Study of Socket Programming and Client Server model using UDP AND TCP

Aim

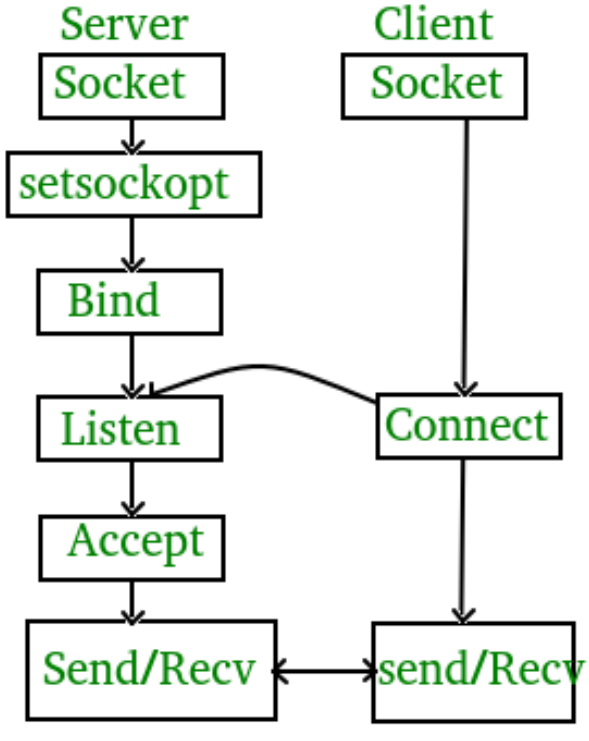
To implement socket programming to send and receive between client and server using TCP and UDP Sockets.

Software Required

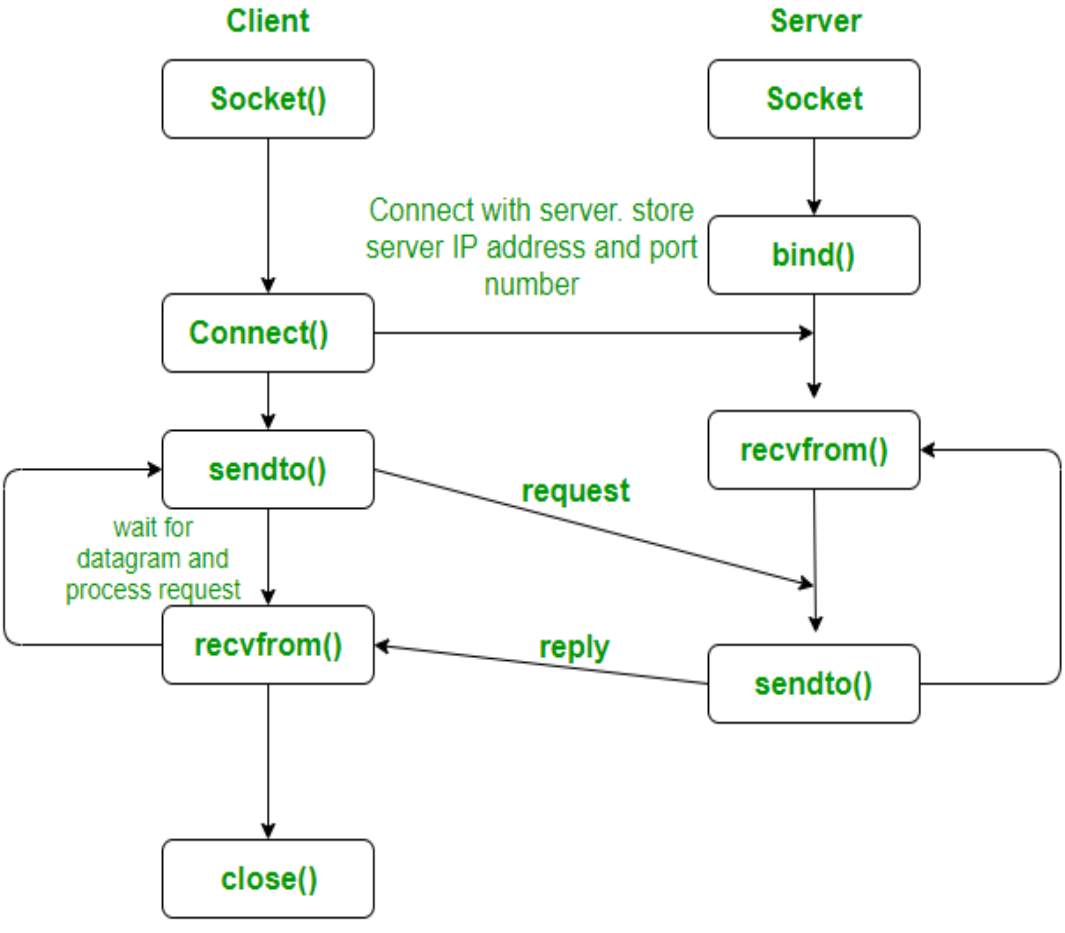
- Linux operating system
- C programming language
- Text editor (e.g. Vim, Nano)
- Terminal emulator (e.g. GNOME Terminal, Konsole)

Flowchart:

TCP



UDP:


Client
Server
Socket()
Socket
Connect with server. store server IP address and port number
bind()
Connect()
recvfrom()
sendto()
request
wait for datagram and process request
recvfrom()
reply
sendto()
close()

Server

```
#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;
}
```

Client

```
#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); }
if (connect(client_fd, (struct sockaddr *)&server_address,
sizeof(server_address)) < 0) {
perror("Connection Failed");
exit(EXIT_FAILURE); }
printf("Enter a message to send to the server: ");
fgets(message, MAX_BUFFER_SIZE, stdin);
send(client_fd, message, strlen(message), 0);
close(client_fd);
return 0;
}
```

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 the server. The subsequent read and send functions are used for reading from and writing to the TCP socket, respectively.

OUTPUT:-

Client

```
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$
```

Server

```
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!
```

Conclusion

Thus, the connection establishment between a client and server, process of sending and receiving messages, and the differences between UDP and TCP has been studied successfully.