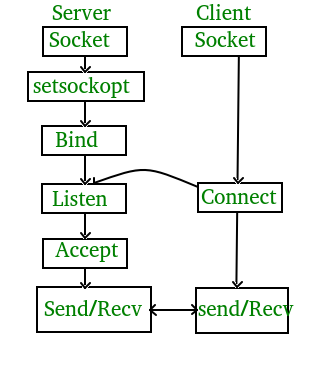
Socket Programming in C/C++

**What is socket programming?**  
Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

**State diagram for server and client model**  
[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Socket-Programming-in-C-C-.jpg)

**Stages for server**

* **Socket creation:**

intsockfd = socket(domain, type, protocol)

**sockfd:** socket descriptor, an integer (like a file-handle)  
**domain:** integer, communication domain e.g., AF\_INET (IPv4 protocol) , AF\_INET6 (IPv6 protocol)  
**type:** communication type  
SOCK\_STREAM: TCP(reliable, connection oriented)  
SOCK\_DGRAM: UDP(unreliable, connectionless)  
**protocol:**Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

* **Setsockopt:**
* intsetsockopt(intsockfd, int level, intoptname,

const void \*optval, socklen\_toptlen);

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

* **Bind:**
* int bind(intsockfd, conststructsockaddr \*addr,

socklen\_taddrlen);

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR\_ANY to specify the IP address.

* **Listen:**

int listen(intsockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

* **Accept:**

intnew\_socket= accept(intsockfd, structsockaddr \*addr, socklen\_t \*addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

**Stages for Client**

* **Socket connection:** Exactly same as that of server’s socket creation
* **Connect:**
* int connect(intsockfd, conststructsockaddr \*addr,

socklen\_taddrlen);

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server’s address and port is specified in addr.

**Implementation**  
Here we are exchanging one hello message between server and client to demonstrate the client/server model.

* server.c

| // Server side C/C++ program to demonstrate Socket programming  #include <stdio.h>  #include <sys/socket.h>  #include <stdlib.h>  #include <netinet/in.h>  #include <string.h>  #define PORT 8080  intmain(intargc, charconst\*argv[])  {      intserver\_fd, new\_socket, valread;      structsockaddr\_in address;      intopt = 1;      intaddrlen = sizeof(address);      charbuffer[1024] = {0};      char\*hello = "Hello from server";        // Creating socket file descriptor      if((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0)      {          perror("socket failed");          exit(EXIT\_FAILURE);      }        // Forcefully attaching socket to the port 8080      if(setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT,                                                    &opt, sizeof(opt)))      {          perror("setsockopt");          exit(EXIT\_FAILURE);      }      address.sin\_family = AF\_INET;      address.sin\_addr.s\_addr = INADDR\_ANY;      address.sin\_port = htons( PORT );        // Forcefully attaching socket to the port 8080      if(bind(server\_fd, (structsockaddr \*)&address,                                   sizeof(address))<0)      {          perror("bind failed");          exit(EXIT\_FAILURE);      }      if(listen(server\_fd, 3) < 0)      {          perror("listen");          exit(EXIT\_FAILURE);      }      if((new\_socket = accept(server\_fd, (structsockaddr \*)&address,                         (socklen\_t\*)&addrlen))<0)      {          perror("accept");          exit(EXIT\_FAILURE);      }      valread = read( new\_socket , buffer, 1024);      printf("%s\n",buffer );      send(new\_socket , hello , strlen(hello) , 0 );      printf("Hello message sent\n");      return0;  } |
| --- |
| * client.c   // Client side C/C++ program to demonstrate Socket programming  #include <stdio.h>  #include <sys/socket.h>  #include <stdlib.h>  #include <netinet/in.h>  #include <string.h>  #define PORT 8080    int main(intargc, char const \*argv[])  {      structsockaddr\_in address;      int sock = 0, valread;      structsockaddr\_inserv\_addr;      char \*hello = "Hello from client";      char buffer[1024] = {0};      if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)      {          printf("\n Socket creation error \n");          return -1;      }        memset(&serv\_addr, '0', sizeof(serv\_addr));        serv\_addr.sin\_family = AF\_INET;      serv\_addr.sin\_port = htons(PORT);        // Convert IPv4 and IPv6 addresses from text to 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;      }        if (connect(sock, (structsockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0)      {          printf("\nConnection Failed \n");          return -1;      }      send(sock , hello , strlen(hello) , 0 );      printf("Hello message sent\n");      valread = read( sock , buffer, 1024);      printf("%s\n",buffer );      return 0;  } |

