Operating Systems(CSL3030)

PracticeSet- Report

Soumen Kumar
Roll No-B22ES006

1 Introduction

As I worked on these practice problems, I aimed to explore and understand the complexities of multi-threading, inter-process communication (IPC), and synchronization. These topics are fascinating because they challenge us to think about how processes and threads interact, how data flows between them, and how to manage shared resources effectively. In this report, I will walk you through my thought process and approach for each of the seven problems.

2 Multi-Threaded Array Sum

The first challenge was to compute the sum of a large array using multiple threads. I broke the array into smaller segments, with each thread handling one segment. This approach ensures that the workload is distributed evenly. After the threads completed their tasks, I gathered their partial sums in the main thread to compute the final total. This method highlighted the power of threading for computational efficiency.

3 Process-Based 3D Array Operations

This problem was particularly intriguing as it combined processes and threads to manage a 3D array. I had to create three processes, each responsible for one dimension of the array. Within each process, I used threads to calculate dot products of 3×3 submatrices. After each process completed its work, it sent its result matrix to the parent process. The parent then combined these matrices into a final result. This task gave me a deeper appreciation for process communication and coordination.

4 Unidirectional IPC with Pipe

For this problem, I set up a simple communication pipeline between a parent and a child process. The parent read data from an input file and sent it through a pipe to the child, which then wrote the data to an output file. It was a straightforward exercise, but it helped reinforce the basics of IPC and file handling in C.

5 Bidirectional IPC with Two Pipes

The fourth task expanded on the previous one by adding bidirectional communication and multiple processes. Here, the parent read data from a file and sent it to one child process for transformation (converting text to uppercase). The transformed data was then sent to a second child for further processing (reversing the text) before being sent back to the parent. Finally, the parent wrote the fully transformed data to an output file. This problem was a great opportunity to practice managing multiple pipes and ensuring proper synchronization.

6 Race Condition Demonstration

The fifth challenge was to demonstrate a race condition using two threads that increment and decrement a shared counter. Without synchronization, the counter's value was unpredictable due to concurrent access. This exercise illustrated the potential pitfalls of unsynchronized threading and the importance of proper locking mechanisms.

7 Thread Synchronization with Mutex

Building on the previous problem, I created a program with 10 threads, each incrementing a shared counter 1000 times. Initially, I ran the program without any locks, which led to incorrect results due to race conditions. Then, I introduced a mutex to synchronize access to the counter, ensuring that the final value was correct. This task was a hands-on way to see how synchronization tools like mutexes can prevent issues in multi-threaded programs.

8 Deadlock Detection with Resource Allocation Graph

Finally, I tackled the problem of deadlock detection using a Resource Allocation Graph (RAG). I represented processes and resources as nodes in a graph and implemented a cycle detection algorithm to identify potential deadlocks. If a cycle existed, it indicated a deadlock, and I could pinpoint the processes involved. This problem emphasized the importance of designing systems that avoid deadlocks through careful resource allocation.

9 Conclusion

These exercises were both challenging and rewarding. They gave me a chance to dive deep into concepts like threading, process communication, synchronization, and deadlock detection. Each problem helped me build a more intuitive understanding of how concurrent systems work and how to handle the complexities that come with them. I'm excited to apply these skills to real-world scenarios and continue exploring this fascinating area of programming.