|  |  |
| --- | --- |
| **Ex No : 01** | **Learning to use basic commands and traceroute PDUs** |
| **DATE :** |

**AIM:**

To learn to use commands like tcpdump, netstat, ifconfig, nslookup and tracerouteping.

**COMMANDS:**

**1.Tcpdump:**

**Display traffic between 2 hosts:**

To display all traffic between two hosts (represented by variables host1 and host2): # tcpdump host host1 and host2

**Display traffic from a source or destination host only:**

To display traffic from only a source (src) or destination (dst) host: # tcpdump src host

# tcpdump dst host

**Display traffic for a specific protocol**

Provide the protocol as an argument to display only traffic for a specific protocol, for example tcp, udp, icmp, arp

# tcpdump protocol

For example to display traffic only for the tcp traffic: # tcpdump tcp

**Filtering based on source or destination port** To

filter based on a source or destination port: # tcpdump src port ftp

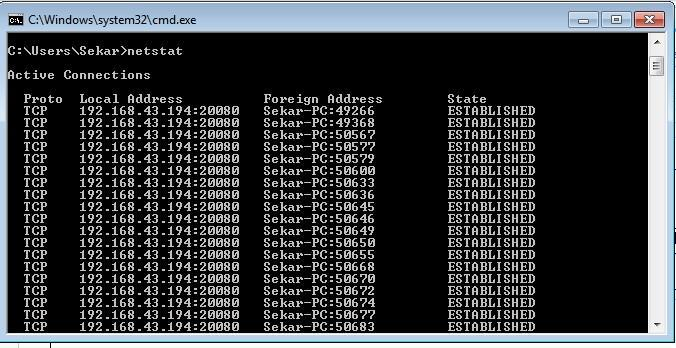
# tcpdump dst port http

**2. Netstat**

Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems. Netstat provides information and statistics about protocols in use and current TCP/IP network connections.

The Windows help screen (analogous to a Linux or UNIX for netstat reads as follows: Displays protocol statistics and current TCP/IP network connections.

#netstat



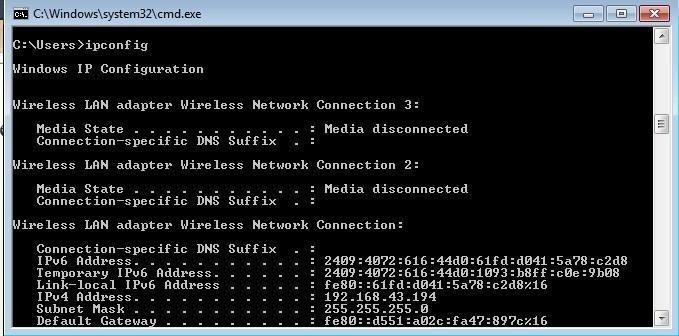
**3.ipconfig**

In Windows, ipconfig is a console application designed to run from the Windows command prompt. This utility allows you to get the IP address information of a Windowscomputer.

**Using ipconfig**

From the command prompt, type ipconfig to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter.

#ipconfig



**4.nslookup**

The nslookup (which stands for name server lookup) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System. The nslookup commandis a powerful tool for diagnosing DNS problems. You know you're experiencing a DNS problem when you can access a resource by specifying its IP address but not its DNS name.

#nslookup

**5.Trace route:**

Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop. Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination.

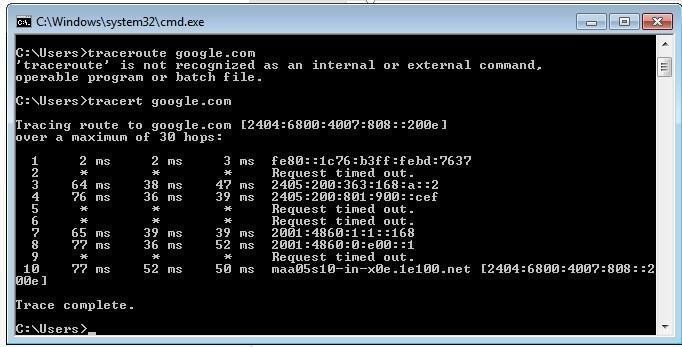
Traceroute also records the time taken for each hop the packet makes during its route to the destination. Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop.

