CS307 - Fall 2020-2021
Airline Reservation System

Date Assigned: 01.11.2020
Due Date Time: 08.11.2020 at 23:55 (sharp, according to server's time)
No Late Submission

1 Introduction

For this homework, you will simulate a simple Airline Reservation System which will involve accessing shared resources. In real life, there are travel agencies who try to reserve seats for their customers. When a customer asks for a seat from a specific flight, the agency checks if there is an empty seat in the aircraft and reserves it immediately (if there is any). Only one reservations can be made on a seat, so there will be no overbooking. There will be one thread per agency simulating the activities of that agency. When an agency thread is booking a seat, the other agency will do busy waiting.

You will implement the following using **POSIX** threads.

2 Threads

2.1 The Main Reservation System (Main)

This is the main thread which initiates 2 other threads called TravelAgency1 and TravelAgency2 together with a 2-D array of fixed size representing the seats in a plane. Note that this 2-D array representing the seats is a shared data structure, so the travel agency threads can access and update it. Accessing a shared data structure may cause a race condition. If the flight is full, meaning that the shared 2-D array is fully marked, then travel agency threads should terminate and main thread should print the seats to the screen.

2.2 Travel Agency Thread (Child)

These two agencies will be the replica of each other since they are processing the same task. They will do the same routine until they are terminated. The agent thread should check the shared memory location, if it can access it, it should check if the flight is full or not, then reserve a seat and then leave the shared memory. If the reservation is done it should say "Seat Number X, is reserved by TravelAgencyTAI" where X is the seat number reserved (technically the index of the marked cells of the array in the shared memory) and TAI is the id of the TravelAgency, it can be 1 or 2 since we are going to have two agencies in total for the sake of simplicity. When an agent thread enters/exits the shared memory area, it should print messages to the console to trace.

3 Implementation Details and Pseudo Algorithm

You will first create a global *Matrix M* with size 2x50 and each of the cells of the matrix will represent a seat in the airplane.

3.1 Main Thread

All the cells will initially be marked as 0 by main thread.

1	2	3	4	...	47	48	49	50
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
51	52	53	54	...	97	98	99	100

Figure 1: Plane orientation.

Where M_{11} corresponds to *seat number 1*, M_{12} corresponds to *seat number 2* and M_{21} corresponds to *seat number 51* etc.

After you create your two threads, your main thread's job is to constantly check whether the *Matrix M* is full or not. It is full when all the cells in the matrix are non-zero. When Main thread sees that the *Matrix M* is full, it will print the *Matrix M* to the console.

3.2 Thread1 & Thread2

They will run concurrently, and their jobs are;

(a) Create a random seat number.

(b) Check if that seat is reserved or not.

- If the seat is full, do not do reservation.
- If the seat is empty, reserve it and mark it. (if thread 1 reserves the seat then thread 1 marks the seat as 1 and if thread 2 reserves the seat, thread 2 marks it as 2).

Since they will run concurrently we need some kind of synchronization mechanism and for that, we will use busy waiting.

