ECEN 602
Network Programming Assignment 2
Simple Broadcast Chat Server and Client

TEAM : 1

SANKET AGARWAL : 329006490
DHIRAJ KUDVA : 829009538

Problem Description:

The objective of this machine problem is to build a chat server and client model where the server acts as a moderator that monitors the entire chatroom. Clients can dynamically join and leave the chat room as and when they want. Also, there is a limit on the maximum number of clients that can be present in the chat room and two clients cannot have the same username.

If any client wants to join the chat room they can use the following way:

./client <IP_address> <Port number> <user_name>

Depending on the situation of the chat room, the joining client would get one of the ACK or NAK messages. Whenever a new client joins the chat room, other clients get an ONLINE message that mentions the username of the new client that joined. Similarly, when the client leaves the chat room, a broadcast message with the OFFLINE tag is sent along with the username of the client that left the chat room.

Team Contribution:
Client code, Test cases and common_def code was written by Sanket Agarwal whereas server code was written by Dhiraj Kudva. Tested over a zoom call on weekends.

Directory Structure:

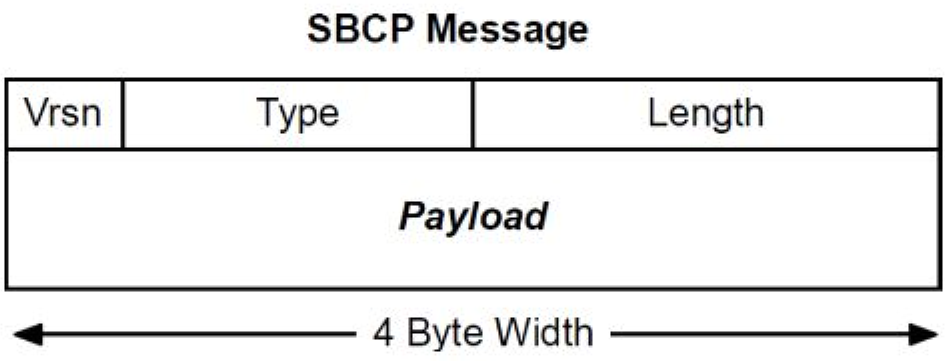
client.c: It contains the code of the client side. Initially the client will check if the connection is IPv4 or IPv6 and then later create a socket depending on the type of IP address that is provided to the client.

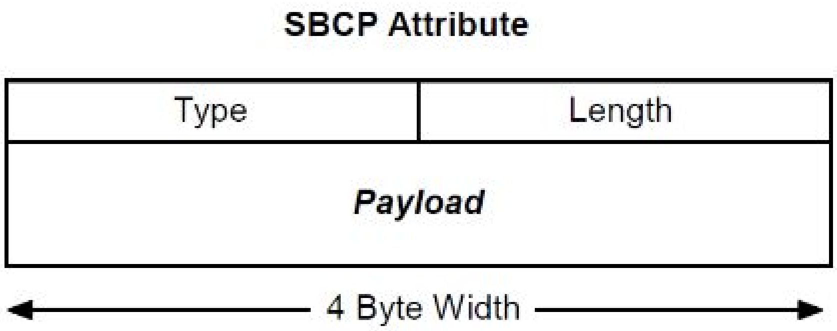
Once the socket is created, it will do the usual process of connecting to the server as done in the previous Machine problem. However, here we have an important function called select that enables the client to listen to both STDIN input as well as the input from the server. Thus at a given moment the client is listening to the server as well as ready to listen to the user via the STDIN input.

server.c: Server monitors the entire chat room and enables the user to send messages to all the clients connected to it. It receives messages from a client and broadcasts them to all the other clients connected to the server. The values specified in the header helps the server to classify messages into different types such as ACK, NAK, IDLE, ONLINE, and OFFLINE.

common_def.h: It contains the basic structure definitions of messages as defined in the manual. For reference purposes they are as show below:

Makefile: It contains the code to generate the executable bin files of the c files.





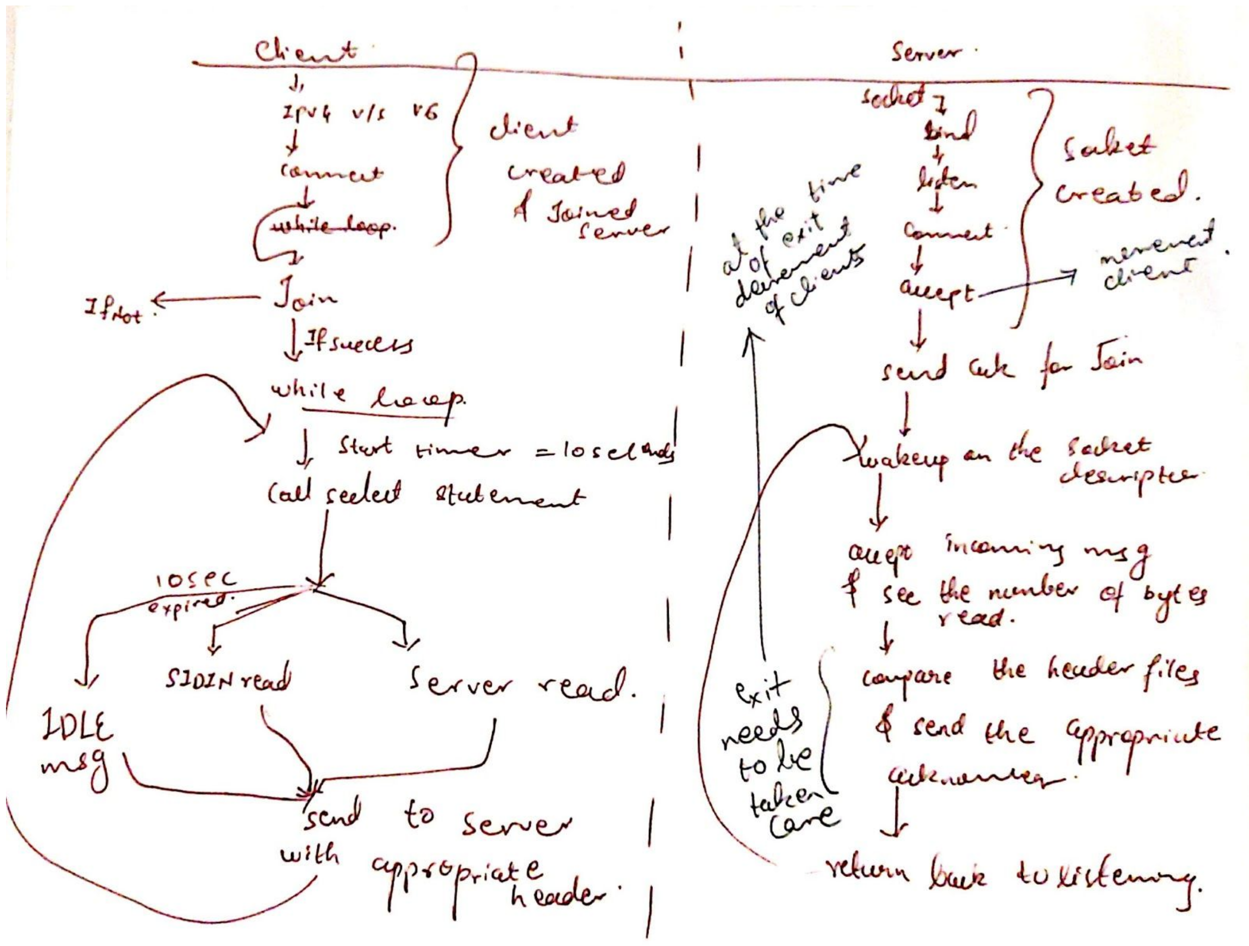
Bonus Features:

1) IPv4 and IPv6 functionality have been added. But there was no way to test it out as of now.

2) ACK, NAK, ONLINE and OFFLINE features have been implemented as described in the manual provided by the TA

3) IDLE messages have also been implemented and as described in the manual, the client will wait for 10 secs and send a response to the server saying he is idle. The server will broadcast this information to the other clients that will indicate the idle client.

Architecture discussed before implementation:


Client
IPV4 v/s V6
connect
~~while loop.~~
client created & Joined server
Join
If not
If success
while loop.
Start timer = 10 seconds
Call select statement
10sec expired.
IDLE msg
STDIN read
Server read.
send to server with appropriate header.
Server
socket
bind
listen
connect
accept
Socket created.
new event client
at the time of exit deletement of clients
send ack for Join
wakeup on the socket descriptor
accept incoming msg & see the number of bytes read.
compare the header files & send the appropriate acknowledge
exit needs to be taken care
return back to listening.

