

**TRIBHUVAN UNIVERSITY**

**INSTITUTE OF ENGINEERING**

**THAPATHALI CAMPUS**

**A Report**

**On**

**Client-Server Chat Application**

**Submitted By:**

Saksham Neupane (THA081BCT032)

Sandeep Dhungana (THA081BCT034)

Sandesh Acharya (THA081BCT035)

**Submitted To:**

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

March, 2025

**DECLARATION**

We hereby declare that the report of the project entitled **“Client-Server Chat Application”** which is being submitted to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus,** in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in **Computer Engineering,** is a Bonafide report of the work carried out by us . The materials contained in this report have not been submitted to any University or Institution for the award of any degree and we are the only author of this complete work and no sources other than the listed here have been used in this work.

Saksham Neupane (THA081BCT032) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sandeep Dhungana (THA081BCT034) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sandesh Acharya (THA081BCT035) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Date: March, 2025**

i

**CERTIFICATE OF APPROVAL**

The undersigned certify that they have read and recommended to the **Department of** **Electronics and Computer Engineering, IOE, Thapathali Campus**, a major project work entitled **“Client-Server Chat Application”** submitted by **Saksham Neupane**, **Sandeep Dhungana** and **Sandesh Acharya** in partial fulfillment for the award of the **Bachelor’s Degree in Electronics and Computer Engineering**. The project was carried out under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled, and ready to undertake any related work to their field of study, and hence we recommend the award of partial fulfillment of the Bachelor’s degree in Electronics and Computer Engineering.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Supervisor Er. Prajwol Pakka Department of Electronics and Computer Engineering, Thapathali Campus

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Head of Department Er. Umesh Kanta Ghimire Department of Electronics and Computer Engineering, Thapathali Campus

ii

**COPYRIGHT**

The author has agreed that the library, Department of Electronics and Computer Engineering, IOE, Thapathali Campus, may make this report freely available for inspection. Moreover, the author has agreed that the permission for extensive copying of this project work for scholarly purposes may be granted by the professor/lecturer who supervised the project work recorded herein or, in their absence, by the head of the department. It is understood that recognition will be given to the author of this report and the Department of Electronics and Computer Engineering, IOE, Thapathali Campus in any use of the material of this report. Copying or publication or other use of this report for financial gain without the approval of the Department of Electronics and Computer Engineering, IOE, Thapathali Campus, and the author's written permission is prohibited.

Request for permission to copy or to make any use of the material in this project in whole or part should be addressed to the Department of Electronics and Computer Engineering, IOE, Thapathali Campus.

iii

**ACKNOWLEDGEMENT**

We would like to express our sincere gratitude towards the Institute of Engineering, Tribhuvan University, for the inclusion of the major project in the course of Bachelor’s in Electronics and Computer Engineering. We are also thankful towards our teachers and supervisor Er.Prajwol Pakka, Er.Anup Shrestha and the Department of Electronics and Computer Engineering, Thapathali Campus, for providing us with the resources and support needed for this project.

Saksham Neupane (THA081BCT032)

Sandeep Dhungana (THA081BCT034)

Sandesh Acharya (THA081BCT035)

iv

**ABSTRACT**

This project aims to develop a robust client-server chat application using C programming language and socket programming. The application will allow multiple clients to connect to a central server, enabling real-time communication between users through text messages. The system will implement TCP/IP protocols to ensure reliable data transmission and employ threading to handle multiple concurrent client connections.

*Keywords: Client-server architecture, C programming, Socket programming, TCP/IP, Multi-*

