Mawlana Bhashani Science and Technology University
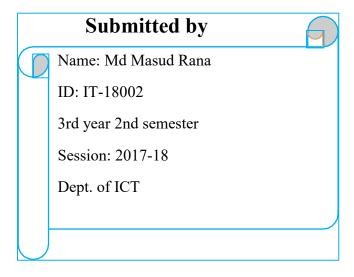
Lab-Report
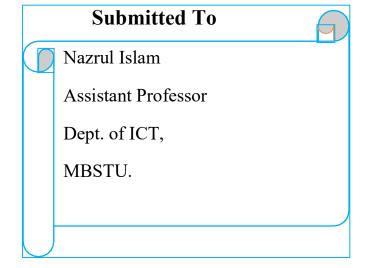
Lab Report No: 06

Lab Report Name: Socket programming(Time protocol).

Course title: Network Planning and Designing Lab

Course code: ICT-3208

Submitted by

Name: Md Masud Rana

ID: IT-18002

3rd year 2nd semester

Session: 2017-18

Dept. of ICT

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT,

MBSTU.

Socket Programming:

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket

. The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets –

• The server instantiates a ServerSocket object, denoting which port number communication is to occur on.

• The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.

• After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.

• The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.

• On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

1) Briefly explain the term IPC in terms of TCP/IP communication.

Answer:

InterProcess Communication 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 operatings 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 initialised 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:

It depends on the underlying protocol i.e., whether you are using IPv4 or IPv6.

• In IPv4, the maximum length of packet size is 65,536. So, for UDP datagram you have maximum data length as:

65,535 bytes - 20 bytes(Size of IP header) = 65, 515 bytes (including 8 bytes UDP header)

- In IPv6, the maximum length of packet size allowed is 64 kB, so, you can have UDP datagram of size greater than that.

NOTE: This size is the theoretical maximum size of UDP Datagram, in practice though, this limit is further constrained by the MTU of data-link layer(which varies for each data-link layer technology, but cannot be less than 576 bytes), considering that, maximum size of UDP datagram can be further calculated as (for IPv4):

- 576 bytes - 20 bytes(IP header) = 556 (including 8 bytes UDP header)

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:

These are used for:

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?
Answer:**

The difference is given below:

• Stream sockets enable processes to communicate using TCP. A stream socket provides a bidirectional, reliable, sequenced, and unduplicated flow of data with no record boundaries. After the connection has been established, data can be read from and written to these sockets as a byte stream. The socket type is SOCK_STREAM.

• Datagram sockets enable processes to use UDP to communicate. A datagram socket supports a bidirectional flow of messages. A process on a datagram socket might receive messages in a different order from the sending sequence. A process on a datagram socket might receive duplicate messages. Messages that are sent over a datagram socket might be dropped. Record boundaries in the data are preserved. The socket type is SOCK_DGRAM.

Time Protocol implementation: 2.2.2

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.

```java
// File Name GreetingClient.java

import java.net.*;

import java.io.*;

public class GreetingClient {

public static void main(String [] args) {

String serverName = args[0];

int port = Integer.parseInt(args[1]);

try {

System.out.println("Connecting to " + serverName + " on port " + port);

Socket client = new Socket(serverName, port);

System.out.println("Just connected to " + client.getRemoteSocketAddress());

OutputStream outToServer = client.getOutputStream();

DataOutputStream out = new DataOutputStream(outToServer);


out.writeUTF("Hello from " + client.getLocalSocketAddress());

InputStream inFromServer = client.getInputStream();

DataInputStream in = new DataInputStream(inFromServer);

System.out.println("Server says " + in.readUTF());

client.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}
```

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 command-line argument –

```java
// File Name GreetingServer.java
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());
```

```java
        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:

```
$ java GreetingClient localhost 6066
Connecting to localhost on port 6066
Just connected to localhost/127.0.0.1:6066
Server says Thank you for connecting to /127.0.0.1:6066
Goodbye!
```

Conclusion: 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.