**ECEN 602**

**NETWORK SIMULATION ASSIGNMENT – 02**

**TEAM 17**

**Mohammad Faisal Khan**

**Amiya Ranjan Panda**

**INTRODUCTION**

We have implemented a client and server for a simple chat service.

It has been successfully compiled, executed and tested on gcc compiler (part of standard LINUX).

Common Errors and Catches:

-If data is not input correctly on the command line as per the ordering given below, it throws a segmentation error.

-This is NOT to be assumed as an error.

-If data is missing from the command line, it throws segmentation error too.

-If a user with a name same as an existing active chat room user joins, he will be required to reenter chat room using a valid name.

-Username can be any size until 512 characters.

-Maximum number of clients active on chat room = argv[3]. If argv[3]+1 client joins, it will throw an ERROR on server and client console.

-IMPORTANT: Please do not use same serverport when running server second time, until the first server process is killed (NOT SUSPENDED).

-If a client disconnects using keyword 'quit', a smooth disconnect happens and resources are deallocated. If not, the other clients enter into hang state.

-This is an iterative server, no use of fork() in this code.

Usage of MAKEFILE:

make -f makefile\_server

make -f makefile\_client

Then, EXECUTE using the following commands:

./team17\_server <ip> <port> <max user>

Example - ./team17\_server 127.0.0.1 50001 12

./team17\_client <user name> <ip> <port>

Example - ./team17\_client user1 127.0.0.1 50001

Usage:

All commands are executed on standard Terminal of LINUX which supports gcc compiler commands.

Special uses:

- SELECT() function has been used in both client and server codes to determine the action.

- SEND() and RECV() are used to send data and receive data respectively into the structs.

Architecture: Modular (Functional) Programming

-We have used modular approach to C Programming in this code.

-Specific functions like sending JOIN, FWD and SEND messages are implemented separetely in the code for higher clarity.

-This code has comments on most lines to fathom the working of the functions.

Data usage: Data Structures (DS)

-Encapsulation of messages into network packets is implemented using DS.

-SBCP frame format for packetization is used.

-As per the requirement, two level encapsulation scheme is employed.

COMMANDS Implemented:

JOIN COMMAND: When a new user connects to the chat room, a JOIN struct is sent across to the server.

This contains the username of the client.

SEND COMMAND: When a user types a chat message, this is encapsulated in the SEND struct and sent to the server.

This contains the username and the chat message of a single client.

FWD COMMAND: Server broadcasts the chat message to the clients using the FWD struct.

ERROR COMMAND: When a client with the same name as a already active client joins the session, Server sends a ERROR command.

This client is disconnected from the server and chat room.

Prints on Server:

-Server starting.

-Listening.

-Shows a connected succesfully message each time a client connects to server IP and server port.

-Shows execution of FWD command.

-Shows execution of ERROR command.

Prints on Client:

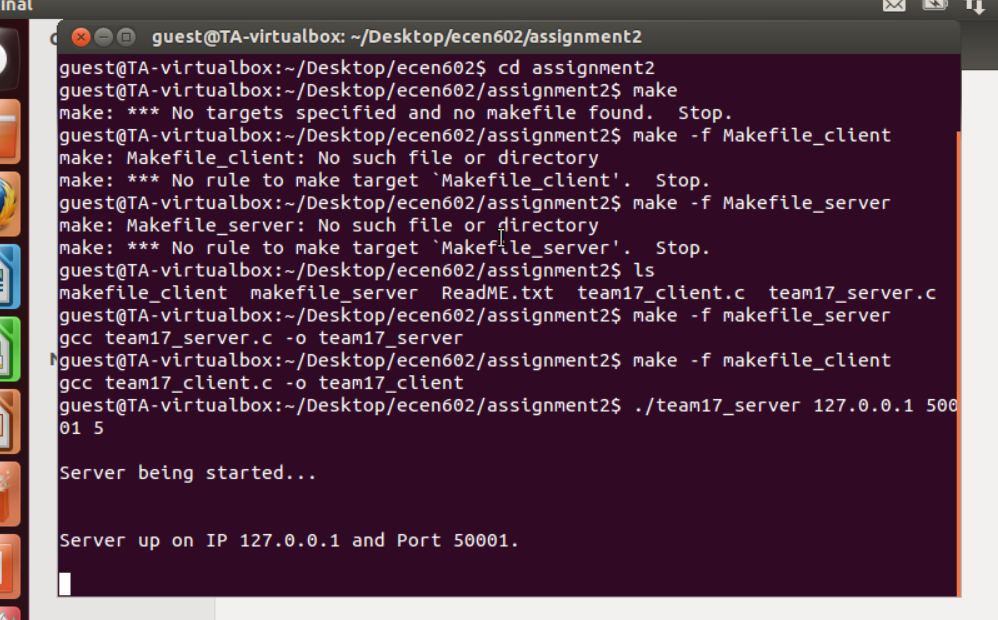
-Client connecting.

