**Expt1 - Internetworking Basics(**Check folder for record pdf)

**Expt2 - Network Configuration Commands(**Check folder for record pdf)

**Expt3.1 - Two Way Chat Application using TCP**

SERVER:

Step 1: Start

Step 2: Create socket with socket() system call and bind it to IP address and

port number using bind().

Step 3: Listen for connection request from client with the listen () system call.

Step 4: Accept connection request with the accept () system call.

Step 5: Receive string from client using recv()

Step 7: Send the string to client using send()

Step 8: Stop

CLIENT:

Step 1: Start

Step 2: Create socket with socket() system call

Step 3: Send connection request to server using connect().

Step 4: Accept string from user

Step 5: Send string to server using send()

Step 6: Receive the string from server using recv()

Step 7: Print the string on the screen

Step 8: Stop

**Expt3.2 - Two Way Chat Application using UDP**

CLIENT:

1. Start: Begin the program execution.

2. Create Socket: Use the `socket()` system call to create a UDP socket.

3. Configure Server Address: Initialize the server address structure (`serverAddr`) with the server's IP address and port number.

4. Send Data to Server:

- Prompt the user to enter a message to send to the server.

- Use the `sendto()` function to send the message to the server.

5. Receive Response from Server:

- Use the `recvfrom()` function to receive a response from the server.

- Print the received message from the server.

6. Repeat: Repeat steps 4-5 indefinitely for continuous communication.

7. Close Socket: Use the `close()` function to close the socket when communication is complete.

8. Stop: End the program execution.

SERVER:

1. Start: Begin the program execution.

2. Create Socket: Use the `socket()` system call to create a UDP socket.

3. Configure Server Address: Initialize the server address structure (`serverAddr`) with the server's IP address and port number.

4. Bind Socket: Use the `bind()` function to bind the socket to the server's address and port number.

5. Receive Data from Client:

- Use the `recvfrom()` function to receive data from a client.

- Print the received message from the client.

6. Send Response to Client:

- Prompt the user to enter a message to send back to the client.

- Use the `sendto()` function to send the response to the client.

7. Repeat: Repeat steps 5-6 indefinitely for continuous communication.

8. Close Socket: Use the `close()` function to close the socket when communication is complete.

9. Stop: End the program execution.

**Expt3.3 - TCP Echo Client and TCP Echo Server**

TCP Echo server

Step 1: Start

Step 2: Create socket with socket() system call and bind it to IP address and port number using bind().

Step 3: Listen for connection request from client with the listen() system call.

Step 4: Accept connection request with the accept() system call.

Step 5: Receive string from client using recv()

Step 6: Send the received string back to client using send()

Step 7: Repeat steps 5-6 until the client closes the connection

Step 8: Stop

TCP Echo client

Step 1: Start

Step 2: Create socket with socket() system call

Step 3: Send connection request to server using connect().

Step 4: Accept string from user

Step 5: Send string to server using send()

Step 6: Receive the echoed string from server using recv()

Step 7: Print the echoed string on the screen

Step 8: Repeat steps 4-7 until the user decides to exit

Step 9: Stop

**Expt3.4 - UDP Echo Client and UDP Echo Server**

UDP Echo server

Step 1: Start

Step 2: Create socket with socket() system call and bind it to IP address and port number using bind().

Step 3: Receive datagram from client using recvfrom()

Step 4: Send the received datagram back to client using sendto()

Step 5: Repeat steps 3-4 until the server is terminated

Step 6: Stop

UDP Echo client

Step 1: Start

Step 2: Create socket with socket() system call

Step 3: Accept string from user

Step 4: Send string to server using sendto()

Step 5: Receive the echoed string from server using recvfrom()

Step 6: Print the echoed string on the screen

Step 7: Repeat steps 3-6 until the user decides to exit

Step 8: Stop

**Expt3.5 - FTP**

Ftp client

Step 1: Start

Step 2: Create socket with socket() system call

Step 3: Send connection request to server using connect()

Step 4: Receive confirmation of connection from server

Step 5: Send requested filename to server

Step 6: Receive file size from server

Step 7: Receive file data from server in chunks until the entire file is received

Step 8: Save the received file data to a file on the client side

Step 9: Print success message

Step 10: Close the socket

Step 11: Stop

FTP server

Step 1: Start

Step 2: Create socket with socket() system call and bind it to IP address and port number using bind()

