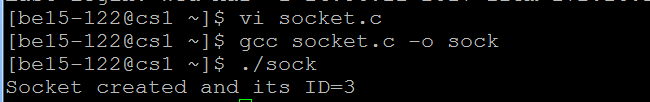
**1. Program to create a socket.**

#include<stdio.h>  
#include<sys/socket.h>  
int main()  
{  
 int sfd;  
 sfd=socket(AF\_INET, SOCK\_STREAM,0);//Create a TCP socket  
 if(sfd<0)//check for error  
 {   
 perror("socket() error:");  
 }  
 printf("Socket created and its ID=%d\n",sfd);  
 return 0;  
}

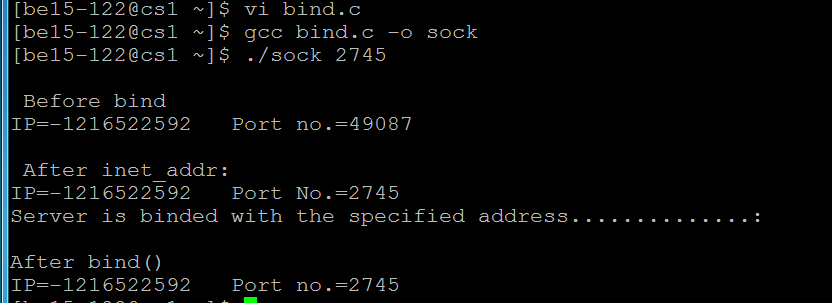
OUTPUT :



**2.Program to demonstrate bind() system call.**

#include<stdio.h>  
#include<netinet/in.h>  
#include<sys/types.h>  
#include<sys/socket.h>  
int main(int argc, char \*argv[])  
{  
 int sfd,s;  
 struct sockaddr\_in servaddr;  
 sfd=socket(AF\_INET, SOCK\_STREAM,0);//creates a socket  
 if(sfd<0)  
 {  
 perror("socket() error:");  
 }  
 servaddr.sin\_family=AF\_INET;//Internet family  
 printf("\n Before bind\n");  
 printf("IP=%d \t Port no.=%d\n",inet\_ntoa(servaddr.sin\_addr),ntohs(servaddr.sin\_port));  
 servaddr.sin\_port=htons(atoi(argv[1])); //convert the port number into network byte order  
 servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);//extracts the machine IP address and convert it into network byte order  
 //servadr.sin\_addr.s\_addr=inet\_addr(“172.16.0.100”);//converts decimal dotted address 172.16.0.100 to network byte order  
 printf("\n After inet\_addr: \n");  
 printf("IP=%d\t Port No.=%d\n",inet\_ntoa(servaddr.sin\_addr),ntohs(servaddr.sin\_port));  
 s=bind(sfd,(struct sockaddr\*)&servaddr,sizeof(servaddr));//binds the server with the IP address and port number  
 if(s<0)//check the return status of bind()  
 {  
 perror("bind() error:");  
 }  
 printf("Server is binded with the specified address..............:\n");  
 printf("\nAfter bind()\n");  
 printf("IP=%d \t Port no.=%d \n",inet\_ntoa(servaddr.sin\_addr),ntohs(servaddr.sin\_port));  
 return 0;  
}

OUTPUT :



**7. Program using raw-socket like packet capturing and filtering**

#Packet sniffer in python for Linux

#Sniffs only incoming TCP packet

import socket, sys

from struct import \*

#create an INET, STREAMing socket

try:

s = socket.socket(socket.AF\_INET, socket.SOCK\_RAW, socket.IPPROTO\_TCP)

except socket.error , msg:

print 'Socket could not be created. Error Code : ' + str(msg[0]) + ' Message ' +

msg[1]

sys.exit()

# receive a packet

while True:

packet = s.recvfrom(65565)

#packet string from tuple

packet = packet[0]

#take first 20 characters for the ip header

ip\_header = packet[0:20]

#now unpack them :)

iph = unpack('!BBHHHBBH4s4s' , ip\_header)

version\_ihl = iph[0]

version = version\_ihl >> 4

ihl = version\_ihl & 0xF

iph\_length = ihl \* 4

ttl = iph[5]

protocol = iph[6]

s\_addr = socket.inet\_ntoa(iph[8]);

d\_addr = socket.inet\_ntoa(iph[9]);

print 'Version : ' + str(version) + ' IP Header Length : ' + str(ihl) + ' TTL : ' +

str(ttl) + ' Protocol : ' + str(protocol) + ' Source Address : ' + str(s\_addr) + '

