MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY,

Santosh, Tangail -1902

Lab Report No	**:**	03
Lab Report Name	**:**	Socket Programming Implementation
Course Name	**:**	Computer Networks Lab

Submitted by,

Name : Sabrin Afroz

ID : IT-17007

Session : 2016-17

Dept. of ICT, MBSTU.

Submitted to,

Nazrul Islam

Assistant Professor

Dept. of ICT, MBSTU.

Socket Programming : Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connection-less.

TCP client-server program : Socket and ServerSocket classes are used for connection-oriented socket programming.

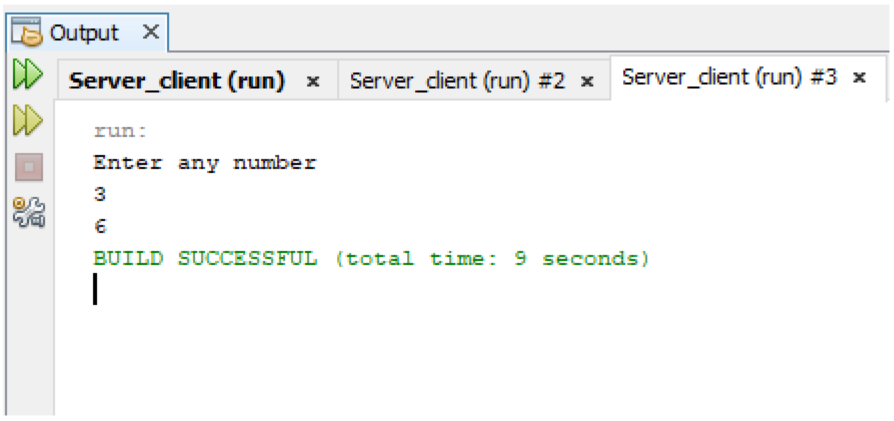
TCP_client.java

```
package MyPackage;


import java.io.IOException;

import java.io.PrintStream;

import java.net.Socket;

import java.util.Scanner;


public class TCP_client {

public static void main(String[] args) throws IOException {

 int number, temp;


 Scanner sc = new Scanner(System.in);

 Socket s = new Socket("127.0.0.1",1312);

 Scanner sc1 = new Scanner(s.getInputStream());

  System.out.println("Enter any number");

  number = sc.nextInt();

  PrintStream p = new PrintStream(s.getOutputStream());

  p.println(number);

  temp = sc1.nextInt();

  System.out.println(temp);

}

}
```

TCP_server.java

```
package MyPackage;

import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class TCP_server {
 public static void main(String[] args) throws IOException{
    int number,temp;
    ServerSocket s1 = new ServerSocket(1312);
    Socket ss = s1.accept();
    Scanner sc = new Scanner(ss.getInputStream());
    number = sc.nextInt();
    temp = number * 2;
    PrintStream p = new PrintStream(ss.getOutputStream());
    p.println(temp);
}
}
```

Output :



```
run:
Enter any number
3
6
BUILD SUCCESSFUL (total time: 9 seconds)
```

UDP client-server program : DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

UDP_client.java

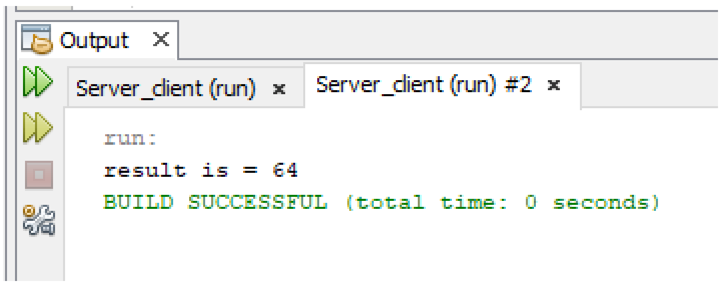
```
package MyPackage;

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;


public class UDP_client {

  public static void main(String[] args) throws Exception {

    DatagramSocket ds = new DatagramSocket();


    int i =8;

    byte[] b = String.valueOf(i).getBytes();


    InetAddress ia = InetAddress.getLocalHost();

    DatagramPacket dp = new DatagramPacket(b,b.length,ia,9999);

    ds.send(dp);
```

```
        byte[] b1 = new byte[1024];
        DatagramPacket dp1 = new DatagramPacket(b1,b1.length);
        ds.receive(dp1);

        String str = new String(dp1.getData(),0,dp1.getLength());
        System.out.println("result is = "+str);
    }
}
```

