Operating Systems Assignment

Introduction

This assignment is designed to help you understand the concepts of multi-threading, semaphores, synchronization, and race conditions in operating systems. You will be implementing several programs in C.

Instructions

- All programs should be written in C.
- Make sure to include necessary comments in your code to explain your logic.

Problems

Problem 1: Multi-threaded Factorial Calculation

Write a multi-threaded program where each thread calculates the factorial of a unique number from 1 to 10. Use a semaphore to ensure that only one thread can print the result at a time, and they must print in ascending order of their numbers. The output should be in the format “Thread [number]: [factorial]”.

Problem 2: Producer-Consumer Problem

Implement the producer-consumer problem using semaphores where the buffer size is 1 (binary semaphore). The producer should generate a sequence of numbers from 1 to 100, and the consumer should print them. Ensure that the consumer only consumes when there is an item in the buffer, and the producer only produces when the buffer is empty.

Problem 3: Multi-process String Generation

There are three processes that generate strings ‘a’, ‘b’, and ‘c’ in an arbitrary order. In the absence of any synchronization mechanism, there will be no order in the generation of ‘a’, ‘b’, and ‘c’. Write a multi-process program in C where each process generates its respective character. Use semaphores to synchronize the processes in such a way that the printed string will follow the pattern ‘aaacbaaacbaaacb’. Print this pattern a total of 10 times.

Problem 4: Race Condition with Global Counter

Consider a multi-threaded program where two threads, namely Thread 1 and Thread 2, are incrementing a global counter initialized to 0. Each thread should increment the counter 100 times. Without proper synchronization, this can lead to a race condition.

Problem 5: Race Condition with Shared Bank Account

Consider a scenario where three threads, namely Thread A, Thread B, and Thread C, are trying to update the balance of a shared bank account initialized to $1000. Each thread should subtract $200 from the account balance 5 times. Without proper synchronization, this can lead to a race condition.

a. The data involved in the race condition is the balance of the shared bank account.

b. The race condition occurs at the location in the code where Thread A, Thread B, and Thread C update (subtract from) the balance of the shared bank account.

c. Write a C program that creates three threads (Thread A, Thread B, and Thread C) which update (subtract from) the balance of a shared bank account initialized to $1000. Each thread should subtract $200 from the account balance 5 times, resulting in a final balance of $1000 - $200 * 3 * 5 = -$2000. Use a semaphore to fix the race condition and ensure that the final balance of the account is always -$2000.

Problem 6: Dining Philosophers Problem

Implement a solution for the dining philosophers problem using semaphores.