CS307 Operating Systems PA2 Report

Hasan Berkay Kürkçü 27851

Firstly, I created a command class with default constructor because we do not have fixed number of commands so it would be painful to decide what of command we are dealing with and be space inefficient. Note that I also keep a record of command entry numbers to be able to work with fixed char* of execvp.

Then for commands with direction ">" all I did was close stdout and open the file given as input so that it is placed instead of stdout and now write end is that output file instead of terminal and of course pipe was not needed here because we do not need anon file or read and write ends to that anon file.

For commands with no direction or "<" first of all note that I created pipe different than default approach like int fileDesc[2] but I used malloc and dynamic allocation. First I used struct and keep readEnd in it but It was complex and pointer approach for our case seemed much clean because we need to send real pipe object through thread print function. Then one critical point is dup2(fileDesc[1], STDOUT_FILENO); line where child process directs output to write pipe's write end instead of terminal. When it comes to parent process another significant part is close(fileDesc[1]); line It did not work without it because I think parent needs to assure that write end is closed and signal that to thread print function otherwise EOF may not be encountered but I am not sure. Then we assing a job to our previously created thread to work for printint and we gave &fileDesc[0] which is read end of pipe. Also for file directioning first I wanted to give pipe's read end as given input file but It did not work, so I just tried to give file as arg to execvp and it worked straight-forward.

For pipe and thread design choices, I created pipe and thread for each command because we have "&" option which is background process and with one pipe (I am not sure if it is possible to implement it, seems not) I believe it would be really complex to close and open that one pipe so this is much clean and reasonable. For threads, you already explained why we need multi-threaded approach to provide shared-variable for mutex and I followed it.

For waiting background processes and threads I followed a really straight way by putting them into array of course with correct types accordingly and then just iterate through the array, and use wait_pid or pthread_join for each object.

You also wanted us to explain how to escape special characters, in PA1 there was a problem but here I did not get any error and they work as expected I think which may be about c vs c++.