Computer Networking Laboratory (IT692)

**Lab Sessional Report Submitted to**

**Maulana Abul Kalam Azad University of Technology, West Bengal**

**for**

****

**B. Tech.**

**in**

**Department of Information Technology**

**Submitted by**

**[MAYUKH CHAKRABARTI] (30000217019)**

**Course Faculty**

## SAIKAT BASU

**Department of Information Technology**

**Maulana Abul Kalam Azad University of Technology, West Bengal Simhat, Haringhata, Nadia**

**Pin-741249**

**May, 2020**

Table of Contents

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Exp. No** | **Name of the Experiments** | **Page no** | **Date of Experiment** | **Date of Submission** | **Faculty Signature** |
| **1** | Fabrication of Cables | **3-4** | **11-05-2020** | **17-05-2020** |  |
| **2** | Peer to Peer Technology | **5-6** | **11-05-2020** | **17-05-2020** |
| **3** | Star Topology | **7-8** | **11-05-2020** | **17-05-2020** |
| **4** | IPv4 Addressing | **9-10** | **11-05-2020** | **17-05-2020** |
| **5** | IPv4 Subnetting | **11-12** | **11-05-2020** | **17-05-2020** |
| **6** | TCP Client-Server | **13-18** | **11-05-2020** | **17-05-2020** |
| **7** | Concurrent Client-Server | **19-25** | **11-05-2020** | **17-05-2020** |
| **8** | Date Client-Server | **26-32** | **11-05-2020** | **17-05-2020** |
| **9** | Echo Client-Server | **33-40** | **12-05-2020** | **17-05-2020** |
| **10** | FTPClient-Server | **41-47** | **12-05-2020** | **17-05-2020** |
| **11** | Math Client-Server | **48-54** | **12-05-2020** | **17-05-2020** |
| **12** | UDP Client-Server | **55-59** | **12-05-2020** | **17-05-2020** |
| **13** | Sliding Window Implementation in C | **60-65** | **16-05-2020** | **17-05-2020** |

## Program No:1

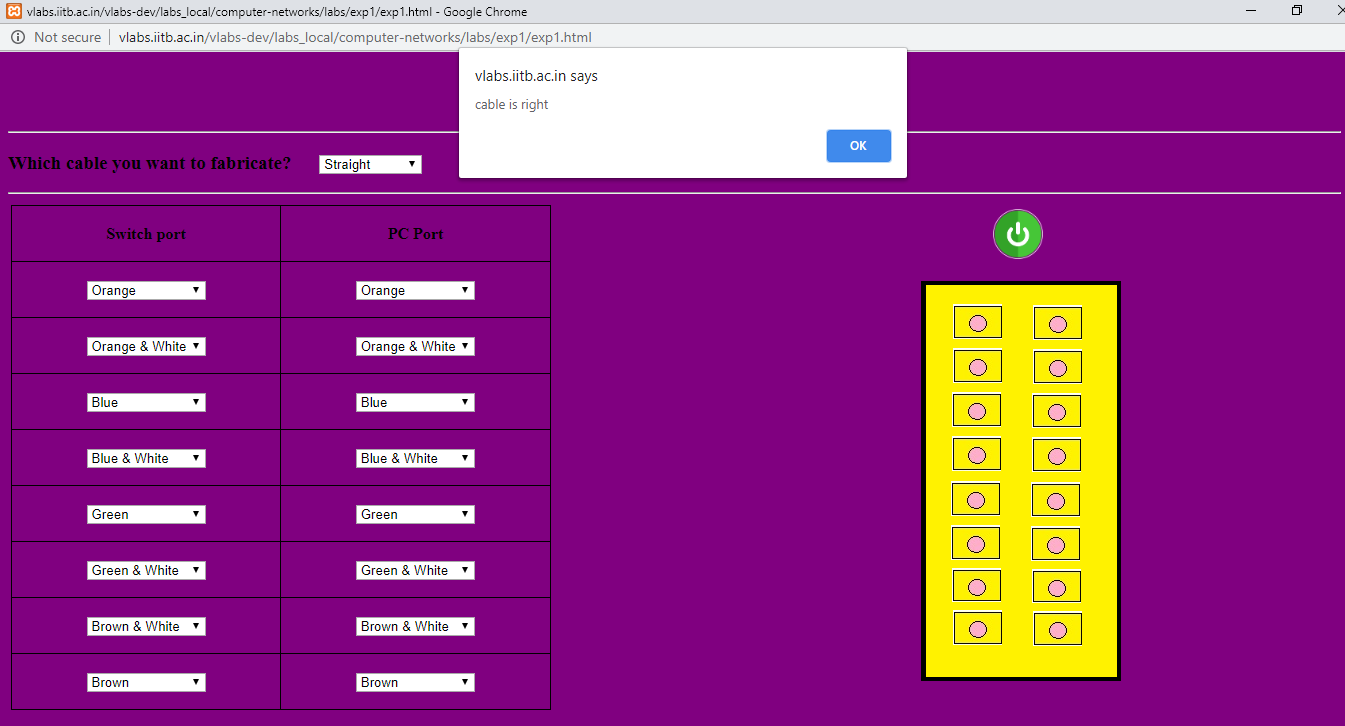
**Program Name:**Fabrication of Cables

**Description:**A twisted pair consists of two insulated conductor twisted together in the shape of a spiral. It can be shielded or unshielded. The unshielded twisted pair cables are very cheap and easy to install. But they are very badly affected by the electromagnetic noise interference.

Twisting of wires will reduce the effect of noise or external interference.The induced emf into the two wires due to interference tends to cancel each other due to twisting. Number of twists per unit length will determine the quality of cable. More twists means better quality.

There are 3 types of UTP cables:-

1. Straight-throughcable
2. Crossovercable
3. Roll-overcable



# Output:-

## Program No:2

**Program Name:**Peer to Peer Topology

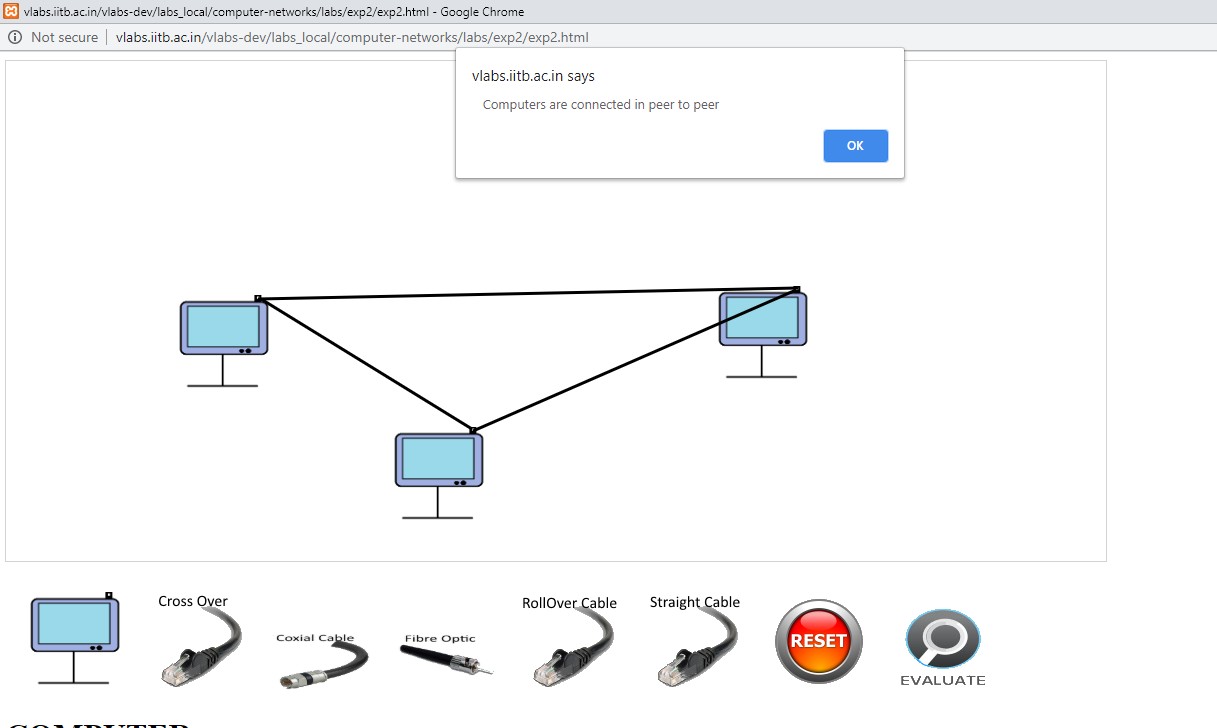
**Description:**The word physical network topology is used to explain the manner in which a network is physically connected. Devices or nodes in a network get connected to each other via communication links and all these links are related to each other in one way or the other. The geometric representation of such a relationship of links and nodes is known as the topology of that network.

These topologies can be classifies into two types:-

* 1. Peer topeer
  2. Primary -Secondary

Peer to peer is the relationship where the devices share the link equally. The examples are ring and mesh topologies.

In Primary - Secondary relationship, one device controls and the other devices have to transmit through it. For example star and tree topology.



# Output:-

## Program No:3

**Program Name:**Star Topology

**Description:**The word physical network topology is used to explain the manner in which a network is physically connected. Devices or nodes in a network get connected to each other via communication links and all these links are related to each other in one way or the other. The geometric representation of such a relationship of links and nodes is known as the topology of that network.

