

Operating System Project Report

National University of Computer and Emerging Sciences



Flight Management System

Batch: 2023

Section: CY-4A

Group Members

Talal Ali (23K-2003)

Muhammad Hammad (23K-2005)

Daniyal Ahmed (23K-2011)

Umer Taiyab (23K-2016)

Lecturer

Ms. Atiya Jokhio

Department of Cyber Security

National University of Computer and Emerging Sciences –

FAST Karachi Campus

Acknowledgement

We would like to extend our sincere thanks to our supervisor, Ms. Atiya Jokhio, for her continuous support, guidance, and insightful feedback throughout the project.

Abstract

This project focuses on solving the reader-writer problem in a flight management system using mutexes and semaphores for synchronization. The system provides functionalities for user login, flight viewing, booking, and cancellation, ensuring safe concurrent access to shared resources. By managing simultaneous user interactions, the system maintains data integrity and offers a reliable solution for handling high-concurrency scenarios in flight management.

Problem Statement

Managing a flight reservation system in a multi-user environment presents challenges in ensuring data consistency while allowing concurrent access. Without proper synchronization, simultaneous actions like flight bookings or cancellations could lead to data corruption, impacting the reliability of the system. The need for an efficient mechanism to handle concurrent operations without compromising data integrity is critical in this domain.

Project Overview

The Flight Management System project aims to design and implement a solution to handle the reader-writer problem using synchronization tools like mutexes and semaphores. It incorporates features like user login/register, flight browsing, booking, and cancellation, ensuring safe and concurrent access to shared resources.

The project seeks to offer a robust and scalable system for flight management that addresses concurrency issues and enhances data integrity.

Dry Run

Scenario #1: Two Threads Trying to Book the Same Seat

1. **Thread 1** and **Thread 2** start simultaneously and both **read flight data** from the file.
2. Both **Thread 1** and **Thread 2** find that the seat is available and attempt to **book the seat** (try to write).
3. **Thread 1** acquires the **file write lock** first (via semaphore/mutex). **Thread 2** waits for **Thread 1** to release the lock.
4. **Thread 1** checks if the seat is still available. It is available, so **Thread 1 books the seat and marks the seat as booked** (`seat_booked_justnow = 1`).
5. **Thread 1** releases the **file write lock**.
6. **Thread 2** now acquires the **file write lock**.
7. **Thread 2** checks the seat status and sees it has already been booked (`seat_booked_justnow = 1`), so **Thread 2 does not book the seat**.
8. **Thread 1** prints **"Seat has been booked successfully."**. **Thread 2** prints **"Seat is already booked. Can't book it twice."** (seat is not available anymore).

```
(kali㉿kali)-[~/Desktop/OS_Project/final/flights_Management_System]
$ make simul
gcc simulation.o view_flights.o cancellation.o book_flight.o -lpthread -o sc1
./sc1
```

```

-----
Simulating Scenario: 2 threads try to book the same seat
-----

Thread 1: trying to book seat 19 on flight 5...
Thread 2: trying to book seat 19 on flight 5...
Seat is already Booked! cant book it twice!
Seat 19 booked successfully for flight 5.

-----
Simulation finished.
-----
```

