LAB MANUAL

Lab Name : Network Programming Lab

Lab Code : 4CS4-23

Branch : Computer Science & Engineering

Year : 2nd Year

Jaipur Engineering College and Research Center, Jaipur

Department of Computer Science & Engineering

(Rajasthan Technical University, KOTA)

INDEX

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTER

Department of Computer Science & Engineering

Branch: Computer Science & Engineering
Semester: 4th

Course Name: NP Lab **Code: 4CS2-23**

External Marks: 30 **Practical hrs:3 hr/week**

Internal Marks: 45 **Total Marks: 75**

1. VISION & MISSION

1 Vision of the Department

To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

2 Mission of the Department

M1- To impart outcome based education for emerging technologies in the field of computer science and engineering.

M2 - To provide opportunities for interaction between academia and industry.

M3 - To provide platform for lifelong learning by accepting the change in technologies

M4 - To develop aptitude of fulfilling social responsibilities.

2. PEO

1. To provide students with the fundamentals of Engineering Sciences with more emphasis in Computer Science &Engineering by way of analyzing and exploiting engineering challenges.
2. To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.
3. To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues.
4. To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self motivated life-long learning needed for a successful professional career.
5. To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge

3. PROGRAM OUTCOMES

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems in IT.
2. **Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences in IT.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations using IT.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions using IT.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations in IT.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice using IT.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development in IT.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice using IT.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings in IT.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project Management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage IT projects and in multidisciplinary environments.
12. **Life –long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological changes needed in IT.

4. COURSE OUTCOMES (COs)

Graduates would be able:

CO1. Understand different protocols for Networking Programming Algorithms.
CO2. Implement socket programming and analyze different (client/server) models.

5. MAPPING OF CO & PO

Semester	Sub	code	L/T/P	CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
IV	Network Programming Lab	4CS4-23	P	Understand different protocols for Networking Programming Algorithms.	H	H	H	H	H	H	M	L	M	L	L	H
			P	Implement socket programming and analyze different (client/server) models.	H	H	H	H	M	L	M	L	L	L	M	H

6. Syllabus

RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

II Year-IV Semester: B.Tech. Computer Science and Engineering

4CS4-23: Network Programming Lab

Credit: 1.5 **Max. Marks: 75(IA:45, ETE:30)**
0L+0T+3P

Objectives: At the end of the semester, the students should have clearly understood and implemented the following:

List of Experiments:

1. Study of Different Type of LAN& Network Equipments.
2. Study and Verification of standard Network topologies i.e. Star, Bus, Ring etc.
3. LAN installations and Configurations.
4. Write a program to implement various types of error correcting techniques.
5. Write a program to implement various types of framing methods.
6. Write two programs in C: hello_client and hello_server
    a. The server listens for, and accepts, a single TCP connection; it reads all the data it can from that connection, and prints it to the screen; then it closes the connection
    b. The client connects to the server, sends the string "Hello, world!", then closes the connection
7. Write an Echo_Client and Echo_server using TCP to estimate the round trip time from client to the server. The server should be such that it can accept multiple connections at any given time.
8. Repeat Exercises 6 & 7 for UDP.
9. Repeat Exercise 7 with multiplexed I/O operations.
10. Simulate Bellman-Ford Routing algorithm in NS2.

Outcomes:
At the end of the semester, the students should have clearly understood and implemented the following:

- Perform the programming by writing programs in scocket programming
- Performed & implemented Network programming

7. Instructional Methods

7.1. Direct Instructions:

- White board presentation

7.2. Interactive Instruction:

- coding

7.3. Indirect Instructions:

- Problem solving

8. LEARNING MATERIALS

8.1. Text/Lab Manual

INSTRUCTIONS OF LAB

DO's

1. Please switch off the Mobile/Cell phone before entering Lab.
2. Enter the Lab with complete source code and data.
3. Check whether all peripheral are available at your desktop before proceeding for program.
4. Intimate the lab In charge whenever you are incompatible in using the system or in case software get corrupted/ infected by virus.
5. Arrange all the peripheral and seats before leaving the lab.
6. Properly shutdown the system before leaving the lab.
7. Keep the bag outside in the racks.
8. Enter the lab on time and leave at proper time.
9. Maintain the decorum of the lab.
10. Utilize lab hours in the corresponding experiment.
11. Get your Cd / Pendrive checked by lab In charge before using it in the lab.

DON'TS

1. No one is allowed to bring storage devices like Pan Drive /Floppy etc. in the lab.
2. Don't mishandle the system.
3. Don't leave the system on standing for long
4. Don't bring any external material in the lab.
5. Don't make noise in the lab.
6. Don't bring the mobile in the lab. If extremely necessary then keep ringers off.
7. Don't enter in the lab without permission of lab Incharge.
8. Don't litter in the lab.
9. Don't delete or make any modification in system files.
10. Don't carry any lab equipments outside the lab.

We need your full support and cooperation for smooth functioning of the

INSTRUCTIONS FOR STUDENT

BEFORE ENTERING IN THE LAB

- All the students are supposed to prepare the theory regarding the next program.
- Students are supposed to bring the practical file and the lab copy.
- Assignment given in previous labs should be written in the practical file.
- Print out of diagram should be pasted in the lab file.
- Any student not following these instructions will be denied entry in the lab.

WHILE WORKING IN THE LAB

- Adhere to experimental schedule as instructed by the lab incharge.
- Get the previously executed program signed by the instructor.
- Get the output of the current program checked by the instructor in the lab copy.
- Each student should work on his/her assigned computer at each turn of the lab.
- Take responsibility of valuable accessories.
- Concentrate on the assigned practical and do not play games.
- If anyone caught red handed carrying any equipment of the lab, then he will have to face serious consequences.

Objective 1: Study of Different Type of LAN& Network Equipments

A **local area network**, or **LAN**, consists of a computer network at a single site, typically an individual office building. A LAN is very useful for sharing resources, such as data storage and printers. LANs can be built with relatively inexpensive hardware, such as hubs, network adapters and Ethernet cables.

The smallest LAN may only use two computers, while larger LANs can accommodate thousands of computers. A LAN typically relies mostly on wired connections for increased speed and security, but wireless connections can also be part of a LAN. High speed and relatively low cost are the defining characteristics of LANs.

LANs are typically used for single sites where people need to share resources among themselves but not with the rest of the outside world. Think of an office building where everybody should be able to access files on a central server or be able to print a document to one or more central printers. Those tasks should be easy for everybody working in the same office, but you would not want somebody just walking outside to be able to send a document to the printer from their cell phone! If a local area network, or LAN, is entirely wireless, it is referred to as a wireless local area network, or WLAN.

Types of Local-Area Networks (LANs)

There are many different types of LANs, with Ethernets being the most common for PCs. Most Apple Macintosh networks are based on Apple's AppleTalk network system, which is built into Macintosh computers. The following characteristics differentiate one LAN from another:

- **Topology:** The geometric arrangement of devices on the network. For example, devices can be arranged in a ring or in a straight line.
- **Protocols:** The rules and encoding specifications for sending data. The protocols also determine whether the network uses a peer-to-peer or client/server architecture.
- **Media:** Devices can be connected by twisted-pair wire, coaxial cables, or fiber optic cables. Some networks do without connecting media altogether, communicating instead via radio waves.

