# SSN College of Engineering, Kalavakkam
# Department of Computer Science and Engineering
# V Semester - CSE 'B'
# UCS1511 NETWORKS LAB

**Date:** 03/08/2020

**Name:** Rahul Ram M

**Exercise:** 02

**Reg No:** 185001121

## a: SIMPLE CLIENT SERVER USING TCP

### Learning Objective:

To develop a socket program to establish a client server communication in which the client sends data to server and the server replies to the client.

### Algorithm for Server:

1. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.
2. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.
3. Using bind() to binf the socket to the address and port number specified in addr(custom data structure). Here, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.
4. listen() function is used to set the server socket in the passive mode, where it waits for the client to approach the server to make a connection, with maximum number of connection in this case is 2.
5. accept() creates a new connected socket and returns a new file descriptor reffering to the socket. After this the connection between server and client is established.
6. read(new_socket, buff, sizeof(buff)) - reads the message sent by the client in the buffer specified in the parameter along with its size preceded by the new socket descriptor.
7. Printing the message from the client using printf().
8. Using scanf(%[^\n]s", msg), reading the input message from the user with space at server side.
9. write(new_socket, msg, sizeof(msg)) - is used to write the message input by the server in the variable msg to br read by the client.
10. close() function shuts down the socket associated with socket descriptor's, and frees resources allocated to the socket.

### Algorithm for Client:

1. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.
2. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.

3. The above two steps are same as the server.
4. The connect() system call connects the socket referred to by the file descriptor socket_fd to the address specified by server_addr. Server's address and port is specified in server_addr.
5. Using scanf(%[^\n]s", msg), reading the input message from the user with space at server side.
6. write(new_socket, msg, sizeof(msg)) - is used to write the message input by the server in the variable msg to br read by the client.
7. read(new_socket, buff, sizeof(buff)) - reads the message sent by the client in the buffer specified in the parameter along with its size preceded by the new socket descriptor.
8. Printing the message from the server using printf().
9. close() function shuts down the socket associated with socket descriptor's, and frees resources allocated to the socket.

## Program for Server:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include <unistd.h>

int main()
{

	int len, server_fd, new_socket, n;
	struct sockaddr_in server_addr, client_addr;
	char buff[1024], msg[1000];

	if((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		perror("Socker error");
	}

	bzero(&server_addr,sizeof(server_addr));

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

	if(bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
	{
		perror("Bind error: ");
	}

	if(listen(server_fd,2) < 0)
	{
		perror("Listen error");
	}

	len = sizeof(client_addr);

	if((new_socket = accept(server_fd, (struct sockaddr*)&client_addr, &len)) < 0)
```

```
	{
		perror("Accept error");
	}

	//Receiving the message
	n = read(new_socket, buff, sizeof(buff));
	printf("Message from Client:%s\n",buff);

	printf("\nEnter message : ");
	scanf("%[^\n]s", msg);
	n = write(new_socket, msg, sizeof(msg));

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

## Program for Client:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include <unistd.h>

int main()
{
	int socket_fd, n, len;
	struct sockaddr_in server_addr, client_addr;

	char msg[1000], buff[1024];

	if((socket_fd=socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		perror("Socket error");
	}

	bzero(&server_addr,sizeof(server_addr));

	server_addr.sin_family = AF_INET;
	server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
	server_addr.sin_port = htons(8080);

	if(connect(socket_fd,(struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
	{
		perror("Connect error");
	}

	//Sending Message
	printf("Enter message : ");
	scanf("%[^\n]s", msg);
	n = write(socket_fd, msg, sizeof(msg));
```

```
        n = read(socket_fd, buff, sizeof(buff));
        printf("Message from Server : %s\n", buff);

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

## Screenshot for Server:

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include <unistd.h>

int main()
{
        int len, server_fd, new_socket, n;
        struct sockaddr_in server_addr, client_addr;
        char buff[1024], msg[1000];

        if((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("Socker error");
        }

        bzero(&server_addr,sizeof(server_addr));

