|  |  |
| --- | --- |
| **Ex No:01** | **Learn to use commands like tcpdump, netstat, ifconfig, nslookup and**  **traceroute. Capture ping and traceroute PDUs using a network protocol**  **analyzer and examine.** |
| **Date:** |

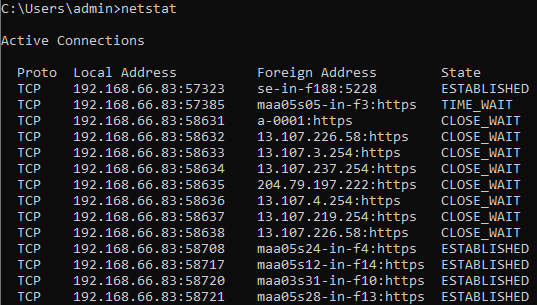
**AIM:**

To Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute ping

**Commands:-**

**netstat:-**

This command displays active TCP connections by default.

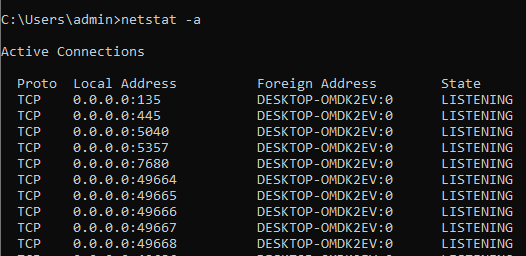
Example:-  


**Netstat commands with their flags:-**

**netstat -a:**

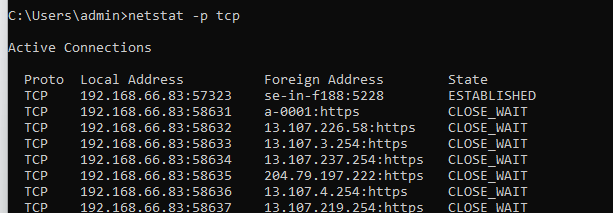
Shows all active connections and listening ports, both TCP and UDP.

Example



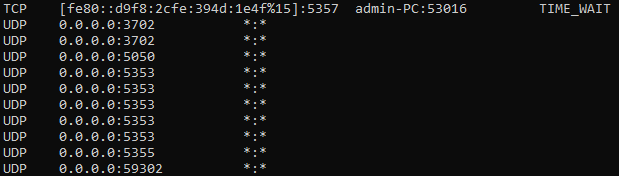
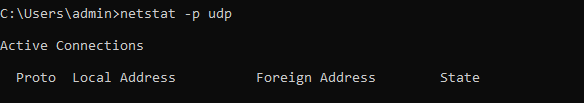
netstat -p tcp:

 Displays only TCP connections.



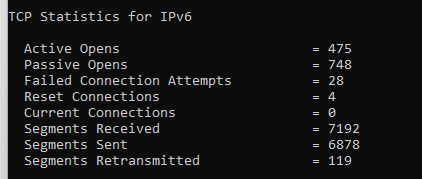
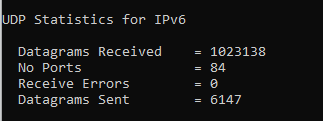
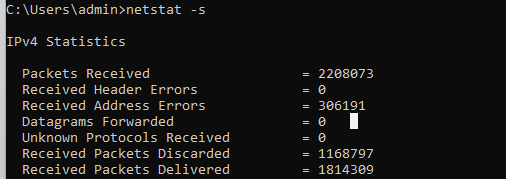
netstat -p udp:

  Displays only UDP connections.



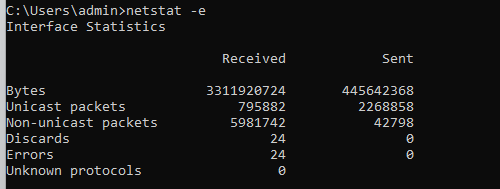
netstat -s:

Provides statistics for each protocol. This includes various network-related statistics such as TCP segments sent/received, UDP datagrams sent/received, etc.



netstat -e:

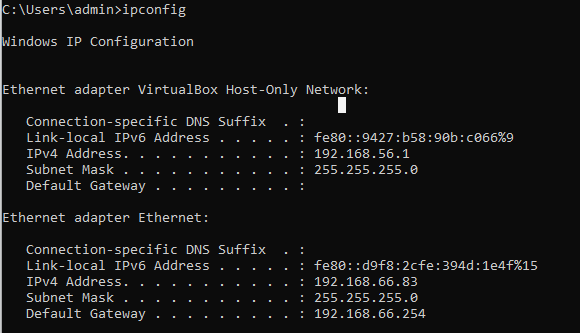
Displays Ethernet statistics, including bytes sent/received, packets sent/received, and other network interface-specific information.



