**Practical File**

of

**Computer Networks Lab**

**(PCC-CS-602)**

submitted in partial fulfillment of the requirement for the award of degree of

**Bachelor of Technology (B.Tech)**

in

**Computer Engineering**

by

**Name: Dhruv Mishra**

**(Roll No.: 21001003036)**

Under the guidance of

**Mr. Umesh Kumar**

**Assistant Professor**



**Department of Computer Engineering**

**J. C. BOSE UNIVERSITY OF SCIENCE & TECHNOLOGY, YMCA**

**SECTOR-6 FARIDABAD**

**HARYANA-121006.**

**Practicals Performed:-**

**Practical 1: Study of different types of Network cables and Practically implement the cross-wired cable and straight through cable using clamping tool.**

**Apparatus (Components):** RJ-45 connector, Climping Tool, Twisted pair Cable.

**Procedure:** To do these practical following steps should be done:

1. Start by stripping off about 2 inches of the plastic jacket off the end of the cable. Be very careful at this point, as to not nick or cut into the wires, which are inside. Doing so could alter the characteristics of your cable, or even worse render is useless. Check the wires, one more time for nicks or cuts. If there are any, just whack the whole end off, and start over.

2. Spread the wires apart, but be sure to hold onto the base of the jacket with your other hand. You do not want the wires to become untwisted down inside the jacket. Category 5 cable must only have 1/2 of an inch of 'untwisted' wire at the end; otherwise it will be 'out of spec'. At this point, you obviously have ALOT more than 1/2 of an inch of un-twisted wire.

3. You have 2 end jacks, which must be installed on your cable. If you are using a pre-made cable, with one of the ends whacked off, you only have one end to install - the crossed over end. Below are two diagrams, which show how you need to arrange the cables for each type of cable end. Decide at this point which end you are making and examine the associated picture below.

**Diagram shows you how to prepare Cross wired connection.**

****

**Diagram shows you how to prepare straight through wired connection.**

****

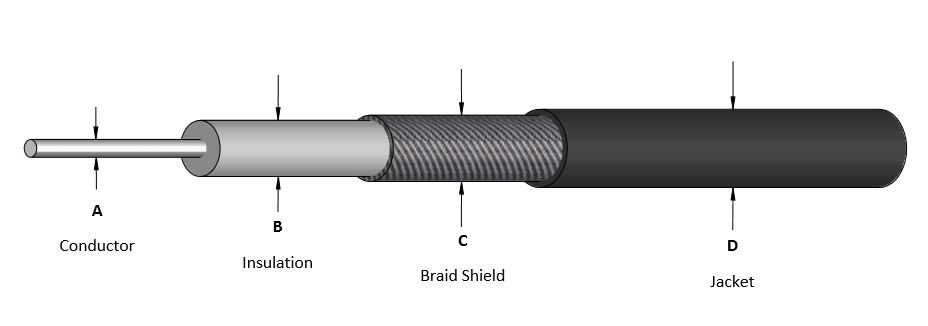
**Theory:**

An ethernet cable allows the user to connect their devices such as computers, mobile phones, routers, etc.; to a network that will allow a user to have internet access, it also carries broadband signals between devices connected through it.

**Types of Ethernet Cables:**

Mainly there are three types of ethernet cables used in LANs i.e., Coaxial cables, Twisted Pair cables, and Fiber optic cables.

**1. Coaxial Cables:**A coaxial cable is used to carry high-frequency electrical signals with low losses. It uses 10Base2 and 10Base5 Ethernet variants. It has a copper conductor in the middle that is surrounded by a dielectric insulator usually made of PVC or Teflon. The dielectric insulator is surrounded by a braided conducting metallic shield which reduces EMI (Electromagnetic Interference) of the metal and outside interference; and finally, the metallic shield is covered by a plastic covering called a sheath usually made of PVC or some other fire-resistant plastic material. Its maximum transmission speed is 10 Mbps. It is usually used in telephone systems, cable TV, etc.

**Design:** Coaxial cable design choices affect physical size, frequency performance, attenuation, power handling capabilities, flexibility, strength, and cost.It consists an inner conductor which might be solid or stranded surrounded by an insulator and, to provide flexibility, it is further surrounded by a copper mesh and which is further surrounded by a plastic or insulating jacket.  


**Types of Coaxial cables:**

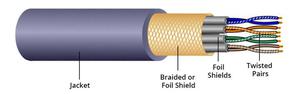
* **Hardline coaxial cable** is used in applications where high signal strength is required; this type is most commonly used. They are used in internet lines and telephone lines.
* **RG-6 Coaxial Cable** is used where better signal quality is required; it has a thicker dielectric insulator, they are used in broadband internet, cable TV, etc.
* **Tri-axial Cable** They offer more bandwidth and interference rejection; they use an additional copper braid shield. Commonly used in connecting cameras and cable TVs. Etc.

**Types of Connectors used in Coaxial cable:**

* BNC (Bayonet Neil Concelman),
* N series Connectors,
* F Type connectors,
* SMA or Subminiature connector,
* TNC (Threaded Neil Concelman), etc.

**2. Twisted Pair Cable:** Twisted pair is a copper wire cable in which two insulated copper wires are twisted around each other to reduce interference or crosstalk. It uses 10BASE-T, 100BASE-T, and some other newer ethernet variants. It uses RJ-45 connectors.

**Design**: A twisted pair cable usually contains two or more conducting wires either shielded by an insulator or not and, further these twisted pairs of wires are coated for protection from any damage.



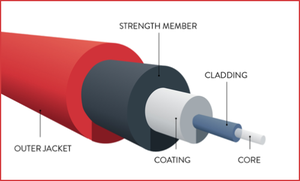
*Twisted Pair Cable*

**Types of twisted pair cable:**

* **Shielded Twisted Pair (STP) Cable:** In STP the wires are covered by a copper braid covering or a foil shield, this foil shield adds a layer that protects it against interference leaking into and out of the cable. Hence, they are used for longer distances and higher transmission rates.
* **Unshielded Twisted Pair (UTP) Cable:** Unshielded twisted pair cable is one of the most commonly used cables in computer networks at present time. UTP consists of two insulated copper wires twisted around one another, the twisting of wires helps in controlling interference.

**3. Fiber Optic Cable:** Fiber optic cables use optical fibers which are made of glass cores surrounded by several layers of cladding material usually made of PVC or Teflon, it transmits data in the form of light signals due to which there are no interference issues in fiber optics. Fiber optics can transmit signals over a very long distance as compared to twisted pairs or coaxial cables. It uses 10BaseF, 100BaseFX, 100BaseBX, 100BaseSX, 1000BaseFx, 1000BaseSX, and 1000BaseBx ethernet variants. Hence, it is capable of carrying information at a great speed.

**Design:**An optical fiber consists of a core and a cladding, chosen for their total internal reflection due to the difference in refractive index between the two. In real optical fibers, the cladding is usually covered with a layer of acrylate or polyimide polymer. The coating protects the fiber from damage and several layers of protective sheathing, depending on the application, are added to form the cable.



