CS433 Programming Assignment 2 Lab Report

Lucas Birkenstock, Alvaro Espinoza

Problem Description

Our task in this assignment was to develop a basic UNIX shell capable of accepting user commands and executing each command in a separate process. We were provided with a starter skeleton program along with a Makefile. We implemented a function which parses user input/commands into tokens, which also checks if the user provided input involves read/writing to/from a file. As the user is continuously prompted for input in a loop, our code parses the commands, handles possible file redirection, and forks a new child process in order to execute a command.

Program Design

Our program is designed as a basic UNIX shell, utilizing algorithms and data structures tailored for parsing user input, handling file redirection, executing commands, and managing processes. It uses a loop to continually prompt the user for input and execute commands until termination. The parse_command() function employs the strtok algorithm (a header from a library not implemented by us) to tokenize user input, efficiently extracting the command and its arguments. File redirection is managed by extracting filenames and using the open() and dup2() functions to redirect standard input/output. Command execution is facilitated by fork() and execvp(), ensuring proper synchronization between parent and child processes. Integer variables and pointer arrays are used for storing flags, numeric values, and command arguments, providing efficient storage and manipulation.

Strtok was used as our method/algorithm of string tokenization because it was the suggested method by the assignment instructions. Its methods are also very simple to use.

System Implementation

Our implementation is mainly described in the project design section. We implemented parse_command() in order to tokenize lines of input from the user, with the strtok library. We made sure to detect “<”/”>” characters in order to determine if the command was

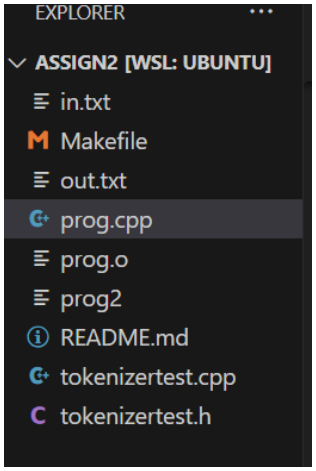
read or write. The read/write state is used later in the main function, adding process states to child processes as they are created. The main function then handles the commands appropriately, depending on several factors and conditional statements associated with them.

One of the main issues we encountered had to do with writing the output of the command to the out.txt file. Finding the exact cause of the problem was difficult to pinpoint, but we eventually discovered that we had to remove the file from the args list after it was parsed out.

Results

Our program passes all gradescope tests, and is functioning as expected when actually driving the code and interacting with it. This is an example of testing the ls command:

```
lucas@DESKTOP-92M2VGR:/mnt/c/Users/Owner/Documents/CS433/CS_433_Assigments/assign2$ ./prog2
osh>ls
Makefile  README.md  in.txt  out.txt  prog.cpp  prog.o  prog2  tokenizertest.cpp  tokenizertest.h
osh>cd out.txt
```



Conclusion

In conclusion, the project was a success. It met all of the criteria needed for the assignment. Our program mimics a UNIX shell; it can take in user command inputs and parse/tokenize them in order to create child processes and execute commands.