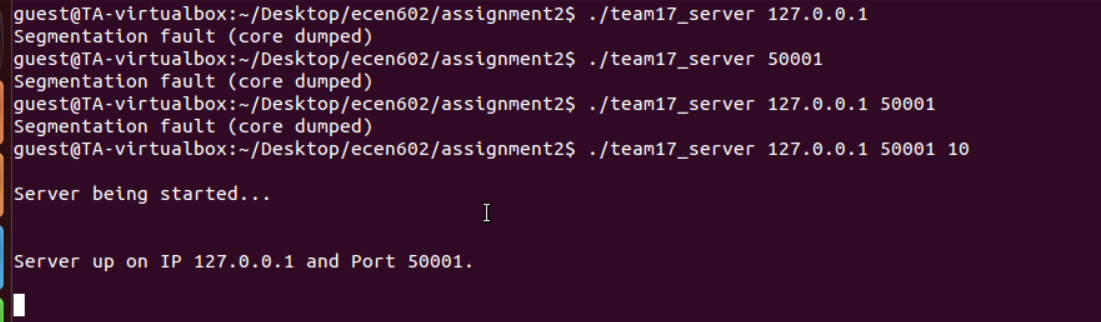
-JOIN Message.

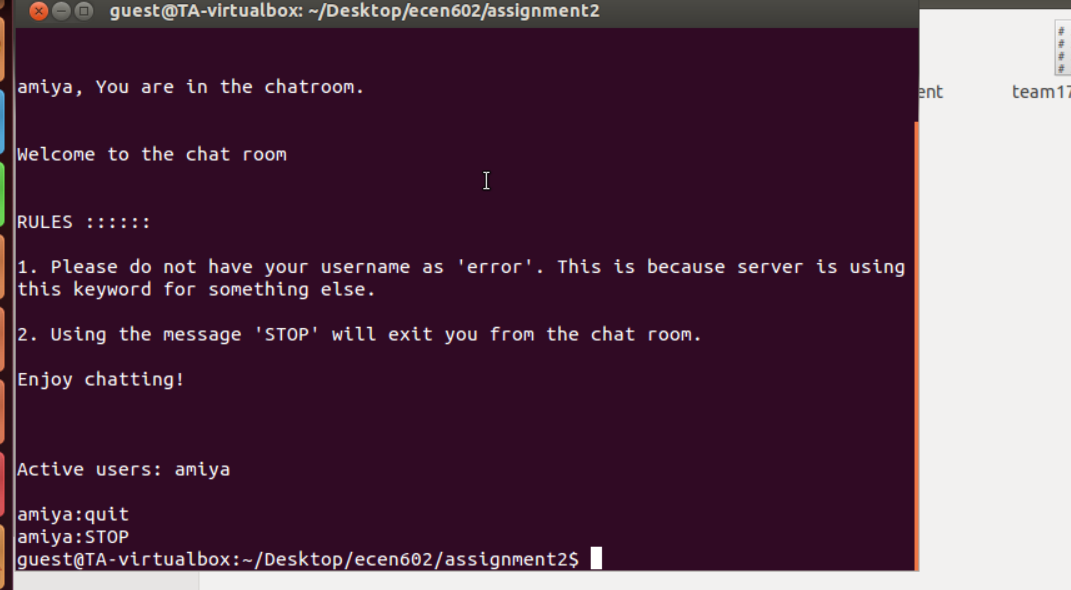
-Chat room rules.

