1. **Implement Data link Layer Framing Methods such as Bit Stuffing &Character Stuffing.**

**i)Program to implement Bit Stuffing**

**PROGRAM:**

#include<stdio.h>

#include<string.h>

#include<conio.h>

#include<mem.h>

void main()

{

char s[50],flg[8];

int i,j,len=0,c=0;

clrscr();

memset(s,0,sizeof(s));

printf("enter the flag\n");

gets(flg);

printf("enter the bit stream\n");

gets(s);

len=strlen(s);

//printf("the length is %d",len);

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

{

if(s[i]=='1')

{

c++;

if(c==5)

{

for(j=len;j>i+1;j--)

s[j]=s[j-1];

s[i+1]='0';

c=0;

len++;

}

}

else

c=0;

}

printf("\n the transmitted frame is %s %s %s",flg,s,flg);

/\*``````````````bit destuffing```````````````````````````\*/

getch();

}

**OUTPUT**

**Enter the flag**

**01111110**

**Enter the bit stream**

**01111110**

**The transmitted frame is 01111110 011111010 01111110**

**ii) Program to implement Character Stuffing.**

**PROGRAM*:***

#include<string.h>

#include<conio.h>

main()

{

char s[30],s2[30];

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

clrscr();

memset(s,0,sizeof(s));

memset(s2,0,sizeof(s2));

puts("Enter String");

gets(s);

while(s[i]!='\0')

{

if((s[i]=='d')&&(s[i+1]=='l')&&(s[i+2]=='e'))

{

j=i+3;

k=0;

while(s[j]!='\0')

{

s2[k]=s[j];

k++;

j++;

}

s[i+3]='d';

s[i+4]='l';

s[i+5]='e';

s[i+6]='\0';

strcat(s,s2);

memset(s2,0,sizeof(s2));

i=i+6;

}

else

i++;

}

printf("The Frame is dle stx %s dle etx",s);

getch();

}

**OUTPUT**

**Enter String**

**dle vvit dle**

**The Frame is dle stx dledle vvit dledle dle etx**

**2)Implement on dataset of characters the three CRC polynomials**

**i) Program to generate CRC**

**PROGRAM**

main()

{

int pl,fl,i,j,k,n;

char p[100],f[100],c[100];

clrscr();

memset(p,0,sizeof(p));

memset(f,0,sizeof(f));

memset(c,0,sizeof(c));

printf("Enter the polynomial\n");

gets(p);

printf("Enter the Frame\n");

gets(f);

pl=strlen(p);

fl=strlen(f);

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

c[i]=f[i];

n=pl+fl-1;

for(i=fl;i<n;i++)

f[i]='0';

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

if(f[i]=='1')

{

for(j=i,k=0;k<pl;k++,j++)

if(f[j]==p[k])

f[j]='0';

else

f[j]='1';

}

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

f[i]=c[i];

printf("\n The DATA TO BE SENT IS %s",f);

getch();

}

**OUTPUT:**

**Enter the Polynomial 1011**

**Enter the Frame: 10011101**

**The DATA TO BE SENT IS 10011101011**

**ii) Write a Program to implement CRC Verifier.**

***PROGRAM:***

#include<stdio.h>

main()

{

int i,j,k,pl,fl,n,cl;

char a[10],b[100],c[100],count=0;

clrscr();

memset(a,0,sizeof(a));

memset(b,0,sizeof(b));

memset(c,0,sizeof(c));

printf("Enter polynomial\n");

scanf("%s",a);

printf("Enter the CRC appended frame\n");

scanf("%s",b);

pl=strlen(a);

fl=strlen(b);

/\* to copy original frame to c[]\*/

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

c[i]=b[i];

n=fl-pl+1;

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

if(c[i]=='1')

{

for(j=i,k=0;k<pl;k++,j++)

if(a[k]==c[j])

c[j]='0';

else

c[j]='1';

}

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

c[i]=b[i];

for(i=n;i<fl;i++)

{

if(c[i]=='1')

count++;

}

if(count==0)

printf("The message is correct\n");

else

printf("The message is not correct\n");

getch();

}

***OUTPUT:***

***Enter Polynomial 101***

***Enter the CRC appended frame***

***10010011***

***The message is correct***

***3. Write a program to implement Dijkstra’s Shortest Path Routing Algorithm***

***PROGRAM:***

/\* SHORTEST PATH \*/

#include<stdio.h>

#include<string.h>

#include<math.h>

void main()

