MEGHNAD SAHA INSTITUTE OF TECHNOLOGY

Nazirabad, P.O. - Uchhepota, Near URBANA Complex, Anandapur, Kolkata 700 150

Computer Science & Engineering

LABORATORY NOTE BOOK

MAKAUT EVEN / ODD SEMESTER 20_____

PAPER NAME: ____

PAPER CODE: ____

STUDENT NAME: ____

ROLL NO:	REGN. NO.:
YEAR:	SEMESTER:
SECTION:	SESSION:

"LIST OF ASSIGNMENT/EXPERIMENT SUBMISSION DETAILS"

SL. NO.	ASSIGNMENT / EXPERIMENT NAME	DATE OF ASSIGNMENT / EXPERIMENT DONE	DATE OF SUBMISION	CHECKED BY	REMARKS (ANY DEVIATION REGARDING SUBMISSION DATES, CONTENT, FORMAT, ETC)

OBSERVATIONS / COMMENTS ON THE OVERALL PERFORMANCE:

Signature in full with date

Faculty / Technical Assistant

Assignment – 1

1. What is computer network and computer internetworking? Explain.

Computer Network:

A computer network is a collection of interconnected devices that communicate with each other to share resources and information. These devices, often referred to as nodes, can include computers, servers, routers, switches, and other networking hardware. The primary purposes of a computer network are:

- **Communication:** Allowing devices to exchange data and messages.
- **Resource Sharing:** Sharing hardware resources like printers and storage devices.
- **Data Sharing:** Enabling users to share files and information.
- **Application and Service Provision:** Facilitating the use of applications like email, internet browsing, and online services.

Computer Internetworking:

Computer internetworking refers to the process of connecting multiple computer networks together to form a larger, cohesive network. This is often achieved using devices such as routers and switches, which manage the flow of data between different networks. Internetworking allows for the seamless transmission of data across different network segments and is fundamental to the operation of the internet.

Key aspects of computer internetworking include:

- **Routing:** The process of determining the path data packets take from the source to the destination across interconnected networks.
- **IP Addressing:** Assigning unique identifiers (IP addresses) to devices on the network to facilitate communication.
- **Protocols:** Standardized rules (such as TCP/IP) that define how data is formatted, transmitted, and received across networks.
- **Network Segmentation:** Dividing a larger network into smaller, manageable segments to improve performance and security.

The difference Between Networking and Internetworking is:

Networking involves connecting and managing devices within a single network, focusing on local communication and resource sharing. Internetworking extends this concept by connecting multiple distinct networks, enabling broader communication and data transfer across larger geographical areas.

2. Explain NIC. How to install and configure NIC card?

Ans: - A Network Interface Card (NIC) is a hardware component that connects a computer to a network. It provides the necessary physical and data link layer functions to transmit and receive data over a network. NICs can be integrated into the motherboard or added as separate expansion cards. They come in various forms, including wired NICs (using Ethernet) and wireless NICs (using Wi-Fi).

Installing a NIC Card:

- **Power Off and Unplug the Computer:** Ensure the computer is turned off and unplugged from the power source.
- **Open the Computer Case:** Use a screwdriver to remove the screws holding the case cover, and then remove the cover to access the motherboard.
- **Locate an Available Expansion Slot:** Find an empty PCI or PCIe slot on the motherboard. These slots are typically white or beige and are positioned parallel to each other.
- **Insert the NIC Card:** Carefully align the NIC card's edge connector with the expansion slot and gently press it down until it is fully seated. Ensure the metal bracket of the NIC is aligned with the case's opening.
- **Secure the NIC Card:** Use a screw to secure the metal bracket of the NIC to the case, ensuring it is firmly in place.
- **Close the Computer Case:** Replace the case cover and secure it with screws.

Configuring the NIC Card:

To configure a NIC (Network Interface Card) on an Ubuntu distribution of Linux, you can use either the graphical interface or command-line tools.

- **Using the Graphical Interface:**
  - **Open Network Settings:**
    Click on the network icon in the top-right corner of the screen.
    Select "Settings" or "Network Settings."
  - **Select the Network Interface:**
    In the Network settings window, find the wired or wireless interface you want to configure.
    Click on the gear icon next to the interface.
  - **Configure the Network Interface:**
    (a) Automatic (DHCP):
    Ensure "Automatic (DHCP)" is selected to get an IP address automatically.
    (b) Manual (Static IP):
    Select "Manual."
    Enter the required IP address, subnet mask, and gateway.
    Add DNS servers in the DNS field if needed.
  - **Save the Configuration:**
    Click "Apply" to save the settings.
    Restart the network interface by turning it off and on again or rebooting the computer.
- **Using the Command Line:**
  - **Identify the Network Interface:**
    i. Open a terminal.
    ii. Run the command **'ip link'** or **'ifconfig'** to list all network interfaces and identify the one you want to configure (e.g., eth0, enp3s0).
  - **Edit the Netplan Configuration File:**
    i. Ubuntu uses Netplan for network configuration starting from version 17.10.
    ii. Open the Netplan configuration file in a text editor with root privileges. For example: **sudo nano /etc/netplan/01-netcfg.yaml**
    iii. The file might be named differently, so use **'ls /etc/netplan'** to find the exact name.

- **Modify the Configuration:**
  a) **Automatic (DHCP):**

```
network:
  version: 2
  ethernets:
    eth0:
      dhcp4: true
```

