IT 307- Exploring the Networks
Lab 7- Socket Programming in C

1. Introduction to Sockets

A socket is an endpoint for sending or receiving data across a computer network. In socket programming, we can establish communication between two machines, using either the TCP (Transmission Control Protocol) or UDP (User Datagram Protocol) protocols.

Sockets allow communication between:

- Two programs on the same machine.
- Two programs on different machines connected by a network.

2. Client-Server Model

In the client-server architecture, a server listens for requests from clients and responds accordingly. The communication typically follows this pattern:

- The server binds to a specific port and listens for incoming client requests.
- The client connects to the server using the server's IP address and port number.
- After establishing a connection, data can be exchanged.

3. Socket Programming Primitives

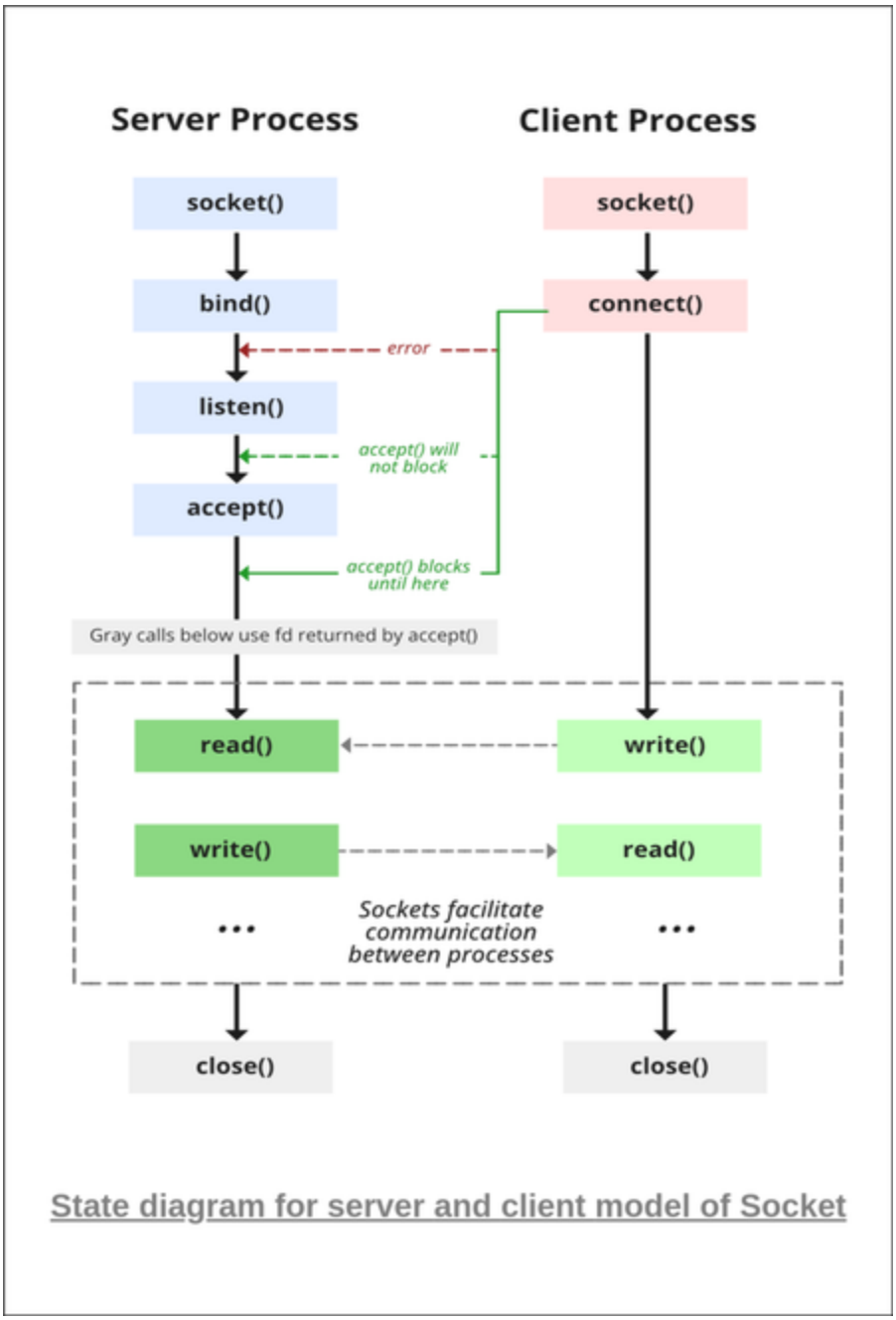
Client-Server Interaction Primitives

Some necessary header files

stdlib.h: Provides functions like `exit()` for exiting the program.