Destination Address : ' + str(d\_addr)

tcp\_header = packet[iph\_length:iph\_length+20]

#now unpack them :)

tcph = unpack('!HHLLBBHHH' , tcp\_header)

source\_port = tcph[0]

dest\_port = tcph[1]

sequence = tcph[2]

acknowledgement = tcph[3]

doff\_reserved = tcph[4]

tcph\_length = doff\_reserved >> 4

print 'Source Port : ' + str(source\_port) + ' Dest Port : ' + str(dest\_port) + '

Sequence Number : ' + str(sequence) + ' Acknowledgement : ' + str(acknowledgement) + '

TCP header length : ' + str(tcph\_length)

h\_size = iph\_length + tcph\_length \* 4

data\_size = len(packet) - h\_size

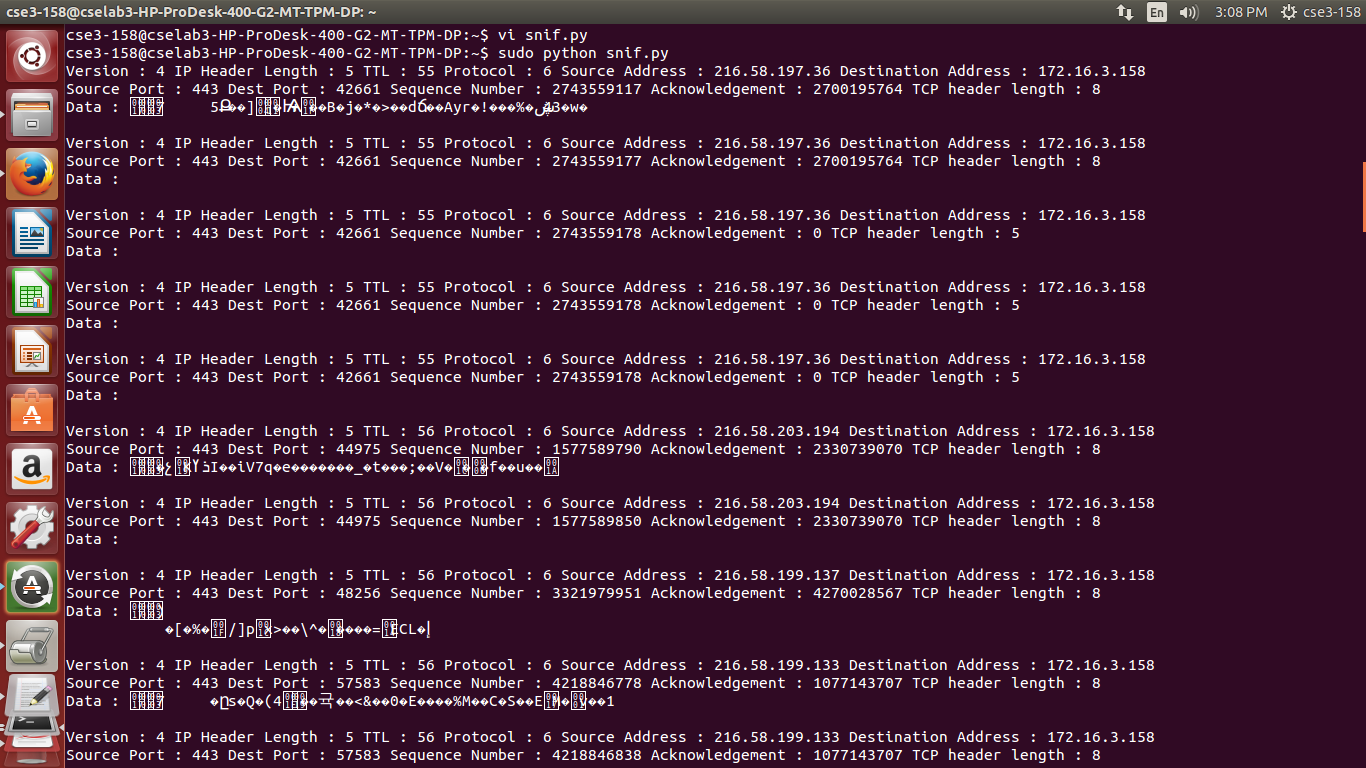
#get data from the packet

data = packet[h\_size:]

print 'Data : ' + data

print

**OUTPUT:**

****

**8. Implementation of RPC**

**date.x:**

program DATE\_PROG

{

version DATE\_VERS

{

long BIN\_DATE(void)=1;

string STR\_DATE(long)=2;