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

        if(bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
        {
                perror("Bind error: ");
        }

        if(listen(server_fd,2) < 0)
        {
                perror("Listen error");
        }

        len = sizeof(client_addr);

        if((new_socket = accept(server_fd, (struct sockaddr*)&client_addr, &len)) < 0)
        {
                perror("Accept error");
        }

        //Receiving the message
        n = read(new_socket, buff, sizeof(buff));
        printf("Message from Client : %s\n",buff);
        printf("\nEnter message : ");
        scanf("%[^\n]s", msg);
        n = write(new_socket, msg, sizeof(msg));

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

## Screenshot for Client:

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include <unistd.h>

int main()
{
        int socket_fd, n, len;
        struct sockaddr_in server_addr, client_addr;

        char msg[1000], buff[1024];

        if((socket_fd=socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("Socket error");
        }

        bzero(&server_addr,sizeof(server_addr));

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(8080);

        if(connect(socket_fd,(struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
        {
                perror("Connect error");
        }

        //Sending Message
        printf("Enter message : ");
        scanf("%[^\n]s", msg);
        n = write(socket_fd, msg, sizeof(msg));

        n = read(socket_fd, buff, sizeof(buff));
        printf("Message from Server : %s\n", buff);

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

## Server Output:

```
rahul@rahul-Ubuntu:~$ ./s1
Message from Client : Hello! How are you?

Enter message : I'm fine.
rahul@rahul-Ubuntu:~$
```

## Client Output:

```
rahul@rahul-Ubuntu:~$ ./c1
Enter message : Hello! How are you?
Message from Server : I'm fine.
rahul@rahul-Ubuntu:~$
```

## Learning Outcomes:

This assignment helped me to

1. Write program for server and client with socket programming.
2. Understand various functions invloved in creating, estabilishing, maintaining, Sending, recieving and termininating the connection between the server and client.
3. Write code to make server and client communicate with each other using read() and write() functions.

# b: ECHO SERVER USING TCP

## Learning Objective:

To develop a socket program to establish a client server communication. The client sends data to server. The server in turn sends the same message back to the client. Transmit multiple lines of text. In client side display the echoed message. In server side display the message which is echoed to client.

## Algorithm for Server:

11. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.
12. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.
13. Using bind() to binf the socket to the address and port number specified in addr(custom data structure). Here, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.
14. listen() function is used to set the server socket in the passive mode, where it waits for the client to approach the server to make a connection, with maximum number of connection in this case is 2.
15. accept() creates a new connected socket and returns a new file descriptor reffering to the socket. After this the connection between server and client is established.
16. read(new_socket, buff, sizeof(buff)) - reads the message sent by the client in the buffer specified in the parameter along with its size preceded by the new socket descriptor.
17. Printing the message from the client using printf().
18. Copying the message from the client to msg.
19. write(new_socket, msg, sizeof(msg)) – writing the message input by the server in the variable msg to br read by the client.
20. Printing the echoed message.
21. close() function - shutting down the socket associated with socket descriptor’s, and frees resources allocated to the socket.

## Algorithm for Client:

1. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.
2. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.
3. The above two steps are same as the server.
4. The connect() system call connects the socket referred to by the file descriptor socket_fd to the address specified by server_addr. Server’s address and port is specified in server_addr.
5. Using scanf(%[^\t]s”, msg), reading the input message from the user with tab as end of the input at server side.
6. write(new_socket, msg, sizeof(msg)) - writing the message input by the server in the variable msg to br read by the client.
7. read(new_socket, buff, sizeof(buff)) – reading the echoed message sent by the client in the buffer specified in the parameter along with its size preceded by the new socket descriptor.

8. Printing the echoed message from the server using printf().
9. close() function - shutting down the socket associated with socket descriptor's, and frees resources allocated to the socket.