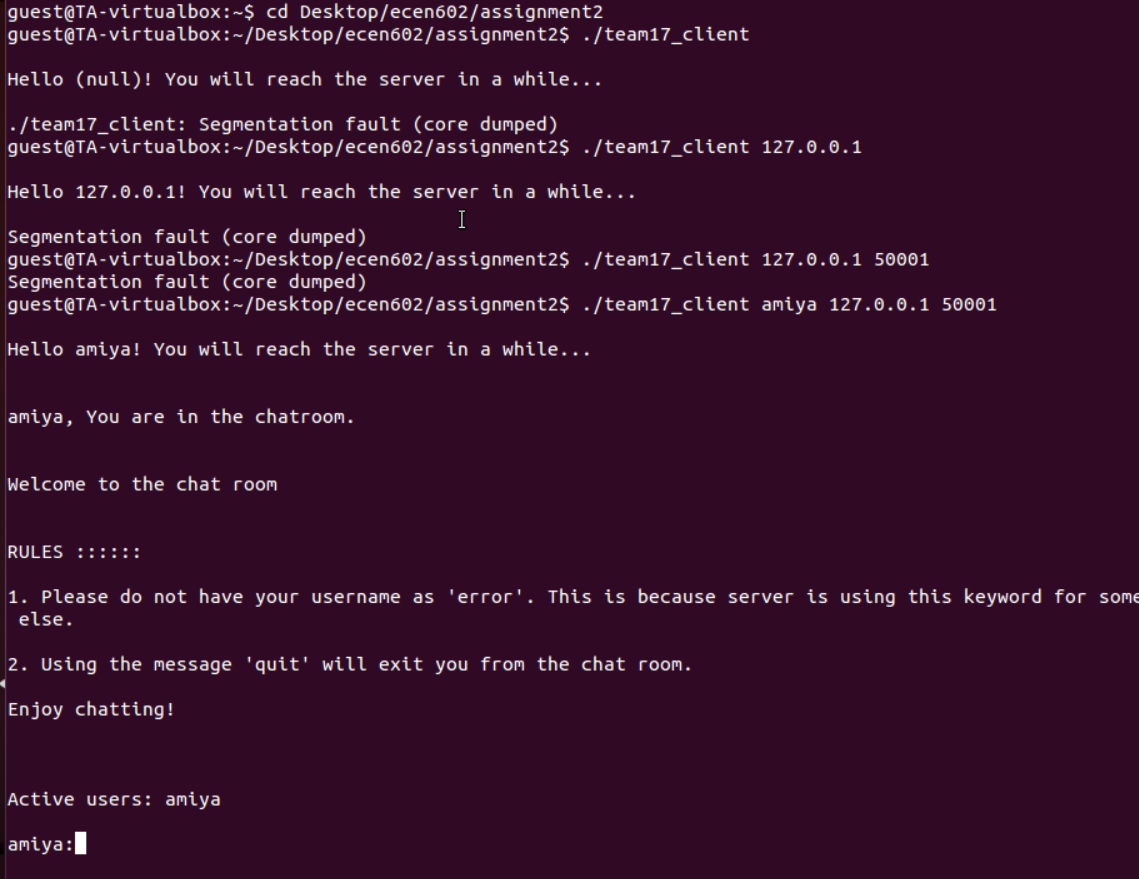
-Active users on the chat room.

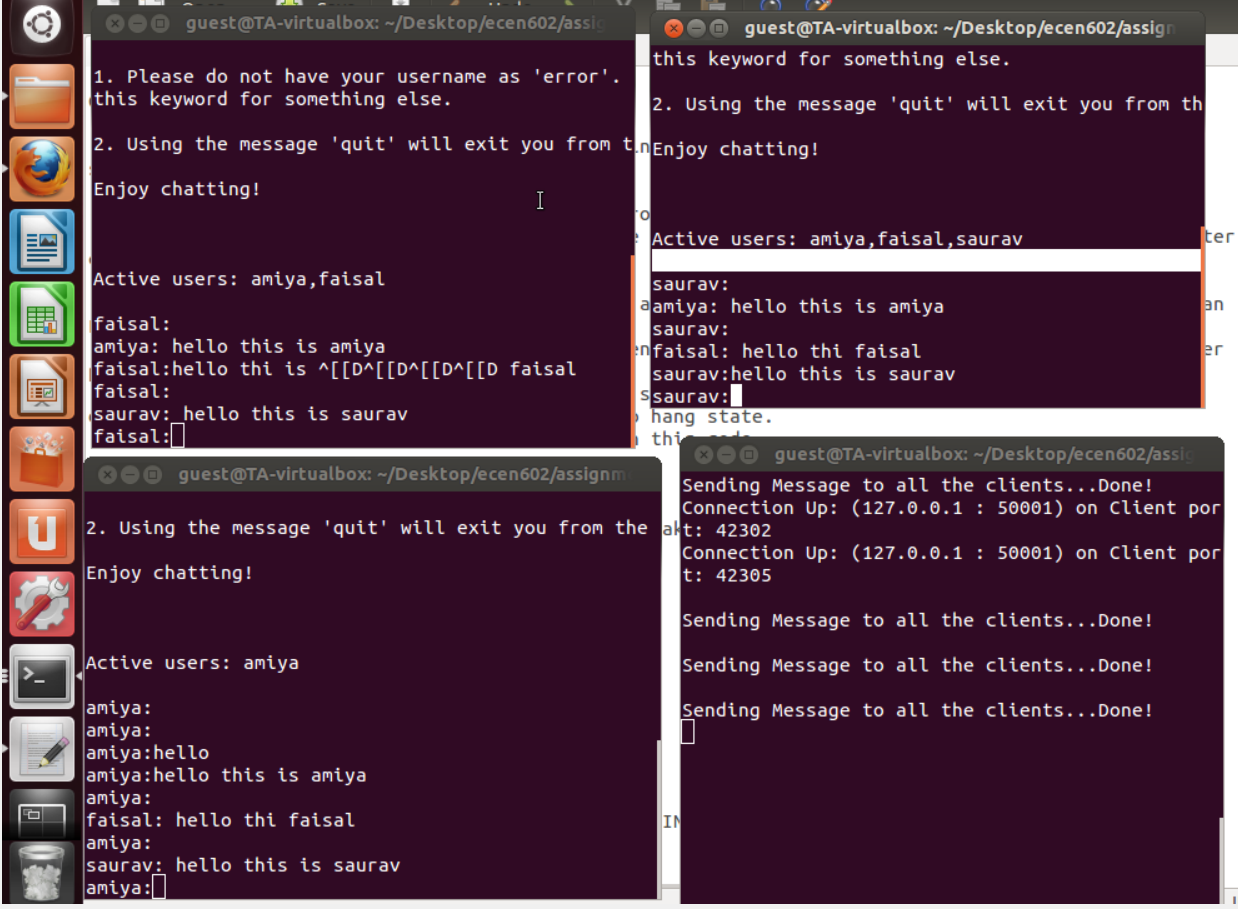
-ERROR when client uses the same username as an already active client.

