CS588: Computer Systems Lab

Supervised by: Dr. Arijit Sur
Professor, Dept. of CSE, IIT Guwahati

Assignment 4 (Q3)

Multi-threaded Event-Reservation System Using Pthreads

1. Introduction

1.1 Background

In today's world, efficient management of event reservations is crucial, especially in large venues such as auditoriums. As the number of events increases, so does the complexity of managing bookings, cancellations, and seat inquiries. A multi-threaded system can significantly improve performance and reliability by handling multiple user requests concurrently.

1.2 Objective

The objective of this project was to design and implement a multi-threaded event-reservation system that can efficiently manage concurrent seat reservations, inquiries, and cancellations. The system was built using the pthread API, focusing on synchronization, concurrency control, and database consistency.

2. System Design

2.1 Problem Statement

The system was designed to handle three types of queries for an auditorium:

- **Inquiry**: Checking the number of available seats for an event.
- **Booking**: Reserving a specified number of seats for an event.
- **Cancellation**: Cancelling a previously booked ticket.

A key challenge was to manage a maximum number of active queries (denoted as MAX) concurrently without overloading the system. Additionally, ensuring mutual exclusion for read/write operations on the same event was essential to maintain database integrity.

2.2 Architecture

The system consists of a master thread and multiple worker threads:

- **Master Thread**: Initializes events, creates worker threads, and manages the overall flow of the system.
- **Worker Threads**: Perform randomly generated queries on the system, simulating real-world scenarios.

2.3 Synchronization Mechanisms

To manage concurrency and ensure mutual exclusion, the following synchronization mechanisms were employed:

- **Semaphores**: Used to limit the number of active queries, ensuring that no more than MAX queries are processed simultaneously.
- **Mutexes**: Utilized to protect shared resources, particularly the event reservation data and the shared table that tracks active queries.

3. Implementation

3.1 Core Functions

- **Check Availability**: Functions to check if a query can be executed based on current active queries.
- **Booking**: Function to book seats, ensuring that no two threads can book the same seats concurrently.
- **Cancellation**: Function to cancel bookings, updating the seat availability accordingly.
- **Query Execution**: Functions that handle the execution of queries, ensuring synchronization and mutual exclusion.

3.2 Shared Table

A shared table was implemented to track active queries. Each entry in this table contained the event number, query type, and thread ID. This allowed the system to determine if a new query could be processed without conflicting with ongoing operations.

3.3 Randomization and Timing

Queries were randomly generated to simulate real-world scenarios. Each thread performed queries with random delays between operations, and the system ran for a specified duration (T).

4. Results and Discussion

4.1 Performance

The system effectively handled multiple concurrent queries while maintaining database consistency. The use of semaphores and mutexes ensured that the system never exceeded the MAX limit of active queries, and mutual exclusion was maintained for read/write operations on events.

4.2 Challenges

- **Deadlock Prevention**: Ensuring that the system was free from deadlocks required careful design of the locking mechanisms.
- **Scalability**: As the number of events and threads increased, managing the shared table and ensuring efficient access became more complex.

4.3 Testing and Validation

The system was tested with various configurations (different numbers of events, threads, and MAX values). The results demonstrated that the system could efficiently manage concurrent operations without any loss of data integrity.

5. Conclusion

The multi-threaded event-reservation system developed in this project successfully demonstrated the ability to manage complex, concurrent reservations in a real-time environment. The implementation of synchronization mechanisms like semaphores and mutexes ensured the system's reliability and performance. This project provides a solid foundation for further development and optimization in more complex and large-scale reservation systems.