Programming Assignment #2

Due: 4th Dec. (Sunday),11:59 PM

1. Introduction

In this assignment, you will make a ticketing server which works under Linux environment. The main goal of this assignment is to learn how to write a program that manages its resource concurrently and consistently.

2. Problem specification

You have to implement a ticketing server which is capable of handling numerous clients simultaneously. The server should manage the ticketing procedure and seat position of each client. A client sends queries to the server to get a ticket for a desired seat. A single query represents an action that client can take. The details of server and query will be explained in this section.

2.1. Server

A server takes queries from clients in an infinite loop with the following condition:

- The total number of seats to be managed by server is 256.
- Each seat must be managed with at least one synchronization mechanism.
- The server must be able to handle up to 1024 clients concurrently.
- A dedicated thread must be created for a client if the connection successfully established with `accept`. The created thread should handle all the queries for that client.
- The server manages log-in status of each "user". Note that the term "user" is different from the term "client". The term "client" represents program or process used to communicate with the server, and a query for a certain "user" can be sent to the server with this client program.
- A user is identified with a single integer (user id).
- If user 1 is currently logged in through client A, every other attempt to log-in as user 1 from the other client must be blocked by the server. And also, the client A cannot send queries of another user before the user 1 successfully log out.
- The server should check if the user is logged in when the user attempts to reserve a seat.
- The server sends a response code (single integer) to the client after receiving a query. Please refer to the section 2.3 about the details of processing queries.
- Please refer to section 2.4 about termination condition of the server.

2.2. Client query

A single query sent from client to server can be represented by following structure:

```
struct query {
	int user;
	int action;
	int data;
};
```

- `user` field represents user id, ranging from 0 to 1023.
- `action` field represents action id, ranging from 1 to 5 (0 to terminate).
- `data` field contains data needed by each action.
- If the `user` of `action` field is out of range, server returns -1.

2.3. Action in query

There are 5 kinds of actions that client can take through a single query:

Action id	Name	Description	`data` field
1	Log in	Try to log in	Passcode
2	Reserve	Try to reserve a seat	Seat number
3	Check reservation	Get the reserved seat number of a user	N/A
4	Cancel reservation	Cancel reservation of a user	Seat number
5	Log out	Log out	N/A

2.3.1. Log in

- If the user trying to log in for the first time, the action will be treated as "registration". Then the server will initialize the passcode for the user with the `data` field in query. The passcode is a 4-byte integer.
- If already registered user tries to log in with the wrong passcode, the action will fail.
- If the user trying to log in is currently logged in at the other client, the action will fail.
- The action will succeed otherwise.
- The server returns 1 on success, -1 on fail.

2.3.2. Reserve

- Server makes a reservation for a user
- The seat number in `data` field ranges from 0 to 255. The action will fail if the seat number is out of range.

- If the user tries to take this action before logging in, the action will fail.
- If the seat requested by the user is already reserved by the other user, the action will fail.
- The action will succeed otherwise.
- The server returns reserved seat number on success, -1 on fail.

2.3.3. Check reservation

- If the user tries to take this action before logging in, the action will fail.
- If the user did not reserve any seat, the action will fail.
- The action will succeed otherwise.
- The server returns the reserved seat number on success, -1 on fail.

2.3.4. Cancel reservation

- If the user tries to take this action before logging in, the action will fail.
- If the user did not reserve any seat, the action will fail.
- The action will succeed otherwise.
- The server returns the cancelled seat number on success, -1 on fail.

2.3.5. Log out

- If the user tries to take this action before logging in, the action will fail.
- The action will succeed otherwise.
- The server returns 1 on success, -1 on fail.

2.3.6. Response code summary

The summary of actions and corresponding response codes are tabulated below:

