NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
(KARACHI CAMPUS)
FAST School of Computing
Spring 2024

OS PROJECT REPORT

CHAT-ROOM USING SOCKETS AND MULTITHREADING

GROUP MEMBERS:

SYED SAMROZE ALI	[22k-4187]
SYED UZAIR HUSSAIN	[22k-4212]
ANAS AHMED	[22k-4371]

SECTION:

BCS-4J

COURSE INSTRUCTOR:

MS. MUBASHRA FAYYAZ

Tools and Technology, Languages and Utilities:

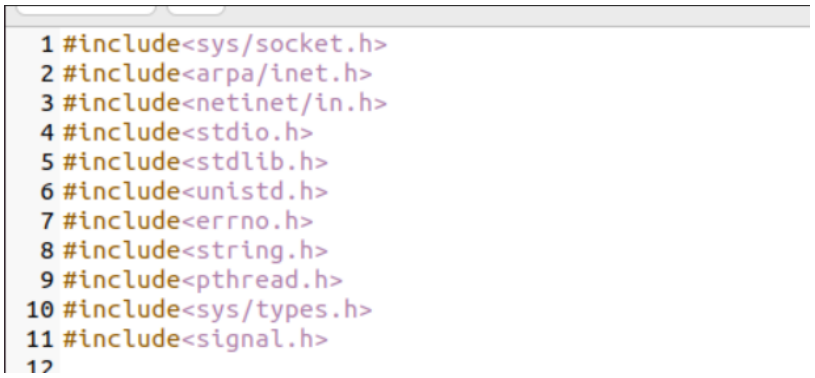
INTRODUCTION:

The aim of our project was to develop a chat-room application using socket programming and multithreading in the C programming language. The application consists of two main components: a server and multiple clients. The server facilitates communication between clients by relaying messages, while clients can send and receive messages within the chatroom.
The Technologies which we have used are:

- C Language
- Linux
- Socket programming
- Multithreading

LIBRARIES:

Following are the libraries we used in our C programs.

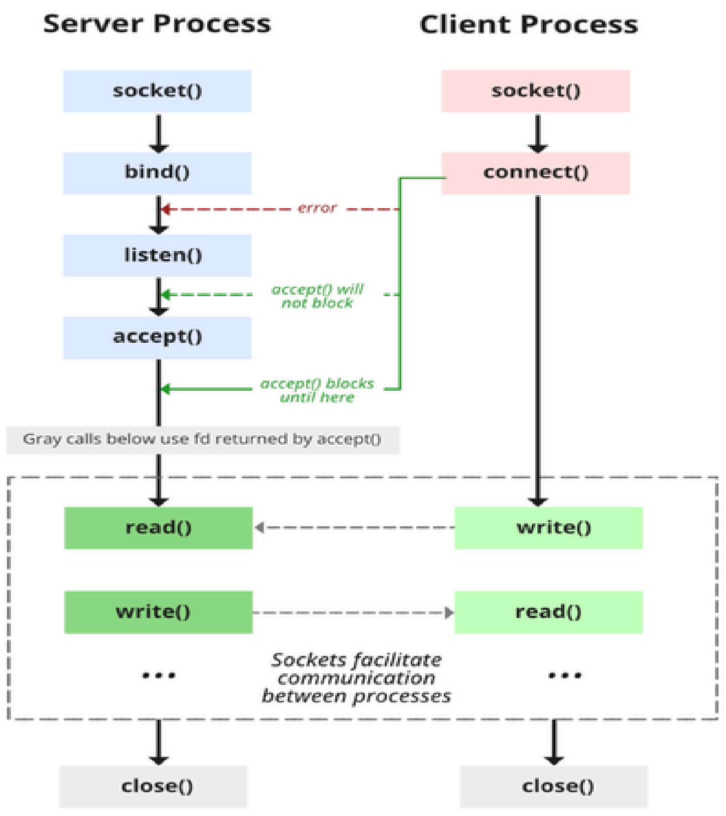


```c
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
#include<string.h>
#include<pthread.h>
#include<sys/types.h>
#include<signal.h>
```

stdio.h	This header file contains declarations for input and output in all c programs.
stdlib.h	Defines Variable types, several macros and functions. Ex: int atoi(const char*)
unistd.h	Provide access to Posix API.
string.h	All functionality regarding strings. Len() , concatenate() e.tc
Arpa/inet.h	Contains definitions for internet operations.(For windows winsock)

sys/socket.h	Definitions for structures needed for sockets. Ex: sockaddr
netinet.h	Constants and structures needed for internet domain address ex sockaddr_in
pthread.h	Give us access to make multiple threads so we can give access to multiple users/clients
sys/types.h	Includes various fundamental data types used by the operating system, such as pthread_t etc. which represents the size of an object in bytes.
signal.h	includes definitions for signals, which are a way for the operating system to notify a process that an event has occurred.
errno.h	defines integer variables that represent system error codes.

State Diagram of Server and Client Model:



FUNCTIONALITY OVERVIEW:

Server

- Accepts incoming client connections.
- Handles multiple client connections concurrently using multithreading.
- Relays messages between clients.
- Monitors client disconnections and updates the chatroom accordingly.

Client

- Connects to the server.
- Sends messages to the server to be broadcasted to other clients.
- Receives messages from other clients via the server.
- Allows users to gracefully exit the chatroom.

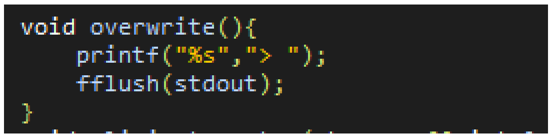
CHALLENGES:

- Ensuring thread safety and avoiding race conditions when accessing shared data structures.
- Handling client connections/disconnections gracefully and updating the chatroom state accordingly.
- Managing buffer sizes to prevent overflow
- Ensuring smooth message transmission.

CLIENT SIDE CODE:

EXPLANATION OF EACH FUNCTION:

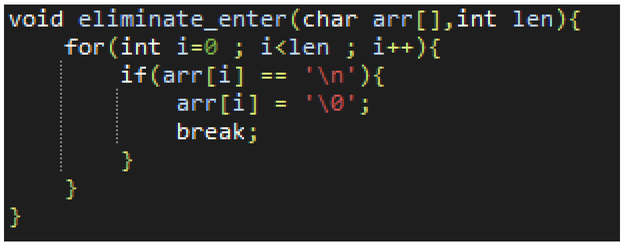
1. **void overwrite()**: This function is responsible for printing the prompt **"> "** to the console, indicating that the user can type a message. It's called whenever the program is ready to receive input.

   ```
   void overwrite(){
       printf("%s","> ");
       fflush(stdout);
   }
   ```

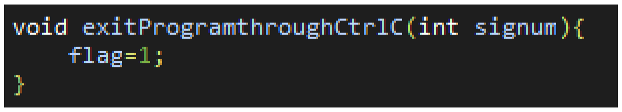
2. **void eliminate_enter(char arr[], int len)**: This function is used to remove the newline character ('\n') from the input string **arr**. It iterates over each character in

the array and replaces the newline character with a null terminator ('\0'). This function is particularly used after reading input with **fgets()** to remove the newline character.



