Socket Programming Assignment

Due Date: April 5th , 11pm

This lab is to be done individually.

Every single code of line should be your own. There will be a written viva later as part of final examination.

Description

In this assignment, we will build a simple client-server system in C/C++. No other programming language is permitted: this is best for learning the internals of socket programming. The idea is to have a simple chat system where you use the client to chat with a server. The protocol between the client and server is as follows.

- The server program is first started. The program should take exactly one command-line argument: the TCP port number on which it should listen. It should exit with an exit code of 1 if given wrong number of arguments or a non-numeric argument. It should exit with an exit code of 2 if it is unable to bind to the given port number, and print the appropriate reason.
- Then the client program is started. The client program takes the server IP address and port as command-line arguments. For this assignment, you are given the client side program: you only have to write the server side.
- The client then connects to the server on a TCP socket.
- The client then asks the user for input. The user types his message on the terminal (e.g., "Hi", “Whatsup?”, "Bye"). The user's input is sent to the server via the connected socket.
- The server reads the user's input from the client socket. If the user has typed "Bye" (without the quotes), the server should reply with "Goodbye <IP:port>", where IP:port is the client side IP address and port for this TCP connection. For any other message, the server must reply with "OK <IP:port>". Note that your server code should figure out the client side IP address and port using appropriate socket calls.
- The client then reads the reply from the server, and checks that it is accurate (either "OK ..." or "Goodbye ...").
- If the user had typed "Bye", and the server replied with a goodbye, the client quits. Otherwise, the client asks the user for the next message to send to the server.

You are provided with the client (source code). You will write **two** versions of the server with increasing functionality.

1. Your server program "server1.c" will be a single process server that can handle only one client at a time. If a second client tries to chat with the server while one client's session is already in progress, the second client's should NOT be able to connect with the server (i.e. it should get a connection error). In this version of the server, the server may choose to exit after interacting with a single client.
2. The second version of your server program "server2.c" will be a single threaded, single process server (no fork or multi-threading) that uses the "select" system call to handle

multiple clients (at least 3 simultaneous clients). Additionally, server2 will support the following client command.

- When a client types “List”, the server should respond with the reply string “OK <IP1:port2>, <IP2:port2>” (and as many clients are currently connected). That is, the current set of clients (including the one asking the listing) are listed in the given format. The clients can be listed in any order.

Several good tutorials and useful documentation on socket programming and designing servers for concurrent clients are available online. Please make use of these resources to learn the intricacies of socket programming on your own during this assignment.

Please write your code for this assignment in C or C++ ONLY.

A good part of your submission will be evaluated automatically, so pay ATTENTION to the specific format of various strings given, as well as see the example output below.

Example output

Below is a sample run of the client. For this example, the client code is first compiled. Then a server is run on port 5000 in another terminal. The client program is then given the server IP (localhost 127.0.0.1 in this case) and port (5000) as command-line inputs. When the user enters messages like Hello or Hi, the server replies with OK <IP:port>. When the user says Bye, the server says Goodbye <IP:port>. The client program will exit after user enters Bye and server replies. Below, I have assumed that the client side port is 8642.

Running server on a terminal:

```
$ gcc server1.c -o server1
$ ./server1 5000
```

Running client on another terminal:

```
$ gcc client.c -o client
$ ./client 127.0.0.1 5000
Connected to server
Please enter the message to the server: Hello
Server replied: OK 127.0.0.1:8642
Please enter the message to the server: Hi
Server replied: OK 127.0.0.1:8642
Please enter the message to the server: Bye
Server replied: Goodbye 127.0.0.1:8642
$
```

In another run of the client, we show what happens when the server sends a wrong reply, say, "NO" instead of "OK". Here, the client exits with error message. This behavior should not happen with your server code.

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the message to the server: Hello
Server replied: NO
```

```
ERROR wrong reply from server
```

Now suppose you run server2.c on a terminal:

```
$ gcc server2.c -o server2
$ ./server2 5000
```

Here we show an example interaction of this server with two simultaneous clients.

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the message to the server: Hello
Server replied: OK 127.0.0.1:8642
```

Now on another terminal, another client process is started without closing the above client. Below we assume that this 2nd client gets a client side port of 7531

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the message to the server: Hello
Server replied: OK 127.0.0.1:7531
Please enter the message to the server: List
Server replied: OK 127.0.0.1:7531, 127.0.0.1:8642
```

Marking scheme: total 20 marks

- 2 marks: correct exit code on wrong (number of) arguments, as specified
- 2 marks: server1.c responds correctly to messages from a connected client
- 2 marks: server1.c closes client connection correctly on seeing “Bye” and responding to it
- 2 marks: second simultaneous client cannot connection to server1.c
- 4 marks: server2.c works responds correctly to clients with up to 3 simultaneous client connections (note: no multi-threading or multiple server processes allowed)
- 2 marks: server2.c works correctly for the “List” command when no client has disconnected yet
- 3 marks: server2.c works correctly for the “List” command even when some clients have disconnected with the “Bye” command
- 2 marks: appropriate variable/function naming, indentation (marking will be binary here; you can lose 2 marks even if you have a single line improperly indented or a single variable/function non-intuitively named)
- 1 mark: documentation, commenting (marking will be binary here too; use appropriate level of commenting: not too little, not too much)

Submission instructions

The directory named <rollnumber>_lab06 that you will submit should contain the following files:

1. server1.c
2. server2.c

Now tar it as follows:
tar -zcvf <rollnumber>_lab06.tgz <rollnumber>_lab06/

Submit the file <rollnumber>_lab06.tgz via moodle for grading.