unistd.h: Provides access to POSIX operating system API, including functions like `close()`.

arpa/inet.h: Contains functions for manipulating Internet addresses (IP addresses and port numbers), such as `inet_ntoa()` and `htons()`.



State diagram for server and client model of Socket

Step 1: Socket Creation:

socket(): Creates a new socket. It returns a file descriptor, which represents the socket.

Syntax: *int sockfd = socket(domain, type, protocol);*

Domain: `AF_INET` for IPv4.
Type: `SOCK_STREAM` for TCP, `SOCK_DGRAM` for UDP.
Protocol: `0` (default protocol).

Step 2: Socket Binding:

- bind(): Binds the socket to an IP address and port number.

Syntax: *int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);*

Step 3: Listening and Accepting:

listen(): Marks the socket as passive (ready to accept incoming connections).
accept(): Accepts an incoming connection from a client.

Step 4: Connecting:

connect(): Used by the client to connect to a server.

Step 5: Sending and Receiving Data:

send(), recv() for TCP communication.

sendto(), recvfrom() for UDP communication.

TCP Server Code Explanation

Step 1: Declare Variables

```
int server_fd, new_socket;
struct sockaddr_in address;
int addrlen = sizeof(address);
char buffer[1024] = {0};
const char *message = "Hello from server";
```

- **server_fd:** This will hold the file descriptor for the server socket.
- **new_socket:** After accepting a client connection, the file descriptor for the client connection is stored here.
- **address:** This `struct sockaddr_in` will hold the server's IP address and port.
- **addrlen:** This holds the size of the `address` structure, used during the `accept()` call.
- **buffer:** A buffer used to store messages received from the client (not used in this example).
- **message:** The message the server will send to the client ("Hello from server").

Step 2: Create the Socket

```
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("Socket failed");
    exit(EXIT_FAILURE);
```

}

- socket(AF_INET, SOCK_STREAM, 0)
    - **AF_INET:** Specifies the address family (IPv4 in this case).
    - **SOCK_STREAM:** Specifies that this is a TCP (stream) socket.
    - **0-** Protocol value (0 means the default protocol for the given socket type will be used, which is TCP for `SOCK_STREAM`).

- This line creates the server socket and stores its file descriptor in `server_fd`. If the socket creation fails (e.g., if the system cannot allocate the resources), the program prints an error message using `perror("Socket failed")` and exits with `EXIT_FAILURE`.

Step 3: Configure the Server Address

```
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(8080);
```

- **address.sin_family = AF_INET:** Specifies that this socket will use the IPv4 address family.
- **address.sin_addr.s_addr = INADDR_ANY**: Binds the server to all available network interfaces on the machine. This means that the server will listen for connections on any IP address assigned to the host machine.
- **address.sin_port = htons(8080)**: Specifies the port number (8080) in ***network byte order***. The function `htons()` (host-to-network short) converts the port number from host byte order to network byte order (which ensures compatibility across different architectures).

Step 4: Bind the Socket to the IP and Port

```
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
   perror("Bind failed");
   exit(EXIT_FAILURE);
}
```

bind(): This function binds the server socket (`server_fd`) to the specified IP address and port (`address`). If the binding fails (e.g., if the port is already in use), the program prints a "Bind failed" error message and exits.

Step 5: Set the Socket to Listen for Incoming Connections

```
if (listen(server_fd, 3) < 0) {
   perror("Listen failed");
   exit(EXIT_FAILURE);
```

}

listen(server_fd, 3): This function tells the operating system that this socket should listen for incoming connections. The second argument (`3`) specifies the backlog, which is the maximum number of pending connections that can be queued before the server starts rejecting new connections. In this case, a maximum of 3 connections can be queued.