{

int

u,v,num,i,j,l,k,s[10],min,cost[10][10],dist[10],path[10],n;

clrscr();

printf("\n ENTER VERTICES:");

scanf("%d",&n);

printf("\n ENTER ADJECENCY MATRIX:\n");

for(i=1;i<=n;i++)

{

printf("\n Enter Matrix for node %d",i);

for(j=1;j<=n;j++)

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

}

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

if(i==j)

cost[i][j]=0;

else

if(cost[i][j]==-1)

cost[i][j]=30000;

printf("\nENTER SOURCE VERTEX:");

scanf("%d",&v);

clrscr();

for(i=1;i<=n;i++)

{

s[i]=0;

path[i]=v;

dist[i]=cost[v][i];

}

dist[v]=0;

for(num=2;num<=n;num++)

{

min=30000;

u=0;

for(i=1;i<=n;i++)

{

if(s[i]!=1)

if(min>dist[i])

{

u=i;

min=dist[i];

}

}

s[u]=1;

for(i=1;i<=n;i++)

{

if(s[i]!=1)

if(dist[i]>(min+cost[u][i]))

{

dist[i]=min+cost[u][i];

path[i]=u;

}

}

}

printf("\n");

printf("\nPATH MATRIX:\n");

printf("\nDISTANCE NODE PATH\n");

for(i=1;i<=n;i++)

{

printf("\n %d",dist[i]);

printf(" %d ",i);

j=i;

do

{

printf(" --> %d ",path[j]);

u=path[j];

j=u;

}

while(u!=v);

}

getch();

}

***OUTPUT:***

***ENTER VERTICES 6***

***ENTER ADJACENCY MATRIX:***

***ENTER MATRIX FOR NODE 1***

***0 2 3 -1 -1 -1***

***ENTER MATRIX FOR NODE 2***

***-1 0 -1 1 1 -1 -1***

***ENTER MATRIX FOR NODE 3***

***-1 -1 0 -1 2 -1***

***ENTER MATRIX FOR NODE 4***

***-1 -1 -1 0 -1 1***

***ENTER MATRIX FOR NODE 5***

***-1 -1 -1 -1 0 1***

***ENTER MATRIX FOR NODE 6***

***-1 -1 -1 -1 -1 0***

***ENTER SOURCE VERTEX:1***

***PATH MATRIX***

***DISTANCE NODE PATH***

***0 1 - -> 1***

***2 2 - -> 1***

***3 3 - -> 1***

***3 4 - -> 2 - -> 1***

***5 5 - -> 3 - -> 1***

***4 5 - -> 4 - -> 2 - -> 1***

***1***

***2 1***

***3 2 1***

***4. Write a program to implement Distance Vector Routing Algorithm.***

***PROGRAM***

#include<stdio.h>  
#include<math.h>  
#include<conio.h>  
main()  
{  
int i,j,k,nv,sn,noadj,edel[20],tdel[20][20],min;  
char sv,adver[20],ch;  
clrscr();  
printf("\n ENTER THE NO.OF VERTECES:");  
scanf("%d",&nv);  
printf("\n ENTER THE SOURCE VERTEX NUM,BER AND NAME:");  
scanf("%d",&sn);  
flushall();  
sv=getchar();  
printf("\n NETER NO.OF ADJ VERTECES TO VERTEX %c",sv);  
scanf("%d",&noadj);  
for(i=0;i<noadj;i++)  
{  
printf("\n ENTER TIME DELAY and NODE NAME:");  
scanf("%d %c",&edel[i],&adver[i]);  
}  
for(i=0;i<noadj;i++)  
{  
printf("\n ENTER THE TIME DELAY FROM %c to ALL OTHER  
NODES: ",adver[i]);  
for(j=0;j<nv;j++)  
scanf("%d",&tdel[i][j]);  
}  
printf("\n DELAY VIA--VERTEX \n ");  
for(i=0;i<nv;i++)  
{  
min=1000;  
ch=0;  
for(j=0;j<noadj;j++)  
if(min>(tdel[j][i]+edel[j]))  
{  
min=tdel[j][i]+edel[j];  
ch=adver[j];  
}  
if(i!=sn-1)  
printf("\n%d %c",min,ch);  
else  
printf("\n0 -");  
}  
getch();

