Overbooking in Airline Systems: A Critical Section Problem

Name: Maksudov Temurbek
ID: 210721

1. Introduction: The Centrum Air Overbooking Case

On 26 September 2025, passengers holding valid tickets for a Centrum Air flight from Istanbul to Namangan were denied boarding due to overbooking. Despite having confirmed reservations, the airline had sold more tickets than available seats on the aircraft. As a result, the passengers were forced to stay at the airport and were later rerouted to another destination. Compensation was paid after the passengers filed a complaint.

This incident demonstrates a common real-world problem in booking systems where multiple users attempt to access and modify shared data simultaneously. From a computer science perspective, this situation can be analyzed using operating system synchronization concepts, particularly the critical section problem.

2. Critical Section Concept

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

In an airline booking system:

- The shared resource is the number of available seats.
- The critical section includes:
    1. Reading the number of available seats
    2. Checking seat availability
    3. Updating the seat count after booking

If multiple booking requests enter this critical section simultaneously without protection, incorrect results may occur.

3. Read/Write Conflict (Race Condition)

Overbooking is caused by a **read/write conflict**, also known as a **race condition**.

Scenario:

- Two users request a ticket at nearly the same time.
- Both processes read the same value: `available_seats = 1`
- Both processes conclude that a seat is available.
- Both write back an updated value after booking.

This results in two tickets being sold for one seat.

The conflict occurs because:

- Reads and writes are not synchronized
- The system does not guarantee atomic execution of the booking operation

4. Producer–Consumer Analogy

The airline booking system can also be explained using the **Producer–Consumer problem**.

Producer–Consumer Model	**Airline Booking System**
Producer	Passenger booking request
Consumer	Seat allocation system
Buffer	Available seats

Conflict	Multiple producers accessing the buffer

When producers (booking requests) generate requests faster than the consumer (seat allocator) can safely process them, data inconsistency occurs. Without synchronization, the same seat can be consumed multiple times.

5. Chosen Solution and Why It Works

Solution: Mutex (Lock-Based Synchronization)

To prevent overbooking, a **mutex lock** is used to protect the critical section.

- Only one booking request can access seat data at a time.
- Other requests must wait until the lock is released.
- The read–check–write sequence becomes **atomic**.

Why This Works

- Eliminates race conditions
- Ensures data consistency
- Guarantees that seat count cannot become negative
- Scales well in multithreaded systems

In real airline systems, this concept is implemented using:

- Database transactions
- Row-level locking
- Distributed locks

6. Conclusion

The Centrum Air overbooking incident is a practical example of a classical **critical section problem**. Overbooking occurs when multiple booking processes access shared seat data without synchronization, leading to a read/write conflict.

By applying operating system concepts such as **mutex locks** and **critical section protection**, booking systems can ensure correctness, prevent overbooking, and protect consumer rights.