}=1;

}=0x31234567;

**date\_proc.c:**

//server

long \*bin\_date\_1\_svc(NULL)

{

static long timeval;

long time();

timeval = time((long \*) 0);

return &timeval;

}

char \*\*str\_date\_1\_svc(long \*lresult)

{

static char\* time;

char\* ctime();

time=ctime(lresult);

return &time;

}

**rdate.c:**

//client

#include<rpc/rpc.h>

#include"date.h"

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

{

CLIENT \*cl;

char \*\*sresult;

long \*lresult;

char \*servername;

servername=argv[1];

cl=clnt\_create(servername,DATE\_PROG,DATE\_VERS,"udp");

lresult=bin\_date\_1(NULL,cl);

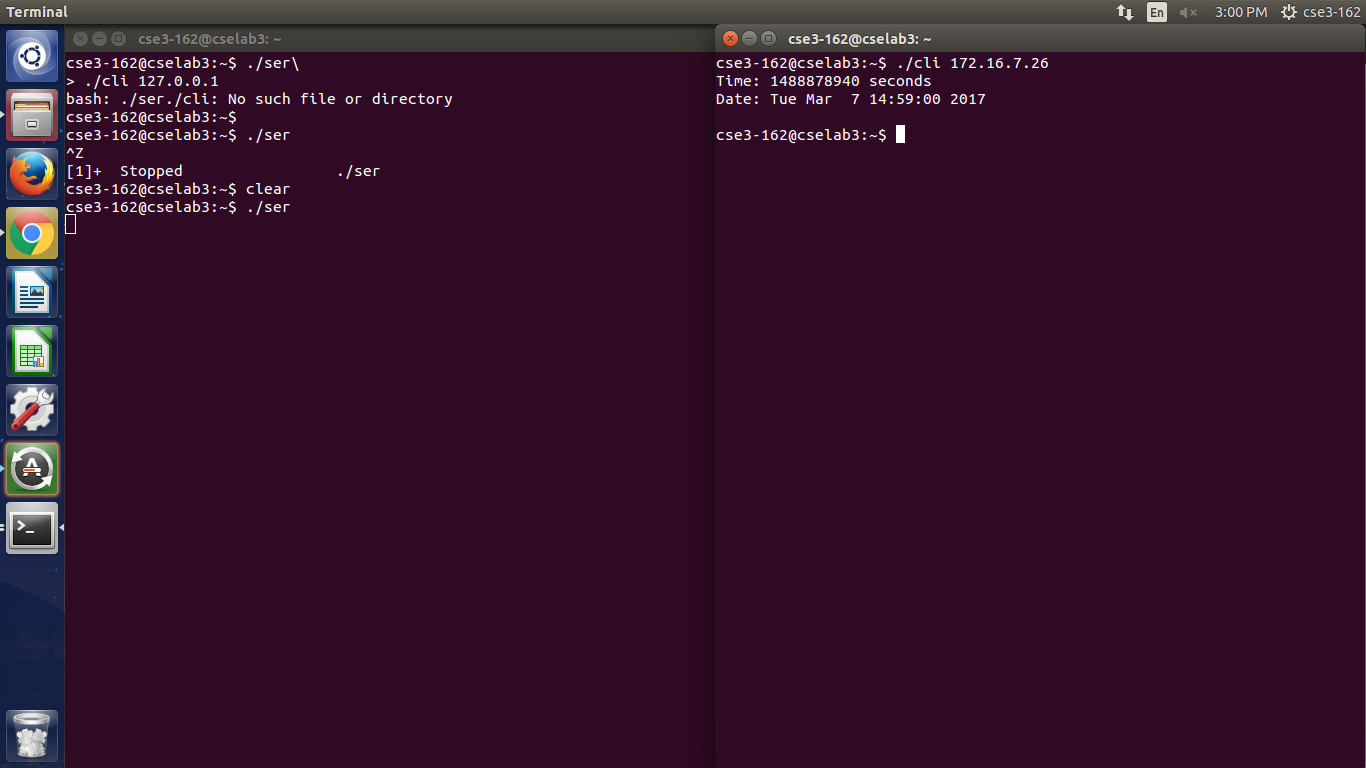
sresult=str\_date\_1(lresult,cl);

printf("Time: %ld seconds\n",\*lresult);

printf("Date: %s\n",\*sresult);