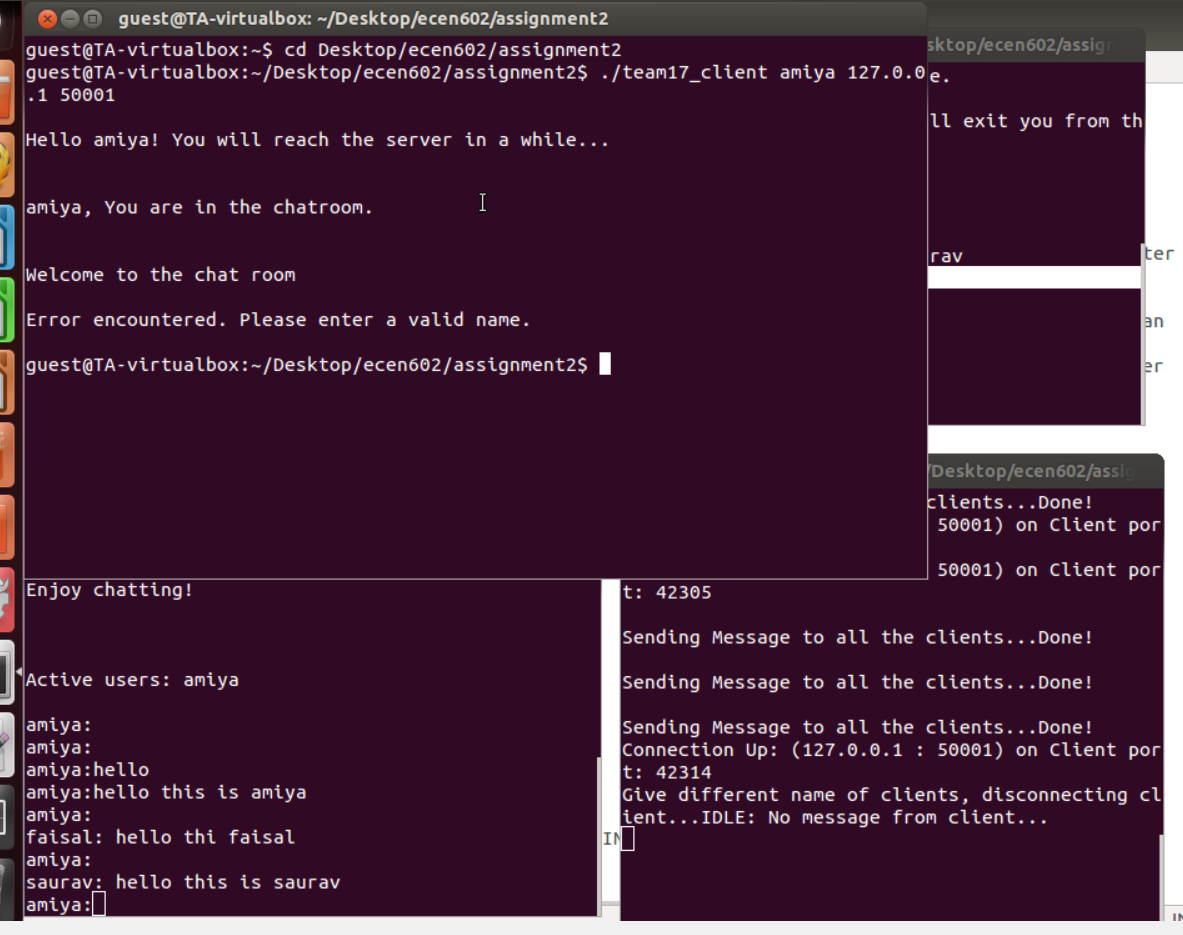


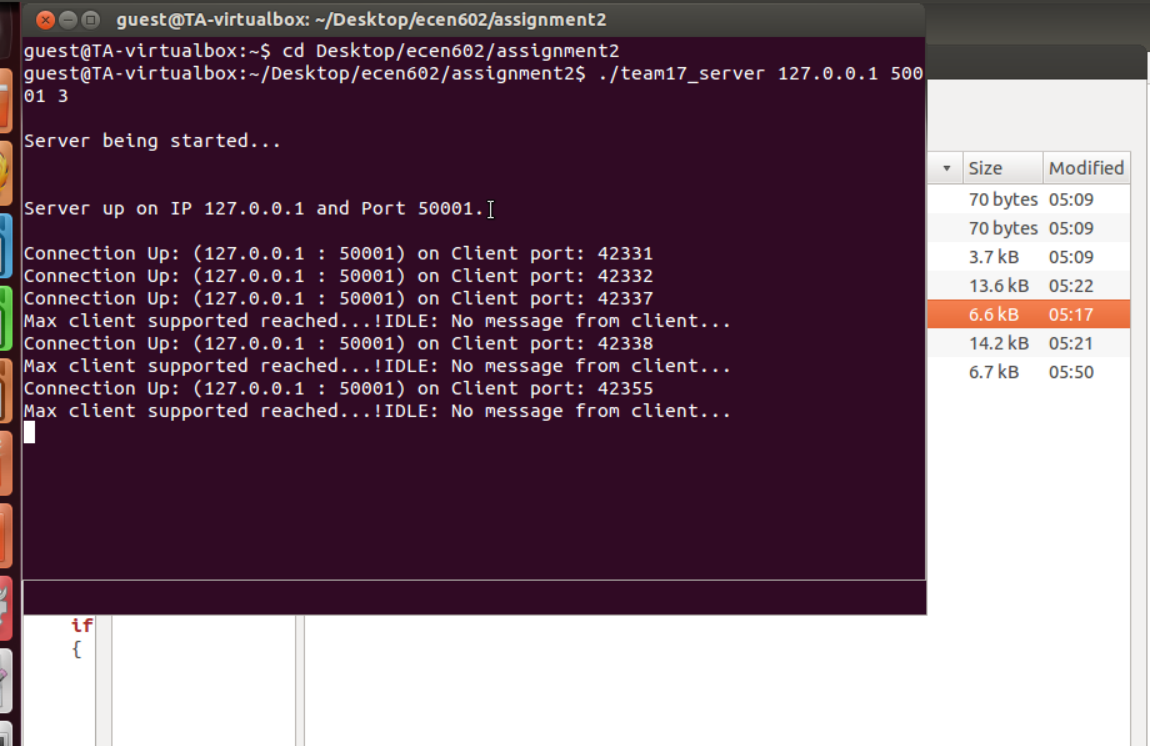












TCP CHAT SERVER CODE ::

# include <stdio.h>

# include <string.h>

# include <sys/types.h>

# include <netinet/in.h>

# include <sys/socket.h>

# include <stdlib.h>

# include <arpa/inet.h>

# include <unistd.h>

# include <netdb.h>

/\* Global Structures \*/

int count;

void reply\_join(int socket\_fd1, int argv3);

char users[1024];

struct attr\_payload

{

char client\_usrname[16];

char message[512];

char reason[32];

int client\_count;

}a\_payload;

struct attr\_sbcp

{

int attr\_type;

int attr\_length;

struct attr\_payload sbcp\_attr\_payload;

}msg\_username, msg, msg\_reason;

struct msg\_sbcp

{

int ver;

int msg\_type;

int msg\_length;

struct attr\_sbcp msg\_payload[4];

}final\_msg, join\_msg;

/\* Declarations of used functions \*/

void connect\_server(int \*socket\_fd,struct sockaddr\_in \*serv\_addr,int argv2, int argv3, char argv1[4]);

void accept\_new\_client(fd\_set \*main\_set, int \*max\_fd, int socket\_fd, struct sockaddr\_in \*client\_addr, char argv2[10], int argv3);

void send\_receive\_data(int num, fd\_set \*main\_set, int socket\_fd, int max\_fd);

void broadcast\_data (int count, int num, int socket\_fd, int recvd\_bytes, struct msg\_sbcp receive\_buffer, fd\_set \*main\_set);

void delete\_client(fd\_set \*main\_set, int \*max\_fd, int socket\_fd, struct sockaddr\_in \*client\_addr, char argv2[10], int argv3);

int main(int argc, char \*argv[])

