Air Traffic Monitoring and Control (ATC)

COEN 320
Introduction to Real-Time Systems

Due Date:
April 10, 2024

A Report
Presented to
The Department of Electrical & Computer Engineering
Concordia University

In Fulfillment
of the Requirements
of COEN 320

by
Omar Dabayeh (40100195)
Email: odabayeh@gmail.com
Hashim Nakhuda (40068968)
Email: hashim.aduhkan@gmail.com
Karyn Gamay (40044481)
Email: k_gamay@live.concordia.ca

Concordia University
April 2024

Table of Contents

Objective

The objective of this project is to implement a simplified Air Traffic Monitoring and Control system using the QNX real time operating system.

Introduction

An ATC, short for Air Traffic Monitoring and Control system, is designed to govern airspace, ensuring the safe movement and navigation of planes under different circumstances.The objectives of air traffic control system is to first maintain orderly air traffic flow and separation between planes for safe plane movement and to avert collisions between planes. For this project, the ATC system involves three categories: the tower control area, the terminal radar control area (TRACON) and the en-route area. The tower control area is responsible for the movement of planes within the airport. The TRACON is responsible for controlling the departure, the arrivals, the overflights of the planes. Finally, the en-route area controls planes when they leave the TRACON area, so when the plane reaches their cruising speed and altitude. This project involves simulating a real-time airspace, within the en-route area, which contains planes in different congestion traffic levels, low, medium and high.

There are a couple of assumptions that our team had to take into consideration when designing the system. For the Operator console, we assumed that the user will not make any mistakes when they enter the task number they want to run, the plane id and the values needed to complete each task. For the safety violation, we assume, but still mentioned, that the user will enter a value between 0 and 180. We also assumed that the airspace begins at 0.

ATC System Interpretation

The ATC system focuses on the en-route control system. The area of this system is a 3d prism with dimensions *x=100 000, y= 100 000, and z=25 000* is located 15 000 ft above sea level from the lowest point on the prism. The system should allow overall management of aircrafts within the space. This includes maintaining and changing their speed or altitudes and alerts for crashes and predicting collisions in the airspace (3 minutes). The requirements include a Data Display that shows the positions of each aircraft every 5 seconds. As well as a logging system that shows a summary of the airspace every 30 seconds. Aircrafts are implemented as individual threads updating their location every second.

As shown below, there is an ontology created for this project. An overview of the main requirements of each component:

- Radar: To detect and track aircrafts while routing the message to the Computer System.
- Computer System: Responsible for determining if there are any safety violations in the airspace. If there is a safety violation within 3 minutes or a current crash occurring, the system will notify the operator. Finally, Computer System routes incoming plane messages from Radar to Data Display
- Operator Console: Enables the user to use a command line interface to change altitude, velocity, period of safety violations and focus on a plane. Also stores cmd logs.
- Data Display: Displays a plane view of the airspace every 5 seconds. As well, stores logs of the airspace.
- Communication System: Responsible for transmission of commands to aircrafts.

