DATE : 16/12/2024

EXPERIMENT :01

NETWORKING COMMANDS IN LINUX

**AIM**: Familiarity with basics of network configuration files and networking commands in Linux - ifconfig, netstat, ping, arp, telnet, ftp, finger.

**PROGRAM OBJECTIVE**: Understanding and using of commands like ifconfig, netstat, ping, arp,telnet, ftp, finger, traceroute, whois.

**PROGRAM DESCRIPTION**: UNIX utilities are commands that, generally, perform a single task. It may be as simple as printing the date and time, or a complex as finding files that match many criteria throughout a directory hierarchy.

# **IFCONFIG**

**Linux (ifconfig)**: ifconfig is a command-line tool used to configure network interfaces. It allows users to set IP addresses, netmasks, and enable or disable interfaces. The ifconfig -a command is particularly useful for displaying all network interfaces, including those that are inactive. This makes it a fundamental tool for network administration and troubleshooting in Linux environments.

**Windows (ipconfig)**: ipconfig serves as the Windows equivalent, providing information about the computer's TCP/IP network configuration. By using commands like ipconfig /all, users can view detailed information about network adapters, IP addresses, DNS servers, and other network-related settings. This tool is crucial for diagnosing network connectivity issues and understanding the network configuration of a Windows system.

# **NETSTAT**

**netstat** is a versatile command-line utility designed to provide insights into network activity and configurations. It displays active network connections, routing tables, and network interface statistics, making it a valuable tool for network troubleshooting and performance monitoring. Common parameters include:

* -a: Shows all active TCP connections and listening ports.
* -e: Displays Ethernet statistics, such as sent and received bytes and packets.
* -f: Displays fully qualified domain names for foreign addresses.
* -i: (Linux/Unix) Displays network interfaces and their statistics.
* -n: Shows active TCP connections with numerical addresses and ports.
* -o: (Windows) Shows active TCP connections with process IDs (PIDs).
* -p: (Linux) Shows which processes are using which sockets.

# These parameters allow users to analyze network traffic, identify potential issues, and measure network performance. PING

**ping** is a command-line network utility used to test the reachability of a host on an Internet Protocol (IP) network. It functions by sending Internet Control Message Protocol (ICMP) echo request packets to the target host and then listening for ICMP echo response replies. This process allows users to determine if a connection exists between their device and the target host.

While often used to estimate round-trip time, ping primarily focuses on verifying connectivity and reporting packet loss. It provides

a statistical summary of the test, highlighting any lost packets and indicating the stability of the connection. Beyond basic connectivity checks, ping can also be employed to self-test a computer's network interface card, ensuring it is functioning correctly. It's a fundamental tool for network troubleshooting, allowing users to quickly identify potential network issues and confirm IP- based device connections.

# **ARP**

In computer networking, the Address Resolution Protocol (ARP) is the method for finding a host's link layer (hardware) address when only its Internet Layer (IP) or some other Network Layer address is known. ARP has been implemented in many types of networks; it is not an IP-only or Ethernet-only protocol. It can be used to resolve many different network layer protocol addresses to interface hardware addresses, although, due to the overwhelming prevalence of IPv4 and Ethernet, ARP is primarily used to translate IP addresses to Ethernet MAC addresses.

# **TELNET**

Telnet (Telecommunication network) is a network protocol used on the Internet or local area network (LAN) connections. In Linux, the telnet command is used to create a remote connection with a system over a TCP/IP network. Typically, telnet provides access to a command-line interface on a remote machine. The term telnet also refers to software which implements the client part of the protocol. Telnet clients are available for virtually all platforms. Protocol details: Telnet is a client server protocol, based on a reliable connection-oriented transport. Typically this protocol is used to establish a connection to TCP port 23

# **FTP**

File Transfer Protocol (FTP): FTP is a network protocol used to transfer data from one computer to another through a network such as the Internet.FTP is a file transfer protocol for exchanging and manipulating files over a TCP computer network. An FTP client may connect to an FTP server to manipulate files on that server.FTP runs over TCP. It defaults to listen on port 21 for incoming connections from FTP clients. A connection to this port from the FTP Client forms the control stream on which commands are passed from the FTP client to the FTP server and on occasion from the FTP server to the FTP client. FTP uses out-of-band control, which means it uses a separate

connection for control and data. Thus, for the actual file transfer to take place, a different connection is required which is called the data stream. To establish an FTP connection to a remote system, use the ftp command with the remote system's IP address:

ftp [IP]

For instance, connecting to a remote server with the IP address 192.168.100.9:

ftp 192.168.100.9

# FINGER

In computer networking, the Name/Finger protocol and the Finger user information protocol are simple network protocols for the exchange of human- oriented status and user information.

# **TRACEROUTE**

**traceroute** is a computer network tool used to determine the route taken by packets across an IP network. An IPv6 variant, traceroute6, is also widely available. Traceroute is often used for network troubleshooting. By showing a list of routers traversed, it allows the user to identify the path taken to reach a particular destination on the network. This can help identify routing problems or firewalls that may be blocking access to a site. Traceroute is also used by penetration testers to gather information about network infrastructure and IP ranges around a given host. It can also be used when downloading data, and if there are multiple mirrors available for the same piece of data, one can trace each mirror to get a good idea of which mirror would be the fastest to use. The traceroute command in Windows is traceroute. On a Linux system, the command is traceroute. man traceroute

# **WHO IS**

WHOIS (pronounced "who is"; not an acronym) is a query/response protocol which is widely used for querying an official database in order to determine the owner of a domain name, an IP address, or an autonomous system number on the Internet. WHOIS lookups were traditionally made using a command line interface, but a number of simplified web-based tools now exist for looking up domain ownership details from different databases. WHOIS normally runs on TCP port 43.

The WHOIS system originated as a method that system administrators could use to look up information to contact other IP address or domain name administrators (almost like "white pages").

whois 216.58.206.46

# **Result:**

Get familiarised with various network commands.

DATE :16/12/2024

# **EXPERIMENT – 02**

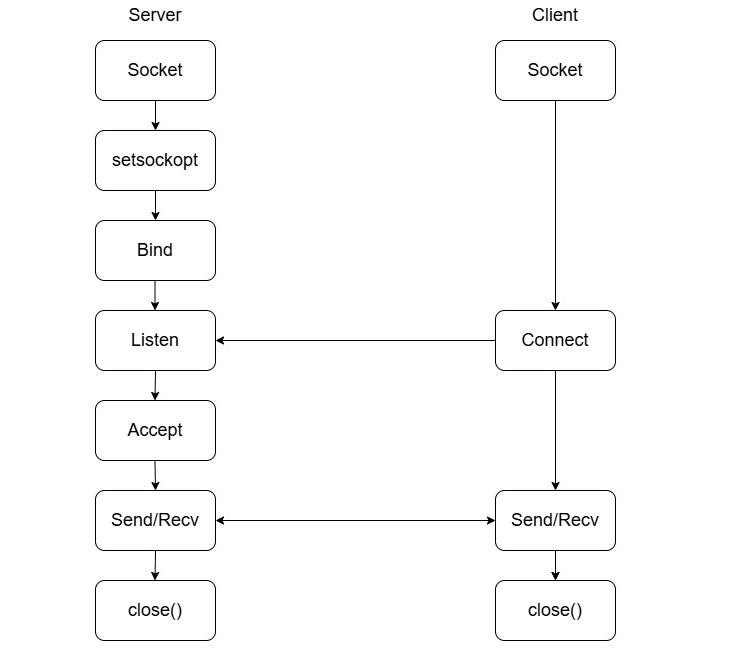
# **a. COMMUNICATION IMPLEMENTATION USING TCP**

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

**PROGRAM OBJECTIVE :** Understand fundamental Linux system calls for network programming,including socket creation,connection handling and data transfer.

**PROGRAM DEFINITION :** This program explores core Linux system calls for network communication,demonstrating socket creation,connection establishment, and data transfer.

The figure represents TCP (Transmission Control Protocol) communication between a client and a server using system calls in C programming. Below is a detailed description of each system call and its parameters.



**Server-Side System Calls :**

1. socket()

Creates a socket for communication.

Syntax:

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

Parameters:

AF\_INET: Address family (IPv4).

SOCK\_STREAM: Stream socket (TCP).

0: Default protocol (TCP).

Returns: A socket descriptor (sockfd) on success, -1 on failure.

1. setsockopt() (Optional, used to configure socket options)

Allows the server to reuse the address immediately after termination.

Syntax:

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &option, sizeof(option));

Parameters:

sockfd: Socket descriptor.

SOL\_SOCKET: Level where the option is applied.

SO\_REUSEADDR: Allows reuse of the address. option: Boolean flag (1 = enable, 0 = disable). sizeof(option): Size of the option variable.

Returns: 0 on success, -1 on failure.