These topologies can be classifies into two types:-

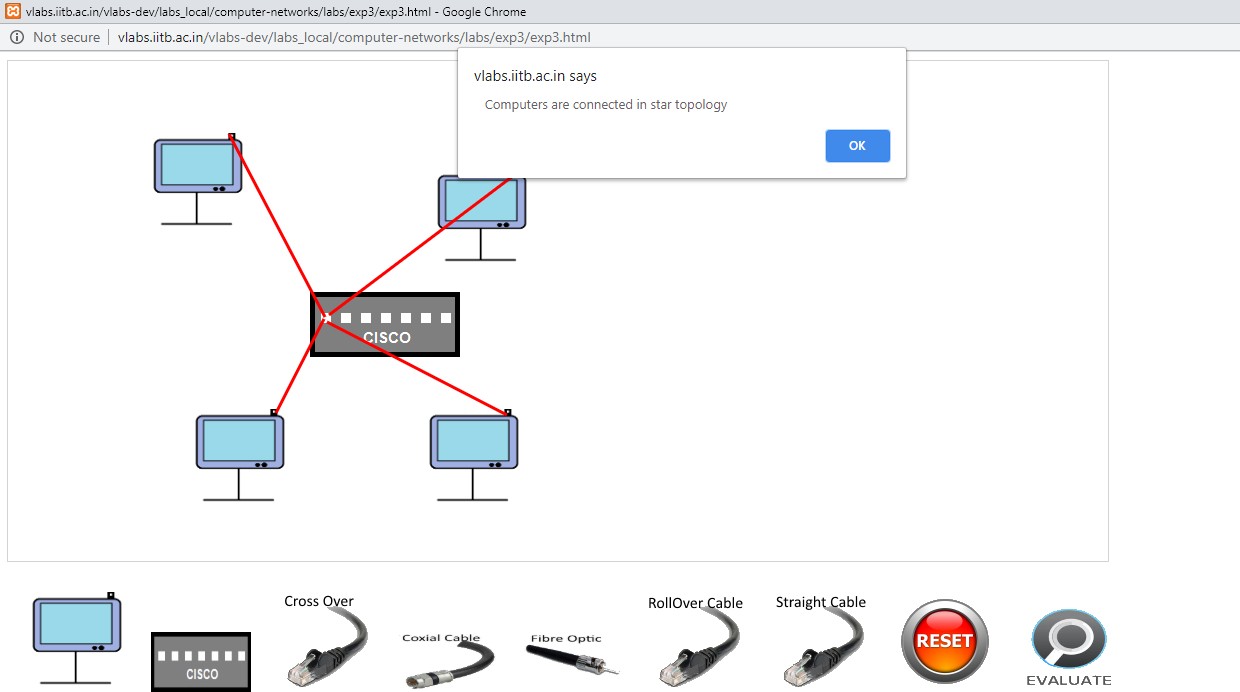
1. Peer topeer
2. Primary -Secondary

Peer to peer is the relationship where the devices share the link equally. The examples are ring and mesh topologies.

In Primary - Secondary relationship, one device controls and the other devices have to transmit through it. For example star and tree topology.

Features of Star Topology:-

1. Every node has its own dedicated connection to thehub.
2. Hub acts as a repeater for dataflow.
3. Can be used with twisted pair, Optical Fibre or coaxialcable.



# Output:-

## Program No:4

**Program Name:**IPv4 Addressing

**Description:**IP addresses enable computers to communicate by providing unique identifiers for the computer itself and for the network over which it is located. An IP address is a 32 bit value that contains a network identifier(net -id) and a host identifier (host-id).

The network administrators need to assign IP addresses to the system on their network. This address needs to be a unique one. All the computers on a particular subnet will have the same network identifier but different host identifiers. The Internet Assigned Numbers Authority (IANA) assigns network identifiers to avoid any duplication of addresses.

Host Identifier Network Identifier 32 bits

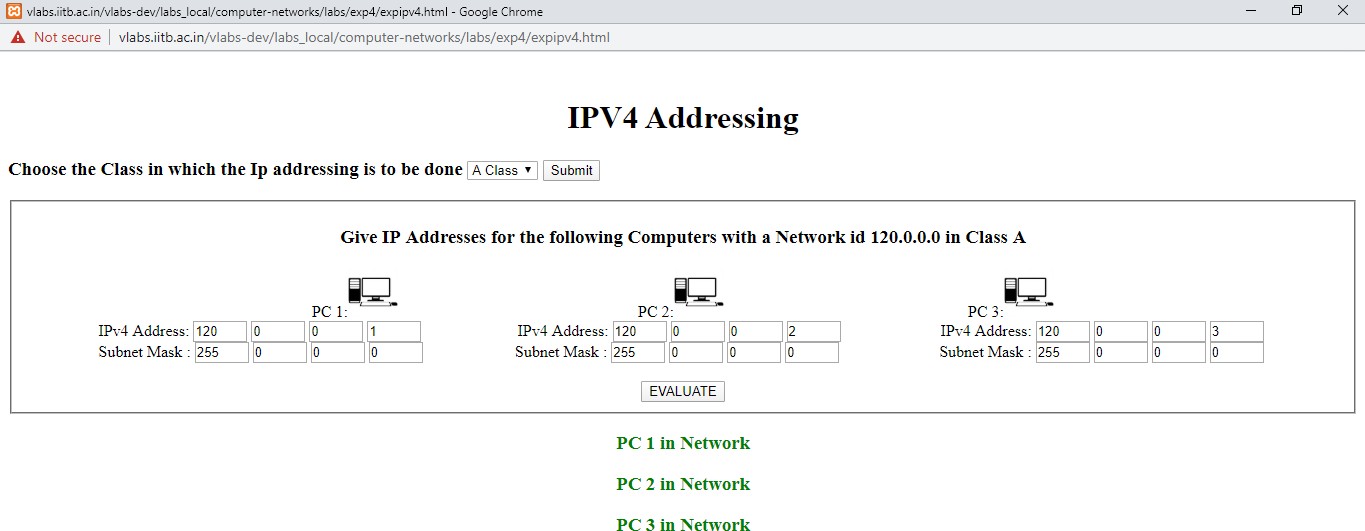
The 32 bit IPv4 address is grouped into groups of eight bits, separated by dots. Each 8 bit group is then converted into its equivalent binary number. Thus each octet (8bit) can take value from 0 to 255. The IPv4 in the dotted decimal notation can range from

0.0.0.0 to 255.255.255.255. The IPv4 Address are classified into 5 types as follows:

1.ClassA 2. ClassB

3.ClassC 4. ClassD

5. Class E

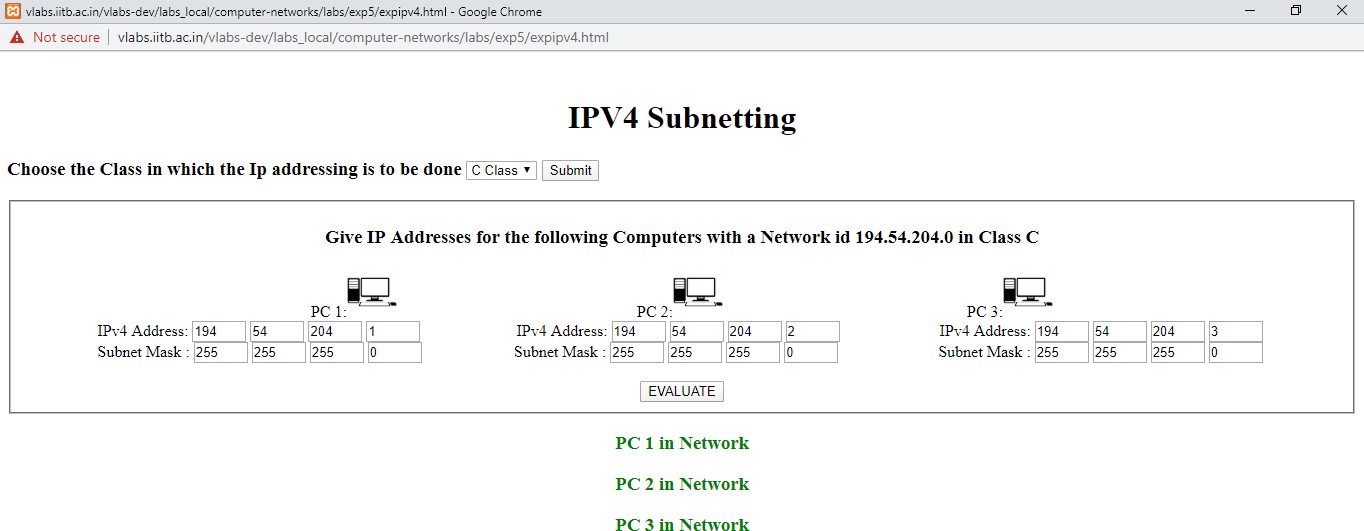


# Output:-

## Program No:5

**Program Name:**IPv4 Subnetting

**Description:**Each IP class is equipped with its own default subnet mask which bounds that IP class to have prefixed number of Networks and prefixed number of Hosts per network. Glassful IP addressing does not provide any flexibility of having less number of Hosts per Network or more Networks per IP Class. CIDR or Classless Inter Domain Routing provides the flexibility of borrowing bits of Host part of the IP address and using them as Network in Network, called Subnet. By using subletting, one singled Class A IP address can be used to have smaller sub\_networks which provides better network managementcapabilities.



# Output:-

## Program No:6

**Program Name:**TCP Client-Server

**Description:**The interSystems IRIS Transmission Control Protocols(TCP) binding establishes a two way connection between a server and a single client. Provides reliable byte stream transmission of data with error checking and correction and message acknowledgement.

## Algorithm:

#### TCP Server –

1. using socket(), Create TCPsocket.
2. using bind(), Bind the socket to serveraddress.
3. using listen(), put the server socket in a passive mode, whereit waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is establishedbetween client and server, and they are ready to transferdata.
5. Go back to Step3.

