Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 03

Course Title: Computer Networking Lab

Course Code: CSE-304 Section:221-D21

Lab Experiment Name: Implementation of socket programming using threading.

<u>Student Details</u>

	Name	ID
1.	Masum Hossain	221902164

Lab Date : 29/12/2024
Submission Date : 15/12/2024
Course Teacher's Name : Md. Saiful Islam Bhuiyan

<u>Lab Report Status</u>

Marks: ………………………………… Signature:.....................
Comments:.. Date:..............................

1. TITLE OF THE LAB REPORT EXPERIMENT

Implementation of Socket Programming Using Threading.

2. OBJECTIVES

This lab experiment involves creating a Java program to perform server and client communication for different scenarios.Here we will implement the client limit for 4 users only instead of unlimited users. The specific objectives of this lab experiment are:

- To understand the principles of socket programming and multithreading.
- To implement a Java-based server-client application that supports multiple client connections.
- To learn how to handle client requests and process data on a multithreaded server.
- To gain practical experience in data communication over TCP sockets.

3. ANALYSIS

This experiment involves implementing a multithreaded server-client application using Java. The server accepts multiple client connections, processes data, and communicates results back to the clients. The key features and processes include:

Server Side:

1. Listens for client connections on a specified port.
2. Accepts up to four clients and assigns each client a separate thread for interaction.
3. Processes sentences sent by clients by capitalizing the middle letters of words.

Client Side:

1. Connects to the server on the specified port.
2. Sends user-provided sentences to the server and receives the processed results.
3. Allows users to terminate the session by typing "Exit."

4. IMPLEMENTATION

The program is implemented in Java, using a simple class structure for both client side and server side.ClientHandler class is used to limit the number of clients.

Server Side Code

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

public class ServerThread {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(5000);
        System.out.println("Server is running on port 5000...");

        int client = 0;

        while (client < 4) {
            Socket clientSocket = serverSocket.accept();
            client++;
            System.out.println("Client " + client + " connected: " + clientSocket);

            DataInputStream dis = new DataInputStream(clientSocket.getInputStream());
            DataOutputStream dos = new DataOutputStream(clientSocket.getOutputStream());

            Thread clientThread = new ClientHandler(clientSocket, dis, dos, client);
            clientThread.start();
        }

        System.out.println("Server lifetime ended. No more clients accepted.");
        serverSocket.close();
    }
}

class ClientHandler extends Thread {
    private final Socket socket;
    private final DataInputStream dis;
    private final DataOutputStream dos;
    private final int client;

    public ClientHandler(Socket socket, DataInputStream dis, DataOutputStream dos, int client)
{
        this.socket = socket;
        this.dis = dis;
        this.dos = dos;
        this.client = client;
    }
```

```
@Override
public void run() {
    try {
        int sentenceCount = 0;

        while (sentenceCount < 3) {
            dos.writeUTF("Client " + client + ": Send a sentence (or type 'Exit' to quit): ");
            String received = dis.readUTF();

            if (received.equalsIgnoreCase("Exit")) {
                System.out.println("Client " + client + " disconnected.");
                break;
            }

            String convertedSen = capitalizeMiddleLetters(received);
            dos.writeUTF("Processed: " + convertedSen);
            System.out.println("Processed for Client " + client + ": " + convertedSen);
            sentenceCount++;
        }

        System.out.println("Client " + client + " session ended.");
        socket.close();
        dis.close();
        dos.close();
    } catch (IOException e) {
        System.err.println("Error handling client " + client + ": " + e.getMessage());
    }
}

private String capitalizeMiddleLetters(String sentence) {
    String[] words = sentence.split(" ");
    StringBuilder result = new StringBuilder();

    for (String word : words) {
        if (word.length() == 0) continue;

        int mid = word.length() / 2;
        char middleChar = Character.toUpperCase(word.charAt(mid));

        String transformedWord = word.substring(0, mid) + middleChar + word.substring(mid
+ 1);
        result.append(transformedWord).append(" ");
    }

    return result.toString().trim();
    }
}
```

Client Side Code

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

public class ClientThread {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 5000);
            System.out.println("Connected to server.");

            DataOutputStream dos = new DataOutputStream(clientSocket.getOutputStream());
            DataInputStream dis = new DataInputStream(clientSocket.getInputStream());
            Scanner scanner = new Scanner(System.in);

            while (true) {
                String serverMessage = dis.readUTF();
                System.out.println(serverMessage);

                System.out.print("Please Enter any sentence: ");
                String input = scanner.nextLine();
                dos.writeUTF(input);

                if (input.equalsIgnoreCase("Exit")) {
                    System.out.println("Exiting session.");
                    break;
                }
                String response = dis.readUTF();
                System.out.println("Server Response: " + response);
            }

            clientSocket.close();
            dis.close();
            dos.close();
        } catch (IOException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

OUTPUT

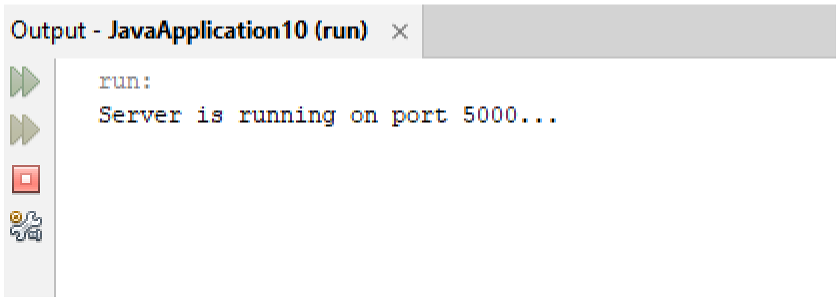
The console output will display the result based on the user's input. For example, if the user queries "," the output will look like this:

Output for Server Side

```
Server is running on port 5000...
Client 1 connected: Socket[addr=/127.0.0.1,port=50689,localport=5000]
Processed for Client 1: tEst Middle woRds
Client 1 session ended.
```

Output for Client Side

```
Connected to server.
Client 1: Send a sentence (or type 'Exit' to quit):
Please Enter any sentence: test middle words
Server Response: Processed: tEst Middle woRds
```



```
run:
Server is running on port 5000...
```

Fig 01: Showing the server is running with the desired port.

6. ANALYSIS AND DISCUSSION

This lab experiment successfully met its objectives by implementing a multithreaded server-client application in Java. Through the practical demonstration of socket programming and threading, it emphasized the importance of designing scalable and interactive networked applications. The experiment also reinforced the significance of concurrent processing in modern networking solutions, paving the way for further exploration of advanced topics in this domain.

The implementation of socket programming using threading provided valuable insights into building concurrent applications. The server's multithreaded design allowed simultaneous handling of multiple clients, demonstrating how threading enhances scalability and responsiveness. Each client was assigned a separate thread, ensuring individualized processing of sentences and seamless interaction. The communication between the client and server, facilitated by Java’s I/O streams, reinforced understanding of data exchange mechanisms in networked systems. Additionally, the program’s functionality to process sentences by capitalizing the middle letters of words showcased efficient string manipulation. Overall, this experiment not only strengthened foundational knowledge of networking and multithreading but also highlighted the practical applications of these concepts in real-world scenarios.