Deploying a Wireless LAN

Wireless networks are relatively easy to implement these days, especially when compared to the prospect of having to route wires when deploying a new wired network or overhauling an existing one. The first step in planning a wireless LAN deployment should be to decide on your wireless networking technology standard. Keep in mind that the standard you need to accommodate your network access points and routers as well as the entire collection of wireless network interface cards (NICs) for your computers and other network resources.

Different networking devices:

Network Hub:

Network Hub is a networking device which is used to connect multiple network hosts. A network hub is also used to do data transfer. The data is transferred in terms of packets on a computer network. So when a host sends a data packet to a network hub, the hub copies the data packet to all of its ports connected to. Like this, all the ports know about the data and the port for whom the packet is intended, claims the packet.

However, because of its working mechanism, a hub is not so secure and safe. Moreover, copying the data packets on all the interfaces or ports makes it slower and more congested which led to the use of network switch.

2.1.1 Network Switch:

Like a hub, a switch also works at the layer of LAN (Local Area Network) but you can say that a switch is more intelligent than a hub. While hub just does the work of data forwarding, a switch does 'filter and forwarding' which is a more intelligent way of dealing with the data packets.

So, when a packet is received at one of the interfaces of the switch, it filters the packet and sends only to the interface of the intended receiver. For this purpose, a switch also maintains a CAM (Content Addressable Memory) table and has its own system configuration and memory. CAM table is also called as forwarding table or forwarding information base (FIB).

2.1.2 Modem:

A Modem is somewhat a more interesting network device in our daily life. So if you have noticed around, you get an internet connection through a wire (there are different types of wires) to your house. This wire is used to carry our internet data outside to the internet world.

However, our computer generates binary data or digital data in forms of 1s and 0s and on the other hand, a wire carries an analog signal and that's where a modem comes in.

A modem stands for (**Mo**dulator+**De**modulator). That means it modulates and demodulates the signal between the digital data of a computer and the analog signal of a telephone line.

2.1.3 Network Router:

A router is a network device which is responsible for routing traffic from one to another network. These two networks could be a private company network to a public network. You can think of a router as a traffic police who directs different network traffic to different directions.

2.1.4 Bridge:

If a router connects two different types of networks, then a bridge connects two subnetworks as a part of the same network. You can think of two different labs or two different floors connected by a bridge.

2.1.5 Repeater:

A repeater is an electronic device that amplifies the signal it receives. In other terms, you can think of repeater as a device which receives a signal and retransmits it at a higher level or higher power so that the signal can cover longer distances.

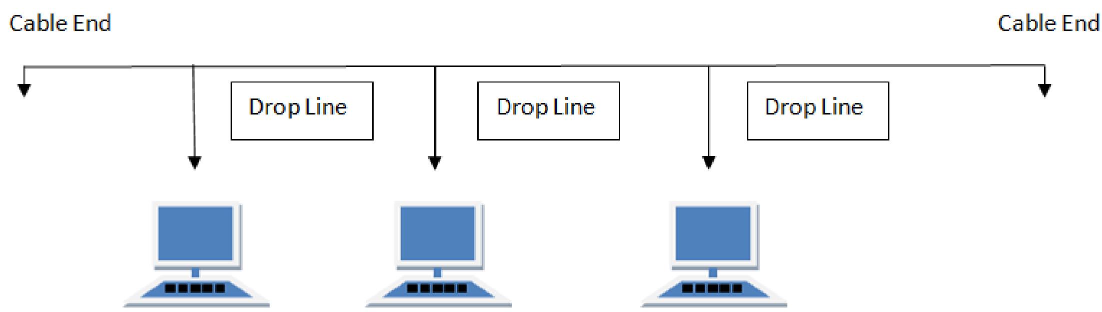
For example, inside a college campus, the hostels might be far away from the main college where the ISP line comes in. If the college authority wants to pull a wire in between the hostels and main campus, they will have to use repeaters if the distance is much because different types of cables have limitations in terms of the distances they can carry the data for.

Objective2: Study and Verification of standard Network topologies i.e. Star, Bus, Ring etc.

A **network topology** is the arrangement of a **network**, including its nodes and connecting lines. There are two ways of defining **network** geometry: the physical **topology** and the logical (or signal)**topology**.

2.2 BUS Topology

Bus topology is a network type in which every computer and network device is connected to single cable. When it has exactly two endpoints, then it is called **Linear Bus topology**.



2.2.1.1 Features of Bus Topology

1. It transmits data only in one direction.
2. Every device is connected to a single cable

2.2.1.2 Advantages of Bus Topology

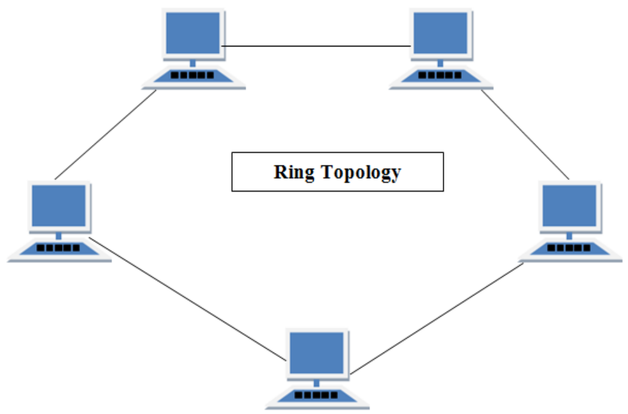
1. It is cost effective.
2. Cable required is least compared to other network topology.
3. Used in small networks.
4. It is easy to understand.
5. Easy to expand joining two cables together.

2.2.1.3 Disadvantages of Bus Topology

1. Cables fails then whole network fails.
2. If network traffic is heavy or nodes are more the performance of the network decreases.
3. Cable has a limited length.
4. It is slower than the ring topology.

2.3 RING Topology

It is called ring topology because it forms a ring as each computer is connected to another computer, with the last one connected to the first. Exactly two neighbours for each device.



2.3.1.1 Features of Ring Topology

1. A number of repeaters are used for Ring topology with large number of nodes, because if someone wants to send some data to the last node in the ring topology with 100 nodes, then the data will have to pass through 99 nodes to reach the 100th node. Hence to prevent data loss repeaters are used in the network.
2. The transmission is unidirectional, but it can be made bidirectional by having 2 connections between each Network Node, it is called **Dual Ring Topology**.
3. In Dual Ring Topology, two ring networks are formed, and data flow is in opposite direction in them. Also, if one ring fails, the second ring can act as a backup, to keep the network up.
4. Data is transferred in a sequential manner that is bit by bit. Data transmitted, has to pass through each node of the network, till the destination node.