**ipconfig:**

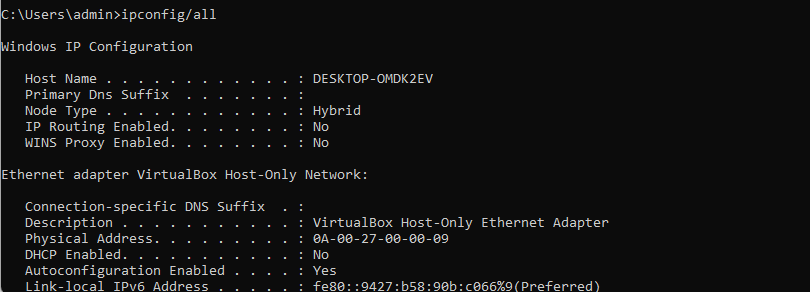
Displays the IP configuration of all network interfaces, including IP addresses, subnet masks, default gateways, and DNS servers.

Example:-



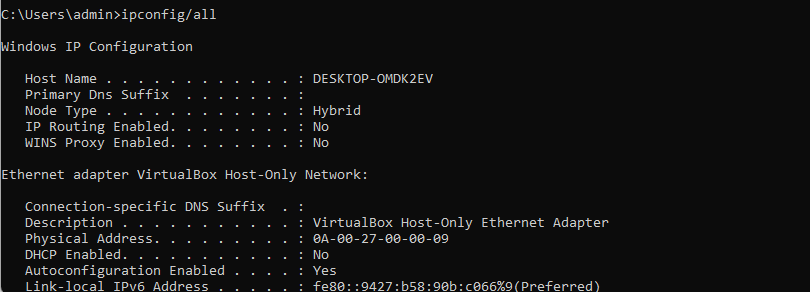
ipconfig /all:

Provides detailed information about the IP configuration of all network interfaces, including physical addresses (MAC addresses), DHCP lease information, and DNS settings.



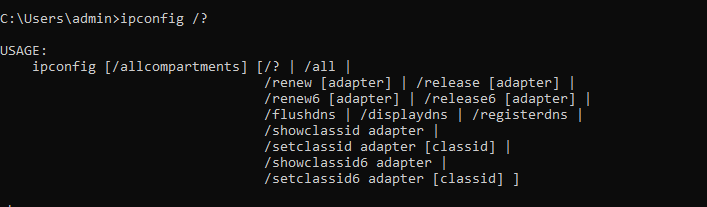
ipconfig /all:

Provides detailed information about the IP configuration of all network interfaces, including physical addresses (MAC addresses), DHCP lease information, and DNS settings.



ipconfig /?:

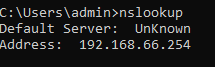
Displays the help information for the ipconfig command, providing details about the available flags and their usage.



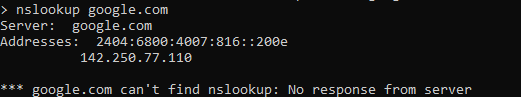
**nslookup:**

 It used for querying DNS (Domain Name System) to obtain domain-related information.

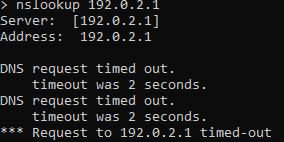
Example:



nslookup <domain>: Queries the DNS for the IP address of a specific domain.



nslookup <IP address>: Performs a reverse DNS lookup to find the domain name associated with a given IP address.