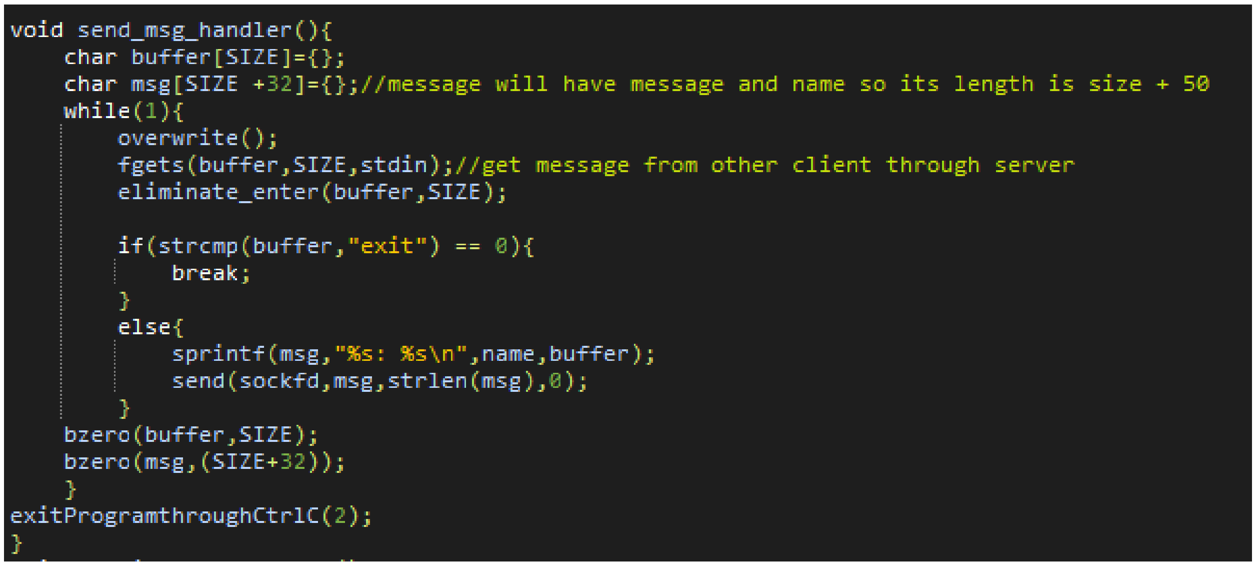
```c
void eliminate_enter(char arr[],int len){
    for(int i=0 ; i<len ; i++){
        if(arr[i] == '\n'){
            arr[i] = '\0';
            break;
        }
    }
}
```

3. **void exitProgramthroughCtrlC(int signum)**: This function is called when the program receives a SIGINT signal (triggered by pressing Ctrl+C). It sets the **flag** variable to 1, indicating that the program should exit now.



```c
void exitProgramthroughCtrlC(int signum){
    flag=1;
}
```

4. **void send_msg_handler()**: This function handles sending messages to the server. It prompts the user for input, reads a message from the console using **fgets()**, eliminates the newline character from the input using **eliminate_enter()**, and sends the message to the server using the **send()** function.



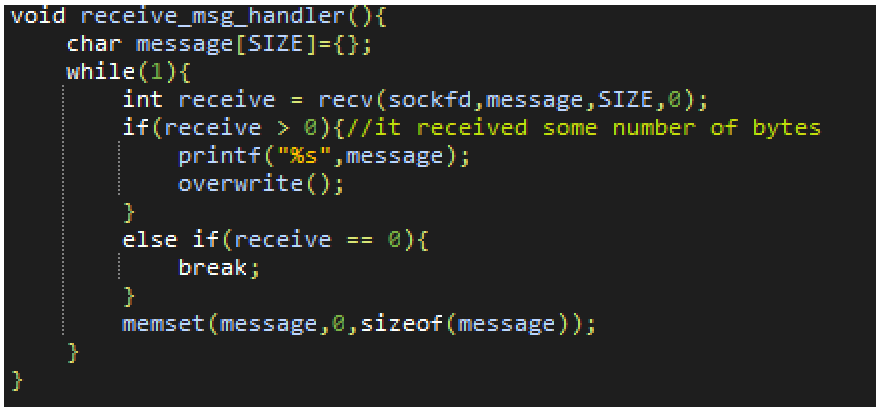
```c
void send_msg_handler(){
    char buffer[SIZE]={};
    char msg[SIZE +32]={};//message will have message and name so its length is size + 50
    while(1){
        overwrite();
        fgets(buffer,SIZE,stdin);//get message from other client through server
        eliminate_enter(buffer,SIZE);

        if(strcmp(buffer,"exit") == 0){
            break;
        }
        else{
            sprintf(msg,"%s: %s\n",name,buffer);
            send(sockfd,msg,strlen(msg),0);
        }
    bzero(buffer,SIZE);
    bzero(msg,(SIZE+32));
    }
exitProgramthroughCtrlC(2);
}
```

Explanation of the statements within send_msg_handler():

- sprintf(msg, "%s: %s\n", name, buffer): This statement constructs the message string msg by formatting it with the user's name (name) followed by a colon and a space (": ") and then the message entered by the user (buffer). The resulting message is terminated with a newline character ("\n"). This formatted message is stored in the msg variable.

- send(sockfd, msg, strlen(msg), 0): This statement sends the constructed message msg to the server using the socket descriptor “sockfd”. strlen(msg) is used to determine the length of the message to be sent. The last argument (0) is generally set to 0 for normal operation.
- bzero(buffer, SIZE): This statement clears the buffer array to prepare it for storing the next message. It sets all bytes in the buffer array to zero.
- bzero(msg, (SIZE + 32)): This statement clears the msg array to prepare it for constructing the next message. It sets all bytes in the msg array to zero.

5. **void receive_msg_handler()**: This function handles receiving messages from the server. It continuously receives messages from the server using the **recv()** function, prints the received message to the client’s terminal, and then prompts the users for input again.

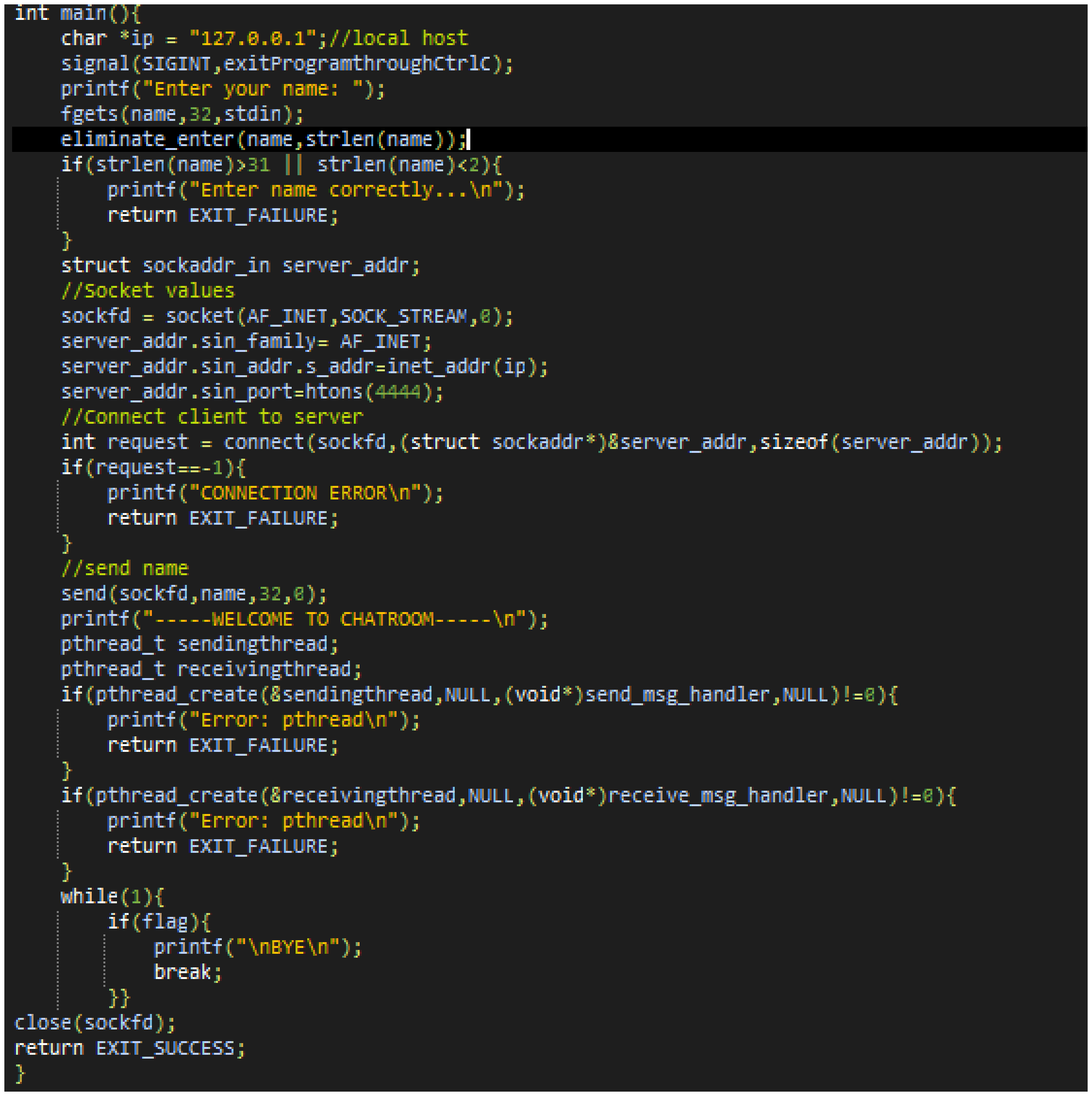


```
void receive_msg_handler(){
    char message[SIZE]={};
    while(1){
        int receive = recv(sockfd,message,SIZE,0);
        if(receive > 0){//it received some number of bytes
            printf("%s",message);
            overwrite();
        }
        else if(receive == 0){
            break;
        }
        memset(message,0,sizeof(message));
    }
}
```

