

System Programming 2022: csieBooking

tags: NTU-SP

1. Problem Description

Due to the coronavirus, people are asked to keep the social distance or even stay at home. This situation strikes the business of stores. Fortunately, as vaccines are maturely developed, we have successfully survived the epidemic and entered the post-epidemic era. In this post-epidemic era, economies are recovering. To help stores resume business, you are expected to implement a simplified multi-service booking system: **csieBooking**.

The csieBooking system is comprised of **read servers** and **write servers**. Both can access a file **bookingRecord** that records information about users' bookings. When a server gets a request from a client, it will respond according to the file's content. A read server tells the client how many items it has booked. A write server can modify the file to book more or less items.

However, the users of csieBooking system are somehow impatient and selfish. A user may grab a coffee after his/her connection is accepted without making request to server, which blocks all subsequent users. Moreover, users cherish their time and are unwilling to wait. Thus, it's your responsibility to equip the csieBooking system with the ability to **serve multiple users simultaneously**.

You are expected to complete the following tasks:

1. Implement the servers. The provided sample source code `server.c` can be compiled as simple read/write servers so you don't have to code them from scratch (but feel free to edit any part of codes). Details will be described in the later part.
2. Modify the code in order that the servers will not be blocked by any single request, but can deal with many requests simultaneously.
(💡 You might want to use `select()` or `poll()` to implement the multiplexing system. However, remember not to do anything that will result in busy waiting.)
3. Guarantee the correctness of file content when it is being modified. You might want to use file lock to protect the file.
(💡 You are supposed to be aware that file lock is a process-oriented tool and a server is exactly a process.)

2. Running the Sample Servers

The provided sample code can be compiled as sample servers.

The sample server has handled the part of socket programming, so you are able to connect to the sample server once you run it.

Feel free to modify any part of the code as you need, or implement your own server from scratch.

Compile

You should write your own `Makefile` to compile your code. You can use the provided command to compile sample source code:

```
$ gcc server.c -D WRITE_SERVER -o write_server
$ gcc server.c -D READ_SERVER -o read_server
```

Your `Makefile` may contain commands above to generate `write_server` and `read_server`.

Also, your `Makefile` should be able to perform cleanup after the execution correctly (i.e, delete `read_server` and `write_server`).

Run

After you compile the code, you can run the server with following command:

```
$ ./write_server {port}
$ ./read_server {port}
```

For example, `./write_server 7777` runs a write server listening on port 7777, and `./read_server 8888` runs a read server listening on port 8888.

Note that port 0~1023 are reserved for common TCP/IP applications, so you should not pick a number within this range.

Port in use

► [Click me](#)

3. Testing your Servers at Client Side

You can use the command `telnet {hostname} {port}` to connect to a running server.

For example, if you run `./read_server 7777` on CSIE linux1 workstation, you can interact with this read server with `telnet linux1.csie.ntu.edu.tw 7777`.

Similarly, if you run `./write_server 10000` on CSIE linux3 workstation, you can interact with this write server with `telnet linux3.csie.ntu.edu.tw 10000`.

If you are running sample servers, try to type something on the client side. You will see some messages from servers.

(TAs will test your code on `linux1` . Make sure your code works as you expect on it.)

4. About the Booking Record File

The servers will access the file `bookingRecord` . The file contains 20 user's information made up with a **user id** (ranged from **902001** to **902020** only) and **user's booking state**. Booking state will be an array, `bookingState` , with 3 integer values. These integer values represent the amount of each item booked by the user. In our case, the correspondence between the items and the indexes will be:


- Food = 0
- Concert = 1
- Electronics = 2

You may need following definitions to use these indexes more flexibly:

```
#define OBJ_NUM 3

#define FOOD_INDEX 0
#define CONCERT_INDEX 1
#define ELECS_INDEX 2
#define RECORD_PATH "../bookingRecord"

static char* obj_names[OBJ_NUM] = {"Food", "Concert", "Electronics"};
```

System Programming 2022: csieBooking  HackMD (https://hackmd.io?utm_source=view-page&utm_medium=logo-nav)

Following is the structure of a record stored in `bookingRecord` :

```
typedef struct {
    int id;          // 902001-902020
    int bookingState[OBJ_NUM]; // amount of booked objects
} record;
```

For those who are not familiar with c language, you might want to check this [website](https://www.geeksforgeeks.org/readwrite-structure-file-c/) (<https://www.geeksforgeeks.org/readwrite-structure-file-c/>).

Note that there is a `bookingRecord` in your repository. You can use this `bookingRecord` to test your code by yourself. However, when judging your homework, TAs will use another `bookingRecord` , so make sure your code does not depend on a fixed `bookingRecord` .

5. Sample input and output

5.1 Read Server

▼ Click me

A client can read an user's booking state. Once it connects to a read server, the terminal will show the following:

Please enter your id (to check your booking state):

If the client types an valid id (for example, 902001) on the client side, the server shall reply:

```
Food: 1 booked
Concert: 3 booked
Electronics: 0 booked
```

(Type Exit to leave...)