2.3.1.2 Advantages of Ring Topology

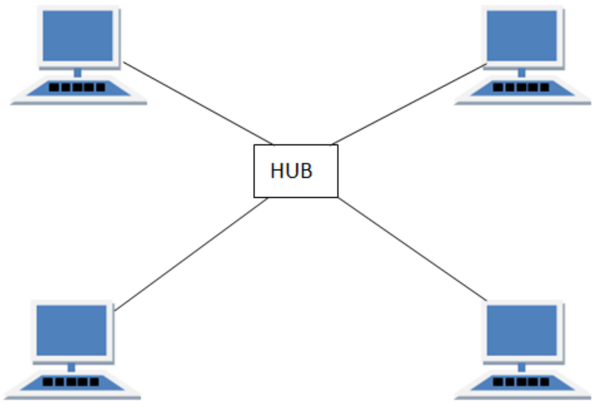
1. Transmitting network is not affected by high traffic or by adding more nodes, as only the nodes having tokens can transmit data.
2. Cheap to install and expand

2.3.1.3 Disadvantages of Ring Topology

1. Troubleshooting is difficult in ring topology.
2. Adding or deleting the computers disturbs the network activity.
3. Failure of one computer disturbs the whole network.

2.4 STAR Topology

In this type of topology all the computers are connected to a single hub through a cable. This hub is the central node and all others nodes are connected to the central node.



2.4.1.1 Features of Star Topology

1. Every node has its own dedicated connection to the hub.
2. Hub acts as a repeater for data flow.
3. Can be used with twisted pair, Optical Fibre or coaxial cable.

2.4.1.2 Advantages of Star Topology

1. Fast performance with few nodes and low network traffic.
2. Hub can be upgraded easily.
3. Easy to troubleshoot.
4. Easy to setup and modify.
5. Only that node is affected which has failed, rest of the nodes can work smoothly.

2.4.1.3 Disadvantages of Star Topology

1. Cost of installation is high.
2. Expensive to use.
3. If the hub fails then the whole network is stopped because all the nodes depend on the hub.
4. Performance is based on the hub that is it depends on its capacity

2.5 MESH Topology

It is a point-to-point connection to other nodes or devices. All the network nodes are connected to each other. Mesh has `n(n-1)/2` physical channels to link `n` devices.

There are two techniques to transmit data over the Mesh topology, they are :

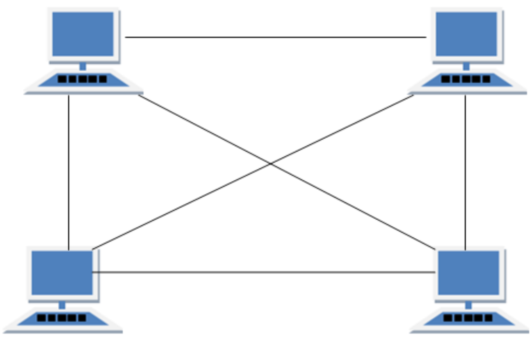
1. Routing
2. Flooding

2.5.1 MESH Topology: Routing

In routing, the nodes have a routing logic, as per the network requirements. Like routing logic to direct the data to reach the destination using the shortest distance. Or, routing logic which has information about the broken links, and it avoids those node etc. We can even have routing logic, to re-configure the failed nodes.

2.5.2 MESH Topology: Flooding

In flooding, the same data is transmitted to all the network nodes, hence no routing logic is required. The network is robust, and the its very unlikely to lose the data. But it leads to unwanted load over the network.



2.5.2.1 Types of Mesh Topology

1. **Partial Mesh Topology :** In this topology some of the systems are connected in the same fashion as mesh topology but some devices are only connected to two or three devices.
2. **Full Mesh Topology :** Each and every nodes or devices are connected to each other.

2.5.2.2 Features of Mesh Topology

1. Fully connected.
2. Robust.
3. Not flexible.

2.5.2.3 Advantages of Mesh Topology

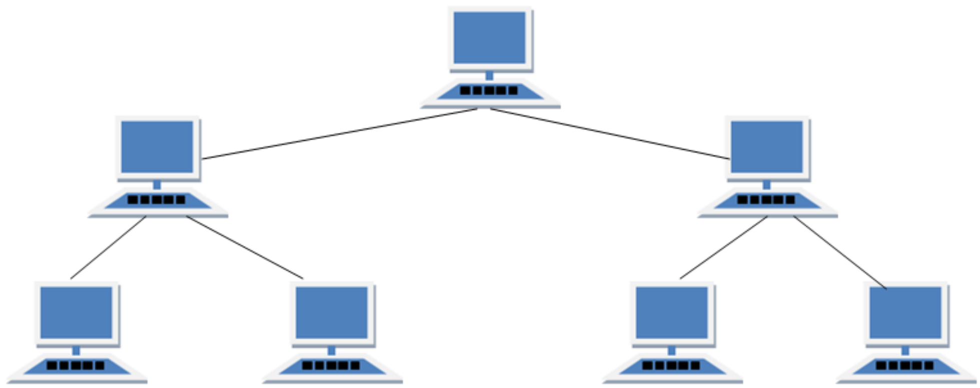
1. Each connection can carry its own data load.
2. It is robust.
3. Fault is diagnosed easily.
4. Provides security and privacy.

2.5.2.4 Disadvantages of Mesh Topology

1. Installation and configuration is difficult.
2. Cabling cost is more.
3. Bulk wiring is required.

2.6 TREE Topology

It has a root node and all other nodes are connected to it forming a hierarchy. It is also called hierarchical topology. It should at least have three levels to the hierarchy.



2.6.1.1 Features of Tree Topology

1. Ideal if workstations are located in groups.
2. Used in Wide Area Network.

2.6.1.2 Advantages of Tree Topology

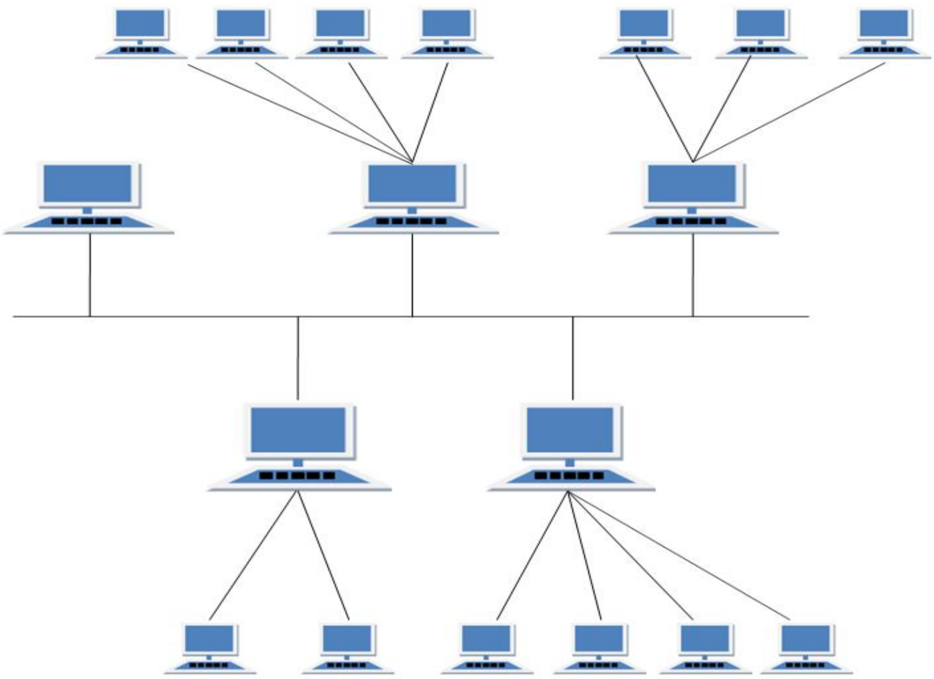
1. Extension of bus and star topologies.
2. Expansion of nodes is possible and easy.
3. Easily managed and maintained.
4. Error detection is easily done.