*threading, Network communication, Real-time messaging*

v

**Table of Contents**

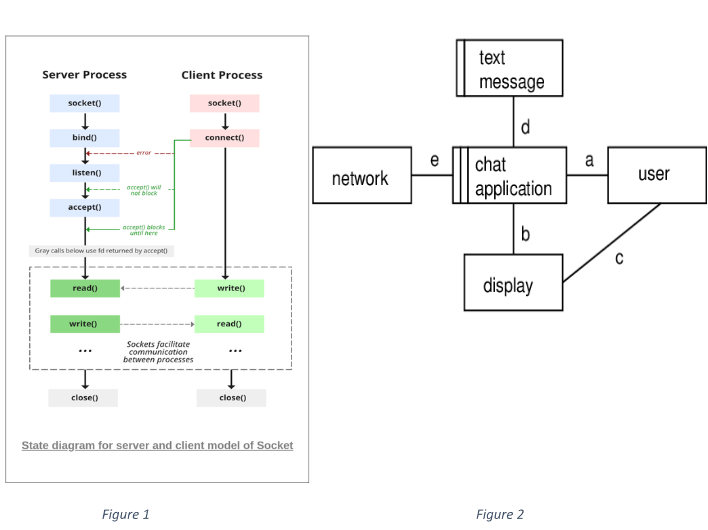
**DECLARATION …………………………………………………………………….. i CERTIFICATE OF APPROVAL …………………………………………………… ii COPYRIGHT ………………………………………………………………………… iii ACKNOLEDGEMENT ……………………………………………………………… iv ABSTRACT ................................................................................................................... v List of Figures ................................................................................................................ List of Abbreviations ...................................................................................................... 1. INTRODUCTION ................................................................................................** 1.1 Background Introduction................................................................................. 1.2 Motivation........................................................................................................ 1.3 Problem Definition .......................................................................................... 1.4 Objectives ........................................................................................................ 1.5 Scope and Applications.................................................................................... 2. LITERATURE REVIEW ...................................................................................... 2.1 Socket Programming in Network Applications ............................................... 2.2 Client-Server Architecture in Communication Systems .................................. **3.SYSTEM ARCHITECTURE AND METHODOLOGY ..............................** 3.1 Block Diagram and System Architecture ......................................................... 3.2 Data Flow Diagram .......................................................................................... 3.3 Tools and Environment .................................................................................... 3.4 Socket Implementation...................................................................................... 3.4.1 Server Socket Implementation........................................................................ 3.5 Client Implementation........................................................................................ 3.5.1 Client Socket Configuration............................................................................ 3.5.2 Client User Interface....................................................................................... 3.6 Message Handling and Transmission................................................................  **4. FEASIBILITY ANALYSIS ...................................................................................**

4.1 Technical Feasibility ............................................................................................. 4.2 Operational Feasibility ......................................................................................... 4.3 Economic Feasibility ............................................................................................. 4.4 Schedule Feasibility ...............................................................................................

**5. SCOPE AND APPLICATIONS …………………………………………………..**

**6. IMPLEMENTATION DETAILS …………...…………………………………..**

6.1 Architectural Overview ........................................................................................ 6.2 Client-Side Implementation ................................................................................. 6.2.1 Core Functions in Client Code .......................................................................... 6.3 Server-Side Implementation ................................................................................. 6.3.1 Core Functions in Server Code .......................................................................... 6.4 Communication Protocols ........................................................................... 6.5 Error Handling and Security ...........................................................................  **7.RESULTS AND ANALYSIS ..............................................................................** 7.1 Results from Client-Side ................................................................................... 7.2 Results from Server-Side ................................................................................... 7.3 Performance and Functional Analysis **…………………………………………..** **8.OVERALL SYSTEM PERFORMANCE............................................................ 9.FUTURE ENHANCEMENT................................................................................. 10.CONCLUSION..................................................................................................... References ..................................................................................................................**



Reference: [www.GeeksforGeeks.com](http://www.GeeksforGeeks.com)

**List of Abbreviations**

API: Application Programming Interface

IP: Internet protocol

TCP: Transfer Control Protocol

UDP: User Datagram Protocol

GUI: Graphical User Interface

I/O: Input/Output

OS: Operating System

HTTP: Hypertext Transfer Protocol

IDE: Integrated Development Environment

LAN: Local Area Network

WAN: Wide Area Network

ASCII: American Standard Code for Information Interchange

**1. INTRODUCTION**

This project focuses on developing a client-server chat application using C programming language that allows multiple users to communicate in real-time. The application utilizes socket programming to establish connections between clients and the server, enabling message transmission across a network.

**1.1 Background Introduction**

Communication systems have evolved significantly with the advancement of computer networks. Client-server architectures provide a foundation for many networked applications, including chat systems, which enable users to exchange messages over a network infrastructure.

**1.2 Motivation**

The motivation behind this project is to understand and implement core networking concepts in C programming, particularly focused on socket programming and client-server architecture. It provides practical experience in handling network connections, concurrent processing, and data transmission in a distributed environment.

