CMPE 322: Project 1- C Sells C Shells by the C Shore Report

Ahmet Fırat Gamsız 2020400180

In this project I aimed to develop a shell that behaves like zsh as much as possible. I took the liberty of making design choices if a part was not clearly defined in the description. I will go over my implementation by following the execution flow.

At the beginning I set the signal handlers. SIGINT handler is set in a way to replicate the zsh behavior on ctrl-C, shell doesn't quit and just prints a new line. SIGQUIT and SIGTERM handlers are logging signal name and exit properly. These were implemented for the sake of learning.

Then, shell reads user input, saves the user input as the candidate for the last executed command (Last executed command always saved as user input rather than formatted executable version of it since zsh saves commands this way in the history). Then I remove quotes from user input (only double quotes, single quotes were not mentioned in description). If you inspect how zsh handles quotes, there are very few cases they make difference. I could find only three: First one is "echo a". zsh says command not found. Second, echo hello > "spaced name.txt". zsh creates a file called spaced name.txt but my shell creates a file called spaced. The third one is alias a = "stuff" &. zsh starts a background process for alias command but my shell saves a = stuff &. I decided these cases are negligible. However, there are more edge cases that my shell matches with zsh, for example ls "-a"hello is executed as ls -a -l by both shells.

Then I tokenize the input, read it token by token and check for redirection and backgrounding. Then if the command is exit, I save the aliases and exit. Saving aliases here might be redundant but it is more failsafe. There is another diversion from zsh here: if you type exit > a.txt in zsh it creates a.txt, I don't.

Then I check for the second built-in command alias. I check the alias syntax, save the command as last executed and create the alias. I simply keep the tokens after = sign and append it to the alias array. I also print the whole array to a text file. This might create I/O overhead, however; I designed this way for simplicity and dependability. These saved aliases are loaded when shell starts. I use same parsing mechanism, just read from the file and call create alias function for each line.

Third built-in command is bello. Bello prints given list of information. Since others are trivial only current number of processes requires explanation. Here I use ps command with shell's tty. I get the line count of ps using system function (I don't use fork and exec here since it would just require duplicated unnecessary code). I decrease the output by 5. This number might be OS dependent, but it just removes extra processes. When there are no background processes it outputs 0 (shell's itself is not counted, just a design choice). Also reverse redirected background commands create 2 processes.

Then the code checks whether user command exists in aliases. If found, it adds arguments user entered to command and send the full list for execution. The final part is execution. First, I search through the path to find the user command executable. If found, execution function forks, if it is not a background process parent process waits for child and child process runs execv function. If wait returns zero exit status last executed command is set to user input. Also signal handlers are modified here (for zsh behavior). If it is a background process parent just logs the pid of process (zsh behavior) and doesn't wait. Collection of zombie processes is done by SIGCHILD handler. This handler prints pid of collected process and inserts a new line for clarity. Also last executed is directly changed to user input here since shell can't know future execution status. Redirection is handled by simply redirecting stdout / printing to a file. However reversed redirection and backgrounding constitutes a special case. Here after parent forks the child, child also forks a grandchild. This is a must to have a nonblocking behavior without multithreading. Child creates a pipe to record grandchild's output. Parent doesn't wait for child, but child waits for grandchild and grandchild executes the command. When grandchild completes, child prints output reverse by reading from the pipe. Also, when this type of command is executed and finished two pids printed to command line (zsh prints one).

When the whole flow is completed, loop returns to start and waits for user input.