



Green University of Bangladesh
Department of Computer Science and Engineering(CSE)
Faculty of Sciences and Engineering
Semester: (Fall, Year:2022), B.Sc. in CSE (Day)

LAB REPORT NO : 03

Course Title: Computer Networking Lab

Course Code: CSE 312 Section: DB

Lab Experiment Name:
Implementation of Socket Programming Using Threading to Perform Basic Arithmetic Operations

Student Details

Name		ID
1.	Hamad Ismail	201902046

Lab Date : 17/12/2022
Submission Date : 28/12/2022
Course Teacher's Name : Rusmita Halim Chaity

1. TITLE OF THE LAB EXPERIMENT

Implementation of Socket Programming Using Threading for Basic Arithmetic Operations.

2. OBJECTIVES/AIM

The main problem of the simple two way socket programming is that it can not handle multiple client requests at the same time. Server can only provide service to that client that has come first to connect with the server. The other clients can not connect with that server.

To resolve this problem, the server opens a different thread for each client and every client communicates with the server using that thread. In this lab, we will create a simple Date-Time server for handling multiple client requests at the same time.

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.

3. PROCEDURE / ANALYSIS / DESIGN

In the multithreaded socket programming, at first client creates a connection with server through serversocket. Then Server creates a different thread namely ClientHandler for each client. In the ClientHandler class, the server passes communication port, inputStream and outputStream as parameters.

Then the Client conducts all types of communications with the server through the ClientHandler class. And create an Arithmetic class to perform all of the arithmetic operations. The detailed step-by-step procedures are discussed as follows.

3.1 Server Side Programming

On the server side, we need to create two classes. One is Server class and another one is ClientHandler Class.

3.1.1 Server class

The steps involved on server side are similar to the article Socket Programming in Java with a slight change to create the thread object after obtaining the streams and port number.

- **Establishing the Connection:** Server socket object is initialized and inside a while loop a socket object continuously accepts incoming connection.
- **Obtaining the Streams:** The inputStream object and outputStream object is extracted from the current requests' socket object.
- **Creating a handler object:** After obtaining the streams and port number, a new clientHandler object (the above class) is created with these parameters.
- **Invoking the start() method:** The start() method is invoked on this newly created thread object.

3.1.2 ClientHandler class

As we will be using separate threads for each request, let's understand the working and implementation of the ClientHandler class extending Threads. An object of this class will be instantiated each time a request comes.

- First of all this class extends Thread so that its objects assume all properties of Threads.
- Secondly, the constructor of this class takes three parameters, which can uniquely identify any incoming request, i.e., a Socket, a DataInputStream to read from, and a DataOutputStream to write to. Whenever we receive any request from the client, the server extracts its port number, the DataInputStream object, and DataOutputStream object and creates a new thread object of this class and invokes the start() method on it.
- Inside the run() method of this class, it performs three operations: request the user to specify which arithmetics operations are needed, read the answer from the input stream object and accordingly write the output on the output stream object.

3.2 Client Side Programming

Client-side programming is similar to in general socket programming program with the following steps-

- **Establish a Socket Connection:** Create a Socket object which takes IP address and port number as input.
- **Communication:** Client can Send the data to the server side using writeUTF() function and Client can read any data using readUTF() function.
- **Closing the Connection:** Client can continue its communication with the server until client sends "ENDS".

3.3 Arithmetic Class

Performs all of the arithmetic operations here. Client class takes input from the user and passes those input via some methods to the Arithmetic class. Then the Arithmetic class performs all arithmetic operations and returns the result when its method is called in the ClientHandler class.