}

**OUTPUT:**



**9. Program to implement Slide window server.**

//Slide Window Server:

#include<string.h>

#include<stdlib.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

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

{

int sfd,lfd,len,choice,s,n;

char str[100],str1[100],err[100];

struct sockaddr\_in saddr,caddr;

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

if(sfd<0)

{

perror("FdError");

exit(-1);

}

bzero(&saddr,sizeof(saddr));

saddr.sin\_family=AF\_INET;

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

saddr.sin\_port=htons(atoi(argv[1]));

s=connect(sfd,(struct sockaddr\*)&saddr,sizeof(saddr));

if(s<0)

{

perror("connect error");

exit(-1);

}

for(;;)

{

n=recv(sfd,&str,100,0);

if(!strncmp(str,"exit",4))

{

printf("Exiting.............\n");

break;

}

str[n]='\0';

printf("\nReceived message is: %s\n Are there any errors?(1-Yes 0-No): ",str);

scanf("%d",&choice);

if(!choice)

write(sfd,"-1",sizeof("-1"));

else

{

printf("Enter the sequence no of the frame where error has occured: ");

scanf("%s",&err);

write(sfd,&err,sizeof(err));

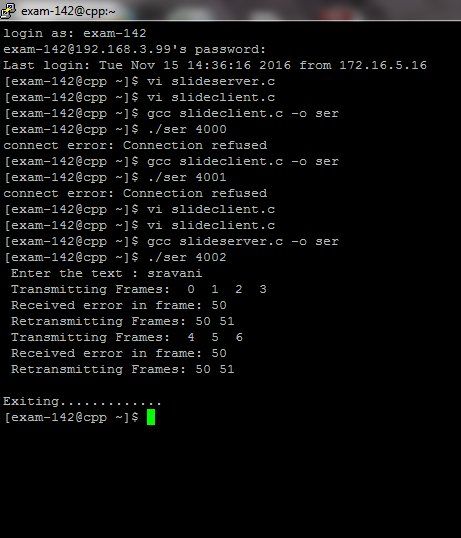
n=read(sfd,&str,20);

str[n]='\0';

printf("\n\nReceived the re-transmitted frames: %s\n\n",str);

}

}



//Slide window client

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

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

{

int sfd,lfd,len,choice,s,n;

char str[100],str1[100],err[100];

struct sockaddr\_in saddr,caddr;

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

if(sfd<0)

{

perror("FdError");

exit(-1);

}

bzero(&saddr,sizeof(saddr));

saddr.sin\_family=AF\_INET;

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

saddr.sin\_port=htons(atoi(argv[1]));

s=connect(sfd,(struct sockaddr\*)&saddr,sizeof(saddr));

if(s<0)

{

perror("connect error");

exit(-1);

}

for(;;)

{

n=recv(sfd,&str,100,0);

if(!strncmp(str,"exit",4))

{

printf("Exiting.............\n");

break;

}

str[n]='\0';

printf("\nReceived message is: %s\n Are there any errors?(1-Yes 0-No): ",str);

scanf("%d",&choice);

if(!choice)

write(sfd,"-1",sizeof("-1"));

else

{

printf("Enter the sequence no of the frame where error has occured: ");

scanf("%s",&err);

write(sfd,&err,sizeof(err));

n=read(sfd,&str,20);

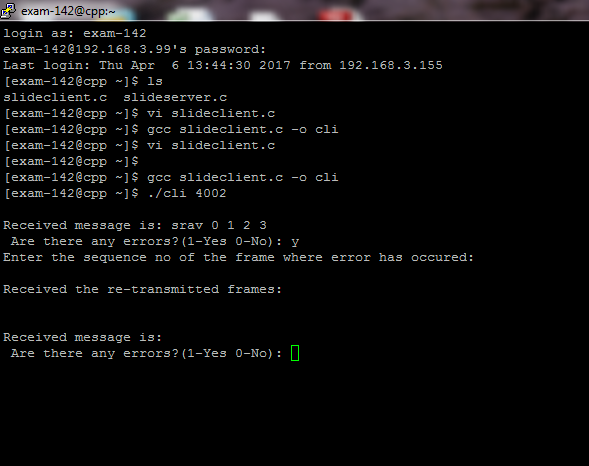
str[n]='\0';

printf("\n\nReceived the re-transmitted frames: %s\n\n",str);

