SYSC 3303 FINAL REPORT

Justin Whalley, Yisheng Li, Yixiang Huang, Gurjit Gill

GROUP: Lab 5 Group 5

Table of Contents

Table of Figures

Table of Tables

Responsibility Breakdown

All:

- All diagrams
- Report
- Bug fixes in weekly meetings
- README.txt

Justin:

- Junit testing
- GUI related programs
- Test.java to run input file
- UDP in floor subsystem
- Timer class
- Elevator UDP and edits
- Motor state interface and related classes
- Half of common class
- Integration of settings (not including port settings)

Yisheng:

- Scheduler class
- ElevtState class
- FloorState class

Yixiang:

- Elevator Subsystem class
- Floor subsystem class
- Command bridge
- Half of common class
- RPC class
- Port settings

Gurjit:

- Elevator class
- FileLoader logic

UML Sequence Diagram

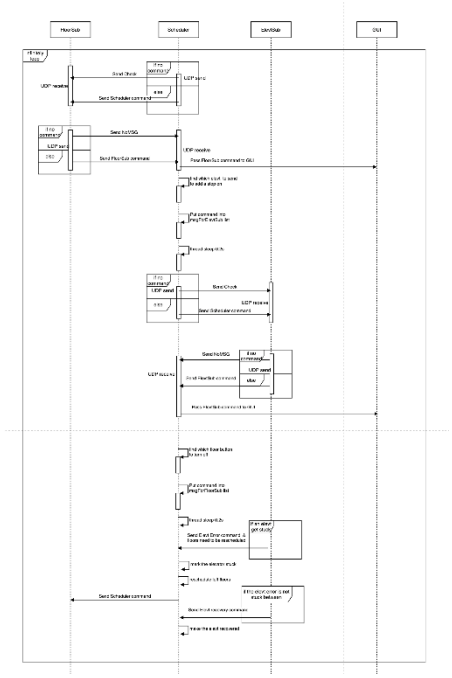

Figure 1 sequence diagram

UML Class diagrams

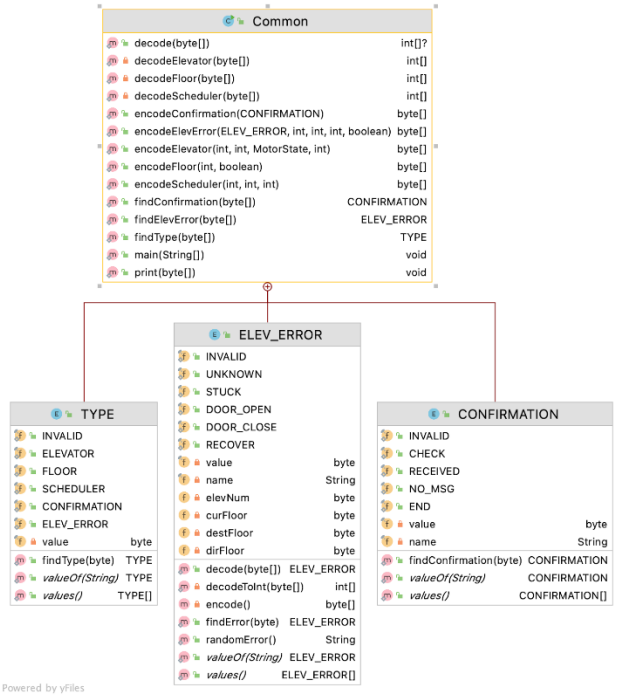

Figure 2 Common class diagram

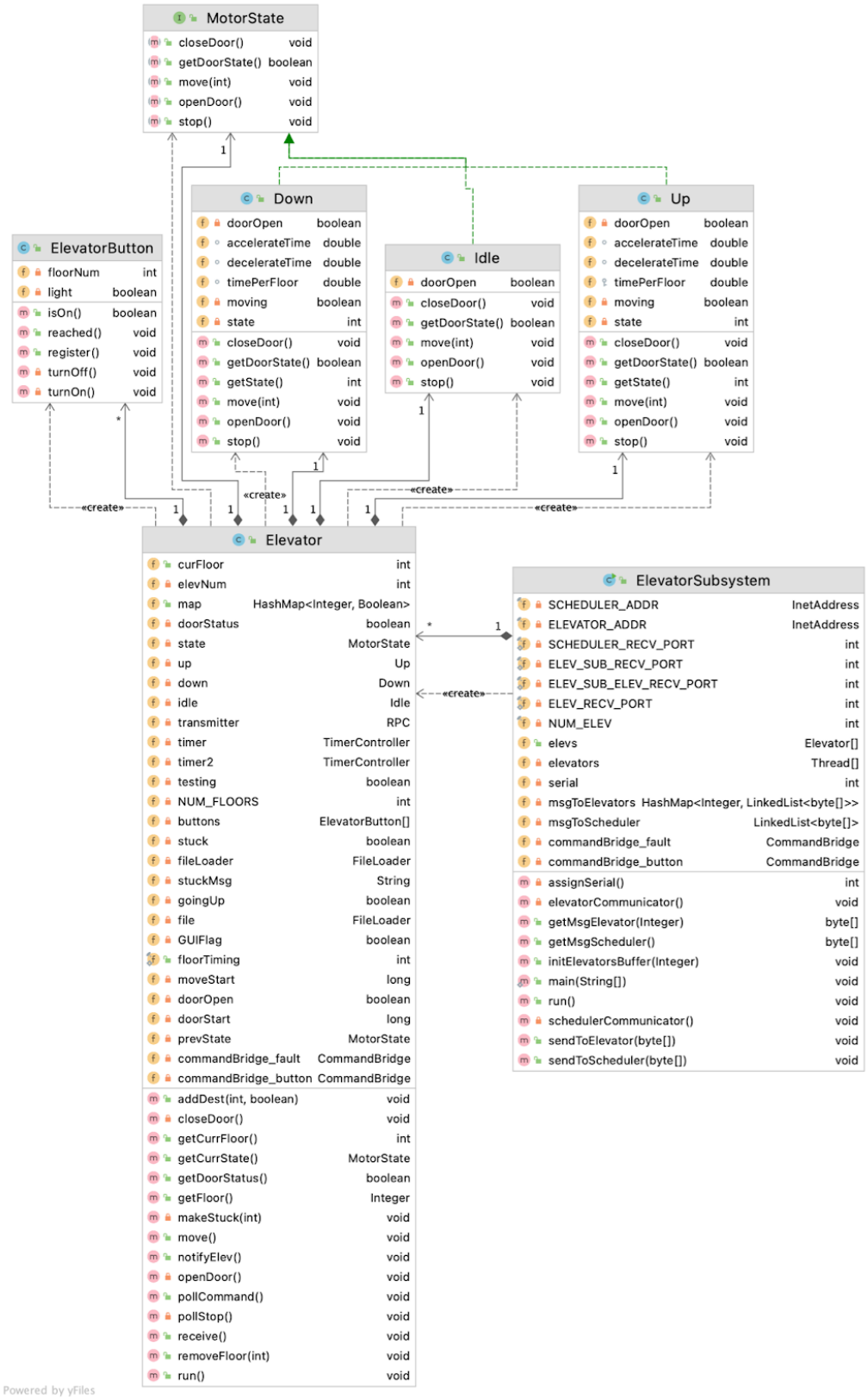

Figure 3 Elevator Subsystem class diagram

Figure 4 Floor Subsystem class diagram

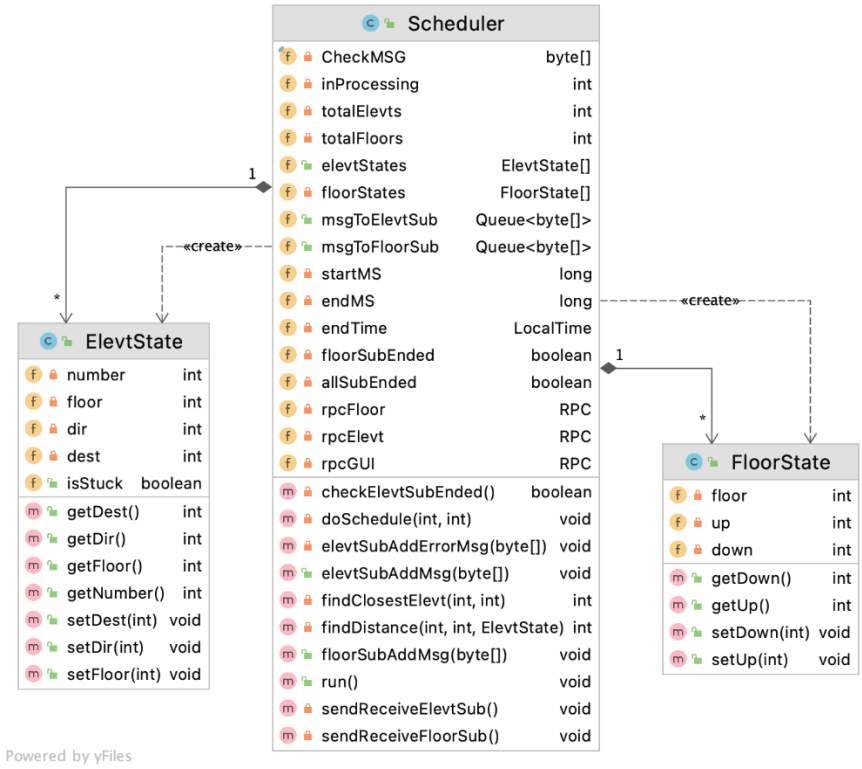

Figure 5 Scheduler class diagram

Figure 6 GUI class diagram

Timing Diagrams

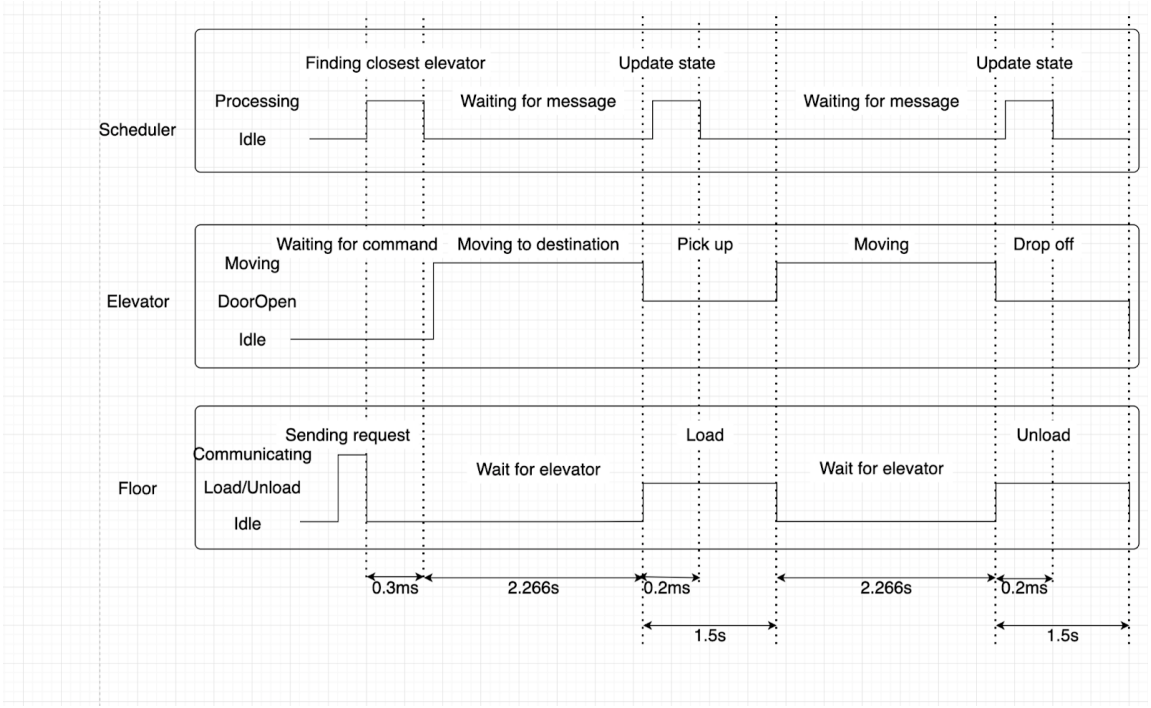

