ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

SHASHANK KAMATH KALASA MOHANDAS
UIN: **627003580**

SRINIVAS PRAHALAD SUMUKHA
UIN: **627008254**

TEAM 6

ASSIGNMENT 2

Computer Communication and Networks

ECEN 602

FALL 2018

Role of SHASHANK KAMATH KALASA MOHANDAS

- server.cpp
- README file

Role of SRINIVAS PRAHALAD SUMUKHA

- client.cpp
- MAKEFILE
- README file
- run.sh

How to run:

1. Run the shell script in the terminal by typing the command '. run.sh' and enter the super user password
2. Two object files named client and server gets created. Execute the server program by typing './server *ip_address port_number max_clients*'. The server gets executed.
3. In a new terminal run the client program by typing './client username *ip_address port_number*'. The client gets executed. Now type any message to communicate with other clients.

Files included in this assignment:

- server.cpp
- client.cpp
- makefile
- run.sh
- README

MAKEFILE:

```
# Owner: Srinivas Prahalad Sumukha\
UIN: 627008254\
Functionality: Makefile to compile client and server\
CC = gcc
CFLAGS = -c #-Wall #wall is optional as all the warnings are already shown
output: client_sbcp server_sbcp #make if there is any changes to client or server

A: client.cpp #compiles client.cpp
    $(CC) $(CFLAGS) client.cpp
    #gcc -o client client.cpp

B: server.cpp #complies server.cpp
```

```
    $(CC) $(CFLAGS) server.cpp
    #gcc -o server server.cpp

clean:
    rm client server
```

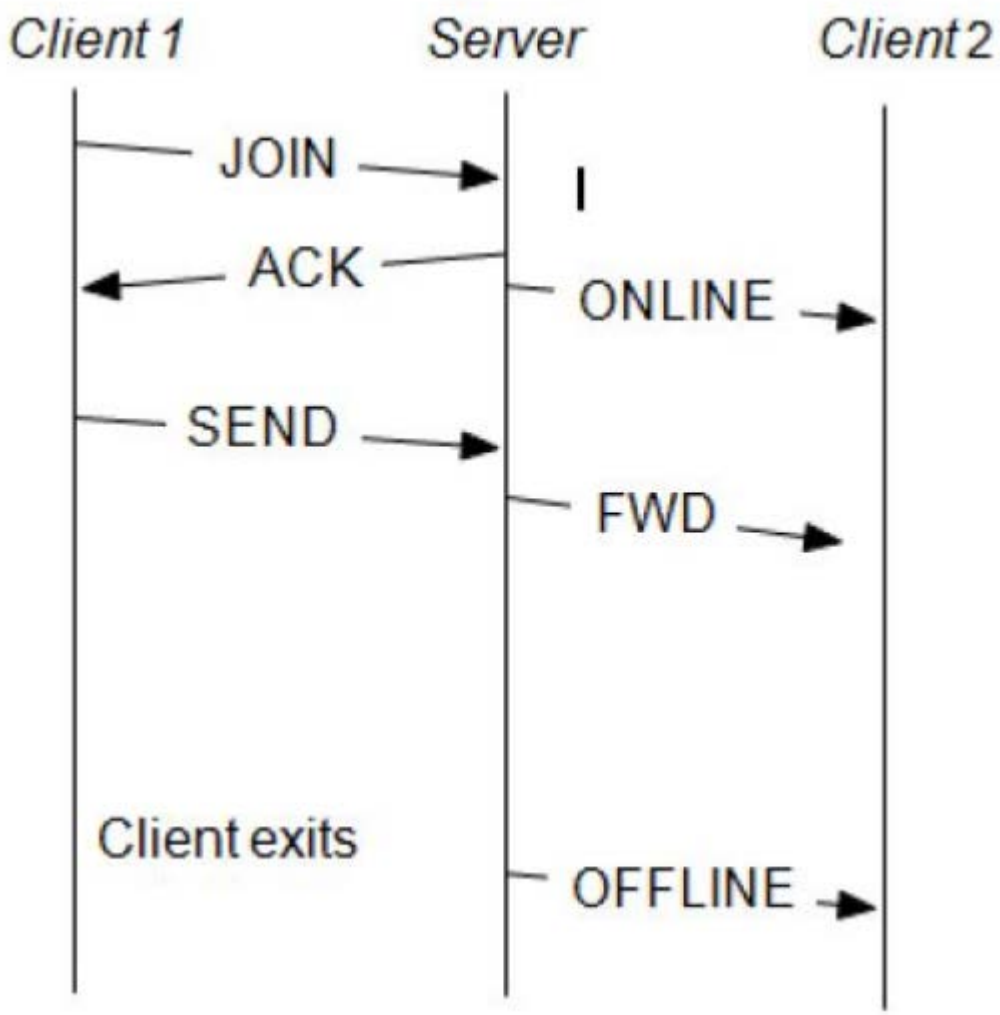
run.sh:

```
# Owner: Srinivas Prahalad Sumukha
# UIN: 627008254
# Functionality: Shell script to run make and to change permissions of the client
and the server
make
sudo chmod 777 server client #used to change the permissions of objects files
```

ARCHITECTURE

Server Implementation:

- Using the functions getaddrinfo(), socket(), bind(), listen() and accept(), a connection from the client is accepted.
- The select() function is used for I/O multiplexing.
- If there is something to read from the new client trying to connect to the socket file descriptor, then the new client is accepted provided the username is usable and there is space for the client to join.
- On the client successfully joining the chat window, it receives a *ACK* message which contains the number of clients in the chatroom along with their names. Correspondingly all the other clients in the chatroom receive a ONLINE message which specifies that a new client has joined the chatroom.
- If the client could not join successfully, then a *NAK* message is sent to the client stating the reason either as “Username used” or “Client limit exceeded”
- On receiving IDLE message from a client, the IDLE message is forwarded to the rest of the client along with the client name which is idle.
- FWD message is used to forward the message sent by 1 client to the rest of the clients.
- Unpacking and packing functions have been used to change the message from network byte order to normal order and vice-versa.
- When a client sends a message, the server unpacks the message, checks the version and message type and determines the type of packet being received.



Client Implementation:

- Using the functions getaddrinfo(), socket() and connect() a connection is established with the server.
- The client uses a unique username to connect with the server.
- Unpacking and packing functions have been used to change the message from network byte order to normal order and vice-versa.
- Firstly, the client packs the username into a packet and sends to the server while connecting and receives *ACK* or *NAK* message correspondingly. If it receives *NAK* then the client exits.
- The select() function is used for I/O multiplexing.

