Assignment - 1

• Write a simple socket program in c to echo the message from client to server and back to client.

1 TCP

Description of Steps to be taken- 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. Take the data from socket of client connection and then store it in the buffer.
6. Echo the message to the server console.
7. Close the connection.

TCP Client –

1. Create TCP socket.
2. connect newly created client socket to server.
3. Store the message from stdin to buffer.
4. Write the buffer message to the server socket.
5. Close the connection.

Code –

TCP SERVER CODE

```
#include <stdio.h>

#include <netdb.h>

#include <netinet/in.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>


#define MAX 80

#define PORT 8080

#define SA struct sockaddr
```

```
// Function designed for chat between client and server.
void func(int sockfd)
{
	char buff[MAX];
	// read the message from client and copy it in buffer
	read(sockfd, buff, sizeof(buff));
	//print the message from client buffer to the console
	printf("From client: %s ", buff);
}

// Driver function
int main()
{
	int sockfd, connfd, len;
	struct sockaddr_in servaddr, cli;

	// socket create and verification
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd == -1) {
		printf("socket creation failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully created..\n");
	bzero(&servaddr, sizeof(servaddr));

	// assign Protocol, IP, PORT
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
	servaddr.sin_port = htons(PORT);
```

```
// Binding newly created socket to given IP and verification
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
}
else
        printf("Socket successfully binded..\n");

// Now server is ready to listen and verification
if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
}
else
        printf("Server listening..\n");
len = sizeof(cli);

// Accept the data packet from client and verification
connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
        printf("server accept failed...\n");
        exit(0);
}
else
        printf("server accept the client...\n");

// Function for chatting between client and server
func(connfd);

// After chatting close the socket
```

