# SSN College of Engineering, Kalavakkam

## Department of Computer Science and Engineering

V Semester - CSE 'B'

UCS1511 NETWORKS LAB

Name: Likhitha Verma A
REG No: 185001084
Date: 08/09/2020

**Exercise 3 : MULTIPLE CLIENTS TO SERVER CHAT USING TCP**

**Aim :**

To develop a socket program to establish a multi-client server communication. Multiple clients can send data to the server. The server replies to each client.

**Learning objectives** :

- To learn C socket programming.
- To learn how to create a TCP multiple clients to server connection.
- To learn how application programs use protocol software to communicate across networks and internets.
- To learn about various system calls used in socket programming.

**Algorithm :**

SERVER SIDE:

1. Create readfds as **fd_set** which is a set of sockets to "monitor" for some activity.
2. Create a socket descriptor with **socket()** system call with AF_INET, SOCK STREAM, default protocol and store as sockfd.
3. If sockfd returns a negative value, print error message.
4. Create sockaddr_in object and initialize with values.

5. Bind newly created socket to address given in sockaddr _in using **bind()** system call.
6. If bind() returns negative value, bind failed, print error message.
7. Listen for connections using **listen()** system call.
8. Clear readfds using **FD_ZERO()** and add sockfd descriptor to the fd_set readfds.
9. Monitor the list of sockets using **select().** It will block until an activity occurs or time limit exceeds.
10. Check if the descriptor is already available in the fd_set using **FD_ISSET()** , and accept connections from socket using **accept()** system call and store client socket descriptor in a separate variable. Then, remove the descriptor from the fd_set using **FD_CLR()**.
11. Find the highest file descriptor number(size) and store it as a limit.
12. Use **select()** system call to monitor activity from new clients using "limit" as size.
13. For each activity monitored in the set of descriptors
    - Read message into buffer using **read()** system call and print the same.
    - Read the message to send to the client into the buffer from the server.
    - Write the message onto the client using **write()** system call.
14. Whenever a client terminates, close that client connection using **close()**.
15. Close connections on socket using **close()** and terminate program.

CLIENT SIDE:

1. Create a socket descriptor with **socket()** system call with AF_INET, SOCK STREAM, default protocol and store as sockfd.
2. If sockfd returns a negative value, print error message.
3. Create sockaddr_in object and initialize with values.

4. Connect the client to the server at the address given in the socket descriptor using **connect()** system call.
5. If connect() returns negative value, connection failed, print error message.
6. Read a message from the user and write it onto the server socket using **write()** system call.
7. Read the response from the server into a buffer variable using read() system call and display received message from the server.
8. Close connections on socket using **close()** and terminate program.

**Program:**

SERVER SIDE

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include <unistd.h> //for read, write and close
#include <stdlib.h>

#define MAX 4

int fd_limit(int arr[MAX], int sockfd, fd_set *readfds)
{
    int max = -1;
    for (int i = 0; i < MAX; ++i)
    {
        if(arr[i] > max)
            max=arr[i];
        if (arr[i] > 0)
            FD_SET(arr[i], readfds);
    }
    if(sockfd > max)
        max=sockfd;
```

```
    return max + 1;
}


int main()
{
    int sockfd,newfd[MAX]={0};
    struct sockaddr_in seraddr, cli;
    socklen_t n;
    char buff[100];

    struct timeval tv;
    tv.tv_sec = 1;
    tv.tv_usec = 0;

    fd_set readfds;

    //Socket Creation
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    if(sockfd<0)
        perror("Socket Creation failed...!");
    else
        printf("Socket Created Successfully...!\n");

    bzero(&seraddr,sizeof(seraddr));

    seraddr.sin_family=AF_INET;
    seraddr.sin_port=htons(3335);
    seraddr.sin_addr.s_addr=htonl(INADDR_ANY);

    //binding to the socket
    if(bind(sockfd,(struct sockaddr *) &seraddr, sizeof(seraddr))<0)
       perror("Bind failed...!");
    else
       printf("Bound Successfully...!\n");

    //listen for connections
    if(listen(sockfd,MAX)>=0)
        printf("Waiting for client ... \n");
```

```
    else
        perror("listen failed ... \n");

    n=sizeof(cli);
    bzero(newfd, sizeof(int ) * MAX );
    while(1)
    {

        FD_ZERO(&readfds);
        FD_SET(sockfd,&readfds);

        int activity=select(sockfd+1,&readfds,NULL,NULL,&tv);

        if(activity<0){
            perror("Error occurred in select()\n ");
            exit(0);
        }