- Using the select function the data is either read from the keyboard or the from the incoming packet.
- In case of keyboard input the data is read using fgets and displayed using fputs to avoid byte buffer overflow attack. Fgets and fputs are implemented in user defined functions namely writen and readline.
- The input data is then packed and transmitted across to the server using send function.
- The incoming data from the server is initially received using the recv function and further unpacked as per SBCP standard message format.
- The received message can indicate if the received message is either *ACK*, *NAK*, online status, offline status e.t.c

SERVER CODE:

server.cpp

```
/*****************************************************************************
**********************
OWNER: SHASHANK KAMATH KALASA MOHANDAS
UIN: 627003580

*****************************************************************************
**********************/

#include<string.h>
#include<ctype.h>
#include<stdarg.h>
#include<stdint.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdio.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
#include<inttypes.h>
#include<signal.h>

#define BACKLOG 10

#define JOIN 2
#define FWD 3
#define SEND 4
#define NAK 5
#define OFFLINE 6
#define ACK 7
#define ONLINE 8
#define IDLE 9
```

```
//Declaring the variables
char *PORT;
char s[INET6_ADDRSTRLEN];
fd_set master;
fd_set read_fds;
int maxfd;
int socketfd, newsocketfd, flag, numbytes, yes=1;
int rcv,length=1;
int i, j, z,b,k=1;
int client_count=0;
struct addrinfo addressinfo, *servicelist, *loopvariable;
struct sockaddr_storage str_addr;
socklen_t addr_size;
char buffer[800];
char buffer_send[600];
char buffer_message_send[560];
char buffer_message_recv[512];
char buffer_username[16];
char usernames[100][16];
char list_name[500];
char client_exit[200];
struct sbcp_msg message_recv,message_send;
struct sbcp_attribute attribute_recv,attribute_send;
int16_t packetsize;

struct sbcp_attribute
{
    int16_t type;
    int16_t length;
    char* payload;
}attribute_recv,attribute_send;

struct sbcp_msg
{
    int8_t vrsn;
    int8_t type;
    int16_t length;
    struct sbcp_attribute* attribute;
}message_recv,message_send;

//Packing and Unpacking functions are taken from Beej's Guide to Network
Programming.
void packi16(char *buf, unsigned int i)
{
*buf++ = i>>8;
*buf++ = i;
}

int32_t pack(char *buf, char *format, ...)
{
    va_list ap;
    int16_t h;
    int8_t c;
    char *s;
    int32_t size = 0;
    int32_t len;
    va_start(ap, format);
    for(; *format != '\0'; format++) {
```

```
        switch(*format) {
            case 'h': // 16-bit
                size += 2;
                h = (int16_t)va_arg(ap, int);
                packi16(buf, h);
                buf += 2;
                break;
            case 'c': // 8-bit
                size += 1;
                c = (int8_t)va_arg(ap, int);
                *buf++ = (c>>0)&0xff;
                break;
            case 's': // string
                s = va_arg(ap, char*);
                len = strlen(s);
                size += len + 2;
                packi16(buf, len);
                buf += 2;
                memcpy(buf, s, len);
                buf += len;
                break;
        }
    }
    va_end(ap);
    return size;
}

unsigned int unpacki16(char *buf)
{
    return (buf[0]<<8) | buf[1];
}

void unpack(char *buf, char *format, ...)
{
    va_list ap;
    int16_t *h;
    int8_t *c;
    char *s;
    int32_t len, count, maxstrlen=0;
    va_start(ap, format);
    for(; *format != '\0'; format++) {
        switch(*format) {
            case 'h': // 16-bit
                h = va_arg(ap, int16_t*);
                *h = unpacki16(buf);
                buf += 2;
                break;
            case 'c': // 8-bit
                c = va_arg(ap, int8_t*);
                *c = *buf++;
                break;
            case 's': // string
                s = va_arg(ap, char*);
                len = unpacki16(buf);
                buf += 2;
                if (maxstrlen > 0 && len > maxstrlen) count = maxstrlen - 1;
                else count = len;
                memcpy(s, buf, count);
                s[count] = '\0';
```

```
                buf += len;
                break;
            default:
                if (isdigit(*format)) {
                    maxstrlen = maxstrlen * 10 + (*format-'0');
                }
        }
        if (!isdigit(*format)) maxstrlen = 0;
    }
    va_end(ap);
}

void *getaddress(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET)
    {
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }
        return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

void *getport(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET)
    {
        return &(((struct sockaddr_in*)sa)->sin_port);
    }
        return &(((struct sockaddr_in6*)sa)->sin6_port);
}

// TO send the full packet data if not fully sent
int sendall(int socket_id, char *buf, int length)
{
int cnt = 0;
int bytesleft = length;
int bytessent;
while(cnt < length)
    {
        bytessent = send(socket_id, buf+cnt, bytesleft, 0);
        if (bytessent == -1)
        {
            break;
        }
        cnt = cnt + bytessent;
        bytesleft = bytesleft - bytessent;
    }
length = cnt;
return bytessent==-1?-1:0; // return -1 on failure, 0 on success
}

void sighandler(int signum)
{
    sprintf(buffer_message_send,"Server Terminated");
    attribute_send.payload=buffer_message_send;
    attribute_send.type=1;
    attribute_send.length=36;//32+4
    message_send.vrsn='3';
    message_send.type=NAK;
    message_send.length=40;
```

```
    message_send.attribute=&attribute_send;

packetsize=pack(buffer_send,"cchhhs",message_send.vrsn,message_send.type,message_
send.length,attribute_send.type,attribute_send.length,buffer_message_send);
    for(z=4; z<=maxfd; z++)
    {
        if(sendall(z, buffer_send, packetsize)==-1)
        {
            perror("Server: Send Error during client exit \n");
        }
    }
    exit(0);
}

int main(int argc, char *argv[])
{
    FD_ZERO(&master);
    FD_ZERO(&read_fds);
    int max_clients=atoi(argv[3]);
    if(argc !=4)
    {
        printf("Server: Excess Arguments Passed /n");
        exit(1);
    }

    memset(&addressinfo,0,sizeof (addressinfo)); // Making the addressinfo struct
zero
    addressinfo.ai_family = AF_UNSPEC;// Not defining whether the connection is
IPv4 or IPv6
    addressinfo.ai_socktype = SOCK_STREAM;
    addressinfo.ai_flags = AI_PASSIVE;

    if ((flag = getaddrinfo(argv[1], argv[2], &addressinfo, &servicelist)) != 0)
    {
        printf("GetAddrInfo Error");
        exit(1);
    }
    printf("Server: Done with getaddrinfo \n");
    // Traversing the linked list for creating the socket
    for(loopvariable = servicelist; loopvariable != NULL; loopvariable =
(loopvariable -> ai_next ))
    {
        if((socketfd = socket(loopvariable -> ai_family, loopvariable ->
ai_socktype, loopvariable -> ai_protocol)) ==  -1 )
        {
            printf("Server: Socket Created.\n");
            continue;
        }
        if (setsockopt(socketfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) ==
-1)
        {
            perror("Server: SetSockOpt.\n");
            exit(1);
        }
        //Binding the socket
        if (bind(socketfd, loopvariable->ai_addr, loopvariable->ai_addrlen) ==  -
1)
        {
            close(socketfd);
```

```
            perror("Server: Bind Error.\n");
            continue;
        }
        break;
    }
    // Freeing the linked  list
    freeaddrinfo(servicelist);

    if (loopvariable == NULL)
    {
        printf("Server: Failed to bind.\n");
        exit(1);
    }

    // Listen to the connection
    if (listen(socketfd, BACKLOG) ==  -1)
    {
        perror("Server: Listen Error");
        exit(0);
    }
    printf("Server: Listening in progress \n");

    FD_SET(socketfd,&master);
    maxfd=socketfd;

    while(1)
    {
        read_fds=master;
        if(select(maxfd+1,&read_fds,NULL,NULL,NULL) == -1)
        {
            perror("Server: Select Error");
            exit(1);
        }

        for(i=0; i<=maxfd; i++)
        {
            if(FD_ISSET(i,&read_fds))
            {
                signal(SIGINT, sighandler);//checking for CTRL+C input
                if(i==socketfd)//Checking for a new connection
                {
                        addr_size = sizeof str_addr;
                        newsocketfd = accept(socketfd, (struct sockaddr
*)&str_addr, &addr_size);
                        //printf("Accepted new connection: \n");
                        if (newsocketfd == -1)
                        {
                            perror("Server: Accept Error");
                            exit(1);
                        }
                        else
                        {
                            FD_SET(newsocketfd,&master);
                            if(newsocketfd>maxfd)
                            {
                                maxfd=newsocketfd;
                            }
                            client_count=client_count+1;
                        }
```

```
                }
                else
                {

                    if((rcv=recv(i,buffer,600,0))<=0)
                    {
                        if(rcv==0)//If client exits unceremonously
                        {
                            printf("Client %s has disconnected\n",usernames[i]);
                            attribute_send.payload=usernames[i];
                            attribute_send.type=2;
                            attribute_send.length=20;//2+2+16
                            message_send.vrsn='3';
                            message_send.type=OFFLINE;
                            message_send.length=24;
                            message_send.attribute=&attribute_send;

packetsize=pack(buffer_send,"cchhhs",message_send.vrsn,message_send.type,message_
send.length,attribute_send.type,attribute_send.length,attribute_send.payload);
                            for(j = 0; j <= maxfd; j++)
                            {
                                if (FD_ISSET(j, &master))
                                {
                                    if (j != socketfd && j != i)
                                    {
                                        if(sendall(j, buffer_send, packetsize)==-
1)
                                        {
                                            perror("Server: Send Error during
client exit \n");