}

***INPUT/OUTPUT:***  
***ENTER THE NO.OF VERTECES:12  
ENTER THE SOURCE VERTEX NUMBER AND NAME:10 J  
ENTER NO.OF ADJ VERTECES TO VERTEX 4  
ENTER TIME DELAY and NODE NAME:8 A  
ENTER TIME DELAY and NODE NAME:10 I  
ENTER TIME DELAY and NODE NAME:12 H  
ENTER TIME DELAY and NODE NAME:6 K  
ENTER THE TIME DELAY FROM A to ALL OTHER NODES:  
0 12 25 40 14 23 18 17 21 9 24 29  
ENTER THE TIME DELAY FROM I to ALL OTHER NODES:  
24 36 18 27 7 20 31 20 0 11 22 33  
ENTER THE TIME DELAY FROM H to ALL OTHER NODES:  
20 31 19 8 30 19 6 0 14 7 22 9  
ENTER THE TIME DELAY FROM K to ALL OTHER NODES:  
21 28 36 24 22 40 31 19 22 10 0 9  
DELAY VIA--VERTEX  
8 a  
20 a  
28 i  
20 h  
17 i  
30 i  
18 h  
12 h  
10 i  
0 -  
6 k  
15 k***

***5. Understanding and using of commands***

## **Purpose**

Sends an echo request to a network host.

## **Syntax**

**ping**[ [**-d**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81067)] [[ **-D**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__ping-d) ] [  [**-n**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81160)] [  [**-q**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81178)] [ [**-r**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81184)] [ [**-v**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81250)] [  [**-R**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81196)] [ [**-a**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__ping-a) addr\_family ] [  [**-c**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81047)Count ] [ [**-w**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81058) timeout ] [  [**-f**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81079)|  [**-i**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81126)Wait ] [  [**-l**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81146)Preload ] [  [**-p**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81167)Pattern ] [  [**-s**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81219)PacketSize ] [ [**-S**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81227) hostname/IP addr ] [ [**-L**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81140) ] [ [**- I**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81101) a.b.c.d. ] [ [**-o**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81115) interface ] [ [**-T**](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81239) ttl ] Host [  [PacketSize](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81268)] [  [Count](https://www.ibm.com/docs/en/aix/7.1?topic=p-ping-command#ping__row-d3e81277)]

## **Description**

The **/usr/sbin/ping** command sends an Internet Control Message Protocol (ICMP) ECHO\_REQUEST to obtain an ICMP ECHO\_RESPONSE from a host or gateway. The **ping** command is useful for:

* Determining the status of the network and various foreign hosts.
* Tracking and isolating hardware and software problems.
* Testing, measuring, and managing networks.

If the host is operational and on the network, it responds to the echo. Each echo request contains an Internet Protocol (IP) and ICMP header, followed by a ping PID and a **timeval** structure, and enough bytes to fill out the packet. The default is to continuously send echo requests until an Interrupt is received (Ctrl-C).

The **ping** command sends one datagram per second and prints one line of output for every response received. The **ping** command calculates round-trip times and packet loss statistics, and displays a brief summary on completion. The **ping** command completes when the program times out or on receipt of a **SIGINT** signal. The Host parameter is either a valid host name or Internet address.

By default, the **ping** command will continue to send echo requests to the display until an Interrupt is received (Ctrl-C). The Interrupt key can be changed by using the **stty** command.

Because of the load that continuous echo requests can place on the system, repeated requests should be used primarily for problem isolation.

## **Flags**

| Item | Description |
| --- | --- |
| -c Count | Specifies the number of echo requests, as indicated by the Count variable, to be sent (and received). |
| -w timeout | This option works only with the -c option. It causes ping to wait for a maximum of 'timeout' seconds for a reply (after sending the last packet). |
| -d | Starts socket-level debugging. |
| -D | This option causes a hex dump to standard output of ICMP ECHO\_REPLY packets. |