2.6.1.3 Disadvantages of Tree Topology

1. Heavily cabled.
2. Costly.
3. If more nodes are added maintenance is difficult.
4. Central hub fails, network fails.

2.7 HYBRID Topology

It is two different types of topologies which is a mixture of two or more topologies. For example if in an office in one department ring topology is used and in another star topology is used, connecting these topologies will result in Hybrid Topology (ring topology and star topology).



2.7.1.1 Features of Hybrid Topology

1. It is a combination of two or topologies
2. Inherits the advantages and disadvantages of the topologies included

2.7.1.2 Advantages of Hybrid Topology

1. Reliable as Error detecting and trouble shooting is easy.
2. Effective.
3. Scalable as size can be increased easily.
4. Flexible.

2.7.1.3 Disadvantages of Hybrid Topology

1. Complex in design.
2. Costly.

Objective 3: LAN installations and Configurations

1. Take the computer for which you are making server, insert the second LAN in that computer.
2. Connect your internet connection into the first LAN (inbuilt) on that computer.
3. Enter the IP address which you got from your ISP and check whether you can able to use internet on that system.
4. Now make sure that the second LAN is detected and is showing Unplugged.
5. Open properties of the first LAN (inbuilt LAN) and then go to "Advanced" option which is available on the top, then check both the boxes and say ok. and close everything.
6. Now take an Internet cable which is crimped on both the sides with same colors of wires.
7. Connect one end to the second LAN and the other end to the switch.
8. Now open your second LAN properties and go to the TCP/IP properties and there enter IP address as (192.168.0.1) or anything you wish Subnet Mask (255.255.255.0) and the gateway as (192.168.0.1).

   NOTE :- THE GATEWAY SHOULD BE SAME AS THE IP ADDRESS ONLY FOR THE SERVER.
9. Now open click on the switch and you will get a notification on your server saying that "Local Area Connection 2" is connected.
10. Now take an another Internet cable and one end of that cable should be in any one port of the Switch and the other should be in the second computer.
11. Now you will get a notification that you are connected to internet, open the LAN properties and enter the IP address as (192.168.0.2) subnet mask and gateway should be same as server.
12. You will now be able to browse Internet on that particular system now.
13. Do the same with the rest of the systems.

    NOTE :- THE IP ADDRESSES SHOULD NOT BE USED SAME FOR TWO SYSTEMS. SO BETTER GO WITH 192.168.0.1 (FOR SERVER) 192.168.0.2 (1ST CLIENT) 192.168.0.3(SECOND CLIENT) AND SO ON.......BUT THE SUBNET MASK AND GATEWAYS SHOULD BE SAME FOR ALL THE CLIENT AND SERVER SYSTEM.

Objective 4: Write a program to implement various types of error correcting techniques.

1. Hamming Code

Hamming code is a popular error detection and error correction method in data communication. Hamming code can only detect 2 bit error and correct a single bit error which means it is unable to correct burst errors if may occur while transmission of data.

Hamming code uses redundant bits (extra bits) which are calculated according to the below formula:-

$\mathbf{2^r \geq m+r+1}$

Where **r** is the number of redundant bits required and **m** is the number of data bits.

R is calculated by putting **r = 1, 2, 3 …** until the above equation becomes true.

R1 bit is appended at position $\mathbf{2^0}$

R2 bit is appended at position $\mathbf{2^1}$

R3 bit is appended at position $\mathbf{2^2}$ and so on.

These redundant bits are then added to the original data for the calculation of error at receiver's end.

At receiver's end with the help of even parity (generally) the erroneous bit position is identified and since data is in binary we take complement of the erroneous bit position to correct received data.

Respective index parity is calculated for **r1, r2, r3, r4** and so on.

Programming for Hamming Code in C

```
#include<stdio.h>

void main() {
   int data[10];
   int dataatrec[10],c,c1,c2,c3,i;
```

```
printf("Enter 4 bits of data one by one\n");
scanf("%d",&data[0]);
scanf("%d",&data[1]);
scanf("%d",&data[2]);
scanf("%d",&data[4]);

//Calculation of even parity
data[6]=data[0]^data[2]^data[4];
        data[5]=data[0]^data[1]^data[4];
        data[3]=data[0]^data[1]^data[2];

        printf("\nEncoded data is\n");
        for(i=0;i<7;i++)
    printf("%d",data[i]);

printf("\n\nEnter received data bits one by one\n");
for(i=0;i<7;i++)
    scanf("%d",&dataatrec[i]);

c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];
        c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
        c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];
        c=c3*4+c2*2+c1 ;

if(c==0) {
                printf("\nNo error while transmission of data\n");
}
        else {
                printf("\nError on position %d",c);

                printf("\nData sent : ");
```

```
        for(i=0;i<7;i++)
          printf("%d",data[i]);

                printf("\nData received : ");
        for(i=0;i<7;i++)
          printf("%d",dataatrec[i]);

                printf("\nCorrect message is\n");

                //if errorneous bit is 0 we complement it else vice versa
                if(dataatrec[7-c]==0)
                        dataatrec[7-c]=1;
        else
                        dataatrec[7-c]=0;

                for (i=0;i<7;i++) {
                        printf("%d",dataatrec[i]);
                }
          }
}
```

Output

Enter 4 bits of data one by one

1

0

1

0

Encoded data is

1010010

Enter received data bits one by one

1

0

1

0

0

1

0

Objective 5: Write a program to implement various types of framing methods.

Solution:

Security and Error detection are the most prominent features that are to be provided by any application which transfers data from one end to the other end. One of such a mechanism in tracking errors which may add up to the original data during transfer is known as Stuffing. It is of two types namely Bit Stuffing and the other Character Stuffing. Coming to the Bit Stuffing, 01111110 is appended within the original data while transfer of it. The following program describes how it is stuffed at the sender end and de-stuffed at the reciever end.