**Compiling:**  
gccclient.c -o client  
gccserver.c -o server

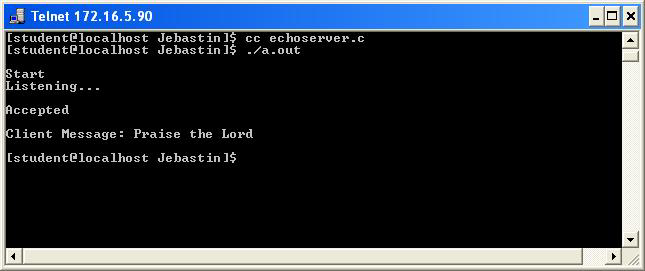
**Output:**

Client:Hello message sent

Hello from server

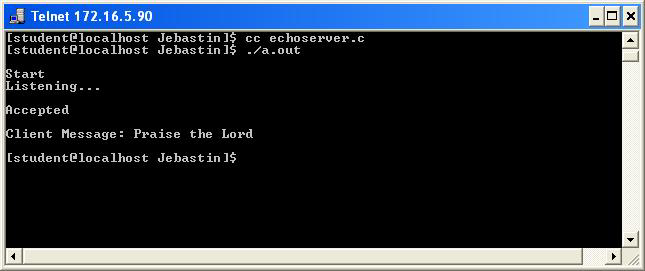
Server:Hello from client

Hello message sent

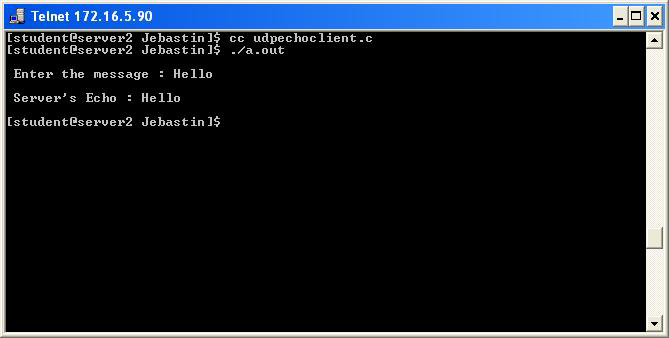
**AIM:  
           To write a program for TCP echo client server**.  
  
**ALGORITHM:SERVER:**  
    STEP 1: Start  
    STEP 2: Declare the variables for the socket  
    STEP 3: Specify the family, protocol, IP address and port number  
    STEP 4: Create a socket using socket() function  
    STEP 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  
  
**SOURCE CODE:  
SERVER:**  
#include<stdio.h>  
#include<netinet/in.h>  
#include<netdb.h>  
#define SERV\_TCP\_PORT 5035  
int main(intargc,char\*\*argv)  
{  
       intsockfd,newsockfd,clength;  
       structsockaddr\_inserv\_addr,cli\_addr;  
       char buffer[4096];  
       sockfd=socket(AF\_INET,SOCK\_STREAM,0);  
       serv\_addr.sin\_family=AF\_INET;  
       serv\_addr.sin\_addr.s\_addr=INADDR\_ANY;  
       serv\_addr.sin\_port=htons(SERV\_TCP\_PORT);  
       printf("\nStart");  
       bind(sockfd,(structsockaddr\*)&serv\_addr,sizeof(serv\_addr));  
       printf("\nListening...");  
       printf("\n");  
       listen(sockfd,5);  
       clength=sizeof(cli\_addr);  
       newsockfd=accept(sockfd,(structsockaddr\*)&cli\_addr,&clength);  
       printf("\nAccepted");  
       printf("\n");  
       read(newsockfd,buffer,4096);  
       printf("\nClient message:%s",buffer);  
       write(newsockfd,buffer,4096);  
       printf("\n");  
       close(sockfd);  
       return 0;  
}  
**CLIENT:**  
#include<stdio.h>  
#include<sys/types.h>  
#include<sys/socket.h>  
#include<netinet/in.h>  
#include<netdb.h>  
#define SERV\_TCP\_PORT 5035  
int main(intargc,char\*argv[])  
{  
       intsockfd;  
       structsockaddr\_inserv\_addr;  
       structhostent \*server;  
       char buffer[4096];  
       sockfd=socket(AF\_INET,SOCK\_STREAM,0);  
       serv\_addr.sin\_family=AF\_INET;  
       serv\_addr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");  
       serv\_addr.sin\_port=htons(SERV\_TCP\_PORT);  
       printf("\nReady for sending...");  
       connect(sockfd,(structsockaddr\*)&serv\_addr,sizeof(serv\_addr));  
       printf("\nEnter the message to send\n");  
       printf("\nClient: ");  
       fgets(buffer,4096,stdin);  
       write(sockfd,buffer,4096);  
       printf("Serverecho:%s",buffer);  
       printf("\n");  
       close(sockfd);  
       return 0;  
}  
  
