CMPE322

OPERATING SYSTEMS

PROJECT#2

FLIGHT RESERVATION SYSTEM SIMULATOR

HASAN BASRİ BALABAN

2016400297

PROJECT REPORT

The program contains four global arrays: One of them (*seat_array*) is used for keeping the current status of the seats and the others (*requested_seat_array, is_reserved_by_server, is_requested_by_client*) are used for communication between client and server threads. Also, there is a mutex array called *mymtx*. This array holds 101 mutexes which are used for safe access to each element of *seat_array*. (According to the project description, there will be at most 100 empty seats.)

In main function, after the global arrays and mutexes initialized client threads are created by using a for loop. Then, another loop is used to join on each of the client threads. And finally, the memory which is dynamically allocated is deleted for avoiding memory leaks.

In order to create and make the threads functional, two additional functions are used beside the main function. First function is *client_runner*. This function is used for client threads, and it runs in the given order:

-> Create the server thread

-> Sleep for a random time between 50-200 ms

-> Select a random seat number

    -> Lock the corresponding mutex in mutex array

    -> Send a request to server to check if the seat is empty or not

    -> Wait until server response

    -> Unlock the mutex after server response

-> If the selected seat is reserved

    -> Join on the server thread and exit the thread

-> If the selected seat is not reserved, repeat the same process until a seat is reserved

And the second function is *server_runner*. This function is used for server threads, and it runs in the given order:

-> Wait until the client requests for a seat

-> Check for the availability of the requested seat

    -> If the requested seat is empty, then mark it as not empty and serve it to client thread

    -> If the requested seat is not empty, wait for another request

While implementing this project, I initially use only one mutex which is locked by server thread while checking if a seat is empty or not and unlocked after this check. But I realize that if I lock the entire seat array while only a single thread checks for a seat, then the functionality will be restricted. So, I used a mutex array (one mutex for every seat) to make the program more functional and faster in order to reduce the busy-waitings during the execution.

Mutexes are locked when the client threads generate a random seat number. In this way, if a client thread selects a seat and is preempted before it requests the seat, the mutex of the seat will be locked and no other clients will be able to reserve that seat.

HOW TO RUN

The submitted zip file contains a Makefile. This makefile contains only one command line:

g++ main_101mutex_functional.cpp -o program -pthread -std=c++11

The program can be compiled by typing *make* command in command line. There will be an executable file in the current directory, named as *program.*

As an alternative the program can be compiled from command line directly by typing:

g++ main_101mutex_functional.cpp -o [program name] -pthread –std=c++11

This command will create an executable file called *[program name].*

The executable file will be executed by the command:

./[program name] [a number between 50-100]

This execution will create a txt file called *output.txt* .