Step 6: Accept an Incoming Client Connection

```
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {
   perror("Accept failed");
   exit(EXIT_FAILURE);
}
```

- **accept():** This function accepts an incoming connection from a client.
- **new_socket:** The file descriptor for the accepted connection (client socket) is stored in `new_socket`.
    - The function returns the client's socket descriptor or `-1` if an error occurs.
    - If an error occurs (such as if the system runs out of resources or no connections are available), the program prints "Accept failed" and exits.

Step 7: Send a Message to the Client

```
send(new_socket, message, strlen(message), 0);
printf("Message sent to client\n");
```

- **send(new_socket, message, strlen(message), 0)**: This function sends the string "Hello from server" to the client over the `new_socket` (the file descriptor representing the client connection).
    - new_socket: The file descriptor representing the client's socket.
    - message: The message to be sent.
    - strlen(message): The length of the message.
    -0: Optional flags (none are used here).

After sending the message, the server prints a confirmation message: "Message sent to client".

Step 8: Close the Sockets

```
close(new_socket);
close(server_fd);
```

- **close(new_socket):** Closes the client socket once the communication is complete.

- **close(server_fd):** Closes the server socket to free up system resources.

Code Implementation- TCP Server (TCP_server.c)

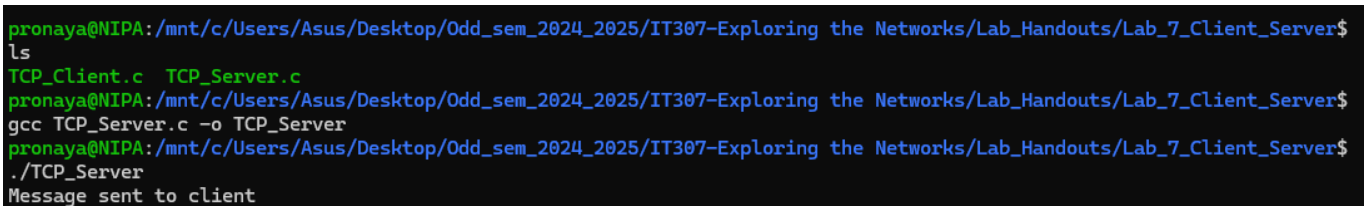
This code provides a simple implementation of a TCP server in C that listens for connections on port 8080, accepts a client connection, sends a message to the client, and then terminates the connection.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    const char *message = "Hello from server";
    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }
    // Define the server address
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(8080);
    // Binding the socket to the port
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }
    // Listening for client connections
    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    // Accepting the connection
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0)
{
```

```
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }
    // Sending a message to the client
    send(new_socket, message, strlen(message), 0);
    printf("Message sent to client\n");
    close(new_socket);
    close(server_fd);
    return 0;
}
```

Server Snapshot-



TCP Client (TCP_client.c)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    // Creating socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(8080);
    // Converting address to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not supported \n");
```

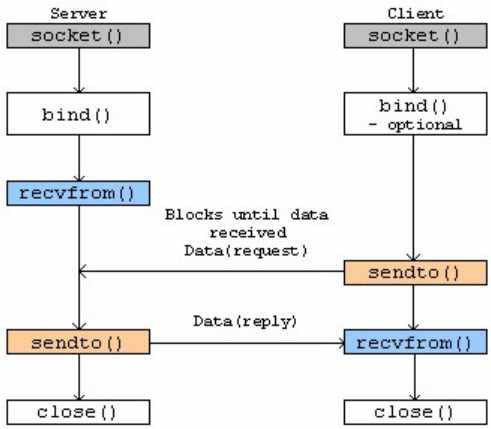
```
        return -1;
    }
    // Connecting to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }
    // Reading the server's response
    read(sock, buffer, 1024);
    printf("Message from server: %s\n", buffer);
    close(sock);
    return 0;
}
```

Client Snapshot

```
pronaya@NIPA:/mnt/c/users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
gcc TCP_Client.c -o TCP_Client
pronaya@NIPA:/mnt/c/users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
./TCP_Client
Message from server: Hello from server
```

Note: To run the TCP server and client on different machines, update the client code by replacing `"127.0.0.1"` with the actual IP address of the server (e.g., `"192.168.1.10"`). This ensures the client connects to the server across the network. Additionally, ensure that the server listens on INADDR_ANY (as it does by default) and that firewall settings on both machines allow traffic on the specified port (e.g., `8080`). If necessary, configure port forwarding for the server if the client is outside the local network.

