This C program is a simple TCP client that connects to a server on the local machine (127.0.0.1) and communicates by sending and receiving messages. Here's a detailed explanation of the code:

**1. Include necessary libraries:**

* stdio.h: Standard input/output functions like printf.
* stdlib.h: General utilities, including memory allocation and process control (exit).
* string.h: Provides string manipulation functions such as strlen, bzero, and strcpy.
* unistd.h: Provides access to the POSIX operating system API, including the close() function.
* arpa/inet.h: Provides definitions for internet operations such as inet\_addr (converting IP address to network format).

**2. Main Function:**

The main function handles setting up the client, connecting to the server, sending a message, receiving a response, and then closing the connection.

**3. Variable Declaration:**

* char \*ip = "127.0.0.1";: The IP address of the server (localhost in this case).
* int port = 8080;: The port number to which the client will connect.
* int client\_socket;: The socket descriptor used for communication.
* struct sockaddr\_in addr;: Struct that holds the server's address information (IP address and port).
* socklen\_t addr\_size;: Size of the addr structure.
* char buffer[1024];: A buffer to store messages sent and received.
* int n;: Placeholder for the length of data sent or received.

**4. Socket Creation:**

c

Copy code

client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

* The socket() function creates a new socket.
* AF\_INET: The address family used for IPv4.
* SOCK\_STREAM: Indicates that the socket will use TCP (reliable, connection-based communication).
* 0: The protocol, set to default for TCP.
* If client\_socket < 0, the program prints an error and exits.

**5. Address Setup:**

c

Copy code

memset(&addr, '\0', sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = port;

addr.sin\_addr.s\_addr = inet\_addr(ip);

* memset() initializes the addr structure to zero.
* addr.sin\_family: Set to AF\_INET for IPv4 addresses.
* addr.sin\_port: The port number (8080 in this case).
* addr.sin\_addr.s\_addr: The server's IP address, converted to a network-compatible format using inet\_addr().

**6. Connecting to the Server:**

c

Copy code

connect(client\_socket, (struct sockaddr\*)&addr, sizeof(addr));

* The connect() function attempts to establish a connection to the server at the specified IP and port.
* client\_socket: The socket descriptor.
* (struct sockaddr\*)&addr: The server's address information, cast to a struct sockaddr\*.
* If the connection is successful, a message confirming the connection is printed.

**7. Send a Message to the Server:**

c

Copy code

bzero(buffer, 1024);

strcpy(buffer, "Hello, This is Client.");

send(client\_socket, buffer, strlen(buffer), 0);

* bzero(buffer, 1024): Clears the buffer to avoid garbage values.
* strcpy(buffer, "Hello, This is Client.");: Copies a message into the buffer.
* send(): Sends the contents of the buffer to the server.
  + client\_socket: The socket descriptor.
  + buffer: The data to send.
  + strlen(buffer): Length of the message.
  + 0: Flags (not used here).

**8. Receive a Response from the Server:**

c

Copy code

bzero(buffer, 1024);

recv(client\_socket, buffer, sizeof(buffer), 0);

printf("Server: %s \n", buffer);

* bzero(buffer, 1024): Clears the buffer before receiving data.
* recv(): Receives data from the server and stores it in buffer.
  + client\_socket: The socket descriptor.
  + buffer: Where the received data will be stored.
  + sizeof(buffer): Maximum number of bytes to receive.
  + 0: Flags (not used here).
* The message received from the server is then printed.

**9. Close the Connection:**

c

Copy code

close(client\_socket);

* The close() function closes the socket, terminating the connection with the server.

**10. Program End:**

The program ends after the connection is closed, printing a message to indicate that the client has disconnected.

**Overall Flow:**

1. The client creates a TCP socket.
2. The client connects to the server using IP 127.0.0.1 and port 8080.
3. A message is sent from the client to the server.
4. The client receives a response from the server.
5. The connection is closed, and the client terminates.

Reciever

This C program implements a simple TCP server that listens for connections on the local machine (127.0.0.1) and communicates with clients. Here’s an explanation of the code:

**1. Include necessary libraries:**

* stdio.h: For input/output functions like printf.
* stdlib.h: For memory management and process control (exit).
* string.h: For string manipulation functions like strcpy, bzero.
* unistd.h: Provides access to operating system functions, including close().
* arpa/inet.h: Contains definitions for internet operations, like inet\_addr to convert an IP address from a string to binary format.

