FORT HAYS STATE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

# DuoChat App - Application for Users Communication

**Project Report**

Noreliz Alorico, Kristina Botova

Fort Hays State University

Department of Computer Science

CSCI 682 - Application Layer Programming

FORT HAYS STATE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

## Table of Content

## 1. Introduction

Human communication has seen several tremendous transformations in the last few years. Real-time chat messaging systems and platforms have become an essential part of our daily routines. In fact, 41 million [1] messages are sent every minute from messaging apps. A report from Statista [2] notes that as of January 2023, WhatsApp has two billion monthly active users.

Chatting is a method of using technology to enable instantaneous user communication. It helps us to stay connected with our friends, colleagues and relatives even if they are far away at the moment. A chat application is a type of software that enables users to communicate with each other in real time.

Our project is an example of a multi-threaded chat room service that allows multiple clients to communicate with each other through a central server. A program for communication between clients through server will be developed using socket programming in C language. It focuses on network communication via Transmission Control Protocol (TCP). The system comprises two key components: the server and the client, both capable of running independently on separate computers.

Despite the existence of widely-adopted chat applications, many businesses have good reasons to build their own chat applications. Chat applications can be customized to suit different industries and businesses, providing a unique platform for users to communicate and collaborate. Overall, chat applications have become indispensable tools.

## 2. Problem Statement

- Create a multi-threaded chat room service that allows multiple clients to communicate with each other through a central server.

- Develop an instant messaging communication feature such as sending and receiving messages seamlessly.
- Develop a program using socket programming in C language that focuses on network communication via TCP.

To summarize the development of a chat application that allows users to communicate with each other in real-time. The application should include the server and the client, both capable of running independently on separate computers.

## 3. Solution

In this project, a multithreaded chat application is implemented. Simply, it makes communication with people from anywhere in the world easy by sending and receiving messages. Users can access the application from anywhere with an internet connection, making it easy to stay connected. The system has one server and multiple clients. Key features:

1. **Socket Communication:** The application employs TCP sockets to establish reliable connections. This ensures that messages are transmitted accurately and in order, making it ideal for chat applications where message integrity is crucial.
2. **Real-time Messaging:** Users can send and receive messages instantly, fostering an engaging and interactive user experience. The chat interface allows users to see messages as they come in, making conversations feel natural and fluid.
3. **User-Friendly Interface:** Although the application is console-based, it is designed to be intuitive. Users can easily enter messages, view incoming messages, and respond without complicated commands or navigation.

Building a real-time chat application requires careful planning to ensure it meets desired functionality and user experience. It should include the following:

1. Socket interface is used to implement network communications.

2. The C chat application is a console application that is launched from the command line using TCP connection.
3. Multiple clients can connect to a server and they can chat to each other.
4. The application consists of two parts: Server and Client. Each part can run independently on separate computers.
5. Server manages the session, it is listening for client connections forever when it is started. When a connection request is received, the server communicates with each client through a separate thread (multitrade). Server forwards incoming messages.
6. Client allows users to connect to the server for sending and receiving messages. Client can exit the chat by typing a specific command ("bye").

## 4. Methodology

DuoChat App is a streamlined, intuitive messaging solution designed for seamless communication between two users. Ideal for personal chats, collaborative discussions, or customer support interactions, this application focuses on delivering an efficient, and secure messaging experience**.**

The chat application follows a client-server architecture, which is a common design pattern for real-time communication systems. In this architecture, the server is responsible for managing client connections, handling incoming messages, and broadcasting them to other clients. The clients, on the other hand, connect to the server, send messages, and receive messages from the server in real-time. Communication between the clients and server is established using TCP sockets, which provide reliable, ordered, and error-checked delivery of data between the connected entities. Multithreading is a critical part of the server-side implementation, as it allows the server to manage multiple clients simultaneously without blocking, enabling real-time communication between all connected users.

On the client side, the application begins by prompting the user for a **username**, which is essential for identifying the user during communication. Once the username is entered, the client establishes a TCP connection to the server and sends the username for identification purposes. After the

connection is established and the username is sent, the client can send messages to the server. The client continuously prompts the user for input, sending each message to the server. To avoid blocking the main thread and ensure that the user can send messages without interruption, a separate thread is created to handle incoming messages from the server. This thread listens for messages and displays them to the user as they are received. This multithreaded approach allows the user to maintain a seamless experience, where they can type messages while also receiving updates in real time without having to wait for one action to complete before starting the next.

On the server side, the application starts by **listening** for incoming connections on a specified port. When a client attempts to connect, the server **accepts** the connection and creates a new thread to handle that specific client. Each client is managed independently within its thread, allowing the server to communicate with multiple clients concurrently. When a client sends a message, the server processes the message and forwards it to all other connected clients, ensuring that every client receives the message. The server formats the message by appending the sender's username to help recipients identify who sent the message. If a client disconnects by sending the message "bye", the server detects this command and broadcasts a disconnection message to the other clients, notifying them that the client has left the chat. The server then removes the client from its active list, ensuring that no further messages are sent to the disconnected client.

