Experiment No-2

Socket Programming

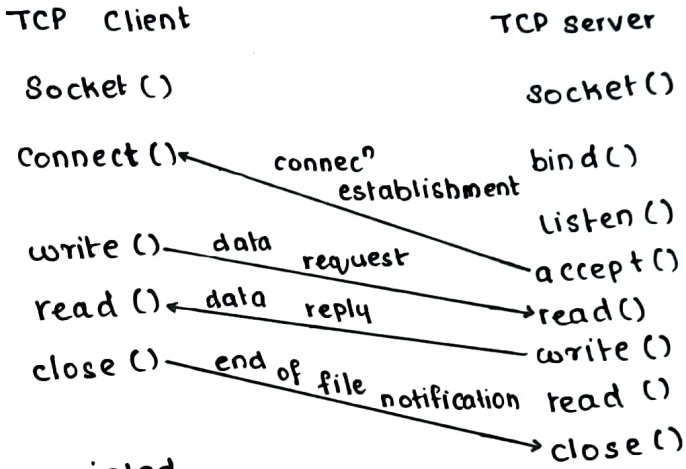
Date : 28/01/25

Aim: To familiarise with the basic system calls used for creating sockets and familiarise and transfering data b/w two nodes.

Description for socket programming.

Socket: a host-local, application-created, os-controlled interface into which application process can send and recieve messages to/from another application process.

Socket programming with TCP interaction Between Client & server



System Calls associated

1 Socket()

- it is used to create a socket

Syntax: Socket(int domain, Inl type, int protocol)

- domain: AF_INET (IPV4) or AF_INET6 (IPV6)
- type: sock_stream (TCP) or sock_DGRAM (UDP)
- protocol: usually 0 (default for type).

2 bind()

- Assume an address (IP and port) to the socket

Syntax:

bind(int sockfd, struct sockaddr *addr,)

socklen_t addrlen)

3 Listen()

- marks the socket as a passive socket to accept incoming connections

Syntax:

listen(int sockfd, int baklog)

4 Connect()

- Initiates a connection to a server

Syntax: connect(int sockfd, struct sockaddr *addr, socklen_t addrlen)

5 Send()

- Sends data over a connected socket(TCP)

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

6 sendto()

- Sends data over an UDP socket

Sendto(int sockfd, const void *buf, size_t len, int flags
const sockaddr *dest_addr, socklen_t addrlen);

7 accept()

- Accepts a connection from a client

accept(int sockfd, struct sockaddr * addr, socklen_t *addrlen

8 recv()

- recieves data from a connected socket(TCP)

Syntax: recv(int sockfd, void * buffer, size_t length, int flags)

9 recvfrom()

- recieves data from an UDP socket.

recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr
* src_addr, socklen_t *addrlen);

10 close()

- Close the socket and release sources

close(int sockfd)

RESULT

Familiarised with the Basic system calls used for creating sockets and transfering data between nodes.

Experiment-No-3

Maximum, Minimum, Average using Socket Programming (TCP)

Date : 28/01/25

Aim: To demonstrate a program showing the client server interaction by finding the maximum, minimum, and average of an average of an array of integer using socket programming.

Algorithm

→ Server side

1 Create a TCP server socket and bind it to 127.0.0.1 and port 5566
2 Listen for incoming connections using listen()
3 Accept a client connection using accept()
4 Recieve an array of integer from client
5 Compute the min, max & avg of the array using a single loop
6 Send the processed data (array, max, and min) back to client using send()
7 Close the client socket and continue listening for new connections using close()

→ Client side

1 Create a TCP client socket and connect to the server using socket() and connect().
2 Send an array of integer to the server using send()
3 Recieve the processed array, max and min values from server.
4 Display the recieved result.
5 Close the connection

RESULT

Send a list of integer to server via a client and successfully recieved their avg, max and min after server processing.

Experiment No-4

String reversal using single server and Two client (TCP)

Date : 28/01/25

Aim : To demonstrate the client-client communication via a single server by showing string reversal from one client to another.