*Fiber Optic Cable*

**Types of Fiber Optics:**

* **SMF (Single-mode fiber)-** it uses one single ray of light to transmit data, it is used for long-distance transmission.
* **MMF (Multi-mode Fiber)-** it uses multiple light rays to transmit data, it is comparatively less expensive.

**Types of Connectors Used:**Mainly these four connectors are used with fiber optic cable:

* ST (Straight-tip) Connector
* FC (Fiber Channel) Connector
* SC (Subscriber) Connector
* LC (Lucent) Connector

**Practical 2: Study of following Network Devices in Detail:-**

**• Repeater**

**• Hub**

**• Switch**

**• Bridge**

**• Router**

**• Gate Way**

**Apparatus (Software):** No software or hardware needed.

**Procedure:** Following should be done to understand this practical:-

**1. Repeater:** Functioning at Physical Layer.Arepeater is an electronic device that receives a signal and retransmits it at a higher level and/or higher power, or onto the other side of an obstruction, so that the signal can cover longer distances. Repeater have two ports ,so cannot be use to connect for more than two devices

**2. Hub:** An Ethernet hub, active hub, network hub, repeater hub, hub or concentrator is a device for connecting multiple twisted pair or fiber optic Ethernet devices together and making them act as a single network segment. Hubs work at the physical layer (layer 1) of the OSI model. The device is a form of multiport repeater. Repeater hubs also participate in collision detection, forwarding a jam signal to all ports if it detects a collision.

**3. Switch:** A network switch or switching hub is a computer networking device that connects network segments. The term commonly refers to a network bridge that processes and routes data at the data link layer (layer 2) of the OSI model. Switches that additionally process data at the network layer (layer 3 and above) are often referred to as Layer 3 switches or multilayer switches.

**4. Bridge:** A network bridge connects multiple network segments at the data link layer (Layer 2) of the OSI model. In Ethernet networks, the term bridge formally means a device that behaves according to the IEEE 802.1D standard. A bridge and switch are very much alike; a switch being a bridge with numerous ports. Switch or Layer 2 switch is often used interchangeably with bridge .Bridges can analyze incoming data packets to determine if the bridge is able to send the given packet to another segment of the network.

**5. Router:** A router is an electronic device that interconnects two or more computer networks, and selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network, or if the data packet must be transferred from one network to another. Where multiple routers are used in a large collection of interconnected networks, the routers exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.

**6. Gate Way:** In a communications network, a network node equipped for interfacing with another network that uses different protocols.

• A gateway may contain devices such as protocol translators, impedance matching devices, rate converters, fault isolators, or signal translators as necessary to provide system interoperability. It also requires the establishment of mutually acceptable administrative procedures between both networks.

• A protocol translation/mapping gateway interconnects networks with different network protocol technologies by performing the required protocol conversions.

**Practical 3: Implementation of Various Networking commands.**

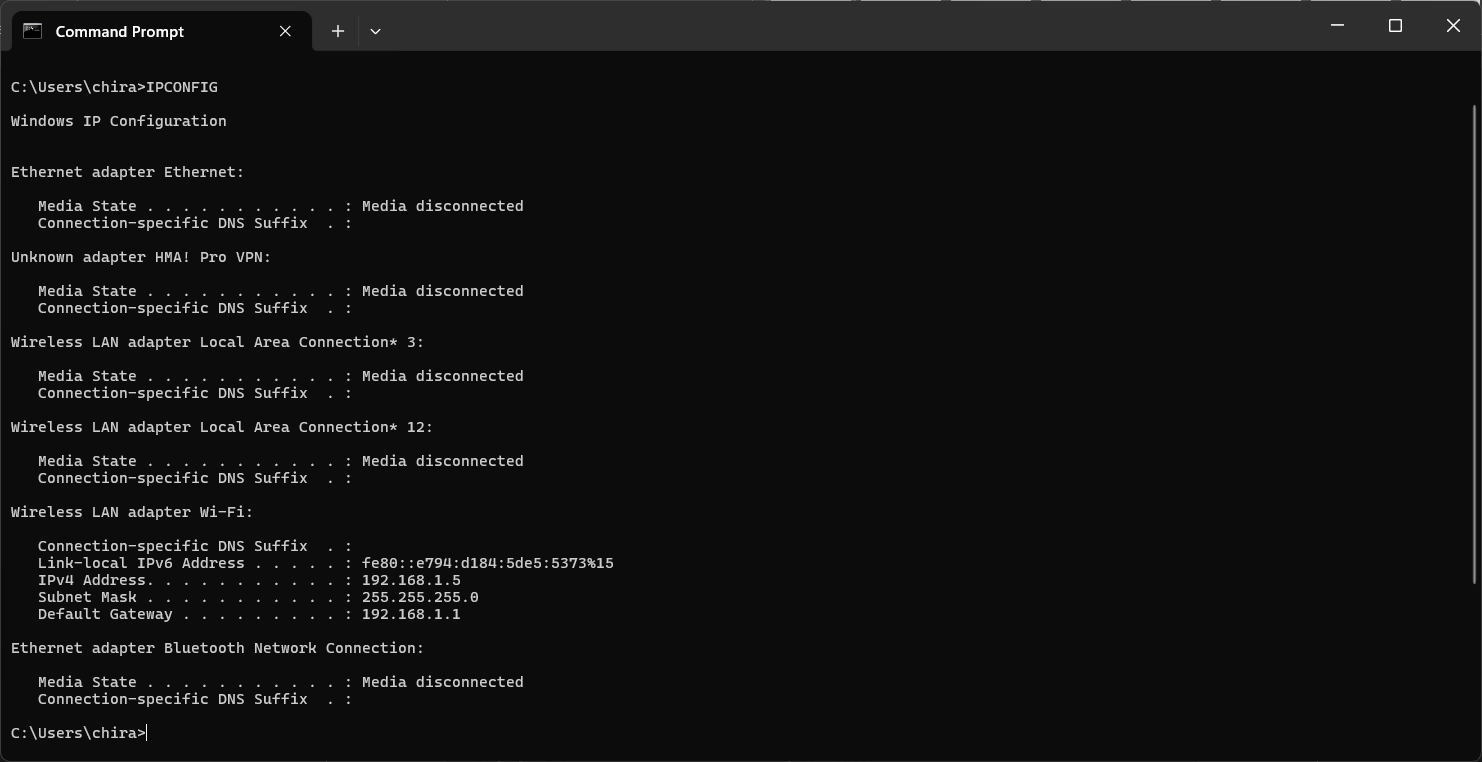
**1.HOSTNAME:**

The HostName command is used to display the host name of the computer or device in a network. When you run the HostName command in a command prompt or terminal, it will return the name assigned to the device on the local network. The host name is a unique identifier used to distinguish devices in a network and is used in various networking protocols and configurations.

**2.IPCONFIG:**

As the command name suggests, it gives information about the IP address. It not only gives the IP address of the computer it is executed on but also much more information as DNS addresses are stored in the cache. It has options to show even the computer's MAC address, renew the IP address, release the current IP address, flush the DNS cache, and help.