1. bind()

Binds the socket to a specific IP address and port.

Syntax:

bind(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

Parameters:

sockfd: Socket descriptor. server\_addr: Server address structure (contains IP & port).

sizeof(server\_addr): Size of the address structure.

Returns: 0 on success, -1 on failure.

## 4. listen()

Prepares the socket to accept incoming connections.

Syntax:

listen(sockfd, backlog);

Parameters:

sockfd: Socket descriptor.

backlog: Number of pending connections allowed in the queue.

Returns: 0 on success, -1 on failure.

1. accept()

Accepts an incoming client connection.

Syntax: int new\_sock = accept(sockfd, (struct sockaddr \*)&client\_addr, &addr\_len);

Parameters:

sockfd: Socket descriptor. client\_addr: Structure to store the client's address.

addr\_len: Size of the client's address structure.

Returns: New socket descriptor for communication, -1 on failure.

1. send() / recv()

send(): Sends data to the client.

recv(): Receives data from the client.

Syntax for send():

send(new\_sock, buffer, buffer\_size, flags);

Syntax for recv():

recv(new\_sock, buffer, buffer\_size, flags);

Parameters:

new\_sock: Socket descriptor for communication. buffer: Data buffer for sending/receiving. buffer\_size: Size of the buffer.

flags: Special flags (usually 0).

Returns: Number of bytes sent/received, -1 on failure.

**Client-Side System Calls:**

1. socket()

Creates a socket for communication (same as server).

## 2. connect()

Establishes a connection with the server.

Syntax:

connect(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

Parameters:

sockfd: Socket descriptor. server\_addr: Server's address structure.

sizeof(server\_addr): Size of the address structure.

Returns: 0 on success, -1 on failure.

## 3. send() / recv()

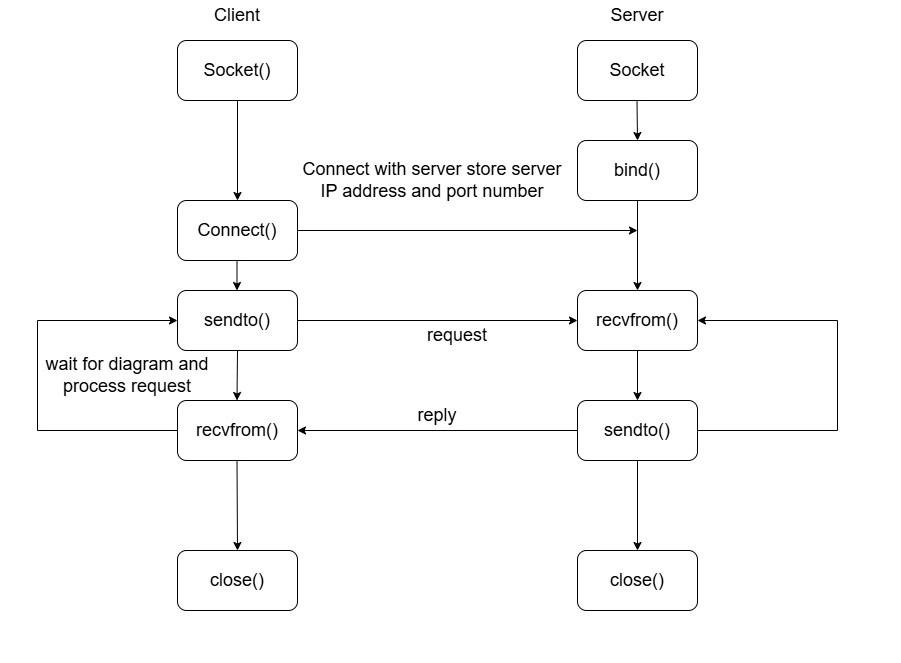
Same as the server side, used for data exchange.

Flow Explanation:

1. Server: Creates a socket (socket()) and sets options (setsockopt()).
2. Server: Binds (bind()) the socket to an IP and port.
3. Server: Listens (listen()) for incoming connections.
4. Client: Creates a socket (socket()) and connects (connect()) to the server.
5. Server: Accepts (accept()) the connection.
6. Both Client & Server: Use send() and recv() to exchange data.

This is a connection-oriented model since TCP establishes a reliable connection before data transfer.

# **b.COMMUNICATION IMPLEMENTATION USING UDP**



The diagram represents UDP (User Datagram Protocol) communication between a client and a server using system calls in C programming. UDP is a connectionless protocol, meaning data is sent without establishing a persistent connection.

**Client-Side System Calls:**

## 1. socket()

Creates a UDP socket.

Syntax:

int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

Parameters:

AF\_INET: Address family (IPv4).

SOCK\_DGRAM: Datagram socket (UDP).

0: Default protocol (UDP).

Returns: A socket descriptor (sockfd) on success; -1 on failure.

2. connect() (Optional in UDP)

Associates the socket with a specific server address and port.

Syntax: connect(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

Parameters:

sockfd: Socket descriptor. server\_addr: Server address structure. sizeof(server\_addr): Size of the structure.

Returns: 0 on success; -1 on failure.

## 3. sendto()

Sends a datagram (message) to the server.

Syntax: sendto(sockfd, buffer, buffer\_size, flags, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

Parameters:

sockfd: Socket descriptor. buffer: Pointer to the data being sent. buffer\_size: Size of the data. flags: Special flags (usually 0). server\_addr: Address of the destination server. sizeof(server\_addr): Size of the address structure.

Returns: Number of bytes sent; -1 on failure.

## 4. recvfrom()

Receives a response from the server.

Syntax:

recvfrom(sockfd, buffer, buffer\_size, flags, (struct sockaddr \*)&server\_addr,

&addr\_len);

Parameters:

sockfd: Socket descriptor. buffer: Pointer to store received data. buffer\_size: Maximum size of data to receive.

flags: Special flags (usually 0). server\_addr: Structure to store sender's address. addr\_len: Pointer to store address length.

Returns: Number of bytes received; -1 on failure.

## 5. close()

Closes the socket connection.

Syntax:

close(sockfd);

Parameters:

sockfd: Socket descriptor.

**Server-Side System Calls:**

1. socket()

Creates a UDP socket (same as the client).

## 2. bind()

Binds the socket to a specific IP address and port.

Syntax: bind(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

Parameters:

sockfd: Socket descriptor. server\_addr: Server address structure (contains IP & port). sizeof(server\_addr): Size of the address structure.

Returns: 0 on success; -1 on failure.

## 3. recvfrom()

Waits for an incoming datagram from a client.

Same syntax and parameters as on the client side.

## 4. sendto()

Sends a response back to the client.

Same syntax and parameters as on the client side.

Flow Explanation:

1. Server: Creates a socket (socket()) and binds (bind()) it to an IP and port.
2. Client: Creates a socket (socket()) and optionally connects (connect()).
3. Client: Sends a request using sendto(), which reaches the server.
4. Server: Receives the request using recvfrom(), processes it, and sends a reply using sendto().
5. Client: Receives the response using recvfrom(), processes it, and closes the socket (close()).

This follows a connectionless model since UDP does not establish a dedicated connection before data transfer.

DATE : 30/12/2024

**EXPERIMENT - 03**

**SOCKET PROGRAMMING USING TCP**

**AIM :**

Implement client-server communication using socket programming and TCP as Transport Layer Protocol

**ALGORITHM :**

SERVER :

1. Start
2. Include Required Header Files
3. Declare Variables:
   1. buff[100], k, len, sock\_desc, temp\_sock\_desc

○ struct sockaddr\_in server, client

1. Create Socket:
   1. sock\_desc = socket(AF\_INET, SOCK\_STREAM, 0) ○ If sock\_desc == -1 → Print "Error in socket creation"
2. Configure Server and Client Address:
   1. server.sin\_family = AF\_INET

○ server.sin\_addr.s\_addr = INADDR\_ANY

○ server.sin\_port = 3003

○ client.sin\_family = AF\_INET

○ client.sin\_port = 3003

1. Bind Socket:
   1. k = bind(sock\_desc, (struct sockaddr\*)&server, sizeof(server))

○ If k == -1 → Print "Error in binding"

1. Listen for Connections:

* 1. k = listen(sock\_desc, 5)

○ If k == -1 → Print "Error in listening"

1. Accept Client Connection:
   1. len = sizeof(client)

○ temp\_sock\_desc = accept(sock\_desc, (struct sockaddr\*)&client, &len)

○ If temp\_sock\_desc == -1 → Print "Error in temporary socket creation"

1. Receive Data from Client:
   1. k = recv(temp\_sock\_desc, buff, 100, 0)

○ If k == -1 → Print "Error in receiving"

1. Display Received Message:
   1. Print "Message got from client : " followed by buff
2. Close Client Socket:
   1. close(temp\_sock\_desc)
3. End

CLIENT :

