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

Your Name

September 12, 2025

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). The process tree should only have one node for each process. 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.

Figure 1: Process Tree

Problem 2 (4 points)

Write a program that creates the following process tree:

```
      A
     / \
    B   C
   / \
  D   E
```

Where A is the initial process. Each process should print its own process ID.

```
% Insert your code for Problem 2 here.  Remember to comment your code.
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
  pid_t pid;
  printf("Process A: %d\n", getpid());
```

```
  pid = fork();
  if (pid == 0) { // Child B
    printf("Process B: %d\n", getpid());
    pid = fork();
    if (pid == 0) { // Grandchild D
      printf("Process D: %d\n", getpid());
    } else { // Child B continues
      pid = fork();
      if (pid == 0){ // Grandchild E
        printf("Process E: %d\n", getpid());
      }
    }
  } else { //Parent A continues
    pid = fork();
    if (pid == 0) { // Child C
      printf("Process C: %d\n", getpid());
    }
  }
  return 0;
}
```

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). The program should then create a child process. The child process computes the greatest common divisor (GCD) of a and b using Euclid's algorithm and prints the result. The parent waits for the child to exit and then prints the least common multiple (LCM) of a and b (calculated using the GCD). Do not use IPC.

```
% Insert your code for Problem 3 here. Remember to comment your code.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

unsigned long long gcd(unsigned long long a, unsigned long long b) {
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <a> <b>\n", argv[0]);
        return 1;
    }

    unsigned long long a = atoll(argv[1]);
    unsigned long long b = atoll(argv[2]);

    pid_t pid = fork();

    if (pid == 0) { // Child process
        unsigned long long result = gcd(a, b);
        printf("Child: GCD(%llu, %llu) = %llu\n", a, b, result);
        exit(0);
    } else if (pid > 0) { // Parent process
```

```
        wait(NULL);
        unsigned long long result = (a * b) / gcd(a, b);
        printf("Parent: LCM(%llu, %llu) = %llu\n", a, b, result);
    } else {
        perror("fork");
        return 1;
    }

    return 0;
}
```

Problem 4 (2 points)

Explain the difference between the ‘exec‘ family of functions and the ‘fork()‘ system call. Give examples of when you would use each.

Problem 5 (8 points)

Write a program that simulates a simple producer-consumer problem using two processes and a shared memory segment. The producer process generates random numbers and writes them to the shared memory. The consumer process reads the numbers from the shared memory and calculates their sum. Use semaphores to synchronize access to the shared memory. The program should take an integer N as a command line argument, indicating the number of random numbers to produce.

```
% Insert your code for Problem 5 here. Remember to comment your code.
// (This problem requires significant code and understanding of shared memory and semaphores.  A com
```

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 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.