**ECEN 602**
**Network Programming Assignment # 02**

**Team No: 1**
**Ishan Tyagi**
**Sushrut Kaul**

Contributions:
Ishan did the Client code and tests.
Sushrut did the Server code.

Bonus features in this code include:-
a. Server and Client side IPv4/IPv6 implementaion.
b. ACK,NAK,ONLINE,OFFLINE attributes.
c. Server and Client side IDLE(>10 seconds) feature.

The package contains 5 files:
1.chatc.c
2.chats.c
3.README.txt
4.Makefile
5.Test Cases.pdf

To compile the code, run the **Makefile: make**
Run the server by using the command line: **./chats server_ip server_port max_no_of_clients**
Run the client by using the command line: **./chatc username server_ip server_port**
To clean the executables **run: make clean**

**SERVER:**

Both the server and the client have been written in C.

The server accepts both a IPv4 or IPv6 IP address. Also the client can connect using corresponding IPv4 or IPv6 server IP address.
A master fd set is created to hold connected fds including listener.
Then select() is called to know if any file descripter is ready to recv() data.when a new connection is requested, it's accepted and passed through a check() function to get validated and modify attributes accordingly.

The server send following message types:-
1. NAK - If max no clients online or username already exists, a NAK message is sent to the client and connection closed.
2. ACK - List of usernames ONLINE and Total no of clients is sent to the client.
3. ONLINE - When a new client is acknowledged in the chatroom, ONLINE message goes to all other client
4. OFFLINE - When a client leaves chatroom, OFFLINE message goes to all other client.

5. IDLE - When a client is idle more than 10 seconds after sending message, it sends an IDLE message to server.
The server then sends the message to all other clients.
6. FWD - Server forwards message sent a client to all other clients.
The server uses a clear() function to remove resources when a client leaves chat session. Also the clear() function closes any client which sends any wrong attribute messages.

CLIENT:

The script itself can be run into several parts: header and socket initialization, connecting to the server, and subsequent chat-related events and responses.

The bulk of the socket initialization is of the same procedure as of the previous assignment, save for the fact that the socket was set up as IPv4 or IPv6 compatible as to keep with current practices.

Once connected, the setup for usage of the select() function began, followed by the main loop of code pertaining to the actual chat client.

The Client begins by sending the JOIN case to the server, with the appropriate header values chosen. This section is followed by the prompt to send a message.

However, if the channels pertaining to receiving a message trigger instead, the select() function chooses to enable the receive functionality, at which the header values are compared against one another to determine what type of message was received. These messages are then processed into terminal messages in readable format.

The optional features (ACK, NAK, etc.) were included and accounted for in this code.

In addition, the IDLE feature was included. When a client is idle for more than 10 seconds (calculated using time() function), it sends an idle message to server and server in turn forward it to everyone.

/*

```
Program Name:
Author Name : Sushrut Kaul , Ishan Tyagi
Department of Electrical and Computer Engineering , Texas A&M
University
*/


/*
Include the header files
*/

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

struct SBCP_attribute{
      uint16_t type;
      uint16_t length;
      uint16_t client_count;
      char reason[32];
      char *attr_payload;
};

struct SBCP_message{
      uint8_t vrsn; // we have assigned a bit field to version.vrsn=3
in our case
      uint8_t type; //assign type=2 for JOIN,3 for forward and 4 for
send
      uint16_t length;
      struct SBCP_attribute *attribute;
};

/*The following function is used to write n bytes.
This function offers an advantage over the standard
write method call. It sends 'n' bytes while the
standard write can sometimes send less and that
condition is not an error.
*/

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

```
}

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

int32_t pack(char *buf, char *format, ...)
{
    va_list ap;
    int16_t h;
    int8_t c;
    char *s;
    int32_t size = 0, 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;
}