1. Start
2. Include Required Header Files
3. Declare Variables:
   1. buff[100], k, sock\_desc ○ struct sockaddr\_in client
4. Create Socket:
   1. sock\_desc = socket(AF\_INET, SOCK\_STREAM, 0) ○ If sock\_desc == -1 → Print "Error in socket creation"
5. Configure Client Address:
   1. client.sin\_family = AF\_INET

○ client.sin\_port = 3003

○ client.sin\_addr.s\_addr = INADDR\_ANY

1. Connect to Server:
   1. k = connect(sock\_desc, (struct sockaddr\*)&client, sizeof(client))

○ If k == -1 → Print "Error in connecting to server"

1. Input Data to Send:
   1. Print "Enter the data to be sent:" ○ Use fgets(buff, 100, stdin) to read input
2. Send Data to Server:
   1. k = send(sock\_desc, buff, 100, 0)

○ If k == -1 → Print "Error in sending"

1. Close Socket:
   1. close(sock\_desc)
2. End

**PROGRAM :**

SERVER :

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <netinet/in.h> #include<unistd.h> int main()

{

    char buff[100];

    int k;

    socklen\_t len;

    int sock\_desc, temp\_sock\_desc;

    struct sockaddr\_in server, client;

    sock\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock\_desc == -1)

        printf("Error in socket creation!!!");

    server.sin\_family = AF\_INET;

    server.sin\_addr.s\_addr = INADDR\_ANY;

    server.sin\_port = 3003;

    client.sin\_family = AF\_INET;

    client.sin\_port = 3003;

    k = bind(sock\_desc, (struct sockaddr \*)&server, sizeof(server));

    if (k == -1)

        printf("Error in binding!!!");

    k = listen(sock\_desc, 5);

    if (k == -1)

        printf("Error in listening!!!");

    len = sizeof(client);

    temp\_sock\_desc = accept(sock\_desc, (struct sockaddr \*)&client, &len);

    if (temp\_sock\_desc == -1)

        printf("Error in temporary socket creation!!!");

    k = recv(temp\_sock\_desc, buff, 100, 0);

    if (k == -1)

        printf("Error in receiving!!!");

    printf("Message got from client : %s", buff);

    close(temp\_sock\_desc);

    return 0;

}

CLIENT :

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <netinet/in.h> #include<unistd.h> int main()

{

    char buff[100];

    int k;

    int sock\_desc;

    struct sockaddr\_in client;

    sock\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock\_desc == -1)

        printf("Error in socket creation!!!");

    client.sin\_family = AF\_INET;

    client.sin\_port = 3003;

    client.sin\_addr.s\_addr = INADDR\_ANY;

    k = connect(sock\_desc, (struct sockaddr \*)&client, sizeof(client));

    if (k == -1)

        printf("Error in connecting to server!!!");

    printf("\nEnter the data to be send : ");

    fgets(buff, 100, stdin);

    k = send(sock\_desc, buff, 100, 0);

    if (k == -1)

        printf("Error in sending!!!");

    close(sock\_desc);

    return 0;

}

**OUTPUT :**

**s6@m16**:**~/Desktop/MhdRayan/TCP**$ gcc client.c -o c

**s6@m16**:**~/Desktop/MhdRayan/TCP** $ ./c

Enter the data to be send : Hello I am Rayan

**s6@m16**:**~/Desktop/MhdRayan/TCP** $

**s6@m16**:**~/Desktop/MhdRayan/TCP** $ gcc server.c -o s

**s6@m16**:**~/Desktop/MhdRayan/TCP** $ ./s

Message got from client : Hello I am Rayan

**RESULT :**

The program is successfully executed and output is obtained.

DATE : 06/01/2025

**EXPERIMENT - 04**

**SOCKET PROGRAMMING USING UDP**

**AIM :**

Implement addition of two numbers using client-server communication using socket programming and UDP as Transport Layer Protocol.

**ALGORITHM :**

SERVER :

1. Start
2. Include Required Header Files
3. Define Constants:
   * SERVER\_PORT = 12345
   * BUFFER\_SIZE = 1024
4. Declare Variables:
   * sockfd (Socket descriptor)
   * struct sockaddr\_in server\_addr, client\_addr
   * socklen\_t addr\_len
   * ○ char buffer[BUFFER\_SIZE]
   * ○ int num1, num2, result
5. Create Socket:

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0) ○ If sockfd < 0 → Print "Socket creation failed" and exit

1. Initialize Server Structure:

memset(&server\_addr, 0, sizeof(server\_addr))

1. Configure Server Address:

* server\_addr.sin\_family = AF\_INET
* server\_addr.sin\_addr.s\_addr = INADDR\_ANY
* server\_addr.sin\_port = htons(SERVER\_PORT)

1. Bind Socket to Server Address:

* bind(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr))
* If binding fails → Print "Bind failed" and exit

1. Print Server Status:
   1. Print "Server listening on port 12345..."
2. Infinite Loop to Handle Client Requests: ● Loop:

* Receive data using recvfrom()
* If recv\_len < 0 → Print "Recvfrom failed" and continue
* Add null terminator to received buffer: buffer[recv\_len] = '\0' ○ If sscanf(buffer, "%d %d", &num1, &num2) == 2

Calculate result: result = num1 + num2

Print received numbers and their sum

Else → Print "Invalid input. Please send two integers."

1. Close the Socket:
2. close(sockfd)
3. End

CLIENT :

1. Start

2. Include Required Header Files

3. Define Constants

* SERVER\_PORT = 12345
* BUFFER\_SIZE = 1024

4. Declare Variables

* sockfd (Socket descriptor)
* struct sockaddr\_in server\_addr
* char buffer[BUFFER\_SIZE]
* int num1, num2

5. Create Socket

* sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)
* If sockfd < 0 → Print "Socket creation failed" and exit

6. Initialize Server Structure

* memset(&server\_addr, 0, sizeof(server\_addr))

7. Configure Server Address

* server\_addr.sin\_family = AF\_INET
* server\_addr.sin\_port = htons(SERVER\_PORT)
* server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1")

8. Get User Input

* Print "Enter the first number: " → Read num1
* Print "Enter the second number: " → Read num2

9. Format Input Data

* snprintf(buffer, sizeof(buffer), "%d %d", num1, num2)

10. Send Data to Server

* Use sendto() to send data to the server
* If sendto() fails → Print "Sendto failed" and exit

11. Receive Response from Server

* recv\_len = recvfrom()
* If recv\_len < 0 → Print "Recvfrom failed" and exit

12. Add Null Terminator to Received Data

* buffer[recv\_len] = '\0'

13. Close the Socket

14. End

**PROGRAM :**

SERVER :

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netdb.h>

int main(int argc, char \*argv[])

{

    struct sockaddr\_in server, client;

    if (argc != 2)

        printf("input format not correct");

    int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

    if (sockfd == -1)

        printf("Error in socket();");

    server.sin\_family = AF\_INET;

    server.sin\_addr.s\_addr = INADDR\_ANY;

    server.sin\_port = htons(atoi(argv[1]));

    if (bind(sockfd, (struct sockaddr \*)&server, sizeof(server)) < 0)

        printf("Error in blind()!\n");

    char buffer[100];

    socklen\_t server\_len = sizeof(server);

    printf("server waiting.......");

    if (recvfrom(sockfd, buffer, 100, 0, (struct sockaddr \*)&server, &server\_len) < 0)

        printf("Error in recufrom()!");

    printf("\nGot a datagram :%s", buffer);

    int first = atoi(buffer);

    if (recvfrom(sockfd, buffer, 100, 0, (struct sockaddr \*)&server, &server\_len) < 0)

        printf("Error in recufrom()!");

    printf("Got a datagram :%s", buffer);

    int second = atoi(buffer);

    int k = first + second;

    sprintf(buffer, "%d", k);

    if (sendto(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr \*)&server, sizeof(server)) < 0)

        printf("error in sending");

    printf("sum is :%d", k);

    return 0;

}

CLIENT :

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netdb.h>

int main(int argc, char \*argv[])

{

    struct sockaddr\_in server, client;

    if (argc != 3)

        printf("Input format not correct");

    int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

    if (sockfd == -1)

        printf("Error in socket();");

    server.sin\_family = AF\_INET;

    server.sin\_addr.s\_addr = INADDR\_ANY;

    server.sin\_port = htons(atoi(argv[2]));

    socklen\_t server\_len = sizeof(server);

    char buffer[100];

    printf("Enter the message to send to server");

    fgets(buffer, 100, stdin);

    if (sendto(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr \*)&server, sizeof(server)) < 0)

        printf("error in sending");

    printf("Enter the second number to be sent to server");

    fgets(buffer, 100, stdin);

    if (sendto(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr \*)&server, sizeof(server)) < 0)

        printf("error in sending");

    if (recvfrom(sockfd, buffer, 100, 0, (struct sockaddr \*)&server, &server\_len) < 0)

        printf("Error in recufrom()!");

    printf("sum is %s", buffer);

    return 0;

}

**OUTPUT :**

**s6@m16:~/Desktop/Rayan/UDP$** gcc sever.c -o s

**s6@m16:~/Desktop/Rayan/UDP$**./s 5886

server waiting.......