  b) **Manual (Static IP):**

```
network:
  version: 2
  ethernets:
    eth0:
      addresses:
        - 192.168.1.100/24
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

  Replace eth0 with your actual interface name, and adjust the IP addresses, subnet mask, gateway, and DNS servers as needed.

- **Apply the Configuration:**
  i. Save the file and exit the text editor.
  ii. Run the command to apply the configuration:
  sudo netplan apply
- **Verify the Configuration:**
  Check the IP address and network settings with:
  ip addr show eth0
  Replace 'eth0' with your interface name.

By following these steps, you can successfully configure a NIC card on an Ubuntu distribution of Linux. The graphical method is more user-friendly, while the command-line method provides more control and is useful for remote or server configurations.

3. Explain: (a) RJ45 (b) Networking cables (CAT-5, UTP).

a) RJ45:

RJ45 stands for Registered Jack 45, and it is a standardized physical network interface commonly used for connecting Ethernet cables in wired networks. The RJ45 connector looks similar to a telephone jack but is slightly wider and has eight pins or contacts, allowing it to support Ethernet connections.

Features of RJ45:

- **Eight Pins:** RJ45 connectors have eight pins, enabling eight wires to be connected, which are typically arranged in a specific sequence.
- **Ethernet Connectivity:** It is used primarily for Ethernet networking and is the standard connector for most types of network cabling.

- **Plug and Play:** RJ45 connectors are designed to be easily inserted and removed from an Ethernet port, facilitating quick and easy connections.
- **Wiring Standards:** The most common wiring standards for RJ45 connectors are T568A and T568B, which define the pinout arrangement for the wires in the cable.

b) Networking Cables:

Networking cables are essential components for wired networking, used to connect different devices such as computers, routers, and switches to establish a network. Two common types of networking cables are CAT-5 and UTP.

- **CAT-5 (Category 5) Cable:**
  Category 5 (CAT-5) cable is a type of twisted pair cable designed for Ethernet networks.
  Speed and Bandwidth: CAT-5 cables support speeds up to 100 Mbps and a bandwidth of 100 MHz, making them suitable for 100BASE-TX (Fast Ethernet) networks.
  Structure: CAT-5 cables consist of four twisted pairs of copper wires, reducing interference and crosstalk.
  Maximum Length: The maximum length for a CAT-5 cable without signal degradation is 100 meters (328 feet).

- **UTP (Unshielded Twisted Pair) Cable:**
  Unshielded Twisted Pair (UTP) cable is a type of twisted pair cable that lacks additional shielding to protect against electromagnetic interference (EMI).
  Types: UTP cables come in various categories, including CAT-5, CAT-5e, CAT-6, and CAT-7, with each higher category providing improved performance and reduced crosstalk.
  Use Case: UTP cables are widely used in Ethernet networks due to their cost-effectiveness and ease of installation.
  Twisted Pairs: The wires in UTP cables are twisted into pairs to cancel out EMI and reduce crosstalk between the pairs.

4. What do you mean by network class and subnet mask-Explain.

Network Class:

In the context of IP addressing, network classes refer to a method used in the original IP addressing architecture to allocate IP address ranges for different sizes of networks. There are five classes (A, B, C, D, E), but classes A, B, and C are the most commonly used for unicast addressing. Each class has a specific range of addresses and a default subnet mask.

i. Class A:

- Range: 1.0.0.0 to 126.255.255.255
- Default Subnet Mask: 255.0.0.0 (/8)
- Number of Networks: 128 (only 1-126 are usable)
- Number of Hosts per Network: ~16.7 million

Class A is used for very large networks.

ii. **Class B:**
   - Range: 128.0.0.0 to 191.255.255.255
   - Default Subnet Mask: 255.255.0.0 (/16)
   - Number of Networks: 16,384
   - Number of Hosts per Network: ~65,536

Class B is used for medium-sized networks.

iii. **Class C:**
   - Range: 192.0.0.0 to 223.255.255.255
   - Default Subnet Mask: 255.255.255.0 (/24)
   - Number of Networks: 2,097,152
   - Number of Hosts per Network: 256 (254 usable addresses)

Class C is used for small networks.

iv. **Class D (Multicast):**
   - Range: 224.0.0.0 to 239.255.255.255

Class D addresses are used for multicast groups rather than individual hosts.

v. **Class E (Experimental):**
   - Range: 240.0.0.0 to 255.255.255.255

Class E addresses are reserved for experimental purposes and are not used for regular network traffic.

Subnet Mask:

A subnet mask is a 32-bit number that divides an IP address into a network and host portion. It helps determine which part of the IP address represents the network and which part represents the host. The subnet mask works by performing a bitwise AND operation between the IP address and the subnet mask to extract the network address.

How Subnet Mask Works:

i. **Default Subnet Masks:** These are used with the different classes of IP addresses to define the default network portion.

   Class A: 255.0.0.0

   Class B: 255.255.0.0

   Class C: 255.255.255.0

ii. **CIDR Notation:** Instead of writing the subnet mask in its dotted-decimal form, it can be represented using CIDR (Classless Inter-Domain Routing) notation, which specifies the number of bits set to 1 in the subnet mask. For example:

   255.0.0.0 = /8

   255.255.0.0 = /16

   255.255.255.0 = /24

5. Given IP Address: 192.168.10.0 /25

a) Find Subnet mask

b) Find the number of networks

c) Find the number of IP address on each network.

d) Find number of hosts in each network

Ans: -

Given the IP address 192.168.10.0 with a subnet mask of /25, we can determine the following:

a. Subnet Mask

The subnet mask for a /25 prefix is:

- /25 in binary: '11111111.11111111.11111111.10000000'
- In decimal notation: '255.255.255.128'

b. Number of Networks

Since the original IP address (192.168.10.0) is a Class C address with a default subnet mask of /24 (255.255.255.0), and we are using a /25 subnet mask, we are borrowing 1 bit from the host portion to create subnets.

- Class C default subnet mask: /24
- Custom subnet mask: /25

The number of additional subnets created is $\mathbf{2^1 = 2}$

Thus, there are 2 subnets.

c. Number of IP Addresses on Each Network

Each /25 subnet has 7 bits allocated for host addresses (32 total bits - 25 network bits = 7 host bits).

- Number of IP addresses per subnet: $\mathbf{2^7 = 128}$

So, there are 128 IP addresses in each subnet.

d. Number of Hosts in Each Network

The total number of IP addresses includes the network address and the broadcast address, which are not assignable to hosts.

- Number of usable host addresses: $\mathbf{2^7 - 2 = 128 - 2 = 126}$

So, there are 126 usable hosts in each subnet.

6. **Given IP Address: 192.168.10.0 /26.**
   a) **Find Subnet mask**
   b) **Find the number of networks**
   c) **Find the number of IP address on each network.**
   d) **Find number of hosts in each network**

Ans: -

Given the IP address 192.168.10.0 with a subnet mask of /26, let's determine the following:

1. **Subnet Mask**
   - The subnet mask for a /26 prefix is:
   - /26 in binary: ‘11111111.11111111.11111111.11000000’
   - In decimal notation: ‘255.255.255.192’
2. **Number of Networks**

   Since the original IP address (192.168.10.0) is a Class C address with a default subnet mask of /24 (255.255.255.0), and we are using a /26 subnet mask, we are borrowing 2 bits from the host portion to create subnets.

   - Class C default subnet mask: /24
   - Custom subnet mask: /26

   The number of additional subnets created is $\mathbf{2^2 = 4}$

   Thus, there are 4 subnets.
3. **Number of IP Addresses on Each Network**

   Each /26 subnet has 6 bits allocated for host addresses (32 total bits - 26 network bits = 6 host bits).

   - Number of IP addresses per subnet: $\mathbf{2^6 = 64}$

   So, there are 64 IP addresses in each subnet.
4. **Number of Hosts in Each Network**

   The total number of IP addresses includes the network address and the broadcast address, which are not assignable to hosts.

   - Number of usable host addresses: $\mathbf{2^6 - 2 = 64 - 2 = 62}$

   So, there are 62 usable hosts in each subnet.

Assignment – 2

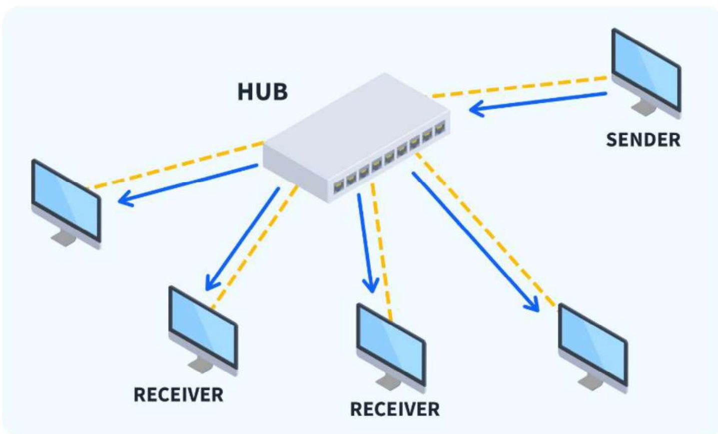
1. **Explain functions of HUB and Switches with suitable diagram.**

HUB:

A hub is a basic networking device that connects multiple computers or other network devices in a LAN (Local Area Network). It operates at the physical layer (Layer 1) of the OSI model. Hubs are simple devices with minimal intelligence and functionality

Functions of a Hub:

- **Broadcasting:** When a hub receives a data packet from one of its ports, it broadcasts the packet to all other ports. This means that all devices connected to the hub will receive the packet, regardless of the intended recipient.
- **Simple Connectivity:** Hubs provide basic connectivity for devices in a network, allowing data to be transmitted between connected devices.
- **No Filtering or Addressing:** Hubs do not filter data or use any addressing. They do not distinguish between the different data packets or their destinations.
- **Diagram of a Hub:**



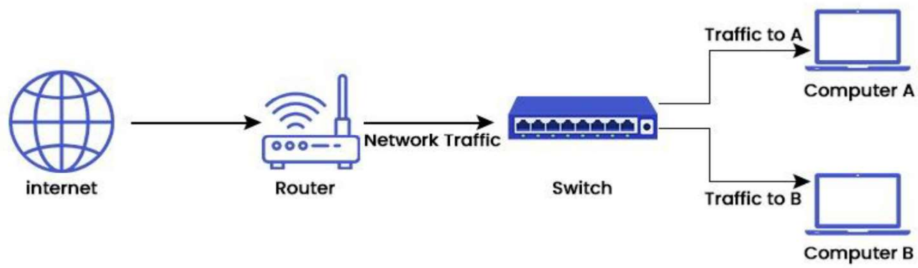
Switches:

A switch is a more advanced networking device that connects multiple devices in a LAN. It operates at the data link layer (Layer 2) of the OSI model, although some switches also have Layer 3 capabilities (network layer).

Functions of a Switch:

- **Frame Filtering and Forwarding:** A switch can inspect incoming data frames and forward them only to the specific port that connects to the destination device. This reduces unnecessary traffic and improves network efficiency.
- **MAC Address Table:** Switches maintain a table of MAC (Media Access Control) addresses. When a frame is received, the switch records the source MAC address and the port it arrived on. It uses this table to direct future frames to the correct destination port.
- **Collision Domain Segmentation:** Each port on a switch represents a separate collision domain, which reduces collisions and improves network performance.

- **VLANs:** Some switches support Virtual LANs (VLANs), which allow network segmentation logically rather than physically, improving security and traffic management.
- **Diagram of a Switch:**



2. HOW to configure/setup physical LAN – Explain in detail.

Ans: -

LAN (Local Area Network) is a data communication network that locally connects network devices such as workstations, servers, routers, etc. to share the resources within a small area such as a building or campus. Physical or wireless connections are set up between workstations to share the resources. Ethernet and Wi-fi are the most important technologies of LAN. Personal networks at home, school, office, etc. are examples of LAN. These are generally privately-owned networks.

Requirements to set up LAN Network:

- Workstation/Personal devices: laptop, computer, mobile phones, etc.
- Network devices: router, switch, modem (if not already present in the router)
- Sharing resources: printers, disk drives, etc.
- Cables: Ethernet cables, wires for connecting other devices (in case of wired LAN)
- Internet connection: Wi-Fi (in case of wireless LAN)

Instructions to set up LAN Network:

Following steps should be followed to set up a LAN network:

- **Identify services:** Identify the network services such as printers, disk drives, data, etc. that will be shared among workstations.
- **Identify devices:** Identify devices such as computers, mobile phones, laptops, etc. with a unique address that will be connected to the network.
- **Plan connections:** Design the network by laying out cable wires between network devices or by making wireless connections. Wired LAN is set up using Ethernet cables while wireless LAN is set up using Wi-Fi that connects network devices without making any physical connection. A wired LAN network is more secure than a wireless LAN network but it is difficult to relocate.
- **Select networking device:** Select switch or router with enough ports to connect all workstations within the network. The choice of networking device is based on the requirements of the network.
- **Configure ports:** Configure WAN ports according to the information provided by ISP (Internet Service Provider). Also, configure LAN ports of cable routers such that there are enough addresses available for all the workstations within the network. A cable router acts as DHCP

(Dynamic Host Configuration Server) server that automatically allocates addresses to all the devices connected to the network.

- **Make connections:** Connect all the devices using wires to configure a LAN network. Standard Ethernet cables are used to connect workstations and servers while Ethernet crossover cable is used to connect the switch to cable routers by connecting the standard port of the switch with router's LAN port. For wireless LAN, connect all the devices to Wi-Fi with SSID (Service Set Identifier) provided by the router or switch to configure the LAN network.
- **Test the network:** Test each of the workstation connected to the network and ensure every workstation have access to network services.

3. Explain: (a) ping (b) getmac (c) arp (d) nslookup (e) tracert.

a) Ping:

Ping is a utility used to test the reachability of a host on an Internet Protocol (IP) network. It sends ICMP (Internet Control Message Protocol) echo request packets to the target host and waits for ICMP echo reply packets. The time taken for the round trip is measured, known as the ping time or latency, and is displayed in milliseconds (ms).

Example: $ ping 192.168.0.102