                                        }
                                    }
                                }
                            }
                            usernames[i][0]='\0';
                        }
                        else
                        {
                            perror("Server: Receive Error \n");
                        }
                        client_count=client_count-1;
                        close(i);
                        FD_CLR(i,&master);
                    }
                    else
                    {

unpack(buffer,"cchhh",&message_recv.vrsn,&message_recv.type,&message_recv.length,
&attribute_recv.type,&attribute_recv.length);
                        if(message_recv.vrsn=='3')
                        {
                            if(message_recv.type==JOIN && attribute_recv.type==2)
                            {
                                if(client_count>max_clients)//If the client limit
is reached then sending NAK message
                                {
                                    //send NAK message
                                    sprintf(buffer_message_send,"Client Limit
Exceeded");
```

```
                                        attribute_send.payload=buffer_message_send;
                                        attribute_send.type=1;
                                        attribute_send.length=36;//32+4
                                        message_send.vrsn='3';
                                        message_send.type=NAK;
                                        message_send.length=40;
                                        message_send.attribute=&attribute_send;

packetsize=pack(buffer_send,"cchhhs",message_send.vrsn,message_send.type,message_
send.length,attribute_send.type,attribute_send.length,buffer_message_send);
                                        if(sendall(i, buffer_send, packetsize)==-1)
                                        {
                                            perror("send");
                                        }
                                        client_count=client_count-1;
                                        close(i);
                                        FD_CLR(i, &master); // remove from master set
                                        break;
                                    }
                                    unpack(buffer+8,"s",buffer_username);
                                    for(j=4;j<=maxfd;j++)
                                    {
                                        if(strcmp(buffer_username,usernames[j])==0)
                                        {
                                            k=0;//if username is already used then
NAK message sent
                                            sprintf(buffer_message_send,"USERNAME
used");

attribute_send.payload=buffer_message_send;
                                            attribute_send.type=1;
                                            attribute_send.length=36;//32+4
                                            message_send.vrsn='3';
                                            message_send.type=NAK;
                                            message_send.length=40;
                                            message_send.attribute=&attribute_send;

packetsize=pack(buffer_send,"cchhhs",message_send.vrsn,message_send.type,message_
send.length,attribute_send.type,attribute_send.length,buffer_message_send);
                                            if(sendall(i, buffer_send, packetsize)==-
1)
                                            {
                                                perror("send");
                                            }
                                            client_count=client_count-1;
                                            close(i);
                                            FD_CLR(i, &master); // remove from master
set
                                            break;
                                        }
                                    }
                                    if (k==1)//Username usable
                                    {
                                        sprintf(usernames[i],"%s",buffer_username);
                                        printf("Client %s connected\n",usernames[i]);
                                        strcpy (list_name,"Names of clients in the
chat room: ");
                                        for(b=4;b<=maxfd;b++)
                                        {
```

```
                                        strcat (list_name,usernames[b]);
                                        strcat (list_name,"|");
                                    }
                                    //sending ACK messageupon successfull
connection
                                    sprintf(buffer_message_send,"Number of
Clients: %d\n",client_count);
                                    strcat(buffer_message_send,list_name);
                                    attribute_send.payload=buffer_message_send;
                                    attribute_send.type=4;
                                    attribute_send.length=516;
                                    message_send.vrsn='3';
                                    message_send.type=ACK;
                                    message_send.length=520;
                                    message_send.attribute=&attribute_send;

packetsize=pack(buffer_send,"cchhhs",message_send.vrsn,message_send.type,message_
send.length,attribute_send.type,attribute_send.length,buffer_message_send);
                                    if(sendall(i, buffer_send, packetsize)==-1)
                                    {
                                        perror("send");
                                    }
                                    //sending ONLINE message
                                    attribute_send.payload=usernames[i];
                                    attribute_send.type=2;
                                    attribute_send.length=20;
                                    message_send.vrsn='3';
                                    message_send.type=ONLINE;
                                    message_send.length=24;
                                    message_send.attribute=&attribute_send;

packetsize=pack(buffer_send,"cchhhs",message_send.vrsn,message_send.type,message_
send.length,attribute_send.type,attribute_send.length,attribute_send.payload);
                                    for(j = 0; j <= maxfd; j++)
                                    {
                                        if (FD_ISSET(j, &master))
                                        {
                                            if (j != socketfd && j != i)
                                            {
                                                if(sendall(j, buffer_send,
packetsize)==-1)
                                                {
                                                    perror("send");
                                                }
                                            }
                                        }
                                    }

                                }
                            }
                            if(message_recv.type==SEND &&
attribute_recv.type==4)//received chat message
                            { // Forwarding the message back
                                    unpack(buffer+8, "s", buffer_message_recv);
                                    sprintf(buffer_message_send,
"<%s>:%s",usernames[i],buffer_message_recv);
                                    attribute_send.payload=buffer_message_send;
                                    attribute_send.type=4;
                                    attribute_send.length=516;
```

```
                                        message_send.vrsn='3';
                                        message_send.type=FWD;
                                        message_send.length=520;
                                        message_send.attribute=&attribute_send;

packetsize=pack(buffer_send,"cchhhs",message_send.vrsn,message_send.type,message_
send.length,attribute_send.type,attribute_send.length,buffer_message_send);
                                        for(j = 0; j <= maxfd; j++)
                                        {
                                            if (FD_ISSET(j, &master))
                                            {
                                                if (j != socketfd && j != i)
                                                {
                                                    if(sendall(j, buffer_send,
packetsize)==-1)
                                                    {
                                                        perror("send");
                                                    }
                                                }
                                            }
                                        }
                                    }
                                    if(message_recv.type==IDLE)
                                    {   //sending IDLE message
                                        attribute_send.payload=usernames[i];
                                        attribute_send.type=2;
                                        attribute_send.length=20;
                                        message_send.vrsn='3';
                                        message_send.type=IDLE;
                                        message_send.length=24;
                                        message_send.attribute=&attribute_send;

packetsize=pack(buffer_send,"cchhhs",message_send.vrsn,message_send.type,message_
send.length,attribute_send.type,attribute_send.length,attribute_send.payload);
                                        for(j = 0; j <= maxfd; j++)
                                        {
                                            if (FD_ISSET(j, &master))
                                            {
                                                if (j != socketfd && j != i)
                                                {
                                                    if(sendall(j, buffer_send,
packetsize)==-1)
                                                    {
                                                        perror("send");
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }//end of for loop
        }//end of infinite while loop
}//end main
```