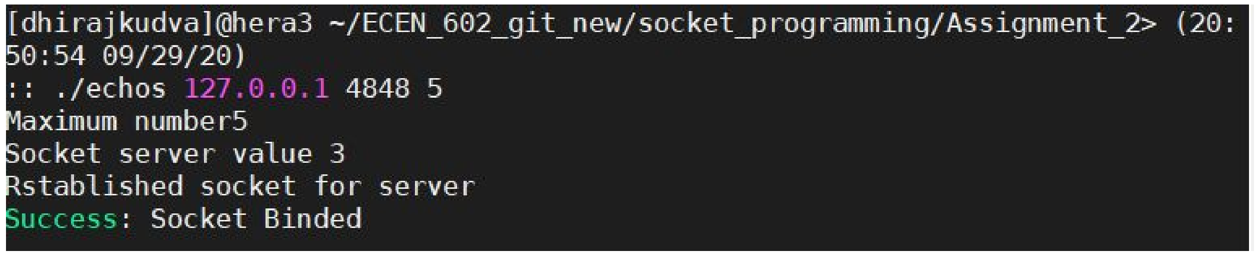
Usage:
The directory shared as the following necessary files:

1. client.c
2. server.c
3. common_def.h
4. Makefile

First step would be to generate the object files for client and server.
Once you are in this working directory, run “make” on the command line. This will result in two object files namely echos (generated from server.c) and echo (generated from client).

To setup client and socket environment, open another terminal and point to the same working directory where all the collaterals (necessary files) are generated.

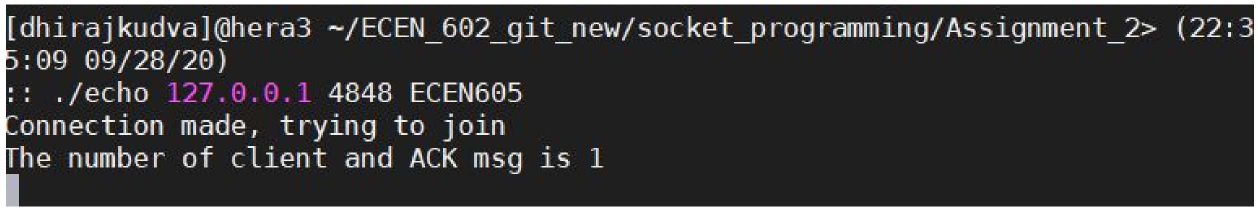
The next step would be to start the server. Run “./echos 127.0.0.1 4848 5” where 127.0.0.1 is the IP address of the server 4848 is the port number and 5 is the maximum number of clients that can be connected to the server. The client joining this server must provide the same IP address and port number. Any port number can be selected within the valid range.
Once this is setup, server will show the below message



```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (20:
50:54 09/29/20)
:: ./echos 127.0.0.1 4848 5
Maximum number5
Socket server value 3
Rstablished socket for server
Success: Socket Binded
```

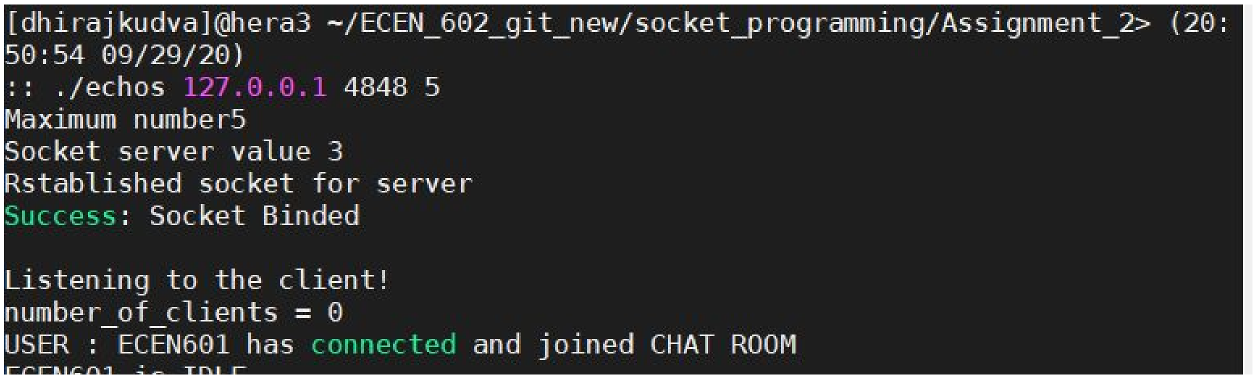
This indicates that the server is listening for the clients ready to join.

In the second terminal, run “./echo 127.0.0.1 4848 ECEN605” . The IP address 127.0.0.1 corresponds to the IP address of the server. 4848 here is the port number and this number has to same as the one mentioned in the earlier command to start the server. ECEN605 is the username given to the client. (This username has to be different from other clients connected to the server)



```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:3
5:09 09/28/20)
:: ./echo 127.0.0.1 4848 ECEN605
Connection made, trying to join
The number of client and ACK msg is 1
```

And at the same time you will see some extra messages printed on the server console indicating that the client is connected.



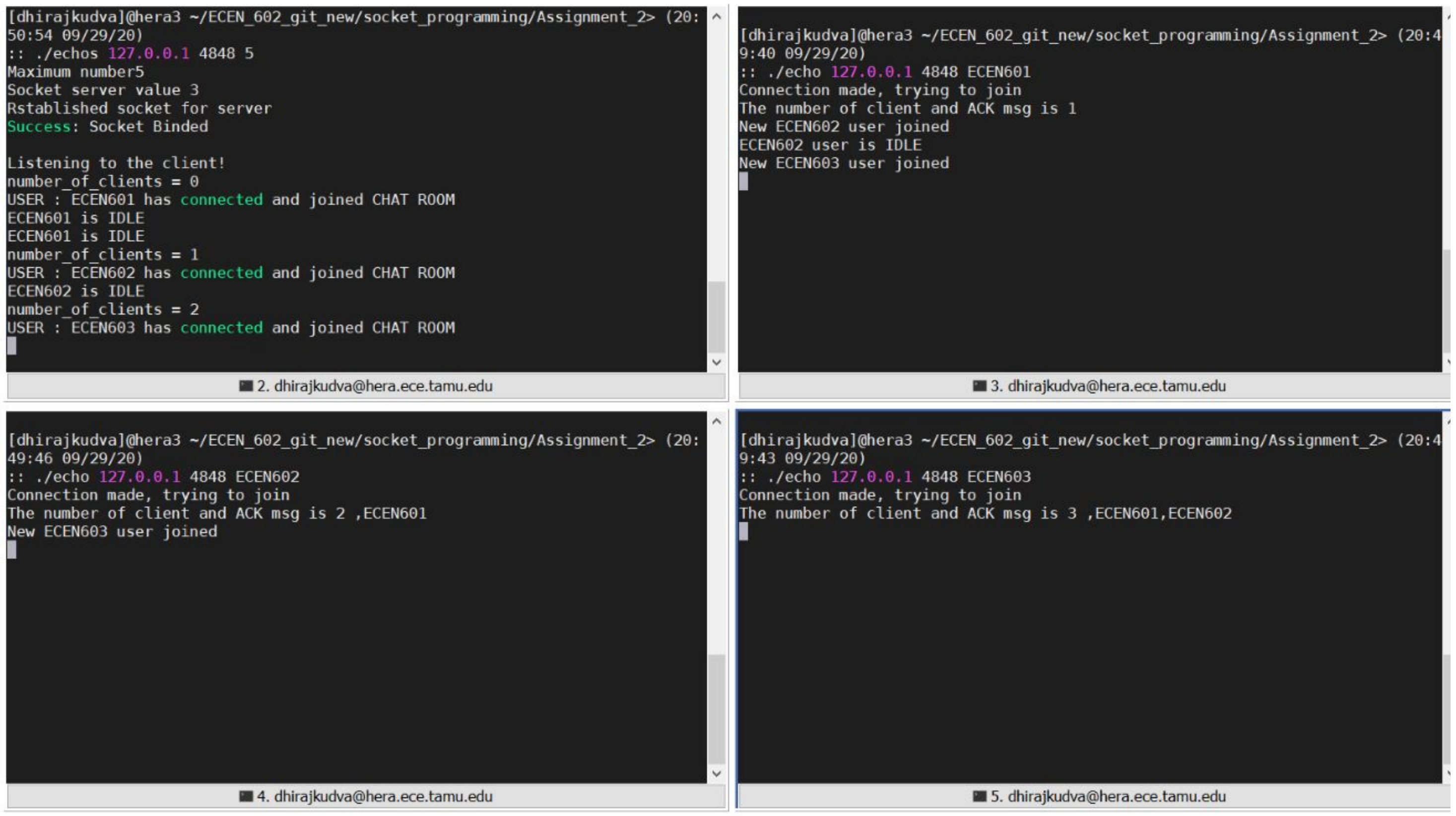
```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (20:
50:54 09/29/20)
:: ./echos 127.0.0.1 4848 5
Maximum number5
Socket server value 3
Rstablished socket for server
Success: Socket Binded

Listening to the client!
number_of_clients = 0
USER : ECEN601 has connected and joined CHAT ROOM
```

Any message typed on the client side will then be displayed on the server side and other connected clients. This can be seen below in the output snapshots of various test cases.

