New York University
Tandon School of Engineering
Department of Electrical & Computer Engineering

Introduction to Operating Systems (CS-GY6233)
Spring 2022

Assignment 4
(15 points)

1) (2 points) If you create a main() routine that calls fork() three times, i.e. if it includes the following code:
   pid_t pid1=33, pid2=22, pid3=11;
   pid1 = fork();
   if(pid1>0) pid2 = fork();
   if(pid2==0) pid3 = fork();

   Draw a process tree similar to that on slide **22** of lecture 4, clearly indicating the values of pid1,pid2 and pid3 for each process in the tree (i.e. whether 0,11,22,33, smaller than 0 or larger than 0).

   Note that the process tree should only have one node for each process and thus the number of nodes should be equal to the number of processes.

   The process tree should be a snapshot just after all forks completed but before any process exists.

   Each line/arrow in the process tree diagram shall represent a creation of a process, or alternatively a parent/child relationship.

2) (4 points) Write a program in which the main routine creates a child process that prints the process ID of the parent. You may use `getpid()` but you may NOT use `getppid()`. You may not use IPC for this part (neither shared memory nor message passing).

3) (9 points) Write a program whose main routine obtains one parameter *n* from the user, i.e. passed to your program when it was invoked from the shell. Your program shall then create a shared memory and a child process.

   The child process should obtain the value of *n* (you actually have multiple options for doing that) and create a sequence of length *n*, whose elements implement the equation $z = (0.5y)^2$ where y is the index of the element, for example, the fourth element (index 3) shall have a value of 2.25. If n=4, then the sequence shall be 0, 0.25, 1, 2.25.

   The child process shall create the elements, one at a time, and wait for a random interval of time (0 to 4.999 seconds) between generating elements of the sequence. As soon as an element is generated, the child places the element in the shared buffer by organizing it as described in slides 7-10 of lecture 5.

   The parent process shall NOT wait for the child to exit but instead shall print an element as soon as it arrives into the shared buffer (again, in a manner similar to slides 7-10 of lecture 5)

Hint; use fflush() to ensure printf's are printed immediately into the screen.

Submission file structure:

Please submit a **single .zip file** named **[Your Netid]_lab#.zip**. It shall have the following structure (replace # with the actual assignment number):

└── [Your Netid] hw# (Single folder includes all your submissions)

- ├── lab#_1.c (Source code for problem 1)
- ├── lab#_2a.c (Source code for problem 2a, and so on)
- ├── lab#_1.h (Source code header file, if any)
- ├── Makefile (makefile used to build your program, if any)
- ├── lab#.pdf (images + Report/answers to short-answer questions)

What to hand in (using Brightspace):

- Source files (.c or .h) with appropriate comments.
- Your Makefile if any.
- A .pdf file named **"lab#.pdf"** (# is replaced by the assignment number), containing:
  - Screen shot(**s**) of your terminal window showing the current directory, the command used to compile your program, the command used to run your program and the output of your program.

RULES:

- You shall **use kernel version 4.x.x or above**. You shall not use kernel version 3.x.x.
- You may consult with other students about GENERAL concepts or methods but copying code (or code fragments) or algorithms is NOT ALLOWED and is considered cheating (whether copied form other students, the internet or any other source).
- If you are having trouble, please ask your teaching assistant for help.
- You must submit your assignment prior to the deadline.