Figure 7 Timing diagram no faults

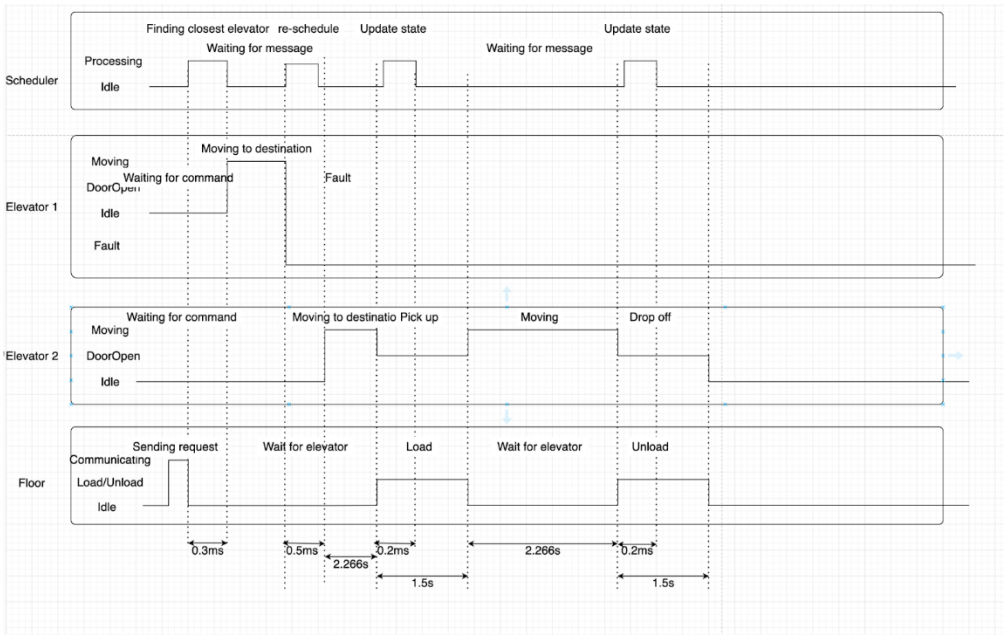

Figure 8 Timing diagram with permanent fault

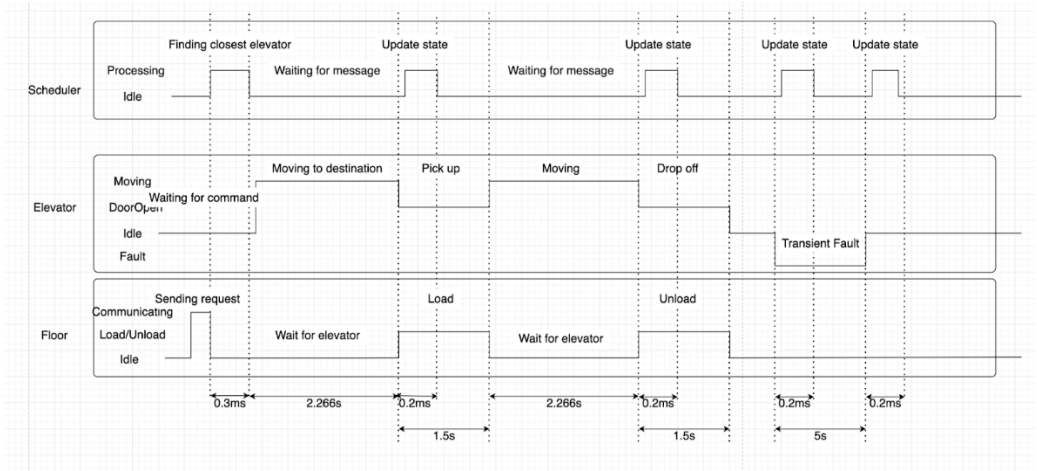

Figure 9 Timing Diagram with transient fault

State Machine Diagrams

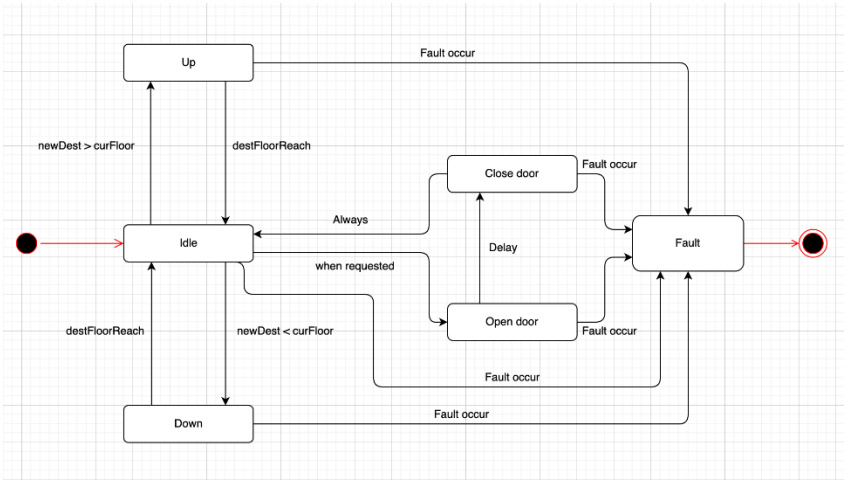

Figure 10 Elevator state machine diagram

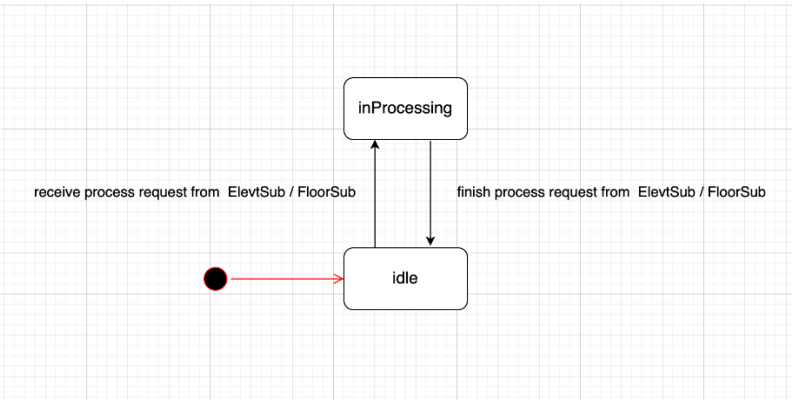

Figure 11 Scheduler state machine diagram

Set Up Instruction

1. Unzip source code folder
2. Ensure ports are not in use on the specific system
    a. If port is taken, modify ports in settings.txt
3. Choose the speed of the system
    a. Modify SPEED value in settings.txt
4. Choose number of floors for the system
    a. Modify FLOORS value in settings.txt
5. Choose a number of elevators for the system
    a. Modify ELEVATORS value in settings.txt
6. If GUI mode
    a. Run GUI.java
7. If test file
    a. Set number of errors you want
        i. Modify ELEV_ERR value in settings.txt
    b. Set number of commands to create
        i. Modify ROWS value in settings.txt
    c. Run Test.java

NOTE: GUI is currently going to create all subsystems, if wanting to run on separate computers, change variable CREATE_ALL_SUBSYSTEMS to false then run the subsystems on their own computers.

