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

Faculty of Sciences and Engineering
Semester: (Spring , Year:2024), B.Sc. in CSE (Day)

LAB REPORT NO #03

Course Title: Computer Networking Lab

Course Code: CSE - 312 Section: 213_D5

Lab Experiment Name: Implementation of socket programming using threading .

<u>Student Details</u>

Name	ID
MD Dulal Hossain	213902116

Lab Date	:	**11– 05 - 2024**
Submission Date	:	**18– 05 - 2024**
Course Teacher's Name	:	**MS. Rusmita Halim Chaity**

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status
Marks: ………………………………… Signature:.....................
Comments:... Date:..............................

1. TITLE OF THE LAB EXPERIMENT

Mathematical operations are very useful to be implemented using TCP socket programming. Create two processes, a server and a client using JAVA codes. The client will send two separate integer values and a mathematical operator to the server. The server receives these integers and a mathematical operator, then performs a mathematical operation based on the user input. The server then sends this answer to the client, who then displays it. The client sends requests as many times as he wishes. However, The server can serve at most 5 clients in its lifetime. The individual client ends its connection by saying “ENDS”.

2. OBJECTIVES/AIM

- If the client sends 10, 20, and Sum, the server sends 30 to the client.
- If the client sends 20, 5, and Subtract, the server sends 15 to the client.
- If the client sends 20, 5, and Multiplication, the server sends 100 to the client.
- If the client sends 20, 5, and Division, the server sends 4 to the client.
- If the client sends 16, 3, and Modules, the server sends 1 to the client.

3. PROCEDURE

Algorithms :

MathServer Algorithm:

```
1. Setup Server Socket: Create a ServerSocket object with port 5000.
2. Print Server Information: Print server port and status messages.
3. Initialize Client Count: Set clientCount to 0 to track the number of clients served.
4. Accept Client Connections: Enter a loop to accept client connections until 5 clients are served.
        Accept a new client connection using serverSocket.accept().
        Print a message indicating a new client connection.
        Create input and output streams for the client.
        Create a new ClientHandler thread passing client socket, input stream, and output stream.
        Start the thread.
        Increment clientCount.
5. Close Server Socket: Close the server socket after serving 5 clients.
```

MathClient Algorithm:

```
1. Connect to Server: Attempt to connect to the server at localhost and port 5000.
2. Print Connection Information: Print client port and connection status.
3. Setup I/O Streams: Create input and output streams for communication with the server.
4. Main Interaction Loop: Enter a loop for interacting with  server until the client decides to end.
        Prompt the user to enter two integers and an operation.
        Send integers and operation to the server.
        If the operation is "ENDS", close the connection and break the loop.
        Receive and print the result from the server.
5. Close Streams and Socket: Close input stream, output stream, scanner, and client socket.
```

ClientHandler Algorithm:

```
1. Setup Thread: Initialize the thread with the client socket, input stream, and output stream.
2. Main Thread Loop: Enter a loop to handle client requests until the client ends the connection.
        Read two integers and an operation from the client.
        Calculate the result based on the operation.
        Send the result back to the client.
3. Handle Client Disconnect: If the client sends "ENDS", close the client socket and end the thread.
4. Cleanup Resources: Close input and output streams when the thread ends.
```

4. IMPLEMENTATION

Source Code :

