OS LAB PROJECT

CHAT ROOM IN C USING THREADS

Submitted to: Sir Usman

Submitted By:

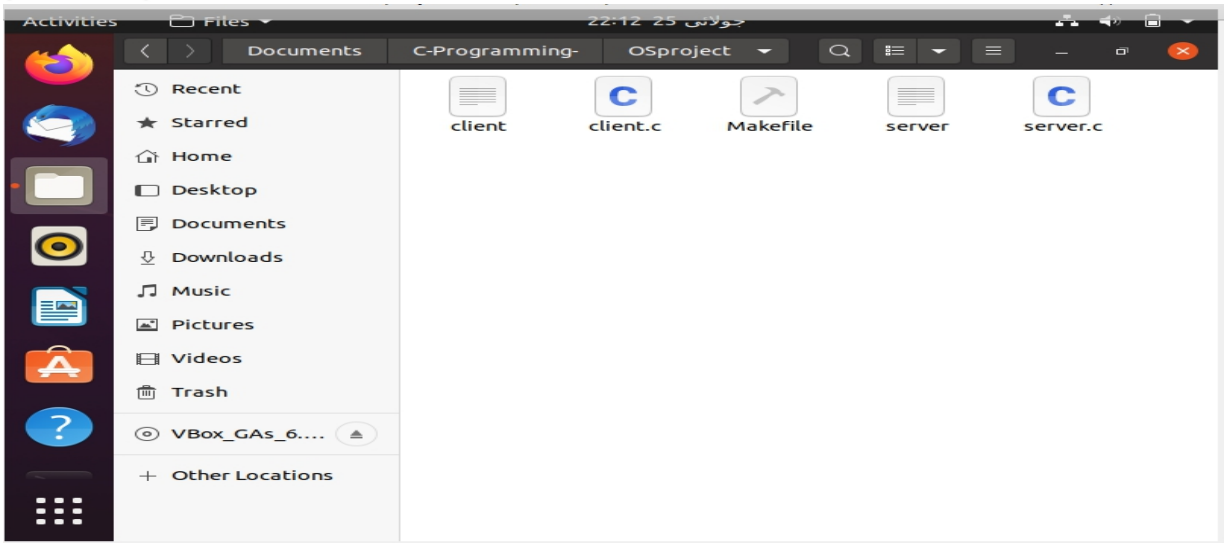
Haris Younas(2019-CS-683)
Muhammad Bilal(2019-CS-682)
Talha Pervaiz(2019-CS-672)

INTRODUCTION:

Multiple Clients will be able to communicate simultaneously in the same system via server communication.Server will keep track of each of its clients communication and it will also receive the same message and will show message that wich client joined the room.

PROJECT DETAILS:

There are three major files client.c,server.c and make the rest two are the compile version of the client.c and server.c files respectively.



The client.c file contains the code for connecting clients with the server file and server code is contained in server.c file which contains code for connecting different clients to the server.

Software Required:

Any distribution of UBUNTO will be sufficient to run this client server chat room implemented in c with the fact that c language compilation is available in the system using gcc compiler and the make is also required to be installed for running and using testing this project.We will go through each and every step required to run this code.Make is a command line tool used to build and maintain groups of programs and files from the source code.

Unfortunately many system calls and threads used in this code are not supported by windows c compilers as windows have different system calls implementation so you can test it on ubunto.

Installations required:

(Note:You can skip this if you have installed Make and c language compilation in your system.)

```
sudo apt install build-essential
```

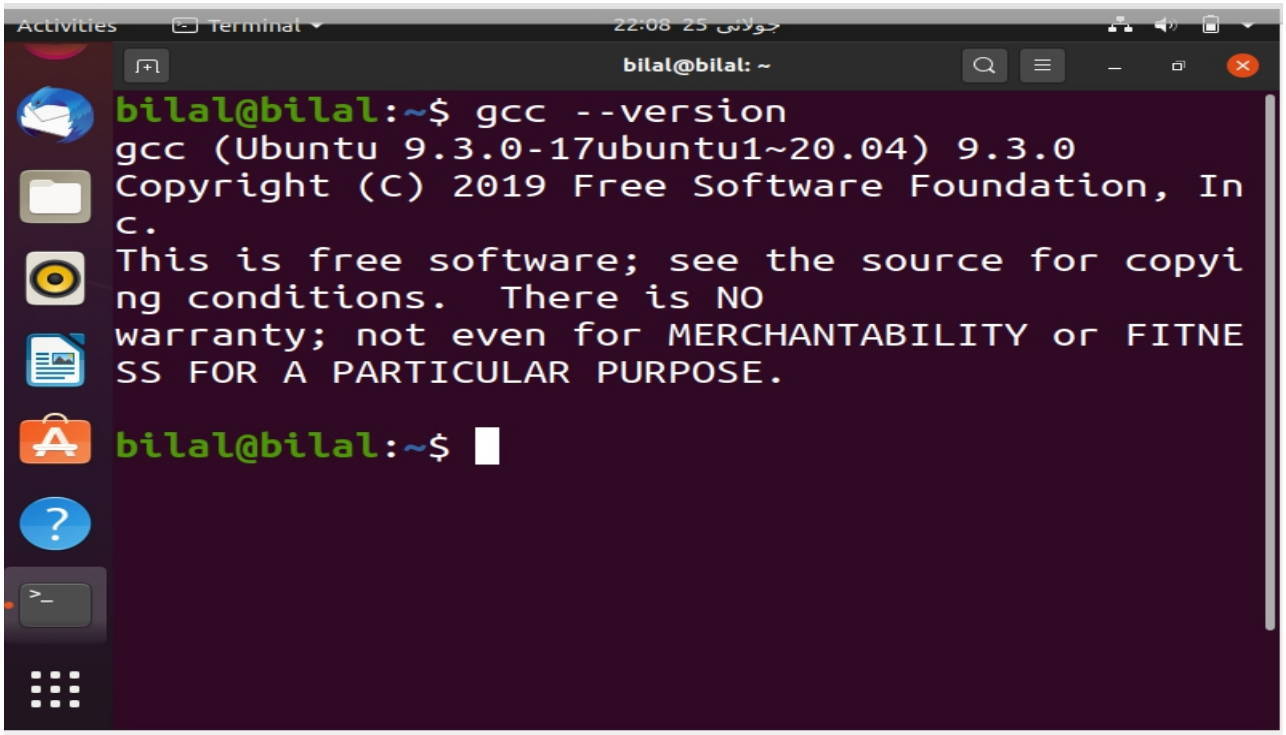
This command will install the gcc,g++ and make that are essential for this project to get running.

