# **SIMPLE CHAT CLIENT SERVER**

*A*

*Mini Project Report*

*Submitted in partial fulfilment of the*

*Requirements for the award of the Degree of*

**BACHELOR OF ENGINEERING**

IN

**INFORMATION TECHNOLOGY**

By

MANOJ KUMAR – 1602-19-737-139

SHITIJ REDDY – 1602-19-737-168



**Department of Information Technology**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Ibrahimbagh, Hyderabad-31**

**2020**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Hyderabad-500 031**

**Department of Information Technology**



**DECLARATION BY THE CANDIDATE**

We, MANOJ KUMAR and SHITIJ REDDY bearing hall ticket numbers, 1602-19-737-139 and 1602-19-737-168, hereby declare that the project report entitled “SIMPLE CHAT CLIENT SERVER” is submitted in partial fulfilment of the requirement for the award of the degree of Bachelor of Engineering in Information Technology.

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

MANOJ KUMAR

1602-19-737-139

SHITIJ REDDY

1602-19-737-168

(Faculty In-Charge) (Head.Dept IT)

**Acknowledgements**

The satisfaction that accompanies the successful completion of this project would not be in complete without the mention of the people who made it possible, without whose constant guidance and encouragement would have made efforts go in vain. We consider ourselves privileged to express gratitude and respect towards all those who guided us through the completion of this project.

We convey thanks to my project guide Mrs. Divya of Information technology Department for providing encouragement, constant support and guidance which was of a great help to complete this project successfully.

Last but not the least, we wish to thank our parents for financing our studies in this privileged Vasavi College of Engineering as well as for constantly encouraging us to learn engineering. Their personal sacrifice in providing this opportunity to learn engineering is gratefully acknowledged.

**Abstract**

Our Project is “Simple Chat Client Server”. Chat refers to the process of communications, interacting and (or) exchanging messages over the Internet. It involves two or more individuals that communicate through a chat enabled Service or software. Chat may be delivered through text via the Internet. A server is a computer program or a device that provides two or more functionality for other programs or devices. Clients want to chat with each other connect to the server. The chat Application we are going to make will be more like a chat room, rather than peer to peer chat. So, this means multiple users can Connect to the chat server and send their messages.

**Table of contents**

|  |  |
| --- | --- |
| **TOPIC** | **PAGE NUMBER** |
| INTRODUCTION | **6** |
| TECHNOLOGY | **7** |
| PROPOSED WORK | **8** |
| RESULTS | **32** |
| ADDITIONAL KNOWLEDGE | **34** |
| CONCLUSION | **35** |
| REFERENCE | **36** |

**INTRODUCTION**

Chat server is an application which does the following operations: Client running in any PC can connect to the server if IP address of the server is known. Listens for messages from all the connected clients .Broadcasts the message from clients to all the clients connected to the server.

**ADVANTAGES:**

The **advantages** of a **client**-**server** network are greater security of the network, more control e.g. network traffic passing through the network, being able to see what each computer is doing and limiting certain actions and preventing things such as viruses spreading, also the amount of data storage available to each

**CLIENT SIDE:**

1. Enter server details

2. Join chat room

3. User details

4. Client status

5. Leave chat room

6. Send messages

Firstly, in user side, client join a chat room by entering an appropriate Server IP address, then they can chat with each other with ease.

**SERVER SIDE:**

1. Create Chat room

2. Authenticate server details

3. Notifications (or) Client Status

4. Display messages

5. Leave (or) End Chat room

Firstly , Server can create a chat room with a particular IP address and port number. Then, the clients can join the server i.e., the chat room by entering that port number created by the server.

Now, Server prompts the user to enter their name and likewise details. The server will be able to read the messages sent by clients. Also, it can end the chat room by leaving the chat room.