Algorithm

→ Server side

1 Create a TCP socket using socket()
2 Configure the server address
3 Assign loopback IP and port no.
4 Bind the socket to the IP & port using bind()
5 listen for client connection.
6 Accept connection from client1 (sender)
7 Accept connection from client2 (reciever)
8 recieve the string from client2.
9 Reverse the string
10 Send the reversed string to client2
11 close connection (both client1 and client2) using close.

→ Client 1 (sender)

1 Create a TCP socket using socket()
2 Configure the server address
3 Assign loopback IP and port no.
4 Connect to the server using connect
5 Send the reversal string using string
6 Close the socket using close().

→ Client 2 (Reciever)

1 Create a TCP socket using socket()
2 Configure the server address
3 Assign loopback IP and Port no.
4 Connect to the server using connect()
5 recieve the reversal string using recv()
6 print the reversed string
7 close the socket using close

RESULT

The program set up a server that recieves a string from client1, reverses it, and sends the reversal string to client2, which then prints the reversed string before all process terminates.

Experiment No-8

Experiment-5

Squaring Integers using Single Server and Two client (TCP)

Date: 28/01/25

Aim: To demonstrate the client-client communication via a single server by means of integer squaring.

Algorithm

→ Client 1 (sender)

1 Create a socket using socket()
2 Specify server details (IP addr 127.0.0.1, port 5566)
3 Connect to the server using connect()
4 Send an integer to the server using send()
5 Close the connection using close()

→ Client 2 (reciever)

1 Create a socket using socket (AF_net, sock_stream, 0)
2 Specify server details (IP addr & port no.)
3 Connect to the server using connect
4 recieve the squared integer from the server using recv()
5 print the recieved result.
6 Close the connection using close()

→ Server

1 Create a socket using socket (AF_NET, sock_stream, 0)
2 Bind the socket to a specific IP (127.0.0.1) and port (5566)
3 Listen for incoming connections using listen()
4 Accept client 1's connection recieves an integer from it using recv()
5 Compute the square of the recieved no.
6 Accept client 2's connection, send the squared result.
7 close both client connections

RESULT

An integer is send to the server via a client 1 the server squares the integer and forwards the result to client 2, which then prints the squared value Before all connections are closed.

Experiment - 6

Scaling a number using single server & two clients (TCP)

Date : 28/1/25

Aim: To implement a socket programming system, where a client sends a float value to the server, which increases it by a power of 1.5 and forward it client 2 for display.

Algorithm

→ Client 1 (Sender)

1 create a TCP socket using socket()
2 define the server IP and port
3 connect to the server using connect()
4 send a floating number to the server using send()
5 Close the socket & disconnect.

→ Server

1 Create a TCP socket using socket()
2 Bind the socket to the IP and port
3 Listen for client connection.
4 Accept connections from client 1 & client 2
5 Recieve a floating point number from Client 2
6 Compute the new value using formula

$$y = x + x(1.5)$$

7 print the recieved no. and the processed result.
8 Send the computed result to client 2
9 Close the connection.

→ Client 2 (Reciever)

1 Create a TCP socket using socket()
2 Define the server IP and port
3 connect to the server
4 Recieve the processed floating no. from the server
5 print the recieved no.
6 Close the socket & disconnect

RESULT

Successfully established a TCP connection between 2 client via a server, where client 1 sends a floating no., the server process it by using $y = x + x(1.5)$ and client 2 recieves computed result.

Experiment-7

Mulli-User Chat Server (TCP)

Date: 28/01/25

Aim: To implement a multi-user chat server using TCP as a transport layer Protocol.

Algorithm

→ Server

1 Initialise the server socket by creating the socket using socket() and defining address and port.
2 Run an infinite loop to accept client conditions.
3 When a client i connects retrieve its socket descriptor using accept().
4 Store the clients socket in client[] array.
5 Create seperate thread for each client handling using pthread_create().
6 if a client disconnects, its socket is removed from clients[] list.
7 Remain in the loop to accept new client and create threads for them.
8 once finished close the server using close().