{

struct sockaddr\_in serv\_addr, client\_addr;

fd\_set main\_set;

fd\_set temp\_set;

int max\_fd;

int i;

int argv2=atoi(argv[2]);

int argv3=atoi(argv[3]);

int socket\_fd=0;

FD\_ZERO(&main\_set);

FD\_ZERO(&temp\_set);

connect\_server(&socket\_fd, &serv\_addr,argv2,argv3,argv[1]);

FD\_SET(socket\_fd, &main\_set);

max\_fd=socket\_fd;

fflush(stdout);

printf("\nServer being started...\n\n");

fflush(stdout);

sleep(1);

fflush(stdout);

printf("\nServer up on IP %s and Port %s.\n\n",argv[1], argv[2]);

fflush(stdout);

while(1)

{

temp\_set=main\_set;

if(select(max\_fd+1,&temp\_set,NULL,NULL,NULL)==-1)

{

error("Error in select()..!");

exit(4);

}

for(i=0;i<=max\_fd;i++)

{

if(FD\_ISSET(i, &temp\_set))

{

if(i==socket\_fd)

{

accept\_new\_client(&main\_set,&max\_fd,socket\_fd,&client\_addr,argv[2],argv3);

}

else

send\_receive\_data(i,&main\_set,socket\_fd,max\_fd);

}

}

}

return 0;

}

void connect\_server(int \*socket\_fd,struct sockaddr\_in \*serv\_addr, int argv2,int argv3, char argv1[4])

{

if((\*socket\_fd = socket(AF\_INET,SOCK\_STREAM,0))==-1)

error("Error in socket()..!");

serv\_addr->sin\_family = AF\_INET;

serv\_addr->sin\_port = htons(argv2);

serv\_addr->sin\_addr.s\_addr = inet\_addr(argv1);

memset(serv\_addr->sin\_zero, '\0', sizeof serv\_addr->sin\_zero);

int flag=1;

if(setsockopt(\*socket\_fd,SOL\_SOCKET,SO\_REUSEADDR,&flag,sizeof(int))==-1)

{

error("Error in socket opt()...\n");

exit(1);

}

if(bind(\*socket\_fd, (struct sockaddr \*)serv\_addr, sizeof (struct sockaddr))==-1)

{

error("Error in bind()...\n");

exit(1);

}

if(listen(\*socket\_fd,argv3) == -1)

{

error("Error in listen()...\n");

exit(1);

}

}

void accept\_new\_client(fd\_set \*main\_set, int \*max\_fd, int socket\_fd, struct sockaddr\_in \*client\_addr,char argv2[10], int argv3)

{

int new\_socket\_fd;

socklen\_t length;

length=sizeof(struct sockaddr\_in);

if((new\_socket\_fd=accept(socket\_fd,(struct sockaddr\*)client\_addr,&length))==-1)

{

error("Client error... Cannot accept...");

exit(1);

}

else

{

FD\_SET(new\_socket\_fd,main\_set);

if(new\_socket\_fd>\*max\_fd)

{

\*max\_fd=new\_socket\_fd;

}

printf("Connection Up: (%s : %s) on Client port: %d\n",inet\_ntoa(client\_addr->sin\_addr),argv2,ntohs(client\_addr->sin\_port));

int new\_socket\_fd1=new\_socket\_fd;

reply\_join(new\_socket\_fd1,argv3);

}

}

void send\_receive\_data(int num, fd\_set \*main\_set, int socket\_fd, int max\_fd)

{

int recvd\_bytes;

memset(&final\_msg, 0, sizeof(final\_msg));

if((recvd\_bytes= recv(num, &final\_msg, sizeof(final\_msg), 0)) > 0)

{

int count;

for(count=0;count<=max\_fd;count++)

{

broadcast\_data(count,num,socket\_fd,recvd\_bytes,final\_msg,main\_set);

}

fflush(stdout);

printf("\nSending Message to all the clients...Done!\n");

fflush(stdout);

}

else

{

if(recvd\_bytes==0)

{

printf("IDLE: No message from client...\n", num);

}

else

{

error("Error in data sent...");

}

close(num);

FD\_CLR(num,main\_set);

}

}

void reply\_join(int socket\_fd1, int argv3)