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);
}



void init_sbcp_attribute(struct SBCP_attribute *struct_ptr, int
type_attr ,char *ptr_to_buffer,int len){
           (struct_ptr->type)= type_attr;
           (struct_ptr->attr_payload)= ptr_to_buffer;
           struct_ptr->length= 4+len;
}

void init_sbcp_message(struct SBCP_message *struct_msg_ptr,int
version,int type_msg,struct SBCP_attribute *attr123){
           (struct_msg_ptr->vrsn)=version;
           (struct_msg_ptr->type)= type_msg;
           (struct_msg_ptr->attribute)=attr123;
           (struct_msg_ptr->length)=(attr123->length) +4;
}



void sigchld_handler(int s){
      int saved_errno= errno;
```

```
       while(waitpid(-1,NULL,WNOHANG )>0);
       errno =saved_errno;
}

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

void err_sys(const char* x){
         perror(x);
          exit(1);
}


#define MAXDATASIZE 100
char active_users_string[160];
char existing_usernames[10][16];
char list[10][16];
int BACKLOG;  // Backlog is the maximum number of the clients that can
connect
int main(int argc,char *argv[]) {


if(argc!=4) {
      printf("Wrong format :::: Enter in the following order : ./server
localhost PORT-NO Max_client\n");
      }

   BACKLOG=atoi(argv[3]);
      printf("%d\n",BACKLOG);
//Variables definitions begin
fd_set master_set;
fd_set read_set;
int fdmax,i,j;  //maximum file descriptor number
int sockfd,new_fd,numbytes,bytes_sent;
 char buf[MAXDATASIZE];
struct addrinfo hints, *servinfo , *p;
 struct sockaddr_storage their_addr;//connector's address
 socklen_t sin_size;
 struct sigaction sa;
 int yes=1;
int val=0;
uint16_t count=0;
 char s[INET6_ADDRSTRLEN];
 int rv,size,k=0;
int t=0;
char username_buffer[16];
char recv_buffer[25];
char reject_message[512];
char online_message[100]="Our newest member is:";
char offline_message[100]="The following user has left us
unceremoniously:";
char my_buffer[16];
char message_buffer[530];
  struct SBCP_message sbcp_msg;
  struct SBCP_attribute sbcp_attr1;
  struct SBCP_attribute sbcp_attr2;
  struct SBCP_message sbcp_msg1;
  struct SBCP_attribute sbcp_attr11;
// Variables definitions end
char user[16];
int g,g1,x;

int comp_res;



memset(&hints,0,sizeof hints);
hints.ai_family =AF_UNSPEC;// We are not specifying the ADDRESS TYPE
hints.ai_socktype =SOCK_STREAM;// Using the Stream Socket
hints.ai_flags =AI_PASSIVE ; // AI_PASSIVE REFERS TO "MY-IP ADDRESS"

FD_ZERO(&master_set);
FD_ZERO(&read_set);





if((rv =getaddrinfo(NULL,argv[2],&hints,&servinfo )) !=0) {
      fprintf(stderr,"getaddrinfo: %s\n",gai_strerror(rv));

      return 1;
}


/*ai_next refers to the pointer to the next node
in the linked-list that we have obtained
*/



for(p=servinfo ;p!=NULL ; p->ai_next ) {
      if((sockfd = socket(p->ai_family , p->ai_socktype, p-
>ai_protocol))==-1){
            err_sys("server :socket");
            continue;
      }

if(setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int))==-1) {
```

```
            err_sys("setsockopt");

            exit(1);
}
if(bind(sockfd,p->ai_addr,p->ai_addrlen)==-1) {
            close(sockfd);

            err_sys("server:bind");

            continue;
}
 break    ; // break when you get a usable configuration
}
      freeaddrinfo(servinfo) ;//servinfo is no longer required

if(p==NULL) {
      fprintf(stderr,"server:failed to bind\n") ;

exit(1);
 }
if(listen(sockfd,BACKLOG) ==-1) {
            err_sys("listen");
            exit(1);

}
sa.sa_handler =sigchld_handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags= SA_RESTART ;
if(sigaction(SIGCHLD ,&sa,NULL)==-1) {

err_sys("sigaction");
exit(1);
}


 printf("server waiting for connections...\n");