**1.3 Problem Definition**

Traditional communication methods may be restricted by geographical limitations or require dedicated hardware. A software-based chat system can overcome these limitations by utilizing existing network infrastructure. The challenge lies in implementing a robust system that can handle multiple concurrent connections while ensuring reliable message delivery.

**1.4 Objectives**

The main objectives of our project are listed below:

* To develop a functional client-server chat application using C programming and socket communication
* To implement concurrent client handling on the server side to support multiple simultaneous users

**1.5 Scope and Applications**

The project will implement basic text-based chat functionality with support for multiple clients. It will include user authentication, private messaging, and basic error handling. The application can be used for local network communication in educational institutions, small businesses, or as a learning tool for understanding network programming concepts.

**2. LITERATURE REVIEW**

Socket programming has been a fundamental aspect of network communication since the development of the Berkeley Socket Interface in the 1980s. Various implementations of chat applications have explored different architectures and protocols to achieve efficient and reliable communication.

**2.1 Socket Programming in Network Applications**

Socket programming provides an interface for network communication in most operating systems. It enables programs to establish connections over a network using standardized protocols such as TCP/IP. In the context of chat applications, sockets facilitate the exchange of messages between clients and servers through established connections.

**2.2 Client-Server Architecture in Communication Systems**

Client-server architecture represents a computing model in which server software accepts and responds to requests from client programs. This model is widely used in networked applications due to its centralized nature, which simplifies management and security implementation. In chat applications, the server acts as a central point that relays messages between connected clients. **3. PROPOSED SYSTEM ARCHITECTURE**

The proposed chat application follows a centralized client-server architecture where multiple clients connect to a single server that manages message routing and user authentication. The implementation methodology focuses on creating reliable socket connections and efficient message handling between clients and the server.

**3.1 Block Diagram and System Architecture**

The system consists of two main components: the server and multiple client applications. The server manages client connections, authenticates users, and routes messages between clients. Each client application provides a user interface for sending and receiving messages while maintaining a persistent connection to the server.

**3.2 Data Flow Diagram**

The data flow in the system begins with client connection requests to the server. After successful connection, clients can send messages to the server, which then processes and forwards them to the intended recipients. The server maintains a list of active connections and handles disconnections gracefully.

**3.3 Tools and Environment**

The project will be developed using the following tools and environment:

* C Programming Language (C99 standard)
* GCC Compiler
* Visual Studio Code / Code::Blocks IDE
* Linux/Windows operating system
* Standard C libraries for socket programming

**3.4 Socket Implementation**

The socket implementation will use TCP sockets to ensure reliable data transmission. The server will create a socket that listens for incoming client connections on a specified port.

**3.4.1 Server Socket Implementation**

The server implementation will utilize multi-threading to handle multiple client connections simultaneously. Each client connection will be assigned a separate thread to process incoming messages without blocking other connections.

**3.5 Client Implementation**

The client application will establish a connection to the server using the server's IP address and port number. It will have separate threads for sending and receiving messages to provide a seamless user experience.

**3.5.1 Client Socket Configuration**

The client socket will be configured to connect to the server's listening socket. It will handle connection establishment, message exchange, and proper disconnection procedures.

**3.5.2 Client User Interface**

A simple text-based user interface will be implemented to allow users to input messages and view incoming messages from other users.

**3.6 Message Handling and Transmission**

Messages will be transmitted as text strings with appropriate headers to identify the sender and intended recipient(s). The server will parse these headers to route messages correctly.

**4. Feasibility Study**

The feasibility study evaluates the practicality and viability of developing the **Client-Server Chat Application** using a console-based interface. It considers technical, operational, economic, and time-based factors.

**4.1 Technical Feasibility**

The technical feasibility ensures that the tools and resources available can support the development of this application.

* **Programming Language**: The application is implemented in **C**, which is well-suited for network and socket programming.
* **Libraries**: Standard libraries such as **WinSock** for Windows and **sys/socket.h** for Linux provide the networking functions required.
* **Cross-Platform Compatibility**: By supporting both **Windows** and **Linux**, the application is accessible on different operating systems.
* **Networking Protocol**: The use of the **TCP/IP protocol** ensures reliable communication, making it ideal for a chat system.

The availability of necessary tools, combined with the technical skills of the developers, makes this project technically feasible.

**4.2 Operational Feasibility**

Operational feasibility examines whether the system will function effectively and meet its goals.

