Final Report
Overbooking Problem and Critical Section Demonstration

Introduction

In modern booking systems such as airline ticketing, hotel reservations, and logistics platforms, multiple users often try to book the same resource at the same time. If the system is not designed to handle concurrent access correctly, it can lead to serious issues such as overbooking. Overbooking occurs when more reservations are confirmed than the actual available capacity.

This project demonstrates how overbooking can occur due to race conditions and improper handling of critical sections in concurrent systems. The main objective of this project is to simulate the problem and explain why synchronization mechanisms are necessary in real-world applications.

Problem Description

A race condition happens when two or more processes or threads access shared data simultaneously and the final result depends on the order in which the processes are executed. In booking systems, this situation is very common because multiple users can request the same seat, room, or slot at nearly the same moment.

The booking process usually follows these steps:

1. Check available capacity
2. Confirm the booking
3. Update the remaining availability

If these steps are executed without synchronization, multiple requests may read the same availability value and all of them may be approved. As a result, the system confirms more bookings than it should, causing overbooking.

Critical Section Explanation

A critical section is a part of a program where shared resources are accessed or modified. Only one process or thread should be allowed to enter this section at a time.

In this project, the critical section includes:

- Reading the number of available booking slots
- Updating the availability after a booking request

Because there is no synchronization mechanism protecting this critical section, multiple threads can enter it simultaneously. This leads to inconsistent data and overbooking.

Project Structure

The project consists of a backend, a simulation script, and a simple frontend interface. The backend handles booking logic and availability management. The simulation script generates concurrent booking requests to demonstrate race conditions. The frontend provides a basic interface to observe booking behavior.

The main files used in this project are:

- app.py: contains the backend logic and booking operations
- simulate.py: simulates multiple booking requests occurring at the same time
- index.html: a simple frontend interface
- requirements.txt: list of required Python libraries

System Workflow

First, the system initializes a limited number of available booking slots. Then, multiple booking requests are sent to the backend almost simultaneously. Each request checks the availability and attempts to confirm a booking.

Since there is no synchronization, several requests may succeed even if there is only one slot available. This clearly demonstrates how overbooking occurs when critical sections are not properly managed.

Simulation and Testing

The simulate.py script is used to test the system under concurrent conditions. By running the script, multiple booking requests are triggered at nearly the same time.

The observed results show that:

- Multiple bookings are confirmed for a single available slot
- The shared availability value becomes incorrect
- The system enters an inconsistent state

These results confirm the presence of race conditions and the absence of critical section protection.

Results and Analysis

The results of the simulation prove that improper handling of critical sections leads directly to overbooking. Even in a small system with simple logic, concurrent access without synchronization causes errors.

This behavior reflects real-world failures in reservation and booking systems, highlighting the importance of using synchronization mechanisms such as mutex locks or semaphores.

Limitations

This project uses in-memory data instead of a real database, which limits its scalability. Synchronization mechanisms are not implemented, as the purpose of the project is to demonstrate the problem rather than the solution. Additionally, the frontend interface is minimal and serves only for basic interaction.

Conclusion

This project successfully demonstrates how overbooking occurs due to race conditions and improper handling of critical sections. It shows that synchronization is essential when multiple processes or users access shared resources.

Understanding and preventing such issues is crucial in designing reliable booking systems. This project provides a clear and practical example of why critical section protection is necessary in concurrent systems.