**New York University**
**Tandon School of Engineering**
**Department of Computer Science and Engineering**
**Introduction to Operating Systems**
**Fall 2024**
**Assignment 4**
(70 points)

**Problem 1 (10 points)** If you create a main() routine that calls fork() four times, i.e., if it includes the following code:

```
pid_t w=-1, x=-2, y=-3, z=-4;
w = fork();
if(w==0) x = fork();
if(x==0) y = fork();
if(y>0) z = fork();
```

Assuming all fork() calls succeed, draw a process tree similar to that of Fig. 3.8 (page 116) in your textbook, clearly indicating the values of w, x, y, and z for each process in the tree (i.e., whether 0, -1, -2, -3, -4, 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.

**Problem 2 (10 points)** Write a program that creates the following process tree: [Insert process tree image here - A tree with at least 5 processes]

**Problem 3 (10 points)** Write a program whose main routine obtains parameters n and m from the user (passed to your program when invoked from the shell, n¿2, m¿1) and creates m child processes. Each child process shall create and print a sequence of n multiples of its process ID. The parent waits for all children to exit before printing a confirmation message. Do not use IPC.

**Problem 4 (10 points)** Consider a scenario where a process needs to create a specific tree structure of processes. The structure is defined recursively: a process creates two child processes if its process ID is even, and one child process if its process ID is odd. The recursion depth is limited to 3 levels (the initial process is level 0). Draw the process tree that results from this algorithm starting with a process ID of 10. Clearly label each node with its process ID.

**Problem 5 (10 points)** Write a C program that uses fork() to create a child process. The parent process sends a signal (e.g., SIGUSR1) to the child process. The child process catches the signal and prints a message indicating that it received the signal and its process ID. The parent process then waits for the child to finish.

**Problem 6 (10 points)** Write a program that uses fork() to create two child processes. The parent process sends a different signal to each child (e.g., SIGUSR1 to one and SIGUSR2 to the other). Each child process catches its respective signal and prints a message indicating the signal received and its process ID. The parent process waits for both children to finish. Error handling for signal delivery failures should be included.

**Problem 7 (10 points)** Design a program that creates a process tree where each process calculates a portion of a large array sum. The parent process creates N child processes (N is a user input). The parent divides a large array of integers into N equal-sized sub-arrays. Each child process is responsible for calculating the sum of its assigned sub-array. After all children finish, the parent collects the partial sums from each child and computes the total sum of the array. Use pipes for inter-process communication. Include error handling for pipe creation and communication.

**What to hand in (using Brightspace):** Please submit the following files individually for all 7 problems:

1. Source file(s) with appropriate comments. The naming should be similar to "lab#_$.c" (# is replaced with the assignment number and $ with the question number within the assignment, e.g. `lab4_b.c`, for lab 4, question b OR `lab5_1a` for lab 5, question 1a).
2. A single pdf file (for images + report/answers to short-answer questions), 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.
3. Your Makefile, if any. This is applicable only to kernel modules.

**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.

**Operating Systems - Prof. Omar Mansour**