}

}

}



**10. Program to implement Distance Vector Routing.**

#include<stdio.h>

struct node

{

unsigned dist[20];

unsigned from[20];

}rt[10];

int main()

{

int costmat[20][20];

int nodes,i,j,k,count=0;

printf("\nEnter the number of nodes : ");

scanf("%d",&nodes);

printf("\nEnter the cost matrix :\n");

for(i=0;i<nodes;i++)

{

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

{

scanf("%d",&costmat[i][j]);

costmat[i][i]=0;

rt[i].dist[j]=costmat[i][j];

rt[i].from[j]=j;

}

}

do

{

count=0;

for(i=0;i<nodes;i++)

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

for(k=0;k<nodes;k++)

if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])

{

rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];

rt[i].from[j]=k;

count++;

}

}while(count!=0);

for(i=0;i<nodes;i++)

{

printf("\n\n For router %d\n",i+1);

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

{

printf("\t\nnode %d via %d Distance %d

",j+1,rt[i].from[j]+1,rt[i].dist[j]);

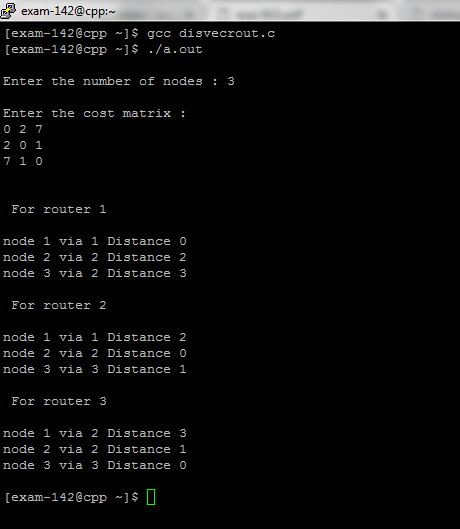
}

}

printf("\n\n");

// getch();

}



**11. Program to implement FTP.**

//ftp Server

#include <sys/socket.h>

#include <netinet/in.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

/\*for getting file size using stat()\*/

#include<sys/stat.h>

/\*for sendfile()\*/

#include<sys/sendfile.h>

/\*for O\_RDONLY\*/

#include<fcntl.h>

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

{

struct sockaddr\_in server, client;

struct stat obj;

int sock1, sock2;

char buf[100], command[5], filename[20];

int k, i, size, len, c;

int filehandle;

sock1 = socket(AF\_INET, SOCK\_STREAM, 0);

if(sock1 == -1)

{

printf("Socket creation failed");

exit(1);

}

server.sin\_port = atoi(argv[1]);

server.sin\_addr.s\_addr = 0;

k = bind(sock1,(struct sockaddr\*)&server,sizeof(server));

if(k == -1)

{

printf("Binding error");

exit(1);

}

k = listen(sock1,1);

if(k == -1)

{

printf("Listen failed");

exit(1);

}

len = sizeof(client);

sock2 = accept(sock1,(struct sockaddr\*)&client, &len);

i = 1;

while(1)

{

recv(sock2, buf, 100, 0);

sscanf(buf, "%s", command);

if(!strcmp(command, "ls"))

{

system("ls >temps.txt");

i = 0;

stat("temps.txt",&obj);

size = obj.st\_size;

send(sock2, &size, sizeof(int),0);

filehandle = open("temps.txt", O\_RDONLY);

sendfile(sock2,filehandle,NULL,size);

}

else if(!strcmp(command,"get"))

{

sscanf(buf, "%s%s", filename, filename);

stat(filename, &obj);

filehandle = open(filename, O\_RDONLY);

size = obj.st\_size;

if(filehandle == -1)

size = 0;

send(sock2, &size, sizeof(int), 0);

if(size)

sendfile(sock2, filehandle, NULL, size);

}

else if(!strcmp(command, "put"))

{

int c = 0, len;

char \*f;

sscanf(buf+strlen(command), "%s", filename);

recv(sock2, &size, sizeof(int), 0);

i = 1;

while(1)

{

filehandle = open(filename, O\_CREAT | O\_EXCL | O\_WRONLY, 0666);

if(filehandle == -1)

{

sprintf(filename + strlen(filename), "%d", i);