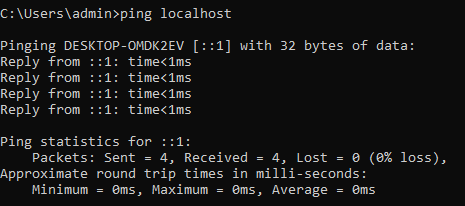


**ping:**

 command in Windows is used to test network connectivity and measure the response time between a source and destination. Here are some common ping commands with their flags and examples:

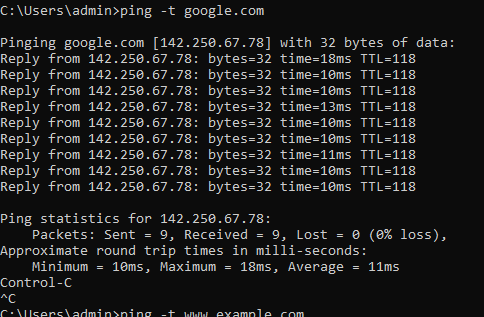
ping <hostname or IP address>: Sends ICMP echo requests to the specified destination and displays the round-trip time and status of each reply.

Example:

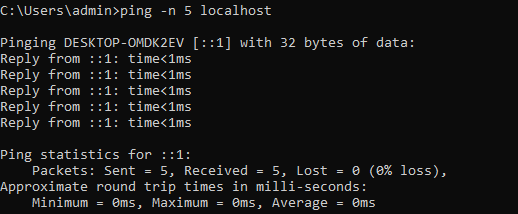


ping -t <hostname or IP address>:

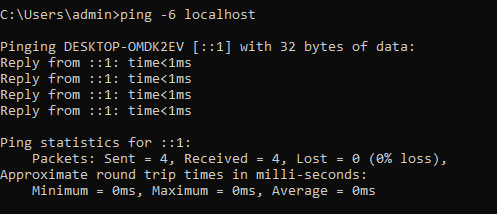
Continuously pings the specified destination until manually stopped by pressing Ctrl+C.



ping -n <count> <hostname or IP address>: Sends the specified number of ICMP echo requests to the destination.



ping -6 <hostname or IP address>: Forces the use of IPv6 for the ping request.



**RESULT**

Thus the various networks commands like tcpdump, netstat, ifconfig, nslookupand traceroute ping are executed successfully

|  |  |
| --- | --- |
| **EX.No: 2** | **Write a HTTP web client program to download a web page using TCP sockets** |
| **Date:** |

**AIM :**

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

**ALGORITHM:**

Client:

Step1:Start.

Step2:Create socket and establish the connection with the server. Step3:Read the image to be uploaded from the disk

Step4:Send the image read to the server Step5: Terminate the connection

Step6: Stop.

Server:

Step1:Start

Step2:Create socket, bind IP address and port number with the created socket and make server a listening server.

Step3:Accept the connection request from the client Step4:Receive the image sent by the client.

Step5:Display the image. Step6:Close the connection. Step7:Stop

**PROGRAM:**

Client:

import java.io.\*;

import java.net.HttpURLConnection;

import java.net.URL;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.StandardOpenOption;

public class SS {

private static final String SERVER\_URL = "http://localhost:3001/image";

public static void main(String[] args) {

try {

URL url = new URL(SERVER\_URL);

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("GET");

int responseCode = connection.getResponseCode();

if (responseCode == HttpURLConnection.HTTP\_OK) {

String contentType = connection.getContentType();

if (contentType != null && contentType.startsWith("image/")) {

int contentLength = connection.getContentLength();

// Read the image data

InputStream inputStream = connection.getInputStream();

ByteArrayOutputStream imageData = new ByteArrayOutputStream();

byte[] buffer = new byte[1024];

int bytesRead;

while ((bytesRead = inputStream.read(buffer)) != -1) {

imageData.write(buffer, 0, bytesRead);

}

Path filePath = Path.of("./image.jpg");

Files.write(filePath, imageData.toByteArray(), StandardOpenOption.CREATE,

StandardOpenOption.TRUNCATE\_EXISTING);

System.out.println("Image downloaded and saved to: " + filePath.toAbsolutePath());

} else {

System.out.println("The requested URL is not an image.");

}

} else {

System.out.println("Failed to download image. Response code: " + responseCode);

}

connection.disconnect();

} catch (IOException e) {

e.printStackTrace();

}

SERVER:

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.StandardCopyOption;

public class Ser {

private static final int PORT = 3001;

private static final String IMAGE\_PATH = "Java.jpg";

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(PORT)) {

System.out.println("Server listening on port " + PORT);

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected: " + clientSocket.getInetAddress().getHostAddress());

BufferedReader request = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

OutputStream response = clientSocket.getOutputStream();