TESTCASES:

(1) normal operation of the chat client with three clients connected

In the below image you can see there are four quadrants. The top right quadrant is the server console and the remaining three correspond to the clients. As you can see that the server displays each connected client and also updates the number of clients. When 1st client is joined it shows the number of clients =0 and then increases by one each time a client is added. As the parameter for maximum number of clients that can be connected is given as 3 while starting the server in the command “./echos 4848 3”. Also the clients get updated about the other clients that are connected after they are joined and the clients which joined before them.



```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (20:
50:54 09/29/20)
:: ./echos 127.0.0.1 4848 5
Maximum number5
Socket server value 3
Rstablished socket for server
Success: Socket Binded

Listening to the client!
number_of_clients = 0
USER : ECEN601 has connected and joined CHAT ROOM
ECEN601 is IDLE
ECEN601 is IDLE
number_of_clients = 1
USER : ECEN602 has connected and joined CHAT ROOM
ECEN602 is IDLE
number_of_clients = 2
USER : ECEN603 has connected and joined CHAT ROOM
```

2. dhirajkudva@hera.ece.tamu.edu

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (20:4
9:40 09/29/20)
:: ./echo 127.0.0.1 4848 ECEN601
Connection made, trying to join
The number of client and ACK msg is 1
New ECEN602 user joined
ECEN602 user is IDLE
New ECEN603 user joined
```

3. dhirajkudva@hera.ece.tamu.edu

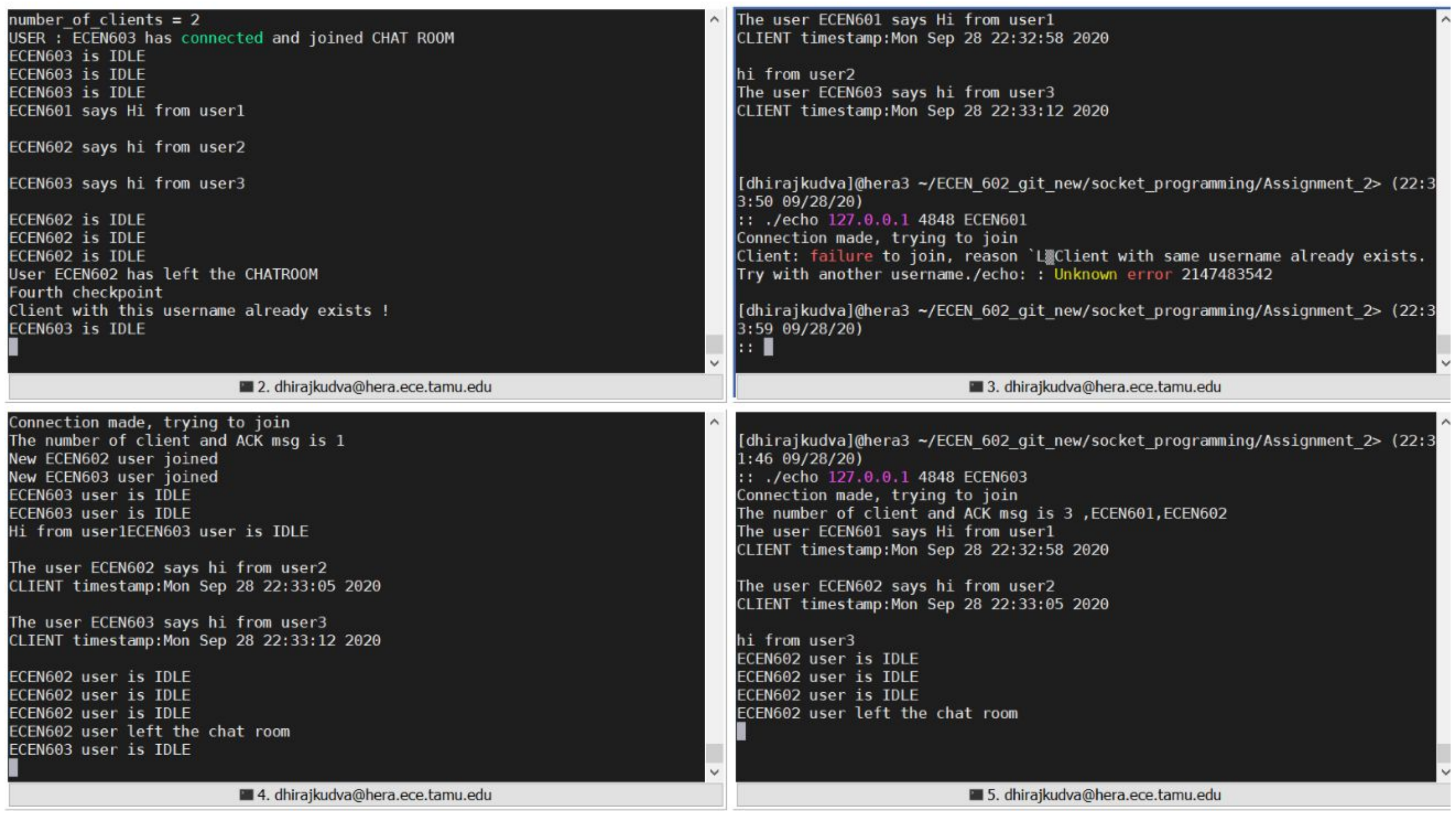
```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (20:
49:46 09/29/20)
:: ./echo 127.0.0.1 4848 ECEN602
Connection made, trying to join
The number of client and ACK msg is 2 ,ECEN601
New ECEN603 user joined
```

4. dhirajkudva@hera.ece.tamu.edu

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (20:4
9:43 09/29/20)
:: ./echo 127.0.0.1 4848 ECEN603
Connection made, trying to join
The number of client and ACK msg is 3 ,ECEN601,ECEN602
```

5. dhirajkudva@hera.ece.tamu.edu

(2) server rejects a client with a duplicate username,
In the below diagram, it can be seen that the client with username ECEN601 already exists. So in the top right quadrant when we try to create another client with same username, it cannot be done. The server rejects the connection, sends a NAK and also displays on its console that the username already exists. So in order to make a connection, all the clients connected to the server should have unique usernames.


2. dhirajkudva@hera.ece.tamu.edu
3. dhirajkudva@hera.ece.tamu.edu
4. dhirajkudva@hera.ece.tamu.edu
5. dhirajkudva@hera.ece.tamu.edu

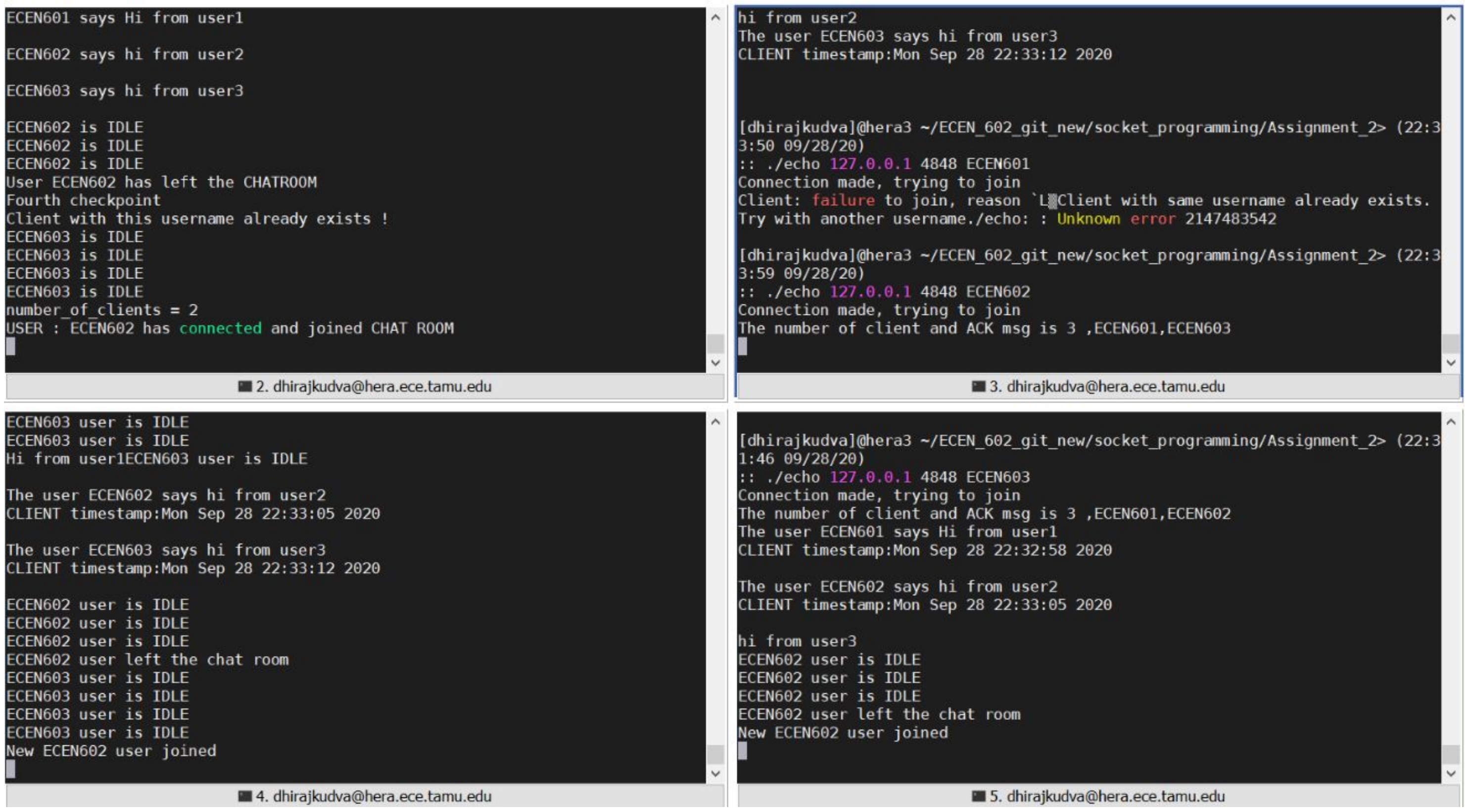
```
number_of_clients = 2
USER : ECEN603 has connected and joined CHAT ROOM
ECEN603 is IDLE
ECEN603 is IDLE
ECEN603 is IDLE
ECEN601 says Hi from user1

ECEN602 says hi from user2

ECEN603 says hi from user3

ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
User ECEN602 has left the CHATROOM
Fourth checkpoint
Client with this username already exists !
ECEN603 is IDLE
```

```
The user ECEN601 says Hi from user1
CLIENT timestamp:Mon Sep 28 22:32:58 2020

hi from user2
The user ECEN603 says hi from user3
CLIENT timestamp:Mon Sep 28 22:33:12 2020



[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:3
3:50 09/28/20)
:: ./echo 127.0.0.1 4848 ECEN601
Connection made, trying to join
Client: failure to join, reason `L▒Client with same username already exists.
Try with another username./echo: : Unknown error 2147483542

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:3
3:59 09/28/20)
::
```

```
Connection made, trying to join
The number of client and ACK msg is 1
New ECEN602 user joined
New ECEN603 user joined
ECEN603 user is IDLE
ECEN603 user is IDLE
Hi from user1ECEN603 user is IDLE

The user ECEN602 says hi from user2
CLIENT timestamp:Mon Sep 28 22:33:05 2020

The user ECEN603 says hi from user3
CLIENT timestamp:Mon Sep 28 22:33:12 2020

ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user left the chat room
ECEN603 user is IDLE
```

