Computer Network

Assignment : 1

List of Topics : Socket Programming

Group No. : 22

Urja Gandhi	202112039
Sharvi Gabani	202112066
Sagar Variya	202112114

1. Create TCP server and client using socket programming. Make them communicate with each other by making a question and answer system between them. (At Least 4 different questions should be there)

TCPserver.c :

```
#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 8080


int main(int argc, char const *argv[])

{

    int server_fd, new_socket, valread;

    struct sockaddr_in address;

    int opt = 1;

    int addrlen = sizeof(address);

    char buffer[1024] = {0};

    char *exit_msg = "exit", *msg;


    // Creating socket file descriptor
```

```
   // The socket() - creates a socket in the specified domain and of the specified type.
   // AF_INET - communicating between processes on different hosts connected by IPV4
   // SOCK_STREAM - TCP (Transmission Control Protocol)
   // 0 - Protocol value for Internet Protocol(IP)
   if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
   {
      perror("socket failed");
      exit(EXIT_FAILURE);
   }

   // Forcefully attaching socket to the port 8080 - For address reuse
   // This is completely optional, but it helps in reuse of address an
   if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,&opt,
sizeof(opt)))
   {
      perror("setsockopt");
      exit(EXIT_FAILURE);
   }

   address.sin_family = AF_INET; // match the socket() call
   address.sin_addr.s_addr = INADDR_ANY; // bind to any local address
   // The htons() - used to convert an IP port number in host byte order to the IP port
number in network byte order
   address.sin_port = htons( PORT ); // specify port to listen on

   // Forcefully attaching socket to the port 8080
   // bind() - binds the socket to the address and port number specified in addr
   if (bind(server_fd, (struct sockaddr *)&address,sizeof(address))<0)
   {
      perror("bind failed");
```

```
        exit(EXIT_FAILURE);
    }

    // listen() - It puts the server socket in a passive mode, where it waits for the client to
approach the server to make a connection.
    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    // The server gets a socket for an incoming client connection by calling accept()
    if ((new_socket = accept(server_fd, (struct sockaddr
*)&address,(socklen_t*)&addrlen))<0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    while(1) {
        // memset() - used to fill buffer variable with 0.
        memset(buffer, 0, 1024);
        // The read() - reads data on a socket with descriptor fs and stores it in a buffer.
        valread = read( new_socket , buffer, 1024);
        buffer[valread]='\0';
        if(strlen(buffer)==0)
        {
            printf("Client exited...\n");//if buffer is empty then print message client exit
            }
            else{
```

```
          printf("Client : %s\n",buffer );//if any message send from server then it will print
          }

        memset(buffer, 0, 1024);
        printf("Server : ");
        // taking input of string
        fgets(msg,100,stdin);
        msg[strlen(msg)-1] = '\0';
        // while loop break when message will be "exit"
        if(!strcmp(msg, exit_msg)){
           // close() - shuts down the socket associated with the socket descriptor socket, and
frees resources allocated to the socket.
           close(server_fd);
           return 0;
        }
        // The send() - sends data on the socket with descriptor socket.
        send(new_socket , msg , strlen(msg) , 0 );
    }
    return 0;
}
```

TCPclient.c :