Program:

```
#include
main()
{
int a[15];
int i,j,k,n,c=0,pos=0;
clrscr();
printf("\n Enter the number of bits");
scanf("%d",&n);
printf("\n Enter the bits");
for(i=0;i<n;i++)
 scanf("%d",&a[i]);
for(i=0;i<n;i++)
```

```
 {
if(a[i]==1)
{
c++;
if(c==5)
{
pos=i+1;
c=0;
for(j=n;j>=pos;j--)
{
k=j+1;
a[k]=a[j];
}
a[pos]=0;
n=n+1;
}
}
else
c=0;
}

printf("\n DATA AFTER STUFFING \n");
printf(" 01111110 ");

for(i=0;i<n;i++)
 {
printf("%d",a[i]);
}
printf(" 01111110 ");
getch();
}
```

Output:

Objective 6: Write two programs in C: hello_client and hello_server

a. The server listens for, and accepts, a single TCP connection; it reads all the data it can from that connection, and prints it to the screen; then it closes the connection

b. The client connects to the server, sends the string "Hello, world!", then closes the connection.

Solution:

(a) The server listens for, and accepts, a single TCP connection; it reads all the data it can from that connection, and prints it to the screen; then it closes the connection

```
// TCP Server
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
```

```
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
int main(int argc, char const *argv[])
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    char *hello = "Hello from server";
    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {  perror("socket failed");
       exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,&opt,
sizeof(opt)))
    {   perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons( PORT );

    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address))<0)
    {
```

```
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen))<0)
    { perror("accept");
        exit(EXIT_FAILURE);
    }
    valread = read( new_socket , buffer, 1024);
    printf("%s\n",buffer );
    send(new_socket , hello , strlen(hello) , 0 );
    printf("Hello message sent\n");
    return 0;
}

// Tcp Client
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080

int main(int argc, char const *argv[])
{
    struct sockaddr_in address;
    int sock = 0, valread;
```

```
    struct sockaddr_in serv_addr;
    char *hello = "Hello from client";
    char buffer[1024] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    // Convert IPv4 and IPv6 addresses from text to binary form
    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }
    send(sock , hello , strlen(hello) , 0 );
    printf("Hello message sent\n");
    valread = read( sock , buffer, 1024);
    printf("%s\n",buffer );
    return 0;
}
```

Steps to compile and Run

Open one Terminal in Linux and compile and run TCP server program

1. Compile the server program

gcc tcpserver.c -o tcpser)

2. Run server using

./tcpser

Output

[root@localhost TCP]# gcc tcpserver.c -o tcpser

[root@localhost TCP]# ./tcpser

Client : Hello from client

Hello message sent.

Open one Terminal in Linux and compile and run TCP client program

1. compile tcp client program

gcc tcpclient.c -o tcpcli)

2. Run tcp client

./tcpcli)

Output

[root@localhost TCP# gcc tcpclient.c -o tcpcli

[root@localhost TCP]# ./tcpcli

Hello message sent.

(b) The client connects to the server, sends the string “Hello, world!”, then closes the connection

```
#include<stdlib.h>

#include<sys/types.h>

#include<sys/socket.h>
```

```
#include<netinet/in.h>

#include<errno.h>

main()

{

        int sock, cli;

        struct sockaddr_in server, clientDetail;

        unsigned int len;

        char mesg[] = “Hello world with Socket programming!”;

        int sentconf;

         if((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)

        {

                perror(“There are some issue in creating socket : “);

                exit(-1);

        }

         server.sin_family = AF_INET;

        server.sin_port = htons(10012);

        server.sin_addr.s_addr = INADDR_ANY;

        bzero(&server.sin_zero, 8);

         len = sizeof(struct sockaddr_in);
```

```
 if((bind(sock, (struct sockaddr *)&server, len)) == -1)
{
        perror("Error in Binding");
        exit(-1);
}
 if((listen(sock, 5)) == -1)
{
        perror("Error in Listening");
        exit(-1);
}
while(1)
{
        if((cli = accept(sock, (struct sockaddr *)&clientDetail, &len)) ==-1)
        {
                perror("Error in Accepting");
                exit(-1);
        }
         sentconf =  send(cli, mesg, strlen(mesg), 0);
```

```
		printf("Sending %d bytes to clientDetail : %c\n",
sentconf,					inet_ntoa(clientDetail.sin_addr));

		close(cli);

	}

}
```

Objective: 7 write an Echo_Client and Echo_server using TCP to estimate the round trip time from client to the server. The server should be such that it can accept multiple connections at any given time.

Solution:

Sockets Overview

The operating system includes the Berkeley Software Distribution (BSD) interprocess communication (IPC) facility known as sockets. Sockets are communication channels that enable unrelated processes to exchange data locally and across networks. A single socket is one end point of a two-way communication channel.

Sockets Overview: In the operating system, sockets have the following characteristics:

- A socket exists only as long as a process holds a descriptor referring to it.
- Sockets are referenced by file descriptors and have qualities similar to those of a character special device. Read, write, and select operations can be performed on sockets by using the appropriate subroutines.
- Sockets can be created in pairs, given names, or used to rendezvous with other sockets in a communication domain, accepting connections from these sockets or exchanging messages with them.

Sockets Background: Sockets were developed in response to the need for sophisticated interprocess facilities to meet the following goals:

- Provide access to communications networks such as the Internet.

- Enable communication between unrelated processes residing locally on a single host computer and residing remotely on multiple host machines.

Socket Facilities: Socket subroutines and network library subroutines provide the building blocks for IPC. An application program must perform the following basic functions to conduct IPC through the socket layer:

- Create and name sockets.
- Accept and make socket connections.
- Send and receive data.
- Shut down socket operations.

Socket Interface: The Socket interface provides a standard, well-documented approach to access kernel network resources.

Socket Header Files to be Included: Socket header files contain data definitions, structures, constants, macros, and options used by socket subroutines. An application program must include the appropriate header file to make use of structures or other information a particular socket subroutine requires. Commonly used socket header files are:

/usr/include/netinet/in.h Defines Internet constants and structures. /usr/include/netdb.h Contains data definitions for socket subroutines. /usr/include/sys/socket.h Contains data definitions and socket structures. /usr/include/sys/types.h Contains data type definitions. /usr/include/arpa.h Contains definitions for internet operations. /usr/include/sys/errno.h Defines the errno values that are returned by drivers and other kernel-level code.

Internet address translation subroutines require the inclusion of the inet.h file. The inet.h file is located in the /usr/include/arpa directory.

ELEMENTARY SOCKET SYSTEM CALLS

Socket() System Call: Creates an end point for communication and returns a descriptor.

Syntax

#include <sys/socket.h> #include <sys/types.h>

int socket (int AddressFamily, int Type, int Protocol);

Description: The socket subroutine creates a socket in the specified AddressFamily and of the specified type. A protocol can be specified or assigned by the system. If the protocol is left unspecified (a value of 0), the system selects an appropriate protocol from those protocols in the address family that can be used to support the requested socket type.