#### TCP Client –

1. Create TCPsocket.
2. Build server addressstructure.
3. connect newly created client socket to server.

# TCP Client-Server:-

### Server:-

#include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <errno.h> #include <string.h>

int main()

{

int sock, connected, bytes\_recieved , true = 1; char send\_data [1024] , recv\_data[1024];

struct sockaddr\_in server\_addr,client\_addr; int sin\_size;

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) { perror("Socket");

exit(1);

}

if (setsockopt(sock,SOL\_SOCKET,SO\_REUSEADDR,&true,sizeof(int)) == -1) { perror("Setsockopt");

exit(1);

}

server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(5000); server\_addr.sin\_addr.s\_addr = INADDR\_ANY; bzero(&(server\_addr.sin\_zero),8);

if (bind(sock, (struct sockaddr \*)&server\_addr, sizeof(struct sockaddr)) == -1) { perror("Unable to bind");

exit(1);

}

if (listen(sock, 5) == -1) { perror("Listen"); exit(1);

}

printf("\nTCPServer Waiting for client on port 5000"); fflush(stdout);

while(1)

{

sin\_size = sizeof(struct sockaddr\_in);

connected = accept(sock, (struct sockaddr \*)&client\_addr,&sin\_size); printf("\n I got a connection from (%s , %d)",

inet\_ntoa(client\_addr.sin\_addr),ntohs(client\_addr.sin\_port));

while (1)

{

printf("\n SEND (q or Q to quit) : ");

gets(send\_data);

if (strcmp(send\_data , "q") == 0 || strcmp(send\_data , "Q") == 0)

{

send(connected, send\_data,strlen(send\_data), 0); close(connected);

break;

}

else

send(connected, send\_data,strlen(send\_data), 0); bytes\_recieved = recv(connected,recv\_data,1024,0); recv\_data[bytes\_recieved] = '\0';

if (strcmp(recv\_data , "q") == 0 || strcmp(recv\_data , "Q") == 0)

{

close(connected); break;

}

else

printf("\n RECIEVED DATA = %s " , recv\_data); fflush(stdout);

}

}

close(sock); return 0;

}

### Client:-

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

int main()

{

int sock, bytes\_recieved;

char send\_data[1024],recv\_data[1024]; struct hostent \*host;

struct sockaddr\_in server\_addr; host = gethostbyname("127.0.0.1");

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) { perror("Socket");

exit(1);

}

server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(5000);

server\_addr.sin\_addr = \*((struct in\_addr \*)host->h\_addr); bzero(&(server\_addr.sin\_zero),8);

if (connect(sock, (struct sockaddr \*)&server\_addr, sizeof(struct sockaddr)) == -1)

{

perror("Connect"); exit(1);

}

while(1)

{

bytes\_recieved=recv(sock,recv\_data,1024,0); recv\_data[bytes\_recieved] = '\0';

if (strcmp(recv\_data , "q") == 0 || strcmp(recv\_data , "Q") == 0)

{

close(sock); break;

}

else

printf("\nRecieved data = %s " , recv\_data);

printf("\nSEND (q or Q to quit) : "); gets(send\_data);

if (strcmp(send\_data , "q") != 0 && strcmp(send\_data , "Q") != 0)

send(sock,send\_data,strlen(send\_data), 0);

else

{

send(sock,send\_data,strlen(send\_data), 0); close(sock);

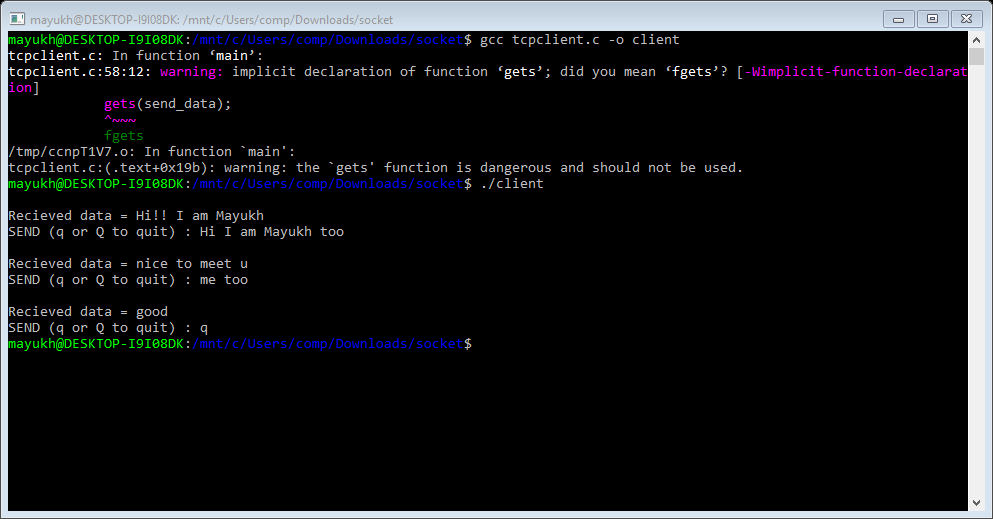
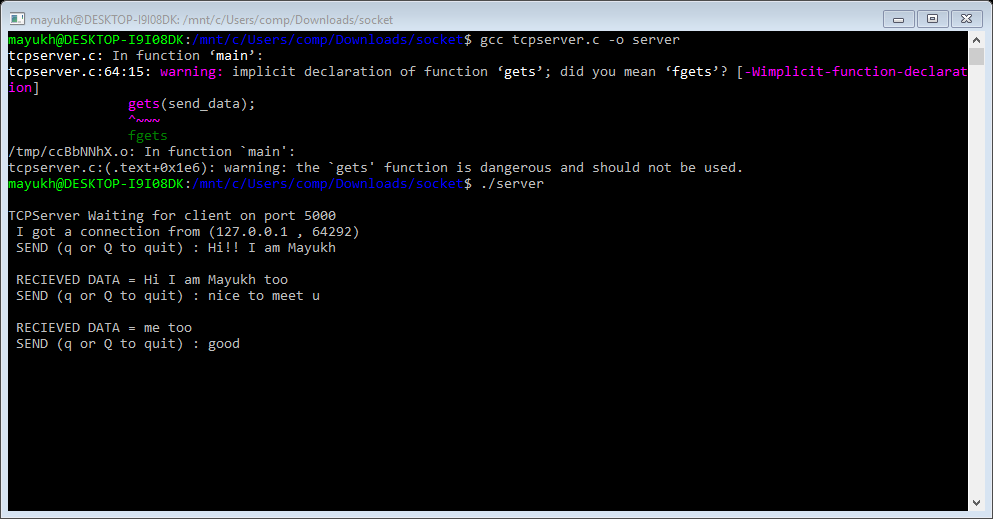
break;

}

}

return 0;

}



**Output:-**

**Server:-**

**Client:-**

**Program No:**7

**Program Name:**Concurrent Client-Server

**Description:**Establishes a two way connection between a server and a single client. Provides reliable byte stream transmission of data with error checking and correction and messageacknowledgement.

## 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: Close the socket

STEP 9: 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: Close the socket

STEP 9: Stop

# Concurrent Client-Server:-

### Server:-

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

#define MAX\_MSG 100

int main (int argc, char \*argv[]) { int sd, newSd, cliLen, n;

struct sockaddr\_in cliAddr, servAddr; char line[MAX\_MSG];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* check command line args \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(argc < 3) {

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build server address structure \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero(&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* create stream socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0){

printf("%s : cannot create stream socket \n", argv[0]); exit(-1);

}

else

printf("%s : successfully created stream socket \n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* bind local port number \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(bind(sd, (struct sockaddr \*) &servAddr, sizeof(servAddr)) < 0) { printf("%d\n", bind(sd, (struct sockaddr \*) &servAddr, sizeof(servAddr))); printf("%s : cannot bind port \n", argv[0]);

exit(-1);

}

else

printf("%s : bound local port successfully\n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* specify number of concurrent \*/

/\* clients tolistenfor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if ((listen(sd, 5)) != 0) { printf("Listen failed...\n"); exit(0);

}

else

printf("Server listening..\n"); int i=0;

while(i<5) { i++;

printf("%s %d: waiting for client connection on port TCP %u\n",argv[0], getpid(), atoi(argv[2]));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* wait for client connection\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ cliLen=sizeof(cliAddr);

newSd = accept(sd, (struct sockaddr \*) &cliAddr, &cliLen);

if(newSd<0) {

printf("%s : cannot accept connection \n", argv[0]); exit(-1);

}

else

printf("%s %d: received connection from host [IP %s ,TCP port

%d]\n",argv[0],getpid(),inet\_ntoa(cliAddr.sin\_addr),ntohs(cliAddr.sin\_port));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* wait for data from client \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(fork()==0)

{

do{

memset(line,0x0,MAX\_MSG);

n=recv(newSd, line, MAX\_MSG, 0); line[n]='\n';

printf("%s %d: received from host [IP %s ,TCP port %d] : %s\n", argv[0],getpid(), inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port), line);

}while(abs(strcmp(line, "quit")));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* close client connection\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("%s %d: closing connection with host [IP %s ,TCP port

%d]\n",argv[0],getpid(),inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port));

close(newSd); exit(1);

}

else

close(newSd);

}

}

### Client:-

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

#define MAX\_MSG 100

int main (int argc, char \*argv[]) {

int sd, rc, i;

struct sockaddr\_in servAddr, clientAddr; char line[MAX\_MSG];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* check command line args \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(argc < 3) {

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build server address structure \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero(&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build client address structure \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero(&clientAddr, sizeof(clientAddr)); clientAddr.sin\_family = AF\_INET; clientAddr.sin\_addr.s\_addr = INADDR\_ANY; clientAddr.sin\_port = htons(0);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* create stream socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0){

printf("%s: cannot create stream socket\n",argv[0]); exit(-1);

}

else

printf("%s : successfully created stream socket \n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* bind local port number \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

rc = bind(sd, (struct sockaddr \*) &clientAddr, sizeof(clientAddr)); if(rc<0){

printf("%s: cannot bind port TCP %s\n",argv[0], argv[1]); exit(1);

}

else

printf("%s: bound local port successfully\n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* connect to server \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

rc = connect(sd, (struct sockaddr \*) &servAddr,sizeof(servAddr)); if(rc<0){

printf("%s: cannot connect to server\n", argv[0]); exit(1);

}

else

printf("%s: connected to server successfully\n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* send data to server \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

do{

printf("Enter string to send to server : "); scanf("%s", line);

rc = send(sd, line, strlen(line) + 1, 0); if(rc<0) {

printf("%s: cannot send data\n", argv[0]); close(sd);

exit(1);