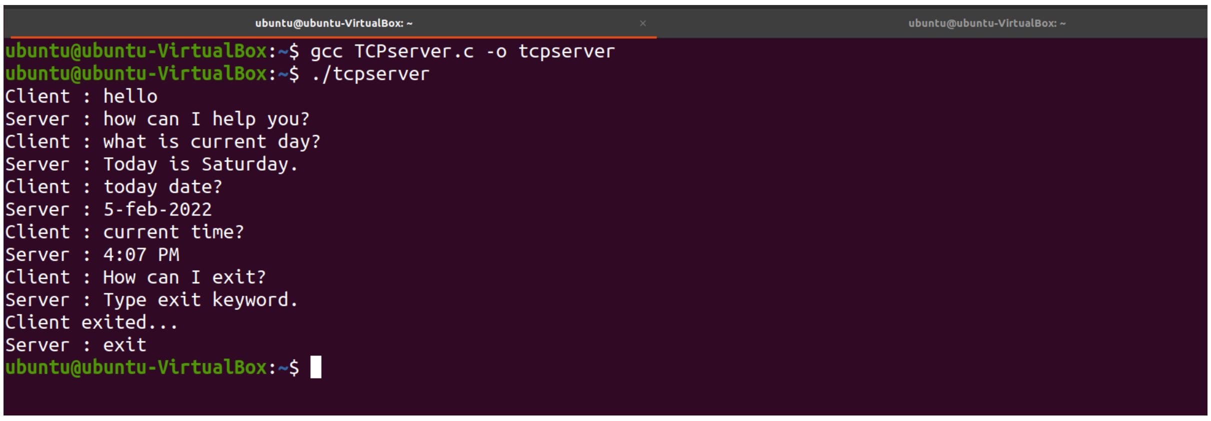
```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080

int main(int argc, char const *argv[])
```

```
{
   int sock = 0, valread;
   struct sockaddr_in serv_addr;
   char *exit_msg = "exit";
   char buffer[1024] = {0};
   char *msg;

   // The socket() - creates a socket in the specified domain and of the specified type.
   // AF_INET - communicating between processes on different hosts connected by IPV4
   // SOCK_STREAM - TCP (Transmission Control Protocol)
   // 0 - Protocol value for Internet Protocol(IP)
   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
      printf("\n Socket creation error \n");
      return -1;
   }

   serv_addr.sin_family = AF_INET;
   serv_addr.sin_port = htons(PORT);

   // Convert IPv4 and IPv6 addresses from text to binary form
   // inet_pton() - converts an Internet address in its standard text format into its numeric
binary form.
   if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
   {
      printf("\nInvalid address/ Address not supported \n");
      return -1;
   }
   // The connect() - connects the socket referred to by the file descriptor sockfd to the
address specified by addr.
   if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
   {
       printf("\nConnection Failed \n");
       return -1;
   }

   while(1) {
       //memset() - used to fill buffer variable with 0.
       memset(buffer, 0, 1024);
       // taking input of string
           fgets(msg,100,stdin);
           msg[strlen(msg)-1] = '\0';
       // while loop break when message will be "exit"
           if(!strcmp(msg, exit_msg)){
          // close() - shuts down the socket associated with the socket descriptor socket, and
frees resources allocated to the socket.
          close(sock);
          return 0;
       }
       // The send() - sends data on the socket with descriptor socket.
       send(sock , msg , strlen(msg) , 0 );
       // The read() - reads data on a socket with descriptor fs and stores it in a buffer.
       valread = read( sock , buffer, 1024);
       buffer[valread]='\0';
       if(strlen(buffer)==0)
       {
           printf("Server exited...\n");// if buffer is empty then print message server exit
      }
```

```
        else{

              printf("From Server : %s\n",buffer );// if any message send from server then it will
print

        }

    }

    return 0;

}
```

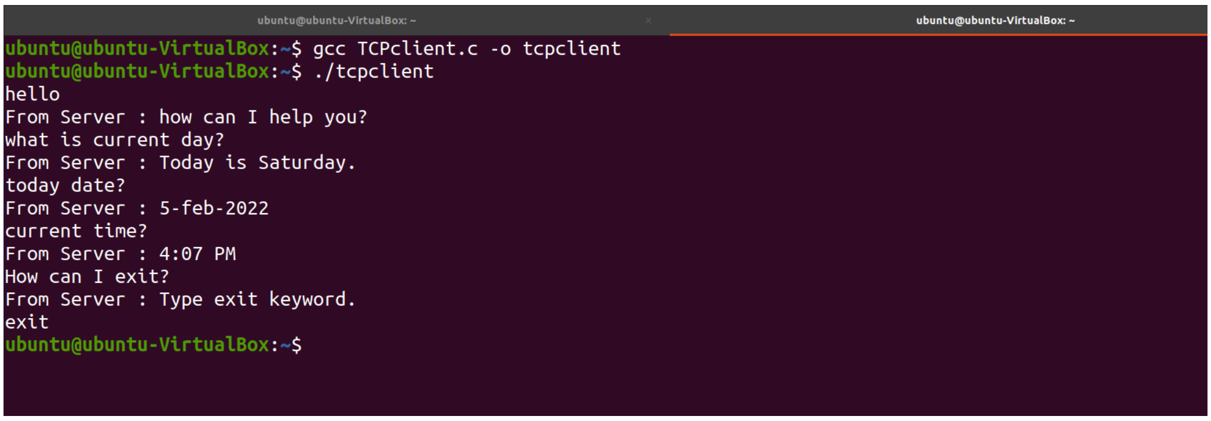
Output :