**OUTPUT:  
SERVER:**  
  
[](http://4.bp.blogspot.com/_V648gqhagYA/TJjdjHLdRzI/AAAAAAAAAGQ/8ckw3ZEBw98/s1600/TCP+ECHO+SERVER.JPG)  
  
**CLIENT:**

[](http://2.bp.blogspot.com/_V648gqhagYA/TJjdlUy2OwI/AAAAAAAAAGY/6gwTkbc_1M0/s1600/TCP+ECHO+CLIENT.JPG)

**AIM:  
           To write a program for UDP echo client server.  
  
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 and Display the client’s message  
    STEP 8: 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  
  
**SOURCE CODE:  
SERVER:**  
#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  
int main(intargc,char \*\*argv)  
{  
intsockfd;  
int n;  
socklen\_tlen;  
char msg[1024];  
structsockaddr\_inservaddr,cliaddr;  
sockfd=socket(AF\_INET,SOCK\_DGRAM,0);  
bzero(&servaddr,sizeof(servaddr));  
servaddr.sin\_family=AF\_INET;  
servaddr.sin\_addr.s\_addr=INADDR\_ANY;  
servaddr.sin\_port=htons(5035);  
printf("\n\n Binded");  
bind(sockfd,(structsockaddr\*)&servaddr,sizeof(servaddr));  
printf("\n\n Listening...");  
for(;;)  
{  
      printf("\n ");  
     len=sizeof(cliaddr);  
     n=recvfrom(sockfd,msg,MAXLINE,0,(structsockaddr\*)&cliaddr,&len);  
     printf("\n Client's Message : %s\n",msg);  
     if(n<6)  
        perror("send error");  
     sendto(sockfd,msg,n,0,(structsockaddr\*)&cliaddr,len);  
}  
return 0;  
}  
  
**CLIENT:**  
#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(intargc,char\* argv[])  
{  
intsockfd;  
int n;  
socklen\_tlen;  
char sendline[1024],recvline[1024];  
structsockaddr\_inservaddr;  
strcpy(sendline,"");  
printf("\n Enter the message : ");  
scanf("%s",sendline);  
sockfd=socket(AF\_INET,SOCK\_DGRAM,0);  
bzero(&servaddr,sizeof(servaddr));  
servaddr.sin\_family=AF\_INET;  
servaddr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");  
servaddr.sin\_port=htons(5035);  
connect(sockfd,(structsockaddr\*)&servaddr,sizeof(servaddr));  
len=sizeof(servaddr);  
sendto(sockfd,sendline,MAXLINE,0,(structsockaddr\*)&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:SERVER:**

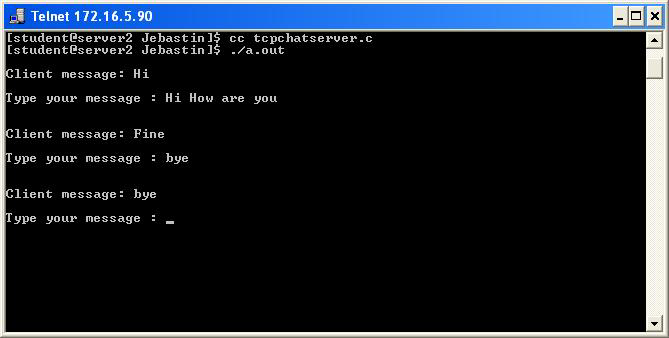
[](http://3.bp.blogspot.com/_V648gqhagYA/TJj3dT4rKsI/AAAAAAAAAIY/LNfTmxZGCow/s1600/TCP+ECHO+SERVER.JPG)

