Programming Assignment 2 Report

Hüseyin Alper Karadeniz

Operating Systems (CS 307) / Fall 2022-2023

1 Program Flow and Design Decisions

I used C++ programming language for the implementation of this homework. My program flow and my design decisions are explained in detail in this report. First of all, my program begins with reading "commands.txt" file line by line, so that it detects the existing information of each command such as command name, input, option, redirection, redirection file name, and background job enable status. It stores all these detected information of each command into a vector of vectors of strings. After storing the details of each command, my program writes all of them into a file called "parse.txt" in the dedicated format in the description of the homework.

For each command, there is an algorithm performing necessary operations according to the some specific details of the command, my program decides what to do for the case. In order to track the current status of processes running in the background, my program uses a vector of integers, which stores the ids of the threads running in the background. Also, there is another vector of threads, which stores the threads being created and currently running. These two vectors are used when waiting is needed, so in the termination phase of the program, and also in a wait status. Therefore, I have control on currently running processes and have the ability to solve the possible concurrency and synchronization issues. During my program executions, all commands are executed with execvp() function, but they are stored and saved with some different ways, which are explained in the later parts of this report.

If the command is a "wait" command, then my program waits for all background operations to be completed with waitpid() function, and also it joins the threads with join() function, ensuring that all threads have completed. After wait operations, it clears both of these two vectors in order to prevent some errors resulting in concurrency problems in the upcoming steps.

If the command has an output redirectioning part, then a child process is created by the shell process with fork() function. In the child process, the descriptor is released and assigned to the opened file. Thus, execution result (output) of the executed command will be written to the dedicated output file.

During this time, the shell process has two options: if background job is enabled, the shell process adds this running process to background operations vector; however, if background job is disabled, the shell process waits for the process to be completed with waitpid() function.

For all other commands, my program creates a pipe, and then creates a child process. In each of these commands, there is a unique pipe providing information exchange between the child process and the shell process from its write end and read end. In the child process, if there is no redirection, then with the help of dup2() function, by using the write end of the pipe, my program writes the result of the execution (output) to the pipe. Else, if there is a redirectioning part, meaning that there is an input file to read to get input rather than getting it from the console, file is opened from its file name and the result of the execution (output) is written to the pipe with the same method used before. During this time, the shell process creates a thread. And, it has two options: if background job is not enabled, then the newly created thread is joined immediately, so that the shell process will wait until the thread execution in the child process to be completed. Else, if background job is enabled, then my program adds newly created thread to the vector of threads, it will have a concurrency controlled with this vector.

After starting all command operations, so that at the end of the command loop, before the termination of the program, my program waits for all processes and threads to be completed, ensuring the stability of the overall program.

Besides, while printing outputs of the executions, the read ends of the pipes are used inside of the threads. And, in order to ensure that the outputs of the threads will not intervene with each other, my program uses mutex structure in its printing function in its critical section. After printing, fflush() function is called, so the effects of these print statements will be immediately visible.