TCP Server Side Algorithm

Include Necessary Headers

Define Constants: Define a constant for the port number (8080).

Initialize Variables

Create Socket: Create a socket using the socket() function.If the socket creation fails, exit the program.

Set Socket Options: Set socket options using the setsockopt() function to allow reuse of local addresses.

Define Address Structure:

Set sin_family to AF_INET for IPv4.

Set sin_addr.s_addr to INADDR_ANY to accept connections from any address.

Set sin_port to the specified port number (8080), converting it to network byte order using htons().

Bind Socket: Bind the socket to the specified address and port using the bind() function.

If the binding fails, print an error message and exit the program.

Listen for Connections: Put the server socket in passive mode using the listen() function to listen for incoming connections.

If the listening fails, print an error message and exit the program.

Accept Connection: Accept an incoming connection using the accept() function.

If accepting the connection fails, print an error message and exit the program.

Read from Client: Read data from the accepted socket into the buffer using the read() function.

Print Client Message: Print the message received from the client.

Send Response to Client: Send a response message to the client using the send() function.

Print Response Message: Print a message indicating that the response has been sent.

TCP Client Side Algorithm

Initialize: Declare variables and structures (sock, valread, serv_addr, buffer, hello).

Create Socket: Use socket() to create a socket.

Define Server Address: Configure serv_addr with AF_INET, PORT (using htons(PORT)), and IP address (using inet_pton()).

Connect to Server: Connect to the server with connect().

Send Data: Send a message to the server with send().

Read Response: Read the server's response into buffer with read().

Print Response: Print the received message.

Exit: Return 0 to end the program.

UDP Server Side Algorithm

Initialize: Declare variables and structures (`sockfd`, `buffer`, `hello`, `servaddr`, `cliaddr`).

Create Socket: Create a socket using socket(AF_INET, SOCK_DGRAM, 0).If socket creation fails, print an error message and exit.

Zero Memory: Use memset() to zero the servaddr and cliaddr structures.

Define Address: Set servaddr with AF_INET, INADDR_ANY, and PORT (using htons(PORT)).

Bind Socket: Bind the socket to the address with bind().If binding fails, print an error message and exit.

Receive Data: Use recvfrom() to receive data from a client into buffer.

Print Client Message: Print the message received from the client.

Send Response: Send a response message to the client using sendto().

Print Response Message: Print a message indicating the response has been sent.

UDP Client Side Algorithm

Initialize: Declare variables and structures (sockfd, buffer, hello, servaddr).

Create Socket: Create a socket using socket(AF_INET, SOCK_DGRAM, 0).If socket creation fails, print an error message and exit.

Zero Memory: Use memset() to zero the servaddr structure.

Define Server Address: Set servaddr with AF_INET, INADDR_ANY, and PORT (using htons(PORT)).

Send Data: Send a message to the server using sendto().

Print Sent Message: Print a message indicating that the message has been sent.

Receive Response: Use recvfrom() to receive the server's response into buffer.

Print Server Response: Print the message received from the server.

Close Socket: Close the socket using close().

MULTI CHAT Client Side Algorithm

Setup: Include necessary headers and define constants (PORT, BUF_SIZE).

Check Arguments: Ensure the server IP address is provided as a command-line argument.

Initialize: Declare variables

Create Socket: Create a socket using socket(AF_INET, SOCK_STREAM, 0).If socket creation fails, print an error message and exit.

Configure Server Address: Zero the addr structure and set addr.sin_family to AF_INET,addr.sin_addr.s_add

r to the server IP, and addr.sin_port to PORT.
Connect to Server: Use connect() to connect to the server.If connection fails, print an error message and exit.
Communication Loop:Zero the buffer.Prompt the user to enter a message.Read the user's input using fgets().Send the message to the server using send().Receive the server's response using recv().Print the server's response.**Exit**: Return 0 to end the program.

MULTI CHAT SERVER Side Algorithm

Server Side Algorithm

Setup: Include necessary headers and define constants (PORT, BUF_SIZE, CLADDR_LEN).
Initialize: Declare variables (addr, cl_addr, sockfd, len, ret, newsockfd, buffer, childpid, clientAddr).
Create Socket: Create a socket using socket(AF_INET, SOCK_STREAM, 0).If socket creation fails, print an error message and exit.
Configure Server Address: Zero the addr structure and set addr.sin_family to AF_INET, addr.sin_addr.s_addr to INADDR_ANY, and addr.sin_port to PORT.
Bind Socket: Bind the socket to the address with bind().If binding fails, print an error message and exit.**Listen for Connections**: Use listen() to set the socket to listen mode.**Accept and Handle Connections**:Enter an infinite loop to accept incoming connections using accept().If a connection is accepted, print a message.Use fork() to create a child process to handle the client.In the child process:
Close the server socket.
Enter an infinite loop to handle communication with the client:
Zero the buffer.
Receive data from the client using recv().
Print the received data.
Send data back to the client using send().
Exit: Close the connection socket and return 0 to end the program.

