MULTIPLE CLIENT SERVER CHAT APPLICATION

# Project Scope:

This project is a socket programming, threads, and synchronization. You are supposed to implement a rudimentary chatting application. The application will run on a client/server model which means that you’ll build two different programs: a server and a client. For simplicity we’d say that one instance of the server will be running and multiple clients can connect to the server and chat to each other using this server program as an intermediary. The server and clients may run on the same PC or on different ones.

# Functionality:

The aim of this project is to develop a reliable and secure network programming (Client-Server chat model) which can perform a multithreaded server client chat application based on C socket programming using Transport Control Protocol (TCP).

Server

Listen on port:8888

IP: 172.16.24.252

Connect

to:172.16.24.252

Connect

to:172.16.24.252

|  |  |  |
| --- | --- | --- |
| Client-1 | …… | Client-N |

# TO DO:

* ­ You basically run the server passing it a port number:

**./anyport**

* The server program creates a socket and starts listening on the port passed to it at command line. ­ You run a client passing it the server programs computer IP, the port it’s listening on, and a name for your client:
* **./client** (public ip of server) anyport client1
* The   server   program accept   connections   from   multiple   clients. For each connection established the server store the client’s identifier and keep a record of which port is being used for communication with this client.
* In this applications, a sever creates a separate thread for each connection with the client.
* If a new connection request comes from a a client with a client identifier which is already in use, the server inform the client with an appropriate error message and close that connection
* The client can send “commands” to the server. Each command starts with a ‘/’ character. When the server program receives a command from a client, it take the appropriate action and send a response to the client. Few commands:

**/list-**if the client sends a “/list” command to the server, it means the client is asking for a list of all the connected clients. Upon receiving this command, the server should put the names of all the clients connected to itself in  string and send it back to the client. The client shall receive this string and display all these names to the user.

**/msg –** users can type this command to send messages to other clients. The general syntax is “/msg clientname message”. i.e. if the user at client­1 types “/msg client­2 hi there”, the server receive this message from client­1 and send it to client­2. The user at client­2 should receive the message “hi there” along with the information that it came from client­1.

Proper error handling for this command at server and client end i.e. missing destination (client) identifier or incorrect destination identifier generate appropriate response.

**/quit  –** users can type this command to disconnect from the server and quit the client. Upon reception of this command from a client, the server closes the connection with that client and terminate that particular thread.

Source Code:

Server.c

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <pthread.h>

#define MAX\_CLIENTS 100 //The max clients the server can entertain

struct client\_info

{

int sockno;

char ip[INET\_ADDRSTRLEN];

};

//GLOBAL Variables

int client\_sockets[MAX\_CLIENTS];//stores client sockets

char client\_ids[MAX\_CLIENTS][MAX\_CLIENTS];//for storing names of clients

int n = 0;//stores number of active clients to the server

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;//Mutex lock for pthreads

int find\_client\_by\_id(char\* name) //returns -1 if specified client not connected, index in id/sock array otherwise

{

for (int i = 0; i < n; i++)

{

if (strcmp(client\_ids[i], name) == 0)

{

return i;

}

}

return -1;

}

int find\_client\_by\_sock(int socket) //returns -1 if specified client not connected, index in id/sock array otherwise

{

for (int i = 0; i < n; i++)

{

if (client\_sockets[i] == socket)

{

return i;

}

}

return -1;

}

//helper for sending and receiving messages between threads

void sendtoallorone(char \*msg\_complete, int curr, int self) //self=0 means will send to all,1 means will send only to passed socket at curr

{

int i;

pthread\_mutex\_lock(&mutex);

for (i = 0; i < n; i++)

{

if (self == 0)

{

if (client\_sockets[i] != curr)

{

if (send(client\_sockets[i], msg\_complete, strlen(msg\_complete), 0) < 0) {

perror("sending failure");

continue;

}

}

}

else

{

if (client\_sockets[i] == curr)

{

if (send(client\_sockets[i], msg\_complete, strlen(msg\_complete), 0) < 0)

{

perror("sending failure");

continue;

}

}

}

}

pthread\_mutex\_unlock(&mutex);

}

//helper for receiving messages, handles actions as well

void \*recvmg(void \*sock)