UDP_server.java

```
package MyPackage;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class UDP_server {
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket(9999);

        byte[] b1 = new byte[1024];

        DatagramPacket dp = new DatagramPacket(b1,b1.length);
        ds.receive(dp);
        String str = new String(dp.getData());
        int num = Integer.parseInt(str.trim());

        int result = num*num;

        byte[] b2 = (result + "").getBytes();

        InetAddress ia = InetAddress.getLocalHost();

        DatagramPacket dp1 = new DatagramPacket(b2,b2.length,ia,dp.getPort());

        ds.send(dp1);


    }

}
```

Output :



```
run:
result is = 64
BUILD SUCCESSFUL (total time: 0 seconds)
```

Time Protocol :

This protocol provides a site-independent, machine readable date and time. The Time service sends back to the originating source the time in seconds since midnight on January first 1900. This protocol may be used either above the Transmission Control Protocol (TCP) or above the User Datagram Protocol (UDP).

When used via **TCP** the time service works as follows:

S: Listen on port 37 (45 octal).

U: Connect to port 37.

S: Send the time as a 32 bit binary number.

U: Receive the time.

U: Close the connection.

S: Close the connection.

The server listens for a connection on port 37. When the connection is established, the server returns a 32-bit time value and closes the connection. If the server is unable to determine the time at its site, it should either refuse the connection or close it without sending anything.

When used via **UDP** the time service works as follows:

S: Listen on port 37 (45 octal).

U: Send an empty datagram to port 37.

S: Receive the empty datagram.

S: Send a datagram containing the time as a 32 bit binary number.

U: Receive the time datagram.

The server listens for a datagram on port 37. When a datagram arrives, the server returns a datagram containing the 32-bit time value. If the server is unable to determine the time at its site, it should discard the arriving datagram and make no reply.

TFTP : TFTP uses client and server software to make connections between two devices. From a TFTP client, individual files can be copied (uploaded) to or downloaded from the server. The server hosts the files and the client requests or sends files. TFTP relies on UDP to transport data.

TFTPServer.java

```
package tftpserver;

import java.net.*;

import java.util.Hashtable;

import java.io.*;
import tftp.*;

public class TFTPServer {
static final int MAX_CONNECTIONS = 10;
protected String ftproot;
protected int port;
protected ServerSocket serverSocket;
protected Socket clientSocket = null;
protected Hashtable<Socket, TFTPServerClient> clients;
public TFTPServer(int port, String ftproot) {
this.port = port;
this.ftproot = ftproot;
clients = new Hashtable<Socket, TFTPServerClient>();
try {
serverSocket = new ServerSocket(port);
} catch (IOException e) {
TFTPUtils.fatalError("Could not listen on port: " + port);
}
TFTPUtils.puts("Server successfully started. Listening on port " + port + " started");
while (true) {
try {
clientSocket = serverSocket.accept();
} catch (Exception e) {
TFTPUtils.fatalError("An exeption was thrown while accepting a client connection");
}
if (clients.size() == MAX_CONNECTIONS) {
TFTPPacket packet_error = new TFTPPacket();
```

```
packet_error.createError(0, "Server is full, please try again later");
try {
packet_error.sendPacket(new BufferedOutputStream(clientSocket.getOutputStream()));
} catch (Exception e) {
TFTPUtils.puts("Unable to send error message to client");
}
} else {
clients.put(clientSocket, new TFTPServerClient(this, clientSocket));
}
}
}
public void removeConnection(Socket clientSocket) {
TFTPUtils.puts("Removing client connection " + clientSocket.getInetAddress().getHostAddress());
if (clients.get(clientSocket) != null) {
try {
clients.get(clientSocket).fin.close();
} catch (Exception ie) {}
try {
clients.get(clientSocket).fout.close();
} catch (Exception ie) {}
try {
clients.get(clientSocket).in.close();
} catch (Exception ie) {}
try {
clients.get(clientSocket).out.close();
} catch (Exception ie) {}
try {
```

```
clients.get(clientSocket).interrupt.close();
} catch (Exception ie) {}
}
try {
clientSocket.close();
} catch (Exception ie) {}
try {
clients.remove(clientSocket);
} catch (Exception ie) {}
}

public void shutdown() {
try {
serverSocket.close();
} catch (IOException e) {
}
}
}
```