4. IMPLEMENTATION

4.1 Server Class

```
package javaSocket;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
```

```

import java.net.Socket;

public class Server {
    public static void main(String args[]) throws IOException {
        ServerSocket handshake = new ServerSocket (5000);
        System.out.println("Server connected at " + handshake.getLocalPort());
        System.out.println("Server is connecting\n");
        System.out.println("Wait for the client\n");
        while(true){
            Socket com_socket = handshake.accept();
            System.out.println("A new client is connected "+ com_socket);
            DataOutputStream dos = new DataOutputStream(com_socket.getOutputStream());

            DataInputStream dis = new DataInputStream(com_socket.getInputStream());

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

```

4.2 Client Class

```

package javaSocket;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;

public class Client {

    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 scn = new Scanner(System.in);
            DataOutputStream dos = new DataOutputStream(clientsocket.getOutputStream());
            DataInputStream dis = new DataInputStream(clientsocket.getInputStream());

```

```

while(true){
    String inLine = dis.readUTF();
    System.out.println(inLine);
    String outLine = scn.nextLine();
    dos.writeUTF(outLine);

    if(outLine.equals("ENDS")){
        System.out.println("Closing the connecting "+ clientsocket);
        clientsocket.close();
        System.out.println("Connection Closed");
        break;
    }

    switch (outLine) {
        case "Add" -> {
            Scanner scanner = new Scanner(System.in);
            int a = scanner.nextInt();
            int b = scanner.nextInt();
            Arithmetics.sum(a,b);
        }
        case "Sub" -> {
            Scanner scanner = new Scanner(System.in);
            int a = scanner.nextInt();
            int b = scanner.nextInt();
            Arithmetics.sub(a,b);
        }
        case "Mul" -> {
            Scanner scanner = new Scanner(System.in);
            int a = scanner.nextInt();
            int b = scanner.nextInt();
            Arithmetics.mul(a,b);
        }
        case "Div" -> {
            Scanner scanner = new Scanner(System.in);
            int a = scanner.nextInt();
            int b = scanner.nextInt();
            Arithmetics.div(a,b);
        }
        case "Mod" -> {
            Scanner scanner = new Scanner(System.in);
            int a = scanner.nextInt();
            int b = scanner.nextInt();
            Arithmetics.mod(a,b);
        }
    }
}

```

```

        default -> System.out.println("Invalid input");
    }

    String received = dis.readUTF();
    System.out.println(received);
}
dos.close();
dis.close();
clientsocket.close();
}
catch (Exception ex){
    System.out.println(ex);
}
}
}

```

4.3 ClientHandler Class

```

package javaSocket;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

class ClientHandler extends Thread {
    final Socket com_tunnel;
    final DataInputStream dis_tunnel;
    final DataOutputStream dos_tunnel;
    String received = "";
    String toreturn = "";

    public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos) throws
    IOException {
        this.com_tunnel = s;
        this.dis_tunnel = dis;
        this.dos_tunnel = dos;
    }
    public void run() {
        while (true) {
            try {
                dos_tunnel.writeUTF("What do you want [Add/Sub/Mul/Div/Mod]");
                received = dis_tunnel.readUTF();
            }

```

```

        if (received.equals("ENDS")) {
            System.out.println("Client " + this.com_tunnel + " sends exits");
            System.out.println("Closing the connection");
            this.com_tunnel.close();
            break;
        }
        switch (received) {
            case "Add" -> {
                toreturn = Arithmetics.sum();
                dos_tunnel.writeUTF(toreturn);
            }
            case "Sub" -> {
                toreturn = Arithmetics.sub();
                dos_tunnel.writeUTF(toreturn);
            }
            case "Mul" -> {
                toreturn = Arithmetics.mul();
                dos_tunnel.writeUTF(toreturn);
            }
            case "Div" -> {
                toreturn = Arithmetics.div();
                dos_tunnel.writeUTF(toreturn);
            }
            case "Mod" -> {
                toreturn = Arithmetics.mod();
                dos_tunnel.writeUTF(toreturn);
            }

            default -> System.out.println("Invalid input");
        }

    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
}
}

