Concordia University
Department of Electrical and Computer Engineering

Introduction to Real-Time Systems
COEN 320

“Real-Time Air Traffic Monitoring and Control”

Team #04

Due Date: 07/04/2025
Instructor: Dr. Bahareh Goodarzi

Table of Contents

Contents

Objectives

The goal of this project is to implement a simplified real-time air traffic monitoring and control (ATC) system that simulates the movement of aircraft in 3D airspace, which aims to support timely detection of safety violations, provide responsive operator control and maintain accurate visualization of air traffic and to handle varying traffic loads reliably.

Introduction

Air Traffic Control (ATC) is a real-time system designed to manage and monitor orderly flow of aircrafts in controlled airspace. Its main purpose is to prevent collisions by keeping safe separation between aircrafts and ensuring efficient traffic movement in both air and ground when landing.

ATC is divided into three main operational areas:

1) Tower Control Area: Which manages aircraft on the ground and near the airport.
2) Approach or Terminal Radar Control (TRACON) area: which handles aircrafts approaching or departing major airports.
3) En-route Area: which oversees aircraft in high-altitude cruising between airports.

This project focuses on a simplified En-route ATC controlling aircraft within a three-dimensional airspace that extends roughly 100,000 × 100,000 horizontally and 25,000 vertically, starting around 15,000 ft above sea level. The system also enforces a minimum separation of 1,000 units vertically and 3,000 units horizontally, issuing alerts whenever a safety threshold is violated. Each aircraft flies at a constant speed and altitude unless the operator intervenes. The intervention occurs when an aircraft is out of the airspace, or two aircrafts are violating the minimum separation. The other main features such as detecting the position of aircrafts and displaying information of aircrafts will facilitate the control of air traffic.

The system can be divided into 6 subsystems: aircraft, radar, computer, operator, display, and communication. Each subsystem is independent that adding a new feature to one subsystem shall not affect the other subsystems.

To communicate among subsystems, share memory and client-server inter-process communication (IPC) are used in the system:

Share memory is one of the techniques for IPC. It creates a share region where processes can read and write. It is important to implement mutex or semaphore to prevent multiple processes from accessing it at the same time.

Client-server IPC is to have one process constantly listening to any request (server). Once another process fires a request (client), the server will receive it and reply to client. The connection between two processes is called channels, and channels shall be attached and detached properly to ensure the message transfer can be completed.

The assumption is made that the priority of aircrafts depends on its ID number. The larger its ID number is, the higher priority it will obtain. For example, aircraft 2 has higher priority than aircraft 1. If aircraft 1 and aircraft 2 are having a collision, the operator shall send a command to change aircraft 1's direction.

An input file is pre-made before the system starts. It contains initial values for each aircraft for testing: ID, x-coordinate, y-coordinate, z-coordinate, speed in x direction, speed in y direction, and speed in z direction.

Analysis and Design

Aircraft Subsystem:
Aircraft subsystem generates aircrafts by reading the data from input file. It must update all aircrafts' position every second and write the updated data into share memory. Computer subsystem reads from share memory and checks if aircrafts had collision or went out of the airspace. Radar subsystem also uses the same share memory to detect aircrafts' current position.

Aircraft subsystem receives message from communication subsystem to change an aircraft's speed. Once the change is made, it replies to communication subsystem.

Additionally, aircraft subsystem shall log the current airspace information into a log file every 20 seconds.

Radar Subsystem:
Radar subsystem reads data from share memory that is created by aircraft subsystem. It converts the data into a x-y plan view and x-z plan view to visualize each aircrafts' position on the screen.

Computer Subsystem:
Computer system reads data from aircraft's share memory every second to check if there is collision or out-of-airspace incident. Once the alert is triggered, it must send a command to operator subsystem to demand a change to flights' speed. If collision alert and out-of-airspace alert occur at the same time, collision alert shall be taken care of first since it is much more dangerous if collision happened. Out-of-bound alert shall be fired after resolving collision alert. Computer subsystem also takes command from operator subsystem to display an aircraft's

information. The command is sent through computer subsystem and arrived at displayed subsystem to be displayed.

To handle sporadic event such as collision alert and out-of-airspace alert, client-server IPC is the best solution to them. Once the server receives messages from clients, it will respond immediately. Thus, all subsystems will implement client-server IPC except radar subsystem since it only needs to read data from share memory.

Operator Subsystem:
Operator subsystem shall be responsive to computer subsystem. When an alert occurs, it must prompt the user to enter the change to an aircraft's speed. When there is no alert, it can prompt the user to enter which aircraft the user wants to display.

Display Subsystem:
Display subsystem receives a message from computer subsystem to display an aircraft's information that the user requests to see. It shall always listen to computer subsystem for a new message.

Communication Subsystem:
Communication subsystem helps computer subsystem to communicate with aircraft subsystem.