Explanation of the statements within send_msg_handler():

- int receive = recv(sockfd, message, SIZE, 0): This statement receives data from the server using the recv() function. It reads data from the socket descriptor “sockfd” and stores it in the message buffer. SIZE gives the maximum number of bytes received.
- memset(message, 0, sizeof(message)): This is a function used to fill a block of memory with a particular value. It sets every byte in the message array to 0, effectively clearing the contents of the array. This is commonly used to reset or initialize arrays and buffers before using them to ensure that there are no residual values left from previous usage.

6. **int main()**: This is the main function where the program execution begins. It initializes variables, establishes a connection to the server, prompts the user for their name, creates two threads for sending and receiving messages (**sendingthread** and **receivingthread**), and then enters an infinite loop to wait for the exit signal (**flag**) set by **exitProgramthroughCtrlC()**. Once the exit signal is received, the program closes the socket and exits.



```
int main(){
    char *ip = "127.0.0.1";//local host
    signal(SIGINT,exitProgramthroughCtrlC);
    printf("Enter your name: ");
    fgets(name,32,stdin);
    eliminate_enter(name,strlen(name));
    if(strlen(name)>31 || strlen(name)<2){
        printf("Enter name correctly...\n");
        return EXIT_FAILURE;
    }
    struct sockaddr_in server_addr;
    //Socket values
    sockfd = socket(AF_INET,SOCK_STREAM,0);
    server_addr.sin_family= AF_INET;
    server_addr.sin_addr.s_addr=inet_addr(ip);
    server_addr.sin_port=htons(4444);
    //Connect client to server
    int request = connect(sockfd,(struct sockaddr*)&server_addr,sizeof(server_addr));
    if(request==-1){
        printf("CONNECTION ERROR\n");
        return EXIT_FAILURE;
    }
    //send name
    send(sockfd,name,32,0);
    printf("-----WELCOME TO CHATROOM-----\n");
    pthread_t sendingthread;
    pthread_t receivingthread;
    if(pthread_create(&sendingthread,NULL,(void*)send_msg_handler,NULL)!=0){
        printf("Error: pthread\n");
        return EXIT_FAILURE;
    }
    if(pthread_create(&receivingthread,NULL,(void*)receive_msg_handler,NULL)!=0){
        printf("Error: pthread\n");
        return EXIT_FAILURE;
    }
    while(1){
        if(flag){
            printf("\nBYE\n");
            break;
        }}
close(sockfd);
return EXIT_SUCCESS;
}
```

Explanation of the statements within main():

- char *ip = "127.0.0.1";: This line declares a pointer ip and assigns it the string value "127.0.0.1", which represents the IP address of the server. This address (127.0.0.1) is the loopback/localhost address(used it to test server through “telnet” command).
- sockfd = socket(AF_INET, SOCK_STREAM, 0);: This line creates a socket using the socket() function. It takes three arguments:
    - AF_INET: Address family, which specifies the communication domain(over here its IPv4 address family).
    - SOCK_STREAM: Type of socket (in this case, a stream socket, which provides sequenced, reliable, two-way, connection-based byte streams i.e a TCP(transmission control protocol).

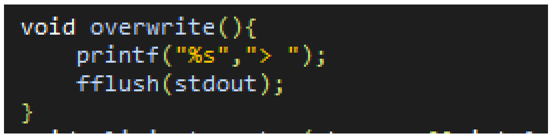
        - 0: Protocol (in this case, the protocol is chosen automatically based on the specified socket type and address family).
- server_addr.sin_family = AF_INET;: This line sets the address family of the server_addr structure to AF_INET.
- server_addr.sin_addr.s_addr = inet_addr(ip);: This line converts the IP address string (ip) to a binary format (in_addr_t) and assigns it to the sin_addr.s_addr field of the server_addr structure. The inet_addr() function converts the string representation of an IPv4 address into a network address structure.
- server_addr.sin_port = htons(4444);: This line sets the port number (4444) in the server_addr structure. The port number is specified in network byte order using the htons() function, which converts a 16-bit quantity from host byte order to network byte order (big-endian).
- int request = connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));: This line establishes a connection to the server using the connect() function. It takes three arguments:
    - sockfd: Socket descriptor representing the client socket.
    - (struct sockaddr*)&server_addr: Pointer to a struct sockaddr containing the server address information.
    - sizeof(server_addr): Size of the server_addr structure.

  The connect() function returns 0 on success and -1 on failure. The return value is stored in the variable request.
- close(sockfd);: This line closes the socket descriptor sockfd after the connection is established or attempted. It releases the resources associated with the socket.

SERVER CODE:

EXPLANATION OF EACH FUNCTION:

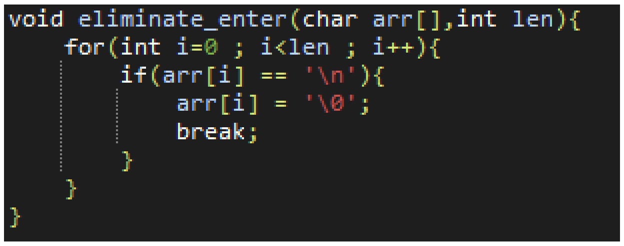
1. **void overwrite()**: This function is responsible for printing the prompt **"> "** to the console, indicating that the user can type a message. It's called whenever the program is ready to receive input.



```
void overwrite(){
    printf("%s","> ");
    fflush(stdout);
}
```

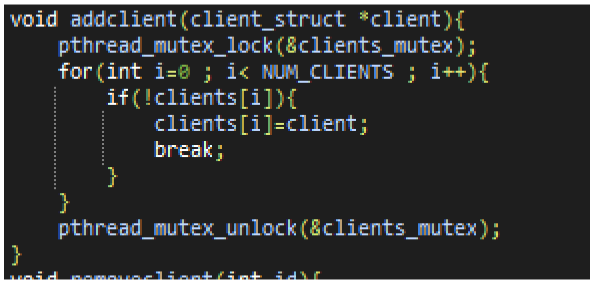
2. **void eliminate_enter(char arr[], int len)**: This function is used to remove the newline character ('\n') from the input string **arr**. It iterates over each character in the array and replaces the newline character with a null terminator ('\0'). This

function is particularly used after reading input with **fgets()** to remove the newline character.



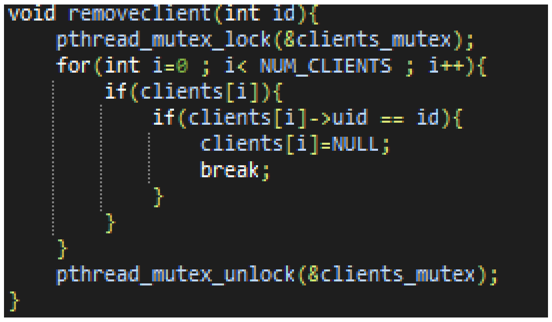
```c
void eliminate_enter(char arr[],int len){
    for(int i=0 ; i<len ; i++){
        if(arr[i] == '\n'){
            arr[i] = '\0';
            break;
        }
    }
}
```

3. **addclient(client_struct client):** This function adds a client to the clients array. It locks the mutex to prevent concurrent access to the clients array, then iterates over the array to find an empty slot to insert the new client.



