|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

Experiment No. 10

|  |  |
| --- | --- |
| Semester | B.E. Semester VIII – Computer Engineering |
| Subject | Distributed Computing Lab |
| Subject Professor In-charge | Dr. Umesh Kulkarni |
| Assisting Professor | Prof. Prakash Parmar |
| Academic Year | 2024-25 |

|  |  |
| --- | --- |
| Student Name | Deep Salunkhe |
| Roll Number | 21102A0014 |

**Title:**  Multi-Threaded Distributed Application in Java

**1. Introduction**

A distributed application is a system where multiple clients interact with a server over a network. In this lab, we develop a multi-threaded client-server application using Java, where the server handles multiple clients concurrently.

**2. Objectives**

* To understand the concept of distributed computing.
* To implement a multi-threaded server that can handle multiple clients.
* To develop a client program that communicates with the server.

**3. Theory**

**3.1 Distributed Computing**

Distributed computing involves multiple computers working together to solve a problem or provide a service. In a client-server model, clients request services from a central server, which processes these requests and sends responses.

**3.2 Multi-Threading in Java**

Multi-threading allows a program to execute multiple tasks concurrently. In a multi-threaded server, each client connection is handled in a separate thread, ensuring simultaneous communication with multiple clients.

**3.3 Sockets in Java**

Sockets enable network communication between processes. Java provides ServerSocket for creating server connections and Socket for client connections. The server listens on a port and accepts incoming connections.

**4. Implementation Details**

**4.1 Server Implementation**

* A ServerSocket is created to listen for incoming client connections.
* When a client connects, a new thread (ClientHandler) is spawned to handle communication.
* The server echoes messages received from the client.

**4.2 Client Implementation**

* A Socket is used to connect to the server.
* The client sends messages to the server and receives responses.
* The client terminates communication when the user types "exit".

**5. Execution Steps**

1. **Compile the Java Files**

javac MultiThreadedServer.java Client.java

1. **Run the Server**

java MultiThreadedServer

1. **Run Multiple Clients**

java Client

1. **Interact with the Server**
   * Clients can send messages.
   * The server responds with an echo message.
   * Typing "exit" disconnects the client.

**6. Observations and Results**

* The server successfully handles multiple clients simultaneously.
* Each client receives responses independently.
* Communication is maintained until the client chooses to disconnect.

**7. Conclusion**

This lab demonstrates the implementation of a distributed application using Java sockets and multi-threading. The server efficiently manages multiple clients using separate threads, showcasing the power of concurrent programming.

**Code:**

**import** **java.io.\***;

**import** **java.net.\***;

**class** ClientHandler **extends** Thread {

**private** **Socket** socket;

**public** ClientHandler(**Socket** socket) {

**this**.socket **=** socket;

    }

    @**Override**

**public** **void** run() {

**try** (**BufferedReader** in **=** **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

**PrintWriter** out **=** **new** PrintWriter(socket.getOutputStream(), **true**)) {

            out.println("Connected to the server. Type 'exit' to disconnect.");

**String** message;

**while** ((message **=** in.readLine()) **!=** **null**) {

**if** ("exit".equalsIgnoreCase(message)) {

                    out.println("Goodbye!");

**break**;

                }

                System.out.println("Client: " **+** message);

                out.println("Server Echo: " **+** message);

            }

        } **catch** (**IOException** e) {

            System.out.println("Client disconnected.");

        } **finally** {

**try** {

                socket.close();

            } **catch** (**IOException** e) {

                e.printStackTrace();

            }

        }

    }

}

**public** **class** MultiThreadedServer {

**public** **static** **void** main(**String**[] args) {

**int** port **=** 5000;

**try** (**ServerSocket** serverSocket **=** **new** ServerSocket(port)) {

            System.out.println("Server is running on port " **+** port);

**while** (**true**) {

**Socket** clientSocket **=** serverSocket.accept();

                System.out.println("New client connected");

**new** ClientHandler(clientSocket).start();

            }

        } **catch** (**IOException** e) {

            e.printStackTrace();

        }

    }

}

**import** **java.io.\***;

**import** **java.net.\***;

**public** **class** Client {

**public** **static** **void** main(**String**[] args) {

**String** host **=** "localhost";

**int** port **=** 5000;

**try** (**Socket** socket **=** **new** Socket(host, port);

**BufferedReader** in **=** **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

**PrintWriter** out **=** **new** PrintWriter(socket.getOutputStream(), **true**);

**BufferedReader** userInput **=** **new** BufferedReader(**new** InputStreamReader(System.in))) {

            System.out.println("Connected to server");

            System.out.println(in.readLine());

**String** message;

**while** (**true**) {

                System.out.print("You: ");

                message **=** userInput.readLine();

                out.println(message);

**if** ("exit".equalsIgnoreCase(message)) **break**;

                System.out.println("Server: " **+** in.readLine());

            }

        } **catch** (**IOException** e) {

            e.printStackTrace();

        }

    }

}

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**