[SWE2024: System Programming Lab - Spring 2023]

Programming Assignment #3

Due: 11th Jun. (Sunday), 23:59:59

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 must 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 conditions:

- The total number of seats managed by the 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 synchronization variable must be used for only one seat.
- 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 a program, or a 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. Also, the client A cannot send queries of another user before the user 1 successfully logs 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

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 의 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` or `action` field is out of range, server returns -1.
- Client can be run in two different ways:
    - When receiving query file as command line input
        - Query file contains more than one query.
        - When client is done handling all the queries in the Query file, it sends a query that can terminate the program.
    - Without query file.
        - Client runs on an infinite loop until the termination condition is met.
- The only available format of a query is "number<space>number<space>number"

```
/* Query File Example */
$ cat query.txt
5 2 12
3 5 18
```

```
/* Client Example with Query File, check with 2.3.7 */
$ ./client ip_addr port query.txt
Reservation failed
Logout failed
```

```
/* Client Example without Query File, check with 2.3.7 */
$ ./client ip_addr port
5 2 12
Reservation failed
3 5 18
Logout failed
```

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 a 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.
- Passcodes are positive integer and does not exceed the INT_MAX.
- If an 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 from another client, the action will fail.
- If a user tries to log in again after a successful log in, the action will fail.
- The action will succeed otherwise.
- Server returns 1 on success, -1 on fail.

2.3.2. Reserve

- Server makes reservation for users.
- 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.
- Server returns the seat number on success, -1 on fail.

2.3.3. Check reservation

- The data field in Log out query is ignored.

- 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.
- Server returns the array of seats on success and returns -1 on fail.
- Returned array contains 1 if the corresponding seat is reserved by the user, otherwise 0.
- The type of array is up to you.
- Client must use the returned array to print all the reserved seats in ascending order.

2.3.4. Cancel reservation

- 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 user did not reserve any seat, the action will fail.
- Actions to empty seats will fail.
- Actions to the seats reserved by other users will fail.
- The action will succeed otherwise.
- Server returns the seat number on success, -1 on fail.

2.3.5. Log out

- The data field in Log out query is ignored.
- 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(s): [0, 255]	-1
4	Cancel reservation	Cancelled seat number: [0-255]	-1
5	Log out	1	-1

2.3.7. Print format

The summary of actions and corresponding terminal outputs are tabulated below:

Action ID	**Name**	**On success**	**On fail**
1	Log in	Login success	Login failed
2	Reserve	Seat [0, 255] is reserved	Reservation failed
3	Check reservation	Reservation: [0, 255], …	Reservation check failed
4	Cancel reservation	Seat [0, 255] is canceled	Cancel failed
5	Log out	Logout success	Logout failed

2.4. Termination condition

- In case of receiving query file in command line, after all the actions in the query file are executed, the client sends the termination query to the server automatically.
- Otherwise, the client receives the termination query through input and sends the termination query.
- When the server receives the query in which all the fields (`action, user, data`) are zero, server must return an integer value of 256.
- After sending an integer value of 256, the Server terminates the connection with the client. If there are users that are logged in, they must be logged out by the server.
- After the Client receives 256 from the server, it terminates.

3. Examples of client/server

The two figures below are example scenarios of client query and desired server response.

	<Client> [user, action, data]	<Terminal>
These are performed before login. Therefore, not accepted.	5 2 12	Reservation failed
	3 5 18	Logout failed
	1 3 2	Reservation check failed
User 7 login success with password '1234'	7 1 1234	Login success
Not logged in user's query.	1 3 2	Reservation check failed
User 7 reserved seat 12	7 2 12	Seat 12 is reserved
Logged out	7 5 0	Logout success
User 3 login success with password '4321'	3 1 4321	Login success
User 7 already reserved seat 12	3 2 12	Reservation failed
User 3 reserved seat 21	3 2 21	Seat 21 is reserved
User 3 reserved seat 22	3 2 22	Seat 22 is reserved
User 3 check reservation	3 3 0	Reservation: 21, 22
Logged out	3 5 21	Logout success
Invalid password	7 1 4321	Login failed
Invalid query	1 2	Invalid query

<Client 1>	<Client 2>	
5 1 1		
	5 1 1	Failed because user 5 already logged in at Client 1
	3 1 7	
5 2 10		
	3 2 10	Failed because user 5 already reserved seat 10
5 4 10		
	3 2 10	
5 5 1		
	3 5 1	
	5 1 11	Failed because user 5's password is 1

4. Restrictions

- This assignment must be done in Linux environment.
- Setting up IP and port are the same as we learned in socket, concurrent programming lab.
- If a resource is dynamically allocated, it must be freed before the program terminates. Resources refer to files, memory, and child processes.
- You must submit assignment in the **ji cluster server with "`submit`" command**.
- `$ ~swe2024-41_23s/bin/submit pa3 [your working directory]`
- You only have to submit **3 files**. `Makefile` and two `.c` source code files.
- Your executable file name must be "`pa3_server`" and "`pa3_client`".
- If you don't follow the submission format or failure of compilation with `make` command, you will get penalty.
- We will grade the only most recently submitted file.
- Skeleton codes are available.
- `$ cp ~swe2024-41_23s/2023s/pa3_skeleton.tar.gz [your working directory]`
- `$ tar xvf pa3_skeleton.tar.gz`
- **NO LATE SUBMISSIONS ARE ALLOWED FOR PA3.**
- **DUE 6/11 23:59 (HARD DEADLINE)**