UDP Client-Server Implementation



Explanation of UDP Server Code-

1. **UDP** uses SOCK_DGRAM (datagram) instead of SOCK_STREAM (used in TCP). This specifies that the communication will be connectionless, meaning no need for a connection establishment (like the TCP three-way handshake).
2. **recvfrom()** is used in **UDP** to receive data from the client. It does not rely on an established connection, so it requires both the socket and the client's address (cliaddr) to be passed in. The recvfrom() function captures both the data and the client's address in one call.
3. **sendto()** is used to send data in **UDP**, which, unlike TCP, does not maintain a persistent connection. The client's address (cliaddr) must be explicitly specified with every send operation to direct the message to the correct recipient.
4. Unlike **TCP**, **UDP** does not have connection management steps (listen() and accept() in TCP). This is because **UDP** is connectionless, meaning the server is always ready to receive messages without establishing or maintaining a connection.
5. In **UDP**, the client and server always explicitly specify the address structures (sockaddr_in) for sending and receiving messages (sendto() and recvfrom()), whereas **TCP** tracks these addresses as part of the connection state.

Implementation Code- UDP Server (UDP_Server.c)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
int main() {
   int sockfd;
   char buffer[1024];
   const char *message = "Hello from UDP server";
   struct sockaddr_in servaddr, cliaddr;
   // Creating socket file descriptor
   if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      perror("Socket creation failed");
      exit(EXIT_FAILURE);
   }
   memset(&servaddr, 0, sizeof(servaddr));
   memset(&cliaddr, 0, sizeof(cliaddr));
   // Filling server information
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr = INADDR_ANY;
   servaddr.sin_port = htons(8080);
   // Binding the socket with the server address
```

```
    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }
    int len, n;
    len = sizeof(cliaddr);
    // Receive message from client
    n = recvfrom(sockfd, (char *)buffer, 1024, MSG_WAITALL, (struct sockaddr *)&cliaddr, &len);
    buffer[n] = '\0';
    printf("Client: %s\n", buffer);
    // Send response to client
    sendto(sockfd, (const char *)message, strlen(message), MSG_CONFIRM, (const struct sockaddr
*)&cliaddr, len);
    printf("Message sent to client\n");
    close(sockfd);
    return 0;
}
```

Server Snapshot

```
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
gcc UDP_Server.c -o UDP_Server
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
./UDP_Server
Client: Hello from UDP client
Message sent to client
```

Implementation of UDP Client (UDP_Client.c)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
int main() {
    int sockfd;
    char buffer[1024];
    const char *message = "Hello from UDP client";
    struct sockaddr_in servaddr;
    // Creating socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
```

```
    memset(&servaddr, 0, sizeof(servaddr));
    // Server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(8080);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    // Send message to server
    sendto(sockfd, (const char *)message, strlen(message), MSG_CONFIRM, (const struct sockaddr
*)&servaddr, sizeof(servaddr));
    // Receive message from server
    int n, len;
    len = sizeof(servaddr);
    n = recvfrom(sockfd, (char *)buffer, 1024, MSG_WAITALL, (struct sockaddr *)&servaddr, &len);
    buffer[n] = '\0';
    printf("Server: %s\n", buffer);
    close(sockfd);
    return 0;
}
```

Client Snapshot

```
pronaya@NIPA:/mnt/c/users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
gcc UDP_Client.c -o UDP_Client
pronaya@NIPA:/mnt/c/users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
./UDP_Client
Server: Hello from UDP server
```