CLIENT CODE:

client.cpp

```
/*****************************************************************************
******************
Owner: Srinivas Prahalad Sumukha
Uin: 627008254
Questions solved: All situations including the bonus questions
Function: client in SBC Protocol

*****************************************************************************
******************/
#include <ctype.h>
#include <stdarg.h>
#include <stdint.h>
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/time.h>

#define debug 0
#define String_size 512
#define JOIN 2
#define FWD 3
#define SEND 4
#define NAK 5
#define OFFLINE 6
#define ACK 7
#define ONLINE 8
#define IDLE 9


struct sbcp_message
{
    int8_t vrsn;
    int8_t type;
    int16_t length;
    struct sbcp_attribute *payload;
}sbcp_msg,sbcp_msg_rcv;

struct sbcp_attribute
{
    int type;
    int length;
    char *payload_attribute;
}sbcp_attr,sbcp_attr_rcv;
```

```
int time_des = 0;
struct timeval time_count;
fd_set timefds;
fd_set readfds;
int status,flag = 0;
char buffer[String_size];
int receive_from_ack;
int receiving_size = sizeof (buffer);
socklen_t *fromlength;
addrinfo server, *server_ptr;
void *address_ptr_void;
char *address, *port, *name;
char username_of_client[16];
char ipstr[INET6_ADDRSTRLEN];
char buff[1000];
char buff_rcv[1024];
char *msg;
int socketfd;
int16_t size_of_packet;
int i_return;
int string_len;
char input_string[String_size];

void packi16(char *buf, unsigned int i)
{
    *buf++ = i>>8; *buf++ = i;
}

int32_t pack(char *buf, char *format, ...)
{
    va_list ap;
    int16_t h;
    int8_t c;
    char *s;
    int32_t size = 0;
    int32_t len;
    va_start(ap, format);
    for(; *format != '\0'; format++) {
        switch(*format) {
            case 'h': // 16-bit
                size += 2;
                h = (int16_t)va_arg(ap, int); // promoted
                packi16(buf, h);
                buf += 2;
                break;
            case 'c': // 8-bit
                size += 1;
                c = (int8_t)va_arg(ap, int); // promoted
                *buf++ = (c>>0)&0xff;
                break;
            case 's': // string
                s = va_arg(ap, char*);
                len = strlen(s);
                size += len + 2;
                packi16(buf, len);
                buf += 2;
                memcpy(buf, s, len);
```

```
                buf += len;
                break;
            }
        }
        va_end(ap);
        return size;
    }

    unsigned int unpacki16(char *buf)
    {
        return (buf[0]<<8) | buf[1];
    }

    void unpack(char *buf, char *format, ...)
    {
        va_list ap;
        int16_t *h;
        int8_t *c;
        char *s;
        int32_t len, count, maxstrlen=0;
        va_start(ap, format);
        for(; *format != '\0'; format++) {
            switch(*format) {
                case 'h': // 16-bit
                    h = va_arg(ap, int16_t*);
                    *h = unpacki16(buf);
                    buf += 2;
                    break;
                case 'c': // 8-bit
                    c = va_arg(ap, int8_t*);
                    *c = *buf++;
                    break;
                case 's': // string
                    s = va_arg(ap, char*);
                    len = unpacki16(buf);
                    buf += 2;
                    if (maxstrlen > 0 && len > maxstrlen) count = maxstrlen - 1;
                    else count = len;
                    memcpy(s, buf, count);
                    s[count] = '\0';
                    buf += len;
                    break;
                default:
                    if (isdigit(*format)) { // track max str len
                        maxstrlen = maxstrlen * 10 + (*format-'0');
                    }
            }
            if (!isdigit(*format)) maxstrlen = 0;
        }
        va_end(ap);
    }

    int writen(int socketfd, int flag) {
    #if debug
        printf("inside writen\n");
    #endif
        //int string_len;
        fgets(input_string,100,stdin);
```

```
    string_len = strlen(input_string) -1;
    //msg = strtok(msg, "\n");
    if (input_string[string_len] == '\n')
        input_string[string_len] = '\0';
    msg = &input_string[0];
#if debug
    printf("Client %s: %s \n",name,msg);
#endif
    return 0;
}
int sendall(int socket_id, char *buf, int length)
{
    int cnt = 0;
    int bytesleft = length;
    int bytessent;
    while(cnt < length)
    {
        bytessent = send(socket_id, buf+cnt, bytesleft, 0);
        if (bytessent == -1)
        {
            break;
        }
        cnt = cnt + bytessent;
        bytesleft = bytesleft - bytessent;
    }
    length = cnt;
    return bytessent==-1?-1:0; // return -1 on failure, 0 on success
}
int readline(int socketfd, char buffer[], int receiving_size, int flag) {
#if debug
    printf("inside readline");
#endif
    int receive_from_ack = recv(socketfd, buffer, receiving_size, flag);
#if debug
    printf("after readline");
#endif
    if(receive_from_ack < 0)
    {
#if debug
        printf("receiving error");
#endif
        perror("error while receiving");
    }
    buffer[receive_from_ack] = '\0';

unpack(buffer,"cchhhs",&sbcp_msg_rcv.vrsn,&sbcp_msg_rcv.type,&sbcp_msg_rcv.l
ength,&sbcp_attr_rcv.type,&sbcp_attr_rcv.length,buff_rcv);
    //put case statement here
    if(sbcp_msg_rcv.vrsn=='3')
    {
        if(sbcp_msg_rcv.type==FWD)
        {
            printf("FWD MESSAGE> ");
            fputs(buff_rcv,stdout);
            printf("\n");
        }
        if(sbcp_msg_rcv.type==ONLINE)
        {
```

```
            printf("ONLINE MESSAGE> ");
            fputs(buff_rcv,stdout);
            printf("\n");
        }
        if(sbcp_msg_rcv.type==OFFLINE)
        {
            printf("OFFLINE MESSAGE> ");
            fputs(buff_rcv,stdout);
            printf("\n");
        }
        if(sbcp_msg_rcv.type==ACK)
        {
            printf("ACK MESSAGE> ");
            fputs(buff_rcv,stdout);
            printf("\n");
        }
        if(sbcp_msg_rcv.type==NAK)
        {
            printf("NAK MESSAGE> ");
            fputs(buff_rcv,stdout);
            printf("\n");
            exit(1);
        }
        if(sbcp_msg_rcv.type==IDLE)
        {
            printf("IDLE MESSAGE> ");
            fputs(buff_rcv,stdout);
            printf("\n");
        }
    }



#if debug
    printf("\nClient Received: %s \n",buff_rcv);
    printf("\n at the end\n");
    printf("\nClient Received:");
#endif
    return receive_from_ack;
}

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

#if debug
    printf("\nEnter text\n");

#endif
    if(argc != 4)
    {
        printf("enter valid args\n");
    }
    name = argv[1];
    address = argv[2];
    port = argv[3];
    server.ai_family = AF_UNSPEC; //can be AF_INET
    server.ai_socktype = SOCK_STREAM;
    server.ai_flags = AI_PASSIVE;
```

```
    status = getaddrinfo(address, port, &server, &server_ptr);
    if(status !=0) {
#if debug
        printf("\nunable to connect to ip address %s\n",address);
#endif
        exit(0);
    }
    addrinfo *ptr_copy;
    ptr_copy = server_ptr;
    for(ptr_copy = server_ptr; ptr_copy != NULL; ptr_copy= ptr_copy-
>ai_next) {
        struct sockaddr_in *ptr_with_address = (struct sockaddr_in *)
ptr_copy->ai_addr;
        address_ptr_void = &(ptr_with_address->sin_addr);
        socketfd = socket(server_ptr->ai_family, server_ptr->ai_socktype,
server_ptr->ai_protocol);
        if (socketfd == -1) {
#if debug
            printf("\nError at the socket call\n");
#endif
            perror("Error at socket");
            continue;
        }

        //printf("before connect ack");
        int connect_ack = connect(socketfd,server_ptr->ai_addr, server_ptr-
>ai_addrlen); //returns negitive value on failure to connect
        //printf("after connect ack");
        if(connect_ack == -1) {
#if debug
            printf("\ncould not connect\n");
#endif
            perror("Problem with connect");
            //exit(0);
        }
        break;
    }
    //printf("before init");
    strcpy(username_of_client , name);
    sbcp_attr.payload_attribute = username_of_client;
    sbcp_attr.type = 2;
    sbcp_attr.length = 20;
    sbcp_msg.vrsn = '3';
    sbcp_msg.type = 2;
    sbcp_msg.length = 24;
    sbcp_msg.payload = &sbcp_attr;
    size_of_packet = pack(buff, "cchhhs", sbcp_msg.vrsn, sbcp_msg.type,
sbcp_msg.length, sbcp_attr.type, sbcp_attr.length,
sbcp_attr.payload_attribute);
    //printf("after init");
    if(sendall(socketfd, buff, size_of_packet)==-1)
        //if(send(socketfd, buff, size_of_packet, flag) < 0)
    {
        printf("error sending username to client\n");
        perror("send usename error");
        exit(0);
    }
    //writen(socketfd, flag);
```