**Output:**

****

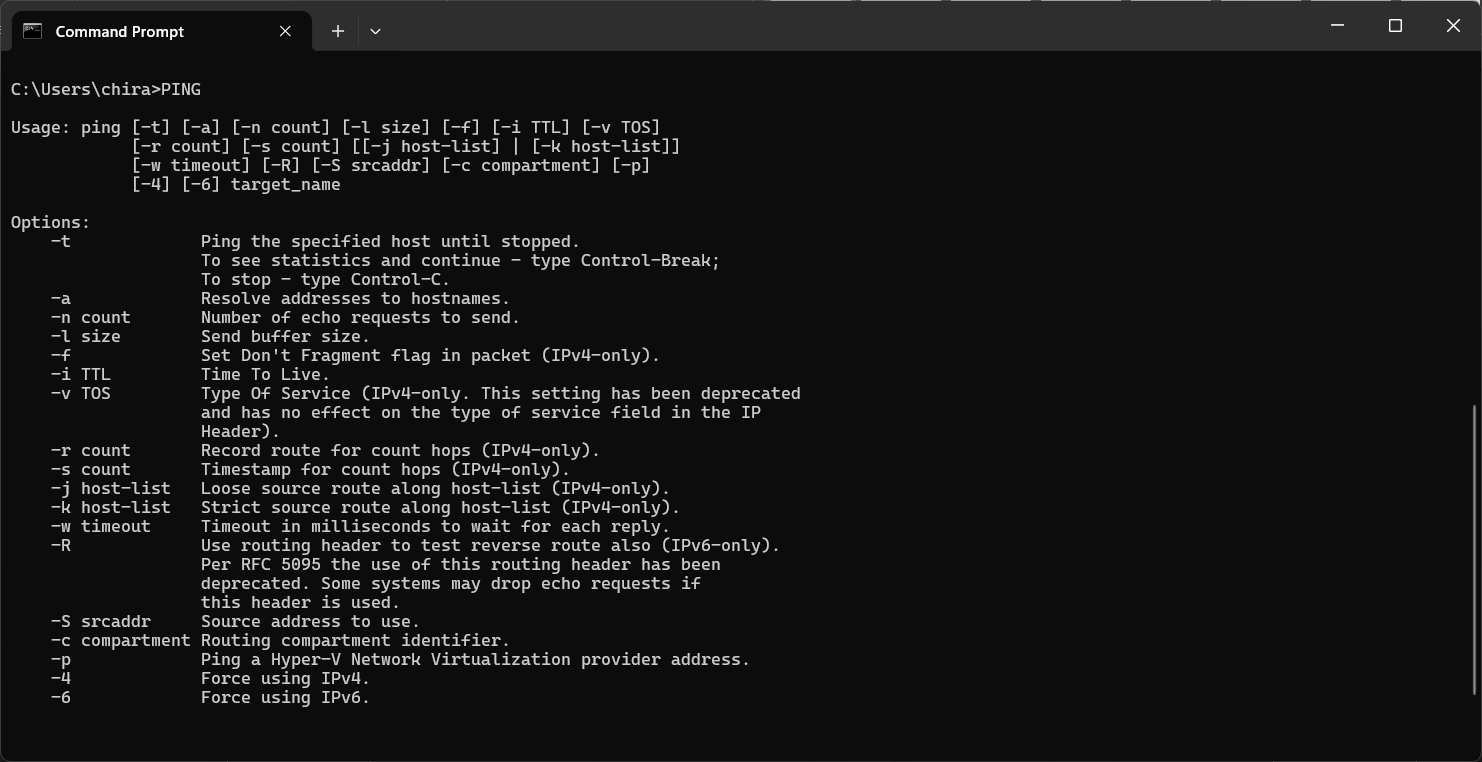
**3.NSLOOKUP:**

The NsLookup command is a command-line utility. It is used in networking to query the Domain Name System (DNS). Then it obtains information about domain names, IP addresses, and other DNS-related data. When you run the NsLookup command followed by a domain name or IP address, it will display the corresponding DNS records associated with that domain or IP. This command is commonly used to troubleshoot DNS-related issues, check DNS resolution, and gather information about domain configurations.

**4.PING:**

It is one of the basic networking commands to test the connection between the local machine and the host server. This command sends a small amount of data to the host server, and in return, the host server sends a reply to the computer. Information like the IP address of the host server, the amount of data sent, time to live, and time needed for sending and receiving the data are recorded and displayed to the user.

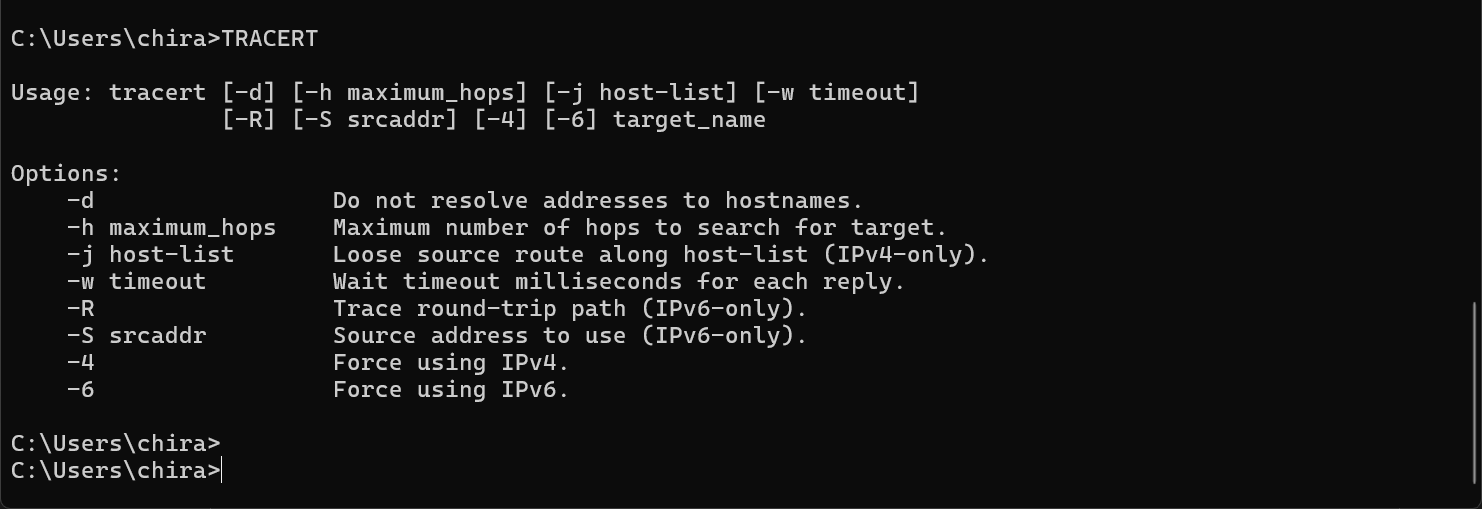
**Output:**

****

**5.TRACERT:**

The tracert command traces the route from a computer to a host server. It traces the connection for a fixed maximum number of hops. It is one of the basic networking commands. It is used to diagnose path-related problems. The information it displays about the connection route includes the IP addresses for each intermediate server and 3 round trips for each server. You can set various options for this command, like whether to resolve names of intermediate server IP addresses, the maximum number of hops in the path, and the maximum amount of time to wait for a reply.

