Client :

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
// to read(),write() and close()
#include <sys/types.h>
//#include <sys/types.h> -> this contain definatoins of number of data types
usen in system call
// ans also used for #include <sys/socket.h> and #include <netinet/in.h>
#include <sys/socket.h>
//#include <sys/socket.h> -> contain defination and stuctures such as
'sockaddr'
#include <netinet/in.h>
//#include <netinet/in.h> -> contain defination,constat and sturcture (e.g.-
'sockaddr_in')for internet domain address
#include <arpa/inet.h>
//#include <arpa/inet.h> -> contains definitions for internet operations .
#include <pthread.h> // for threading
#define LENGTH 2048
// Global variables
volatile sig_atomic_t flag = 0;
int sockfd = 0; // socket id
char name[32];
void str_overwrite()
{
    printf("%s", "> ");
    fflush(stdout);
    // to flush the output buffer of the stream
}
void trim(char *arr, int length)
{
    int i;
    for (i = 0; i < length; i++)
    { // trim the length of the buffer/message.
        if (arr[i] == '\n')
        {
           arr[i] = '\0';
            break;
        }
    }
}
void LEAVE_ROOM(int sig)
{
    // function to disconnect the client
    flag = 1;
```

```
}
void send_msg()
{
    char message[LENGTH] = {};
    char buffer[LENGTH + 32] = {};
    while (1)
    {
        str_overwrite();
        fgets(message, LENGTH, stdin);
        trim(message, LENGTH);
        if (strcmp(message, "Bye") == 0)
        {
            // if the client types "Bye",it instructs the code to shut doen
the client.
            break;
        }
        else
        {
            sprintf(buffer, "%s: %s\n", name, message); // prints the input
message on the same screen.
            send(sockfd, buffer, strlen(buffer), 0); // sends the message to
the server.
        }
        bzero(message, LENGTH); // clears the message.
        bzero(buffer, LENGTH + 32); // clears the buffer.
    }
    LEAVE_ROOM(2); // instructs the code to exit the client.
}
void recv_msg()
{
    char message[LENGTH] = {};
    while (1)
    {
        int receive = recv(sockfd, message, LENGTH, 0); // receives the
message from the server.
        if (receive > 0)
        {
            // prints the message if no error occurs.
            printf("%s", message);
            str_overwrite(); // clears stdout.
        }
        else if (receive == 0) // connection lost.
            break;
        else
        {
            // -1->error
        }
        memset(message, 0, sizeof(message));
```

```
    }
}
int main(int argc, char **argv)
{
    // argc-> server parameter
    // agrv-> port number
    // to verify the server parameter and port number =>
    if (argc != 2)
    {
        printf("Usage: %s <port>\n", argv[0]);
        return EXIT_FAILURE;
    }
    char *ip = "127.0.0.1";
    int port = atoi(argv[1]); // converting ip string to int (port)
    signal(SIGINT, LEAVE_ROOM); // signal to check if to close the client
    printf("Please enter your name: ");
    fgets(name, 32, stdin);
    trim(name, strlen(name));
    // rejecting the name if greater than 32 characters
    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;
    // sockadd_in structure...
    // 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) // if err is -1 then do Exit
    {
        printf("ERROR: connect\n");
        return EXIT_FAILURE;
    }
    // Send name
    send(sockfd, name, 32, 0);
    printf("\t\t=====WELCOME TO THE CHATROOM=====\n");
    // creating thread to send the message
    pthread_t send_msg_thread;
    if (pthread_create(&send_msg_thread, NULL, (void *)send_msg, NULL) != 0)
    {
        printf("ERROR: pthread\n");
        return EXIT_FAILURE;
```