Now, if the client types `Exit` , server should close the connection from the client (If typing other input, ignore it (<https://github.com/NTU-SP/SP-Hw1-release/issues/3#issue-1391876971>)). Note that you should keep the booking state of this id **unwritable** by others until server receive `Exit` from the client.

Booking state of the same id could be **read** simultaneously. However, if someone else is **writing** the booking state of the same id, the server shall reply:

```
Locked.
```

and close the connection from client.

(Note that there always exists a newline character in the end of output.)

5.2 Write Server

▼ Click me

A client can edit an user's booking states. It will first show the user's previous booking state just like a read server, then ask for a booking command.

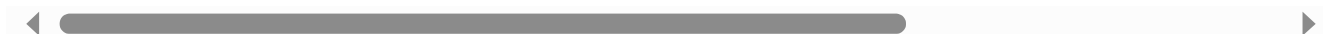
Once the client connects to a write server, the terminal will show the following

Please enter your id (to check your booking state):

If you type an valid id (for example, 902001) on the client side, the server shall reply the booking state of the id, following by a prompt:

```
Food: 1 booked
Concert: 3 booked
Electronics: 0 booked
```

Please input your booking command. (Food, Concert, Electronics. Positive/negative value in



A valid booking command consists of 3 integers, separated by a single space. If the client types a command (for example, 3 -2 4), the server shall modify the `bookingRecord` file and reply:

```
Bookings for user 902001 are updated, the new booking state is:
Food: 4 booked
Concert: 1 booked
Electronics: 4 booked
```

And close the connection from client. Note that you should keep the booking state of this id **unwritable** & **unreadable** by others once the client determines the id until closing the connection from client.

If someone else is **reading/writing** the booking state of the same id, the server shall reply:

```
Locked.
```

and close the connection from client.

(Note that there always exists a newline character in the end of output.)

5.3 Bounds of the booking amount

► [Click me](#)

5.4 Input checking

► [Click me](#)

6. Grading

▼ Warning

- Please strictly follow the output format above, or you will lose the point of each task, respectively.
- Make sure your `Makefile` compile files correctly on linux1, or your score will be **0**.
- Make sure your `Makefile` can clean unnecessary files with `make clean`, or you will lose **0.25 point**.
- You should not use `fork()` or threading in this assignment, or your score will be **0**.

For task 1 to 4, it's guaranteed that client will not disconnect unless the output is incorrect or the operation is completed.

1. Handle valid requests on 1 server, while the server will only have 1 connection simultaneously.
 - 1 read server **(0.5 point)**
 - 1 write server **(0.5 point)**
2. Handle valid and invalid requests on 1 server, while the server will only have 1 connection simultaneously. **(0.25 point)**

3. Handle valid requests on 1 server, while there will be multiple connections simultaneously.
(1.5 point)
4. Handle valid requests on multiple servers (at most 4 servers), while each server will have only 1 connection simultaneously and different servers may access `bookingRecord` simultaneously.
(1.5 point)
5. Handle valid requests on multiple servers (at most 4 servers), while each server will have multiple connections simultaneously and different servers may access `bookingRecord` simultaneously.
 - Clients will always finish their operations **(1.75 point)**
 - Clients may close their connections at any stage of their operations **(1 point)**
6. Report **(1 point)**
 - Please answer the problems on Homework section of NTU COOL

Your server should respond to a request from client **in 0.2s**, or you will fail to pass the testdata. For every task, you have to pass all the testdata to receive point for that part. So make sure that you think twice during coding.

7. Submission

7.1 To GitHub

► **Click me**

7.2 To NTU COOL

► **Click me**

8. Reminder

▼ **Click me**

1. Plagiarism is **STRICTLY** prohibited.
2. Late policy (D refers to formal deadline, **10/14**)
 - If you submit your assignment **on D+1 or D+2**, your score will be multiplied by **0.85**.
 - If you submit your assignment **between D+3 and D+5**, your score will be multiplied by **0.7**.
 - If you submit your assignment **between D+6 and 10/31**, your score will be multiplied by **0.5**.
 - Late submission after **10/31** will not be accepted.
3. If you have any question, feel free to:
 - evoke issues in [NTU-SP/SP-Hw1-release](https://github.com/NTU-SP/SP-Hw1-release/issues) (<https://github.com/NTU-SP/SP-Hw1-release/issues>).
 - send mails to ntusp2022@gmail.com (<mailto:ntusp2022@gmail.com>).
 - ask during TA hours (you should inform TA by email first).

4. Any form of cheating, lying, or plagiarism will not be tolerated. Students can get **zero scores** or **fail the class** for those kinds of misconducts.
5. Please start your work as soon as possible, do **NOT** leave it until the last day!
6. Watch out for the **typos**!
7. It's suggested that you should check return value of system calls like open, select, etc.
8. Reading manuals of system calls may be tedious, but it's definitely helpful.