Christopher Lebovitz

CSC 306: Operating Systems-G001

Programming Project 1

10/19/2020

This project aimed to build and implement a shell in a Linux system with a high-level programming language. For this project, I chose to use python for the project because python allows developers to interact with string and lists. There are three tasks outlined in the project,

1. Create a chile process and execute a command in the child and allow the parent and child to run
2. Provide a history feature
3. Adding support of input and output redirecting
4. Allow the parent and child process to communicate via a pipe.

Creating a child process for me was the most challenging because I first tried to use C to Implement the shell, and unfortunately, it has been a while since I've touched a C style language. The main problem I was having was reading input into a character array and figuring out how to pass those commands to the execvp function. It was around this time that designing a shell will be easier to do In python because of how easy it is to manipulate string in python, and from then on, it was pretty easy to implement the rest of the first task. In the run child process I check if the process id is equal to zero to see if it's the child process and if it is, that's where I call the execvp() command and pass it the command to execute. If it's not, then tell the parent to wait until the child is done. I created a function checking if the character limit has been reached and check if the user has entered anything. If not, keep asking for input until the user enters a command.

Also, in the first task, it says to consider when the user enters '&' or not, which allows the parent and child to run concurrently together. The only issue I kept running into Into was running the parent and child together. It throws off my output.

```
Python Shell>ls &
parent process running concurrently with child
Python Shell>exit  filename                header.txt  hello.c    output.txt  README   Work
file  final-src-osc10e.zip  hello       input.txt  ps.txt      Welcome
```

A way I was able to prevent this from happening was addring a shor wait time to the parent to be able to get the output to pring correctly.

It prints the output of the command on the same line that's asking for user input. Not sure if this is intended, but I have yet to find a workaround for the problem. In the concurrent function, I defined the process to check if the user has entered a '&' character by first checking the command list for that character, then removing it from the list at that position and returning true. The value is then used in the run child function to determine if the parent waits on the child or continues.

Creating a history feature was pretty straight forward. First, check if the user enters '!!'. If so, remove '!!' from the command list and add the last command in the last index in the history list. I found the last index instead of seeing it or keeping track of the number of items in the list using -1 in history[].

Python can let negative numbers be the opposite end of the list, making it easier to set command equal to the last element in the history list.

The redirecting input and output were a bit more challenging than the other two tasks because we had to send the console's output to a file and feed the content of a file to a command to be executed. To accomplish this, I used the dup2 function accessed by importing the os library in python. dup2 allows for the output to be written somewhere else, such as a file. It can also take input from a file and pass it to stdout in the console to be used by a system command. Then I use sys.stdout = sys.__stdout__ to clear stout to prevent anything else after the command to be written by the output file.

Communicating via a pipe at first was something relatively easy to implement since the only thing new I had to do was create a pipe and pass it between two child processes. Initially, I could get the pipe to work by having the parent process create two child processes and using pipes that allowed the two-child processes to send information to each other. Still, I realized that was wrong after I went over the requirement for this functionality. The assignment asks us to create a child and then have that child create their child to execute the first half of the command and then pass the output to the second half of the upper child's command. I was able to get the child process to create a grandchild process, but I was having issues thinking through the logic of how the command work in a child and grandchild process. After researching it a bit, I found out that it is supposed to work like a recursive function in a sense. The grandchild should handle the base case, so I had the grandchild execute the first half of the command then write the pipe so that the upper child can read and execute the rest of the user's command.

I learned a lot from this project, mostly working in a Linux system. I've always wanted to learn how to working in a terminal. Still, I never got around to it, or if I tried to run Linux on an old computer, I always gave up because I didn't know how to navigate the system. Doing this assignment not only taught me how to use system commands to make a shell, but it has also taught me how to work in Linux.