Project Report

Group: Ali Awais Safdar/Huzaifa Liaqat

Introduction:

In the vast landscape of modern computing, operating systems serve as the foundation upon which all other software and applications are built. They provide a crucial layer of abstraction between the hardware and software, enabling users to interact with their computers in a meaningful and intuitive manner. One of the most prominent aspects of an operating system is the terminal, a command-line interface that allows users to execute various commands and control the system. Our project's basic idea is to allow the user to simulate an experience of using all the necessary, typical shell commands but instead those commands are manually coded by us.

Our project aims to delve into the intricate workings of operating systems, particularly focusing on the shell component, by developing a mini shell from scratch. By manually implementing shell commands in Linux, we will gain a deeper understanding of the underlying mechanisms that facilitate command execution, process management, and interprocess communication.

The primary motivation behind this project is to empower users with a comprehensive understanding of the internal mechanisms that drive the command-line interfaces we interact with on a daily basis. By delving into the nitty-gritty details of shell command execution, we can unravel the magic behind the scenes and demystify the complex interactions between users, shells, and the operating system. Concepts hidden behind built-in commands can be expanded on and realized to a greater extent by our stripping back of used libraries and built in functions and instead a more direct approach to the implementations.

Through this project, we were able to have the opportunity to enhance our knowledge of operating systems, Linux internals, and system programming. By writing code to parse commands, execute processes, and handle input/output redirection, we gained hands-on experience in low-level programming and system-level interactions. This project did not only sharpen our technical skills but also deepened our appreciation for the intricacies of operating systems that power modern computing.

The project involved a step-by-step approach, starting with a basic shell skeleton and gradually implementing various features and functionalities, such as command parsing, process execution, and handling background processes. We also explored the concepts of piping, input/output redirection, and signal handling, which are fundamental aspects of any comprehensive shell implementation.

Timeline, Chosen Prompt, Task Division:

Our chosen prompt was the second listed repository i.e., an implementation of a mini shell. Our vision for this prompt was to make our implemented commands general and practical, commands you would see in Linux otherwise as well, while not sacrificing our use of important concepts such as pipes and forking.

Our project took six days to complete with the bulk of the time taken being in practical implementation of the functions we had researched and planned. This research was conducted by Ali Awais, along with the debugging of all the functions and coding of half the functions, while Huzaifa handled the documentation and implementation of the other half of the functions.

Implementation:

Our project was implemented in C++ in Linux using a combination of virtual machines and dual boot. We implemented file handling commands, general shell-based instructions, and used a single pipe and forking in one command as well. A list of the implemented commands with a description of their functionality is provided below:

pwd: Print current working directory

help: Show available commands

exit: Exit the shell

date: Display the current Date

echo: prints a statement into the terminal

list: List files and directories in the current directory

mkdir <directory>: Create a new directory

change <directory>: Change the current Directory

rename <file>: Rename a file

perm: Show Permissions of a file

size: Show size of a file

count: Show line count of a file

read: Display contents of a file

pipe: Give Simple Working of Pipes by executing a forked ls - l command

write: Overwrite content to a file

append: Write content to the end of a file

copy <source>: Copy a file

rm <source>: Remove File

Our used libraries were:

iostream

cstring

fstream

unistd.h

cstdlib

sys/types.h

sys/wait.h

sys/stat.h

ctime

cstdio

dirent.h

Pitfalls:

While generally a smooth implementation, we faced a few complications in our design of the project. Firstly, we had a lot of trouble setting up our Virtual Machines due to unexpected errors and lack of experience dealing with them. We had to manually check host I/O device usage in the Controller: SATA tab in storage settings, which allowed for the machine to be functioning. Beyond this, the system ran slow at times and had issues opening and running the terminal as well. To solve all of this, we used a dual boot system instead which made the rest of our implementation relatively smooth.

Secondly, we had an error coming up with our write/append commands which made the system go into an infinite loop upon pressing the exit condition (CTRL+D). To solve this issue, we had to re-make and re-imagine the two functions entirely.

Conclusion

While having some issues and complications rise up with the timeline and implementation of our project, we ended up with a set of commands allowing the user to simulate a shell system which is in reality fully manually implemented. This allowed for us to gain significant experience and clarification regarding our course and also gain a full understanding of the workings behind the shell. Experience with pipes, filing and the shell is something extremely valuable for us as it allows for us to feel better equipped with the workings of our operating system; we leave the project with a feeling of having significantly developed our understanding of the course and coming to better grips with it as a whole.