

# Computer Networks I

CS3530 - Aug 2019

## Assignment 1:

**Extending echo client/server functions and making it  
protocol independent**

**INSTRUCTOR: Kotaro Kataoka**

**Report By:**  
**Vijay Tadikamalla**  
**CS17BTECH11040**

---



भारतीय प्रौद्योगिकी संस्थान हैदराबाद  
Indian Institute of Technology Hyderabad

## **Part 1: Extending echo client/server functions**

### **Features 1: A Server-Client Chat system.**

A client first connects with the server and then the server assigns a unique ID to each client. This ID can be used by the clients to message each other. A client can join the server and leave the server any time they want.

```
(base) vijayphoenix🐉:~/.../Comp Network/a1$ g++ server.cpp -lpthread -o server
(base) vijayphoenix🐉:~/.../Comp Network/a1$ ./server
Enter: <Server Address>
Enter: <Port>
Server hosted on IP: 127.0.0.1
Listening on Port: 8002 .....
Client 1 connected.
Username Registered: Vijay
█
```

```
(base) vijayphoenix🐉:~/.../Comp Network/a1$ g++ client.cpp -lpthread -o client
(base) vijayphoenix🐉:~/.../Comp Network/a1$ ./client
Enter: <Server Address>
Enter: <Port>
Connected to server: 127.0.0.1
Port: 8002
Enter: <User_ID>
Vijay
----- Welcome Vijay-----

Enter:
Command 1 -> User_List - To get list of all the users
Command 2 -> Send <ID_Number> <Msg> - To send msg to user with <ID_Number> ID
Command 3 -> Command <CMD> - Execute <CMD> bash command on the server and get the output
Command 4 -> Bye - Bye!

Happy Chatting!!!

>>
```

**Figure 1: Client 1 connects to the server with User\_ID Vijay.**

```

(base) vijayphoenix🐉:~/.../Comp Network/a1$ ./server
Enter: <Server Address>
Enter: <Port>
Server hosted on IP: 127.0.0.1
Listening on Port: 8002 .....
Client 1 connected.
Username Registered: Vijay
Client 2 connected.
Username Registered: Turing

(base) vijayphoenix🐉:~/.../Comp Network/a1$ ./client
Enter: <Server Address>
Enter: <Port>
Connected to server: 127.0.0.1
Port: 8002
Enter: <User_ID>
Vijay
----- Welcome Vijay-----

Enter:
Command 1 -> User_List - To get list of all the users
Command 2 -> Send <ID_Number> <Msg> - To send msg to user with <ID_Number> ID
Command 3 -> Command <CMD> - Execute <CMD> bash command on the server and get the output
Command 4 -> Bye - Bye!

Happy Chatting!!!

>> Send 2 Hello Turing
>>
Msg from Client 2: Turing -> Hello Vijay

(base) vijayphoenix🐉:~/.../Comp Network/a1$ ./client
Enter: <Server Address>
Enter: <Port>
Connected to server: 127.0.0.1
Port: 8002
Enter: <User_ID>
Turing
----- Welcome Turing-----

Enter:
Command 1 -> User_List - To get list of all the users
Command 2 -> Send <ID_Number> <Msg> - To send msg to user with <ID_Number> ID
Command 3 -> Command <CMD> - Execute <CMD> bash command on the server and get the output
Command 4 -> Bye - Bye!

Happy Chatting!!!

>>
Msg from Client 1: Vijay -> Hello Turing

>> Send 1 Hello Vijay
>> █