Note: In case UDP client and server are on different machines

Replace INADDR_ANY with the **server's IP address**, for example, "192.168.1.10".

```
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(8080);
// Replace INADDR_ANY with the server's IP address
inet_pton(AF_INET, "192.168.1.10", &servaddr.sin_addr);
```

TCP Chat application between client and server

TCP_Chat_Server.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
int main() {
```

```
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    const char *exit_message = "exit";

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    // Define the server address
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(8080);

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

    // Listening for client connections
    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }
    printf("Server is listening on port 8080...\n");
    // Accepting the connection
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0)
{
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }
    printf("Connection established with the client!\n");
    while (1) {
        // Receive message from client
        memset(buffer, 0, sizeof(buffer));
        int n = read(new_socket, buffer, 1024);
        printf("Client: %s\n", buffer);

        // Exit condition
        if (strncmp(buffer, exit_message, 4) == 0) {
            printf("Client has exited the chat.\n");
            break;
```

```
        }

        // Send message to client
        printf("Enter message: ");
        fgets(buffer, 1024, stdin);
        buffer[strcspn(buffer, "\n")] = 0;  // Remove newline character
        send(new_socket, buffer, strlen(buffer), 0);

        // Exit condition for server
        if (strncmp(buffer, exit_message, 4) == 0) {
            printf("Server has exited the chat.\n");
            break;
        }
    }

    close(new_socket);
    close(server_fd);
    return 0;
}
```

TCP_Chat_Client.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    const char *exit_message = "exit";

    // Creating socket
    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(8080);
```

```
    // Convert address to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("Invalid address / Address not supported\n");
        return -1;
    }

    // Connecting to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("Connection failed\n");
        return -1;
    }

    printf("Connected to the server!\n");

    while (1) {
        // Send message to server
        printf("Enter message: ");
        fgets(buffer, 1024, stdin);
        buffer[strcspn(buffer, "\n")] = 0;  // Remove newline character
        send(sock, buffer, strlen(buffer), 0);

        // Exit condition for client
        if (strncmp(buffer, exit_message, 4) == 0) {
            printf("Client has exited the chat.\n");
            break;
        }

        // Receive message from server
        memset(buffer, 0, sizeof(buffer));
        int n = read(sock, buffer, 1024);
        printf("Server: %s\n", buffer);

        // Exit condition if server sends exit
        if (strncmp(buffer, exit_message, 4) == 0) {
            printf("Server has exited the chat.\n");
            break;
        }
    }

    close(sock);
    return 0;
}
```

Server Snapshot

```
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
gcc TCP_Chat_Server.c -o TCP_Chat_Server
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
./TCP_Chat_Server
Server is listening on port 8080...
Connection established with the client!
Client: Hello Server how are you??
Enter message: I am fine Client..!!!
Client: Great weather
Enter message: Yes, it is nice weather
Client: exit
Client has exited the chat.
```

Client Snapshot

```
pronaya@NIPA:/mnt/c/users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
gcc TCP_Chat_Client.c -o TCP_Chat_Client
pronaya@NIPA:/mnt/c/users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
./TCP_Chat_Client
Connected to the server!
Enter message: Hello Server how are you??
Server: I am fine Client..!!!
Enter message: Great weather
Server: Yes, it is nice weather
Enter message: exit
Client has exited the chat.
```

UDP Multicasting (Group Chat Formation)

Basic Concept: UDP multicasting allows data to be sent from one sender to multiple receivers using a multicast group address.

UDP Multicast Server

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>

#define MULTICAST_GROUP "239.0.0.1"
#define PORT 8080

int sockfd;
```

```
void handle_sigint(int sig) {
    printf("\nCaught signal %d. Exiting multicast server gracefully...\n", sig);
    close(sockfd);
    exit(0);
}

int main() {
    struct sockaddr_in multicast_addr, client_addr;
    char message[1024];
    socklen_t client_len = sizeof(client_addr);

    // Register the signal handler for SIGINT (Ctrl + C)
    signal(SIGINT, handle_sigint);

    // Create a UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Allow multiple sockets to reuse the address and port
    int reuse = 1;
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse)) < 0) {
        perror("Setting SO_REUSEADDR failed");
        exit(EXIT_FAILURE);
    }