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:3
1:46 09/28/20)
:: ./echo 127.0.0.1 4848 ECEN603
Connection made, trying to join
The number of client and ACK msg is 3 ,ECEN601,ECEN602
The user ECEN601 says Hi from user1
CLIENT timestamp:Mon Sep 28 22:32:58 2020

The user ECEN602 says hi from user2
CLIENT timestamp:Mon Sep 28 22:33:05 2020

hi from user3
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user left the chat room
```

(3) server allows a previously used username to be reused,
In the below figure you can see that in the server quadrant, there was a client with username ECEN602 and then the client closed the connection and that client and its username where freed from the server's list of connected clients. So later in the top right quadrant when you want to create another client, that name can be reused and hence a successful connection is established between the client and the server, with reusing the client name.



```
ECEN601 says Hi from user1

ECEN602 says hi from user2

ECEN603 says hi from user3

ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
User ECEN602 has left the CHATROOM
Fourth checkpoint
Client with this username already exists !
ECEN603 is IDLE
ECEN603 is IDLE
ECEN603 is IDLE
ECEN603 is IDLE
number_of_clients = 2
USER : ECEN602 has connected and joined CHAT ROOM
```

2. dhirajkudva@hera.ece.tamu.edu

```
hi from user2
The user ECEN603 says hi from user3
CLIENT timestamp:Mon Sep 28 22:33:12 2020


[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:3
3:50 09/28/20)
:: ./echo 127.0.0.1 4848 ECEN601
Connection made, trying to join
Client: failure to join, reason `L▒Client with same username already exists.
Try with another username./echo: : Unknown error 2147483542

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:3
3:59 09/28/20)
:: ./echo 127.0.0.1 4848 ECEN602
Connection made, trying to join
The number of client and ACK msg is 3 ,ECEN601,ECEN603
```

3. dhirajkudva@hera.ece.tamu.edu

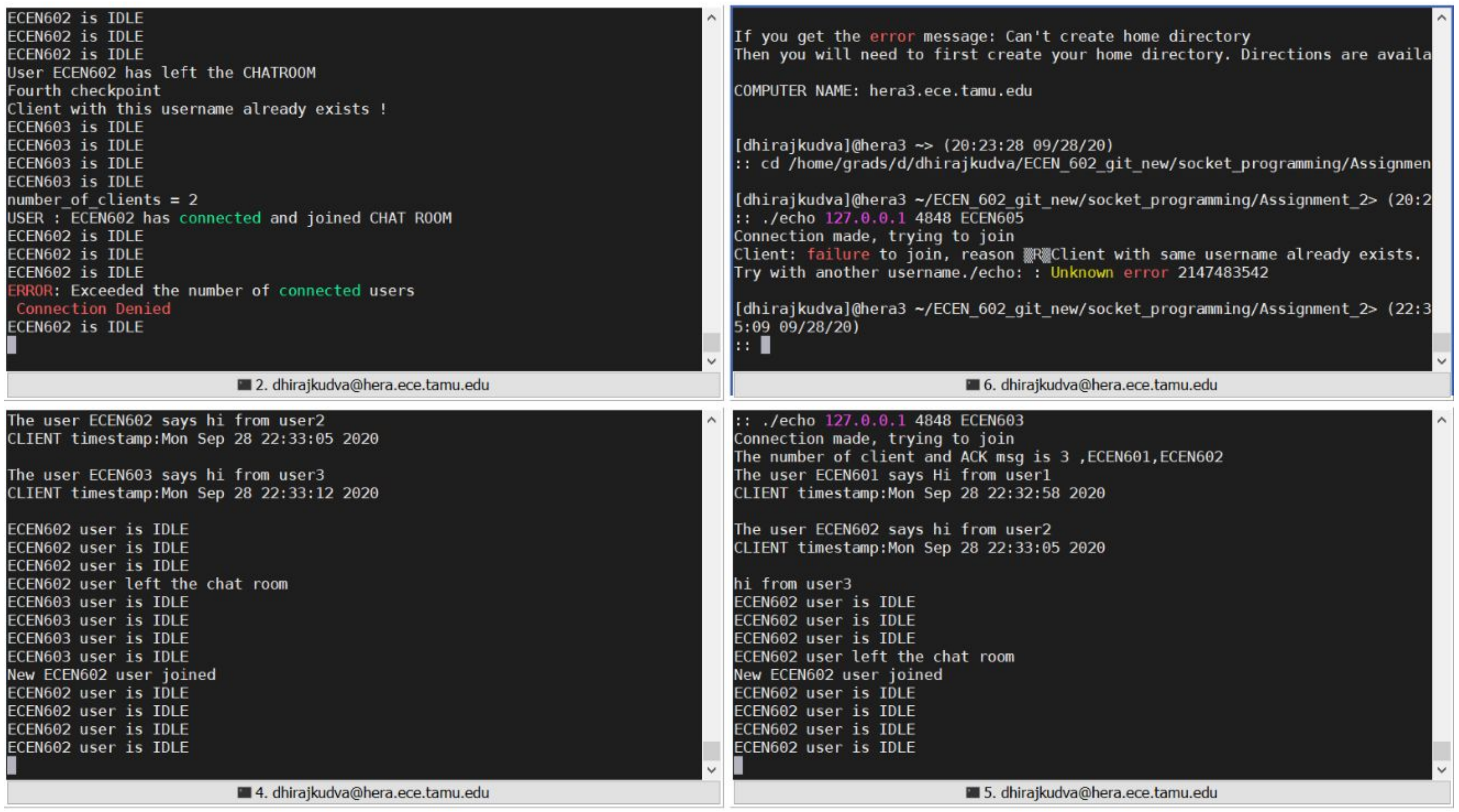
```
ECEN603 user is IDLE
ECEN603 user is IDLE
Hi from user1ECEN603 user is IDLE

The user ECEN602 says hi from user2
CLIENT timestamp:Mon Sep 28 22:33:05 2020

The user ECEN603 says hi from user3
CLIENT timestamp:Mon Sep 28 22:33:12 2020

ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user left the chat room
ECEN603 user is IDLE
ECEN603 user is IDLE
ECEN603 user is IDLE
ECEN603 user is IDLE
New ECEN602 user joined
```

4. dhirajkudva@hera.ece.tamu.edu

```

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:3
1:46 09/28/20)
:: ./echo 127.0.0.1 4848 ECEN603
Connection made, trying to join
The number of client and ACK msg is 3 ,ECEN601,ECEN602
The user ECEN601 says Hi from user1
CLIENT timestamp:Mon Sep 28 22:32:58 2020

The user ECEN602 says hi from user2
CLIENT timestamp:Mon Sep 28 22:33:05 2020

hi from user3
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user left the chat room
New ECEN602 user joined
```

5. dhirajkudva@hera.ece.tamu.edu

(4) server rejects the client because it exceeds the maximum number of clients allowed,
In the below figure it can be seen that the server is already connected with three clients and 3 is the maximum number of clients value that is passed while creating the server at the start. Hence, when fourth client with username ECEN605 is created for connection with the server, the server rejects the client stating an error that the number of connected users is exceeded and connection is denied.