```

4.4 Arithmetics Class

```
package javaSocket;

public class Arithmetics {

    public static String sum(int x, int y){

        int c = x + y;
        System.out.print(c);
        return String.valueOf(c);
    }
    public static String sub(int x, int y){
        int c;
        c = x - y;
        System.out.print(c);
        return String.valueOf(c);
    }
    public static String mul(int x, int y){
        int c;
        c = x * y;
        System.out.print(c);
        return String.valueOf(c);
    }
    public static String div(int x, int y){
        int c;
        c = x / y;
        System.out.print(c);
        return String.valueOf(c);
    }
    public static String mod(int x, int y){
        int c;
        c = x % y;
        System.out.print(c);
        return String.valueOf(c);
    }
}

    public static String sum() {

        return String.valueOf("");
    }
    public static String sub() {
```



```

        return String.valueOf("");
    }
    public static String mul() {

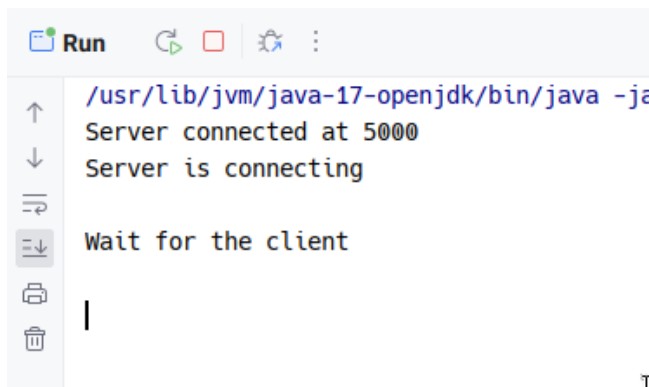
        return String.valueOf("");
    }
    public static String div() {

        return String.valueOf("");
    }
    public static String mod() {

        return String.valueOf("");
    }
}

```

5. TEST RESULT / OUTPUT

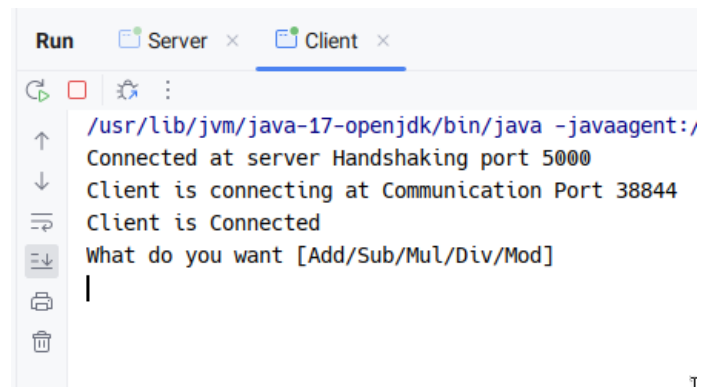


```

Run
/usr/lib/jvm/java-17-openjdk/bin/java -j
Server connected at 5000
Server is connecting
Wait for the client
|

```

Fig-5.1: Run Server Class



```

Run Server x Client x
/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/
Connected at server Handshaking port 5000
Client is connecting at Communication Port 38844
Client is Connected
What do you want [Add/Sub/Mul/Div/Mod]
|

```

Fig-5.2: Run Client Class



```

Run Server x Client x Client_2 x
/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar
Server connected at 5000
Server is connecting
Wait for the client
A new client is connected Socket[addr=/127.0.0.1,port=46830,localport=5000]
A new thread is assigning
A new client is connected Socket[addr=/127.0.0.1,port=43488,localport=5000]
A new thread is assigning
|

```

Fig-5.3: Client Informations in the Server Class

```

Run  Server  Client
/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/usr/sh
Connected at server Handshaking port 50000
Client is connecting at Communication Port 38844
Client is Connected
What do you want [Add/Sub/Mul/Div/Mod]
Add
4 6
10
What do you want [Add/Sub/Mul/Div/Mod]

