EX.NO: 07	Study of Socket Programming and client server model using UDP AND TCP
DATE:	

Aim

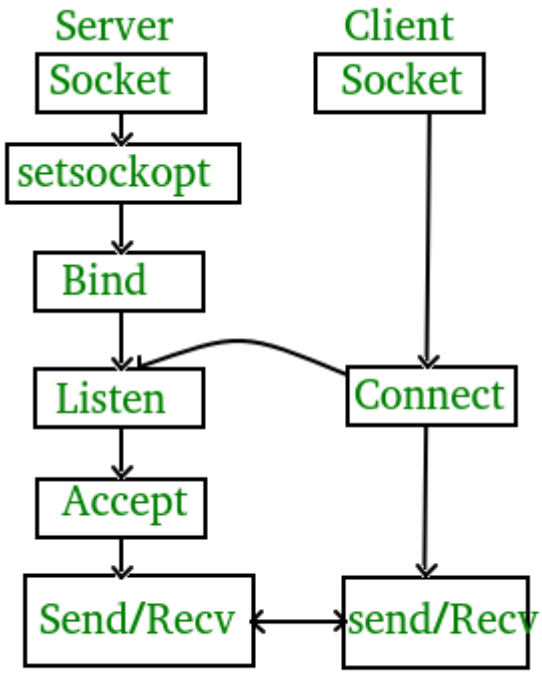
To understand the functioning and differences between hubs, switches, routers, gateways, and modems in a network environment.

Software Required

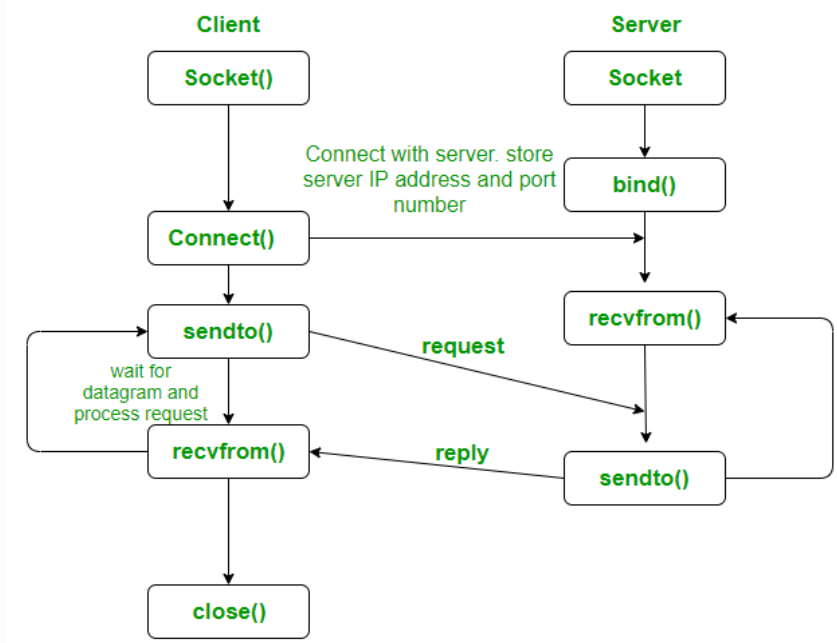
- Linux operating system
- C programming language
- Text editor (e.g. Vim, Nano)
- Terminal emulator (e.g. GNOME Terminal, Konsole

Flowchart

- **TCP**



- **UDP**



Lab Setup:

1.Setting up the environment

- Ensure that you have a Linux environment with a C compiler installed (e.g., gcc).
- Open a terminal

2.Creating the Project Directory

- **Create a new directory for your project.**

Part 1:Server Side

Step 1:Writing the Server Code

- **Create a file named server .c in the project directory.**

1.// server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#define PORT 8080
```

```
#define MAX_BUFFER_SIZE 1024
int main() {
int server_fd, new_socket, valread;
struct sockaddr_in address;
int addrlen = sizeof(address);
char buffer[MAX_BUFFER_SIZE] = {0};
// Create a socket
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
perror("Socket creation failed");
exit(EXIT_FAILURE);
}
// Set up server address struct
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Bind the socket to the address
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
perror("Bind failed");
exit(EXIT_FAILURE);
}
// Listen for incoming connections
if (listen(server_fd, 3) < 0) {
perror("Listen failed");
exit(EXIT_FAILURE);
}
// Accept incoming connection
if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
perror("Accept failed");
exit(EXIT_FAILURE);
}
// Read data from the client using TCP
valread = read(new_socket, buffer, MAX_BUFFER_SIZE);
printf("Received message from client: %s\n", buffer);
// Close the connection
close(new_socket);
```

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

Step 2 :Compiling and Running the Server Code

- **Compile the server code.**

```
        Gcc server.c -o server
        Run the server
```

- **./server**

Part 2: Client Side

Step 1:Writing the Client Code

- **Create a file named client.c in the project directory**

1.// client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#define PORT 8080
#define MAX_BUFFER_SIZE 1024
int main() {
int client_fd;
struct sockaddr_in server_address;
char message[MAX_BUFFER_SIZE];

// Create a socket
if ((client_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
perror("Socket creation failed");
exit(EXIT_FAILURE);
}
// Configure server address
server_address.sin_family = AF_INET;
```

```
server_address.sin_port = htons(PORT);
if (inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr) <= 0) {
perror("Invalid address/ Address not supported");
exit(EXIT_FAILURE);
}
// Connect to the server using TCP
if (connect(client_fd, (struct sockaddr *)&server_address,
sizeof(server_address)) < 0) {
perror("Connection Failed");
exit(EXIT_FAILURE);
}
// Get user input for the message
printf("Enter a message to send to the server: ");
fgets(message, MAX_BUFFER_SIZE, stdin);
// Send the message to the server using TCP
send(client_fd, message, strlen(message), 0);
// Close the connection
close(client_fd);
return 0;
}
```

Step 2 : Compiling and Running the Client Code

- **Compile the client code**

    Gcc client.c -o client

    Run the client

In both the server and client code, the SOCK_STREAM parameter in the socket function indicates the use of TCP. This sets up a reliable, connection-oriented communication channel between the client and theserver. The subsequent read and send functions are used for reading from and writing to the TCP socket, respectively.

OUTPUT:-

Client Side

```
user@administrator-ThinkCentre-M72e:~/tcp_message_lab$ gcc server.c -o server
user@administrator-ThinkCentre-M72e:~/tcp_message_lab$ ./server
Received message from client: Hello, World!
```

Server Side:-

```
user@administrator-ThinkCentre-M72e:~/tcp_message_lab$ gcc client.c -o client
user@administrator-ThinkCentre-M72e:~/tcp_message_lab$ ./client
Enter a message to send to the server: Hello, World!
user@administrator-ThinkCentre-M72e:~/tcp_message_lab$
```

Conclusion

Through this lab, participants gained practical experience in socket programming and the client-server model using both UDP and TCP protocols. They learned how to establish a connection between a client and server, send and receive messages, and analyze the differences between UDP and TCP. By completing this lab, participants are now equipped with the knowledge and skills to develop more complex network applications and troubleshoot network-related issues in the future.