Rylan Casanova

02/26/23

CST-315

Project 1: Improved Unix/Linux Command Line Interpreter

**Explanation of Design:**

In this Project we were tasked with creating new features and improving the programming and overall design for the command line interpreter created in Project 1. In my improved CLI, I added a history feature which allows the user to use the up/down arrows to search for and execute previously executed commands. Additionally, I improved some parts of the parseCommand() function and quitting the program.

**Concept and Analysis:**

In order to accomplish creating a history feature for my CLI, I implemented the library readline.h This gives us access to reading previously executed lines, and outputting them to the user for execution. The history of commands executed in my CLI is stored in an array using the add\_history() function implemented by the readline library, providing command line editing functionality.

**Test and Validate:**

In order to successfully add a history feature, the readline library was implemented. This can be done by simply running the command “sudo apt-get install libreadline-dev“ in your Linux shell if it is not already installed. With some testing, I was able to utilize this library’s history functionality for my command line interpreter. It is difficult to demonstrate in screenshots to validate, so I attached a short video of executing multiple different commands and navigating through the history: <https://www.loom.com/share/a5f0af41cf254307ad57b9910d5e9fc9>

**Previous Project 1 Documentation:**

**Project description:**

In this project we are tasked with creating a CLI (command line interpreter) for Unix/Linux. The command line interpreter works as a mini shell, which prompts the user for an input command, creates a child process, and executes the command. Additionally, if multiple commands are entered separated by semicolons they are run simultaneously.

**Methodology/Approach:**

I chose to create my CLI in C++. In order to accomplish the tasks given in the project description, I created multiple functions. The first function is a parseCommand() function, which splits the input command into separate arguments. The executeCommand() function creates a child process and executes the command using the C++ commands execvp() and fork(). If multiple commands are entered separated by semicolons, they are executed simultaneously using the wait() function. The prompt is continuously displayed after each of the user’s commands are executed using cout <<. The user’s input is then read with the cin.getline() command.

**Algorithm for parsing and processing shell commands:**

To parse the shell commands I used the function I built called parseCommand(). This function splits the arguments of a string input into tokens, separated by the delimiter “ “ (space). To process the shell commands, the executeCommand() function is run. This function will output an error if there is no parent id (pid == -1), or if the inputted arguments are invalid/not found.

**Screenshot of execution:**

**Text

Description automatically generated**