String line = request.readLine();

if (line != null) {

String[] requestParts = line.split(" ");

if (requestParts.length >= 3 && requestParts[0].equals("GET")) {

String path = requestParts[1];

if (path.equals("/image")) {

serveImage(response);

} else {

serve404NotFound(response);

}

}

}

clientSocket.close();

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

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void serveImage(OutputStream response) throws IOException {

File imageFile = new File(IMAGE\_PATH);

if (imageFile.exists()) {

byte[] imageBytes = Files.readAllBytes(Path.of(IMAGE\_PATH));

PrintWriter pw = new PrintWriter(response);

pw.println("HTTP/1.1 200 OK");

pw.println("Content-Type: image/jpeg");

pw.println("Content-Length: " + imageBytes.length);

pw.println();

pw.flush();

response.write(imageBytes);

response.flush();

} else {

serve404NotFound(response);

}

}

private static void serve404NotFound(OutputStream response) throws IOException {

PrintWriter pw = new PrintWriter(response);

pw.println("HTTP/1.1 404 Not Found");

pw.println("Content-Type: text/html");

pw.println();

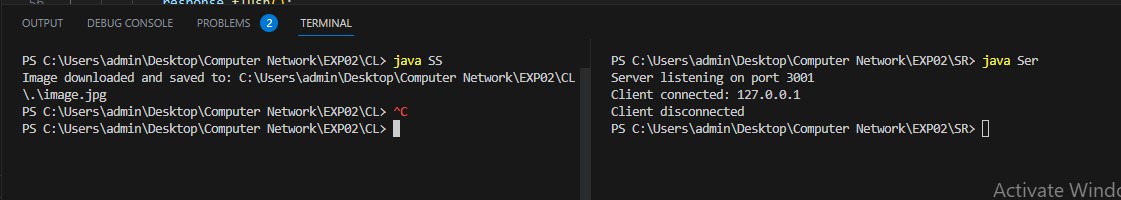
pw.println("<h1>404 Not Found</h1>");

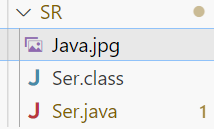
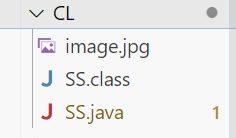
pw.flush();

}

}

**Output:-**





**RESULT:**

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

|  |  |
| --- | --- |
| **Ex No: 3a** | **Applications using TCP sockets: Echo client and echo server** |
| **Date:** |

**AIM :**

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

**ALGORITHM:**

Echo Client and Server Client:

Step1: Start

Step2: Create the TCP socket Step3:Establish connection with the server

Step4: Get the message to be echoed from the user Step5: Send the message to the server

Step6: Receive the message echoed by the server Step7: Display the message received from the server Step8: Terminate the connection

Step9: Stop

Server:

Step1: Start

Step2: Create TCP socket, make it a listening socket

Step3: Accept the connection request sent by the client for connection establishment Step4: Receive the message sent by the client

Step5: Display the received message

Step6: Send the received message to the client from which it receives

Step7: Close the connection when client initiates termination and server becomes a listening server, waiting for clients.

Step8: Stop.

**PROGRAM:**

EClient.java

package CL;

import java.net.\*;

import java.io.\*;

public class client {

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.print("Client:");

line = is.readLine();

os.println(line);

System.out.println("Server: " + is1.readLine());

}

} catch (IOException e) {

System.out.println("Socket Closed!");

}

}

}

EServer.java

package SR;

import java.net.\*;

import java.io.\*;

public class server {

public static void main(String args[]) {

ServerSocket s = null;

String line;

DataInputStream is = null;

PrintStream ps = null;

Socket c = null;

try {

s = new ServerSocket(9000);

c = s.accept();

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

ps = new PrintStream(c.getOutputStream());

while (true) {

line = is.readLine();

if (line == null) {

break;

}

ps.println(line);

}

} catch (IOException e) {

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

} finally {

try {

if (is != null) is.close();

if (ps != null) ps.close();

if (c != null) c.close();

if (s != null) s.close();

} catch (IOException ex) {

System.out.println("Error while closing resources: " + ex.getMessage());