A simple block diagram about the system is shown in Fig.1:

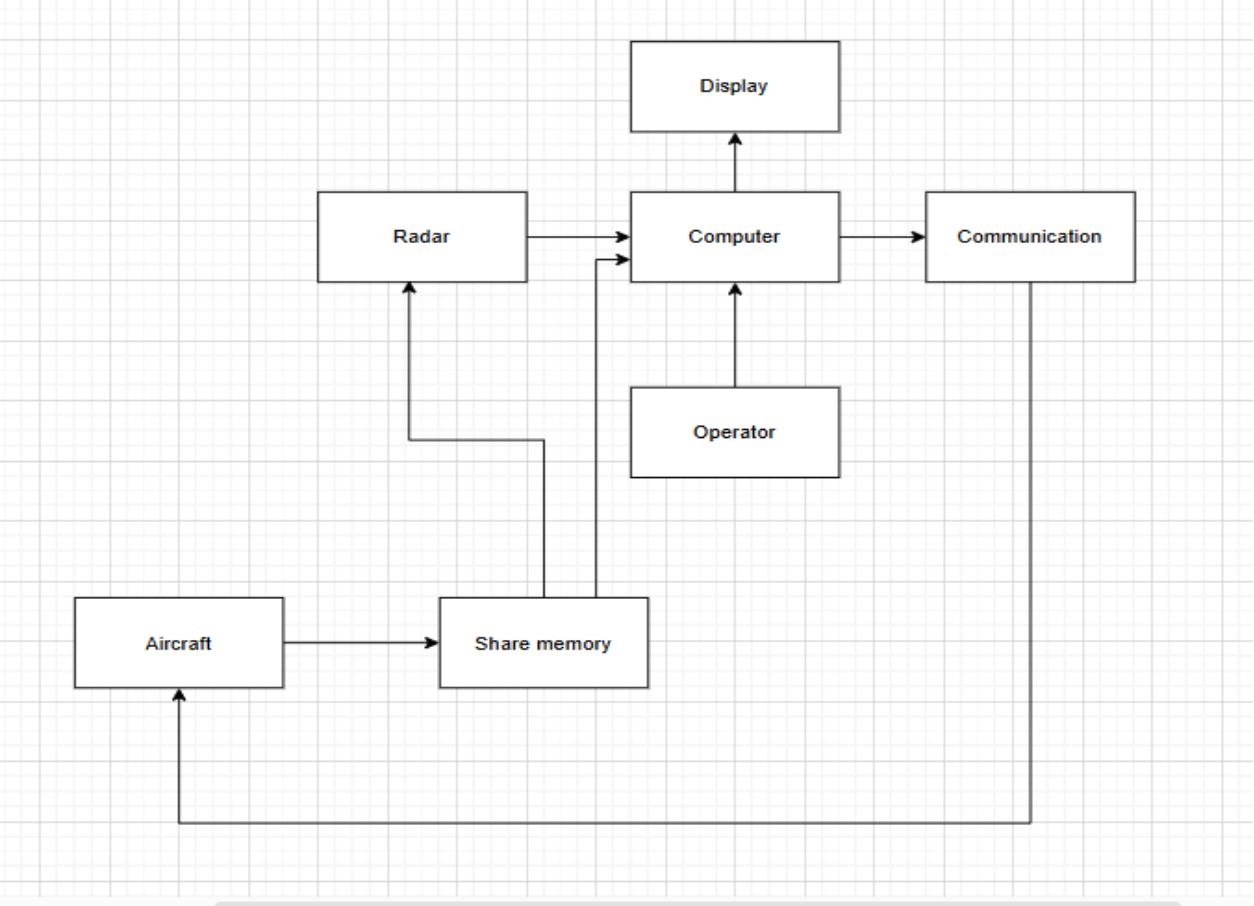

Figure 1. Block diagram

Implementation

Aircraft Subsystem:

Aircraft subsystem has three main tasks: generate aircrafts, receive message from communication subsystem, and log airspace. All tasks shall run at the same time; therefore, multiple threads need to be created. Timers need to be implemented so that the subsystem can update aircrafts' information and log airspace periodically. Fig. 2 demonstrates how aircraft subsystem shall be implemented:

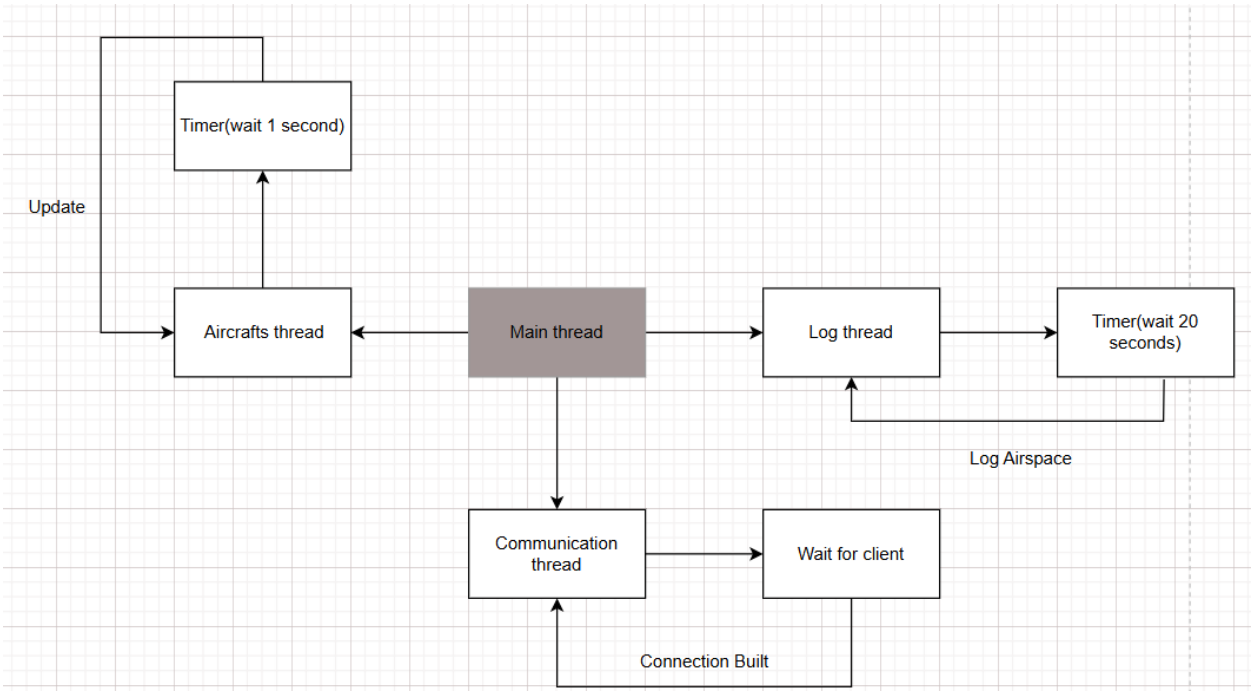

Figure 2. Aircraft Block Diagram

After executing aircraft subsystem, the console prints out aircrafts' position, speed, and status.

```
Flight ID: 1
Flight Position: (50000, 5000, 22000)
Flight Speed: (100, 100, 10)
Flight Status: 1
****************************
Flight ID: 2
Flight Position: (25000, 3000, 30000)
Flight Speed: (100, 500, 0)
Flight Status: 1
****************************
Flight ID: 3
Flight Position: (90000, 500, 15000)
Flight Speed: (1000, 60, 100)
Flight Status: 1
****************************
Flight ID: 4
Flight Position: (21000, 3000, 30000)
Flight Speed: (200, 500, 0)
Flight Status: 1
****************************
Flight ID: 5
Flight Position: (8000, 6000, 17000)
Flight Speed: (500, 50, 60)
Flight Status: 1
****************************
Flight ID: 6
Flight Position: (600, 600, 30000)
Flight Speed: (100, 100, 0)
Flight Status: 1
****************************
```

Figure 3. Aircraft Subsystem Console

```
1 50000 5000 22000 100 100 10
2 25000 3000 30000 100  500  0
3 90000  500  15000 1000 60 100
4 21000 3000 30000 200 500 0
5 8000 6000 17000 500 50 60
6 600 600 30000 100 100 0
```

Figure 4. Input File

Fig 4 shows the pre-made input file which has the same initial value in Fig 3. That means the subsystem has successfully read the file and prints the data in the console.

After 1 second:

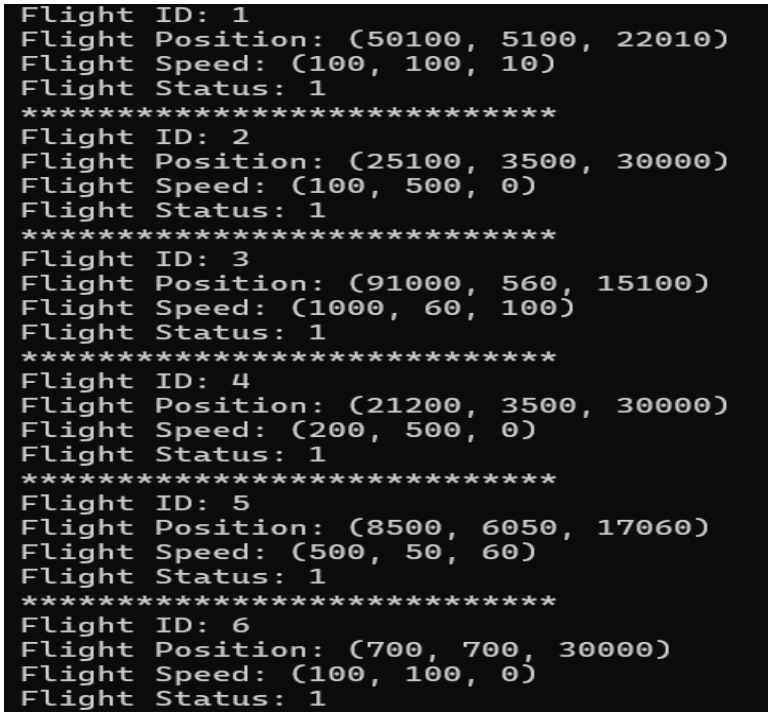

Figure 5. Updated Position

Each aircraft updates its position by adding its own speed because period is one second.
An additional variable called “status” indicates if aircrafts are in the airspace or not. This variable is written in share memory along with aircrafts’ position and speed. The functionalities of radar subsystem and computer subsystem depends on it. When status is one, that means aircrafts are in the airspace. Aircrafts’ status is checked when they are updated.

Radar Subsystem:

Radar subsystem uses one thread to read data from share memory and generate x-y plan view and x-z plan view. Timer needs to be implemented to detect new position every second.

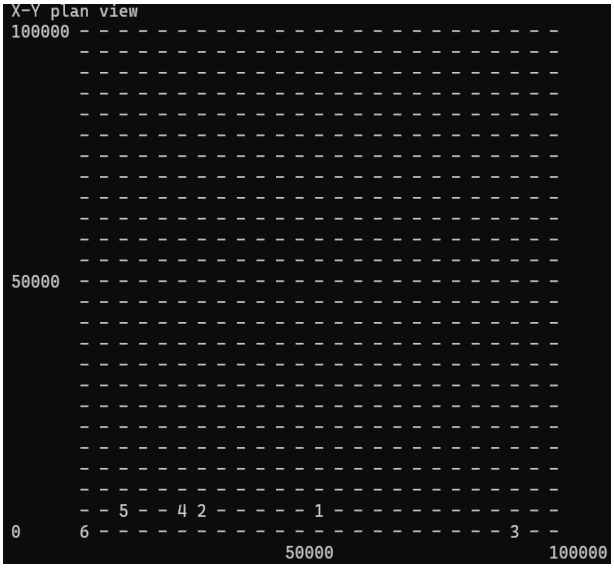

Figure 6. X-Y Plan View

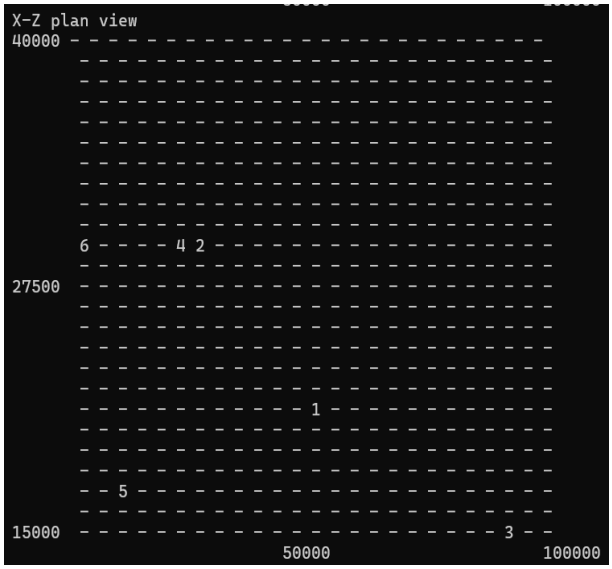

Figure 7. X-Z Plan View

The two plan views in Fig.6 and Fig.7 help user to visualize where the aircrafts are and how far they are from each other. The left bottom corner is the center of radar. User can quickly see that aircraft 6 is the closest to the center, and aircraft 3 is the furthest in terms of x coordinate.

Additional information reagarding each aircraft is printed out as well to verify accuracy of the plan views. Noticed that the values in Fig. 8 is slightly larger than initial value. It is because radar subsystem is started a bit later than aircraft subsystem.

```
Flight ID: 1
Flight level: 72276.9 feet
Flight Speed: 141.774 m/s
Flight Position: (50300, 5300, 22030)
Flight ID: 2
Flight level: 98425.2 feet
Flight Speed: 509.902 m/s
Flight Position: (25300, 4500, 30000)
Flight ID: 3
Flight level: 50196.9 feet
Flight Speed: 1006.78 m/s
Flight Position: (93000, 680, 15300)
Flight ID: 4
Flight level: 98425.2 feet
Flight Speed: 538.516 m/s
Flight Position: (21600, 4500, 30000)
Flight ID: 5
Flight level: 56364.8 feet
Flight Speed: 506.063 m/s
Flight Position: (9500, 6150, 17180)
Flight ID: 6
Flight level: 98425.2 feet
Flight Speed: 141.421 m/s
Flight Position: (900, 900, 30000)
```

Figure 8. Additional Information

Radar subsystem only prints out aircraft that are within airsapce. When an aircraft's status is 0, it must not be printed in the console.