**Technology**

1. **SOFTWARE REQUIREMENTS**

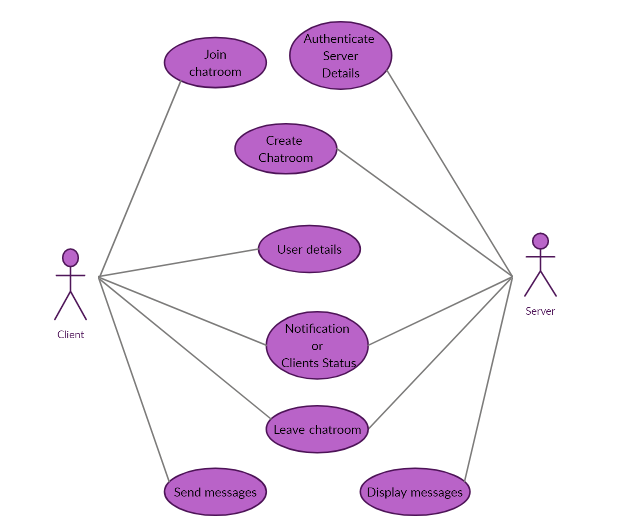
* **OS –** Windows 7 and above
* **Editor –** Any basic editor is preferable
* **C Compiler -** Linux

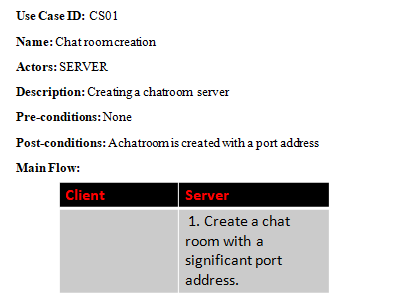
1. **HARDWARE REQUIREMENTS**

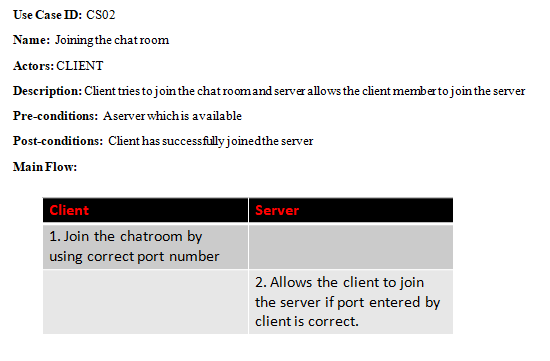
* **RAM –** 512 MB
* **INPUT DEVICES –** Keyboard
* **OUTPUT DEVICES –** Monitor Software

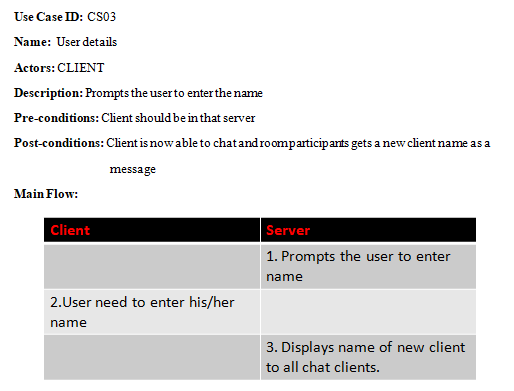
**Proposed work**

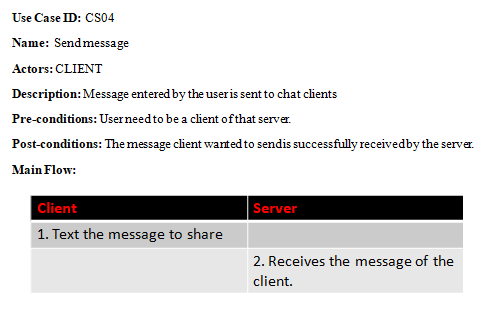
1. **Design**
2. **Use case diagram and descriptions for all the use-cases**

