Report - OS Lab 3 - 150050031

1. Given the tokens, first goal was to execute simple linux commands using execvp(). execvp() automatically searched in the PATH variable so we don't need to pass the absolute location of linux binaries. For that I made the exec_builtin() function which forks the current process and calls execvp() in the child. The parent waits for this this time for the foreground process to finish. If the command is not found in PATH directories, it is reported.

2. Next aim was to get `cd` command working. For this I wrote the change_dir() function which checks for the correct format and then tries to cd into the given directory. If the directory is not present and error is printed.

3. Next is to get redirection working. For this we need to get the file to have its file descriptor as that of STDOUT. Also now a little pre-processing is required on the command, so I wrote execute_one() function. It checks if `>` is present in the command. Then checks for proper format and if everything is good it used dup() and dup2() to backup stdout, and restore it. During this time the child process write to file descriptor 1 which now belongs to the file we opened.

4. Next is to get sequential commands working. For this I separated the commands in the list of sequential commands and called execute_one on them one by one. Since the parent waits for each child to complete, the running is sequential.

5. Now let's get parallel commands running. Using similar algorithm for parsing as for sequential commands, I separated the commands. Now we need to tell the parent not to wait on these processes since they're running in background so I use the `parallel` flag.

6. Note that redirection is managed in each of the individual commands separately.

7. Two things left now. Managing `exit` and SIGINT (Ctrl+C). For SIGINT, the foreground process in sequence should stop running so I used a ctrlc flag which stops execution of current child and those further coming up in the sequence are not executed. Now the tricky part. SIGINT should not go to background processes. For this I made a dummy background process which is always running and which has a separate process group than our shell. All the background processes will be put in this group. So SIGINT will not affect background processes.

8. For exit, we need to close all background processes, so I killed the whole group of background processes and exit() from the shell.

9. Another thing was to print something on the completion of all background processes. So since at a time only 1 foreground process can be running, I saved it’s ID and didn’t print for it in SIGCHLD handler.

10. For handling SIGINT in parent (so shell doesn’t close on Ctrl+C), I used a SIGINT handler which does nothing.

x---------------------------------------x