Implement a simple TCP client-server chatting program algorithms

Experiment-2: A simple TCP Client-Server program

Server:

step 1: Include appropriate header files.
step 2: Create a TCP Server Socket using socket() system call
step 3: Fill in the socket address structure
step 4: Specify the server port no.
step 5: Bind the address and port using bind() system call.
step 6: Server executes listen() system call to indicate its willingness to receive connections.
step 7:Accept the connection from the client process by using an accept() system call.
step 8:Receive a message from the Client using recv() system call.
step 9:Print the recieved message from client
step 10: close the socket.

Client:

step 1: Include appropriate header files
step 2: Create a TCP Client Socket.
step 3: Fill in the socket address structure
step 4: Specify the server port no.
step 5: Establish connection to the Server using connect() system call.
step 6: Send a message to client using send() system call.
step 7: close the socket.

Experiment-3: A simple UDP Client-Server program

Server:

Step 1: Include appropriate header files.
Step 2: Create a UDP Server Socket using socket() system call.
Step 3: Fill in the socket address structure
Step 4: Specify the server port number.

Step 5: Bind the address and port using bind() system call.
Step 6: Receive a message from the client using recvfrom() system call.
Step 7: Print the received message from the client.
Step 8: Close the socket.

Client:

Step 1: Include appropriate header files.
Step 2: Create a UDP Client Socket.
Step 3: Fill in the socket address structure
Step 4: Specify the server port number.
Step 5: Prompt the user to enter a string.
Step 6: Send the entered string to the server using sendto() system call.
Step 7: Close the socket.

Experiment - 4: Client-Server Communication using TCP

Server:

Step 1: Include appropriate header files.
Step 2: Create a TCP Server Socket using socket() system call.
Step 3: Fill in the socket address structure
Step 4: Specify the server port number.
Step 5: Bind the address and port using bind() system call.
Step 6: Server executes listen() system call to indicate its willingness to receive connections.
Step 7: Accept the connection from the client process by using an accept() system call.
Step 8: Loop:
a. Receive a message from the client using recvfrom() system call.
b. Print the received message from the client.
c. Check if the message is "bye". If yes, break from the loop.
d. If not, prompt the server to enter a message and send it to the client using sendto() system call.
Step 9: Close the server socket.

Client:

Step 1: Include appropriate header files.
Step 2: Create a TCP Client Socket.
Step 3: Fill in the socket address structure

Step 4: Specify the server port number.

Step 5: Establish connection to the server using connect() system call.

Step 6: Implement a chat function to send and receive messages with the server.

Step 7: Loop:

a. Prompt the user to enter a message for the server.

b. Send the entered message to the server using sendto() system call.

c. Check if the message is "bye". If yes, break from the loop.

d. Receive a message from the server using recvfrom() system call.

e. Print the received message from the server.

Step 8: Close the client socket.

Experiment - 5: Implement a simple UDP client-server chatting programming

Server:

Step 1: Include appropriate header files.

Step 2: Create a UDP Server Socket using socket() system call.

Step 3: Fill in the socket address structure

Step 4: Specify the server port number.

Step 5: Bind the address and port using bind() system call.

Step 6: Loop:

a. Receive a message from the client using recvfrom() system call.

b. Print the received message from the client.

c. Check if the message is "bye". If yes, break from the loop.

d. If not, prompt the server to enter a message and send it to the client using sendto() system call.

Step 7: Close the server socket.

Client:

Step 1: Include appropriate header files.

Step 2: Create a UDP Client Socket.

Step 3: Fill in the socket address structure

Step 4: Specify the server port number.

Step 5: Loop:

a. Prompt the user to enter a message for the server.

b. Send the entered message to the server using sendto() system call.

c. Check if the message is "bye". If yes, break from the loop.

d. Receive a message from the server using recvfrom() system call.

e. Print the received message from the server.

Step 6: Close the client socket.