}

else

break;

}

f = malloc(size);

recv(sock2, f, size, 0);

c = write(filehandle, f, size);

close(filehandle);

send(sock2, &c, sizeof(int), 0);

}

else if(!strcmp(command, "pwd"))

{

system("pwd>temp.txt");

i = 0;

FILE\*f = fopen("temp.txt","r");

while(!feof(f))

buf[i++] = fgetc(f);

buf[i-1] = '\0';

fclose(f);

send(sock2, buf, 100, 0);

}

else if(!strcmp(command, "cd"))

{

if(chdir(buf+3) == 0)

c = 1;

else

c = 0;

send(sock2, &c, sizeof(int), 0);

}

else if(!strcmp(command, "bye") || !strcmp(command, "quit"))

{

printf("FTP server quitting..\n");

i = 1;

send(sock2, &i, sizeof(int), 0);

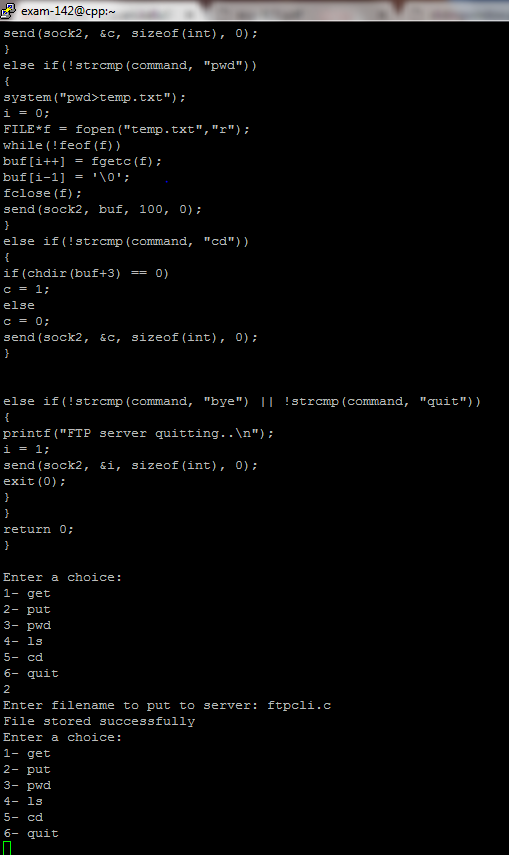
exit(0);

}

}

return 0;

}



//ftp Client

#include <sys/socket.h>

#include <netinet/in.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

/\*for getting file size using stat()\*/

#include<sys/stat.h>

/\*for sendfile()\*/

#include<sys/sendfile.h>

/\*for O\_RDONLY\*/

#include<fcntl.h>

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

{

struct sockaddr\_in server;

struct stat obj;

int sock;

int choice;

char buf[100], command[5], filename[20], \*f;

int k, size, status;

int filehandle;

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

if(sock == -1)

{

printf("socket creation failed");

exit(1);

}

server.sin\_family = AF\_INET;

server.sin\_port = atoi(argv[1]);

server.sin\_addr.s\_addr = 0;

k = connect(sock,(struct sockaddr\*)&server, sizeof(server));

if(k == -1)

{

printf("Connect Error");

exit(1);

}

int i = 1;

while(1)

{

printf("Enter a choice:\n1- get\n2- put\n3- pwd\n4- ls\n5- cd\n6- quit\n");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter filename to get: ");

scanf("%s", filename);

strcpy(buf, "get ");

strcat(buf, filename);

send(sock, buf, 100, 0);

recv(sock, &size, sizeof(int), 0);

if(!size)

{

printf("No such file on the remote directory\n\n");

break;

}

f = malloc(size);

recv(sock, f, size, 0);

while(1)

{

filehandle = open(filename, O\_CREAT | O\_EXCL | O\_WRONLY, 0666);

if(filehandle == -1)

{

sprintf(filename + strlen(filename), "%d", i);//needed only if same directory is used for both server and client

}

else break;

}

write(filehandle, f, size, 0);

close(filehandle);

strcpy(buf, "cat ");

strcat(buf, filename);

system(buf);

break;

case 2:

printf("Enter filename to put to server: ");

scanf("%s", filename);

filehandle = open(filename, O\_RDONLY);

if(filehandle == -1)

{

printf("No such file on the local directory\n\n");

break;

