***Question** :*

Simple Client Server Program

***Aim** :*

To establish a client server communication (TCP) using UNIX Socket Programming.

***Algorithm** :*

a. *Server -*

   i. Create a socket, initialize server details and bind the server with the socket.

   ii. Listen for a client request.

   iii. Accept the client's request to connect.

   iv. Read the client's message and display it on the standard output.

   v. Close the connection.

b. *Client -*

   i. Create a socket and initialize server details.

   ii. Make a request to connect with any available server.

   iii. Once connected successfully, read message from user and send it to the server.

   iv. Close the connection.

Learning Outcome :

***Question** :*

Echo Client Server Program

***Aim** :*

To develop a socket program to simulate an Echo Server.

***Algorithm** :*

a. *Server -*

   i. Create a socket, initialize server details and bind the server with the socket.

   ii. Listen for a client request.

   iii. Accept the client's request to connect.

   iv. Read the message and send it back to the client.

   v. Close the connection.

b. *Client -*

   i. Create a socket and initialize server details.

   ii. Make a request to connect with any available server.

   iii. Once connected successfully, read message from user and send it to the server.

   iv. Read the message echoed by the server and display it on the standard output.

   v. Close the connection.

Learning Outcome :

***Question** :*

File Transfer using TCP

***Aim** :*

To transfer a file from server to client using TCP.

***Algorithm** :*

a. *Server -*

   i. Create a socket, initialize server details and bind the server with the socket.

   ii. Listen for a client request.

   iii. Accept the client's request to connect.

   iv. Read the client's message (path of the file to be transferred)

   v. Initialize a file descriptor to the file and open it in read mode.

   vi. Read the file and store the contents in a buffer.

   vii. Send the buffer contents to the client.

   viii. Close the connection.

b. *Client -*

   i. Create a socket and initialize server details.

   ii. Make a request to connect with any available server.

   iii. Input the path of file to be transferred and send the path to server.

   iv. Input the destination path to store the file transferred.

   v. Initialize a file descriptor to the destination file and open it in write mode.

   vi. Receive the file contents from server and store it in a buffer.

   vii. Write the contents of buffer in the destination file.

   viii. On completion of time, close the connection.

Learning Outcome :

***Question** :*

Chat using TCP

***Aim** :*

To establish client-server chat using UNIX Socket Programming.

***Algorithm** :*

a. *Server -*

   i. Create a socket, initialize server details and bind the server with the socket.

   ii. Listen for a client request.

   iii. Accept the client's request to connect.

   iv. Start the timer for 60 seconds.

   v. Keep reading the client's message and reply back for the above specified time.

   vi. On completion of time, close the connection.

b. *Client -*

   i. Create a socket and initialize server details.

   ii. Make a request to connect with any available server.

   iii. Once connected successfully, start the timer for 60 seconds.

   iv. Keep reading the server's message and reply back for the above specified time.

   v. On completion of time, close the connection.

Learning Outcome :

***Question** :*

ARP using Socket Programming

***Aim** :*

To simulate ARP (Address Resolution Protocol) using Socket Programming.

***Algorithm** :*

a. *Server -*

i. Create a socket, initialize server details and bind the server with the socket.

ii. Create a structure ARP packet to store information such as MAC address and IP address of source and destination, and also the 16-bit data.

iii. Listen for any number of client requests.

iv. Accept the client requests one by one.

v. Develop an ARP request packet by initialising it with the values of source IP address, source MAC address and destination IP address by getting it as input from the user. For the destination MAC address and 16-bit data, initialise with all 0's.

vi. Broadcast the ARP request to all the clients.

vii. Wait for a reply from one of the clients.

viii. Once a reply is received from a client, extract the destination MAC address from the ARP reply packet. This is the client to which the ARP packet must be sent.

ix. Concatenate the 16-bit data to the ARP reply packet and send it to the same client.

x. Close the connection.

b. *Client -*

i. Create a socket and initialize server details.

ii. Make a request to connect with any available server. Get the IP address and MAC address of this client as input from the user.

iii. Once connected successfully, receive the ARP packet from the server.

iv. Extract the destination address from this packet.

v. If the IP address of the client matches with the destination address value, concatenate the MAC address of the client with the received packet and send it back to the server.

vi. Receive another packet from the server which contains the 16-bit data.

vii. Close the connection.

Learning Outcome :

***Question** :*

Simulate the concept of Domain Name Server using UDP.

***Aim** :*

To create server and client programs to simulate Domain Name Server (DNS) using UDP.