```
ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
User ECEN602 has left the CHATROOM
Fourth checkpoint
Client with this username already exists !
ECEN603 is IDLE
ECEN603 is IDLE
ECEN603 is IDLE
ECEN603 is IDLE
number_of_clients = 2
USER : ECEN602 has connected and joined CHAT ROOM
ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
ERROR: Exceeded the number of connected users
 Connection Denied
ECEN602 is IDLE
```

2. dhirajkudva@hera.ece.tamu.edu

```
If you get the error message: Can't create home directory
Then you will need to first create your home directory. Directions are availa

COMPUTER NAME: hera3.ece.tamu.edu


[dhirajkudva]@hera3 ~> (20:23:28 09/28/20)
:: cd /home/grads/d/dhirajkudva/ECEN_602_git_new/socket_programming/Assignmen

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (20:2
:: ./echo 127.0.0.1 4848 ECEN605
Connection made, trying to join
Client: failure to join, reason ▒R▒Client with same username already exists.
Try with another username./echo: : Unknown error 2147483542

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:3
5:09 09/28/20)
::
```

6. dhirajkudva@hera.ece.tamu.edu

```
The user ECEN602 says hi from user2
CLIENT timestamp:Mon Sep 28 22:33:05 2020

The user ECEN603 says hi from user3
CLIENT timestamp:Mon Sep 28 22:33:12 2020

ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user left the chat room
ECEN603 user is IDLE
ECEN603 user is IDLE
ECEN603 user is IDLE
ECEN603 user is IDLE
New ECEN602 user joined
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
```

4. dhirajkudva@hera.ece.tamu.edu

```
:: ./echo 127.0.0.1 4848 ECEN603
Connection made, trying to join
The number of client and ACK msg is 3 ,ECEN601,ECEN602
The user ECEN601 says Hi from user1
CLIENT timestamp:Mon Sep 28 22:32:58 2020

The user ECEN602 says hi from user2
CLIENT timestamp:Mon Sep 28 22:33:05 2020

hi from user3
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user left the chat room
New ECEN602 user joined
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
```

5. dhirajkudva@hera.ece.tamu.edu

(5) BONUS FEATURES

a. IPv4 and IPv6
The above snapshots were generated using IPv4 addressing as the local host address was an IPv4 address. Hence, IPv6 could not be checked.
But the code to support IPv6 is included in the client.C, where we read the IP address, gets the type of address using “getaddrinfo” and based on this the socket is formatted with the correct address type.

b. ACK,NAK,OFFLINE and ONLINE.

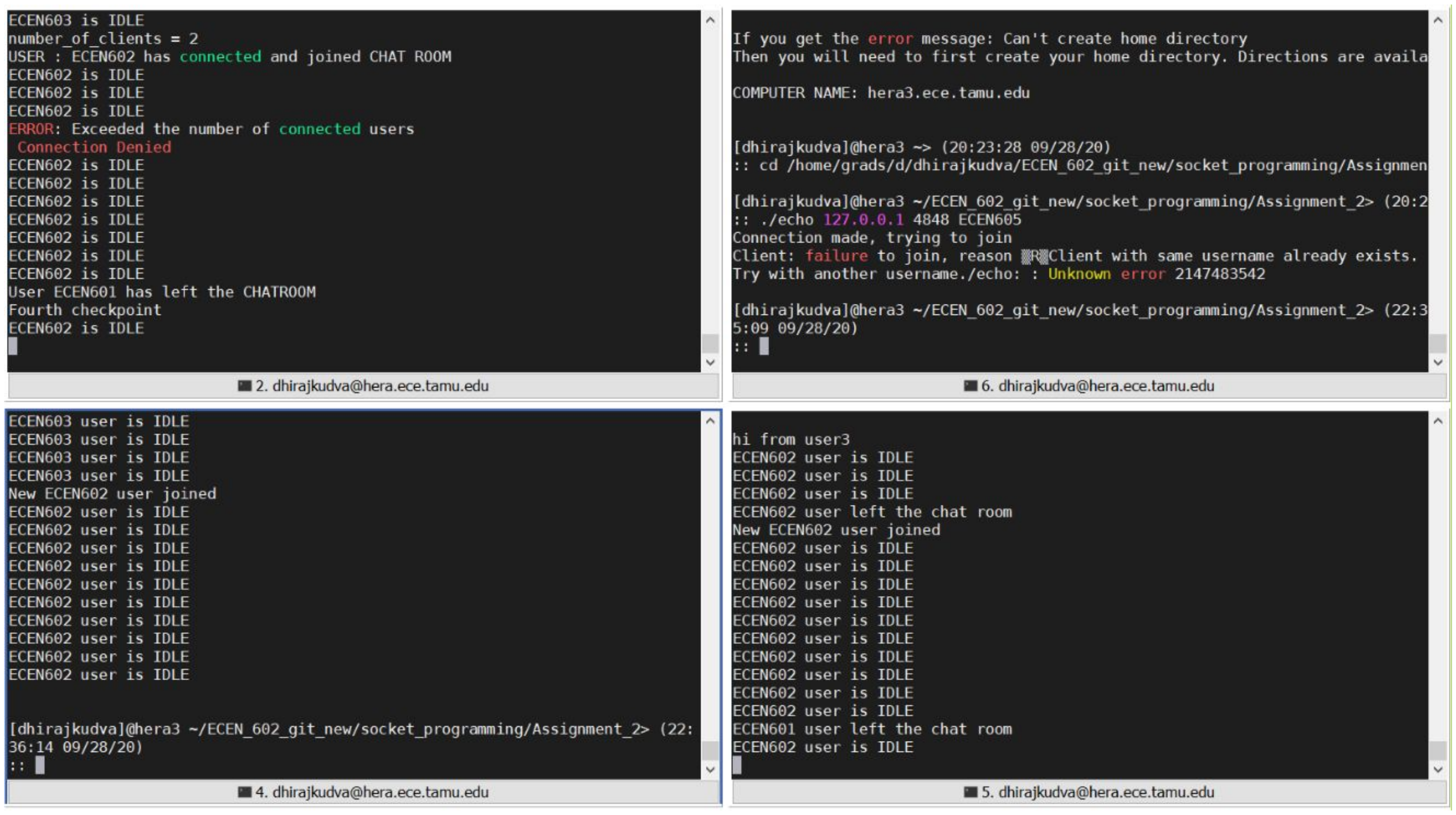
In the below figure, it can be seen that all the four type of bonus messages are implemented.

ACK - acknowledge message is sent when a client joins. And other clients are notified about the user joined and also the new client is provided information about the clients that are already registered.

NAK- this is sent when the client cannot be connected to the server : either because of same username or more than the maximum number of clients that can be registered.

OFFLINE: When a client leaves the server, it is notified to other clients by the server that the client is offline. Also the user has left the room is also displayed on the server.

ONLINE : When the client joins, the message that the client is available and in the chat room is also displayed on the server and other clients.



```
ECEN603 is IDLE
number_of_clients = 2
USER : ECEN602 has connected and joined CHAT ROOM
ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
ERROR: Exceeded the number of connected users
 Connection Denied
ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
ECEN602 is IDLE
User ECEN601 has left the CHATROOM
Fourth checkpoint
ECEN602 is IDLE
```

2. dhirajkudva@hera.ece.tamu.edu

```
If you get the error message: Can't create home directory
Then you will need to first create your home directory. Directions are availa

COMPUTER NAME: hera3.ece.tamu.edu


[dhirajkudva]@hera3 ~> (20:23:28 09/28/20)
:: cd /home/grads/d/dhirajkudva/ECEN_602_git_new/socket_programming/Assignmen

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (20:2
:: ./echo 127.0.0.1 4848 ECEN605
Connection made, trying to join
Client: failure to join, reason ▒R▒Client with same username already exists.
Try with another username./echo: : Unknown error 2147483542

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:3
5:09 09/28/20)
::
```

6. dhirajkudva@hera.ece.tamu.edu

```
ECEN603 user is IDLE
ECEN603 user is IDLE
ECEN603 user is IDLE
ECEN603 user is IDLE
New ECEN602 user joined
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE


[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_2> (22:
36:14 09/28/20)
::
```

4. dhirajkudva@hera.ece.tamu.edu

```
hi from user3
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user left the chat room
New ECEN602 user joined
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN602 user is IDLE
ECEN601 user left the chat room
ECEN602 user is IDLE
```

5. dhirajkudva@hera.ece.tamu.edu

c. IDLE message.

In the above snapshot, it can be seen that whichever client is IDLE, its username is displayed on the server and other clients stating that the particular client is IDLE. This is checked at interval of 10 seconds.

CODE:

Client.c

```
/**************************client.c**********************/
//Author: Sanket Agarwal and Dhiraj Kudva
//Organisation: Texas A&M University
//Description: Client code of the Simple Broadcast Chat Server

//header files required

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/un.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>
#include <netinet/in.h>
#include <time.h>
#include <stdbool.h>

//Simple Broadcast Chat server structures.
#include "common_def.h"
time_t local_time;

//connection is made, time to join the chat room

void join_chat (int socket_descriptor, char * argv[]){

       struct simple_broadcast_chat_server_header header;
       struct simple_broadcast_chat_server_attribute attribute;
       struct simple_broadcast_chat_server_message message;

       int join_status = 0;

       header.version = '3'; // protocol version as defined in the mannual
       header.type    = '2'; //join request header

       attribute.type = 2;//sending the username
       attribute.length = strlen(argv[3]) + 1; //length of username + null char
       strcpy(attribute.payload_data, argv[3]); // copy the username

       message.header = header; //encapsulate
       message.attribute[0] = attribute;// just one attribute for joining.
```

```
        write(socket_descriptor,(void *)&message, sizeof(message));

        join_status = read_server_message(socket_descriptor);

        if (join_status==1)
              {system_error("username already present.");
               close(socket_descriptor);}

}

//read server message

int read_server_message(int socket_descriptor){

        struct simple_broadcast_chat_server_message server_message;
        int i;
        int return_status = 0;
        int number_of_bytes = 0;

// reading the bytes from the server.
        number_of_bytes = read(socket_descriptor,(struct
simple_broadcast_chat_server_message*)& server_message, sizeof(server_message));

        int size_of_payload = 0;//will be used to check the size of the payload
received.