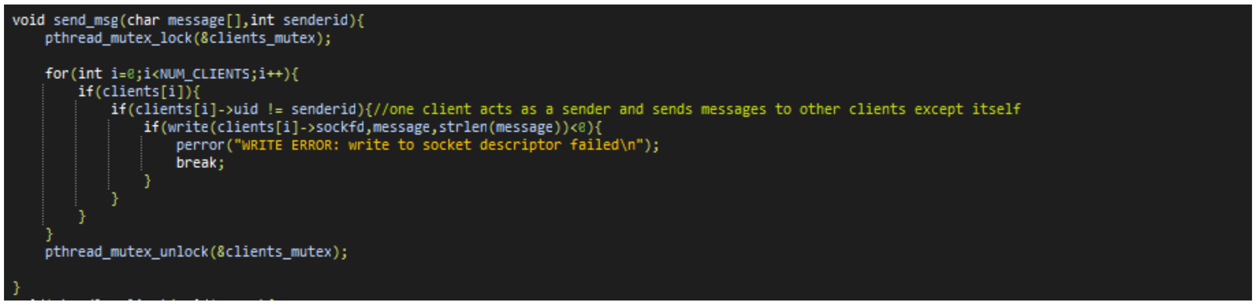
```c
void addclient(client_struct *client){
    pthread_mutex_lock(&clients_mutex);
    for(int i=0 ; i< NUM_CLIENTS ; i++){
        if(!clients[i]){
            clients[i]=client;
            break;
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}
```

4. **removeclient(int id):** This function removes a client from the clients array based on the given id. It locks the mutex to prevent concurrent access to the clients array, then iterates over the array to find the client with the matching id and removes it.



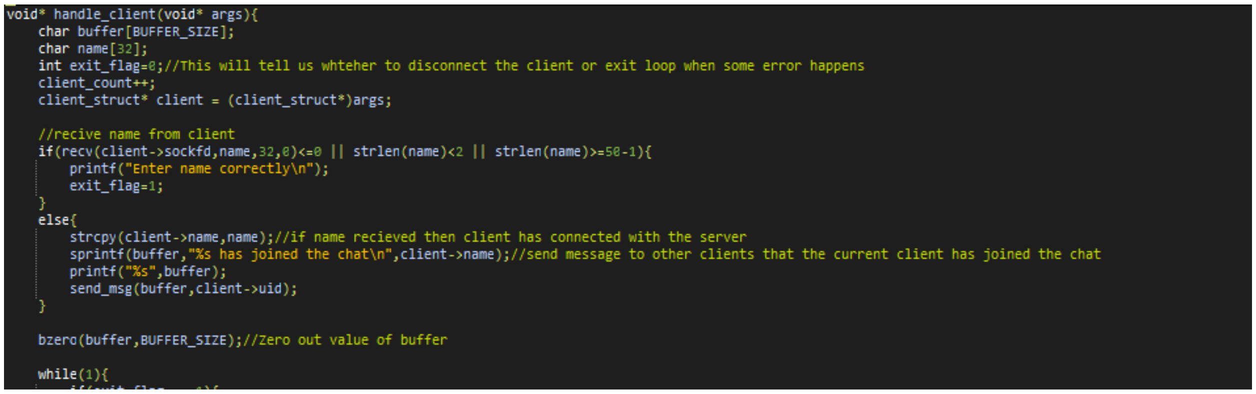
```c
void removeclient(int id){
    pthread_mutex_lock(&clients_mutex);
    for(int i=0 ; i< NUM_CLIENTS ; i++){
        if(clients[i]){
            if(clients[i]->uid == id){
                clients[i]=NULL;
                break;
            }
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}
```

5. **send_msg(char message[],int senderid):** This function sends a message to all clients except the one identified by senderid. It locks the mutex to prevent concurrent access to the clients array, then iterates over the array to send the message to each client's socket descriptor using the write() function.

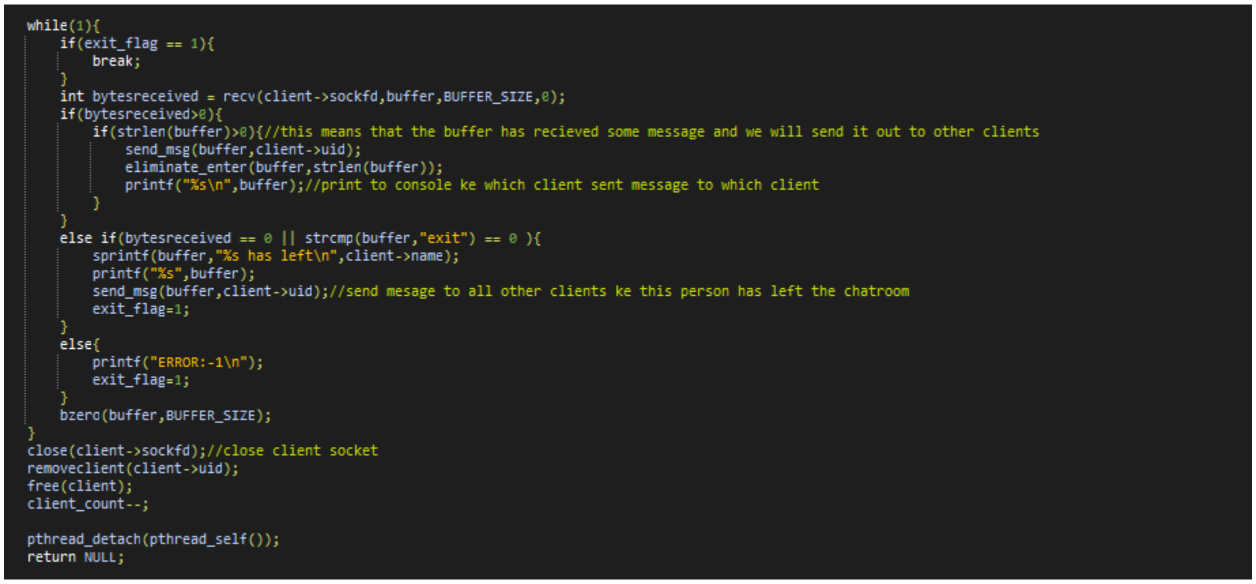


```c
void send_msg(char message[],int senderid){
    pthread_mutex_lock(&clients_mutex);

    for(int i=0;i<NUM_CLIENTS;i++){
        if(clients[i]){
            if(clients[i]->uid != senderid){//one client acts as a sender and sends messages to other clients except itself
                if(write(clients[i]->sockfd,message,strlen(message))<0){
                    perror("WRITE ERROR: write to socket descriptor failed\n");
                    break;
                }
            }
        }
    }
    pthread_mutex_unlock(&clients_mutex);

}
```

6. **handle_client(void args):** This function is executed in a separate thread for each connected client. It handles communication with the client. It receives the client's name, sends a join message to other clients, continuously receives messages from the client, and sends them to other clients. It also handles disconnection of the client.



```c
void* handle_client(void* args){
    char buffer[BUFFER_SIZE];
    char name[32];
    int exit_flag=0;//This will tell us whteher to disconnect the client or exit loop when some error happens
    client_count++;
    client_struct* client = (client_struct*)args;

    //recive name from client
    if(recv(client->sockfd,name,32,0)<=0 || strlen(name)<2 || strlen(name)>=50-1){
        printf("Enter name correctly\n");
        exit_flag=1;
    }
    else{
        strcpy(client->name,name);//if name recieved then client has connected with the server
        sprintf(buffer,"%s has joined the chat\n",client->name);//send message to other clients that the current client has joined the chat
        printf("%s",buffer);
        send_msg(buffer,client->uid);
    }

    bzero(buffer,BUFFER_SIZE);//Zero out value of buffer

    while(1){
```