Verify your gcc version by this command

```
gcc --version
```

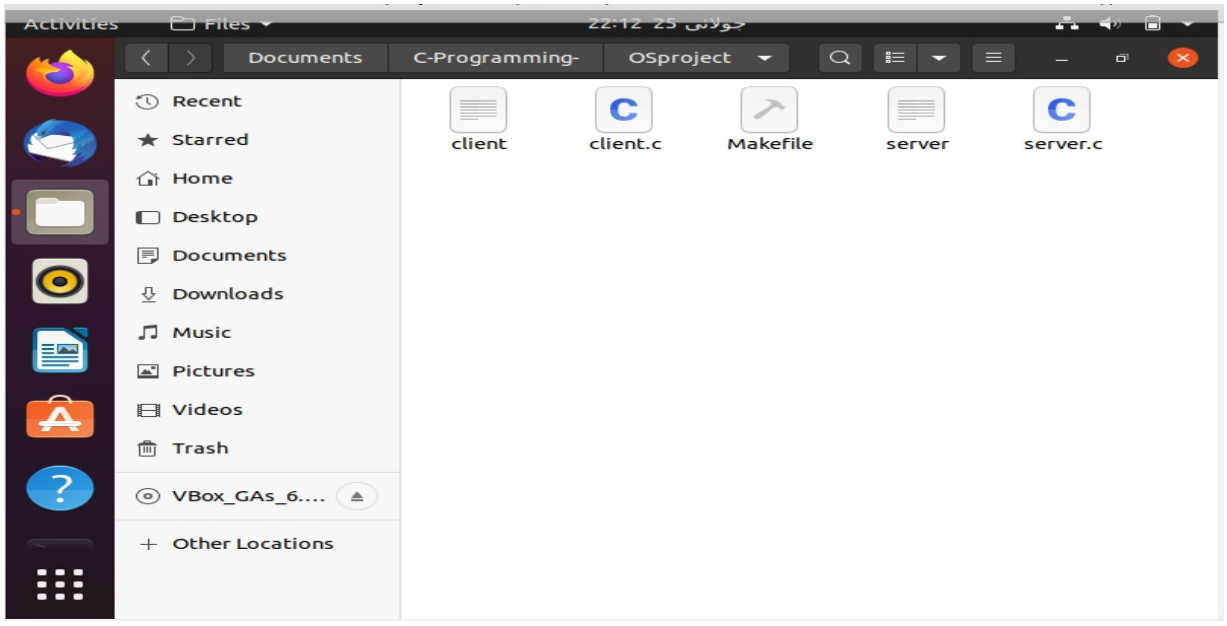
This command will give the output if the version of gcc is installed otherwise it will say you to install a gcc version and will

also give you a command to install so run that in other case otherwise the output will be like this expectedly.

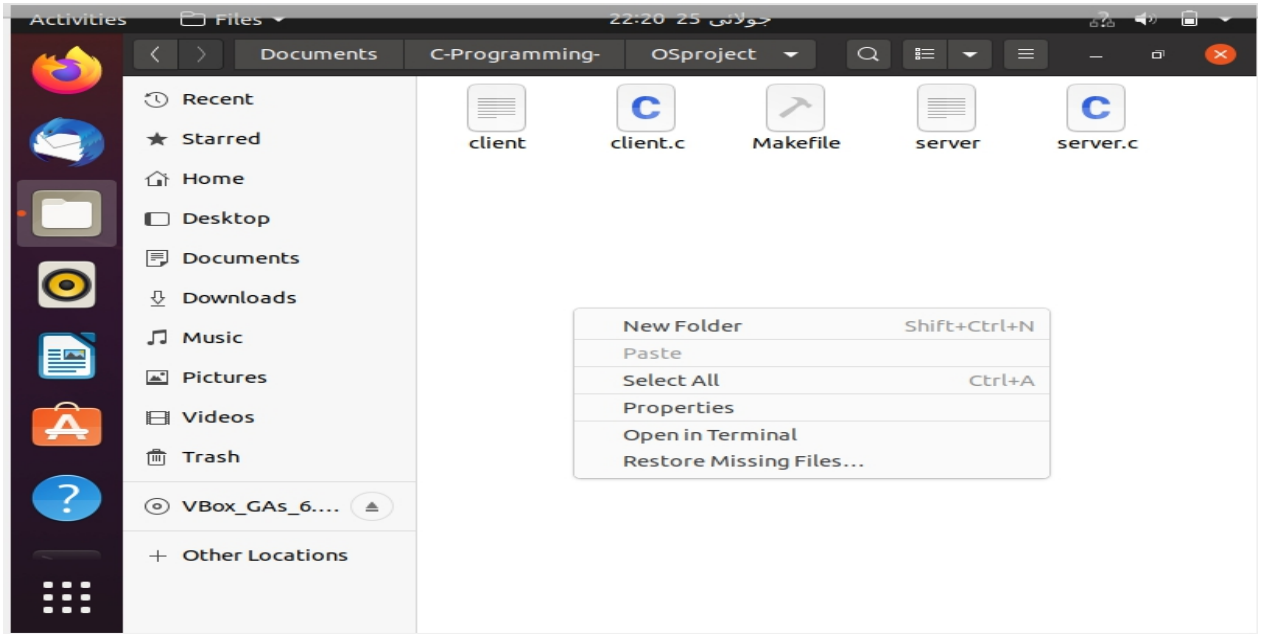


Steps required to run this project:

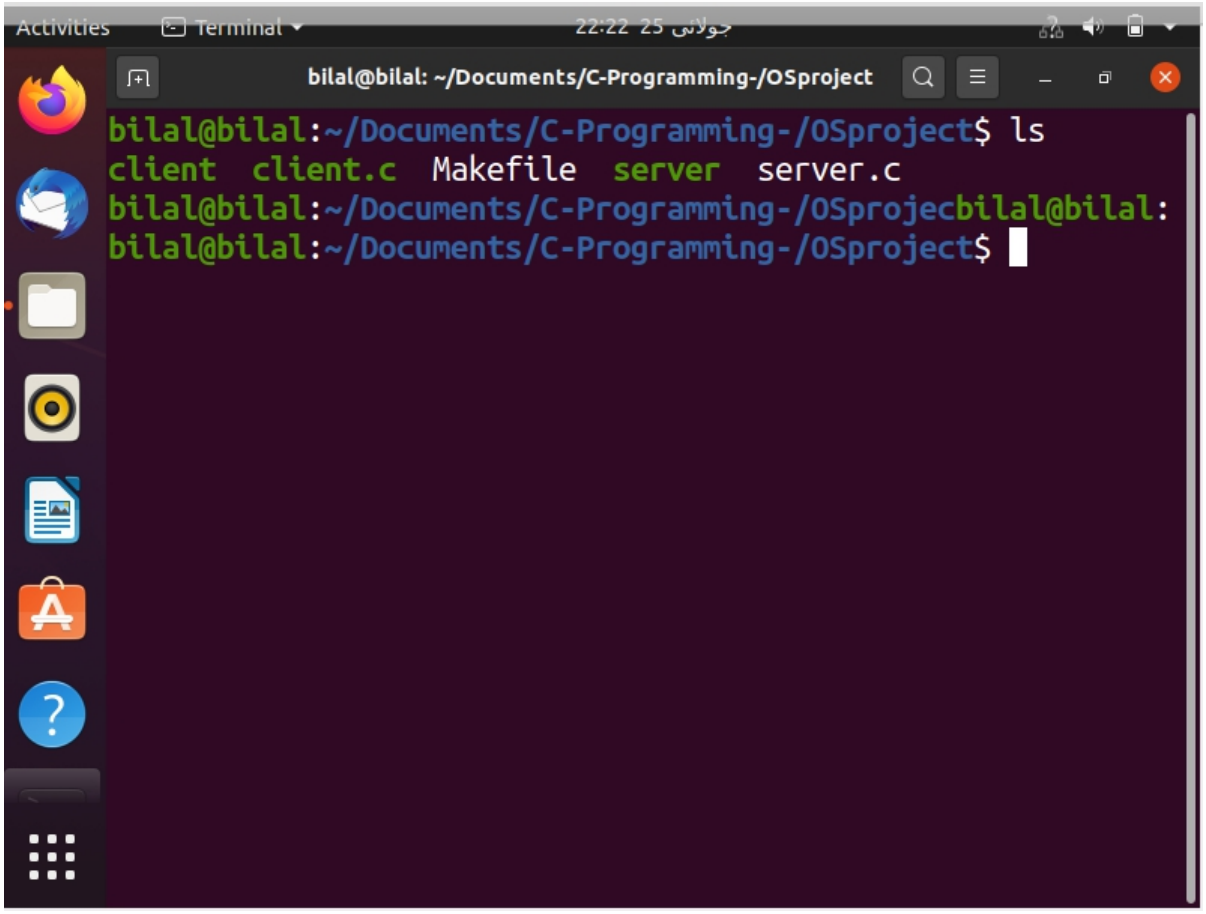
- ✓ Navigate to the project root directory. Like this all these three files should be present in that directory.



✓ Right click in the directory and open up terminal in that directory.

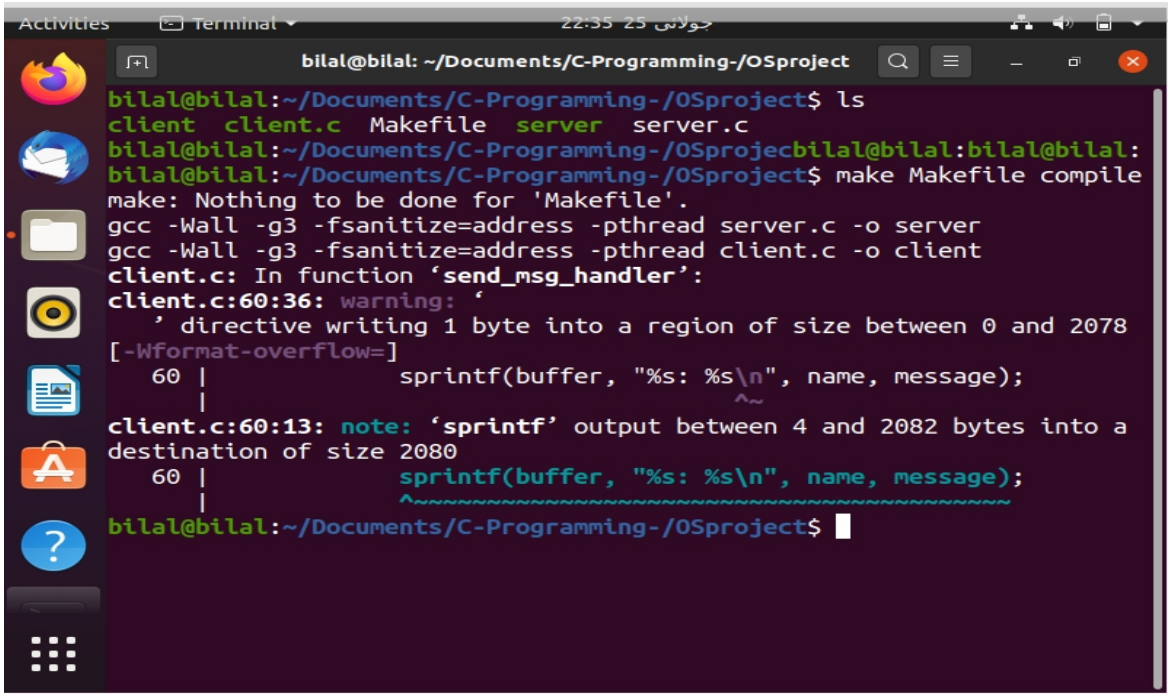


✓ Write the command ls in terminal to check whether all files are present.



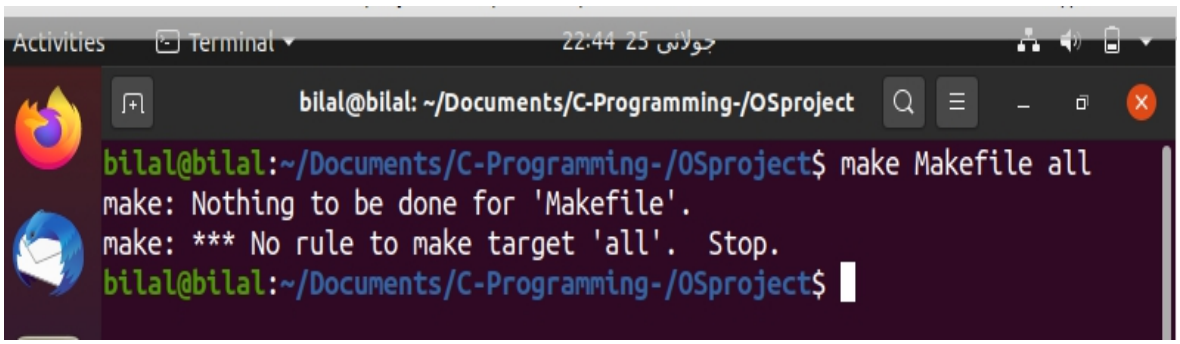
✓ Now that you have installed the required command line tools like make and gcc.Run the command

make Makefile compile



✓ Run the command

make Makefile all



✓ Run the command.

