Project Overview: Shell Command Line Interface Implementation

Summary

This project entailed the development of a command line interface (CLI) similar to a Unix shell, which enabled learning and implementing various aspects of operating systems, interpreters, and system programming. Throughout the project, knowledge was gained on parsing user input, managing processes, handling signals, and implementing built-in commands.

Key Learnings

1. Command Line Parsing and Interpretation

Parsing a command line requires splitting user input into separate tokens to be interpreted and executed. A command line parser was developed that was able to interpret complex command sequences, including pipelines, background tasks, and redirection of standard input and output.

2. Process and Job Management

The project involved gaining deep insights into how Unix-like operating systems handle processes. It included understanding process lifecycles, spawning new processes using fork() and exec(), and coordinating multiple processes concurrently. Also, a simple job control was implemented, providing an understanding of how Unix-like operating systems handle background and foreground jobs.

3. Signal Handling

This section of the project included learning about Unix signals and how to handle them in a program. In particular, it involved handling signals such as `SIGINT`, `SIGSTOP`, `SIGCONT`, and `SIGCHLD`, and ensuring the shell behaved correctly when receiving these signals.

4. Built-in Command Implementation

The project also involved the implementation of various shell built-in commands, including `kill`, `fg`, `bg`, `jobs`, `stop`, `exit`, as well as commands to navigate directories (`cd`) and manipulate environment variables. This allowed for a practical understanding of what functionality is usually built into a shell and what functionality is provided by external programs.

Moreover, two additional built-in commands were implemented – history and grep, a simple one and a more complex one. This not only allowed for creativity in the project but also presented a challenge in understanding what functionality could be effectively added to the shell, adhering to the Unix philosophy of implementing functionality using many small programs and utilities.

Conclusion

This project provided a comprehensive, hands-on experience on Unix system programming, enhancing understanding of operating systems, process management, signal handling, and shell programming. By developing a Unix-like shell from scratch, theoretical knowledge was reinforced with practical skills, providing a deep insight into the functionalities and behaviors of a shell.