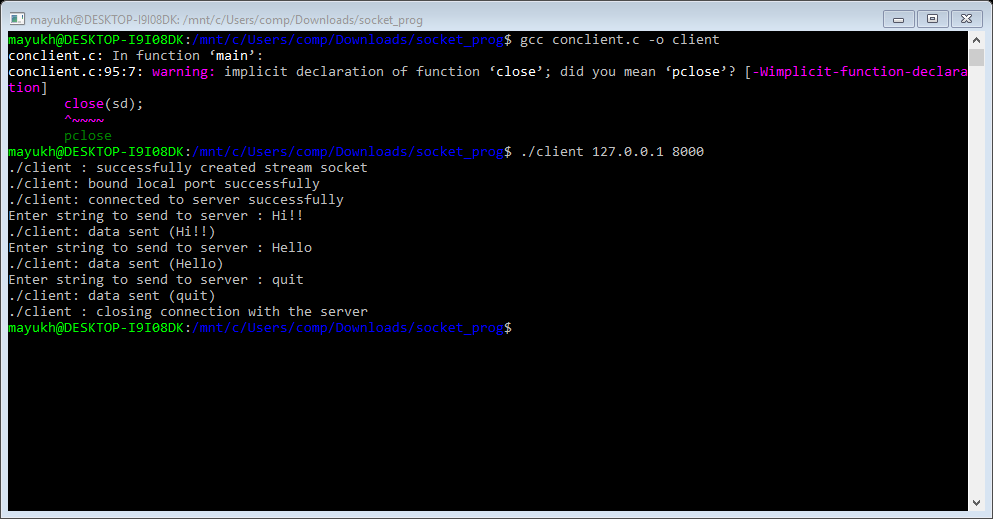
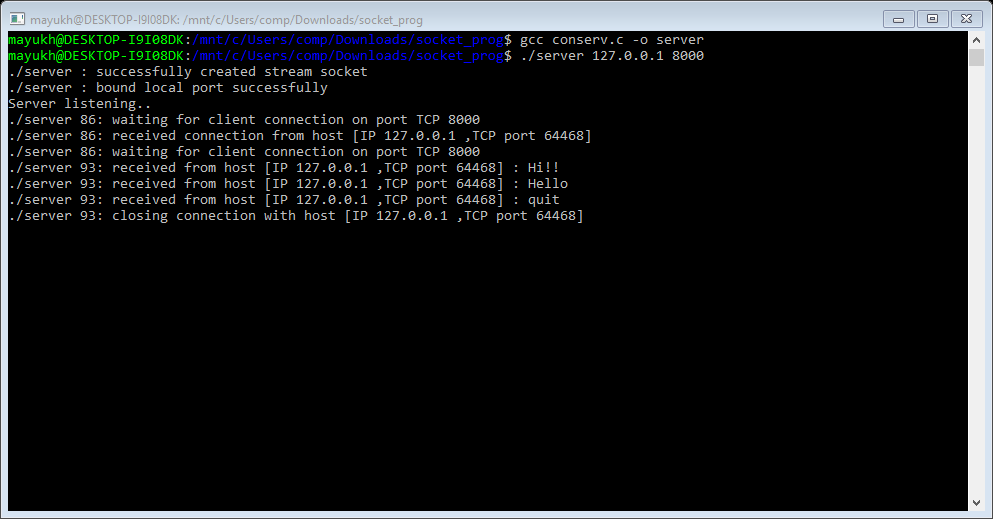
}

printf("%s: data sent (%s)\n",argv[0], line);

}while(strcmp(line, "quit"));

printf("%s : closing connection with the server\n", argv[0]); close(sd);

}



# Output:-

## Server:-

**Client:-**

**Program No: 8**

**Program Name:**Date Client-Server

**Description:**Establishes a two way connection between a server and a single client. Provides reliable byte stream transmission of data from client to server and server returns the date andtime.

## 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: If client says yes then send the date and time to the client 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: Receive the date and time from server and display it STEP 9: Close the socket

STEP 10: Stop

# Date Client-Server:-

### Server:-

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

#define MAX\_MSG 100

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

{

int sd,sd1, newSd, cliLen=0,n,i=0; floatjj=0;

FILE\*fp;

struct sockaddr\_in cliAddr, servAddr,clientAddr; char line[MAX\_MSG],ch;

if(argc < 3) {

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

bzero((char \*)&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0)

{

printf("%s : cannot create stream socket \n", argv[0]); exit(-1);

}

else

printf("%s : successfully created stream socket \n",argv[0]);

if(bind(sd, (struct sockaddr \*) &servAddr,sizeof(servAddr))<0)

{

printf("%s : cannot bind port \n", argv[0]); exit(-1);

}

else

printf("%s : bound local port successfully\n", argv[0]); listen(sd,5);

printf("%s: waiting for client connection on port TCP %u\n",argv[0],atoi(argv[2])); cliLen=sizeof(cliAddr);

newSd = accept(sd, (struct sockaddr \*) &cliAddr, &cliLen); if(newSd<0)

{

}

else

printf("%s : cannot accept connection \n", argv[0]); exit(-1);

printf("%s: received connection from host [IP %s ,TCP port

%d]\n",argv[0],inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port));

memset(line,0x0,MAX\_MSG); n=recv(newSd, line, MAX\_MSG, 0);

printf("%s: received from host [IP %s ,TCP port %d] : %s\n", argv[0],inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port),line);

if(strcmp(line,"yes")==0)

{

system("date>day.txt"); fp=fopen("day.txt","r+"); if(fp==NULL)

{

}

else

{

printf("ERROR opening file or no such file"); strcpy(line,"UNAVAILABLE"); send(newSd,line,strlen(line)+1,0);

strcpy(line,"AVAILABLE"); send(newSd,line,strlen(line)+1,0); for(jj=0;jj<=100;)

jj=jj+.00001;

printf("\nSENDINGDATA. \n");

printf("\n \n");do

{

ch=fgetc(fp); send(newSd,&ch,sizeof(ch),0); if(ch!=EOF)

printf("%c",ch);

}while(ch!=EOF);fclose(fp);

printf("\n \n");

printf("DATA SENT::\n");

}

else

{

}

strcpy(line,"UNAVAILABLE"); send(newSd,line,strlen(line)+1,0);

printf("%s: closing connection with host [IP %s ,TCP port

%d]\n",argv[0],inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port));

close(newSd); exit(1);

}

printf("%s: closing connection with host [IP %s ,TCP port

%d]\n",argv[0],inet\_ntoa(cliAddr.sin\_addr),ntohs(cliAddr.sin\_port));

close(newSd); exit(1);

}

### Client:-

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

#define MAX\_MSG 100

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

{

int sd,newsd, rc, i=0; FILE \*fp;

struct sockaddr\_in clientAddr, servAddr,cliaddr; char line[MAX\_MSG],line1[MAX\_MSG],ch;

if(argc < 3)

{

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

bzero((char \*)&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

bzero((char \*)&clientAddr, sizeof(clientAddr)); clientAddr.sin\_family = AF\_INET; clientAddr.sin\_addr.s\_addr = INADDR\_ANY; clientAddr.sin\_port = htons(0);

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0)

{

}

else

printf("%s: cannot create stream socket\n", argv[0]); exit(-1);

printf("%s : successfully created stream socket \n", argv[0]);

rc = bind(sd, (struct sockaddr \*) &clientAddr, sizeof(clientAddr)); if(rc<0)

{

}

else

printf("%s: cannot bind port TCP %s\n",argv[0], argv[1]); exit(1);

printf("%s: bound local port successfully\n", argv[0]);

rc = connect(sd, (struct sockaddr \*) &servAddr, sizeof(servAddr));

if(rc<0)

{

}

else

printf("%s: cannot connect to server\n", argv[0]); exit(1);

printf("%s: connected to server successfully\n", argv[0]);

printf("DO YOU WANT TO SEE THE DATE(TYPE yes) : ");

scanf("%s", line);

rc = send(sd, line, strlen(line) + 1, 0); if(rc<0)

{

}

else

printf("%s: cannot send data\n", argv[0]); close(sd);

exit(1);

printf("%s: data sent (%s)\n",argv[0], line);

recv(sd,line1,MAX\_MSG,0); if(strcmp(line1,"UNAVAILABLE")==0)

{

}

else

{

printf("SORRY INFORMATION UNAVAILABLE \n");

printf("\nTODAY'S DATE & TIME:\n"); do

{

recv(sd,&ch,sizeof(ch),0); if(ch!=EOF) printf("%c",ch);

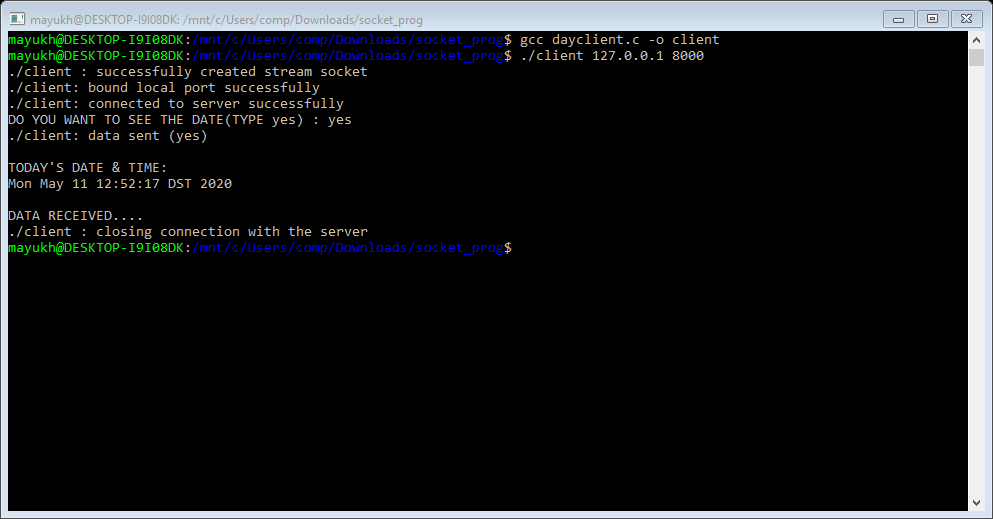
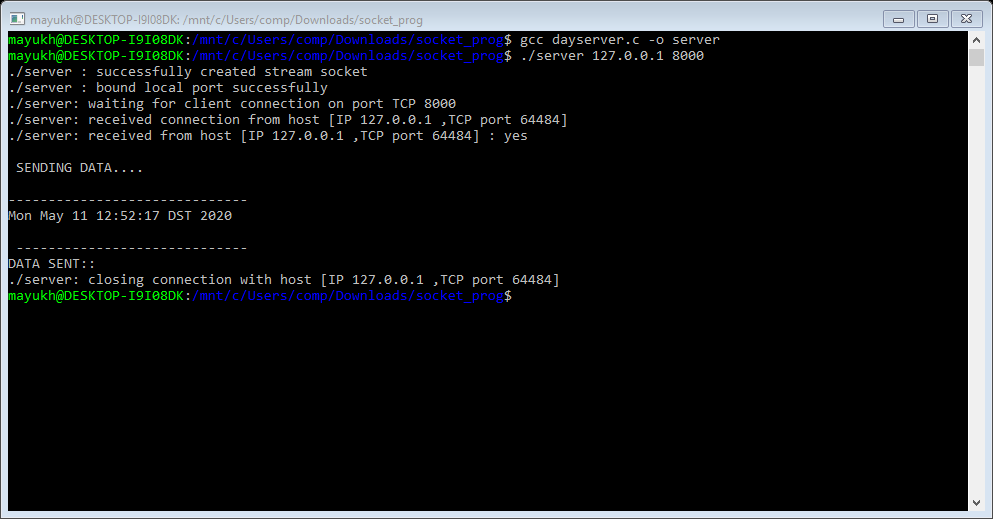
}while(ch!=EOF); printf("\nDATARECEIVED. \n");

}

printf("%s : closing connection with the server\n", argv[0]); close(sd);

exit(1);

}



# Output:-

## Server:-

**Client:-**

**Program No: 9**

**Program Name:**Echo Client-Server

**Description:**Establishes a two way connection between a server and a single client. Provides reliable byte stream transmission of data from client to server and server returns the same message to theclient.