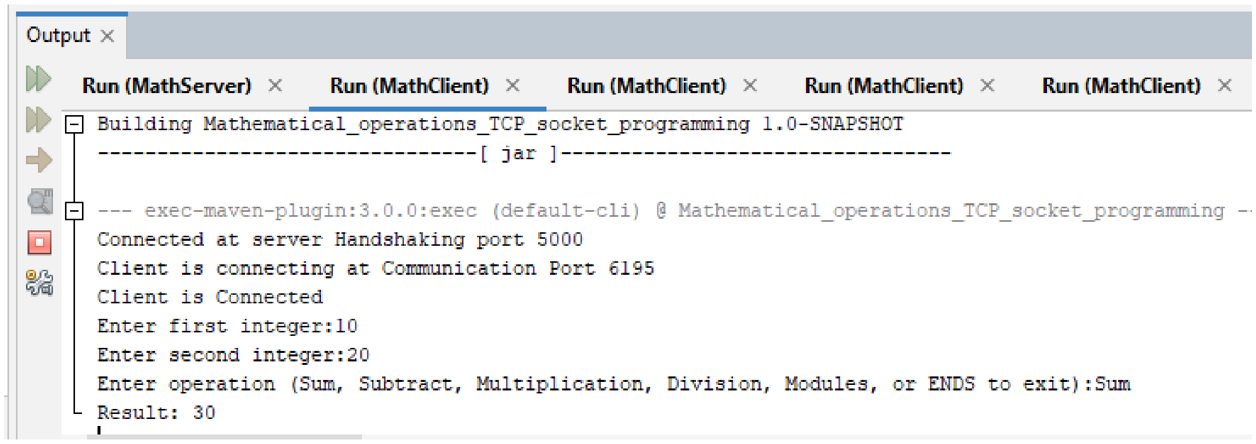
//MathServer Part

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

public class MathServer {
    public static void main(String args[]) throws IOException {
        ServerSocket serverSocket = new ServerSocket(5000);
        System.out.println("Server connected at " + serverSocket.getLocalPort());
        System.out.println("Server is connecting\n");
        System.out.println("Waiting for the client\n");

        int clientCount = 0;
        while (clientCount < 5) {
            Socket clientSocket = serverSocket.accept();
            System.out.println("A new client is connected " + clientSocket);

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

            System.out.println("A new thread is assigning");
            Thread clientHandlerThread = new ClientHandler(clientSocket, dis, dos);
            clientHandlerThread.start();

            clientCount++;
        }
        serverSocket.close();
    }
}
```

```
// MathClient part
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class MathClient {
    public static void main(String args[]) throws IOException {
        try {
            Socket clientSocket = new Socket("localhost", 5000);
            System.out.println("Connected at server Handshaking port " + clientSocket.getPort());
            System.out.println("Client is connecting at Communication Port " +
clientSocket.getLocalPort());
            System.out.println("Client is Connected");
            Scanner scanner = new Scanner(System.in);
            DataOutputStream dos = new DataOutputStream(clientSocket.getOutputStream());
            DataInputStream dis = new DataInputStream(clientSocket.getInputStream());
            while (true) {
                System.out.println("Enter first integer:");
                int num1 = scanner.nextInt();
                System.out.println("Enter second integer:");
                int num2 = scanner.nextInt();
                System.out.println("Enter operation (Sum, Subtract, Multiplication, Division,
Modules, or ENDS to exit):");
                String operator = scanner.next();
                dos.writeInt(num1);
                dos.writeInt(num2);
                dos.writeUTF(operator);
                if (operator.equals("ENDS")) {
                    System.out.println("Closing the connection " + clientSocket);
                    clientSocket.close();
                    System.out.println("Connection Closed");
                    break;
                }

                int result = dis.readInt();
                System.out.println("Result: " + result);
            }