```
Pinging 192.168.0.102 with 32 bytes of data:
Reply from 192.168.0.102: bytes=32 time<1ms TTL=128
Reply from 192.168.0.102: bytes=32 time<1ms TTL=128
Reply from 192.168.0.102: bytes=32 time<1ms TTL=128
Reply from 192.168.0.102: bytes=32 time<1ms TTL=128
Ping statistics for 192.168.0.102:
   Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
   Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

b) Getmac:

Getmac is a command-line utility in Linux that retrieves the Media Access Control (MAC) address and the list of network protocols associated with each address for all network cards in a computer. The MAC address is a unique identifier assigned to the network interface card (NIC) by the manufacturer.

Example: $ GETMAC

```
Physical Address    Transport Name
=================== ==========================================================
FC-5C-EE-95-31-DB   Media disconnected
60-45-2E-6C-B7-4F   \Device\Tcpip_{3D3C291C-F876-4A04-83EF-E244754DF90D}
```

c) **ARP (Address Resolution Protocol):**

ARP is a protocol used for mapping an IP address to a MAC address in a local network. When a device needs to communicate with another device on the same subnet, it uses ARP to find the MAC address associated with the destination IP address.

Example: $ arp -a

```
Interface: 192.168.0.102 --- 0x6
  Internet Address      Physical Address      Type
  192.168.0.1           a8-63-7d-40-98-4a     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

d) **Nslookup:**

Nslookup is a command-line tool used to query Domain Name System (DNS) servers to obtain domain name or IP address mapping, or other DNS records. It can be used to troubleshoot DNS-related problems or to obtain information about a domain.

Example: $ nslookup

```
Default Server:  UnKnown
Address:  ::

> server 8.8.8.8
Default Server: [8.8.8.8]
Address:  8.8.8.8

> google.com
Server: [8.8.8.8]
Address:  8.8.8.8

Non-authoritative answer:
Name:    google.com
Addresses:  2404:6800:4009:82b::200e
          142.250.192.14
```

e) **Tracert (Traceroute):**

Tracert is a command-line tool that traces the route packets take to reach a destination host. It shows the IP addresses of the routers in the path to the destination, along with the round-trip times (latency) to each router. Tracert helps in diagnosing network connectivity issues and identifying bottlenecks.

Example: $ tracert www.google.com

```
Tracing route to www.google.com [142.250.182.228]
over a maximum of 30 hops:

  1     2 ms   243 ms     1 ms  192.168.0.1
  2     3 ms     3 ms     2 ms  172.23.129.1
```

4. Explain: (a) netstat (b) route (c) ipconfig (d)ifconfig (e) scp.

a) netstat:

The **'netstat'** command displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.

Example: $ netstat -a

```
Active Connections

Proto  Local Address          Foreign Address        State
  TCP    127.0.0.1:27060        IMPULSE-PRO:0          LISTENING
  TCP    127.0.0.1:49682        IMPULSE-PRO:0          LISTENING
  TCP    127.0.0.1:49682        IMPULSE-PRO:50844      ESTABLISHED
  TCP    127.0.0.1:49688        IMPULSE-PRO:49689      ESTABLISHED
  TCP    127.0.0.1:49689        IMPULSE-PRO:49688      ESTABLISHED
```

b) route:

The **'route'** command is used to view and manipulate the IP routing table in a Unix-like operating system. It displays or modifies entries in the kernel's routing tables. It is used to set up static routes to specific hosts or networks and to manage the routing table.

Example:

```
$ route PRINT
===========================================================================
Interface List
 12...60 45 2e 6c b7 50 ......Microsoft Wi-Fi Direct Virtual Adapter
 13...62 45 2e 6c b7 4f ......Microsoft Wi-Fi Direct Virtual Adapter #2
  6...60 45 2e 6c b7 4f ......Intel(R) Wi-Fi 6E AX211 160MHz
  3...fc 5c ee 95 31 db ......Realtek PCIe GbE Family Controller
  1...........................Software Loopback Interface 1
===========================================================================
```

IPv4 Route Table

===

Active Routes:

Network Destination	Netmask	Gateway	Interface	Metric
0.0.0.0	0.0.0.0	192.168.0.1	192.168.0.102	40
127.0.0.0	255.0.0.0	On-link	127.0.0.1	331
127.0.0.1	255.255.255.255	On-link	127.0.0.1	331
127.255.255.255	255.255.255.255	On-link	127.0.0.1	331
192.168.0.0	255.255.255.0	On-link	192.168.0.102	296
192.168.0.102	255.255.255.255	On-link	192.168.0.102	296
192.168.0.255	255.255.255.255	On-link	192.168.0.102	296

===

c) ipconfig:

The ‘ipconfig’ command is used in Unix-like operating systems to configure, manage, and query network interfaces. It is used to assign an address to a network interface and enable or disable interfaces. It also displays information about currently active network interfaces.

Example: $ ipconfig

```
Windows IP Configuration


Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : Home
   Link-local IPv6 Address . . . . . : fe80::743f:6ba:b204:fe71%6
   IPv4 Address. . . . . . . . . . . : 192.168.0.102
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::1%6
                                       192.168.0.1

Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

d) scp:

Secure Copy (scp) is a command-line tool used to securely transfer files between a local and a remote host or between two remote hosts over an SSH (Secure Shell) connection. It is used to copy files and directories securely over the network, similar to the **‘cp’** command but with the added security of encryption provided by SSH.

Example: $ scp file1.txt file2.txt file3.txt file4.txt user1@host1.com:/home/user1/Desktop

Assignment – 3

1. **Write and run a TCP client and a TCP server program using C / 'JAVA' language in UNIX / LINUX / Windows as per the following details:**
   i. **Establish a TCP connection between a TCP client and a TCP server.**
   ii. **Client program will send some messages (echo client) to the server.**
   iii. **Server will return the messages/acknowledgement (echo server) to the client.**

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of 'vim' editor. Then we can write the code to setup TCPserver and TCPclient locally.