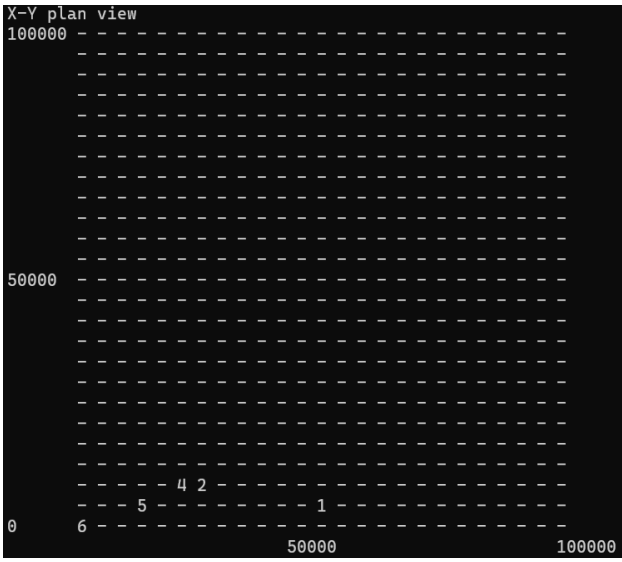

Figure 9. New X-Y Plan View

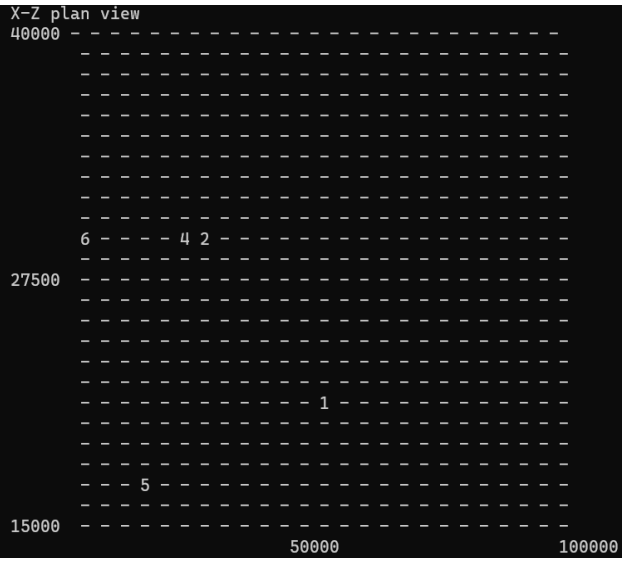

Figure 10. New X-Z Plan View

Aircraft 3 disappears because its status is 0 at this moment.

Computer Subsystem and Operator Subsystem:

Computer subsystem and operator subsystem relies on each other to solve collision alert and out-of-airspace alert. Fig. 11 demonstrates how they work together.