//add listener socket to the master set
FD_SET(sockfd,&master_set);

fdmax = sockfd; //sockfd is the biggest value till now
while(1) {
     read_set=master_set;
     if(select(fdmax+1,&read_set,NULL,NULL,NULL) ==-1) {
      perror("select");
      exit(4);
       }


  for(i=0;i<=fdmax;i++) {
      if(FD_ISSET(i,&read_set)) {
            if(i==sockfd) {
                sin_size=sizeof their_addr;
                new_fd=accept(sockfd,(struct sockaddr
*)&their_addr,&sin_size);

                if(new_fd==-1) {
                      perror("accept");
                }
                else {
                     FD_SET(new_fd,&master_set);
                     if(new_fd>fdmax) {
                          fdmax=new_fd;
                     }
                     printf("selectserver :new connection from %s on
socket %d\n",inet_ntop(their_addr.ss_family,get_in_addr((struct
sockaddr *)&their_addr),s,INET6_ADDRSTRLEN),new_fd);
                 }
              }

           else {
                if((numbytes =recv(i,buf,sizeof buf,0)) <=0) {
                     int f,k1,z1;int flag=1; int h;
                     //printf(" The number of bytes are
%d\n",numbytes);
                     if(numbytes ==0) {
                          for(h=0;h<i;h++) {
                               if(!(strcmp(list[h],username_buffer)))
{
                                    flag=0;
                               }
                          }
                               if(flag!=0) {
                               for(g1=0;g1<=fdmax;g1++) {

     if(FD_ISSET(g1,&master_set)) {
                                                     if(g1!=sockfd &&
g1!=i) {
                                          size=
pack(buf,"cchhhhshhs",'3','6',sbcp_msg.length,2,sbcp_attr1.length,coun
t,list[i],4,sbcp_attr2.length,offline_message);


     if(send(g1,buf,size,0) ==-1) {
                                                     perror("send");
                                                 }
                                                 }
                                                 }
                                                 }
                          strcpy(user,list[i]);
                          count--;
                          for(f=0;f<10;f++) {

     if(!strcmp(existing_usernames[f],user)) {
```

```
                                for(k1=f;k1<9;k1++) {

        strcpy(existing_usernames[k1],existing_usernames[k1+1]);
                                }
                            }
                        }
                        //for(z1=0;z1<=9;z1++) {
                            //
        printf("%s\n",existing_usernames[z1]);
                            //}
                        memset(list[i],0,sizeof(list[i]));
                        //printf("selectserver:socket %d hung
up\n",i);
                    }

                    }
                    else {
                        perror("recv");
                    }
                    close(i);
                    FD_CLR(i,&master_set);
                }
                else {
                    //printf("I am here\n");

        unpack(buf,"cchhhshhs",&sbcp_msg.vrsn,&sbcp_msg.type,&sbcp_msg.le
ngth,&sbcp_attr1.type,&sbcp_attr1.length,username_buffer,&sbcp_attr2.t
ype,&sbcp_attr2.length,message_buffer);
                    //printf("%c\n",sbcp_msg.type);
                    //printf("%s\n",recv_buffer);
                    //printf("This is my username_buffer
:%s\n",username_buffer);

                    if(sbcp_msg.type=='2')  { // this means that you
have got a send message on your hand
                                int u,current_pos;
                                  comp_res=0;
                                int t9=0;
                            //printf("I am here\n");int z;

                            //printf("count is=%d\n",count);
                            //strcat(welcome_message,recv_buffer);
                            //printf("%s",welcome_message);

                            if(count==0) {

        strcpy(existing_usernames[0],username_buffer);
                        strcpy(list[i],username_buffer);
                        current_pos=1;
char welcome_message[512]="From now on, you will be recognized as";
```

```
                          //        printf("count second
:%d",count);

pack(buf,"cchhhhshhs",'3','7',sbcp_msg.length,2,sbcp_attr1.length,coun
t,username_buffer,4,sbcp_attr2.length,welcome_message);
                          //        printf("count again
:%d\n",count);
                                    t9=1;
                                    count++;
                          //        printf("Count bhosdi %d
\n",count);
                               }

                                             else {

                                    for(u=0;u<10;u++) {

if(!strcmp(existing_usernames[u],username_buffer)) {
                                          comp_res=10;
                                           }
                                      }
                                 }