        // forward_message

        //check the username, compare the actual length with the length specified in
the header, check the header type with the attribute index. If all are correct print
the message.

        if (server_message.header.type==3){// forward message


if((server_message.attribute[0].payload_data!=NULL||server_message.attribute[0].payloa
d_data!='\0')

&&(server_message.attribute[1].payload_data!=NULL||server_message.attribute[1].payload
_data!='\0')
               &&(server_message.attribute[0].type==4)
               &&(server_message.attribute[1].type==2)){
              //checking the size of the payload matches the payload length specified.
                    for(i=0; i<sizeof(server_message.attribute[0].payload_data);i++){
                          if
(server_message.attribute[0].payload_data[i]=='\0'){//end of string found
                                size_of_payload = i-1;
                                break;
                          }
                    }//end for
                    if(size_of_payload==server_message.attribute[0].length){

                            printf("The user %s says 
%s",server_message.attribute[1].payload_data,
                                   server_message.attribute[0].payload_data);
                            local_time = time (NULL);

                            printf("CLIENT timestamp:%s 
\n",asctime(localtime(&local_time)));

                    }

                    else system_error("length of payload mismatch at client \n");
             }//mismatch of the any data such as payload data type or username data 
type
             else system_error("CLIENT: header type mismatch or null recevied\n");

             return_status = 0; //sucess/
      }//if (server_message.header.type)


      //if the server sends a NACK
      if(server_message.header.type ==5){

if((server_message.attribute[0].payload_data!=NULL||server_message.attribute[0].payloa
d_data!='\0')
               &&(server_message.attribute[0].type==1)){// indicating the reason of 
failure
                    printf("Client: failure to join, reason 
%s",server_message.attribute[0].payload_data);
             }
             return_status = 1;
      }// if (server_message.header.type)

//offline message received from the server.
      if (server_message.header.type==6){

             if 
((server_message.attribute[0].payload_data!=NULL||server_message.attribute[0].payload_
data!='\0')
                &&server_message.attribute[0].type ==2){ //i.e.sending the username 
that left the chat

                    printf("%s user left the chat 
room\n",server_message.attribute[0].payload_data);
             }
      return_status =0;//successfully read.
      }
```

```
//ACK with the client count
        if (server_message.header.type==7){
                if
((server_message.attribute[0].payload_data!=NULL||server_message.attribute[0].payload_
data!='\0')             &&server_message.attribute[0].type ==4){
                    printf("The number of client and ACK msg is
%s\n",server_message.attribute[0].payload_data);
                }
        return_status =0;
        }

//new chat participant has arrived.
        if (server_message.header.type ==8){
                if
((server_message.attribute[0].payload_data!=NULL||server_message.attribute[0].payload_
data!='\0')
                &&server_message.attribute[0].type ==2){ //i.e.sending the username
that joinded
                        printf("New %s user joined
\n",server_message.attribute[0].payload_data);
                }

        return_status = 0;
        }

        if (server_message.header.type ==9)
        {
                if
((server_message.attribute[1].payload_data!=NULL||server_message.attribute[1].payload_
data!='\0')
                &&server_message.attribute[1].type ==2){ //i.e.sending the username
that joinded
                        printf("%s user is IDLE
\n",server_message.attribute[1].payload_data);
                }

        return_status = 0;
        }



        return return_status;
}



void send_to_server(int socket_descriptor, bool timeout)
{
        struct simple_broadcast_chat_server_header header;
        header.version = '3';
```

```
	header.type    = '4';//as defined in the manual.

	struct simple_broadcast_chat_server_message message;
	struct simple_broadcast_chat_server_attribute attribute;


	message.header = header;// copying the header to the message header.

	int number_of_bytes_read_from_user = 0;
	char temp_message_holder[512];
//	char *pointer = temp_message_holder;
	struct timeval wait_time_to_send;
	fd_set read_file_descriptor;
	if (timeout == true)
	{
		message.header.type = 9;
		attribute.type =4;//idle message
		char idle_array[29]  = "I am IDLE please talk to me.";
		idle_array[29] = '\0';
		strcpy(attribute.payload_data, idle_array);
		message.attribute[0] = attribute;
		message.attribute[0].length = 28; // the length of the string above.
	}
	else
	{

	wait_time_to_send.tv_sec = 2;
	wait_time_to_send.tv_usec = 0;

	FD_ZERO(&read_file_descriptor); //clearing the read descriptor
	FD_SET(STDIN_FILENO, &read_file_descriptor);// set to read from the input

	select(STDIN_FILENO+1, &read_file_descriptor, NULL, NULL, &wait_time_to_send);

	if(FD_ISSET(STDIN_FILENO, &read_file_descriptor)){
		number_of_bytes_read_from_user = read(STDIN_FILENO,temp_message_holder,
sizeof(temp_message_holder));

		if (number_of_bytes_read_from_user >0)
			temp_message_holder[number_of_bytes_read_from_user] = '\0';


	attribute.type = 4;//message as specified in the mannual
	strcpy(attribute.payload_data, temp_message_holder);// copy the message to
payload
	message.attribute[0] = attribute;
	message.attribute[0].length = number_of_bytes_read_from_user -1 ; //excluding
the extra read char
	}else {
		printf("CLIENT:Timeout occrued \n");
	}

	}
	write(socket_descriptor, (void *)&message, sizeof(message));


}
int main (int argc, char*argv[]){

	//int idle_count = 0;
	struct timeval idle_count;
	char *username, *IPaddress;
	int select_return_value =0;
	if (argc!=4)
	{
		printf("CLIENT:USAGE:./echo <IP_address> <port_number> <user_name> \n");
		system_error("Please specify the right arguments as above");
	}

	 // Wwould work for both IPv4 AND IPv6

	char * p; // used to point to an array if not converted,Same as MP1
	//server add.
	int port_number = strtol(argv[2],&p,10);

//	struct hostent* IP =  gethostbyname(argv[1]);//IP address

	struct sockaddr_in server_address;
	struct sockaddr_in6 server_address_6; // IPv6 address
	struct addrinfo check , *get_addr_info=NULL; // will be used to check ipv4 vs
6.
	int return_value_address_resolution;
	int inet_return;
	int socket_descriptor;

	IPaddress = argv[1];
	username = argv[3];

	memset(&check, '\0', sizeof check);

	check.ai_family = PF_UNSPEC;
	check.ai_flags = AI_NUMERICHOST;

// get the inforamtion about the address being ipv4 or ipv6
	return_value_address_resolution = getaddrinfo(IPaddress, NULL, &check,
&get_addr_info);

	if (return_value_address_resolution)
	{
```

```
              system_error ("invalid address");
       }

       if (get_addr_info->ai_family == AF_INET)// ipv4 address
       {
        socket_descriptor = socket(AF_INET, SOCK_STREAM, 0);

       bzero(&server_address, sizeof(server_address)); // same as MP 1
       server_address.sin_family = AF_INET; //IPv4
       server_address.sin_port   = htons(port_number); // port number as MP1
       //memcpy(&server_address.sin_addr.s_addr, IP->h_addr, IP->h_length);
       inet_return = inet_pton(AF_INET, IPaddress, &(server_address.sin_addr));
       //connec to the server/or we can say the chatroom.
       int connect_status = connect(socket_descriptor, (struct sockaddr
*)&server_address, sizeof(server_address));
       if (connect_status < 0)//error
              system_error("Error connecting to the server");

       printf("Connection made, trying to join\n");
       }

       else if(get_addr_info->ai_family == AF_INET6)
       {
        socket_descriptor = socket(AF_INET6, SOCK_STREAM,0);
       bzero(&server_address_6, sizeof(server_address_6));
       server_address_6.sin6_family = AF_INET6;
       server_address_6.sin6_port = htons(port_number);
       inet_pton(AF_INET6, IPaddress,&server_address_6.sin6_addr);

       int connect_status6 = connect(socket_descriptor, (struct
sockaddr*)&server_address_6, sizeof(server_address_6));

       if (connect_status6<0)
              system_error("Client:ipv6 cant connect");
       }

       else
       {
              system_error("Invalid address.");
       }

       freeaddrinfo(get_addr_info);

