SERVER

This C program is a simple implementation of a UDP client-server communication where the client sends and receives messages to/from a server using the User Datagram Protocol (UDP). Here’s an explanation of the code:

**1. Include necessary libraries:**

* stdio.h: For standard input/output operations like printf.
* stdlib.h: For general utilities such as memory allocation, conversion, and process control.
* string.h: For string manipulation functions like strcpy and bzero.
* sys/socket.h, sys/types.h, netinet/in.h, arpa/inet.h: These libraries are required for socket programming. They handle socket creation, address manipulation, and internet protocols.

**2. Main Function:**

The main function performs the core tasks of setting up a UDP socket, sending a message to the server, and receiving a response.

**3. Command-Line Argument Check:**

c

Copy code

if(argc != 2){

printf("Usage: %s <port>\n", argv[0]);

exit(0);

}

* This checks if the user provided the port number as an argument while running the program. The program expects exactly one argument (the port number), otherwise it prints usage instructions and exits.

**4. IP and Port Setup:**

c

Copy code

char \*ip = "127.0.0.1";

int port = atoi(argv[1]);

* ip: The IP address is set to 127.0.0.1, which is the loopback address (local machine).
* port = atoi(argv[1]): The port number is taken from the command-line argument and converted from a string to an integer using atoi.

**5. Socket Creation:**

c

Copy code

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

* This line creates a UDP socket (SOCK\_DGRAM specifies UDP).
* AF\_INET refers to the IPv4 address family.
* The socket descriptor (sockfd) is used for subsequent operations like sending and receiving data.

**6. Address Setup:**

c

Copy code

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

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port);

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

* memset() clears the addr structure to avoid any garbage values.
* addr.sin\_family = AF\_INET: Specifies that this is an IPv4 address.
* addr.sin\_port = htons(port): Sets the port number, converting it to network byte order using htons (Host-to-Network Short).
* addr.sin\_addr.s\_addr = inet\_addr(ip): Converts the IP address (127.0.0.1) from a string format to binary form.

**7. Sending Data to Server:**

c

Copy code

bzero(buffer, 1024);

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

sendto(sockfd, buffer, 1024, 0, (struct sockaddr\*)&addr, sizeof(addr));

printf("[+] Data sent successfully\n");

* bzero(buffer, 1024): Clears the buffer to ensure there is no garbage data.
* strcpy(buffer, "Hello, This is Server."): Copies the message to be sent into the buffer.
* sendto(sockfd, buffer, 1024, 0, (struct sockaddr\*)&addr, sizeof(addr)): Sends the message stored in buffer to the server.
  + sockfd: The socket descriptor used for sending.
  + buffer: The message being sent.
  + 1024: The size of the buffer.
  + addr: The address of the server where the message is being sent.
* printf("[+] Data sent successfully\n");: Prints a success message after sending the data.

**8. Receiving Data from Server:**

c

Copy code

bzero(buffer, 1024);

addr\_size = sizeof(addr);

recvfrom(sockfd, buffer, 1024, 0, (struct sockaddr\*)&addr, &addr\_size);

printf("[+] Data received: %s\n", buffer);

* bzero(buffer, 1024): Clears the buffer to ensure no old data remains.
* recvfrom(sockfd, buffer, 1024, 0, (struct sockaddr\*)&addr, &addr\_size): Receives data from the server.
  + sockfd: The socket descriptor used for receiving.
  + buffer: The buffer to store the incoming data.
  + addr: The address from which the data is received.
  + addr\_size: Stores the size of the address structure.
* The received message is then printed using printf().

**9. Close the Socket:**

c

Copy code

close(sockfd);

* This closes the UDP socket after communication is done.

**Overall Flow:**

1. The client program is run with a specified port number as a command-line argument.
2. A UDP socket is created.
3. The client sends a message ("Hello, This is Server.") to the specified server IP (127.0.0.1) and port.
4. The client then waits to receive a response from the server.
5. The received message is printed, and the socket is closed.

This code demonstrates basic UDP communication where the client sends and receives messages without establishing a formal connection, typical of UDP's connectionless nature.

