CS 307 PA 2 Report

Emir Asal 27933

I. Reading & Parsing

At the beginning of my program, I read the commands and parse it using a function I implemented. I created a struct named 'commandLine' to put the information of the command lines which I've read. Parsing works in a way that explained below:

Index position is set 0 at the beginning.

- First word is directly assigned to command field of my struct and position index is incremented.
- Later it is checked if the next position word contains any of the following: '-', '<', '>', '&' if they do now we know that it is the Input field. If it is the input field. Index position is incremented if not counter stays the same. (Repeated for every step)
- Else if next position contains '-' it is marked as Options field.
- Else if next position contains '<' or '>' it is marked as Redirection Field. (Also file name of the redirection handled here)
- For the last we check if next position contains '&' and set if it is background job.

Length of this command line is also counted so that when creating array for 'execvp' command I will use it.
At the end, struct with the information of the command line is returned to main.

Because of the way my parsing is done I did not need anything extra to handle the special character '-' for execvp command.

II. Command Executions (Pipes & Threads)

Before starting I created two global vectors for processes and threads to keep track of the background processes. In this program I mostly used vector since they are easier to work with than arrays.

After parsing is done, every command is stored in a vector and executed one by one from there.

- Firstly, we check if the command contains '>' if it does, we create a new process. In the child process I assign everything inside vector to an array and run the execvp command. In the parent process, it is checked if the command is a background process if it is 'pid' is pushed into the process vector and continued. If it is not a background job, the child will be waited using waitpid().
- Else if it is a 'wait' command 'waitpid()' function is called for each process inside the processes vector. Furthermore, 'pthread_join() is called for each thread inside the threads vector.
- Else if the command does not have output redirecting, we initialize a pipe, thread and a new process. Inside the child process we make sure that output will be done to a pipe using dup2() and information inside the command line is transferred to an array for 'execvp' function. In the parent process (shell) we initialize the thread with 'threadFunctionForPrint' in this function we acquire the mutex and read from the pipe and print everything to the console. At the end of the thread fflush() is used and mutex is released.
  After shell process creates the thread, it checks if it is a background job if not: process and thread is pushed into a vector and program continues but if not: both thread and process will be waited.

  In this program new pipe is created for every console output command.

  At the end, when commands are executed program will wait for every thread and process to finish using 'pthread_join()' and 'waitpid()' since they are stored in a vector.