STOP AND WAIT GENERAL

Step 1: Start the program
Step 2: import all the necessary libraries
Step 3: Create 2 Application client and server
Step 4: Connect both Application using socket
Step 5: Sender frame is sent to the receiver and displayed by the receiver
Step 6: Receiver sends the acknowledgement to the sender if the frame is received else negative acknowledgement is sent
Step 7: Sender waits for the acknowledgement from the receiver
Step 8: If the acknowledgement is received then the sender sends the next frame
Step 9: If the negative acknowledgement is received then the sender sends the same frame again

STOP AND WAIT Client Side Algorithm

Initialize:Create and set up a UDP socket.Define server address.

Main Loop:If `ack_recv` is 1:
Prepare frame with `sq_no`, `frame_kind`, `ack`, and data.
Send frame to server.
Print confirmation of frame sent.
Receive acknowledgment from server.
If valid acknowledgment received, set `ack_recv` to 1 and print confirmation.
If acknowledgment not received, set `ack_recv` to 0.
Increment `frame_id`.
Exit: Close the socket.

STOP AND WAIT Server Side Algorithm

Initialize:Create and set up a UDP socket.Define and bind server address.
Main Loop:Receive frame from client.If valid frame received, print data and prepare acknowledgment frame.
Send acknowledgment frame to client and print confirmation.
If frame not valid, print a message.
Increment `frame_id`.
Exit: Close the socket.

<u>Algorithm - Leaky Bucket</u>

Step 1: Input the bucket size, outgoing rate, and no of inputs
Step 2: While n is not equal to 0,
Step 3: Input the incoming packet size
Step 4: Print the incoming packet size
Step 5: If the incoming packet size is less than or equal to the bucket size - store,
Step 6: Add the incoming packet size to the store

Step 7: Print the bucket buffer size and the store

Step 8: Subtract the outgoing rate from the store

Step 9: If the store is less than 0,

Step 10: Set the store to 0

Step 11: Print the after outgoing packets left out of the bucket buffer size and the store

Step 12: Subtract 1 from n

Step 13: End while

Step 14: End program

Algorithm - Time Server Application - UDP Server side

Step 1: Start the program

Step 2: create a socket with the help of socket() function

Step 3: bind the socket to the address and port number using the bind() function

S**tep 4**: listen for the incoming requests using the listen() function

Step 5: receive request from the client using the recvfrom() function

Step 6: send the current time to the client using the sendto() function

Step 7: close the socket

Algorithm - Time Server Application - UDP Client side

Step 1: Start the program

Step 2: Send a request to the server asking for the current time

Step 3: Receive the current time from the server using the recvfrom() function

Step 4: Display the current time on the screen

Step 5: close the socket

SERVER ADD TWO NUMBER TCP

```
int main(int argc, char const* argv[])
{ int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    int opt = 1;
    int num1, num2, sum;
    if ((server_fd = socket.. Write bal
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);   }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE); }
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);  }
    if ((new_socket =
accept(server_fd, (struct
sockaddr*)&address,
(socklen_t*)&addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE);  }
    valread = read(new_socket,
buffer, 1024);
    sscanf(buffer, "%d %d", &num1,
&num2);
    printf("Received numbers: %d
and %d\n", num1, num2);
     sum = num1 + num2;
    sprintf(buffer, "Sum: %d", sum);
    send(new_socket, buffer,
strlen(buffer), 0);
    printf("Sum message sent\n");
    return 0;}
```

CLIENT ADD TWO NUMBER TCP

```
int main(int argc, char const*
argv[]){
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    char message[1024];
    int num1, num2;
    if ((sock = socket(AF_INET,
SOCK_STREAM, 0)) < 0) {
printf("Socket creation
error\n");return -1;
serv_addr.sin_family = AF_INET;
  serv_addr.sin_port =
htons(PORT);
    if (inet_pton(AF_INET,
"127.0.0.1", &serv_addr.sin_addr)
<= 0) {printf("\nInvalid address/
Address not supported\n");
        return -1;}
    if (connect(sock, (struct
sockaddr*)&serv_addr,
sizeof(serv_addr)) < 0) {
        printf("\nConnection
Failed\n");
        return -1; }
    printf("Enter two numbers: ");
    scanf("%d %d", &num1,
&num2);
    sprintf(message, "%d %d",
num1, num2);
 send(sock, message,
strlen(message), 0);
    printf("Numbers sent\n");
    valread = read(sock, buffer,
1024);
    printf("%s\n", buffer);
  return 0;}
```