CHAT APPLICATION USING SOCKET IN C LANGUAGE

A COURSE PROJECT REPORT

By
PALAASH SURANA(RA2111003010319)
KAMYA OJHA(RA2111003010343)

Under the guidance of

DR. R.S. PONMAGAL

In partial fulfilment for the Course

of

18CSC302J - COMPUTER NETWORKS

in CTECH

FACULTY OF ENGINEERING AND

TECHNOLOGY SRM INSTITUTE OF SCIENCE

AND TECHNOLOGY

Kattankulathur, Chenpalpattu District

OCTOBER 2023

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
SRM NAGAR, KATTANKULATHUR- 603203,
CHENGALPATTU DISTRICT

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this mini project report " CHAT APPLICATION USING SOCKET IN C LANGUAGE" **is the bonafide work of Palaash Surana(RA2111003010319) kamya Ojha(RA2111003010343) who carried out the project work under my supervision.**

Date: 31/10/2023

SIGNATURE

Dr.R.S.Ponmagal
Associate Professor
Department of Computing TechnologiesSRM IST,
Kattankulathur

TABLE OF CONTENTS

CHAPTERS CONTENTS

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable Vice Chancellor **Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors. We would like to express my warmth of gratitude to our Registrar **Dr. S. Ponnusamy**, for his encouragement.

We express our profound gratitude to our Dean (College of Engineering and Technology) **Dr. T. V.Gopal**, for bringing out novelty in all executions. We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman**, for imparting confidence to complete my course project We wish to express my sincere thanks to Course Audit Professor **Dr.Annapurani Panaiyappan**, Professor and Head, Department of Networking and Communications and Course Coordinators for their constant encouragement and support.

We are highly thankful to our my Course project Faculty **Dr.R.S.Ponmagal** , Associate Professor , Department of Computing Technologies, for his/her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our HoD **Dr.M.Pushpalatha**, Head of the Department, Department of Computing Techonogies and my Departmental colleagues for their Support. Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

ABSTRACT

In the ever-changing landscape of digital communication, the Chat Application using TCP Server-Client implementation in C emerges as a fusion of classic networking principles and contemporary user needs.

This project is a testament to the essential role of instant messaging in our daily lives and the underlying complexities of ensuring reliable and secure data transmission over a network. The choice of the C programming language not only provides the necessary control and efficiency for networking applications but also underscores the project's commitment to a robust foundation. By adopting the TCP protocol, the application ensures not only the accurate delivery of messages but also maintains the chronological order of communication, enriching the user experience.

This project recognizes the omnipresence of instant messaging in modern life and seeks to provide a versatile and extensible solution. The utilization of TCP adds a layer of reliability that is paramount in ensuring that messages traverse the network intact and in the intended order. The introduction of this Chat Application not only caters to the present demand for seamless communication but also lays the groundwork for future enhancements and innovations in the realm of networked applications.

INTRODUCTION

In the vast realm of computer networks, the client-server architecture has proven to be a stalwart foundation for numerous applications, and this project contributes to this legacy with a focused lens on real-time chat. As we navigate the digital age, the need for instantaneous communication has become non-negotiable, and this project addresses this need with a nuanced understanding of both the timeless principles and contemporary nuances of networking.

The choice of the C programming language is deliberate, offering a level of control and efficiency crucial for networking applications. The TCP protocol is strategically employed to establish a dependable channel for communication. This protocol not only aligns with the reliability required for messaging applications but also adheres to the connection-oriented nature essential for maintaining the integrity of real-time conversations.

This project acknowledges the ubiquity of instant messaging in modern life and aims to provide a solution that is not only functional but also adaptable to the evolving demands of users. The introduction of this Chat Application signifies a commitment to not just meeting present expectations for seamless communication but also laying the groundwork for future enhancements and innovations in the realm of networked applications. Through this exploration, the project embarks on a journey to encapsulate the essence of connectivity in an ever-expanding digital universe.

REQUIREMENTS:

Programming Language: C

Networking Library: Socket Programming

Development Environment: Code::Blocks, GCC Compiler

Operating System: Cross-platform compatibility (Linux, Windows)

User Interface: Simple command-line interface for initial implementation

IMPLEMENTATION

The intricate implementation of the Chat Application, realized through TCP Server-Client architecture in C, unveils a seamless dance of coding intricacies and user-centric design principles. At the server's core lies a carefully crafted symphony of socket programming and multithreading, laying the groundwork for a robust and responsive communication hub.