**2. Main Function:**

The main function handles creating the server, binding it to an IP address and port, listening for incoming connections, accepting client connections, and sending/receiving messages.

**3. Variable Declaration:**

* char \*ip = "127.0.0.1";: The IP address where the server will listen (localhost).
* int port = 8080;: The port number on which the server listens for connections.
* int server\_socket, client\_socket;: Socket descriptors for the server and client.
* struct sockaddr\_in server\_addr, client\_addr;: Structures that hold server and client address information (IP address and port).
* socklen\_t addr\_size;: The size of the address structure.
* char buffer[1024];: A buffer to store messages sent/received.
* int n;: Holds the return value of certain functions (like bind).

**4. Create TCP Socket:**

c

Copy code

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

* Creates a TCP socket using IPv4 (AF\_INET) and stream-based communication (SOCK\_STREAM).
* If server\_socket < 0, an error occurred, and the program exits.

**5. Address Setup:**

c

Copy code

memset(&server\_addr, '\0', sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = port;

server\_addr.sin\_addr.s\_addr = inet\_addr(ip);

* memset() initializes the server\_addr structure to zero.
* server\_addr.sin\_family = AF\_INET: Sets the address family to IPv4.
* server\_addr.sin\_port = port: Sets the port number (8080 in this case).
* server\_addr.sin\_addr.s\_addr = inet\_addr(ip): Converts the IP address from a string to a format usable by the socket.

**6. Bind the IP and Port to the Socket:**

c

Copy code

n = bind(server\_socket, (struct sockaddr\*)&server\_addr, sizeof(server\_addr));

* bind() associates the server’s IP address and port with the server socket.
* If bind() fails (returns < 0), an error is printed, and the program exits.

**7. Listen for Incoming Connections:**

c

Copy code

listen(server\_socket, 5);

* listen() makes the server ready to accept incoming connections.
* The second argument (5) specifies the maximum number of pending connections allowed in the queue.

**8. Accept Client Connections in a Loop:**

c

Copy code

while(1){

addr\_size = sizeof(client\_addr);

client\_socket = accept(server\_socket, (struct sockaddr\*)&client\_addr, &addr\_size);

* accept() waits for a client to connect. When a client connects, a new socket (client\_socket) is created for communication with that client.
* The server keeps running indefinitely, accepting and processing one client at a time.

**9. Receive Data from the Client:**

c

Copy code

bzero(buffer, 1024);

recv(client\_socket, buffer, sizeof(buffer), 0);

* bzero(buffer, 1024): Clears the buffer to avoid garbage data.
* recv() receives data from the client and stores it in buffer.
  + client\_socket: The socket descriptor for the client.
  + buffer: Where the received data will be stored.
  + sizeof(buffer): Maximum size of the data to receive (1024 bytes).
* The message received from the client is printed.

**10. Send a Response to the Client:**

c

Copy code

bzero(buffer, 1024);

strcpy(buffer, "Hello, message received. This is Server.");

send(client\_socket, buffer, strlen(buffer), 0);

* bzero(buffer, 1024): Clears the buffer.
* strcpy(buffer, "Hello, message received. This is Server.");: Copies a response message into the buffer.
* send(): Sends the contents of buffer back to the client.
  + client\_socket: The socket descriptor for the client.
  + buffer: The data to send.
  + strlen(buffer): The length of the message.

**11. Close the Client Connection:**

c

Copy code

close(client\_socket);

* close() closes the connection with the client after the message exchange is complete.

**12. Repeat for Other Clients:**

The server continues to listen for and accept new clients in the infinite while loop.

**Overall Flow:**

1. The server creates a TCP socket.
2. It binds to IP 127.0.0.1 and port 8080.
3. The server listens for incoming connections.
4. When a client connects, the server accepts the connection.
5. The server receives a message from the client.
6. The server sends a response back to the client.
7. The connection with the client is closed, and the server waits for the next client.

This is a basic server implementation that processes one client at a time.

4o