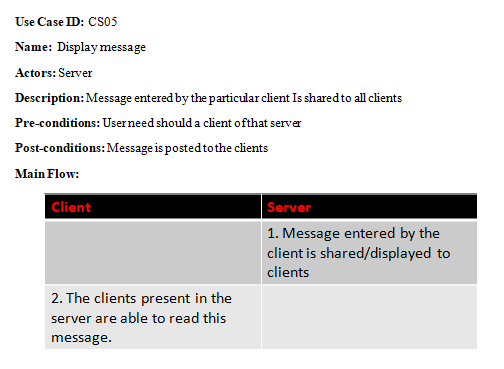


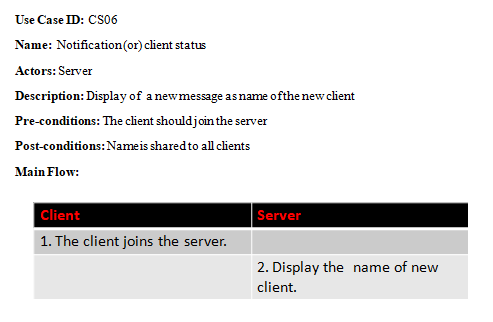


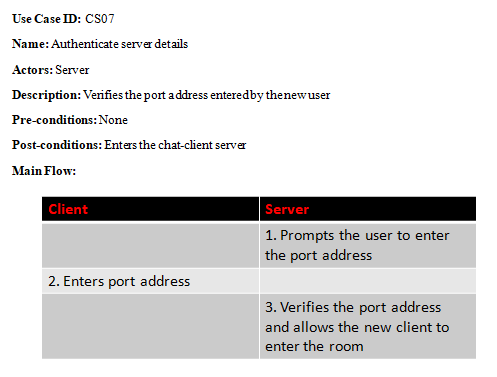


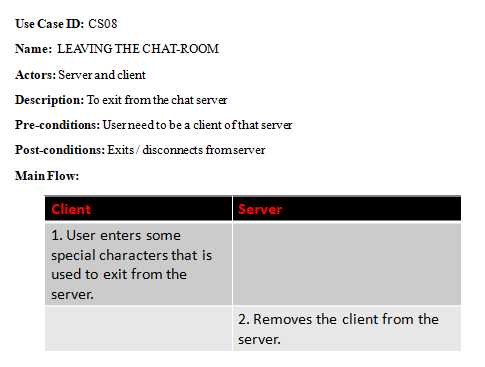












**b . IMPLEMENTATION**

**i . Module wise code for the entire project**

**FILE NAME (file number 1):** Client.c

**HEADER FILES:**

#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>

**MACROS:**

#define LENGTH 2048

#define MAX\_CLIENTS 10

**GLOBAL VARIABLES:**

/\* Global variables \*/

volatile sig\_atomic\_t flag = 0;

int sockfd = 0;

**FUNCTION NAME**  **:** str\_overwrite\_stdout ()

void str\_overwrite\_stdout(){

printf("%s", "-> ");

fflush(stdout);

}

**FUNCTION NAME** **:** str\_new\_line ()

void str\_new\_line(char \*arr, int length)

{

int i;

for(i=0;i<length;i++){

if(arr[i] == '\n'){

arr[i] = '\0';

break;

}

}

}

**FUNCTION NAME** **:** send\_msg\_handler ()

void send\_msg\_handler(){

char message[LENGTH] = {};

char buffer[LENGTH + 32] = {};

while(1) {

str\_overwrite\_stdout();

fgets(message, LENGTH, stdin);

str\_new\_line(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\_and\_exit(2);

}

**FUNCTION NAME**  **:** recv\_msg\_handler ()

void recv\_msg\_handler(){

char message[LENGTH] = {};

while(1) {

int recieve = recv(sockfd, message, LENGTH, 0); // Recieve message

if(recieve > 0)

{

printf("%s", message);

str\_overwrite\_stdout();

} else if (recieve == 0)

{

break;

}

bzero(message, LENGTH);

}

}

**FUNCTION NAME** **:** catch\_and\_exit ()

void catch\_and\_exit(int sig) {

flag = 1;