```

**Figure 2: Client 2 (User\_ID Turing) sends a message to Client 1 (User\_ID Vijay).**

## Features 2: Remote Execution of Commands

When a client sends a command to the server, the server executes the command and sends back the results to the client. This feature is a small step towards cryptographic network protocols like **ssh** where we log in from one computer to remote server/computer and execute our commands on the server through a shell.

```
(base) vijayphoenix@~/.Comp Network/a1$ g++ server.cpp >> Command ls
(base) vijayphoenix@~/.Comp Network/a1$ ./server >>
Enter: <Server Address> client
Enter: <Port> client.c
Server hosted on IP: 127.0.0.1 client.cpp
Listening on Port: 8002 ..... man.txt
Client 1 connected. server
Username Registered: User1 server.c
Entering Command mode server.cpp
Exiting Command mode TCPEchoClient4.c
Entering Command mode TCPEchoServer4.c
Exiting Command mode
[] >> Command ls -al
>>
total 104
drwxr-xr-x 2 vijay vijay 4096 Nov 27 05:29 .
drwxr-xr-x 6 vijay vijay 4096 Nov 25 19:08 ..
-rwxr-xr-x 1 vijay vijay 17584 Nov 27 05:17 client
-rw-r--r-- 1 vijay vijay 3525 Nov 26 23:12 client.c
-rw-r--r-- 1 vijay vijay 3485 Nov 27 00:19 client.cpp
-rw-r--r-- 1 vijay vijay 21936 Nov 27 00:25 man.txt
-rwxr-xr-x 1 vijay vijay 26648 Nov 27 05:29 server
-rw-r--r-- 1 vijay vijay 2984 Nov 26 22:33 server.c
-rw-r--r-- 1 vijay vijay 4053 Nov 27 00:20 server.cpp
-rw-r--r-- 1 vijay vijay 2297 Nov 26 00:00 TCPEchoClient4.c
-rw-r--r-- 1 vijay vijay 2937 Nov 26 22:42 TCPEchoServer4.c
>> []
```

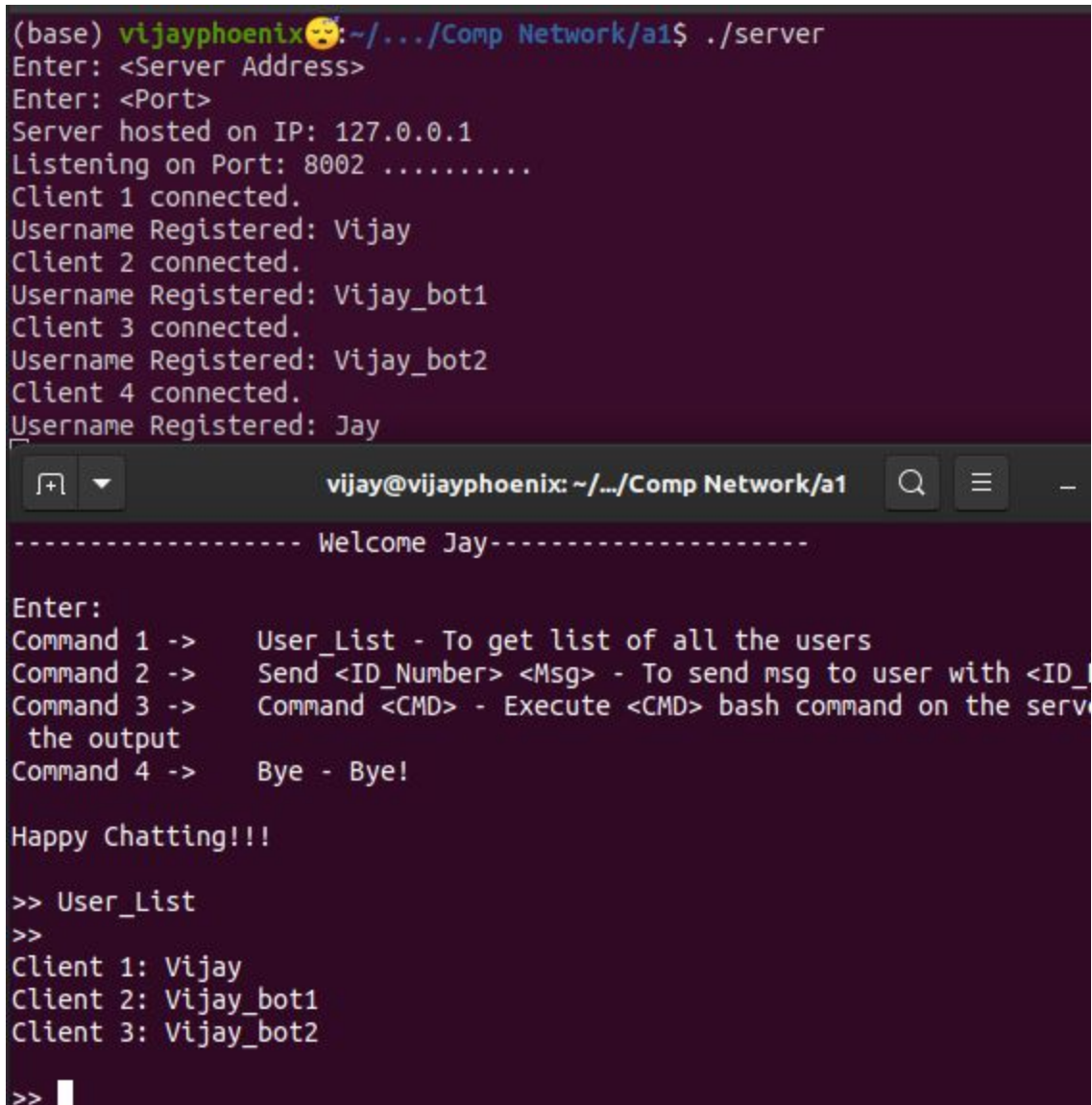
**Figure 3:** Client 1 (User\_ID User1) executes commands like **ls, ls -al** and receives the corresponding output from the server.