Got a datagram : 10

Got a datagram : 20

sum is :30

**s6@m16:~/Desktop/Rayan/UDP$**

**s6@m16:~/Desktop/Rayan/UDP$** gcc client.c -o c

**s6@m16:~/Desktop/Rayan/UDP$** c localhost 5886

Enter the message to send to server 10

Enter the second number to be sent to server 20

**s6@m16:~/Desktop/Rayan/UDP$**

**RESULT :**

The program is successfully executed and output is obtained

DATE : 20/01/2025

**EXPERIMENT - 05**

**FILE TRANSFER PROTOCOL**

**AIM :**

Write a program to implement File Transfer Protocol

**ALGORITHM :**

SERVER :

1. Start

2. Create a Socket

* sock = socket(AF\_INET, SOCK\_STREAM, 0)
* If sock < 0 → Print "Socket creation failed" and exit

3. Configure Server Address

* server\_address.sin\_family = AF\_INET
* server\_address.sin\_port = 5000
* server\_address.sin\_addr.s\_addr = INADDR\_ANY

4. Bind the Socket

* status = bind(sock, (struct sockaddr \*)&server\_address, sizeof(struct sockaddr))
* If status < 0 → Print "Binding failed" and exit

5. Listen for Incoming Connections

* status = listen(sock, 5)
* If status < 0 → Print "Listening failed" and exit

6. Accept a Client Connection

* length = sizeof(client\_address)
* client = accept(sock, (struct sockaddr \*)&client\_address, &length)
* If client < 0 → Print "Couldn't establish connection" and exit

7. Receive File Name from Client

* recv(client, rev, 100, 0)
* Add null terminator: rev[100] = '\0'

8. Open the Requested File in Read Mode

* fp = fopen(rev, "r")

9. Read File Content and Send it to Client

* While fgets(temp, 100, fp) != NULL  
  → send(client, temp, 100, 0)

10. Display Success Message

* Print "File copied"
* Print "Message from file: " followed by the last read data (temp)

11. Close the File

* fclose(fp)

12. Close Socket Connections

* close(sock)
* close(client)

13. End

CLIENT

1. Start

2. Include Required Header Files

3. Declare Variables

* FILE \*fp (File pointer)
* int sock (Socket descriptor)
* struct sockaddr\_in client\_address (To store client address details)
* int length (For client structure size)
* int status (To check socket function return values)
* Character arrays:
  + name[100]
  + rcvsmsg[100]
  + fname[100]
  + rcvg[100]
  + temp[100]

4. Create a Socket

* sock = socket(AF\_INET, SOCK\_STREAM, 0)
* If sock < 0 → Print "Socket creation failed" and exit

5. Configure Client Address

* client\_address.sin\_family = AF\_INET
* client\_address.sin\_port = 5000
* client\_address.sin\_addr.s\_addr = INADDR\_ANY

6. Connect the Socket to the Server

* status = connect(sock, (struct sockaddr \*)&client\_address, sizeof(client\_address))
* If status < 0 → Print "Connection failed" and exit

7. Get User Input for File Names

* Print "Enter the existing file name: " and read name using scanf()
* Print "Enter the new file name: " and read fname using scanf()

8. Send the Requested File Name to the Server

* send(sock, name, 100, 0)

9. Open the New File in Write Mode

* fp = fopen(fname, "w")

10. Receive File Content from Server and Write to the New File

* While recv(sock, temp, 100, 0) is successful:
  + Print the received data → printf("%s", temp)
  + Write the received data to the new file → fprintf(fp, "%s", temp)

11. Close the Socket Connection

* close(sock)

12. Close the Newly Created File

* fclose(fp)

1. End

**PROGRAM :**

SERVER :

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

int main() {

    FILE \*fp;

    int sock, client;

    struct sockaddr\_in server\_address, client\_address;

    int length;

    char name[100], fileread[100], temp[100], fname[100], ch, file[100], rev[100];

    sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock < 0) {

        printf("Socket creation failed.");

        exit(0);

    }

    server\_address.sin\_family = AF\_INET;

    server\_address.sin\_port = 5000;

    server\_address.sin\_addr.s\_addr = INADDR\_ANY;

    int status = bind(sock, (struct sockaddr\*)&server\_address, sizeof(struct sockaddr));

    if (status < 0) {

        printf("Binding failed.");

        exit(0);

    }

    status = listen(sock, 5);

    if (status < 0) {

        printf("Listening failed.");

        exit(0);

    }

    length = sizeof(client\_address);

    client = accept(sock, (struct sockaddr\*)&client\_address, &length);

    if (client < 0) {

        printf("Couldn't establish connection.");

    }

    recv(client, rev, 100, 0);

    rev[100] = '\0';

    fp = fopen(rev, "r");

    while (fgets(temp, 100, fp) != NULL) {

        send(client, temp, 100, 0);

    }

    printf("File copied\n");

    printf("Message from file: %s", temp);

    fclose(fp);

    close(sock);

    close(client);

    return 0;

}

CLIENT :

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

int main() {

    FILE \*fp;

    int sock;

    struct sockaddr\_in client\_address;

    int status;

    char name[100], rcvmsg[100], fname[100], rcv[100], temp[100];

    sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock < 0) {

        printf("Socket creation failed.");

        exit(0);

    }

    client\_address.sin\_family = AF\_INET;

    client\_address.sin\_port = 5000;

    client\_address.sin\_addr.s\_addr = INADDR\_ANY;

    status = connect(sock, (struct sockaddr\*)&client\_address, sizeof(client\_address));

    if (status < 0) {

        printf("Connection failed.");

        exit(0);

    }

    printf("Enter the existing file name: ");

    scanf("%s", name);

    printf("Enter the new file name: ");

    scanf("%s", fname);

    send(sock, name, 100, 0);

    fp = fopen(fname, "w");

    while (recv(sock, temp, 100, 0)) {

        printf("%s", temp);

        fprintf(fp, "%s", temp);

    }

    close(sock);

    fclose(fp);

    return 0;

}

**OUTPUT :**

**s6@m16**:**~/Desktop/MhdRayan/FTP$** gcc server.c -o s

**s6@m16**:**~/Desktop/MhdRayan/FTP$** ./s

Enter the existing file name: c.txt

Enter the new file name: n.txt

**s6@m16**:**~/Desktop/MhdRayan/FTP$**

**s6@m16**:**~/Desktop/MhdRayan/FTP$** gcc ftpc.c -o c

**s6@m16**:**~/Desktop/MhdRayan/FTP$** ./c

File copied

Message from client : Hello

**s6@m16**:**~/Desktop/MhdRayan/FTP$**

**RESULT :**

The program is successfully executed and output is obtained.

DATE : 03/03/2025

**EXPERIMENT - 06**

**SIMPLE MAIL TRANSFER PROTOCOL**

**AIM**

Write a program to implement Simple Mail Transfer Protocol.

**ALGORITHM**

SERVER

1. Start
2. Include Required Header Files
   * Include necessary libraries: <stdio.h>, <stdlib.h>, <string.h>, <unistd.h>, <sys/socket.h>, <netinet/in.h>
3. Define check Function
   * This function accepts a string str and an integer value
   * If value < 0, print "<str> FAILED" to indicate an error
4. Initialize Variables
   * int sid, sbind, size, slisten, saccept, srec, ssend, i, n = 5
   * Buffers for data exchange: char recbuff[500], sendbuff[500], fromaddress[500], toaddress[500], mailbody[1000]
5. Socket Creation
   * sid = socket(AF\_INET, SOCK\_STREAM, 0)
   * Call check("SOCKET CREATION", sid) to verify socket creation
6. Configure Server Address Structure
   * saddr.sin\_family = AF\_INET
   * saddr.sin\_port = htons(8082)
   * saddr.sin\_addr.s\_addr = INADDR\_ANY
   * size = sizeof(saddr)
7. Bind Socket
   * sbind = bind(sid, (struct sockaddr\*)&saddr, size)
   * Call check("BINDING", sbind) to verify binding
8. Listen for Client Connections
   * slisten = listen(sid, 5)
   * Call check("LISTEN", slisten) to verify
9. Accept Client Connection
   * saccept = accept(sid, (struct sockaddr\*)&saddr, &size)
   * Call check("ACCEPT", saccept) to verify
10. Receive and Respond to Client's Initial Message
    * Receive data from the client: recv(saccept, recbuff, 500, 0)
    * Print "MESSAGE FROM CLIENT: <received message>"
    * Send response: "220 192.168.9.138"
11. Handle "HELLO" Message
    * Receive message
    * If message starts with "HELLO", respond with "250 OK"
12. Handle "MAIL FROM:" Message
    * Receive message
    * If message starts with "MAIL FROM:", respond with "250 OK"
13. Handle "MAIL TO:" Message
    * Receive message
    * If message starts with "MAIL TO:", respond with "250 OK"
14. Handle "DATA" Command
    * Receive "DATA" from the client
    * Respond with "354 Go AHEAD"
15. Receive Mail Body
    * Loop to receive multiple lines of the email body
    * Stop receiving when the body starts with "$"
16. Send Final Response
    * Send "221 OK" as the final acknowledgment
17. Handle "QUIT" Command
    * Receive "QUIT" command from the client
    * Send "221 OK" to confirm termination
18. Close Connections
    * Close client socket (close(saccept))
    * Close server socket (close(sid))
19. End