Step 3: Listen for connection request from client with the listen() system call

Step 4: Accept connection request with the accept() system call

Step 5: Receive filename from client

Step 6: Open the requested file for reading

Step 7: If the file exists, send file size to client

Step 8: Send file data to client in chunks until the entire file is sent

Step 9: Close the file

Step 10: Close the socket

Step 11: Stop

**Expt3.6 - TCP Concurrent Server**

TCP concurremt server

Step 1: Start

Step 2: Create a server socket with socket() system call

Step 3: Set socket options with setsockopt() to enable address reuse

Step 4: Bind the server socket to IP address and port number with bind()

Step 5: Listen for connections with listen()

Step 6: Accept incoming connections from clients with accept() in a loop

Step 7: Create a new thread for each client connection

Step 8: In the new thread, handle communication with the client

Step 8.1: Receive data from the client with read()

Step 8.2: Process the received data

Step 8.3: Send response back to the client with write()

Step 9: Close the client socket in each thread when communication is done

Step 10: Close the server socket when no longer needed

Step 11: Stop

TCP concurrent client

Step 1: Start

Step 2: Create a client socket with socket() system call

Step 3: Configure server address (IP and port)

Step 4: Connect to the server with connect()

Step 5: Start a loop for communication

Step 5.1: Prompt user to enter message

Step 5.2: Read user input

Step 5.3: Send the message to the server with send()

Step 5.4: Receive response from the server with read()

Step 5.5: Display the server's response

Step 6: Close the client socket when communication is done

Step 7: Stop

**Expt3.7 - Broadcast**

Client

Step 1: Start

Step 2: Create a client socket with socket() system call specifying the domain as AF\_INET and type as SOCK\_DGRAM for UDP.

Step 3: Set socket options with setsockopt() to enable broadcast by setting SO\_BROADCAST option.

Step 4: Configure the server address (IP and port) to the broadcast address.

Step 5: Start a loop for communication.

Step 5.1: Prompt the user to enter a message.

Step 5.2: Read user input.

Step 5.3: Send the message to the server using sendto().

Step 5.4: Wait for a response from the server using recvfrom().

Step 5.5: Print the response received from the server.

Step 6: Close the client socket when communication is done.

Step 7: Stop

Server

Step 1: Start

Step 2: Create a server socket with socket() system call specifying the domain as AF\_INET and type as SOCK\_DGRAM for UDP.

Step 3: Set socket options with setsockopt() to enable broadcast by setting SO\_BROADCAST option.

Step 4: Bind the server socket to a specific port and network interface using bind() system call.

Step 5: Listen for incoming messages from clients in a loop.

Step 6: When a message is received, print the message.

Step 7: Broadcast the received message back to all clients by sending it to the broadcast address.

Step 8: Repeat steps 5-7 for each incoming message.

Step 9: Close the server socket when no longer needed.

Step 10: Stop

**Expt3.8 - TCP Iterative Server**

Client

Step 1: Start

Step 2: Create a client socket with socket() system call specifying the domain as AF\_INET and type as SOCK\_STREAM for TCP.

Step 3: Configure the server address (IP and port) to the address of the server you want to connect to.

Step 4: Connect to the server using connect() system call.

Step 5: Enter a loop to continuously send messages to the server and receive responses.

Step 5.1: Prompt the user to enter a message.

Step 5.2: Read user input.

Step 5.3: Send the message to the server using send() system call.

Step 5.4: Receive response from the server using recv() system call.

Step 5.5: Print the response received from the server.

Step 6: Continue the loop until the user decides to exit.

Step 7: Close the client socket when communication is done.

Step 8: Stop

Server

Step 1: Start

Step 2: Create a server socket with socket() system call specifying the domain as AF\_INET and type as SOCK\_STREAM for TCP.

Step 3: Bind the server socket to a specific port and network interface using bind() system call.

Step 4: Listen for incoming connections with listen() system call.

Step 5: Enter an infinite loop to continuously accept incoming connections.

Step 5.1: Accept an incoming connection from a client using accept() system call.

Step 5.2: Receive data from the client using recv() system call.

Step 5.3: Print the received data.

Step 5.4: Close the connection with the client using close() system call.

Step 6: Continue listening for new connections in the loop.

Step 7: Close the server socket when no longer needed.

Step 8: Stop