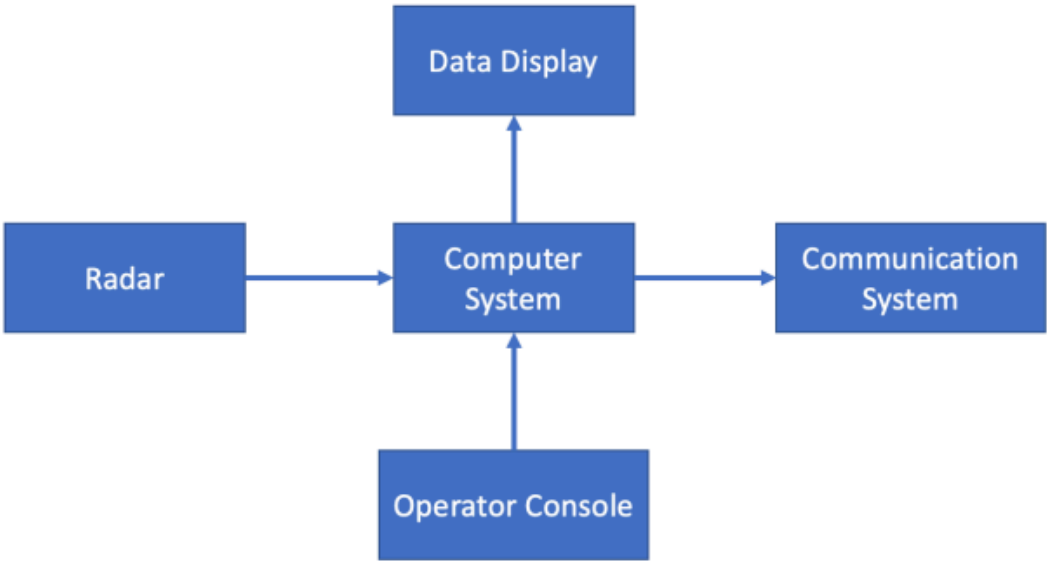
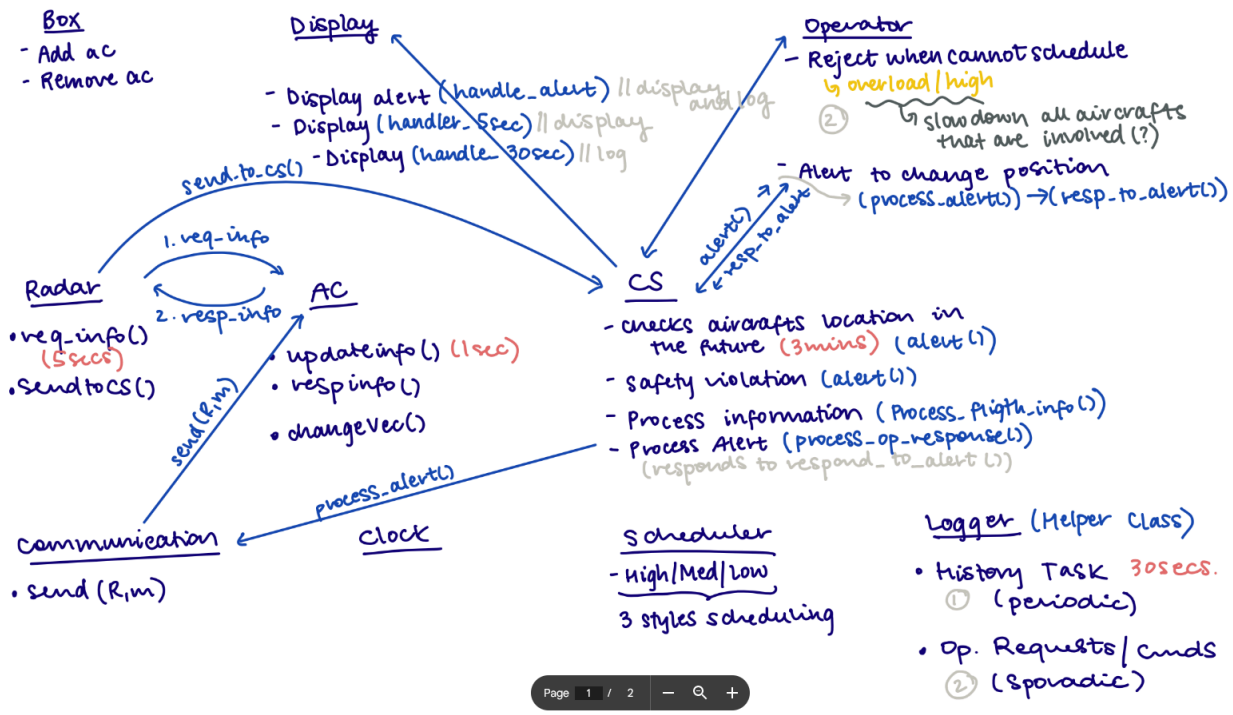