      //for accepting new clients
        if (FD_ISSET(sockfd, &readfds))
        {
            int client = accept(sockfd, (struct sockaddr *)&cli, &n);
            if (client < 0)
                perror("Accept connection failed..!!\n ");

            for (int i = 0; i < MAX; i++)
                if (newfd[i] == 0)
                {
                    newfd[i] = client;
                    break;
                }
            FD_CLR(sockfd, &readfds);
        }

        int limit = fd_limit(newfd, sockfd, &readfds);

        //new message from existing clients
        activity = select(limit, &readfds, NULL, NULL, &tv);

        if (activity < 0){
```

```
            perror("Error occurred in select()\n ");
            exit(0);
        }

        for (int i = 0; i < MAX; i++)
        {
            if (newfd[i] < 0)
                continue;

            // Message from client
            if (FD_ISSET(newfd[i], &readfds))
            {
                int count = read(newfd[i], buff, 100);

                // Client has terminated
                if (strcmp(buff, "*") == 0)
                {
                    close(newfd[i]);
                    newfd[i] = 0;
                    printf("Client %d disconnected!\n", i+1);
                }
                else
                {
                    printf("Client %d: %s \n", (i+1), buff);
                    bzero(buff, 100);
                    printf("Server: ");
                    scanf("%[^\n]", buff);
                    getchar();

                    // Write response to client
                    write(newfd[i], buff, 100);
                }
            }
        }
    }
    close(sockfd);
    return 0;
    }
```

CLIENT SIDE:

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include <unistd.h> //for read write and close
#include <arpa/inet.h> //for inet_addr()

int main(){

	int sockfd,n;
	struct sockaddr_in seraddr;
	char buff[100],cname[20];

	//Socket Creation
	sockfd=socket(AF_INET,SOCK_STREAM,0);
	if(sockfd<0)
		perror("Socket Creation failed...!");
	else
		printf("Socket Created Successfully...!\n");

	bzero(&seraddr,sizeof(seraddr));

	seraddr.sin_family=AF_INET;
	seraddr.sin_port=htons(3335);
	seraddr.sin_addr.s_addr=inet_addr("127.0.0.1");

	//Connect to a socket
	connect(sockfd,(struct sockaddr *)&seraddr,sizeof(seraddr));

	//Send message
	printf("Enter Client Name : ");
	scanf(" %[^\n]",cname);

	while (1)
	{
```

```
            bzero(buff, 100);
            printf("%s : ",cname);
            scanf(" %[^\n]",buff);

            write(sockfd, buff, 100);

            if(strcmp(buff, "*") == 0) break;

            read(sockfd, buff, 100);
            printf("Server: %s\n", buff);
      }
      printf("Connection terminated..!!!\n");
      close(sockfd);

      return 0;
}
```

**OUTPUT :**

SERVER SIDE :

```
Last login: Sun Sep 13 12:12:12 on console
msml@MSMLs-MacBook-Pro ~ % cd Likkie/cn-lab/ex-3
msml@MSMLs-MacBook-Pro ex-3 % gcc server.c -o s
msml@MSMLs-MacBook-Pro ex-3 % ./s
Socket Created Successfully...!
Bound Successfully...!
Waiting for client ...
Client 1: Hi server
Server: hello client1
Client 2: good morning server
Server: GM client2
Client 2: How are you?
Server: I am good.
Client 1: It's time for me to leave. bye
Server: bye.
Client 1 disconnected!
Client 2: See you later
Server: okay!!
Client 2 disconnected!
```

CLIENT-1 SIDE:

```
Last login: Sun Sep 13 13:25:43 on ttys001
msml@MSMLs-MacBook-Pro ~ % cd Likkie/cn-lab/ex-3
msml@MSMLs-MacBook-Pro ex-3 % gcc client.c -o c1
msml@MSMLs-MacBook-Pro ex-3 % ./c1
Socket Created Successfully...!
Enter Client Name : Alice
Alice : Hi server
Server: hello client1
Alice : It's time for me to leave. bye
Server: bye.
Alice : *
Connection terminated..!!!
```

CLIENT-2 SIDE:

```
Last login: Sun Sep 13 13:25:16 on ttys002
msml@MSMLs-MacBook-Pro ~ % cd Likkie/cn-lab/ex-3
msml@MSMLs-MacBook-Pro ex-3 % gcc client.c -o c2
msml@MSMLs-MacBook-Pro ex-3 % ./c2
Socket Created Successfully...!
Enter Client Name : Bob
Bob : good morning server
Server: GM client2
Bob : How are you?
Server: I am good.
Bob : See you later
Server: okay!!
Bob : *
Connection terminated..!!!
```

**Learning Outcomes**:

- I have learnt to create a multi-client chat socket program in C.
- I have learnt about various system calls used in socket programming.
- I have learnt to create a TCP server/client program to exchange messages from each other.