CLIENT

1. Start
2. Include Required Header Files
   * Include <stdio.h>, <string.h>, <stdlib.h>, <unistd.h>, <sys/socket.h>, and <netinet/in.h> for networking and I/O operations.
3. Define check Function
   * Accept a string str and an integer value.
   * If value < 0, print "<str> FAILED" indicating an error.
4. Initialize Variables
   * int cid, con, size, csend, crec, i; for socket operations and loops.
   * Buffers for data exchange:
     + char sendbuff[500], recbuff[500] for general messages.
     + char fromaddress[500], toaddress[500] for email addresses.
     + char mailbody[1000] for the message body.
5. Create a Socket
   * cid = socket(AF\_INET, SOCK\_STREAM, 0);
   * Call check("SOCKET CREATION", cid) to confirm successful creation.
6. Configure Client Address Structure
   * caddr.sin\_family = AF\_INET;
   * caddr.sin\_port = htons(8082);
   * caddr.sin\_addr.s\_addr = INADDR\_ANY;
   * size = sizeof(caddr);
7. Connect to the Server
   * con = connect(cid, (struct sockaddr\*)&caddr, size);
   * Call check("CONNECTION", con) to confirm successful connection.
8. Send Initial Greeting
   * Print "SENDING HI TO SERVER"
   * Copy "HI" into sendbuff
   * Send the message using send()
   * Wait for the server's response using recv()
   * Print "MESSAGE FROM SERVER: <received message>"
9. Send "HELLO" Message
   * Copy "HELLO" into sendbuff
   * Send the message using send()
   * Wait for and check the response
   * If the response is "250 OK", proceed; otherwise, print "OK NOT RECEIVED"
10. Send "MAIL FROM:" Message
    * Prompt the user to enter the sender's address.
    * Format the message as "MAIL FROM:<sender\_address>".
    * Send the message and verify the server's response.
11. Send "MAIL TO:" Message
    * Prompt the user to enter the recipient's address.
    * Format the message as "MAIL TO:<recipient\_address>".
    * Send the message and verify the server's response.
12. Send "DATA" Command
    * Send "DATA" using send()
    * Wait for the server's response ("354 Go AHEAD" expected)
    * If "354" is not received, print "OK NOT RECEIVED"
13. Send Mail Body
    * Prompt the user to enter the mail content.
    * Use a loop (for i = 0 to 5) to send multiple lines of text.
    * If the user enters "$", break the loop to indicate the end of the mail body.
14. Close Connection
    * Print "CONNECTION CLOSED"
    * Close the socket using close(cid).
15. End

### **PROGRAM**

SERVER

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/socket.h>

#include<netinet/in.h>

void check(char str[], int value) {

if(value < 0)

printf("%s FAILED\n", str);}

void main() {

int sid, sbind, size, slisten, saccept, srec, ssend, i, n = 5;

char recbuff[500], sendbuff[500], fromaddress[500], toaddress[500], mailbody[1000];

sid = socket(AF\_INET, SOCK\_STREAM, 0);

check("SOCKET CREATION", sid);

struct sockaddr\_in saddr;

saddr.sin\_family = AF\_INET;

saddr.sin\_port = htons(8082);

saddr.sin\_addr.s\_addr = INADDR\_ANY;

size = sizeof(saddr);

sbind = bind(sid, (struct sockaddr\*)&saddr, size);

check("BINDING", sbind);

slisten = listen(sid, 5);

check("LISTEN", slisten);

saccept = accept(sid, (struct sockaddr\*)&saddr, &size);

check("ACCEPT", saccept);

printf("\n......\n");

srec = recv(saccept, recbuff, 500, 0);

check("RECEIVE", srec);

printf("MESSAGE FROM CLIENT: %s\n", recbuff);

printf("SENDING RESPONSE FOR HI...\n");

strcpy(sendbuff, "220 192.168.9.138");

ssend = send(saccept, sendbuff, 500, 0);

check("SENDING", ssend);

srec = recv(saccept, recbuff, 500, 0);

check("RECEIVE", srec);

if (strncmp(recbuff, "HELLO", 5))

printf("ERROR\n");

else {

printf("MESSAGE FROM CLIENT: %s\n", recbuff);

printf("SENDING RESPONSE FOR HELLO...\n");

strcpy(sendbuff, "250 OK");

ssend = send(saccept, sendbuff, 500, 0);

check("SENDING", ssend); }

printf("WAITING FOR FROM ADDRESS\n");

srec = recv(saccept, fromaddress, 500, 0);

check("RECEIVE", srec);

if (strncmp(fromaddress, "MAIL FROM:", 10))

printf("From address error\n");

else {

printf("FROM ADDRESS: %s\n", fromaddress);

printf("SENDING OK TO FROM ADDRESS\n");

strcpy(sendbuff, "250 OK");

ssend = send(saccept, sendbuff, 500, 0);

check("SENDING", ssend); }

printf("WAITING FOR TO ADDRESS\n");

srec = recv(saccept, toaddress, 500, 0);

check("RECEIVE", srec);

if (strncmp(toaddress, "MAIL TO:", 8))

printf("To address error\n");

else {

printf("TO ADDRESS: %s\n", toaddress);

printf("SENDING OK TO TO ADDRESS\n");

strcpy(sendbuff, "250 OK");

ssend = send(saccept, sendbuff, 500, 0);

check("SENDING", ssend); }

printf("WAITING FOR DATA...\n");

srec = recv(saccept, recbuff, 500, 0);

check("RECEIVE", srec);

if (strncmp(recbuff, "DATA", 4))

printf("DATA NOT RECEIVED\n");

else {

printf("MESSAGE FROM CLIENT: %s\n", recbuff);

printf("SENDING OK TO CLIENT\n");

strcpy(sendbuff, "354 Go AHEAD");

ssend = send(saccept, sendbuff, 500, 0);

check("SENDING", ssend); }

printf("MAILBODY:\n");

for (i = 0; i < n; i++) {

srec = recv(saccept, mailbody, 1000, 0);

check("MAIL BODY RECEIVE", srec);

if (strncmp(mailbody, "$", 1) == 0)

break;

else

printf("%s", mailbody); }

strcpy(sendbuff, "221 OK");

ssend = send(saccept, sendbuff, 500, 0);

printf("\nSENDING OK TO CLIENT\n");

check("SENDING", ssend);

srec = recv(saccept, mailbody, 1000, 0);

if (strncmp(mailbody, "QUIT", 4) == 0) {

strcpy(sendbuff, "221 OK");

ssend = send(saccept, sendbuff, 500, 0);

printf("Sending 221 OK...\n"); }

printf("CONNECTION CLOSED\n");

close(saccept);

close(sid);}

CLIENT

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/socket.h>

#include<netinet/in.h>

void check(char str[], int value) {

if(value < 0)

printf("%s FAILED\n", str);}

void main() {

int cid, con, size, csend, crec, i;

char sendbuff[500], recbuff[500], fromaddress[500], toaddress[500], mailbody[1000];

cid = socket(AF\_INET, SOCK\_STREAM, 0);

check("SOCKET CREATION", cid);

struct sockaddr\_in caddr;

caddr.sin\_family = AF\_INET;

caddr.sin\_port = htons(8082);

caddr.sin\_addr.s\_addr = INADDR\_ANY;

size = sizeof(caddr);

con = connect(cid, (struct sockaddr\*)&caddr, size);

check("CONNECTION", con);

printf("\n<-->\n");

printf("SENDING HI TO SERVER\n");

strcpy(sendbuff, "HI");

csend = send(cid, sendbuff, 500, 0);

check("SENDING", csend);

printf("WAITING FOR SERVER RESPONSE..\n");

crec = recv(cid, recbuff, 500, 0);

check("RECEIVE", crec);

printf("MESSAGE FROM SERVER: %s\n", recbuff);

printf("SENDING HELLO TO SERVER...\n");

strcpy(sendbuff, "HELLO");