}

**FUNCTION NAME** **:** main()

int main(int argc, char \*\*argv)

{

if(argc != 2)

{

printf("Usage: ./Client <port number>\n");

return EXIT\_FAILURE;

}

char \*ip = "127.0.0.1"; // IP Address

int port = atoi(argv[1]);

signal(SIGINT, catch\_and\_exit);

printf("Please enter your name: ");

fgets(name, 32, stdin);

str\_new\_line(name, strlen(name));

if(strlen(name) > 32 || strlen(name) <2){

printf("Name must be more than 2 and less than 32 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 the Server \*/

int err = connect(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

if(err == -1) {

printf("ERROR: Connection Failed, Kindly check the port number you've entered\n");

return EXIT\_FAILURE;

}

/\* Send name \*/

send(sockfd, name, 32, 0);

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n === WELCOME TO CHAT CLIENT SERVER === \n");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\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("\nBye, I'm leaving.\n");

break;

}

}

close(sockfd);

return EXIT\_SUCCESS;

}

**FILE NAME (file number 2):** Server.c

**HEADER FILES :**

#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>

**MACROS :**

#define MAX\_CLIENTS 10

#define BUFFER\_SZ 2048

#define NAME\_LENGTH 32

**GLOBAL VARIABLES AND STRUCTURES:**

static int cli\_count = 0;

static int uid = 10;

/\* Client structure \*/

typedef struct{

struct sockaddr\_in address;

int sockfd;

int uid; // uid of the client

char name[NAME\_LENGTH]; // Name of the client

} client\_t;

client\_t \* clients[MAX\_CLIENTS];

pthread\_mutex\_t clients\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

**FUNCTIONS :**

**FUNCTION NAME**  **:** str\_overwrite\_stdout

**/\* Function for overwriting std\_out \*/**

void str\_overwrite\_stdout() {

printf("\r%s", "> ");

fflush(stdout);

}

**FUNCTION NAME**  **:** str\_new\_line

**/\* Function: If new line entered send message \*/**

void str\_new\_line(char\* arr, int length){

int i;

for(i=0;i<length;i++){

if(arr[i] == '\n'){

arr[i] = '\0';

break;

}

}

}

**FUNCTION NAME :** print\_client\_addr

**/\* Function to print client address \*/**

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);

}

**FUNCTION NAME :** queue\_add

**/\* Add clients to queue \*/**

void queue\_add(client\_t \*cl){

pthread\_mutex\_lock(&clients\_mutex);

int i=0;

for(i=0;i<MAX\_CLIENTS; ++i){

if(!clients[i]){

clients[i] = cl;

break;

}

}

pthread\_mutex\_unlock(&clients\_mutex);

}

**FUNCTION NAME :** queue\_remove

**/\* Remove clients from queue \*/**

void queue\_remove(int uid){

pthread\_mutex\_lock(&clients\_mutex);

int i=0;

for(i=0;i<MAX\_CLIENTS; ++i){

if(clients[i]){

if(clients[i]->uid == uid){

clients[i] = NULL;

break;

}

}

}

pthread\_mutex\_unlock(&clients\_mutex);

}

**FUNCTION NAME :** send\_message

**/\* Send message to all clients except sender \*/**

void send\_message(char \*s, int uid){

pthread\_mutex\_lock(&clients\_mutex);

int i=0;

for(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);

}

**FUNCTION NAME :** handle\_client

**/\* Handle all communication with the client \*/**

void \*handle\_client(void \*arg){

char buffer[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) >= NAME\_LENGTH-1){

printf("Please enter the name correctly.\n");

leave\_flag = 1;

} else {

strcpy(cli->name, name);

sprintf(buffer, "%s has joined\n", cli->name);

printf("%s", buffer);

send\_message(buffer, cli->uid);

}

bzero(buffer, BUFFER\_SZ);

while(1)

