New York University
Tandon School of Engineering
Department of Computer Science and Engineering
Introduction to Operating Systems
Fall 2024
Assignment 4 (10 points)

Problem 1 (2 points)

If you create a main() routine that calls fork() three times, i.e., if it includes the following code:

```
   pid_t x=-11, y=-22, z=-33;
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 x, y, and z for each process in the tree (i.e., whether 0, -11, -22, -33, 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 (4 points)

Write a program that creates the process tree shown below:

Problem 3 (4 points)

Write a program whose main routine obtains a parameter n from the user (i.e., passed to your program when it was invoked from the shell, n¿2) and creates a child process. The child process shall then create and print a sequence of n numbers where each number is the sum of the two preceding numbers, starting with 1 and 1. The parent waits for the child to exit and then prints two additional numbers following the same pattern, i.e., the total number of numbers printed by the child and the parent is n+2. Do not use IPC in your solution to this problem (i.e., neither shared memory nor message passing).

Problem 4 (5 points)

Explain the concept of a Zombie process. Describe a scenario where a zombie process might be created. Detail how a zombie process can be avoided. Give a code example (in C) demonstrating the creation of a zombie process and a modified version that avoids it.

Problem 5 (5 points)

Consider a scenario with two processes, A and B, that need to share a single resource (e.g., a printer). Describe how you would use semaphores to prevent a race condition, ensuring that only one process can access the resource at any given time. Provide C code snippets illustrating the use of semaphores to implement mutual exclusion in this scenario. Discuss the potential issues with using only semaphores for synchronization and suggest alternative solutions or improvements.

What to hand in (using Brightspace):

Please submit the following files individually:

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 c 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:

- Screenshot(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 from 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