Programming Assignment 2

Bilal Berkam Dertli 29267

In this programming assignment, I wrote cli.cpp file to create a shell simulator where shell reads commands and creates threads/ child processes to execute them.

Commands were given in a file called “commands.txt”. From that txt file, I first extracted the commands, considering in what category of a command could a specific word be. (cmd_name, option, input, redirectioning, background job &)

First, my program reads every line, and categorizes whitespace-seperated string in it. These strings are put into a vector, and when the line ends, I gather every line’s command in a vector of vector of string. I simply push back all vector of strings to it when a line ends.

After getting the necessary commands, the program flow is as follows:

Before iterating over the list of commands, I created **a global mutex that will be shared among threads of the shell process.** Additionally, I decided to **hold background tasks (both processes and threads running in the background) in a vector** with their id, and when the incoming command is “wait” (which does not include any input/option/redirectioning/ background task, as indicated in the file), program would iterate over these two lists, and use **waitpid for background processes and join for background threads to wait.**

Get the command.

> If the command is wait, don’t create a child but wait for all child processes and threads **(waitpid and join)** until they finish their execution.
>
> If the command is output redirectioning:
>
> > If this command is NOT a background task:
> >
> > > Shell: Fork a child and wait for it
> > >
> > > Command: close stdout, open the output file, execute the command.
> >
> > If this command is a background task:
> >
> > > Shell Fork a child, DO NOT WAIT, just add it to the background tasks list
> > >
> > > Command: close stdout, open the output file, execute the command.

NOTE: In case of output redirectioining, synchronization of writing to the same file is out of the scope of this homework, so no need to create a pipe. Also, at this point I realized that **behavior of command process does not change whether it is a background job or not.**

If the command is input redirectioning:

In this case, **piping is needed to synchronize console output**. **For every shell and its child (command process), a unique pipe is allocated from the heap before fork call.**

Shell: Close write end of the pipe since shell threads will not write to it but read from it.

If this command is NOT a background task:

Call join() for thread, wait for the thread to finish its execution

If this command is a background task:

Add thread to the background threads vector

Command:

Close the standard input, since this command has input redirectioning, open the desired file to read from, call **dup2 for write end of the pipe and standard output and execute the command so that instead of directly writing to the console, write to the write end of the pipe for synchronization of outputs**.

If the command has no redirectioning:

In this case, **piping is again needed to synchronize console output**. **For every shell and its child (command process), a unique pipe is allocated from the heap before fork call.**

Shell: Close write end of the pipe since shell threads will not write to it but read from it.

If this command is NOT a background task:

Call join() for thread, wait for the thread to finish its execution

If this command is a background task:

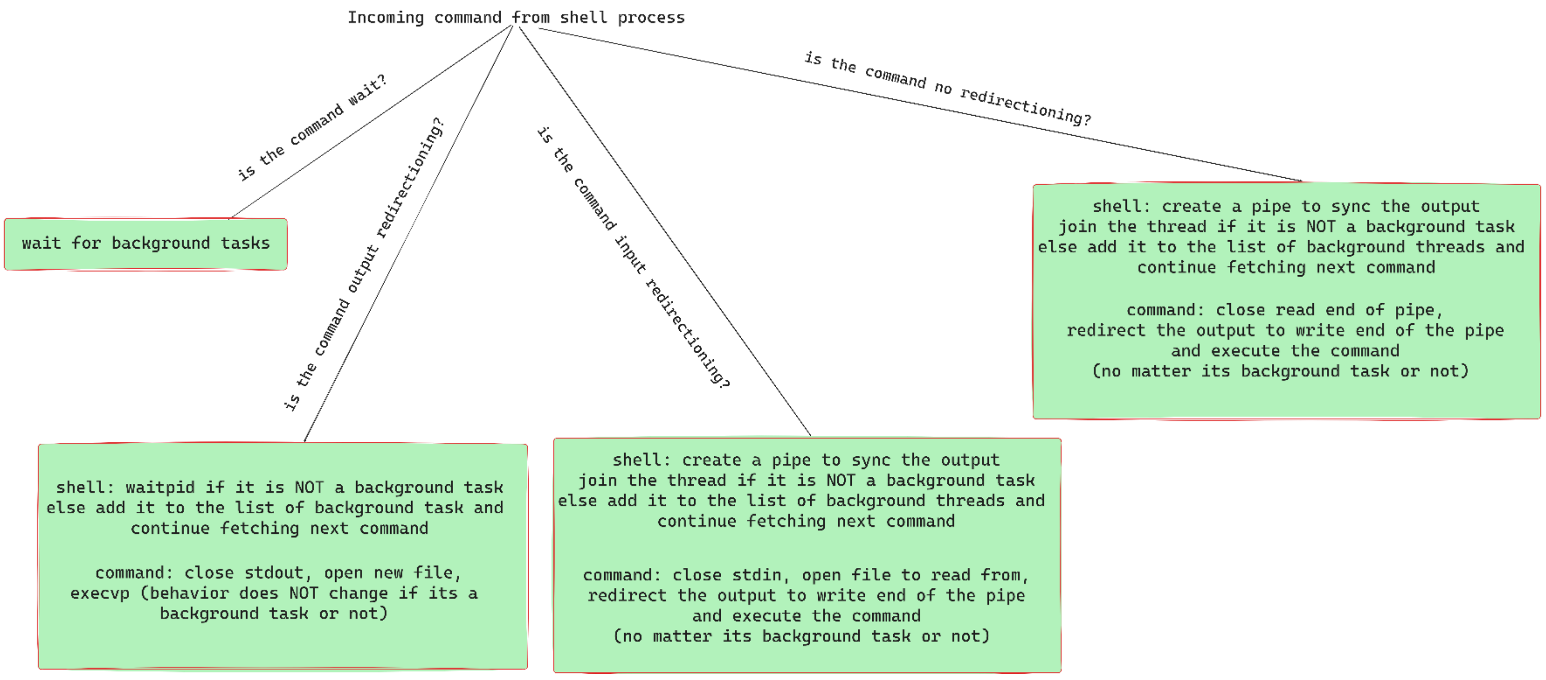
Add thread to the background threads vector

Command:

Close the read end of the pipe since we will not read from it. Call **dup2 for write end of the pipe and standard output and execute the command so that instead of directly writing to the console, write to the write end of the pipe for synchronization of outputs.**

After executing all of the commands, shell process waits for all background tasks to finish just like an incoming wait command.

In short, when a command is fetched, the following logic is implemented:


Incoming command from shell process
is the command wait?
wait for background tasks
is the command output redirectioning?
shell: waitpid if it is NOT a background task else add it to the list of background task and continue fetching next command
command: close stdout, open new file, execvp (behavior does NOT change if its a background task or not)
is the command input redirectioning?
shell: create a pipe to sync the output join the thread if it is NOT a background task else add it to the list of background threads and continue fetching next command
command: close stdin, open file to read from, redirect the output to write end of the pipe and execute the command (no matter its background task or not)
is the command no redirectioning?
shell: create a pipe to sync the output join the thread if it is NOT a background task else add it to the list of background threads and continue fetching next command
command: close read end of pipe, redirect the output to write end of the pipe and execute the command (no matter its background task or not)

Creation of pipes: I created a pipe for input redirectioning and no redirectioning commands between the shell process and command (child of shell) process for synchronization. Pipes are allocated from the heap every time, so they are unique per command process. Pipes are used for synchronization of console output. For synchronization, command process writes to the write end of the unique pipe between it and shell process, while threads of shell process reads from the read end of the unique pipe.

Creation of mutex: There is only 1 global mutex used for all of the shell process' threads since mutexes provide synchronization between threads of one process. Threads of shell process are using lock and unlock methods to prevent garbled output in the console, and fflush() is called to make sure that output is printed to the console.

Creation of threads: If there is a need for synchronizing the output (which is the case for input redirectioning and no redirectioning), threads are created for printing to console. In case that the command needs synchronizing, if it is a background task, shell process adds this thread to background threads vector and continues fetching. If it is not a background task, shell must wait for the thread to finish, so shell joins the thread before fetching next command.

How to wait for all background tasks? In my implementation, I have two lists: one for background threads, one for background processes. Background processes are kept in a list by their id, and when they should be waited, this list is iterated and waitpid is called over all processes of the list. Similarly, for threads, join is called for all of the background threads that are kept in a list if they should be waited.

How to Handle Escape Character: For grep command, whether it has redirectioning or not, an input could be starting with dash (-) which would be interpreted as an option, not an input. In order to handle this, I added -e option right after the grep which indicates that the input which comes after the -e option should be interpreted as an input, not a command. This -e option is indicated in the man page of grep command.