    // Bind the socket to the multicast group port
    memset(&multicast_addr, 0, sizeof(multicast_addr));
    multicast_addr.sin_family = AF_INET;
    multicast_addr.sin_addr.s_addr = htonl(INADDR_ANY); // Bind to any interface
    multicast_addr.sin_port = htons(PORT);

    if (bind(sockfd, (struct sockaddr *)&multicast_addr, sizeof(multicast_addr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }
```

```
    printf("UDP Multicast Server is running. Type messages to send to the group:\n");

    // Multicast message to all clients
    while (1) {
        printf("Enter message to multicast (type 'exit' to quit): ");
        fgets(message, sizeof(message), stdin);
        message[strcspn(message, "\n")] = 0;  // Remove newline character

        // Check if the server wants to quit
        if (strncmp(message, "exit", 4) == 0) {
            printf("Exiting multicast server.\n");
            break;
        }

        // Send the message to the multicast group
        multicast_addr.sin_family = AF_INET;
        multicast_addr.sin_addr.s_addr = inet_addr(MULTICAST_GROUP);
        multicast_addr.sin_port = htons(PORT);

        if (sendto(sockfd, message, strlen(message), 0, (struct sockaddr *)&multicast_addr,
sizeof(multicast_addr)) < 0) {
            perror("Multicast sendto() failed");
            exit(EXIT_FAILURE);
        }

        printf("Multicast message sent to group %s: %s\n", MULTICAST_GROUP, message);

        // Check for any messages from clients
        int n = recvfrom(sockfd, message, sizeof(message) - 1, MSG_DONTWAIT, (struct
sockaddr *)&client_addr, &client_len);
        if (n > 0) {
            message[n] = '\0';
            printf("Received message from client: %s\n", message);
        }
    }

    close(sockfd);
    return 0;
}
```

UDP Multicast Client

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>

#define MULTICAST_GROUP "239.0.0.1"
#define PORT 8080

int sockfd;

void handle_sigint(int sig) {
    printf("\nCaught signal %d. Exiting multicast client gracefully...\n", sig);
    close(sockfd);
    exit(0);
}

int main() {
    struct sockaddr_in multicast_addr, server_addr;
    struct ip_mreq multicast_request;
    char message[1024];
    socklen_t server_len = sizeof(server_addr);

    // Register the signal handler for SIGINT (Ctrl + C)
    signal(SIGINT, handle_sigint);

    // Create a UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Allow multiple sockets to reuse the port
    int reuse = 1;
```

```
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse)) < 0) {
        perror("Setting SO_REUSEADDR failed");
        exit(EXIT_FAILURE);
    }

    // Set up the address for the multicast group to bind to
    memset(&multicast_addr, 0, sizeof(multicast_addr));
    multicast_addr.sin_family = AF_INET;
    multicast_addr.sin_addr.s_addr = htonl(INADDR_ANY); // Accept any incoming messages
    multicast_addr.sin_port = htons(PORT);

    // Bind the socket to the multicast port
    if (bind(sockfd, (struct sockaddr *)&multicast_addr, sizeof(multicast_addr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Join the multicast group
    multicast_request.imr_multiaddr.s_addr = inet_addr(MULTICAST_GROUP); // Multicast
group address
    multicast_request.imr_interface.s_addr = htonl(INADDR_ANY);         // Use default network
interface
    if (setsockopt(sockfd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &multicast_request,
sizeof(multicast_request)) < 0) {
        perror("Joining multicast group failed");
        exit(EXIT_FAILURE);
    }

    printf("UDP Multicast Client is running. You can send messages to the server.\n");

    fd_set readfds;
    int max_sd = sockfd;

    while (1) {
        FD_ZERO(&readfds);
        FD_SET(0, &readfds);  // Standard input (keyboard)
        FD_SET(sockfd, &readfds);  // Client socket (receiving messages)

        int activity = select(max_sd + 1, &readfds, NULL, NULL, NULL);
```

```
        if (FD_ISSET(0, &readfds)) {
            // Get user input
            printf("Enter message to send (type 'exit' to quit): ");
            fgets(message, sizeof(message), stdin);
            message[strcspn(message, "\n")] = 0;  // Remove newline character

            // Exit condition for client
            if (strncmp(message, "exit", 4) == 0) {
                printf("Client exiting...\n");
                break;
            }

            // Send message to the server
            server_addr.sin_family = AF_INET;
            server_addr.sin_addr.s_addr = inet_addr(MULTICAST_GROUP); // Send to multicast
group
            server_addr.sin_port = htons(PORT);

            if (sendto(sockfd, message, strlen(message), 0, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0) {
                perror("Sending message to server failed");
                exit(EXIT_FAILURE);
            }

            printf("Message sent to server: %s\n", message);
        }

        if (FD_ISSET(sockfd, &readfds)) {
            // Receive multicast message from the server
            int n = recvfrom(sockfd, message, sizeof(message) - 1, 0, NULL, NULL);
            if (n > 0) {
                message[n] = '\0';
                printf("Received multicast message: %s\n", message);
            }
        }
    }

    close(sockfd);
    return 0;
}
```

