OS HACKATHON-2

Problem Statement:

Chat Applications

• Application: Applications like WhatsApp or Slack use IPC mechanisms for message exchange between processes handling user input, storage, and network communication.

• Solution: Implement message queues or shared memory for efficient data sharing between processes.

Server Code

The server listens for client connections and processes their messages.

```java
import java.io.*;

import java.net.*;

import java.util.concurrent.*;


public class ChatServer {

    private static final int PORT = 12345; // The port to listen for connections

    private static final int MAX_MESSAGES = 10; // Message queue size limit

    private static BlockingQueue<String> messageQueue = new
LinkedBlockingQueue<>(MAX_MESSAGES);


    public static void main(String[] args) {

        try (ServerSocket serverSocket = new ServerSocket(PORT)) {

            System.out.println("Server is running, waiting for client connections...");


            // Continuously accept client connections

            while (true) {

                Socket clientSocket = serverSocket.accept();

                System.out.println("Client connected: " + clientSocket.getInetAddress());
                // Start a new thread for each client
                new Thread(new ClientHandler(clientSocket)).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Handle communication with a single client
    static class ClientHandler implements Runnable {
        private final Socket clientSocket;

        public ClientHandler(Socket clientSocket) {
            this.clientSocket = clientSocket;
        }

        @Override
        public void run() {
            try (BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                 PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) {

                String message;
                while ((message = in.readLine()) != null) {
                    if ("exit".equalsIgnoreCase(message)) {
                        System.out.println("Client disconnected.");
                        break;
                    }
                    messageQueue.put(message); // Add the message to the queue
                    System.out.println("Received message: " + message);
                    out.println("Server: " + message); // Send a confirmation to the client
                }
            } catch (IOException | InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Client Code

The client connects to the server and exchanges messages.

```java
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class ChatClient {
    private static final String SERVER_IP = "192.168.150.122"; // Replace with the server's IP address
    private static final int SERVER_PORT = 12345; // The port the server is listening on

    public static void main(String[] args) {
        try (Socket socket = new Socket(SERVER_IP, SERVER_PORT);
             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {

            Scanner scanner = new Scanner(System.in);

            System.out.println("Connected to the server!");
            System.out.println("Type 'exit' to quit.");

            // Continuously read user input and send to the server
            while (true) {
                System.out.print("Enter message: ");
                String message = scanner.nextLine();

                out.println(message); // Send the message to the server

                if ("exit".equalsIgnoreCase(message)) {
                    System.out.println("Exiting chat...");
                    break;
                }

                // Receive the response from the server
                String response = in.readLine();
                System.out.println("Server response: " + response);
            }

            scanner.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Steps to Run the Code:

1. **Run the Server Code:**

   - Execute the `ChatServer` program on one system (or instance) to start the server.

   - The server will wait for client connections on port `12345`.

2. **Run the Client Code:**

   - Execute the `ChatClient` program on a different system.

   - Replace `SERVER_IP` with the IP address of the server system.

   - Connects to the server and allows sending messages.

3. **Test Communication:**

   - Enter messages on the client console.

   - The server will display received messages and echo them back.

4. **Stop Communication:**

   - Type `exit` on the client to disconnect.

This setup enables basic communication between multiple clients and the server over a network.