St. Thomas College of Engineering & Technology
Vellilode, Sivapuram P.O., Mattanur, Kannur District, Kerala
Approved by AICTE New Delhi, Govt. of Kerala and Affiliated to APJ Abdul Kalam Technological University

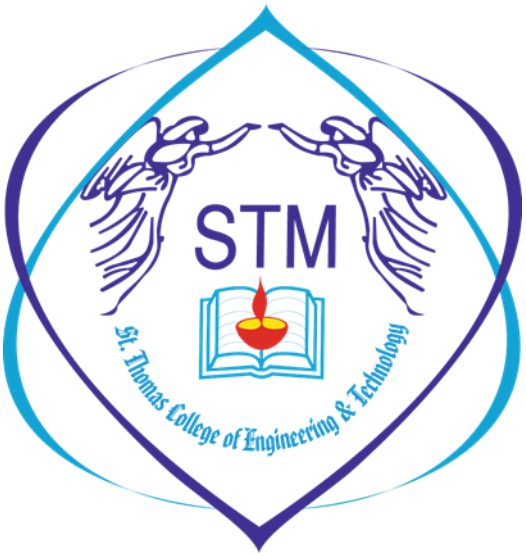
LAB MANUAL

CSL332- NETWORKING LAB

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

CSL332- NETWORKING LAB

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING



Revision	Date	Prepared by			Approved by		
		Name	Designation	Signature	Name	Designation	Signature
Rev1.0	16/12/24	Ms.VAISHAKHI V K	AP, CSE		Dr.AMITHA I.C	HOD, CSE	

VISION OF THE INSTITUTION

To be an Institute of repute recognized for excellence in education, innovation and social contribution.

MISSION OF THE INSTITUTION

M1: Infrastructural Relevance

Develop, maintain and manage our campus for our stakeholders.

M2: Life-Long Learning

Encourage our stakeholders to participate in lifelong learning through industry and academic interactions.

M3: Social Connect

Organize socially relevant outreach programs for the benefit of humanity.

VISION OF THE DEPARTMENT

To produce globally competent and socially responsible Computer Science Engineers.

MISSION OF THE DEPARTMENT

M1: Professional Skills

Provide students with opportunities to become industry- ready professionals and entrepreneurs through analytical learning.

M2: Lifelong Learning

Maintain a lifelong learning attitude and stay current in their profession to foster personal and organizational development.

M3: Engage with Society

Encourage students to focus on sustainable solutions, to improve quality of life and social welfare.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1. Professional Practices

Apply engineering practices required for Software development, Hardware development and Embedded systems.

PEO2. Intrapreneurial Skills

Exhibit innovation, Self – confidence and teamwork skills in the organization and society.

PEO3. Lifelong Learning

Offer continuing education programmes in the emerging areas for the knowledge upgradation of stakeholders.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: Computer Science and Engineering students can analyse, design, develop, test and apply management principles, mathematical foundations in the development of computational solutions, making them experts in designing computer hardware and software.

PSO2: Develop their skills to solve problems in the broad area of programming concepts and appraise environmental and social issues with ethics and manage different projects in interdisciplinary fields.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

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.

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.

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.

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.

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.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

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 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 change.

CSL332- NETWORKING LAB

SYLLABUS

*Mandatory(Note: At least one program from each topic in the syllabus should be completed in the Lab)

1. Getting started with the basics of network configuration files and networking commands in Linux.*
2. To familiarize and understand the use and functioning of system calls used for network programming in Linux.*
3. Implement client-server communication using socket programming and TCP as transport layer protocol*
4. Implement client-server communication using socket programming and UDP as transport layer protocol*
5. Simulate sliding window flow control protocols.* (Stop and Wait, Go back N, Selective Repeat ARQ protocols)
6. Implement and simulate algorithms for Distance Vector Routing protocol or Link State Routing protocol.*
7. Implement Simple Mail Transfer Protocol.
8. Implement File Transfer Protocol.*
9. Implement congestion control using a leaky bucket algorithm.*
10. Understanding the Wireshark tool.*
11. Design and configure a network with multiple subnets with wired and wireless LANs using required network devices. Configure commonly used services in the network.*
12. Study of NS2 simulator*

Course Objectives:

- The course enables the learners to get hands-on experience in network programming using Linux System calls and network monitoring tools.
- It covers implementation of network protocols and algorithms, configuration of network services and familiarization of network simulators.
- This helps the learners to develop, implement protocols and evaluate its performance for real world networks.

Course outcome:

- **CO1-** Use network related commands and configuration files in Linux Operating System.(Cognitive Knowledge Level: Understand).
- **CO2-** Develop network application programs and protocols.
  (Cognitive Knowledge Level: Apply)
- **CO3-** Analyze network traffic using network monitoring tools.
  (Cognitive Knowledge Level: Apply)
- **CO4-** Design and setup a network and configure different network protocols.
  (Cognitive Knowledge Level: Apply)
- **CO5-** Develop simulation of fundamental network concepts using a network simulator.(Cognitive Knowledge Level: Apply)

Mapping of course outcomes with program outcomes

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	2	2					2		2		2
CO2	2	2	2	2				2		2		2
CO3	1	2	2	1	2			2		2		2
CO4	1	2	2	2	1	2		1		1		2
CO5	1	2	2	2	1			1		1		2

Mark Distribution

Total Marks	CIE Marks	ESE Marks	ESE Duration
150	75	75	3 hours

References Books:

1. W. Richard Stevens, Bill Fenner, Andy Rudoff, UNIX Network Programming: Volume 1, The Sockets Networking API, 3rd Edition, Pearson, 2015

2. Lisa Bock, Learn Wireshark: Confidently navigate the Wireshark interface and solve real-world networking problems, Packt Publishing, 2019

3. Teerawat Issariyakul, Ekram Hossain, Introduction to Network Simulator NS2,2nd Edition, Springer,2019

Assessment Rubrics

Class Performance/ Continuous Assessment Test:

Sl no	Parameters	Marks	High	Medium	Low

1	Lab Performance	10	Complete in Minimum no. of steps(Optimal), Logically correct, Steps should be computable, sequentially executable and gives expected result for all possible inputs	Must be Logically correct, Steps should be computable, Sequentially executable and gives expected result for most of the inputs	Must be Logically correct, Sequentially executable and produces correct results but not in specified format.
			8-10	5-7	1-4
2	Documentation	10	Neat, well labelled and organized and completed within time	Neat, work is somewhat organized and completed.	Record is somewhat organized but not neat.
			8-10	5-7	1-4
3	Viva & Discussion	15	Very good knowledge of both theory and experimental procedure	Good knowledge of both theory and experimental procedure	Fair idea of theory and experimental procedure
			12-15	8-11	1-7
4	Code of Conduct & Team work	10	Punctual in all aspects , Proper dress code, Excellent work with proper attitude	Punctual ,Proper dress code, Good work with proper attitude	Lacking Punctuality (while coming to lab, Viva/submission of record), Fair dress code, Minimum work with lack of proper attitude
			8-10	5-7	1-4

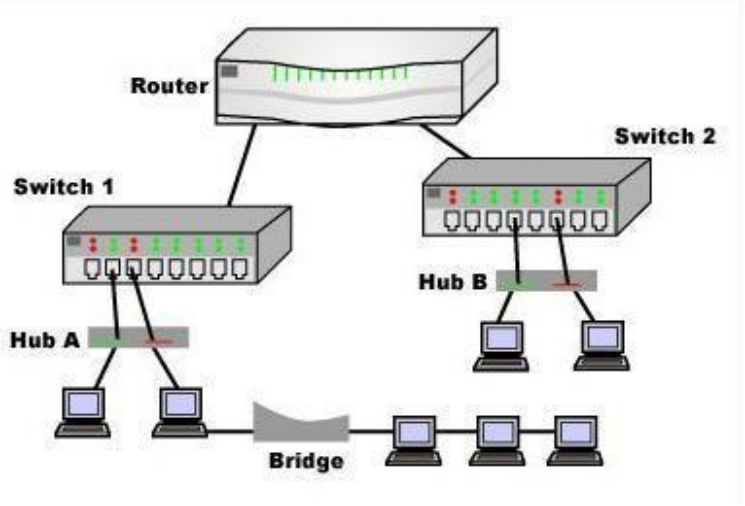
Contents

Experiment 1

Familiarization Of Network Devices

Aim: Getting started with the basics of network configuration files and networking commands in Linux.