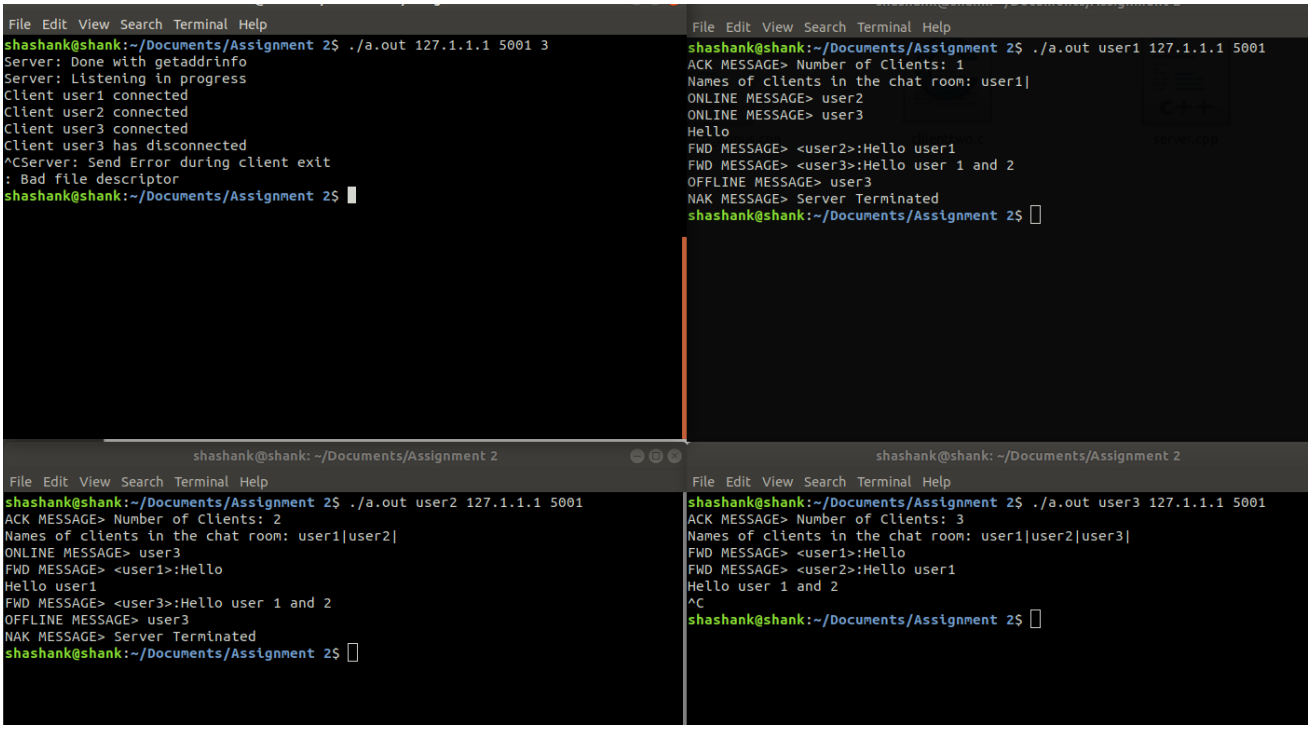
```
    sbcp_attr.payload_attribute = input_string;
    sbcp_attr.type = 4;
    sbcp_attr.length = 516;
    sbcp_msg.vrsn = '3';
    sbcp_msg.type = 4;
    sbcp_msg.length = 520;
    sbcp_msg.payload = &sbcp_attr;
    int STDIN = 0;
    FD_SET(STDIN , &readfds);
    FD_SET(socketfd , &readfds);
#if debug
    printf("socketfd %d",socketfd);
#endif

    time_count.tv_sec = 10;
    time_count.tv_usec = 0;
    int idle = 0;

    while(1)
    {
        int select_value;
        select_value = select(socketfd+1, &readfds, NULL,NULL, &time_count);
        if(select_value < 0)
        {
            printf("error in select\n");
            exit(0);
        }
        if(!(FD_ISSET(STDIN,&readfds)) && time_count.tv_sec == 0 && idle ==
0){
            //strcpy(sbcp_attr.payload_attribute,"\0");
            sbcp_attr.type = NULL;
            sbcp_attr.length = 0;
            sbcp_msg.vrsn = '3';
            sbcp_msg.type = 9;
            sbcp_msg.length = 4;
            sbcp_msg.payload = &sbcp_attr;
            size_of_packet = pack(buff, "cchhh", sbcp_msg.vrsn,
sbcp_msg.type, sbcp_msg.length, sbcp_attr.type, sbcp_attr.length,
input_string);
            //packet = sbcp_to_string(VERSION,IDLE,0,NULL);
            int sendto_ack = send(socketfd, buff, size_of_packet, flag);
//checking for any error while sending
            if(sendto_ack <= -1)
            {
                printf("\nerror sending\n");
                exit(0);
            }
            time_count.tv_sec = 10;
            printf("You are idle\n");
            idle = 1;
            FD_SET(STDIN , &readfds);
            FD_SET(socketfd , &readfds);
        };
        for(i_return = 0 ; i_return <= socketfd ; i_return++)
        {
            if(FD_ISSET(i_return,&readfds))//if(FD_ISSET(socketfd,&readfds))
            {
                idle = 0;
```

```
#if debug
                //strtok(msg, "\n");
#endif

#if debug
                printf("i1 %d\n",i_return);
#endif
                if(i_return == 0)
                {
#if debug
                    printf("before writen\n");
#endif
                    time_count.tv_sec = 10;
                    idle = 0;
                    writen(socketfd, flag); // function call to write
                    size_of_packet = pack(buff, "cchhhs", sbcp_msg.vrsn,
sbcp_msg.type, sbcp_msg.length, sbcp_attr.type, sbcp_attr.length,
input_string);
#if debug
                    printf("msg after pack %s\n",msg);
#endif
                    sbcp_attr.payload_attribute = input_string;
                    sbcp_attr.type = 4;
                    sbcp_attr.length = 516;
                    sbcp_msg.vrsn = '3';
                    sbcp_msg.type = 4;
                    sbcp_msg.length = 520;
                    sbcp_msg.payload = &sbcp_attr;
                    int sendto_ack = sendall(socketfd, buff,
size_of_packet); //checking for any error while sending
                    if(sendto_ack <= -1)
                    {
                        printf("\nerror sending\n");
                        exit(0);
                    }
                    //time_count.tv_sec = 10;
                    //idle = 0;
#if debug
                    printf("i_return %d and socketfd %d
\n",i_return,socketfd);
#endif
                }

#if debug
                printf("i1 %d\n",i_return);
#endif
                if(i_return == socketfd)
                {
                    int receiving_data =
readline(socketfd,buffer,receiving_size, flag); // function call to read
                    if(receiving_data == -1) {
                        printf("\nerror receiving\n");
                    }
                }//printf("I am going out of FD_ISSET\n");
            }FD_SET(0, &readfds);
            FD_SET(socketfd, &readfds);
#if debug
            printf("socketfd %d i_return %d\n",socketfd, i_return);
```

```
#endif
        }time_count.tv_sec = 10;
    }
    close(socketfd);
    freeaddrinfo(server_ptr);
    return 0;
}
```