**CLIENT:**

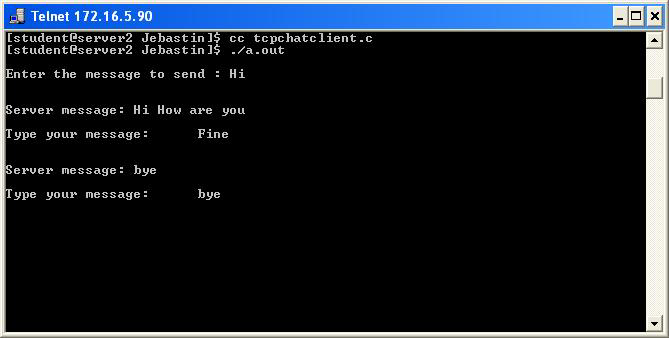
[](http://2.bp.blogspot.com/_V648gqhagYA/TJj3fE0_INI/AAAAAAAAAIg/lSxR1mfMens/s1600/UDP+ECHO+CLIENT.JPG)

**AIM:  
           To write a program for TCP chat between client and server.**  
  
**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: Continue the chat  
    STEP 10: Terminate the chat  
    STEP 11: Close the socket  
    STEP 12: 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 reply  
    STEP 9: Continue the chat  
    STEP 10: Terminate the chat  
    STEP 11: Close the socket  
    STEP 12: Stop

**SOURCE CODE:  
SERVER:**  
#include<stdio.h>  
#include<sys/types.h>  
#include<sys/socket.h>  
#include<netinet/in.h>  
#include<netdb.h>  
#define SERV\_TCP\_PORT 5035  
int main(intargc,char\*\*argv)  
{  
       intsockfd,newsockfd,clength;  
       structsockaddr\_inserv\_addr,cli\_addr;  
       char buffer[4096];  
       sockfd=socket(AF\_INET,SOCK\_STREAM,0);  
       serv\_addr.sin\_family=AF\_INET;  
       serv\_addr.sin\_addr.s\_addr=INADDR\_ANY;  
       serv\_addr.sin\_port=htons(SERV\_TCP\_PORT);  
       bind(sockfd,(structsockaddr\*)&serv\_addr,sizeof(serv\_addr));  
       listen(sockfd,5);  
       clength=sizeof(cli\_addr);  
       newsockfd=accept(sockfd,(structsockaddr\*)&cli\_addr,&clength);  
       read(newsockfd,buffer,4096);  
       while(buffer!="quit")  
       {  
         printf("\nClient message: %s",buffer);  
         printf("\nType your message : ");  
         fgets(buffer,4096,stdin);  
         write(newsockfd,buffer,4096);  
         printf("\n");  
         read(newsockfd,buffer,4096);  
       }  
       close(sockfd);  
       return 0;  
}  
  
**CLIENT:**  
#include<stdio.h>  
#include<sys/types.h>  
#include<sys/socket.h>  
#include<netinet/in.h>  
#include<netdb.h>  
#define SERV\_TCP\_PORT 5035  
int main(intargc,char\*\*argv)  
{  
       intsockfd;  
       structsockaddr\_inserv\_addr;  
       structhostent \*server;  
       char buffer[4096];  
       sockfd=socket(AF\_INET,SOCK\_STREAM,0);  
       serv\_addr.sin\_family=AF\_INET;  
       serv\_addr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");  
       serv\_addr.sin\_port=htons(SERV\_TCP\_PORT);  
       connect(sockfd,(structsockaddr\*)&serv\_addr,sizeof(serv\_addr));  
       printf("\nEnter the message to send : ");  
       fgets(buffer,4096,stdin);  
       fputs(buffer,stdout);  
       while(buffer!="quit")  
       {  
         if(buffer=="quit")  
         break;  
         write(sockfd,buffer,4096);  
         read(sockfd,buffer,4096);  
         printf("\n");  
         printf("\nServer message:\t%s",buffer);  
         printf("\nType your message:\t");  
         fgets(buffer,4096,stdin);  
       }  
       close(sockfd);  
       return(0);  
}  
  