***6. Implementation of TCP***

***1. CLIENT SIDE***

***#include <netdb.h>***

***#include <stdio.h>***

***#include <stdlib.h>***

***#include <string.h>***

***#include <sys/socket.h>***

***#define MAX 80***

***#define PORT 8080***

***#define SA struct sockaddr***

***void func(int sockfd)***

***{***

***char buff[MAX]***

***int n;***

***for (;;) {***

***bzero(buff,sizeof(buff));***

***printf("Enter the string : ");***

***n = 0;***

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

***;***

***write(sockfd, buff, sizeof(buff));***

***bzero(buff, sizeof(buff));***

***read(sockfd, buff, sizeof(buff));***

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

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

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

***break;***

***}***

***}***

***}***

***int main()***

***{***

***int sockfd, connfd;***

***struct sockaddr\_in servaddr, cli;***

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

***if (sockfd == -1) {***

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

***exit(0);***

***}***

***else***

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

***bzero(&servaddr, sizeof(servaddr));***

***servaddr.sin\_family = AF\_INET;***

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

***servaddr.sin\_port = htons(PORT);***

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

***printf("connection with the server failed...\n");***

***exit(0);***

***}***

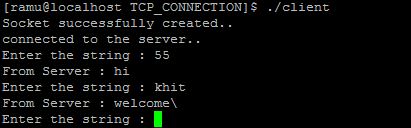
***else***

***printf("connected to the server..\n");***

***func(sockfd);***

***close(sockfd);***

***}***



***ii. SERVER SIDE***

***#include <netdb.h>***

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

***#include<stdlib.h>***

***#include <string.h>***

***#include <sys/socket.h>***

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

***#define MAX 80***

***#define PORT 8080***

***#define SA struct sockaddr***

***#include<stdio.h>***

***void func(int sockfd)***

***{***

***char buff[MAX];***

***int n;***

***for (;;) {***

***bzero(buff, MAX);***

***read(sockfd, buff, sizeof(buff));***

***printf("From client: %s\t To client : ", buff);***

***bzero(buff, MAX);***

***n = 0;***

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

***;***

***write(sockfd, buff, sizeof(buff));***

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

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

***break;***

***}***

***}***

***}***

***int main()***

***{***

***int sockfd, connfd, len;***

***struct sockaddr\_in servaddr, cli;***

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

***if (sockfd == -1) {***

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

***exit(0);***

***}***

***else***

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

***bzero(&servaddr, sizeof(servaddr));***

***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");***

***if ((listen(sockfd, 5)) != 0) {***

***printf("Listen failed...\n");***

***exit(0);***

***}***

***else***

***printf("Server listening..\n");***

***len = sizeof(cli);***

***connfd = accept(sockfd, (SA\*)&cli, &len);***

***if (connfd < 0) {***

***printf("server acccept failed...\n");***

***exit(0);***

***}***

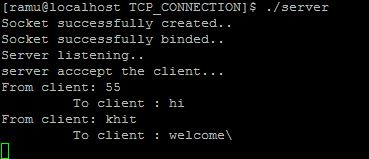
***else***

***printf("server acccept the client...\n");***

***func(connfd);***

***close(sockfd);***

***}***



***7. Implementation of UDP***

***i. CLIENT SIDE***

***#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>***

***#include <arpa/inet.h>***

***int main()***

***{***

***int CreateSocket = 0,n = 0;***

***char dataReceived[1024];***

***struct sockaddr\_in ipOfServer;***

***memset(dataReceived, '0' ,sizeof(dataReceived));***

***if((CreateSocket = socket(AF\_INET, SOCK\_STREAM, 0))< 0)***

***{***

***printf("Socket not created \n");***

***return 1;***

***}***

***ipOfServer.sin\_family = AF\_INET;***

***ipOfServer.sin\_port = htons(2017);***

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

***if(connect(CreateSocket, (struct sockaddr \*)&ipOfServer, sizeof(ipOfServer))<0)***

***{***

***printf("Connection failed due to port and ip problems\n");***

***return 1;***

***}***

***while((n = read(CreateSocket, dataReceived, sizeof(dataReceived)-1)) > 0)***

***{***

***dataReceived[n] = 0;***

***if(fputs(dataReceived, stdout) == EOF)***

***{***

***printf("\nStandard output error");***

***}***

***printf("\n");***

***}***

***if( n < 0)***

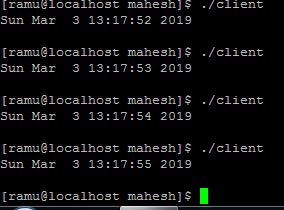
***{***

***printf("Standard input error \n");***

***}***

***return 0;***

***}***



***ii. SERVER SIDE***

***#include <stdio.h> // standard input and output library***

***#include <stdlib.h> // this includes functions regarding memory allocation***

***#include <string.h> // contains string functions***

***#include <errno.h> //It defines macros for reporting and retrieving error conditions through error codes***

***#include <time.h> //contains various functions for manipulating date and time***

***#include <unistd.h> //contains various constants***

***#include <sys/types.h> //contains a number of basic derived types that should be used whenever appropriate***

***#include <arpa/inet.h> // defines in\_addr structure***

***#include <sys/socket.h> // for socket creation***

***#include <netinet/in.h> //contains constants and structures needed for internet domain addresses***

***int main()***

***{***

***time\_t clock;***

***char dataSending[1025]; // Actually this is called packet in Network Communication, which contain data and send through.***