Traceroute sends packets with TTL values that gradually increase from packet to packet, starting with TTL value of one. Routers decrement TTL values of packets by one when routing and discard packets whose TTL value has reached zero, returning the ICMP error message ICMP Time Exceeded. For the first set of packets, the first router receives the packet, decrements the TTL value and drops the packet because it then has TTL value zero. The router sends an ICMP Time Exceeded message back to the source. The next setof packets are given a TTL value of two, so the first router forwards the packets, but the second router drops them and replies with ICMP TimeExceeded.

Proceeding in this way, traceroute uses the returned ICMP Time Exceeded messages to build a list of routers that packets traverse, until the destination is reached and returns an ICMP Echo Reply message. With the tracert command shown above, we're asking tracert to show us the path from the local computer all the way to the network device with the host name

[www.google.com.](http://www.google.com/)

#tracert google.com



**6.Ping:**

The ping command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not. Tracking and isolating hardware and software problems. Determining the status of the network and various foreign hosts. The ping command is usually used as a simple way to verify that a computer can communicate over the network with another computer or network device. The ping command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the destination computer and waiting for a response

# ping172.16.6.2

**RESULT:**

Thus the various networks commands like tcpdump, netstat, ifconfig, nslookup and traceroute ping are executed successfully.

|  |  |
| --- | --- |
| **Ex No : 02** | **Downloading a web pageusing TCP sockets** |
| **DATE :** |

**AIM:**

To write a java program for socket for HTTP for web page upload and download.

**ALGORITHM:**

**Client:**

* 1. Start.
  2. Create socket and establish the connection with the server.
  3. Read the image to be uploaded from the disk.
  4. Send the image read to the server.
  5. Terminate the connection.
  6. Stop.

**Server:**

1. Start.
2. Create socket, bind IP address and port number with the created socket and make server a listening server.
3. Accept the connection request from the client.
4. Receive the image sent by the client.
5. Display the image.
6. Close the connection.

**PROGRAM:**

**Client:**

import javax.swing.\*;

import java.net.\*;

import java.awt.image.\*;

import javax.imageio.\*;

import java.io.\*;

import java.awt.image.BufferedImage;

import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

public class Client {

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

Socket soc;

BufferedImage img = null;

soc = new Socket("localhost", 4000);

System.out.println("Client is running.");

try {

System.out.println("Reading image from disk.");

img = ImageIO.read(new File("digital\_image\_processing.jpg"));

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ImageIO.write(img, "jpg", baos);

baos.flush();

byte[] bytes = baos.toByteArray();

baos.close();

System.out.println("Sending image to server.");

OutputStream out = soc.getOutputStream();

DataOutputStream dos = new DataOutputStream(out);

dos.writeInt(bytes.length);

dos.write(bytes, 0, bytes.length);

System.out.println("Image sent to server.");

dos.close();

out.close();

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage());

soc.close();

}

soc.close();

}

}

**SERVER:**

import java.net.\*;

import java.io.\*;

import java.awt.image.\*;

import javax.imageio.\*;

import javax.swing.\*;

class Server {

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

ServerSocket server = null;

Socket socket;

server = new ServerSocket(4000);

System.out.println("Server Waiting for image");

socket = server.accept();

System.out.println("Client connected.");

InputStream in = socket.getInputStream();

DataInputStream dis = new DataInputStream(in);

int len = dis.readInt();

System.out.println("Image Size: " + len / 1024 + " KB");

byte[] data = new byte[len];

dis.readFully(data);

dis.close();

in.close();

InputStream ian = new ByteArrayInputStream(data);

BufferedImage bImage = ImageIO.read(ian);

System.out.println("Image received");

// Optionally display the image using JLabel

JFrame frame = new JFrame();

frame.setSize(400, 400);

JLabel label = new JLabel(new ImageIcon(bImage));

frame.add(label);

frame.setVisible(true);

ian.close();

socket.close();

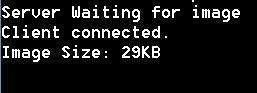
server.close();

}

}

**OUTPUT:**

When we run the client code, following output screen would appear on client side.



**RESULT:**

Thus the socket program for HTTP for web page upload and download was developed and executed successfully.

|  |  |
| --- | --- |
| **Ex No : 03** | **Applications using TCP Socket** |
| **DATE :** |

1. **Echo client and echo server:**

**AIM:**

To write a java program for application using TCP Sockets Link.

**ALGORITHM:**

**Client:**

* 1. Start.
  2. Create the TCP socket.
  3. Establish connection with the server.
  4. Get the message to be echoed from the user.
  5. Send the message to the server.
  6. Receive the message echoed by the server.
  7. Display the message received from the server.
  8. Terminate the connection.
  9. Stop.

