Assignment 2

Georgiy Krylov

September 21, 2024

ASSIGNMENT IS TO BE COMPLETED INDIVIDUALLY BY ALL STUDENTS!

There was an announcement from our IT department the last time we did this exercise, that while working on CS3413 assignment #2, it is possible to create too many processes using forks. As per the email, this "makes the computer completely unusable to anyone and causes problems for other students." Therefore, while working on the `processtree` command, use the input no bigger than 7 when testing.

Additionally, to avoid that, make sure you set the limits for the maximum number of processes you can create before running your program. Please follow these steps to limit the number of processes:

1. Identify the number of processes currently running on your machine by typing `ps -aux | wc -l`. Suppose my output is 300.
2. Add 150 to the number you have seen and remember it - this will be a new limit for the maximum number of running processes.
3. If you are connected to a lab machine remotely, start by typing "limit" command.
    - If you see an output other than "command not found" - type "limit maxproc 450", where instead of 450 should be the number you remembered from before.

- Otherwise, you are likely using Bash interpreter. Use "ulimit -S -u 450", where instead of should be the number you remembered from before.

This will limit the number process you can create. If you are seeing a message along the lines "no more processes" or "fork retry" - you have reached the specified limit. Please close the terminal program, and repeat the procedure of setting the limits again, every time you open a new terminal window. Also, please make sure you are terminating the processes properly. It will help you debug your program and keep you from locking up and getting kicked out of the computer. You will need to run this every time in each new command window.

1 Description

This assignment is for the class to have hands-on experience with creating process, understanding the principles of process trees and parent-child relationships between processes. Additionally, this assignment allows practicing the inter-process communication.

The assignment is Due at 11:59 p.m., Saturday the 28th of September 2024. (1 minute before Sunday).

2 Task

To complete this assignment, you have to write a C program that would serve as an "enhanced" terminal, which has extra functionality as well as able to invoke some simple programs.

The items your program should be able to do:

First of all, your program should terminate when it reads the **stop** command. The terminal otherwise expects a command in an infinite loop, prompting the user for input. The infinite loop might be configured to only run if a value of a certain variable is set to 1. This value would never change in a parent, but in children it might change. Alternatively, your program can call `exit()` explicitly in every child process.

Every line generated by you program should contain a process ID of the processes that issue it, using `getpid()` command. You are not allowed to use a variable to store the PID of the current process when printing a prompt.

To avoid "zombie" processes, please make sure to call the `wait()` command in every process. Note that the `wait()` command is waiting for termination of just one child process, you may need to call it in a loop if you want for multiple processes.

Upon reading the **onechild** command, your shell should spawn a child process and report parent and child process IDS. Hint: **fork()** and **getpid()** can be used for that.

Upon receiving the **addnumbers** command, your program should be able to sum the numbers from consecutive input, until a zero is typed in. This should

be done by setting up two pipes and forking a process. Later, the parent process should receive the inputs from the user and sending them to the child process. The child process should be reading from the pipe until a zero is encountered, storing the sum into a temporary variable. After a zero is read, the child process should send the total sum to the parent process. The parent process should print the result.

Upon reading the **exec** input into the command line, your program should read arguments until the "**;**" literal is encountered. After that, the command should be executed. To do this, your process should read the commands into an array of **char*** (i.e. char**) (up to five arguments). Further, your process should call the **fork** syscall. The parent should **wait(NULL)** for the child process, which in turn should execute the command using either of the **execvp(), exec(), execl(),** and **execlp()** commands. The child process does NOT have to print its PID when using one of the **exec()** calls. Note, make sure you initialize your buffer strings to NULL explicitly as well as allocate and de-allocate memory for them appropriately.

The final task is such that upon entering the **processtree** command, followed by an integer, all numbers from 0 to $2^{integer} - 1$ are printed. You should use a **for** loop for spawning the processes, can use a local variable, and can use the loop variable for computing the output. The order of output does not matter. (Please use the integers no bigger than 5 as inputs to this command, before you are certain that your processes terminate correctly). Creating other than $2^{integer} - 1$ processes, as well as underutilizing the parallel execution capabilities might affect the grade for this part of the assignment. You should use a **for** loop for spawning the processes, and a variable as the generation counter.

Hint: Chapter 3 actually provides a lot of code that can be reused. Another hint: To display all the buffered output, you can use **fflush(stdout)**.

3 Input/Output format

For this assignment, you don't have to make sure that input/output redirection works. Please refer to **sample_execution.txt** and **sample_execution_2.txt**.

4 Submission instructions

Please submit your C file to D2L Assignment box. Make sure your code compiles and runs. Make sure your code follows the specified input/output format. You must use C programming language to solve this assignment. Important to note in this course: if your program produces correct output, it doesn't necessarily mean you have properly managed the resources and penalties are possible. This assignment focuses on process management and communication, and therefore the TA will be asked to make sure the processes are properly created and terminated (emphasis on not having zombie processes).

NOTE: THE INPUT AND OUTPUT OF YOUR PROGRAM IS SPECIFIED SUCH THAT THE MARKER CAN AUTO TEST YOUR SUBMISSIONS. PLEASE FOLLOW THE SPECIFICATIONS! INPUT WILL BE READ VIA stdin (E.G. KEYBOARD). OUTPUT SHOULD BE PRINTED TO stdout (E.G. MONITOR).