***int clintListn = 0, clintConnt = 0;***

***struct sockaddr\_in ipOfServer;***

***clintListn = socket(AF\_INET, SOCK\_STREAM, 0); // creating socket***

***memset(&ipOfServer, '0', sizeof(ipOfServer));***

***memset(dataSending, '0', sizeof(dataSending));***

***ipOfServer.sin\_family = AF\_INET;***

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

***ipOfServer.sin\_port = htons(2017); // this is the port number of running server***

***bind(clintListn, (struct sockaddr\*)&ipOfServer , sizeof(ipOfServer));***

***listen(clintListn , 20);***

***while(1)***

***{***

***printf("\n\nHi,Iam running server.Some Client hit me\n"); // whenever a request from client came. It will be processed here.***

***clintConnt = accept(clintListn, (struct sockaddr\*)NULL, NULL);***

***clock = time(NULL);***

***snprintf(dataSending, sizeof(dataSending), "%.24s\r\n", ctime(&clock)); // Printing successful message***

***write(clintConnt, dataSending, strlen(dataSending));***

***close(clintConnt);***

***sleep(1);***

***}***

***return 0;***

***}***



**8.** Implementation of Select system call.

#include <stdio.h>

#include <string.h> //strlen

#include <stdlib.h>

#include <errno.h>

#include <unistd.h> //close

#include <arpa/inet.h> //close

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <sys/time.h> //FD\_SET, FD\_ISSET, FD\_ZERO macros

#define TRUE 1

#define FALSE 0

#define PORT 8888

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

{

int opt = TRUE;

int master\_socket , addrlen , new\_socket , client\_socket[30] ,

max\_clients = 30 , activity, i , valread , sd;

int max\_sd;

struct sockaddr\_in address;

char buffer[1025]; //data buffer of 1K

//set of socket descriptors

fd\_set readfds;

//a message

char \*message = "ECHO Daemon v1.0 \r\n";

//initialise all client\_socket[] to 0 so not checked

for (i = 0; i < max\_clients; i++)

{

client\_socket[i] = 0;

}

//create a master socket

if( (master\_socket = socket(AF\_INET , SOCK\_STREAM , 0)) == 0)

{

perror("socket failed");

exit(EXIT\_FAILURE);

}

//set master socket to allow multiple connections ,

//this is just a good habit, it will work without this

if( setsockopt(master\_socket, SOL\_SOCKET, SO\_REUSEADDR, (char \*)&opt,

sizeof(opt)) < 0 )

{

perror("setsockopt");

exit(EXIT\_FAILURE);

}

//type of socket created

address.sin\_family = AF\_INET;

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

address.sin\_port = htons( PORT );

//bind the socket to localhost port 8888

if (bind(master\_socket, (struct sockaddr \*)&address, sizeof(address))<0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

printf("Listener on port %d \n", PORT);

//try to specify maximum of 3 pending connections for the master socket

if (listen(master\_socket, 3) < 0)

{

perror("listen");

exit(EXIT\_FAILURE);

}

//accept the incoming connection

addrlen = sizeof(address);

puts("Waiting for connections ...");

while(TRUE)

{

//clear the socket set

FD\_ZERO(&readfds);

//add master socket to set

FD\_SET(master\_socket, &readfds);

max\_sd = master\_socket;

//add child sockets to set

for ( i = 0 ; i < max\_clients ; i++)

{

//socket descriptor

sd = client\_socket[i];

//if valid socket descriptor then add to read list

if(sd > 0)

FD\_SET( sd , &readfds);

//highest file descriptor number, need it for the select function

if(sd > max\_sd)

max\_sd = sd;

}

//wait for an activity on one of the sockets , timeout is NULL ,

//so wait indefinitely

activity = select( max\_sd + 1 , &readfds , NULL , NULL , NULL);

if ((activity < 0) && (errno!=EINTR))

{

printf("select error");

}

//If something happened on the master socket ,

//then its an incoming connection

if (FD\_ISSET(master\_socket, &readfds))

{

if ((new\_socket = accept(master\_socket,

(struct sockaddr \*)&address, (socklen\_t\*)&addrlen))<0)

