Mawlana Bhashani Science and Technology University

# Lab-Report

Course code: ICT-3208

Course title: Computer Networks Lab

Date of Performance:03/06/2021

Date of Submission:10/06/2021


**Submitted by**

Name:Md Asikur Rahman

ID:IT-18025

3rd year 2nd semester

Session: 2017-2018

Dept. of ICT

MBSTU.

**Submitted To**

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

**Experiment No: 06**

**Experiment Name: Socket Programming-Time protocol Implementation.**

**Objectives:**

The main objectives of this lab how to know Sockets provide the communication mechanism between two computers using TCPand java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

**Theory:**

How do we build Internet applications? In this lecture, we will discuss the socket API and support for TCP communications between end hosts. Socket programing is the key API for programming distributed applications on the Internet. Note, we do not cover the UDP API in the course. If interested take CS60 Computer Networks.

Socket program is a key skill needed for the robotics project for exerting control - in this case the controller running on your laptop will connect to the server running on the bot

**Time Protocol:** The Time Protocol is a network protocol in the Internet Protocol Suite defined in 1983 in RFC 868 by Jon Postel and K. Harrenstein. Its purpose is to provide a site-independent, machine readable date and time.
The Time Protocol may be implemented over the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP). A host connects to a server that supports the Time Protocol on port 37. The server then sends the time as a 32-bit unsigned integer in binary format and in network byte order, representing the number of seconds since 00:00 (midnight) 1 January, 1900 GMT, and closes the connection. Operation over UDP requires the sending of any datagram to the server port, as there is no connection setup for UDP

**1. Briefly explain the term IPC in terms of TCP/IP communication. Answer:**

In computer science, inter-process communication (IPC) refers specifically to the mechanisms an operating system provides to allow the processes to manage shared data.
IPC is a term we use for interactions between two processes on the same host. William Westlake mentions TCP/IP, which is used for interactions with another host. It can be used locally as well, but it is relatively inefficient. Unix domain sockets are used in the same way, but are only for local use and a bit more efficient.
The answer will be different each Operating System. Unix offers System V IPC, which gives you message queues, shared memory, and semaphores.
Message queues are easy to use: processes and threads can send variable-sized messages by appending them to some queue and others can receive messages from them so that each message is received at most once. Those operating can be blocking or non-blocking. The difference with UDP is that messages are received in the same order as they are sent. Pipes are simpler, but pass a stream of data instead of distinct messages. Line feeds can be used as delimiters.
Shared memory allows different processes to share fixed regions of memory in the same way that threads have access to the same memory. Unix allocates a certain amount of memory after which it can be accessed like private memory. Since two threads updating the same data can lead to inconsistencies, semaphores can be used to achieve mutual exclusion. A similar mechanism is the memory-mapped file: the difference is that the memory segment it initialized from a disc file and changes can be permanent. The size of a file can change, which complicates shared files.

**2. What is the maximum size of a UDP datagram? What are the implications of using a packet-based protocol as opposed to a stream protocol for transfer of large files?**

**Answer:**

The size datagram would contain no data-only an IP header with no options and a UDP header. It depends on the underlying protocol i.e., whether you are using IPv4 or IPv6.
This field specifies the length in bytes of the UDP header and UDP data. The minimum length is 8 bytes, the length of the header. The field size sets a theoretical limit of 65,535 bytes (8 byte header + 65,527 bytes of data) for a UDP datagram. However the actual limit for the data length, which is imposed by the

underlying IPv4 protocol, is 65,507 bytes (65,535 − 8 byte UDP header − 20 byte IP header).

Using IPv6 it is possible to have UDP datagrams of size greater than 65,535 bytes. Specifies that the length field is set to zero if the length of the UDP header plus UDP data is greater than 65,535.

**3. TCP is a reliable transport protocol, briefly explain what techniques are used to provide this reliability.**

**Answer:**

A number of mechanisms help provide the reliability TCP guarantees. Each of these is described briefly below.

**Checksums:** All TCP segments carry a checksum, which is used by the receiver to detect errors with either the TCP header or data.

**Duplicate data detection:** It is possible for packets to be duplicated in packet switched network; therefore TCP keeps track of bytes received in order to discard duplicate copies of data that has already been received.

**Retransmissions:** In order to guarantee delivery of data, TCP must implement retransmission schemes for data that may be lost or damaged. The use of positive acknowledgements by the receiver to the sender confirms successful reception of data. The lack of positive acknowledgements, coupled with a timeout period calls for a retransmission.

**Sequencing:** In packet switched networks, it is possible for packets to be delivered out of order. It is TCP's job to properly sequence segments it receives so it can deliver the byte stream data to an application in order.

**Timers:** TCP maintains various static and dynamic timers on data sent. The sending TCP waits for the receiver to reply with an acknowledgement within a bounded length of time. If the timer expires before receiving an acknowledgement, the sender can retransmit the segment.

**4. Why are the htons(), htonl(), ntohs(), ntohl() functions used? Answer:**

When you write a network program in C + +, you will often encounter the problem of byte network order and host order. This is the 4 functions that may be used to htons (), Ntohl (), Ntohs (), htons ().