{

if(leave\_flag){

break;

}

int recieve = recv(cli->sockfd, buffer, BUFFER\_SZ, 0);

if(recieve > 0){

if(strlen(buffer) > 0){

send\_message(buffer, cli->uid);

str\_new\_line(buffer, strlen(buffer));

printf("%s\n", buffer);

}

} else if(recieve == 0 || strcmp(buffer, "exit") == 0){

sprintf(buffer, "%s has left\n", cli->name);

printf("%s", buffer);

send\_message(buffer, cli->uid);

leave\_flag = 1;

} else {

printf("ERROR: -1\n");

leave\_flag = 1;

}

bzero(buffer, 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;

}

**FUNCTION NAME :** main

int main(int argc, char \*\*argv)

{

if(argc != 2){

printf("Usage: ./Server <port\_number>\n");

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);

/\* Ignoring 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("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n === WELCOME TO CHAT CLIENT SERVER === \n");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\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("Sorry, 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); // Provided by unistd header file(#include<unistd.h>)

}

return EXIT\_SUCCESS;

}

1. **Any specific algorithms / logic to be highlighted in bold**

**EXAMPLE ALGORITHM FOR AUTHENTICATNG PORT NUMBER**

**STEP 1:** Start.

**STEP 2:** Read the port number through command line argument

**STEP 3:** Convert the value of command line to integer using **atoi()** function.

**STEP 4:** Check for connection using **signal()** function.

**STEP 5:** Socket created successfully if port number is valid otherwise user will be prompted with a message “**Invalid Details**, **Please enter valid port number”**.

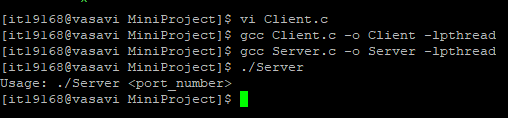
**STEP 6:** End.

1. **Github links**

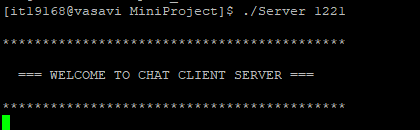
<https://github.com/ShitijReddy>

<https://github.com/bashaboinamanojkumar>

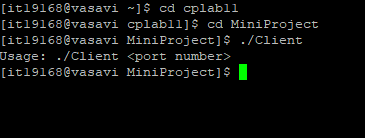
**C.TESTING**



|  |  |  |  |
| --- | --- | --- | --- |
| TEST CASE TEMPLATE | | | |
| **Test case ID** : TC01 | | | Use case ID :  **CS01** |
| **Test case title** : CHAT ROOM CREATION (Server) | | |
| **Test case description** : creating a chat room server with a port number. | | |
| **Test steps** | **Expected result** | **Actual result** | |
| User didn’t entered any port number. | A chat room is created successfully. | Message that port number is invalid. | |



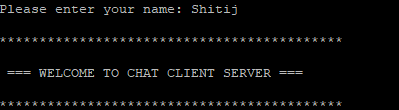
|  |  |  |  |
| --- | --- | --- | --- |
| TEST CASE TEMPLATE | | | |
| **Test case ID** : TC02 | | | Use case ID :  **CS01** |
| **Test case title** : CHATROOM CREATION (Server) | | |
| **Test case description** : User has to enter a valid port number | | |
| **Test steps** | **Expected result** | **Actual result** | |
| User enters a valid port number of 4 digits. | A chat room is successfully created. | A chat room is successfully created. | |



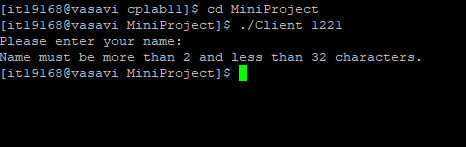
|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC03 | | | Use case ID :  **UC02** |
| **Test case title** : Join the chat room (Client) | | |
| **Test case description** : User has to enter port number created by server. | | |
| **Test steps** | **Expected result** | **Actual result** | |
| The system prompts the user to enter port number and join the chat room. | Client has successfully joined the chat room. | A message saying wrong port is displayed on the screen. | |