TCPserver:

```
import java.io.*;

import java.net.*;


public class TCPserver {

    public static void main(String[] args) {

        try (ServerSocket serverSocket = new ServerSocket(6789)) {

            System.out.println("Server is listening on port 6789");

            while (true) {

                Socket socket = serverSocket.accept();

                System.out.println("New client connected");

                new ServerThread(socket).start();

            }

        } catch (IOException ex) {

            ex.printStackTrace();

        }

    }

}


class ServerThread extends Thread {

    private Socket socket;


    public ServerThread(Socket socket) {

        this.socket = socket;
```

```java
    }

    public void run() {
        try (InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input));
             OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true)) {

            String text;
            while ((text = reader.readLine()) != null) {
                System.out.println("Received: " + text);
                writer.println("Echo: " + text);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

TCPclient:

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

public class TCPclient {
    public static void main(String[] args) {
        String hostname = "localhost";
        int port = 6789;

        try (Socket socket = new Socket(hostname, port);
             OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true);
             InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input));
             BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {
```

```
            String text;

            while (true) {
                System.out.print("Enter message: ");
                text = consoleReader.readLine();
                writer.println(text);
                String response = reader.readLine();
                System.out.println(response);

                if ("exit".equalsIgnoreCase(text)) {
                    break;
                }
            }
        } catch (UnknownHostException ex) {
            System.out.println("Server not found: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

Output:

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash
$ vim TCPserver.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash
$ javac TCPserver.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash
$ java TCPserver
Server is listening on port 6789
New client connected
Received: Hello server
Received: It is the client
Received: exit


Panther@IMPULSE-PRO MINGW64 /d/Codes/bash
$
```

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash
$ vim TCPclient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash
$ javac TCPclient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash
$ java TCPclient
Enter message: Hello server
Echo: Hello server
Enter message: It is the client
Echo: It is the client
Enter message: exit
Echo: exit

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash
$
```

Assignment – 4

1. **Write and run a TCP client and a TCP server program using C / 'JAVA' language in Unix/Linux according to the following specifications:**
    i. **Establish a TCP connection between a TCP client and a TCP server.**
    ii. **Client program will send a string of characters to the specific server.**
    iii. **Server program will count the number of characters (Length) in the given string and return back to the respective client.**

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of ‘vim’ editor. Then we can write the code to setup TCPserver and TCPclient locally.

TCPserver:

```
import java.io.*;

import java.net.*;


public class TCPserver {

    public static void main(String[] args) {

        try (ServerSocket serverSocket = new ServerSocket(6789)) {

            System.out.println("Server is listening on port 6789");

            while (true) {

                Socket socket = serverSocket.accept();

                System.out.println("New client connected");

                new ServerThread(socket).start();

            }

        } catch (IOException ex) {

            ex.printStackTrace();

        }

    }

}


class ServerThread extends Thread {

    private Socket socket;


    public ServerThread(Socket socket) {
```

```
        this.socket = socket;
    }

    public void run() {
        try (InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input));
             OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true)) {

            String text;
            while ((text = reader.readLine()) != null) {
                System.out.println("Received: " + text);
                int length = text.length();
                writer.println("Length: " + length);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

TCPclient:

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

public class TCPclient {
    public static void main(String[] args) {
        String hostname = "localhost";
        int port = 6789;
        try (Socket socket = new Socket(hostname, port);
             OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true);
             InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input));
             BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {
```

```
            String text;

            while (true) {
                System.out.print("Enter string: ");
                text = consoleReader.readLine();
                writer.println(text);
                String response = reader.readLine();
                System.out.println(response);

                if ("exit".equalsIgnoreCase(text)) {
                    break;
                }
            }
        } catch (UnknownHostException ex) {
            System.out.println("Server not found: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

Output:

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 4
$ vim TCPserver.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 4
$ javac TCPserver.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 4
$ java TCPserver
Server is listening on port 6789
New client connected
Received: Hello
Received: I am client
Received: exit


Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 4
$ |
```

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 4
$ vim TCPclient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 4
$ javac TCPclient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 4
$ java TCPclient
Enter string: Hello
Length: 5
Enter string: I am client
Length: 11
Enter string: exit
Length: 4

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 4
$ |
```

Assignment – 5

1. **Write and run a TCP client and a TCP server program using C / 'JAVA' language in Unix/Linux according to the following specification Write and run a TCP Client and a TCP Server program using C / 'JAVA' language in Unix/Linux operating system as per the following specifications:**
    i. **Establish a TCP connection between a TCP client and a TCP server.**
    ii. **Client will send a text string to the Server.**
    iii. **Server will check whether the string is a palindrome or not. Result will be sent to the respective Client.**

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of 'vim' editor. Then we can write the code to setup TCPserver and TCPclient locally.

TCPserver:

```
import java.io.*;

import java.net.*;


public class TCPserver {

    public static void main(String[] args) {

        try (ServerSocket serverSocket = new ServerSocket(6789)) {

            System.out.println("Server is listening on port 6789");

            while (true) {

                Socket socket = serverSocket.accept();

                System.out.println("New client connected");

                new ServerThread(socket).start();

            }

        } catch (IOException ex) {

            ex.printStackTrace();

        }

    }

}


class ServerThread extends Thread {

    private Socket socket;
```

```
    public ServerThread(Socket socket) {
        this.socket = socket;
    }


    public void run() {
        try (InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input));
             OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true)) {


            String text;
            while ((text = reader.readLine()) != null) {
                System.out.println("Received: " + text);
                String result = isPalindrome(text) ? "Palindrome" : "Not Palindrome";
                writer.println(result);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }


    private boolean isPalindrome(String text) {
        String clean = text.replaceAll("\\s+", "").toLowerCase();
        int length = clean.length();
        for (int i = 0; i < length / 2; i++) {
            if (clean.charAt(i) != clean.charAt(length - 1 - i)) {
                return false;
            }
        }
        return true;
    }
}
```

TCPclient:

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

public class TCPclient {
  public static void main(String[] args) {
    String hostname = "localhost";
    int port = 6789;

    try (Socket socket = new Socket(hostname, port);
       OutputStream output = socket.getOutputStream();
       PrintWriter writer = new PrintWriter(output, true);
       InputStream input = socket.getInputStream();
       BufferedReader reader = new BufferedReader(new InputStreamReader(input));
       BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {
      String text;
      while (true) {
        System.out.print("Enter string: ");
        text = consoleReader.readLine();
        writer.println(text);
        String response = reader.readLine();
        System.out.println(response);

        if ("exit".equalsIgnoreCase(text)) {
          break;
        }
      }
    } catch (UnknownHostException ex) {
      System.out.println("Server not found: " + ex.getMessage());
    } catch (IOException ex) {
      System.out.println("I/O error: " + ex.getMessage());
    }
  }
}
```

Output:

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 5
$ vim TCPserver.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 5
$ javac TCPserver.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 5
$ java TCPserver
Server is listening on port 6789
New client connected
Received: Bob
Received: Red
Received: exit

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 5
$ |
```

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 5
$ vim TCPclient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 5
$ javac TCPclient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 5
$ java TCPclient
Enter string: Bob
Palindrome
Enter string: Red
Not Palindrome
Enter string: exit
Not Palindrome

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 5
$ |
```

2. **Write and run a TCP client and a server program using C / JAVA-language in Linux/Unix environment to perform the following tasks:**
    i. **Establish a TCP connection between a TCP client and a TCP server.**
    ii. **The client program sends two strings to the server.**
    iii. **The server program returns 'YES' if both the strings are equal and `NO' if both are not equal.**

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of 'vim' editor. Then we can write the code to setup TCPserver and TCPclient locally.

TCPserver:

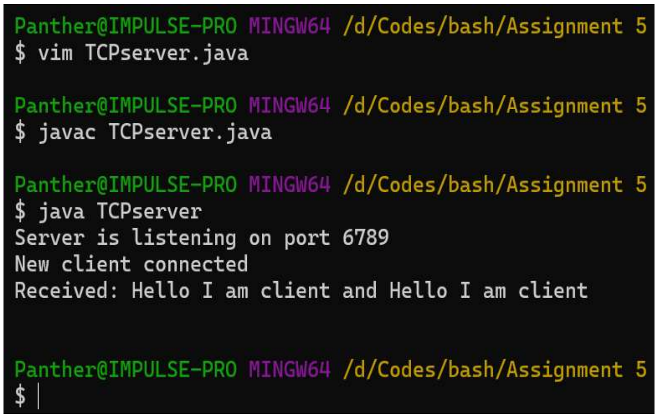
```
import java.io.*;

import java.net.*;


public class TCPserver {

  public static void main(String[] args) {

    try (ServerSocket serverSocket = new ServerSocket(6789)) {

      System.out.println("Server is listening on port 6789");

      while (true) {

        Socket socket = serverSocket.accept();

        System.out.println("New client connected");

        new ServerThread(socket).start();

      }

    } catch (IOException ex) {

      ex.printStackTrace();

    }
```

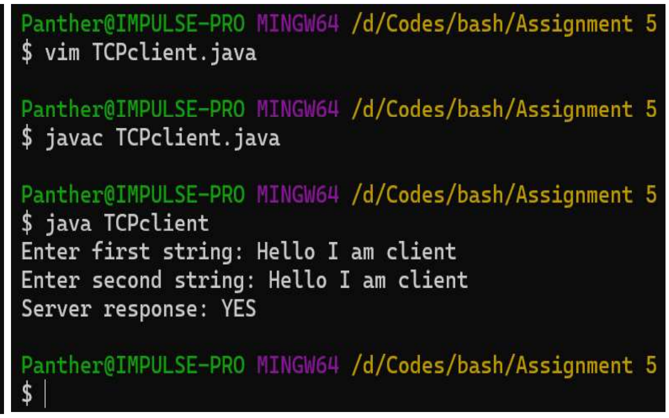
```java
    }
}

class ServerThread extends Thread {
    private Socket socket;

    public ServerThread(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input));
             OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true)) {

