Operating Systems (CSL 3030)

Readme file

Lab Assignment	**03**
Roll No.	**B20CS018**
Name	**Harshita Gupta**

PART II:

Ques 1

First we need to run the server1.c file by giving the input commands –

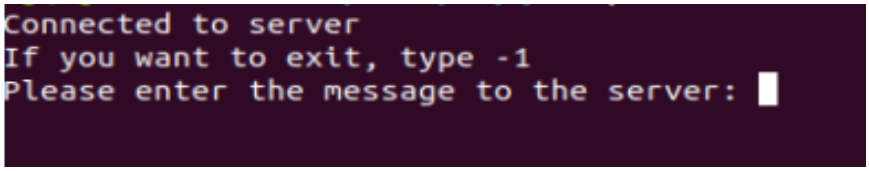
1. gcc server1.c –o server1
2. ./server1 8080 **(8080 is a port number, you can take any such valid no.)**

Then you need to open another terminal which is the client terminal and write the commands –

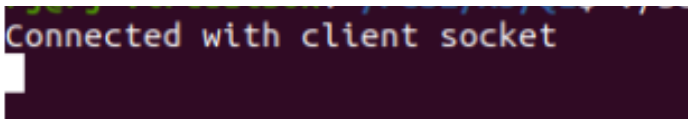
1. gcc client.c –o client
2. ./client 127.0.0.1 8080 **(127.0.0.1 is the IP address and 8080 is the port no. which should be the same as provided in the server terminal)**

(NOTE: If the server is listening on the port the client asked, then comms happens. Otherwise, the connection cannot continue. So the port that the server is bound to (or listening on) must be the same as the port the client specified. **)**

The output at the client side that shows that the client is connected to the server is shown below:

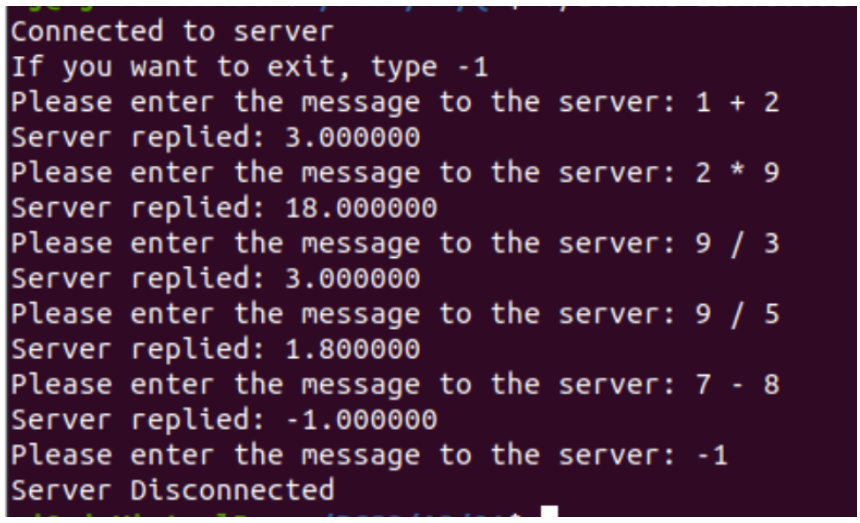


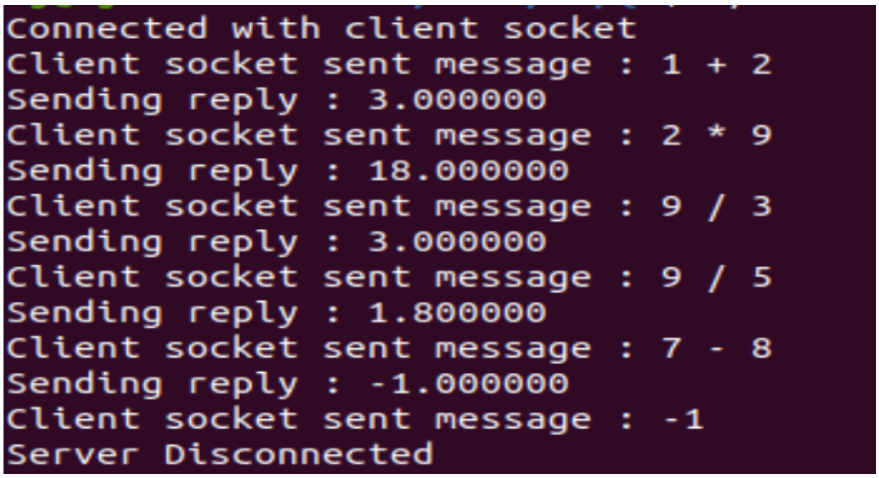
The output at the server terminal that shows that the server is also connected with the client socket is shown below:



Now the client terminal asks you to enter a message to the server.

The basic calculations are all displayed here, and you can repeat them as often as you like up until you type the number -1 into the terminal. When you do, the server and clients will both disconnect.





Some of the things to keep in mind while giving input to the user –

1. Do provide the integers as single-digit integers only, as the code is capable of handling only single-digit integers.
2. Write an integer, give a space, write an operator, give a space and then write another integer, i.e., **2 + 3, 1 / 5 are valid, but 2/ 5, 2*2, 5 +3 are all invalid** and will not show proper results.