Fig. 3. Components of the simplified ATC to be implemented

Design



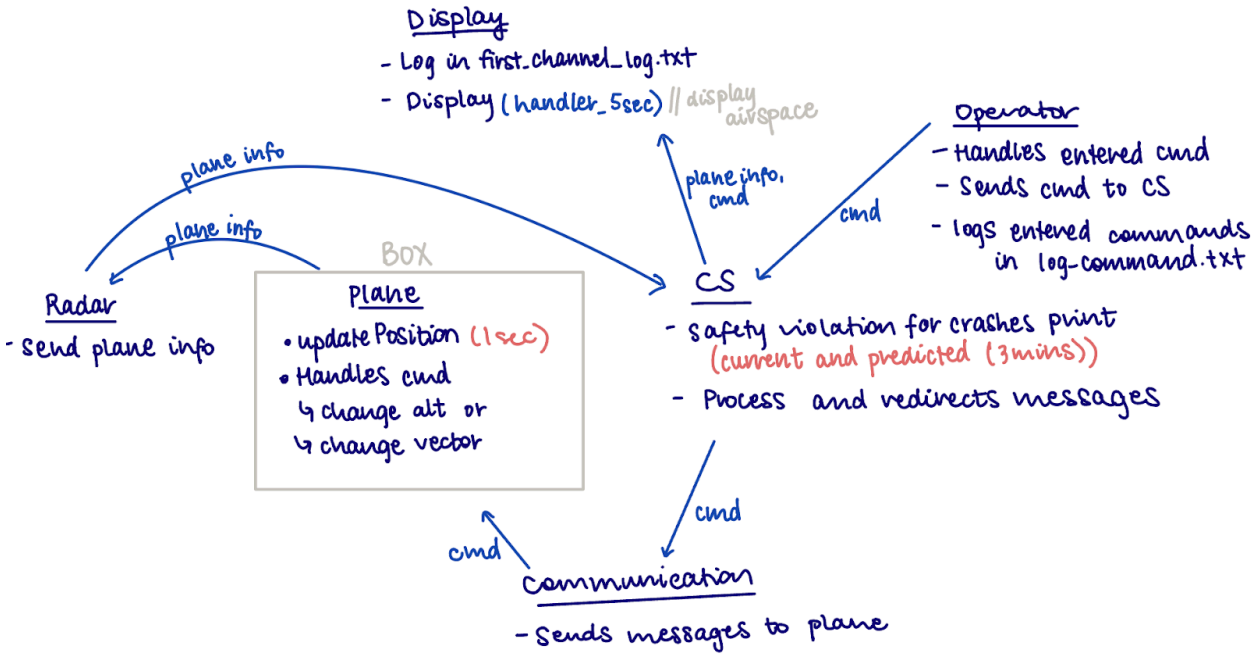
We held multiple meetings to brainstorm the architecture of the system. After some iterations, we came up with the design shown above.

Originally Data Display was supposed to display safety violations (alerts), display a plan view of the space and log the airspace every 30 seconds.

The interaction between radar and plane originally was based on a flawed understanding of how a client to server communication worked. It was understood that a client would make a request for a response then the server would deliver what was requested if it was not busy. With this we chose to make radar the client and plane the server. Later we learnt that this was wrong and that radar should be the server and plane the client. Box was designed to hold planes and handle anything to do with planes entering or leaving the airspace however, later it was seen to be simpler to just have planes handle themselves entirely so in the implementation we will see below box no longer needs remove ac but will need add to start all the plane threads.

When we first were discussing the project, we had the impression that the operator and the computer system had to interact with each other to be able to tackle a given situation. After asking questions, we thought that we did not need a new class for the Operator and that the moment a plane found that they would be heading towards a crash, the Computer System would ask the controller, cin, for new values to change the plane's trajectories.

Implementation



Solutions to some of the requirements(others discussed each component):

- The distance between each plane can be solved by scaling our airspace down to respect these dimensions this way if 2 planes are on the same point we know they are within these boundaries.
- The safety violation period being dynamic is resolved by having a flag be set and storing the changed period by the computer system when it receives a message from the operator.
- Planes leaving the airspace are resolved by planes checking if they are out of bounds and then sending one last message to indicate to the other components by setting its x position to -1. When a computer system or data display sees a -1 on the x position they will know to no longer track the plane.

In safety violation it was desired to get the exact time at which the planes crashed however given the implementation choice of using mathematics instead of using nested forloops every second the time became rather complex to acquire sometimes we would get a division of zero. Since this was more of a want rather than a requirement we chose to omit this.
When implementing message passing there were many issues faced. As mentioned in design for main the actual order in which the threads start were crucial and often caused issues where our clients would try to open an attach point that was not yet created.