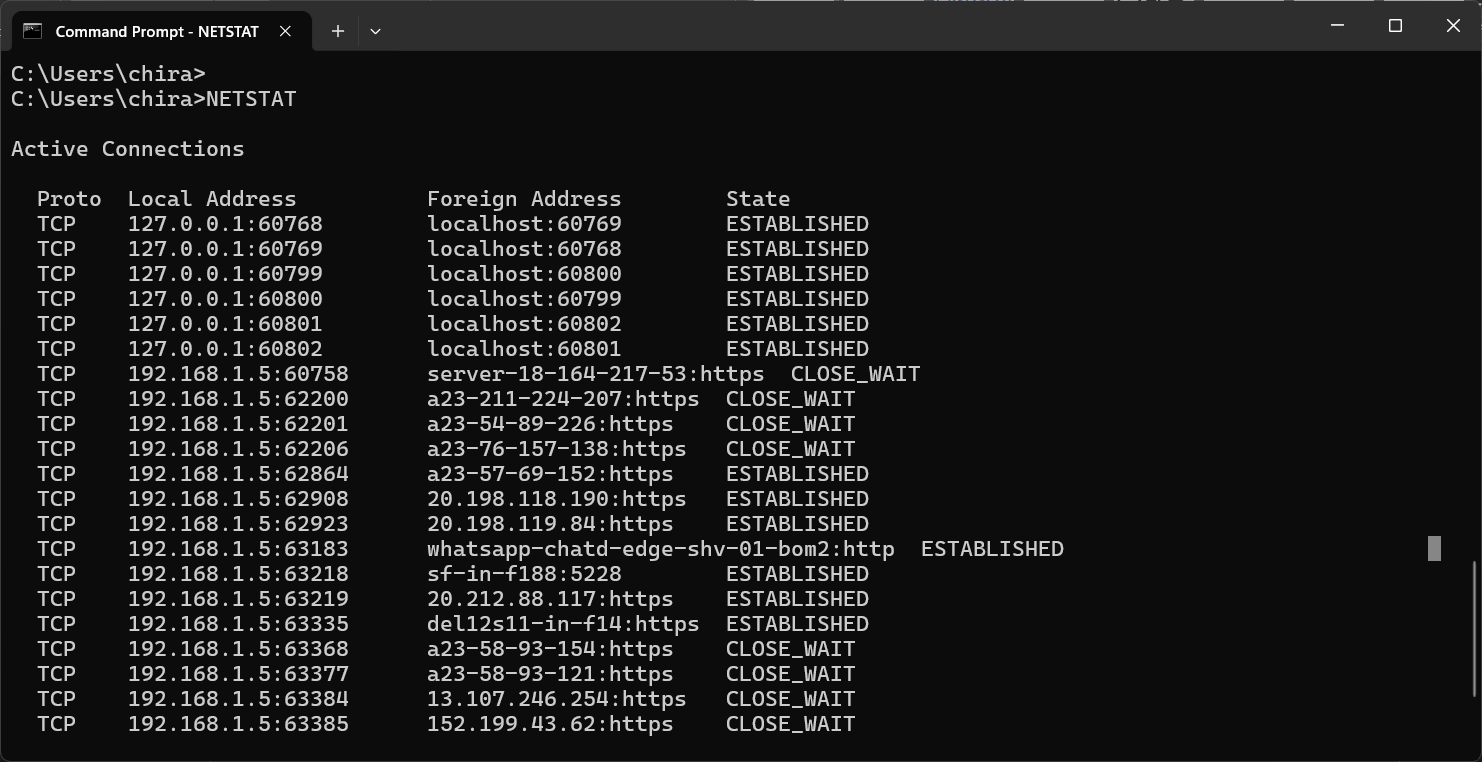
**Output:**

****

**6.NETSTAT:**

This command displays the connections active on the computer and the ports the computer is listening to. The command displays the four parameters: proto, local address, foreign address, and state. The proto column shows the type of connection, and the local address shows the IP address and the port number of the connection local machine. The proto column also indicates the foreign address, specifies the IP address and port number of the connection in the host server, and the state shows whether the connection is established or not.

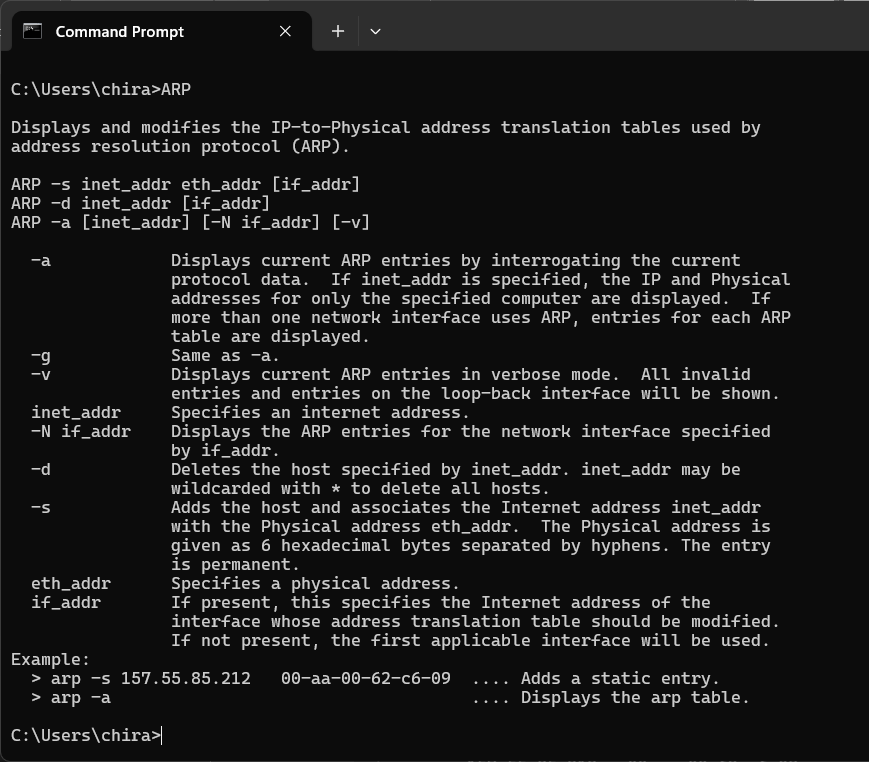
**Output:**

****

**7.ARP:**

The arp command is a short form for Address Resolution Protocol. This command is used to display and modify the IP to the physical address translation table used by the address resolution protocol. It has many options, of which a few are to display current ARP entries, specify an internet address, delete a host in the ARP table, specify a physical address, and many more.