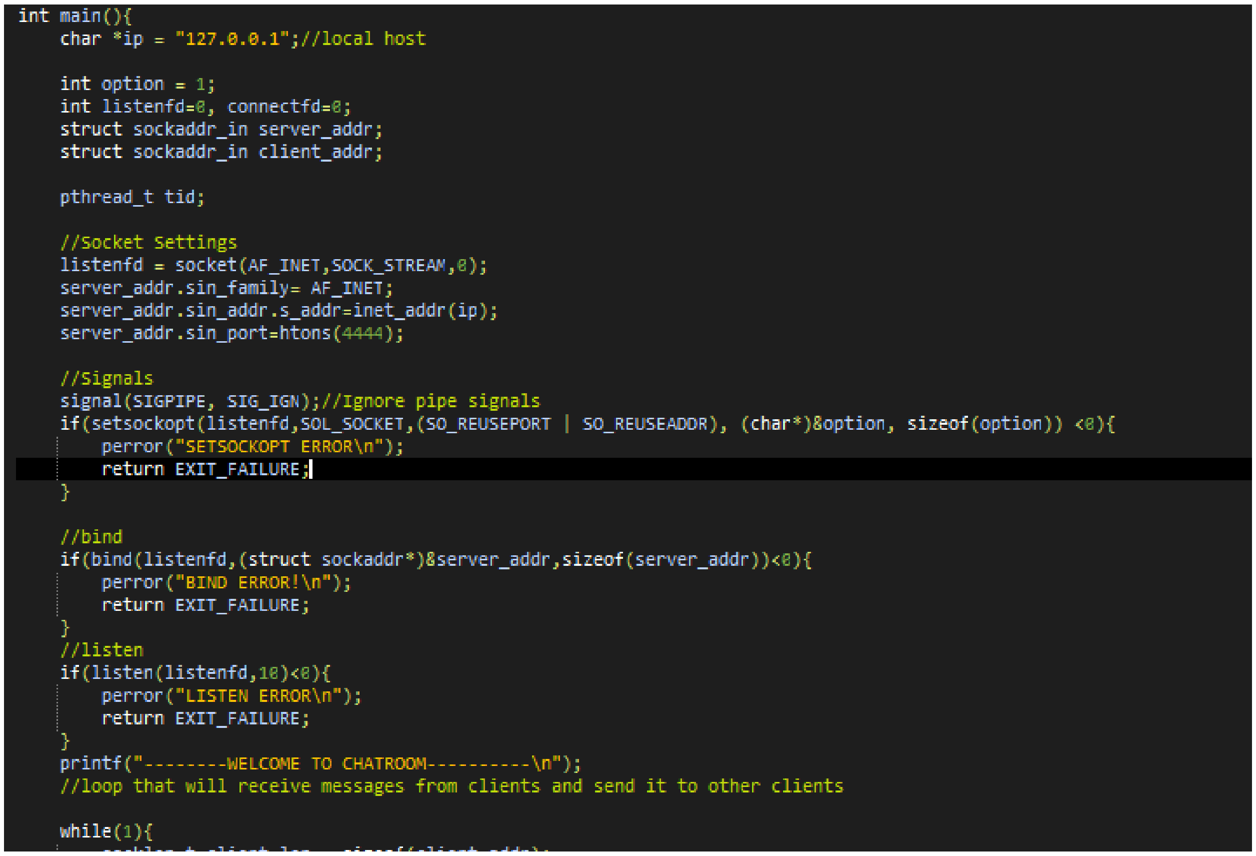
```c
while(1){
    if(exit_flag == 1){
        break;
    }
    int bytesreceived = recv(client->sockfd,buffer,BUFFER_SIZE,0);
    if(bytesreceived>0){
        if(strlen(buffer)>0){//this means that the buffer has recieved some message and we will send it out to other clients
            send_msg(buffer,client->uid);
            eliminate_enter(buffer,strlen(buffer));
            printf("%s\n",buffer);//print to console ke which client sent message to which client
        }
    }
    else if(bytesreceived == 0 || strcmp(buffer,"exit") == 0 ){
        sprintf(buffer,"%s has left\n",client->name);
        printf("%s",buffer);
        send_msg(buffer,client->uid);//send mesage to all other clients ke this person has left the chatroom
        exit_flag=1;
    }
    else{
        printf("ERROR:-1\n");
        exit_flag=1;
    }
    bzero(buffer,BUFFER_SIZE);
}
close(client->sockfd);//close client socket
removeclient(client->uid);
free(client);
client_count--;

pthread_detach(pthread_self());
return NULL;
```

Explanation of the statements within handle_client():

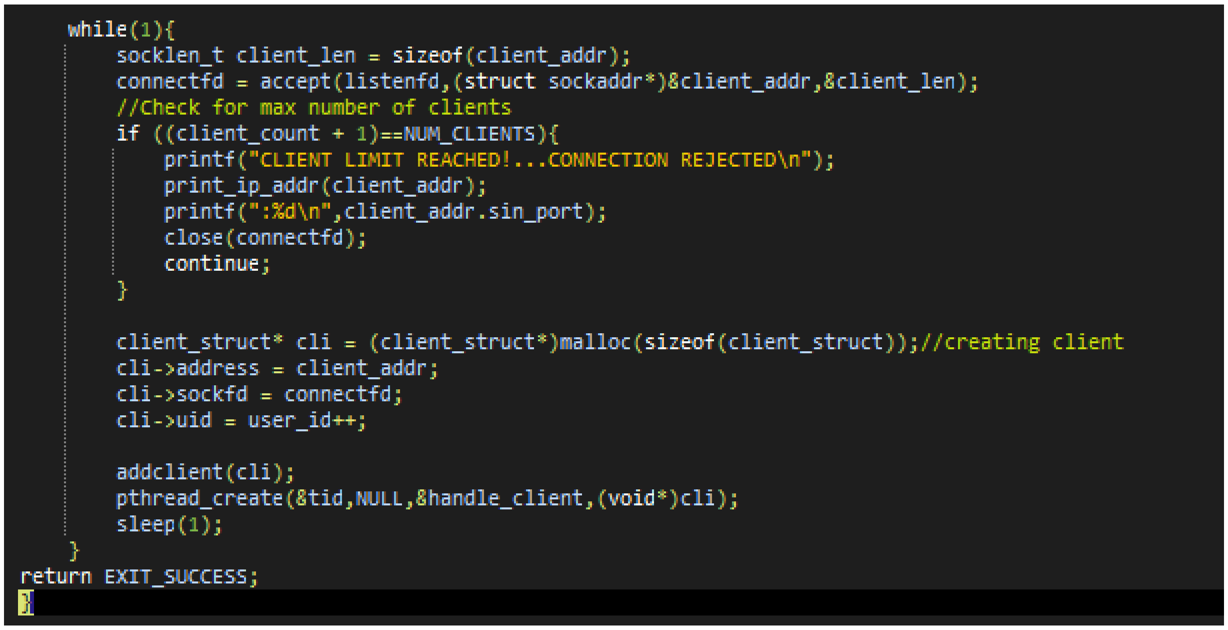
- <u>if(recv(client->sockfd,name,32,0)<=0 || strlen(name)<2 || strlen(name)>=50-1):</u> This line receives the client's name from its socket descriptor (client->sockfd). It checks if the received data is less than or equal to 0 (indicating an error or disconnection), or if the length of the name is less than 2 or greater than or equal to 49. If any of these conditions are met, it signifies an invalid name input.

- sprintf(buffer,"%s has joined the chat\n",client->name); This line constructs a message indicating that the current client has joined the chat. It uses sprintf() to format the message with the client's name.
- send_msg(buffer,client->uid); This line sends the join message to all other clients except the current client. It uses the send_msg() function to accomplish this.
- int bytesreceived = recv(client->sockfd,buffer,BUFFER_SIZE,0); This line receives messages from the client and stores them in the buffer. The number of bytes received is stored in bytesreceived.

