CS 307 PA1 Report

ls command lists the content (files and folders) of the current directory in a lexicographical order. --size option is for listing the items with the information of their sizes. It also gives the total size of the directory. I picked this command and option because I reckon it provides a basic but a significant insight in an easy and a fast way. The information provided is especially useful for monitoring disk usage or for identifying large files that could be taking up valuable space.

For the grep command I have used the -A 1 option to append the following line that continues covering the explanation of the ls --size command. Also, I have added two dashes, (--) which state the end of command options, to imply that the following words will be strings that grep command needs to search for. In this way, I have avoided the special character problem.

In the implementation of the pipe simulation of this command and option, the parent process, in another words "SHELL process," serves as the root of the process hierarchy. First of all, it opens an output file descriptor out of the file output.txt and it creates two pipes. Then it forks off a first child process, known as the "MAN process," which executes the man ls command. The output of this command is written into the write end of the pipe. Without waiting for the MAN process to end, the SHELL process forks off a second child process called the "GREP process." This second child reads from the read end of the pipe and executes the grep -A 1 -- --size command, sending its output to output.txt. Finally, the SHELL process waits for these two children to complete, closes all remaining file descriptors, and exits.

Note: To make sure the man process' print line is executed first, the code uses a secondary pipe (pipe_fd2). After the MAN process prints its line and before executing its main command, it writes a message ("Man comes before grep!") to this secondary pipe. The GREP process, before printing its line and executing its command, reads from this secondary pipe. This ensures a synchronization point between the two processes. The GREP process will block on its read operation until there is something to read from pipe_fd2. This mechanism ensures that the MAN process always has the opportunity to print its line before the GREP process does.

My program is 2a. My man and grep processes have sibling relationship because they have been both forked from the shell process. My man and grep processes can run concurrently because both processes are created as child processes of the parent (SHELL) process using the fork() system call. After the MAN process is forked, the parent does not wait for it to complete before it forks the GREP process. This allows both processes to be in the running state simultaneously. The operating system's scheduler can then schedule both processes for execution, allowing them to potentially run at the same time if multiple CPU cores are available. Even if the system has only one core, they can be context-switched, giving an illusion of concurrency. The interprocess communication through pipes ensures that data flows from the MAN

process to the GREP process, and this data flow is managed by the operating system's pipe mechanism without explicit synchronization in the code.

Note: While both processes can run concurrently, the GREP process is inherently dependent on the output of the MAN process for its input. Thus, while they might be active at the same time, the GREP process might spend some of its time waiting for input (page) from the MAN process as MAN process send the manual pages one by one and GREP process catches and uses them one by one.

Hamit Efe Çınar
30925