## 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

# Echo Client-Server:-

### Server:-

#include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <netdb.h> #include <stdio.h>

#include <unistd.h> #include <stdlib.h> #include <string.h>

#define MAX\_MSG 100

int main (int argc, char \*argv[]) { int sd, newSd, cliLen, n;

struct sockaddr\_in cliAddr, servAddr; char line[MAX\_MSG];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* check command line args \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(argc < 3) {

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build server address structure \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero((char \*)&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* create stream socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0) {

printf("%s : cannot create stream socket \n", argv[0]);

exit(-1);

}

else

printf("%s : successfully created stream socket \n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* bind local port number \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(bind(sd, (struct sockaddr \*) &servAddr, sizeof(servAddr))<0) { printf("%s : cannot bind port \n", argv[0]);

exit(-1);

}

else

printf("%s : bound local port successfully\n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* specify number of concurrent \*/

/\* clients tolistenfor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ listen(sd,5);

while(1) {

printf("%s %d: waiting for client connection on port TCP %u\n",argv[0], getpid(), atoi(argv[2]));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* wait for client connection\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ cliLen = sizeof(cliAddr);

newSd = accept(sd, (struct sockaddr \*) &cliAddr, &cliLen);

if(newSd<0) {

printf("%s : cannot accept connection \n", argv[0]); exit(-1);

}

else

printf("%s %d: received connection from host [IP %s ,TCP port %d]\n",argv[0], getpid(), inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* wait for data from client \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(fork()==0)

{

do{

memset(line,0x0,MAX\_MSG);

n=recv(newSd, line, MAX\_MSG, 0); line[n]='\n';

printf("%s %d: received from host [IP %s ,TCP port %d] : %s\n", argv[0],getpid(),inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port), line);

printf("\nSENDINGTOCLIENT \n");

send(newSd, line, strlen(line) + 1, 0);

}while(abs(strcmp(line, "quit")));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* close client connection\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("%s %d: closing connection with host [IP %s ,TCP port

%d]\n",argv[0],getpid(),inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port));

close(newSd); exit(1);

}

else

close(newSd);

}

}

### Client:-

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

#define MAX\_MSG 100

int main (int argc, char \*argv[]) { int sd, rc, i,n;

struct sockaddr\_in clientAddr, servAddr; char line[MAX\_MSG],line1[MAX\_MSG];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* check command line args \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(argc < 3) {

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build server address structure \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero((char \*)&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build client address structure \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero((char \*)&clientAddr, sizeof(clientAddr)); clientAddr.sin\_family = AF\_INET; clientAddr.sin\_addr.s\_addr = INADDR\_ANY; clientAddr.sin\_port = htons(0);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* create stream socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0){

printf("%s: cannot create stream socket\n",argv[0]); exit(-1);

}

else

printf("%s : successfully created stream socket \n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* bind local port number \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

rc = bind(sd, (struct sockaddr \*) &clientAddr,sizeof(clientAddr)); if(rc<0){

printf("%s: cannot bind port TCP %s\n",argv[0], argv[1]); exit(1);

}

else

printf("%s: bound local port successfully\n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* connect to server \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

rc = connect(sd, (struct sockaddr \*) &servAddr,sizeof(servAddr)); if(rc<0){

printf("%s: cannot connect to server\n", argv[0]); exit(1);

}

else

printf("%s: connected to server successfully\n", argv[0]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* send data to server \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

do{

printf("Enter string to send to server : "); gets(line);

rc = send(sd, line, strlen(line) + 1, 0); if(rc<0) {

printf("%s: cannot send data\n", argv[0]); close(sd);

exit(1);

}

printf("%s: data sent (%s)\n",argv[0], line); printf("\nRECEIVING FROM SERVER:");

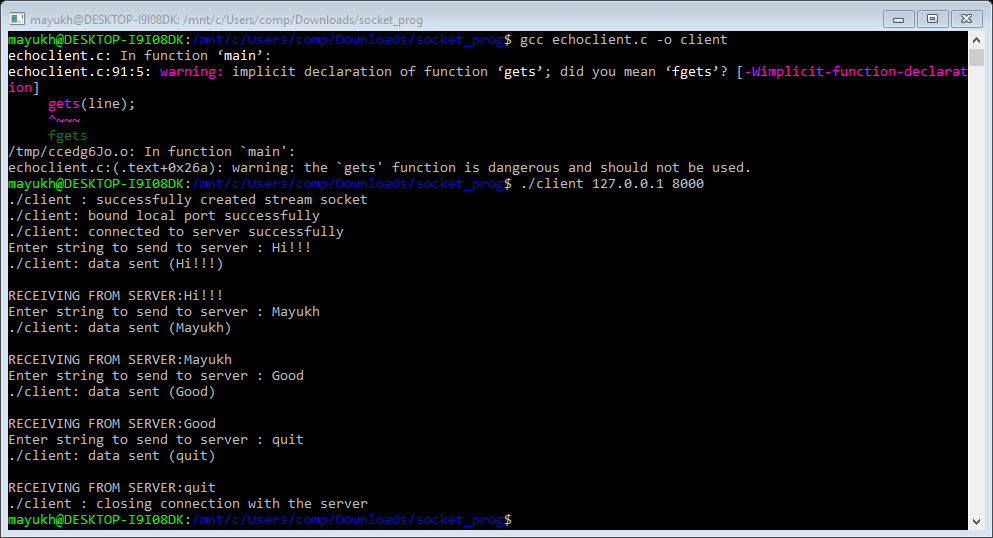
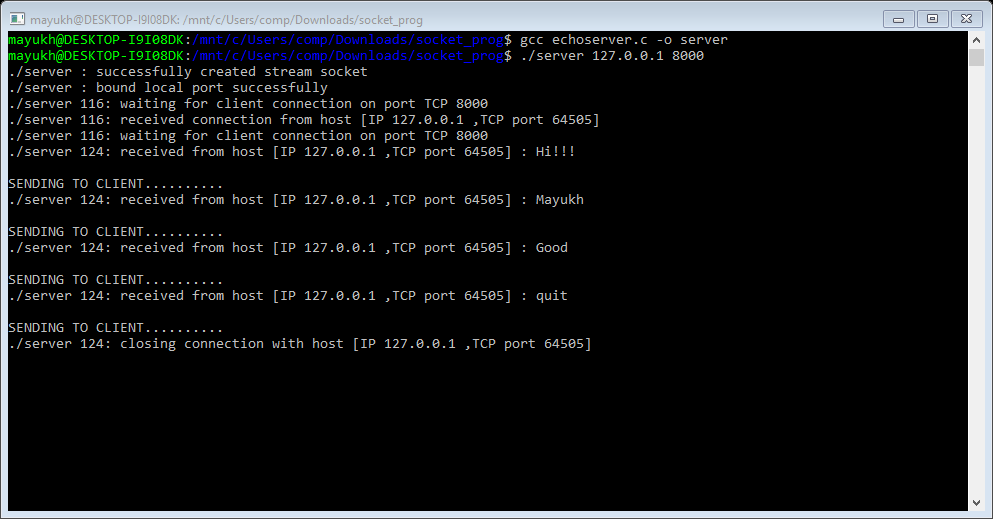
n=recv(sd, line1, MAX\_MSG, 0);

line1[n]='\n'; printf("%s\n",line1);

}while(strcmp(line, "quit"));

printf("%s : closing connection with the server\n", argv[0]); close(sd);

}



# Output:-

## Server:-

**Client:-**

**Program No: 10**

**Program Name:**FTP Client-Server

**Description:**A File Transfer Protocol client (FTP client) is a software utility that establishes a connection between a host computer and a remote server, typically an FTP server. An FTP client provides the dual-direction transfer of data and files between two computers over a TCP network or an Internet connection. An FTP client works on a client/server architecture, where the host computer is the client and the remote FTP server is the centralserver.

