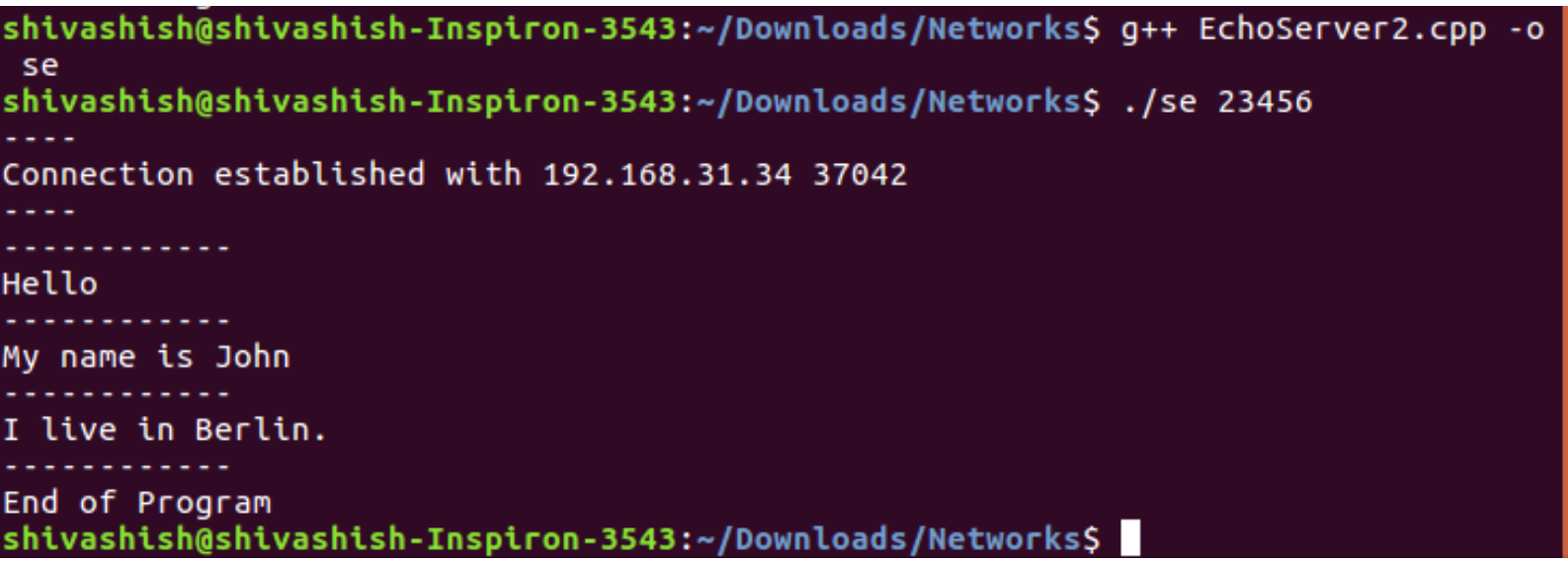
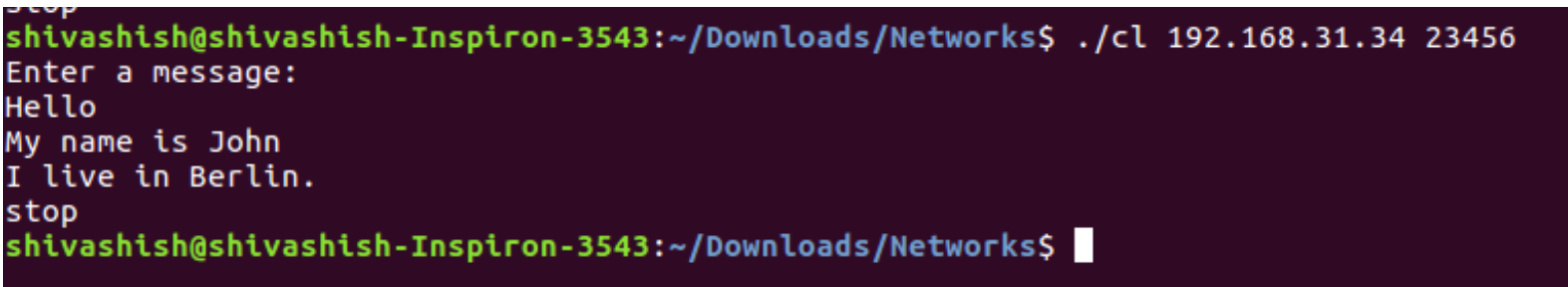
Computer Networks Assignment
Name – Shivashish Suman
Roll No. - CS17BTECH11037