}

strcpy(buf, "put ");

strcat(buf, filename);

send(sock, buf, 100, 0);

stat(filename, &obj);

size = obj.st\_size;

send(sock, &size, sizeof(int), 0);

sendfile(sock, filehandle, NULL, size);

recv(sock, &status, sizeof(int), 0);

if(status)

printf("File stored successfully\n");

else

printf("File failed to be stored to remote machine\n");

break;

case 3:

strcpy(buf, "pwd");

send(sock, buf, 100, 0);

recv(sock, buf, 100, 0);

printf("The path of the remote directory is: %s\n", buf);

break;

case 4:

strcpy(buf, "ls");

send(sock, buf, 100, 0);

recv(sock, &size, sizeof(int), 0);

f = malloc(size);

recv(sock, f, size, 0);

filehandle = creat("temp.txt", O\_WRONLY);

write(filehandle, f, size, 0);

close(filehandle);

printf("The remote directory listing is as follows:\n");

system("cat temp.txt");

break;

case 5:

strcpy(buf, "cd ");

printf("Enter the path to change the remote directory: ");

scanf("%s", buf + 3);

send(sock, buf, 100, 0);

recv(sock, &status, sizeof(int), 0);

if(status)

printf("Remote directory successfully changed\n");

else

printf("Remote directory failed to change\n");

break;

case 6:

strcpy(buf, "quit");

send(sock, buf, 100, 0);

recv(sock, &status, 100, 0);

if(status)

{

printf("Server closed\nQuitting..\n");

exit(0);

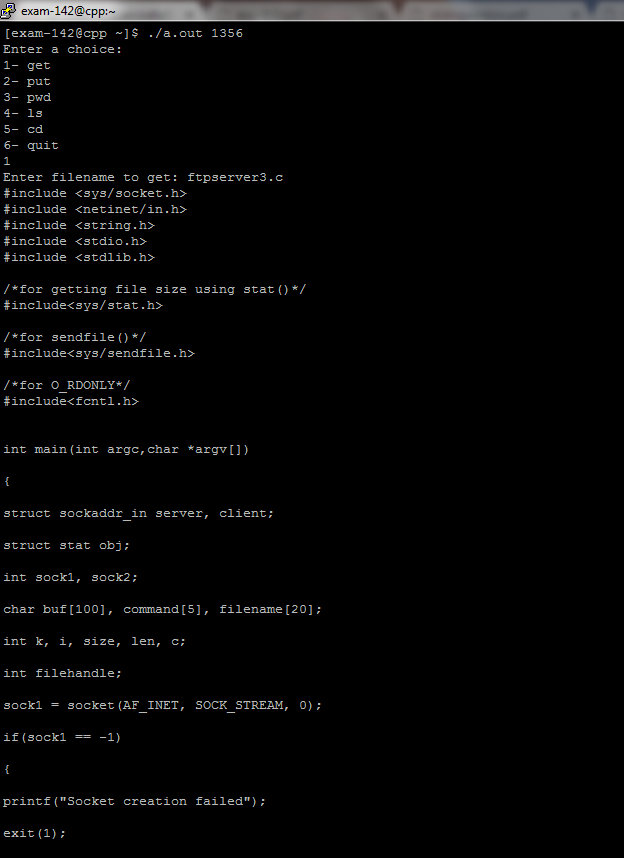
}

printf("Server failed to close connection\n");

}

}

}



**12. Program to demonstrate Ping.**

#include <fcntl.h>

#include <errno.h>

#include <sys/socket.h>

#include <resolv.h>

#include <netdb.h>

#include <netinet/in.h>

#include <netinet/ip\_icmp.h>

#define PACKETSIZE 64

struct packet

{

struct icmphdr hdr;

char msg[PACKETSIZE-sizeof(struct icmphdr)];

};

int pid=-1;

struct protoent \*proto=NULL;

/\*--------------------------------------------------------------------\*/

/\*--- checksum - standard 1s complement checksum ---\*/

/\*--------------------------------------------------------------------\*/

unsigned short checksum(void \*b, int len)

{ unsigned short \*buf = b;

unsigned int sum=0;

unsigned short result;

for ( sum = 0; len > 1; len -= 2 )

sum += \*buf++;

if ( len == 1 )

sum += \*(unsigned char\*)buf;

sum = (sum >> 16) + (sum & 0xFFFF);

sum += (sum >> 16);

result = ~sum;

return result;