{

struct client\_info cl = \*((struct client\_info \*)sock);

char msg\_complete[500];

char msg\_text[500];

char broadcast\_text[500];

int len;

int i;

int j;

int is\_id\_set = 0;

while ((len = recv(cl.sockno, msg\_complete, 500, 0)) > 0)

{

msg\_complete[len] = '\0';

broadcast\_text[len] = '\0';

printf("\nServer transmitting message(s)...\n");

if (strncmp(msg\_complete, "\*", 1) == 0 && is\_id\_set == 0) //Set the id of the lcient in the array to the username sent by the client, identified by the \* character

{

is\_id\_set = 1;

strcpy(client\_ids[n - 1], &(msg\_complete[1]));

int index = find\_client\_by\_id(&(msg\_complete[1]));

if (index != n - 1) //indicates that a duplicate named client is already connected to the server

{

//Duplicate client id, shutdown Client

sendtoallorone("/shutdown", cl.sockno, 1); //Tell client to shut down

break;

}

}

if (strncmp(msg\_complete, "/list", 5) == 0)//sends list of connected clients to the client requesting

{

char temp[100];

snprintf(temp, 100, "\nCONNECTED CLIENTS:\n");

strcat(broadcast\_text, temp);

for (int i = 0; i < n; i++)

{

snprintf(temp, 100, "%d%s%s\n", i + 1, ":", client\_ids[i]);

strcat(broadcast\_text, temp);

}

sendtoallorone(broadcast\_text, cl.sockno, 1);

}

if (strncmp(msg\_complete, "/quit", 5) == 0)//breaks the loop, closes the socket and thread exits.

{

printf("Dropping from server\n");

break;

}

if (strncmp(msg\_complete, "/cmds", 5) == 0)//shows available cmds to the requesting client

{

strcat(broadcast\_text, "\nAVAILABLE CMDS:\n/list - Display active client list\n/msg clientid message - Send message to user at client-id\n/all message - Send message to all active clients\n/cmds - View commands\n/quit - Disconnect and exit from the server");

sendtoallorone(broadcast\_text, cl.sockno, 1);

}

if (strncmp(msg\_complete, "/msg", 4) == 0)

{

//temporary strings

char temp[100];

char id[100];

char message[100];

if (strlen(msg\_complete) >= 6)

{

//slicing received msg to id + msg

strcpy(temp, &(msg\_complete[5]));

strcat(broadcast\_text, temp);

//splitting input to the two id and message strings

sscanf(broadcast\_text, "%s %[^\r\n]", id, message);

if (strlen(id) == 0 || strlen(message) == 0)

{

snprintf(temp, 100, "\nIncorrect Arguments to /msg\n");

strcpy(broadcast\_text, temp);

sendtoallorone(broadcast\_text, cl.sockno, 1);

}

else//PROPER FORMATTED ID AND MESSAGE RECEIVED

{

int socket = find\_client\_by\_id(id);

if (socket == -1)//INVALID CLIENT

{

snprintf(temp, 100, "\nSpecified user not connected to server\n");

strcat(broadcast\_text, temp);

sendtoallorone(broadcast\_text, cl.sockno, 1);

}

else//VALID CLIENT + MESSAGE, SEND

{

socket = client\_sockets[socket];

//add sender id at start of message, then deliver

int index = find\_client\_by\_sock(cl.sockno);

char tmp[500];

strcpy(tmp, client\_ids[index]);

strcat(tmp, ": ");

strcat(tmp, message);

sendtoallorone(tmp, socket, 1);

}

}

strcpy(message, "");

strcpy(id, "");

}

else

{

snprintf(temp, 100, "\nIncorrect Arguments to /msg\n");

strcat(broadcast\_text, temp);

}

}

if (strncmp(msg\_complete, "/all", 4) == 0)//EXTRA TEST CMD, send a message to all connected clients

{

char tmp[500];

if (strlen(msg\_complete) >= 6)

{

int index = find\_client\_by\_sock(cl.sockno);

strcpy(tmp, client\_ids[index]);

strcat(tmp, ": ");

strcat(tmp, &msg\_complete[5]);

sendtoallorone(tmp, cl.sockno, 0);

}

else

{

snprintf(tmp, 100, "\nInvalid arguments to /all\n");

sendtoallorone(tmp, cl.sockno, 1);

