Cenci_Binici_Shell

Progress Report

Table of Contents

Early Setup

Joe created the folder structure and set up the github repository that both members pushed and pulled from on February 4th. Joe and Julian both worked to type up the book code and copied over the csapp.h file from the cmu.edu website in order to include csapp.h in the shellex.c file. Also looked at the basic forkexec.c files provided in class and tested with them in order to interpret how piping works and the way signals can be used to kill and reap children processes. Joe looked up how to create a Makefile and began to implement them instead of having to tediously enter in the gcc command every time along with the files to be compiled and linked. The Makefile took some time and Professor Ordille helped with the small issue of include the LIBS for -pthread at the top of the file because the executable file that was created could not be run until the gcc command was entered for lsh. But after inserting the statement to include LIBS = -pthread the Makefile worked perfectly and allowed for easy compilation after making modifications in the code. Joe explained Makefiles to Julian afterwards and both have a solid comprehension of how to create and use them effectively.

Statistics

Joe and Julian both researched various ways to perform statistics and were lost on how to actually include them. After discovering the beauty of usage and ru_*, Joe implemented stats through the built-in commands method that can be interpreted in the order that the user had specified through the minishell input. Joe worked on formatting the statistics so that they would be printed out in a clean and ordered fashion. Joe had to modify much of the code after realizing

the stats could be printed in different order depending on what was requested by the user. Joe finished up statistics on February 14.

Environment & Variables

Joe and Julian both researched and implemented variables at the beginning but quickly ran into some issues with having to reset a variable to a new value if it was already instantiated. Joe kept troubleshooting how to modify an environment variable and eventually discovered that the integer value at the end of the setenv command determined if it would overwrite the variables existing value if it had one. Joe changed the 0 value to a 1 and just like that environment variables could be changed. Joe also worked on parsing the $ symbol so that the program would know to retrieve the value associated with the specified variable. This value would then be returned to the user. Working on echo was a simple inclusion after having fiddled around with variables for so long and both Joe and Julian had figured out how to parse the $ symbol and produce a return value accordingly.

Execve VS. Execvp

For the longest time, both Joe and Julian couldn't figure out how execve would deal with the new environment variables and continuously received command not found errors from the system. This caused both to question the legitimacy of their code elsewhere and did not make the connection that the execve command would not produced the proper values. Joe stayed after class one day and Professor Ordille explained that the execvp function was what they were looking for. This simple change helped to stop the ongoing circling of changing code elsewhere

and allowed the two to continue on with the rest of the project. Execvp would not require the inclusion of the /bin/ prior to the desire command that the user wanted to run.

Signals

Joe and Julian both researched ways to handle signals. While Joe was implementing piping, Julian worked on implementing the signals. This included the handling of the SIGINT and SIGTSTP signals when the corresponding Ctrl-C and Ctrl-Z commands were input into the shell. Several issues occurred pertaining to where these signal calls were executed. Julian was eventually able to solve these issues. Rereading Chapter 8 and looking at pages 744-745 helped to clarify the way we wanted to modify the default the behavior.

Piping

The parsing of the piping took a long time as Joe forgot that he had already implemented basic parsing of the pipe symbol using strtok prior to the rest of the project. Joe was originally jumping around in memory and using malloc to ensure that no segmentation faults would occur while checking array locations in string commands. But realizing he had already broken up a pipe input into several commands he worked to set up a char* array[] that would save all of the ordered commands and tried to figure out how to pass the pipe commands properly and fork at the same time. Both Joe and Julian put their heads together and tediously worked on myriad of variations in order to see what the issue was. After an innumerable amount of failures and different results that seemed close but incorrect Joe emailed Professor Ordille for help. Joe got even closer but still couldn’t get it to pass appropriately to the shell. However Joe was able to perform piping without tags (ls | sort → works) (ls -l | sort → does not completely).

Jobs

Jobs were one of the items we wanted to implement but did not have the time to work on them effectively. We wanted to focus on piping and signal handling, which we did not figure out until the last day, before we worried about venturing onto job handling.

If We Had More Time

Unfortunately there were several features that we we're unable to fully implement due to time constraints. Our program lacked a job lists to track the status of processes currently running in our shell. This meant that we were unable to include the bg and fg commands to continue a stopped processes after the Ctrl-Z command was passed. We also wanted to better implement signals in a more effective manner. This goes hand in hand with the job lists and process management. We also wanted to clean up and comment out the code effectively so that others looking at the code could get a better interpretation of the purpose of each method and line. We wanted to ensure that despite lacking some functionality that the project required, we could make an organized project that if it were returned to in the future it would be easy to pick up again.