MARMARA UNIVERSITY

FACULTY OF ENGINEERING

CSE 3033

PROJECT 2

150120052 – Kerem Hakkı Koç

150121065 - Turgut Köroğlu

150120995 - Seyyid Ali Koldaş

Introduction

This report explains the implementation of a shell application developed in C. The goal of this project was to create a functional terminal that processes user commands, manages processes, and handles input/output redirection. The project was divided into several key functionalities, each described below.

Command Execution and Process Management

The shell accepts user commands, parses them, and determines their location within the directories listed in the $PATH variable. By utilizing system calls like getenv, access, and execv, the shell ensures that the commands are located and executed correctly. During this process, careful error handling was implemented to ensure that invalid commands or missing files are appropriately reported to the user.

To manage processes, the shell uses fork to create child processes. Foreground processes are managed using waitpid, while background processes run independently, allowing the shell to handle new commands simultaneously. This dual management approach ensured that the shell could remain responsive while running multiple tasks.

Command History

The shell maintains a history of the last 10 commands entered by the user. The history feature is implemented using a circular buffer, enabling efficient storage and retrieval of commands.

Users can:

View the last 10 commands with `history`.

Execute a specific command from history using `history -i <index>`.

Additionally, the implementation handles edge cases such as invalid indices or empty histories to ensure robustness. This feature provides a practical way to recall and reuse previously executed commands without retyping.

Input/Output Redirection

The shell supports redirecting input and output to or from files. This includes:

Writing output to a file (`>`).

Appending output to a file (`>>`).

Reading input from a file (`<`).

Combining input and output redirection (`< input_file > output_file`).

System calls like open, dup2, and close were used to implement these functionalities. File descriptors were manipulated to replace standard input and output streams with files as needed. Detailed error messages guide users when file operations fail due to permissions or missing paths, improving the overall user experience.

Signal Handling

A key feature of the shell is its ability to manage interruptions. Using signal and custom handlers, the shell ensures graceful termination of foreground processes upon receiving signals like SIGINT. This improves usability and aligns with real-world shell behavior. The shell also prevents accidental termination of background tasks, making it more robust in handling user interactions.

Alias Management

The shell allows users to define aliases for commands, enhancing efficiency. Aliases are managed through simple commands:

Add alias: `alias <name>="<command>"`

Remove alias: `unalias <name>`

List all aliases: `aliaslist`

Aliases provide a way to simplify repetitive or complex commands by assigning them shorter names.

Sample Outputs and Use Cases

1. Running foreground and background processes with and without `&`.
2. Viewing command history and executing specific commands from it.
3. Redirecting output to files and handling errors with invalid file paths.
4. Using aliases to streamline commonly used commands.

Conclusion

This project successfully implemented a shell with functionalities including command execution, process management, command history, I/O redirection, and alias handling. It provided valuable insights into system programming, process management, and file handling. The shell's modular design ensures extensibility for additional features in the future.