                                 if(t9!=1) {
                               if(comp_res!=10 && count <BACKLOG) {

     strcpy(list[i],username_buffer);

     strcpy(existing_usernames[current_pos],username_buffer);


     //printf("%s\n",username_buffer);
                                          current_pos++;
                                          count++;
                                          for(x=0;x<=9;x++) {

     strcat(active_users_string,existing_usernames[x]) ;

     strcat(active_users_string," ");

                                          }

     //printf("%s\n",active_users_string);
                                          for(g=0;g<=fdmax;g++) {

     if(FD_ISSET(g,&master_set)) {
                                                  if(g!=sockfd &&
g!=i) {
                                         size=
pack(buf,"cchhhhshhs",'3','8',sbcp_msg.length,2,sbcp_attr1.length,coun
t,username_buffer,4,sbcp_attr2.length,online_message);
```

```
                                        if(send(g,buf,size,0) ==-1) {
                                                        perror("send");
                                                 }
                                                 }
                                                 }
                                                 }


                                            char
welcome_message[512]="Online users in the system are:\n";

     pack(buf,"cchhhhshhs",'3','7',sbcp_msg.length,2,sbcp_attr1.length
,count,active_users_string,4,sbcp_attr2.length,welcome_message);

     memset(active_users_string,0,sizeof(active_users_string));


                                }
                                else{
                                            if(count>=BACKLOG) {

     strcpy(reject_message,"User limit reached....");

                                            }
                                            if(comp_res==10){

     strcpy(reject_message,"User with the same name already
exists....");
                                             }



pack(buf,"cchhhhshhs",'3','5',sbcp_msg.length,2,sbcp_attr1.length,coun
t,username_buffer,4,sbcp_attr2.length,reject_message);


                                   }
                                }
                                //printf("Before removing:\n");
                                //for(z=0;z<=9;z++) {
                                //
     printf("%s\n",existing_usernames[z]);
                                //}
                                //printf("%d",count);
                                if(send(i,buf,100,0)==-1) {
                                                perror("send");
                                            }




                      }
                      if(sbcp_msg.type=='9') {
                           for(j=0;j<=fdmax;j++) {
                                if(FD_ISSET(j,&master_set)) {
                                     if(j!=sockfd && j!=i) {

     init_sbcp_attribute(&sbcp_attr1,2,username_buffer,16);

     init_sbcp_message(&sbcp_msg,'3','9',&sbcp_attr1);

     init_sbcp_attribute(&sbcp_attr2,4,message_buffer,530);
                size=pack(buf,"cchhhhshhs", sbcp_msg.vrsn,
sbcp_msg.type,sbcp_msg.length,
sbcp_attr1.type,sbcp_attr1.length,count,username_buffer,sbcp_attr2.typ
e,sbcp_attr2.length,message_buffer);


                                          if(send(j,buf,size,0) ==-1)
{
                                               perror("send");
                                          }
                                     }
                                     }
                           }
                      }



                       if(sbcp_msg.type=='4') {
                           //printf("I am here at message 3\n");
                           for(j=0;j<=fdmax;j++) {
                                if(FD_ISSET(j,&master_set)) {
                                     if(j!=sockfd && j!=i) {

     init_sbcp_attribute(&sbcp_attr1,2,username_buffer,16);

     init_sbcp_message(&sbcp_msg,'3','3',&sbcp_attr1);

     init_sbcp_attribute(&sbcp_attr2,4,message_buffer,530);
                size=pack(buf,"cchhhhshhs", sbcp_msg.vrsn,
sbcp_msg.type,sbcp_msg.length,
sbcp_attr1.type,sbcp_attr1.length,count,username_buffer,sbcp_attr2.typ
e,sbcp_attr2.length,message_buffer);


                                          if(send(j,buf,size,0) ==-1)
{
                                               perror("send");
                                          }
                                     }
                                     }
                           }
                      }
```

```
                  }//else against i==sockfd
              }
             }
        }
      }
      return 0;
      }
```

Client:

```
*
 * Program Name: Client.c
 * Authors: Sushrut Kaul, Ishan Tyagi
 * Department of Electrical and Computer Engineering
 * Texas A&M University,College Station
 *
*/

/*
 * Standard header file declarations
 *
*/

#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<ctype.h>
#include<stdarg.h>

#define DATA_SIZE_LIMIT 100 // set limit to the Data size to be handled
#define MESSAGE_LEN_LIMIT 512 //512 Bytes is the maximum message length
allowed



//structures to be used
        //struct {
        //char username[16];
        //char message[512];
        //char reason[32];
        //unsigned int client_count :16;
        //}attribute_payload;

        struct SBCP_attribute{
        uint16_t type;
        uint16_t length;
        char *attr_payload;
        };

        struct SBCP_message{
        uint8_t vrsn; // we have assigned a bit field to version.vrsn=3 in
our case
        uint8_t type; //assign type=2 for JOIN,3 for forward and 4 for send
        uint16_t length;
        struct SBCP_attribute *attribute;
        };

        void packi16(unsigned 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, 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);
}


//write methods to initialize your structures
//following is a method to initialize my SBCP_attribute structure

	void init_sbcp_attribute(struct SBCP_attribute *struct_ptr, int
type_attr ,char *ptr_to_buffer,int len)
	{
		(struct_ptr->type)= type_attr;
		(struct_ptr->attr_payload)= ptr_to_buffer;
		struct_ptr->length= len;
	}
```

```
        void init_sbcp_message(struct SBCP_message *struct_msg_ptr,int
version,int type_msg,struct SBCP_attribute *attr123,int len)
        {
                (struct_msg_ptr->vrsn)=version;
                (struct_msg_ptr->type)= type_msg;
                (struct_msg_ptr->attribute)=attr123;
                (struct_msg_ptr->length)=len;
        }

/* Step 1 : Create static variables to maintain the state across calls */

static char my_buffer[DATA_SIZE_LIMIT];
static char *buffer_pointer;
static int  buffer_count;


void err_sys(const char* x)
{
        perror(x);
        exit(1);
}



/*The following function is used to write n bytes.
  This function offers an advantage over the standard
  write method call. It sends 'n' bytes while the
  standard write can sometimes send less and that
  condition is not an error.
  */

int writen(int sock_d,char *str,int n)
{
        ssize_t bytes_sent;
        size_t bytes_remaining=n;
        while(bytes_remaining!=0) {
                bytes_sent = write(sock_d,str,bytes_remaining);
                if(bytes_sent<=0)  {
                        if(errno== EINTR && bytes_sent <0)
                                bytes_sent=0;
                        else
                                return -1;
                }
                str=str+bytes_sent;
                bytes_remaining=bytes_remaining-bytes_sent;
        }
        printf("wrote %d bytes \n",n);
        return n;
}


/*
   my_read function takes the socket descriptor
   and pointer to a character where the read char
   will be stored. This function is used by our
   readline function
   */

int my_read(int sock_d,char *ptr)
{

       //Initially , the buffer_count value is zero.
       if(buffer_count<=0) {
up:            //label

               buffer_count=  read(sock_d,my_buffer,sizeof(my_buffer));
               /*buffer_count is updated in the previous statement.
                 It can be positive,negative or zero. Each value
corresponds
                 to a specific meaning checked below */


               //CASE A: BUFFER_COUNT is less than zero.
               //Caution: Indicates error condition

               if(buffer_count < 0) {
                       //EINTR requires us to call read again .
                       //So, we jump back to label 'up'
                       if(errno == EINTR) {
                               goto up;
                       }
                       else {
                               return -1;
                       }
               }


               //CASE B: Buffer_count==0
               //Indicates an end of file condition

               else if (buffer_count==0)
                       return 0;