TEST CASES:

1. Normal operation of the chat windows:

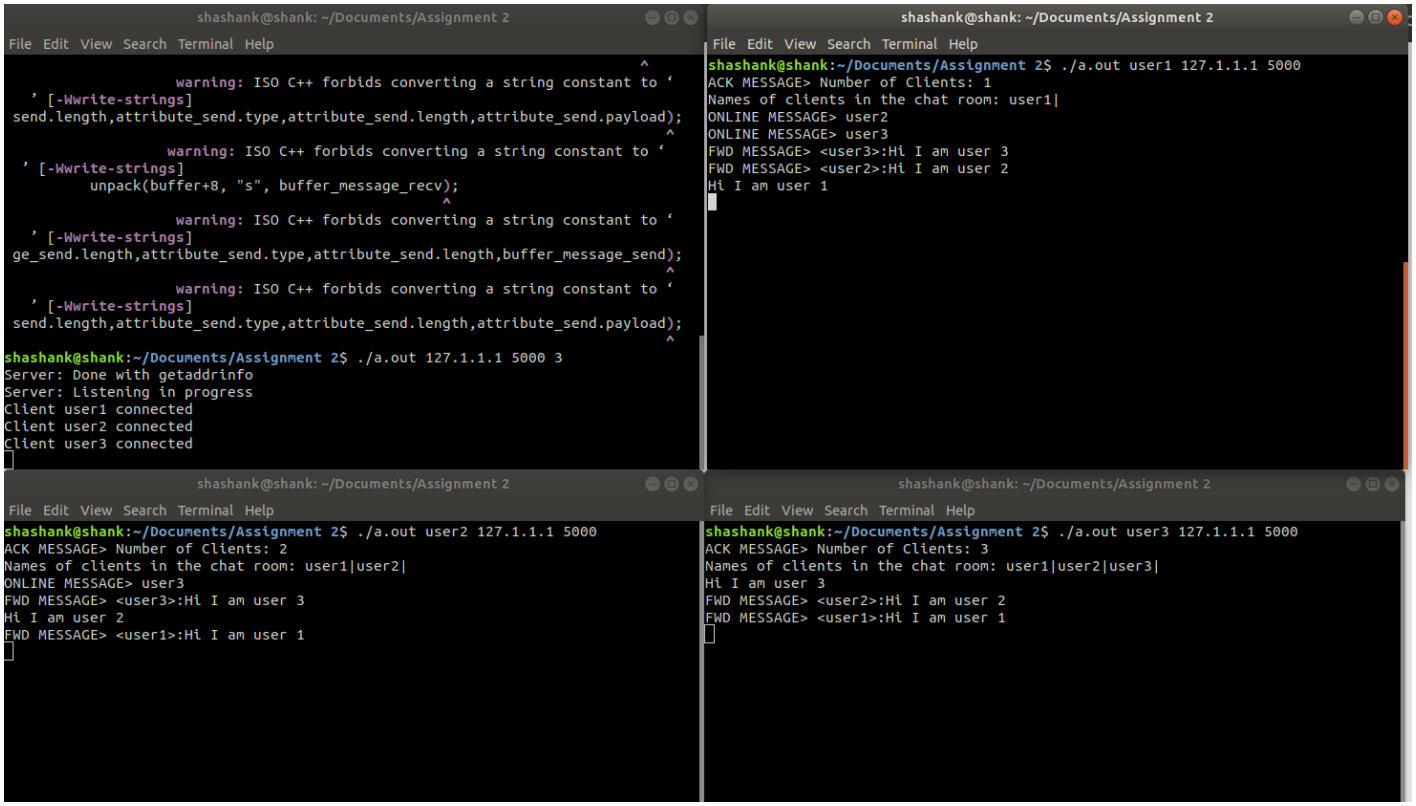


In the above figure, we can see that 3 clients are connected to the server. This is the full execution of the code and the execution will be explained in detail in the following testcases.

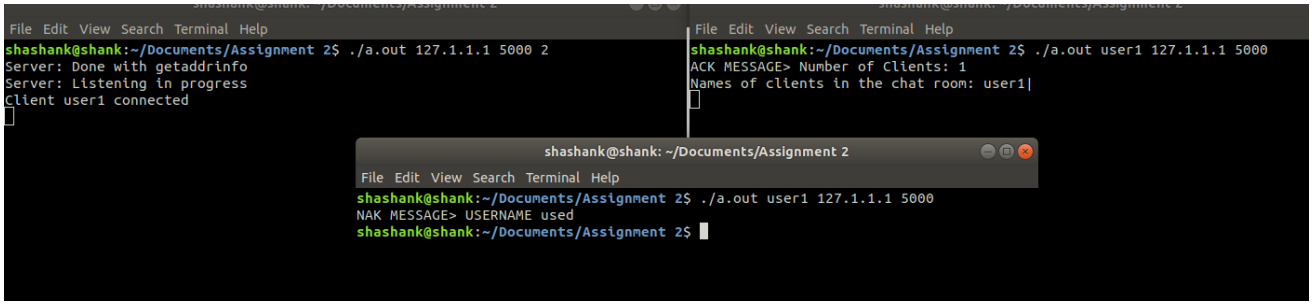
2. 3 Clients connected to the server:

In the screenshot there are 3 clients connected to the server at ip address 127.0.0.2 and port number 5000. The test begins with all the 3 clients trying to connect to the server without any the hinderance. The result of the first phase of the test is displayed by the server “client user 1 connected”, “client user 2 connected” and “client user 3 connected”. The second phase of the testing starts with client user3 sending a text “I am user 3”, this text should be displayed by both the clients (user 2 and user 3). The screenshot suggests that the second phase of testing was successful, and the clients

could successfully display the text. The above experiment is conducted on remaining two clients to verify the normal operations and functionalities of client and the server.



