Anıl Şen - 29556

CS307 PA2 REPORT

“execvp” Options

Regarding the arguments passed to “execvp” function, I have not encountered any issue, therefore, I have passed the arguments, especially options, directly without any further modification.

Redirectioning

For redirectioning, if there is an output redirection, there is no need for parent to open a pipe since the output provided by the execvp function in child process is going to be in the specified output file. To ensure this, I have used dup2 function to set the default output file descriptor as the file specified after the redirection symbol. By doing that we can see the results of execvp in the output file.

However, if there is an input redirection or no redirection at all, we want the child process to write to the write end of the pipe which is created before the parent shell forks, so that in the parent process, we can listen to these writings by sending threads.

If there is an input redirection, by using “dup2” function, I have set the input file as the default input in order for execvp to get the input from that file. After that, to print the results to the write end, I have also set the default output as the write end of the pipe by using “dup2” for execvp to write the results to the pipe.

If there is no redirection at all, setting default output as the write end of the pipe is enough to make sure that after execvp, the results will be visible by looking at the read end of the pipe.

Background jobs bookkeeping

In my program, if the command is a background job, the job’s pid is added to the “pids” vector so that once the command is wait, this vector can be iterated over and all the jobs with these pid’s can be waited by using “waitpid” function. Later, I empty out the list, so that for the next background jobs, the list will be up-to-date and no already waited background jobs will stay in that vector.

While we are waiting for these jobs to finish, it is said that we need to also make sure that the assigned threads for the jobs should also be finished. Therefore, after waiting for these child processes to finish, we wait also for threads to join. Even though it is not stated in the document, I prefer waiting for other threads belonging to non-background jobs if there is any since one does not expect to see jobs still running after wait statement in a command line.

Using a pipe per command

In my program, I have created a new pipe per command. Since for each iteration in the lines of “command.txt” I have used “int fd[2];” depending on the redirection situation as explained

in the chapter *Redirectioning.* It turns out that I am creating a new pipe for each iteration rather than a single one throughout the program.

Operation of Threads

At the end of each line in “command.txt”, I add threads to “threads” vector. And while adding to the vector, I also immediately start them. It is crucial to have a vector while starting them since in the wait command it is expected to wait for threads to finish as well as their processes. The threads use the “readAndWriteToConsole” function with an integer parameter representing the read end of the pipe, so that they can create a file stream using this integer and read the read end of the pipe as long as there is something written after they get the mutex.

Mutex Implementation

I have used “<mutex>” library to benefit from the mutex class. I have created an instance from mutex class named “mutex_obj” and declare it as a global variable since it must be shared among the threads of the parent shell. After ensuring that it is shared, I have used “lock()” and “unlock()” member functions of that class to have mutual exclusion. Each thread is attempting to lock the mutex by using “lock()” method and once they finish their printings, they release the lock by using “unlock()” method.