7. **main():** This is the entry point of the program. It sets up the server socket, listens for incoming connections, accepts connections, creates a new thread to handle each connection, and repeats the process indefinitely.



```c
int main(){
    char *ip = "127.0.0.1";//local host

    int option = 1;
    int listenfd=0, connectfd=0;
    struct sockaddr_in server_addr;
    struct sockaddr_in client_addr;

    pthread_t tid;

    //Socket Settings
    listenfd = socket(AF_INET,SOCK_STREAM,0);
    server_addr.sin_family= AF_INET;
    server_addr.sin_addr.s_addr=inet_addr(ip);
    server_addr.sin_port=htons(4444);

    //Signals
    signal(SIGPIPE, SIG_IGN);//Ignore pipe signals
    if(setsockopt(listenfd,SOL_SOCKET,(SO_REUSEPORT | SO_REUSEADDR), (char*)&option, sizeof(option)) <0){
        perror("SETSOCKOPT ERROR\n");
        return EXIT_FAILURE;
    }

    //bind
    if(bind(listenfd,(struct sockaddr*)&server_addr,sizeof(server_addr))<0){
        perror("BIND ERROR!\n");
        return EXIT_FAILURE;
    }
    //listen
    if(listen(listenfd,10)<0){
        perror("LISTEN ERROR\n");
        return EXIT_FAILURE;
    }
    printf("--------WELCOME TO CHATROOM----------\n");
    //loop that will receive messages from clients and send it to other clients

    while(1){
```



```c
    while(1){
        socklen_t client_len = sizeof(client_addr);
        connectfd = accept(listenfd,(struct sockaddr*)&client_addr,&client_len);
        //Check for max number of clients
        if ((client_count + 1)==NUM_CLIENTS){
            printf("CLIENT LIMIT REACHED!...CONNECTION REJECTED\n");
            print_ip_addr(client_addr);
            printf(":%d\n",client_addr.sin_port);
            close(connectfd);
            continue;
        }

        client_struct* cli = (client_struct*)malloc(sizeof(client_struct));//creating client
        cli->address = client_addr;
        cli->sockfd = connectfd;
        cli->uid = user_id++;

        addclient(cli);
        pthread_create(&tid,NULL,&handle_client,(void*)cli);
        sleep(1);
    }
return EXIT_SUCCESS;
}
```

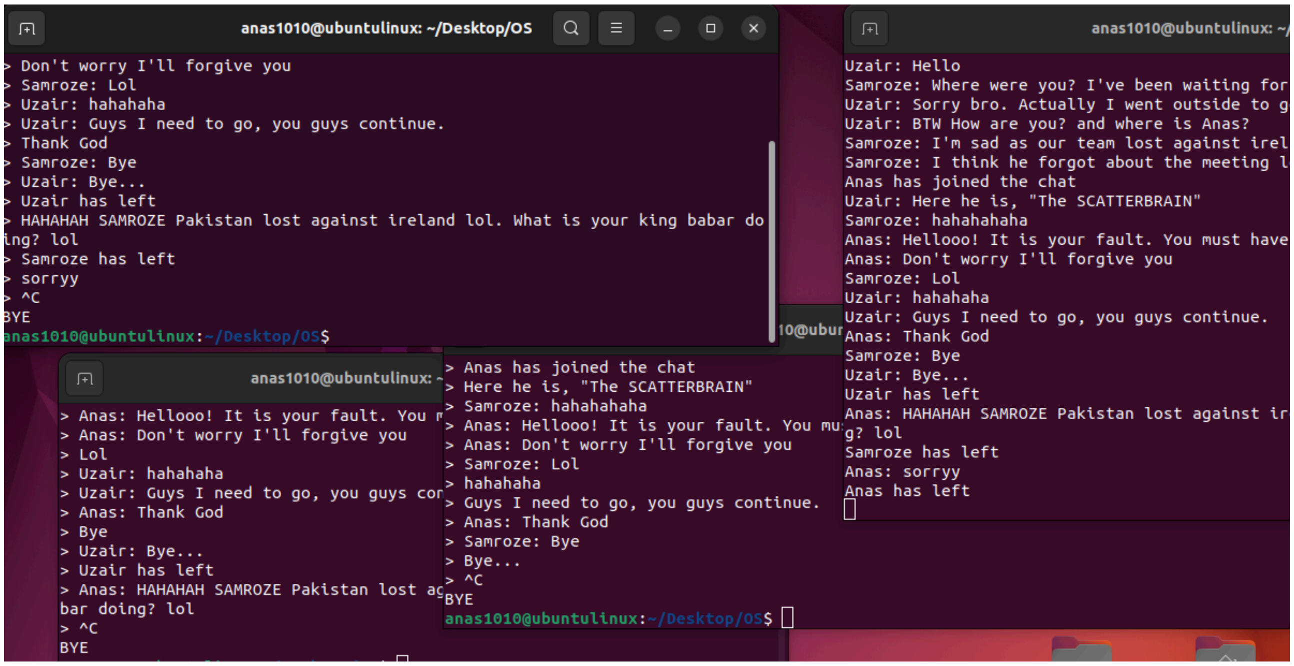
Explanation of the statements within main():

- char *ip = "127.0.0.1";:This statement initializes a pointer ip with the IP address "127.0.0.1". This IP address represents the loopback address.
- listenfd = socket(AF_INET, SOCK_STREAM, 0);:This statement creates a socket using the socket() system call.
    - AF_INET: Specifies the address family.
    - SOCK_STREAM: Specifies the socket type as a stream socket, which provides sequenced, reliable, two-way, connection-based byte streams.
    - 0: Specifies the protocol. Since SOCK_STREAM is used, 0 indicates that the operating system should choose the appropriate protocol.
- Setting up server address (server_addr):The following lines initialize the server_addr structure with the server's IP address and port number.
    - server_addr.sin_family = AF_INET;: Sets the address family to IPv4.
    - server_addr.sin_addr.s_addr = inet_addr(ip);: Sets the IP address. Here, inet_addr() converts the IP address string "127.0.0.1" to its binary representation.
    - server_addr.sin_port = htons(4444);: Sets the port number. htons() converts the port number from host byte order to network byte order.
- signal(SIGPIPE, SIG_IGN);:This statement sets the signal handler for SIGPIPE to SIG_IGN, which means the program will ignore SIGPIPE signals, which can terminate the program.
- if(setsockopt(listenfd, SOL_SOCKET, (SO_REUSEPORT | SO_REUSEADDR), (char*)&option, sizeof(option)) < 0):This statement sets socket options to allow the reuse of the port and address. It prevents the "address already in use" error when restarting the server immediately after it's stopped.
  Parameters:
    - listenfd: The socket descriptor.
    - SOL_SOCKET: The level at which the option is defined (in this case, at the socket level).
    - (SO_REUSEPORT | SO_REUSEADDR): The socket options to set, indicating that both port and address can be reused.
    - (char*)&option: A pointer to the option value, which is cast to a char*.
    - sizeof(option): The size of the option value.

- if(bind(listenfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0):This statement binds the socket to the server address.
  Parameters:
    - listenfd: The socket descriptor of the server socket that is in the listening state.

    - (struct sockaddr*)&server_addr: A pointer to the server address
    - structure.sizeof(server_addr): The size of the server address structure.
- connectfd = accept(listenfd, (struct sockaddr*)&client_addr, &client_len);:
  This line calls the accept() system call to accept a connection on the listening socket listenfd.
  Parameters:
    - connectfd: The socket descriptor of the server socket that is in the listening state.
    - (struct sockaddr*)&client_addr: A pointer to a sockaddr_in structure (client_addr) where the client's address information will be stored.
    - &client_len: A pointer to a variable (client_len) that stores the size of the client address structure. Upon the return of accept(), this variable will be updated with the actual size of the client's address structure.

OUTPUT:

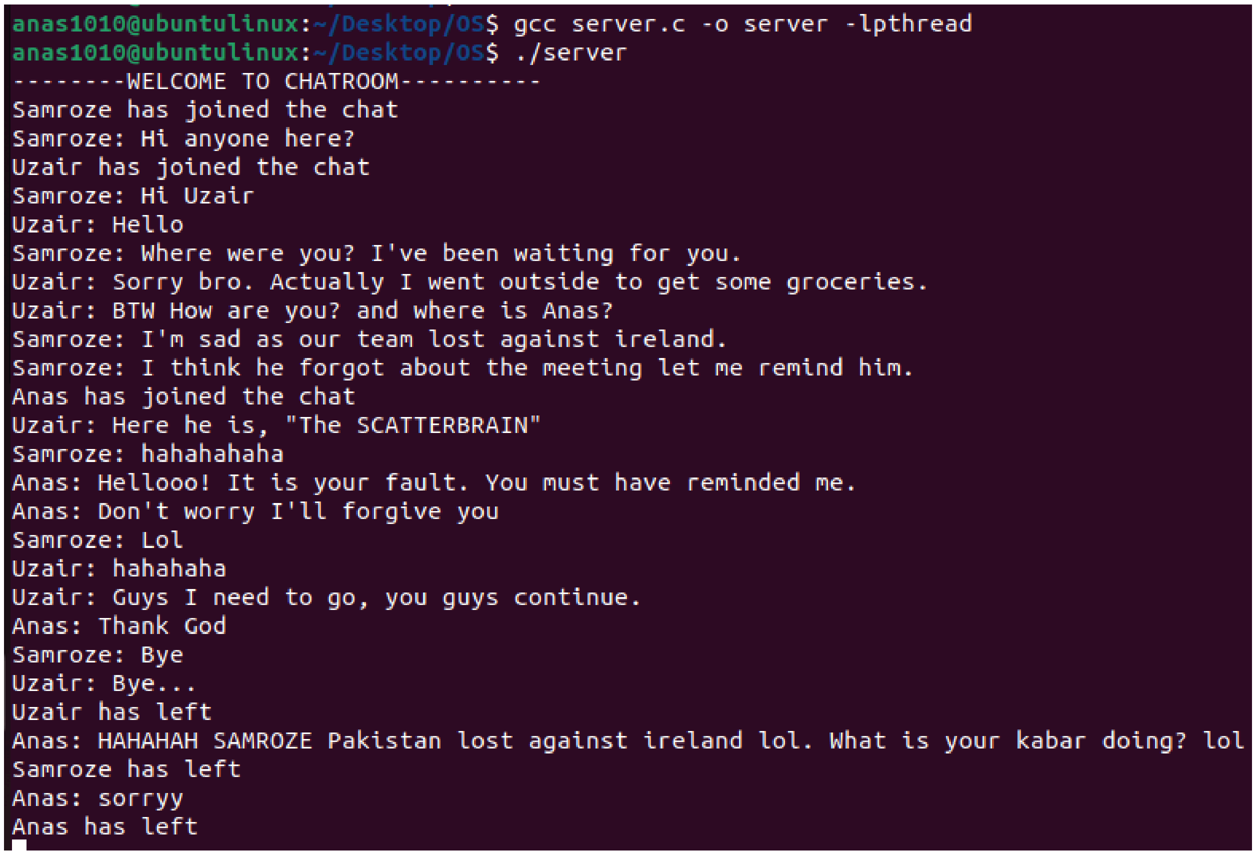


```
> Don't worry I'll forgive you
> Samroze: Lol
> Uzair: hahahaha
> Uzair: Guys I need to go, you guys continue.
> Thank God
> Samroze: Bye
> Uzair: Bye...
> Uzair has left
> HAHAHAH SAMROZE Pakistan lost against ireland lol. What is your king babar do
ing? lol
> Samroze has left
> sorryy
> ^C
BYE
anas1010@ubuntulinux:~/Desktop/OS$
```

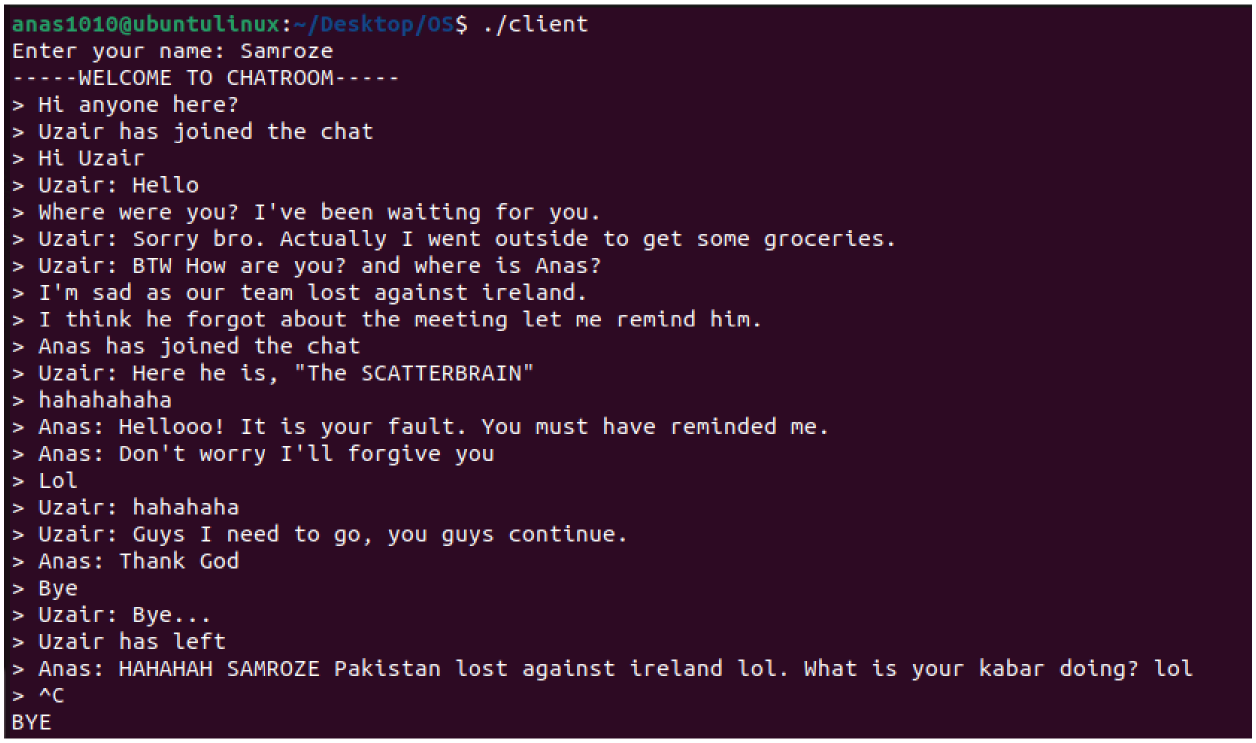
```
Uzair: Hello
Samroze: Where were you? I've been waiting for
Uzair: Sorry bro. Actually I went outside to g
Uzair: BTW How are you? and where is Anas?
Samroze: I'm sad as our team lost against irel
Samroze: I think he forgot about the meeting l
Anas has joined the chat
Uzair: Here he is, "The SCATTERBRAIN"
Samroze: hahahahaha
Anas: Hellooo! It is your fault. You must have
Anas: Don't worry I'll forgive you
Samroze: Lol
Uzair: hahahaha
Uzair: Guys I need to go, you guys continue.
Anas: Thank God
Samroze: Bye
Uzair: Bye...
Uzair has left
Anas: HAHAHAH SAMROZE Pakistan lost against ir
g? lol
Samroze has left
Anas: sorryy
Anas has left
```

```
> Anas: Hellooo! It is your fault. You r
> Anas: Don't worry I'll forgive you
> Lol
> Uzair: hahahaha
> Uzair: Guys I need to go, you guys cor
> Anas: Thank God
> Bye
> Uzair: Bye...
> Uzair has left
> Anas: HAHAHAH SAMROZE Pakistan lost ag
bar doing? lol
> ^C
BYE
```