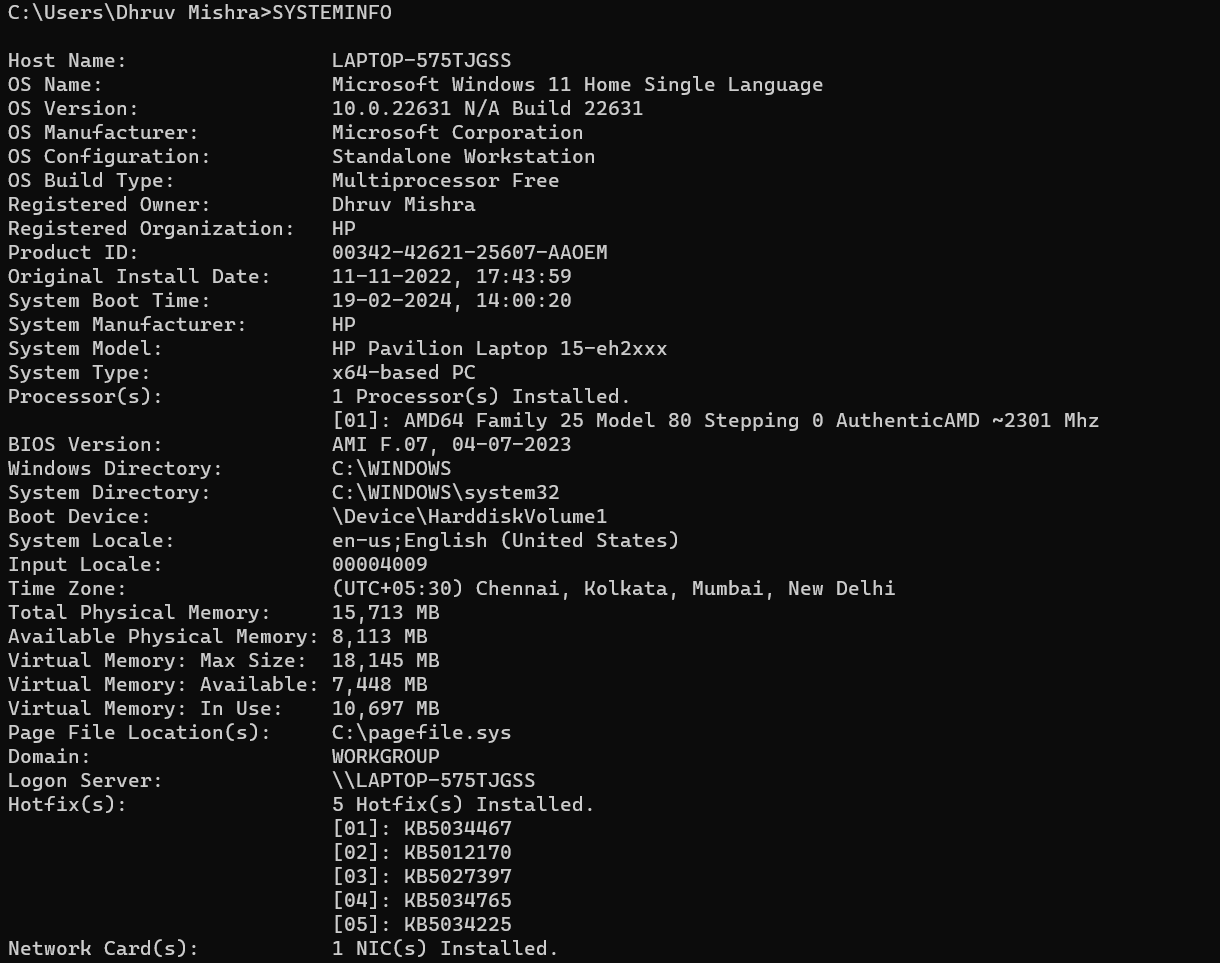
**Output:**

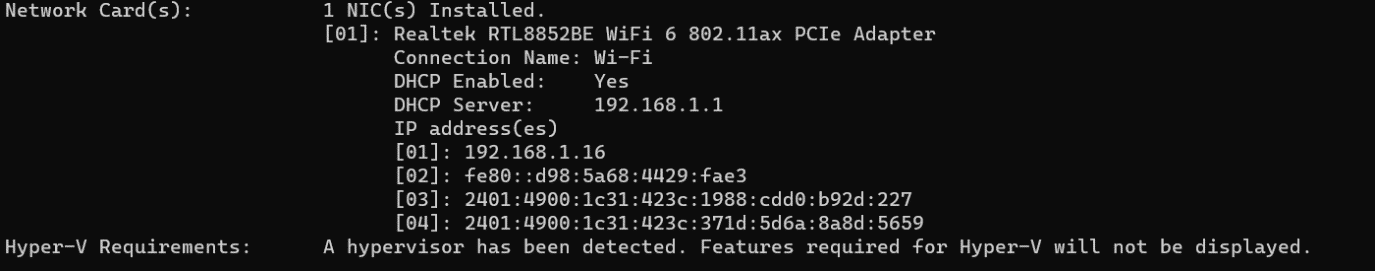
****

**8.SYSTEMINFO:**

The SystemInfo command is used to retrieve detailed information about the hardware and software configuration of a Windows-based computer. When you run the SystemInfo command in a command prompt, it will display a comprehensive report containing information. It will display the operating system version, system manufacturer, processor details, memory size, network adapter details, and more. This command is helpful for system administrators and users to gather system-related information and diagnose issues.