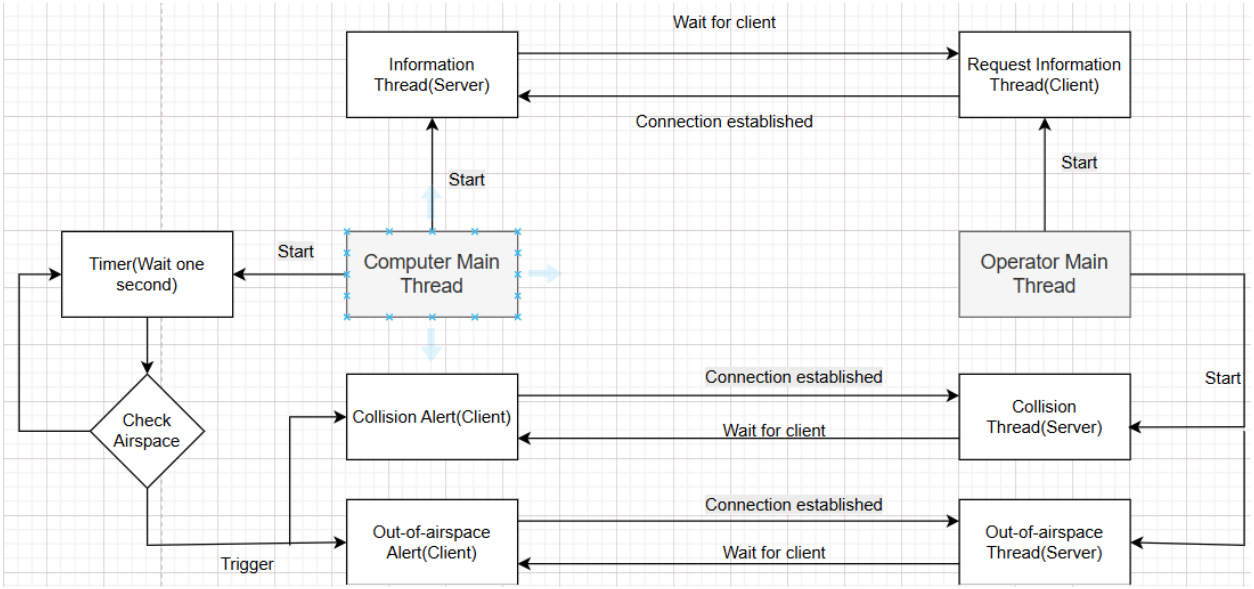

Figure 11. Computer and Operator Block Diagram

Computer subsystem must handle three different tasks: receive display request, emit collision alert, and emit out-of-airspace alert. These three tasks shall have priority that follows the order of collision alert, out-of-airspace alert, and receiving display request. The priority is show as following:

```
[ALERT COLLISION] Aircraft 2 and Aircraft 4 are too close! (Horizontal Dist=2600, Vertical Dist=0)
[ALERT COLLISION] Aircraft 2 and Aircraft 4 are too close! (Horizontal Dist=2600, Vertical Dist=0)
[ALERT COLLISION] Aircraft 2 and Aircraft 4 are too close! (Horizontal Dist=2973.21, Vertical Dist=0)
[ALERT COLLISION] Aircraft 2 and Aircraft 4 are too close! (Horizontal Dist=2973.21, Vertical Dist=0)
[ALERT COLLISION] Aircraft 2 and Aircraft 4 are too close! (Horizontal Dist=2973.21, Vertical Dist=0)
[ALERT OUT OF BOUND] Aircraft 3 is out of bounds!
[COMPUTER SYSTEM] Sending Out of Bound Alert to Operator: Alert detected, request speed change
```

Figure 12. Collision Alert and Out-of-Airspace Alert

Aircarft 2 and 4 is going to collide after the system starts 10 seconds, and at the mean while, aircraft 3 will fly out of airspace after 10 seconds as well. In Fig. 12 , it clearly demonstrates that collision alert is taken care of first. Out-of-airspace alert is only handled after collision alert is clear.

After out-of-airspace alert is clear, the user can request an aircraft's information:

```
[ALERT OUT OF BOUND] Aircraft 3 is out of bounds!
[ALERT OUT OF BOUND] Aircraft 3 is out of bounds!
[ALERT OUT OF BOUND] Aircraft 3 is out of bounds!
[ALERT OUT OF BOUND] Aircraft 3 is out of bounds!
[COMPUTER SYSTEM] Received message: Request Info of Aircraft 2
[COMPUTER SYSTEM] Send message To Display: ID: 2, Pos: (26400,3000,30000), Speed: (0,-500,0), Status: Active
[COMPUTER SYSTEM] Received Reply from Communication: Displayed Successfully
```

Figure 13. Request Aircraft's information

Operator subsystem will respond to computer subsystem depending on which event is triggered:

```
[URGENT COLLISION] Enter new speed command for Aircraft 2
Enter new speeds (e.g., 100 200 300): [Operator Console] Sending message to server: Request Info of Aircraft 1
[Operator Console] Received reply from Computer System: Data Sent to Display
0 -500 0
Captured collision command: '0 -500 0 '
Sent collision speed command: 0 -500 0
[Operator Console] Enter Aircraft ID for Display (or 0 to wait): 1
[Operator Console] Sending message to server: Request Info of Aircraft 1
[Operator Console] Received reply from Computer System: Data Sent to Display
[Operator Console] Enter Aircraft ID for Display (or 0 to wait): 2
Received out-of-bounds alert from ComputerSystem: Alert detected, request speed change
[URGENT OUT OF BOUNDS] Enter new speed command for Aircraft 3
Enter new speeds (e.g., 100 200 300): [Operator Console] Sending message to server: Request Info of Aircraft 2
[Operator Console] Received reply from Computer System: Data Sent to Display
-1000 0 0
Captured out-of-bounds command: '-1000 0 0 '
Sent out-of-bounds speed command: -1000 0 0
[Operator Console] Enter Aircraft ID for Display (or 0 to wait): 2
[Operator Console] Sending message to server: Request Info of Aircraft 2
[Operator Console] Received reply from Computer System: Data Sent to Display
[Operator Console] Enter Aircraft ID for Display (or 0 to wait):
```

Figure 14. Operator Console

Display Subsystem:

Display subsytem shall constantly wait for the message from computer subsystem and display aircraft's information in the console. Figure 15 shows how display subsystem interact with computer subsystem.