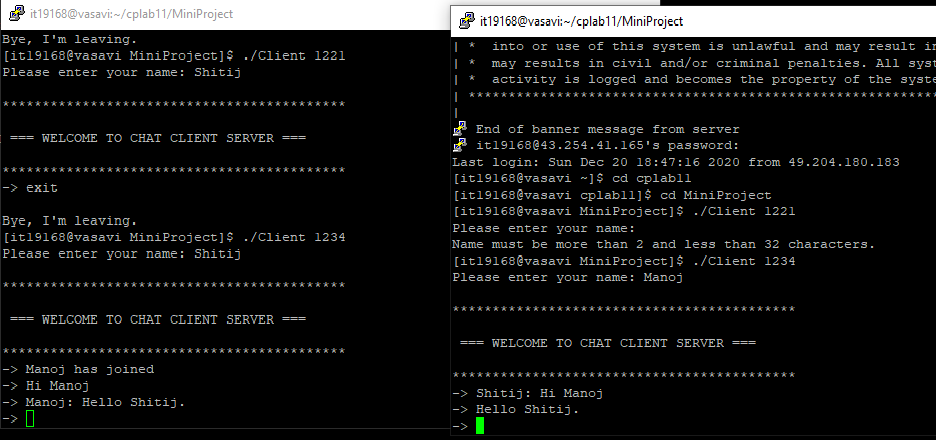
|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC04 | | | Use case ID :  **UC02** |
| **Test case title** : Join (Client) | | |
| **Test case description**: User has to correct and existing port number. | | |
| **Test steps** | **Expected result** | **Actual result** | |
| The system prompts the user to enter port number and join the chat room. | Client has successfully joined the chat room. | Client has successfully joined the chat room. | |
| The system prompts the user to enter password and user name correctly | A message saying login is successful is displayed on the screen and redirects to users home page. | A message saying login is successful is displayed on the screen and redirects to users home page. | |



|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC05 | | | Use case ID :  **UC03** |
| **Test case title** : User Details | | |
| **Test case description** : User has to enter his details such as name | | |
| **Test steps** | **Expected result** | **Actual result** | |
| System prompts the user to enter his name and he enters. | Name is successfully entered and displays name of new client | Name is successfully entered and displays name of new client | |



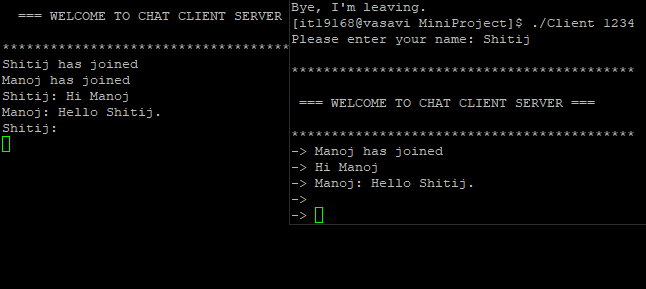
|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC06 | | | Use case ID :  **UC03** |
| **Test case title** : User Details | | |
| **Test case description** : User has to enter his details such as name | | |
| **Test steps** | **Expected result** | **Actual result** | |
| User enters invalid name and it shows an error message. | Message saying invalid name. | Message saying invalid name. | |



|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC07 | | | Use case ID :  **UC04** |
| **Test case title** : Send message | | |
| **Test case description** : Message entered by the user is sent to clients. | | |
| **Test steps** | **Expected result** | **Actual result** | |
| Text the message to share to the clients. | Message is sent successfully to all the clients. | Message is sent successfully to all the clients. | |