{

int recvd\_bytes;

memset(&join\_msg, 0, sizeof(join\_msg));

if((recvd\_bytes= recv(socket\_fd1, &join\_msg, sizeof(join\_msg), 0)) <=0)

{

if(recvd\_bytes==0)

{

printf("Socket not replying...", socket\_fd1);

}

else

{

error("Error in received bytes...");

}

close(socket\_fd1);

}

char name[16];

strcpy(name,join\_msg.msg\_payload[0].sbcp\_attr\_payload.client\_usrname);

if(strstr(users,name))

{

fflush(stdout);

printf("Give different name of clients, disconnecting client...");

fflush(stdout);

char error[]="error\_name";

send(socket\_fd1,error,strlen(error),0);

memset(&join\_msg, 0, sizeof(join\_msg));

}

else

{

if(count<argv3-1)

{

strcat(users,join\_msg.msg\_payload[0].sbcp\_attr\_payload.client\_usrname);

int m=strlen(users);

send(socket\_fd1,users,strlen(users),0);

users[m]=',';

memset(&join\_msg, 0, sizeof(join\_msg));

count++;

}

else

{

printf("Max client supported reached...!");

fflush(stdout);

char error[]="error\_max";

send(socket\_fd1,error,strlen(error),0);

memset(&join\_msg, 0, sizeof(join\_msg));

}

}

}

void broadcast\_data (int count, int num, int socket\_fd, int recvd\_bytes, struct msg\_sbcp final\_msg, fd\_set \*main\_set)

{

if(FD\_ISSET(count,main\_set))

{

if(count!=socket\_fd && count!=num)

{

if(send(count,&final\_msg,recvd\_bytes,0)==-1)

{

error("Error in send()...");

}

memset(&final\_msg, 0, sizeof(final\_msg));

}

}

}

TCP CHAT CLIENT CODE –

# include <stdio.h>

# include <string.h>

# include <sys/types.h>

# include <netinet/in.h>

# include <stdlib.h>

# include <unistd.h>

# include <sys/socket.h>

# include <errno.h>

# include <arpa/inet.h>

void cli\_conn(int \*serv\_filedes, struct sockaddr\_in \*servaddr, int argv3, char argv2[4]);

void sendrecv\_info(int i, int serv\_filedes, char argv1[16]);

void separate(char buffer[256], char username[16], int serv\_filedes);

void consolidate(int serv\_filedes, char argv1[16]);

void join(int serv\_filedes1, char username[16]);

int main(int argc, char \*argv[])

{

struct sockaddr\_in servaddr;

int serv\_filedes=0;

int max\_fd, i;

int argv3=atoi(argv[3]);

fd\_set main\_set, temp\_set;

fflush(stdout);

printf("\nHello %s! You will reach the server in a while...\n\n",argv[1]);

cli\_conn(&serv\_filedes, &servaddr, argv3, argv[2]);

sleep(1);

fflush(stdout);

printf("\n%s, You are in the chatroom.\n\n", argv[1]);

sleep(1);

printf("\nWelcome to the chat room\n\n");

fflush(stdout);

sleep(1);

sleep(1);

int serv\_filedes1=serv\_filedes;

join(serv\_filedes1,argv[1]);

char user\_list[1024]={'\0'};

int l;

l=recv(serv\_filedes1,user\_list,1024,0);

if(strcmp(user\_list,"error\_name")==0)

{

printf("Error encountered. Please enter a different name.\n\n");

exit(0);

}

if(strcmp(user\_list,"error\_max")==0)

{

printf("Error encountered. Maximum users already present in chatroom.\n\n");

exit(0);

}

printf("\nRULES ::::::\n\n");

printf("1. Please do not have your username as 'error'. This is because server is using this keyword for something else.\n\n");

printf("2. Using the message 'STOP' will exit you from the chat room.\n\n");

printf("Enjoy chatting!\n\n");

printf("\n\nActive users: %s\n\n",user\_list);

FD\_ZERO(&main\_set);

FD\_ZERO(&temp\_set);

FD\_SET(0,&main\_set);

FD\_SET(serv\_filedes,&main\_set);

max\_fd=serv\_filedes;

fflush(stdout);

printf("%s:",argv[1]);

fflush(stdout);

while(1)

{

temp\_set=main\_set;

if(select(max\_fd+1,&temp\_set,NULL,NULL,NULL)==-1)

{

error("Cannot select()");

exit(4);

}

for(i=0;i<=max\_fd;i++)

{

if(FD\_ISSET(i,&temp\_set))

{

sendrecv\_info(i,serv\_filedes,argv[1]);

}

}

}

return 0;

}

void separate(char buffer[256], char username[16], int serv\_filedes)