GUI Explanation

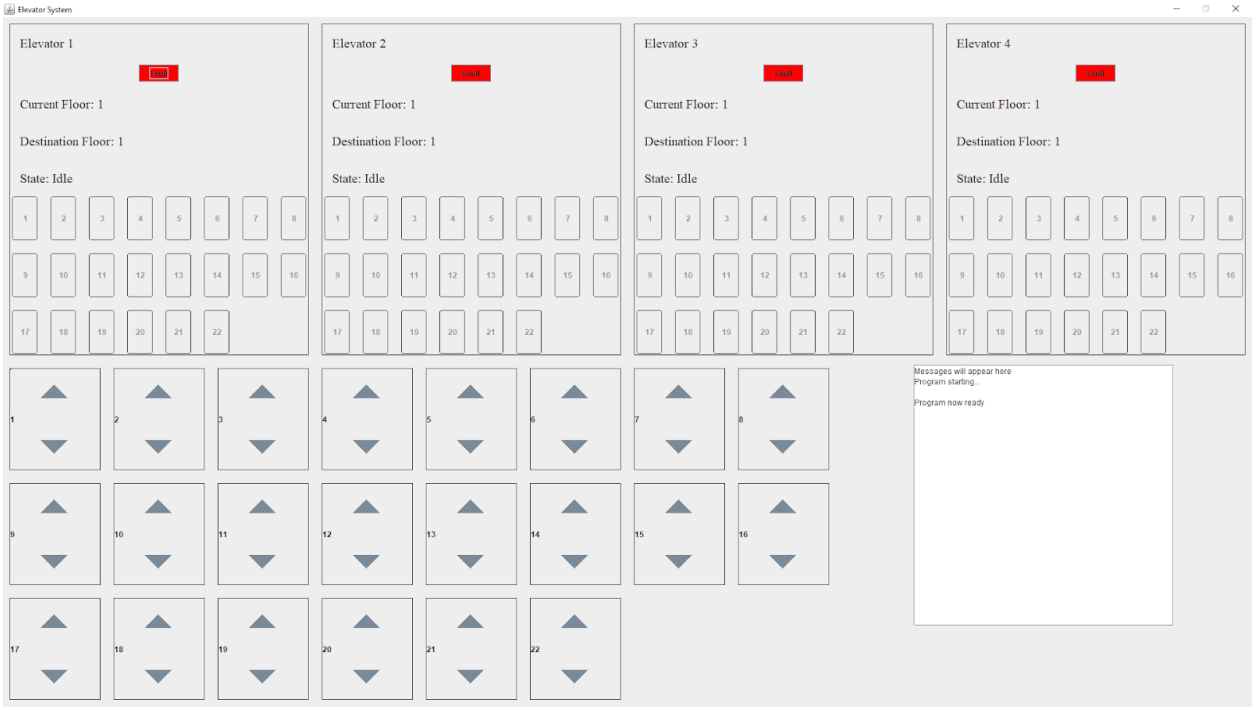

Figure 12 GUI screenshot for explanation

The GUI, as seen above, has 3 main sections. The top half represents each elevator in the system, the bottom left contains the floor buttons with up and down buttons, and the bottom right is the console displaying the GUI events that have taken place. The elevator buttons will be disabled until the elevator picks up the first person and will stay on until program exit. Therefore to activate the elevator buttons a floor button must be clicked and the elevator you want the buttons to be enabled on must be the elevator scheduled. The red fault button in each elevator will fault the elevator to whichever state the elevator is currently in (i.e. if the elevator is between floors, it's a stuck between floors fault). The current floor, destination floor and the state for each elevator is updated whenever the elevator sends an update to the scheduler (at every floor change). The frame is set to not resizable to ensure everything is readable, and so if you withdraw it full its full screen state any way other than the minimize button, it will not be able to be made bigger.

Testing Instructions

1. Unzip source code folder
2. Ensure the ports are not in use on specific system
3. Run JunitTestCases.java

Note: due to the randomized order of the tests, sometimes a test will fail when it is unexpected (the state is assumed to be something that it isn't in an assert statement). If this occurs, highlight the test and run the test individually, it should pass once done.

Measurement Results

Elevator to move between floors with speed = 1	2.266 s
Door open with speed	1.5 s
Scheduler to process a Floor Subsystem request	0.3 ms
Scheduler to process an Elevator Subsystem request	0.2 ms
Scheduler to process a rescheduling request	0.5 ms

Table 1 Measurement Table

Schedulability Analysis

The results from the previous section show that the system takes minimal time to schedule and communicate with the respective elevators with respect to the time to move between floors. This is optimal as the time to schedule should be as low as possible. The scheduler uses saved states and integer representations of the distances between the destination floor to accurately and quickly decide the optimal elevator to go to pick up. Another optimization we included was the elevator will read its own elevator button clicks which then removes the unnecessary scheduling from the scheduler as only the elevator that had a button click is allowed to go to that floor. The timing for the elevator both to move between floors and open the door are hard coded and divided by the value of speed in the settings file. The elevtsub request is the fastest computation as the scheduler only needs to update variables within itself. The floorSub request is the second shortest as it schedules the closest elevator without the requirement to update the states. The rescheduling will take the longest of the 3 scheduler timings due to it not only updating the state of the elevator to a fault state, it will also need to reschedule the floor, in the same way as it schedules a floor request.