3.3 Busy Waiting

Remainder of the busy waiting algorithm and the pseudo code

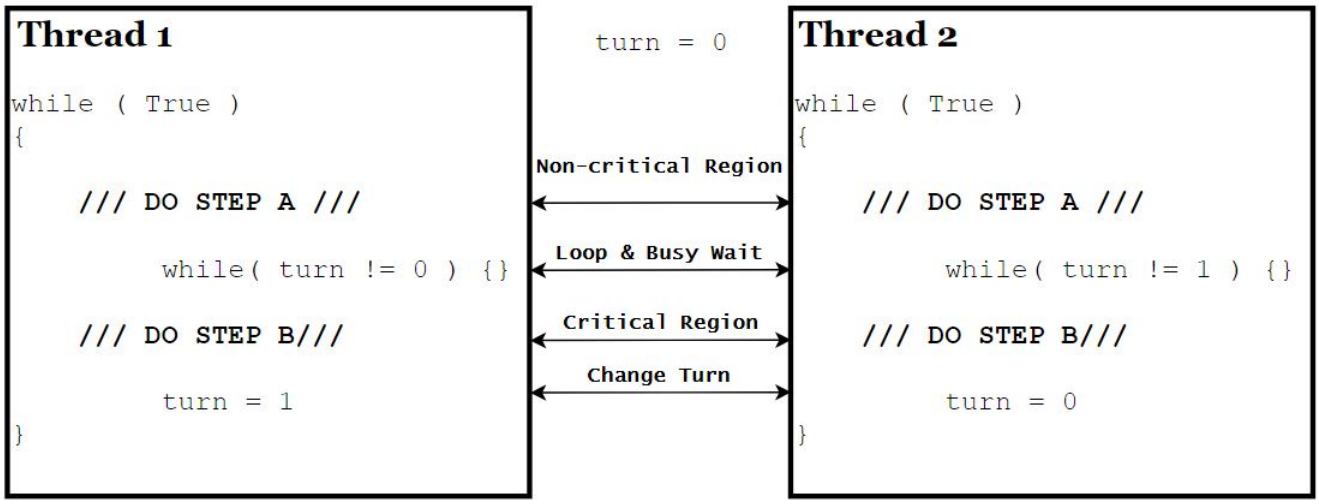

Figure 2: Busy Waiting Algorithm.

4 Useful Information & Tips

- Please refer to the POSIX thread examples in your recitation notes to see the detailed implementation of pthreads.
- If you are generating similar random numbers at each execution, you can add **srand (time(NULL))** to the beginning of the main. So your random generator will generate different seeds at each execution, thus you will have different random numbers.
- Since the matrix is a shared variable, constantly checking the matrix in a loop is not good idea. Use remaining number of seats variable to track the capacity.
- Your program should terminate properly, you will lose points if your program can not exit infinite loop or missing outputs.
- Your program will be tested and graded on the flow server. Please test it before your submission.
- You have to use pthreads (POSIX Threads) library and submit a cpp file, any other thread library will not be accepted as a solution.
- You will lose points if a seat is overbooked, which means each seat should be booked only once.
- Using *printf* statement instead of *cout* statement may help for shuffled output orders.
- Do not forget to use *-lpthread*, command while compiling.

Sample Output

Due to random nature of threads and random generator, your output could not exactly be same but the output format should be similar with the below example.

```
Agency 1 Entered Critical Region
Seat Number 10 is reserved by Agency 1
Agency 1 Exit Critical Region

Agency 2 Entered Critical Region
Seat Number 1 is reserved by Agency 2
Agency 2 Exit Critical Region

Agency 1 Entered Critical Region
Seat Number 12 is reserved by Agency 1
Agency 1 Exit Critical Region

.
.
.
.
.

Agency 1 Entered Critical Region
Seat Number 96 is reserved by Agency 1
Agency 1 Exit Critical Region

Agency 2 Entered Critical Region
Seat Number 48 is reserved by Agency 2
Agency 2 Exit Critical Region


Agency 1 Entered Critical Region
Agency 1 Exit Critical Region
No Seats Left

Plane is full:

2 2 1 2 1 2 2 1 2 1 2 1 2 1 1 1 1 2 1 2 1 1 2 2 1 2 1 2 2 2 2 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1 1 2 1 1
2 2 1 2 2 1 2 1 1 1 1 2 1 2 1 1 1 2 1 1 2 1 1 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 1 2 1 1 1
```

Submission

Please include a report that describes the reasoning and flow of your program. Please briefly explain how your code is working in your report. Your report should be at **MOST** one page long and your report should be in **PDF** format.

Your **CPP** file should be submitted in a **ZIP** archive, name your zip archive as: ***YourNameSurname_ID_hw1.zip*** and submit to **SUcourse**.

You need to add comments to the your code to make it more understandable. You will lose points if there are problems in your submission, so please write your code on a **CPP** file. After that create a **ZIP** file that contains your report and **CPP** file.

Note that, your system time and SUCourse server's time may not be synchronized so do not wait the last minutes to submit your solution. Only the solutions in the SUCourse system will be graded. Other submissions, such as emailing to instructor or assistants, will not be graded.

Good Luck,
Berkant