The socket subroutine returns a descriptor (an integer) that can be used in later subroutines that operate on sockets.

a) Connection Oriented Implementation

<u>Server:</u>

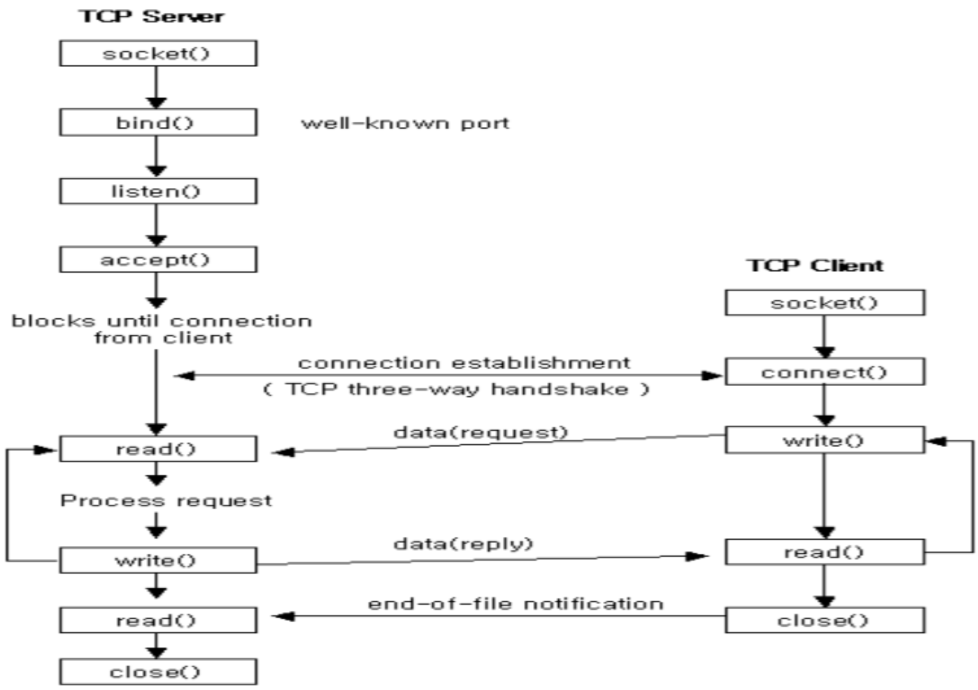
- Include appropriate header files.
- Create a TCP Socket.
- Fill in the socket address structure (with server information)
- Specify the port where the service will be defined to be used by client.
- Bind the address and port using bind() system call.
- Server executes listen() system call to indicate its willingness to receive connections.
- Accept the next completed connection from the client process by using an accept()• system call.
- Receive a message from the Client using recv() system call.
- Send the result of the request made by the client using send() system call.

<u>Client</u>

- Include appropriate header files.
- Create a TCP Socket.
- Fill in the socket address structure (with server information)
- Specify the port of the Server, where it is providing service
- Establish connection to the Server using connect() system call.
- For echo server, send a message to the server to be echoed using send() system call.

- Receive the result of the request made to the server using recv() system call.
- Write the result thus obtained on the standard output.

FLOW-CHART



Execution Procedure: Suppose, the server program is server.c and client program is client.c First compile the Server program as,

$ cc server.c – o obj ⇨ $ ./obj& ⇨ $ cc client.c ⇨ $./a.out

Validation:

Sample Input: Client sends a message that will be echoed by the Server, say “Hello”.

Sample Output: Server echoes the message back to the client i.e “Hello”

Objective 8: Write an Echo_Client and Echo_server using UDP to estimate the round trip time from client to the server. The server should be such that it can accept multiple connections at any given time.

Solution:

<u>Connection less Implementation</u>

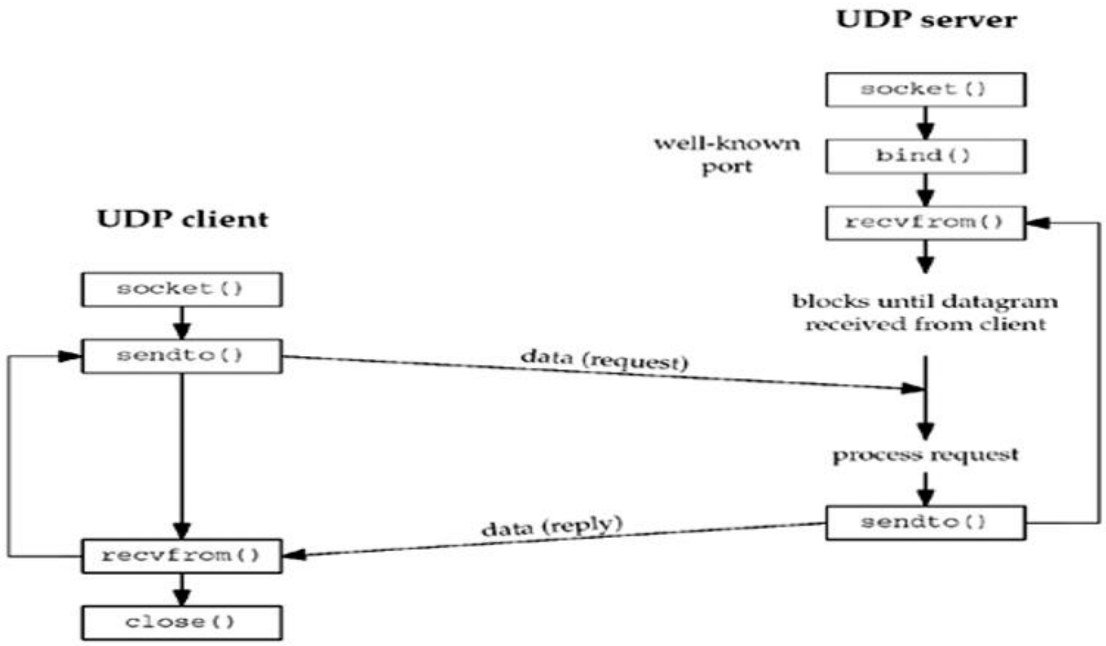
<u>Server:</u>

- Include appropriate header files.
- Create a UDP Socket.
- Fill in the socket address structure (with server information)
- Specify the port where the service will be defined to be used by client.
- Bind the address and port using bind() system call.
- Receive a message from the Client using recvfrom() system call.
- Send the result of the request made by the client using sendto() system call.

<u>Client</u>

- Include appropriate header files.
- Create a UDP Socket.
- Fill in the socket address structure (with server information)
- Specify the port of the Server, where it is providing service
- For echo server, send a message to the server to be echoed using sendto() system call.
- Receive the result of the request made to the server using recvfrom() system call.
- Write the result thus obtained on the standard output.

<u>FLOW CHART</u>



Execution Procedure:

Suppose, the server program is server.c and client program is client.c

First compile the Server program as,

$ cc server.c – o obj

$ ./obj&

$ cc client.c

$./a.out

Validation:

Sample Input: Client sends a message that will be echoed by the Server, say "Hello".