            dos.close();
            dis.close();
            scanner.close();
            clientSocket.close();
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
}
```

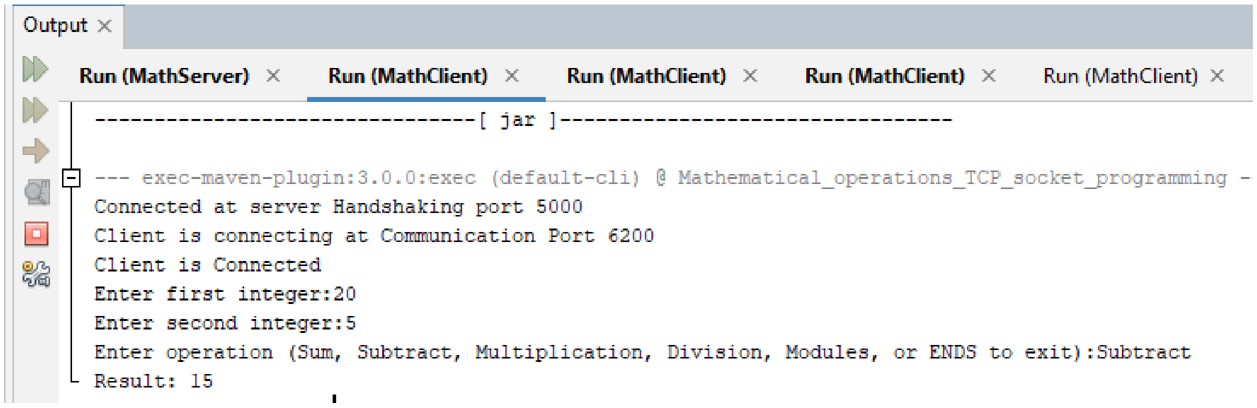
```
// ClientHandler Part
class ClientHandler extends Thread {
   final Socket clientSocket;
   final DataInputStream dis;
   final DataOutputStream dos;
   public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos) {
      this.clientSocket = s;
      this.dis = dis;
      this.dos = dos;    }
   public void run() {
      try {
         while (true) {
            int num1 = dis.readInt(); // Read first integer from client
            int num2 = dis.readInt(); // Read second integer from client
            String operator = dis.readUTF(); // Read operator from client
            int result = 0;
            switch (operator) {
               case "Sum":
                  result = num1 + num2;
                  break;
               case "Subtract":
                  result = num1 - num2;
                  break;
               case "Multiplication":
                  result = num1 * num2;
                  break;
               case "Division":
                  result = num1 / num2;
                  break;
               case "Modules":
                  result = num1 % num2;
                  break;
               case "ENDS":
                  System.out.println("Client " + this.clientSocket + " has ended the connection");
                  this.clientSocket.close();
                  return;                    default:
                  dos.writeUTF("Invalid operator");
                  continue;              }
            dos.writeInt(result);              }
      } catch (IOException e) {
         e.printStackTrace();
      } finally {
         try {
            this.dis.close();
            this.dos.close();
         } catch (IOException e) {
            e.printStackTrace();          }        }    }
}
```

5. TEST RESULT / OUTPUT



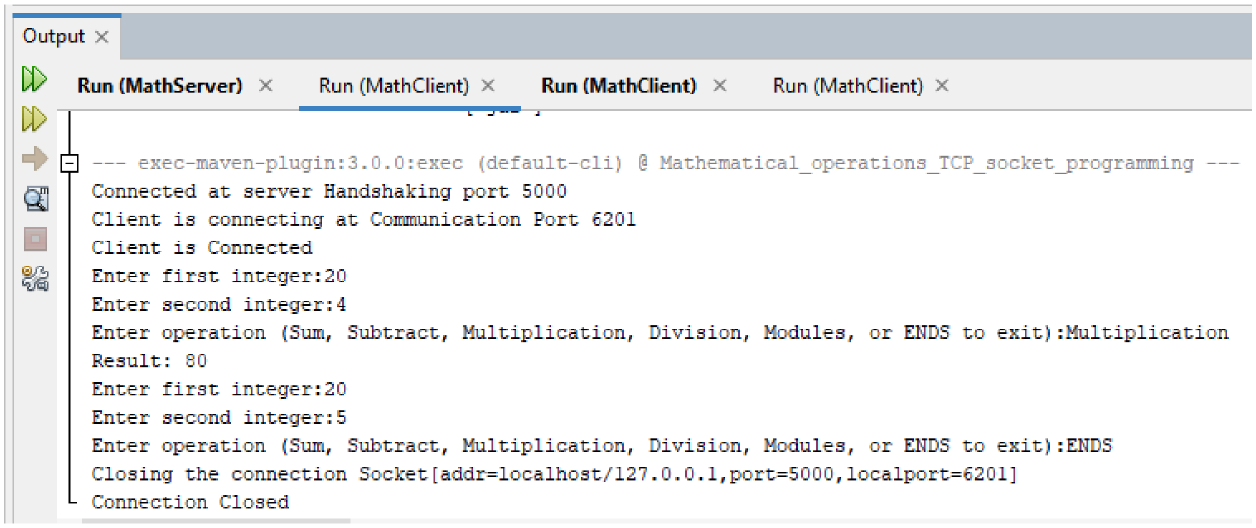
```
Building Mathematical_operations_TCP_socket_programming 1.0-SNAPSHOT
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Mathematical_operations_TCP_socket_programming -
Connected at server Handshaking port 5000
Client is connecting at Communication Port 6195
Client is Connected
Enter first integer:10
Enter second integer:20
Enter operation (Sum, Subtract, Multiplication, Division, Modules, or ENDS to exit):Sum
Result: 30
```

Figure 1 : Sum Result Successfully Show.



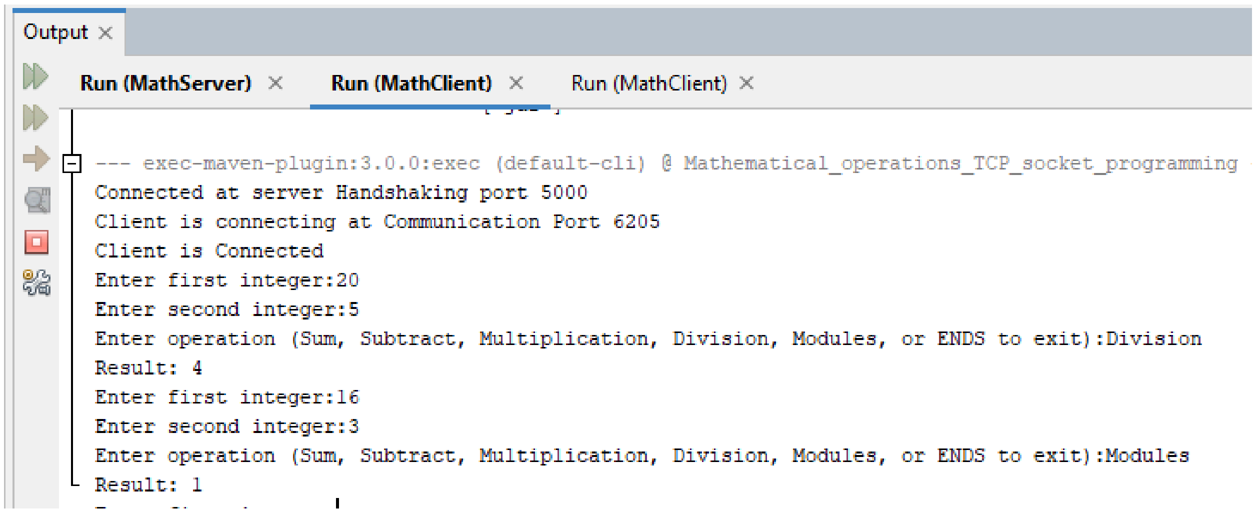
```
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Mathematical_operations_TCP_socket_programming -
Connected at server Handshaking port 5000
Client is connecting at Communication Port 6200
Client is Connected
Enter first integer:20
Enter second integer:5
Enter operation (Sum, Subtract, Multiplication, Division, Modules, or ENDS to exit):Subtract
Result: 15
```

Figure 2 : Subtract Result Successfully Show.