1. Network Hub
2. Network Switch
3. Modem
4. Network Router
5. Bridge
6. Repeater
7. Gateway



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 switches.

Network Switch:

Like a hub, a switch also works at the layer of LAN (Local Area Network) but we 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 a forwarding table or forwarding information base (FIB).

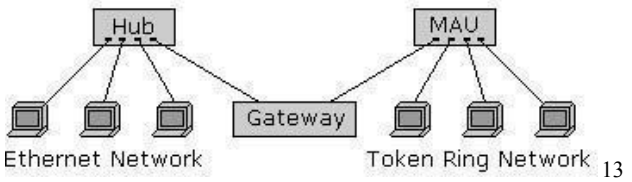
Modem:
The most common use for modems is for both sending and receiving of the digital information between personal computers. This information used to be transmitted over telephone lines using V.92, the last dial-up standard, to an analog modem that would convert the signal back to a digital format for a computer to read.
Now, access to the Internet more commonly takes place using high-speed broadband modems.
A modem stands for (Modulator+Demodulator). That means it modulates and demodulates the signal between the digital data of a computer and the analog signal of a telephone line.

Network Router:
A router is a network device which is responsible for routing traffic from one to another network. Routers perform the traffic directing functions on the Internet. A data packet is typically forwarded from one router to another router through the networks that constitute an internetwork until it reaches its destination node.

Bridge:
If a router connects two different types of networks, then a bridge connects two subnetworks as a part of the same network. A network bridge joins two otherwise separate computer networks to enable communication between them and allow them to work as a single network. Bridges are used with local area networks (LANs) to extend their reach to cover larger physical areas than the LAN can otherwise reach. Bridges are similar to—but more intelligent than—simple repeaters, which also extend signal range.

Repeater:
A repeater is an electronic device that amplifies the signal it receives. A network device used to regenerate or replicate a signal. Repeaters are used in transmission systems to regenerate analog or digital signals distorted by transmission loss. Analog repeaters frequently can only amplify the signal while digital repeaters can reconstruct a signal to near its original quality.

In a data network, a repeater can relay messages between subnetworks that use different protocols or cable types
Gateway
A gateway, as the name suggests, is a passage to connect two networks together that may work upon different networking models. They basically work as the messenger agents that take data from one system, interpret it, and transfer it to another system. Gateways are also called protocol converters and can operate at any network layer. Gateways are generally more complex than switch or router.



Experiment 1.2: Getting started with Basics of Networking commands in Linux

ifconfig

ifconfig (interface configurator) command is use to initialize an interface, assign IP Address to interface and enable or disable interface on demand. With this command we can view the IP Address and Hardware / MAC address assign to interface and also the MTU (Maximum transmission unit) size.

```
# ifconfig
```

- with -a options will display all available interface details if it is disable also. # ifconfig -a

- ifconfig with interface (eth0) command only shows specific interface details like IP Address, MAC Address, etc.

```
# ifconfig eth0
```

- The "**up**" or "**ifup**" flag with interface name (eth0) activates a network interface, if it is not in active state and allows it to send and receive information. For example, "**ifconfig eth0 up**" or "**ifup eth0**" will activate the eth0 interface.

```
#       ifconfig
eth0 up or
# ifup eth0
```

- The "**down**" or "**ifdown**" flag with interface name (**eth0**) deactivates the specified network interface. For example, "**ifconfig eth0 down**" or "**ifdown eth0**" command deactivates the **eth0** interface, if it is in active state.

```
#  ifconfig  eth0
down or
# ifdown eth0
```

- To assign an IP address to an specific interface, use the following command with an interface name (eth0) and ip address that you want to set. For example, "ifconfig eth0 172.16.25.125" will set the IP address to interface eth0.

```
# ifconfig eth0 172.16.25.125
```

- Using the "ifconfig" command with "netmask" argument and interface name as (eth0) allows you to define an netmask to an given interface. For example, "ifconfig eth0 netmask 255.255.255.224" will set the network mask to a given interface eth0.

```
# ifconfig eth0 netmask 225.225.225.224
```

ping

PING (Packet INternet Groper) command is the best way to test connectivity between two nodes. Ping uses ICMP (Internet Control Message Protocol) to communicate to other devices. You can ping the host name of the ip address using the below command.

#ping 8.8.8.8

#ping www.google.com

traceroute

traceroute is a network troubleshooting utility which shows the number of hops taken to reach a destination and also determines packets traveling path. Below we are tracing the route to global DNS server IP Address and being able to reach destination also shows path of that packet is traveling.

traceroute 4.2.2.2

Netstat

Netstat (**Network Statistic**) command display connection info, routing table information etc. To displays routing table information use option as **-r**

#netstat –r

nslookup

nslookup command to find out DNS related queries. The following examples shows A Record (IP Address) of tecmint.com.

nslookup www.google.com

host

host command to find name to IP and also query DNS records #host www.google.com

system-config-network

system-config-network in command prompt to configure network setting and you will get nice Graphical User Interface (GUI) which may also use to configure IP Address**,** Gateway**,** DNS etc

system-config-network

Viva Questions:

1. What is the purpose of networking in Linux?
2. What is the purpose of the ifconfig command in Linux?
3. What does the ping command do?
4. What is the difference between ping and traceroute?
5. What information can be retrieved using the netstat command?
6. What is a **router**, and what is its role in networking?
7. At which OSI layer does a bridge operate?
8. How does a modem differ from a router?

EXPERIMENT NO.2
SYSTEM CALLS

AIM

To familiarize and understand the use and functioning of system calls used for network programming in Linux

Sockets

Sockets are programming constructs used to communicate between processes. There are different types of systems that sockets can be used for, the main one of interest to us are Internet-based sockets (the other commonly used socket is Unix sockets).

TCP (Transmission Control Protocol)

Stream sockets use the Transmission Control Protocol (TCP) to communicate. TCP is stream-oriented, sending a stream of bytes to the receiver. It is also a reliable transport protocol, which means it guarantees that all data arrives at the receiver, and arrives in order. TCP starts be setting up a connection, and then sending data between sender and receiver. TCP is used for most data- oriented applications like web browsing, file transfer and email.

UDP (User Datagram Protocol)