## 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: Open the file if exists and display file content to server

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 file name to the server

STEP 8: Display the server’s message of Available or not STEP 9: Close the socket

STEP 10: Stop

# FTP Client-Server:-

### Server:-

#include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <netdb.h>

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

#define MAX\_MSG 100

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

{

int sd,sd1, newSd, cliLen=0,n,i=0; floatjj=0;

FILE\*fp;

struct sockaddr\_in cliAddr, servAddr,clientAddr; char line[MAX\_MSG],ch;

if(argc < 3) {

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

bzero((char \*)&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0)

{

}

else

printf("%s : cannot create stream socket \n",argv[0]); exit(-1);

printf("%s : successfully created stream socket \n",argv[0]);

if(bind(sd, (struct sockaddr \*) &servAddr, sizeof(servAddr))<0)

{

}

else

printf("%s : cannot bind port \n", argv[0]); exit(-1);

printf("%s : bound local port successfully\n", argv[0]);

listen(sd,1);

printf("%s: waiting for client connection on port TCP

%u\n",argv[0],atoi(argv[2]));

cliLen = sizeof(cliAddr);

newSd = accept(sd, (struct sockaddr \*) &cliAddr, &cliLen); if(newSd<0)

{

printf("%s : cannot accept connection \n", argv[0]); exit(-1);

}

else

printf("%s: received connection from host [IP %s ,TCP port

%d]\n",argv[0],inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port));

memset(line,0x0,MAX\_MSG); n=recv(newSd, line, MAX\_MSG, 0);

printf("%s: received from host [IP %s ,TCP port %d] : %s\n", argv[0],inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port),line);

fp=fopen(line,"r"); if(fp==NULL)

{

}

else

{

printf("ERROR opening file or no such file"); strcpy(line,"UNAVAILABLE"); send(newSd,line,strlen(line)+1,0);

strcpy(line,"AVAILABLE"); send(newSd,line,strlen(line)+1,0); printf("FILE COPY IS IN PROGRESS \n");

while(jj<=100)

jj=jj+.0001; printf("FILE CONTENT:\n");

printf("\n \n");do

{

ch=fgetc(fp); if(ch!=EOF) printf("%c",ch);

send(newSd,&ch,sizeof(ch),0);

}while(ch!=EOF); fclose(fp);

printf("\n \n");printf("END OF FILE::\n"); printf("ONE FILE COPIED\n");

}

printf("%s: closing connection with host [IP %s

,TCPport%d]\n",argv[0],inet\_ntoa(cliAddr.sin\_addr),ntohs(cliAddr.sin\_port)); close(newSd);

exit(1);

}

### Client:-

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

#define MAX\_MSG 100

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

{

int sd,newsd, rc, i=0; FILE \*fp;

struct sockaddr\_in clientAddr, servAddr,cliaddr; char line[MAX\_MSG],line1[MAX\_MSG],ch;

if(argc < 3)

{

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

bzero((char \*)&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

bzero((char \*)&clientAddr, sizeof(clientAddr)); clientAddr.sin\_family = AF\_INET; clientAddr.sin\_addr.s\_addr = INADDR\_ANY; clientAddr.sin\_port = htons(0);

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0)

{

}

else

printf("%s: cannot create stream socket\n", argv[0]); exit(-1);

printf("%s : successfully created stream socket \n", argv[0]);

rc = bind(sd, (struct sockaddr \*) &clientAddr, sizeof(clientAddr)); if(rc<0)

{

}

else

printf("%s: cannot bind port TCP %u\n",argv[0], argv[1]); exit(1);

printf("%s: bound local port successfully\n", argv[0]);

rc = connect(sd, (struct sockaddr \*) &servAddr, sizeof(servAddr)); if(rc<0)

{

}

else

printf("%s: cannot connect to server\n", argv[0]); exit(1);

printf("%s: connected to server successfully\n", argv[0]);

printf("ENTER REQUIRED FILE NAME : ");

scanf("%s", line);

rc = send(sd, line, strlen(line) + 1, 0); if(rc<0)

{

printf("%s: cannot send data\n", argv[0]); close(sd);

}

else

exit(1);

printf("%s: data sent (%s)\n",argv[0], line);

recv(sd,line1,MAX\_MSG,0); printf("\n %s\n",line1);

if(strcmp(line1,"UNAVAILABLE")==0)

{

}

else

{

printf("NO SUCH FILE EXISTS \n");

close(sd); exit(1);

fp=fopen("pro2.txt","w"); if(fp==NULL)

{

puts("COPYING FAILED<CAN NOT CREATE DESTINATION FILE \n");

fclose(fp); close(sd); exit(1);

}

else{

do

{

recv(sd,&ch,sizeof(ch),0);if(ch!=EOF)

putc(ch,fp);

}while(ch!=EOF); fclose(fp);

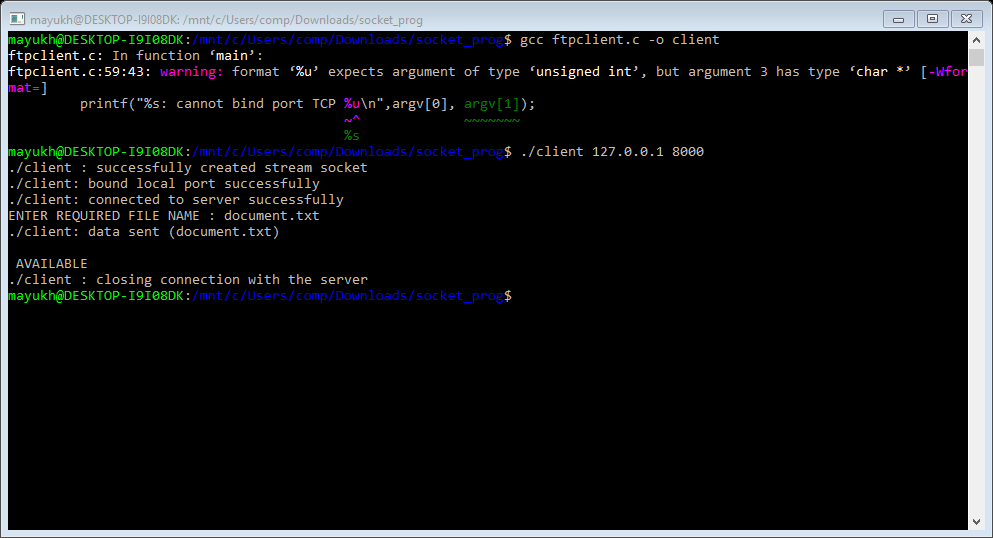
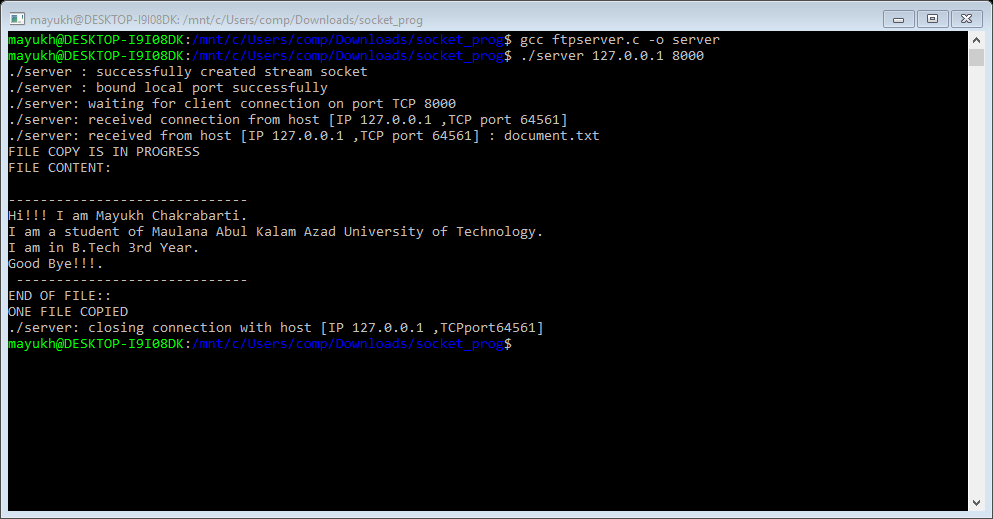
}

}

printf("%s : closing connection with the server\n", argv[0]); close(sd);

exit(1);

}



# Output:-

## Server:-

**Client:-**

**Program No: 11**

**Program Name:**Math Client-Server

**Description:**Establishes a two way connection between a server and a single client. Provides reliable byte stream transmission of data which consists two operands and a operator from client to server and server returns the same output value to theclient.

## 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 and output value and send the output to client

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 output value sent by server STEP 9: Close the socket

STEP 10: Stop

# Math Client-Server:-

### Server:-

#include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h>

#include <netdb.h> #include <stdio.h> #include <unistd.h> #include <stdlib.h> #include <string.h>

#define MAX\_MSG 100

int main (int argc, char \*argv[]) {

int sd,sd1, newSd, cliLen, n,i=0,op[3],res=0; struct sockaddr\_in cliAddr, servAddr,clientAddr; char line[MAX\_MSG],opr[MAX\_MSG];

if(argc < 3) {

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

bzero((char \*)&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0){

printf("%s : cannot create stream socket \n", argv[0]); exit(-1);

}

else

printf("%s : successfully created stream socket \n", argv[0]);

if(bind(sd, (struct sockaddr \*) &servAddr, sizeof(servAddr))<0) { printf("%s : cannot bind port \n", argv[0]);

exit(-1);

}

else

printf("%s : bound local port successfully\n", argv[0]); listen(sd,5);

printf("%s: waiting for client connection on port TCP %u\n",argv[0],atoi(argv[2]));

cliLen=sizeof(cliAddr);

newSd = accept(sd, (struct sockaddr \*) &cliAddr, &cliLen);

if(newSd<0) {

printf("%s : cannot accept connection \n", argv[0]); exit(-1);

}

else

printf("%s: received connection from host[IP%s,TCPport%d]\n",argv[0],inet\_ntoa(cliAddr.sin\_addr),

ntohs(cliAddr.sin\_port));

do{

memset(line,0x0,MAX\_MSG); n=recv(newSd, line, MAX\_MSG, 0); op[i]=atoi(line);

printf("%s: received from host [IP %s ,TCP port %d] : %s\n", argv[0],inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port), line);

i++;

}while(i<2);

memset(opr,0x0,MAX\_MSG); recv(newSd,opr,MAX\_MSG,0);

printf("%s: received from host [IP %s ,TCP port %d] : %s\n", argv[0], inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port), opr);

if(\*opr=='+')

{

res=op[0]+op[1];

printf("The Result obtained : %d\n",res);

}

else if(\*opr=='-')

{

res=op[0]-op[1];

printf("The Result obtained : %d\n",res);

}

else if(\*opr=='\*')

{

res=op[0]\*op[1];

printf("The Result obtained : %d\n",res);

}

else if(\*opr=='/')

{

res=op[0]/op[1];

printf("The Result obtained : %d\n",res);

}

else if(\*opr=='%')

{

res=op[0]%op[1];

printf("The Result obtained : %d\n",res);

}

else

{

res=-9990001;

printf("Unrecognized Symbol ( %s )\n",opr);

}

send(newSd,&res,sizeof(res),0);

printf("%s: Result Sent ( %d )\n",argv[0],res);

printf("%s: closing connection with host [IP %s ,TCP port

%d]\n",argv[0],inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port)); close(newSd);