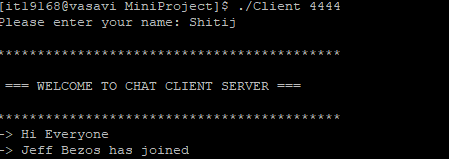
|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC08 | | | Use case ID :  **UC04** |
| **Test case title** : Send message | | |
| **Test case description** : Send message to all clients | | |
| **Test steps** | **Expected result** | **Actual result** | |
| User enters the empty message. | Empty message is sent to all clients. | Empty message is sent to all clients. | |



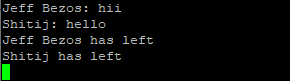
|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC09 | | | Use case ID :  **UC05** |
| **Test case title** : Display message | | |
| **Test case description** : Message entered by client is shared to all clients. | | |
| **Test steps** | **Expected result** | **Actual result** | |
| Message entered by the client is shared/ displayed to clients.  The clients present in the server are able to read this message. | Message displayed on the screen. | Message displayed on the screen. | |



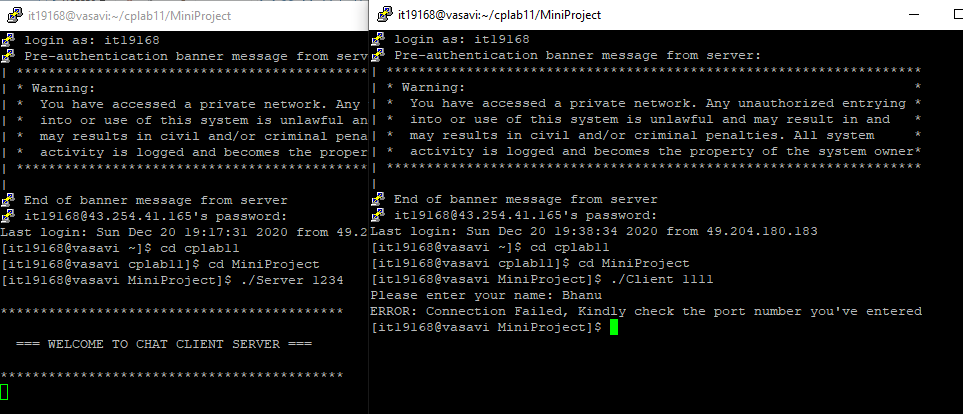
|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC10 | | | Use case ID :  **UC05** |
| **Test case title** : Display empty message | | |
| **Test case description** : Message entered by client is shared to all clients | | |
| **Test steps** | **Expected result** | **Actual result** | |
| Empty message is displayed on the server screen and every client is able to see the message. | Empty message is displayed to all the clients. | Empty message is displayed to all the clients. | |



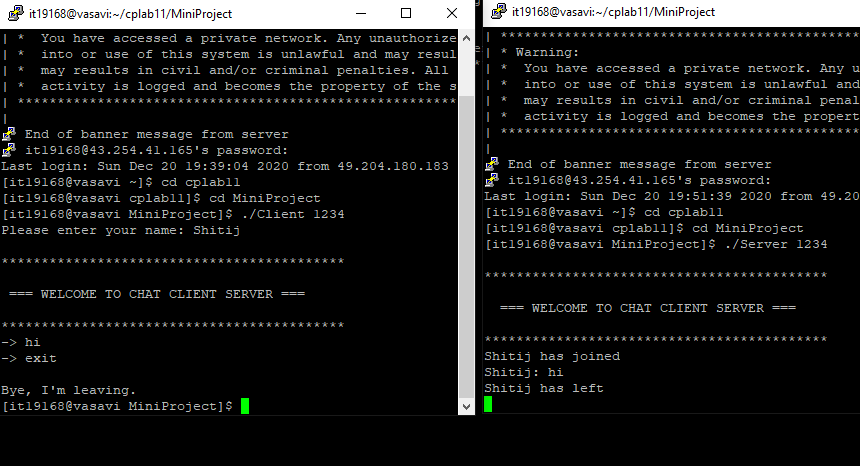
|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC11 | | | Use case ID :  **UC06** |
| **Test case title** : Notifications/ Client Status (Joining) | | |
| **Test case description** : Client status is displayed on the screen | | |
| **Test steps** | **Expected result** | **Actual result** | |
| Displays a message that this client has joined | A message is displayed that he joined the chat client server. | A message is displayed that he joined the chat client server. | |



