MODULE 4

1.Explain the connection procedure followed in client server communication

- In C programming, socket programming is used for client-server communication
- The necessary headers for socket programming in your C program. The most common headers are <sys/socket.h> for socket functions and <netinet/in.h> for internet address manipulation.

Create Socket: Both the client and server need to create a socket. The server creates a socket using the socket() system call. The client also creates a socket in a similar manner.

Server:

```
int server_socket = socket(AF_INET, SOCK_STREAM, 0);
if (server_socket < 0) {
   perror("Socket creation failed");
   exit(EXIT_FAILURE);
}
```

Client:

```
int client_socket = socket(AF_INET, SOCK_STREAM, 0);
if (client_socket < 0) {
   perror("Socket creation failed");
   exit(EXIT_FAILURE);
}
```

Specify Server Address: In the client program, specify the address of the server to connect to. This includes the IP address and port number. In the server program, define and bind the server address.

```
struct sockaddr_in server_address;
server_address.sin_family = AF_INET;
server_address.sin_port = htons(PORT); // Port number
server_address.sin_addr.s_addr = INADDR_ANY; // IP address
```

Connect: In the client program, use the connect() system call to connect to the server.

```
if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
    perror("Connection failed");
    exit(EXIT_FAILURE);
}
```

Accept: In the server program, use the bind() system call to bind the socket to a specific address and port. Then use the listen() system call to listen for incoming connections. Finally, use the accept() system call to accept a connection request.

```
if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
    perror("Binding failed");
    exit(EXIT_FAILURE);
}
if (listen(server_socket, BACKLOG) < 0) {
    perror("Listen failed");
    exit(EXIT_FAILURE);
}

int client_socket = accept(server_socket, (struct sockaddr *)&client_address,
(socklen_t*)&address_length);
if (client_socket < 0) {
    perror("Acceptance failed");
    exit(EXIT_FAILURE);
}
```

Communication: Once the connection is established, the client and server can communicate by sending and receiving data using send() and recv() system calls.

Close Connection: After communication is complete, close the sockets using close() system call.

2. What is the use of bind() function in socket programming ?

In socket programming in C, the bind() function is used to associate a socket with a specific network address, typically the address of the host machine and a port number on that host.

Source code:

```
struct sockaddr_in server_address;
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = INADDR_ANY; // Listen on all interfaces
server_address.sin_port = htons(PORT); // Port number

if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
   perror("Binding failed");
   exit(EXIT_FAILURE);
}
```

3. What is Datagram Socket ?

In C programming, specifically in socket programming, a datagram socket is a type of socket that provides connectionless communication. It is also known as a "datagram-oriented socket" or "connectionless socket." Datagram sockets are commonly used with the User Datagram Protocol (UDP).

Key points about datagram sockets:

Connectionless Communication: Unlike stream sockets, which provide a reliable, connection-oriented communication channel (such as TCP), datagram sockets offer connectionless communication. This means that there is no established connection between the sender and receiver before data transmission.

Unreliable Transmission: Datagram sockets provide unreliable transmission of data. This means that there is no guarantee of delivery, order, or duplication prevention of the transmitted data. It's up to the application to handle these concerns if necessary.

No Persistent Connection: With datagram sockets, each datagram (or packet) is sent individually and may take a different route to reach its destination. There is no persistent connection between the sender and receiver as in stream sockets.

Simple and Lightweight: Datagram sockets are often simpler and more lightweight than stream sockets. They are suitable for applications where low overhead and real-time delivery are more important than reliability, such as multimedia streaming, online gaming, DNS (Domain Name System), and SNMP (Simple Network Management Protocol).

Usage: Datagram sockets are typically created using the socket() system call with the appropriate address family (e.g., AF_INET for IPv4) and socket type (e.g., SOCK_DGRAM for datagram sockets). After creating the socket, you can use functions like sendto() and recvfrom() to send and receive datagrams, respectively.

4. Write a server/client model socket program to exchange hello message between them.

Server Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_address, client_address;
    int client_address_len = sizeof(client_address);
    char buffer[BUFFER_SIZE] = {0};
    char *hello_message = "Hello from server";
    // Create server socket
    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
```

```
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Initialize server address structure
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT);

    // Bind the socket to the server address
    if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_socket, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    // Accept incoming connection
    if ((client_socket = accept(server_socket, (struct sockaddr *)&client_address, (socklen_t
*)&client_address_len)) < 0) {
        perror("Acceptance failed");
        exit(EXIT_FAILURE);
    }

    // Receive data from client
    if (recv(client_socket, buffer, BUFFER_SIZE, 0) < 0) {
```

```
        perror("Receive failed");
        exit(EXIT_FAILURE);
    }

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

    // Send hello message to client
    if (send(client_socket, hello_message, strlen(hello_message), 0) < 0) {
        perror("Send failed");
        exit(EXIT_FAILURE);
    }

    printf("Hello message sent to client\n");

