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() 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 exits. 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: *(Process tree diagram would be inserted here - a simple tree with at least 4 processes)*

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 calculate the sum of the first n even numbers and print the result. The parent waits for the child to exit and then prints double the sum calculated by the child. Do not use IPC in your solution to this problem (i.e., neither shared memory nor message passing).

Problem 4 (2 points) Describe the race condition that might occur if two processes try to increment a shared counter simultaneously without any synchronization mechanisms. Suggest a solution to prevent this race condition.

Problem 5 (3 points) Explain the difference between a process and a thread. Give examples of when you might prefer to use threads over processes, and vice versa.

Problem 6 (4 points) Write a C program that uses threads to calculate the factorial of numbers from 1 to 5 concurrently. Each thread should calculate the factorial of one number and print the result. The main thread should wait for all other threads to complete before exiting.

Problem 7 (2 points) What is a deadlock? Give an example scenario where a deadlock might occur in an operating system.

Problem 8 (3 points) Explain the concept of a semaphore. How can semaphores be used to prevent race conditions?

Problem 9 (4 points) Write a C program that uses two threads and a mutex to protect a shared counter. One thread increments the counter 1000 times, and the other thread decrements the counter 1000 times. Ensure that the final value of the counter is 0.

Problem 10 (2 points) What is a critical section? Why is it important to protect critical sections?

Problem 11 (3 points) Explain the difference between a mutex and a condition variable. When would you use each?

Problem 12 (4 points) Write a C program that uses a condition variable to synchronize two threads. One thread produces data, and the other thread consumes the data. The consumer thread should wait for data to be available before consuming it.

Problem 13 (2 points) Describe the producer-consumer problem.

Problem 14 (3 points) Explain the concept of a reader-writer lock. When would you use a reader-writer lock instead of a mutex?

Problem 15 (4 points) Write a C program that uses a reader-writer lock to protect a shared resource. Multiple reader threads can access the resource concurrently, but only one writer thread can access it at a time.

Problem 16 (2 points) What is a memory map?

Problem 17 (3 points) Explain the concept of virtual memory. What are its advantages?

Problem 18 (4 points) Describe the page replacement algorithms: FIFO, LRU, and Optimal. Discuss their relative performance.

Problem 19 (2 points) What is a page fault?

Problem 20 (3 points) Explain the concept of thrashing. How can it be avoided?

Problem 21 (4 points) Write a short C program that simulates a simple page replacement algorithm (e.g., FIFO) with a fixed number of frames.

Problem 22 (2 points) What is the role of the scheduler in an operating system?

Problem 23 (3 points) Describe different scheduling algorithms (e.g., FCFS, SJF, Round Robin).

Problem 24 (4 points) Compare and contrast preemptive and non-preemptive scheduling.

Problem 25 (2 points) What is context switching?

Problem 26 (3 points) Explain the concept of a process control block (PCB).

Problem 27 (4 points) Describe the different states a process can be in (e.g., running, ready, blocked).

Problem 28 (2 points) What is an interrupt?

29 (3 points) Explain how interrupts are handled by the operating system.

Problem 30 (4 points) Describe the differences between hardware and software interrupts.

Problem 31 (2 points) What is I/O-bound process?

Problem 32 (3 points) What is CPU-bound process?

Problem 33 (4 points) How does the operating system handle I/O operations?

Problem 34 (2 points) What is a file system?

Problem 35 (3 points) Describe different types of file systems (e.g., FAT, NTFS, ext4).

Problem 36 (4 points) Explain the concept of file allocation methods (e.g., contiguous, linked, indexed).

Problem 37 (2 points) What is a directory?

Problem 38 (3 points) Explain the concept of a file descriptor.

Problem 39 (4 points) Write a simple C program that creates a file, writes data to it, and then reads the data back.

Problem 40 (2 points) What is a socket?

Problem 41 (3 points) Explain the client-server model.

Problem 42 (4 points) Write a simple client-server program using sockets (you can use TCP or UDP). The client sends a message to the server, and the server echoes the message back to the client.

Problem 43 (2 points) What is a deadlock?

Problem 44 (3 points) Explain the four conditions necessary for a deadlock to occur.

Problem 45 (4 points) Describe methods for preventing or avoiding deadlocks.

Problem 46 (2 points) What is a race condition?

Problem 47 (3 points) Explain how mutual exclusion can be used to prevent race conditions.

Problem 48 (4 points) Write a C program that demonstrates a race condition and then shows how to fix it using a mutex.

Problem 49 (2 points) What are some common operating system security threats?

What to hand in (using Brightspace): Please submit the following files individually for all 49 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 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 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