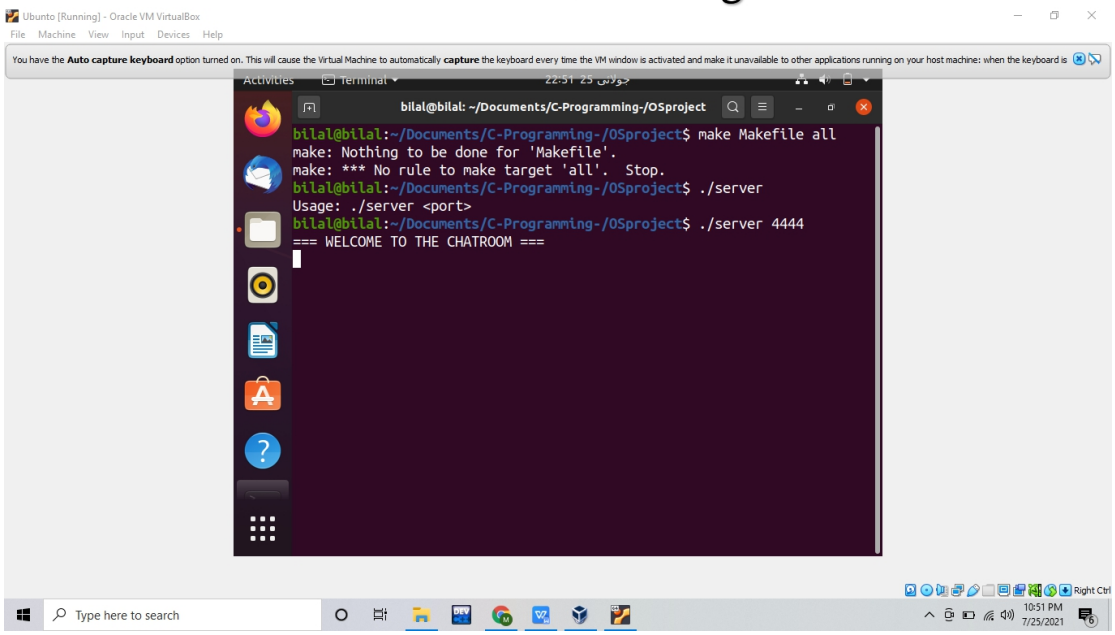
./server

```
bilal@bilal:~/Documents/C-Programming-/OSproject$ ./server
Usage: ./server <port>
bilal@bilal:~/Documents/C-Programming-/OSproject$
```

✓ Then Execute the command.

./server 4444

This will run the server on port 4444 with a WELCOME TO THE CHATROOM Message being displayed in the terminal.This means our server is running now.



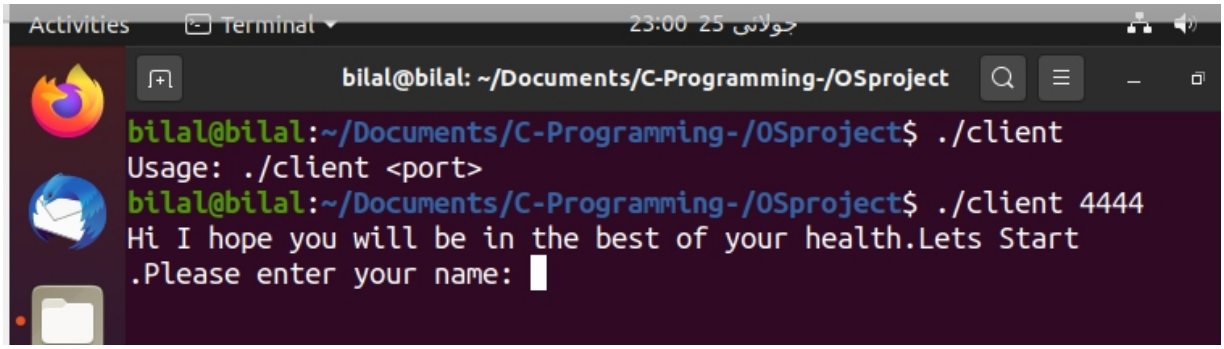
✓ Now lets connect client to the server.Right click in the folder directory of the project and again open up another terminal.Here we will write the command.