This feature properly supports non-interactive commands like **ls, ps, whoami, etc.** Commands which use in streams or some other out streams except stdout may not work properly.

Currently, the output which is sent to the client has a limit of **1024 chars**. So, some commands which have large outputs like **top, etc** may not give the desired output.

## Other Features

- All the commands, messages are non-blocking i.e. no client has to wait to send their message or execute their command
- A client can send a message **"User\_List"** to the server to get the complete list of other client's IDs connected to the server at any point in time.
- Users can exit from the chatroom by sending a message **"Bye"** to the server.
- An atomic boolean variable is used to ensure all the clients exit the server safely when the server is closed
- A logged of all event are printed on the server.

A terminal window with a dark purple background. The top part shows the server being started with the command './server'. It prompts for a server address and port, then displays the IP (127.0.0.1) and port (8002). It shows four clients connecting and registering with usernames: Vijay, Vijay\_bot1, Vijay\_bot2, and Jay. The bottom part shows the client's command menu, including 'User\_List' to get a list of users. The client then enters 'User\_List' and receives a list of the three other users: Vijay, Vijay\_bot1, and Vijay\_bot2.

```
(base) vijayphoenix🐼:~/.../Comp Network/a1$ ./server
Enter: <Server Address>
Enter: <Port>
Server hosted on IP: 127.0.0.1
Listening on Port: 8002 .....
Client 1 connected.
Username Registered: Vijay
Client 2 connected.
Username Registered: Vijay_bot1
Client 3 connected.
Username Registered: Vijay_bot2
Client 4 connected.
Username Registered: Jay

----- Welcome Jay-----

Enter:
Command 1 ->   User_List - To get list of all the users
Command 2 ->   Send <ID_Number> <Msg> - To send msg to user with <ID_
Command 3 ->   Command <CMD> - Execute <CMD> bash command on the serv
the output
Command 4 ->   Bye - Bye!

Happy Chatting!!!

>> User_List
>>
Client 1: Vijay
Client 2: Vijay_bot1
Client 3: Vijay_bot2
>>
```

**Figure 4:** Client 1 (User\_ID Jay) request for the **User\_List** from the server.

## **Part 2: One Server and N Clients (Hard Mode)**

### **About the program:**

- The server manages all the message passing among the users.
- A client can join the server and leave the server any time they want.
- A client can send a message to another client using the following message format:  
Send **<ID\_Number>** message
- The received message has the format:  
Msg from Client: **<ID\_Number>**: **<Sender\_Name>** -> **<Msg>**
- This is very similar to many chatting applications (like WhatsApp, Messenger) where each user can send a message to other users privately. In this type of messaging apps, all the message passing is handled by the company's server.

### **Core Idea:**

- To achieve the above-mentioned goal, we use multithreading.
- So, the server is at any point in time, is able to do message passing between the clients.

```
(base) vijayphoenix🐉:~/../Comp Network/a1$ ./client
Enter: <Server Address>
Enter: <Port>
Connected to server: 127.0.0.1
Port: 8002
Enter: <User_ID>
Turing
----- Welcome Turing-----

Enter:
Command 1 -> User_List - To get list of all the users
Command 2 -> Send <ID_Number> <Msg> - To send msg to user with <ID_Number> ID
Command 3 -> Command <CMD> - Execute <CMD> bash command on the server and get the output
Command 4 -> Bye - Bye!

Happy Chatting!!!

>> Send 1 This is a great app
>>
Message received from Client 1: Vijay -> Thanks Turing!

>> |
```



```

(base) vijayphoenix🐉:~/.../Comp Network/a1$ ./client
Enter: <Server Address>
Enter: <Port>
Connected to server: 127.0.0.1
Port: 8002
Enter: <User_ID>
Vijay
----- Welcome Vijay-----

Enter:
Command 1 -> User_List - To get list of all the users
Command 2 -> Send <ID_Number> <Msg> - To send msg to user with <ID_Number> ID
Command 3 -> Command <CMD> - Execute <CMD> bash command on the server and get the output
Command 4 -> Bye - Bye!

Happy Chatting!!!

>>
Message received from Client 2: Turing -> This is a great app

>> Send 2 Thanks Turing!
>>
Message received from Client 3: TA -> I think I will give you full marks

>> 

