CSE 4300/5307 Group Project

Arman Chowdhury
Swamy Narayan Jignaas Pattipati
Ya-Sine Agrignan

01 Proposal

Multi-threaded CLI Chat Server

Pitch

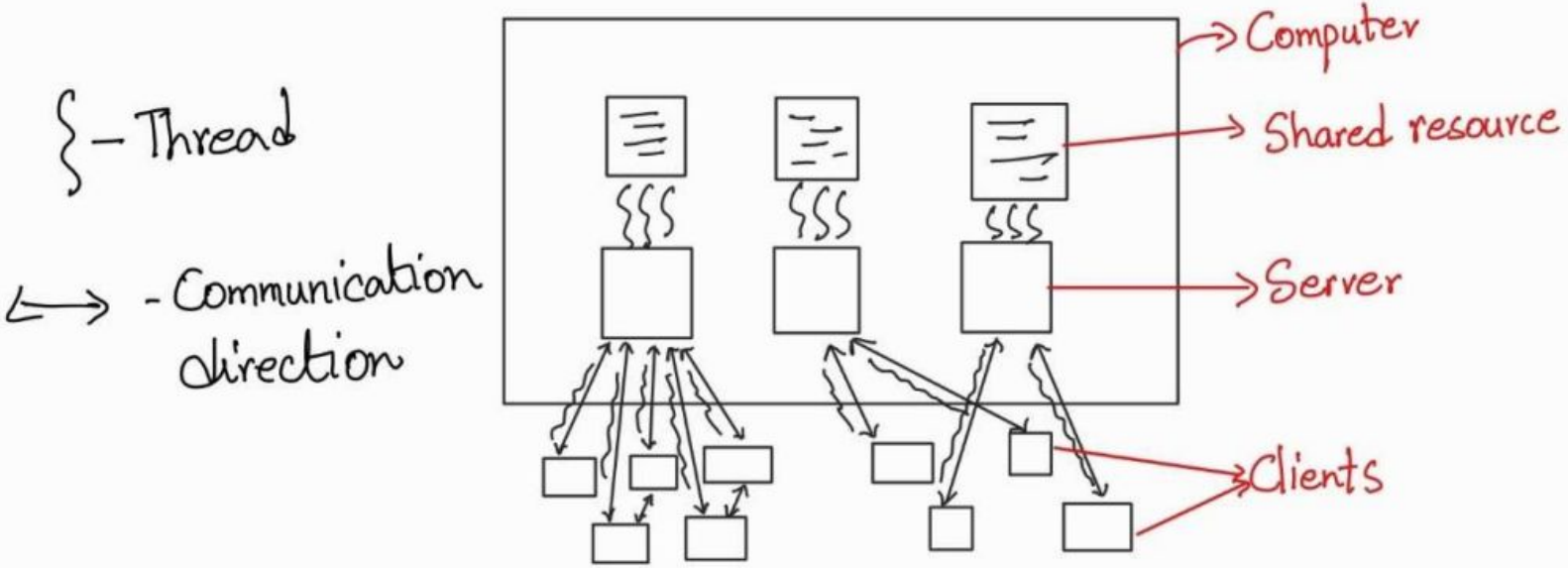
Our proposal was to **design a functional CLI application that creates group chat servers and allows for multiple clients to connect to it**.

Functionally, **each different thread serves each client**, which stores all the history of the chats within a file. Every thread should be able to access this file and append any messages sent by their respective client, giving the application both storage and memory capacities.

We rely on **threads** for this implementation of a chat server in order to maximize the ability of the application to **serve multiple clients simultaneously.** Each threads by necessity will use synchronization primitives to eliminate potential race conditions in appending any messages to the shared resource.

Within the scope of our designed server, the **user will be able to create multiple chat servers** that are analogous to group chat applications. In the application, **clients will be able to broadcast the message** to every client on the server **and individually message** other clients with simple commands.

High Level Overview



Please Note: We are *not* art students.

Takeaways

Many chat servers can be created on one computer, and many clients can connect to the server of their interest

Clients send messages to servers, and the server sends the message to every other client

Every client is served with a different thread, which will take care of reading the packet and appending the message to the shared resource. Other threads can send this shared message to their clients upon the request of the server

If a client decides to send a personal message to another client, the server will only transfer the message to the requested client

02

Code Explanation

Using Java Socket Programming

Code Snippet Section 1

```
public class Server{
    public Server(int port) {
        this.port = port;
        sdf = new SimpleDateFormat("HH:mm:ss");
        al = new ArrayList<ClientThread>();
    }
    public void start(){
    //creates tcp socket
    //handles client threads
    }
}
```

```
public class ClientThread{
    ClientThread(Socket socket) {
        // class to handle client threads
        id = ++uniqueId;
        this.socket = socket;
    }
}
```

- Each server handles the client separately.
- Every client is stored in a thread.

- Each thread is associated with a unique Id and is stored in array list

Code Snippet Section 2

```java
public class Client{
    Client(String server, int port, String username) {
        this.server = server;
        this.port = port;
        this.username = username;
    }
}
```

```java
public class MessageObject implements Serializable {
    MessageObject(int type, String message) {
        this.type = type;
        this.message = message;
    }
}
```

- Client takes server IPaddr, port and username as input
- Username is sent to server through IO stream
- Client can join and leave a chat server

- MessageObject contains message and type of message from client and is passed as a serializable object through socket
- Message types are Logout , active users and message

Please note:

Synchronized block is used to send messages through sockets and messages are written to history inside this block to eliminate race conditions

03 Live Demo

(please hold)

Thank you!

Please raise your hand if you have any further questions!