./client

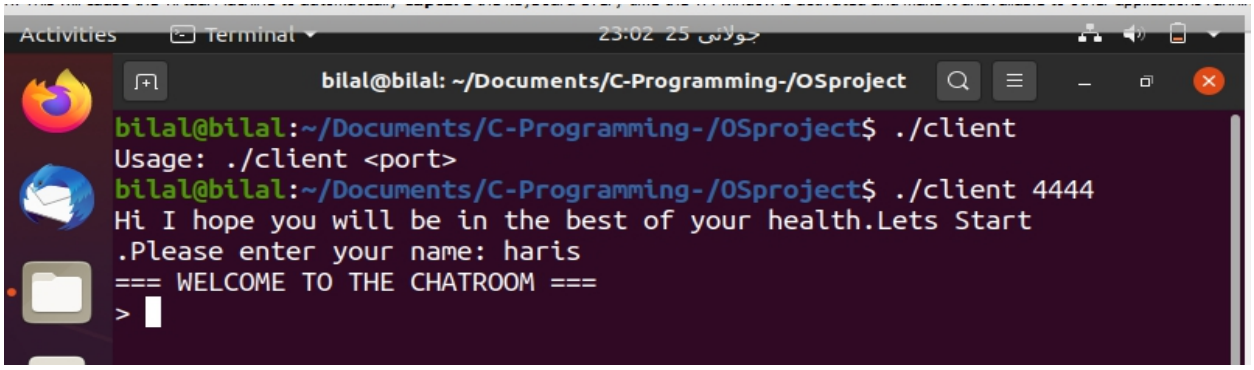
✓ And then write the command

./client 4444

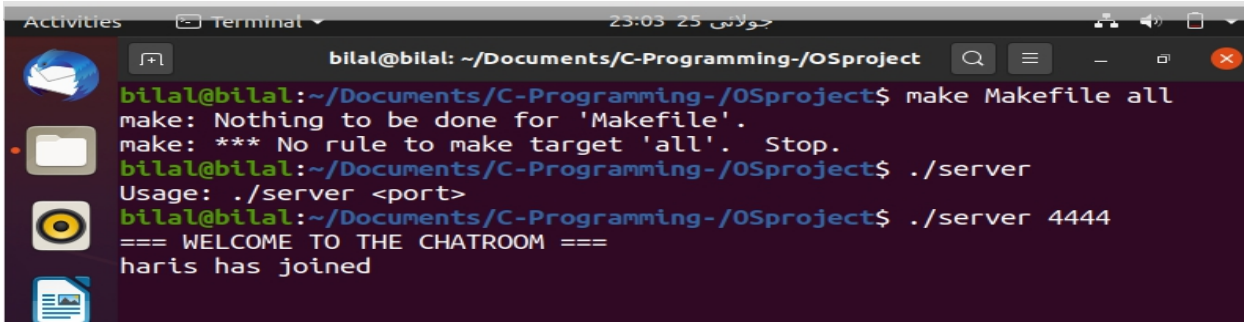
✓ After that this message will be displayed "Hi I hope you will be in the best of your health.Lets Start .Please enter your name:"



✓ Here write you name as a first client.Any thing any name.For example Haris.



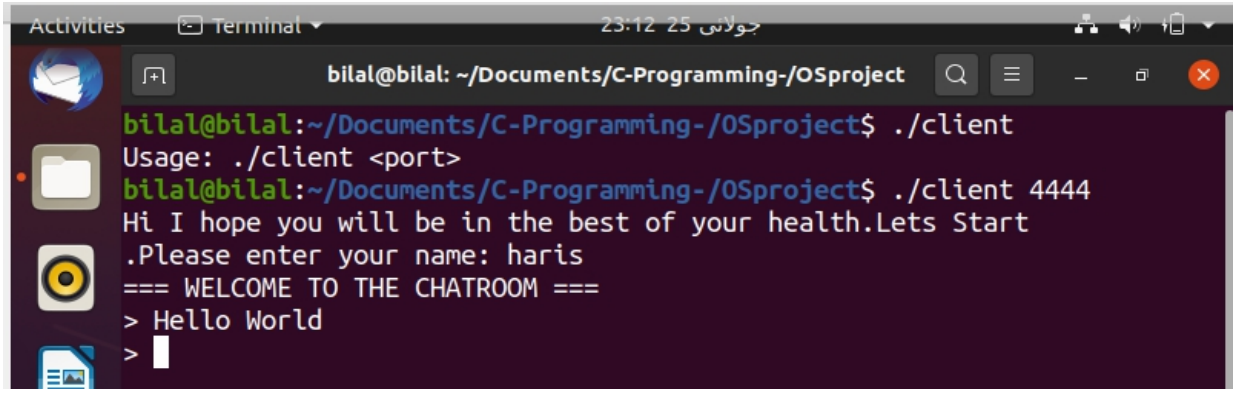
✓ You are now entered in the chat room as soon as you entered your name check back the server terminal.



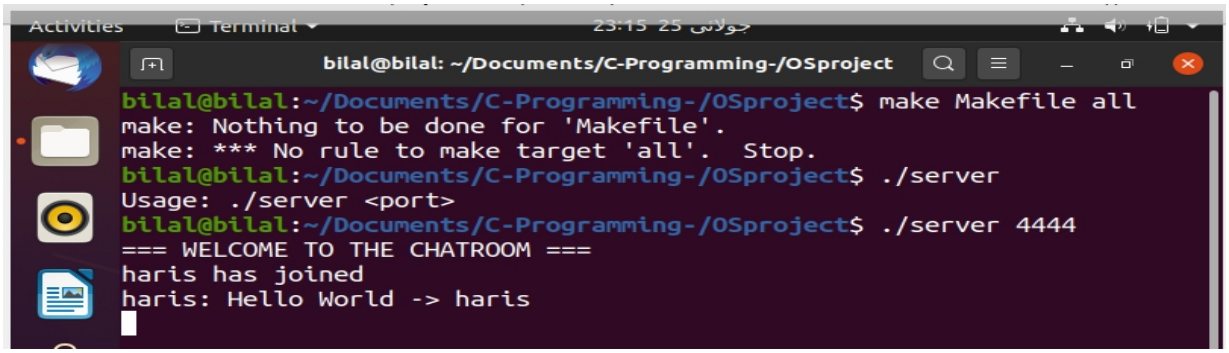
As you can notice on the server terminal that it updated automatically upon the joining of haris as a client and displayed his name on the terminal.So this is what means that server keeps track of every thing that happens under his hood.

We can communicate now but this communication will be entirely client server communication as no second client is involved till now.So lets first examine this.

- ✓ Enter any message in client terminal.For example type “Hello World”.



- ✓ Now again open up server terminal,you will notice that the same message is in the server terminal refrencing haris.



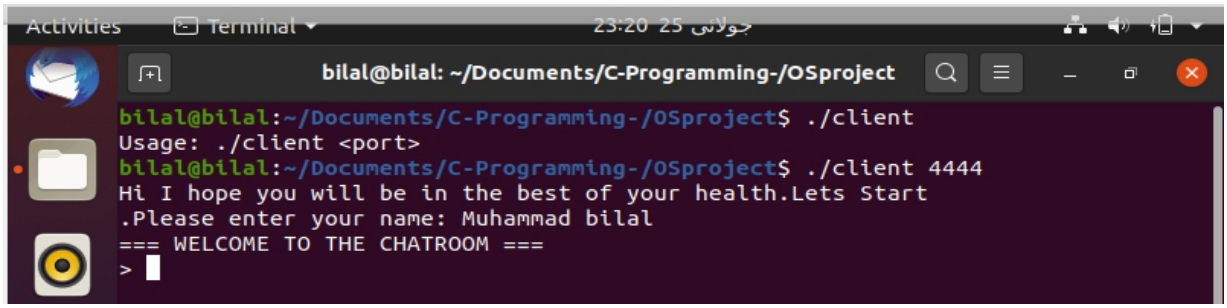
- ✓ Again open up another terminal.