            String str1 = reader.readLine();
            String str2 = reader.readLine();

            System.out.println("Received: " + str1 + " and " + str2);

            if (str1 != null && str1.equals(str2)) {
                writer.println("YES");
            } else {
                writer.println("NO");
            }

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

TCPclient:

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

public class TCPclient {
   public static void main(String[] args) {
      String hostname = "localhost";
      int port = 6789;

      try (Socket socket = new Socket(hostname, port);
           OutputStream output = socket.getOutputStream();
           PrintWriter writer = new PrintWriter(output, true);
           InputStream input = socket.getInputStream();
           BufferedReader reader = new BufferedReader(new InputStreamReader(input));
           BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {

         System.out.print("Enter first string: ");
         String str1 = consoleReader.readLine();
         System.out.print("Enter second string: ");
         String str2 = consoleReader.readLine();

         writer.println(str1);
         writer.println(str2);

         String response = reader.readLine();
         System.out.println("Server response: " + response);

      } catch (UnknownHostException ex) {
         System.out.println("Server not found: " + ex.getMessage());
      } catch (IOException ex) {
         System.out.println("I/O error: " + ex.getMessage());
      }
   }
}
```

Output:





Assignment – 6

1. Write a Socket program to implement Day Time Server-client using TCP/UDP protocol:

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of ‘vim’ editor. Then we can write the code to setup TCPDayTimeServer and TCPDayTimeClient locally.

TCPDayTimeServer:

```
import java.io.*;

import java.net.*;

import java.util.Date;


public class TCPDayTimeServer {

  public static void main(String[] args) {

    try (ServerSocket serverSocket = new ServerSocket(6789)) {

      System.out.println("Day Time Server is listening on port 6789");

      while (true) {

        try (Socket socket = serverSocket.accept();

             PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {

          String dateTime = new Date().toString();

          System.out.println("Client connected, sending date and time: " + dateTime);

          out.println(dateTime);

        } catch (IOException e) {

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

          e.printStackTrace();
```

```
            }
        }
    } catch (IOException e) {
        System.out.println("Could not listen on port 6789");
        e.printStackTrace();
    }
  }
}
```

TCPDayTimeClient:

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

public class TCPDayTimeClient {
  public static void main(String[] args) {
    String hostname = "localhost";
    int port = 6789;

    try (Socket socket = new Socket(hostname, port);
         BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

        String dateTime = in.readLine();
        System.out.println("Current date and time from server: " + dateTime);

    } catch (UnknownHostException ex) {
        System.out.println("Server not found: " + ex.getMessage());
    } catch (IOException ex) {
        System.out.println("I/O error: " + ex.getMessage());
    }
  }
}
```

Output:

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 6
$ vim TCPDayTimeServer.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 6
$ javac TCPDayTimeServer.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 6
$ java TCPDayTimeServer
Day Time Server is listening on port 6789
Client connected, sending date and time: Thu May 23 19:25:50 IST 2024


Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 6
$ |
```

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 6
$ vim TCPDayTimeClient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 6
$ javac TCPDayTimeClient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 6
$ java TCPDayTimeClient
Current date and time from server: Thu May 23 19:25:50 IST 2024

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 6
$ |
```

2. Write a Socket program to implement CALCULATOR in client-server socket using TCP protocol.

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of 'vim' editor. Then we can write the code to setup TCPCalculatorServer and TCPCalculatorClient locally.

TCPCalculatorServer:

```
import java.io.*;

import java.net.*;


public class TCPCalculatorServer {

  public static void main(String[] args) {

    try (ServerSocket serverSocket = new ServerSocket(6789)) {

      System.out.println("Calculator Server is listening on port 6789");

      while (true) {

        new CalculatorThread(serverSocket.accept()).start();

      }

    } catch (IOException ex) {
```

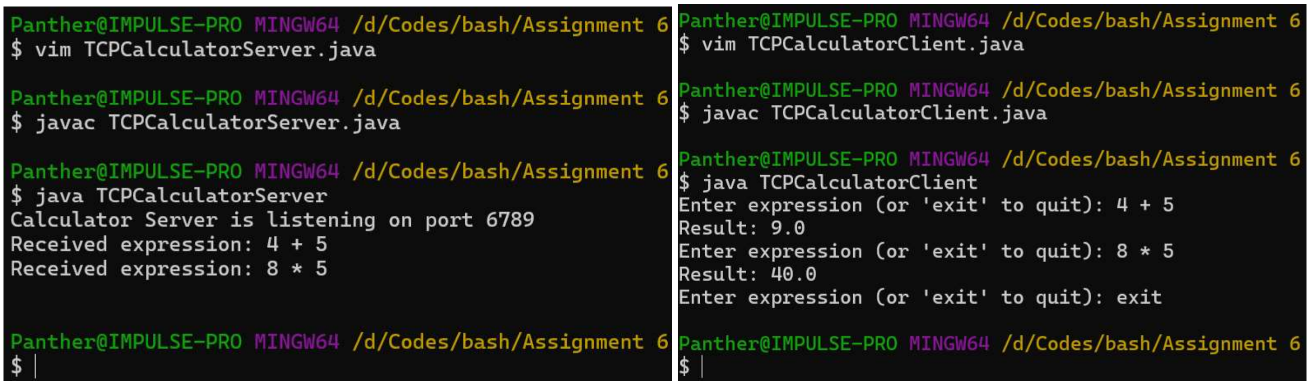
```
            System.out.println("Server exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}

class CalculatorThread extends Thread {
    private Socket socket;

    public CalculatorThread(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input));
             OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true)) {

            String expression;
            while ((expression = reader.readLine()) != null) {
                System.out.println("Received expression: " + expression);
                String result = calculate(expression);
                writer.println(result);
            }
        } catch (IOException ex) {
            System.out.println("Server thread exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }

    private String calculate(String expression) {
        try {
            String[] tokens = expression.split(" ");
```

```
            if (tokens.length != 3) {
                return "Error: Invalid expression format. Use: <number1> <operator> <number2>";
            }

            double num1 = Double.parseDouble(tokens[0]);
            String operator = tokens[1];
            double num2 = Double.parseDouble(tokens[2]);

            double result;
            switch (operator) {
                case "+":
                    result = num1 + num2;
                    break;
                case "-":
                    result = num1 - num2;
                    break;
                case "*":
                    result = num1 * num2;
                    break;
                case "/":
                    if (num2 == 0) {
                        return "Error: Division by zero";
                    }
                    result = num1 / num2;
                    break;
                default:
                    return "Error: Invalid operator. Use one of +, -, *, /";
            }
            return "Result: " + result;
        } catch (NumberFormatException e) {
            return "Error: Invalid number format";
        }
    }
}
```

TCPCalculatorClient:

```
import java.io.*;

import java.net.*;


public class TCPCalculatorClient {

  public static void main(String[] args) {

    String hostname = "localhost";

    int port = 6789;


    try (Socket socket = new Socket(hostname, port);

       OutputStream output = socket.getOutputStream();

       PrintWriter writer = new PrintWriter(output, true);

       InputStream input = socket.getInputStream();

       BufferedReader reader = new BufferedReader(new InputStreamReader(input));

       BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {


      String expression;

      while (true) {

        System.out.print("Enter expression (or 'exit' to quit): ");

        expression = consoleReader.readLine();

        if ("exit".equalsIgnoreCase(expression)) {

          break;

        }

        writer.println(expression);

        String response = reader.readLine();

        System.out.println(response);

      }

    } catch (UnknownHostException ex) {

      System.out.println("Server not found: " + ex.getMessage());

    } catch (IOException ex) {

      System.out.println("I/O error: " + ex.getMessage());

    }

  }

}
```

Output:



Assignment – 7

1. **Write a Socket program: Client Request – Server sends its IPADDRESS using TCP/UDP protocol.**

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of 'vim' editor. Then we can write the code to setup TCPIPAddressServer and TCPIPAddressClient locally.

