

Server-Client Chat
Using
Transmission Control Protocol (TCP)



A
Linux Programming Project Report

Bachelor of Technology
In
Computer Science & Engineering

By
P. NITTHIN REDDY 2103A53009

Under the Guidance of
Mr. T. SAMPATH KUMAR
Professor, School of Computer Science and Engineering

Submitted to





DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the Linux Programming – Course Project Report entitled “**Server-Client Chat using Transmission Control Protocol (TCP)**” is a record of bonafide work carried out by the student **P. NITHIN REDDY** bearing Roll No. 2103A53009 during the academic year 2022-23 in partial fulfilment of the award of the degree of **Bachelor of technology in Computer Science & Engineering** by the SR University.

Lab In-charge

Head of the Department

TABLE OF CONTENTS

TOPICS	PAGE NO
Abstract	4
Objective of project	5
Definition of elements used in project	6 - 8
Design	9
Source Code	10 - 13
Output	14 - 17
Conclusion	18

ABSTRACT

Transmission Control Protocol is referred to as TCP. The packet transmission from source to destination is made easier by this transport layer protocol. Since it is a connection-oriented protocol, communication between computer devices in a network begins with the link being established. Together, they are referred to as a TCP/IP since they are utilised with an IP protocol. The TCP's primary role is to retrieve data from the application layer. The data is then divided into numerous packets, each of which is given a number, before being transmitted to the destination. The packets are reassembled by the TCP and sent to the application layer from the opposite side. Since TCP is a connection-oriented protocol, the connection will continue to exist as long as the sender and receiver are still in contact.

OBJECTIVE

- Gaining expertise in creating a network application based on a client/server architecture is the goal of this project.
- By creating a chat application that consists of a network of two connected Servers and Client using the sockets API.
- To terminate the chat, we have to go for hardware interrupt signal, i.e., ctrl+c.

DEFINITIONS OF THE ELEMENTS USED IN THE PROJECT

- Here I used basics of TCP protocol.
- Among the two users, I assumed one is server and the other is client.
- For Server the following functions are included. They are:
 - Socketfd (Socket Creation)
 - Bind
 - Listen
 - Accept
 - Read
 - Write
 - Close
- For Client the following functions are included. They are:
 - Socketfd (Socket Creation)
 - Connect
 - Write
 - Read
 - Close

STAGES OF SERVER

1. Socket Creation:

int sockfd = socket (domain, type, protocol);

- a. **Socketfd:** Socket descriptor, an integer. If its value is non negative integer, then that means socket created. Else If the value is -1, then there is an error in creating socket.
- b. **Domain:** Communication domain is indicated by an integer. For interprocess communication on the same host, we use the AF_LOCAL provided in the POSIX standard. We utilise AF_INET for processes connected by IPV4 and AF_INET6 for processes linked by IPV6 to communicate with one another.
- c. **Type:** This specifies communication type. For TCP (reliable, connection oriented) communication we specify it as “SOCK_STREAM” and for UDP

(unreliable, connectionless) communication we specify it as “SOCK_DGRAM”.

- d. **Protocol:** Internet Protocol (IP) has a protocol value of 0. This is the same number that is displayed in the protocol field of a packet's IP header.

2. Bind:

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

- The bind function ties the newly created socket to the address and port number supplied in addr. In the sample code, the server is bound to localhost, therefore the IP address is specified using INADDR_ANY.

3. Listen:

int listen (int sockfd, int backlog);

- It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

4. Accept:

*int new_socket = accept (int sockfd, struct sockaddr *addr, socklen_t *addrlen);*

- It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, the connection is established between client and server, and they are ready to transfer data.

5. Read and Write Operations:

- There will be alternate read and write operations done by the server.
- For a corresponding write operation from client, there will be a corresponding read operation from the server.
- For a corresponding read operation from client, there will be a corresponding write operation from the server.

6. Close:

- This will close or disconnect the server and the client.