```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Mathematical_operations_TCP_socket_programming ---
Connected at server Handshaking port 5000
Client is connecting at Communication Port 6201
Client is Connected
Enter first integer:20
Enter second integer:4
Enter operation (Sum, Subtract, Multiplication, Division, Modules, or ENDS to exit):Multiplication
Result: 80
Enter first integer:20
Enter second integer:5
Enter operation (Sum, Subtract, Multiplication, Division, Modules, or ENDS to exit):ENDS
Closing the connection Socket[addr=localhost/127.0.0.1,port=5000,localport=6201]
Connection Closed
```

Figure 3 : Multiplication Result Successfully Show & Exit or Connection Close work Successfully.



```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Mathematical_operations_TCP_socket_programming
Connected at server Handshaking port 5000
Client is connecting at Communication Port 6205
Client is Connected
Enter first integer:20
Enter second integer:5
Enter operation (Sum, Subtract, Multiplication, Division, Modules, or ENDS to exit):Division
Result: 4
Enter first integer:16
Enter second integer:3
Enter operation (Sum, Subtract, Multiplication, Division, Modules, or ENDS to exit):Modules
Result: 1
```

Figure 4 : Division And Modules Result Successfully Show.

Explain Output :

1. Connection Setup:
   The client connects to the server at port 5000.
   Communication occurs through port 6195, 6200, 6201 & 6205.
2. Interaction Flow:
   The user enters two integers.
   Chooses an operation among Sum, Subtract, Multiplication, Division, Modulus, or ENDS (to exit).
   Receives the result of the chosen mathematical operation performed on the entered integers.
3. Sample Operations:
   Sum: 10 + 20 = 30
   Subtraction: 20 - 5 = 5
   Multiplication: 20 * 4 = 80
   Division: 20 / 5 = 4
   Modulus: 16 % 3 = 1
4. Connection Closure:
   The user can exit by entering “ENDS” as the operation.

6. ANALYSIS AND DISCUSSION

The provided code implements a basic client-server architecture for arithmetic operations. The server listens for client connections, assigns each to a separate thread, and handles arithmetic calculations based on client requests. Clients connect to the server, send integer inputs and operation requests, and receive results. The code demonstrates fundamental socket programming concepts, including multithreading for concurrent client handling. However, it lacks advanced features like authentication and encryption, which are crucial for secure communication. Moreover, error handling could be improved to enhance the robustness and reliability of the application. Overall, the code serves as a foundational example for networked applications but requires further refinement for real-world deployment. So we say that our lab report work 100% properly.