TCPIPAddressServer:

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

public class TCPIPAddressServer {
  public static void main(String[] args) {
    try (ServerSocket serverSocket = new ServerSocket(6789)) {
      System.out.println("Server is listening on port 6789");
      while (true) {
        new IPAddressServerThread(serverSocket.accept()).start();
      }
    } catch (IOException ex) {
      System.out.println("Server exception: " + ex.getMessage());
      ex.printStackTrace();
    }
  }
}
class IPAddressServerThread extends Thread {
  private Socket socket;
```

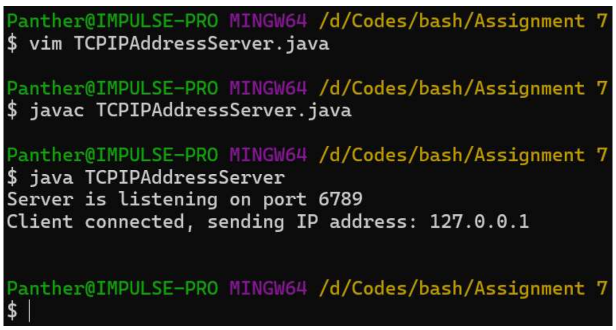
```
    public IPAddressServerThread(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true)) {

            InetAddress serverAddress = socket.getLocalAddress();
            String ipAddress = serverAddress.getHostAddress();
            System.out.println("Client connected, sending IP address: " + ipAddress);
            writer.println(ipAddress);

        } catch (IOException ex) {
            System.out.println("Server thread exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

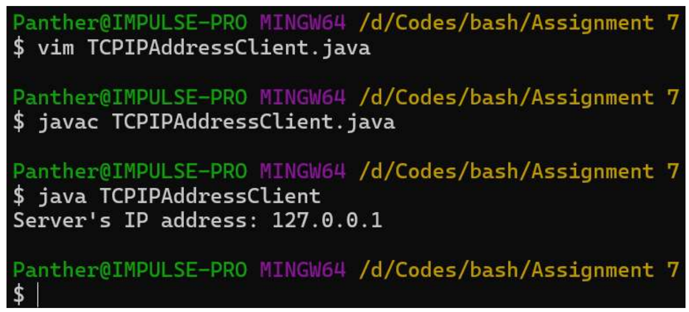
TCPIPAddressClient:

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

public class TCPIPAddressClient {
    public static void main(String[] args) {
        String hostname = "localhost";
        int port = 6789;

        try (Socket socket = new Socket(hostname, port);
             InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input))) {

            String ipAddress = reader.readLine();
```

```
            System.out.println("Server's IP address: " + ipAddress);

        } catch (UnknownHostException ex) {
            System.out.println("Server not found: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

Output:





2. **Write and run a TCP client and a TCP server program using C / 'JAVA' language in Unix/Linux according to the following specifications.**
   i. **Establish a TCP connection between a TCP client and a TCP server.**
   ii. **Client program will send a number to the specific server.**
   iii. **Server program will find the factorial in the given number and return result back to the respective client.**

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of ‘vim’ editor. Then we can write the code to setup TCPFactorialServer and TCPFactorialClient locally.

TCPFactorialServer:

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

public class TCPFactorialServer {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(6789)) {
            System.out.println("Server is listening on port 6789");
```

```
            while (true) {
                new FactorialServerThread(serverSocket.accept()).start();
            }
        } catch (IOException ex) {
            System.out.println("Server exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}

class FactorialServerThread extends Thread {
    private Socket socket;

    public FactorialServerThread(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input));
             OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true)) {

            String numberStr = reader.readLine();
            int number = Integer.parseInt(numberStr);
            System.out.println("Received number: " + number);

            long result = factorial(number);
            writer.println(result);

        } catch (IOException | NumberFormatException ex) {
            System.out.println("Server thread exception: " + ex.getMessage());
            ex.printStackTrace();
        }
```

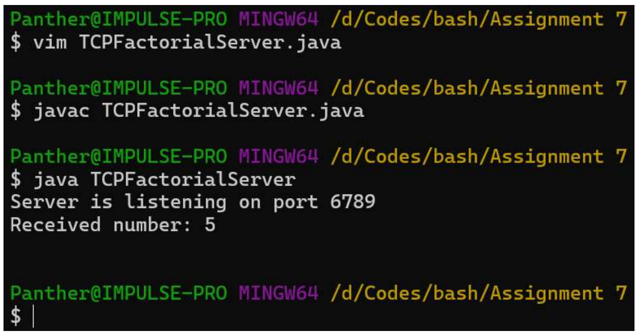
```
    }

    private long factorial(int n) {
        if (n == 0) return 1;
        long result = 1;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```

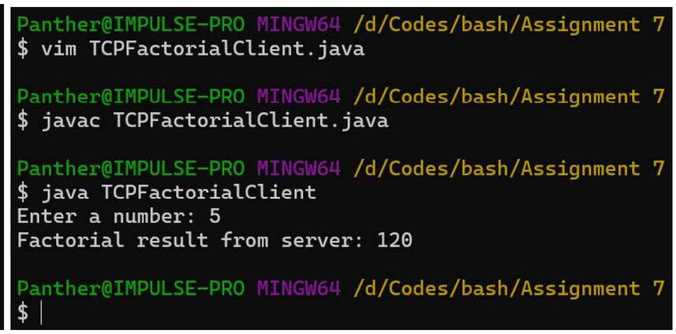
TCPFactorialClient:

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

public class TCPFactorialClient {
    public static void main(String[] args) {
        String hostname = "localhost";
        int port = 6789;

        try (Socket socket = new Socket(hostname, port);
             OutputStream output = socket.getOutputStream();
             PrintWriter writer = new PrintWriter(output, true);
             InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input));
             BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {

            System.out.print("Enter a number: ");
            String number = consoleReader.readLine();

            writer.println(number);
            String response = reader.readLine();
            System.out.println("Factorial result from server: " + response);
```

```
        } catch (UnknownHostException ex) {
            System.out.println("Server not found: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

Output:





Assignment – 8

1. **Write a socket program to implement multi client-server (echo client-server) using TCP protocol.**

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of ‘vim’ editor. Then we can write the code to setup MultiClientEchoServer and EchoClient locally.

MultiClientEchoServer:

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

public class MultiClientEchoServer {
    public static final int PORT = 6789;
    public static List<ClientHandler> clients = new ArrayList<>();

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server is listening on port " + PORT);
```

```
            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("New client connected: " + clientSocket);
                ClientHandler clientHandler = new ClientHandler(clientSocket);
                clients.add(clientHandler);
                clientHandler.start();
            }
        } catch (IOException ex) {
            System.out.println("Server exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }

    public static void broadcastMessage(String message, ClientHandler sender) {
        for (ClientHandler client : clients) {
            if (client != sender) {
                client.sendMessage(message);
            }
        }
    }
}
class ClientHandler extends Thread {
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;

    public ClientHandler(Socket socket) {
        this.clientSocket = socket;
        try {
            this.out = new PrintWriter(socket.getOutputStream(), true);
            this.in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        } catch (IOException e) {
            System.out.println("Error creating client handler: " + e.getMessage());
            e.printStackTrace();
```

```
            }
        }

        public void run() {
            try {
                String message;
                while ((message = in.readLine()) != null) {
                    System.out.println("Received from client: " + message);
                    MultiClientEchoServer.broadcastMessage(message, this);
                }
            } catch (IOException e) {
                System.out.println("Error handling client input: " + e.getMessage());
                e.printStackTrace();
            } finally {
                try {
                    clientSocket.close();
                    MultiClientEchoServer.clients.remove(this);
                } catch (IOException e) {
                    System.out.println("Error closing client socket: " + e.getMessage());
                    e.printStackTrace();
                }
            }
        }

        public void sendMessage(String message) {
            out.println(message);
        }
    }
```

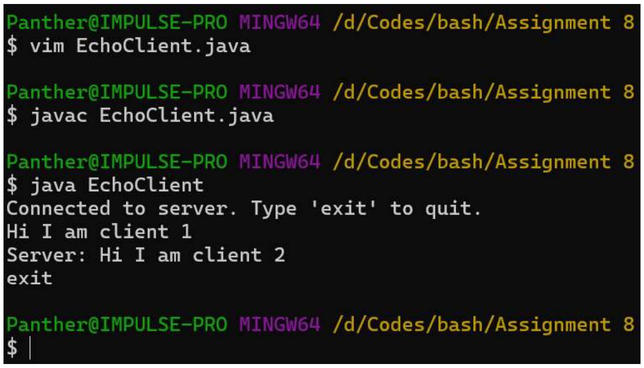
EchoClient:

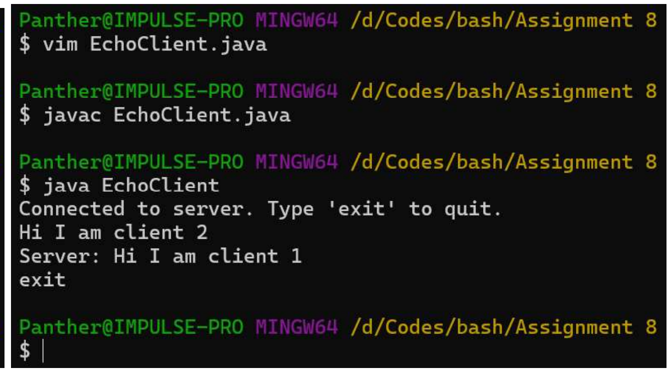
```
import java.io.*;
import java.net.*;

public class EchoClient {
    private static final String SERVER_ADDRESS = "localhost";
```

```java
        private static final int SERVER_PORT = 6789;

        public static void main(String[] args) {

            try (Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT);

                 PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

                 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

                 BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {

                System.out.println("Connected to server. Type 'exit' to quit.");

                String userInput;

                while ((userInput = consoleReader.readLine()) != null) {

                    if ("exit".equalsIgnoreCase(userInput)) {

                        break;

                    }

                    out.println(userInput);

                    String serverResponse = in.readLine();

                    System.out.println("Server: " + serverResponse);

                }

            } catch (IOException e) {

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

                e.printStackTrace();

            }

        }

    }
```

Output:

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ vim MultiClientEchoServer.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ javac MultiClientEchoServer.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ java MultiClientEchoServer
Server is listening on port 6789
New client connected: Socket[addr=/127.0.0.1,port=50297,localport=6789]
New client connected: Socket[addr=/127.0.0.1,port=50298,localport=6789]
Received from client: Hi I am client 1
Received from client: Hi I am client 2


Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ 
```





2. Write a socket program to implement multi client-server (Result sheet) using TCP protocol.

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of 'vim' editor. Then we can write the code to setup MultiClientResultSheetServer and ResultSheetClient locally.

