Operating Systems

(CS-2006)

Spring-2025

Final Project Report

"Airplane Allocator using Parallelism Techniques"

Group Members:

Hafsa Rashid (23K-0064)

Syeda Sara Ali (23K-0070)

Adina Faraz (23K-0008)

Muneeb ur Rehman (23K-0038)

Instructor:

Ms. Saeeda Kanwal

1. Introduction

The Airplane Allocator project is a simulation system created to handle the complex process of managing airplane takeoffs and landings in an efficient and organized manner. This system takes inspiration from real-world air traffic control operations and uses modern computing techniques such as **parallelism**, **thread synchronization**, and **priority-based scheduling**. Written in the **C programming language**, the project demonstrates the use of **multithreading**, **mutex locks**, **condition variables**, and **asynchronous operations** to achieve concurrency.

The main idea behind this project is to allow multiple flight operations—such as takeoffs and landings—to occur concurrently while maintaining safe and synchronized use of limited runways. To achieve this, the system uses both **Data Parallelism** and **Task Parallelism** techniques and compares their performance under similar workloads.

2. Objectives of the Project

This project was designed with the following main goals:

1. Efficient Airplane Scheduling:
   Handle takeoff and landing operations with a focus on proper resource allocation and timing.

2. Use of Priority Queues:
   Give emergency or high-priority flights precedence over regular ones by using a queue-based system.

3. Implementation of Data Parallelism:
   Use dedicated threads for takeoffs and landings to allow them to be handled independently and simultaneously.

4. Implementation of Task Parallelism:
   Incorporate asynchronous function calls to manage different operations more flexibly and effectively.

5. Proper Synchronization:
   Prevent conflicts and race conditions during resource sharing using mutexes and condition variables.

6. Performance Evaluation:
   Measure and compare the effectiveness of the two parallelism techniques to determine which performs better under different conditions.

3. Project Methodology

The project was carried out in several structured steps to ensure clarity, correctness, and efficiency:

Step 1: Designing the Flight Data Structure
A Flight structure was created to hold essential flight information such as:

- Flight ID
- Priority Level (for emergency handling)
- Departure Time
- Landing Time
- Route Information

Each flight is stored in memory and added to either a takeoff or landing queue based on its timing.

Step 2: Implementing the Flight Queues
Two separate queues were maintained:

- takeoffQueue for outgoing flights.
- landingQueue for incoming flights.

These queues used circular arrays and were protected with mutex locks and condition variables to ensure thread-safe access.

Step 3: Reading Input Using the Producer Thread
The Producer thread reads flight data from a CSV file (flights.csv). It parses each line and creates a flight object. Depending on whether the departure time is greater than the landing time or not, the flight is added to the appropriate queue.

Step 4: Creating Consumer Threads
Two Consumer threads are created—one for takeoff and one for landing:

- The Takeoff Consumer dequeues flights from takeoffQueue and simulates the takeoff process.
- The Landing Consumer dequeues flights from landingQueue and simulates the landing process.

Both consumers wait if the queue is empty and resume once new data is available.

Step 5: Synchronization with Mutex and Condition Variables
The system uses pthread_mutex_t to lock the shared queues during enqueue and dequeue operations, ensuring that no two threads access the same data at the same time. pthread_cond_t is used to signal when data becomes available or when a slot is free, helping threads coordinate their execution.

Step 6: Flight Allocation and Priority Handling
Flights are processed based on their priority. Emergency flights (higher priority) are handled first. Regular flights are processed in the order they arrive. Delays are introduced using sleep() and usleep() to simulate real-time takeoffs and landings.

Step 7: Graceful Shutdown
Once all the data from the CSV file has been read and processed, a shutdown signal is sent to both queues. This ensures that consumer threads exit gracefully after processing all remaining flights.

4. Parallelism Techniques Used

Data Parallelism
This technique involves dividing the tasks among different threads that operate on separate data. In this project, two threads handle takeoff and landing independently. This allows both operations to be executed in parallel, improving overall throughput.

Task Parallelism
Task parallelism is achieved by structuring the program in a way where different functions (producer, takeoff consumer, landing consumer) perform different tasks concurrently. Though not explicitly using async in this code, the design imitates asynchronous execution through thread behavior.

5. Performance and Synchronization

The effectiveness of the system heavily relies on proper synchronization. The use of mutexes and condition variables helps avoid problems like:

- Deadlocks (when threads wait forever)
- Race conditions (when multiple threads access shared data unpredictably)
- Starvation (when a thread never gets access to the resource)

The code ensures only one thread modifies the queue at a time and provides signals when data becomes available or space is freed.
Although the code focuses on structure and correctness rather than raw performance benchmarking, its clear organization makes it easy to extend for timing analysis using the <chrono> library or other performance measurement tools.

6. Tools and Libraries Used

The following libraries and tools were used to develop the project:

- C Standard Libraries (stdio.h, stdlib.h, string.h)
- POSIX Thread Library (pthread.h) for multithreading
- Synchronization Primitives: Mutexes and Condition Variables
- Sleep Functions: sleep() and usleep() for timing simulation
- File Handling: To read flight data from the CSV file

7. Results and Expected Outcomes

By the end of this project, we achieved:

- A fully working simulation system for flight takeoff and landing scheduling.
- Clear demonstration of thread-based concurrency using the producer-consumer model.
- Correct use of mutex locks and condition variables for synchronized resource access.
- Application of Data Parallelism and Task Parallelism for better system performance.
- Deeper understanding of multithreading concepts and real-world system simulation in C.

8. Conclusion

The Airplane Allocator project successfully simulates a simplified but realistic model of how airport runways can be managed efficiently using computer programs. It highlights the importance of concurrent programming, resource management, and synchronization when designing systems that handle multiple operations at once.

The distinction between Data Parallelism (same operation on different data) and Task Parallelism (different operations in parallel) is well presented in this project. It provides a great learning experience in implementing these concepts using the C language and POSIX threads.

This simulation can be further improved by adding:

- A Graphical User Interface (GUI)
- Logging and reporting features
- Real-time priority adjustments
- Dynamic runway availability management

Overall, this project serves as a solid foundation for students and developers who wish to explore system-level programming, multithreading, and real-time simulations.