       //adding file descriptor to the select.
       fd_set main_file_descriptor;
       fd_set read_file_descriptor;

       //clearing them and setting to zero.
       FD_ZERO(&read_file_descriptor);
       FD_ZERO(&main_file_descriptor);
```

```
	join_chat(socket_descriptor, argv);

	FD_SET(socket_descriptor, &main_file_descriptor);// to see any input on the 
socket line
	FD_SET(STDIN_FILENO, &main_file_descriptor);// to see any input on the command 
line


	while (1){

	read_file_descriptor = main_file_descriptor;

	idle_count.tv_sec =10;// waitin for 10 secs or the readfiledescriptor to wakeup

	if ((select_return_value = select(socket_descriptor+1, 
&read_file_descriptor,NULL,NULL,&idle_count))==-1)
		system_error("CLIENT:SELECT error");

	if (select_return_value == 0)// idle time of 10 secs have passed.
	{
		send_to_server(socket_descriptor, 1);
	}

	if (FD_ISSET(socket_descriptor,&read_file_descriptor))//read from the socket
		read_server_message(socket_descriptor);

	if (FD_ISSET(STDIN_FILENO, &read_file_descriptor))//read from commadn line send 
to the server
		send_to_server(socket_descriptor, 0 );
	}



	printf("The user left, end of chat\n");
	printf("Closing client");
	return 0;

}
```

SERVER.c

```
#include <string.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// #include <sstream.h>
#include <sys/time.h>
#include <sys/socket.h>
```

```
#include <sys/types.h>
#include <sys/un.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>
#include <netinet/in.h>


//Simple Broadcast Chat server structures.
#include "common_def.h"

#define max_string_size 256

int number_of_clients = 0;

//initializing clients structure to store client information
struct simple_broadcast_chat_server_client_user_information *clients;

//ACK function: To send acknowledge signal to client
void ACK_server_client (int fd_client){
       struct simple_broadcast_chat_server_message ack_message;
       char temp_string [256];

       //ACK header format
       ack_message.header.version = 3; //mandatory header type FWD
       ack_message.header.type = 7;
       ack_message.attribute[0].type = 4;

       snprintf(temp_string,5,"%d%s",number_of_clients," ");
       int k = strlen(temp_string);
       int i;
       for (i=0; i < number_of_clients-1; i++)
       {
              strcat(temp_string,",");
              strcat(temp_string,clients[i].username);
       }
       ack_message.attribute[0].length = strlen(temp_string) + 1;
       strcpy (ack_message.attribute[0].payload_data, temp_string);

       write (fd_client,(void *)&ack_message, sizeof(ack_message));
}

void NACK_server_client(int fd_client)
{
       struct simple_broadcast_chat_server_message nack_message;
       char temp_string[256];

       nack_message.header.version = 3;
       nack_message.header.type = 5;
```

```
        nack_message.attribute[0].type = 1;

        strcat(temp_string,"Client with same username already exists. Try with another
username");
        nack_message.attribute[0].length = strlen(temp_string);
        strcpy(nack_message.attribute[0].payload_data, temp_string);

        write(fd_client,(void *) &nack_message,sizeof(nack_message));

        close(fd_client);
}

//checks clients username:
//if username exists, returns true
//else returns false
int check_username (char username[]){
        int i;
        for (i =0 ; i < number_of_clients; i++)
        {
                if(strcmp (username, clients[i].username) == 0)
                {
                        return 1; //username exists
                }
        }
        return 0; //username doesnot exist
}

//checks if the client has joined or not
int client_join_check(int fd_client){
        struct simple_broadcast_chat_server_message join_message;
        struct simple_broadcast_chat_server_attribute join_message_attribute;
        char temp_string [16];
        read(fd_client,(struct simple_broadcast_chat_server_message
*)&join_message,sizeof(join_message));
        join_message_attribute = join_message.attribute[0];
        strcpy(temp_string,join_message_attribute.payload_data);

        if(check_username(temp_string)){
                printf("%s\n","Client with this username already exists !" );
                NACK_server_client(fd_client);
                return 1;
        }
        else{
                strcpy(clients[number_of_clients].username, temp_string);
         printf("number_of_clients = %d\n",number_of_clients);
                clients[number_of_clients].file_descriptor = fd_client;
                clients[number_of_clients].client_count = number_of_clients;
                number_of_clients ++;
                ACK_server_client(fd_client);
```

```
	}
	return 0;
}


int main(int argc, char*argv[])
{
	struct simple_broadcast_chat_server_message receive_message, forward_message,
join_broadcast_message, leave_broadcast_message,idle_message;
	struct simple_broadcast_chat_server_attribute client_attribute;

    if (argc!=4)
    {
        printf("CLIENT:USAGE:./echos <IP_address> <port_number> <number_of_clients>
\n");
        system_error("Please specify the right arguments as above");
    }

    // printf("Here it is\n");
	//Server's address information
	struct sockaddr_in server_address, *clients_address;
    server_address.sin_family = AF_INET;
    // printf("First chec\n");
    // server_address.sin_addr.s_addr = inet_addr(INADDR_ANY);
    // printf("Second checpoint \n");
    server_address.sin_port = htons(atoi(argv[2]));
    socklen_t server_addr_size = sizeof(server_address);

    int maximum_number_of_clients = atoi(argv[3]);
    printf("Maximum number%d\n", maximum_number_of_clients);

    int server_status;
    struct addrinfo addr_hints;
    struct addrinfo *server_info; //point to the results
    memset(&addr_hints,0,sizeof addr_hints); //creating an empty structure

    addr_hints.ai_family = AF_UNSPEC; //don;t care IPv4 OR IPv6
    addr_hints.ai_socktype = SOCK_STREAM; //TCP sockets

    // if((server_status = getaddrinfo(argv[1],&addr_hints, &server_info))!=0){
    //       fprintf(stderr, "ERROR: getaddrinfo : %s\n",gai_strerror(server_status)
);
    //       exit(1);
    // }

    fd_set fd_master; //main master set of file descriptor
    fd_set temp_fd; //temporary file decriptor list
    FD_ZERO (&fd_master); //resetting all entries in the temp and master file
descriptors
    FD_ZERO (&temp_fd);
    int client_new = 0; //count for newly accepted socket descriptor
    struct sockaddr_in addr_client; //client's address
    // addr_client.sin_addr.s_addr = htons(INADDR_ANY);
    addr_client.sin_family = AF_INET;
    /* we don't want to bind the server socket to a specific IP.
    Basically during bind, we want to accept connections to all IPs*/
    addr_client.sin_addr.s_addr = htons(INADDR_ANY);
    addr_client.sin_port = htons (atoi(argv[2]));
    int number_of_bytes = 0; //number of bytes received
    // printf("Third checpoint\n");
    //socket initialization
    // int socket_server =
socket((*server_info).ai_family,(*server_info).ai_socktype,(*server_info).ai_protocol)
;
    int socket_server = socket(AF_INET, SOCK_STREAM,0);
    printf("Socket server value %d\n",socket_server );
    if(socket_server < 0){
      printf("%s\n","Failed to establish connection between client and server" );
      system_error("Trying to establish");
      exit (0);
    }

    int temp_reuse = 1;
    if (setsockopt(socket_server,SOL_SOCKET,SO_REUSEADDR,&temp_reuse,sizeof(int))<0){
      printf("%s\n","Failed : setsockopt(SO_REUSEADDR)" );
    }
    //CHECK THIS CASE: WHEN should exactly the below message needs to be displayed

    printf("Rstablished socket for server\n");

    //binding
    // if(bind(socket_server,(*server_info).ai_addr, (*server_info).ai_addrlen) < 0){
    if (bind(socket_server, (struct sockaddr *) &addr_client ,sizeof(addr_client)) <
0)
    {
      printf("Failed in socket binding");
      system_error("BINDING");
      exit(0);
    }

    printf("%s\n","Success: Socket Binded\n" );

    clients = (struct simple_broadcast_chat_server_client_user_information
*)malloc(maximum_number_of_clients*sizeof(struct
simple_broadcast_chat_server_client_user_information));
    clients_address = (struct sockaddr_in
*)malloc(maximum_number_of_clients*sizeof(struct sockaddr_in));

    //listening phase
    if(listen (socket_server,10) < 0) {
      printf("Fail: To client found\n");
      system_error("LISTEN");
      exit(0);
    }
    printf("Listening to the client!\n" );

