RTPNOTES.vercel.app

Computer-Networks-TCP-Program-Tips

For more notes visit

https://rtpnotes.vercel.app

For lab programs visit

https://pgmsite.netlify.app

RTPNOTES.vercel.app

TCP Client

Include Statements

```
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys.types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

TCP-Client: Basic Algorithm for remembering

1. Create the socket
2. Setup the server address struct
3. Loop
    1. Connect to server
    2. Read input
    3. Send message
    4. Receive message
    5. Logic to end the loop
4. Close the socket

TCP-Client: Steps in more detail

RTPNOTES.vercel.app

1. Creating the socket

- You will need the following lines

```
int client_socket
client_socket = socket(AF_INET, SOCK_STREAM, 0);
```

- Here AF_INET is the address_family
- SOCK_STREAM is the socket type (SOCK_STREAM Indicates its TCP)
- Protocol is 0

2. Setup the server address structure

```
server.sin_family = AF_INET; // AF_INET is used for IPv4 addresses
server.sin_port = 2000; // Port number
server.sin_addr.s_addr = inet_addr("127.0.0.1"); // IP address
```

- Here we setup the family, portnumber and servers IP address.

3. Loop

3.1 Connecting to the server

- We are using TCP server here, since TCP is connection oriented, we need to connect

```
connect(client_socket, (struct sockaddr *)&server, sizeof(server)); // Since
TCP is connection oriented we need to Connect
```

- Connect takes 3 arguments
  - sockfd
    - This is the file descriptor of the socket you want to connect
    - File descriptor is like an integer which uniquely identifies a socket
  - Address
    - This is a pointer to a `struct sockaddr` that contains the address of the server to which you want to connect.

RTPNOTES.vercel.app

- Address length
    - Its the length of the address structure

3.2 Read input

- For reading the input we need two things
    - sBuf variable
        - Variable which contains data to be sent to the server
        - `char sBuf[100] = ""
    - `gets(sBuf)`
        - This will read the content and store in sBuf

3.3 Send Message

- After the input is ready, we need to send it to the server

```
send(client_socket, sBuf, sizeof(sBuf), 0); // Send data to the server
```

- It has the following arguments
    - sockfd
        - This is the file descriptor of the socket you want to connect
    - Buffer
        - Pointer to the buffer we want to send to the server
    - Size of Buffer
        - Size of content to send
    - Flags
        - Set as 0, no flags needed

3.4 Receive Message

- Here we will be receiving the message from server after we have the our message
- We need a variable called `rBuf` to store the received buffer
    - `char rBuf[100] = ""`

RTPNOTES.vercel.app

```
recv(client_socket, rBuf, sizeof(rBuf), 0); // Receive data from the server
```

- It has the following arguments
  - sockfd
  - Buffer
  - Size of Buffer
  - Flags

3.5 Logic to end the loop

```
if(strcmp(sBuf, "end") == 0) // If server sends "end", break the loop
        break;
```

4. Close the socket

- You can use this to close the socket

```
close(client_socket)
```

TCP Server

TCP-Server: Basic Algorithm for remembering

1. Create the socket
2. Setup the server address struct
3. Bind the socket and listen for connections
4. Accept connection from client
5. Loop
   1. Receive data from client and print the message
   2. Read input to send to client
   3. Send message
   4. Logic to end the loop

RTPNOTES.vercel.app

6. Close connection with client
7. Close server socket

TCP-Server: Steps in more detail

1. Create the socket

```
lfd = socket(AF_INET, SOCK_STREAM, 0); // Create a socket
```

2. Setup the server address struct

```
server.sin_family = AF_INET; // IPv4
server.sin_port = 2000; // Port number
server.sin_addr.s_addr = inet_addr("127.0.0.1"); // IP address
```

3. Bind the socket and listen for connections

```
bind(lfd, (struct sockaddr *)&server, sizeof(server)); // Bind the socket to
the server address
listen(lfd, 1); // Listen for incoming connections, since TCP is connection
oriented
```

Bind

- The `bind` function is used to associate a socket with a specific local IP address and port number.
- The parameters used are
  - sockfd
    - The file descriptor of the socket to be bound. In your example, this is `lfd`, which is the listening file descriptor.
  - **`addr` (const struct sockaddr *)**:
    - A pointer to a `struct sockaddr` that contains the address to bind to the socket.
  - **`addrlen` (socklen_t)**:

RTPNOTES.vercel.app

        - The size of the address structure. In your example, this is `sizeof(server)`, which is the size of the `struct sockaddr_in` structure.

Listen

- The `listen` function marks a connection-oriented socket (like a TCP socket) as ready to accept incoming connection requests.
- The parameters used are
    - sockfd
        - The file descriptor of the socket to be set to listening mode. Here its `lfd`
    - backlog
        - The maximum length of the queue of pending connections. In this example, this is `1`, meaning the queue can hold at most one pending connection before additional connections are refused.

4. Accept connection from client

```
confd = accept(lfd, (struct sockaddr *)&client, &n); // Accept a connection
from a client
```

- The `accept` function is used in socket programming to accept incoming connection requests on a listening socket.
- When a client attempts to connect to the server, the `accept` function creates a new socket dedicated to that connection and returns a file descriptor for this new socket.
- Parameters used
    - sockfd
        - The file descriptor of the listening socket that is ready to accept connections.
    - **`addr` (struct sockaddr *)**:
        - A pointer to a `struct sockaddr` structure that will be filled with the address information of the connecting client. In this example, this is `(struct sockaddr *)&client`.
    - **`addrlen` (socklen_t *)**:
        - A pointer to a `socklen_t` variable that initially contains the size of the `addr` structure.

RTPNOTES.vercel.app

5. Loop

5.1 Receive data from client and print the message

```
recv(confd, rBuf, sizeof(rBuf), 0); // Receive data from the client
printf("\nClient: %s", rBuf); // Print the received data
```

- The `recv` function is used to receive data from a connected socket. It reads data from the socket into a buffer,
- Parameters used
  - sockfd
    - The file descriptor of the connected socket from which to receive data. In this example, this is `confd`, which represents the new socket file descriptor created by the `accept` function for the specific client connection.
  - Buffer
    - buffer where the received data will be stored.
  - Size of buffer
  - Flags

5.2 Read input to send to client

```
gets(sBuf); // Get server input
```

5.3 Sending the message

- The `send` function is used to transmit data from a socket to a connected peer. This function is typically used in server applications to send data to a connected client, or in client applications to send data to a server.

```
send(confd, sBuf, sizeof(sBuf), 0); // Send data to the client`
```

- Parameters used
  - sockfd
    - The file descriptor of the connected socket through which data will be sent. In this example, this is `confd`, which represents the new socket file descriptor

RTPNOTES.vercel.app

created by the `accept` function for the specific client connection.

- Buffer
- Size of buffer
- Flags

6. Close connection with client and server socket

```
close(confd); // Close the connection with the client
close(lfd); // Close the server socket`
```