Report of the Shell Project

Main Approach: In the myshell.c code, the one and only code file where everything has been done, there is a main loop, which after the start of the program, gives an input prompt to the user and waits for an input indefinitely. Depending on the input of the user, the program processes the input or just directly exits the program. In case of “exit” command or exit-signal (Ctrl-D), the program terminates. In case of no input, a new prompt was opened. Otherwise the input is being taken into a function named parser, which takes the input, splits the input into tokens, and depending on the tokens, executes the according task (if possible). In case of an error, the function catches the error and returns, while setting an error flag to 1, which indicates that an error has occurred during the execution. After the parser function has finished, depending on our error flag, we store that input in a string named lastlyExecutedCommand if there is no error. Apart from the main code, there is a file named alias_config_file.txt, which holds the variable names and their corresponding commands which were assigned with the alias command in our shell, so that we can reuse these aliases even after restarting our shell. Also, there is a file named bello.txt, which holds the output of the lastly executed bello command. That file is used for redirection to standard output. The code’s input capacity and reverse append capacity are both 1024 characters. During task execution, for commands inside the path, apart from “>>> &” case, only one child is created, otherwise 2 children are created. For other commands, no child is created. The output redirection at >>> was done by using pipes. The grandchild was not counted at the number of total processes as it is part of a single process which is “>>> &”

Some Major Problems Occurred While Writing The Program: My first major problem was about using the corresponding command of the aliased variables. When I assigned a command to a variable by alias and then called that variable, the program always gave an “invalid argument” error. After a long search, I realized that the problem was, when I store the corresponding command in alias_config_file.txt, the last character of the corresponding command of a variable is ‘\n’ as it was written onto a file. But when using the corresponding command, I directly took the command from the file with ‘\n’ at the end, and as it was an invalid argument, the program gave an error. When I changed that ‘\n’ with ‘\0’, the problem was solved.
My second major problem was about the processes at the background. At first, I mistakenly implemented background processes just as foreground processes, as the main process always waits for its child even though I tried to directly exit from the first child and let the grandchild keep executing. This didn’t work unfortunately. I removed the “parent waiting for its child process” part and that issue was fixed. A related problem also occurred while trying to count the number of processes currently being executed. For foreground processes, waitpid function worked well, but at the background processes, neither wait not waitpid functions in the fork-exec parts worked because all these functions reap the child process and as it was difficult to increment the number of reaped child processes inside the fork-exec parts, I didn’t call any of these waiting functions at the parent. The parent just waits for 1 second for its child (in order to avoid race conditions) then returns its child’s process id, then the child keeps going. Instead of collecting the “number of reaped child processes” information inside fork-exec, I wrote a while loop at the beginning of my parser function, so when an input was taken, the parser function first checks the number of “recently terminated background processes” and increments the value stored at reaPointer (the pointer which holds the address of an integer which stores the total number of reaped processes, including both foreground and background processes) by 1. By removing wait functions at the fork-exec parts of background processes and putting a while loop for checking terminated child processes, the counting was successfully done.