System Programming 2019 Fall

Programming Assignment # 1

Due: 23:59 Tue, Oct 22, 2019

1.Problem Description

In assignment 1, you are expected to implement a simplified "**Banking System**". You need to complete the simple servers which we already handled a lot of works such as internet connection, memory allocation, etc. The provided source code can be compiled and run as very simple read/write servers, which can only serve one request per time. Things you need to do is to modify it so that it can support **I/O multiplexing**, deal with **many requests at the same time**, rather than be blocked by only one request, and to action on the same account by using different servers.

To summarize, there are **three** tasks for you:

1. Use **select() to do a multiplexing banking system**. There may be multiple clients/accounts that connect to servers and send request at the same time.

2. Please use **file lock** to guarantee the correctness when there are multiple read/write requests occurring on one account at the same time.

3. Implement the **Account structure**. The structure of an account is a struct in C which have two attributes : id and balance. Each of them belongs to type int. There is a file called ***account_list*** including 20 continuous account structures with id from 1~20, and randomly specified price at beginning.

```
typedef  struct{
    int    id;
    int    balance;
```

```
} Account;
```

2.How to Run the Sample Servers

Compile

To produce execution files, you have to compile *server.c* . If you take a look on makefile, you will see that the execution file write_server is compiled directly. However, if you computing it with *-D READ_SERVER,* the execution file *read_server* will be produce. On the first attempt, you can type the command, make, after extracting the files on the workstation.

Run

```
$ ./read_server {port_num}
```

(Take read_server for example, where {port_num} is the port you want assign to the server.)

3.How to Test Servers at Client Side

Use telnet to connect to your servers

```
$ telnet  {host}  {port_num}
```

Where {host} is the location of your servers run, i.e., linux1.csie.ntu.edu.tw. Note that the {port_num} must be the same with one you run the read/write server. If you're testing the sample server, you can type anything, pressing enter and the content of your input will immediately send back to you. However, if you're testing your banking system, there may be different cases described below. After connecting to the read server, you need to send an account-id to query the details of the correspondings account. And then type single enter to indicate the end of the input. There have **two** different situations.

Case 1 - You can query this account

The banking system should return the details of the account and the connection will be closed.

Case 2 - Anyone else is changing this account

The banking system should return “This account is locked!” and the connection will be closed.

Connect to a writer server

On the other hand, if you connect to a write server, you should send an account-id to specify the account that you want to change at the first line. You might get **two** different types of response immediately.

Case 1 - You get the right to do some action on this account

The banking system should return “This account is modifiable.” Next, you should type an action command, i.e., save, withdraw, transfer or balance. There are four actions which can be done by the banking system.

save: save the money into account. The balance of the account should be increased. For example, “save 20”.

withdraw: withdraw money from account. The balance of the account should be decreased. For example “withdraw 20”

transfer: Transfer money from account A to account B. The balance of account A should be decreased while the balance of account B should be increased. There are two integers in this command. The first integer is the id of account B and the second integer is the money that need to be transfered. For example, “transfer 1 20”.

balance: Set the account balance to a specify value.

If you succeed in changing the details of the account, the connection will be closed.

Case 2 - You get the right to do the actions but you failed

The banking system should return “This account is modifiable.” Next, if you specify an invalid command. The operation will fail and the banking system

should return “Operation failed.” And then the connection will closed. Below shows the definition of invalid command in each command:

save: The integer in this command should be bigger than or equal to 0. Otherwise, it is an invalid command. For example, “save -20” is invalid command and “save 0” is a valid command.

withdraw: The integer in this command should be bigger than or equal to 0 and less than or equal to the balance of the account . Otherwise, it is an invalid command. For example, if the balance of the account is 100, then “withdraw -100” and “withdraw 200” are invalid commands while “withdraw 0” and “withdraw 100” are valid command.

transfer: The second integer should be bigger than or equal to 0 and less than or equal to the balance of the account A. Otherwise, it is an invalid command. For example, if the balance of account A is 200, then “transfer 1 201” and “transfer 1 -100” are invalid commands. “transfer 1 0” and “transfer 1 200” is valid commands. You can assume the first integer is always valid and the account A and account B does not have same account id.

balance: The specify value should greater than 0. Otherwise, it is an invalid command. For example, balance 100 is valid command and balance -100 is invalid command.

Case 3 - Anyone else is changing the same account

For all the command, make sure there is no anyone else is changing the same account. In transfer command, you can always assume that there is no anyone else changing the account B. Otherwise, the banking system should return “This account is locked.” and the connection will be closed.