```
    }
    // creating thread to receive the message
    pthread_t recv_msg_thread;
    if (pthread_create(&recv_msg_thread, NULL, (void *)recv_msg, NULL) != 0)
    {
        printf("ERROR: pthread\n");
        return EXIT_FAILURE;
    }
    while (1)
    {
        if (flag) // to disconnect the client.
        {
            printf("\nBye\n");
            printf("%s has left the ChatRoom.", name);
            break;
        }
    }
    close(sockfd); // closing socket.
    return EXIT_SUCCESS;
}
```

Server:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>
#include <errno.h>
#define MAX_USERS 100
#define BUFFER_SZ 2048
#define NAME_LEN 80

static _Atomic unsigned int cli_count = 0;
// Atomic variables can be accessed concurrently between different threads
without creating race conditions.
static int user_id = 10;

// Client structure will store information about client's address,name, id
typedef struct
{
    struct sockaddr_in address;
    int sockfd; // socket descripter
    int user_id;
    char name[NAME_LEN];
} client_t;

client_t *clients[MAX_USERS]; // CLient array to store clients that join the
server

// initialising mutex lock
pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;

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

void trim(char *arr, int length)
{
    int i;
    for (i = 0; i < length; i++)
    { // trim the length of the buffer/message.
```

```
        if (arr[i] == '\n')
        {
            arr[i] = '\0';
            break;
        }
    }
}

void print_ip_addr(struct sockaddr_in addr, int port)
{
    printf("\t\tCurrent IP Address of the server is : ");
    printf("%d.%d.%d.%d\n",
           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);
    printf("\t\tPort Number is : %d\n", port);
}

// add the client to the queue
void connect_to_server(client_t *cl)
{
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < MAX_USERS; i++)
    { // if client is empty at an index,we assign that index to the client
that is requesting to join
        if (!clients[i])
        {
            clients[i] = cl;
            break;
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}

// remove the client from the queue
void remove_from_server(int user_id)
{
    pthread_mutex_lock(&clients_mutex);

    for (int i = 0; i < MAX_USERS; i++)
    {
        if (clients[i])
        {
            if (clients[i]->user_id == user_id)
            {
                clients[i] = NULL;
                break;
```

```
            }
        }
    }

    pthread_mutex_unlock(&clients_mutex);
}

// this function takes the message and user_id and sends to all the users
except itself
void message(char *s, int user_id)
{
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < MAX_USERS; i++)
    {
        if (clients[i]) // check that if client exists
        {
            if (clients[i]->user_id != user_id)
            {
                if (write(clients[i]->sockfd, s, strlen(s)) < 0)
                {
                    printf("ERROR: write to descriptor failed\n");
                    break;
                }
            }
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}

void *handle_client(void *args)
{
    char buffer[BUFFER_SZ];
    char name[NAME_LEN];
    int leave_flag = 0;
    cli_count++;

    client_t *cli = (client_t *)args;

    // Recieving name from the client
    if (recv(cli->sockfd, name, NAME_LEN, 0) <= 0 || strlen(name) < 2 ||
strlen(name) >= NAME_LEN - 1)
    {
        printf("Enter the name correctly!\n");
        leave_flag = 1;
    }
    else
    {
```

```
        strcpy(cli->name, name);                              // copying client
name recieved from client in client structure
        sprintf(buffer, "\n\t\t%s HAS JOINED\n", cli->name); // sending
message to all other users
        printf("%s\n", buffer);
        message(buffer, cli->user_id);
    }

    bzero(buffer, BUFFER_SZ);

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

        int receive = recv(cli->sockfd, buffer, BUFFER_SZ, 0);

        if (receive > 0)
        {
            if (strlen(buffer) > 0 || strcmp(buffer, "SEND") == 0)
            {
                message(buffer, cli->user_id);
                trim(buffer, strlen(buffer));
                printf("%s\n", buffer);
            }
        }
        else if (receive == 0 || strcmp(buffer, "Bye") == 0)
        {
            sprintf(buffer, "%s has left\n", cli->name);
            printf("\t\t%s\n", buffer);
            message(buffer, cli->user_id);
            leave_flag = 1;
        }
        else
        {
            printf("ERROR in program \n");
            leave_flag = 1;
        }
        bzero(buffer, BUFFER_SZ); // zero() function erases the data in the n
bytes of the memory
    }

    close(cli->sockfd);
    remove_from_server(cli->user_id);
    free(cli);
    cli_count--;
```

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

int main(int argc, char **argv)
{
    if (argc < 2)
    {
        printf("Port Number not provided.Program Terminated.\n");
        printf("Usage: %s <port>\n", argv[0]);
        return EXIT_FAILURE;
    }

    char *ip = "127.0.0.1";
    // '127.0.0.1' is a special-purpose IPv4 address and is called the
localhost or loopback address.
    int port = atoi(argv[1]);
    // stores port number entered by user

    int option = 1;
    // These are the basic structures for all syscalls and functions that deal
with internet addresses.
    struct sockaddr_in serv_addr;
    struct sockaddr_in cli_addr; // Client structure will store information
about client's address,name, id
    pthread_t tid;
    // SOCKET SETTING
    int listenfd = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(ip); // stores IP address
    serv_addr.sin_port = htons(port);          // stores port number

    // SIGNAL
    signal(SIGPIPE, SIG_IGN); // iterrupt

    if (setsockopt(listenfd, SOL_SOCKET, (SO_REUSEPORT | SO_REUSEADDR), (char
*)&option, sizeof(option)) < 0)
    {
        printf("ERROR: setsockopt\n");
        return EXIT_FAILURE;
    }
    //  bind() assigns the address specified by addr to the socket referred to
by the file descriptor sockfd.
    if (bind(listenfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("ERROR caused in bind call bind\n");
        return EXIT_FAILURE;
    }
```

```c
    // listen -  indicates a readiness to accept client connection requests.
    if (listen(listenfd, 10) < 0)
    {
        printf("ERROR cased in listen call\n");
        return EXIT_FAILURE;
    }

    print_ip_addr(serv_addr, port);
    printf("\t\t===== Welcome to my server =====\n");
    while (1)
    {
        socklen_t clilen = sizeof(cli_addr);
        // accepting client and connecting it to server
        int connfd = accept(listenfd, (struct sockaddr *)&cli_addr, &clilen);
        if (connfd < 0)
        {
            printf("ERROR caused in accept call.");
            return EXIT_FAILURE;
        }
        // check for max-clients
        if ((cli_count + 1) == MAX_USERS)
        {
            printf("Maximum clients connected.\n ");
            printf("Sorry,cannot allow you into the ChatRoom.Try again
later.");
            print_ip_addr(cli_addr, port);
            close(connfd);
            continue;
        }
        // Client Settings
        client_t *cli = (client_t *)malloc(sizeof(client_t));
        cli->address = cli_addr;
        cli->sockfd = connfd;
        cli->user_id = user_id++;
        // Add client to queue
        connect_to_server(cli);
        pthread_create(&tid, NULL, &handle_client, (void *)cli);
        // To reduce load on CPU
        sleep(1);
    }
    return EXIT_SUCCESS;
}
```