

# System Programming - 2018 Fall

---

## Programming Assignment #1

Due: 23:59 Tue, Oct 16th, 2018

### 1. Problem Description

In assignment 1, you are expected to implement a simplified "bidding 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 guarantee the correctness of the details of each item when more than one client do actions on the same item by using different servers.

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

1. Use `select()` to do a multiplexing bidding system. There may be multiple clients that connect to servers and send request at the same time.
2. 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 Item structure. The structure of an item is a struct in C which have three members, `id`, `amount`, `price`, of type `int`. There is a file called `item_list` including 20 continuous Item structures with `id` from 1 to 20, `amount` of 10 and randomly specified `price` at beginning.

```
typedef struct {
    int id;
    int amount;
    int price;
} Item;
```

### 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 compiling 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 that you want assign to the server.

## 3. How to Test the 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 the 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 bidding system, there may be different cases described below. After connecting to the read server, you need to send an item-id to query the details of the corresponding item. And then type single enter to indicate the end of the input. There have **two** different situations.

### Case 1 - you can query this item

The bidding system should return the details of the item and the connection will be closed.

### Case 2 - anyone else is changing this item

The bidding system should return "This item is locked." and the connection will be closed.

## Connect to a write server

On the other hand, if you connect to a write server, you should send an item-id to specify the item 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 actions on this account

The bidding system should return "This item is modifiable." Next, you should type an action command, i.e., buy, sell or price, followed by a whitespace and an integer indicated the amount. The amount specified bysell command is used to increase the amount of the item, not to set the amount directly. If you succeed in changing the details of the item, the connection will be closed.

## Case 2 - you get the right to buy but you failed

The bidding system should return "This item is modifiable." Next, if you specify a amount greater than the remaining amount of the item or specify a negative number for price, the operation will fail and the bidding system should return "Operation failed." And then the connection will be closed.

## Case 3 - anyone else is changing the same item

The bidding system should return "This item is locked." and the connection will be closed.

## 4. Input and Output Format

The following text with [Standard In-put] 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 140.112.30.32 12345
```

```
[Standard Output] Trying 140.112.30.32...  
[Standard Output] Connected to linux1.csie.ntu.edu.tw.  
[Standard Output] Escape character is '^'.
```

### Case 1

```
[Standard In-put] 2  
[Standard Output] item2 $300 remain: 10  
[Standard Output] Connection closed by foreign host.
```

### Case 2

```
[Standard In-put] 2  
[Standard Output] This item is locked.  
[Standard Output] Connection closed by foreign host.
```

## Write

### Server Side

```
$ ./write_server 12346
```

```
[Standard Output] starting on linux1, port 12345, 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

```
[Standard In-put] 2  
[Standard Output] This item is modifiable.  
[Standard In-put] sell 20  
[Standard Output] Connection closed by foreign host.
```

### Case 2-1

```
[Standard In-put] 2  
[Standard Output] This item is modifiable.  
[Standard In-put] buy 20  
[Standard Output] Operation failed.  
[Standard Output] Connection closed by foreign host.
```

### Case 2-2

```
[Standard In-put] 2  
[Standard Output] This item is modifiable.  
[Standard In-put] price -200  
[Standard Output] Operation failed.  
[Standard Output] Connection closed by foreign host.
```

### Case 3

```
[Standard In-put] 2  
[Standard Output] This item 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 8 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 item. (1 point)  
There would be only **one** request at a time for this subtask.
3. `write_server` can change the details of item 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 `item-id` but doesn't send the action command. Then a write request `q` connects to `write_server` and it specifies a different `item-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 item locked by a certain request should not be written by other request.
7. Protection on multi-server. (2 points)  
In multi-server schema, the item 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 `b03902068`:

```
$ mkdir b03902068
$ cp Makefile readme.txt *.c b03902068/
$ tar -zcvf SP_HW1_b03902068.tar.gz b03902068/
$ rm -r b03902068/
```

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. Submit the compressed file `SP_HW1_b03902068.tar.gz` to CEIBA.

## 7. Reminder

1. Plagiarism is **STRICTLY** prohibited.
2. Your credits will be deducted 5% for each day delay, but a late submission is still better than absence.
3. If you have any question, please ask questions at the **Fackbook Group**, 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!