The server component kicks off with the initialization of a socket, its binding to a specific port, and the attentive stance of listening for incoming connections. Upon a client's connection, the server gracefully spins off a dedicated thread, ensuring that multiple users can engage concurrently without the worry of collisions or performance bottlenecks.

Message handling, a pivotal aspect of the server-side implementation, is a meticulous dance of parsing and processing. Messages flow in from clients, each undergoing scrutiny to guarantee proper formatting and sequence maintenance. The commitment to preserving the order of messages contributes significantly to the cohesive and intuitive flow of conversation, enhancing the overall user experience.

On the client side, the intricacies continue to unfold. The initiation of a socket and subsequent connection to the server's designated port herald the user into a straightforward command-line interface. This interface acts as a canvas for interaction, providing an intuitive space for users to engage in real-time conversations. The messaging process seamlessly juggles between sending and receiving messages, with the client-side logic acting as a vigilant conductor, ensuring the harmonious transmission of data.

Noteworthy in the client-side implementation is the integration of robust error handling mechanisms. This foresight equips the client to gracefully navigate unexpected scenarios, be it a connection hiccup or a glitch in message transmission. The result is an

application that exudes reliability, demonstrating resilience even in the face of unforeseen challenges.

The chosen communication protocol, a simple yet effective text-based approach, underscores the project's commitment to functionality without unnecessary complexity. Messages are structured with sender identity and content, offering a streamlined processing logic. While this approach suffices for the current scope, it also hints at the project's adaptability, providing a fertile ground for future enhancements such as multimedia message support or encrypted communication.

In conclusion, the implementation journey of the Chat Application stands as a testament to the seamless integration of networking intricacies and user-friendly design. The server and client components collaboratively craft an environment where communication is not just a transaction but an experience. As the project evolves, this implementation lays down the tracks for further innovation, illustrating that in the realm of networked applications, the journey is as enriching as the destination.

Server Side:

Socket Initialization: Establishing a socket for communication.
Binding and Listening: Assigning a port to the socket and listening for incoming connections.
Accepting Connections: Accepting client connections and creating individual threads for each client.
Message Handling: Managing the exchange of messages between clients.

Client Side:

Socket Initialization: Creating a socket for communication.
Connection Establishment: Connecting to the server.

User Interface: Designing a simple command-line interface for user interaction.
Message Exchange: Sending and receiving messages to and from the server.

Communication Protocol:

Using a simple text-based protocol for message exchange.
Implementing error handling mechanisms to ensure data integrity.

CODE

SERVER SIDE:

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.


void func(int connfd)
{
                                                    char buff[MAX];
                                                    int n;
                                                    // infinite loop for chat
                                                    for (;;) {
                                                      bzero(buff, MAX);

                                                      // read the message from
client and copy it in buffer
                                                      read(connfd, buff,
sizeof(buff));
                                                      // print buffer which contains
the client contents
                                                      printf("From client: %s\t To
client : ", buff);
                                                      bzero(buff, MAX);
                                                      n = 0;
                                                      // copy server message in the
buffer
                                                      while ((buff[n++] =
getchar()) != '\n')
                                                              ;
```

```
                                        // and send that
  buffer to client
  write(connfd, buff,
sizeof(buff));

  // if msg contains "Exit" then
  server exit and chat ended.
  if (strncmp("exit", buff, 4)
== 0) {
        printf("Server
Exit...\n");
        break;
  }
}

}

// Driver function

int main()
{
int sockfd, connfd, len;
struct sockaddr_in servaddr,
cli;

// socket create and
verification
sockfd = socket(AF_INET,
SOCK_STREAM, 0);
if (sockfd == -1) {
  printf("socket creation
failed...\n");
  exit(0);
}
else
  printf("Socket successfully
created..\n");
bzero(&servaddr,
sizeof(servaddr));

// assign IP, PORT
servaddr.sin_family =
AF_INET;
```

```
servaddr.sin_addr.s_addr =
htonl(INADDR_ANY);
servaddr.sin_port =
htons(PORT);

// Binding newly created
socket to given IP and verification
if ((bind(sockfd,
(SA*)&servaddr, sizeof(servaddr))) != 0) {
  printf("socket bind
failed...\n");
  exit(0);
}
else
  printf("Socket successfully
binded..\n");

// Now server is ready to listen
and verification
if ((listen(sockfd, 5)) != 0) {
  printf("Listen failed...\n");
  exit(0);
}
else
  printf("Server listening..\n");
len = sizeof(cli);