}

### Client:-

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

#define MAX\_MSG 100

int main (int argc, char \*argv[]) { int sd,newSd, rc, i,res=5,cliLen;

struct sockaddr\_in clientAddr, servAddr,cliAddr;

char line[MAX\_MSG];

if(argc < 3) {

printf("usage: %s <server-addr> <server-port>\n",argv[0]); exit(1);

}

bzero((char \*)&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(argv[1]); servAddr.sin\_port = htons(atoi(argv[2]));

bzero((char \*)&clientAddr, sizeof(clientAddr)); clientAddr.sin\_family = AF\_INET; clientAddr.sin\_addr.s\_addr = INADDR\_ANY; clientAddr.sin\_port = htons(0);

sd = socket(AF\_INET, SOCK\_STREAM, 0); if(sd<0){

printf("%s: cannot create stream socket\n",argv[0]); exit(-1);

}

else

printf("%s : successfully created stream socket \n", argv[0]);

rc = bind(sd, (struct sockaddr \*) &clientAddr,sizeof(clientAddr)); if(rc<0){

printf("%s: cannot bind port TCP %s\n",argv[0], argv[1]); exit(1);

}

else

printf("%s: bound local port successfully\n", argv[0]);

rc = connect(sd,(struct sockaddr \*)&servAddr,sizeof(servAddr)); if(rc<0){

printf("%s: cannot connect to server\n", argv[0]); exit(1);

}

else

printf("%s: connected to server successfully\n", argv[0]);

printf("Enter first operand to send to server : "); scanf("%s", line);

rc = send(sd, line, strlen(line) + 1, 0); printf("%s: data sent (%s)\n",argv[0], line);

printf("Enter second operand to send to server : "); scanf("%s", line);

rc = send(sd, line, strlen(line) + 1, 0); printf("%s: data sent (%s)\n",argv[0], line);

printf("Enter operator to send to server : "); scanf("%s", line);

rc = send(sd, line, strlen(line) + 1, 0); printf("%s: data sent (%s)\n",argv[0], line);

recv(sd,&res,sizeof(res),0); if(res!=-9990001)

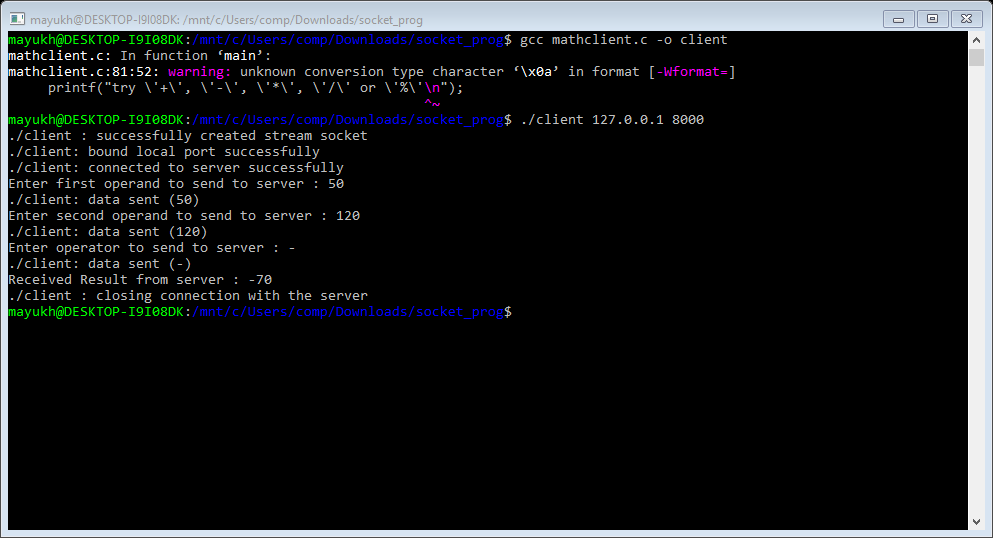
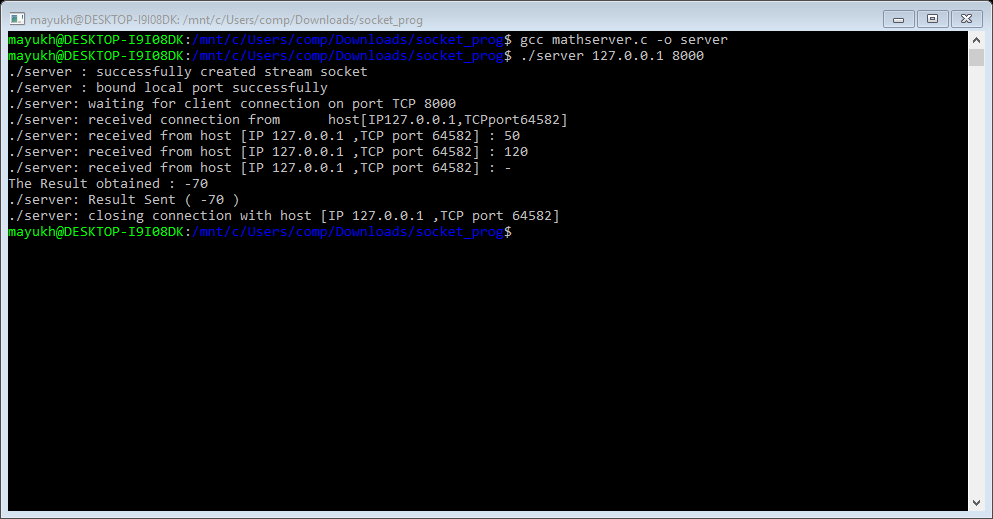
printf("Received Result from server : %d\n",res); else{

printf("Error Message from Server : Unrecognized Symbol ( %s ), ",line); printf("try \'+\', \'-\', \'\*\', \'/\' or \'%\'\n");

}

printf("%s : closing connection with the server\n", argv[0]); close(sd);

}



# Output:-

## Server:-

**Client:-**

**Program No: 12**

**Program Name:**UDP Client-Server

**Description:**In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.

## Algorithm:

**SERVER :**

STEP 1: Create UDP socket.

STEP 2: Bind the socket to server address.

STEP 3: Wait until datagram packet arrives from client.

STEP 4: Process the datagram packet and send a reply to client. STEP 5: Go back to Step 3.

**CLIENT :**

STEP 1: Create UDP socket. STEP 2: Send message to server.

STEP 3: Wait until response from server is recieved.

STEP 4: Process reply and go back to step 2, if necessary. STEP 5: Close socket descriptor and exit.

# UDP Client-Server:-

### Server:-

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

#define MAX\_MSG 100

#define SERVER\_ADDR "127.0.0.1"

#define SERVER\_PORT 8000

int main() {

int sd, rc, n, cliLen;

struct sockaddr\_in cliAddr, servAddr; char msg[MAX\_MSG];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build server address structure \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero((char \*)&servAddr, sizeof(servAddr)); servAddr.sin\_family = AF\_INET; servAddr.sin\_addr.s\_addr = inet\_addr(SERVER\_ADDR); servAddr.sin\_port = htons(SERVER\_PORT);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* create datagram socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sd=socket(AF\_INET, SOCK\_DGRAM, 0); printf("datagram socket created succefully\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* bind local port number \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bind (sd, (struct sockaddr \*) &servAddr,sizeof(servAddr)); printf("successfully bound local address\n");

printf("waiting for data on port UDP %u\n", SERVER\_PORT);

while(1) {

/\* init buffer \*/ memset(msg,0x0,MAX\_MSG);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* receive data from client \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

cliLen = sizeof(cliAddr);

n = recvfrom(sd, msg, MAX\_MSG, 0, (struct sockaddr \*) &cliAddr, &cliLen);

printf("from %s: UDP port %u : %s \n", inet\_ntoa(cliAddr.sin\_addr), ntohs(cliAddr.sin\_port),msg);

puts("enter msg for sending to client:"); gets(msg);

sendto(sd, msg, strlen(msg)+1, 0, (struct sockaddr \*) &cliAddr, sizeof(cliAddr));

}

return0;

}

### Client:-

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

#define MAX\_MSG 100

#define SERVER\_ADDR "127.0.0.1"

#define SERVER\_PORT 8000

int main() {

int sd, rc, tempLen, n;

struct sockaddr\_in cliAddr, remoteServAddr, tempAddr; char msg[MAX\_MSG];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* build server address structure \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero((char \*)&remoteServAddr, sizeof(remoteServAddr)); remoteServAddr.sin\_family = AF\_INET; remoteServAddr.sin\_addr.s\_addr = inet\_addr(SERVER\_ADDR); remoteServAddr.sin\_port = htons(SERVER\_PORT);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* create datagram socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sd = socket(AF\_INET,SOCK\_DGRAM,0); printf("successfully created datagram socket\n");

do {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* send data to server \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