4.Input and Output format

The following text with [standard input] is entered by user, while [standard output] is printed by the program. Suppose the host is at *linux1.csie.org*

Read

Server Side

$ ./read_server 12345

[standard output] starting on linux1, port 12345, fd 3, maxconn 65536

Client Side

$ telnet linux1.csie.ntu.edu.tw 12345

[standard output] Trying 140.112.30.32.....
[standard output] Connected to linux1.csie.ntu.edu.tw.
[stabdard output] Escape character is '^]'.

Case 1 - You can query this account

[standard input] 2
[standard output] 2 4108
[standard output] Connection closed by foreign host.

Case 2 - Anyone else is changing this account

[standard input] 2
[standard output] This account is locked.
[standard output] Connection closed by foreign host.

Write

Server Side

$ ./write_server 12346

[standard output] starting on linux1, port 12346, fd 3, maxconn 65536

Client Side

$ telnet 140.112.30.32 12346

[standard output] Trying 140.112.30.32.....
[standard output] Connected to linux1.csie.ntu.edu.tw.
[standard output] Escape character is '^]'.

Case 1 - You get the right to do some action on this account

[standard input] 2
[standard output] This account is modifiable
[standard input] save 200
[standard output] Connection closed by foreign host.

Case 2 - You get the right to do the actions but you failed

(a)save

[standard input] 2
[standard output] This account is modifiable.
[standard input] save -200
[standard output] Operation failed.
[standard output] Connection closed by foreign host.

(b)withdraw

[standard input] 2
[standard output] This account is modifiable.
[standard input] withdraw 200000
[standard output] Operation failed.
[standard output] Connection closed by foreign host.

(c)transfer

[standard input] 2
[standard output] This account is modifiable.

[standard input] transfer 1 -100
[standard output] Operation failed.
[standard output] Connection closed by foreign host.

Case 3 - Anyone else is changing the same account

[standard input] 2
[standard output] This account is locked.
[standard output] Connection closed by foreign host.

5.Grading

There are 7 subtasks in this assignment. By finishing all subtasks that you earn the full points.

1. You can produce the execution file successfully. (1 point)
   That is, produce read_server and write_server by Makefile .
2. read_server returns the details of specific account. (1 point)
   There would be only one request at a time for this subtask.
3. write_server can change the details of account correctly. (1 point)
   There would be only one request at a time for this subtask.
4. Two requests issued to read_server . (1 point)
   A read request p connects to read_server but hasn't send the request. Then a read request q connects to read_server and sends the request. The read_server should be able to respond to request q .
5. Two requests issued to write_server . (1 point)
   A write request p connects to write_server , and it specifies the account-id but doesn't send the action command. Then a write request q connects to write_server and it specifies a different account-id . The write_server should be able to respond to request q .
6. Protection on single write_server . (1 point)
   If two or more clients connect to single write_server , the account locked by a certain request should not be written by other request.
7. Protection on multi-server. (1 points)
   In multi-server schema, the account locked by a certain server should be protected by advisory file lock to keep correctness

6.Submission

Your Assignment should be submitted to the CEIBA before the deadline. Or you will receive penalty.

At least **three files** should be included:

1. server.c (as well as all other .c files)
2. Makefile
3. readme.txt

Detail

Since we will directly execute your Makefile , therefore you can modify the names of .c files, but Makefile should compile your source into **two** executable files named read_server and write_server . In readme.txt , please briefly state how do you finish your program and something valuable you want to explain. These files should be put inside a folder named with your student ID (in lower case) and you should compress the folder into a .tar.gz before submission. Please do not use .rar or any other file types.

(The commands below will do the trick. Suppose your student ID is b07902000)

$ mkdir b07902000

*$ cp Makefile readme.txt *.c b07902000/*

$ tar –zcvf SP_HW1_b07902000.tar.gz b07902000/ $ rm –r b07902000/

Please do **NOT** add executable files to the compressed file. Errors in the submission file (such as files not in a directory named with your student ID, compiled binary not named read_server and write_server , and so on) may cause

deduction of your credits. Please Submit the compressed file ***SP_HW1_b07902000.tar.gz*** to CEIBA.

7.Reminder

1. Plagiarism is **STRICTLY** prohibited.
2. Your credits will be deducted for each day delay, but a late submission is still better than absence.
3. If you have any question, please feel free to contact us via email *ntucsiesp@gmail.com* or come to R302 during TA hours.
4. Please start your work ASAP and do not leave it until the last day!
5. **Good luck!**