***Algorithm** :*

a. *Server 1 (.com server) -*

i. Initialize a socket, sockfd with AF_INET address family in UDP mode using the socket() function.

ii. Create a table to maintain the list of all ".com" servers along with their IP address and display the table.

iii. Initialize two variables, servaddr (to connect to local DNS) and servaddr2 (to connect to Server2) of the type struct sockaddr_in with the AF_INET address family, server port number and server IP address.

iv. Bind the socket address ***servaddr*** to the socket ***sockfd***.

v. Initialize a while loop with 1 as argument and run it on commands (vi) to (ix)

vi. Wait for a message from the local DNS server through sockfd.

vii. When a message is received, check if the domain name is ".com".

viii. If it is ".com" then search for the domain name in the table created, else pass on the message to Server2 (.org server), that is to servaddr2 through sockfd.

ix. If the domain name was found in the table, return the IP address, else return "Not found". If the message was passed onto Server2 return the reply from Server2 to local DNS server.

x. Close the socket used and terminate the program.

b. *Server 2 (.org server) -*

i. Initialize a socket, sockfd with AF_INET address family in UDP mode using the socket() function.

ii. Create a table to maintain the list of all ".org" servers along with their IP address and display the table.

iii. Initialize two variables, servaddr (to connect to local DNS) and servaddr2 (to connect to Server2) of the type struct sockaddr_in with the AF_INET address family, server port number and server IP address.

iv. Bind the socket address ***servaddr*** to the socket ***sockfd***.

v. Initialize a while loop with 1 as argument and run it on commands (vi) to (ix)

vi. Wait for a message from the local DNS server through sockfd.

vii. When a message is received, check if the domain name is ".org".

viii. If it is ".org" then search for the domain name in the table created, else pass on the message to Server2 (.edu server), that is to servaddr2 through sockfd.

ix. If the domain name was found in the table, return the IP address, else return "Not found". If the message was passed onto Server2 return the reply from Server2 to local DNS server.

x. Close the socket used and terminate the program.

c. *Server 3 (.edu server)* -

i. Initialize a socket, sockfd with AF_INET address family in UDP mode using the socket() function.

ii. Create a table to maintain the list of all ".edu" servers along with their IP address and display the table.

iii. Initialize two variables, servaddr (to connect to local DNS) and servaddr2 (to connect to Server2) of the type struct sockaddr_in with the AF_INET address family, server port number and server IP address.

iv. Bind the socket address ***servaddr*** to the socket ***sockfd***.

v. Initialize a while loop with 1 as argument and run it on commands (vi) to (ix)

vi. Wait for a message from the local DNS server through sockfd.

vii. When a message is received, check if the domain name is ".edu".

viii. If it is ".edu" then search for the domain name in the table created, else pass on the message to Server2 (.net server), that is to servaddr2 through sockfd.

ix. If the domain name was found in the table, return the IP address, else return "Not found". If the message was passed onto Server2 return the reply from Server2 to local DNS server.

x. Close the socket used and terminate the program.

d. *Server 4 (.net server)* -

i. Initialize a socket, sockfd with AF_INET address family in UDP mode using the socket() function.

ii. Create a table to maintain the list of all ".net" servers along with their IP address and display the table.

iii. Initialize two variables, servaddr (to connect to local DNS) and servaddr2 (to connect to Server2) of the type struct sockaddr_in with the AF_INET address family, server port number and server IP address.

iv. Bind the socket address ***servaddr*** to the socket ***sockfd***.

v. Initialize a while loop with 1 as argument and run it on commands (vi) to (ix)

vi. Wait for a message from the local DNS server through sockfd.

vii. When a message is received, check if the domain name is ".net".

viii. If it is ".net" then search for the domain name in the table created.

ix. If the domain name was found in the table, return the IP address, else return "Not found" to Server3.

x. Close the socket used and terminate the program.

e. Local DNS Server -

i. Initialize a socket, sockfd with AF_INET address family in UDP mode using the socket() function.

ii. Initialize two variables, servaddr (to connect to local DNS) and servaddr2 (to connect to Server2) of the type struct sockaddr_in with the AF_INET address family, server port number and server IP address.

iii. Bind the socket address ***servaddr*** to the socket ***sockfd***.

iv. Initialize a while loop with 1 as argument and run it on commands (v) to (viii)

v. Wait for a message from the client through sockfd.

vi. When a message is received, pass on the message to Server1.

vii. Wait for reply from Server1.

viii. Return the reply to client.

ix. Close the socket used and terminate the program

f. Client -

i. Initialize a socket, sockfd, with AF_INET address family in UDP mode using the socket() function.

ii. Initialize a variable, servaddr (to connect to Local DNS), of the type struct sockaddr_in with the AF_INET address family, server port number and server IP address.

iii. Send the domain name to the local DNS server (servaddr) through the sockfd.

iv. Wait for a reply from the Local DNS.

v. If the domain name was found in any one of the servers, the IP address is received as reply from Local DNS else the reply is “Not found”.

vI. Close the socket used and terminate the program.

