Department of
Computer Science and Engineering

Title: Implementation of socket programming using threading

Computer Networking Lab

CSE 312

Green University of Bangladesh

1 Objective(s)

- To gather basic knowledge of socket programming using threading
- To learn about step by step implementation for handling multiple client requests using threaded socket programming.

2 Problem analysis

Socket is mainly the door between application process and end-end-transport protocol. Java Socket programming can be connection-oriented or connection-less. 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 provides 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, server opens 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.

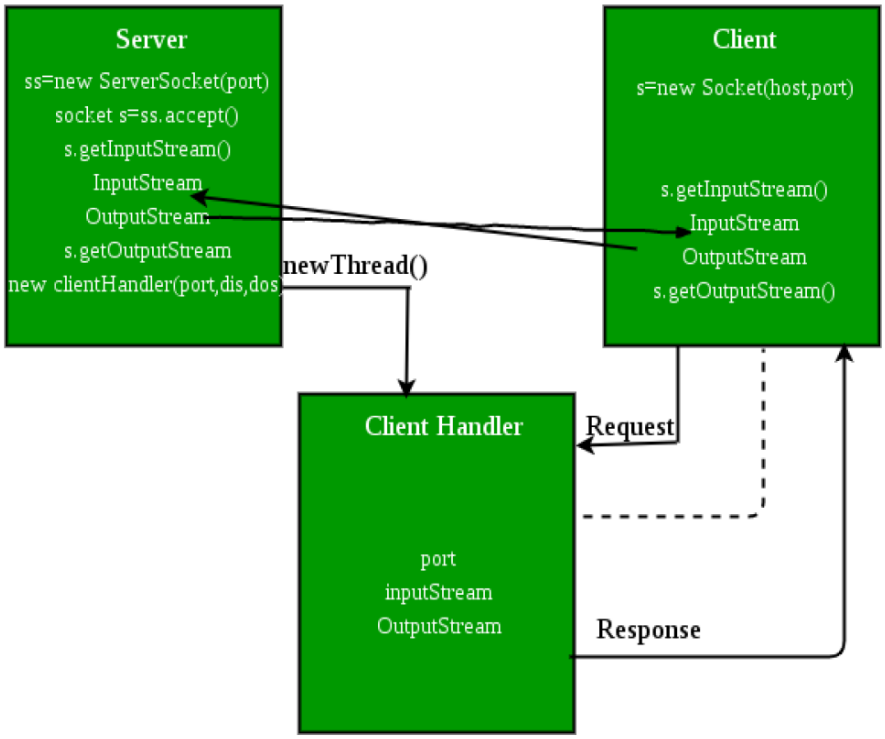

Figure 1: Flow chart

3 Procedure

In the multi-threaded socket programming, at first client creates a connection with server through serversocket which is depicted in Fig. 1. Then Server creates a different thread namely ClientHandler for each client. In the ClientHandler class, server passes communication port, inputStream and outputStream as parameters. Then Client conducts all types of communications with the server through ClientHandler class. The detailed step-by-step procedures is discussed as follows.

3.1 Server Side Programming

In 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.
  Note: Every request will always have a triplet of the socket, input stream, and output stream. This ensures that each object of this class writes on one specific stream rather than on multiple streams.
- Inside the **run()** method of this class, it performs three operations: request the user to specify whether time or date 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**
- **Communication**
- **Closing the Connection**

The detailed algorithms of the server side and client side connections are given in Algorithm 1 and 2.