```

```

(base) vijayphoenix🐉:~/.../Comp Network/a1$ ./client
Enter: <Server Address>
Enter: <Port>
Connected to server: 127.0.0.1
Port: 8002
Enter: <User_ID>
TA
----- Welcome TA-----

Enter:
Command 1 -> User_List - To get list of all the users
Command 2 -> Send <ID_Number> <Msg> - To send msg to user with <ID_Number> ID
Command 3 -> Command <CMD> - Execute <CMD> bash command on the server and get the output
Command 4 -> Bye - Bye!

Happy Chatting!!!

>> Send 1 I think I will give you full marks

```

**Figure 5: Client 1 (User\_ID Vijay), Client 2 (User\_ID Turing), Client 3 (User\_ID TA) send messages to each other.**

## **Part 3: Protocol Independent Echo Client-Server**

About the Server program:

- The program uses **struct hints**, **\*result**, **\*rp** to get the address info.
- We use the following values of **struct hints** as follows:

```
hints.ai_family = AF_INET6;  
hints.ai_socktype = SOCK_STREAM; // Hardcoded TCP as dummy  
hints.ai_protocol = 0; // Hardcoded TCP as dummy  
hints.ai_flags = AI_PASSIVE;  
hints.ai_canonname = NULL;  
hints.ai_addr = NULL;  
hints.ai_next = NULL;
```

- We then use **getaddrinfo()** which returns a list of address structures.
- We then iterate over each address until all the functions like **socket()**, **bind()** and **listen()** are successful.

About the Client program:

- Like in the Server program we iterate over each address until all the functions like **socket()** and **connect()** are successful.
- The value of **rp->ai\_family** tells us if the connection is IPv4 or IPv6.



```

(base) vijayphoenix🐼:~/.../Submit/a3$ g++ client.cpp -o client
(base) vijayphoenix🐼:~/.../Submit/a3$ g++ server.cpp -o server
(base) vijayphoenix🐼:~/.../Submit/a3$ ./server 8000
Waiting for Client
Mesage Sent to client
Case1
Waiting for Client
Mesage Sent to client
Case2
Waiting for Client
Mesage Sent to client
Case3
Waiting for Client
Mesage Sent to client
----
Handling client ::1 32816
Case4
Waiting for Client
Mesage Sent to client
----
Handling client ::1 32822
Case5

```

```

(base) vijayphoenix🐼:~/.../Submit/a3$ ./client 127.0.0.1 8000 Case1
IPv4 detected
Received: Case1
(base) vijayphoenix🐼:~/.../Submit/a3$
(base) vijayphoenix🐼:~/.../Submit/a3$ ./client 192.168.105.228 8000 Case2
IPv4 detected
Received: Case2
(base) vijayphoenix🐼:~/.../Submit/a3$ ./client localhost 8000 Case3
IPv4 detected
Received: Case3
(base) vijayphoenix🐼:~/.../Submit/a3$ ./client ip6-localhost 8000 Case4
IPv6 detected
Received: Case4
(base) vijayphoenix🐼:~/.../Submit/a3$ ./client ::1 8000 Case5
IPv6 detected
Received: Case5

```

**Figure 6:** Outputs of echo server client on different inputs

## References

- Course Slides
- Man page: getAddrInfo
- <https://stackoverflow.com/questions/478898/how-do-i-execute-a-command-and-get-the-output-of-the-command-within-c-using-popen>
- [https://www.ibm.com/support/knowledgecenter/en/ssw\\_ibm\\_i\\_73/rzab6/xacceptboth.htm](https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_73/rzab6/xacceptboth.htm)
- <https://www.geeksforgeeks.org/multithreading-c-2/>