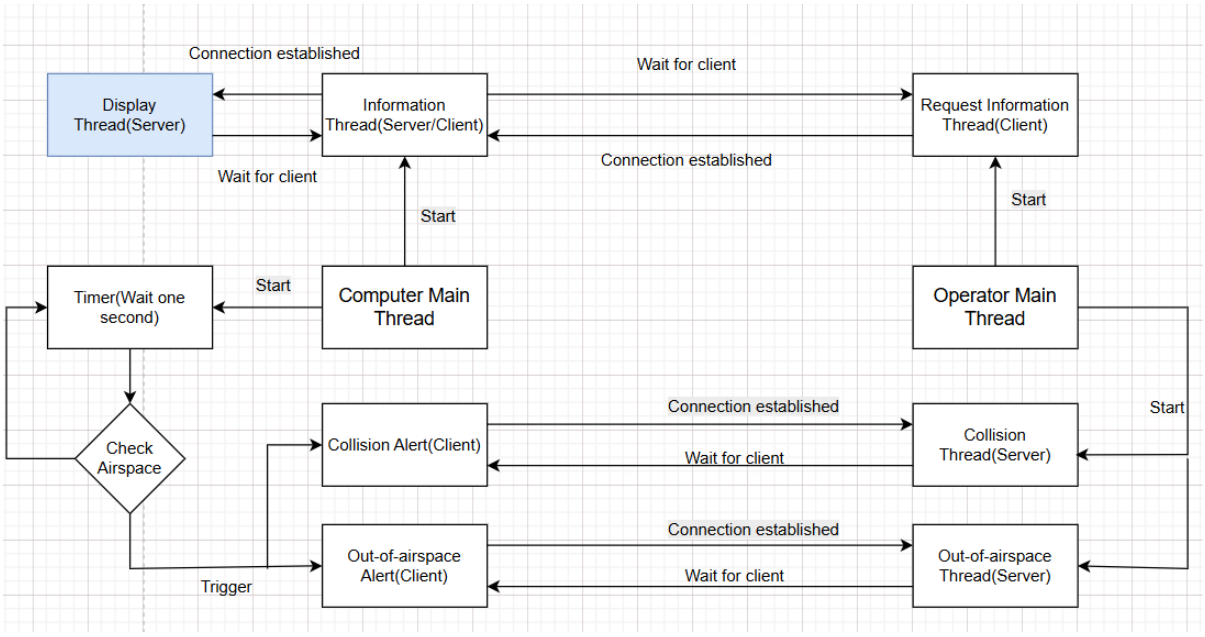

Figure 15. Display Subsystem

Information thread act as server and client at the same time because it waits for message from operator subsystem and sends the same message to display subsystem.

A display console is shown in Figure 16:

```
Display server start........
[Display] Server is running, waiting for messages...
[Display] Receive message from computer
[Display] Aircraft Info: ID: 1, Pos: (50500,5500,22050), Speed: (100,100,10), Status: Active
[Display] Receive message from computer
[Display] Aircraft Info: ID: 2, Pos: (25600,6000,30000), Speed: (100,500,0), Status: Active
[Display] Receive message from computer
[Display] Aircraft Info: ID: 3, Pos: (96000,860,15600), Speed: (1000,60,100), Status: Active
[Display] Receive message from computer
[Display] Aircraft Info: ID: 4, Pos: (22200,6000,30000), Speed: (200,500,0), Status: Active
[Display] Receive message from computer
[Display] Aircraft Info: ID: 5, Pos: (11500,6350,17420), Speed: (500,50,60), Status: Active
[Display] Receive message from computer
[Display] Aircraft Info: ID: 6, Pos: (1300,1300,30000), Speed: (100,100,0), Status: Active
[Display] Receive message from computer
[Display] Aircraft Info: ID: 1, Pos: (51100,6100,22110), Speed: (100,100,10), Status: Active
[Display] Receive message from computer
[Display] Aircraft Info: ID: 1, Pos: (51500,6500,22150), Speed: (100,100,10), Status: Active
[Display] Receive message from computer
[Display] Aircraft Info: ID: 2, Pos: (26400,8500,30000), Speed: (0,-500,0), Status: Active
[Display] Receive message from computer
[Display] Aircraft Info: ID: 2, Pos: (26400,3000,30000), Speed: (0,-500,0), Status: Active
```

Figure 16. Display console

Communication Subsystem:

Communication subsytem will receive the message from computer subsystem and send the same message to aircraft subsystem. Figure 17 demonstrates how communication subsystem interact with computer subsystem and aircraft subsystem.