**Output:**

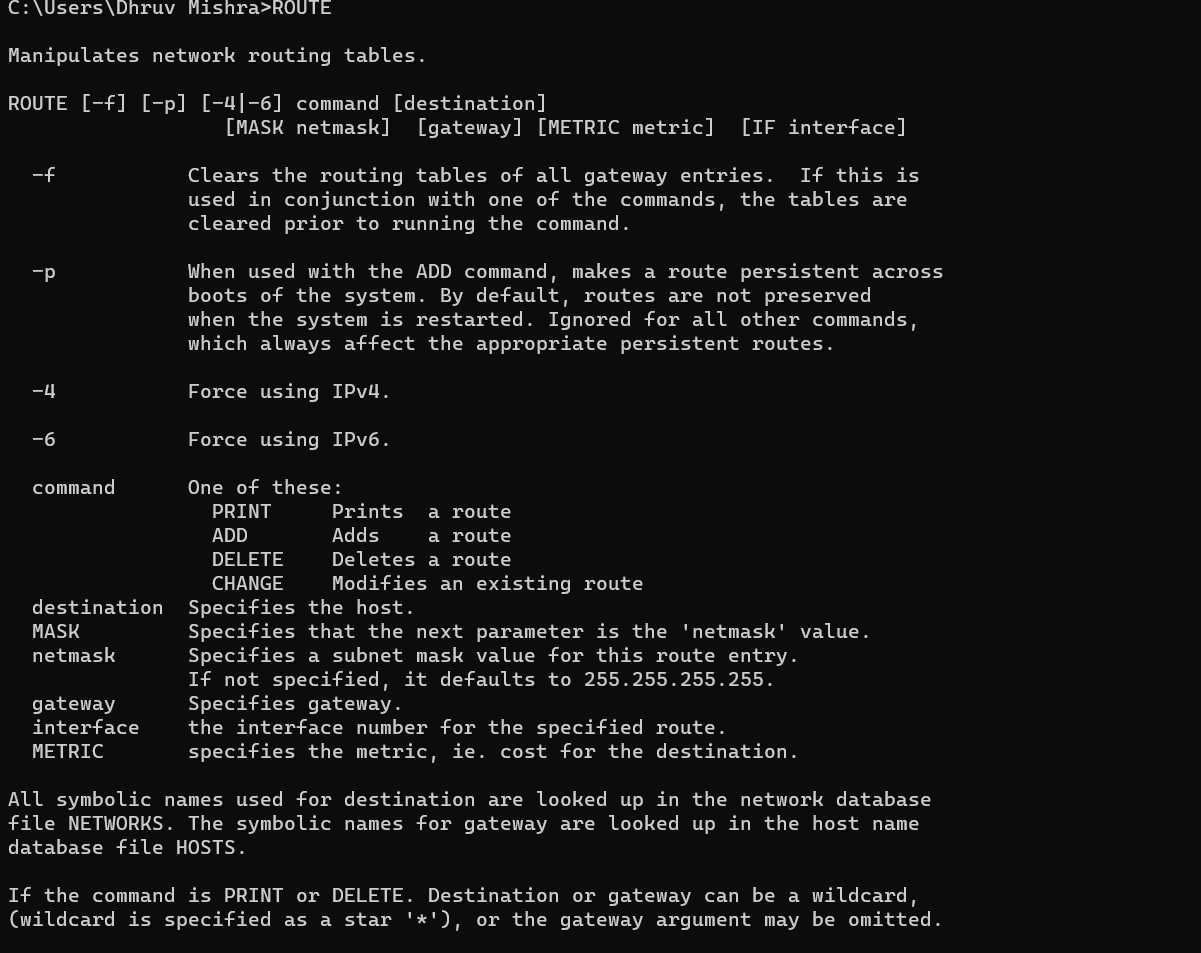
****

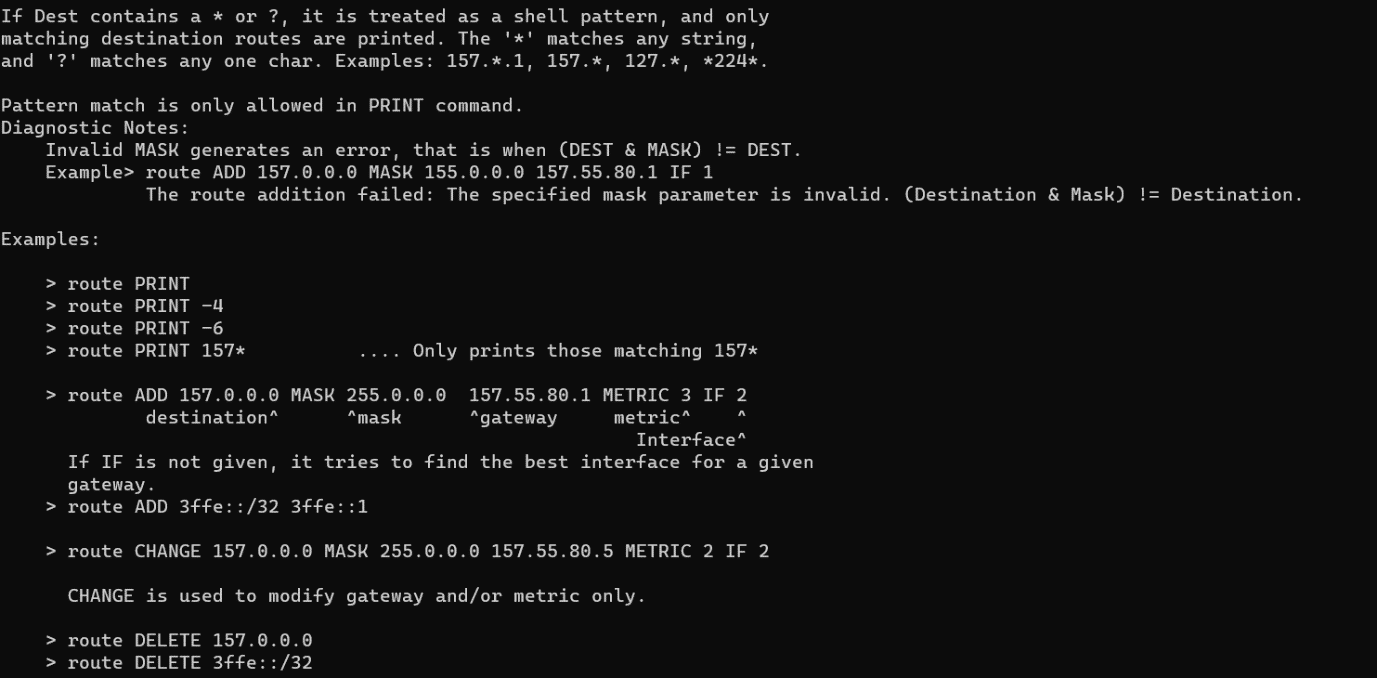
****

**9.ROUTE:**

Using the route command displays or modifies the computer's routing table.

**Output:**

****

****

**Practical 4: Write a program to implement the Cyclic Redundancy Check(CRC) Problem.**

**Code:**

#include <bits/stdc++.h>

**using** **namespace** std;

string xor1(string a, string b)

{

   string result = "";

**int** n = b.length();

**for** (**int** i = 1; i < n; i++) {

**if** (a[i] == b[i])

            result += "0";

**else**

            result += "1";

    }

**return** result;

}

string mod2div(string dividend, string divisor)

{

**int** pick = divisor.length();

    string tmp = dividend.substr(0, pick);

**int** n = dividend.length();

**while** (pick < n) {

**if** (tmp[0] == '1')

            tmp = xor1(divisor, tmp) + dividend[pick];

**else**

tmp = xor1(std::string(pick, '0'), tmp)

                  + dividend[pick];

        pick += 1;

    }

**if** (tmp[0] == '1')

        tmp = xor1(divisor, tmp);

**else**

        tmp = xor1(std::string(pick, '0'), tmp);

**return** tmp;

}

**void** encodeData(string data, string key)

{

**int** l\_key = key.length();

    string appended\_data

        = (data + std::string(l\_key - 1, '0'));

    string remainder = mod2div(appended\_data, key);

    string codeword = data + remainder;

    cout << "Remainder : " << remainder << "\n";

    cout << "Encoded Data (Data + Remainder) :" << codeword

         << "\n";

}

**void** receiver(string data, string key)

{

    string currxor

        = mod2div(data.substr(0, key.size()), key);

**int** curr = key.size();

**while** (curr != data.size()) {

**if** (currxor.size() != key.size()) {

            currxor.push\_back(data[curr++]);

        }

**else** {

            currxor = mod2div(currxor, key);

        }

    }

**if** (currxor.size() == key.size()) {

        currxor = mod2div(currxor, key);

    }

**if** (currxor.find('1') != string::npos) {

        cout << "there is some error in data" << endl;

    }

**else** {

        cout << "correct message received" << endl;

    }

}

**int** main()

{

    string data = "100100";

    string key = "1101";

    cout << "Sender side..." << endl;

    encodeData(data, key);