The conversion function between the network byte order and the local byte order htons() host to network short htonl() host to network long ntohs() network to host short ntohl() network to host long

**5. What is the difference between a datagram socket and a stream socket? Which transport protocols do they correspond to?**

**Answer**:

The difference is given below:

**Stream Socket:**

- Dedicated & end-to-end channel between server and client.
- Use TCP protocol for data transmission.
- Reliable and Lossless.
- Data sent/received in the similar order.
- Long time for recovering lost/mistaken data **Datagram Socket:**

- Not dedicated & end-to-end channel between server and client.
- Use UDP for data transmission.
- Not 100% reliable and may lose data.
- Data sent/received order might not be the same.
- Don't care or rapid recovering lost/mistaken data.

**6. What is a stateful service, as opposed to a stateless service? What are the performance implications of statefulness and statelessness?**

**Answer:**

The opposed of stateful and stateless applications is that stateless applications don't "store" data whereas stateful applications require backing storage. Stateful applications like the Cassandra, MongoDB and mySQL databases all require some type of persistent storage that will survive service restarts.

Keeping state is critical to running a stateful application whereas any data that flows via a stateless service is typically transitory and the state is stored only in a separate back-end service like a database. Any associated storage is typically ephemeral. If the container restarts for instance, anything stored is lost. As organizations adopt containers, they tend to begin with stateless containers as they are more easily adapted to this new type of architecture and better

separated from their monolithic application codebase, thus they are more amenable to independent scaling.

**Time Protocol implementation:** A java program where the following GreetingClient is a client program that connects to a server by using a socket and sends a greeting, and then waits for a response

**Client code:**

*GreetingClient.java GreetingServer.java

```
1  package networking;
2
3
4
5
6  import java.net.*;
7  import java.io.*;
8  public class GreetingClient {
9
10     public static void main(String[] args) {
11         String serverName = args[0];
12         int port = Integer.parseInt(args[1]);
13         try {
14             System.out.println("Connecting to " + serverName + " on port " + port);
15             Socket client = new Socket(serverName, port);
16             System.out.println("Just connected to "
17                     + client.getRemoteSocketAddress());
18             OutputStream outToServer = client.getOutputStream();
19             DataOutputStream out = new DataOutputStream(outToServer);
20             out.writeUTF("Hello from " + client.getLocalSocketAddress());
21             InputStream inFromServer = client.getInputStream();
22             DataInputStream in = new DataInputStream(inFromServer);
23             System.out.println("Server says " + in.readUTF());
24             client.close();
25         } catch (IOException e) {
26             e.printStackTrace();
27         }
28     }
29 }
30
```

The following GreetingServer program is an example of a server application that uses the Socket class to listen for clients on a port number specified by a commandline argument –

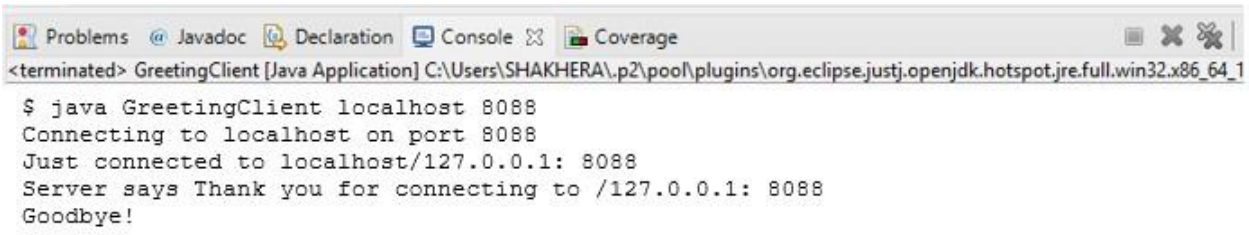
**Server code:**

*GreetingClient.java  *GreetingServer.java

```
package networking;



import java.net.*;
import java.io.*;
public class GreetingServer extends Thread {

    private ServerSocket serverSocket;
    public GreetingServer(int port) throws IOException {
        serverSocket = new ServerSocket(port);
        serverSocket.setSoTimeout(10000);
    }
    public void run() {
        while (true) {
            try {
                System.out.println("Waiting for client on port "
                        + serverSocket.getLocalPort() + "...");
                Socket server = serverSocket.accept();
                System.out.println("Just connected to " + server.getRemoteSocketAddress());
                DataInputStream in = new DataInputStream(server.getInputStream());
                System.out.println(in.readUTF());
                DataOutputStream out = new DataOutputStream(server.getOutputStream());
                out.writeUTF("Thank you for connecting to " + server.getLocalSocketAddress()
                        + "\nGoodbye!");
                server.close();
            } catch (SocketTimeoutException s) {
                System.out.println("Socket timed out!");
                break;
            } catch (IOException e) {
                e.printStackTrace();
                break;
            }
        }
    }
    public static void main(String[] args) {
        int port = Integer.parseInt(args[0]);
        try {
            Thread t = new GreetingServer(port);
            t.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

## Output:

Compile the client and  the server code




## Discussion:

Using this lab, I have to know about Time Protocol Implementation over the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP). To perform this lab, firstly I connected to a server specified by port number. I also know how to communicate client and server by read from and write to the socket. It can how toSockets provide the communication mechanism between two computers using TCP. When the ServerSocket invokes accept(), the method does not return until a client connects. After a client does connect, the ServerSocket creates a new Socket on an unspecified port and returns a reference to this new Socket. A TCP connection now exists between the client and the server, and communication can begin.

We learn from this lab

- What is a socket?
- The client-server model
- Byte order
- TCP socket API
- Concurrent server design
- Example of echo client and iterative server
- Example of echo client and concurrent server