Programming Assignment 2 Report

Andres Hinojosa

Submitted Files:

- **prog.cpp:** Contains the code for the shell

How to compile and run the program:

- To compile the program, use command: make
- To run the program, use command: ./prog2

Results for prog2:

- My program was able to run successfully through each case that I tested successfully. The only thing it would have trouble with is a command such as “ls>out.txt” without any spaces is the command input

Features implemented:

- **Parser**: The first feature I implemented was the parser. It works by having a counter, a temporary command holder, and a while loop. First, the command is separated by spaces. The while loop basically removes each token, one by one while updating the counter and saving the token into the args array until nothing is left in the command array. Finally, the number of tokens is returned.
- **History**: The next feature that I implemented was the History feature. It works by storing the last used command in a separate string pointer called “history.” A Boolean variable called check is initialized as false when the program starts and changes to be true once the first command is executed to show the error if nothing is in the array.
- **Redirection**: The final required feature that I implemented was the Redirection feature. The easier of the two was redirection for input. I noticed that in the UNIX shell, typing a command with and without the '<' operator results in the same output. This led me to remove the operator for that case which worked. Redirection for output was more challenging. I first checked in which index the ‘>’ operator was and saved it into an integer. Next, I used the open() function to create or open a file with the name that is after the ‘>’ operator. I used dup2 to change the output, so it goes into that file. I finally executed the function that came before the operator which ended up working.
- **Pipe communication**: I implemented the extra credit pipe feature after everything else. I first checked for the ‘|’ operator and changed a boolean variable “pipeCheck” to true. Inside of the | case, I forked a child which will write to the pipe using dup2. In this child process, I call the first command. After that, I fork a second child process which will read the output from the pipe using dup2. Finally, the second command gets executed with the last output as the input.

Design and implementation choices:

I mostly went into the program while learning as I go, so my design could be better. The top of my main function initializes multiple Booleans, integers, and the pipe, and the history array. My while

loop starts by resetting each Boolean to be false and creating a pipe. Next, I get and parse the command into the args array. Before I get into the fork, I check to see if there is any special case by looking for a &, |, >, <, !!, or exit. This will cause the appropriate Boolean to become True so that the specific case will run in the child process. After the first fork, I have many if else statements to run the appropriate case, and at the very bottom is the regular case which is a simple execution.

Lessons Learned:

After learning from the previous assignment, I have started this assignment early which allowed me to debug and finish the program early. I also started to go to office hours whenever I got stuck on a problem which helped me to see certain cases that were failing. Since I am mostly familiar with C++, I also learned to look at documentation to learn how C-strings work. Something that caused some bugs were that I was checking for specific cases at first, so next time I would need my checks to be more general so that it works for every case and to prevent messy code.

References:

A reference that I used were official documentations and stack overflow for learning about C-strings, system calls, and their functions. I also watched some videos by a YouTuber called CodeVault who visually explains function calls and pipes to better my understanding of them.

Future improvements:

To improve my program, I would reduce the amount of code I have by organizing it. For example, I have two if statements which check for the same thing which is the '&' operator. I could combine them into one to reduce clutter all around my program. Since my program is finished and I know how everything works, I would further improve by optimizing the code.