}

}

//clean buffer

memset(msg\_complete, '\0', sizeof(msg\_complete));

memset(broadcast\_text, '\0', sizeof(broadcast\_text));

}

pthread\_mutex\_lock(&mutex);

//A client has disconnected, update the two arrays

printf("%s disconnected\n", cl.ip);

//A client has dropped, we remove its entries from the two main arrays (socket and id)

for (i = 0; i < n; i++)

{

if (client\_sockets[i] == cl.sockno)

{

j = i;

while (j < n - 1)

{

client\_sockets[j] = client\_sockets[j + 1];

strcpy(client\_ids[j], client\_ids[j + 1]);

j++;

}

}

}

n--;

pthread\_mutex\_unlock(&mutex);

}

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

{

for (int i = 0; i < 100; i++)

strcpy(client\_ids[i], "");//clean the array

//Initializing socket communication system

struct sockaddr\_in my\_addr, their\_addr;

int my\_sock;

int their\_sock;

socklen\_t their\_addr\_size;

int portno;

pthread\_t sendt, recvt;

char msg\_complete[500];

int len;

struct client\_info cl;

char ip[INET\_ADDRSTRLEN];

if (argc != 2)

{

printf("Usage: ./server port\n");

exit(1);

}

portno = atoi(argv[1]);

my\_sock = socket(AF\_INET, SOCK\_STREAM, 0);

memset(my\_addr.sin\_zero, '\0', sizeof(my\_addr.sin\_zero));

my\_addr.sin\_family = AF\_INET;

my\_addr.sin\_port = htons(portno);

my\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

their\_addr\_size = sizeof(their\_addr);

//Binding and starting to listen to the passed port

if (bind(my\_sock, (struct sockaddr \*)&my\_addr, sizeof(my\_addr)) != 0)

{

perror("binding unsuccessful");

exit(1);

}

if (listen(my\_sock, 5) != 0) {

perror("listening unsuccessful");

exit(1);

}

printf("\nServer up and ready to transmit...\n");

while (1)

{

//If a client connects to our socket, we create a server thread for it for managing ipc and update server information

if ((their\_sock = accept(my\_sock, (struct sockaddr \*)&their\_addr, &their\_addr\_size)) < 0)

{

perror("accept unsuccessful");

exit(1);

}

pthread\_mutex\_lock(&mutex);

inet\_ntop(AF\_INET, (struct sockaddr \*)&their\_addr, ip, INET\_ADDRSTRLEN);

printf("%s connected\n", ip);

cl.sockno = their\_sock;

strcpy(cl.ip, ip);

client\_sockets[n] = their\_sock;

n++;

pthread\_create(&recvt, NULL, recvmg, &cl);

pthread\_mutex\_unlock(&mutex);

}

return 0;

}

Client.c

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <pthread.h>

void \*recvmg(void \*sock)

{

int their\_sock = \*((int \*)sock);

char msg[500];

int len;

while ((len = recv(their\_sock, msg, 500, 0)) > 0) {

msg[len] = '\0';

//If duplicate client id, server sends shutdown cmd, terminate client

if (strncmp(msg, "/shutdown", 9) == 0)

{

printf("\nDuplicate Client ID, Exiting...\n");

exit(0);

}

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

memset(msg, '\0', sizeof(msg));

}

}

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

{

struct sockaddr\_in their\_addr;

int my\_sock;

int their\_sock;

int their\_addr\_size;

int portno;

pthread\_t sendt, recvt;

char msg[500];

char username[100];

username[0] = '\*'; //for establishing uniqueness

char res[600];

char ip[INET\_ADDRSTRLEN];

int len;

if (argc != 4) {

printf("Usage: ip port client-username\n");

exit(1);

}

portno = atoi(argv[2]);

//Concatenating username with '\*' character

for (int i = 0; i < strlen(argv[3]); i++)

{

username[i + 1] = (argv[3][i]);

}

username[strlen(argv[3]) + 1] = '\0';

my\_sock = socket(AF\_INET, SOCK\_STREAM, 0);

memset(their\_addr.sin\_zero, '\0', sizeof(their\_addr.sin\_zero));

their\_addr.sin\_family = AF\_INET;

their\_addr.sin\_port = htons(portno);

