

CS 4323 Design and Implementation of Operating Systems I

Assignment 03: Full Marks 200

(Due Date and End date: 04/30/2021, 11:59 PM CST)

This is the final project assignment that must be done in the group as specified using C programming language. If you do not know the group members, then please refer the module **FinalGroupProject** under **Home** page in Canvas.

This project will familiarize you with synchronization and scheduling for multi-threaded server environment using TCP Unix socket.

You are required to write a multithreaded server capable of receiving messages (or request) from the client, process the request and send the result back to the client.

You are encouraged to make reasonable assumptions in the project. All such assumptions should be clearly specified in the report.

Project Overview:

The project focuses on a complete management of a train reservation/booking system. Customers (or a client) can make reservations/modifications/cancellation online through multiple servers. Let us use a variable N to define the number of servers. The servers should be able to handle many customers simultaneously at any time and each server should be multithreaded. Each thread can serve only one customer and the number of threads in the server defines the number of customer requests that the server can handle in parallel. The conceptual picture would be similar as shown below:

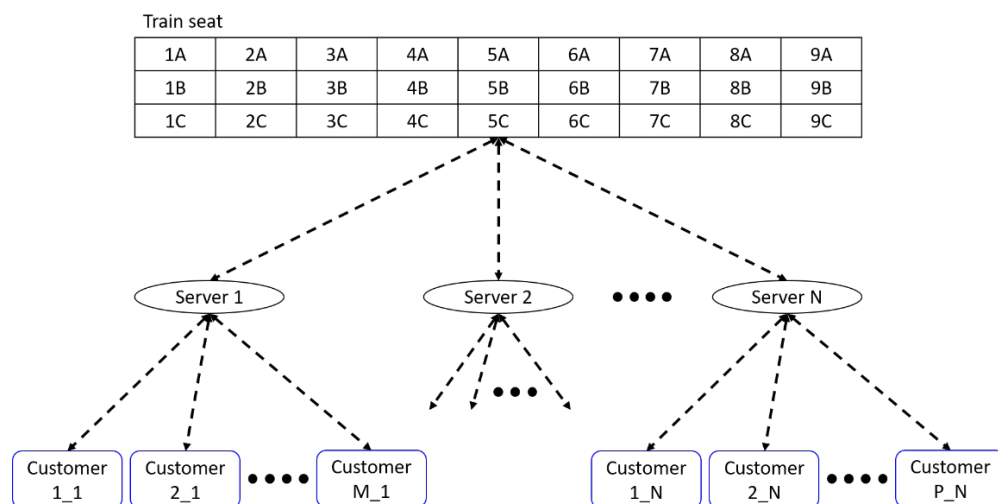


Fig (1). Model for Train reservation System

Each customer will connect to one server at any time to complete any task (reservations/ modifications/ cancellation) at any time. When a connection is established between the customers and the servers, the server will greet the customer and display the menu to the client. The menu should include the following option:

1. Make a reservation.
2. Inquiry about the ticket.
3. Modify the reservation.
4. Cancel the reservation.
5. Exit the program.

The customer is then asked to choose the option, based on the customer requirement. After the completion of requested operation, the program should display the menu, as long as the customer does not exit the program. Once the customer chooses option (5), the thread returns to the pool.

Tasks required to perform in each of the options are discussed below:

Make a reservation

You will need to consider the reservation is allowed for only next 2 days and you can consider:

- there is only a single train on each day,
- all customers are travelling from the same place to the same destination,
- fixed number of seats in the train,

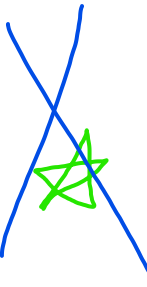
If a customer chooses to make a reservation, then the server needs get the following information:

- Customer's full name
- Date of birth
- Gender
- Government ID number
- Date of travel
- Number of travelers

A customer can make multiple reservation at once for everyone travelling together. Once the customer enters the above information and the customer's request can be granted, then only the program will prompt for confirmation message:

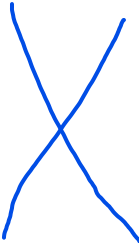
Do you want to make reservation (yes/no):

If the customer enters "yes", then it should be treated as equivalent to purchasing ticket. This means the customer must get the seat. All the customers will be then asked to choose the seats. For that you need to clearly display only the available seats in the train for the given date.



The server then needs to make a receipt of the booking (include the ticket number and the server ID number) and need to send the file to the customer. Issue one ticket number for one receipt. A single receipt will be issued to a customer, even if a customer has booked a ticket for all the accompanying travelers. The server then needs to maintain a single file to keep record of each customer's information for all the reservation, which will be used for other options. We will call this file as a **summary** file. The **summary** file needs to include all customer's information, ticket number, server ID used to make a reservation and a note to include if the reservation is later modified (discussed below).


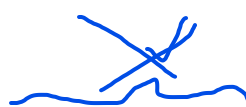
Inquiry about the ticket



A customer can make an inquiry about the ticket purchased, based on the ticket number. During the inquiry the customer cannot make any changes. Customer can just read the ticket. You need to display all the information made for the ticket number.

A customer can connect to any server to make the inquiry. For example, in the Fig (1), if the customer made a reservation from server 1, he/she is able to make inquiry from server N.