    // Close sockets
    close(client_socket);
    close(server_socket);

    return 0;
}
```

Client Program:

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

#define PORT 8080
```

```
#define SERVER_IP "127.0.0.1"
#define BUFFER_SIZE 1024

int main() {
    int client_socket;
    struct sockaddr_in server_address;
    char buffer[BUFFER_SIZE] = {0};
    char *hello_message = "Hello from client";

    // Create client socket
    if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Initialize server address structure
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }  // Connect to server
    if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0)
{
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }
```

```
    // Send hello message to server
    if (send(client_socket, hello_message, strlen(hello_message), 0) < 0) {
        perror("Send failed");
        exit(EXIT_FAILURE);
    }

    printf("Hello message sent to server\n");

    // Receive data from server
    if (recv(client_socket, buffer, BUFFER_SIZE, 0) < 0) {
        perror("Receive failed");
        exit(EXIT_FAILURE);
    }

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

    // Close socket
    close(client_socket);

    return 0;}
```

5.Write a TCP server-client program to check if a given string is Palindrome

Server Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#define PORT 8080
#define BUFFER_SIZE 1024

int isPalindrome(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        if (str[i] != str[len - i - 1]) {
            return 0; // Not a palindrome
        }
    }
    return 1; // Palindrome
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_address, client_address;
    int client_address_len = sizeof(client_address);
    char buffer[BUFFER_SIZE] = {0};
    char *response;

    // Create server socket
    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
    // Initialize server address structure
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT);
```

```
    // Bind the socket to the server address
    if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }
    // Listen for incoming connections
    if (listen(server_socket, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }
    // Accept incoming connection
    if ((client_socket = accept(server_socket, (struct sockaddr *)&client_address, (socklen_t
*)&client_address_len)) < 0) {
        perror("Acceptance failed");
        exit(EXIT_FAILURE);
    }
    // Receive data from client
    if (recv(client_socket, buffer, BUFFER_SIZE, 0) < 0) {
        perror("Receive failed");
        exit(EXIT_FAILURE);
    }
    printf("Received string from client: %s\n", buffer);
    // Check if string is palindrome
    if (isPalindrome(buffer)) {
        response = "Palindrome";
    } else {
        response = "Not a Palindrome";
    }
    // Send response to client
    if (send(client_socket, response, strlen(response), 0) < 0) {
```

```
        perror("Send failed");
        exit(EXIT_FAILURE);
    }
    printf("Response sent to client\n");
    // Close sockets
    close(client_socket);
    close(server_socket);
    return 0;
}
```

Client program:

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

#define PORT 8080
#define SERVER_IP "127.0.0.1"
#define BUFFER_SIZE 1024
int main() {
    int client_socket;
    struct sockaddr_in server_address;
    char buffer[BUFFER_SIZE] = {0};
    char *input_string;

    printf("Enter a string to check if it's a palindrome: ");
    fgets(buffer, BUFFER_SIZE, stdin);
    input_string = strtok(buffer, "\n"); // Remove trailing newline
```

```
    // Create client socket
    if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
    // Initialize server address structure
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);
    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }
    // Connect to server
    if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0)
{
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }
    // Send string to server
    if (send(client_socket, input_string, strlen(input_string), 0) < 0) {
        perror("Send failed");
        exit(EXIT_FAILURE);
    }
    printf("String sent to server\n");

    // Receive response from server
    if (recv(client_socket, buffer, BUFFER_SIZE, 0) < 0) {
        perror("Receive failed");
        exit(EXIT_FAILURE);
```

```
    }
    printf("Response from server: %s\n", buffer);
    // Close socket
    close(client_socket);
    return 0;
}
```

6.Write an example to demonstrate UDP server-client program

UDP Server Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_socket;
    struct sockaddr_in server_address, client_address;
    int client_address_len = sizeof(client_address);
    char buffer[BUFFER_SIZE] = {0};

    // Create server socket
    if ((server_socket = socket(AF_INET, SOCK_DGRAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
```

```
    // Initialize server address structure
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT);

    // Bind the socket to the server address
    if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    printf("UDP Server listening on port %d\n", PORT);

    // Receive data from client
    while (1) {
        int bytes_received = recvfrom(server_socket, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&client_address, (socklen_t *)&client_address_len);
        if (bytes_received < 0) {
            perror("Receive failed");
            exit(EXIT_FAILURE);
        }

        printf("Received from client: %s\n", buffer);

        // Echo back to client
        if (sendto(server_socket, buffer, bytes_received, 0, (struct sockaddr *)&client_address,
client_address_len) < 0) {
            perror("Send failed");
            exit(EXIT_FAILURE);
        }
```

```
    }

    // Close socket
    close(server_socket);

    return 0;
}
```

UDP Client Program:

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

#define PORT 8080
#define SERVER_IP "127.0.0.1"
#define BUFFER_SIZE 1024
int main() {
    int client_socket;
    struct sockaddr_in server_address;
    char buffer[BUFFER_SIZE] = {0};
    char *message = "Hello from client";
    // Create client socket
    if ((client_socket = socket(AF_INET, SOCK_DGRAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
    // Initialize server address structure
```

```
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);
    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }
    // Send message to server
    if (sendto(client_socket, message, strlen(message), 0, (struct sockaddr *)&server_address,
sizeof(server_address)) < 0) {
        perror("Send failed");
        exit(EXIT_FAILURE);
    }
    printf("Message sent to server\n");
    // Receive response from server
    int bytes_received = recvfrom(client_socket, buffer, BUFFER_SIZE, 0, NULL, NULL);
    if (bytes_received < 0) {
        perror("Receive failed");
        exit(EXIT_FAILURE);
    }
    printf("Response from server: %s\n", buffer);

    // Close socket
    close(client_socket);

    return 0;
}
```