

ASSIGNMENT NO 08

Title:

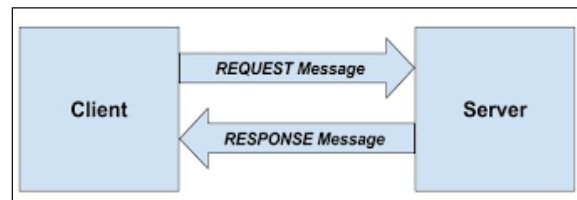
Write a program using TCP socket for wired network for following

- a. Say Hello to Each other
- b. File transfer
- c. Calculator

Theory:

Client-Server Model

Network applications can be divided into two process: a Client and a Server, with a communication link joining the two processes.



Normally, from Client side it is one-one connection. From the Server Side, it is many-one connection. The standard model for network applications is the Client-Server model. A

Server is a process that is waiting to be contacted by a Client process so that server can do something for the client. Typical BSD Sockets applications consist of two separate application level processes; one process (the client) requests a connection and the other process (the server) accepts it.

Client-Server Model Using TCP

TCP Clients sends request to server and server will receives the request and response with acknowledgement. Every time client communicates with server and receive response from it. Algorithm to create client and server process is as below.

ALGORITHM:

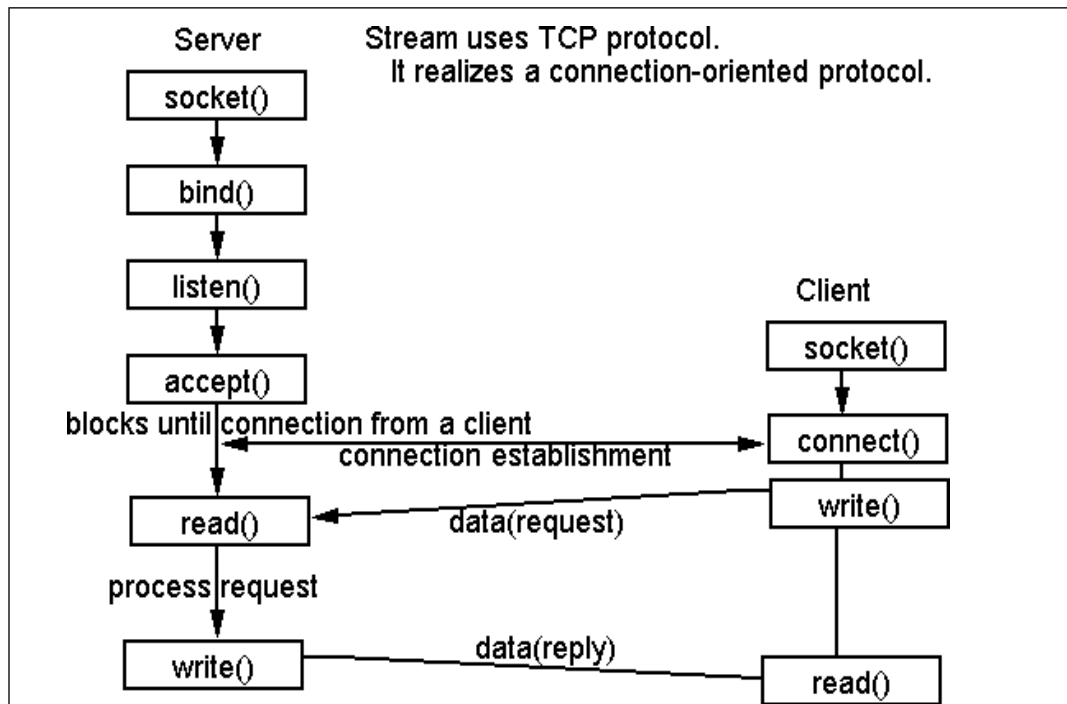
Server

1. Create a server socket and bind it to port
2. Listen for new connection and when a connection arrives, accept it
3. Read Client's message and display it
4. Get a message from user and send it to client
5. Close the server socket
6. Stop

Client

1. Create a client socket and connect it to the server's port number
2. Get a message from user and send it to server
3. Read server's response and display it
4. Close the client socket
5. Stop