✓ Execute the following commands simultaneously.

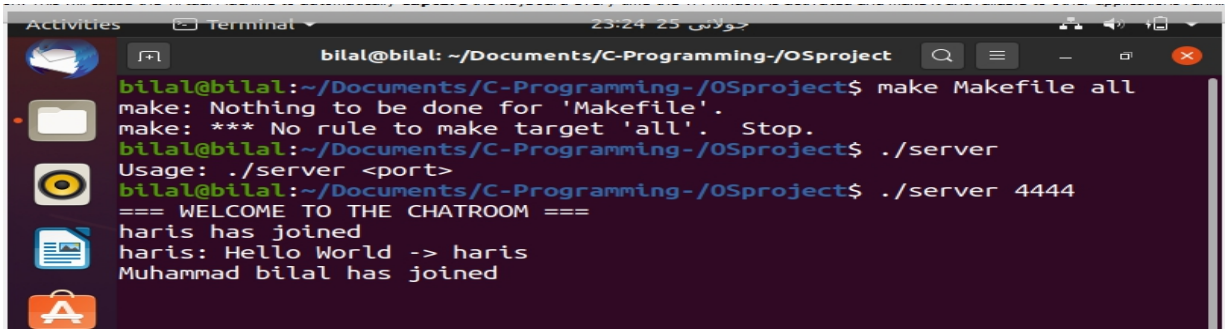
./client

./client 4444

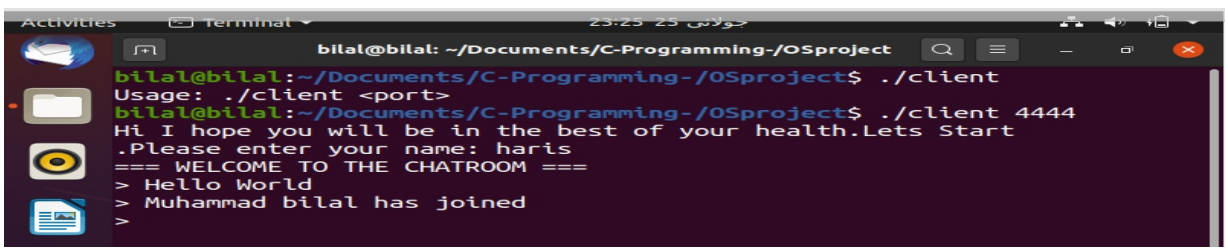
✓ Enter the name as For Example : “Muhammad Bilal”.



Same is the case here that server will display message upon the joining of “Muhammad Bilal”.Open the server terminal to notice that.

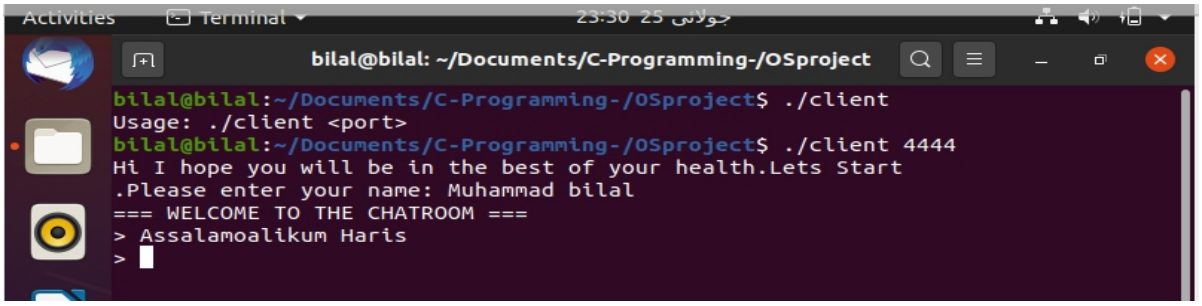


Not only in the server but the joining of “Muhammad Bilal” as a client is notified to all the previous clients which is in case only one client that has joined till now that is “haris”.This is also notified in the haris terminal.

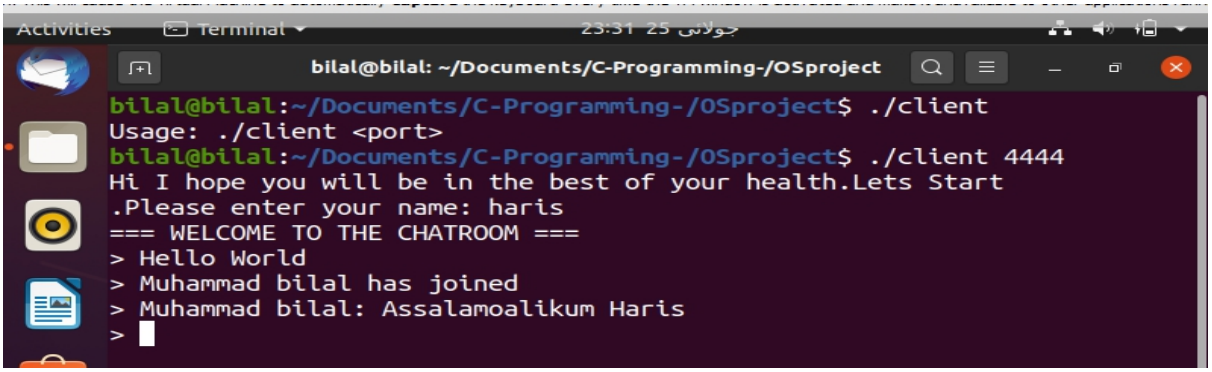


✓ Now enter a message from “Muhammad Bilal” terminal for example saying “Assamoalikum Haris”.

Muhammad Bilal Terminal



Haris Terminal.