csend = send(cid, sendbuff, 500, 0);

check("SENDING", csend);

crec = recv(cid, recbuff, 500, 0);

check("RECEIVE", crec);

if (strncmp(recbuff, "250", 3))

printf("OK NOT RECEIVED\n");

else

printf("MESSAGE FROM SERVER: %s\n", recbuff);

printf("ENTER THE FROM ADDRESS: ");

scanf("%s", fromaddress);

strcpy(sendbuff, "MAIL FROM:");

strcat(sendbuff, fromaddress);

csend = send(cid, sendbuff, 500, 0);

check("SENDING", csend);

crec = recv(cid, recbuff, 500, 0);

check("RECEIVE", crec);

if (strncmp(recbuff, "250", 3))

printf("OK NOT RECEIVED\n");

printf("ENTER TO ADDRESS: ");

scanf("%s", toaddress);

strcpy(sendbuff, "MAIL TO:");

strcat(sendbuff, toaddress);

csend = send(cid, sendbuff, 500, 0);

check("SENDING", csend);

printf("SENDING DATA TO THE SERVER..\n");

strcpy(sendbuff, "DATA");

csend = send(cid, sendbuff, 500, 0);

check("SENDING", csend);

crec = recv(cid, recbuff, 500, 0);

check("RECEIVE", crec);

if (strncmp(recbuff, "354", 3))

printf("OK NOT RECEIVED\n");

printf("Enter mail body:\n");

for (i = 0; i < 5; i++) {

fgets(mailbody, sizeof(mailbody), stdin);

csend = send(cid, mailbody, 1000, 0);

if (strncmp(mailbody, "$", 1) == 0)

break;

check("SENDING", csend);

}

printf("CONNECTION CLOSED\n");

close(cid);

}

**OUTPUT**

**s6@m16:~/Desktop/MhdRayan/SMTP** gcc server.c -o s

**s6@m16:~/Desktop/MhdRayan/SMTP**./s

......................................

MESSAGE FROM CLIENT :HI

SENDING RESPONSE FOR HI

WAITING FOR RESPONSE FROM CLIENT

MESSAGE FROM CLIENT : HELLO

SENDING RESPONSE FOR HELLO

WAITING FOR FROM ADDRESS

FROM ADDRESS: MAIL FROM:rayan6203@3gmail.com

SENDING OK TO FROM ADDRESS

WAITING FOR TO ADDRESS

TO ADDRESS: MAIL TO:aj@gmail.com

SENDING OK TO TO ADDRESS

WAITING FOR DATA

DATA NOT RECEIVED

SENDING OK TO CLIENT

MAIL BODY:

Explain about your lab

SENDING OK TO CLIENT

Sending 221 okCONNECTION CLOSED

**s6@m16:~/Desktop/MhdRayan/SMTP$**

**s6@m16:~/Desktop/MhdRayan/SMTP$** gcc client.c -o c

**s6@m16:~/Desktop/MhdRayan/SMTP$**./c

.....................................

SENDING HI TO SERVER

WAITING FOR RESPONSE FROM SERVER

MESSAGE FROM SERVER :220 192.168.9.138

SENDING HELLO TO SERVER

WAITING FOR OK MESSAGE

MESSAGE FROM SERVER:250 OK

ENTER FROM ADDRESS:rayan@3gmail.com

WAITING FOR OK FROM SERVER

MESSAGE FROM SERVER:250 OK

ENTER TO ADDRESS:ajgmail.com

WAITING FOR OK FROM SERVER

MESSAGE FROM SERVER:250 OK

SENDING DATA TO THE SERVER

WAITING FOR OK FROM SERVER

MESSAGE FROM SERVER:354 GO AHEAD

Enter mail body

Explain about your lab

$

SENDING MAILBODY TO SERBER

WAITING OK FROM SERVER

MESSAGE FROM SERVER:221 OK

SENDING QUIT....

Existing...CONNECTION CLOSED

**s6@m16:~/Desktop/MhdRayan/SMTP**

**RESULT :**

The program is successfully executed and output is obtained.

DATE : 10/03/2025

**EXPERIMENT - 07**

**STOP AND WAIT PROTOCOL**

**AIM**

Implement a Stop-and-Wait ARQ (Automatic Repeat reQuest) protocol using socket programming in C.

**OBJECTIVE**

To understand a Stop-and-Wait ARQ (Automatic Repeat reQuest) protocol using socket programming in C.

**DEFINITION:**

A C program that demonstrates reliable data transmission using the Stop-and-Wait ARQ protocol in a client-server model, ensuring correct packet delivery with acknowledgment handling.

**ALGORITHM**

CLIENT :

1. Start
2. Initialize Variables
   * Declare sock\_desc (Socket descriptor).
   * Create buffers: buf[100], packet[3][100] (Packet data), and ack[100].
3. Create Socket
   * Call socket() to create a client socket.
   * If socket creation fails, print "Error in socket creation".
4. Configure Client
   * Set client properties (AF\_INET, INADDR\_ANY, Port 5651).
5. Connect to Server
   * Use connect() to establish a connection with the server.
   * If connection fails, print "Error".
6. Send Packets
   * For 3 iterations, perform the following:
     + Call send() to transmit the packet.
     + If sending fails, print "Error in sending".
     + Otherwise, print "Sending Packet i".
     + Wait for an acknowledgment (recv()).
     + If acknowledgment is "1", continue to the next packet.
     + Otherwise, resend the packet and print "Resending Packet i".
7. Print "Packet transmitted successfully"
8. Close the Socket
9. End

SERVER :

1. Start
2. Initialize Variables
   * Declare sock\_desc (Server socket) and temp\_sock\_desc (Temporary socket).
   * Create a buffer buf[100] and acknowledgment buffer ack[100].
3. Create Socket
   * Call socket() to create the server socket.
   * If socket creation fails, print "Error in socket creation".
4. Configure Server and Client
   * Set server and client properties (AF\_INET, INADDR\_ANY, Ports 5651 and 5656).
5. Bind Socket
   * Call bind() to bind the socket.
   * If binding fails, print "Error in binding".
6. Listen for Incoming Connections
   * Call listen() to start listening.
   * If listening fails, print "Error in listening".
7. Accept Connection
   * Use accept() to accept client connection.
   * If connection fails, print "Error in temporary socket creation".
8. Receive and Acknowledge Packets
   * For 3 iterations, perform the following:
     + Receive packet using recv().
     + If packet reception fails, print "Error in receiving packet"
     + Continuously prompt for "Packet received or not" until "1" is acknowledged.
     + Resend the acknowledgment until "1" is received.
9. Close the Socket
   * Use close(temp\_sock\_desc) to close the temporary socket.
10. End

### **PROGRAM**

CLIENT

#include<stdio.h>

#include<stdlib.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<unistd.h>

#include<string.h>

int sock\_desc;

void sendPacket(char packet[],int i)

{

    int k,m;

    char ack[100];

    k=send(sock\_desc,packet,100,0);

    if(k==-1)

        printf("Error in sending\n");

    else

        printf("sending packet %d\n",i+1);

    for(;;)

    {

        m=recv(sock\_desc,ack,100,0);

        printf("%s\n",ack);

        if(strcmp(ack,"1")==0)

            break;

        else

        {

            k=send(sock\_desc,packet,100,0);

            printf("Resending packet %d\n",i+1);

        }

    }

}

int main()

{

    char buf[100];

    int k;

    struct sockaddr\_in client;

    char packet[3][100]={"IAM BATMANNN.....","IAM THE SAVIOR OF GOTHAM","IAM WALTER WHITE"};

    sock\_desc=socket(AF\_INET,SOCK\_STREAM,0);

    if (sock\_desc==-1)

        printf("Error in socket creation ");

    client.sin\_family=AF\_INET;

    client.sin\_addr.s\_addr=INADDR\_ANY;

    client.sin\_port=5651;

    k=connect(sock\_desc,(struct sockaddr\*)&client,sizeof(client));

    if(k==-1)

        printf("Error in connecting to server \n");

    for(int i=0;i<3;i++)

    {

        sendPacket(packet[i],i);

    }

    printf("packets transmitted successfully\n");

    close (sock\_desc);

    return 0;

}

SERVER

#include<stdio.h>

#include<stdlib.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<unistd.h>

#include<string.h>

int sock\_desc,temp\_sock\_desc;

void recvPacket(int i)

{

    char buf[100],ack[100];

    int k;

    k=recv(temp\_sock\_desc,buf,100,0);

    if(k==-1)

        printf("Error in receiving packet %d\n",i+1);

    for(;;)

    {

        printf("Packet %d received or not:",i+1);

        scanf("%s",ack);

        ack[strlen(ack)]='\0';

        k=send(temp\_sock\_desc,ack,100,0);

        if(strcmp(ack,"1")==0)

            break;

        else

        {

            k=recv(temp\_sock\_desc,buf,100,0);

