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() twice, i.e., if it includes the following code:

```
pid_t x=-11, y=-22;
x = fork();
if(x==0) y = 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 and y for each process in the tree (i.e., whether 0, -11, -22, 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 exits. Each line/arrow in the process tree diagram shall represent a creation of a process, or alternatively a parent/child relationship.

(Insert process tree diagram here)

Problem 2 (4 points)

Write a program that creates the process tree shown below:

[level distance=1.5cm, level 1/.style=sibling distance=3cm, level 2/.style=sibling distance=1.5cm] A child node B child node D child node E child node C child node F ;

Each node represents a process. The letter represents a unique process ID for identification purposes (in your code, you should use actual PIDs from the fork() system call).

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¿0) and creates n child processes. Each child process shall print its process ID and exit. The parent waits for all child processes to exit before printing "All children have finished." Do not use IPC in your solution to this problem.

Problem 4 (2 points)

Explain the difference between the `fork()` and `exec()` system calls. Give an example scenario where you would use each.

Problem 5 (2 points)

Describe a potential race condition that could occur in a multi-threaded program that increments a shared counter variable. Provide code illustrating the race condition and suggest a solution to prevent it.

Problem 6 (4 points)

Write a program that uses signals to handle the SIGINT signal (Ctrl+C). When the SIGINT signal is received, the program should gracefully terminate and print a message indicating that it is exiting.

Problem 7 (2 points)

Illustrate with code snippets how to use `wait()` and `waitpid()` system calls to wait for a child process to finish. Explain the differences between these two functions and when you might choose to use one over the other.

What to hand in (using Brightspace):

Please submit the following files individually for these 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:
    - 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.