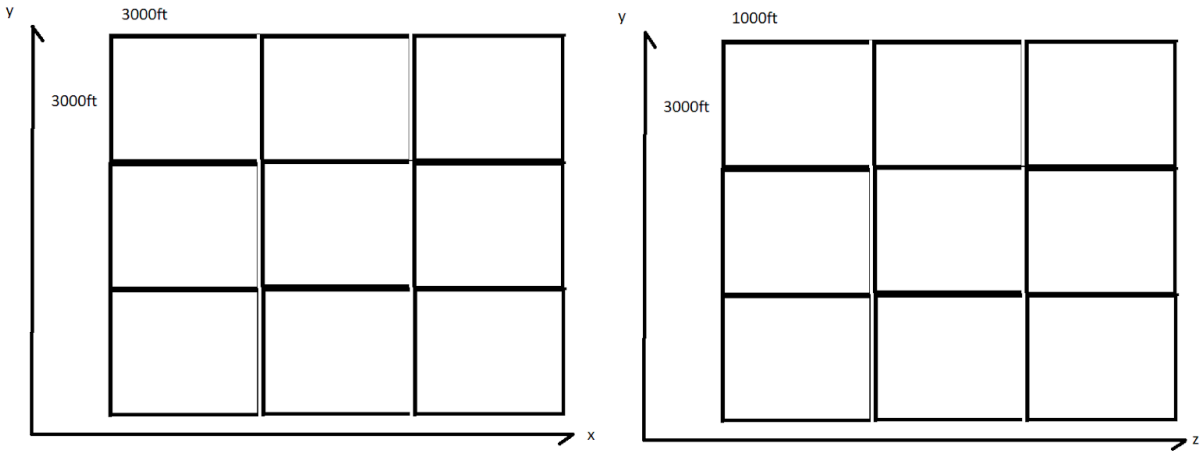
Main

There are currently three mains that can be run. Each one demos a separate simulation with regards to the three congestion levels. They first create a file for the test data of planes. This file will run every time to ensure that the file will exist. It then starts the threads for radar, computer system, data display and communication. Followed by reading the file input to create the corresponding planes. The order here is crucial since planes are the only ones who start all the message passing; this means if planes were initialized first they would be sending messages to a server that doesn't yet exist. After everything is initialized the program simply runs on its own with the given initial values. The corresponding threads are then joined back to main.

Radar

In this project it was a requirement to make our parts modular. This was factored in when designing radar. The current radar is a message passing server waiting for plane clients to send to their updated position every 1 second. It then immediately sends this message to the computer system. No processing is needed here however, this allows us to expand on the functionalities of radar in the future.

Data Display



In the final implementation, Data Display has two main functions. Compared to the old design, the display will no longer handle safety alerts as the Computer System will perform the displaying of alerts itself. DD will still log as planned and it will display a plan view as planned.

The logging occurs every time a message is received. This creates a log file with all the plane info (ID,x,y,z,vx,vy,vz).

The plan view is a 33x33x25 matrix. We chose this size so that the output is more readable. Each unit in the x or y direction is 3030ft, because 100 000/ 33 = 3030.30 ft. Every unit in the z direction is 1000 ft. The conversion of coordinates is completed in this class before displaying planes.

The code utilizes two threads for this class. The first thread is the display thread and the second is the server thread. The display thread runs the function *runServer* where a channel is created for messages from the Computer System. The first thread calls the *runDisplay* function which runs the *displayData* function every 5 seconds. *displayData* prints a matrix as mentioned above and places planes from the messages in the right position. In the second thread, a server is created and listens to incoming messages at *ATC_Server1*.

If *focusOnPlane()* is called then the *focusedPlaneId* is changed and the server only prints new locations of the focused plane.

logMessage() is called every time a message is received. This was one of the complications we had during the project. Storing the messages and periodically updating the log was a challenge.

Computer System

The Computer System is responsible for processing and redirecting received messages. The following are the details of the different tasks that the Computer System is responsible for.

Safety Violation

When receiving a message from the plane it will check if it needs to store or update the plane's information depending if it has the plane's id already this will be used in safety violation. The safety violation is a periodic thread where its period can be changed during runtime. This was done by the computer system setting a flag and storing the new period whenever the operator sent the command to change the period. The safety thread on the next run will then check the flag and it is set to update the period of its timer then resetting the flag. The actual logic on how safety violation works is by first checking for an immediate crash if so displaying it to console as

an alert then checking for future crashes within 3 minutes of the current planes positions we have in a map for tracking all the planes in airspace. To check for future crashing some linear mathematics are used for getting the intersection point of two lines in a 3d space.

