Dennis Huang

dlh4fx

CS 4414-001

Operating Systems Homework 1 Write Up

Problem Description

I was able to complete the assignment and get my code working. The point of this homework assignment was to implement a UNIX shell written in C. Once the shell is started, it will be able to read in a command that is no longer than 100 characters. The program will then be able to parse the command into token groups to execute. The program also contains code that checks the command for any erroneous input and will not execute if an error is found.

Approach

The first step of the shell was to read in and validate user input. The user input was stored as an array of characters. Based on the homework guidelines, a command will never be more than 100 characters, and if it exceeded the limit, an error was thrown. An array containing valid characters was compared to the characters in the command, checking for any invalid characters. Next, the string token command was used to split the command by spaces, and each word was then stored into an array. The array was then iterated through to check the syntax of the command. Examples of bad syntax were given in the homework guideline, and all the cases were checked in the program.

The next step was to interpret the command. After going through main, if the command was deemed free of syntax errors and invalid characters, the interpret function was called. The function's parameters included the array of the words, the length of the array, and the number of pipe tokens found in the command. At first, the interpret function made sure a command without I/O redirection and piping was able to be run. A while loop was used so that the forks of the command would all run through the same process. With a command that had no I/O redirection and piping, there would only be one fork. The words of the command was added to another array called args. Next, the fork function was called. An if statement was used to determine whether the child function was currently executing. At the end of the statement, the execvp was used to execute the commands found in args. After the while loop, the wait function was used to wait until all the processes were finished.

Once a simple command was able to be interpreted, the next step was handling I/O redirection. In the while loop, once the commands have been added to args, a for loop was used to iterate through the array and look for the input and output operators. If found, a boolean variable (one for input, one for output) was set to be true. An if statement that checked the boolean was used to open or create a file, with the word next to the operator in the array being the file name. The indices that contained the operator and file name were then set to null value so that the command would not try to interpret these words as arguments. After fork was called, if statements that checked the input and output booleans determined which type of dup2 statement to be called. The dup2 statements determined whether stdin or stdout was being redirected.

The last part was to handle piping. The first part in this step was to create the correct amount of pipes needed for the command. Each pipe needed two spots: one for stdin, one for stdout. This was achieved by initializing an array with the size of the number of pipes times 2. Next, the pipe function was called in a for loop to dynamically create the pipes needed for the command. In the while loop, the array of the command was iterated through, looking for pipes. Once a pipe was found, several if statements determined where in the command the pipe was. If it was the first pipe found, then it was only necessary to redirect stdout. If it was in the middle of the command, it was necessary to redirect both stdout and stdin. If it was the last pipe, the it was only necessary to redirect stdin. Booleans were used to create the different scenarios for the pipes. After fork was called, the booleans were used to determine which pipe was used to redirect stdin or stdout. At the end, all the pipes were closed to prevent the program from hanging.

Results

The performance of the program was not slow at all. If it was, it was due to some error in the code that needed to be fixed. Since the program only deal with 100 characters of input at a time, it worked very well.

The shell runs very well. All the commands that are input in the native shell return the same results as the program. The program is capable of using multiple pipes, just like the one shown in the homework handout. One place to improve would be the error checking. As of now, if there is no syntax errors in the command, the program will try to interpret it and throw an error saying the command failed. An improvement would be to make the messages more detailed on why exactly the command failed.

Analysis

Before the program was working, multiple ways to try to get I/O redirection and piping were tried. At first, I had hard coded arrays and pipes that displayed I/O and piping just to see how it worked. The hardest part after doing this was to dynamically handle these two functionalities. Some steps that were attempted included brute force iteration and recursion. These two approaches could have worked, but after a while, I got very lost with these two methods and decided it would have been easier to try something else.

There were times when pipes were improperly closed, which left the program able to take in text but not functioning properly. This led to more debugging and fixing the way my code dealt with pipes. Other times, the pipes were not reading from or writing to properly, so after I coded the perror statement, it helped me see where exactly in my code the pipes were breaking. Often times I did not initialize enough pipes for the command, which led to the program not working properly.

Conclusion

This assignment helped me learn a lot about C. Before this class, I had experience in C++ and Embedded C. There were concepts in C that I had never worked with before, such as storing a string of characters as an array. Dealing with pointers and dereferencing was also something that I was not particularly good at, and it was necessary to learn how to properly do these for this assignment.

After getting the shell working, I now have a much better understanding of how the operating system handles command line arguments. It was interesting to learn about the fork and execvp functions in class and then write code to handle these functions.

Pledge

I pledge that I have neither given nor received any help on this assignment.

Dennis Huang

09/11/2016