Socket functions for TCP client/server in Connection-oriented Scenario



Elementary Socket System Calls

1. socket()	<p>socket() System Call: Creates an end point for communication and returns a descriptor.</p> <pre>#include <sys/socket.h> #include <sys/types.h></pre> <pre>int socket (int Address Family, int <u>Type</u>, int <u>Protocol</u>);</pre> <p><u>Return Values:</u> Upon successful completion, the socket subroutine returns an integer (the socket descriptor). It returns -1 on error.</p>
2. bind()	<p>bind() System call: Binds a name to a socket.</p> <p>Description: The bind subroutine assigns a Name parameter to an unnamed socket. It assigns a local protocol address to a socket.</p> <pre>#include <sys/socket.h></pre> <pre>int bind (int sockfd, struct sockaddr *myaddr, int addrlen);</pre> <p><u>Return Values:</u> Upon successful completion, the bind subroutine returns a value of 0. Otherwise, it returns a value of -1 to the calling program.</p>

3. connect()	<p>connect() System call: The connect function is used by a TCP client to establish a connection with a TCP server.</p> <pre>#include <sys/socket.h></pre> <pre>int connect(int sockfd, struct sockaddr *servaddr, int addrlen);</pre> <p><u>Return Values:</u> Upon successful completion, the connect subroutine returns a value of 0. Otherwise, it returns a value of -1 to the calling program.</p>
4. listen()	<p>listen() System call: This system call is used by a connection-oriented server to indicate that it is willing to receive connections.</p> <pre>#include <sys/socket.h></pre> <pre>int listen (int sockfd, int backlog);</pre> <p><u>Return values:</u> Returns 0 if OK, -1 on error</p>
5. accept()	<p>accept() System call: The actual connection from some client process is waited for by having the server execute the accept system call.</p> <pre>#include <sys/socket.h></pre> <pre>int accept (int sockfd, struct sockaddr *cliaddr, int *addrlen);</pre> <p><u>Return Values:</u> This system call returns up to three values: an integer return code that is either a new socket descriptor or an error indication, the protocol address of the client process (through the cliaddr pointer), and the size of this address (through the addrlen pointer).</p>
6. read() and write()	<p>read() and write() System call: - read/write from a file descriptor</p> <pre>#include <unistd.h></pre> <pre>ssize_t read(int fd, void *buf, size_t count); ssize_t write(int fd, const void *buf, size_t count);</pre> <p><u>Return Values:</u> On success, the number of bytes read or written is returned (zero indicates nothing was written/read). On error, -1 is returned.</p>
7. Send(), sendto(), recv()	<p>Send(), sendto(), recv() and recvfrom() system calls:</p> <p>These system calls are similar to the standard read and write functions, but one additional argument is required.</p>
and recvfrom()	<pre>#include <sys/socket.h></pre> <pre>int send(int sockfd, char *buff, int nbytes, int flags);</pre> <pre>int sendto(int sockfd, char void *buff, int nbytes, int flags, struct sockaddr *to, int addrlen);</pre> <pre>int recv(int sockfd, char *buff, int nbytes, int flags);</pre> <pre>int recvfrom(int sockfd, char *buff, int nbytes, int flags, struct sockaddr *from, int *addrlen);</pre> <p>The first three arguments, sockfd, buff and nbytes are the same as the first three arguments to read and write. The flags argument is either 0 or is formed by logically OR'ing one or more of the constants.</p>

	Return Values: All four system calls return the length of the data that was written or read as the value of the function. Otherwise it returns, -1 on error.
8. Close()	Close() system call: The normal Unix close function is also used to close a socket and terminate a TCP connection. #include <unistd.h> int close (int sockfd);

Testing

1. Run Wireshark tool
2. Run client and server program
3. Exchange message
4. Capture TCP packets in Wireshark.

Program:

TCPSocketServer.java

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;

public class TCPSocketServer {
    private ServerSocket serverSocket = null;
    private Socket socket = null;
    private InputStream inStream = null;
    private OutputStream ouStream = null;
    public TCPSocketServer() { }
    public void communicate() {
        try {
            serverSocket = new ServerSocket(4545);
            socket = serverSocket.accept();
            inStream = socket.getInputStream();
            ouStream = socket.getOutputStream();
```

```

        while (socket.isConnected()) {
            byte[] readBuffer = new byte[1024];
            int numBytes = inStream.read(readBuffer);
            byte[] tempBuffer = new byte[numBytes];
            System.arraycopy(readBuffer, 0, tempBuffer, 0, numBytes);
            String message = new String(tempBuffer, "UTF-8");
            System.out.println("message from client = " + message);
            String reply = "Thank you !!.This is from server";
            byte[] replyBytes = reply.getBytes();
            ouStream.write(replyBytes);
            System.out.println("reply has written to socket");
            Thread.sleep(2000);
            socket.close();
        }
    }

    catch (SocketException se) {
        System.exit(0);
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

public static void main(String[] args) {
    TCPSocketServer server = new TCPSocketServer();
    server.communicate();
}
}

```

TCPSocketClient.java