{

perror("accept");

exit(EXIT\_FAILURE);

}

//inform user of socket number - used in send and receive commands

printf("New connection , socket fd is %d , ip is : %s , port : %d

\n" , new\_socket , inet\_ntoa(address.sin\_addr) , ntohs

(address.sin\_port));

//send new connection greeting message

if( send(new\_socket, message, strlen(message), 0) != strlen(message) )

{

perror("send");

}

puts("Welcome message sent successfully");

//add new socket to array of sockets

for (i = 0; i < max\_clients; i++)

{

//if position is empty

if( client\_socket[i] == 0 )

{

client\_socket[i] = new\_socket;

printf("Adding to list of sockets as %d\n" , i);

break;

}

}

}

//else its some IO operation on some other socket

for (i = 0; i < max\_clients; i++)

{

sd = client\_socket[i];

if (FD\_ISSET( sd , &readfds))

{

//Check if it was for closing , and also read the

//incoming message

if ((valread = read( sd , buffer, 1024)) == 0)

{

//Somebody disconnected , get his details and print

getpeername(sd , (struct sockaddr\*)&address , \

(socklen\_t\*)&addrlen);

printf("Host disconnected , ip %s , port %d \n" ,

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

//Close the socket and mark as 0 in list for reuse

close( sd );

client\_socket[i] = 0;

}

//Echo back the message that came in

else

{

//set the string terminating NULL byte on the end

//of the data read

buffer[valread] = '\0';

send(sd , buffer , strlen(buffer) , 0 );

}

}

}

}

return 0;