3. Server rejects a client with a duplicate username:

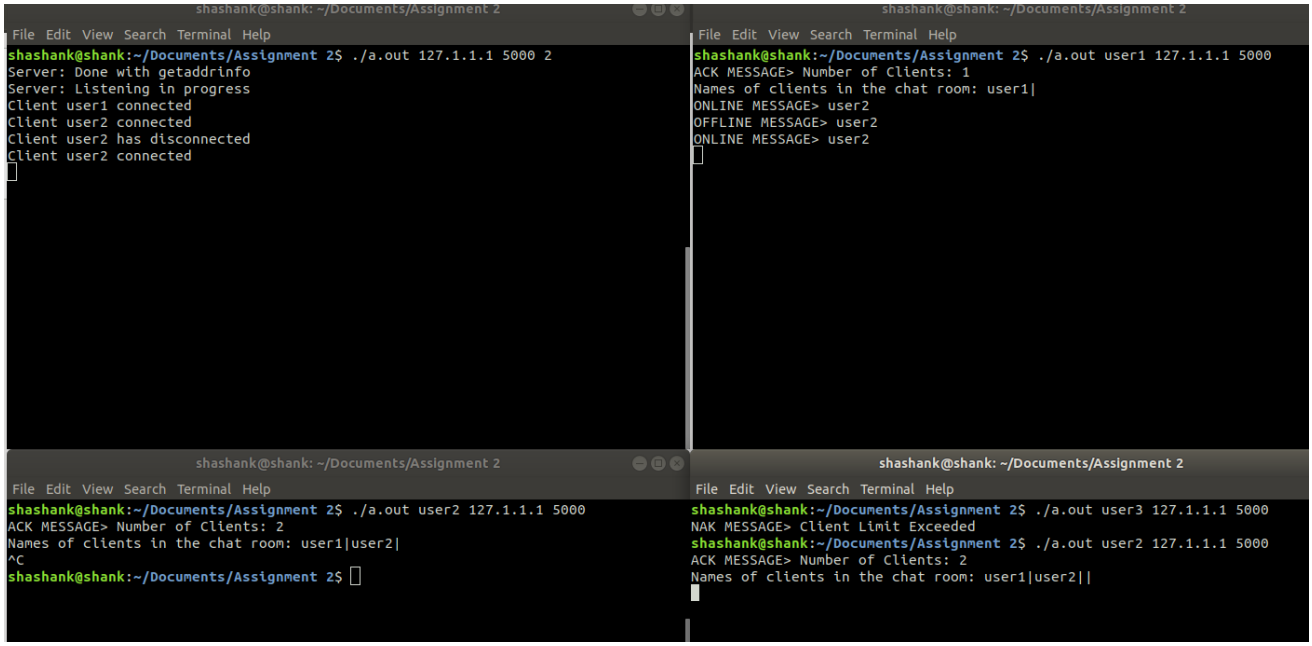


This test-case proves the fact that the server will not accept two connections with same username. The above screenshot suggests the same, initially client user 1 got connected to the server and after the connection was acknowledged (*ACK*) by the server another client with same username tried to connect to the server. In the later case the server denied the connection and sent a *NAK* back to the client with duplicate username.

Whenever the server accept the connection the server sends a *ACK* to the client, acknowledging the successful connection between the client and server for a given username. In other cases where the connection between the server and client fails, the

server sends a *NAK* back to the client and also sends the reason for rejection. In the above case the server has denied the connection and the reason stated is “USERNAME USED”.

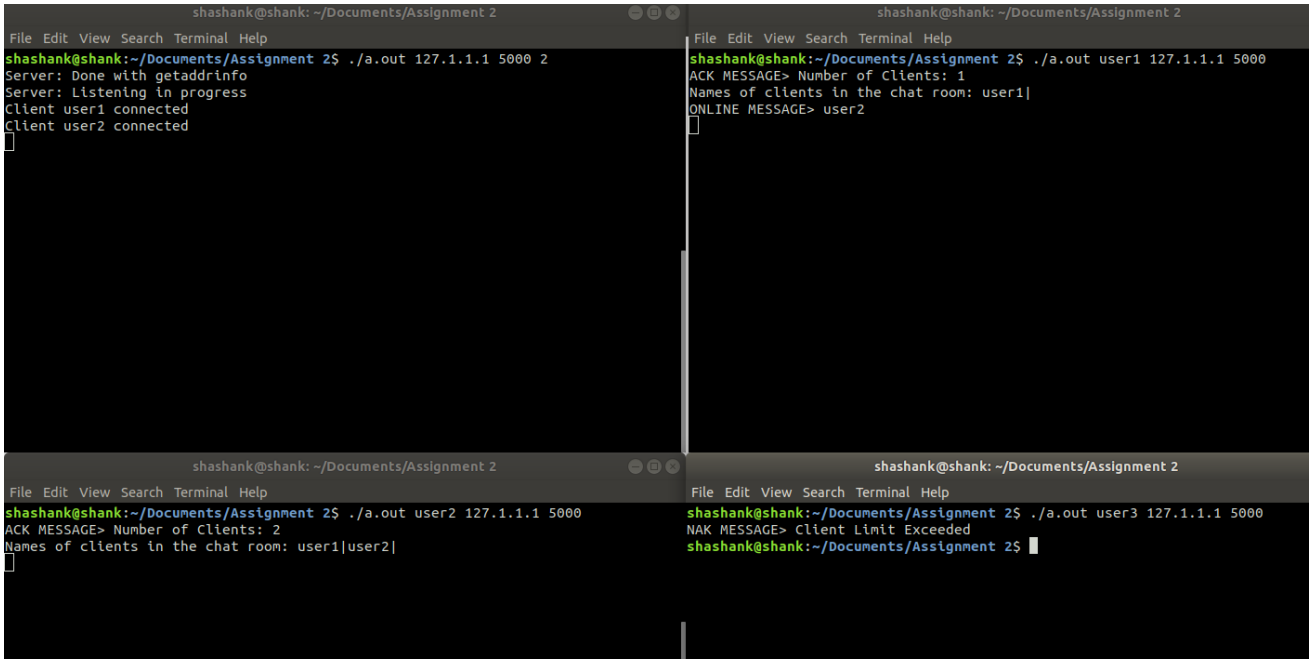
4. Server allows a previously used username to be reused:



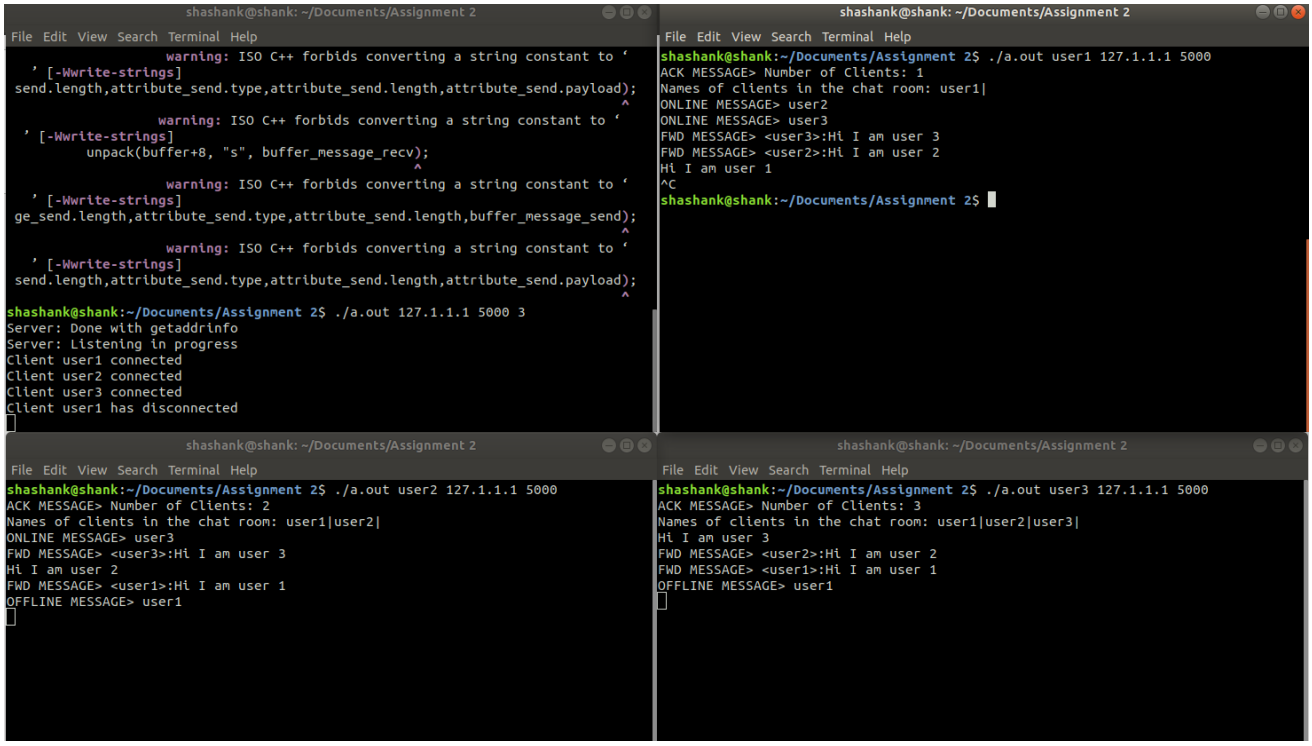
This test case is an extension of the previous two test cases which signifies the fact that the server has adopted functionalities wherein a client username can be reused under special conditions. In the above screenshot attached, client username “user 2” is reused. Initially the client “user 2” is connected to the server and after some time “t” “user 2” left the connection, then a new user coming in after time “t” can reuse the client name, in the above case it is “user 2”.