Server Side Terminal :



```
ubuntu@ubuntu-VirtualBox:~$ gcc TCPserver.c -o tcpserver
ubuntu@ubuntu-VirtualBox:~$ ./tcpserver
Client : hello
Server : how can I help you?
Client : what is current day?
Server : Today is Saturday.
Client : today date?
Server : 5-feb-2022
Client : current time?
Server : 4:07 PM
Client : How can I exit?
Server : Type exit keyword.
Client exited...
Server : exit
ubuntu@ubuntu-VirtualBox:~$
```

Client Side Terminal :



```
ubuntu@ubuntu-VirtualBox:~$ gcc TCPclient.c -o tcpclient
ubuntu@ubuntu-VirtualBox:~$ ./tcpclient
hello
From Server : how can I help you?
what is current day?
From Server : Today is Saturday.
today date?
From Server : 5-feb-2022
current time?
From Server : 4:07 PM
How can I exit?
From Server : Type exit keyword.
exit
ubuntu@ubuntu-VirtualBox:~$
```

Explanation :

[TCP Server]

Step 1: A Socket of specified type will be created in the specified domain using mentioned protocol.

And socket file descriptor will be returned.

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

Code:

```
/* here AF_INET means the the communication is over the internet
domain.SOCL_STREAM indicates its a stream type of communication and 0 indicates
the protocol used is TCP/IP.*/

if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
  {
     perror("socket failed");
     exit(EXIT_FAILURE);
  }
```

Step 2: Socket will be mapped to server address.

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

Code:

```
/*binds a socket to an address. Here, the address would be the IP address of the
current machine and the port number*/

if (bind(server_fd, (struct sockaddr *)&address,sizeof(address))<0)
  {
     perror("bind failed");
     exit(EXIT_FAILURE);
  }
```

Step 3: allows a process to listen on socket for communication

```
listen(socket_fd, no_of_waiting_connections)
```

Code:

```
/* 3 connections can wait at the max*/

if (listen(server_fd, 3) < 0)
  {
     perror("listen");
     exit(EXIT_FAILURE);
  }
```

Step 4: At this point, connection is established between client and server, and they are ready to transfer data.

int accept(sockfd,pointer_to_address_of client, addr_storing_size_of_client_address)

Code:

/* so the new_socket has the new socket address which will be used for communication. So, this command blocks until the read() of data is complete that is till the client has finished its write().*/

```
if ((new_socket = accept(server_fd, (struct sockaddr
*)&address,(socklen_t*)&addrlen))<0)

  {

     perror("accept");

     exit(EXIT_FAILURE);

  }
```

Step 5:

Code:

/*Communication between server and client takes place in this infinite loop which will break if server exits. Till then, server will receive message from client (read) and send message to client (send)*/

```
while(1) {

     memset(buffer, 0, 1024);

     valread = read( new_socket , buffer, 1024);

     buffer[valread]='\0';

     if(strlen(buffer)==0)

     {

        printf("Client exited...\n");

        }

        else{

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

        }


     memset(buffer, 0, 1024);

     printf("Server : ");
```

```
      fgets(msg,100,stdin);
      msg[strlen(msg)-1] = '\0';
         if(!strcmp(msg, exit_msg)){
        close(server_fd);
        return 0;
      }
      send(new_socket , msg , strlen(msg) , 0 );
    }
```

[TCP Client]

Step 1: A Socket of specified type will be created in the specified domain using mentioned protocol.