STAGES OF CLIENT

1. Socket Connection:

int sockfd = socket (domain, type, protocol);

- a. **Sockfd:** Socket descriptor, an integer. If its value is non negative integer, then that means socket created. Else If the value is -1, then there is an error in creating socket.
 - b. **Domain:** Communication domain is indicated by an integer. For interprocess communication on the same host, we use the AF_LOCAL provided in the POSIX standard. We utilise AF_INET for processes connected by IPV4 and AF_INET6 for processes linked by IPV6 to communicate with one another.
 - c. **Type:** This specifies communication type. For TCP (reliable, connection oriented) communication we specify it as “SOCK_STREAM” and for UDP (unreliable, connectionless) communication we specify it as “SOCK_DGRAM”.
- **Protocol:** Internet Protocol (IP) has a protocol value of 0. This is the same number that is displayed in the protocol field of a packet's IP header.

2. Connect:

*int connect (int sockfd, const struct sockaddr *addr, socklen_t addrlen);*

- The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

3. Read and Write Operations:

- There will be alternate read and write operations done by the client.
- For a corresponding read operation from server, there will be a corresponding write operation from the client.
- For a corresponding write operation from server, there will be a corresponding read operation from the client.

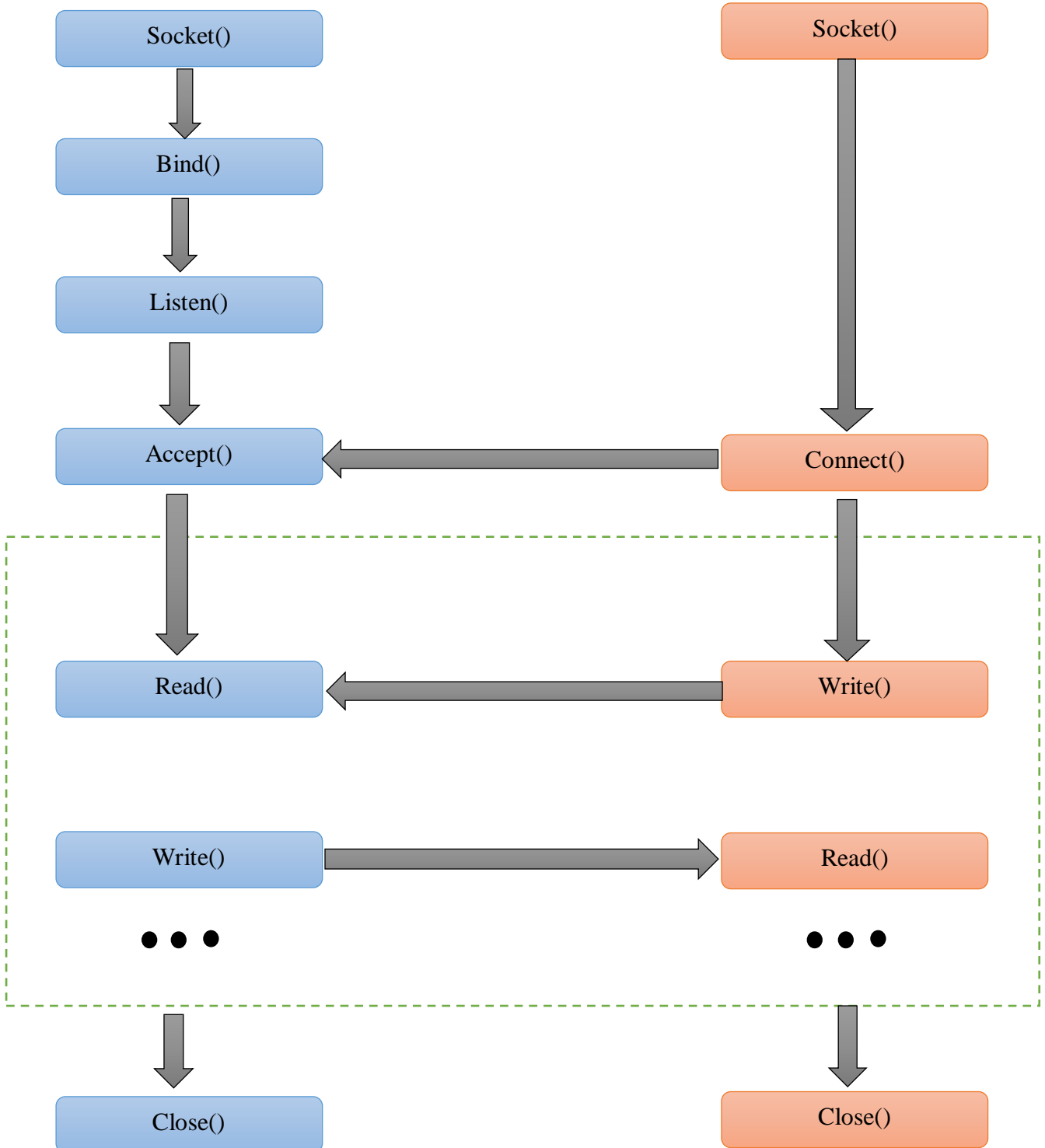
4. Close:

- This will close or disconnect the server and the client.

DESIGN

Server Process

Client Process



Source Code

Server.c

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<arpa/inet.h>

int main()

{

    char *ip = "127.0.0.1:;

    int port = 8080;

    int server_sock, client_sock;

    struct sockaddr_in server_addr, client_addr;

    socklen_t addr_size;

    char buffer[1024];

    int n;

    server_sock = socket(AF_INET, SOCK_STREAM, 0);

    if(server_sock < 0){

        perror("[-]Socket error");

        exit(1);

    }

    printf("[+]TCP Server Socket Created.\n");

    memset(&server_addr, '\0', sizeof(server_addr));
```

```

server_addr.sin_family = AF_INET;

server_addr.sin_port = port;

server_addr.sin_addr.s_addr = inet_addr(ip);

n = bind(server_sock, (struct sockaddr *)&server_addr, sizeof(server_addr));

if(n < 0){

    perror("[-]Bind Error:");

    exit(1);

}

printf("[+]Bind to the port number: %d\n", port);

listen(server_sock, 5);

printf("Listening....\n");

while(1){

    addr_size = sizeof(client_addr);

    client_sock = accept(server_sock, (struct sockaddr *)&client_addr,
&addr_size);

    printf("[+]Client Connected.\n");

    while(1){

        bzero(buffer, 1024);

        recv(client_sock, buffer, sizeof(buffer), 0);

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

        bzero(buffer, 1024);

        printf("Your Message:");

        gets(buffer);

        send(client_sock, buffer, sizeof(buffer), 0);

    }

```

```

        close(client_sock);

        printf("Disconnected from the server.\n");
    }

    return 0;
}

```

Client.c

```

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<arpa/inet.h>

int main()
{
    char *ip = "127.0.0.1:;

    int port = 8080;

    int server_sock, client_sock;

    struct sockaddr_in server_addr, client_addr;

    socklen_t addr_size;

    char buffer[1024];

    int n;

    server_sock = socket(AF_INET, SOCK_STREAM, 0);

    if(server_sock < 0){

        perror("[-]Socket error");
    }
}

```

```

        exit(1);
    }

    printf("[+]TCP Server Socket Created.\n");

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

    addr.sin_family = AF_INET;

    addr.sin_port = port;

    addr.sin_addr.s_addr = inet_addr(ip);

    connect(sock, (struct sockaddr *)&addr, sizeof(addr));

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

    while(1){

        bzero(buffer, 1024);

        printf("Your Message:");

        gets(buffer);

        send(sock, buffer, sizeof(buffer), 0);

        bzero(buffer, 1024);

        recv(sock, buffer, sizeof(buffer), 0);