}

/\*--------------------------------------------------------------------\*/

/\*--- display - present echo info ---\*/

/\*--------------------------------------------------------------------\*/

void display(void \*buf, int bytes)

{

int i;

struct iphdr \*ip = buf;

struct icmphdr \*icmp = buf+ip->ihl\*4;

printf("----------------\n");

for ( i = 0; i < bytes; i++ )

{

if ( !(i & 15) )

printf("\nX: ", i);

printf("1X ", ((unsigned char\*)buf)[i]);

}

printf("\n");

printf("IPv%d: hdr-size=%d pkt-size=%d protocol=%d TTL=%d src=%s ",

ip->version, ip->ihl\*4, ntohs(ip->tot\_len), ip->protocol,

ip->ttl, inet\_ntoa(ip->saddr));

printf("dst=%s\n", inet\_ntoa(ip->daddr));

if ( icmp->un.echo.id == pid )

{

printf("ICMP: type[%d/%d] checksum[%d] id[%d] seq[%d]\n", icmp->type, icmp->code, ntohs(icmp->checksum), icmp->un.echo.id, icmp->un.echo.sequence);

}

}

/\*--------------------------------------------------------------------\*/

/\*--- listener - separate process to listen for and collect messages--\*/

/\*--------------------------------------------------------------------\*/

void listener(void)

{

int sd;

struct sockaddr\_in addr;

unsigned char buf[1024];

sd = socket(PF\_INET, SOCK\_RAW, proto->p\_proto);

if ( sd < 0 )

{

perror("socket");

exit(0);

}

for (;;)

{ int bytes, len=sizeof(addr);

bzero(buf, sizeof(buf));

bytes = recvfrom(sd, buf, sizeof(buf), 0, (struct sockaddr\*)&addr, &len);

if ( bytes > 0 )

display(buf, bytes);

else

perror("recvfrom");

}

exit(0);

}

/\*--------------------------------------------------------------------\*/

/\*--- ping - Create message and send it. ---\*/

/\*--------------------------------------------------------------------\*/

void ping(struct sockaddr\_in \*addr)

{ const int val=255;

int i, sd, cnt=1;

struct packet pckt;

struct sockaddr\_in r\_addr;

sd = socket(PF\_INET, SOCK\_RAW, proto->p\_proto);

if ( sd < 0 )

{

perror("socket");

return;

}

if ( setsockopt(sd, SOL\_IP, IP\_TTL, &val, sizeof(val)) != 0)

perror("Set TTL option");

if ( fcntl(sd, F\_SETFL, O\_NONBLOCK) != 0 )

perror("Request nonblocking I/O");

for (;;)

{ int len=sizeof(r\_addr);

printf("Msg #%d\n", cnt);

if ( recvfrom(sd, &pckt, sizeof(pckt), 0, (struct sockaddr\*)&r\_addr, &len) > 0 )

printf("\*\*\*Got message!\*\*\*\n");

bzero(&pckt, sizeof(pckt));

pckt.hdr.type = ICMP\_ECHO;

pckt.hdr.un.echo.id = pid;

for ( i = 0; i < sizeof(pckt.msg)-1; i++ )

pckt.msg[i] = i+'0';

pckt.msg[i] = 0;

pckt.hdr.un.echo.sequence = cnt++;

pckt.hdr.checksum = checksum(&pckt, sizeof(pckt));

if ( sendto(sd, &pckt, sizeof(pckt), 0, (struct

sockaddr\*)addr, sizeof(\*addr)) <= 0 )

perror("sendto");

sleep(1);

}

}

/\*--------------------------------------------------------------------\*/

/\*--- main - look up host and start ping processes. ---\*/

/\*--------------------------------------------------------------------\*/

int main(int count, char \*strings[])

{ struct hostent \*hname;

struct sockaddr\_in addr;

if ( count != 2 )

{

printf("usage: %s <addr>\n", strings[0]);

exit(0);

}

if ( count > 1 )

{

pid = getpid();

proto = getprotobyname("ICMP");

hname = gethostbyname(strings[1]);

bzero(&addr, sizeof(addr));

addr.sin\_family = hname->h\_addrtype;

addr.sin\_port = 0;

addr.sin\_addr.s\_addr = \*(long\*)hname->h\_addr;

if ( fork() == 0 )

listener();

else

ping(&addr);

wait(0);}

else

printf("usage: myping <hostname>\n");

return 0;

}