{

int buff\_len=strlen(buffer);

int user\_len=strlen(username);

struct attr\_payload

{

char client\_username[16];

char message[512];

char reason[32];

int client\_count;

};

struct sbcp\_attribute

{

int attribute\_type;

int attribute\_length;

struct attr\_payload sbcp\_payload;

}msg\_username, msg, msg\_reason;

struct sbcp\_message

{

int ver;

int msg\_type;

int msg\_length;

struct sbcp\_attribute msg\_payload[4];

}final\_msg,join\_msg;

memset(&final\_msg, 0, sizeof(final\_msg));

strcpy(msg\_username.sbcp\_payload.client\_username,username);

msg\_username.attribute\_type=2;

msg\_username.attribute\_length=user\_len+2+2;

strcpy(msg.sbcp\_payload.message,buffer);

msg.attribute\_type=4;

msg.attribute\_length=buff\_len+2+2;

final\_msg.ver=3;

final\_msg.msg\_type=4;

final\_msg.msg\_length=sizeof(msg\_username)+sizeof(msg)+2+2;

final\_msg.msg\_payload[0]=msg\_username;

final\_msg.msg\_payload[1]=msg;

send(serv\_filedes, &final\_msg,sizeof(final\_msg),0);

memset(&final\_msg, 0, sizeof(final\_msg));

fflush(stdout);

printf("%s:",username);

fflush(stdout);

}

void sendrecv\_info(int i, int serv\_filedes, char argv1[16])

{

char send\_buffer[256];

char receive\_buffer[256];

int received\_bytes;

if(i==0)

{

fgets(send\_buffer,256,stdin);

if(strcmp(send\_buffer,"STOP\n")!=0)

separate(send\_buffer,argv1,serv\_filedes);

else

exit(0);

fflush(stdout);

}

else

{

consolidate(serv\_filedes, argv1);

}

}

void consolidate(int serv\_filedes, char argv1[16])

{

struct attr\_payload

{

char client\_username[16];

char message[512];

char reason[32];

int client\_count;

};

struct sbcp\_attribute

{

int attribute\_type;

int attribute\_length;

struct attr\_payload sbcp\_payload;

};

struct sbcp\_message

{

int ver;

int msg\_type;

int msg\_length;

struct sbcp\_attribute msg\_payload[4];

}final\_msg;

int received\_bytes;

char receive\_buffer[256];

received\_bytes=recv(serv\_filedes, &final\_msg, sizeof(final\_msg),0);

strcpy(receive\_buffer,final\_msg.msg\_payload[1].sbcp\_payload.message);

receive\_buffer[received\_bytes]='\0';

printf("\n%s: %s",final\_msg.msg\_payload[0].sbcp\_payload.client\_username,receive\_buffer);

fflush(stdout);

printf("%s:",argv1);

fflush(stdout);

memset(&final\_msg, 0, sizeof(final\_msg));

fflush(stdout);

}

void cli\_conn(int \*serv\_filedes, struct sockaddr\_in \*servaddr, int argv3, char argv2[4])

{

if((\*serv\_filedes=socket(AF\_INET,SOCK\_STREAM,0))==-1)

{

error("Cannot create socket.");

exit(1);

}

servaddr->sin\_family=AF\_INET;

servaddr->sin\_port=htons(argv3);

servaddr->sin\_addr.s\_addr=inet\_addr(argv2);

memset(servaddr->sin\_zero,'\0',sizeof servaddr->sin\_zero);

if(connect(\*serv\_filedes,(struct sockaddr \*) servaddr, sizeof(struct sockaddr))==-1)

{

error("Failed to connect to socket.");

exit(1);

}

}

void join(int serv\_filedes1, char username[16])

{

int user\_len=strlen(username);

struct attr\_payload

{

char client\_username[16];

char message[512];

char reason[32];

int client\_count;

};

struct sbcp\_attribute

{

int attribute\_type;

int attribute\_length;

struct attr\_payload sbcp\_payload;

}msg\_username, msg, msg\_reason;

struct sbcp\_message

{

int ver;

int msg\_type;

int msg\_length;

struct sbcp\_attribute msg\_payload[4];

}join\_msg;

memset(&join\_msg, 0, sizeof(join\_msg));

strcpy(msg\_username.sbcp\_payload.client\_username,username);

msg\_username.attribute\_type=2;

msg\_username.attribute\_length=user\_len+2+2;

join\_msg.ver=3;

join\_msg.msg\_type=2;

join\_msg.msg\_length=sizeof(msg\_username)+2+2;

join\_msg.msg\_payload[0]=msg\_username;

send(serv\_filedes1, &join\_msg,sizeof(join\_msg),0);

memset(&join\_msg, 0, sizeof(join\_msg));

}