In conclusion, the performance of the scheduler is good based on measurement results. It can process and respond quickly enough to schedule properly for Elevator Subsystem and Floor Subsystem.

Reflection

Overall we are very happy with the end result of the project. Our system (if running in GUI mode) can be run on 4 different computers, where a computer is running either, GUI.java, FloorSubsystem.java, ElevatorSubsystem.java, or Scheduler.java. Something we would have changed if we were to start over would be to integrate the test file mode to utilize command bridge class to create a UDP connection between the system and the test file. This would make it so the test file mode can be run on multiple computers similar to the GUI. This would also make it so the program natively supports the GUI rather than having flags to decide how the program will work. Another part we did well is the integration of a settings file. This file can be used to configure ports (in cases of local system already is using a port), as well as number of elevators, number of floors and the speed multiplier. We created a scheduler with minimal calculations to ensure the smallest scheduling time possible. This results in the timing of the scheduling to be small (0.2-0.5ms) which is negligible in comparison to the time for the elevator to move between floors (2.266s). Another design decision we made is to create a common class which holds the methods that are used by different subsystems, such as the encode, decode and print outs. This is used to clean up the code as well as keep the communication consistent throughout the systems. The common class printouts were also created to convert the byte array to English for easier reading for the user rather than printing out the actual values sent in the array. The GUI was also created with a console on the right-hand side of the frame which prints the GUI actions that were recorded. If more detail is wanted the console will also print out all commands as normal if GUI mode is chosen. We also chose to use RPC communication to ensure that the messages are received. This helps to ensure that packet losses do not happen. The final design decision we believe benefited the project is the creation of the external timer thread. This thread is used by the elevator to set the time between floors and to set the time between the door open and closed. The use of the thread means the elevator can continue receiving and sending while waiting to arrive at a floor meaning there will be no messages lost.