→ Client

1 Create a socket using socket()
2 define servers IP address & port no.
3 Connect to server using connect()
4 Create a seperate thread using pthread_create() to handle incoming msg.
5 Using recieve_message() function, continously checks for the msg from server and point them.
6 Read the user input using fgets()
7 Send to server using send()
8 loop 7 & 8 indefinitly allowing client
9 close the socket using close() once finished
10 server removes client from list.

RESULT

Successfully demonstrated real-time msg. exchange between multiple chats using TCP, with proper sychronizalion messages brodcasting & client disconnection handling.

Experiment No-8

'Ping' & 'Traceroute'

Date:

Aim: To understand how network packets travel from a source to a destination by using 'ping' to measure RTT and trace route to analyse the path takes by packets across multiple network hops.

Theory:

1 Ping:

- Ping is a network diaganostic tool used to check the reachability of a host and measure the time taken for the packet to travel between the source end destination.
  - key outputs:
  - → Packet loss: indicates network congestion of host unavailable.
  - → Round trip time: Time taken for a packet to travel to the destination & back

Syntax:

Ping <hostname/ip>

2 Trace route

- 'trace route' help analyze the path packets take from the source to the destination by showing intermideate hops
- it sends packets with incrementing TTL (Time To Live) value to trace the route
- if there is no response within a certain timeout, an "*" (astrisk) is printed for that probe.

Syntax:

traceroute <hostname/ip>

RESULT

Successfully used ping and tranceroute to analyse network connectivity and routing path.

Experiment - No. 9

"Whois" Command

Date:

Aim: To use the 'whois' utility to query the internet registries and determine

(a) The IP address allocated to the student's network

(b) The internet service provider of students network

(c) Major ISP's in the region.

Theory

- Whois is a network utility tool used to query database of domain name registers and IP address allocation.
- it helps retrieve info on IP, domain name, autonomous system
  - → Provide details like
    - The organisa° / ISP to manage IP addr / domain
    - IP allocation (CIDR)
    - Contact info for domain owners
    - geographical locations of network
- How whois work?

  when a user executes the whois command, it queries a set of RIRS (Regional Internet Registers) responsible for managing IP address
- Usage

  (i) Getting info on Domain name.

  → whois <Domain-name>

  (ii) Getting info about IP

  → Whois < IP addr>

  (iii) Some specific whois server set up by ICANN

  → whois -h <whois server> < domain-name>

  (iv) domain name from spesific region

  → whois -h < whois_server> <TLD / Domain>

RESULT

Successfully used the whois tool to retrieve domain and IP details, identified the systems Public IP and ISP, detected major ISPs like Reliance Jio, and proved useful for cyber security and networking.

Experiment No-10

ARP Command

Date:

Aim: To find the hardware / MAC address of a robber computer on the network, using the ARP (Address Resolution Protocol) command.

Theory

- ARP is used to map IP to MAC address within a local network
- Every device in the network has an unique Mac addr assigned to it NIC
- The ARP command allows user to view and manipulate the ARP cache which stores IP-to-MAC mapping of recently communicated device
- By using arp-a we can list all available MAC addr in the network.

Observation

- There is an entry for the IP addr 192.168.160.96
- [ether] → it is an ethernet (MAC) address.
- It is a wireless network interface

Result

Successfully retreived the MAC address of another computer in the network using ARP commands.

Experiment No:-11

Leaky Bucket Algorithm for Congestion control

Date: 11/03/25

Aim: To implement the Leaky Bucket Algorithm for congestion control in a network so that packet transmission at a regulated rates, are ensured.

Algorithm

1 Initialize a queue (q[]) to store packets, with front (f) and rear (r) pointers set to -1.

2 Enqueue all the incoming packets into the queue.
- if the, queue is full, reject the packet.
- otherwise, insert the packet at the rear of the queue

3 Process the packets using the Leaky Bucket mechanism.
- set the counter (n) representing the bucket transmission capacity per clock tick.
- Initialize a count variable to track the clock ticks.
- while the queue is not empty.
  - if the available capacity (temp) is sufficient to accomodate the front packet, send it and update the capacity
- otherwise reset the available capacity temp in and increment the clock ticks (counts).