And socket file descriptor will be returned.

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

Code:

```
/* here AF_INET means the the communication is over the internet domain.SOCL_STREAM indicates its a stream type of communication and 0 indicates the protocol used is TCP/IP.*/
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    printf("\n Socket creation error \n");
    return -1;
  }
```

Step 2: Connection is established to server

connect(sockfd, host_to_which_itconnects, sizeof_addr)

Code:

```
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  {
    printf("\nConnection Failed \n");
    return -1;
  }
```

Step 3: Communication process continues till client exits using methods like send() and read()

Code:

```
while(1) {
        memset(buffer, 0, 1024);
             fgets(msg,100,stdin);
             msg[strlen(msg)-1] = '\0';
             if(!strcmp(msg, exit_msg)){
            close(sock);
            return 0;
        }
        send(sock , msg , strlen(msg) , 0 );
        valread = read( sock , buffer, 1024);
        buffer[valread]='\0';
        if(strlen(buffer)==0)
        {
             printf("Server exited...\n");// if buffer is empty then print message server exit
             }
             else{
                             printf("From Server : %s\n",buffer );// if any message send from
server then it will print
             }
    }
```

2. Create UDP server and client using socket programming. Make them communicate with each other by sending packets between them.

UDPsever.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT 8080
#define MAXLINE 1024

int main(int argc, char const *argv[])
{
  int server_fd;
  struct sockaddr_in servaddr, cliaddr;
  char buffer[MAXLINE] = {0};
  char *msg;

  // The socket() - creates a socket in the specified domain and of the specified type.
  // AF_INET - communicating between processes on different hosts connected by IPV4
  // SOCK_DGRAM - UDP (User Datagram Protocol)
  // 0 - Protocol value for Internet Protocol(IP)
  if ((server_fd = socket(AF_INET, SOCK_DGRAM, 0)) == 0)
  {
    perror("socket failed");
    exit(EXIT_FAILURE);
  }

  //memset() - used to fill servaddr and cliaddr variable with 0.
  memset(&servaddr, 0, sizeof(servaddr));
  memset(&cliaddr, 0, sizeof(cliaddr));
```

```
    servaddr.sin_family = AF_INET; // match the socket() call

    servaddr.sin_addr.s_addr = INADDR_ANY;// bind to any local address

    // The htons() - used to convert an IP port number in host byte order to the IP port
number in network byte order

    servaddr.sin_port = htons(PORT);// specify port to listen


    // Forcefully attaching socket to the port 8080

    // bind() - binds the socket to the address and port number specified in servaddr

    if (bind(server_fd, (struct sockaddr *)&servaddr,sizeof(servaddr))<0)

    {

        perror("bind failed");

        exit(EXIT_FAILURE);

    }


    int len, n;


    len = sizeof(cliaddr);


    while(1){

        // recvfrom() - places the received message into the buffer. This function is typically
used with connectionless sockets.

        // MSG_WAITALL - flag requests that the operation block until the full request is
satisfied.

            n = recvfrom(server_fd, (char *)buffer, MAXLINE, MSG_WAITALL, ( struct sockaddr *)
&cliaddr, &len);

            buffer[n] = '\0';

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


            memset(buffer, 0, 1024);

        printf("Server : ");

        // taking input of string
        fgets(msg,100,stdin);

        msg[strlen(msg)-1] = '\0';

        // The sendto() - sends data on the socket with descriptor socket.

        // MSG_CONFIRM - This flag is used to tell the kernel that the neighbour has
successfully replied to a message of ours.

        sendto(server_fd, (const char *)msg, strlen(msg), MSG_CONFIRM, (const struct
sockaddr *) &cliaddr,len);

    }

    return 0;