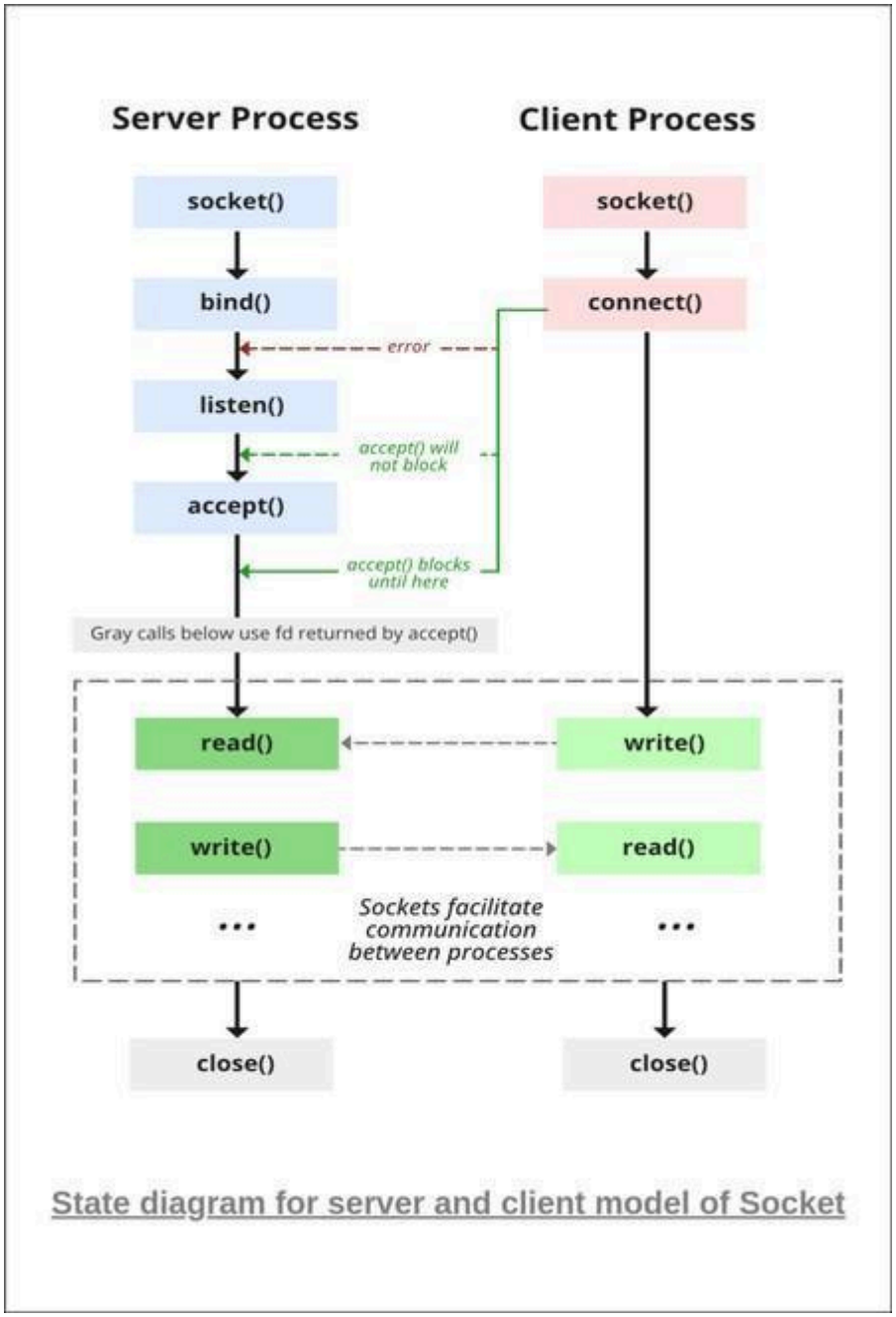
Datagram sockets use the User Datagram Protocol (UDP) to communicate. UDP is an unreliable protocol. There is no connection setup or retransmissions. The sender simply sends a packet (datagram) to the receiver, and hopes that it arrives. UDP is used for most real-time oriented applications like voice over IP and video conversations.

Socket Functions

The client creates a socket and then connects to the server. The connect() system call from the client triggers a TCP SYN segment from client to server. The server accepts the connection from the client. The accept() system call is actually a blocking function—when the program calls accept(), the server does not return from the function until it receives a TCP SYN segment from a client, and completes the 3-way handshake. After the client returns from the connect() system call, and the server returns from the accept() system call, a connection has been established. Now the two can send data. Sending and receiving data is performed using the write() and read() functions.

read() is a blocking function—it will only return when the socket receives data. You (the application programmer) must correctly coordinate reads and writes between the client and server. If a client calls the read() function, but no data is sent from the server, then the client will wait forever!

Client and Server Model



State diagram for server and client model of Socket

Socket creation:

int sockfd = socket(domain, type, protocol)

sockfd: socket descriptor, an integer (like a file-handle)

domain: integer, specifies communication domain. We use AF_ LOCAL as defined in the POSIX standard for communication between processes on the same host. For communicating between processes on different hosts connected by IPV4, we use AF_INET and AF_I NET 6 for processes connected by IPV6.

type: communication type

SOCK_STREAM: TCP(reliable, connection

oriented) SOCK_DGRAM: UDP(unreliable,

connectionless)

protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on the protocol field in the IP header of a packet.

1. Connect Function

The connect() function is used by a TCP client to establish a connection with a TCP

server/ The function is defined as follows:

#include <sys/socket.h>

int connect (int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);

where sockfd is the socket descriptor returned by the socket function, the second and third arguments are pointers to the socket address structure and its size.

The function returns 0 if it succeeds in establishing a connection (i.e., successful TCP three-way handshake, -1 otherwise.

2. Bind Function

he bind() assigns a local protocol address to a socket. With the Internet protocols, the address is the combination of an IPv4 or IPv6 address (32-bit or 128-bit) address along with a 16 bit TCP port number.

The function is defined as follows:

#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);

where sockfd is the socket descriptor, myaddr is a pointer to a protocol-specific address and addrlen is the size of the address structure.

bind() returns 0 if it succeeds, -1 on error.

struct sockaddr_in serv; /* IPv4 socket address structure */ bind(sockfd, (struct sockaddr*)

&serv, sizeof(serv))

3. Listen Function

The listen() function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket. It is defined as follows:

#include <sys/socket.h>

listen(int sockfd, int backlog);

where **sockfd** is the socket descriptor and **backlog** is the maximum number of connections the kernel should queue for this socket.

4. Accept Function

The accept() is used to retrieve a connect request and convert that into a request. It is defined as follows:

#include <sys/socket.h>

accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);

where sockfd is a new file descriptor that is connected to the client that called the connect(). The cliaddr and addrlen arguments are used to return the protocol address of the client. The new socket descriptor has the same socket type and address family of the original socket. The original socket passed to accept() is not associated with the connection, but instead remains available to receive additional connect requests.

Close Procedure.

The normal close() function is used to close a socket and terminate a TCP socket. It returns 0 if it succeeds, -1 on error. It is defined as follows:

#include

<unistd.h> int close(int

sockfd);

5. Send and Receive function

The recv() function is similar to read(), but allows to specify some options to control how the data are received. We will not consider the possible options in this course. We will assume it is equal to 0.

receive is defined as follows:

#include <sys/socket.h>

recv(int sockfd, void *buf, size_t nbytes, int flags);

Since a socket endpoint is represented as a file descriptor, we can use read and write to communicate with a socket as long as it is connected. However, if we want to specify options we need another set of functions.

For example, send() is similar to write() but allows to specify some options. send() is defined as follows:

#include <sys/socket.h>

send(int sockfd, const void *buf, size_t nbytes, int flags);

where buf and nbytes have the same meaning as they have with write. The additional argument flags is used to specify how we want the data to be transmitted.

Viva Questions:

1. What is the purpose of the socket() function?
2. What is the difference between AF_INET and AF_INET6?
3. What is the role of bind() in socket programming?
4. Why do we need the listen() function in a server program?
5. What does the accept() function do in a server socket?
6. How does the connect() function work in a client socket?
7. What is the difference between send() and write() functions?
8. What is the difference between recv() and read() functions?
9. What is the purpose of sendto() and send() functions?
10. What is a datagram socket?

Experiment 3
Client-server communication using socket programming and TCP as transport layer protocol

Aim

Implement client-server communication using socket programming and TCP as transport layer protocol.

Server Algorithm:

1. Create a socket using socket(AF_INET, SOCK_STREAM, 0).
2. Bind the socket to an IP address (127.0.0.1) and port (2000).
3. Listen for incoming connections using listen().
4. Accept a client connection using accept().
5. Repeat until "end" is received:
   - Receive a message from the client.
   - If the message is "end", close the connection and exit.
   - Display the client's message.
   - Take user input (server response).
   - Send the response to the client.
   - If the response is "end", close the connection and exit.
6. Close the connection and terminate.

Client Algorithm:

1. Create a socket using socket(AF_INET, SOCK_STREAM, 0).
2. Connect to the server at 127.0.0.1 and port 2000.
3. Repeat until "end" is sent or received:
   - Take user input (client message).
   - Send the message to the server.
   - If the message is "end", close the connection and exit.
   - Receive a response from the server.
   - If the response is "end", close the connection and exit.
   - Display the server's message.
4. Close the connection and terminate.