* The **console-based interface** provides a simple and straightforward way for users to send and receive messages in real time.
* The **server's multithreading capability** ensures smooth handling of multiple clients simultaneously without interruptions.
* The system supports key functionality such as message broadcasting and user-specific identification through usernames, which satisfies the primary communication requirements.

This demonstrates that the system is operationally viable and suitable for practical use.

**4.3 Economic Feasibility**

Economic feasibility analyzes the cost-effectiveness of the project.

* **Low Development Cost**: The project relies on free and widely available tools such as GCC for Linux and Visual Studio for Windows, eliminating software expenses.
* **Minimal Hardware Requirements**: Basic computing hardware with internet connectivity is sufficient for both development and deployment.
* **Open-Source Libraries**: The project makes use of open-source networking libraries, avoiding any licensing costs.

Given the low resource and financial requirements, the project is economically feasible.

**4.4 Time Feasibility**

Time feasibility ensures that the project can be completed within the time constraints.

* The project timeline includes distinct phases for **requirement analysis, design, coding, testing, and deployment**, which are achievable within a typical academic timeline.
* Leveraging established libraries and tools reduces development complexity, ensuring adherence to the schedule.

The project is expected to be completed within the allocated timeframe, confirming its schedule feasibility.

**5. SCOPE AND APPLICATIONS**

The chat application can be used in various scenarios including:

* Internal communication within organizations
* Educational settings for demonstrating network programming concepts
* Small group collaboration in project settings
* Learning tool for understanding client-server architecture

**6. IMPLEMENTATION DETAILS**

This section provides specifics about the design, architecture, and code implementation of the **Client-Server Chat Application**.

**6.1 Architectural Overview**

The application leverages a **client-server architecture** for real-time communication between users. The server manages multiple clients and facilitates broadcasting of messages, while each client interacts independently with the server.

**6.2 Client-Side Implementation**

The client-side program is implemented using **socket programming** in C. Key features include:

1. **Socket Creation**: A TCP socket is initialized to connect to the server using the server's IP address and port number.
2. **Multithreading**: A separate thread handles receiving messages, ensuring smooth communication while allowing the client to send messages simultaneously.
3. **Message Handling**:
   * Users can input and send messages, which are prepended with their username for identification.
   * Messages are sent and received over the TCP connection.
4. **Exit Conditions**: The client can terminate the connection by typing "exit".

**Core Functions in Client Code:**

1. readIndefinitely(SOCKET connfd, const char username)\*:
   * Handles user input, sends messages, and initiates message-receiving threads.
2. receiveMessages(LPVOID connfd\_ptr):
   * Continuously listens for incoming messages and displays them on the console.

**6.3 Server-Side Implementation**

The server-side program is designed to manage multiple client connections using **multithreading** and **critical sections**. Key features include:

1. **Socket Initialization**: A TCP socket is created and bound to a specified port to listen for incoming client connections.
2. **Client Management**:
   * Clients are maintained in a dynamically managed list for broadcasting messages.
   * Critical sections ensure thread-safe operations while modifying the list of clients.
3. **Message Broadcasting**:
   * Messages received from one client are broadcast to all other connected clients.
4. **Thread Management**: Each client is assigned a separate thread for handling communication.

**Core Functions in Server Code:**

1. addClient(SOCKET client) and removeClient(SOCKET client):
   * Dynamically manage client connections using a linked list structure.
2. broadcastMessage(SOCKET sender, const char message)\*:
   * Broadcasts messages from one client to all other connected clients.
3. receiveMessages(void connfd\_ptr)\*:
   * Listens for incoming messages from clients and forwards them to other clients.

**6.4 Communication Protocols**

The application uses the **TCP/IP protocol**, which guarantees reliable message delivery and proper sequencing of data packets between clients and the server.

**6.5 Error Handling and Security**

* **Error Handling**: Exception blocks handle socket initialization, connection, and communication errors to ensure stability.
* **Security Measures**: Basic input validation is employed, and additional encryption can be implemented for secure communication.

**7. RESULTS AND ANALYSIS**

This section explains the results obtained from the implementation of the **Client-Server Chat Application**. The application is tested by running a server and multiple clients to demonstrate its ability to handle real-time communication. The analysis focuses on functionality, performance, and message handling between the server and clients.