}
```

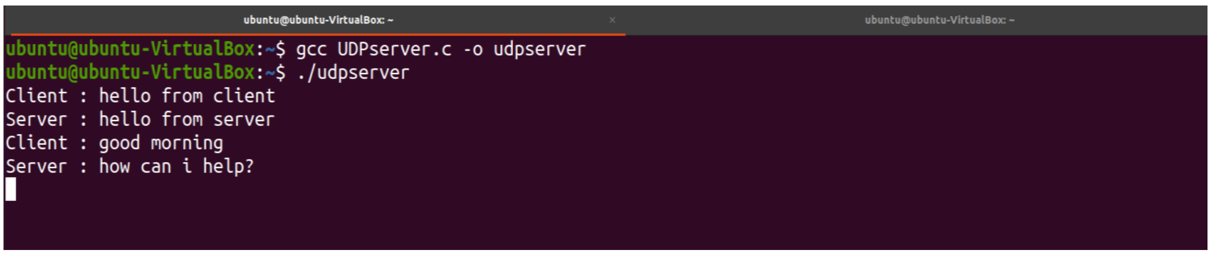
UDPclient.c :

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>


#define PORT 8080

#define MAXLINE 1024


int main(int argc, char const *argv[])

{

    int sock = 0;

    struct sockaddr_in servaddr;

    char *exit_msg = "exit";

    char *msg;
    char buffer[MAXLINE] = {0};


    // The socket() - creates a socket in the specified domain and of the specified type.

    // AF_INET - communicating between processes on different hosts connected by IPV4

    // SOCK_DGRAM - UDP (User Datagram Protocol)

    // 0 - Protocol value for Internet Protocol(IP)

    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)

    {

        printf("\n Socket creation error \n");

        return -1;

    }

    //memset() - used to fill servaddr variable with 0.

    memset(&servaddr, 0, sizeof(servaddr));


    servaddr.sin_family = AF_INET;// match the socket() call

    // The htons() - used to convert an IP port number in host byte order to the IP port
number in network byte order

    servaddr.sin_port = htons(PORT);// specify port to listen

    servaddr.sin_addr.s_addr = INADDR_ANY;// bind to any local address


    int n, len;

    while(1){

            memset(buffer, 0, 1024);

        // taking input of string

            fgets(msg,100,stdin);

            msg[strlen(msg)-1] = '\0';

        // while loop break when message will be "exit"

            if(!strcmp(msg, exit_msg)){

            // close() - shuts down the socket associated with the socket descriptor socket, and
frees resources allocated to the socket.
```

```
            close(sock);

            return 0;

        }

             // The sendto() - sends data on the socket with descriptor socket.

        // MSG_CONFIRM - This flag is used to tell the kernel that the neighbour has
successfully replied to a message of ours.

             sendto(sock, (const char *)msg, strlen(msg),MSG_CONFIRM, (const struct sockaddr
*) &servaddr,sizeof(servaddr));


        // recvfrom() - places the received message into the buffer. This function is typically
used with connectionless sockets.

        // MSG_WAITALL - flag requests that the operation block until the full request is
satisfied.

        n = recvfrom(sock, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr *)
&servaddr,&len);

             buffer[n] = '\0';

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

    }

    return 0;

}
```

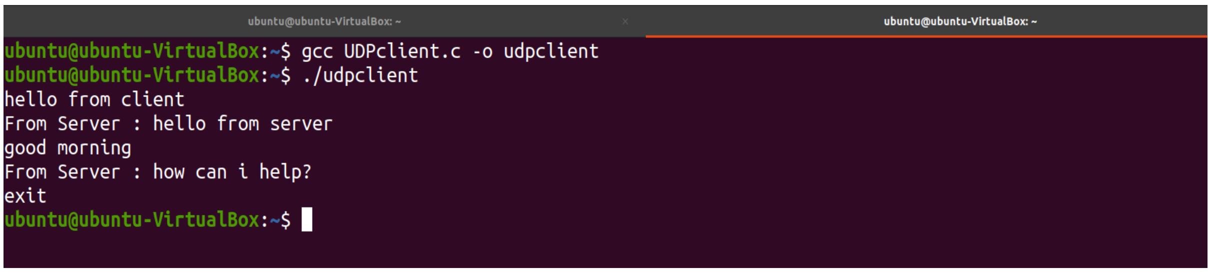
Output :

