Report

Semester Project Part-1

[Client-Server architecture]

[Computer networks]

[2024-2028]

- **[Muhammad Saad Saif | 68408]**
- **[Laiba Zahir | 68864]**
- **[Anzal Faheem | 67954]**

[2nd Semester]

Submitted to:

[Miss Poma]

[Department of _____<u>Computer Science</u>______]

Faculty of Information and Communication Technology (FICT), BUITEMS, Quetta

Abstract

This Project works on the design and implementation of communication chat application between several clients using java language and java socket programming **“The client server architecture”.** Multiple clients connect through multi-threading method, broadcast the messages and handle the errors. Key features include:

- GUI
- Automatic reconnection
- Thread safe client management.
- Input validation

This project of client and server handles 150 clients with maximum 85% of usage of CPU.

Table of Contents

1. Problem Statement:

In the modern world, communication is so important for offices work, education purpose and furthermore. Many existing chat apps face issues like:

- ❖ **Connection:**
  Clients may keep losing connections due to network issues.
- ❖ **Scalability issues:**
  Single threaded server that struggles in multi-client connection.
- ❖ **Disconnection:**
  Whenever a server disconnected or restarted client stayed disconnected without a smooth of reconnecting after the server restarted.

The client-server architecture is a foundational model that separates service providers (servers) from service requesters (clients), enabling scalable, organized, and reliable network communication. The task is to ensure managing multiple client connections, maintaining data integrity, and smooth interaction between the client and server, also the reconnectivity phase. Now we have to build a basic client-server chat application that highlights main issues like connection handling, multiple clients, timeouts, errors, or specific character errors using java language.

2. Introduction:

➢ Problem Overview:

Many chat applications suffer from errors and issues such as connection instability, lack of automatic reconnection, scalability issues, and poor error handling. These problems lead clients to frequent disconnections, manual reconnection efforts, and inefficient message handling, all these problems prevent clients from smoothly chatting.

As I mentioned above, the interconnected world is basically bound to a communication system that plays the role of multi-client handling, facing errors, also the reconnection when servers stop and the smooth disconnection.

So, to overcome the problems we have made a client server architecture using java socket programming with a GUI that contains a centralized server which manages multiple client requests using multithread, broadcast the messages. Furthermore, we have also added reconnection to server after disconnecting logic, that prevents clients from manually connecting themselves to server instead smoothly reconnect without any error.

➢ Problem analysis:

The problems are:

- ❖ **Connection stability:**
  - All clients lose connection when the server stops.
  - It requires manual client reconnection, that leads user's bad impression and experience.
- ❖ **Session management:**
  - In case the server restarts all clients lost their IP Port and cannot reconnect.
  - No logic is there to restore the previous session.
- ❖ **Scalability:**
  - Single thread server cannot handle multiple clients, which is not useful.
  - And without proper synchronization the collision of messages and delays in messages can occur.
- ❖ **Input validation:**
  - Special characters can cause formatting issues in text.

The solutions are:

- ❖ **Multi-thread server:**
  Handle multiple clients using server sockets and Client handler thread.
- ❖ **Automatic reconnection:**
  After the server stops and restarts, the client detects and connects themselves to the server with their previous username and IP address.
- ❖ **Broadcast Messages:**
  Whenever the client messages, it will be sent to all other clients including the server.
- ❖ **Input Validation:**
  Whenever the client enters special characters, the server detects it and send error to that specific client.
- ❖ **GUI interaction:**
  Both the clients and server get the best UI interface which attract the user for communication.

➢ Project Phase:

The phases of the project were:

- ❖ **Downloading and installation:**
  The downloading and installation of NetBeans IDE for the implementation of the code of java, the installation of JDK the compiler that compiles java code, and JavaFX for the UI components.
- ❖ **Logic building, articles and videos:**
  As we are new to java language so don't know much about this, to overcome this we read the article regarding client and server that how basically it works and watch the videos from many resources, then ourselves we made a simple logic using IPO chart and Flowchart.
- ❖ **Requirements Analysis:**
  Identify the key features like client handler, message broadcasting, GUI, server's start stop functions, reconnection.
- ❖ **Design phase:**
  Structure client server architecture with multi-threading and giving better GUI.
- ❖ **Implementation:**
  After all the above thing we implement the code in NetBeans application software in java programming language, swing for GUI components, socket programming for network connection.
- ❖ **Testing and debugging:**
  In this phase we implement reconnection logic, and the error handling using try-catch block of code.
- ❖ **Documentation:**
  After the code side is completed, we have done the documentation.
- ❖ **Video presentation:**
  The video presentation means our each and everything in code will be described in it. (link mentioned at the end of the document).