**OUTPUT:  
SERVER:**

[](http://3.bp.blogspot.com/_V648gqhagYA/TJjiMwABccI/AAAAAAAAAGg/nvKyG2h450Q/s1600/TCP+CHAT+SERVER.JPG)

**CLIENT:**

[](http://1.bp.blogspot.com/_V648gqhagYA/TJjiOoD8jHI/AAAAAAAAAGo/nM7FrUzjF_c/s1600/TCP+CHAT+CLIENT.JPG)

**UDP SOCKETS CHAT APPLICATION (SERVER & CLIENT) USING C**

SERVER

#include<stdio.h>

#include<netinet/in.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netdb.h>

#include<string.h>

#include<stdlib.h>

#define MAX 80

#define PORT 43454

#define SA structsockaddr

voidfunc(intsockfd)

{

char buff[MAX];

intn,clen;

structsockaddr\_in cli;

clen=sizeof(cli);

for(;;)

{

bzero(buff,MAX);

recvfrom(sockfd,buff,sizeof(buff),0,(SA \*)&cli,&clen);

printf("From client %s To client",buff);

bzero(buff,MAX);

n=0;

while((buff[n++]=getchar())!='\n');

sendto(sockfd,buff,sizeof(buff),0,(SA \*)&cli,clen);

if(strncmp("exit",buff,4)==0)

{

printf("Server Exit...\n");

break;

}

}

}

int main()

{

intsockfd;

structsockaddr\_inservaddr;

sockfd=socket(AF\_INET,SOCK\_DGRAM,0);

if(sockfd==-1)

{

printf("socket creation failed...\n");

exit(0);

}

else

printf("Socket successfully created..\n");

bzero(&servaddr,sizeof(servaddr));

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

servaddr.sin\_port=htons(PORT);

if((bind(sockfd,(SA \*)&servaddr,sizeof(servaddr)))!=0)

{

printf("socket bind failed...\n");

exit(0);

}

else

printf("Socket successfully binded..\n");

func(sockfd);

close(sockfd);

}

CLIENT

#include<sys/socket.h>

#include<netdb.h>

#include<string.h>

#include<stdlib.h>

#include<stdio.h>

#define MAX 80

#define PORT 43454

#define SA structsockaddr

int main()

{

char buff[MAX];

intsockfd,len,n;

structsockaddr\_inservaddr;

sockfd=socket(AF\_INET,SOCK\_DGRAM,0);

if(sockfd==-1)

{

printf("socket creation failed...\n");

exit(0);

}

else

printf("Socket successfully created..\n");

bzero(&servaddr,sizeof(len));

servaddr.sin\_family=AF\_INET;

servaddr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");

servaddr.sin\_port=htons(PORT);

len=sizeof(servaddr);

for(;;)

{

printf("\nEnter string : ");

n=0;

while((buff[n++]=getchar())!='\n');

sendto(sockfd,buff,sizeof(buff),0,(SA \*)&servaddr,len);

bzero(buff,sizeof(buff));

recvfrom(sockfd,buff,sizeof(buff),0,(SA \*)&servaddr,&len);

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

if(strncmp("exit",buff,4)==0)

{

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

break;

}

}

close(sockfd);

}

OUTPUT

SERVER SIDE

$ cc udpchatserver.c

$ ./a.out

Socket successfully created..

Socket successfully binded..

From client hai

To client hello

From client exit

To client exit

Server Exit...

$

CLIENT SIDE

$ cc udpchatclient.c

$ ./a.out

Socket successfully created..

Enter string :hai

From Server : hello

Enter string : exit

From Server : exit

Client Exit...

