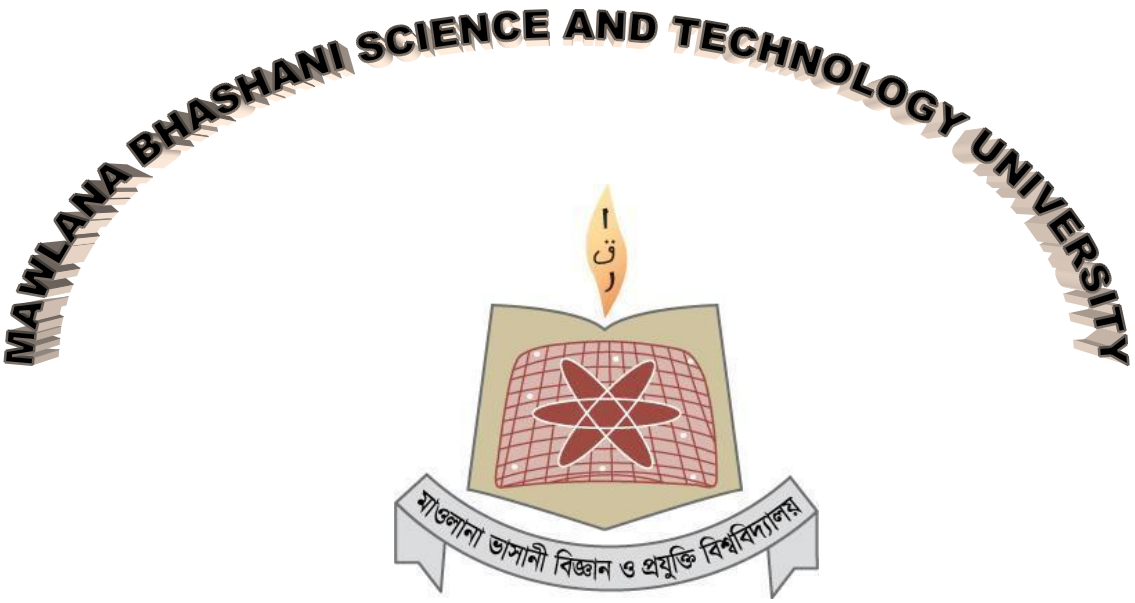


Lab Report: 07

Report Name: Socket Programming-Echo protocol

Course code: ICT-3208
Course title: Computer Networks Lab

Date of Performance: 23/02/2021

Date of Submission: 24/02/2021

SUBMITTED BY

Name: Shakhera khanom
ID: IT-18033
3rd year 2nd semester
Session: 2017-18
Department of ICT,
MBSTU.

SUBMITTED TO

Nazrul Islam
Assistant Professor
Department of ICT,
MBSTU.

Experiment No: 07

Experiment Name: Socket Programming-Echo protocol Implementation.

Objectives:

- To implement a echo server that can handle clients
- Connect server and client
- Sent message to client
- Print input from server

Theory: Echo Protocol: The Echo Protocol is a service in the Internet Protocol Suite defined in RFC 862. It was originally proposed for testing and measurement of round-trip times in IP networks.

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.

Client code:

*EchoClient.java EchoServer.java

```java
package networking;

/**
 *
 * @author SHAKHERA
 */
import java.io.*;
import java.net.*;

public class EchoClient {

    public static void main(String[] args)
    {
        try
        {
            Socket s = new Socket("127.0.0.1", 9999);
            BufferedReader r = new BufferedReader(new InputStreamReader(s.getInputStream()));
            PrintWriter w = new PrintWriter(s.getOutputStream(), true);
            BufferedReader con = new BufferedReader(new InputStreamReader(System.in));
            String line;
            do
            {
                line = r.readLine();
                if ( line != null )
                    System.out.println(line);
                line = con.readLine();
                w.println(line);
            }
            while ( !line.trim().equals("bye") );
        }
        catch (Exception err)
        {
            System.err.println(err);
        }
    }
}
```

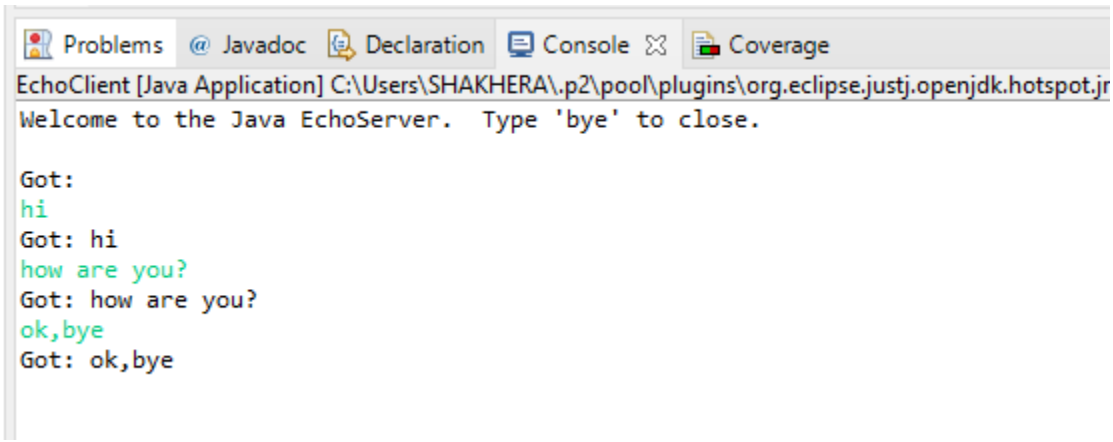
Server code:

*EchoClient.java *EchoServer.java

```
package networking;
/**
 *
 * @author SHAKHERA
 */
import java.io.*;
import java.net.*;
public class EchoServer {
    public EchoServer(int portnum) {
        try {
            server = new ServerSocket(portnum);
        } catch (Exception err) {
            System.out.println(err);
        }
    }
    public void serve() {
        try {
            while (true) {
                Socket client = server.accept();
                BufferedReader r = new BufferedReader(new InputStreamReader(client.getInputStream()));
                PrintWriter w = new PrintWriter(client.getOutputStream(), true);
                w.println("Welcome to the Java EchoServer.  Type 'bye' to close.");
                String line;
                do {
                    line = r.readLine();
                    if (line != null)
                        w.println("Got: " + line);
                } while (!line.trim().equals("bye"));
                client.close();
            }
        } catch (Exception err) {
            System.err.println(err);
        }
    }
    public static void main(String[] args) {
        EchoServer s = new EchoServer(9999);
        s.serve();
    } private ServerSocket server;
}
```

Output:

Compile the client and then the server code



Discussion:

Using this lab, I have to know about Echo Protocol Implementation over the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP).

It was originally proposed for testing and measurement of round-trip times [citation needed] 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.

It reads input from the user on the standard input stream, and then forwards that text to the echo server by writing the text to the socket. The server echoes the input back through the socket to the client. The client program reads and displays the data passed back to it from the server.