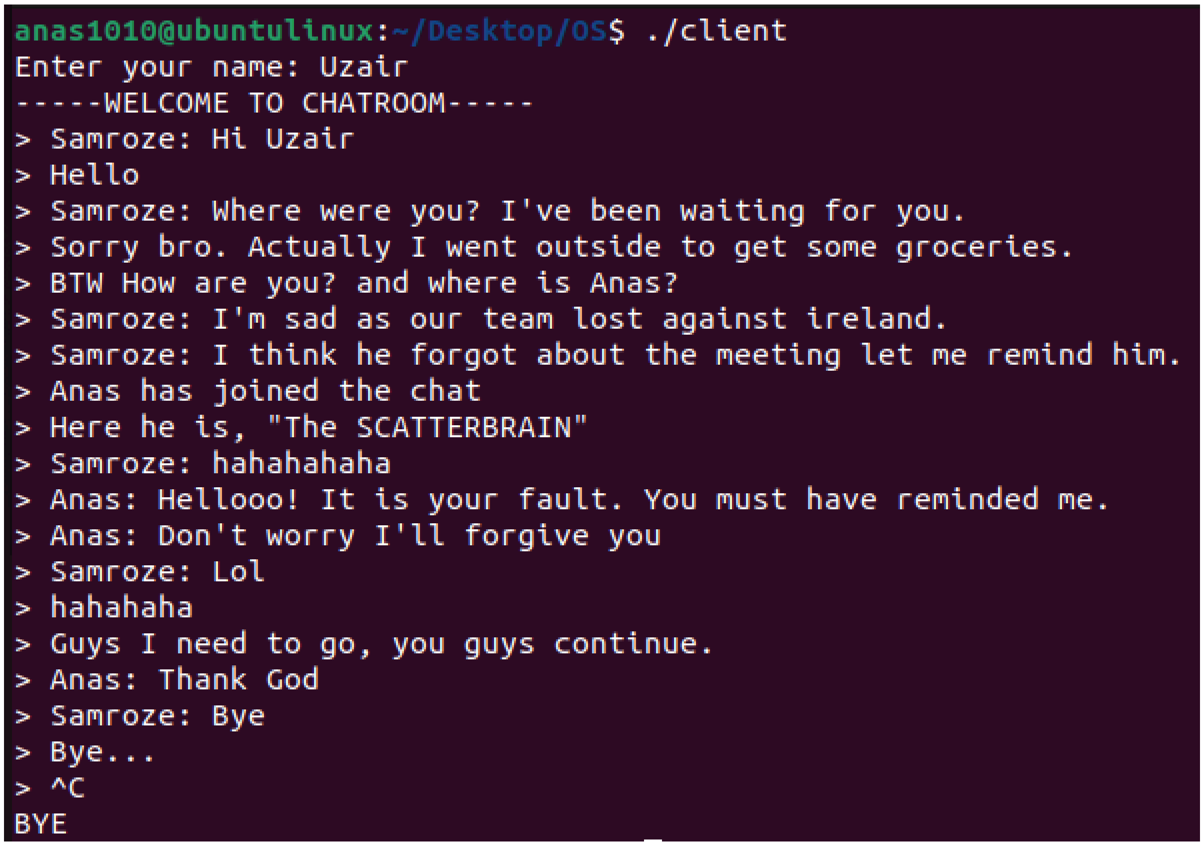
```
> Anas has joined the chat
> Here he is, "The SCATTERBRAIN"
> Samroze: hahahahaha
> Anas: Hellooo! It is your fault. You mu
> Anas: Don't worry I'll forgive you
> Samroze: Lol
> hahahaha
> Guys I need to go, you guys continue.
> Anas: Thank God
> Samroze: Bye
> Bye...
> ^C
BYE
anas1010@ubuntulinux:~/Desktop/OS$
```



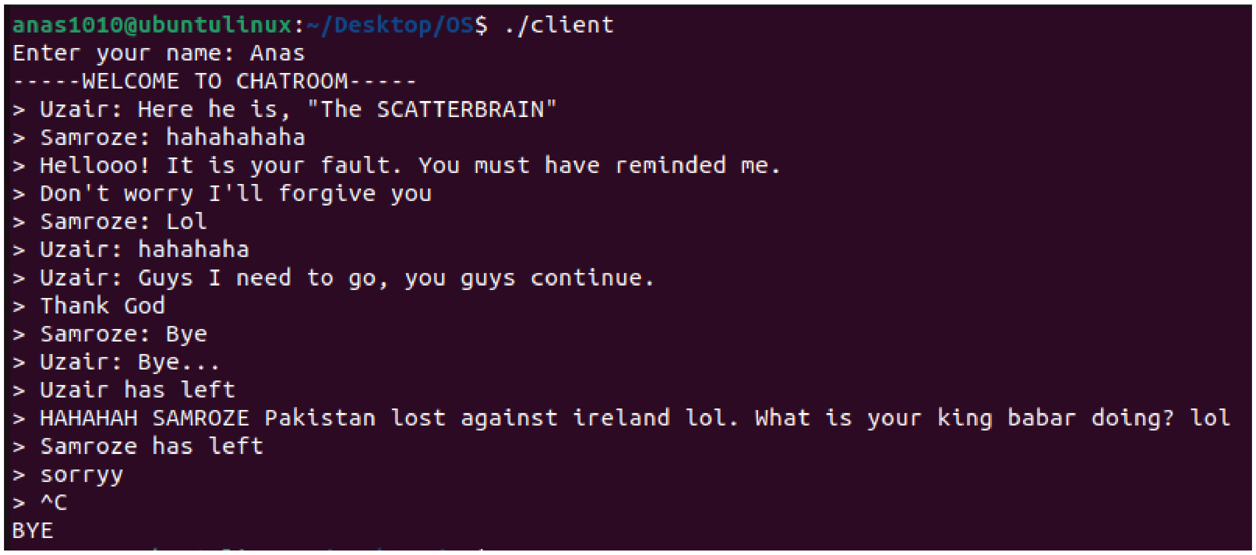
```
anas1010@ubuntulinux:~/Desktop/OS$ gcc server.c -o server -lpthread
anas1010@ubuntulinux:~/Desktop/OS$ ./server
--------WELCOME TO CHATROOM----------
Samroze has joined the chat
Samroze: Hi anyone here?
Uzair has joined the chat
Samroze: Hi Uzair
Uzair: Hello
Samroze: Where were you? I've been waiting for you.
Uzair: Sorry bro. Actually I went outside to get some groceries.
Uzair: BTW How are you? and where is Anas?
Samroze: I'm sad as our team lost against ireland.
Samroze: I think he forgot about the meeting let me remind him.
Anas has joined the chat
Uzair: Here he is, "The SCATTERBRAIN"
Samroze: hahahahaha
Anas: Hellooo! It is your fault. You must have reminded me.
Anas: Don't worry I'll forgive you
Samroze: Lol
Uzair: hahahaha
Uzair: Guys I need to go, you guys continue.
Anas: Thank God
Samroze: Bye
Uzair: Bye...
Uzair has left
Anas: HAHAHAH SAMROZE Pakistan lost against ireland lol. What is your kabar doing? lol
Samroze has left
Anas: sorryy
Anas has left
```



```
anas1010@ubuntulinux:~/Desktop/OS$ ./client
Enter your name: Samroze
-----WELCOME TO CHATROOM-----
> Hi anyone here?
> Uzair has joined the chat
> Hi Uzair
> Uzair: Hello
> Where were you? I've been waiting for you.
> Uzair: Sorry bro. Actually I went outside to get some groceries.
> Uzair: BTW How are you? and where is Anas?
> I'm sad as our team lost against ireland.
> I think he forgot about the meeting let me remind him.
> Anas has joined the chat
> Uzair: Here he is, "The SCATTERBRAIN"
> hahahahaha
> Anas: Hellooo! It is your fault. You must have reminded me.
> Anas: Don't worry I'll forgive you
> Lol
> Uzair: hahahaha
> Uzair: Guys I need to go, you guys continue.
> Anas: Thank God
> Bye
> Uzair: Bye...
> Uzair has left
> Anas: HAHAHAH SAMROZE Pakistan lost against ireland lol. What is your kabar doing? lol
> ^C
BYE
```



```
anas1010@ubuntulinux:~/Desktop/OS$ ./client
Enter your name: Uzair
-----WELCOME TO CHATROOM-----
> Samroze: Hi Uzair
> Hello
> Samroze: Where were you? I've been waiting for you.
> Sorry bro. Actually I went outside to get some groceries.
> BTW How are you? and where is Anas?
> Samroze: I'm sad as our team lost against ireland.
> Samroze: I think he forgot about the meeting let me remind him.
> Anas has joined the chat
> Here he is, "The SCATTERBRAIN"
> Samroze: hahahahaha
> Anas: Hellooo! It is your fault. You must have reminded me.
> Anas: Don't worry I'll forgive you
> Samroze: Lol
> hahahaha
> Guys I need to go, you guys continue.
> Anas: Thank God
> Samroze: Bye
> Bye...
> ^C
BYE
```



```
anas1010@ubuntulinux:~/Desktop/OS$ ./client
Enter your name: Anas
-----WELCOME TO CHATROOM-----
> Uzair: Here he is, "The SCATTERBRAIN"
> Samroze: hahahahaha
> Hellooo! It is your fault. You must have reminded me.
> Don't worry I'll forgive you
> Samroze: Lol
> Uzair: hahahaha
> Uzair: Guys I need to go, you guys continue.
> Thank God
> Samroze: Bye
> Uzair: Bye...
> Uzair has left
> HAHAHAH SAMROZE Pakistan lost against ireland lol. What is your king babar doing? lol
> Samroze has left
> sorryy
> ^C
BYE
```

==