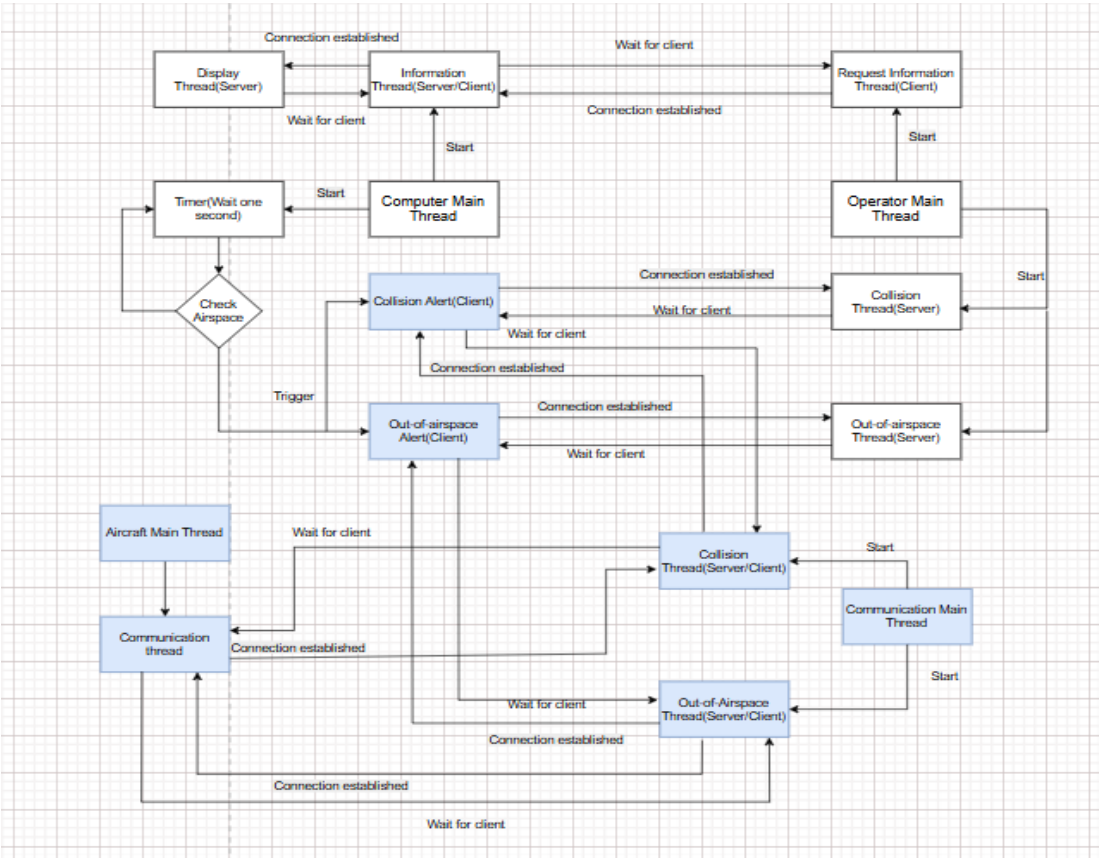

Figure 17. Communication Subsystem

Difficulties :
Timer, share memory, and client-server IPC are new concepts in this course. It takes time to understand and implement them in the code. On top of that, both timer and share memory require to use pointer; therefore, a lot of memory allocation errors have occured when the code is compiled. It actually takes more time to debug those errors than write the code.

Apart from writing the code, testing the system is another important part of this project. A system shall be robust and reliable to handle different scenarios. A lot of errors happen after testing the system especially task scheduling. Scheduling sporadic tasks are tricky that when two sporadic tasks occur at the same time, the higher-priority shall be handled first. It involves multiple if-else statements , variables, and functions to ensure priority rule is followed strictly. In this project, collision must not happen. Thus, it must be taken care of and resolved first.

Improvement:
The console of each subsystem could have been more user-friendly especially computer subsystem and operator subsystem. Due to the functionalities of these two subsystems, their consoles are full of text for communicating with other subsystems. It makes users extremely hard to understand the system and send the commands to solve current issues. Enhancement of understandability of the code would be first priority if the system is implemented in reality.

The code shall be refactored to make it more readable that other developers can easily locate what they are looking for.

Lesson Learned

Instead of testing all the features after the code is complete, it is better to test each subsystem individually once the code of that subsystem is done. It will make debugging process a lot easier and modify the code before it is too late. For example, display subsystem is supposed to display the current aircraft's position and its position in the next n seconds (n is positive integer); however, the way display subsystem is designed can only print the current position. If display subsystem were to print aircraft's future position, it must access share memory to fetch new data. The change will require modifying the code in display subsystem and computer subsystem which can be annoying.

A good understanding of what features the system needs will also help reducing the complexity of the system. If the block diagram in Fig. 17 were present before writing the code, it would have made the code simpler and more understandable.

Conclusion

The system has completed both periodic tasks and sporadic tasks successfully. Aircraft subsystem can update the share memory periodically so that radar subsystem and computer subsystem read the latest data. Client-server IPC works well between computer subsystem, operator subsystem, display subsystem, communication subsystem, and aircraft subsystem. The channels are attached and detached properly when receiving or sending messages. Sporadic tasks are handled according to tasks' priority, and collision alert shall be clear first as soon as possible. The out-of-airspace alert will be handled once collision alert disappears. Requesting aircraft's information is executed when there is no alert occurring.

The system must pass more tests before it is released to the public. There are only 6 aircrafts feeding to the system, and that is clearly not enough in reality. The system shall handle at least 1000 aircrafts to ensure the reliability of it. User interface shall be more user-friendly that any person can get familiar with the system quickly. Apart from these flaws, the system is well-functional, and it can manage air traffic.