CS 307 – Operating Systems

Fall 2019

Homework 1

Date Assigned: 7.10.2019
Due Date Time: 15.10.2019 at 23:55 (sharp, according to server's time)

Airline Reservation System

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). A customer can reserve more than one seat if needed. Since there are a lot of travel agencies out there, the reservations will be done based on FCFS (first come, first served) strategy. 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 rest will do busy waiting.

You will implement the following using POSIX threads:

- **<u>The Main Reservation System:</u>** This is the main thread which initiates 3 other threads called `TravelAgency1`, `TravelAgency2` and `TravelAgency3` together with a 2-D array of fixed size representing the seats in a plane. It will then let the three travel agency threads access the aircraft seats and do a reservation if possible. 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 main thread terminates the execution of travel agency threads and print the seats to the screen.
- **<u>Travel Agency Thread:</u>** These three 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 any seats and then leave the shared memory. If the reservation is done it should say "Seat Number X, Seat Number Y … are reserved by TravelAgencyTAI" where X, Y, .. is the seat numbers reserved (technically the indexes 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 or 3 since we are going to have three 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.

<u>Implementation Details and Pseudo Algorithm</u>.

<u>Main thread:</u>
You will first create a Matrix M with size 2x50 and each of the cells of the matrix will represent a seat in the airplane. All the cells will initially be marked as 0.

Seat plan is done according to this convention:

1	3	5	7	9	11	...	...	99
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
2	4	6	8	10	12	...	...	100

Where M_{11} corresponds to seat number 1, M_{21} corresponds to seat number 2 and M_{12} corresponds to seat number 3 etc.

After you create your three 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 is full, it will first finish the execution of the other 3 threads and then it will print the Matrix to the console.

Thread 1 & Thread 2 & Thread 3:

They will run concurrently, and their jobs are;

a) Create a random seat number and check if that seat is reserved or not.
b) If it is reserved create another random seat number until you find an empty seat.
c) Reserve it and mark it as 1 or 2 or 3 according to the reserver thread number (if thread 1 reserves it then it marks it as 1; if thread 2 reserves it, it marks it as 2 and if thread 3 reserves it, it marks it as 3).

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

Remainder of the busy waiting algorithm and the pseudo code:

T1	T2	T3

`while(true) {`
`  while(turn!=0);`
`  //Critical region`
`  //Do steps a, b and c`
`}`
	`while(true) {`
	`  while(turn!=1);`
	`  //Critical region`
	`  //Do steps a, b and c`
	`}`
		`while(true) {`
		`  while(turn!=2);`
		`  //Critical region`
		`  //Do steps a, b and c`
		`}`

Please refer to the POSIX thread examples in your recitation notes to see the detailed implementation of pthreads.

Submission Guides:

Solutions should be submitted in a zip achieve, name your zip achieve as: ***YourNameSurname_ID_hw1.zip*** and submit to **SUcourse**. 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.