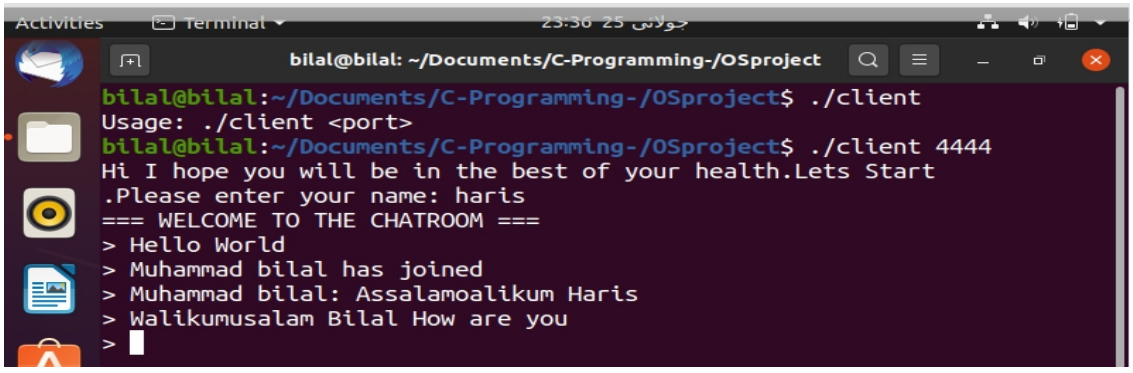


As you can notice the message from muhammad bilal was received to the haris.

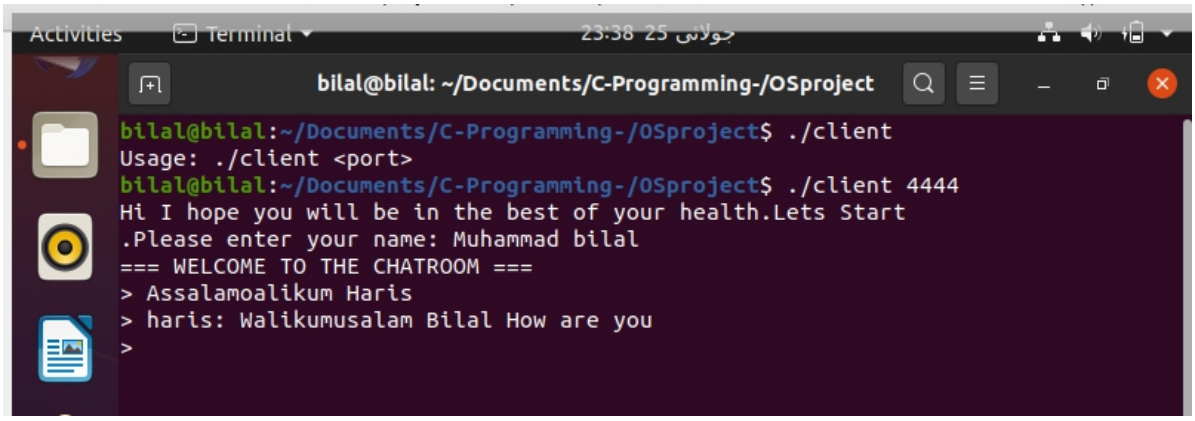
✓ Now lets reply back to muhammad bilal i.e message from Haris to Muhammad Bilal.

For example Haris replied “Walikumusalam How are you Bilal”.Then.

Haris Terminal



Bilal Terminal.



.

CONCLUSION:

In this way a communication is established from the server to the clients and multiple clients simultaneosly.So we discussed the communication for two clients but the clients can be as many as we need.This was just an example.You can open up more terminals and repeat the procedure and check if that works for you.Thanks.

DRAW BACKS:

The communication is just local.This isnt the scenario in the real world as the two clients will be in the different places and at different systems.So for that network programming would be needed in order to communicate for the two clients remotely aslo sockets are used for that purpose.Thanks.

Makefile

```
compile:
	gcc -Wall -g3 -fsanitize=address -pthread server.c -o server
	gcc -Wall -g3 -fsanitize=address -pthread client.c -o client
```

Server.c

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.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>

#define MAX_CLIENTS 100
#define BUFFER_SZ 2048

static _Atomic unsigned int cli_count = 0;
static int uid = 10;

/* Client structure */
typedef struct
{
   struct sockaddr_in address;
   int sockfd;
   int uid;
   char name[32];
} client_t;

client_t *clients[MAX_CLIENTS];

pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;

void str_overwrite_stdout()
{
   printf("\r%s", "> ");
   fflush(stdout);
}

void str_trim_lf(char *arr, int length)
{
   int i;
```

```
    for (i = 0; i < length; i++)
    { // trim \n
        if (arr[i] == '\n')
        {
            arr[i] = '\0';
            break;
        }
    }
}

void print_client_addr(struct sockaddr_in addr)
{
    printf("%d.%d.%d.%d",
           addr.sin_addr.s_addr & 0xff,
           (addr.sin_addr.s_addr & 0xff00) >> 8,
           (addr.sin_addr.s_addr & 0xff0000) >> 16,
           (addr.sin_addr.s_addr & 0xff000000) >> 24);
}

/* Add clients to queue */
void queue_add(client_t *cl)
{
    pthread_mutex_lock(&clients_mutex);

    for (int i = 0; i < MAX_CLIENTS; ++i)
    {
        if (!clients[i])
        {
            clients[i] = cl;
            break;
        }
    }

    pthread_mutex_unlock(&clients_mutex);
}

/* Remove clients to queue */
void queue_remove(int uid)
{
    pthread_mutex_lock(&clients_mutex);

    for (int i = 0; i < MAX_CLIENTS; ++i)
```

```
    {
        if (clients[i])
        {
            if (clients[i]->uid == uid)
            {
                clients[i] = NULL;
                break;
            }
        }
    }

    pthread_mutex_unlock(&clients_mutex);
}

/* Send message to all clients except sender */
void send_message(char *s, int uid)
{
    pthread_mutex_lock(&clients_mutex);

    for (int i = 0; i < MAX_CLIENTS; ++i)
    {
        if (clients[i])
        {
            if (clients[i]->uid != uid)
            {
                if (write(clients[i]->sockfd, s, strlen(s)) < 0)
                {
                    perror("ERROR: write to descriptor failed");
                    break;
                }
            }
        }
    }

    pthread_mutex_unlock(&clients_mutex);
}

/* Handle all communication with the client */
void *handle_client(void *arg)
{
    char buff_out[BUFFER_SZ];
    char name[32];
```

```
int leave_flag = 0;

cli_count++;
client_t *cli = (client_t *)arg;