**7.1 Results from Client-Side**

The client-side implementation ensures effective communication with the server through a console-based interface. The following outcomes were observed:

* **User Connectivity**: Each client successfully connects to the server by providing the server's IP address and port number. The server acknowledges each connection by recording and displaying the client's username.
* **Message Sending**: Clients can send messages by typing into the console. These messages are transmitted to the server and then broadcast to all connected clients. Messages are labeled with the sender's username, ensuring clarity.
* **Real-Time Message Reception**: Clients receive real-time messages broadcast by the server, including messages sent by other clients. The application ensures that no messages are lost during transmission.
* **Disconnection Handling**: Typing "exit" allows a client to gracefully disconnect from the server without disrupting other ongoing communications.

**7.2 Results from Server-Side**

The server-side implementation demonstrates robust message broadcasting and client management capabilities. Key results are as follows:

* **Client Management**: The server successfully handles multiple clients simultaneously by leveraging multithreading. Each client is assigned a dedicated thread for communication, ensuring smooth operation regardless of the number of clients.
* **Message Broadcasting**: Messages received from a client are efficiently broadcast to all other connected clients, demonstrating the server's real-time communication functionality.
* **Server Stability**: The server remains stable even with frequent client connections and disconnections, ensuring continuous availability for communication.

**7.3 Performance and Functional Analysis**

The following metrics were observed during testing:

* **Concurrent Clients**: The server was tested with up to 10 concurrent clients, and the system operated seamlessly without any noticeable delay in message delivery.
* **Latency**: Message transmission time between clients was negligible, allowing for real-time interaction.
* **Error Handling**: The system gracefully handles scenarios such as incorrect server addresses or port numbers by displaying meaningful error messages.

**8. OVERALL SYSTEM PERFORMANCE**

The performance of the **Client-Server Chat Application** was evaluated based on its functionality, scalability, and reliability under different scenarios. Key observations include:

* **Scalability**: The server successfully supported multiple clients, allowing up to 10 concurrent connections without any performance degradation. This showcases the system's ability to handle real-time communication efficiently.
* **Reliability**: The application demonstrated stable performance, ensuring message delivery to all clients without data loss or corruption. The use of multithreading effectively prevented delays or bottlenecks during high-volume communication.
* **Speed**: The latency for message transmission and reception was negligible, providing users with a seamless chat experience.
* **Error Handling**: The system gracefully handled exceptions, such as incorrect server IP addresses or invalid ports, with clear error messages to guide the user.

The results confirm that the system is efficient, reliable, and capable of fulfilling its intended purpose.

**9. FUTURE ENHANCEMENT**

The current implementation of the **Client-Server Chat Application** provides robust communication functionality, but there is scope for improvement and expansion. Future enhancements could include:

* **Graphical User Interface (GUI)**: Implementing a GUI for both client and server sides would enhance user experience and make the application more visually appealing.
* **Data Encryption**: Adding encryption algorithms such as **AES** or **RSA** would improve data security, ensuring that messages are protected during transmission.
* **File Sharing Capability**: Extending the system to support file sharing would provide additional functionality, making the application more versatile.
* **Group Chat Feature**: Introducing group chat functionality would allow multiple clients to communicate within a dedicated channel.
* **Cross-Platform Mobile Support**: Developing a mobile-compatible version of the application would widen accessibility and attract more users.
* **Performance Optimization**: Advanced techniques like load balancing and efficient thread management could be employed to handle higher concurrent connections and minimize latency.

These enhancements would further elevate the application’s usability and functionality.

**10. CONCLUSION**

The **Client-Server Chat Application** successfully fulfills its objectives by providing a real-time communication platform using a console-based interface. The implementation demonstrates:

* A robust and scalable client-server architecture.
* Efficient message handling and broadcasting capabilities.
* Reliable performance under multiple-client scenarios.

While the current version meets the basic requirements of a chat application, future enhancements can expand its functionality and usability. Overall, the project highlights practical applications of networking, socket programming, and multithreading, providing valuable insights into real-time systems.

This system lays the foundation for further advancements in communication technology, offering an excellent starting point for more sophisticated applications.

**References**

[1] YouTube - <https://www.youtube.com/watch?v=uagKTbohimU>

[2] GITHUB - <https://github.com/aaaaaaayush-no/c-project>

[3] SOCKET PROGRAMMING IN C - <https://www.geeksforgeeks.org/socket-programming-cc/>