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

    }

    close(sock);

    printf("Disconnected from the server.\n");

    return 0;
}

```

OUTPUT

Step 1: Compile Server.c and run server.c. But compile client.c but don't run. Output will be like server will wait for client to connect as shown below:

Server Output:

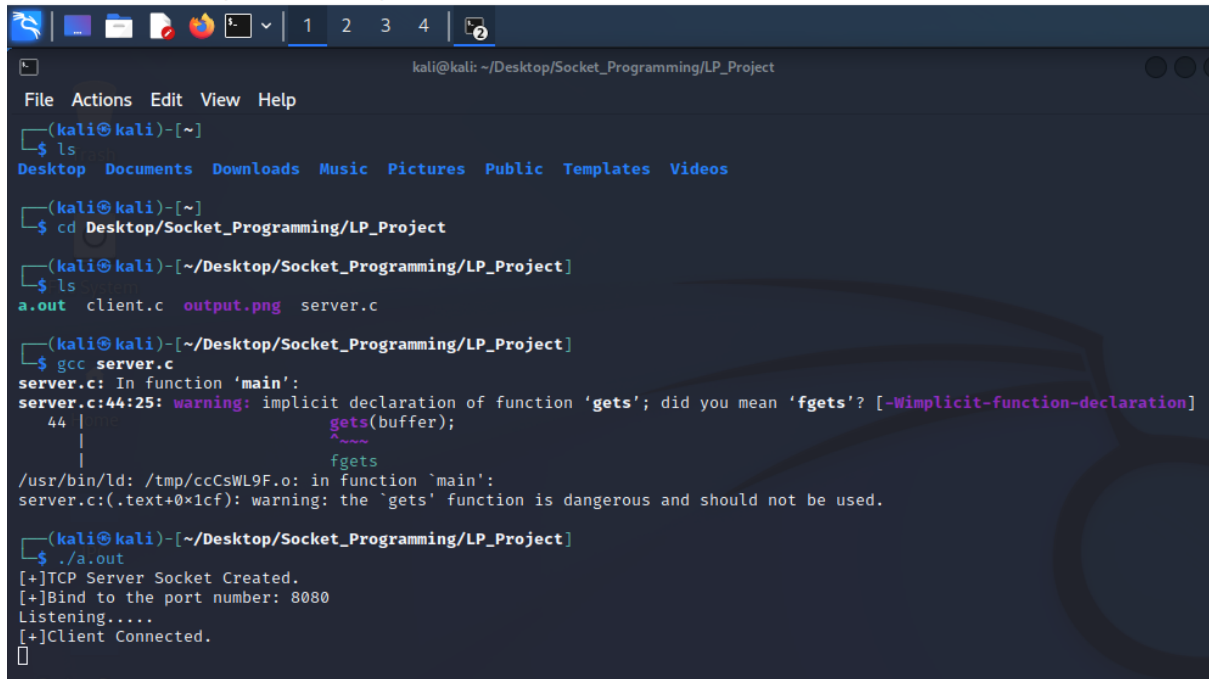
```
kali@kali: ~/Desktop/Socket_Programming/LP_Project
File Actions Edit View Help
(kali@kali)-[~]
$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
(kali@kali)-[~]
$ cd Desktop/Socket_Programming/LP_Project
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ls
a.out client.c output.png server.c
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ gcc server.c
server.c: In function 'main':
server.c:44:25: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   44 |         gets(buffer);
      |         ^~~~~
      |         fgets
/usr/bin/ld: /tmp/ccCsWL9F.o: in function 'main':
server.c:(.text+0x1cf): warning: the 'gets' function is dangerous and should not be used.
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ./a.out
[+]TCP Server Socket Created.
[+]Bind to the port number: 8080
Listening.....
█
```

Client output:

```
kali@kali: ~/Desktop/Socket_Programming/LP_Project
File Actions Edit View Help
(kali@kali)-[~]
$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
(kali@kali)-[~]
$ cd Desktop/Socket_Programming/LP_Project
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ls
a.out client.c output.png server.c
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ gcc client.c
client.c: In function 'main':
client.c:31:17: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   31 |         gets(buffer);
      |         ^~~~~
      |         fgets
/usr/bin/ld: /tmp/ccS7McUP.o: in function 'main':
client.c:(.text+0xfa): warning: the 'gets' function is dangerous and should not be used.
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ █
```

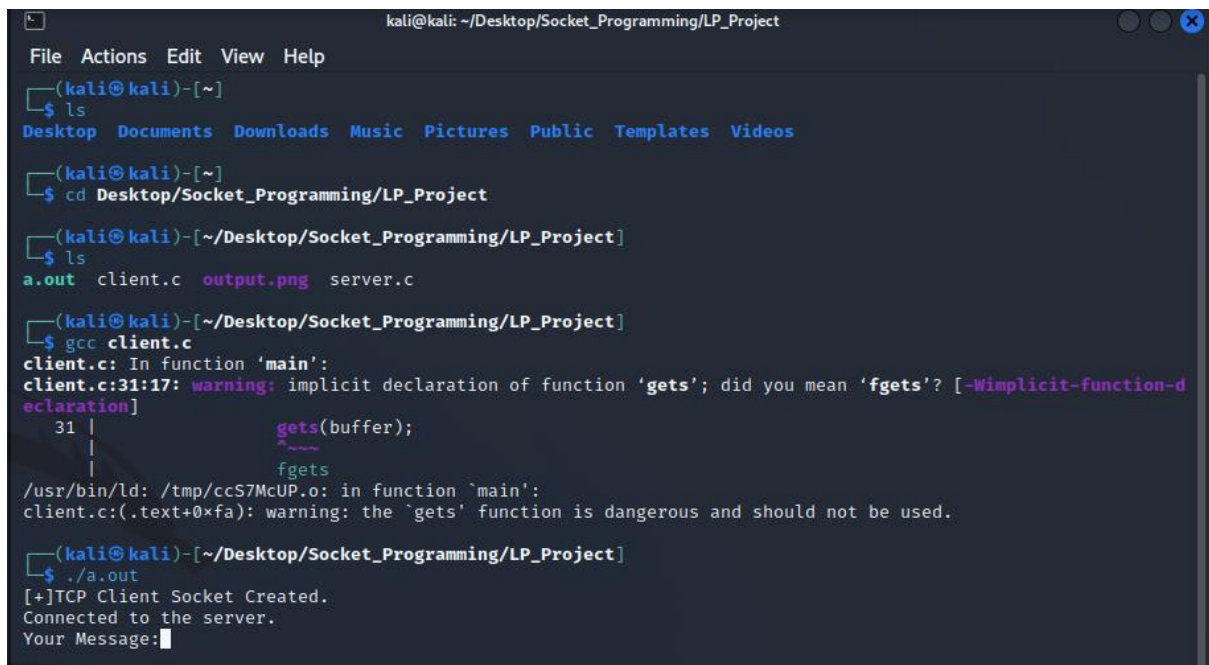
Step 2: Now run the client.c code. You will observe that server as connected to the client as shown below:

Server Output:



```
kali@kali: ~/Desktop/Socket_Programming/LP_Project
File Actions Edit View Help
(kali@kali)-[~]
$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
(kali@kali)-[~]
$ cd Desktop/Socket_Programming/LP_Project
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ls
a.out client.c output.png server.c
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ gcc server.c
server.c: In function 'main':
server.c:44:25: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   44 |         gets(buffer);
      |         ^~~~~
      |         fgets
/usr/bin/ld: /tmp/ccCsWL9F.o: in function `main':
server.c:(.text+0x1cf): warning: the `gets' function is dangerous and should not be used.
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ./a.out
[+]TCP Server Socket Created.
[+]Bind to the port number: 8080
Listening.....
[+]Client Connected.
[]
```

Client Output:



```
kali@kali: ~/Desktop/Socket_Programming/LP_Project
File Actions Edit View Help
(kali@kali)-[~]
$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
(kali@kali)-[~]
$ cd Desktop/Socket_Programming/LP_Project
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ls
a.out client.c output.png server.c
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ gcc client.c
client.c: In function 'main':
client.c:31:17: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   31 |         gets(buffer);
      |         ^~~~~
      |         fgets
/usr/bin/ld: /tmp/ccS7McUP.o: in function `main':
client.c:(.text+0xfa): warning: the `gets' function is dangerous and should not be used.
(kali@kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ./a.out
[+]TCP Client Socket Created.
Connected to the server.
Your Message:[]
```

Step 3: Now You can start the chat as shown below:

Server Output:

```
(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ls
a.out  client.c  output.png  server.c

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ gcc server.c
server.c: In function 'main':
server.c:44:25: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   44 |         |           gets(buffer);
       |         |           ^~~~
       |         |           fgets
/usr/bin/ld: /tmp/ccCsWL9F.o: in function `main':
server.c:(.text+0x1cf): warning: the `gets' function is dangerous and should not be used.