Multithreading is used throughout the server to allow it to handle multiple clients at once. Each time a new client connects, a new thread is created to handle that client's communication, and each client's interactions are processed in isolation from others. This approach makes the system scalable, as it allows multiple clients to communicate with the server and each other in parallel without affecting performance. Additionally, the server is designed to broadcast messages to all connected clients, making it function as a central hub for communication. The server is also responsible for managing client disconnections and ensuring that resources are properly released when clients leave.

Message formatting is an important feature of the system, as it ensures that all communications are clear and easy to read. Each message that is broadcasted to other clients is prefixed with the sender's username, allowing users to identify the source of the message. To improve the readability of the conversation, messages are displayed with added spacing, ensuring that each message stands out. This is particularly important in a real-time chat system, where users might be sending and receiving messages rapidly, and clear formatting makes it easier to follow the conversation.

Error handling is another key aspect of the application. The system is designed to handle common errors such as connection failures, message sending issues, and client disconnections. If a client unexpectedly disconnects or if there is an issue with message delivery, the server gracefully handles these scenarios by removing the client from the active list and notifying other clients as needed. This helps ensure that the system remains stable even in the event of failures, and users experience minimal disruptions during communication.

Overall, the methodology emphasizes efficient and concurrent communication between clients, seamless message reception and sending, and clear message formatting for better user experience. The use of multithreading ensures that the server can handle multiple clients simultaneously, enabling real-time, uninterrupted communication. Error handling mechanisms and disconnection management maintain system stability and user experience, even when a client unexpectedly disconnects. The chat application offers a foundation for real-time messaging and can be further enhanced with additional features such as encryption, user authentication, and file transfers to make it more secure and versatile for future use cases.

## 5. Technical Requirements

The technical requirements are structured to ensure that both the client and server components operate efficiently and reliably. On the hardware side, the client device, which can be a computer, laptop, or mobile device, needs a minimum of a 1 GHz processor, 1 GB of RAM, and 50 MB of available disk space. An active internet connection or a local network connection is required for communication with the server. The server, which is responsible for managing multiple client

connections simultaneously, requires a more powerful configuration. It should have at least a 2 GHz multi-core processor, 2 GB of RAM, and 100 MB of free disk space for efficient operation. A stable internet connection with a static IP address is crucial for the server, as this allows clients to reliably connect and communicate with it over the network.

In terms of software requirements, the chat application is designed to run on Unix-like operating systems such as Linux or macOS. It is also compatible with Windows when using a POSIX-compatible environment like Cygwin or Windows Subsystem for Linux (WSL). The operating system should have a modern kernel version to ensure compatibility with networking and threading libraries. For compiling and building the application, a C compiler like GCC or Clang is required. The build system can be managed using tools like Make or CMake, which help automate the process of compiling the C source code and managing dependencies.

The application is developed in the C programming language, which provides low-level control over system resources, making it ideal for socket communication and multithreading. To handle concurrent communication, the application uses the POSIX threading library (pthread), enabling the server to manage multiple client connections in parallel. The standard C libraries such as **stdio.h** for input and output, **stdlib.h** for memory management, and **string.h** for string manipulation are employed to facilitate basic functionality. For network communication, the application relies on the **sys/socket.h** and **netinet/in.h** libraries, which provide functions for creating and managing sockets and handling network communication. The **arpa/inet.h** library is used to convert IP addresses between string and binary formats.

The communication protocol used for the application is **TCP/IP**, a reliable, connection-oriented protocol that ensures the messages between the client and server are delivered in the correct order without errors. The server listens for incoming connections on a predefined port (port 8080 by default), and both the client and the server must agree on this port for successful communication. The server must be accessible either via a **static IP address** (for remote clients) or a local network

address (for local communication), depending on the setup. Clients need to know the server's IP address to establish the connection, making proper IP addressing critical for communication.

The server-side component of the application uses multithreading to handle multiple clients concurrently. Each client connection is managed by a separate thread, allowing the server to independently process messages from each client without blocking other connections. This approach enhances the scalability of the server, as it can handle numerous clients without performance degradation. Additionally, error handling is built into both the client and server components to ensure smooth operation. For instance, the server gracefully handles client disconnections and ensures that resources such as memory and sockets are properly released when a client leaves. This prevents memory leaks or system instability.

In summary, the technical requirements for the chat application ensure its smooth operation by specifying the necessary hardware and software configurations for both the client and server. The system relies on modern networking technologies, multithreading, and error handling to provide a reliable and scalable chat solution. The use of TCP/IP ensures reliable message delivery, while multithreading enables concurrent client handling, making the application suitable for real-time communication scenarios in both local and remote network setups. With these requirements in place, the chat application can efficiently handle multiple clients, manage connections, and provide a seamless messaging experience.

## 6. Limitations

Despite the advantages of the proposed chat application, there are several limitations that need to be addressed. One of the main limitations is the **scalability** of the system. As the number of clients increases, the server may experience performance issues due to the overhead associated with creating and managing multiple threads. This issue can become more pronounced if the server is running on hardware with limited processing power or if there are too many concurrent

connections. The current server design might not be able to efficiently manage hundreds or thousands of clients without experiencing slowdowns or increased response times.