RECIEVER

This C program implements a basic UDP (User Datagram Protocol) server that listens on a specified port, receives data from a client, and sends a response back. Let’s go through the code in detail:

**1. Include necessary libraries:**

* stdio.h: For standard input/output functions like printf.
* stdlib.h: For general utility functions like memory allocation and process control (exit).
* string.h: For string manipulation functions like bzero and strcpy.
* sys/socket.h, sys/types.h, netinet/in.h, arpa/inet.h: These are required for socket programming, specifically for working with sockets, networking protocols, and address manipulation.

**2. Main Function:**

The core of the program, where the server socket is created, data is received, and a response is sent.

**3. Command-Line Argument Check:**

c

Copy code

if(argc != 2){

printf("Usage: %s <port>\n", argv[0]);

exit(0);

}

* This checks if the user provided the port number as an argument. If not, it displays the correct usage and exits the program.

**4. IP and Port Setup:**

c

Copy code

char \*ip = "127.0.0.1";

int port = atoi(argv[1]);

* ip = "127.0.0.1": The server will listen on the local loopback address (localhost).
* port = atoi(argv[1]): Converts the provided port number (as a string from the command line) into an integer using atoi.

**5. Socket Creation:**

c

Copy code

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

if(sockfd < 0){

perror("[-] Socket error");

exit(1);

}

* A UDP socket is created using SOCK\_DGRAM.
* AF\_INET: Specifies the IPv4 address family.
* The program checks whether the socket creation is successful by checking if sockfd is negative. If it fails, an error is printed and the program exits.

**6. Server Address Setup:**

c

Copy code

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

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

server\_addr.sin\_port = htons(port);

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

* memset(&server\_addr, '\0', sizeof(server\_addr)): Clears the server\_addr structure to ensure it contains no garbage values.
* server\_addr.sin\_family = AF\_INET: Specifies IPv4 as the address family.
* server\_addr.sin\_port = htons(port): Converts the provided port number to network byte order using htons.
* server\_addr.sin\_addr.s\_addr = inet\_addr(ip): Converts the IP address string (127.0.0.1) to its binary form.

**7. Binding the Socket:**

c

Copy code

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

if(n < 0){

perror("[-] Bind error");

exit(1);

}

* bind() associates the socket (sockfd) with the server's IP address and port number. If bind() fails (returns a negative value), the program prints an error and exits.

**8. Receiving Data from Client:**

c

Copy code

bzero(buffer, 1024);

addr\_size = sizeof(client\_addr);

recvfrom(sockfd, buffer, 1024, 0, (struct sockaddr\*)&client\_addr, &addr\_size);

printf("[+] Data received: %s\n", buffer);

* bzero(buffer, 1024): Clears the buffer to avoid holding any leftover or garbage values.
* recvfrom(): This function listens for data from the client without needing a connection (since UDP is connectionless).
  + sockfd: The socket file descriptor to receive data on.
  + buffer: The buffer where the received message will be stored.
  + 1024: The maximum size of the buffer.
  + client\_addr: The address of the client that sent the data.
* The received data is then printed to the console.

**9. Sending Data to the Client:**

c

Copy code

bzero(buffer, 1024);

strcpy(buffer, "Welcome to Server. This is UDP Server.");

sendto(sockfd, buffer, 1024, 0, (struct sockaddr\*)&client\_addr, sizeof(client\_addr));

printf("[+] Data sent: %s\n", buffer);

* bzero(buffer, 1024): Clears the buffer before preparing the response message.
* strcpy(buffer, "Welcome to Server. This is UDP Server."): The server prepares a response message.
* sendto(): Sends the response back to the client.
  + client\_addr: This is the address of the client to whom the message is sent.

**10. Program Flow:**

1. The server is run with a specified port number as a command-line argument.
2. A UDP socket is created, and the server binds it to the specified IP address (127.0.0.1) and port.
3. The server listens for incoming messages from a client using recvfrom.
4. After receiving a message, the server prints it, prepares a response, and sends the message back to the client using sendto.
5. The server continues running, waiting for more data from clients.

**Key Points:**

* **Connectionless**: UDP is a connectionless protocol, meaning no formal connection is established between client and server.
* **Unreliable**: There is no guarantee that packets are delivered in UDP, unlike TCP.
* **Low Overhead**: UDP is faster and has lower overhead than TCP because it lacks the connection establishment, reliability checks, and error correction features.

This is a simple UDP server implementation that can interact with a corresponding UDP client by sending and receiving messages.

4o

Top of Form

Bottom of Form

ChatGPT can ma