```

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

```

```
import java.net.Socket;
import java.net.SocketException;

public class TCPSocketClient {
    private Socket socket = null;
    private InputStream inStream = null;
    private OutputStream ouStream = null;

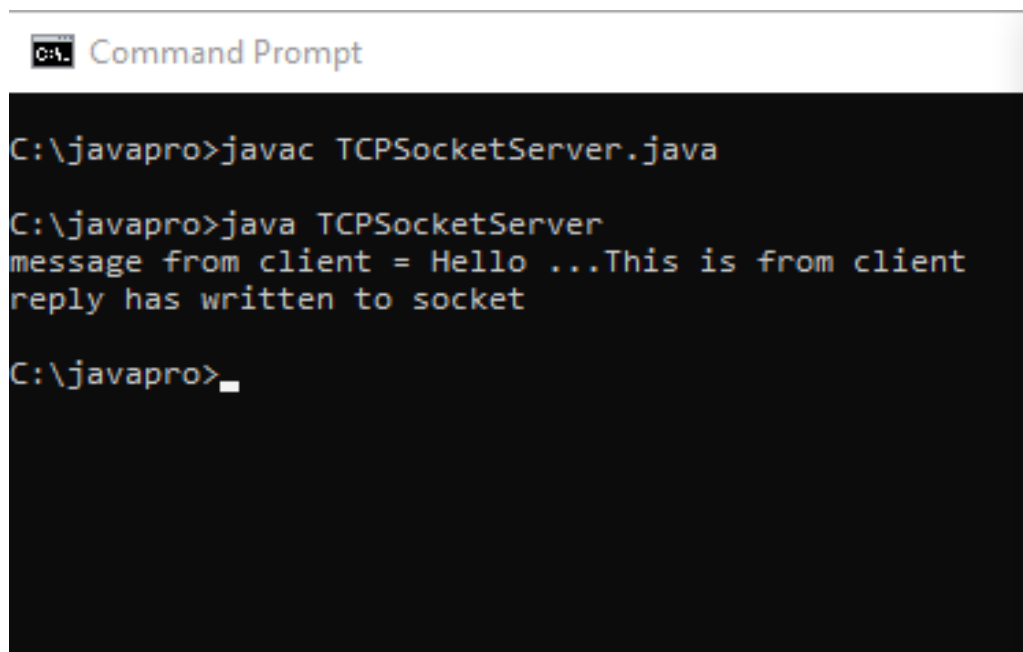
    public TCPSocketClient() {

    }

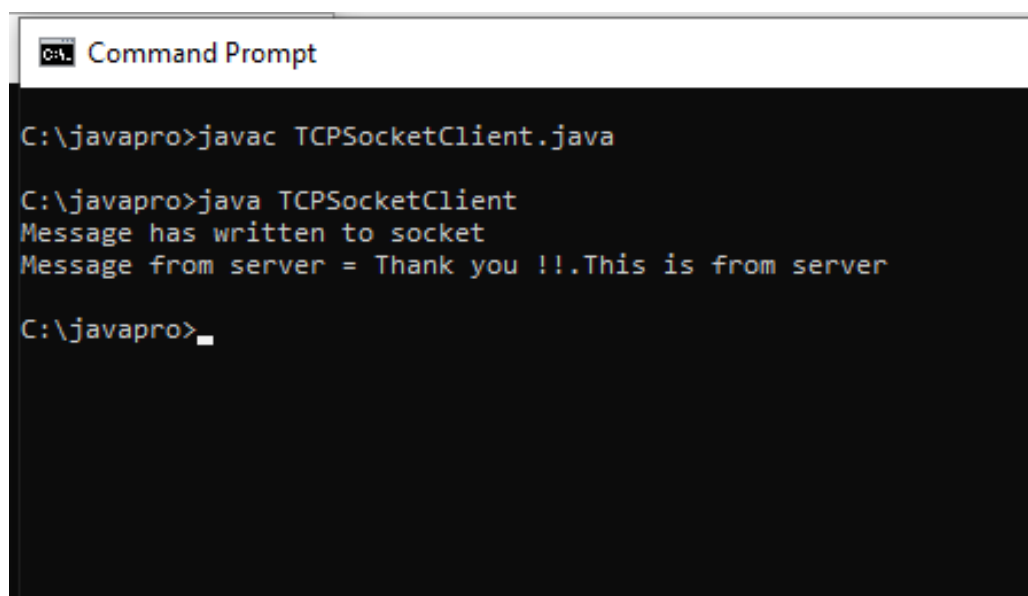
    public void communicate() {
        try {
            socket = new Socket("localhost", 4545);
            inStream = socket.getInputStream();
            ouStream = socket.getOutputStream();
            while (socket.isConnected()) {
                String messageString = "Hello ...This is from client";
                ouStream.write(messageString.getBytes());
                System.out.println("Message has written to socket");
                byte[] byteBuffer = new byte[1024];
                int numBytes = inStream.read(byteBuffer);
                byte[] tempBuffer = new byte[numBytes];
                System.arraycopy(byteBuffer, 0, tempBuffer, 0, numBytes);
                String message = new String(tempBuffer, "UTF-8");
                System.out.println("Message from server = " + message);
                Thread.sleep(2000);
                socket.close();
            }
        } catch (SocketException se) {
            System.exit(0);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
} catch (InterruptedException e) {  
e.printStackTrace();  
}  
}  
  
