memory and has no `valgrind` errors. One aspect where I use a non-system call (although I was told it is permitted) was the use of `opendir()` and `readdir()`. These files are listed in the lab2 for use, so I was under the impression they are fine for use. Attached is a screenshot of my program with the given test input using `valgrind`. I return an output file with the correct outputs and leak no memory and have no errors. The only problem I could see my program running into is if someone defines the input file as `"output.txt"`, then my program will write to a file that it is reading from and may cause undefined behavior. Additionally, on the rubric it states a case where the input uses the `"&&"` to run two commands. My program does not read `"&&"` and instead delimits at `","`, so it will not work when `"&&"` is used. However, it does not state in the rubric that it needs to work with `"&&"`, it only states it needs to work with `","`. Also, in file mode, if I read an unrecognized command, I don't read all subsequent commands on that same

line. So for instance, if the command was: `sldkfj; ls`; I do not read the `ls` command. However, if the `ls` was on a new line, then I would read it.



```

Terminal - me@DebianXfce23F: ~/cs/cs415/project1
File Edit View Terminal Tabs Help
me@DebianXfce23F:~/cs/cs415/project1$ make all
gcc -c main.c
gcc -c command.c
gcc -c string_parser.c
gcc -o pseudo-shell.exe main.o command.o string_parser.o
me@DebianXfce23F:~/cs/cs415/project1$ valgrind ./pseudo-shell.exe -f input.txt
==29306== Memcheck, a memory error detector
==29306== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==29306== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==29306== Command: ./pseudo-shell.exe -f input.txt
==29306==
==29306== HEAP SUMMARY:
==29306==    in use at exit: 0 bytes in 0 blocks
==29306==   total heap usage: 188 allocs, 188 frees, 400,893 bytes allocated
==29306==
==29306== All heap blocks were freed -- no leaks are possible
==29306==
==29306== For lists of detected and suppressed errors, rerun with: -s
==29306== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
me@DebianXfce23F:~/cs/cs415/project1$

```

Conclusion

This project gave me the opportunity to learn a ton about system calls and their functionality. I'm excited to move forward in this class and use system calls in future projects to implement bigger and more complex tasks. As well as to learn more about how a computer's operating system functions with regard to system calls.