## Program for Server:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include <unistd.h>

int main()
{
	int server_fd, new_socket, n, len;
	struct sockaddr_in server_addr,client_addr;
	char buff[1024], msg[1000];

	if((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		perror("Socket error");
	}

	bzero(&server_addr,sizeof(server_addr));

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

	if(bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
	{
		perror("Bind error");
	}

	if(listen(server_fd,2) < 0)
	{
		perror("Listen error");
	}

	len = sizeof(client_addr);
	if((new_socket = accept(server_fd, (struct sockaddr*)&client_addr, &len)) < 0)
	{
		perror("Accept error");
	}

	//Receiving the message
	n = read(new_socket, buff, sizeof(buff));
	printf("Message from Client : %s\n",buff);

	strcpy(msg, buff);
	n = write(new_socket, msg, sizeof(msg));
	printf("Message echoed to client : %s\n", msg);
```

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

## Program for Client:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include <unistd.h>

int main()
{

	int len;
	int socket_fd,n;
	struct sockaddr_in server_addr,client_addr;

	char msg[1000]; char buff[1024];

	if((socket_fd=socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		perror("Socket error");
	}

	bzero(&server_addr,sizeof(server_addr));

	server_addr.sin_family = AF_INET;
	server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
	server_addr.sin_port = htons(8080);

	if(connect(socket_fd,(struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
	{
		perror("Connect error");
	}

	//Sending Message
	printf("Enter message : ");
	scanf("%[^\t]s",msg);
	n = write(socket_fd, msg, sizeof(msg));

	n = read(socket_fd, buff, sizeof(buff));
	printf("Echoed Message : %s\n",buff);

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

## Screenshot for Server:

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include <unistd.h>

int main()
{
        int server_fd, new_socket, n, len;
        struct sockaddr_in server_addr,client_addr;
        char buff[1024], msg[1000];

        if((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("Socket error");
        }

        bzero(&server_addr,sizeof(server_addr));

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

        if(bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
        {
                perror("Bind error");
        }

        if(listen(server_fd,2) < 0)
        {
                perror("Listen error");
        }

        len = sizeof(client_addr);
        if((new_socket = accept(server_fd, (struct sockaddr*)&client_addr, &len)) < 0)
        {
                perror("Accept error");
        }

        //Receiving the message
        n = read(new_socket, buff, sizeof(buff));
        printf("Message from Client : %s\n",buff);

        strcpy(msg, buff);
        n = write(new_socket, msg, sizeof(msg));
        printf("Message echoed to client : %s\n", msg);
        close(server_fd);
        close(new_socket);
        return 0;
}
```

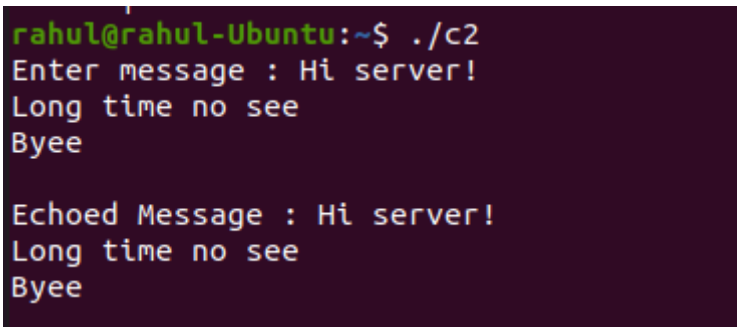
## Screenshot for Client:

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include <unistd.h>

int main()
{

        int len;
        int socket_fd,n;
        struct sockaddr_in server_addr,client_addr;

        char msg[1000]; char buff[1024];

        if((socket_fd=socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("Socket error");
        }

        bzero(&server_addr,sizeof(server_addr));

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(8080);

        if(connect(socket_fd,(struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
        {
                perror("Connect error");
        }

        //Sending Message
        printf("Enter message : ");
        scanf("%[^\t]s",msg);
        n = write(socket_fd, msg, sizeof(msg));

        n = read(socket_fd, buff, sizeof(buff));
        printf("Echoed Message : %s\n",buff);

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

## Server I/O:

```
rahul@rahul-Ubuntu:~$ ./s2
Message from Client : Hi server!
Long time no see
Byee

Message echoed to client : Hi server!
Long time no see
Byee
```

## Client I/O:




## Learning Outcomes:

This assignment helped me to

1. Write program for server and client with socket programming.
2. Understand various functions invloved in creating, estabilishing, maintaining, Sending, recieving and termininating the connection between the server and client.
3. Write a code where message from client is sent to server, which echoes the same message to the client.