Mert Kılıçaslan

Shell Command Execution Simulation in C (PA – 1) Report

CS307 - Operating Systems

Command simulation: man ping **|** grep ping -A 5 -e -A > output.txt

man command: ping

grep options: -A 5 -e -A

The ping command is used to check the capability of the main computer to reach a specific computer. It's generally used as a path to confirm that a computer can communicate over the network with another network device. This is achieved by sending ICMP Echo requests and waiting for responses from them. Two main components of these responses are how many responses are returned and how long does it takes for them to return.
I picked ping as my man command because the way of communication between network devices is very interesting and exciting to me.

In the grep command, the first '-A 5 'option is to indicate that there will be five trailing lines after matching the option that I am looking for since its description takes five lines. The ‘-e’ option is to protect the next pattern beginning with "-" to accept it as a search string rather than an option. And the last option ‘-A’ is my main search option on the man page of the ping. After its detection in the man page, its line and the following five lines (description) are directed to the output.txt. I picked ‘-A’ as my search option I think because it’s a very simple and fundamental option. It prints the num lines of trailing context after matching lines.

Process Hierarchy

Firstly, I include all the necessary files to my code such as ‘unistd.h’, ‘sys/wait.h’ and ‘sys/types.h’ to create pipe and child processes. Shortly after I print the parent id to the console. Then I continue with opening the pipe and checking if it is successfully opened. By using pipe() I will have a communication channel between children and the parent when I will create children. Therefore, I will earn the ability to write command from one end and read it from the other end.

Then, I create the first child process by using fork() and check again if it is successful or not and now I have two processes which are a child and the parent process. Since I want to first execute man ping, I go to the child process by checking if the current id is zero and if it is that means the current process is a child. Now I am in the child condition block, therefore I print the current id (id of the first child) to the console. After that, I use the dup2(fd[1], STDOUT_FILENO) to take the standard output of the ping to write to the pipe. I close the read end of the pipe which is fd[0] since I don’t need it. Then; for executing man ping command, I use execvp which takes commands as an element of an array and executes them. Execvp does replace everything in the executing process (child in this case) because of this, the process will be replaced by another program. After the condition block ends, everything can only be executed by parent processes. After the first child’s condition block terminates I use waitpid(first_child_id) to wait for the first child to return 0 in its program. Because I will create another child and want to make sure that they are all in a process hierarchy.

After the first child return 0, the wait command continues where it left off. For the grep command, I create my second child by using fork() and check it again. I continue with checking if the id is zero since I want to use only the second child here. Now I am in the second child's condition block, I print the second child’s id, and call dup2(fd[0], STDINT_FILENO) function. Because grep is waiting to read from the pipe and it should read from the reading end which is fd[0], and therefore I replace that with standard input. I close fd[1] because the second child process doesn't write anything. After that, I open an output.txt file, to redirect all the output to a file instead of the console. For this purpose, I use the same implementation with the first child condition block where I call dup2(). And finally, in this block, I use execvp to execute the program with the options that I picked (-A 5 -e -A). Since I use exec function, similar to the first child, the second child will be replaced by another program therefore it never comes back. I use waitpid(second_child_id) after the condition block ends to make sure the first child, second child, and parent process are in the process hierarchy.

Finally, after the second child returns 0 in its program, the second waitpid continues where it left off. Now there is a single process left which is parent, I close the parent's pipe ends to stop grep to wait for reading in the console. Lastly, I print out the parent process id and finish the execution.