TFTPClient.java

```
package MyPackage;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.SocketAddress;
import java.nio.ByteBuffer;

import java.nio.channels.DatagramChannel;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.nio.file.StandardOpenOption;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Set;


public class TFTPClient{

        private String TFTP_SERVER_ADDRESS = "192.168.1.11";

        private int TFTP_SERVER_PORT = 69;

        private Selector selector;

        private List<DatagramChannel> datagramChannels;

        private SocketAddress socketAddress;

        private int fileCount;


        private final byte OP_RRQ = 1;

        private final byte OP_DATAPACKET = 3;

        private final byte OP_ACK = 4;

        private final byte OP_ERROR = 5;

        private final int PACKET_SIZE = 1024;
```

```
public static void main(String args[]) throws Exception {
        TFTPClientNIO tFTPClientNio = new TFTPClientNIO();
        String[] files = { "demo.txt", "TFTP.pdf" };
        tFTPClientNio.getFiles(files);
}

public void getFiles(String[] files) throws IOException {
        registerAndRequest(files);
        readFiles();
}
private void registerAndRequest(String[] files) throws IOException {
        fileCount = files.length;
        selector = Selector.open();
        socketAddress = new InetSocketAddress(TFTP_SERVER_ADDRESS,
                        TFTP_SERVER_PORT);
        datagramChannels = new ArrayList<DatagramChannel>();
        for (int i = 0; i < fileCount; i++) {
                DatagramChannel dc = DatagramChannel.open();
                dc.configureBlocking(false);
                SelectionKey selectionKey = dc.register(selector,
                                SelectionKey.OP_READ);
                selectionKey.attach(files[i]);
                sendRequest(files[i], dc);
                datagramChannels.add(dc);
        }
}
```

```
private void sendRequest(String fileName, DatagramChannel dChannel)
                throws IOException {
        String mode = "octet";
        ByteBuffer rrqByteBuffer = createRequest(OP_RRQ, fileName, mode);
        System.out.println("Sending Request to TFTP Server.");
        dChannel.send(rrqByteBuffer, socketAddress);
        System.out.println("Request Sent Success.");
}

private void readFiles() throws IOException {
        int counter = 0;
        while (counter < fileCount) {
                counter++;
                int readyChannels = selector.select();
                if (readyChannels == 0)
                        continue;
                Set<SelectionKey> selectedKeys = selector.selectedKeys();
                Iterator<SelectionKey> keyIterator = selectedKeys.iterator();
                while (keyIterator.hasNext()) {
                        SelectionKey key = keyIterator.next();
                        if (key.isAcceptable()) {
                                System.out.println("connection accepted: " +
                               key.channel());
                        } else if (key.isConnectable()) {
                                System.out.println("connection established: "
                                                + key.channel());
                        } else if (key.isReadable()) {
```