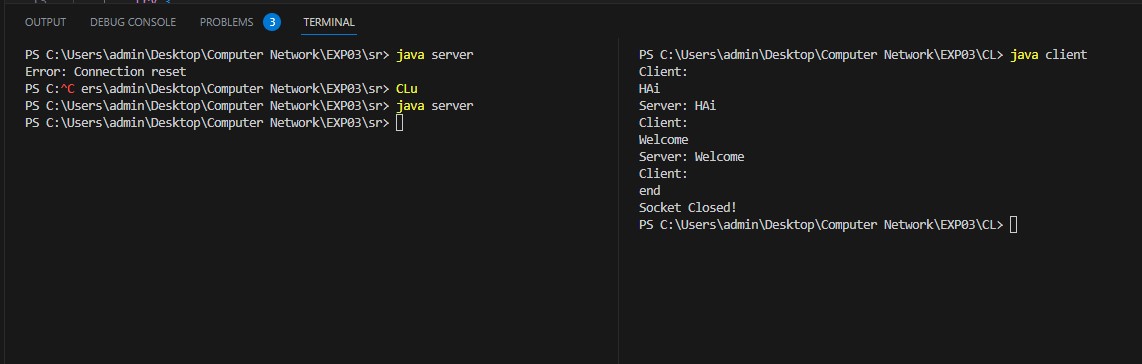
}

}

}

}

**Output:-**



**RESULT:**

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

|  |  |
| --- | --- |
| **Ex No: 3b** | **Applications using TCP sockets: Chat** |
| **Date:** |

**AIM:**

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

**ALGORITHM:**

Client:

Step1: Start

Step2: Create the UDP datagram socket

Step3: Get the request message to be sent from the user Send the request message to the server

Step4: If the request message is ―END‖ go to step 10 Step5: Wait for the reply message from the server Step6: Receive the reply message sent by the server

Step8: Display the reply message received from the server Step9: Repeat the steps from 3 to 8

Step10: Stop

Server:

Step1: Start

Step2: Create UDP datagram socket, make it a listening socket Step3: Receive the request message sent by the client

Step4: If the received message is ―END‖ go to step 10

Step5: Retrieve the client‘s IP address from the request message received Step6: Display the received message

Step7: Get the reply message from the user

Step8: Send the reply message to the client

**PROGRAM:**

UDPserver.java

import java.io.\*;

import java.net.\*;

public class server {

private static int indexOf(String[] array, String str) {

str = str.trim();

for (int i = 0; i < array.length; i++) {

if (array[i].equals(str))

return i;

}

return -1;

}

public static void main(String arg[]) {

String[] hosts = { "yahoo.com", "gmail.com", "cricinfo.com", "facebook.com" };

String[] ip = { "68.180.206.184", "209.85.148.19", "80.168.92.140", "69.63.189.16" };

System.out.println("Press Ctrl + C to Quit");

while (true) {

try {

DatagramSocket serversocket = new DatagramSocket(1362);

byte[] receivedata = new byte[1021];

DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);

serversocket.receive(recvpack);

String sen = new String(recvpack.getData()).trim();

InetAddress ipaddress = recvpack.getAddress();

int port = recvpack.getPort();

String capsent;

System.out.println("Request for host " + sen);

int index = indexOf(hosts, sen);

if (index != -1) {

capsent = ip[index];

} else {

capsent = "Host Not Found";

}

byte[] senddata = capsent.getBytes();

DatagramPacket pack = new DatagramPacket(senddata, senddata.length, ipaddress, port);

serversocket.send(pack);

serversocket.close();

} catch (IOException e) {

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

}

}

}

}

UDPClient.java

import java.io.\*;

import java.net.\*;

public class cl {

public static void main(String args[]) {

try {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

DatagramSocket clientsocket = new DatagramSocket();

InetAddress ipaddress;

if (args.length == 0)

ipaddress = InetAddress.getLocalHost();

else

ipaddress = InetAddress.getByName(args[0]);

byte[] senddata = new byte[1024];

byte[] receivedata = new byte[1024];

int portaddr = 1362;

System.out.print("Enter the hostname: ");

String sentence = br.readLine();

senddata = sentence.getBytes();

DatagramPacket pack = new DatagramPacket(senddata, senddata.length, ipaddress, portaddr);

clientsocket.send(pack);

DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);

clientsocket.receive(recvpack);

String modified = new String(recvpack.getData()).trim();

System.out.println("IP Address: " + modified);

clientsocket.close();