Scenario #2: Two Threads Trying to Cancel the Same Seat

1. **Thread 1** and **Thread 2** start simultaneously and both **read the flight data** from the file.
2. Both **Thread 1** and **Thread 2** find that the seat is **booked** and attempt to **cancel the booking** (try to write).
3. **Thread 1** acquires the **file write lock** first (via semaphore/mutex). **Thread 2** waits for **Thread 1** to release the lock.
4. **Thread 1** verifies if the seat is booked. It is booked, so **Thread 1** **cancels the booking** and marks the seat as available.
5. **Thread 1** releases the **file write lock**.
6. **Thread 2** now acquires the **file write lock**.
7. **Thread 2** checks the seat status and sees that the seat has already been canceled (no longer booked), so **Thread 2 does not perform cancellation** again.
8. **Thread 1** prints "Seat has been cancelled successfully."
Thread 2 prints "Cannot cancel because you have not booked that seat."

```

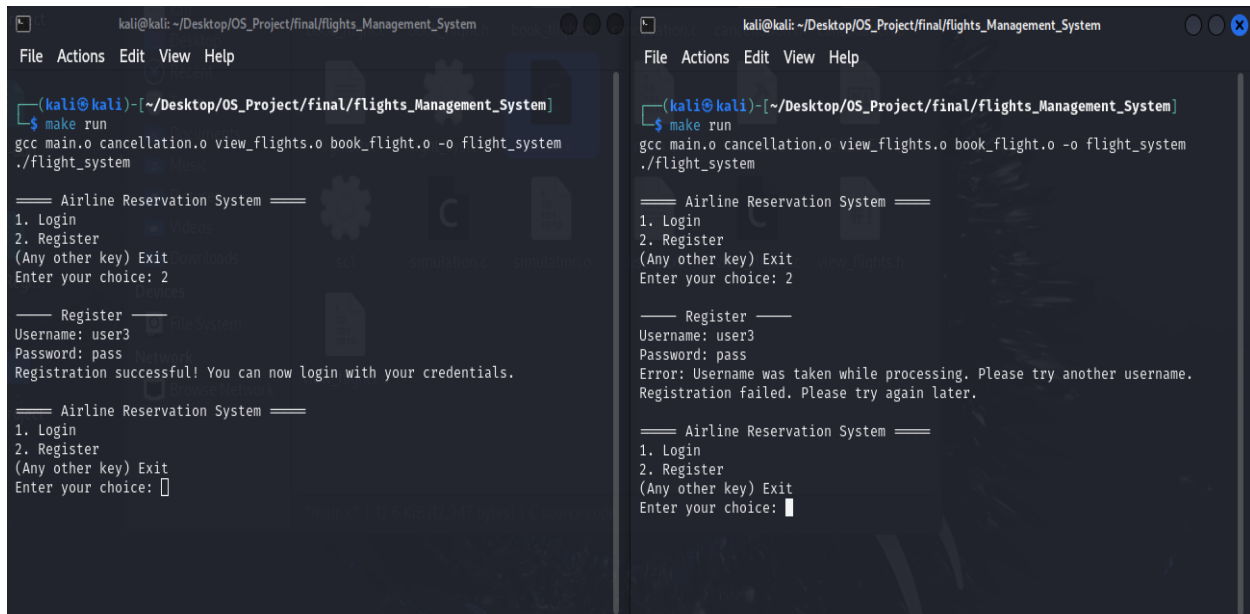
CN: A1
-----
Simulating Scenario: 2 threads try to cancel the same seat
-----
Thread 4: trying to cancel seat 19 on flight 5 ...
Thread 3: trying to cancel seat 19 on flight 5 ...
You havent booked flight number 5!
Booking cancelled successfully!
-----
Simulation finished.
-----

```

Scenario #3: Two Processes Trying to Register/Login Simultaneously

1. **Process 1** and **Process 2** both attempt to **register/login** at the same time by running separate instances of the program.
2. Both processes try to access the **user database (users.txt) file** concurrently.
3. **Process 1** acquires the **user_file_sem semaphore** first. **Process 2** waits for the semaphore to be released.
4. **Process 1** reads the user data:
 - If **registering**, it checks if the username already exists and writes a new user if available.
 - If **logging in**, it verifies the entered credentials.
5. **Process 1** releases the **user_file_sem semaphore** after completing its operation.
6. **Process 2** acquires the **user_file_sem semaphore** after Process 1.
7. **Process 2** reads the updated user data:
 - If **registering**, and the username is already taken (registered by Process 1), it fails the registration.
 - If **logging in**, it verifies against the updated credentials.

- Each process prints appropriate success or failure messages based on the outcome.



```
kali@kali: ~/Desktop/OS_Project/final/flights_Management_System
File Actions Edit View Help

(kali@kali)-[~/Desktop/OS_Project/final/flights_Management_System]
$ make run
gcc main.o cancellation.o view_flights.o book_flight.o -o flight_system
./flight_system

===== Airline Reservation System =====
1. Login
2. Register
(Any other key) Exit
Enter your choice: 2

===== Register =====
Username: user3
Password: pass
Registration successful! You can now login with your credentials.

===== Airline Reservation System =====
1. Login
2. Register
(Any other key) Exit
Enter your choice:

kali@kali: ~/Desktop/OS_Project/final/flights_Management_System
File Actions Edit View Help

(kali@kali)-[~/Desktop/OS_Project/final/flights_Management_System]
$ make run
gcc main.o cancellation.o view_flights.o book_flight.o -o flight_system
./flight_system

===== Airline Reservation System =====
1. Login
2. Register
(Any other key) Exit
Enter your choice: 2

===== Register =====
Username: user3
Password: pass
Error: Username was taken while processing. Please try another username.
Registration failed. Please try again later.

===== Airline Reservation System =====
1. Login
2. Register
(Any other key) Exit
Enter your choice: 
```