4 Print the results indicating the packets and the total ticks required to transmit all packets.

RESULT

The Leaky Bucket Algorithm was successfully implements, ensuring controlled transmission of packets in the network.

Experiment No:-12

Routing Algorithm

Date: 11/03/25

Aim: To implement the distance vector routing algorithm and Linkstate routing algorithm in order to find the shortest distance from a given node to others.

→ Algorithm

Distance Vector

1 Initialize the routing table
   - Each node maintaining a table with distances to all other nodes.
   - if a direct connection exists, set the distances otherwise set it to large value.
   - The next hop for each node is initially set to itself.

2 Update the routing table.
   - Each node shares its distances information with it's neighbour
   - if a shorter path is found via an intermediate node, update the table.
   - Repeat this process - until no further updates occur.

3 Print the final - routing table for each node.

Link State

1 Initialize distances and visted nodes.
   - set the node of existence all to infinity except the source node(0)
   - Mark all nodes as unvisited.

2 Find the node with minimum distance
   - Select the unvisited node with smallest known distance

3 Update distances value
   - For each neighBour of the selected node, update the distance if a shorter path is found.

4 Repeat until all nodes are visited

5 Print the routing table for source node

RESULT

The distance vector Routing Algorithm has been successfully implemented using Bellman - ford - Algorithm and the link state routing algorithm has been successfully implemented using Dijkstra's Algorithm.

Experiment No :- 13

Ethtool Command

Date:

Aim: To use network tools like ethtool to observe and analyse link layer packet satisfies and errors

Theory

Linux provides various networking tools for link-layer analysis, with ethtool being a powerful utility for querying and modifying ethernet device setting. It can retreive detailed interface satisfies, such as.

→ Tx (Transmitter) and Rx (Reciever) packets: No. of packets transmitted and recieved.

→ Dropped packets: packets discarded due to buffer overflow/congestion/error.

→ CRC Errors :- Errors caused by data corruption during transmission.

→ Collisions : Instances where two devices attempt to send data simultaneously on a half-duplex network.

→ Frame Errors : Issues related to frame formatting, often due to mismatched configurations / faulty hardware.

By analysing these statistics network adminstration can identify & resolve network issues, optimistize performance and ensure data integerity

Commands

(i) ethtool -s <interface_name>

- provides detailed statistics such as transmitted and recieved. packets, dropped packets and errors.

(ii) ethtool <interface_name>

- display the current link status, supported link speed and duplex setting.

(iii) watch -n 2 ethtool -s <interface_names>

- updates the statistics every 2 sec. to observe real-time changes in packet transmission & errors.

(iv) ethtool -i <interface_name>

- provides information about the network interface driven useful for diagnosing hardware compatibility issues.

Observation

- The Rx and Tx packets count increase over time as network traffic flows through the interface.
- Dropped packets indicate potential congestion / buffer issues
- CRC and frame errors suggests possible cable faults or mismatched configurations
- Collisions mainly occur in half duplex network leading to re-transmission.

RESULT

Successfully observed and analyzed link layer packets statistics and errors using ethtools.

Experiment No:-14

Stop and wait Protocol

Date:

Aim: To implement stop and wait ARQ flow control protocol.

Algorithm

1 Begin
2 declare variables fr, ACK
    // fr - no. of frames
    // ACK - acknowledgment status
3 read fr
4 for i=0 to i=fr-1 loop
5 do
6 print "sending frame i+1"
7 sleep(1)
8 generate random acknowledgments ack = rand()%2
9 if ACK=1
        print "success full acknowledgement"
10 else
        print "negative acknowledgment"
11 i-- // resending frame with NACK
12 end loop
13 end.

RESULT

Successfully implemented stop and wait ARQ flow control protocol.

Experiment NO-15

Go-Back-N Protocol

Date:

Aim: To implement Go-Back-N ARQ flow control Protocol

Algorithm

1 Begin
2 declare variables
    // fr - total no. of frames
    // wi - window size
    // start=0, ack (acknowledgment status)
3 read fr, wi
4 while start < fr, do
5 loop send frames from start to start + wi, ensuring it doesn't exceed fr.
6 sleep(1)
7 ack = rand() % (wi+1) // random acknowledgment
8 if ack > 0, then
9 Acknowledgment recieved for first 'ack' frames
10 set start = start + wi
11 else (is ack=0)
12 resend the entire window starting from first acknowledged frames.
13 end loop
14 loop

RESULT

Successfully implemented Go-back-N ARQ flow control protocol.