Server Side Terminal :



```
ubuntu@ubuntu-VirtualBox:~$ gcc UDPserver.c -o udpserver
ubuntu@ubuntu-VirtualBox:~$ ./udpserver
Client : hello from client
Server : hello from server
Client : good morning
Server : how can i help?
```

Client Side Terminal :



```
ubuntu@ubuntu-VirtualBox:~$ gcc UDPclient.c -o udpclient
ubuntu@ubuntu-VirtualBox:~$ ./udpclient
hello from client
From Server : hello from server
good morning
From Server : how can i help?
exit
ubuntu@ubuntu-VirtualBox:~$ 
```

Explanation :

->User Datagram Protocol is called a connectionless, unreliable protocol.

->It has a very limited error checking capability and it can be used with minimum overhead.

Communication process:

[UDP Server]

Step 1: A Socket of specified type will be created in the specified domain using mentioned protocol.

And socket file descriptor will be returned.

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

<u>Code:</u>

```
/*Socket of type SOCK_DGRAM (for UDP) in AF_NET domain (for IPv4) with default protocol is created. And if it fails to create the socket, error message will be printed.*/

if ((server_fd = socket(AF_INET, SOCK_DGRAM, 0)) == 0)
  {
     perror("socket failed");
     exit(EXIT_FAILURE);
  }
```

Step 2: Socket will be mapped to server address.

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

<u>Code:</u>

```
//File descriptor of the socket (server_fd) is bounded to server address(servaddr)
```

```
if (bind(server_fd, (struct sockaddr *)&servaddr,sizeof(servaddr))<0)
   {
       perror("bind failed");
       exit(EXIT_FAILURE);
   }
```

Step 3:Server will now wait for the datagram packet to arrive from the client.

And after receiving the datagram packet, it will send the reply to client

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr,
socklen_t *addrlen)
```

```
ssize_t sendto(int sockfd, void *buf, size_t len, int flags,const struct sockaddr
*dest_addr, socklen_t *addrlen)
```

Code:

```
/* As it is while(1), this will be in infinite loop. It means server is waiting for the client
to send the message. After receiving the message (recvfrom), we will print that
message. And now server would send the reply (sendto) */

while(1){
            n = recvfrom(server_fd, (char *)buffer, MAXLINE, MSG_WAITALL, ( struct
sockaddr *) &cliaddr, &len);
            buffer[n] = '\0';
            printf("Client : %s\n",buffer );


            memset(buffer, 0, 1024);
       printf("Server : ");
       fgets(msg,100,stdin);
       msg[strlen(msg)-1] = '\0';
       sendto(server_fd, (const char *)msg, strlen(msg), MSG_CONFIRM, (const struct
sockaddr *) &cliaddr,len);
   }
```

[UDP Client]

Step 1: A Socket of specified type will be created in the specified domain using mentioned protocol.

And socket file descriptor will be returned.

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

Code:

```
/*Socket of type SOCK_DGRAM (for UDP) in AF_NET domain (for IPv4) with default
protocol is created. And if it fails to create the socket, error message will be
printed.*/

if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
   {
       printf("\n Socket creation error \n");
       return -1;
   }
```

Step 2:Client will now send datagram packet to Server.

And after sending the datagram packet, it will wait for the reply.

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr,
socklen_t *addrlen)

ssize_t sendto(int sockfd, void *buf, size_t len, int flags,const struct sockaddr
*dest_addr, socklen_t *addrlen)
```

Code:

```
/* message will be sent (sendto) and server response would be received
(recvfrom)*/

while(1){
         memset(buffer, 0, 1024);
         fgets(msg,100,stdin);
         msg[strlen(msg)-1] = '\0';
         if(!strcmp(msg, exit_msg)){
        close(sock);
        return 0;
     }
                   sendto(sock, (const char *)msg, strlen(msg),MSG_CONFIRM, (const
struct sockaddr *) &servaddr,sizeof(servaddr));


     n = recvfrom(sock, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr *)
&servaddr,&len);
```

```
        buffer[n] = '\0';
        printf("From Server : %s\n",buffer );
}
```