$  
##################################################################################################################################################################################

TCP SOCKET(DATE AND TIME) in C

SERVER:

#include"netinet/in.h"

#include"sys/socket.h"#include"stdio.h"

#include"string.h"

#include"time.h"

main( )

{

structsockaddr\_insa;

structsockaddr\_incli;intsockfd,conntfd;intlen,ch;charstr[100];

time\_t tick;

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

if(sockfd<0)

{

printf("error in socket\n");

exit(0);

}

elseprintf("Socket opened");

bzero(&sa,sizeof(sa));

sa.sin\_port=htons(5600);

sa.sin\_addr.s\_addr=htonl(0);

if(bind(sockfd,(structsockaddr\*)&sa,sizeof(sa))<0)

{

printf("Error in binding\n");

}

else

printf("Binded Successfully");

listen(sockfd,50);

for(;;)

{

len=sizeof(ch);

conntfd=accept(sockfd,(structsockaddr\*)&cli,&len);

printf("Accepted");

tick=time(NULL);

snprintf(str,sizeof(str),"%s",ctime(&tick));

printf("%s",str);write(conntfd,str,100);

}

}

CLIENT:

#include"netinet/in.h"

#include"sys/socket.h"

#include"stdio.h"

main()

{

structsockaddr\_insa,cli;

intn,sockfd;

intlen;char buff[100];

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

if(sockfd<0){ printf("\nError in Socket");

exit(0);

}

elseprintf("\nSocket is Opened");

bzero(&sa,sizeof(sa));

sa.sin\_family=AF\_INET;

sa.sin\_port=htons(5600);

if(connect(sockfd,(structsockaddr\*)&sa,sizeof(sa))<0)

{

printf("\nError in connection failed");

exit(0);

}

else

printf("\nconnected successfully");

if(n=read(sockfd,buff,sizeof(buff))<0)

{

printf("\nError in Reading");

exit(0);

}

else

{printf("\nMessage Read %s",buff);

}}

SERVER SIDE:  
[shanthi@linuxserver ~]$ cc tcpcd.c  
[shanthi@linuxserver ~]$ cc tcpcd.c -o ll  
[shanthi@linuxserver ~]$ ./ll  
Socket is Openedconnected successfully  
Message Read Fri Sep 24 15:16:26 2010  
[shanthi@linuxserver ~]$  
CLIENT SIDE:  
[shanthi@linuxserver ~]$ cc tcpd.c  
[shanthi@linuxserver ~]$ cc tcpd.c -o kk  
[shanthi@linuxserver ~]$ ./kk  
SocketOpenedBindedSuccessfullyAccepted  
Fri Sep 24 15:16:26 2010

#########################################################################################

**TCP stop and wait ARQ implementation(client/server program)**

**Stop&wait.c**

include<stdio.h>

int timer=0,wait\_for\_ack=-1,frameQ=0,cansend=1,t=0;

main()

{

inti,j,k;

int frame[5];

//clrscr();

printf("enter the time when data frame will be ready\n");

for(j=0;j<3;j++)

{

sender(i, &frame);

recv(i);

}

}

{

wait\_for\_ack++;

if(wait\_for\_ack==3)

{

}

if(i==frame[t])

{

frameQ++;

t++;

}

if(frameQ==0)

printf("NO FRAME TO SEND at time=%d \n",i);

if(frameQ>0 &&cansend==1)

{

printf("FRAME SEND AT TIME=%d\n",i);

cansend=-1;

frameQ--;

timer++;

printf("timer in sender=%d\n",timer);

}

if(frameQ>0 &&cansend==-1)

printf("FRAME IN Q FOR TRANSMISSION AT TIME=%d\n",i);

if(frameQ>0)

t++;

printf("frameQ=%d\n",frameQ);

printf("i=%d t=%d\n",i,t);

printf("value in frame=%d\n",frame[t]);

// return 0;

}

recv(inti )

{ printf("

timer in recvr=%d\n",timer);

if(timer>0)

{

timer++;

}

if(timer==3)

{

printf("

FRAME ARRIVED AT TIME=%d\n",i);

wait\_for\_ack=0;

timer=0;

}

else

printf("

WAITING FOR FRAME AT TIME %d\n",i);

// return 0;

}

Now the next one is :