Algorithm 1: Algorithm of Server Side Socket Connection

1: Create a ServerSocket object namely handshaking socket which takes port number as input
2: Create a plain Socket namely communication socket object that accepts client request
3: Create two objects of DataOutputStream and DataInputStream classes which is used for send and read data, respectively.
4: Create an object for ClientHandler Thread class and pass communication socket, object of DataOutputStream and DataInputStream class as the parameters
5: Start the thread class by calling start() method.
6: In the ClientHandler Thread class, create a constructor of that class.
7: Do the necessary communication until client sends "Exit"
8: Close the connection

Algorithm 2: Algorithm of Client Side Socket Connection

1: Create a Socket object which takes IP address and port number as input.
2: Create two objects of DataOutputStream and DataInputStream classes which is used for send and read data, respectively.
3: Client can Send the data to the server side using writeUTF() function and Client can read any data using readUTF() function
4: Client can continue its communication with the server until client sends "Exit"
5: Step -5: Close the connection

4 Implementation in Java

```
1  /* Server Side Code */
2  import java.io.*;
3  import java.net.*;
4
5  public class ServerThread {
6      public static void main(String args[]) throws IOException{
7          ServerSocket handshake = new ServerSocket (5000);
8          System.out.println("Server connected at " + handshake.getLocalPort());
9          System.out.println("Server is connecting\n");
10         System.out.println("Wait for the client\n");
11         while(true){
12             Socket com_socket = handshake.accept();
13             System.out.println("A new client is connected "+ com_socket);
14             DataOutputStream dos = new DataOutputStream(com_socket.
                   getOutputStream());
15             DataInputStream dis = new DataInputStream(com_socket.getInputStream
                   ());
16             System.out.println("A new thread is assigning");
17             Thread new_tunnel = new ClientHandler(com_socket, dis, dos);
18             new_tunnel.start();
19
20         }
21     }
22 }
```

```
1  /* ClientHandler Class Code */
2  import java.io.*;
3  import java.net.*;
4  import java.text.DateFormat;
5  import java.text.SimpleDateFormat;
6  import java.util.*;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9
10 class ClientHandler extends Thread{
11     DateFormat fordate = new SimpleDateFormat("yyyy/MM/dd");
12     DateFormat fortime = new SimpleDateFormat("hh:mm:ss");
13     final Socket com_tunnel;
14
15     final DataInputStream dis_tunnel;
16     final DataOutputStream dos_tunnel;
17     String received = "";
18     String toreturn = "";
```

```
19      public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos)
20      {
21          this.com_tunnel = s;
22          this.dis_tunnel = dis;
23          this.dos_tunnel = dos;
24
25      }
26      public void run(){
27         while(true){
28             try {
29                 dos_tunnel.writeUTF("What do you want [Date/Time]");
30                 received = dis_tunnel.readUTF();
31                 if(received.equals("Exit")){
32                     System.out.println("Client " + this.com_tunnel + " sends
                           exits");
33                     System.out.println("Closing the connection");
34                     this.com_tunnel.close();
35                     break;
36                 }
37
38                 Date date = new Date();
39                 switch (received) {
40
41                      case "Date" :
42                          toreturn = fordate.format(date);
43                          dos_tunnel.writeUTF(toreturn);
44                          break;
45
46                      case "Time" :
47                          toreturn = fortime.format(date);
48                          dos_tunnel.writeUTF(toreturn);
49                          break;
50
51                      default:
52                          dos_tunnel.writeUTF("Invalid input");
53                          break;
54                  }
55
56             } catch (IOException ex) {
57                 Logger.getLogger(ClientHandler.class.getName()).log(Level.SEVERE,
                       null, ex);
58             }
59         }
60         try {
61             this.dos_tunnel.close();
62             this.dis_tunnel.close();
63         } catch (IOException ex) {
64             Logger.getLogger(ClientHandler.class.getName()).log(Level.SEVERE,
                   null, ex);
65         }
66     }
67 }
```

```
1  /* Client Side Code */
2  import java.io.*;
3  import java.net.*;
```

```
4  import java.util.*;
5
6  public class ClientThread {
7      public static void main(String args[]) throws IOException{
8          try{
9          Socket clientsocket = new Socket ("localhost", 5000);
10         System.out.println("Connected at server Handshaking port " +
               clientsocket.getPort());
11         System.out.println("Client is connecting at Communication Port " +
               clientsocket.getLocalPort());
12         System.out.println("Client is Connected");
13         Scanner scn = new Scanner(System.in);
14         DataOutputStream dos = new DataOutputStream(clientsocket.getOutputStream
               ());
15         DataInputStream dis = new DataInputStream(clientsocket.getInputStream())
               ;
16         while(true){
17             String inLine = dis.readUTF();
18             System.out.println(inLine);
19             String outLine = scn.nextLine();
20             dos.writeUTF(outLine);
21
22             if(outLine.equals("Exit")){
23                 System.out.println("Closing the connecting "+ clientsocket);
24                 clientsocket.close();
25                 System.out.println("Connection Closed");
26                 break;
27             }
28             String received = dis.readUTF();
29             System.out.println(received);
30         }
31         dos.close();
32         dis.close();
33         clientsocket.close();
34         }
35         catch (Exception ex){
36             System.out.println(ex);
37         }
38     }
39 }
```