**Server:**

1. Start.
2. Create TCP socket, make it a listening socket.
3. Accept the connection request sent by the client for connection establishment.
4. Receive the message sent by the client.
5. Display the received message.
6. Send the received message to the client from which it receives.
7. Close the connection when client initiates termination and server becomes a listening server, waiting for clients.
8. Stop.

**PROGRAM:**

**EServer.java**

import java.net.\*;import java.io.\*;

public class EServer {

public static void main(String args[]) {ServerSocket s = null;

String line; DataInputStream is;PrintStream ps; Socket c = null; try {

s = new ServerSocket(9000);

} catch (IOException e)

{

System.outprintln(e);

}

try {

c = s.accept();

is = new DataInputStream(c.getInputStream());

ps = new PrintStream(c.getOutputStream());while (true) { line = is.readLine();ps.println(line);

}

} catch (IOException e) { System.out.println(e

);

}

}

}

**EClient.java**

import java.net.\*;import java.io.\*;

public class EClient {

public static void main(String arg[]) {Socket c = null; String line; DataInputStream is, is1;PrintStream os; try {

InetAddress ia = InetAddress.getLocalHost();c = new Socket(ia, 9000);

} catch (IOException e) { System.out.println(e);

}

try {

os = new PrintStream(c.getOutputStream());is = new DataInputStream(System.in);

is1 = new DataInputStream(c.getInputStream());while (true) { System.out.println("Client:");line = is.readLine(); os.println(line); System.out.println("Server:" + is1.readLine());

}

} catch (IOException e) { System.out.println("Socket Closed!");

}

}

}

**OUTPUT:**

**Server:**

C:\Program Files\Java\jdk1.5.0\bin>javac EServer.java C:\Program Files\Java\jdk1.5.0\bin>java EServer C:\Program Files\Java\jdk1.5.0\bin>Client C:\Program Files\Java\jdk1.5.0\bin>javac EClient.java

C:\Program Files\Java\jdk1.5.0\bin>java EClient Client: Hai Server

Server:Hai Server Client: Hello Server:Hello Client:end Server:end Client:ds

Socket Closed!

1. **Chat:**

**AIM:**

To write a java program for application using TCP Sockets Link for Chat Application.

**ALGORITHM:**

**Client:**

1. Start.
2. Create the UDP datagram socket.
3. Get the request message to be sent from the user.
4. Send the request message to the server.
5. If the request message is ―END‖ go to step 10.
6. Wait for the reply message from the server.
7. Receive the reply message sent by the server.
8. Display the reply message received from the server.
9. Repeat the steps from 3 to 8.
10. Stop.

**Server:**

1. Start.
2. Create a UDP datagram socket and make it a listening socket.
3. Receive the request message sent by the client.
4. If the received message is "END", go to step 10.
5. Retrieve the client's IP address from the request message received.
6. Display the received message.
7. Get the reply message from the user.
8. Send the reply message to the client.
9. Repeat steps 3 to 8.
10. Stop.

**PROGRAM:**

**UDPserver.java**

import java.io.\*; import java.net.\*;

class UDPserver {

public static DatagramSocket ds;

public static byte buffer[] = new byte[1024]; public static int clientport = 789, serverport = 790;

public static void main(String args[]) throws Exception { ds = new DatagramSocket(clientport); System.out.println("press ctrl+c to quit the program"); BufferedReader dis = new BufferedReader(new InputStreamReader(System.in)); InetAddress ia = InetAddress.geyLocalHost();

while (true) {

DatagramPacket p = new DatagramPacket(buffer, buffer.length);ds.receive(p); String psx = new String(p.getData(), 0, p.getLength()); System.out.println("Client:" + psx); System.out.println("Server:");

String str = dis.readLine();if (str.equals("end")) break;

buffer = str.getBytes();

ds.send(new DatagramPacket(buffer, str.length(), ia, serverport));

}

}

}

**UDPclient.java**

import java.io.\*; import java.net.\*;

class UDPclient {

public static DatagramSocket ds;

public static int clientport = 789, serverport = 790;

public static void main(String args[]) throws Exception {byte buffer[] = new byte[1024]; ds = new DatagramSocket(serverport);

BufferedReader dis = new BufferedReader(new InputStreamReader(System.in)); System.out.println("server waiting");