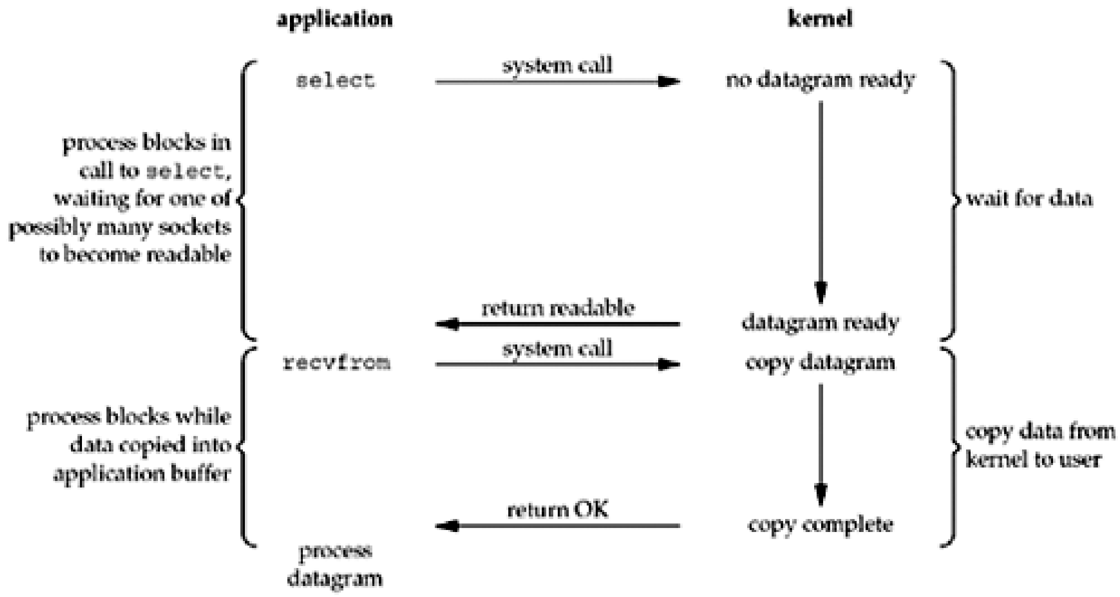
Sample Output: Server echoes the message back to the client i.e "Hello" .

Objective 9: Write an Echo_Client and Echo_server using TCP to estimate the round trip time from client to the server. The server should be such that it can accept multiple connections at any given time. multiplexed I/O operations.

Solution:

1. **I/O Multiplexing Model**

With **I/O multiplexing**, we call select or poll and block in one of these two system calls, instead of blocking in the actual I/O system call. The figure is a summary of the I/O multiplexing model:



We block in a call to select, waiting for the datagram socket to be readable. When select returns that the socket is readable, we then call recvfrom to copy the datagram into our application buffer.

2. **Comparing to the blocking I/O model** *

Comparing both the figures

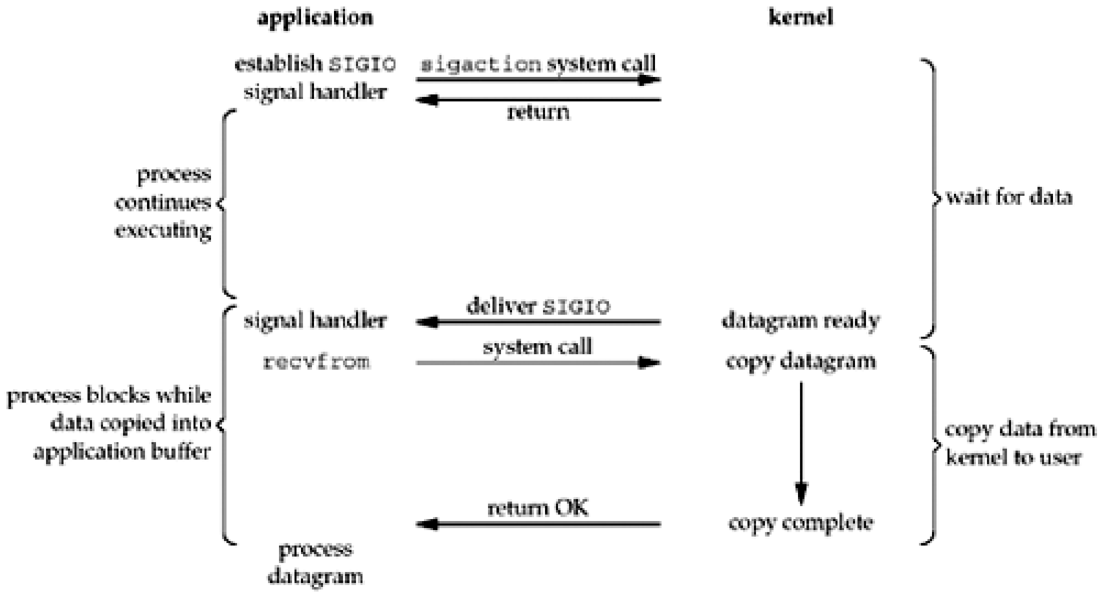
- Disadvantage: using select requires two system calls (select and recvfrom) instead of one
- Advantage: we can wait for more than one descriptor to be ready
  (see the select function later in this chapter)

   a. *Multithreading with blocking I/O* *

Another closely related I/O model is to use multithreading with blocking I/O. That model very closely resembles the model described above, except that instead of using select to block on multiple file descriptors, the program uses multiple threads (one per file descriptor), and each thread is then free to call blocking system calls like recvfrom.

2. Signal-Driven I/O Model

The **signal-driven I/O model** uses signals, telling the kernel to notify us with the SIGIO signal when the descriptor is ready. The figure is below:



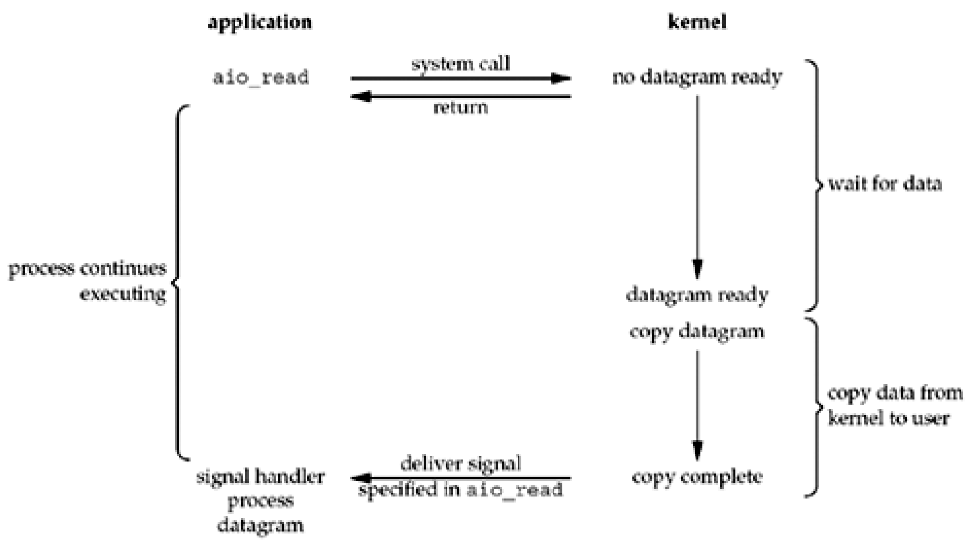
- We first enable the socket for signal-driven I/O and install a signal handler using the sigaction system call. The return from this system call is immediate and our process continues; it is not blocked.
- When the datagram is ready to be read, the SIGIO signal is generated for our process. We can either:
    - read the datagram from the signal handler by calling recvfrom and then notify the main loop that the data is ready to be processed.
    - notify the main loop and let it read the datagram.

