Experiment No. 09

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 Server for Distributed Computing

---

1. Introduction

Distributed computing involves multiple interconnected nodes working together to solve computational problems. A critical aspect of distributed systems is efficient task handling across multiple nodes. This report presents a multi-threaded server that manages multiple client connections simultaneously, ensuring responsiveness and scalability.

2. Objective

The primary objective of this lab work is to implement a multi-threaded server capable of handling multiple clients concurrently. This demonstrates fundamental concepts of distributed computing such as concurrency, parallelism, synchronization, and efficient resource management.

3. Theoretical Background

3.1 Multi-Threading in Distributed Systems

Multi-threading is a technique where multiple threads run concurrently within a single process. This approach enables efficient CPU utilization and enhances performance by handling multiple requests simultaneously instead of sequentially processing them.

3.2 Client-Server Model

The client-server model is a fundamental architecture in distributed computing. The server

listens for incoming connections and processes client requests, while clients send requests and receive responses.

3.3 Thread Pooling

Thread pooling is a technique used to manage a fixed number of worker threads that handle tasks. Instead of creating a new thread for each request, which is inefficient, a thread pool reuses a limited number of threads, improving performance and reducing system overhead.

4. Implementation Details

4.1 Server-Side Functionality

- The server listens on a designated port for incoming client connections.
- A thread pool is used to efficiently manage concurrent client requests.
- Each client request is assigned to a separate worker thread for processing.
- The server echoes back the received messages and disconnects when the client sends an exit command.

4.2 Client-Side Functionality

- The client establishes a connection with the server.
- It sends user-input messages to the server.
- It continuously listens for responses from the server.
- The connection terminates when the user types 'exit'.

5. Advantages of Multi-Threaded Servers in Distributed Computing

- **Concurrency**: Handles multiple clients simultaneously without blocking operations.
- **Efficiency**: Thread pooling reduces the overhead of thread creation and destruction.
- **Scalability**: Can accommodate increasing client requests without significant performance degradation.
- **Responsiveness**: Ensures that one slow client does not affect the performance of others.

6. Real-World Applications

- Web servers handling multiple HTTP requests.
- Chat applications managing real-time user interactions.

- Database servers processing multiple queries in parallel.
- Cloud computing environments distributing workloads efficiently.

7. Conclusion

This lab work demonstrates the principles of distributed computing through a multi-threaded server-client architecture. By employing multi-threading and thread pooling, the system efficiently handles concurrent requests, showcasing an essential concept in distributed systems and network programming.

Code:

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

public class MultiThreadedServer {
    private static final int PORT = 5000;
    private static final int THREAD_POOL_SIZE = 5;

    public static void main(String[] args) {
        ExecutorService threadPool = Executors.newFixedThreadPool(THREAD_POOL_SIZE);

        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server started on port " + PORT);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("New client connected: " +
clientSocket.getInetAddress());
                threadPool.execute(new ClientHandler(clientSocket));
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            threadPool.shutdown();
        }
    }
}

class ClientHandler implements Runnable {
    private final Socket clientSocket;

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

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

            out.println("Welcome to the server. Type 'exit' to disconnect.");
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                System.out.println("Received: " + inputLine);
                if ("exit".equalsIgnoreCase(inputLine)) {
                    out.println("Goodbye!");
                    break;
                }
                out.println("Echo: " + inputLine);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                clientSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            System.out.println("Client disconnected.");
        }
    }
}

class Client {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 5000);
             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.");
            new Thread(() -> {
                try {
                    String serverMessage;
                    while ((serverMessage = in.readLine()) != null) {
                        System.out.println("Server: " + serverMessage);
                    }
                } catch (IOException e) {
                    e.printStackTrace();
```

```
                }
            }).start();

            String userMessage;
            while ((userMessage = userInput.readLine()) != null) {
                out.println(userMessage);
                if ("exit".equalsIgnoreCase(userMessage)) {
                    break;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
PS E:\GIt\Sem-8\DC\Lab9> java MultiThreadedServer
Server started on port 5000
New client connected: /127.0.0.1
New client connected: /127.0.0.1
New client connected: /127.0.0.1
New client connected: /127.0.0.1
Received: exit
Client disconnected.
```

```
PS E:\GIt\Sem-8\DC\Lab9> java Client
Connected to server.
Server: Welcome to the server. Type 'exit' to disconnect.
```

```
PS E:\GIt\Sem-8\DC\Lab9> java Client
Connected to server.
Server: Welcome to the server. Type 'exit' to disconnect.
```

```
PS E:\GIt\Sem-8\DC\Lab9> java Client
Connected to server.
Server: Welcome to the server. Type 'exit' to disconnect.
```

```
PS E:\GIt\Sem-8\DC\Lab9> java Client
Connected to server.
Server: Welcome to the server. Type 'exit' to disconnect.
exit
Server: Goodbye!
java.io.IOException: Stream closed
        at java.base/java.io.BufferedReader.ensureOpen(BufferedR
eader.java:121)
```