Server Program

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define PORT 2000
```

```
int main() {
    struct sockaddr_in client_address, server_address;
    int socket_descriptor_1, socket_descriptor_2;
    char message_1[100], message_2[100];

    // Create socket
    socket_descriptor_1 = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_descriptor_1 < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Define server address
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);  // Convert port to network byte order
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Bind socket to the specified IP and port
    if (bind(socket_descriptor_1, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
        perror("Bind failed");
        return 1;
    }

    // Listen for incoming connections
    if (listen(socket_descriptor_1, 1) < 0) {
        perror("Listen failed");
        return 1;
    }

    printf("Server is listening on port %d...\n", PORT);

    int size_of_client_address = sizeof(client_address);
    socket_descriptor_2 = accept(socket_descriptor_1, (struct sockaddr *)&client_address,
&size_of_client_address);
    if (socket_descriptor_2 < 0) {
        perror("Accept failed");
        return 1;
    }

    printf("Client connected.\n");

    // Chat loop
    while (1) {
        memset(message_1, 0, sizeof(message_1));
        recv(socket_descriptor_2, message_1, sizeof(message_1), 0);

        if (strcmp(message_1, "end") == 0) {
            printf("Client disconnected.\n");
            break;
        }

        printf("\nClient: %s", message_1);
        printf("\nServer: ");
```

```
    scanf("%s", message_2);

    send(socket_descriptor_2, message_2, sizeof(message_2), 0);

    if (strcmp(message_2, "end") == 0) {
        printf("Server ending chat.\n");
        break;
    }
}

    // Close sockets
    close(socket_descriptor_2);
    close(socket_descriptor_1);

    return 0;
}
```

Client Program

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define PORT 2000

int main() {
    struct sockaddr_in server_address;
    int socket_descriptor;
    char message_1[100], message_2[100];

    // Create socket
    socket_descriptor = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_descriptor < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Define server address
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);  // Convert port to network byte order
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Connect to the server
    if (connect(socket_descriptor, (struct sockaddr *)&server_address,
sizeof(server_address)) < 0) {
        perror("Connection failed");
```

```
        return 1;
    }

    printf("Connected to server.\n");

    // Chat loop
    while (1) {
        printf("\nClient: ");
        scanf("%s", message_2);

        send(socket_descriptor, message_2, sizeof(message_2), 0);

        if (strcmp(message_2, "end") == 0) {
            printf("Client ending chat.\n");
            break;
        }

        memset(message_1, 0, sizeof(message_1));
        recv(socket_descriptor, message_1, sizeof(message_1), 0);

        if (strcmp(message_1, "end") == 0) {
            printf("Server ended chat.\n");
            break;
        }

        printf("\nServer: %s", message_1);
    }

    // Close socket
    close(socket_descriptor);

    return 0;
}
```

Output

Client Side

Client: Hello
Server: Hi

Client: How are you?
Server: Good

Client: end

Client ending chat.

Server Side:
Client: Hello
Server: Hi

Client: How are you?
Server: Good

Client: end
Client disconnected.

Viva Questions:

1. What is client-server communication in networking?
2. What is the difference between TCP and UDP in socket programming?
3. Why is TCP preferred over UDP for reliable communication?
4. What are the key characteristics of TCP as a transport layer protocol?
5. How does TCP ensure reliable data transmission?

Experiment 4
Client-server communication using socket programming and UDP as transport layer protocol

Aim

Implement client-server communication using socket programming and UDP as transport layer protocol.

Server Algorithm:

1. Create a UDP socket using socket(AF_INET, SOCK_DGRAM, 0).
2. Bind the socket to a specific IP (127.0.0.1) and port (3000).
3. Wait for client messages in a loop:
   - Receive a message from the client.
   - If the message is "end", print a message and exit.
   - Display the received message.
   - Get user input (server's reply).
   - Send the reply to the client.
   - If the reply is "end", exit the loop.
4. Close the socket and terminate.

Client Algorithm:

1. Create a UDP socket using socket(AF_INET, SOCK_DGRAM, 0).
2. Define the server address (IP: 127.0.0.1, Port: 3000).
3. Repeat until "end" is sent or received:
   - Get user input (client's message).
   - Send the message to the server.
   - If the message is "end", exit the loop.
   - Receive a response from the server.
   - If the response is "end", exit the loop.
   - Display the received message.
4. Close the socket and terminate.

Server Program

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define PORT 3000
#define BUFFER_SIZE 100

int main() {
   struct sockaddr_in server_address, client_address;
```

```
int socket_descriptor;
char message[BUFFER_SIZE];
socklen_t client_len = sizeof(client_address);

// Create UDP socket
socket_descriptor = socket(AF_INET, SOCK_DGRAM, 0);
if (socket_descriptor < 0) {
    perror("Socket creation failed");
    return 1;
}

// Define server address
server_address.sin_family = AF_INET;
server_address.sin_port = htons(PORT);
server_address.sin_addr.s_addr = INADDR_ANY;

// Bind socket to port
if (bind(socket_descriptor, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
    perror("Bind failed");
    return 1;
}

printf("Server is waiting for messages...\n");

while (1) {
    // Receive message from client
    recvfrom(socket_descriptor, message, BUFFER_SIZE, 0,
            (struct sockaddr *)&client_address, &client_len);

    if (strcmp(message, "end") == 0) {
        printf("Client ended the chat.\n");
        break;
    }

    printf("Client: %s\n", message);

    // Get server's reply
    printf("Server: ");
    scanf("%s", message);

    // Send response to client
    sendto(socket_descriptor, message, strlen(message) + 1, 0,
           (struct sockaddr *)&client_address, client_len);

    if (strcmp(message, "end") == 0) {
        printf("Server exiting...\n");
        break;
    }
}

// Close socket
close(socket_descriptor);
return 0;
```

```
}
```

Client Program

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define PORT 3000
#define BUFFER_SIZE 100

int main() {
    struct sockaddr_in server_address;
    int socket_descriptor;
    char message[BUFFER_SIZE];

    // Create UDP socket
    socket_descriptor = socket(AF_INET, SOCK_DGRAM, 0);
    if (socket_descriptor < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Define server address
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");

    while (1) {
        printf("Client: ");
        scanf("%s", message);

        // Send message to server
        sendto(socket_descriptor, message, strlen(message) + 1, 0,
               (struct sockaddr *)&server_address, sizeof(server_address));

        if (strcmp(message, "end") == 0) {
            printf("Client exiting...\n");
            break;
        }

        // Receive response from server
        socklen_t server_len = sizeof(server_address);
        recvfrom(socket_descriptor, message, BUFFER_SIZE, 0,
                 (struct sockaddr *)&server_address, &server_len);

        printf("Server: %s\n", message);
    }

    // Close socket
```

```
    close(socket_descriptor);
    return 0;
}
```

Output

Client Side

Client: Hello
Server: Hi

Client: How are you?
Server: Good

Client: end
Client exiting...

Server Side

Server is waiting for messages...
Client: Hello
Server: Hi

Client: How are you?
Server: Good

Client: end
Client ended the chat.
Server exiting...

Viva Questions:

1. Why is UDP considered a connectionless protocol?
2. In which applications is UDP preferred over TCP?
3. How does UDP handle packet loss?
4. What is the role of ports and IP addresses in UDP communication?
5. Why is UDP faster than TCP?

Experiment 5
File Transfer Protocol

Aim

To Implement File Transfer Protocol

Server Algorithm:

1. Create a TCP socket using `socket(AF_INET, SOCK_STREAM, 0)`.
2. Bind the socket to `INADDR_ANY` and port `6000`.
3. Listen for incoming client connections using `listen(socket_descriptor, QSIZE)`.
4. Accept a client request using `accept(socket_descriptor, ...)`.
5. Receive the filename requested by the client.
6. Check if the file exists.
    - If the file exists:
        - Open the file and read its content.
        - Send the file content to the client.
        - Print the content for verification.
        - Close the file after sending.
    - If the file does not exist, notify the client.
7. Close the client connection.
8. Close the server socket.

Client Algorithm:

1. Create a TCP socket using `socket(AF_INET, SOCK_STREAM, 0)`.
2. Connect to the server at `127.0.0.1:6000`.
3. Ask the user to enter a filename to request from the server.
4. Send the filename to the server.
5. Wait for the file response.
    - If the file exists on the server:
        - Receive the file content.
        - Print the content.
        - Indicate that the requested file has been received.
    - If the file is missing, display an error message.
6. Close the client socket.