Learning Outcome :

***Question** :*

HTTP Webpage Download

***Aim** :*

To develop a socket program to download a HTTP webpage.

***Algorithm** :*

i. Set the command line argument.

ii. Call the ***gethostbyname*** function to return a structure of type ***hostent*** for the given host name or null if not present.

iii. Create a socket on the client side and connect to the server.

iv. Send the data (website name) to the HTTP server.

v. Create a file in write mode.

vi. Accept contents of the file and store in a buffer.

vii. Write this buffer to the file.

viii. Once all the contents are written to the file, terminate the loop, close the file and the connection.

Learning Outcome :

***Question** :*

Error Checking using Hamming Code

***Aim** :*

To implement Hamming Code for Single Error Correction using C Socket Programming.

***Algorithm** :*

a. *Server -*

   i. Read the input from user (0's and 1's).

   ii. Calculate the number of redundant bits.

   iii. Position the redundant bits.

   iv. Calculate the values of each redundant bit.

   v. Introduce error(s) (single bit error or no error).

   vi. Send the data to the client.

b. *Client -*

   i. Receive the data from the server.

   ii. Calculate of the number of redundant bits.

   iii. Position the redundant bits.

   iv. Ckeck the parity.

   v. If any error, correct it and display the original message.

Learning Outcome :

***Question** :*

Performance Evaluation of TCP and UDP

***Aim** :*

To write TCL scripts to evaluate the performance of TCP and UDP sharing a bottleneck link.

***Algorithm** :*

i. Create six nodes and the links between the nodes as

a. 0 → 2 2Mb 10ms duplex link

b. 1 → 2 2Mb 10ms duplex link

c. 2 → 3 0.3Mb 100ms simplex link

d. 3 → 2 0.3Mb 100ms simplex link (link 2 → 3 is a bottleneck)

e. 3 → 4 0.5Mb 40ms duplex link

f. 3 → 5 0.5Mb 40ms duplex link.

ii. Align the nodes properly.

iii. Set Queue Size of link (2 → 3) to 10 (or) 5.

iv. Setup a TCP connection over 0 and 4 with its flow id, window size and packet size.

v. Setup a UDP connection over 1 and 5 with its flow id, type, packet size, rate and random fields.

vi. Set different colors for TCP and UDP.

vii. Run the simulation for 5 seconds, and show the simulation in the network animator and in trace files.

viii. Analyze the performance of TCP and UDP from the simulation.

Learning Outcome :

***Question** :*

Simulation of Distance Vector Routing Protocol

***Aim** :*

To write a NS2 program for implementing unicast routing protocol.

***Algorithm** :*

i. Start the program.

ii. Declare global variable ***ns*** for creating a new simulator.

iii. Set the colors for the packets.

iv. Open the network animator file in the name of ***file2*** in write mode.

v. Open the trace file in the name of ***file1*** in write mode.

vi. Set the unicast routing protocol (Distance Vector Routing Protocol) to transfer the packets in the network.

vii. Create the required no. of nodes.

viii. Create a duplex-link between the nodes including the delay time, bandwidth and dropping queue mechanism.

ix. Give the position for the links between the nodes.

x. Set a TCP reno connection for the source node.

xi. Set the destination node using TCP sink.

xii. Set up a FTP connection over the TCP connection.

xiii. Down the connection between any nodes at a particular time.

xiv. Reconnect the downed connection at a particular time.

xv. Define the ***finish*** procedure and declare global variables ***ns, file1 and file2.***

xvi. Close the trace file and name file and execute the network animation file.

xvii. Call the ***finish*** procedure at a particular time.

xviii. Stop the program.

Learning Outcome :

***Question** :*

Simulation of Link State Routing Protocol

***Aim** :*

To write a NS2 program for implementing unicast routing protocol.

***Algorithm** :*

i. Start the program.

ii. Declare global variable ***ns*** for creating a new simulator.

iii. Set the colors for the packets.

iv. Open the network animator file in the name of ***file2*** in write mode.

v. Open the trace file in the name of ***file1*** in write mode.

vi. Set the unicast routing protocol (Link State Routing Protocol) to transfer the packets in the network.

vii. Create the required no. of nodes.

viii. Create a duplex-link between the nodes including the delay time, bandwidth and dropping queue mechanism.

ix. Give the position for the links between the nodes.

x. Set a TCP reno connection for the source node.

xi. Set the destination node using TCP sink.

xii. Set up a FTP connection over the TCP connection.

xiii. Down the connection between any nodes at a particular time.

xiv. Reconnect the downed connection at a particular time.

xv. Define the ***finish*** procedure and declare global variables ***ns, file1 and file2.***

xvi. Close the trace file and name file and execute the network animation file.

xvii. Call the ***finish*** procedure at a particular time.

xviii. Stop the program.

Learning Outcome :