// Accept the data packet from
client and verification
connfd = accept(sockfd,
(SA*)&cli, &len);
if (connfd < 0) {
  printf("server accept
failed...\n");
  exit(0);
}
else
  printf("server accept the
client...\n");

// Function for chatting
between client and server
func(connfd);
```

```
                                                    // After chatting close the
socket
}                                                   close(sockfd);
CLIENT SIDE:

#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr


void func(int sockfd)
{
                                                    char buff[MAX];
                                                    int n;
                                                    for (;;) {
                                                      bzero(buff, sizeof(buff));
                                                      printf("Enter the string : ");
                                                      n = 0;
                                                      while ((buff[n++] =

getchar()) != '\n')
                                                              ;
                                                      write(sockfd, buff,

sizeof(buff));
                                                      bzero(buff, sizeof(buff));
                                                      read(sockfd, buff,

sizeof(buff));
                                                      printf("From Server : %s",

buff);
                                                      if ((strncmp(buff, "exit", 4))

== 0) {
                                                              printf("Client

Exit...\n");
                                                              break;
                                                      }
```

```
}

int main()
{

cli;

verification

SOCK_STREAM, 0);

failed...\n");

created..\n");

sizeof(servaddr));

AF_INET;

inet_addr("127.0.0.1");

htons(PORT);

server socket

(SA*)&servaddr, sizeof(servaddr))

server failed...\n");
}

int sockfd, connfd;
struct sockaddr_in servaddr,

// socket create and

sockfd = socket(AF_INET,

if (sockfd == -1) {
  printf("socket creation

  exit(0);
}
else
  printf("Socket successfully

bzero(&servaddr,

// assign IP, PORT
servaddr.sin_family =

servaddr.sin_addr.s_addr =

servaddr.sin_port =

// connect the client socket to

if (connect(sockfd,

  != 0) {
  printf("connection with the

  exit(0);
}
else
```

```
                                        printf("connected to the
server..\n");

                                        // function for chat
                                        func(sockfd);

                                        // close the socket
                                        close(sockfd);
}
```

OUTPUT

Server Side:

```
~$ gcc server.c
~$ ./a.out
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...
From client: hello this is client
         To client : hey there how are you?
From client: I am good
         To client : bye
From client: bye
         To client :
```

Client Side:

```
~$ gcc client.c
~$ ./a.out
Socket successfully created..
connected to the server..
Enter the string : hello this is client
From Server : hey there how are you?
Enter the string : I am good
From Server : bye
Enter the string : bye
```

CONCLUSION:

In the culmination of the Chat Application project utilizing TCP Server-Client implementation in C, it becomes evident that the convergence of classic networking principles and contemporary user needs results in a robust and versatile solution. This endeavor has not merely scratched the surface of real-time communication but has delved into the intricacies of data transmission, network reliability, and user interaction, showcasing the depth and breadth of its contributions.

The implementation of a client-server architecture, rooted in the C programming language, has allowed for meticulous control over networking intricacies, providing a solid foundation for the application. The selection of the TCP protocol, with its emphasis on reliability and ordered communication, has proven to be pivotal. The ability to guarantee message delivery and maintain conversational order is fundamental in ensuring a seamless and intuitive user experience.

As we draw conclusions from this project, it is essential to underscore its significance in the context of modern communication paradigms. Instant messaging has become a staple in our interconnected world, and the Chat Application not only meets but anticipates the evolving demands of users. The journey through the development process has not been just about code and protocols; it has been a voyage into understanding the dynamics of human interaction in the digital realm.

Future enhancements to this Chat Application could include the integration of a graphical user interface for a more engaging user

experience, support for multimedia messages, and the implementation of robust security measures to ensure the confidentiality of conversations. These potential additions highlight the extensibility and adaptability of the project, positioning it as a foundation for continuous innovation in the dynamic landscape of networked applications.

In essence, the Chat Application project stands not only as a functional tool for communication but as a testament to the synergy between technological prowess and human-centric design. It symbolizes the ever-expanding possibilities within the realm of networked applications, encouraging further exploration and innovation. As we bid adieu to this project, it is with the realization that it is not merely the end but a stepping stone towards a future where communication knows no bounds, and connectivity becomes an ever-enriching tapestry of shared experiences.

REFERENCES

Stevens, W. R., Fenner, B., & Rudoff, A. M. (2004). UNIX Network Programming, Volume 1: The Sockets Networking API.
Forouzan, B. A. (2006). TCP/IP Protocol Suite.
These references served as valuable resources in understanding the principles of socket programming and TCP/IP protocols during the development of this project.