public static void main(String[] args) {  
TCPSocketClient client = new TCPSocketClient();  
client.communicate();  
}  
}
```

Output:



```
C:\javapro>javac TCPSocketServer.java  
  
C:\javapro>java TCPSocketServer  
message from client = Hello ...This is from client  
reply has written to socket  
  
C:\javapro>_
```



```
C:\javapro>javac TCPSocketClient.java  
  
C:\javapro>java TCPSocketClient  
Message has written to socket  
Message from server = Thank you !!.This is from server  
  
C:\javapro>_
```

No.	Time	Source	Destination	Protocol	Length	Info
5	0.071970	2402:e280:3e16:16ca...	2620:1ec:c11::200	TCP	74	54337 → 443 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
6	2.969728	20.198.118.190	192.168.1.5	TCP	54	443 → 53860 [ACK] Seq=1 Ack=1 Win=7374 Len=0
7	2.969920	192.168.1.5	20.198.118.190	TCP	54	[TCP ACKed unseen segment] 53860 → 443 [ACK] Seq=1 Ack=2 Win=508 Len=0
11	3.396832	2402:e280:3e16:16ca...	2404:6000:4003:c04::	TCP	75	53965 → 5228 [ACK] Seq=1 Ack=1 Win=510 Len=1
13	3.545403	2404:6000:4003:c04::	2402:e280:3e16:16ca...	TCP	86	5228 → 53965 [ACK] Seq=1 Ack=2 Win=265 Len=0 SLE=1 SRE=2
33	12.288428	52.231.199.126	192.168.1.5	TCP	54	443 → 54345 [ACK] Seq=1 Ack=1 Win=2048 Len=0
34	12.288428	52.231.199.126	192.168.1.5	TCP	54	443 → 54346 [ACK] Seq=1 Ack=1 Win=2050 Len=0
35	12.288691	192.168.1.5	52.231.199.126	TCP	54	[TCP ACKed unseen segment] 54345 → 443 [ACK] Seq=1 Ack=2 Win=509 Len=0
36	12.288839	192.168.1.5	52.231.199.126	TCP	54	[TCP ACKed unseen segment] 54346 → 443 [ACK] Seq=1 Ack=2 Win=512 Len=0
58	18.535141	117.18.237.29	192.168.1.5	TCP	54	80 → 54336 [ACK] Seq=1 Ack=1 Win=60 Len=0
61	18.536091	192.168.1.5	117.18.237.29	TCP	54	[TCP ACKed unseen segment] 54336 → 80 [ACK] Seq=1 Ack=2 Win=509 Len=0

> Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{08DEAEDA-E451-41EE-890E-18A966408F50}, id 0

> Ethernet II, Src: IntelCor_6c:39:a5 (cc:2f:71:6c:39:a5), Dst: TaicangT_65:92:8c (40:33:06:65:92:8c)

> Internet Protocol Version 6, Src: 2402:e280:3e16:16ca:5961:ea:f1f54:2e, Dst: 2620:1ec:c11::200

> Transmission Control Protocol, Src Port: 54337, Dst Port: 443, Seq: 1, Ack: 1, Len: 0

0000	40 33 06 65 92 8c cc 2f 71 6c 39 a5 86 dd 60 04	@3.e.../ q19...
0010	ad 4d 00 14 06 40 24 02 e2 80 3e 16 16 ca 59 61	.M...@\$. -->...Ya
0020	0e af 1f 54 00 2e 25 20 01 ec 0c 11 00 00 00 00	...T...8
0030	00 00 00 00 02 00 d4 41 01 bb 3f f5 62 42 28 c4A --? bb(-
0040	25 41 50 14 00 00 d0 84 00 00	%AP.....

Conclusion:

Hence we have studied TCP Socket Programming in C for echo message, file transfer, arithmetic and trigonometric calculator and captured TCP packets in Wireshark tool.