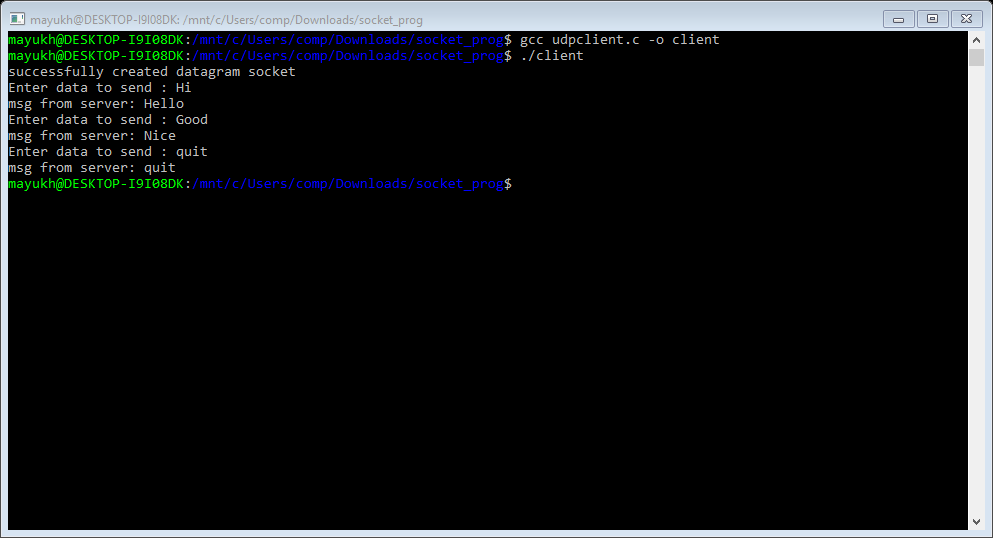
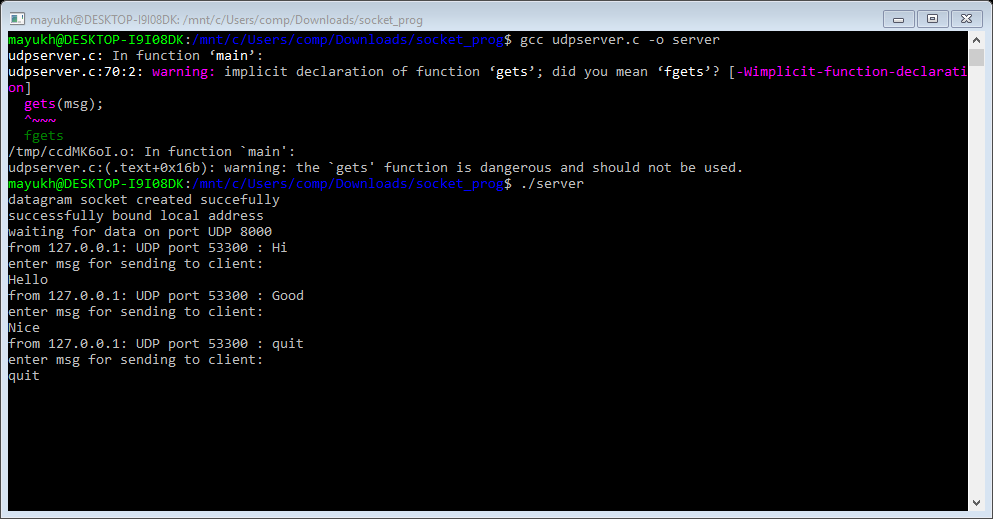
printf("Enter data to send : "); scanf("%s", msg);

sendto(sd, msg, strlen(msg)+1, 0, (struct sockaddr \*) &remoteServAddr, sizeof(remoteServAddr));

recvfrom(sd, msg, MAX\_MSG, 0, (struct sockaddr \*) &tempAddr, &tempLen); printf("msg from server: %s\n",msg);

}while(strcmp(msg, "quit")); close(sd);

}



# Output:-

## Server:-

**Client:-**

**Program No: 13**

**Program Name:**Sliding Window implementation in C

**Description:**In computer networks sliding window protocol is a method to transmit data on a network. Sliding window protocol is applied on the Data Link Layer of OSI model. At data link layer data is in the form of frames. In Networking, Window simply means a buffer which has data frames that needs to be transmitted.

Both sender and receiver agrees on some window size. If window size=w then after sending w frames sender waits for the acknowledgement (ack) of the first frame.

As soon as sender receives the acknowledgement of a frame it is replaced by the next frames to be transmitted by the sender. If receiver sends a collective or cumulative acknowledgement to sender then it understands that more than one frames are properly received, for eg:- if ack of frame 3 is received it understands that frame 1 and frame 2 are received properly.

## Algorithm:

* 1. We compute the sum of first k elements out of n terms using a linear loopand storethe sum in variablewindow\_sum.
  2. Then we will graze linearly over the array till it reaches the endand simultaneously keep track of maximumsum.
  3. To get the current sum of block of k elements just subtract the first elementfrom the previous block and add the last element of the current block.

## Sliding Window implementation in C :-

**Server:-**

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

int main()

{

int sock,size,connect;

char senddata[50],data[50]; int val,count,i,port;

struct sockaddr\_in ser,cli; printf("\n\nServerRunning ");

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

perror("\n Socket Creation Error"); exit(-1);

}

printf("\nEnter the port number : "); scanf("%d",&port);

ser.sin\_family = AF\_INET; ser.sin\_port = htons(port); ser.sin\_addr.s\_addr=INADDR\_ANY; bzero(&(ser.sin\_zero),8);

if(bind(sock,(struct sockaddr \*)&ser,sizeof(struct sockaddr)) == -1)

{

perror("\n\t Error in Bind"); exit(-1);

}

if (listen(sock,2)==-1)

{

perror("\n\t Error in Listen"); exit(-1);

}

printf("\n\t Waiting for connection "); size=sizeof(struct sockaddr); connect=accept(sock,(struct sockaddr\*)&cli,&size);

if(connect==-1)

{

perror("\n\t Connection Failed :"); exit(-1);

}

printf("\n\t ConnectedSuccessfully"); printf("\n");

// get the pocket number from client recv(connect,&val,sizeof(val),0); count=val;

while(1)

{

i=recv(connect,&data,sizeof(data),0); data[i]='\0';

if (strcmp(data,"end")==0)

{

printf("\n\t Finished"); break;

}

if(count!=val)

{

strcpy(senddata,"packet missing");

send(connect,&count,sizeof(count),0); send(connect,senddata,strlen(senddata),0);

}

else

{

printf("\n The packet Number is : %d",val); printf("\n The data is :%s",data);

count++;

strcpy(senddata,"send nextdata"); send(connect,&count,sizeof(count),0); send(connect,senddata,strlen(senddata),0);

}

printf("\n The Expected Packet now is: %d \n",count); recv(connect,&val,sizeof(val),0);

}

close(connect); close(sock); return 0;

}

Client:-

#include <sys/types.h> #include <netinet/in.h> #include <arpa/inet.h> #include <netdb.h> #include <stdio.h> #include <string.h> #include <stdlib.h> #include <unistd.h> #include <errno.h>

int main()

{

int sock,val,i,count,port;

char recvdata[50],sentdata[50]; struct sockaddr\_in server\_addr; printf("\n\nClientRunning ");

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

perror("Socket"); exit(1);

}

printf("\nEnter the port number"); scanf("%d",&port); server\_addr.sin\_family = AF\_INET; server\_addr.sin\_port = htons(port);

server\_addr.sin\_addr.s\_addr= htonl(INADDR\_ANY); bzero(&(server\_addr.sin\_zero),8);

if (connect(sock, (struct sockaddr \*)&server\_addr, sizeof(struct sockaddr)) == -1)

{

perror("Connect"); exit(1);

}

while(1)

{

//get the pack number from client printf("\n Enter packet number "); scanf("%d",&val);

// sent the value to server send(sock,&val,sizeof(val),0);

// get the data from the user printf("\n\n Enter data "); scanf("%s",sentdata);

// sent the to server send(sock,sentdata,strlen(sentdata),0);

if(strcmp(sentdata,"end")==0) break;

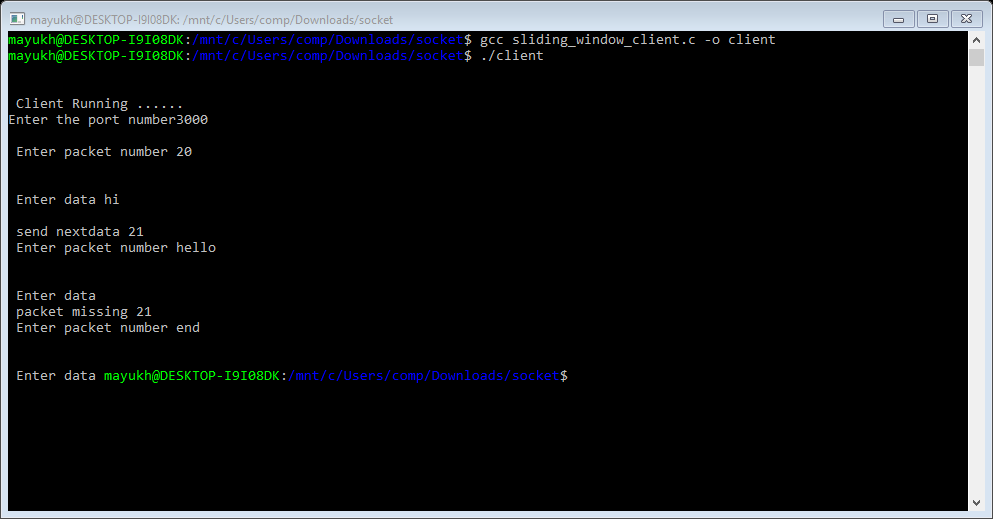
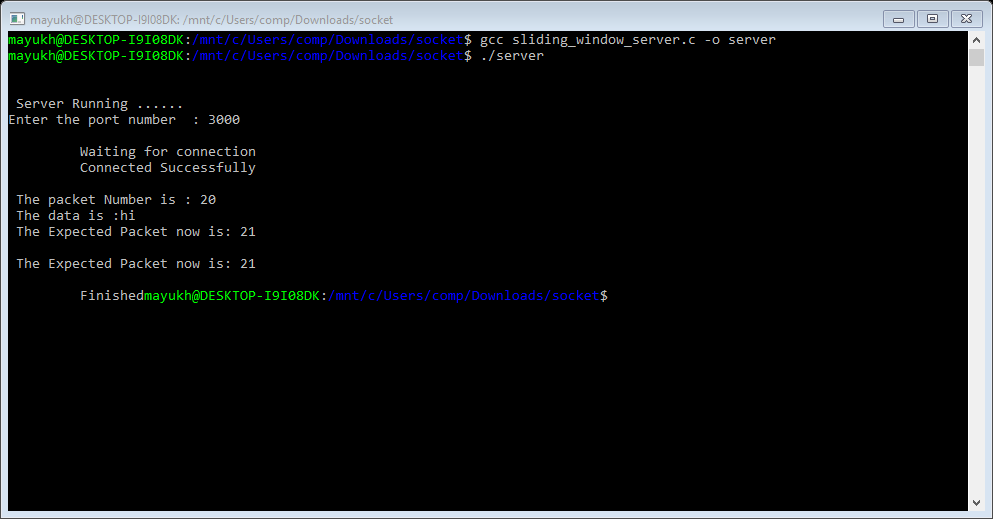
// recev the result from server recv(sock,&count,sizeof(count),0); i=recv(sock,recvdata,50,0); recvdata[i]='\0';

printf("\n %s %d",recvdata,count);

}

close(sock); return 0;

}



# Output:-

## Server:-

**Client:-**