The advantage to this model is that we are not blocked while waiting for the datagram to arrive. The main loop can continue executing and just wait to be notified by the signal handler that either the data is ready to process or the datagram is ready to be read.

3. Asynchronous I/O Model

Asynchronous I/O is defined by the POSIX specification, and various differences in the *real-time* functions that appeared in the various standards which came together to form the current POSIX specification have been reconciled.

These functions work by telling the kernel to start the operation and to notify us when the entire operation (including the copy of the data from the kernel to our buffer) is complete. <u>The main difference between this model and the signal-driven I/O model is that with signal-driven I/O, the kernel tells us when an I/O operation can be initiated, but with asynchronous I/O, the kernel tells us when an I/O operation is complete.</u> See the figure below for example:



- We call aio_read (the POSIX asynchronous I/O functions begin with aio_ or lio_) and pass the kernel the following:
  - descriptor, buffer pointer, buffer size (the same three arguments for read),
  - file offset (similar to lseek),
  - and how to notify us when the entire operation is complete.

  This system call returns immediately and our process is not blocked while waiting for the I/O to complete.

- We assume in this example that we ask the kernel to generate some signal when the operation is complete. This signal is not generated until the data has been copied into our application buffer, which is different from the signal-driven I/O model.

Objective 10: Simulate Bellman-Ford Routing algorithm in NS2.

Bellman-Ford algorithm is used to find minimum distance from the source vertex to any other vertex. The main difference between this algorithm with Dijkstra's the algorithm is, in Dijkstra's algorithm we cannot handle the negative weight, but here we can handle it easily.
Bellman-Ford algorithm finds the distance in a bottom-up manner. At first, it finds those distances which have only one edge in the path. After that increase the path length to find all possible solutions.

Input and Output

Input:

The cost matrix of the graph:

0 6 ∞ 7 ∞

∞ 0 5 8 -4

∞ -2 0 ∞ ∞

∞ ∞ -3 0 9

2 ∞ 7 ∞ 0

Output:

Source Vertex: 2

Vert: 0 1 2 3 4

Dist: -4 -2 0 3 -6

Pred: 4 2 -1 0 1

The graph has no negative edge cycle

2.7.1.4 Algorithm

bellmanFord(dist, pred, source)

Input: Distance list, the predecessor list, and the source vertex.
Output: True, when a negative cycle is found.

```
Begin
  iCount := 1
  maxEdge := n * (n - 1) / 2     //n is number of vertices

  for all vertices v of the graph, do
    dist[v] := ∞
    pred[v] := ϕ
  done

  dist[source] := 0
  eCount := number of edges present in the graph
  create edge list named edgeList

  while iCount < n, do
    for i := 0 to eCount, do
      if dist[edgeList[i].v] > dist[edgeList[i].u] + (cost[u,v] for edge i) dist[edgeList[i].v] >
dist[edgeList[i].u] + (cost[u,v] for edge i) pred[edgeList[i].v] := edgeList[i].u
    done
  done

  iCount := iCount + 1
  for all vertices i in the graph, do
    if dist[edgeList[i].v] > dist[edgeList[i].u] + (cost[u,v] for edge i),
      then return true
  done

  return false
End
```

Source Code:

```c
#include<stdio.h>
struct node
{
   unsigned dist[20];
   unsigned from[20];
}rt[10];
int main()
{
   int costmat[20][20];
   int nodes,i,j,k,count=0;
   printf("\nEnter the number of nodes : ");
   scanf("%d",&nodes);//Enter the nodes
   printf("\nEnter the cost matrix :\n");
   for(i=0;i<nodes;i++)
   {
      for(j=0;j<nodes;j++)
      {
         scanf("%d",&costmat[i][j]);
         costmat[i][i]=0;
         rt[i].dist[j]=costmat[i][j];//initialise the distance equal to cost matrix
         rt[i].from[j]=j;
      }
   }
      do
      {
         count=0;
         for(i=0;i<nodes;i++)//We choose arbitary vertex k and we calculate the direct distance
from the node i to k using the cost matrix
```

```
            //and add the distance from k to node j
            for(j=0;j<nodes;j++)
            for(k=0;k<nodes;k++)
                if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
                {//We calculate the minimum distance
                    rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
                    rt[i].from[j]=k;
                    count++;
                }
        }while(count!=0);
        for(i=0;i<nodes;i++)
        {
            printf("\n\n For router %d\n",i+1);
            for(j=0;j<nodes;j++)
            {
                printf("\t\nnode %d via %d Distance %d ",j+1,rt[i].from[j]+1,rt[i].dist[j]);
            }
        }
    printf("\n\n");
    getch();
}
/*
A sample run of the program works as:-
Enter the number of nodes :
3
Enter the cost matrix :
0 2 7
2 0 1
7 1 0
For router 1
node 1 via 1 Distance 0
```

```
node 2 via 2 Distance 2
node 3 via 3 Distance 3
For router 2
node 1 via 1 Distance 2
node 2 via 2 Distance 0
node 3 via 3 Distance 1
For router 3
node 1 via 1 Distance 3
node 2 via 2 Distance 1
node 3 via 3 Distance 0
*/
```

We have used the concept of I/O multiplexing for managing both TCP and UDP port in the same server.

```
fd_set fdvar;
FD_ZERO(&fdvar);
FD_SET(tcp_sfd,&fdvar);
FD_SET(udp_sfd,&fdvar);
int maxpl = max(tcp_sfd,udp_sfd);
cout << "Waiting for a client...\n";

if(select(maxpl+2 ,&fdvar,NULL,NULL,NULL)==-1)
{
    perror("error in select");
 }
if(FD_ISSET(udp_sfd,&fdvar))
{
    // UDP
 }
 else
 {
```

```
    //TCP
  }
```

struct fd_set can be set to monitor activity on many ports at the same time.

FD_SET(tcp_sfd,&fdvar) sets the appropriate bit in fdvar so that select may monitor tcp_sfd

similarly FD_SET(udp_sfd,&fdvar) sets the appropriate bit in fdvar so that select may monitor the udp_sfd.

We use select system call to monitor socket fd's for TCP & UDP servers which are listening to the same port ie. port 3001

FD_ISSET checks which sfd has received a packet.

Once this is known we handle the request as in case of TCP client or UDP client