Experiment No-16

Selective Repeat Protocol

Date:

Aim: To implemented selective repeat ARQ flow control protocol

Algorithm

1 Begin
2 declare variable, fr, wi, start = 0
3 declare array ack[fr]
4 read fr, wi
5 for i from 0 to fr-1
6 ack[i] = rand() % 2
7 end for loop
8 while start < fr do.
9 send frames from 'start' to 'start+wi', ensuring it doesn't exeed 'fr'
10 sleep(1)
11 for each frames in window do
12 if ack=1, print "success"
13 else
14 sleep(1)
15 ack[i]=1
16 print "success Acknowledgment after resending"
17 end loop
18 if all frames in window are acknowled ged
(ack[i]=1 for all frames in start → start+wi)
19 update start = start+wi
20 end loop
21 end.

RESULT

Successfully implemented selective repeat ARQ flow control protocol.

Experiment No-17

Maximum, Minimum, Average using Socket Programming (UDP)

Date:

Aim: To demonstrate a program showing the client server interaction. by finding the maximum, minimum and average of an array of integer using UDP socket programming.

Algorithm

1 Begin
2 define sock addr_in structure for server (ser) and client (cli)
3 declare array num[10], sum=10, port = 5566
4 create UDP socket using socket()
5 Bind the socket to the port using bind() after setting server adress properties (AF_INET, port, INADDR_ANY)
6 if binding fails prints an error and exit.
7 while (1) do
8 addr_size = sizeof(cli)
9 use recvfrom() to recieve an array of 10 integer from client.
10 set min = num[0], max = num[0] sum = 0.
11 iterate through array to compile.
12 sum of all elements min (smallest), max (largest)
13 end loop
14 avg = sum/10.0
15 use sendto() to send results back to the client
16 client()
17 end.

→ Client side

1 Begin
2 Set port = 5566, arr[10] = {1, 2, ····10} min, max avg
3 Create UDP socket using socket()
4 define server applica^{n} properties
5 use sendto() to send the 'arr' array to server
6 use recvfrom() to recieve max, min, avg.
7 close the socket
8 end

RESULT

Successfully demonstrated client-server interaction by finding the max, min & avg of an array of integer using UDP.

Experiment No-18

String Reversed using Single Server And two client (UDP)

Date:

Aim: To demonstrate the client class communication via a single server by showing string reversal from one client to another.

Algorithm

→ Server side

1 Begin
2 Create a UDP socket using socket() system call
3 Configure the sockaddr_in structure

sinfamily = AF_INET
sin_port = htons (5566)
sin_addr.sin_addr.s_addr = INADDR_ANY

4 use bind to associate the socket with the specified port
5 use recvfrom() to recieve data from client1
6 reverse the string recieved (x)
7 set i=0 u = strlen(x) - 1
8 swap x[i] and x[j] until they meet
9 use recvfrom() to recive 'ready' msg from client2
10 use sendto() to send the modified string to client2
11 close() // socket
12 end

→ Client 2 side (server)

1 Begin, create socket using socket()
2 configure sock_addr_in structure with

sin_family = AF_INET
sin_port = htons(5566)
sin_addr.s_addr = 127.0.0.1

3 send a string "Hello world" to the server using sendto()
4 print "msg sent"
5 close (server_sock)
6 end.

→ Client 2 side (Recievер)

1 Begin
2 Create an UDP socket using socket()
3 Configure sock_addr_in as in client 2.
4 notify the server that it is ready, using sendto()
5 use recvfrom() to recieve the reversed string from the server.
6 display reversed string.
7 close (server_sock)
8 end

RESULT

Success fully demonstrated the client-client communication via single server by showing string reversal from one client to another.

Experiment No-19

Squaring Integer using Single Server And Two Client (UDP)

Date:

Aim: To demonstrate the client-client communication via a single server by means of integer squaring using UDP socket.