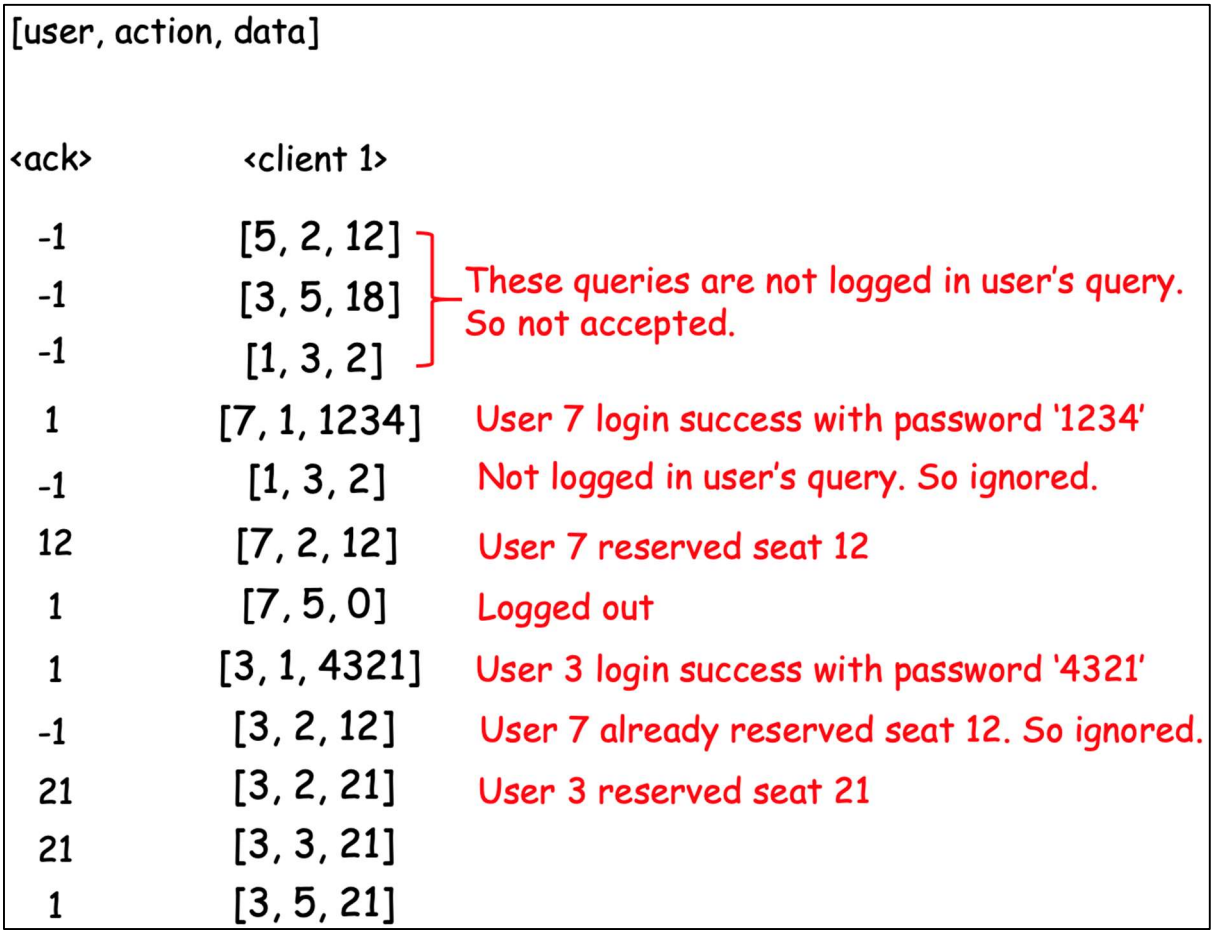
Action id	Name	On success	On fail
1	Log in	1	-1
2	Reserve	Reserved seat number: [0, 255]	-1
3	Check reservation	Reserved seat number: [0, 255]	-1
4	Cancel reservation	Cancelled seat number: [0, 255]	-1
5	Log out	1	-1

2.4. Termination condition

- This termination query is sent when all the reservations are done. (No log-in required)
- When the server receives the query in which all the fields (`action, user, data`) are zero, server must return the whole integer seat array.
- The size of the array must be 256 and the value of the array element must be the owner of the corresponding seat. For example, if user 3 successfully reserved seat 14, the value of 14th element in the array should be 3.
- If the seat is not reserved any user, the value of the element must be -1.

3. Client/server examples

The two figures below show the example scenario of client query and desired server response.



<client 1>	<client 2>	<client 3>
[5, 1, 1]		
	[5, 1, 1]	Ignored because user 5 already logged in at client 1
	[3, 1, 7]	
[5, 2, 10]		
	[3, 2, 10]	Ignored because user 5 already reserved seat 10
[5, 4, 10]		
	[3, 2, 10]	
[5, 5, 1]		
	[3, 5, 1]	
	[5, 1, 11]	Ignored because user 5's password is 11
	[5, 1, 1]	
	[5, 3, 1]	Ack will be -1, because user 5 didn't reserve the seat

4. Restrictions & hand-in instruction

- You have to do your assignment in **Linux environment.**
- If a resource is dynamically allocated, it must be freed before the program terminates. Resources refer to files, memory, and child processes.
- You have to submit assignment in the icampus
- You have to submit **2 files**. `Makefile` and `.c` source code file of server.
- You may submit report.docx (or .pdf) for the report. The report score policy is the same as the assignment 2: not mandatory, bonus credit 5 which does not exceed full credit …
- Your executable file name must be "`pa2.out`".
- If you don't follow submission format or we can't compile your source code with "`make`" command, you will get penalty.
- We will grade the only most recently submitted file.
- The deadline is **December 4th, 11:59 P.M. (12월 4일 오후 11시 59분)**

5. Precautions

- You can discuss the task together, but you must write the source code by yourself.
- You'll get 15% reduced point per day, up to 45% if you cannot meet deadline. You'll get 0 point after 3 days.
- If you copy someone else's assignment, **you both will get 0 points**, even if you copy the source code you **found on the Internet**. These are the minimum penalties. Therefore, there may be additional penalties other than assignment's 0 point.