Server Program

```
#include <arpa/inet.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#define BUFSIZE 1024
```

```
#define PORT_ADDR 6000
#define QSIZE 5
int main() {
   struct sockaddr_in server_address, client_address;
   int socket_descriptor_1, socket_descriptor_2, size_of_client_address, file_descriptor,
character_count;
   char buffer_1[BUFSIZE], buffer_2[BUFSIZE];
   // Creating a socket
   socket_descriptor_1 = socket(AF_INET, SOCK_STREAM, 0);
   if (socket_descriptor_1 < 0) {
      perror("Socket creation failed");
      exit(EXIT_FAILURE);
   }
   // Configure server address
   server_address.sin_family = AF_INET;
   server_address.sin_port = htons(PORT_ADDR);
   server_address.sin_addr.s_addr = htonl(INADDR_ANY);
   // Bind the socket
   if (bind(socket_descriptor_1, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
      perror("Binding failed");
      exit(EXIT_FAILURE);
   }
   // Listen for client connections
   listen(socket_descriptor_1, QSIZE);
   printf("Server listening on port %d...\n", PORT_ADDR);
size_of_client_address = sizeof(client_address);
   socket_descriptor_2 = accept(socket_descriptor_1, (struct sockaddr *)&client_address, (socklen_t
*)&size_of_client_address);
   if (socket_descriptor_2 < 0) {
      perror("Accept failed");
      exit(EXIT_FAILURE);
   }
   printf("Client connected...\n");
   // Receive filename from client
   character_count = read(socket_descriptor_2, buffer_1, BUFSIZE);
   buffer_1[character_count] = '\0';
   printf("Requested file: %s\n", buffer_1);
   // Open the requested file
   if ((file_descriptor = open(buffer_1, O_RDONLY)) >= 0) {
      printf("File found. Sending data...\n");

      // Read the file and send content to the client
      while ((character_count = read(file_descriptor, buffer_2, BUFSIZE)) > 0) {
         write(socket_descriptor_2, buffer_2, character_count);
      }
      printf("File transfer complete.\n");
      close(file_descriptor);
   } else {
      printf("File not found.\n");
      write(socket_descriptor_2, "File not found", 15);
   }
```

```
    // Close connections
    close(socket_descriptor_2);
    close(socket_descriptor_1);
    return 0;
}
```

Client Program

```
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define BUFSIZE 1024
#define PORT_ADDR 6000
int main() {
   struct sockaddr_in server_address;
   int socket_descriptor, character_count;
   char buffer_1[BUFSIZE], buffer_2[BUFSIZE];
   // Creating socket
   socket_descriptor = socket(AF_INET, SOCK_STREAM, 0);
   if (socket_descriptor < 0) {
      perror("Socket creation failed");
      exit(EXIT_FAILURE);
   }
   // Configure server address
   server_address.sin_family = AF_INET;
   server_address.sin_port = htons(PORT_ADDR);
   server_address.sin_addr.s_addr = inet_addr("127.0.0.1");

   // Connect to server
   if (connect(socket_descriptor, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
      perror("Connection failed");
      exit(EXIT_FAILURE);
   }
   // Get filename from user
   printf("Enter the filename: ");
   scanf("%s", buffer_1);
      // Send filename to server
   write(socket_descriptor, buffer_1, strlen(buffer_1));
   // Receive file content from server
   printf("\nReceiving file...\n");
   character_count = read(socket_descriptor, buffer_2, BUFSIZE);
   if (character_count > 0) {
      printf("File Content:\n");
      write(1, buffer_2, character_count);
      printf("\nFile received successfully.\n");
   } else {
      printf("File not found on server.\n");
   }
   // Close socket
   close(socket_descriptor);
   return 0;
}
```

Output

Client Side

```
$ gcc FileClient.c -o client
$ ./client
Enter the filename: sample.txt
Receiving file...
File Content:
This is a sample text file.
File received successfully.
```

Server Side

```
$ gcc FileServer.c -o server
$ ./server
Server listening on port 6000...
Client connected...
Requested file: sample.txt
File found. Sending data...
File transfer complete.
```

Viva Questions:

1. What is FTP, and what is its purpose?
2. How does FTP work?
3. What is the difference between FTP and HTTP for file transfers?
4. What are the advantages and disadvantages of FTP?
5. What are some real-world applications of FTP?

Experiment 6
Distance Vector Routing protocol

Aim

Implement and simulate algorithm for Distance Vector Routing protocol.

Algorithm

1. Start
2. Input the number of nodes.
3. Input the cost matrix (where 0 represents the distance to itself, and other values represent link costs).
4. Initialize each router's distance table:
5. Set the distance to itself as 0.
6. Set the distance to other nodes based on the input cost matrix.
7. Set the next-hop for each destination as the node itself.
8. Repeat until no more updates occur:
9. For each router:
10. Check all possible paths to each destination.
11. If a shorter path is found via an intermediate node, update the distance and next-hop information.
12. Print the final routing table for each router, showing the shortest path to all other nodes.
13. End

Program

```
#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;

   // Input: Number of nodes
   printf("\nEnter the number of nodes: ");
   scanf("%d", &nodes);

   // Input: Cost matrix
   printf("\nEnter the cost matrix:\n");
   for (i = 0; i < nodes; i++) {
      for (j = 0; j < nodes; j++) {
         scanf("%d", &costmat[i][j]);
         if (i == j)
            costmat[i][j] = 0; // Distance from a node to itself is always 0
         rt[i].dist[j] = costmat[i][j]; // Initialize distance matrix
         rt[i].from[j] = j; // Set the source of each route
      }
```

```
    }

    // Bellman-Ford Algorithm
    do {
       count = 0;
       for (i = 0; i < nodes; i++) { // Pick each node
          for (j = 0; j < nodes; j++) { // Pick each destination
             for (k = 0; k < nodes; k++) { // Pick an intermediate node
                if (rt[i].dist[j] > costmat[i][k] + rt[k].dist[j]) {
                   // Update distance and path if a shorter path is found
                   rt[i].dist[j] = costmat[i][k] + rt[k].dist[j];
                   rt[i].from[j] = k;
                   count++;
                }
             }
          }
       }
    } while (count != 0); // Continue until no updates

    // Output: Distance Vector Table
    for (i = 0; i < nodes; i++) {
       printf("\n\nRouting table for router %d\n", i + 1);
       for (j = 0; j < nodes; j++) {
          printf("Node %d via %d Distance %d\n", j + 1, rt[i].from[j] + 1, rt[i].dist[j]);
       }
    }
    printf("\n");
    return 0;
}
```

Output

```
Enter the number of nodes: 3
Enter the cost matrix:
0 2 7
2 0 1
7 1 0

Routing table for router 1
Node 1 via 1 Distance 0
Node 2 via 2 Distance 2
Node 3 via 2 Distance 3

Routing table for router 2
Node 1 via 1 Distance 2
Node 2 via 2 Distance 0
Node 3 via 3 Distance 1

Routing table for router 3
Node 1 via 2 Distance 3
Node 2 via 2 Distance 1
Node 3 via 3 Distance 0
```

Viva Questions:

1. What is a Distance Vector Routing Protocol?
2. How does Distance Vector Routing work?
3. What information does a router exchange in Distance Vector Routing?
4. What is a routing table, and how is it updated in Distance Vector Routing?
5. How does a router determine the best path to a destination?

Experiment 7
Stop and wait

Aim

Simulate sliding window flow control protocol stop and wait.

Algorithm for Server:

1. Create a UDP socket.
2. Bind the socket to a port (8080).
3. Wait for a connection request from the client.
4. Receive the total number of frames from the client.
5. Enter a loop to receive each frame:
    - Receive a frame number.
    - If the received frame number is `-99`, terminate the connection.
    - Display the received frame number.
    - Ask for user input to send an acknowledgment (`1` for success, `-1` for resend).
    - Send the acknowledgment back to the client.
6. Close the socket.

Algorithm for Client:

1. Create a UDP socket.
2. Send a connection request to the server.
3. Receive confirmation from the server.
4. Input the total number of frames and send it to the server.
5. Enter a loop to send frames:
    - Send the frame number to the server.
    - Wait for an acknowledgment.
    - If the acknowledgement is `-1`, resend the frame.
    - Repeat until a positive acknowledgment (`1`) is received.
6. After all frames are sent, send `-99` to indicate completion.
7. Close the socket.