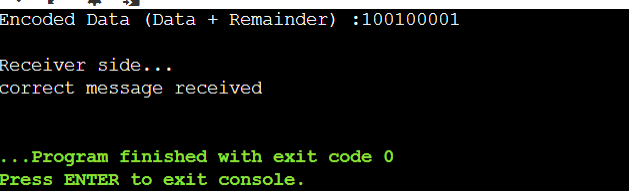
    cout << "\nReceiver side..." << endl;

    receiver(data+mod2div(data+std::string(key.size() - 1, '0'),key), key);

**return** 0;

}

**Output:**

****

**Practical 5: Write a program to implement the CHECKSUM Problem.**

**Code:**

#include <bits/stdc++.h>

**using** **namespace** std;

string Ones\_complement(string data)

{

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

**if** (data[i] == '0')

            data[i] = '1';

**else**

            data[i] = '0';

    }

**return** data;

}

string checkSum(string data, **int** block\_size)

{

**int** n = data.length();

**if** (n % block\_size != 0) {

**int** pad\_size = block\_size - (n % block\_size);

**for** (**int** i = 0; i < pad\_size; i++) {

            data = '0' + data;

        }

    }

    string result = "";

**for** (**int** i = 0; i < block\_size; i++) {

        result += data[i];

    }

**for** (**int** i = block\_size; i < n; i += block\_size) {

        string next\_block = "";

**for** (**int** j = i; j < i + block\_size; j++) {

            next\_block += data[j];

        }

        string additions = "";

**int** sum = 0, carry = 0;

**for** (**int** k = block\_size - 1; k >= 0; k--) {

            sum += (next\_block[k] - '0')

                   + (result[k] - '0');

            carry = sum / 2;

**if** (sum == 0) {

                additions = '0' + additions;

                sum = carry;

            }

**else** **if** (sum == 1) {

                additions = '1' + additions;

                sum = carry;

            }

**else** **if** (sum == 2) {

                additions = '0' + additions;

                sum = carry;

            }

**else** {

                additions = '1' + additions;

                sum = carry;

            }

        }

        string final = "";

**if** (carry == 1) {

**for** (**int** l = additions.length() - 1; l >= 0;

                 l--) {

**if** (carry == 0) {

                    final = additions[l] + final;

                }

**else** **if** (((additions[l] - '0') + carry) % 2

                         == 0) {

                    final = "0" + final;

                    carry = 1;

                }

**else** {

                    final = "1" + final;

                    carry = 0;

                }

            }

            result = final;

        }

**else** {

            result = additions;

        }

    }

**return** Ones\_complement(result);

}

**bool** checker(string sent\_message,

             string rec\_message,

**int** block\_size)

{

    string sender\_checksum

        = checkSum(sent\_message, block\_size);

    string receiver\_checksum = checkSum(

        rec\_message + sender\_checksum, block\_size);

**if** (count(receiver\_checksum.begin(),

              receiver\_checksum.end(), '0')

        == block\_size) {

**return** **true**;

    }

**else** {

**return** **false**;

    }

}

**int** main()

{

    string sent\_message

        = "10000101011000111001010011101101";

    string recv\_message

        = "10000101011000111001010011101101";

**int** block\_size = 8;

**if** (checker(sent\_message,

                recv\_message,

                block\_size)) {

        cout << "No Error";

    }

**else** {

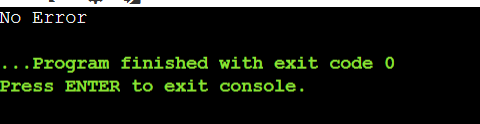
        cout << "Error";

    }

**return** 0;

}

**Output:**

****

**Practical 6: Write a program to implement Simplest Protocol.**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

FILE \*fp;

void openFile() {

fp = fopen("frames.txt", "w+");

if (fp == NULL) {

printf("Could not open file frames.txt");

\_exit(1);

}

}

void closeFile() {

fclose(fp);

}

void sendFrame() {

fprintf(fp, "Sender: Sending frame\n");

}

void receiveFrame() {

char frame[255];

if (fgets(frame, 255, fp) != NULL) {

printf("Receiver: Receiving frame\n");

printf("%s", frame);

}

}

void getData() {

printf("Sender: Getting data\n");

}

void makeFrame() {

printf("Sender: Making frame\n");

}

void extractData() {

printf("Receiver: Extracting data\n");

}

void deliverData() {

printf("Receiver: Delivering data to network layer\n");

}

bool waitForEvent() {

sleep(1);

return true;

}

int main() {

bool isSender = true;

openFile();

while (true) {

if (waitForEvent()) {

if (isSender) {

getData();

makeFrame();

sendFrame();

} else {

receiveFrame();

extractData();

deliverData();

}

isSender = !isSender;

}

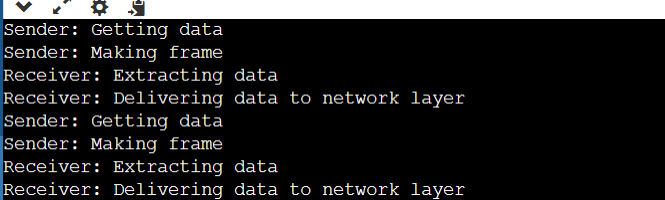
}

closeFile();

return 0;

}

**Output:**

****

**Practical 7: Write a program to implement Stop and Wait Protocol.**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

FILE \*fp;

void openFile() {

fp = fopen("frames.txt", "w+");

if (fp == NULL) {

printf("Could not open file frames.txt");

\_exit(1);

}

}

void closeFile() {

fclose(fp);

}

void sendFrame() {

fprintf(fp, "Sender: Sending frame\n");

printf("Sender: Sending frame\n");

}

void receiveFrame() {

char frame[255];

if (fgets(frame, 255, fp) != NULL) {

printf("Receiver: Receiving frame\n");

printf("%s", frame);

}

}

void sendAck() {

fprintf(fp, "Receiver: Sending Acknowledgement\n");

printf("Receiver: Sending Acknowledgement\n");

}

void receiveAck() {

char ack[255];

if (fgets(ack, 255, fp) != NULL) {

printf("Sender: Receiving Acknowledgement\n");

printf("%s", ack);

}

}

void getData() {

printf("Sender: Getting data\n");

}

void makeFrame() {

printf("Sender: Making frame\n");

}

void extractData() {

printf("Receiver: Extracting data\n");

}

void deliverData() {

printf("Receiver: Delivering data to network layer\n");

}

bool waitForEvent() {

sleep(1);

return true;

}

int main() {

bool isSender = true;

openFile();

while (true) {

if (waitForEvent()) {

if (isSender) {

getData();

makeFrame();

sendFrame();

receiveAck();

} else {

receiveFrame();

extractData();

deliverData();

sendAck();

}

isSender = !isSender;

}

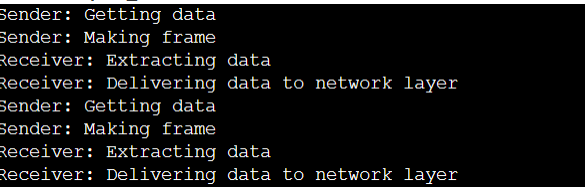
}

closeFile();

return 0;

}

**Output:**

****

**Practical 8: Write a program to implement Stop and Wait ARQ Protocol.**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