// Name
if (recv(cli->sockfd, name, 32, 0) <= 0 || strlen(name) < 2 || strlen(name) >= 32 - 1)
{
   printf("Didn't enter the name.\n");
   leave_flag = 1;
}
else
{
   strcpy(cli->name, name);
   sprintf(buff_out, "%s has joined\n", cli->name);
   printf("%s", buff_out);
   send_message(buff_out, cli->uid);
}

bzero(buff_out, BUFFER_SZ);

while (1)
{
   if (leave_flag)
   {
      break;
   }

   int receive = recv(cli->sockfd, buff_out, BUFFER_SZ, 0);
   if (receive > 0)
   {
      if (strlen(buff_out) > 0)
      {
         send_message(buff_out, cli->uid);

         str_trim_lf(buff_out, strlen(buff_out));
         printf("%s -> %s\n", buff_out, cli->name);
      }
   }
   else if (receive == 0 || strcmp(buff_out, "exit") == 0)
   {
      sprintf(buff_out, "%s has left\n", cli->name);
```

```
            printf("%s", buff_out);
            send_message(buff_out, cli->uid);
            leave_flag = 1;
        }
        else
        {
            printf("ERROR: -1\n");
            leave_flag = 1;
        }

        bzero(buff_out, BUFFER_SZ);
    }

    /* Delete client from queue and yield thread */
    close(cli->sockfd);
    queue_remove(cli->uid);
    free(cli);
    cli_count--;
    pthread_detach(pthread_self());

    return NULL;
}

int main(int argc, char **argv)
{
    if (argc != 2)
    {
        printf("Usage: %s <port>\n", argv[0]);
        return EXIT_FAILURE;
    }

    char *ip = "127.0.0.1";
    int port = atoi(argv[1]);
    int option = 1;
    int listenfd = 0, connfd = 0;
    struct sockaddr_in serv_addr;
    struct sockaddr_in cli_addr;
    pthread_t tid;

    /* Socket settings */
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
```

```
    serv_addr.sin_addr.s_addr = inet_addr(ip);
    serv_addr.sin_port = htons(port);

    /* Ignore pipe signals */
    signal(SIGPIPE, SIG_IGN);

    if (setsockopt(listenfd, SOL_SOCKET, (SO_REUSEPORT | SO_REUSEADDR), (char
*)&option, sizeof(option)) < 0)
    {
        perror("ERROR: setsockopt failed");
        return EXIT_FAILURE;
    }

    /* Bind */
    if (bind(listenfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        perror("ERROR: Socket binding failed");
        return EXIT_FAILURE;
    }

    /* Listen */
    if (listen(listenfd, 10) < 0)
    {
        perror("ERROR: Socket listening failed");
        return EXIT_FAILURE;
    }

    printf("=== WELCOME TO THE CHATROOM ===\n");

    while (1)
    {
        socklen_t clilen = sizeof(cli_addr);
        connfd = accept(listenfd, (struct sockaddr *)&cli_addr, &clilen);

        /* Check if max clients is reached */
        if ((cli_count + 1) == MAX_CLIENTS)
        {
            printf("Max clients reached. Rejected: ");
            print_client_addr(cli_addr);
            printf(":%d\n", cli_addr.sin_port);
            close(connfd);
            continue;
```

```
        }

        /* Client settings */
        client_t *cli = (client_t *)malloc(sizeof(client_t));
        cli->address = cli_addr;
        cli->sockfd = connfd;
        cli->uid = uid++;

        /* Add client to the queue and fork thread */
        queue_add(cli);
    }
        pthread_create(&tid, NULL, &handle_client, (void *)cli);

        /* Reduce CPU usage */
        sleep(1);

    return EXIT_SUCCESS;
}
```

Client.c

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

#define LENGTH 2048

// Global variables
volatile sig_atomic_t flag = 0;
```

```
int sockfd = 0;
char name[32];

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

void str_trim_lf(char *arr, int length)
{
   int i;
   for (i = 0; i < length; i++)
   { // trim \n
      if (arr[i] == '\n')
      {
         arr[i] = '\0';
         break;
      }
   }
}

void catch_ctrl_c_and_exit(int sig)
{
   flag = 1;
}

void send_msg_handler()
{
   char message[LENGTH] = {};
   char buffer[LENGTH + 32] = {};

   while (1)
   {
      str_overwrite_stdout();
      fgets(message, LENGTH, stdin);
      str_trim_lf(message, LENGTH);

      if (strcmp(message, "exit") == 0)
      {
         break;
      }
```

```
        else
        {
            sprintf(buffer, "%s: %s\n", name, message);
            send(sockfd, buffer, strlen(buffer), 0);
        }

        bzero(message, LENGTH);
        bzero(buffer, LENGTH + 32);
    }
    catch_ctrl_c_and_exit(2);
}

void recv_msg_handler()
{
    char message[LENGTH] = {};
    while (1)
    {
        int receive = recv(sockfd, message, LENGTH, 0);
        if (receive > 0)
        {
            printf("%s", message);
            str_overwrite_stdout();
        }
        else if (receive == 0)
        {
            break;
        }
        else
        {
            // -1
        }
        memset(message, 0, sizeof(message));
    }
}

int main(int argc, char **argv)
{
    if (argc != 2)
    {
        printf("Usage: %s <port>\n", argv[0]);
        return EXIT_FAILURE;
    }
```

```
  char *ip = "127.0.0.1";
  int port = atoi(argv[1]);

  signal(SIGINT, catch_ctrl_c_and_exit);

  printf("Hi I hope you will be in the best of your health.Lets Start\n.Please enter your
name: ");
  fgets(name, 32, stdin);
  str_trim_lf(name, strlen(name));

  if (strlen(name) > 32 || strlen(name) < 2)
  {
    printf("Name must be less than 30 and more than 2 characters.\n");
    return EXIT_FAILURE;
  }

  struct sockaddr_in server_addr;

  /* Socket settings */
  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(port);

  // Connect to Server
  int err = connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
  if (err == -1)
  {
    printf("ERROR: connect\n");
    return EXIT_FAILURE;
  }

  // Send name
  send(sockfd, name, 32, 0);

  printf("=== WELCOME TO THE CHATROOM ===\n");

  pthread_t send_msg_thread;
  if (pthread_create(&send_msg_thread, NULL, (void *)send_msg_handler, NULL) !=
0)
  {
```

```
        printf("ERROR: pthread\n");
        return EXIT_FAILURE;
    }

    pthread_t recv_msg_thread;
    if (pthread_create(&recv_msg_thread, NULL, (void *)recv_msg_handler, NULL) != 0)
    {
        printf("ERROR: pthread\n");
        return EXIT_FAILURE;
    }

    while (1)
    {
        if (flag)
        {
            printf("\nAllah Hafiz\n");
            break;
        }
    }

    close(sockfd);

    return EXIT_SUCCESS;
}
```