5 Input/Output

To execute the above code, at first you need to run server side code, then run client side code. Here I have run code for two clients.

Output window of the client # 1:

```
Connected at server Handshaking port 5000
Client is connecting at Communication port 50198
Client is Connected
What do you want [Date/Time]
Date
2021/07/24
What do you want [Date/Time]
Time
11:14:26
What do you want [Date/Time]
Exit
Closing the connecting Socket[addr=localhost/127.0.0.1,port=5000,localport=50198]
Connection Closed
```

Output window of the client # 2:

```
Connected at server Handshaking port 5000
Client is connecting at Communication port 49581
Client is Connected
What do you want [Date/Time]
Time
11:17:40
What do you want [Date/Time]
Exit
Closing the connecting Socket[addr=localhost/127.0.0.1,port=5000,localport=49581]
Connection Closed
```

Output window of server side is given below:

```
Server is connected at port no: 5000
Server is connecting
Waiting for the client
A new client is connected Socket[addr=/127.0.0.1,port=50198,localport=5000]
A new thread is assigning
Client Socket[addr=/127.0.0.1,port=50198,localport=5000] sends exits
Closing the connection
A new client is connected Socket[addr=/127.0.0.1,port=49581,localport=5000]
A new thread is assigning
Client Socket[addr=/127.0.0.1,port=49581,localport=5000] sends exits
Closing the connection
```

6 Discussion & Conclusion

Based on the focused objective(s) to understand about multi-threaded socket programming, this task helped us to learn about the basic structure of multi-threaded socket programming. The additional lab exercise of multi-threaded socket programming will help us to be confident towards the fulfilment of the objectives(s) and guide us to implement some real-life problem using multi-threaded socket programming.

7 Lab Task (Please implement yourself and show the output to the instructor)

Create a simple basic thread application for multiple clients to be connected with a single server. Here each client sends a string to the single server. The server returns the string in all uppercase to that client. After receiving the uppercase string from the server, that client closes down automatically. The server can serve at most 5 clients in its lifetime. If the 6th client comes to get connected or just completing giving service to exactly 5 clients, the server closes down automatically.

7.1 Problem analysis

In our previous task, server can not be closed automatically. Server can create infinite connections in its lifetime. However, in this task, server can not create connections with infinite number of client. The server can serve at most 5 clients in its lifetime. If the 6th client comes to get connected or just completing giving service to exactly 5 clients, the server closes down automatically.

For that reason, you need to convert the infinite loop to a conditional loop. You can do it using while/for loop and you have to keep a counter for counting the number of client those have been connected till now. When a new client has come, the value of counter has been incremented by one and when the counter value is equal to 10, then the server close its connection.

Then, you need to do your necessary task i.e., converting a string to uppercase string in the **run()** method under the **ClientHandler** class by changing simple Date-Time server code. You can easily convert a string to Uppercase string by using java build-in function **toUpperCase()** method.

8 Lab Exercise (Submit as a report)

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". For example:

- 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.

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.