Modify the reservation



A customer can make a change to the seat, a travel date or even the number of travelers. As before, a customer can connect to any server to modify the reservation. After the completion of this task, the changes should be included in the **summary** file. In the summery file, you need to clearly show the server ID which is used to modify the reservation and what modification has been made.

Cancel the reservation

A customer can also make a cancellation of the reservations made, based on the ticket number. As before, a customer can connect to any server to cancel the reservation. After the completion of this task, the changes should be included in the **summary** file. The respective ticket number information should be removed from the **summary** file.

Project Specification:

You can choose the appropriate value for **N** (i.e. the number of servers). But these values should be more than 2. You can choose appropriate values for the number of customers and number of total seats. The main objective of the assignment is to handle proper synchronization. So, you need to make sure the boundary conditions are met. This means that you cannot arbitrarily chose unlimited seats or very few customers.

Managing concurrency:

You need to design the policy for managing concurrency in each sever and this policy will be same across all the servers. Each of the server needs to maintain a thread pool. Each server will have a specific number of threads in the pool (to handle concurrent requests) in advance and this number will not change. Let us define the number of threads in the pool as **THREAD_NUMBER** and you need to hardcore this value in the program. Each customer's request will be handled using the thread from this pool. When the server receives the customer's request, it will be served by one of the thread from the pool and when the customer's request is done, the thread will be returned in the pool. If all the threads are occupied (or currently busy), then the customer needs to wait for the thread to be available. When the there is no customer, the threads remain inactive in the pool.

Managing synchronization:

You need to use semaphore to provide all necessary synchronization for the project. You need to identify where synchronization is required.

In addition to that you need to explicitly allow only one server and one customer to access the **summary** file at a time for write purpose only. For read purpose, you can allow multiple servers/customers to access the **summary** file.



When making the reservation of the seat, you need to give priority to the customer who have larger number of accompanying travelers. For example, in the Fig (1), if customers 1_1 and M_1 are making reservation through server 1, and customer 1_1 has 3 accompanying travelers while customer M_1 has 5 accompanying travelers, then the server need to give first chance to customer M_1. However, you need to make sure that customer M_1 do not starve or is blocked from getting the seat.

Points distribution for this project will be as follows:

- Creation of multithreaded servers and successful interaction with clients **[20 points]**
 - Concurrency management is considered a part of multi-threader servers
- Implementation of each menu:
 - Make a reservation **[12 points]** ✓
 - Inquiry about the ticket. **[8 points]** ✓
 - Modify the reservation **[12 points]** ✓
 - Cancel the reservation **[8 points]** ✓**[40 points]**
- Implementation of semaphore to provide synchronization among servers. **[60 points]**
- Implementation of semaphore to provide synchronization among customers within a server. **[40 points]** ✓



- Implementation of priority for seat reservation and avoidance of starvation [20 points]
- Final report and group presentation [10 + 10 points]

Grading Criteria:

This is a group project. So, grading will focus on students' ability to work in the group and successfully complete the work. Each member in the group should coordinate as a team rather than individual. And that's the key to score maximum points.

Points to remember:

- Failure to complete the project as a group will deduct 20 point, even though every individual has completed their part. As this is a group project, you need to learn to work in the group and meet the team's objective, and not individual objective.
- Work must be distributed uniformly among all group member and should be clearly specified in the report.
 - Be sure to submit a progress report that summarizes your design and all the test cases you have performed.
 - The progress report needs to include the code and the output.
 - Each student will write the code in their own file. No two students will share their code in a single file.
 - Students can have multiple .c files. Each .c file should have a header information.
 - Include each student's contribution clearly. Failure to be specific on this part will make us difficult to give fair grades. So, it is your responsibility to clearly indicate what specific task each member is undertaking in the project.
- It is not necessary that all group members will get equal points. It depends on what responsibility each student has taken, whether or not the student has delivered the task.
- It is the group's responsibility to alert instructor with any issue that is happening in the group.
 - Students are supposed to use the group created in the Canvas for all correspondence. That will serve as the proof, in case there is any dispute among the group members.
 - Time is very important. Group members are expected to respond quickly.
 - You need to have sufficient days to combine individual work. You are going to make one submission as a group and not individually.

Submission Guidelines:

- Final Progress report need to be submitted by the progress due date/end date (04/19/2021, 11:59 PM CDT). The due date and the end date are same for the progress report. During the progress report, each group need to submit:
 - Please refer to the progress report file for further instruction.
 - Make sure you list any assumption you have made in the report.
- Group presentation needs to be completed by 05/07/2021, 5:00 PM CDT.
 - All the group members should be present for the final presentation.
 - Each group should book an appointment with the TA beforehand. It is the group's responsibility to schedule the appropriate date and time with the TA.
 - Make sure that the group do not wait for the last moment to book an appointment for the presentation.
- Final project submission instruction:
 - Include a single readMe.txt file that shows how to run your program.
 - Each student's work needs to be in a separate .c file.
 - You can have .h file.
 - Client side and server side program would have a separate driver .c file.
 - Each .c file should include the header information, which should include:
 - Group Number
 - Group member name
 - Email
 - There should be sufficient comments in the program.
 - Each function should be clearly described along with the input arguments and return values.
 - All the codes should be submitted in both .pdf format (copy paste in textual form, no screenshot) as well as in .c files (including any .h file).
- Final project will be evaluated together with the progress report. So, they should have a correlation. Commitments on the progress report should be reflected on the final submissions.