5. Server rejects the client because it exceeds the maximum number of clients:

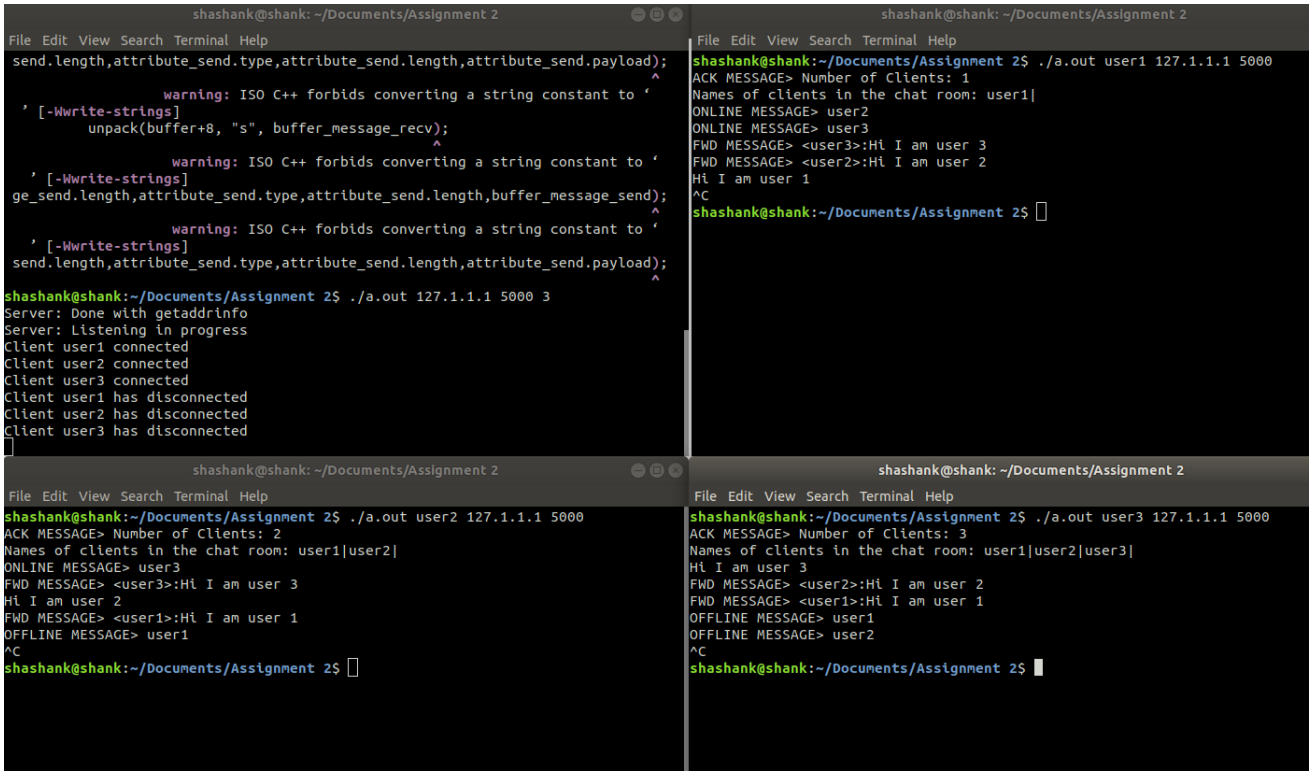
The designed server can only handle a specific number of connections “n” and this number is fed in as an input initially through the terminal. If the number of clients exceed this number “n” then the server will not accept the connections. That is the server send a *NAK* with the reason stating that the number of clients have exceeded the maximum number of clients the server can handle. In the above case the server supported a maximum of 2 connections and the the third user was rejected and with a reason “CLIENT LIMIT EXCEEDED”



6. Implementation of FEATURE 1- ACK, NAK, ONLINE, OFFLINE:

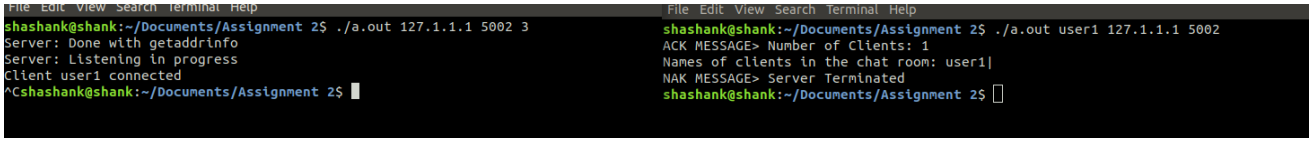


The bonus feature is implemented here and the above screenshot shows the same. 3 clients are connected and as each client gets connected the rest of the client already in the chat room get a ONLINE message which indicates the arrival of a new client. Similarly when a client leaves a chatroom, OFFLINE message is sent to the rest of the clients. The ACK and NAK messages are discussed in the previous test cases.



This screenshot indicates that as each client enters and exits the chatroom, correspondingly ACK, ONLINE and OFFLINE messages are sent.

7. Server exiting unceremoniously:



When the server exits due to input of CTRL+C, the clients connected to the server receive a “Server Terminated” message and the client program also exits.

8. IDLE time

The screenshot below demonstrates the idle time scenario, when a user does not send or receive anything for a given time t then it is called to be idle in the network. The client recognises this and sends a idle packet to the server and the server notifies all the other clients about the idle state of that client.

```
sumukh110@sumukh110-Inspiron-3537:~/sumukh/ccn/csce602/assignment2/test/final$ ./server 127.0.0.2 9600 5
Server: Done with getaddrinfo
Server: Listening in progress
Client user1 connected
Client user2 connected
```

```
sumukh110@sumukh110-Inspiron-3537: ~/sumukh/ccn/csce602/assignment2/test/final$ ./client user2 127.0.0.2 9600
ACK MESSAGE> Names of clients in the chat room: user1|user2|
hi user2
FWD MESSAGE> <user1>:hi
so username
IDLE MESSAGE> <user1>:is IDLE
IDLE MESSAGE> <user1>:is IDLE
idYou are idle
idle?
FWD MESSAGE> <user1>:user1
FWD MESSAGE> <user1>:sorry
why idle
FWD MESSAGE> <user1>:waiting too long
```

```
sumukh110@sumukh110-Inspiron-3537: ~/sumukh/ccn/csce602/assignment2/test/final$ ./client user1 127.0.0.2 9600
ACK MESSAGE> Names of clients in the chat room: user1|
ACK MESSAGE> Names of clients in the chat room: user1|user2|
FWD MESSAGE> <user2>:hi user2
hi
FWD MESSAGE> <user2>:so username
is userYou are idle
1
user1
IDLE MESSAGE> <user2>:is IDLE
IDLE MESSAGE> <user2>:is IDLE
sorry
FWD MESSAGE> <user2>:why idle
waiting too long
```