Server Program

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT 8080

int main() {
   struct sockaddr_in servaddr, cliaddr;
   int sockfd, len, n, frame, ack;
```

```
    printf("\nWaiting for client...\n");

    // Creating socket file descriptor
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Clear memory
    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    // Assign server details
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);

    // Bind the socket
    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("Bind failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    len = sizeof(cliaddr);

    // Wait for client connection
    recvfrom(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&cliaddr, &len);
    printf("\nClient connected successfully.\n");

    // Receive total number of frames
    printf("\nWaiting for total number of frames...\n");
    recvfrom(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&cliaddr, &len);
    printf("\nTotal frames to receive: %d\n", n);

    while (1) {
        recvfrom(sockfd, &frame, sizeof(frame), 0, (struct sockaddr *)&cliaddr, &len);

        if (frame == -99) {
            printf("\nTransmission completed. Closing server.\n");
            break;
        }

        printf("\nReceived Frame-%d", frame);
        printf("\nEnter 1 for +ve ACK and -1 for -ve ACK: ");
        scanf("%d", &ack);

        sendto(sockfd, &ack, sizeof(ack), 0, (struct sockaddr *)&cliaddr, len);
    }

    close(sockfd);
    return 0;
}
```

Client Program

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT 8080

int main() {
   struct sockaddr_in servaddr;
   int sockfd, n, ack, len;

   printf("\nSearching for server...\n");

   // Creating socket file descriptor
   sockfd = socket(AF_INET, SOCK_DGRAM, 0);
   if (sockfd == -1) {
      perror("Socket creation failed");
      exit(EXIT_FAILURE);
   }

   // Clear memory
   memset(&servaddr, 0, sizeof(servaddr));

   // Assign server details
   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(PORT);
   servaddr.sin_addr.s_addr = INADDR_ANY;

   // Notify server of connection
   n = -1;
   sendto(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
   printf("\nServer connected successfully.\n");

   // Get total number of frames from user
   printf("\nEnter the total number of frames: ");
   scanf("%d", &n);

   sendto(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));

   len = sizeof(servaddr);

   // Sending frames
   for (int i = 1; i <= n; i++) {
      do {
         printf("\nSending Frame-%d", i);
         sendto(sockfd, &i, sizeof(i), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
         printf("\nWaiting for ACK...\n");

         recvfrom(sockfd, &ack, sizeof(ack), 0, (struct sockaddr *)&servaddr, &len);

         if (ack == -1) {
            printf("\nNegative ACK received.. Resending...\n");
         }
```

```
        } while (ack == -1);
    }

    // End transmission
    n = -99;
    sendto(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
    printf("\nSuccessfully sent all frames.\n");

    close(sockfd);
    return 0;
}
```

Output
Server Side

```
Waiting for client...
Client connected successfully.
Waiting for total number of frames...
Received frame-1
Enter 1 for +ve ack and -1 for -ve ack.
1
Received frame-2
Enter 1 for +ve ack and -1 for -ve ack.
-1
Received frame-2
Enter 1 for +ve ack and -1 for -ve ack.
1
...
Successfully received all frames.
```

Client Side

```
Searching for server...
Server connected successfully.
Enter the total number of frames: 5
Sending frame-1
Waiting for ACK...
Sending frame-2
Waiting for ACK...
Negative ACK received.. resending...
Sending frame-2
Waiting for ACK...
...
Successfully sent all the frames.
```

Viva Questions:

1. What is the Stop-and-Wait protocol?
2. How does the Stop-and-Wait protocol work?
3. Why is Stop-and-Wait considered a simple flow control mechanism?
4. What is the difference between Stop-and-Wait Flow Control and Stop-and-Wait ARQ?
5. What are the advantages and disadvantages of the Stop-and-Wait protocol?

Experiment 8

Implement Selective Repeat protocol

Aim
To implement Selective repeat protocol.

Server Algorithm

1. Create a UDP socket and bind it to a port.
2. Wait for a connection request from the client.
3. Request window size from the client.
4. Compute total packets and frames to be sent.
5. Send packets in frames based on the window size.
6. Wait for acknowledgements:
   - If positive acknowledgment, continue sending.
   - If negative acknowledgment, retransmit the frame.
7. Repeat until all packets are sent successfully.
8. Close the connection.

Client Algorithm

1. Create a UDP socket and send a connection request to the server.
2. Receive a request for window size and send the window size.
3. Receive total packets and frames.
4. Start receiving frames from the server.
5. Process the received frames:
   - If correct, send a positive acknowledgment.
   - If corrupted/missing, send a negative acknowledgment.
6. Repeat until all packets are received.
7. Close the connection.

Program

Server Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>

#define MAX_WINDOW_SIZE 4
#define PORT 5018
#define MAX_PACKETS 40

struct frame {
   int packet[MAX_PACKETS];
};
```

```
struct ack {
   int acknowledge[MAX_PACKETS];
};

void clear_screen() {
   system("clear");  // Use "cls" for Windows
}

int main() {
   int socket_fd;
   struct sockaddr_in serveraddr, clientaddr;
   socklen_t len;
   struct frame f1;
   struct ack acknowledgement;

   int windowsize, totalpackets, totalframes;
   int i = 0, j = 0, framessend = 0, k, m, n, l;
   int repacket[MAX_PACKETS];
   char req[50];

   // Create UDP socket
   socket_fd = socket(AF_INET, SOCK_DGRAM, 0);
   if (socket_fd < 0) {
      perror("Socket creation failed");
      exit(1);
   }

   bzero(&serveraddr, sizeof(serveraddr));
   serveraddr.sin_family = AF_INET;
   serveraddr.sin_port = htons(PORT);
   serveraddr.sin_addr.s_addr = INADDR_ANY;

   if (bind(socket_fd, (struct sockaddr*)&serveraddr, sizeof(serveraddr)) < 0) {
      perror("Binding failed");
      close(socket_fd);
      exit(1);
   }

   len = sizeof(clientaddr);

   // Waiting for client connection
   printf("\nWaiting for client connection...\n");
   recvfrom(socket_fd, req, sizeof(req), 0, (struct sockaddr*)&clientaddr, &len);
   printf("\nClient connected: %s\n", req);

   // Request window size
   sendto(socket_fd, "REQUEST FOR WINDOW SIZE", strlen("REQUEST FOR WINDOW SIZE"), 0, (struct
sockaddr*)&clientaddr, sizeof(clientaddr));
   recvfrom(socket_fd, &windowsize, sizeof(windowsize), 0, (struct sockaddr*)&clientaddr, &len);

   clear_screen();
   printf("Window size obtained: %d\n", windowsize);

   // Calculate packets and frames
   totalpackets = windowsize * 5;
   totalframes = 5;

   // Send total packets and frames info
   sendto(socket_fd, &totalpackets, sizeof(totalpackets), 0, (struct sockaddr*)&clientaddr, sizeof(clientaddr));
   recvfrom(socket_fd, req, sizeof(req), 0, (struct sockaddr*)&clientaddr, &len);
```

```
    sendto(socket_fd, &totalframes, sizeof(totalframes), 0, (struct sockaddr*)&clientaddr, sizeof(clientaddr));
    recvfrom(socket_fd, req, sizeof(req), 0, (struct sockaddr*)&clientaddr, &len);

    printf("\nPress Enter to start transmission...\n");
    fgets(req, 2, stdin);
    clear_screen();

