External Project Presentation of Computer Networking(CSE 3034)

on

Chat Application using TCP/IP

Supervisor

Ashutosh Chakrabarty

Presented by

P. Biswanath Patra (2141016196)
P. Padmaja Patra (2141016197)
Trishna Tanaya Patra (2141016198)
Aryan Patri (2141016225)
Ayusha Pattnaik (2141016248)

Department of Computer Science and Engineering

Institute of Technical Education & Research (FET)

Siksha 'O' Anusandhan Deemed to be University, Bhubaneswar

Jan, 2024

Contents

- Introduction
- Problem statement
- Methodology
- Implementation
- Result and Interpretation
- Conclusion

Introduction

- The project is a basic client-server chat application implemented in Java, using socket programming. It consists of two main components:

Chat-Server

- Upon connection, establishes input and output streams for communication with the client.
- The server operator input messages from the console, which are sent to the client.

Chat-Client

- Establishes input and output streams for communication with the server.
- The client user take input messages from the console, which are then transmitted to the server.

Problem Statement

User Input Identification

Users input messages through the console, and the application needs to identify these messages.

Processing User Messages

The Server's Client-Handler continuously reads and process messages from the connected client.

Reflection of Results in the Console

Processed messages, whether from the client or server, are currently displayed in the console with print statements.

Persisting Results

The current implementation does not persist the chat history. If there's a need to store the chat log, it could be saved to a file or a database

Methodology

The following pseudocode outlines the basic algorithm for the chat application, covering the main functionality of the server and client components.

Server Pseudocode:

1. Initialize ServerSocket on specified port.
2. Listen for incoming connections.
3. Upon connection, create a new Socket.
4. Create input/output streams for communication.
5. Start clientHandler thread.
6. In main thread:
    a. Read messages from console.
    b. If message is "exit," close connection and terminate.
    c. Send message to connected client.

Client Pseudocode:

1. Initialize Socket with server's IP and port.
2. Create input/output streams for communication.
3. Start server-Handler thread.
4. In main thread:
    a. Read messages from console.
    b. If message is "exit," close connection and terminate.
    c. Send message to server.

ClientHandler/ ServerHandler Threads:

1. Continuously read messages.
2. If message is null or "exit," terminate thread.
3. Display received message in console.
4. Process or broadcast message.

Implementation

Networking:

- Utilizes Java's ServerSocket and Socket classes for network communication.
- Establishes connections between the server and clients.

Multithreading:

- Implements multithreading to handle concurrent communication with clients.
- Dedicated threads (clientHandler and serverHandler) manage message processing.

Input/Output Streams:

- Utilizes BufferedReader and PrintWriter for efficient data exchange between server and clients.

Console Interface:

- The user interacts with the application through a console-based interface.
- Server and client display messages and await user input.

- **The implementation provides a foundational structure for a chat application, showcasing key concepts in Java programming.**
- **Future enhancements can transform this basic implementation into a more feature-rich and scalable communication platform.**

Result and Interpretation

Server Output:

1.Server Start:
- The server starts and listens on a specified port (e.g., port 12345).
- You see a message like: "Server listening on port 12345."

2.Client Connection:
- When a client connects, you see a message indicating the client's IP address.
- Example: "Client connected: 127.0.0.1"

3.Server Console Input:
- The server awaits input from the console, prompting with "Server: ".
- You type a message and press Enter.

4.Client Message Display:
- The server outputs the message you entered to the connected client.
- Example: "Client: Hello, how are you?“

Client Disconnection:
- If a client enters "exit" in the console, the server displays "Client disconnected" and terminates the connection.

Client Output:

1.Client Start:
- The client starts and attempts to connect to the server's IP address and port (e.g., 127.0.0.1:12345).
- You see a message indicating a successful connection.

2.Server Message Display:
- The client's serverHandler thread continuously displays messages received from the server.
- Example: "Server: Welcome to the chat!"

3.Client Console Input:
- The client awaits input from the console, prompting with "Client: ".
- You type a message and press Enter.

4.Server Message Display:
- The client sends the message to the server, and the server displays it in the console.
- Example: "Server: How can I assist you?"

Server Disconnection:
- If you enter "exit" in the client console, the client displays "Disconnected from server" and terminates the connection.

Conclusion

Key Achievements:

- Successful establishment of server-client connections.
- Effective implementation of multithreading for concurrent communication.
- Basic message processing for sending and receiving messages.

Limitations and Future Opportunities:

- Identified single-client limitation and basic message processing.
- Opportunities include GUI integration, feature expansion, and scalability considerations.

References

- Computer Networks, Andrew S. Tannenbaum, Pearson India
- Java Network Programming by Harold, O'Reilly (Shroff Publishers).
- Google - https://www.google.co.in/

Thank You!!!

Any Questions?