Another limitation is related to **security**. The system lacks advanced security features such as message encryption, authentication, or access control mechanisms. Without encryption, the messages sent between clients can be intercepted and read by unauthorized parties, compromising the confidentiality of the communication. Furthermore, without authentication, malicious users could impersonate others or gain unauthorized access to the system. While basic error detection is included, the system does not provide robust security against more sophisticated attacks, such as denial-of-service (DoS) or man-in-the-middle (MITM) attacks.

Additionally, the **error detection** methods employed, while useful, are somewhat limited in their ability to handle complex transmission errors. More advanced error correction mechanisms, such as **Forward Error Correction (FEC)**, could be incorporated to improve message reliability in situations where the network conditions are less than ideal. Without these enhancements, the system may not perform optimally in environments with high packet loss or network instability.

The **user interface** of the client-side application is rudimentary and could benefit from improvements to make it more intuitive and user-friendly. The current interface only supports basic functionality, and it lacks features such as chat room creation, file sharing, or direct image transmission. Improving the user experience with more advanced features would make the application more appealing to a wider audience.

## 7. Future Enhancements

To elevate DuoChat App from its current educational prototype to a fully-featured application, several enhancements are recommended. First, integrating a graphical user interface (GUI) would significantly improve the user experience. A GUI would allow users to interact with the application more easily, with features like drag-and-drop message sending, a chat window, and a user-friendly way to view the list of active clients. A GUI could also enable features such as message formatting, emoticons, or the

ability to send multimedia content like images or files. In terms of security, implementing encryption protocols such as TLS would safeguard communications against eavesdropping and data breaches. To improve **security**, encryption mechanisms should be added to the system to protect the confidentiality of messages during transmission. Implementing encryption standards such as **AES** (Advanced Encryption Standard) would ensure that even if messages are intercepted, they cannot be read by unauthorized users. Additionally, **authentication** features could be introduced, requiring clients to log in with a username and password before connecting to the server. This would prevent unauthorized users from joining the chat system or impersonating other users. Enhancing error handling is also crucial; robust mechanisms to manage network issues, invalid inputs, and unexpected disconnections will improve stability and user experience. Integrating more advanced error detection and correction algorithms, such as Forward Error Correction (FEC) or automatic retransmission requests (ARQ), could significantly improve the system's ability to recover from transmission errors, ensuring message integrity even in unreliable network conditions. Introducing message persistence features, such as logging and conversation history, would allow users to retain their chat records across sessions. Additionally, to ensure the application can handle growth and increased user demand, the server should be adapted for scalability through multi-threading or asynchronous processing, enabling it to manage multiple clients simultaneously without performance degradation. Scalability can be improved by optimizing the server-side logic to handle a larger number of concurrent connections. This could involve using a non-blocking I/O model or load balancing techniques to distribute client requests more efficiently across multiple servers. Another approach would be to replicate the server to handle more clients by distributing the load across different servers, ensuring that the system can handle higher volumes of traffic.

These enhancements will collectively contribute to a more secure, reliable, and user-centric chat application.

- Group chats - Multiple users can chat with each other within a group chat channel.
- Reactions, stickers, and emojis - Users can react to chat messages through the use of emojis or stickers without needing to type full messages.
- Online presence and status indicators - Users can expose their status (online, away, do not disturb) for other users to see.

- Messaged editing and deletion - Users can edit messages that have already been sent, and delete sent messages so that they are no longer available on a recipient's device.
- Message delivery/read status - A sender can see if the intended recipient has received and read a message that was sent.
- Notifications - To keep users informed, push notifications alert them to new messages even when the application is not open. Notification settings are customizable, allowing users to manage their alert preferences according to their needs.
- Search - The in-app search functionality enables users to quickly find specific messages or media.

## 8. Conclusion

DuoChat App is a streamlined messaging solution designed to facilitate seamless communication between users. There are two components to the chat application: the server and the client. Every component can function separately on different machines. A server may have several clients connected, allowing them to communicate with one another. Network communications are implemented via the socket interface. Launched from the command line via TCP connection, the chat application is a console application.

Developed entirely in C, the DuoChat App leverages the power of sockets for networking, allowing real-time messaging between clients. The application architecture is built to support instant communication, ensuring that users can exchange messages without delays. In addition to overseeing the chat session, the chat server keeps track of all active clients. It delivers incoming messages to the correct person. The server is multithreaded, meaning that a different thread is used for communication with each client. The server waits for client connections after it has started. The server initiates a thread to serve the client upon receiving a connection request. For every client it is required to have its own socket.

This application is particularly well-suited for various scenarios, including personal chats, collaborative discussions, and customer support interactions, with a strong emphasis on efficiency and security.

## 9. References

[1] Damjan. “Text, Don’t Call: Messaging Apps Statistics for 2024.” *KommandoTech*, 12 Sept. 2023, kommandotech.com/statistics/messaging-apps-statistics/

[2] Dixon, Stacy Jo. “Most Popular Global Mobile Messenger Apps as of April 2024, Based on Number of Monthly Active Users.” *Statista*, 27 Aug. 2024, www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/