    // Start sending packets
    j = 0;
    l = 0;
    while (l < totalpackets) {
        bzero(&f1, sizeof(f1));

        printf("Sending frame %d\n", framessend);
        for (m = 0; m < j; m++) {
            f1.packet[m] = repacket[m];
        }

        while (j < windowsize && i < totalpackets) {
            f1.packet[j] = i;
            i++;
            j++;
        }

        sendto(socket_fd, &f1, sizeof(f1), 0, (struct sockaddr*)&clientaddr, sizeof(clientaddr));
        printf("Waiting for acknowledgment...\n");

        recvfrom(socket_fd, &acknowledgement, sizeof(acknowledgement), 0, (struct sockaddr*)&clientaddr, &len);
        clear_screen();

        j = 0;
        m = 0;
        k = 0;
        n = 1;

        while (m < windowsize && n < totalpackets) {
            if (acknowledgement.acknowledge[m] == -1) {
                printf("\nNegative ACK received for packet: %d\n", f1.packet[m]);
                repacket[j] = f1.packet[m];
                j++;
                k = 1;
            } else {
                l++;
            }
            m++;
            n++;
        }

        if (k == 0) {
            printf("\nAll packets acknowledged successfully.\n");
        }

        framessend++;
        printf("\nPress Enter to proceed...\n");
        fgets(req, 2, stdin);
        clear_screen();
    }
    printf("All frames sent successfully. Closing connection.\n");
    close(socket_fd);
    return 0;
}
```

Client Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAX_PACKETS 40
#define PORT 5018

struct frame {
   int packet[MAX_PACKETS];
};

struct ack {
   int acknowledge[MAX_PACKETS];
};

void clear_screen() {
   system("clear");  // Use "cls" for Windows
}

int main() {
   int client_socket_fd;
   struct sockaddr_in serveraddr;
   socklen_t len;
   struct frame f1;
   struct ack acknowledgement;

   int windowsize, totalpackets, totalframes;
   int i = 0, j = 0, framesreceived = 0, k, m, n, l;
   int repacket[MAX_PACKETS];
   char req[50];

   client_socket_fd = socket(AF_INET, SOCK_DGRAM, 0);
   if (client_socket_fd < 0) {
      perror("Socket creation failed");
      exit(1);
   }

   bzero(&serveraddr, sizeof(serveraddr));
   serveraddr.sin_family = AF_INET;
   serveraddr.sin_port = htons(PORT);
   serveraddr.sin_addr.s_addr = inet_addr("127.0.0.1");

   // Connect to server
   printf("Sending request to server...\n");
   sendto(client_socket_fd, "HI IAM CLIENT", strlen("HI IAM CLIENT"), 0, (struct sockaddr*)&serveraddr,
sizeof(serveraddr));

   recvfrom(client_socket_fd, req, sizeof(req), 0, (struct sockaddr*)&serveraddr, &len);
   printf("Server response: %s\n", req);

   printf("\nEnter window size: ");
   scanf("%d", &windowsize);

   sendto(client_socket_fd, &windowsize, sizeof(windowsize), 0, (struct sockaddr*)&serveraddr, sizeof(serveraddr));

   recvfrom(client_socket_fd, &totalpackets, sizeof(totalpackets), 0, (struct sockaddr*)&serveraddr, &len);
```

```
    printf("Total packets: %d\n", totalpackets);

    sendto(client_socket_fd, "RECEIVED", strlen("RECEIVED"), 0, (struct sockaddr*)&serveraddr, sizeof(serveraddr));

    printf("Receiving frames...\n");
    while (i < totalpackets) {
        recvfrom(client_socket_fd, &f1, sizeof(f1), 0, (struct sockaddr*)&serveraddr, &len);
        printf("Received frame %d\n", framesreceived);

        for (m = 0; m < windowsize; m++) {
            printf("\nPacket %d: ", f1.packet[m]);
            scanf("%d", &acknowledgement.acknowledge[m]);
        }

        sendto(client_socket_fd, &acknowledgement, sizeof(acknowledgement), 0, (struct sockaddr*)&serveraddr,
sizeof(serveraddr));
        framesreceived++;
        clear_screen();
    }

    printf("All frames received. Closing connection.\n");
    close(client_socket_fd);
    return 0;
}
```

OUTPUT

Client

Sending request to the server...
Waiting for reply...
Server response: REQUEST FOR WINDOW SIZE

Enter the window size: 4
Waiting for server response...
Total packets: 20
Total frames: 5

Starting to receive frames...
Expected frame 1 with packets: 0 1 2 3
Received frame 1
Enter negative acknowledgment for corrupted packets (-1 for errors, 1 for correct):
Packet 0: 1
Packet 1: 1
Packet 2: -1
Packet 3: 1

Waiting for retransmission...
Received frame 1 with packets: 2 3
...
All frames received successfully.
Closing connection with the server.

Server

Waiting for client connection...
Client connection obtained: HI IAM CLIENT.

Sending request for window size...
Waiting for the window size...
The window size obtained as: 4

Total packets obtained: 20
Total frames to be transmitted: 5

Sending frame 1 with packets: 0 1 2 3
Waiting for the acknowledgement...
Received negative acknowledgment for packet: 2
Retransmitting frame 1 with packets: 2 3

Sending frame 2 with packets: 4 5 6 7
Positive acknowledgment received.

Sending frame 3 with packets: 8 9 10 11
...
All frames sent successfully.
Closing connection with client.

Viva

1. What is the Selective Repeat (SR) protocol?
2. How does Selective Repeat differ from Go-Back-N (GBN)?
3. What is the significance of the window size in Selective Repeat?
4. How does Selective Repeat improve efficiency compared to Go-Back-N?
5. What is the main advantage of Selective Repeat over Go-Back-N?

Experiment 9

Implement Go Back N protocol

Aim

To implement Go Back N

Algorithm for Server

1. Create a socket using socket().
2. Initialize the server address structure with IP, port, and protocol type.
3. Bind the socket to the specified address and port using bind().
4. Listen for incoming connections using listen().
5. Accept a client connection using accept().
6. Read data from the client (frame of data).
7. Check for errors in the received data.
   - If there is an error, retransmit the affected frame.
   - If there is no error, continue sending the next frame.
8. Continue sending frames until the entire message is transmitted.
9. Send an exit message when transmission is complete.
10. Close the socket and terminate the server.

Algorithm for Client

1. Create a socket using socket().
2. Initialize the server address structure with IP and port.
3. Connect to the server using connect().
4. Receive frames from the server.
5. Check for errors in the received frame:
   - If there is no error, send acknowledgement (-1).
   - If error occurs, send the sequence number of the incorrect frame.
6. Receive retransmitted frames (if any).
7. Continue receiving frames until the server sends an exit message.
8. Close the socket and terminate the client.

Server Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>

#define SIZE 4  // Frame size
```

```
int main() {
    int sfd, lfd, len, i, j, status;
    char str[20], frame[20], temp[20], ack[20];
    struct sockaddr_in saddr, caddr;

    // Creating socket
    sfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sfd < 0) {
        perror("Socket creation failed");
        exit(1);
    }

    // Initializing server structure
    memset(&saddr, 0, sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);
    saddr.sin_port = htons(5465);

    // Binding socket
    if (bind(sfd, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
        perror("Bind error");
        exit(1);
    }

    // Listening for connections
    listen(sfd, 5);
    len = sizeof(caddr);

    // Accepting client connection
    lfd = accept(sfd, (struct sockaddr *)&caddr, (socklen_t *)&len);
    if (lfd < 0) {
        perror("Accept error");
        exit(1);
    }

    printf("Enter the text: ");
    scanf("%s", str);

    i = 0;
    while (i < strlen(str)) {
        memset(frame, 0, 20);
        strncpy(frame, str + i, SIZE);  // Extract 4-character frame

        // Transmitting Frames
        printf("Transmitting Frames: ");
        len = strlen(frame);
        for (j = 0; j < len; j++) {
            printf("%d", i + j);
            sprintf(temp, "%d", i + j);
            strcat(frame, temp);
        }
```

```
        printf("\n");

        write(lfd, frame, sizeof(frame));
        read(lfd, ack, sizeof(ack));  // Read acknowledgment

        sscanf(ack, "%d", &status);
        if (status == -1) {
            printf("Transmission is successful\n");
        } else {
            printf("Received error in %d\n", status);
            printf("\nRetransmitting frame: ");