➢ Literature and technologies:

This project is built upon:

- **Java socket programming** for networking connections.
- **Swing** is used for GUI developments and components.
- **Concurrent HashMap & Synchronized Collections** used for thread safe client management.
- **Reconnection** used for auto connection of clients after disconnection.

3. Literature Review:

Client-server architecture is a way computers work together. In this system, there are two main parts: the client and the server. The client asks for help or information, and the server gives it. This architecture is mainly used in web applications, communication systems and more.

The server is like a big computer that stores data, manages data, processes or runs programs. The client is usually a smaller device and also known as end-user device, like your phone or laptop, that connects to the server over a network or the internet. And typically, TCP/IP protocols are used for saving, reliability and gate to gate data transfer.

This system is very common and helps many applications work well. People study how to make client-server systems faster, safer, and able to handle many users at the same time.

Today, with cloud computing, servers can be very powerful, and clients can get services from far away. Still, there are challenges like slow connections and keeping data safe.

In short, client-server architecture is important because it helps different devices and programs talk to each other easily

4. Methodology (Design and Simulation):

5. FLowchart

- **Overview:**

  This project follows the structure to design and implement multi-client chat application using java programming language.

  The methodology includes:

  - Requirement analysis.
  - System design.
  - Implementation.
  - Testing.

- **Tools and Purposes:**

  Below are the **tools** and **purposes.**

  - **Java language:**

- Backend logic, networking etc.

- **Java swing:**
  - GUI development and components for both server side and client side for user better experience.
- **Server socket:**
  - Establish connection using IP addresses.
- **Multi-threading:**
  - Multi-clients handling.
- **Datainputstream \ Dataoutputstream:**
  - Reading and writing messages.
- **HashMap:**
  - Storage of active clients

Design:

This system follows a centralized client and server model.

- **Server:**
  - It manages client connections.
  - Broadcast messages to all active clients.
  - Maintain a thread safe list for active clients.
- **Client:**
  - Connect to server through IP.
  - Send and receive messages.

Material:

- **Threading function:**

In the threading function the server uses thread per client method:

- Each clientHandler thread, manages and targets each single client.
- Prevent blocking of input and output streams

