Mawlana Bhashani Science and Technology University

Lab-Report

Report No: 07

Course code:ICT-3110

Course title: Operating Systems Lab

Lab Report Name: Socket Programming(Echo protocol)

Date of Performance:

Date of Submission: 30/06/2021

Submitted by

Name:Golam Kibria Tuhin

ID:IT-18015

3th year 1ndsemester

Session: 2017-2018

Dept. of ICT

MBSTU.

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.

Question 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.

Question 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
#In IPv6 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).

Question 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

Question 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

Question 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.

Echo Protocol implementation: 2.2.3

Client Program:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

class Client {
   public static void main(String[] args) throws IOException {
      Scanner s = new Scanner(System.in);
      String serverHostname;
      System.out.println("Enter an IP value: ");
      serverHostname = s.next();
      if (args.length > 0) serverHostname = args[0];
      System.out.println ("Attemping to connect to host "
            + serverHostname + " on port 10007.");
      Socket echoSocket = null;
      PrintWriter out = null;
      BufferedReader in = null;
      try {
         echoSocket = new Socket(serverHostname, 10007);
         out = new PrintWriter(echoSocket.getOutputStream(), true);
         in = new BufferedReader(new InputStreamReader(

               echoSocket.getInputStream()));
      } catch (UnknownHostException e) {
```

```
            System.err.println("Don't know about host: " + serverHostname);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for "
                    + "the connection to: " + serverHostname);
            System.exit(1);
        }
        BufferedReader stdIn = new BufferedReader(
                new InputStreamReader(System.in));
        String userInput;
        System.out.print ("input: ");
        while ((userInput = stdIn.readLine()) != null)
        { out.println(userInput);
            System.out.println("echo: " + in.readLine());
            System.out.print ("input: ");
        }

        out.close();
        in.close();
        stdIn.close();
        echoSocket.close();
    }

}
```

Server Program:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

 class EchoServer
{
    public static void main(String[] args) throws IOException
    {
        ServerSocket serverSocket = null;
        try{
            serverSocket = new ServerSocket(10007);
        }
```

```
        catch (IOException e)
        {
           System.err.println("Could not listen on port: 10007.");
           System.exit(1);
        }
        Socket clientSocket = null;
        System.out.println ("Waiting for connection.....");

        try {
           clientSocket = serverSocket.accept();
        }
        catch (IOException e)
        {

           System.err.println("Accept failed.");
           System.exit(1);
        }
        System.out.println ("Connection successful");

        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
              true);
        BufferedReader in = new BufferedReader(
              new InputStreamReader( clientSocket.getInputStream()));
        String cAddress = "";
        String inputLine;
        cAddress = clientSocket.getInetAddress().toString();
        while ((inputLine = in.readLine()) != null)
        {
           System.out.println ("Server: " + inputLine + " " + cAddress + " ");
           out.println(inputLine + " " + cAddress);
           if (inputLine.equals("bye"))
              break;
        }
        out.close();
        in.close();
        clientSocket.close();
        serverSocket.close();
     }

}
```

Conclusion:

The Echo Protocol is a service in the Internet Protocol Suite defined in RFC 862. It was originally proposed for testing and measurement in IP networks. A host may connect to a server that supports the Echo Protocol using the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) on the well-known port number 7. The server sends back an identical copy of the data it received