#include <signal.h>

#include <setjmp.h>

FILE \*fp;

bool ackReceived = false;

jmp\_buf env;

void handle\_alarm(int sig) {

if (!ackReceived) {

longjmp(env, 1);

}

}

void openFile() {

fp = fopen("frames.txt", "w+");

if (fp == NULL) {

printf("Could not open file frames.txt");

\_exit(1);

}

}

void closeFile() {

fclose(fp);

}

void sendFrame() {

fprintf(fp, "Sender: Sending frame\n");

printf("Sender: Sending frame\n");

}

void receiveFrame() {

char frame[255];

if (fgets(frame, 255, fp) != NULL) {

printf("Receiver: Receiving frame\n");

printf("%s", frame);

}

}

void sendAck() {

fprintf(fp, "Receiver: Sending Acknowledgement\n");

printf("Receiver: Sending Acknowledgement\n");

}

void receiveAck() {

char ack[255];

if (fgets(ack, 255, fp) != NULL) {

printf("Sender: Receiving Acknowledgement\n");

printf("%s", ack);

ackReceived = true;

}

}

void getData() {

printf("Sender: Getting data\n");

}

void makeFrame() {

printf("Sender: Making frame\n");

}

void extractData() {

printf("Receiver: Extracting data\n");

}

void deliverData() {

printf("Receiver: Delivering data to network layer\n");

}

bool waitForEvent() {

sleep(1);

return true;

}

int main() {

bool isSender = true;

openFile();

signal(SIGALRM, handle\_alarm);

while (true) {

if (waitForEvent()) {

if (isSender) {

if (setjmp(env) == 0) {

getData();

makeFrame();

sendFrame();

alarm(5); // Set a timer for 5 seconds

receiveAck();

alarm(0); // Cancel the timer

} else {

printf("Timeout: Resending frame\n");

sendFrame();

alarm(5); // Set a timer for 5 seconds

receiveAck();

alarm(0); // Cancel the timer

}

} else {

receiveFrame();

extractData();

deliverData();

sendAck();

isSender = !isSender; // Only switch to sender after receiving Ack

}

}

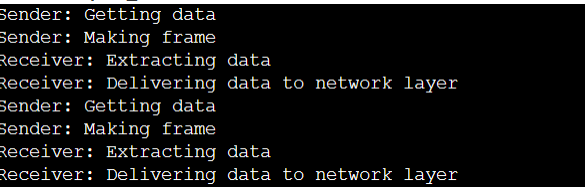
}

closeFile();

return 0;

}

**Output:**

****

**Practical 9: Write a program to implement Go-Back-N Protocol.**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

#include <signal.h>

#include <setjmp.h>

FILE \*fp;

bool ackReceived = false;

jmp\_buf env;

void handle\_alarm(int sig) {

if (!ackReceived) {

longjmp(env, 1);

}

}

void openFile() {

fp = fopen("frames.txt", "w+");

if (fp == NULL) {

printf("Could not open file frames.txt");

exit(1);

}

}

void closeFile() {

fclose(fp);

}

void sendFrame() {

fprintf(fp, "Sender: Sending frame\n");

printf("Sender: Sending frame\n");

}

void receiveFrame() {

char frame[255];

if (fgets(frame, 255, fp) != NULL) {

printf("Receiver: Receiving frame\n");

printf("%s", frame);

}

}

void sendAck() {

fprintf(fp, "Receiver: Sending Acknowledgement\n");

printf("Receiver: Sending Acknowledgement\n");

}

void receiveAck() {

char ack[255];

if (fgets(ack, 255, fp) != NULL) {

printf("Sender: Receiving Acknowledgement\n");

printf("%s", ack);

ackReceived = true;

}

}

void getData() {

printf("Sender: Getting data\n");

}

void makeFrame() {

printf("Sender: Making frame\n");

}

void extractData() {

printf("Receiver: Extracting data\n");

}

void deliverData() {

printf("Receiver: Delivering data to network layer\n");

}

bool waitForEvent() {

sleep(1);

return true;

}

int main() {

bool isSender = true;

openFile();

signal(SIGALRM, handle\_alarm);

while (true) {

if (waitForEvent()) {

if (isSender) {

if (setjmp(env) == 0) {

getData();

makeFrame();

sendFrame();

alarm(5); // Set a timer for 5 seconds

receiveAck();

alarm(0); // Cancel the timer

} else {

printf("Timeout: Resending frame\n");

sendFrame();

alarm(5); // Set a timer for 5 seconds

receiveAck();

alarm(0); // Cancel the timer

}

} else {

receiveFrame();

extractData();

deliverData();

sendAck();

isSender = !isSender; // Only switch to sender after receiving Ack

}

}

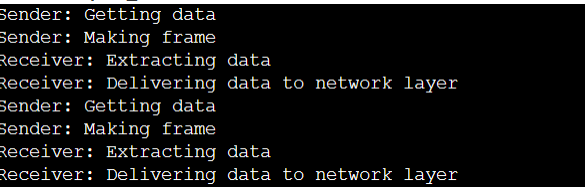
}

closeFile();

    return 0;

}

**Output:**

****

**Practical 10: Write a program to implement Selective Repeat Protocol.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int windowsize = 4;

int noofPacket, morePacket;

int receiver(int tmp1) {

srand(time(0));

return rand() % tmp1;

}

int negack(int tmp1) {

srand(time(0));

return rand() % tmp1;

}

int simulate(int windowsize) {

srand(time(0));

int tmp1 = rand();

if(tmp1 == 0)

tmp1 = simulate(windowsize);

int i = tmp1 % windowsize;

if(i == 0)

return windowsize;

else

return tmp1 % windowsize;

}

int main() {

srand(time(0));

noofPacket = rand() % 10;

printf("Number of frames is: %d\n", noofPacket);

morePacket = noofPacket;

int tmp1, tmp2, tmp3 = 0, tmp4 = 0, tmp5 = 0;

while(morePacket >= 0) {

tmp1 = simulate(windowsize);

windowsize -= tmp1;

tmp4 += tmp1;

if(tmp4 > noofPacket)

tmp4 = noofPacket;

for(int i = noofPacket - morePacket; i <= tmp4; i++)

printf("\nSending Frame %d", i);

tmp2 = receiver(tmp1);

tmp3 += tmp2;

if(tmp3 > noofPacket)

tmp3 = noofPacket;

tmp2 = negack(tmp1);

tmp5 += tmp2;

if(tmp5 != 0) {

printf("\nNo acknowledgement for the frame %d", tmp5);

printf("\nRetransmitting frame %d", tmp5);

}

morePacket -= tmp1;

if(windowsize <= 0)

windowsize = 4;

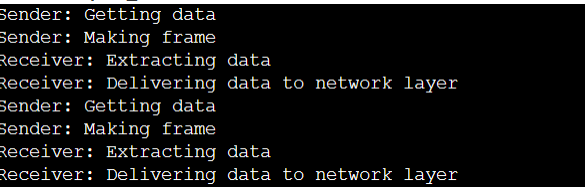
}

printf("\nSelective Repeat Protocol Ends. All packets are successfully transmitted.\n");

return 0;

}

**Output:**

****