

## **ASSIGNMENT-5**



**Name: Sayantan Roy**

**Roll Number: GCECTB-R18-3025**

**Subject: Computer Network Lab**

**Dept: CSE**

**Year:3rd**

# **ASSIGNMENT -5**

## **TCP ECHO CLIENT SERVER PROGRAM**

**IN C**

**TCP ECHO CLIENT SERVER PROGRAM  
IN C**

**SYSTEM USED :**

**OPERATING SYSTEM : kali linux 2020.1**

**KERNEL: 4.19.153**

**GUI: DEFAULT XSESSION (XFCE4)**

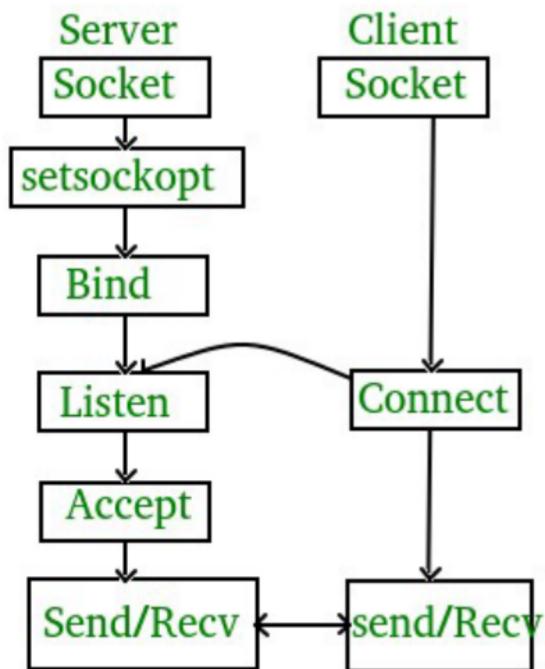
**TEXT EDITOR :NANO**

**system owned and project done by :Sayantan Roy**

## TCP ECHO CLIENT SERVER PROGRAM IN C

If we are creating a connection between client and server using TCP then it has few functionality like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPS, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

The entire process can be broken down into following steps:



**The entire process can be broken down into following steps:**

**TCP Server –**

- 1.using **create()**, Create TCP socket.
- 2.using **bind()**, Bind the socket to server address.
- 3.using **listen()**, put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
- 4.using **accept()**, At this point, connection is established between client and server, and they are ready to transfer data.
- 5.Go back to Step 3.

**TCP Client –**

- 1.Create TCP socket.
- 2.connect newly created client socket to server.

## **Arguments :**

**domain** –Specifies the communication domain ( AF\_INET for IPv4/ AF\_INET6 for IPv6 )

**type** –Type of socket to be created ( SOCK\_STREAM for TCP / SOCK\_DGRAM for UDP )

**protocol** –Protocol to be used by socket.  
0 means use default protocol for the address family.

**bind** : assigns address to the unbound socket

**sockfd** –File descriptor of socket.

**addr** –Structure in which address to be binded to is specified

## **Functions:**

`socket(int domain, int type, int protocol)`

Creates an unbound socket in the specified domain.  
Returns socket file descriptor.

**AIM:**

To write a program for TCP echo client server.

**ALGORITHM:****SERVER:**

- STEP 1:** Start
- STEP 2:** Declare the variables for the socket
- STEP 3:** Specify the family, protocol, IP address and port number
- STEP 4:** Create a socket using socket() function
- STEP 5:** Bind the IP address and Port number
- STEP 6:** Listen and accept the client's request for the connection
- STEP 7:** Read the client's message
- STEP 8:** Display the client's message
- STEP 9:** Close the socket
- STEP 10:** Stop

**CLIENT:**

- STEP 1:** Start
- STEP 2:** Declare the variables for the socket
- STEP 3:** Specify the family, protocol, IP address and port number
- STEP 4:** Create a socket using socket() function
- STEP 5:** Call the connect() function
- STEP 6:** Read the input message
- STEP 7:** Send the input message to the server
- STEP 8:** Display the server's echo
- STEP 9:** Close the socket
- STEP 10:** Stop

**SOURCE CODE:****FILENAME: tcpechoserver.c****SERVER:**

```
#include<stdio.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5035
int main(int argc,char**argv)
{
    int sockfd,newsockfd,clength;
    struct sockaddr_in serv_addr,cli_addr;
    char buffer[4096];
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=INADDR_ANY;
    serv_addr.sin_port=htons(SERV_TCP_PORT);
    printf("\nStart");
    bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    printf("\nListening... ");
    printf("\n");
    listen(sockfd,5);
    clength(sizeof(cli_addr));
    newsockfd=accept(sockfd,(struct sockaddr*)&cli_addr,&clength);
    printf("\nAccepted");
    printf("\n");
    read(newsockfd,buffer,4096);
    printf("\nClient message:%s",buffer);
    write(newsockfd,buffer,4096);
    printf("\n");
    close(sockfd);
    return 0;
}
```