            // Retransmitting frames
            for (j = 0;;) {
                frame[j] = str[j + status];
                printf("%d", j + status);
                j++;
                if ((j + status) % SIZE == 0) break;
            }
            printf("\n");

            frame[j] = '\0';
            len = strlen(frame);
            for (j = 0; j < len; j++) {
                sprintf(temp, "%d", j + status);
                strcat(frame, temp);
            }
            write(lfd, frame, sizeof(frame));
        }
        i += SIZE;
    }

    write(lfd, "Exit", sizeof("Exit"));
    printf("Exiting\n");

    close(lfd);
    close(sfd);
    return 0;
}
```

Client Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
int main() {
    int sfd, choice;
```

```
    char str[20], err[20];
    struct sockaddr_in saddr;

    // Creating socket
    sfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sfd < 0) {
        perror("Socket creation failed");
        exit(1);
    }
    // Initializing server structure
    memset(&saddr, 0, sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);
    saddr.sin_port = htons(5465);

    // Connecting to the server
    if (connect(sfd, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
        perror("Connection error");
        exit(1);
    }

    for (;;) {
        // Receive Frame
        read(sfd, str, sizeof(str));
        if (strcmp(str, "Exit") == 0) {
            printf("Exiting\n");
            break;
        }
        printf("\n\nReceived: %s\nDo you want to report an error? (0 for No, 1 for Yes): ", str);
        scanf("%d", &choice);
        if (!choice) {
            write(sfd, "-1", sizeof("-1"));  // Send acknowledgment (-1 means no error)
        } else {
            printf("Enter the sequence number of the frame where the error occurred: ");
            scanf("%s", err);
            write(sfd, err, sizeof(err));  // Send error acknowledgment
            // Receive retransmitted frame
            read(sfd, str, sizeof(str));
            printf("\nReceived the retransmitted frame: %s\n", str);
        }
    }
    close(sfd);
    return 0;
}
```

Server Side Output

```
Enter the text: helloworld
Transmitting Frames...0123
Received error in 2
Retransmitting frame 23
Transmitting Frames...4567
```

Received error in 4
Retransmitting frame 4567
Transmitting Frames...89
Transmission is successful
Exiting

Client Side Output
Received 0123
Do you want to report an error? 1
Enter the sequence of frame where error has occurred: 2
Received the retransmitted frames 23

Received 4567
Do you want to report an error? 1
Enter the sequence of frame where error has occurred: 4
Received the retransmitted frames 4567

Received 89
Do you want to report an error? 0
Transmission successful
Exiting

Viva

1. What is Go-Back-N (GBN) protocol?
2. What is the window size in Go-Back-N?
3. How does Go-Back-N handle errors?
4. What happens if an acknowledgment (ACK) is lost?
5. Why is Go-Back-N called ‘Go-Back’?

Experiment 10

Implement Leaky bucket protocol

Aim

To implement Leaky bucket protocol

Algorithm

1. Start the program.
2. Read bucket size, outgoing rate, and number of inputs.
3. Initialize store = 0.
4. Repeat for n times:
   i. Read incoming packet size.
   ii. If the packet fits in the bucket, add it to the store.
   iii. Else, drop excess packets and fill the bucket to its maximum capacity.
   iv. Reduce the store by the outgoing rate.
   v. Ensure the store does not go below zero.
   vi. Print the current bucket status.
5. End the program.

Program

```
#include <stdio.h>
int main() {
    int incoming, outgoing, bucket_size, n, store = 0;
    // Read bucket size, outgoing rate, and number of packets
    printf("Enter bucket size, outgoing rate, and number of inputs: ");
    scanf("%d %d %d", &bucket_size, &outgoing, &n);
    while (n > 0) {
        // Read incoming packet size
        printf("Enter incoming packet size: ");
        scanf("%d", &incoming);
        printf("Incoming packet size: %d\n", incoming);
        // If incoming packets fit in the bucket, store them
        if (incoming <= (bucket_size - store)) {
            store += incoming;
            printf("Bucket buffer size: %d out of %d\n", store, bucket_size);
        } else {
            // Drop excess packets and fill bucket to max
            printf("Dropped %d packets\n", incoming - (bucket_size - store));
            store = bucket_size;
            printf("Bucket buffer size: %d out of %d\n", store, bucket_size);
        }
        // Send outgoing packets
        store -= outgoing;
        if (store < 0)
            store = 0;
        printf("After outgoing, %d packets left out of %d in buffer\n", store, bucket_size);
```

```
        n--; // Reduce number of inputs
    }
    return 0;
}
```

Output

```
Enter bucket size, outgoing rate and number of inputs: 10 4 3
Enter incoming packet size: 6
Incoming packet size is 6
Bucket buffer size is 6 out of 10
After outgoing, 2 packets left out of 10 in buffer

Enter incoming packet size: 8
Incoming packet size is 8
Dropped 4 no of packets
Bucket buffer size is 10 out of 10
After outgoing, 6 packets left out of 10 in buffer

Enter incoming packet size: 3
Incoming packet size is 3
Bucket buffer size is 9 out of 10
After outgoing, 5 packets left out of 10 in buffer
```

Viva

1. What is the Leaky Bucket Algorithm?
2. Why is the Leaky Bucket Algorithm used in network traffic shaping?
3. How does the Leaky Bucket control congestion in a network?
4. What are the key parameters of the Leaky Bucket model?
5. What is the difference between the Leaky Bucket and Token Bucket algorithms?

Experiment 11
Understanding the Wireshark tool

Aim
To Familiarizing Wireshark Tool and NS2 Simulator

Install Wireshark (if not already installed) using:

bash
sudo apt install wireshark -y
Grant non-root user permissions if needed:

bash

sudo usermod -aG wireshark $(whoami)
sudo dpkg-reconfigure wireshark-common

Launch Wireshark:

- Open Wireshark from the applications menu or run wireshark in the terminal.

Start Packet Capture:

- Select an active network interface (Wi-Fi, Ethernet) and click Start to begin capturing live network traffic.

Observe Captured Packets:

- Analyze packets in real-time, showing details such as source/destination IP, protocol, port numbers, and payload.

Apply Filters:

- Use display filters to narrow down the packet view. Examples:
  - Capture only HTTP traffic: http
  - Filter packets from a specific IP: ip.addr == 192.168.1.1
  - Show only TCP traffic: tcp

Packet Inspection:

- Click on a packet to expand its details, including Ethernet, IP, TCP/UDP, and application-layer data.

Save and Export Data:

- Go to File > Save As and save the captured packets in .pcap format for future analysis.

Stop Capture & Exit:

- Click Stop and close Wireshark after completing the analysis.

Viva

1. What is Wireshark?
2. What are the key features of Wireshark?
3. What is packet sniffing?
4. What is a packet in networking?
5. What are the different types of packets captured by Wireshark?

Experiment 12
Understanding the NS2 Simulator tool

Introduction to NS2: NS2 is an open-source network simulator used for simulating wired and wireless networks. It is widely used in research and academic fields to study network protocols, routing, and performance analysis.

Key Features of NS2:
Simulation of Wired and Wireless Networks – Supports TCP, UDP, FTP, CBR, and various routing protocols.
Event-Driven Simulation – Works based on time-scheduled network events.
Support for Various Network Protocols – Includes routing, MAC, and application-layer protocols.
Trace File Generation – Produces trace files (.tr) for analyzing network behavior.
Integration with NAM (Network Animator) – Provides visual representation of simulations.

Basic NS2 Workflow:
1. Install NS2 sudo apt install ns2 -y

2. Verify Installation:
Run the following command to check if NS2 is installed: ns
If installed successfully, it will display % indicating the NS2 interpreter is running.
Type exit to close.

3. Writing a Simple NS2 Simulation Script:
Create a TCL (Tool Command Language) script to define network topology and traffic flow. simple.tcl

```
# Create a simulator instance
set ns [new Simulator]
# Define network nodes
set n1 [$ns node] set n2 [$ns node]
# Create a link between nodes
$ns duplex-link
$n1 $n2 1Mb 10ms DropTail
# Define a TCP connection
set tcp [new Agent/TCP]
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent
$n2 $sink
$ns connect
$tcp $sink
# Create a simulation trace file set tracefile [open out.tr w]
$ns trace-all
$tracefile
# Run simulation
$ns run
```

4. Run the Simulation:

Execute the script using the following command: ns simple.tcl
This will generate a trace file (out.tr) containing network simulation details.

5. Analyze Results:
Use the trace file (out.tr) to analyze network performance. Use NAM (Network Animator) for visualization: nam out.nam

6. Modify and Experiment: Change bandwidth, delay, or protocol types in the TCL script and observe different results.

Viva
1. What is NS2?
2. What are the key features of NS2?
3. What are the main components of NS2?
4. Which programming languages are used in NS2?
5. How do you run an NS2 script?