Algorithm

→ Server side

1 Begin
2 Create a UDP socket using socket() system call.
3 Configure the sock_addr_in structure;

```
sin_family = AF_INET
sin_port = htons(5566)
sin_addr.s_addr = INADDR_ANY
```

4 Use bind() to associate the socket with the specified port.
5 use recvfrom() to recieve data from client1
6 Square the integer recieved(y)
7 ans = y*y
8 use recvfrom() to recieve a 'ready' msg from client2
9 use sendto() to send the squared integer to client2
10 close() // socket
11 end.

→ Client 1 side (sender)

1 Begin
2 Create socket using socket()
3 Configure sock_addr_in structure with

```
sin_family = AF_INET
sin_port = htons(5566)
sin_addr.s_addr = 127.0.0.1
```

4 Send an integer (y=8) to the server using sendto()
5 print "msg sent"
6 close (server_sock)
7 end

→ Client 2 side (Reciever)

1 Begin
2 Create an UDP socket using socket()
3 Configure sock_addr_in as in client1
4 Notify the server that it is ready using sendto()
5 use recvfrom() to recieve the squared_integer from the server.
6 display the squared integer.
7 close (server_sock)
8 end

RESULT

Successfully demonstrated the client-client communication via a single server by means of integer squaring using UDP sockets.

Experiment No-20

Scaling a Number Using Single Server & Two Clients (UDP)

Date:

Aim: To demonstrate the client-client communication via a single server by means of scaling an integer using UDP sockets.

Algorithm

→ Server side

1 Begin
2 Create a UDP socket using socket() system call.
3 Configure the sock_addr_in structure
    sin_family = AF_INET
    sin_port = htons (5566)
    sin_addr.s_addr = INADDR_ANY
4 Use Bind() to associate the socket with specified port.
5 use recvfrom() to recieve the data from client1
6 ans = 4 + pow(4, 1.5)
7 use recvfrom() to recieve a 'really' msg from client2
8 use sendto to send the socket no. 'ans' to client2
9 close()
10 end.

→ Client 1 side (sender)

1 Begin
2 Create an UDP socket using socket()
3 Configure sock_addr_in structure with
    sin_family = AF_INET
    sin_port = htons (5566)
    sin_addr.s_addr = 127.0.0.1
4 Send a no. x=7 to the server using sendto()
5 print 'Date sent'
6 close (Server_sock)
7 end.

→ Client 2 (Reiever)

1 Begin
2 Create a UDP socket using socket().
3 Configure sock_addr as in client1.
4 notify the server that it is ready, using sendto().
5 use recvfrom() to recive the scaled no. from the server.
6 display scaled no.
7 close (server_sock)
8 end.

RESULT

Success fully demonstrated the client-client communication via a single server by showing number scaling from 1 client to another.

Experiment No- 21

Concurrent Time server using UDP

Date:

Aim: To implement a concurrent time server application using UDP to execute the program at a remote server.

Algorithm

→ Server- Side

1 Begin
2 Creak a UDP socket using socket() sys·call
3 Configure the sock-addr-in structure
    sin_family = AF_INET
    sin_port = htons (5566)
    sin_addr.s_addr = INADDR_ANY
4 use bind() to associate the socket with specified port.
5 declare a sock_addr_in structure to store the client's address
6 use recvfrom() to recieve a "Time request" msg from client.
7 print a confirmation msg.
8 use time (&t) to set current time
9 Convert it to human readble format using local time (&t)
10 Format the time using strftime(t)
11 use sendto() to send the formatted time string back to client.
12 close (sever_sock)
13 END

→ Client

1 Begin
2 create a UDP socket using socket()
3 Configure sock_addr_in structure
    sin_family = AF_INET
    sin_port = htons(5566)
    sinddr.saddr = inet_addr (127.0.0.1)
4 use sendto() call to demand a request for the current time.
5 use recvfrom() to recive the response from server
6 print recieved time
7 close (server_sock)

RESULT

Successfully implemented a concurrent time server application using UDP to execute the program at a remote server.