SENDER.C  
// ARQ - client server  
// sender.c  
//connection if he ready to receive else wait  
// till he does not ready or timeout fails .  
/\*server.c and client.c - implementation of go-back-n ARQ in C  
Server.c implements a reliable data transfer over UDP in C  
client.c implements a reliable data transfer client over UDP in C  
Both of these programs use the go-back-n ARQ, that is lost data is  
automatically resent. These programs are hardcoded to transfer  
to compile:  
gcc -o server server.c  
gcc -o client client.c  
\*/  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <unistd.h>  
#include <string.h>  
#include <fcntl.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <sys/stat.h>  
#define MAXLINE 80  
char buffer[255] ;  
#define SERVER\_ADDR "127.0.0.1"  
#define SERVER\_PORT\_NUM 6000  
#define t\_max\_retransmission 3  
typedefint bit\_32\_var ;  
typedef char bit\_8\_var ;

typedefstructpdu\_field  
{  
bit\_32\_var SN ;  
bit\_8\_var data[MAXLINE] ;  
bit\_8\_var status ;  
}PDU\_FIELD;  
void str\_cli ( FILE \*fp , bit\_32\_var sfd )

{

bit\_32\_var no\_of\_data , counter\_1 , counter\_2 ;  
static bit\_32\_var retransmission\_counter , intial\_readycheck\_counter ;  
PDU\_FIELD send\_data[MAXLINE] ,recv\_data[MAXLINE] ;  
printf ( "enter how many data you have to send : " ) ;  
scanf ( "%d", &no\_of\_data ) ;  
for ( counter\_1 = 0 ; counter\_1 <no\_of\_data ; counter\_1++ )  
{  
printf ( "enter %d'th data : ",counter\_1 ) ;  
scanf ( "%s", send\_data[counter\_1].data ) ;  
send\_data[counter\_1].SN = counter\_1 ;  
send\_data[counter\_1].status = 0 ;  
}

for ( counter\_1 = 0 ; counter\_1 <no\_of\_data ; counter\_1++ )  
{  
write ( sfd , &send\_data[counter\_1] , sizeof ( send\_data[counter\_1] ) ) ;  
read ( sfd , &recv\_data[counter\_1] , sizeof ( recv\_data[counter\_1] ) ) ;

if ( counter\_1 == 0 && ( strcmp ( recv\_data[counter\_1].data , "yes" ) != 0 ) )

{

if ( intial\_readycheck\_counter == t\_max\_retransmission )

{

printf ( "client : receiver not there , better to exit : \n" ) ;

exit ( 1 ) ;

}

intial\_readycheck\_counter++ ;

printf ( "client : receiver is not ready : wait 10sec... \n" ) ;

sleep ( 10 ) ;

counter\_1 = counter\_1- 1 ;

}

if ( counter\_1 > 0 )

{

if ( recv\_data[counter\_1].SN == send\_data[counter\_1].SN + 1 )

printf ( "\n server responding = %s ", recv\_data[counter\_1].data ) ;

else

{

if ( retransmission\_counter<t\_max\_retransmission )

{

printf ( "incorrect ack - sending the same data - \n" ) ;

counter\_1 -= 1 ;

retransmission\_counter++ ;

}

else

{

printf ( "Time out : sending next data \n " ) ;

retransmission\_counter = 0 ;

}

}

}

}

write ( sfd , "Nothingtotransmit:\n" , 10 ) ;  
printf ( "Transmission finished : \n" ) ;  
exit ( 1 ) ;  
}

int main ()

{

bit\_32\_var socket\_fd ;  
structsockaddr\_inconfig\_client ;

socket\_fd = socket ( AF\_INET , SOCK\_STREAM , 0 ) ;

if ( socket\_fd< 0 )  
{  
printf ( "client : failed to create socket \n" ) ;  
exit ( 1 ) ;  
}

memset ( &config\_client , 0 , sizeof ( structsockaddr\_in )) ;

config\_client.sin\_family = AF\_INET ;

config\_client.sin\_port = htons( SERVER\_PORT\_NUM ) ;

inet\_aton( SERVER\_ADDR , &config\_client.sin\_addr ) ;

connect ( socket\_fd , (structsockaddr \*)&config\_client , sizeof ( config\_client ) ) ;

printf ( "connect successfully\n" ) ;

system ( "clear" ) ;

str\_cli( stdin , socket\_fd ) ;

exit ( 0 );

