

CL-IX (Distributed Computing Systems Lab)

Assignment 1 (A)

Roll No.: 43141

Name: Sahil Naphade

Batch: P-9

Problem Statement: To develop distributed application through implementing client-server communication programs based on Java Sockets (TCP and UDP).

TCP – Chatting application

UDP – Quotes server

Objectives: By the end of this assignment, the student will be able to implement any distributed multi-threaded client-server programs using Java sockets.

Tools: Java Programming Environment, JDK 1.8+.

Theory:

Java Socket programming is used for communication between the applications running on different JRE.

- Java Socket programming can be connection-oriented or connectionless.
- Socket and ServerSocket classes are used for connection-oriented socket programming (TCP).
- DatagramSocket and DatagramPacket classes are used for connectionless socket programming (UDP).
- Socket class and methods:
 - A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.
 - public InputStream getInputStream(): returns the InputStream attached with this socket.
 - public OutputStream getOutputStream(): returns the OutputStream attached with this socket.
 - public synchronized void close(): closes this socket

1. Java Connection-oriented sockets:

- a. The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.
- b. public Socket accept(): returns the socket and establish a connection between server and client.
- c. public synchronized void close(): closes the server socket.
- d. Steps:
 - i. Create two java files, Server.java and Client.java.
 - ii. Server file contains two classes namely: Server (public class for creating server) and ClientHandler (for handling any client using multithreading).

- iii. Client file contain only one public class Client (for creating a client).
 - iv. Java API networking package (java.net) to be imported which takes care of all network programming.
- e. Working:
 - i. When a client, say client1 sends a request to connect to server, the server assigns a new thread to handle this request.
 - ii. The newly assigned thread is given the access to streams for communicating with the client.
 - iii. After assigning the new thread, the server via its while loop, again comes into accepting state.
 - iv. When a second request comes while first is still in process, the server accepts these requests and again assigns a new thread for processing it. In this way, multiple requests can be handled even when some requests are in process.
- f. Implementation
 - i. SERVER-SIDE PROGRAMMING (Server.java)

Server class: The steps involved on server side:

 1. Establishing the Connection: Server socket object is initialized and inside a while loop a socket object continuously accepts incoming connection.
 2. Obtaining the Streams: The inputstream object and outputstream object is extracted from the current requests' socket object. Streams reads data (numbers) instead of just bytes.
 3. Creating a handler object: After obtaining the streams and port number, a new clientHandler object is created with these parameters.
 4. Invoking the start() method : The start() method is invoked on this newly created thread object.
 - ii. CLIENT-SIDE PROGRAMMING (Client.java)
 1. Client-side programming is similar as in general socket program with the following steps-
 - a. Establish a Socket Connection
 - b. Communication
 - c. Close the connection
 - iii. Output:

```
Itsnasa@Nasa-Rlg: ~/DistributedComputingLab/AssignmentITCP/src/serverPackage$ java Server.java
Enter the port no. on which you wish to start server: 8089
Exception in thread "main" java.net.BindException: Address already in use (Bind failed)
    at java.base/java.net.PlainSocketImpl.socketBind(Native Method)
    at java.base/java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java:436)
    at java.base/java.net.ServerSocket.bind(ServerSocket.java:395)
    at java.base/java.net.ServerSocket.<init>(ServerSocket.java:257)
    at java.base/java.net.ServerSocket.<init>(ServerSocket.java:249)
    at ServerPackage.Server.main(Server.java:15)
Itsnasa@Nasa-Rlg: ~/DistributedComputingLab/AssignmentITCP/src/serverPackage$ java Server.java
Enter the port no. on which you wish to start server: 8089
Server waiting on port no. 8089
Client: Hello server!!
Enter your response: Hello!
Client: How are you??
Enter your response: I am fine!
Client: Roll No. 43141
Enter your response: Batch P9
Client: Exit
Enter your response: Exit
Itsnasa@Nasa-Rlg: ~/DistributedComputingLab/AssignmentITCP/src/serverPackage$

Itsnasa@Nasa-Rlg: ~/DistributedComputingLab/AssignmentITCP/src/clientPackage$ java Client.java
Enter the port no on which you want to connect: 8089
Enter your message: Hello server!!
Server: Hello!
Enter your message: How are you??
Server: I am fine!
Enter your message: Roll No. 43141
Server: Batch P9
Enter your message: Exit
Server: Exit
Itsnasa@Nasa-Rlg: ~/DistributedComputingLab/AssignmentITCP/src/clientPackage$
```

2. Java Connection-less sockets:

a. Java.net.DatagramSocket class:

- i. Every packet sent from a datagram socket is individually routed and delivered.
- ii. It can also be used for sending and receiving broadcast messages.
- iii. Datagram Sockets is the java's mechanism for providing network communication via UDP instead of TCP.
- iv. DatagramSocket() : Creates a datagramSocket and binds it to any available port on local machine. If this constructor is used, the OS would assign any port to this socket.

