Başak Amasya – 26628

CS307 – HW1 – Report

The aim of this homework was to practice multithreading and shared variable access by using busy waiting mechanism via simulating an airplane reservation system.

Program starts execution with main thread initiating each seat of the plane to 0 by using 2 simple nested for loops and then creating the child threads. Main thread then calls the join function for both threads and waits for the threads to finish their jobs. When they are done, meaning the plane is full, the final seating plan of the plane is displayed and program is terminated successfully.

For the implementation of child threads which work concurrently, different approaches can be considered. But in this homework, both threads work on the same function containing simple if-else statement for differentiating the threads via their ID's. Although, both threads process the same task, this distinction was done because of the nature of the busy waiting mechanism. This TravelAgency function first generates a random seat number, then the thread who has the turn checks whether that seat is empty or not. If it's empty, it does the reservation. If not, it does not do anything. In both cases, after the thread is done using the shared variable, which is the plane matrix, it gives the turn to the other thread and starts to busy wait until the other thread gives the turn back. This function is being executed until there are no more remaining seats. It should be noted that, only checking the seat and doing the reservation part is done inside the critical region since they require the usage of the shared variable, other parts are done outside the critical region to minimize the time spent within the critical region.

Since busy waiting is used and global "turn" variable is initiated as 0 at the beginning, first thread will always start first, then the threads will work in an alternating manner. Due to randomness of the generated numbers, the order of the seat reservations or how many seats one thread managed to reserve will change in each run. Also, the fact that a thread has entered and exited the critical region is being displayed, despite whether the reservation is done or not.

The code has been tested several times to see if a problem of overbooking or leaving a seat/seats empty occurred. Such problem was not encountered, the seating of the plane was correctly done in each run. Below, the compilation, beginning and termination of a sample run of the program can be found.

```
[bamasya@flow ~]$ g++ -o basakamasya_26628_hw1.out basakamasya_26628_hw1.cpp -lpthread
[bamasya@flow ~]$ ./basakamasya_26628_hw1.out
Agency 1 Entered Critical Region
Seat Number 74 is reserved by Agency 1
Agency 1 Exit Critical Region
```

```
Agency 1 Entered Critical Region
Seat Number 75 is reserved by Agency 1
Agency 1 Exit Critical Region

Agency 2 Entered Critical Region
Agency 2 Exit Critical Region
No Seats Left

Plane is full:
2 2 1 1 2 1 1 2 1 1 2 2 2 1 1 1 1 1 2 1 2 2 1 1 2 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 1
1 2 1 1 1 2 2 1 1 2 2 2 2 2 1 1 2 1 2 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 2 2 1 1 1 1 1 2 1 1
[bamasya@flow ~]$
```