//Initiliazing socket communication system

if (inet\_pton(AF\_INET, argv[1], &their\_addr.sin\_addr) < 0)

{

perror("Client Error: IP not initialized succesfully");

exit(1);

}

if (connect(my\_sock, (struct sockaddr \*)&their\_addr, sizeof(their\_addr)) < 0) {

perror("connection not esatablished");

exit(1);

}

inet\_ntop(AF\_INET, (struct sockaddr \*)&their\_addr, ip, INET\_ADDRSTRLEN);

printf("Connected to server at %s, type /cmds to view commands.\n", ip);

pthread\_create(&recvt, NULL, recvmg, &my\_sock);

//transmitting username to server

if ( write(my\_sock, username, strlen(username)) < 0 )

{

perror("message not sent");

exit(1);

}

memset(msg, '\0', sizeof(msg));

memset(res, '\0', sizeof(res));

//block for sending messages to server

while (fgets(msg, 500, stdin) > 0)

{

strcpy(res, msg);

len = write(my\_sock, res, strlen(res));

if (len < 0) {

perror("message not sent");

exit(1);

}

if (strncmp(res, "/quit", 5) == 0)

{

close(my\_sock);

return 0;

}

memset(msg, '\0', sizeof(msg));

memset(res, '\0', sizeof(res));

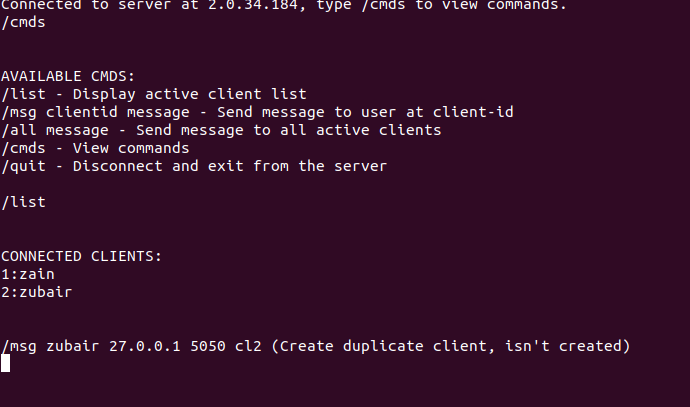
}

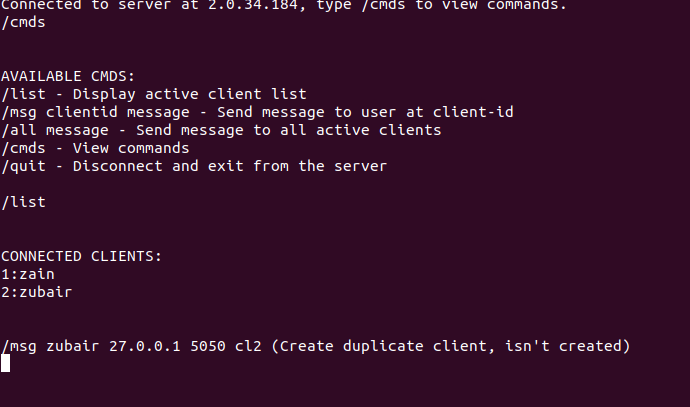
pthread\_join(recvt, NULL);

close(my\_sock);

}

Screen shot





OS PROJECT ‘21

Group Members:

Muhammad Zubair

Muhammad Zain Ul Abideen

Sample usage through terminal:

(Terminal 1 ->) make

(Terminal 1 ->) ./se 5050 (Start server)

(Terminal 2 ->) ./cl 172.16.24.252 8888 zain (Create unique client)

(Terminal 3 ->) ./cl 172.16.24.252 8888 zubair (Create unique client)

(Terminal 4 ->) ./cl 172.16.24.252 8888 ali (Create duplicate client, isn't created)

(Terminal 3 ->) /list (Displays list of clients)

(Terminal 2 ->) /msg zain hello (Send message to inexistent client from zain)

(Terminal 2 ->) /msg zubair hello (Send message to zubair from zain)

(Terminal 3 ->) /quit (Exit zain)

(Terminal 2 ->) /list (Check updated list from server)

## TEAM MEMBER:

Muhammad Zubair

Muhammad Zain Ul Abideen