**FILENAME:tcpechoclient.c****CLIENT:**

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5035
int main(int argc,char*argv[])
{
    int sockfd;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[4096];
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
    serv_addr.sin_port=htons(SERV_TCP_PORT);
    printf("\nReady for sending...");
    connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    printf("\nEnter the message to send\n");
    printf("\nClient: ");
    fgets(buffer,4096,stdin);
    write(sockfd,buffer,4096);
    printf("Serverecho:%s",buffer);
    printf("\n");
    close(sockfd);
    return 0;
}
```

**output :**

sayantan@localhost:~\$ gcc tcpechoserver.c -o tcpechoserver

sayantan@localhost:~\$ gcc tcpechoclient.c -o tcpechoclient

sayantan@localhost:~\$ ./tcpechoserver

Start  
Listening...

Accepted

Client message:hi ! how are you ?

sayantan@localhost:~\$ ./tcpechoclient

Ready for sending...  
Enter the message to send

Client: hi ! how are you ?

Serverecho:hi ! how are you ?

sayantan@localhost:~\$

The screenshot shows a terminal window titled "Shell No.1" with the command "01:52 PM". The window contains the code for a TCP echo server in C, written in the nano text editor. The code includes headers for stdio.h, netinet/in.h, and netdb.h, defines SERV\_TCP\_PORT as 5035, and implements a basic server loop with accept, read, and write operations. The terminal window has a dark theme with light-colored text. The bottom of the window shows the nano editor's command bar with various keyboard shortcuts.

```
GNU nano 5.3                                     tcpechoserver.c
#include<stdio.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5035
int main(int argc,char**argv)
{
    int sockfd,newsockfd,clength;
    struct sockaddr_in serv_addr,cli_addr;
    char buffer[4096];
    socklen_t clilen=sizeof(cli_addr);
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    if(sockfd<0)
    {
        perror("Server error");
        exit(1);
    }
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=INADDR_ANY;
    serv_addr.sin_port=htons(SERV_TCP_PORT);
    printf("\nStart");
    bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    printf("\nlistening ...");
    printf("\n");
    listen(sockfd,5);
    clength=sizeof(cli_addr);
    newsockfd=accept(sockfd,(struct sockaddr*)&cli_addr,&clength);
    printf("\nAccepted");
    printf("\n");
    read(newsockfd,buffer,4096);
    printf("\nClient message:%s",buffer);
    write(newsockfd,buffer,4096);
    printf("\n");
    close(sockfd);
    return 0;
}
```

filename:tcpechoserver.c

The screenshot shows a terminal window with a dark theme. At the top, there are several tabs: "TCP DATETIM...", "TCP ECHO CLI...", "New Tab - Chr...", "Shell No.1", and "No Data". The status bar at the bottom shows the time as "01:53 PM EN" and various system icons. The main area of the terminal is a text editor displaying the code for "tcpechoclient.c". The code is written in C and includes headers like stdio.h, sys/types.h, sys/socket.h, netinet/in.h, and netdb.h. It defines a port number SERV\_TCP\_PORT as 5035 and sets up a socket to connect to a server at 127.0.0.1:5035. The program then reads input from the user and writes it back to the server. The file ends with a closing brace. Below the code, there is a menu bar with "File", "Actions", "Edit", "View", and "Help". At the bottom of the terminal window, there is a toolbar with various keyboard shortcut keys for file operations like Help, Exit, Write Out, Read File, Where Is, Replace, Cut, Paste, Execute, Justify, Location, Go To Line, Undo, Redo, Set Mark, To Bracket, Copy, Where Was, and Next.

```
GNU nano 5.3                                     tcpechoclient.c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5035
int main(int argc,char*argv[])
{
    int sockfd;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[4096];
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
    serv_addr.sin_port=htons(SERV_TCP_PORT);
    printf("\nReady for sending ...");
    connect(sockfd,(STRUCT sockaddr*)&serv_addr,sizeof(serv_addr));
    printf("\nEnter the message to send\n");
    printf("\nClient ");
    fgets(buffer,4096,stdin);
    write(sockfd,buffer,4096);
    printf("Server echo:%s",buffer);
    printf("\n");
    close(sockfd);
    return 0;
}

^G Help      ^C Write Out     ^W Where Is      ^K Cut          [ Read 28 lines ]   M-U Undo
^X Exit      ^R Read File     ^L Replace       ^U Paste        ^T Execute      C Location
                                         ^J Justify      ^I Go To Line   M-E Redo
                                         M-A Set Mark  M-Q To Bracket M-Q Where Was M-W Next
```

filename:tcpechoclient.c

A screenshot of a terminal window titled "Shell No.1". The window has a dark background and a light-colored text area. At the top, there is a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu, the command "sayantan@localhost:~\$ ./tcpechoserver" is entered. The server starts listening for connections, indicated by the message "Listening ...". Once a connection is accepted, the server prints "Accepted" followed by the client's message "Client message: hi ! How are you ?". The terminal prompt "sayantan@localhost:~\$ " is visible at the bottom.

```
sayantan@localhost:~$ ./tcpechoserver
Listening ...
Accepted
Client message: hi ! How are you ?
sayantan@localhost:~$
```

**fig : TCP echo server is running**

A screenshot of a terminal window titled "Shell No.1". The window shows a session between a client and a server. The client sends the message "hi ! How are you ?" and the server echoes it back. The terminal has a dark background with white text.

```
sayantan@localhost:~$ ./tcpechoclient
./tcpechoclient: No such file or directory
Ready for sending ... ./tcpechoserver
Enter the message to send
Client: hi ! How are you ?
Serverecho:hi ! How are you ?
sayantan@localhost:~$
```

**fig : TCP echo client is running , client's message got echo from server**