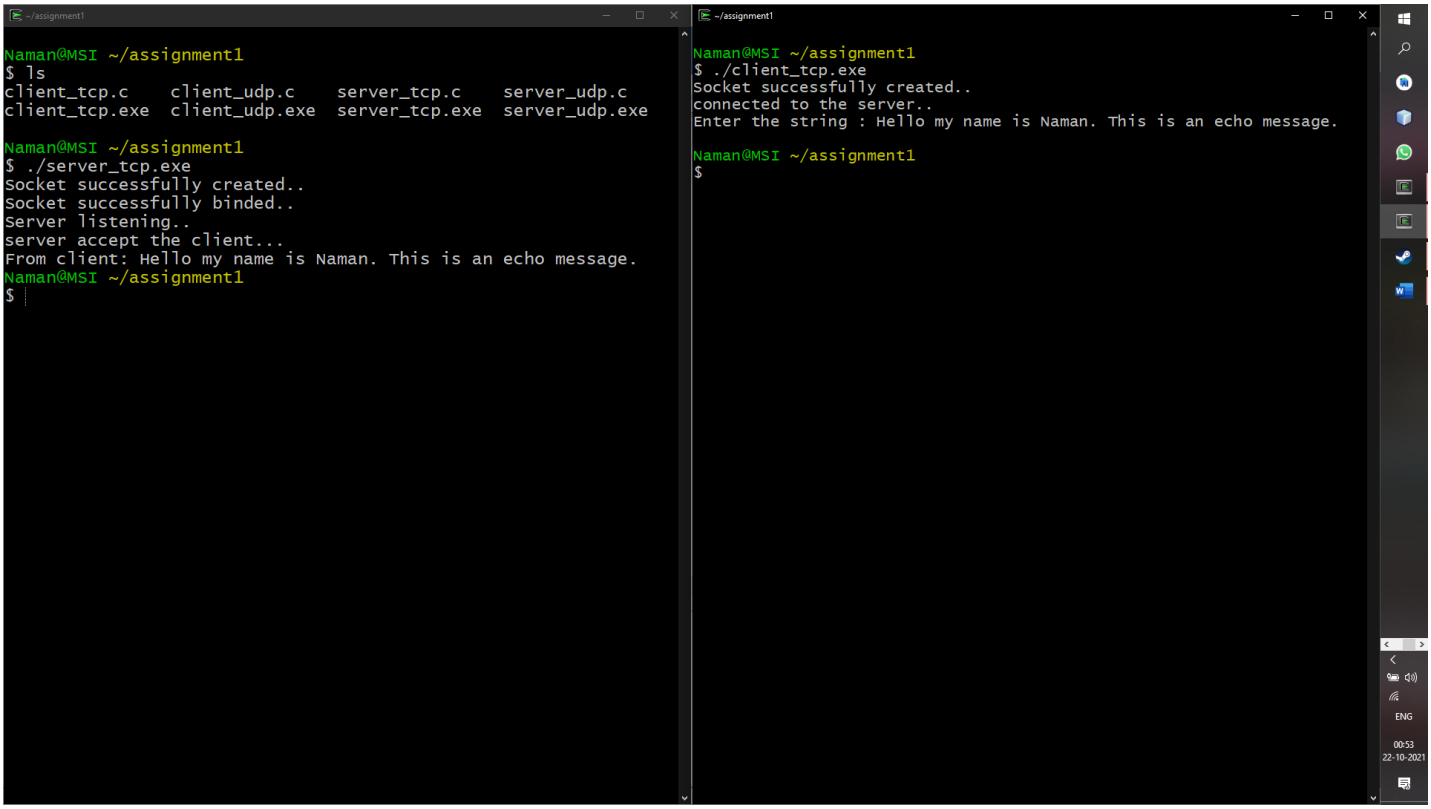
```
        close(sockfd);
}
```

TCP CLIENT CODE-

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

#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
   char buff[MAX];
   int n;
      bzero(buff, sizeof(buff));
      printf("Enter the string : ");
                    gets(buff);
      write(sockfd, buff, sizeof(buff));
}

int main()
{
   int sockfd, connfd;
   struct sockaddr_in servaddr, cli;

   // socket create and varification
```

```
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");

    // function for chat
    func(sockfd);

    // close the socket
    close(sockfd);
}
```

Output-



2 UDP

Description-

UDP Server :

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.

UDP Client :

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is received.
4. Process reply.
5. Close socket descriptor and exit.

UDP Server Code-

```
#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>
```

```
#include<unistd.h>
#include<netdb.h>
#include<stdio.h>
#include<string.h>
#include<arpa/inet.h>
#define MAXLINE 1024
//Driver code
int main(int argc,char **argv)
{
int sockfd;
int n;
socklen_t len;
//message buffer
char msg[1024];
//defining server and client address
struct sockaddr_in servaddr,cliaddr;
//creation of server socket
sockfd=socket(AF_INET,SOCK_DGRAM,0);
bzero(&servaddr,sizeof(servaddr));
//setting server protocol, address, port
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=INADDR_ANY;
servaddr.sin_port=htons(5035);
printf("\n\n Binded");
//binding the socket to the port
bind(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
//listening to the client
printf("\n\n Listening...");
for(;;)
{
    printf("\n ");
```

```
    len=sizeof(cliaddr);
//receiving the message from the client and storing it in msg variable
    n=recvfrom(sockfd,msg,MAXLINE,0,(struct sockaddr*)&cliaddr,&len);
//printing the client message to the console
    printf("\n Client's Message : %s\n",msg);
    if(n<6)
      perror("send error");
//send reply to client
    sendto(sockfd,msg,n,0,(struct sockaddr*)&cliaddr,len);
}
return 0;
}
```

UDP Client Code –

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<arpa/inet.h>
#include<string.h>
#include<arpa/inet.h>
#include<stdio.h>
#define MAXLINE 1024
int main(int argc,char* argv[])
{
int sockfd;
int n;
socklen_t len;
char sendline[1024], recvline[1024];
struct sockaddr_in servaddr;
strcpy(sendline,"");
```

```
printf("\n Enter the message : ");
gets(sendline);
//creating the socket
sockfd=socket(AF_INET,SOCK_DGRAM,0);
bzero(&servaddr,sizeof(servaddr));
//setting the server protocol, address, port
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
servaddr.sin_port=htons(5035);
//connecting to the server socket
connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
len=sizeof(servaddr);
//sending the data to server
sendto(sockfd,sendline,MAXLINE,0,(struct sockaddr*)&servaddr,len);
n=recvfrom(sockfd,recvline,MAXLINE,0,NULL,NULL);
recvline[n]=0;
printf("\n Server's Echo : %s\n\n",recvline);
return 0;
}
```

Output of Client Server Echo through UDP –