1. I added two features in the echo client/server program.
First feature enables the client to send multiple messages to the server after a connection is established. After ‘stop’ message is sent to the server, both client and server stops sending/receiving the message and program exits.

Program on server side receiving message:



Program on client side sending message:



Second feature is that server receives messages from different clients at different times and each time connection is established with client. At the end if any one of the client sends the message “stop” then server stops listening to the message and prints messages from different IP addresses by grouping messages from same IP address together . After then program exits.Also I added feature to merge all the arguments after port number to form one string (allows space)message which is sent to the server.

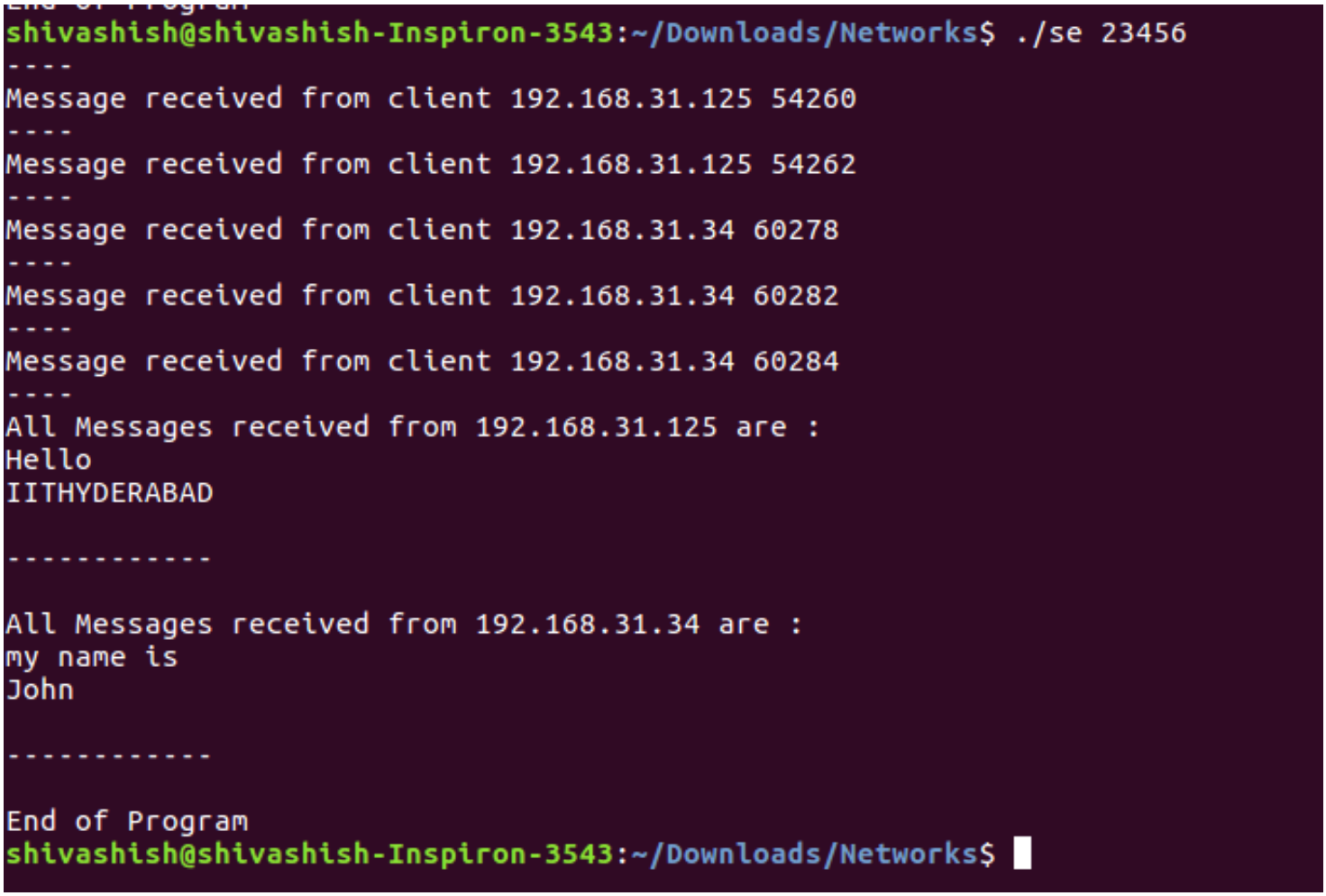
Message sent from client 1 :

```
rushi@rt:~/network-lab$ ./cl 192.168.31.34 23456 Hello
Received: Hello
rushi@rt:~/network-lab$ ./cl 192.168.31.34 23456 IITHYDERABAD
Received: IITHYDERABAD
rushi@rt:~/network-lab$
```