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ./a.out
[+]TCP Server Socket Created.
[+]Bind to the port number: 8080
Listening.....
[+]Client Connected.
Client: hi
Your Message:hi
Client: how r u
Your Message:fine. What about u
Client: i am also great. by the what do you want
Your Message:nothing
Client: then good bye
Your Message:bye bye
█
```

Client Output:

```
kali@kali: ~/Desktop/Socket_Programming/LP_Project
File Actions Edit View Help

(kali㉿kali)-[~]
$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

(kali㉿kali)-[~]
$ cd Desktop/Socket_Programming/LP_Project

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ls
a.out  client.c  output.png  server.c

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ gcc client.c
client.c: In function 'main':
client.c:31:17: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   31 |         |           gets(buffer);
       |         |           ^~~~
       |         |           fgets
/usr/bin/ld: /tmp/ccS7McUP.o: in function `main':
client.c:(.text+0xfa): warning: the `gets' function is dangerous and should not be used.

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ./a.out
[+]TCP Client Socket Created.
Connected to the server.
Your Message:hi
Server: hi
Your Message:how r u
Server: fine. What about u
Your Message:i am also great. by the what do you want
Server: nothing
Your Message:then good bye
Server: bye bye
Your Message:█
```


Step 4: Now to stop the chat we have to do normal termination of the program i.e., ctrl+c as shown below:

Server Output:

```
(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ gcc server.c
server.c: In function 'main':
server.c:44:25: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   44 |         gets(buffer);
      |         ^~~~
      |         fgets
/usr/bin/ld: /tmp/ccCsWL9F.o: in function 'main':
server.c:(.text+0x1cf): warning: the 'gets' function is dangerous and should not be used.

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ./a.out
[+]TCP Server Socket Created.
[+]Bind to the port number: 8080
Listening.....
[+]Client Connected.
Client: hi
Your Message:hi
Client: how r u
Your Message:fine. What about u
Client: i am also great. by the what do you want
Your Message:nothing
Client: then good bye
Your Message:bye bye
^C

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$
```

Client Output:

```
kali@kali: ~/Desktop/Socket_Programming/LP_Project
File Actions Edit View Help

(kali㉿kali)-[~]
$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos

(kali㉿kali)-[~]
$ cd Desktop/Socket_Programming/LP_Project

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ls
a.out client.c output.png server.c

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ gcc client.c
client.c: In function 'main':
client.c:31:17: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   31 |         gets(buffer);
      |         ^~~~
      |         fgets
/usr/bin/ld: /tmp/ccS7McUP.o: in function 'main':
client.c:(.text+0xfa): warning: the 'gets' function is dangerous and should not be used.

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$ ./a.out
[+]TCP Client Socket Created.
Connected to the server.
Your Message:hi
Server: hi
Your Message:how r u
Server: fine. What about u
Your Message:i am also great. by the what do you want
Server: nothing
Your Message:then good bye
Server: bye bye
Your Message:^C

(kali㉿kali)-[~/Desktop/Socket_Programming/LP_Project]
$
```

CONCLUSION

Here for this experiment, I used Ip address as 127.0.0.1. Since this is conducted on same host, we used that Ip address. And for port number I used 8080, since port number below 1000 are reserved for some particular websites. So, I used port number greater than 1000 i.e., 8080.

Now you can use this chat program and perform the communication with each other. Here I followed TCP Protocol. And for termination, we have to go for normal termination of the program i.e., ctrl+c. This is done by the basics of TCP protocol.

**THANK
YOU**