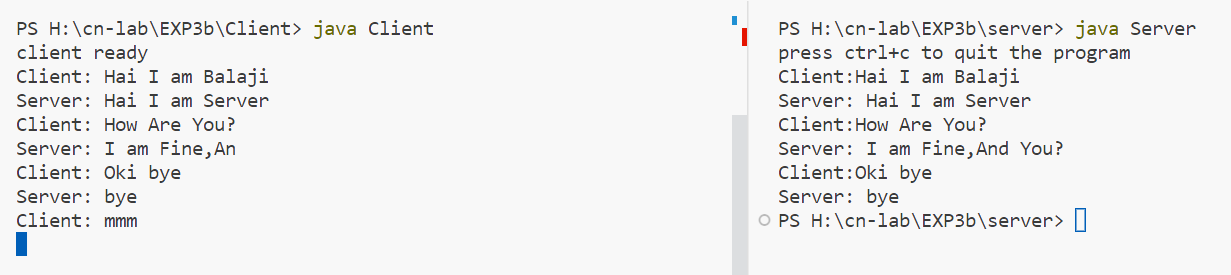
} catch (IOException e) {

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

}

}

}

**Output:-**

**RESULT:**

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

|  |  |
| --- | --- |
| **Ex No: 3c** | **Applications using TCP sockets: File Transfer** |
| **Date:** |

**AIM:**

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

**ALGORITHM:**

Server:

Step 1:Import java packages and create class file server.

Step 2:Create a new server socket and bind it to the port.

Step 3:Accept the client connection

Step 4:Get the file name and stored into the BufferedReader.

Step 5:Create a new object class file and realine.

Step 6:If file is exists then FileReader read the content until EOF is reached.

Step 7:Stop the program.

Client:

Step 1:Import java packages and create class file server.

Step 2:Create a new server socket and bind it to the port.

Step 3:Now connection is established.

Step 4:The object of a BufferReader class is used for storing data content which has beenretrieved from socket object.

Step 5:The connection is closed.

Step 6:Stop the program.

**PROGRAM:**

File Server :

import java.io.\*;

import java.net.\*;

public class Server {

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

// Initialize ServerSocket

ServerSocket ssock = new ServerSocket(5000);

Socket socket = ssock.accept();

// Specify the file

File file = new File("index.html");

FileInputStream fis = new FileInputStream(file);

BufferedInputStream bis = new BufferedInputStream(fis);

OutputStream os = socket.getOutputStream();

byte[] contents;

long fileLength = file.length();

long current = 0;

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.print("Sending file... " + (current \* 100) / fileLength + "% complete!\n");

}

os.flush();

socket.close();

ssock.close();

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

}

}

File Client:

import java.io.\*;

import java.net.\*;

public class Client {

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

// Initialize socket

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

byte[] contents = new byte[10000];

int bytesRead;

InputStream is = socket.getInputStream();

FileOutputStream fos = new FileOutputStream("receivedFile.html");

BufferedOutputStream bos = new BufferedOutputStream(fos);

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

bos.write(contents, 0, bytesRead);

}

bos.flush();

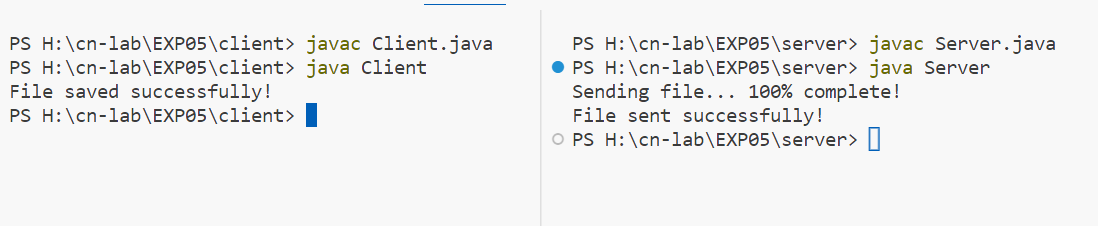
socket.close();

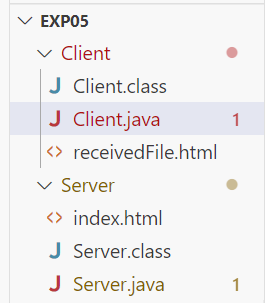
System.out.println("File saved successfully!");

}

}

**Output:-**



**RESULT:**

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