}

RECEIVE.C

// ARQ - client server  
// Stop & Wait basics  
// receive.c  
// written at asmaitha wireless  
// contact - [contacts@asmaitha.com](mailto:contacts@asmaitha.com)  
/\*server.c and client.c - implementation of go-back-n ARQ in C  
by Elijah Jordan Montgomery [elijah.montgomery@uky.edu](mailto:elijah.montgomery@uky.edu)

Server.c implements a reliable data transfer over UDP in C  
client.c implements a reliable data transfer client over UDP in C

Both of these programs use the go-back-n ARQ, that is lost data is  
automatically resent. These programs are hardcoded to transfer  
to compile:

gcc -o server server.c  
gcc -o client client.c  
\*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <string.h>  
#define SUCCESS 1  
#define FAILURE 0  
#define NO\_OF\_FRAMES 10  
typedefint BIT\_32\_VAR\_INT ;  
typedef char BIT\_8\_VAR\_CHAR ;  
void error ( BIT\_8\_VAR\_CHAR \*msg )  
{  
perror ( msg ) ;  
exit ( 1 ) ;

}  
typedefstructpdu\_data  
{  
BIT\_32\_VAR\_INT SN ;  
BIT\_8\_VAR\_CHAR data[80] ;  
BIT\_32\_VAR\_INT status ;  
}PDU\_DATA;

int main () {

PDU\_DATA send\_data[10] , recv\_data[10] ;

BIT\_32\_VAR\_INT socket\_fd ,newsocket\_fd , port\_no , client\_len ;

// char buffer[256] ;

structsockaddr\_inserv\_addr , cli\_addr ;

intsizeof\_read\_data , counter\_1 ;

//char buf[256] ;

socket\_fd = socket ( AF\_INET , SOCK\_STREAM , 0 ) ;

if ( socket\_fd< 0 )

{

error ( "error in socket opening " ) ;

}

bzero ( ( char \* ) &serv\_addr , sizeof ( serv\_addr ) ) ;

port\_no = atoi( "7100" ) ;

serv\_addr.sin\_family = AF\_INET ;

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY ;

serv\_addr.sin\_port = htons( 6000 ) ;

if ( bind ( socket\_fd , ( structsockaddr \* ) &serv\_addr , sizeof ( serv\_addr ) ) < 0 )

error ( "error in binding" ) ;

listen ( socket\_fd , 5 ) ;

client\_len = sizeof( cli\_addr ) ;

newsocket\_fd = accept ( socket\_fd , ( structsockaddr \* ) &cli\_addr , &client\_len ) ;

if ( newsocket\_fd< 0 )

error ( "error in accept " ) ;

// bzero ( buffer , 256 ) ;

while ( 1 )

{

for ( counter\_1 = 0 ; counter\_1 < NO\_OF\_FRAMES ; counter\_1++ )

{

sizeof\_read\_data = read ( newsocket\_fd , &recv\_data[counter\_1] , sizeof ( recv\_data[counter\_1]) ) ;

if ( sizeof\_read\_data< 0 )

error ( "error in reading from socket " ) ;

printf ( "client sended : %s with SN %d :\n" , recv\_data[counter\_1].data , recv\_data[counter\_1].SN) ;

printf ( "enter a reply to client : " ) ;

scanf ( "%s",send\_data[counter\_1].data ) ;

printf ( "\nenter a SN for next you want to receive : " ) ;

scanf ( "%d",&send\_data[counter\_1].SN ) ;

if ( send\_data[counter\_1].SN <= recv\_data[counter\_1].SN )

counter\_1 -= 1 ;

send\_data[counter\_1].status = SUCCESS ;

write ( newsocket\_fd , &send\_data[counter\_1] , sizeof ( send\_data[counter\_1]) ) ;

}

bzero ( &recv\_data[counter\_1].data , sizeof ( recv\_data[counter\_1].data ) ) ;

read ( newsocket\_fd , &recv\_data[counter\_1] , sizeof ( recv\_data[counter\_1] ) ) ;

printf ( "it's end Thank YOu : \n" ) ;

exit ( 1 ) ;

}  
return 0 ;

}