Server Snapshot-

```
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
gcc  UDP_multicast_server.c -o mserver
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
./mserver
UDP Multicast Server is running. Type messages to send to the group:
Enter message to multicast (type 'exit' to quit): Hello Group Members Client 1 and Client 2
Multicast message sent to group 239.0.0.1: Hello Group Members Client 1 and Client 2
Received message from client: Hello Group Members Client 1 and Client 2
Enter message to multicast (type 'exit' to quit): How are you both
Multicast message sent to group 239.0.0.1: How are you both
Received message from client: Hello Server, this is Client 1
Enter message to multicast (type 'exit' to quit): server is leaving now
Multicast message sent to group 239.0.0.1: server is leaving now
Received message from client: Hello Server, this is Client 2
Enter message to multicast (type 'exit' to quit): exit
Exiting multicast server.
```

Client 1 in Group 239.0.0.1 (MULTICAST_GROUP)

```
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
gcc  UDP_multicast_client.c -o mclient1
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
./mclient1
UDP Multicast Client is running. You can send messages to the server.
Received multicast message: Hello Group Members Client 1 and Client 2
Hello Server, this is Client 1
Enter message to send (type 'exit' to quit): Message sent to server: Hello Server, this is Client 1
Received multicast message: Hello Server, this is Client 1
Received multicast message: Hello Server, this is Client 2
Received multicast message: How are you both
Client 1 is fine
Enter message to send (type 'exit' to quit): Message sent to server: Client 1 is fine
Received multicast message: Client 1 is fine
Received multicast message: Client2 is fine
Client 1 is leaving now
Enter message to send (type 'exit' to quit): Message sent to server: Client 1 is leaving now
Received multicast message: Client 1 is leaving now
exit
Enter message to send (type 'exit' to quit): Client exiting...
```

Client 2 in Group 239.0.0.1 (MULTICAST_GROUP)

```
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
gcc  UDP_multicast_client.c -o mclient2
pronaya@NIPA:/mnt/c/Users/Asus/Desktop/Odd_sem_2024_2025/IT307-Exploring the Networks/Lab_Handouts/Lab_7_Client_Server$
./mclient2
UDP Multicast Client is running. You can send messages to the server.
Received multicast message: Hello Group Members Client 1 and Client 2
Received multicast message: Hello Server, this is Client 1
Hello Server, this is Client 2
Enter message to send (type 'exit' to quit): Message sent to server: Hello Server, this is Client 2
Received multicast message: Hello Server, this is Client 2
Received multicast message: How are you both
Received multicast message: Client 1 is fine
Client2 is fine
Enter message to send (type 'exit' to quit): Message sent to server: Client2 is fine
Received multicast message: Client2 is fine
Received multicast message: Client 1 is leaving now
client 2 is leaving now
Enter message to send (type 'exit' to quit): Message sent to server: client 2 is leaving now
Received multicast message: client 2 is leaving now
exit
Enter message to send (type 'exit' to quit): Client exiting...
```