               //CASE C:When Buffer_count>0,we set set the buffer pointer
               //equal to a pointer to the string from which we are
reading
               buffer_pointer= my_buffer;
       }

       buffer_count=buffer_count-1; //one element read from the buffer
       *ptr=*buffer_pointer;        //assign pointer to the character
pointer
       buffer_pointer=buffer_pointer+1; // Increment pointer
       return 1;
}


int readline(int sock_d,char *ptr,int max)
```

```
{
        //Readline will call our my_read function for better control
        int bytes_received;
        int i=1;
        char c,*ptr1;
        ptr1=ptr;
        while(i<max)
        {
                bytes_received =my_read(sock_d,&c); //read a character from
buffer into c

                if(bytes_received== 1)
                {
                        *ptr1=c;
                        ptr1=ptr1+1;
                        if(c == '\n') //null termination
                                break;
                }
                else if (bytes_received == 0) {
                        *ptr1=0; //this is the End of file situation
                        return (i-1);
                }
                else {
                        return -1;
                }


                 /*
                switch(bytes_received) {
                        //case 1:If bytes_received is 1, we got one char
from buffer
                        case 1:
                                *ptr1=c;
                                ptr1=ptr1+1;
                                if(c=='\n') // New line character read
                                        break;     //line read complete
                                break;
                        case 0:
                                //End of file situation
                                *ptr1=0;
                                return (i-1);
                        default:
                                return -1;
                }
                */