InetAddress ia = InetAddress.getLocalHost();while (true) { System.out.println("Client:");String str = dis.readLine(); if (str.equals("end"))break;

buffer = str.getBytes();

ds.send(new DatagramPacket(buffer, str.length(), ia, clientport)); DatagramPacket p = new DatagramPacket(buffer, buffer.length);ds.receive(p); String psx = new String(p.getData(), 0, p.getLength()); System.out.println("Server:" + psx);

}

}

}

**OUTPUT:**

**Server**

C:\Program Files\Java\jdk1.5.0\bin>javac UDPserver.java C:\Program Files\Java\jdk1.5.0\bin>java UDPserver

press ctrl+c to quit the program Client:Hai ServerServer:Hello Client Client:How are You

Server:I am Fine

**Client**

C:\Program Files\Java\jdk1.5.0\bin>javac UDPclient.java C:\Program Files\Java\jdk1.5.0\bin>java UDPclient server waiting

Client:Hai Server Server:Hello Clie Client:How are You Server:I am Fine Client:end

1. **File Transfer:**

**AIM:**

To write a java program for file transfer using TCP Sockets.

**ALGORITHM:**

**Server:**

1. Import java packages and create class file server.
2. Create a new server socket and bind it to the port.
3. Accept the client connection
4. Get the file name and stored into the BufferedReader.
5. Create a new object class file and realine.
6. 6.If file is exists then FileReader read the content until EOF is reached.
7. Stop the program.

**Client:**

1. Import java packages and create class file server.
2. Create a new server socket and bind it to the port.
3. Now connection is established.
4. The object of a BufferReader class is used for storing data content which has been retrieved from socket object.
5. The connection is closed.
6. Stop the program.

**PROGRAM:**

**File Server:**

import java.io.BufferedInputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.OutputStream;

import java.net.InetAddress;

import java.net.ServerSocket;

import java.net.Socket;

public class FileServer {

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

// Initialize Sockets

ServerSocket ssock = new ServerSocket(5000);

System.out.println("Server is listening on port 5000...");

Socket socket = ssock.accept();

System.out.println("Client connected.");

// The InetAddress specification (localhost)

InetAddress IA = InetAddress.getByName("localhost");

// Specify the file

File file = new File("e:\\Bookmarks.html");

FileInputStream fis = new FileInputStream(file);

BufferedInputStream bis = new BufferedInputStream(fis);

// Get socket's output stream

OutputStream os = socket.getOutputStream();

// Read File Contents into contents array

byte[] contents;

long fileLength = file.length();

long current = 0;

long start = System.nanoTime();

// File transfer loop

while (current != fileLength) {

int size = 10000;

if (fileLength - current >= size) {

current += size;

} else {

size = (int) (fileLength - current);

current = fileLength;

}

contents = new byte[size];

bis.read(contents, 0, size);

os.write(contents);

System.out.println("Sending file ... " + (current \* 100) / fileLength + "% complete!");

}

os.flush();

// File transfer done. Close the socket connection!

socket.close();

ssock.close();

bis.close();

fis.close();

System.out.println("File sent successfully!");

}

}

**File Client:**

import java.io.BufferedOutputStream;

import java.io.FileOutputStream;

import java.io.InputStream;

import java.net.InetAddress;

import java.net.Socket;

public class FileClient {

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

// Initialize socket

Socket socket = new Socket(InetAddress.getByName("localhost"), 5000);

byte[] contents = new byte[10000];

// Initialize the FileOutputStream to the output file's full path

FileOutputStream fos = new FileOutputStream("e:\\Bookmarks1.html");

BufferedOutputStream bos = new BufferedOutputStream(fos);

InputStream is = socket.getInputStream();

// No of bytes read in one read() call

int bytesRead = 0;

while ((bytesRead = is.read(contents)) != -1) {

bos.write(contents, 0, bytesRead);

}

// Flush and close streams

bos.flush();

bos.close();

fos.close();

socket.close();

System.out.println("File saved successfully!");

}

}

**OUTPUT:**

**Server:**

E:\nwlab>java FileServer

Sending file ... 9% complete!

Sending file ... 19% complete!

Sending file ... 28% complete!

Sending file ... 38% complete!

Sending file ... 47% complete!

Sending file ... 57% complete!

Sending file ... 66% complete!

Sending file ... 76% complete!

Sending file ... 86% complete!

Sending file ... 95% complete!

Sending file ... 100% complete!

**Client:**

E:\nwlab>client E:\nwlab>java FileClient File saved successfully!

E:\nwlab>

**RESULT:**

Thus the java application program using TCP Sockets was developed and executed successfully.