Project 2 – Concurrency Control Using Semaphores

Topic – The Elevator Simulator

*Simulation*

The simulation for the project consists of two runnable sections, one is the elevator runnable thread, and the other is the runnable code for all the elevator threads. The runnable for the elevator have the responsibilities to control the amount of people enter the elevator and limit the amount of people that enter the elevator, also it have to take the people to their floor and only come back to floor 1 if there are still people in the elevator to take to other floor or if there is people waiting at floor 1. Without these two conditions, the elevator will assume that there is no one waiting on floor 1 and there are no more people in the elevator and so simulation will end. Logically flow of the elevator is that it waits for the elevator to get down to floor 1. The elevator should be on floor 1 initially but after the first transport, it has to wait for the elevator to come back down to floor 1 to carry the 2nd batch of people. The elevator then signal door to open and signal the vacancy to how many number of people that can fit the elevator. Then it waits for takeoff signal from all of the person’s thread in the elevator. This take off signal should be the number of signal that the person thread sends. Elevator will not take off if the total number of people in the elevator has not sent the signal. Then elevator will close the door and take people to their floor. The elevator will check for the next floor based on the floor the person thread set and go to the next lowest floor close to floor 1. Then after take people to that floor, elevator closes the door and move to the next closest floor. After elevator takes all people to their floor, it goes back to floor 1 if there are still people want to use the elevator.

The runnable section for the person threads is simple. Each person thread will wait for a room available in the elevator and try to enter, if wait signal are full then that person thread will wait until there are room available. Then after enter the elevator, which the person thread will select a floor and signal elevator to take off. After the elevator take off, the person thread will wait for the elevator to open and get out at their floor and signal the elevator that person thread have gotten out of the elevator.

*Difficulties and Learning Experiences*

The difficult and probably the hardest part of this project are to control the people and signal the elevator when to have people wait for available room. Often the case with concurrency in multithreading is that when there is threads that wait on a resources, such as the elevator room available, once the elevator door open, every thread enter since the computer run so fast. However, correct implementation and usage of semaphore can eliminate this problem. This entire project only depending on the semaphores to signal and wait for one another to control the flow of person thread that enter the elevator and this is the core function of this project. The harder part in this specifically is getting the people out of the elevator because after the person thread enter the elevator, every thread is waiting to get out and if semaphore is not use correctly here, the output will not print out correctly. In the beginning of this project, I use the semaphore value as a way to control the elevator simulation however, after talking to the Professor about how to efficiently use the semaphore, I am able to utilize and use the semaphore wait and signal concept and was able to get the project working. Another thing I learned during this project is that using the semaphore permit value is a really bad and inefficient way to implement this project. The reason is because the computer run so fast, so at certain point in time we get the value for the semaphore permit, also at that time, the permit no longer exist since other thread have already claim it but the value of the permit will still display to the user however that information is not correct.

*Result*

By apply the technique wait and signaling to implement semaphores to control concurrency in this elevator simulation. The project was able to get all 49 people to ride the elevator in 7 batches. Each batch consist of only 7 people maximum and the elevator was successfully print out the correct output for the project like the one shown in the project example. Semaphore was implemented correctly and the project did not use waiting technique to force a thread to wait on other thread and this project did not use the permit information in the semaphore as based to control the elevator simulation.