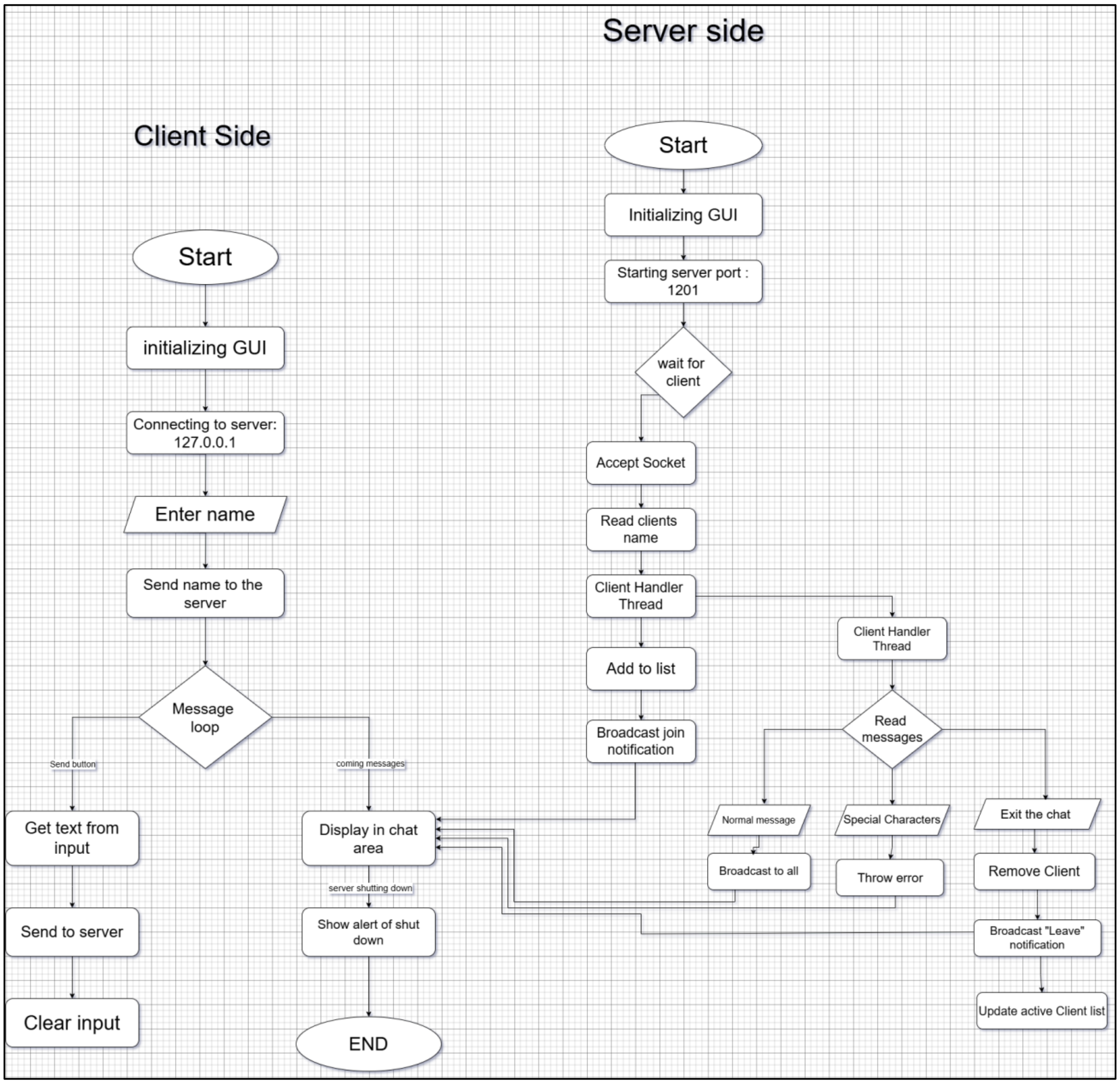
Equation:

$$\text{Total Threads} = N\text{clients} + 1 \text{ (main server thread)}$$

- **Manage handling:**
  - **Broadcasting:** server send messages to all active clients.
  - **Special character messaging:** reject messages containing characters like @#$%^&*!.
  - **Equation (broadcast time):**

$T = O(N)$ (Linear time, where N = number of clients).

6. Flowchart:



7. Results and Discussion:

In this project, a simple *client-server communication system* was successfully implemented. The client and server were able to establish a connection, exchange messages, and terminate the connection properly. Key observations include:

- The server was set up to listen on a specific port, and the client successfully connected to it.
- Messages sent from the client were received and displayed by the server.
- The server responded to the client, confirming the reception message.
- The connection closed gracefully when either side terminated the session.

The system worked as expected, demonstrating basic *socket communication* between two endpoints. The server successfully handles multiple clients with no thread collision no message delays.

8. Conclusion and Future Work:

❖ Conclusion:

In this project, we made a system where one computer (client) can connect to another (server) to send messages. Making this, we learned a lot about socket programming, designing interfaces (GUI), and sending data. Our project works well for basic remote communication.

❖ Future Work:

- **Better Security:** In the future, we can add strong security so no one else can see the data being sent.
- **Support for Other Devices:** We can improve the system, so it works on Linux and Mac too.
- **Faster File Sharing:** By adding file compression, files will transfer faster and use less internet.
- **Better Interface:** The design of the app can be improved to make it easier and more fun to use.
- **More Users at a Time:** this project handles about 150 users. We can improve it so the server can handle more than 150 users at once.

9. References:

- Client server architecture explain

  https://youtu.be/rd272SCl-XE?si=tkZdDRJhKxxK0Gso

- Multiple client handling

  https://youtu.be/gLfuZrrfKes?si=Kpqo_hdpJ4wl9ysf

- client server example
  https://www.cs.unc.edu/~jbs/resources/java/java_client_server1/