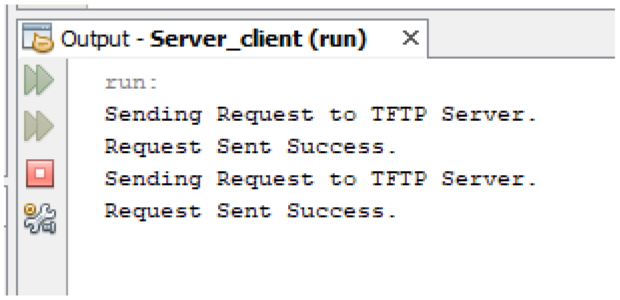
```
                        System.out.println("Channel Readable: " + key.channel());
                        receiveFile((DatagramChannel) key.channel(),
                                        (String) key.attachment());
                        System.out.println("File received.");
                    } else if (key.isWritable()) {
                        System.out.println("writable: " + key.channel());
                    }
                    keyIterator.remove();
                }
            }
        }
        private void receiveFile(DatagramChannel dc, String fileName)
                throws IOException {
            ByteBuffer dst = null;
            do {
                dst = ByteBuffer.allocateDirect(PACKET_SIZE);
                SocketAddress remoteSocketAddress = dc.receive(dst);
                byte[] opCode = { dst.get(0), dst.get(1) };
                if (opCode[1] == OP_ERROR) {
                    System.out.println("Type of PACKET Received is ERROR!");
                } else if (opCode[1] == OP_DATAPACKET) {
                    System.out.println("Type of PACKET Received is DATA.");

                    byte[] packetBlockNumber = { dst.get(2), dst.get(3) };
                    System.out.println("Packet Block Number: "
                            + packetBlockNumber[0] + ", " +
                     packetBlockNumber[1]);
```

```
                        readPacketData(dst, fileName);
                        sendAcknowledgment(packetBlockNumber,
                    remoteSocketAddress, dc);
                }
            } while (!isLastPacket(dst));
        }
        private byte[] readPacketData(ByteBuffer dst, String fileName)
                throws IOException {
            System.out.println("Read contents of this packet..");
            byte fileContent[] = new byte[PACKET_SIZE];
            dst.flip();
            int m = 0, counter = 0;
            while (dst.hasRemaining()) {

                if (counter > 3) {
                    fileContent[m] = dst.get();
                    m++;
                } else {
                    dst.get();
                }
                counter++;
            }
            System.out.println("Packet Reading Done.");
            Path filePath = Paths.get(fileName);
            byte[] toWrite = new byte[m];
            System.arraycopy(fileContent, 0, toWrite, 0, m);
```

```
		writeToFile(filePath, toWrite);

		return fileContent;

	}

	private void writeToFile(Path filePath, byte[] toWrite) throws IOException {

		if (Files.exists(filePath)) {

			Files.write(filePath, toWrite, StandardOpenOption.APPEND);

		} else {

			Files.write(filePath, toWrite, StandardOpenOption.CREATE);

		}

	}

	private boolean isLastPacket(ByteBuffer bb) {

		if (bb.limit() < 512)

			return true;

		else

			return false;

	}

	private void sendAcknowledgment(byte[] blockNumber,

			SocketAddress socketAddress, DatagramChannel dc) throws IOException
{

		System.out.println("Sending ACK.. " + blockNumber[0] + ", "

				+ blockNumber[1]);

		byte[] ACK = { 0, OP_ACK, blockNumber[0], blockNumber[1] };

		dc.send(ByteBuffer.wrap(ACK), socketAddress);

		System.out.println("Acknowledgement Sent");

	}

	private ByteBuffer createRequest(final byte opCode, final String fileName,

			final String mode) {
```

```
			byte zeroByte = 0;
			int rrqByteLength = 2 + fileName.length() + 1 + mode.length() + 1;
			byte[] rrqByteArray = new byte[rrqByteLength];


			int position = 0;
			rrqByteArray[position] = zeroByte;
			position++;
			rrqByteArray[position] = opCode;
			position++;
			for (int i = 0; i < fileName.length(); i++) {
				rrqByteArray[position] = (byte) fileName.charAt(i);
				position++;
			}
			rrqByteArray[position] = zeroByte;
			position++;
			for (int i = 0; i < mode.length(); i++) {
				rrqByteArray[position] = (byte) mode.charAt(i);
				position++;
			}
			rrqByteArray[position] = zeroByte;
			ByteBuffer byteBuffer = ByteBuffer.wrap(rrqByteArray);
			return byteBuffer;
		}

	}
```

Output:



```
run:
Sending Request to TFTP Server.
Request Sent Success.
Sending Request to TFTP Server.
Request Sent Success.
```

Conclusion :

The socket programming will provide the ability of the implement in analytics, streaming in binary, document collaboration and so on. On the positive side, The IO can control the connection in sockets. For this reason, both the server and also client side is consists of IO libraries. It makes the real-time application are feasible in the mobile devices in Embedded Operating System.On the Internet, stream sockets are typically implemented using TCP so that applications can run across any networks using TCP/IP protocol. Raw sockets. Allow direct sending and receiving of IP packets without any protocol-specific transport layer formatting.Socket programming (not to be confused with web sockets) is the fundamental way in which any software can access TCP/IP network and communicate with another applications over the network.So essentially, web browsers are built using socket programming and so do are web servers built. Web servers use sockets to listen on tcp port 80 waiting for an incoming connection. Web browser connects to web server using socket and sends request to download a web page.