```

Fig-5.4: Perform addition operation

```

Run  Server  Client  Client_2
/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/usr/sh
Connected at server Handshaking port 50000
Client is connecting at Communication Port 43488
Client is Connected
What do you want [Add/Sub/Mul/Div/Mod]
Mod
10 3
1
What do you want [Add/Sub/Mul/Div/Mod]
|

```

Fig-5.5: Perform modulus operation

```

Run  Server  Client  Client_2
/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=39
Connected at server Handshaking port 50000
Client is connecting at Communication Port 46830
Client is Connected
What do you want [Add/Sub/Mul/Div/Mod]
Add
4 6
10
What do you want [Add/Sub/Mul/Div/Mod]
ENDS
Closing the connecting Socket[addr=localhost/127.0.0.1,port=50000,localport=46830]
Connection Closed

```

Fig-5.6: Closing the Client

```

Project  Server.java  Client.java  ClientHandler.java  Arithmetics.java
arithmeticSocket ~/IdeaProjects/arithmetic
  .idea
  .out
  src
    javaSocket
      Arithmetics
      Client
      ClientHandler
      Server
    arithmeticSocket.iml
  External Libraries
  Scratches and Consoles

public class Server {
    no usages
    public static void main(String args[]) throws IOException {
        ServerSocket handshake = new ServerSocket ( port: 50000);
        System.out.println("Server connected at " + handshake.getLocalPort());
        System.out.println("Server is connecting\n");
        System.out.println("Wait for the client\n");
        while(true){
            Socket com_socket = handshake.accept();
            System.out.println("A new client is connected "+ com_socket);
            DataOutputStream dos = new DataOutputStream(com_socket.getOutputStream());

            DataInputStream dis = new DataInputStream(com_socket.getInputStream());

            System.out.println("A new thread is assigning");
            Thread new_tunnel = new ClientHandler(com_socket, dis, dos);
        }
    }
}

Run  Server  Client  Client_2
A new client is connected Socket[addr=/127.0.0.1,port=46830,localport=50000]
A new thread is assigning
A new client is connected Socket[addr=/127.0.0.1,port=43488,localport=50000]
A new thread is assigning
Client Socket[addr=/127.0.0.1,port=43488,localport=50000] sends exits
Closing the connection
Client Socket[addr=/127.0.0.1,port=46830,localport=50000] sends exits
Closing the connection

```

Fig-5.7: Perform all arithmetic operations

Fig-5.1: Run Server Class - Run the Server class first, and it gives output that it waits for the clients.

Fig-5.2: Run Client Class - Client class shows the port number and waits for the user input when it runs.

Fig-5.3: Client Informations in the Server Class - Then back in Server class it gives the information that which clients are connected with their local ip and port numbers.

Fig-5.4: Perform addition operation - User can perform any arithmetics operations. Clients just need to give input which operations they want to perform as well as give the corresponding value. Here the client chooses the Addition operation and provides two inputs then the server returns the result to the clients.

Fig-5.5: Perform modulus operation - It's already said that the program can perform multiple operations. But here a new client request to the server to perform modulus operation at the same time where the previous client requested for additional operation. By using the multithreading server can serve both clients at the same time. Here first client requests to the server for additional operation at the same time client_2 requests the server for the modulus operations. Server returns the result for both clients.

Fig-5.6: Closing the Client - While clients choose ENDS as input then clients close its connections with the server.

Fig-5.7: Perform all arithmetic operations - Here the platform where we made this program. We choose IntelliJ IDEA where we can write our program code simultaneously and we can show the results in the output console. In the server console it shows the detailed information about which clients are close to the connection with the server along with their communications port numbers.

6. ANALYSIS AND DISCUSSION

- Based on the focused objective(s) to understand about multithreaded socket programming, this task helped us to learn about the basic structure of multi-threaded socket programming.
- The client sends requests as many times as he wishes. The individual client ends its connection by saying “ENDS”.
- The additional lab exercise of multi-threaded socket programming will help us to be confident towards the fulfillment of the objectives(s) and guide us to implement some real-life problems using multi-threaded socket programming.