                i=i+1; //loop counter
        }
        *ptr1=0; // null termination
        return i; // return the number of bytes read
}
```

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








int main(int argc,char *argv[])
{
        int sockfd,numbytes,i;
        char buf[DATA_SIZE_LIMIT];
        struct addrinfo hints , *server_info, *p;
        char input_stdin[100];
        char username[16];
        int rv,len,size;
        int bytes_written;
        char s[INET6_ADDRSTRLEN];
        char buffer[1024];
        fd_set read_set,read_set1;
        struct timeval tv;
        char message[MESSAGE_LEN_LIMIT];
        struct SBCP_message sbcp_msg;
        struct SBCP_attribute sbcp_attr1;
        char data[530];
        char data1[20];
        struct SBCP_message sbcp_msg1;
        struct SBCP_attribute sbcp_attr11;
        struct SBCP_attribute sbcp_attr22;
        struct SBCP_attribute sbcp_attr2;
        uint16_t count;
        int cnt=0;
        //zero out the master set and the read set
        FD_ZERO(&read_set);


        if(argc!=4) {
                fprintf(stderr,"usage :client username hostname port-
number\n");
                exit(1);
        }






        memset(&hints ,0,sizeof hints);
        hints.ai_family =AF_UNSPEC;     // can be IpV4/IpV6
        hints.ai_socktype =SOCK_STREAM; // Stream Socket


        if((rv=getaddrinfo(argv[2],argv[3],&hints,&server_info)) !=0) {
                fprintf(stderr,"getaddrinfo: %s\n",gai_strerror(rv));
                return 1;
        }

        for(p=server_info; p!= NULL; p=p->ai_next) { // Find and use the
first working configuration
                //ai_next is the pointer to next node in the linked list
                if((sockfd=socket(p->ai_family,p->ai_socktype,p-
>ai_protocol))==-1) {
                        err_sys("client:socket");
                        continue;
                }

                if(connect(sockfd,p->ai_addr,p->ai_addrlen)==-1) {
                        close(sockfd);
                        err_sys("client : connect");
                        continue;
                }
                break;
        }


        if(p==NULL) {
                fprintf(stderr,"client:failed to connect\n");
                return 2;
        }

        inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p-
>ai_addr),s,sizeof s);
        printf("client:connecting to %s and my username is
%s\n",s,argv[1]);
        freeaddrinfo(server_info);//server_info is not longer required


//copy the username from the argv[1] into username
strcpy(username,argv[1]);
len=strlen(username);
//start by sending the join request
//data for JOIN :  type =2 ,version =3,payload=username entered by the
client on the console
//void init_sbcp_attribute( int type_attr ,int *ptr_to_buffer,int len) --
Method to initialize the sbcp_attribute

init_sbcp_attribute(&sbcp_attr1,2,username,20);

//void init_sbcp_message(int version,int type_msg,struct SBCP_attribute
*attr123)
init_sbcp_message(&sbcp_msg,'3','2',&sbcp_attr1,24);
```

```
size = pack(buf,"cchhhshhs", sbcp_msg.vrsn, sbcp_msg.type,
sbcp_msg.length, sbcp_attr1.type,sbcp_attr1.length,username,3,530," ");

        // send data of username
    if (send(sockfd, buf, size, 0) == -1){
                                            perror("send");
                                            exit(1);
}

//Next is the send operation
init_sbcp_attribute(&sbcp_attr1,4,message,520);
init_sbcp_message(&sbcp_msg,'3','4',&sbcp_attr1,524);
init_sbcp_attribute(&sbcp_attr2,2,username,16);

//Add the standard input and the socket to the read set

        //FD_SET(0, &read_set); // add the keyboard input to the read_fds
set
        //FD_SET(sockfd, &read_set);
        int ret_val;
        while(1) // Infinite loop
        {
                //read_set=master_set;
            FD_ZERO(&read_set);
             FD_SET(0,&read_set);
             FD_SET(sockfd,&read_set);
            tv.tv_sec=10;
               if((ret_val=select(sockfd+1,&read_set,NULL,NULL,&tv))==-1)
{
                        perror("select");
                        exit(4);
               }

               if(ret_val==0) {
                        //nothing is sec for 10 s elapsed time

        size=pack(buf,"cchhhshhs",'3','9',300,2,200,username,4,300,message)
;
                        if(send(sockfd,buf,size,0) ==-1) {
                                     perror("send");
                                                 }
               }



               if(FD_ISSET(0,&read_set)) {
                              if((fgets(message,512,stdin)==NULL)||
feof(stdin))  {
                              /* fgets() returns a NULL when we type the
END-OF-FILE(CNTRL-D)
                                 However, this does not take care of the
case when we input
```

```
                                    some text followed by the END-OF-FILE. For
that, we use the
                                    feof() function. feof will return true if
it encounters an
                                    end of file.*/


                                printf("\nEnd of file detected\n");
                                printf("\nclosing the socket on client
side\n");
                                printf("Client disconnected\n");
                                close(sockfd);
                                exit(1);
                                }

                                else {
                                strcpy(username,argv[1]);
                                size=pack(buf,"cchhhshhs", sbcp_msg.vrsn,
sbcp_msg.type,sbcp_msg.length,
sbcp_attr1.type,sbcp_attr1.length,username,sbcp_attr2.type,sbcp_attr2.leng
th,message);
                                if(send(sockfd,buf,size,0) ==-1) {
                                        perror("send");
                                                }

                                }
                        }


                if (FD_ISSET(sockfd,&read_set)){//there is data to read
from server
                                        // receive chat message to server

                                        if ((numbytes = recv(sockfd, buf,
DATA_SIZE_LIMIT-1, 0)) <= 0) {
                                                perror("recv");
                                                exit(1);
                                        }


        unpack(buf,"cchhhhshhs",&sbcp_msg1.vrsn,&sbcp_msg1.type,&sbcp_msg1.
length,&sbcp_attr11.type,&sbcp_attr11.length,&count,data,&sbcp_attr22.type
,&sbcp_attr22.length,data1);



                                        if(sbcp_msg1.type=='5') {
                                                printf("%s\n",data1);
                                        }

                                        if(cnt==2) cnt=0;
                                        if(sbcp_msg1.type=='7' && cnt!=1) {
                                        cnt++;
                                        printf("%s:",data1);
                                        printf("%s\n",data);
                                        if(count==0)
                                               count=1;
                                        printf("The count is :%d\n",count);
                                        //printf("%d\n",cnt);


                                        }

                                        if(sbcp_msg1.type=='8') {
                                          printf("ONLINE:%s",data1);
                                          printf("%s\n",data);
                                        }

                                        if(sbcp_msg1.type=='6') {
                                          printf("OFFLINE:%s",data1);
                                          printf("%s\n",data);
                                        }

                                        if(sbcp_msg1.type=='9') {
                                        printf("IDLE MESSAGE:");
                                        printf("%s has been idle for more
than 10 seconds\n",data);
                                        }



                                        else if(sbcp_msg1.type=='3') {
                                         printf("Received data from %s is :
%s \n",data,data1);
                                        }
                   }

               }
       return 0;
       }
```