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 - A simple tree with 3 processes: parent with x ¿ 0, child 1 with x == 0, y ¿ 0, child 2 with x == 0, y == 0)

Problem 2 (4 points)

Write a program that creates the process tree shown below:

(Insert process tree diagram here - A tree with a parent and three children, each child having two children. Total of 10 processes.)

Problem 3 (4 points)

Write a program whose main routine obtains two parameters, a and b, from the user (passed to your program when invoked from the shell, a and b are integers). The program then creates a child process. The child process calculates the greatest common divisor (GCD) of a and b using Euclid's algorithm and prints the result. The parent process waits for the child to finish and then prints the least common multiple (LCM) of a and b, calculated using the formula LCM(a, b) = (a * b) / GCD(a, b). Do not use IPC in your solution.

Problem 4 (2 points)

Explain the difference between a zombie process and an orphan process. Provide an example scenario for each.

Problem 5 (2 points)

Consider the following code snippet:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();
    if (pid == 0) {
        printf("Child process: PID = %d, PPID = %d\n", getpid(), getppid());
        exit(0);
    } else if (pid > 0) {
        wait(NULL);
        printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);
    } else {
        fprintf(stderr, "Fork failed\n");
        return 1;
    }
    return 0;
}
```

Draw the execution flow and show the output. Explain what the 'wait(NULL)' function call does.

Problem 6 (4 points)

Write a program that simulates a simple producer-consumer scenario using processes and pipes. The producer process generates random numbers between 1 and 100 and sends them through a pipe to the consumer process. The consumer process receives these numbers and calculates their sum. After generating and sending 10 numbers, the producer process closes its end of the pipe. The consumer process should continue to read and sum numbers until the pipe is closed, at which point it prints the total sum.

Problem 7 (4 points)

Write a program that forks a child process. The parent process sends a signal (e.g., SIGUSR1) to the child. The child process catches this signal and prints a message indicating that the signal was received. The parent process then waits for the child to finish before exiting. Explain the purpose of using signals in this scenario. Include necessary signal handling using the 'signal()' function.

What to hand in (using Brightspace):

Please submit the following files individually for each of the 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 for each problem.
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.