MultiClientResultSheetServer:

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

public class MultiClientResultSheetServer {
  public static final int PORT = 6789;
  public static Map<String, Integer> resultSheet = new HashMap<>();
  public static List<ClientHandler> clients = new ArrayList<>();

  public static void main(String[] args) {
    try (ServerSocket serverSocket = new ServerSocket(PORT)) {
      System.out.println("Server is listening on port " + PORT);
      while (true) {
        Socket clientSocket = serverSocket.accept();
        System.out.println("New client connected: " + clientSocket);
        ClientHandler clientHandler = new ClientHandler(clientSocket);
        clients.add(clientHandler);
        clientHandler.start();
      }
    } catch (IOException ex) {
      System.out.println("Server exception: " + ex.getMessage());
```

```
            ex.printStackTrace();
        }
    }

    public static synchronized void updateResult(String studentName, int marks) {
        resultSheet.put(studentName, marks);
        broadcastResultSheet();
    }

    public static synchronized void removeResult(String studentName) {
        resultSheet.remove(studentName);
        broadcastResultSheet();
    }

    public static synchronized void broadcastResultSheet() {
        for (ClientHandler client : clients) {
            client.sendResultSheet(resultSheet);
        }
    }
}

class ClientHandler extends Thread {
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;

    public ClientHandler(Socket socket) {
        this.clientSocket = socket;
        try {
            this.out = new PrintWriter(socket.getOutputStream(), true);
            this.in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        } catch (IOException e) {
            System.out.println("Error creating client handler: " + e.getMessage());
            e.printStackTrace();
```

```
    }
}

public void run() {
    try {
        while (true) {
            String request = in.readLine();
            if (request == null) break;
            String[] tokens = request.split(" ");
            if (tokens.length < 2) continue;
            String command = tokens[0];
            String studentName = tokens[1];
            if ("ADD".equalsIgnoreCase(command)) {
                if (tokens.length < 3) continue;
                int marks = Integer.parseInt(tokens[2]);
                MultiClientResultSheetServer.updateResult(studentName, marks);
            } else if ("REMOVE".equalsIgnoreCase(command)) {
                MultiClientResultSheetServer.removeResult(studentName);
            }
        }
    } catch (IOException | NumberFormatException e) {
        System.out.println("Error handling client input: " + e.getMessage());
        e.printStackTrace();
    } finally {
        try {
            clientSocket.close();
            MultiClientResultSheetServer.clients.remove(this);
        } catch (IOException e) {
            System.out.println("Error closing client socket: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```java
    public void sendResultSheet(Map<String, Integer> resultSheet) {
        out.println("RESULT_SHEET");
        for (Map.Entry<String, Integer> entry : resultSheet.entrySet()) {
            out.println(entry.getKey() + " " + entry.getValue());
        }
        out.println("END");
    }
}
```

ResultSheetClient:

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

public class ResultSheetClient {
    private static final String SERVER_ADDRESS = "localhost";
    private static final int SERVER_PORT = 6789;

    public static void main(String[] args) {
        try (Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
             BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {

            System.out.println("Connected to server. Type 'exit' to quit.");

            String userInput;
            while ((userInput = consoleReader.readLine()) != null) {
                if ("exit".equalsIgnoreCase(userInput)) {
                    break;
                }
                out.println(userInput);
                String serverResponse;
                while (!(serverResponse = in.readLine()).equals("END")) {
                    System.out.println("Server: " + serverResponse);
```

```
                }
            }
        } catch (IOException e) {
            System.out.println("Error connecting to server: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Output:

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ vim MultiClientResultSheetServer.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ javac MultiClientResultSheetServer.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ java MultiClientResultSheetServer
Server is listening on port 6789
New client connected: Socket[addr=/127.0.0.1,port=50504,localport=6789]
New client connected: Socket[addr=/127.0.0.1,port=50505,localport=6789]


Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ 
```

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ vim ResultSheetClient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ javac ResultSheetClient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ java ResultSheetClient
Connected to server. Type 'exit' to quit.
ADD Bob 85
Server: RESULT_SHEET
Server: Alice 95
ADD Dave 65
Server: RESULT_SHEET
Server: Bob 85
Server: Alice 95
exit

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ 
```

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ vim ResultSheetClient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ javac ResultSheetClient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ java ResultSheetClient
Connected to server. Type 'exit' to quit.
ADD Alice 95
Server: RESULT_SHEET
Server: Alice 95
ADD Charlie 75
Server: RESULT_SHEET
Server: Bob 85
Server: Alice 95
REMOVE Bob
Server: RESULT_SHEET
Server: Bob 85
Server: Alice 95
Server: Charlie 75
RESULT_SHEET
Server: RESULT_SHEET
Server: Bob 85
Server: Alice 95
Server: Charlie 75
Server: Dave 65
exit

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 8
$ 
```

Assignment – 9

1. **Write and run a UDP client and a UDP server program using 'JAVA' language in UNIX / LINUX / Windows as per the following details.**
   i. **Client program will send some messages (echo client) to the server.**
   ii. **Server will return the messages/acknowledgement (echo server) to the client.**

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of 'vim' editor. Then we can write the code to setup UDPEchoServer and UDPEchoClient locally.

UDPEchoServer:

```
import java.io.*;

import java.net.*;


public class UDPEchoServer {

   private static final int PORT = 6789;

   private static final int BUFFER_SIZE = 1024;


   public static void main(String[] args) {

      try (DatagramSocket socket = new DatagramSocket(PORT)) {

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

         byte[] buffer = new byte[BUFFER_SIZE];

         while (true) {

            DatagramPacket request = new DatagramPacket(buffer, buffer.length);

            socket.receive(request);

            String message = new String(request.getData(), 0, request.getLength());

            System.out.println("Received from client: " + message);


            InetAddress clientAddress = request.getAddress();

            int clientPort = request.getPort();

            DatagramPacket response = new DatagramPacket(buffer, buffer.length, clientAddress, clientPort);

            socket.send(response);

         }

      } catch (IOException ex) {

         System.out.println("Server exception: " + ex.getMessage());
```

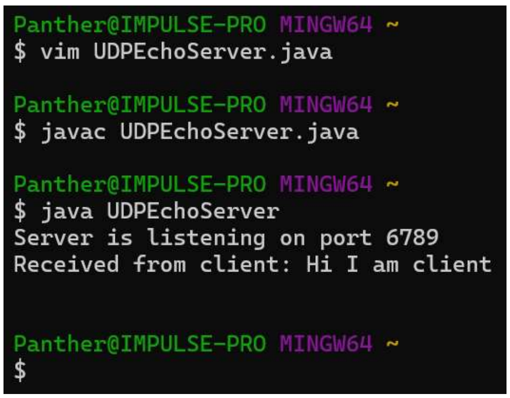
```java
            ex.printStackTrace();
        }
    }
}
```

UDPEchoClient:

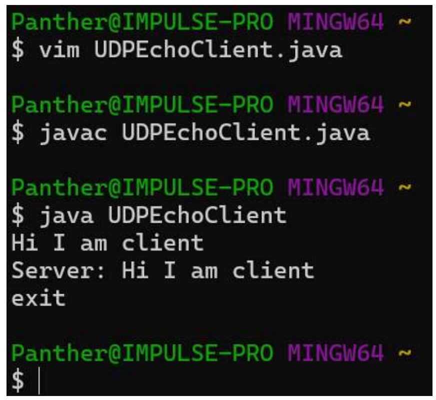
```java
import java.io.*;
import java.net.*;

public class UDPEchoClient {
    private static final String SERVER_ADDRESS = "localhost";
    private static final int SERVER_PORT = 6789;
    private static final int BUFFER_SIZE = 1024;

    public static void main(String[] args) {
        try (DatagramSocket socket = new DatagramSocket();
             BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {

            InetAddress serverAddress = InetAddress.getByName(SERVER_ADDRESS);
            byte[] buffer = new byte[BUFFER_SIZE];

            String userInput;
            while ((userInput = consoleReader.readLine()) != null) {
                if ("exit".equalsIgnoreCase(userInput)) {
                    break;
                }
                buffer = userInput.getBytes();
                DatagramPacket request = new DatagramPacket(buffer, buffer.length, serverAddress,
SERVER_PORT);
                socket.send(request);

                DatagramPacket response = new DatagramPacket(buffer, buffer.length);
                socket.receive(response);
                String serverResponse = new String(response.getData(), 0, response.getLength());
                System.out.println("Server: " + serverResponse);
```

```
            }
        } catch (IOException e) {
            System.out.println("Client exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Output:



```
Panther@IMPULSE-PRO MINGW64 ~
$ vim UDPEchoServer.java

Panther@IMPULSE-PRO MINGW64 ~
$ javac UDPEchoServer.java

Panther@IMPULSE-PRO MINGW64 ~
$ java UDPEchoServer
Server is listening on port 6789
Received from client: Hi I am client


Panther@IMPULSE-PRO MINGW64 ~
$
```



```
Panther@IMPULSE-PRO MINGW64 ~
$ vim UDPEchoClient.java

Panther@IMPULSE-PRO MINGW64 ~
$ javac UDPEchoClient.java

Panther@IMPULSE-PRO MINGW64 ~
$ java UDPEchoClient
Hi I am client
Server: Hi I am client
exit

Panther@IMPULSE-PRO MINGW64 ~
$
```

2. Write a socket program to find addition of two input numbers using UDP Client-server socket.

Ans: -

To perform this at first, we have to create two separate Java files on our directory one for Server and other one for the Client with the help of 'vim' editor. Then we can write the code to setup UDPAdditionServer and UDPAdditionClient locally.

UDPAdditionServer:

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


public class UDPAdditionServer {
    private static final int PORT = 6789;
    private static final int BUFFER_SIZE = 1024;


    public static void main(String[] args) {
        try (DatagramSocket socket = new DatagramSocket(PORT)) {
            System.out.println("Server is listening on port " + PORT);
            byte[] buffer = new byte[BUFFER_SIZE];
            while (true) {
```

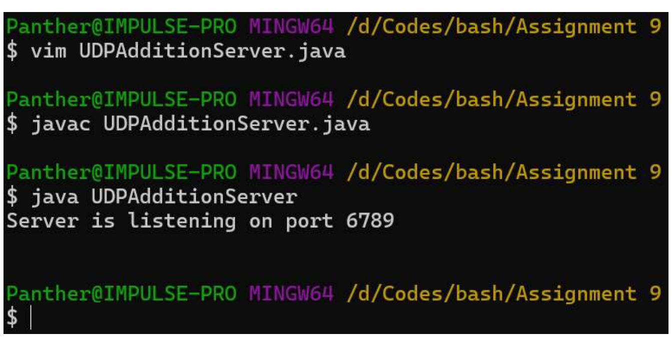
```java
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            socket.receive(request);
            String requestData = new String(request.getData(), 0, request.getLength());
            String[] numbers = requestData.split(" ");
            if (numbers.length < 2) {
                System.out.println("Invalid input received");
                continue;
            }
            int num1 = Integer.parseInt(numbers[0]);
            int num2 = Integer.parseInt(numbers[1]);
            int result = num1 + num2;
            String responseData = String.valueOf(result);
            buffer = responseData.getBytes();
            InetAddress clientAddress = request.getAddress();
            int clientPort = request.getPort();
            DatagramPacket response = new DatagramPacket(buffer, buffer.length, clientAddress, clientPort);
            socket.send(response);
        }
    } catch (IOException | NumberFormatException ex) {
        System.out.println("Server exception: " + ex.getMessage());
        ex.printStackTrace();
    }
  }
}
```

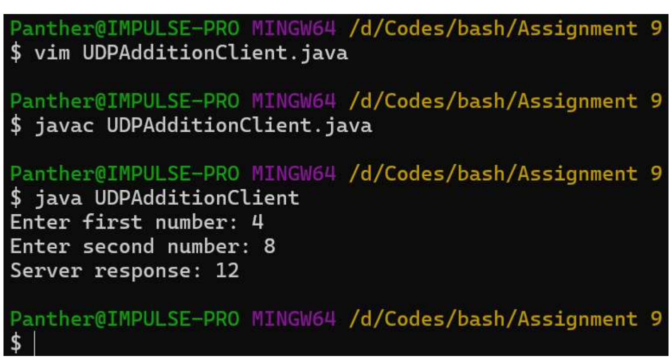
UDPAdditionClient:

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

public class UDPAdditionClient {
    private static final String SERVER_ADDRESS = "localhost";
    private static final int SERVER_PORT = 6789;
    private static final int BUFFER_SIZE = 1024;
    public static void main(String[] args) {
        try (DatagramSocket socket = new DatagramSocket();
```

```java
            BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {
            InetAddress serverAddress = InetAddress.getByName(SERVER_ADDRESS);
            byte[] buffer = new byte[BUFFER_SIZE];
            System.out.print("Enter first number: ");
            String num1 = consoleReader.readLine();
            System.out.print("Enter second number: ");
            String num2 = consoleReader.readLine();
            String requestData = num1 + " " + num2;
            buffer = requestData.getBytes();
            DatagramPacket request = new DatagramPacket(buffer, buffer.length, serverAddress, SERVER_PORT);
            socket.send(request);
            DatagramPacket response = new DatagramPacket(buffer, buffer.length);
            socket.receive(response);
            String result = new String(response.getData(), 0, response.getLength());
            System.out.println("Server response: " + result);
        } catch (IOException e) {
            System.out.println("Client exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Output:





Assignment – 10

1. **Write a socket program to implement stop and wait protocol.**

Ans: -

The Stop-and-Wait Protocol is a simple protocol used for reliable communication over an unreliable channel, such as UDP. In this protocol, the sender sends one packet at a time and waits for an acknowledgment (ACK) from the receiver before sending the next packet.

StopAndWaitServer:

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

public class StopAndWaitServer {
  private static final int PORT = 6789;
  private static final int PACKET_SIZE = 1024;
  public static void main(String[] args) {
    try (DatagramSocket socket = new DatagramSocket(PORT)) {
      System.out.println("Server is listening on port " + PORT);
      byte[] receiveBuffer = new byte[PACKET_SIZE];
      DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
      while (true) {
        socket.receive(receivePacket);
        String message = new String(receivePacket.getData(), 0, receivePacket.getLength());
        System.out.println("Received from client: " + message);
        InetAddress clientAddress = receivePacket.getAddress();
        int clientPort = receivePacket.getPort();
        byte[] sendData = message.toUpperCase().getBytes();
        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress,
clientPort);
        socket.send(sendPacket);
      }
    } catch (IOException ex) {
      System.out.println("Server exception: " + ex.getMessage());
      ex.printStackTrace();
    }
  }
}
```

StopAndWaitClient:

```
import java.io.*;

import java.net.*;


public class StopAndWaitClient {

    private static final String SERVER_ADDRESS = "localhost";

    private static final int SERVER_PORT = 6789;

    private static final int PACKET_SIZE = 1024;

    public static void main(String[] args) {

        try (DatagramSocket socket = new DatagramSocket();

             BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in))) {


            InetAddress serverAddress = InetAddress.getByName(SERVER_ADDRESS);

            while (true) {

                System.out.print("Enter message to send (type 'exit' to quit): ");

                String message = consoleReader.readLine();

                if ("exit".equalsIgnoreCase(message)) {

                    break;

                }

                byte[] sendData = message.getBytes();

                DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress,
SERVER_PORT);

                socket.send(sendPacket);

                byte[] receiveBuffer = new byte[PACKET_SIZE];

                DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);

                socket.receive(receivePacket);

                String modifiedMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());

                System.out.println("Server response: " + modifiedMessage);

            }

        } catch (IOException e) {

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

            e.printStackTrace();

        }

    }

}
```

Output:

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 10
$ vim StopAndWaitServer.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 10
$ javac StopAndWaitServer.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 10
$ java StopAndWaitServer
Server is listening on port 6789
Received from client: Hi I am client.


Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 10
$ |
```

```
Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 10
$ vim StopAndWaitClient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 10
$ javac StopAndWaitClient.java

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 10
$ java StopAndWaitClient
Enter message to send (type 'exit' to quit): Hi I am client.
Server response: HI I AM CLIENT.
Enter message to send (type 'exit' to quit): exit

Panther@IMPULSE-PRO MINGW64 /d/Codes/bash/Assignment 10
$ |
```