}

# 9. Implementation of gesockopt(), setsockopt() system calls

# #include<stdio.h>

# #include<stdlib.h>

# #include<string.h>

# #include<unistd.h>

# #include<netinet/in.h>

# #include<arpa/inet.h>

# #include<sys/types.h>

# #include<sys/socket.h>

# #include<errno.h>

# int main()

# {

# int sockfd,maxseg,sendbuff,optlen;

# if((sockfd=socket(AF\_INET,SOCK\_STREAM,0))<0)

# {

# perror(“socket”);

# exit(0);

# }

# optlen=sizeof(maxseg);

# if(getsockopt(sockfd,IPPROTO\_TCP,TCP\_MAXSEG,(char \*)&maxseg,&optlen)<0)

# {

# perror(“get sockopt1”);

# exit(0);

# }

# printf(“\n TCP maxseg=%d”,maxseg);

# sendbuff=12324;

# if(setsockopt(sockfd,SOL\_SOCKET,SO\_SNDBUF,(char \*)&sendbuff,sizeof(sendbuff))<0)

# {

# perror(“set socketopt”);

# exit(0);

# }

# optlen=sizeof(sendbuff);

# if(getsockopt(sockfd,SOL\_SOCKET,SO\_SNDBUF,(char \*)&sendbuff,&optlen)<0)

# {

# perror(“getsockopt2”);

# exit(0);

# }

# printf(“\n send buffer size=%d\n”,sendbuff);

# }

# 10. Implementation of getpeername() system call

# #include<stdio.h>

# #include<stdlib.h>

# #include<string.h>

# #include<unistd.h>

# #include<netinet/in.h>

# #include<arpa/inet.h>

# #include<sys/types.h>

# #include<sys/socket.h>

# #include<errno.h>

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

# {

# int s,s2,t,len;

# struct sockaddr\_in local,rem;

# char str[100];

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

# if(s==-1)

# {

# perror(“socket”);

# exit(1);

# }

# bzero((char \*)&local,sizeof(local));

# local.sin\_family=AF\_INET;

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

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

# if(bind(s,(struct sockaddr \*)&local,sizeof(local))==-1)

# {

# perror(“bind”);

# exit(1);

# }

# if(listen(s,5)==-1)

# {

# perror(“listen”);

# exit(1);

# }

# for(;;)

# {

# int done,n;

# printf(“waiting for a connection……\n”);

# t=sizeof(rem);

# s2=accept(s,(struct sockaddr \*)&rem,&t);

# if(s2==-1)

# {

# perror(“accept”);

# exit(1);

# }

# }

# close(s2);

# return 0;

# }

# /\*GET PEER NAME CLIENT\*/

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

# {

# int s;

# struct sockaddr\_in server,addr;

# socklen\_t len;

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

# server.sin\_family=AF\_INET;

# inet\_aton(“172.16.0.1”,&server.sin\_addr);

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

# if(connect(s,(struct sockaddr \*)&server,sizeof(server))<0)

# {

# perror(“connect”);

# exit(0);

# }

# len=sizeof(addr);

# getpeername(s,(struct sockaddr \*)&addr,&len);

# printf(“peer IP address:%s\n”,inet\_ntoa(addr.sin\_addr));

# printf(“peer port:%d\n”,ntohs(addr.sin\_port));

# return 0;

# }

**11.** Implementation of remote command execution using socket system calls.

# Server program

# #include<stdio.h>

# #include<stdlib.h>

# #include<string.h>

# #include<unistd.h>

# #include<netinet/in.h>

# #include<arpa/inet.h>

# #include<sys/types.h>

# #include<sys/socket.h>

# #include<errno.h>

# int main()

# {

# int sd,acpt,len,bytes,port;

# char send[50],receiv[50];

# struct sockaddr\_in serv,cli;

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

# {

# printf("Error in socket\n");

# exit(0);

# }

# bzero(&serv,sizeof(serv));

# printf("Enter the port number : ");

# scanf("%d",&port);

# serv.sin\_family=AF\_INET;

# serv.sin\_port=htons(port);

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

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

# {

# printf("Error in bind\n");

# exit(0);

# }

# if(listen(sd,3)<0)

# {

# printf("Error in listen\n");

# exit(0);

# }

# if((acpt=accept(sd,(struct sockaddr\*)NULL,NULL))<0)

# {

# printf("\n\t Error in accept");

# exit(0);

# }

# while(1)

# {

# bytes=recv(acpt,receiv,50,0);

# receiv[bytes]='\0';

# if(strcmp(receiv ,"end")==0)

# {

# close(acpt);

# close(sd);

# exit(0);

# }

# else

# {

# printf("Command received : %s",receiv);

# system(receiv);

# printf("\n");

# }

# }

# }

# /\*Client program \*/

# #include<stdio.h>

# #include<stdlib.h>

# #include<string.h>

# #include<unistd.h>

# #include<netinet/in.h>

# #include<arpa/inet.h>

# #include<sys/types.h>

# #include<sys/socket.h>

# #include<errno.h>

# int main()

# {

# int sd,acpt,len,bytes,port;

# char send1[50],receiv[50];

# struct sockaddr\_in serv,cli;

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

# {

# printf("Error in socket\n");

# exit(0);

# }

# bzero(&serv,sizeof(serv));

# printf("Enter the port number : ");

# scanf("%d",&port);

# serv.sin\_family=AF\_INET;

# serv.sin\_port=htons(port);

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

# if(connect(sd,(struct sockaddr \*)&serv,sizeof(serv))<0)

# {

# printf("Error in connection\n");

# exit(0);

# }

# while(1)

# {

# printf("Enter the command:");

# gets(send1);

# if(strcmp(send1,"end")!=0)

# {

# send(sd,send1,50,0);

# }

# else

# {

# send(sd,send1,50,0);

# close(sd);

# break;

# }

# }

# }

**12.** Implementation of Distance Vector Routing Algorithm.

# #include<stdio.h>

# #include<stdlib.h>

# #include<string.h>

# #include<unistd.h>

# #include<netinet/in.h>

# #include<arpa/inet.h>

# #include<sys/types.h>

# #include<sys/socket.h>

# #include<errno.h>

#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);//Enter the 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];//initialise the distance equal to cost matrix

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

}

}

do

{

count=0;

for(i=0;i<nodes;i++)//We choose arbitary vertex k and we calculate the direct distance from the node i to k using the cost matrix

//and add the distance from k to node j

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

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

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

{//We calculate the minimum distance

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();

}

13.Implementation of SMTP.

# #include<stdio.h>

# #include<stdlib.h>

# #include<string.h>

# #include<unistd.h>

# #include<netinet/in.h>

# #include<arpa/inet.h>

# #include<sys/types.h>

# #include<sys/socket.h>

# #include<errno.h>

#define MAX 1024

int main()

{

int i,sockfd;

struct sockaddr\_in server;

char buf[MAX];

char tmp[1024];

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

if(sockfd<0)

{

perror("Error:");

exit(0);

}

server.sin\_family=AF\_INET;

server.sin\_port=htons(25);

server.sin\_addr.s\_addr=inet\_addr("192.168.10.55");

i=connect(sockfd,(struct sockaddr \*)&server,sizeof(server));

if(i<0)