Message sent from client 2:

```
shivashish@shivashish-Inspiron-3543:~/Downloads/Networks$ ./cl 192.168.31.34 23456 my name is
Received: my name is
shivashish@shivashish-Inspiron-3543:~/Downloads/Networks$ ./cl 192.168.31.34 23456 John
Received: John
shivashish@shivashish-Inspiron-3543:~/Downloads/Networks$ ./cl 192.168.31.34 23456 stop
Received: stop
shivashish@shivashish-Inspiron-3543:~/Downloads/Networks$
```

Message received by server and grouped together based on IP address:



2. In the second program I used multi-threading to receive and establish connection among multiple clients at the same time. Firstly, all the clients establish connection to the server and then they start sending and receiving messages. It can be used to handle more than 2 clients at a time. In the code it is mentioned “# define

N 4” which is number of clients and can be changed according to the clients available.
In the client side, one thread continuously runs to check if there is any message received so that it can print. In the server side, each thread runs to handle request from each client and then directs the message to the specified client. So if there are N clients then there are N threads running. Input should be <message>-<client number> i.e. send “message” to client “client number”. Client number is given in the order they connect to server.

Following is the screenshot of messages sent by multiple clients to each other:

Client – 0

Client - 1

Client – 2

```
shivashish@shivashish-Inspiron-3543:~/Downloads/Networks$ ./cl 192.168.31.34 23456
Enter a message:
Message Received:
whats your name

my name is John - 1
whats your name - 0
Message Received:
my name is Brad
```

Client – 3

```
shivashish@shivashish-Inspiron-3543:~/Downloads/Networks$ ./cl 192.168.31.34 23456
Enter a message:
Message Received:
whats your name

my name is Robin - 0
where do u live - 1
Message Received:
 i live in US
```

Server (handling 4 clients at a time) -

```
shivashish@shivashish-Inspiron-3543:~/Downloads/Networks$ ./se 23456
----
Handling client 192.168.31.34 42784
----
Handling client 192.168.31.34 42788
----
Handling client 192.168.31.34 42790
----
Handling client 192.168.31.34 42794
```