    //select - waiting for events
    FD_SET(socket_server, &fd_master); //add server socket to master set
    //keeping count of the file descriptors
    int max_fd = socket_server; //number of file descriptors should be highest file +1
    // printf("Max FD= %d\n",max_fd);
    int temp;

    while(1){
      temp_fd = fd_master;
      if (select(max_fd + 1,&temp_fd, NULL, NULL,NULL) == -1){
            printf("%s\n","Error occured when selecting \n" );
            system_error("SELECT:");
            exit(0);
      }
      int i;
      for ( i=0; i<=max_fd; i++){ //looping through the file descriptors
           // printf("Max FD = %d\n",max_fd);
            if(FD_ISSET(i,&temp_fd)){ //taking one from existing file descriptors
                  if(i == socket_server){ //if this is server soket , then check for
clients that wants to connect or send any message
                        //Accept and the address of the new client in the array
                        socklen_t size_client_address =
sizeof(clients_address[number_of_clients]);
                        client_new = accept(socket_server, (struct sockaddr
*)&clients_address[number_of_clients], &size_client_address );
                  // client_new = accept(socket_server,(struct sockaddr*)NULL,
NULL);
                        if(client_new == -1){
                              printf("ERROR : Occured when accepting new client \n
Error Number%d\n",(int)errno );
                        }else{
                              temp = max_fd;
                              FD_SET(client_new, &fd_master); //Adding the new
client/connection to the connected clients' list

                              //updating the maximum number of file descriptors
                              if(client_new > max_fd){
                                    max_fd = client_new;
                              }
                              if(number_of_clients + 1 >
maximum_number_of_clients){
                                    printf("ERROR: Exceeded the number of
connected users\n Connection Denied \n");
```

```
                                        max_fd = temp;
                                        FD_CLR(client_new, & fd_master);
                                        NACK_server_client(client_new);
                                }else{
                                        if(client_join_check(client_new) == 0){
                                                //new online client
                                                //broadcast this information to other
user
                                                printf("USER : %s has connected and
joined CHAT ROOM\n", clients[number_of_clients-1].username);
                                                join_broadcast_message.header.version =
3;
                                                join_broadcast_message.header.type = 8;

join_broadcast_message.attribute[0].type = 2;

strcpy(join_broadcast_message.attribute[0].payload_data,
clients[number_of_clients-1].username)  ;
                                                //go through the file descriptor list
and except the new client and server, broadcast the
                                                //information of new client to
everybody
                                                int j;
                                                for(j=0; j<=max_fd;j++){ //loop again
the file descriptors and broadcast
                                                        if(FD_ISSET(j, &fd_master))
                                                        {
                                                                if(j != socket_server && j
!= client_new){
                                                                        if((write(j,(void
*)&join_broadcast_message,sizeof(join_broadcast_message))) == -1){

system_error("Error while broadcasting JOIN message");
                                                                        }
                                                                }
                                                        }


                                                }
                                        } else{
                                                max_fd = temp;
                                                FD_CLR(client_new, &fd_master); //clear
newclient if username is already used and hence not available
                                        }
                                }
                        }
                }else{
                                //data from existing connection
                                number_of_bytes = read(i, (struct
simple_broadcast_chat_server_message *)&receive_message,sizeof(receive_message));
                        //printf("TYPE HELLO = %d\n",receive_message.header.type );
                        if(receive_message.header.type == 9){
                            // printf("Entered IDLE message\n");
                            client_attribute = receive_message.attribute[0]; //gets
message
                        idle_message = receive_message;
                        idle_message.header.type = 9;
                        idle_message.attribute[1].type =2;
                        //////check this line below : I thing needs to be changed to
attribute[1]
                        // idle_message.attribute[0].length =
receive_message.attribute[0].length;
                        char name[16];
                        strcpy(name,receive_message.attribute[1].payload_data);

                        int k;
                        for(k=0; k<number_of_clients;k++){
                            if(clients[k].file_descriptor == i){

strcpy(idle_message.attribute[1].payload_data,clients[k].username);
                            }
                        }
                        printf("%s is IDLE\n",idle_message.attribute[1].payload_data);

                        //Forward the message to all the clients except the current
one and server
                        int j;
                        for (j=0; j<=max_fd; j++){
                            //send forward message
                              receive_message.header.type = 69;
                            if(FD_ISSET(j , &fd_master)){
                                if(j!=socket_server && j!=i){
                                    //CHECK: I guess the condition should be fals
                                    if((write(j,
(void*)&idle_message,number_of_bytes))==-1){
                                        system_error("Forwarding IDLE message");
                                    }
                                }
                            }
                        }



                        }
                        else{
                        if(number_of_bytes <= 0){

                                        if(number_of_bytes == 0){
                                               int k,flag=0,x;
                                               for(k=0; k < number_of_clients; k++){
                                                    if(clients[k].file_descriptor ==
i){

leave_broadcast_message.attribute[0].type = 2;

strcpy(leave_broadcast_message.attribute[0].payload_data,clients[k].username);
                                        flag = 1;
                                                    }

                                    // printf("Flag value = %d\n",flag );
                                    if(flag == 1){
                                        // printf("Inside Flag\n" );
                                        for(x=k; x<(number_of_clients-1); x++){
                                            clients[x] = clients[x+1];
                                            // printf("Client k username sarkar =
%s\n",clients[x].username );
                                        }
                                        number_of_clients--;
                                        flag=0;
                                    }
                                              }
                                              printf("User %s has left the
CHATROOM\n", leave_broadcast_message.attribute[0].payload_data );
                                              leave_broadcast_message.header.version
= 3;
                                              leave_broadcast_message.header.type =
6;
                                              int j;
                                              for (j = 0; j <=(max_fd); j++){
                                                    if(FD_ISSET(j,&fd_master)){
                                                           if(j!=socket_server){

if((write(j,(void*)&leave_broadcast_message,sizeof(leave_broadcast_message))) == -1){

system_error("BROADCASTING LEAVE MESSAGE");
                                                                  }
                                                           }
                                                    }
                                              }
                                  // max_fd -= 1;
                                  // printf("Fourth checkpoint\n");
                                 }else if(number_of_bytes < 0){
                                        // system_error("RECEIVING MESSAGE, WAITING");
                           printf("RECEIVING MESSAGE, WAITING\n");
                                 }
                                 close(i);
                                 FD_CLR(i, &fd_master); //client is removed from the
master set of connected clients
                                 // int x;
                                 // for(x=i; x<number_of_clients; x++){
```

```
                                 //     clients[x] = clients[x+1];

                                 // }
                                 // number_of_clients--;
                          }else{
                                 //number_of_bytes > 0
                                 client_attribute = receive_message.attribute[0];
//gets message
                                 forward_message = receive_message;
                                 forward_message.header.type = 3;
                                 forward_message.attribute[1].type =2;
                                 //////check this line below : I thing needs to be
changed to attribute[1]
                                 forward_message.attribute[0].length =
receive_message.attribute[0].length;
                                 char name[16];

strcpy(name,receive_message.attribute[1].payload_data);

                                 int k;
                                 for(k=0; k<number_of_clients;k++){
                                        if(clients[k].file_descriptor == i){

strcpy(forward_message.attribute[1].payload_data,clients[k].username);
                                        }
                                 }
                                 printf("%s says
%s\n",forward_message.attribute[1].payload_data,
forward_message.attribute[0].payload_data );

                                 //Forward the message to all the clients except the
current one and server
                                 int j;
                                 for (j=0; j<=max_fd; j++){
                                        //send forward message
                                        if(FD_ISSET(j , &fd_master)){
                                               if(j!=socket_server && j!=i){
                                                      //CHECK: I guess the condition
should be false
                                                      if((write(j,
(void*)&forward_message,number_of_bytes))==-1){
                                                             system_error("Forwarding
message");
                                                      }
                                               }
                                        }
                                 }
                          }
                   }//End forward message
            }//End dealing with data from client
```

```
       }//else{
//            printf("Garbage\n");}//end new connection
    }//end loop through file descriptors
} //while loop end


    close(socket_server);
    return 0;
}
```

COMMON_DEF.h

```
/************************common_def.h**********************************/
//Author:Sanket Agarwal & Dhiraj Kudva (agarwal.220196, dhirajkudva)@tamu.edu
//Organisation: Texas A&M University
//Description: Simple Broadcast Chat Server structure files.

#ifndef common_def
#define common_def

struct simple_broadcast_chat_server_header{

       unsigned int version:9; //9 bits as defined in the mannual.
       unsigned int type:   7; //7 bits as defined.
       int length;
};

struct simple_broadcast_chat_server_attribute{

       int type;
       int length;
       char payload_data [512];

};

struct simple_broadcast_chat_server_message{

       struct simple_broadcast_chat_server_header header;
       struct simple_broadcast_chat_server_attribute attribute[2];// to identify the
two different msgs. i.e. the actual payload message and the username.
};


struct simple_broadcast_chat_server_client_user_information{

       char username[16];//as definedin the manual
       int file_descriptor;
       int client_count;
};
```

```
#endif
int system_error(const char* error_string) //display and exit
{
       error(error_string);
       exit(1);
}


MAKEFILE:
# **************************************** Make file code ********************
# Author: Sanket Agarwal & Dhiraj Kudva (agarwal.220196, dhirajkudva)@tamu.edu
#Organisation: Texas A&M University
#Description: Machine problem 1, Compiles server and client source code.



#for compiling server.c
sample : echos.o echo.o


echos.o: server.c common_def.h
       gcc -I. server.c -o echos


#for compiling client.c

echo.o: client.c common_def.h
       gcc -I. client.c -o echo

#clean to discard previous .o files
clean:
       rm -f sample *.o core
```