Page No.
Date

# Assignment - 05

* Title: UDP Socket Programming

* Problem Statement: Write a program using UDP sockets to enable file transfer (Script, Text, Audio and Video one file each) between two machines. Demonstrate the packets captured traces using wireshark Packet Analyzer Tool for peer to peer mode.

* Objectives:
  - Getting familiar with the client-server communication model.
  - Designing simple client or server application for datagram
  - Learn important libraries & methods, classes used for network programming.

* Hardware & Software Required:
  Java SE-11, IntelliJ IDE, Windows 10 (64 bit).

* Theory:

User Datagram Protocol (UDP):

UDP is a simple transport layer protocol. The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram which is further encapsulated in an IP datagram which is sent to the destination.

There is no guarantee that a UDP will reach the destination and that the order of the datagrams will be preserved across the network or that datagrams arrive only once

Teacher's Sign: _____

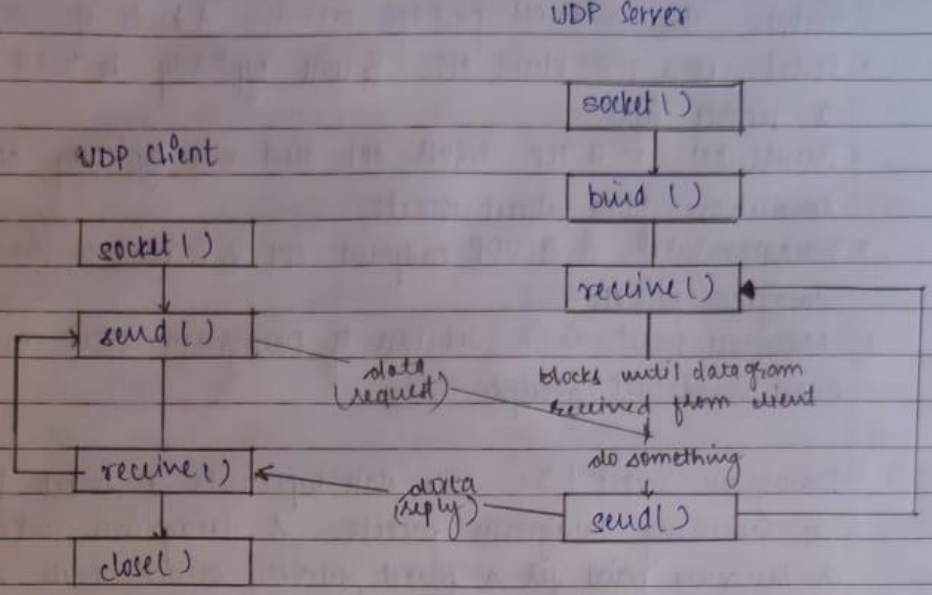
The problem of UDP is its lack of reliability if a datagram reaches its final distination but the check sum detects an error of it the datagram is dropped in network, It is not automatically retransmitted. Each UDP datagram is passed to the receiving application alongwith the data.

- Client - Server model and UDP:

The client - server model is one of the most used communication paradigms in communicated network systems. Clients normally communicate with one server at a time. From a server's perspective at any point in time, it is not unusual for a server to be communicating with multiple clients. Client need to know of the existence of & the address of the server, but the server does not need to know address of client prior to connection being established.

Following figure shows interaction between a UDP and client & server. First of all, client doesn't establish a connection with server. Instead it sends a datagram to server using sendto function which requires address of distination as a parameter. Similarly server does not accept connection from client. Instead server just calls receive() from, which waits until data arrives from some client. Receive() returns IP address and port of client, along with datagram. so server can send a response to client.

Teacher's Sign.: _____




- Java.net Package:

Java net package provides the classes for implementing networking applications. The Java.net package can be roughly divided into 2 sections.

- A low level API, which deals with the following abstraction
  - Addresses, which are networking identifiers like IP addresses
  - sockets, which are networking basic bidirectional data communication
- A High level API which deals with the following abstraction
  - URI's, • URL's, • connections which represents connections to the resource pointed by URLs.

Addresses: Addresses are used throughout Java.net APIs as either host identifiers or socket end point identifiers. The Inet Adress class is the abstraction representing an IP address. It has two subclasses:

- Inet 4 Address for IPv4 addresses
- Inet 6 Address for IPv6 addresses.

Page No.
Date

Sockets: The java.net package provides 4 kinds of sockets:

- Socket: is a TCP client API & will typically be used to connect to a remote host.
- ServerSocket: is a TCP server API and will typically accepts connection from client sockets.
- DatagramSocket: is a UDP endpoint API & is used to send & receive datagram packets.
- Multicast Socket: is a subclass of Datagram socket used when dealing with multicast groups.

- DatagramSocket(): This class represents a socket for sending & receiving datagram packets. A datagram socket is sending or receiving point for a packet delivery service. Each packet sent or received on it individually addressed and routed.
eg: DatagramSocket socket = new DatagramSocket(8888); will create a Datagram Socket able to receive broadcasts UDP Port 8888.

- DatagramPacket(): They are used to implement a connection less packet delivery service. Each message is routed from one machine to another based solely on information contained within packet. Multiple packets sent from one to another machine might be routed differently & might arrive in any order packet delivery is not guaranteed.
eg: DatagramPacket(byte[] buf, int length): constructs datagram packet for receiving packets of length 'length'.