To Data Display

When a message struct from the Operator, of the form id+“focus” is received, the Computer System will redirect the command to Data Display to ask to print out the display of that plane.

To Communication System

The Computer System sends messages to the Communication System if the command from the Operator asks to change either the altitude or the vector of a specific Plane.

From Operator

The Computer System receives messages from the Operator because it will be the one to decide where the received messages will go next. Messages received from the Operator are of type 0x01. Once the received messages are within the if statement, it will search the cmd in the struct to find one of the following words: safety, focus, alt or vector. If the word safety is found, the Computer System will set the safety period to the new value. If the word focus is found, the received message will be sent to the Data Display. If the words alt or focus are found, the received message will be sent to the Communication System which will then send it to the Plane.

Operator

The Operator is the console who waits for inputs from the controller. Our project assumes that the controller is fully knowledgeable on the different tasks and how they are used. The Operator is a thread and uses the plane_data struct to send messages to Computer Systems. It starts by introducing a menu of 4 tasks of the different tasks the operator can perform. The first task on the menu is to change safety violation time. It asks the user to enter a new value for the safety validation period. We assume that the operator will enter a value that is between 0 and 180. Once the controller has entered a value, the value will be appended to the word safety and will become the cmd in the struct that will be sent to the Computer System. The cmd string that will be sent is “safety/n”. The second task on the menu is to focus on a plane. For this task to be selected, the controller will have to enter 2. Then the Operator will ask for which plane ID would

the controller want to focus on. Once the value has been entered it will be equal to the id within the struct and the word focus will be equal to the cmd in the struct then will be sent to the Computer Systems. The third task is to change the altitude by changing the acceleration on the z-axis of a specific plane. For this task to be selected, the controller will have to enter 3. Then the Operator will ask for which plane ID would the controller want to change their altitude and by how much. Once that is done, the plane ID will be equal to the id variable in the struct and “alt/entered value” will be appended together and equal to the cmd of the struct. Once that is done, it will be sent to the Computer System. Finally the final task, where the controller must type in 4 for it to be selected, is to change the vector. If this task is selected, the Operator will ask for which plane ID and new values for X,Y and Z. Note that, we don’t allow for the user to enter values for both X and Y to avoid diagonal movement of the plane. If both X and Y are not equal to 0, the controller will be warned and will have to re-enter 4 to be able to restart the task. If all goes well, the id in the struct will be equal to the entered plane ID and the cmd will be equal to “vector/x/y/z” then the whole will be sent to the Computer System.

Communication System

The communication system’s sole responsibility is to send commands entered in the Operator Console, processed within the Computer System then finally to the specified plane. When the message is received in the Communication System, it will take the message and use the send(plane_data struct) within the system that will send the message over to the specified Plane thread.

Box

This class is simply a list of all the planes. When the constructor is called it will read the given in file corresponding to the specific congestion level then create planes for that line and once that is done it will start their threads. It also holds a function to join the threads for all the planes for convenience.

Planes

Plane represents a simulated plane with functionalities for updating its position, periodically sending its position to Radar and receiving commands from the Communication System. Each plane is initialized with 8 attributes: ID, the time it enters the airspace, x/y/z positions, and x/y/z

acceleration. After updating its position it will check if it left the airspace and if it did the next time it runs it will set its x position to -1 and send the message to radar. This indicates it has left the airspace then after the message is sent it will break out of the while loop thus ending its run.

Lessons Learned

We assumed that we were able to separate the systems amongst ourselves to work on them individually, but soon realized that the different components relied on each other. So we had to start working from the bottom up, meaning starting with planes. When having to debug various issues it was best to just have a second project with only the required components to have a sort of isolated system and then once that feature worked we could re-integrate the fixed version. Another lesson was as stated earlier how message passing actually works and that client is not meant to make a request for information. In the future when designing a system with message passing this will be easier to design.

Conclusion

We worked with software to simulate a real time system. Creating our own design as well as implementing the design gave us a more in depth understanding on how to better design these systems and what to look out for or consider during the design phase in future projects. Overall our project worked relatively well but took a lot of time to implement due to some issues faced and some lack of understanding. For instance how client to server works in message passing, the ordering of starting threads, how to divide the workload for this kind of project and what aspects are more complicated than others which was different than originally thought because each component was dependent on another. However, we were able to overcome these obstacles and completed the majority, if not all, the tasks we had wanted to complete for this project.