|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC12 | | | Use case ID :  **UC06** |
| **Test case title** : Notifications / Client status ( leaving) | | |
| **Test case description** : Client status is displayed on the screen | | |
| **Test steps** | **Expected result** | **Actual result** | |
| Displays a message that this client has left. | A message is displayed that he left the chat client server. | A message is displayed that he left the chat client server. | |

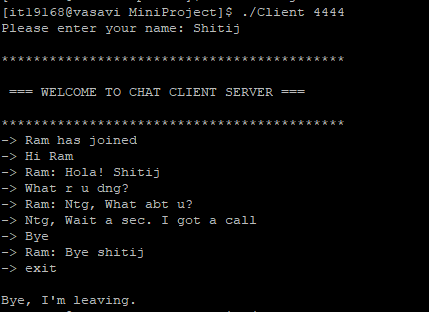


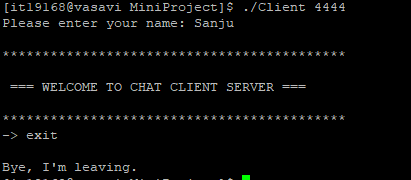
|  |  |  |  |
| --- | --- | --- | --- |
| TEST CASE TEMPLATE | | | |
| **Test case ID** : TC13 | | | Use case ID :  **UC07** |
| **Test case title** : Authenticate server details. | | |
| **Test case description** : Verifies the port address entered by the new user. | | |
| **Test steps** | **Expected result** | **Actual result** | |
| 1. Prompts the user to enter the port address.  2. Enters the port address.  3. Verifies the port address and allows the new client to enter the room. | An error message saying no such chat room. | An error message saying no such chat room. | |



|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** : TC14 | | | Use case ID :  **UC08** |
| **Test case title** : Leave the chat room | | |
| **Test case description** : User wants to exit from the server. | | |
| **Test steps** | **Expected result** | **Actual result** | |
| User enters a particular message used to quit from the chat room. | Client has successfully left the chat client server. | Client has successfully left the chat client server. | |

**RESULT**

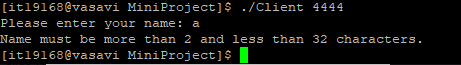












These are the test cases of our project.

**What is the additional knowledge gained as a result of implementing this mini project apart from the syllabus covered in the course programming for problem solving?**

I have learnt socket programming and threads especially pthreads. Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

**Socket creation:**

int sockfd = socket(domain,type,protocol)

**sockfd:** socket descriptor, an integer (like a file-handle)  
**domain:** integer, communication domain e.g., AF\_INET (IPv4 protocol) , AF\_INET6 (IPv6 protocol)  
**type:** communication type  
SOCK\_STREAM: TCP(reliable, connection oriented)  
SOCK\_DGRAM: UDP(unreliable, connectionless)  
**protocol:**Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

In the process of research I came to know about the latest technologies and different algorithms.

**Conclusion and Future work**

The main objective of the project is to develop a Secure Chat Application. I had taken a wide range of literature review in order to achieve all the tasks, where I came to know about some of the products that are existing in the market. I made a detailed research in that path to cover the loop holes that existing systems are facing and to eradicate them in our application.

The purpose of the CLIENT AND SERVER are achieved by providing a console-based user interface, this client and server model is highly flexible for the user to operate. The project is provided a client and server for users to chat with each other conveniently.

Apart from these features, we are interested in adding a few features such as clients can send audio and videos to the server and other clients. And even to make to a user-interactive model using java and python.

**References**

<https://www.geeksforgeeks.org/socket-programming-cc/>

<https://stackoverflow.com/>