This was our single client single server implementation as asked in Q1, if **another client tries to connect with the same IP address and port no. , it will not receive any outputs from the server side, nor it will be able to send any data to the server for calculations.**

Ques 2

First you need to write the commands –

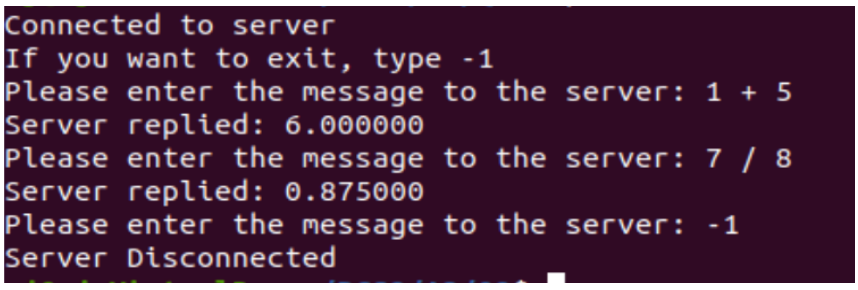
1. gcc server2.c –o server2
2. ./server2 8000

And next you need to open the terminals, as many clients you need, for eg., here **3 terminals are shown at once which are simultaneously connecting with the server and the server is responding to each client correctly.**

For this, you need to run the commands –

1. gcc client.c –o client
2. ./client 127.0.0.1 8000

These are the three client terminals that are concurrently connected to the server, and as you can see, they are all receiving responses from it. The server also indicates to which client it is responding.



```
Connected to server
If you want to exit, type -1
Please enter the message to the server: 1 + 9
Server replied: 10.000000
Please enter the message to the server:
```

```
Connected to server
If you want to exit, type -1
Please enter the message to the server: 9 * 8
Server replied: 72.000000
Please enter the message to the server: 9 / 9
Server replied: 1.000000
Please enter the message to the server:
```

Additionally, you can see from the screenshots that even **if one client disconnects, the other two continue to function flawlessly.**

Ques 3

First you need to write the commands –

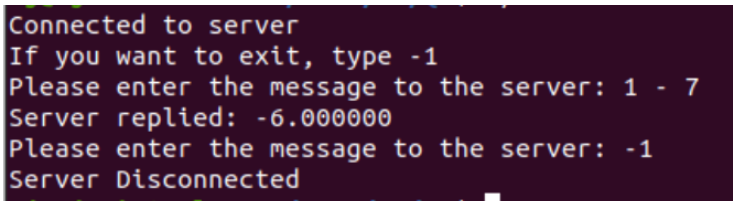
1. gcc server3.c –o server3
2. ./server3 3333

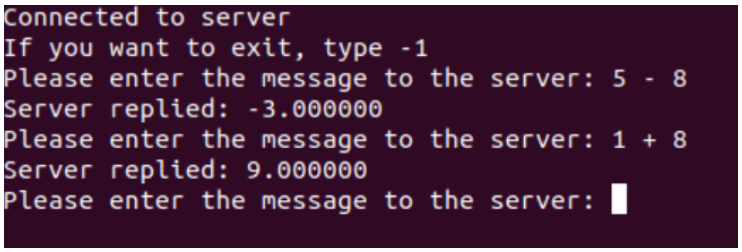
```
Connected with client socket 127.0.0.1 37904
Client socket 127.0.0.1 37904 sent message : 1 - 7
Sending reply : -6.000000
Connected with client socket 127.0.0.1 37906
Client socket 127.0.0.1 37906 sent message : 5 - 8
Sending reply : -3.000000
Connected with client socket 127.0.0.1 37908
Client socket 127.0.0.1 37908 sent message : 2 * 7
Sending reply : 14.000000
Client socket 127.0.0.1 37908 sent message : -1
Disonnected with client socket 127.0.0.1 37908
Client socket 127.0.0.1 37908 sent message : 1 + 8
Sending reply : 9.000000
```

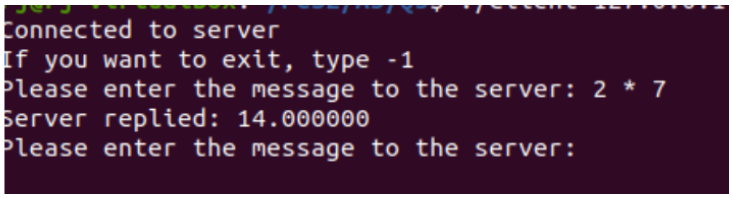
And next you need to open the terminals, as many clients you need, for eg., here **3 terminals are shown at once which are simultaneously connecting with the server and the server is responding to each client correctly.**

For this, you need to run the commands –

1. gcc client.c –o client
2. ./client 127.0.0.1 3333







(One crucial point to note is that because it is stated in the code, **you can adjust the maximum number of clients that can be connected at once to suit your needs**. However, if one of the ten clients disconnects, you can connect one more client because the disconnected client is removed from the fd set after disconnecting, and any value of -1 is checked in the fd set before connecting with any new client to identify any open slots.)

Comparative Performance Analysis -

1. Since there is only one client and one server communicating with one another in the first section and no fork, select, or multithreading is used, it is the simplest client-server connection possible.

2. The server forks multiple processes in the second section, each of which is capable of accepting connections on a single listening socket that has been opened.

3. A single listening socket is opened in the third section as well, but this time the select function is used to check if any of the file descriptors are receiving activity. The file descriptor in this server code is of the order O because it scans through each connected socket fd (n). In contrast to creating separate threads for each incoming client, the benefit of a select system call is that a single thread can manage multiple sockets and is scalable for a large number of clients. However, it is considerably slower.