b. JAVA API FOR DATAGRAM COMMUNICATION

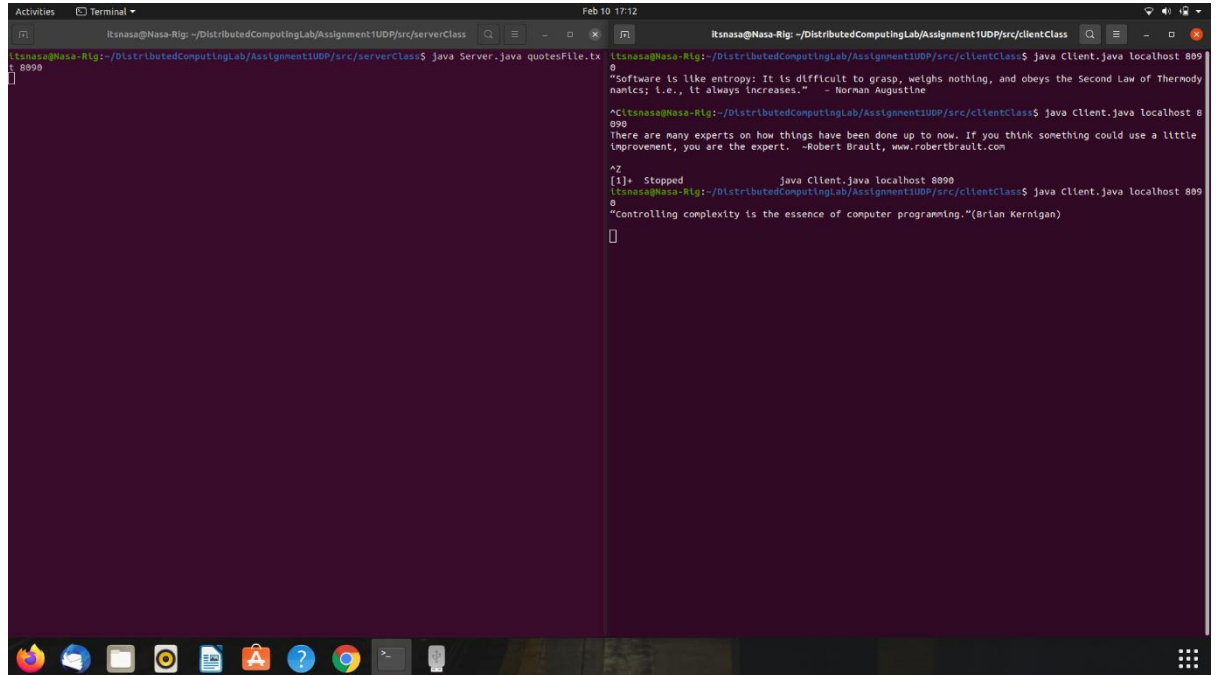
- i. DatagramPacket : A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. In Java, DatagramPacket represents a datagram.
- ii. You can create a DatagramPacket object by using one of the following
- iii. Constructors:
 1. DatagramPacket(byte[] buf, int length)
 2. DatagramPacket(byte[] buf, int length, InetAddress address, int port)
- iv. The data must be in the form of an array of bytes.
- v. The first constructor is used to create a DatagramPacket to be received.
- vi. The second constructor creates a DatagramPacket to be sent, so you need to specify the address and port number of the destination host.
- vii. The parameter length specifies the amount of data in the byte array to be used, usually is the length of the array (buf.length).

c. Datagram socket

- i. DatagramSocket is used to send and receive DatagramPackets.
- ii. In Java, DatagramSocket is used for both client and server. There are no separate classes for client and server like TCP sockets.
- iii. DatagramSocket creates object to establish a UDP connection for sending and receiving datagram, by using the following constructors:

- iv. DatagramSocket(int port, InetAddress laddr): This constructor binds the server to the specified IP address (in case the computer has multiple IP addresses).
- v. The key methods of the DatagramSocket include:
 - 1. send(DatagramPacket p): sends a datagram packet.
 - 2. receive(DatagramPacket p): receives a datagram packet
 - 3. close(): closes the socket.

d. Output:



```
Itsnasa@Nasa-Rlg: ~/DistributedComputingLab/Assignment1UDP/src/serverClass$ java Server.java quotesFile.txt
8090

Itsnasa@Nasa-Rlg:~/DistributedComputingLab/Assignment1UDP/src/clientClass$ java Client.java localhost 8090
0
"Software is like entropy: It is difficult to grasp, weighs nothing, and obeys the Second Law of Thermodynamics; i.e., it always increases." - Norman Augustine
^C
Itsnasa@Nasa-Rlg:~/DistributedComputingLab/Assignment1UDP/src/clientClass$ java Client.java localhost 8090
0
There are many experts on how things have been done up to now. If you think something could use a little improvement, you are the expert. -Robert Brault, www.robertbrault.com
^Z
[1]+  Stopped                  java Client.java localhost 8090
Itsnasa@Nasa-Rlg:~/DistributedComputingLab/Assignment1UDP/src/clientClass$ java Client.java localhost 8090
0
"Controlling complexity is the essence of computer programming."(Brian Kernigan)
```

Conclusion:

Thus, I learnt how to develop a client/server distributed application relying on TCP and UDP protocol. Based on this knowledge, we are able to develop client programs that communicate with servers via TCP and UDP, and developing their own TCP/UDP client/server applications.