**Project 3: Improved Unix/Linux Command Line Interpreter**

Atuhaire Ambala and Ricardo Escarcega

Prof. Isac Artzi

CST 315

June 2nd 2024

**Project 3: Improved Unix/Linux Command Line Interpreter**

The Unix/Linux Command Line Interpreter represents a refined shell program designed to execute user commands easily in both interactive and batch modes. This shell accepts user input, executes a wide array of commands, and handles the concurrent execution of multiple commands separated by semicolons. This foundational project sets the stage for a full-functional, full-featured shell by focusing on the essential aspects of basic command execution and signal handling. In this rendition, we have introduced the `History` feature, which displays the command history, and the `cd` command, which allows users to navigate through directories. The goal of this project is add more layers to build a full functional shell.

**Responsibilities**

Atu

* Documentation
* Feature algorithm design
* Code review

Ricardo

* Coding
* Documentation review
* Feature Research

**Methodology/Approach**

1. **Prompt:** The shell displays a custom prompt (`$lopeShell >`) and waits for the user to enter commands.
2. **Command Parsing**: The input is parsed to separate multiple commands using semicolons (`;`).
3. **Process Creation**: Each command is executed in a separate child process using `fork()` and `execvp()`.
4. **Concurrent Execution**: Commands separated by semicolons are executed concurrently. The shell uses `wait()` to ensure all child processes complete before returning to the prompt.
5. **Command History**: The shell maintains a history of executed commands, which can be displayed using the `History` command.
6. **Directory Navigation**: The `cd` command is implemented to allow users to change directories within the shell.

**Parsing and Processing Algorithm:**

1. Read Input: Read input from the user (interactive mode) or from a file (batch mode).
2. Tokenize Commands: Split the input into individual commands using `strtok` with semicolon (`;`) as the delimiter.
3. Execute Commands: For each command:

* Split the command into arguments using `strtok` with whitespace as the delimiter.
* Create a child process using `fork()`.
* In the child process, replace the current process with the new command using `execvp()`.
* In the parent process, wait for the child processes to complete using `wait()`.

**Features**:

1. **Interactive Mode**: Features a custom prompt (`$lopeShell >`) and waits for the user to enter commands

A screen shot of a computer code

Description automatically generated

1. **Batch Mode**: Supports executing commands from a file.

A screen shot of a computer program

Description automatically generated

1. **Concurrent Execution**: Allows the execution of multiple commands concurrently.

Commands entered by the user are parsed and tokenized. Multiple commands separated by semicolons (;) are executed concurrently

A computer screen with text on it

Description automatically generated

1. **Command History**: Maintains and displays a history of executed commands.

A computer screen shot of a program code

Description automatically generated

1. **Directory Navigation**: Implements the `cd` command for changing directories allowing users to navigate through directories within the shell

A computer screen with text on it

Description automatically generated

1. **Signal Handling**: Gracefully handles termination signals (`SIGINT` and `SIGQUIT`).A screen shot of a computer program

   Description automatically generated
2. **Process Creation and Command Exectuion**: Each command is executed in a separate child process using fork() and execvp().

A computer screen with white text

Description automatically generated

**Code**

Github Repo : <https://github.com/AtuAmbala/CST-315/tree/main/Project%203:%20Improved%20Unix%20Linux%20Command%20Line%20Interpreter>

A computer screen shot of a program code

Description automatically generated

A computer screen shot of text

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

**Testing and Validation**

Testing and validation of the Unix/Linux Command Line Interpreter involved a systematic approach to ensure its robustness and reliability. Interactive mode was thoroughly tested by entering various commands, including built-in commands like `cd` and `history`, as well as external commands. Multiple commands separated by semicolons were executed to verify concurrent execution and process handling. The command history feature was validated by checking the accuracy and order of stored commands. Batch mode testing involved creating batch files with multiple commands and observing the correct execution of these commands in sequence. Signal handling was tested by sending `SIGINT` and `SIGQUIT` signals to ensure graceful termination and cleanup. Edge cases, such as invalid commands and excessive command lengths, were tested to confirm proper error handling and stability. Overall, the testing phase confirmed that the shell operates as expected under various conditions and inputs, demonstrating its reliability and functionality.

**Code Execution**

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer

Description automatically generated