        }

    }

}

int main()

{

    char buf[100];

    char ack[100];

    int k,m;

    socklen\_t len;

    struct sockaddr\_in server,client;

    sock\_desc=socket(AF\_INET,SOCK\_STREAM,0);

    if (sock\_desc==-1)

        printf("Error in socket creation ");

    server.sin\_family=AF\_INET;

    server.sin\_addr.s\_addr=INADDR\_ANY;

    server.sin\_port=5651;

    client.sin\_family=AF\_INET;

    client.sin\_addr.s\_addr=INADDR\_ANY;

    client.sin\_port=5656;

    k=bind(sock\_desc,(struct sockaddr\*)&server,sizeof(server));

    if(k==-1)

        printf("Error in binding\n");

    k=listen(sock\_desc,5);

    if(k==-1)

        printf("Error in listening\n");

    len=sizeof(client);

    temp\_sock\_desc=accept(sock\_desc,(struct sockaddr\*)&client,&len);

    if(temp\_sock\_desc==-1)

        printf("Error in temperory socket creation");

    for(int i=0;i<3;i++)

    {

        recvPacket(i);

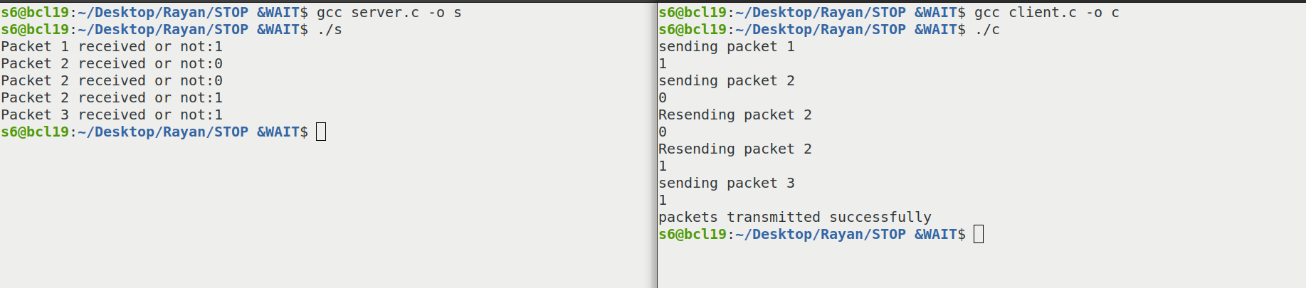
    }

    close(temp\_sock\_desc);

    return 0;

}

**OUTPUT**



**RESULT**

The program is successfully executed and output is obtained.

DATE : 11/02/2025

**EXPERIMENT - 08**

**DISTANCE VECTOR ROUTING**

**AIM**

To implement the Distance Vector Routing algorithm for network communication.

**OBJECTIVE**

To understand the functionality of the Distance Vector Routing algorithm in determining the shortest path between network nodes.

**DEFINITION:**

A C program that uses the Distance Vector Routing algorithm to compute and display the shortest paths and next-hop details for each router in a given network topology

**ALGORITHM**

1. Start
2. Input Number of Nodes
   * Read nodes (number of nodes in the network).
3. Input Cost Matrix
   * For each node i and j:
     + Read the cost from node i to node j.
     + If i == j, set costmatrix[i][j] = 0.
     + Initialize:

RT[i].distance[j] = costmatrix[i][j]

RT[i].from[j] = j

1. Distance Vector Calculation (Loop Until Stable)
   * Repeat until no updates are made (count == 0):
     + Set count = 0.
     + For each node i:
       - For each node j:
         * For each intermediate node k:

If RT[i].distance[j] > RT[i].distance[k] + RT[k].distance[j]:  
 RT[i].distance[j] = RT[i].distance[k] + RT[k].distance[j]

RT[i].from[j] = k

count++

1. Display Routing Tables
   * For each node i:
     + Print "For Router i"
     + For each node j:
       - Display "Node j via k distance d" where:
         * k = RT[i].from[j] (Next hop)
         * d = RT[i].distance[j] (Minimum distance)
2. End

### **PROGRAM**

#include<stdio.h>

#include<stdlib.h>

struct node{

    unsigned dist[20];

    unsigned from[20];

}rt[10];

void main(){

    int cost\_matrix[20][20];

    int nodes,i,j,k,count=0;

    printf("Enter the nodes : ");

    scanf("%d",&nodes);

    printf("Enter the Cost Matrix : ");

    for(i=0;i<nodes;i++){

        for(j=0;j<nodes;j++){

            scanf("%d",&cost\_matrix[i][j]);

            cost\_matrix[i][j] = 0;

            rt[i].dist[j] = cost\_matrix[i][j];

            rt[i].from[j] = j;

        }

    }

    do{

        count=0;

        for(i=0;i<nodes;i++){

            for(j=0;j<nodes;j++){

                for(k=0;k<nodes;k++){

                    if(rt[i].dist[j]>cost\_matrix[i][k]+rt[k].dist[j]){

                        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("For router %d",i+1);

        for(j=0;i<nodes;j++){

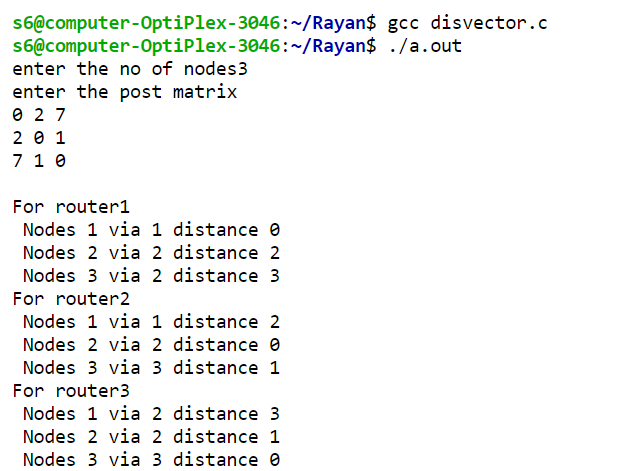
            printf("\t\n node %d via %d dist %d",j+1,rt[i].from[j+1],rt[i].dist[j]);

        }

    }

}

**OUTPUT**



**RESULT**

The program is successfully executed and output is obtain

DATE : 12/02/2025

**EXPERIMENT - 09**

**LEACKY BUCKET ALGORITHM**

**AIM**

To implement the Leaky Bucket algorithm for network traffic management.

**OBJECTIVE**

To understand the Leaky Bucket algorithm to regulate data flow and control congestion by managing incoming and outgoing packet rates.

**DEFINITION:**

A C program that demonstrates the Leaky Bucket algorithm, managing packet flow by tracking buffer size, incoming packets, and outgoing rates to prevent data loss due to buffer overflow.

**ALGORITHM**

1. Start
2. Input Values
   * Read the number of inputs n.
   * Read the bucket size bsize.
   * Read the outgoing packet rate out.
3. Initialize Storage
   * Set storage = 0.
4. Process Each Incoming Packet
   * For each packet i (from 1 to n):
     + Read the incoming packet size psize.
     + Calculate available space:  
        size\_left = bsize - storage
     + If Incoming Packet Fits in Bucket:
       - Add psize to storage.
     + Else (If Overflow Occurs):
       - Print the number of dropped packets:
       - Dropped = psize - size\_left
       - Set storage = bsize.
5. Display Buffer Status
   * Print "Bucket buffer size storage out of bsize".
6. Outgoing Packet Process
   * If (storage - out) < 0:
     + Set storage = 0.
   * Else:
     + Reduce storage by outgoing packets:
     + storage -= out
   * Print "After outgoing out packet, storage packets left in buffer".
7. Repeat Steps 4 to 6 for all packets.
8. End

### **PROGRAM**

#include<stdio.h>

int main()

{

    int n,storage=0,out,psize,bsize,size\_left;

    printf("Enter the No.of inputs:");

    scanf("%d",&n);

    printf("Enter the bucket size:");

    scanf("%d",&bsize);

    printf("Enter outgoing rate:");

    scanf("%d",&out);

    for(int i=0;i<n;i++)

    {

        printf("\nEnter incoming packet size:");

        scanf("%d",&psize);

        size\_left=bsize-storage;

        if(psize<=size\_left)

        {

            storage+=psize;

        }

        else

        {

            printf("Dropped %d no.of packets\n",psize-(bsize-storage));

            storage=bsize;

        }

        printf("\nBucket buffer size %d out of %d",storage,bsize);

        if(storage-out<0)

            storage=0;

        else

            storage-=out;

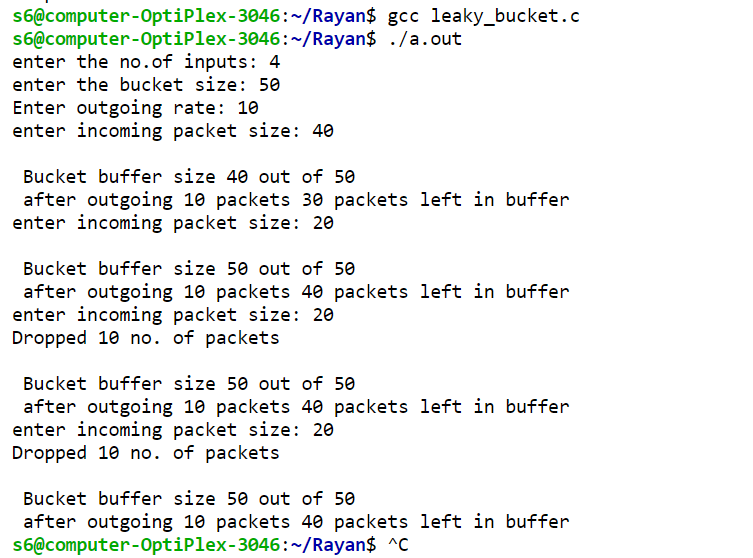
        printf("\nAfter outgoing %d packets %d packets left in buffer\n",out,storage);

    }

        return 0;

}

**OUTPUT**



**RESULT**

The program is successfully executed and output is obtained.