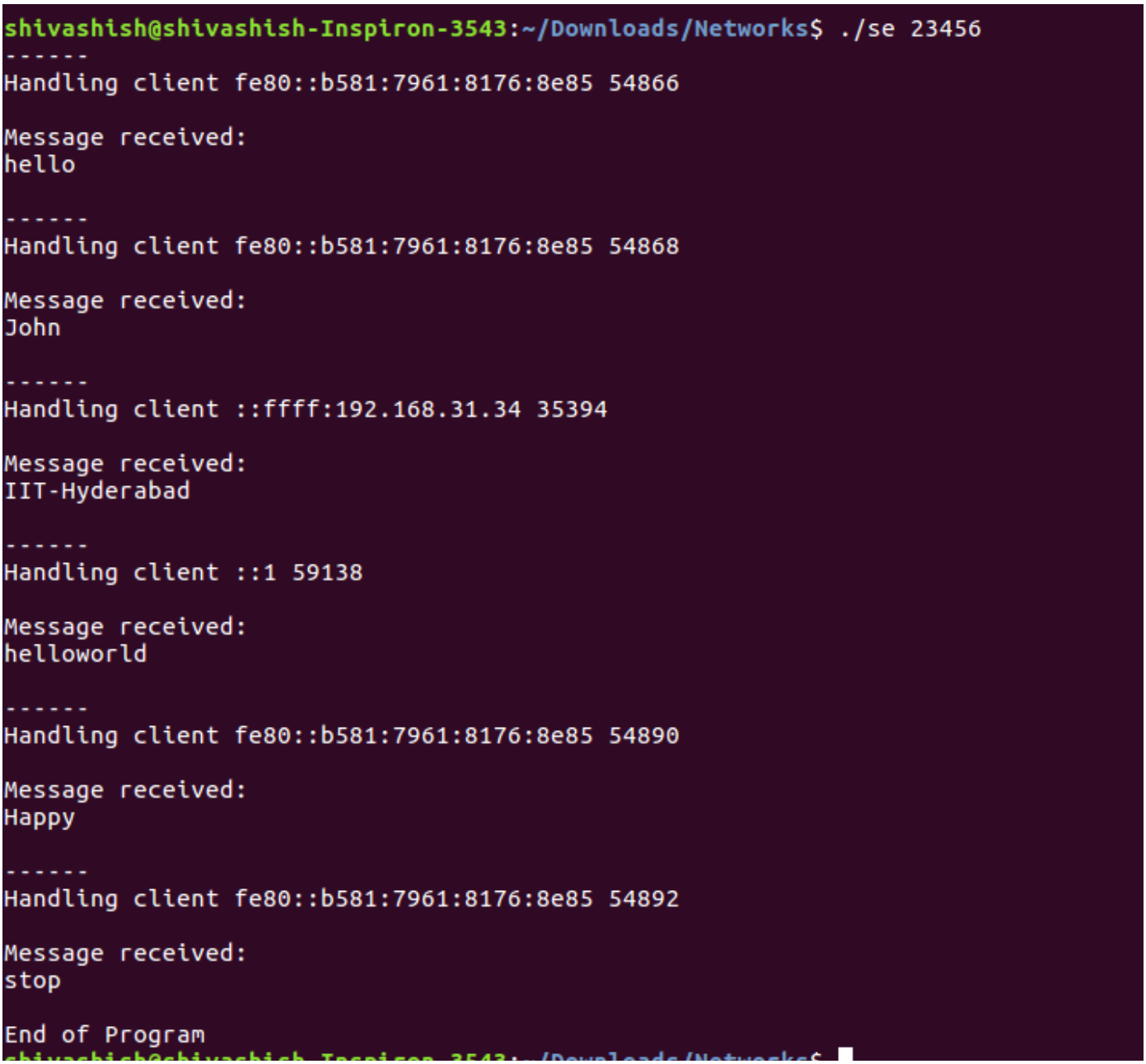
3. Reference used :

(a.)https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzab6/rzab6soxoverview.htm

(b.)https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_73/rzab6/ip6scen.htm

In this program server continues to receive message from client until it receives the message “stop”. It is protocol independent and can support both IPv4 and Ipv6 as shown in the screenshots.
Server Program takes one argument which is port no.
Client Program takes three arguments (IP address of Server, Port no of Server, message).

Server side



Client Side

I have used sockaddr_in6 in place of sockaddr_in so as to support both the protocols. Size of BUFFER is 1024 i.e. max length of message to be transmitted. In the code, we set the address to in6addr_any, which (by default) allows connections to be established from any IPv4 or IPv6 client. getpeername() API is used which returns the client's address to the application. If the client is an IPv4 client, the address is shown as an IPv4–mapped IPv6 address. I have used getaddrinfo() which retrieves the address information needed for the subsequent socket() and connect() calls.