DatagramPacket(byte[] buf, int length, InetAddress address, int port): constructs a packet for sending packets of length 'length' to specified port number on specified host.

Teacher's Sign.:

Page No.
Date

Methods:

- InetAddress getAddress(): returns IP address of machine to which this datagram is being sent or from which was received.
- int getLength(): returns length of data to be sent or received
- int getPort(): returns port number on remote host to which this datagram is being sent or from which was received.
- SocketAddress getSocketAddress(): gets socket address of remote host that this packet is being sent to or is coming from.
- DatagramSocket send(DatagramPacket): sends a datagram packet from socket. The packet includes information indicating data to be sent, its length, IP add. of remote host and it's portno.
- DatagramSocket receive(DatagramPacket): receives a packet from socket; when this method returns the Datagram packet's buffer is filled with the data received. The packet also contains sender's IP add and it's portnumber.
- DatagramSocket close(): closes datagram socket. Any thread currently blocked in receive() upon the call to close() on the socket will throw a SocketException.

* Algorithm:

a) server:

i) create a new DatagramSocket & bind at localhost, port
ii) create a datagrampacket an receive file name to be received along with file extension
iii) get sender's socket add. from above packet in previous step
iv) Receive total no of packets to be received in total (let's say 'n')
v) for i=0 to n repeat:
   - request $i^{th}$ packet from sender
   - receive sent packet and write it to a file in sequence
vi) when i reaches n (all packets received) send completion acknowledgement to the sender.

Teacher's Sign.: _____

Page No.
Date

b) Client:

i) creates a new DatagramSocket (null) & get receiver's InetAddress & port from user. also the filepath to be sent.

ii) send file name to receiver

iii) open file in read mode and calculate the no. of packets to be sent (i.e. filesize / packetsize)

iv) Send no. of packets calculated in the previous step to server (receiver) using datagram packet with InetAddress & port

v) Break the file into packets and store it into sequenced data structure.

vi) while completion ack not received repeat: - receive packet sequence number from receiver
- send the packet [sequence no.] to the receiver

vii) When completion ack received close the socket & print no. of bytes sent.

* Conclusion: In this assignment we studied about client-server communication model and UDP thus implemented client-server program to send and receive files (script, text, audio & video) using UDP for machine connected in same wired network using fastEthernet.

Teacher's Sign.: _____

# Code:

## Server:

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {
    private static DataOutputStream dataOutputStream = null;
    private static DataInputStream dataInputStream = null;

    public static void main(String[] args) {
        try(ServerSocket serverSocket = new ServerSocket(5000)){
            System.out.println("listening to port:5000");
            Socket clientSocket = serverSocket.accept();
            System.out.println(clientSocket+" connected.");
            dataInputStream = new DataInputStream(clientSocket.getInputStream());
            dataOutputStream = new
DataOutputStream(clientSocket.getOutputStream());

            receiveFile("D:/Dhcp1.mp4");
            receiveFile("D:/server.txt");
            receiveFile("D:/sample1.mp3");
            receiveFile("D:/script.dat");

            dataInputStream.close();
            dataOutputStream.close();
            clientSocket.close();
        } catch (Exception e){
            e.printStackTrace();
        }
    }

    private static void receiveFile(String fileName) throws Exception{
        int bytes = 0;
        FileOutputStream fileOutputStream = new FileOutputStream(fileName);

        long size = dataInputStream.readLong();     // read file size
        byte[] buffer = new byte[4*1024];
        while (size > 0 && (bytes = dataInputStream.read(buffer, 0,
(int)Math.min(buffer.length, size))) != -1) {
            fileOutputStream.write(buffer,0,bytes);
            size -= bytes;      // read upto file size
        }
        fileOutputStream.close();
    }
}
```

## Client:

```
import java.io.*;
import java.net.Socket;
```

```java
public class Client {
    private static DataOutputStream dataOutputStream = null;
    private static DataInputStream dataInputStream = null;

    public static void main(String[] args) {
        try(Socket socket = new Socket("localhost",5000)) {
            dataInputStream = new DataInputStream(socket.getInputStream());
            dataOutputStream = new DataOutputStream(socket.getOutputStream());

            sendFile("D:/dhcp.mp4");
            sendFile("D:/client.txt");
            sendFile("D:/sample.mp3");
            sendFile("D:/UKLog.dat");

            dataInputStream.close();
            dataInputStream.close();
        }catch (Exception e){
            e.printStackTrace();
        }
    }

    private static void sendFile(String path) throws Exception{
        int bytes = 0;
        File file = new File(path);
        FileInputStream fileInputStream = new FileInputStream(file);

        // send file size
        dataOutputStream.writeLong(file.length());
        // break file into chunks
        byte[] buffer = new byte[4*1024];
        while ((bytes=fileInputStream.read(buffer))!=-1){
            dataOutputStream.write(buffer,0,bytes);
            dataOutputStream.flush();
        }
        fileInputStream.close();
    }
}
```

# Ouput:

## Server:

listening to port:5000

Socket[addr=/127.0.0.1,port=64861,localport=5000] connected.

File Transfer Complete!

Process finished with exit code 0

## Client:

File Transfer Complete!

Process finished with exit code 0