DATE : 01/04/2025

**EXPERIMENT - 10**

**STATIC ROUTING WITH SUBNETTING**

**Aim:**

An organization needs 500 subnet each with 100 usable host addresses. Design the network with class B. Communicate second & fourth subnet using static routing.

**Procedure:**

**Design:**

No: of subnet = 500

2n = 500

n = 9 i.e, No: of borrowed bits = 9

No: of host bit = 7

No: of host = 27=128.

**Subnets:**

172.27.0.0 – first subnet

172.27.0.128 – second subnet

172.27.1.0 – third subnet

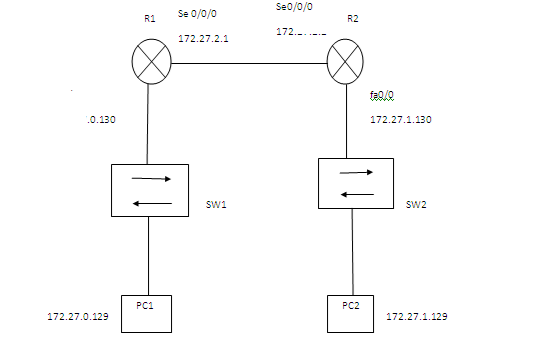
172.27.1.128 – fourth subnet

172.27.2.0 – fifth subnet

172.27.2.128 – sixth subnet

**Subnet mask**: 255.255.255.128

**Diagram:**



**Procedure :**

**Configuration:**

Routing table for router1

|  |  |
| --- | --- |
| Destination network address | Next hop address |
|  |  |
| 172.27.0.128 | Directly connected |
|  |  |
| 172.27.2.0 | Directly Connected |
|  |  |
| 172.27.1.28 | 172.27.2.2 |
|  |  |

Routing table for router2

|  |  |
| --- | --- |
| Destination network address | Next hop address |
|  |  |
| 172.27.0.128 | 172.27.2.1 |
|  |  |
| 172.27.2.0 | Directly Connected |
|  |  |
| 172.27.1.128 | Directly connected |
|  |  |

Configuring PC’s

|  |  |  |  |
| --- | --- | --- | --- |
| PC | IP address | Subnet Mask | Default Gateway |
|  |  |  |  |
| PC1 | 172.27.0.129 | 255.255.255.128 | 172.27.1.130 |
|  |  |  |  |
| PC2 | 172.27.1.129 | 255.255.255.129 | 172.27.0.130 |
|  |  |  |  |

**Router Configuration of R1**

Configuring Fast Ethernet in R1

Router>enable

Router # config terminal

Router (config) # hostname R1

R1(config) # int fast Ethernet 0/0

R1 (config –if) # ip address 172.27.0.130 255.255.255.128

R1 (config – if) # no shut

R1 (config – if) # exit

Configuring Serial interface in R1

R1(config) # int serial 0/0/0

R1 (config –if) # ip address 172.27.2.1 255.255.255.128

R1 (config –if) #clock rate 64000

R1nfig – if) # no shut

R1 (config – if) # exit

Configuring static routing in R1

R1 (config) # ip route 172.27.0.128 255.255.255.128 172.27.2.2

R1 (config) # exit

**Router Configuration of R2**

Configuring Fast Ethernet in R2

Router>enable

Router # config terminal

Router (config) # hostname R2

R2(config) # int fast Ethernet 0/0

R2 (config –if) # ip address 172.17.1.130 255.255.255.128

R2 (config – if) # no shut

R2 (config – if) # exit

Configuring Serial interface in R2

R2(config) # int serial 0/0/0

R2 (config –if) # ip address 172.27.2.2 255.255.255.128

R2 (config – if) # no shut

R2 (config – if) # exit

Configuring static routing in R2

R2 (config) # ip route 172.27.0.128 255.255.255.128 172.27.2.1

R2 (config) # exit

**Output :**

Devices are configured and successfully communicated.

DATE : 01/04/2025

**EXPERIMENT - 10**

**DYNAMIC ROUTING USING RIP**

**Aim:**

A subnet mask of 255.255.248.0 is given for a network 150.11.0.0. How many host & subnet are possible? Implement dynamic routing using RIP with two routers for any two subnets.

**Design :**

Network address 150.110.0.0 & Subnet mask 255.255.248.0

No: of bits borrowed = 5

No: of subnets = 25

No: of hosts = 211

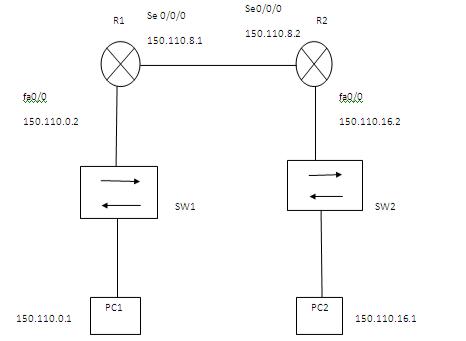
**Subnets:**

150.110.0.0 - first

150.110.8.0 - second

150.110.16.0 - third

**Diagram:**



**Procedure :**

**Configuration:**

Routing table for router1

|  |  |
| --- | --- |
| Destination network address | Next hop address |
|  |  |
| 150.110.0.0 | Directly connected |
|  |  |
| 150.110.8.0 | Directly Connected |
|  |  |
| 150.110.16.0 | 150.110.8.2 |
|  |  |
| Routing table for router2 |  |
|  |  |
| Destination network address | Next hop address |
|  |  |
| 150.110.0.0 | 150.110.8.1 |
|  |  |
| 150.110.8.0 | Directly Connected |
|  |  |
| 150.110.16.0 | Directly connected |
|  |  |

Configuring PC’s

|  |  |  |  |
| --- | --- | --- | --- |
| PC | IP address | Subnet Mask | Default Gateway |
|  |  |  |  |
| PC1 | 150.110.0.1 | 255.255.248.0 | 150.110.16.2 |
|  |  |  |  |
| PC2 | 150.110.16.1 | 255.255.248.0 | 150.110.0.2 |
|  |  |  |  |

**Router Configuration of R1**

Configuring Fast Ethernet in R1

Router>enable

Router # config terminal

Router (config) # hostname R1

R1(config) # int fast Ethernet 0/0

R1 (config –if) # ip address 150.110.0.2 255.255.248.0

R1 (config – if) # no shut

R1 (config – if) # exit

Configuring Serial interface in R1

R1(config) # int serial 0/0/0

R1 (config –if) # ip address 150.110.8.1 255.255.248.0

R1 (config –if) #clock rate 64000

R1 (config – if) # no shut

R1 (config – if) # exit

Configuring Dynamic routing in R1

R1 (config) # router RIP

R1 (config - router) # network 150.110.0.0

R1 (config - router) # network 150.110.8.0

R1 (config - router) # exit

**Router Configuration of R2**

Configuring Fast Ethernet in R2

Router>enable

Router # config terminal

Router (config) # hostname R2

R2(config) # int fast Ethernet 0/0

R2 (config –if) # ip address 150.110.16.2 255.255.248.0

R2 (config – if) # no shut

R2 (config – if) # exit

Configuring Serial interface in R2

R2(config) # int serial 0/0/0

R2 (config –if) # ip address 150.110.8.2 255.255.248.0

R2 (config – if) # no shut

R2 (config – if) # exit

Configuring Dynamic routing in R2

R2 (config) # router RIP

R2 (config - router) # network 150.110.16.0

R2 (config - router) # network 150.110.8.0

R2 (config - router) # exit

**Output :**

Devices are configured and successfully communicate