Use Cases

Case #1: Register → Login → Book Flight → Logout

- **User registers** with a new username and password.
- **System checks** if the username already exists.
- Upon successful registration, **user logs in**.
- After login, **user views available flights** and **books a seat** on the selected flight.
- System **updates** the booking records.
- **User logs out** after booking.
- **User then exits** the program.

```
(kali@kali)-[~/Desktop/OS_Project/final/flights_Management_System]
$ make run
gcc -c cancellation.c
gcc main.o cancellation.o view_flights.o book_flight.o -o flight_system
./flight_system
```

```
===== Airline Reservation System =====
```

```
1. Login
2. Register
(Any other key) Exit
Enter your choice: 2
```

```
----- Register -----
```

```
Username: User5
```

```
Password: pass
```

```
Registration successful! You can now login with your credentials.
```

```
===== Airline Reservation System =====
```

```
1. Login
2. Register
(Any other key) Exit
Enter your choice: 1
```

```
----- Login -----
```

```
Username: User5
```

```
Password: pass
```

```
Welcome, User5! You are now logged in.
```

```
===== User Menu =====
1. View Flights
2. Cancel Flights
(Any other key) Logout
Enter your choice: 1
```

View Flights section

Following Are the flights Available:-

- 1) Karachi to Delhi
- 2) New York to London
- 3) Tokyo to Sydney
- 4) Paris to Cairo
- 5) Toronto to Dubai

(Note: enter any other character to abort)
Select flight: 1

Flight Information:

Flight no: 1
Route: Karachi to Delhi
Date: Tuesday, 22 May 2025
Time: 7:30 AM (GMT+5)

Seats Available:-

{ }	{ }	{ }	{ }	{ }	{ }
{ }	{08}	{09}	{10}	{11}	{12}
{13}	{14}	{ }	{16}	{17}	{18}
{19}	{20}	{21}	{22}	{23}	{24}
{25}	{26}	{27}	{28}	{29}	{30}
{31}	{32}	{33}	{34}	{35}	{36}
{37}	{38}	{39}	{40}	{41}	{42}
{43}	{44}	{45}	{46}	{47}	{48}
{49}	{50}	{51}	{52}	{53}	{54}
{55}	{56}	{57}	{58}	{59}	{60}

(Note: enter 0 or any invalid seat number to cancel)
Which seat to book: 10
Seat 10 booked successfully for flight 1.

(Note: enter 0 or any invalid seat number to cancel)
Which seat to book: 0

Aborted Seat Booking: Returning to Flights section

Following Are the flights Available:-

- 1) Karachi to Delhi
- 2) New York to London
- 3) Tokyo to Sydney
- 4) Paris to Cairo
- 5) Toronto to Dubai

(Note: enter any other character to abort)
Select flight: y

Aborted Flight Selection: Returning to User Menu

==== User Menu ====

1. View Flights
2. Cancel Flights

(Any other key) Logout
Enter your choice: 3

Logout Successful.

==== Airline Reservation System ====

1. Login
2. Register

(Any other key) Exit
Enter your choice: 3

Airline System Exited. Goodbye!

Case #2: Login → Cancel Booking → Logout

- **User logs in** with existing credentials.
- After login, **user cancels** a booking for a flight they had reserved.
- System **updates** the seat availability.
- **User logs out** after cancellation and **exits** the program.

```
===== Airline Reservation System =====
1. Login
2. Register
(Any other key) Exit
Enter your choice: 1

===== Login =====
Username: User5
Password: pass
CN_Project

Welcome, User5! You are now logged in.

===== User Menu =====
1. View Flights
2. Cancel Flights
(Any other key) Logout
Enter your choice: 2

Flights Cancellation section

(Enter '0' to navigate back to the User Menu)
Enter flight number: 1
Enter seat number: 10
Booking cancelled successfully!

Returning to User Menu

===== User Menu =====
1. View Flights
2. Cancel Flights
(Any other key) Logout
Enter your choice: 3

Logout Successful.

===== Airline Reservation System =====
1. Login
2. Register
(Any other key) Exit
Enter your choice: 3

Airline System Exited. Goodbye!
```

Theoretical Concepts Used

Multithreading:

Our project uses multithreading to allow multiple operations (booking, cancellation) to occur simultaneously without blocking each other, improving system responsiveness and simulating real-world concurrent access.

Reader-Writer Problem:

The system addresses the classic reader-writer problem by allowing multiple read operations (viewing flights) but ensuring exclusive access during write operations (booking/canceling flights) to maintain data integrity.

Mutex (Mutual Exclusion):

Mutex locks are used to protect critical sections where shared data is accessed or modified. Only one thread can hold the mutex at a time, ensuring that no two threads write simultaneously.

Semaphore:

Semaphores are used to manage concurrent access to shared resources, particularly when updating user flight records, providing a lightweight synchronization mechanism across threads.

Conclusion

This project successfully implements a multithreaded flight management system by addressing synchronization challenges using mutexes and semaphores. It demonstrates the practical application of the reader-writer problem, ensuring that flight data remains consistent even under concurrent booking and cancellation attempts. By simulating real world user actions like registration, login, booking, and cancellation, the project highlights the importance of safe concurrent programming practices in modern systems.