{

perror("Error:");

close(sockfd);

exit(0);

}

memset(buf,0,MAX);

read(sockfd,buf,MAX);

printf("\n%s",buf);

sleep(1);

memset(buf,0,MAX);

strcpy(buf,"HELO localhost.localdomain\r\n");

write(sockfd,buf,strlen(buf));

memset(buf,0,MAX);

read(sockfd,buf,MAX);

printf("\n%s\nSender mail:",buf);

scanf("%s",tmp);

strcat(tmp,"\r\n");

sleep(1);

memset(buf,0,MAX);

strcpy(buf,"MAIL FROM:");

strcat(buf,tmp);

write(sockfd,buf,strlen(buf));

memset(buf,0,MAX);

read(sockfd,buf,MAX);

printf("\n%s\nReceipient mail:",buf);

scanf("%s",tmp);

strcat(tmp,"\r\n");

sleep(1);

memset(buf,0,MAX);

strcpy(buf,"RCPT TO:");

strcat(buf,tmp);

write(sockfd,buf,strlen(buf));

memset(buf,0,MAX);

read(sockfd,buf,MAX);

printf("\n%s",buf);

memset(buf,0,MAX);

strcpy(buf,"DATA\r\n");

write(sockfd,buf,strlen(buf));

memset(buf,0,MAX);

read(sockfd,buf,MAX);

printf("\n%s",buf);

memset(buf,0,MAX);

while(1)

{

memset(tmp,0,1024);

gets(tmp);

if(!strcmp(tmp,"."))

break;

strcat(buf,tmp);

}

strcat(buf,"\r\n.\r\n");

write(sockfd,buf,strlen(buf));

memset(buf,0,MAX);

read(sockfd,buf,MAX);

printf("\n%s",buf);

sleep(1);

close(sockfd);

return 0;

}

14.Implementation of 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");

}

}

}

15.Implementation of HTTP

|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include <stdlib.h> |
|  | #include <unistd.h> |
|  | #include <fcntl.h> |
|  |  |
|  | #include <netinet/tcp.h> |
|  | #include <sys/socket.h> |
|  | #include <sys/types.h> |
|  | #include <netinet/in.h> |
|  | #include <netdb.h> |
|  |  |
|  | int socket\_connect(char \*host, in\_port\_t port){ |
|  | struct hostent \*hp; |
|  | struct sockaddr\_in addr; |
|  | int on = 1, sock; |
|  |  |
|  | if((hp = gethostbyname(host)) == NULL){ |
|  | herror("gethostbyname"); |
|  | exit(1); |
|  | } |
|  | copy(hp->h\_addr, &addr.sin\_addr, hp->h\_length); |
|  | addr.sin\_port = htons(port); |
|  | addr.sin\_family = AF\_INET; |
|  | sock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP); |
|  | setsockopt(sock, IPPROTO\_TCP, TCP\_NODELAY, (const char \*)&on, sizeof(int)); |
|  |  |
|  | if(sock == -1){ |
|  | perror("setsockopt"); |
|  | exit(1); |
|  | } |
|  |  |
|  | if(connect(sock, (struct sockaddr \*)&addr, sizeof(struct sockaddr\_in)) == -1){ |
|  | perror("connect"); |
|  | exit(1); |
|  |  |
|  | } |
|  | return sock; |
|  | } |
|  |  |
|  | #define BUFFER\_SIZE 1024 |
|  |  |
|  | int main(int argc, char \*argv[]){ |
|  | int fd; |
|  | char buffer[BUFFER\_SIZE]; |
|  |  |
|  | if(argc < 3){ |
|  | fprintf(stderr, "Usage: %s <hostname> <port>\n", argv[0]); |
|  | exit(1); |
|  | } |
|  |  |
|  | fd = socket\_connect(argv[1], atoi(argv[2])); |
|  | write(fd, "GET /\r\n", strlen("GET /\r\n")); // write(fd, char[]\*, len); |
|  | bzero(buffer, BUFFER\_SIZE); |
|  |  |
|  | while(read(fd, buffer, BUFFER\_SIZE - 1) != 0){ |
|  | fprintf(stderr, "%s", buffer); |
|  | bzero(buffer, BUFFER\_SIZE); |
|  | } |
|  |  |
|  | shutdown(fd, SHUT\_RDWR); |
|  | close(fd); |
|  |  |
|  | return 0; |
|  | } |