**code for CRC**

#include<iostream>

using namespace std;

char xor1(char a,char b)

{

if(a!=b)

return '1';

else

return '0';

}

int main()

{

string divsr,divdnd,quot,rem;

cout<<"enter data and generator";

cin>>divdnd;

cin>>divsr;

int t=divdnd.length();

for(int m=0;m<divsr.length()-1;m++)

{

divdnd = divdnd+'0';

}

string h=divdnd;

int k=0,l=0;

for(int j=0;j<=(divdnd.length()-divsr.length());j++)

{

if(divdnd[j]=='1')

{ quot[k++]='1';

for(int i=0;i<divsr.length();i++)

{

divdnd[i+j]=xor1(divdnd[i+j],divsr[i]);

}

}

else

{

quot[k++]='0';

}

}

int p=0,m=divdnd.length()-divsr.length()+1;

for(int i=(divdnd.length()-divsr.length()+1);i<=divdnd.length();i++)

{

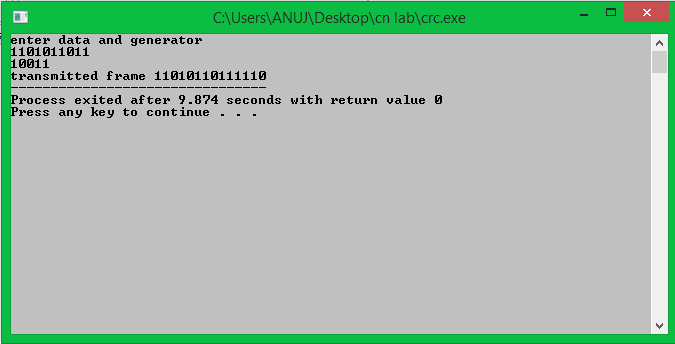
cout<<divdnd[i]<<endl;

h[m++]= xor1(divdnd[i],'0' );

}

cout<<"transmitted frame"<<h;

}



**code for bit destuffing**

#include<iostream>

using namespace std;

int main()

{

string s,m="";

cout<<"enter string to destuff"<<endl;

cin>>s;

int count=0,k=0;

for(int i=0;i<s.length();i++)

{

if(s[i]=='0')

{

count=0;m+=s[i];

}

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

{

count++;m+=s[i];

}

if(count==5)

{

count=0;

i=i+2;

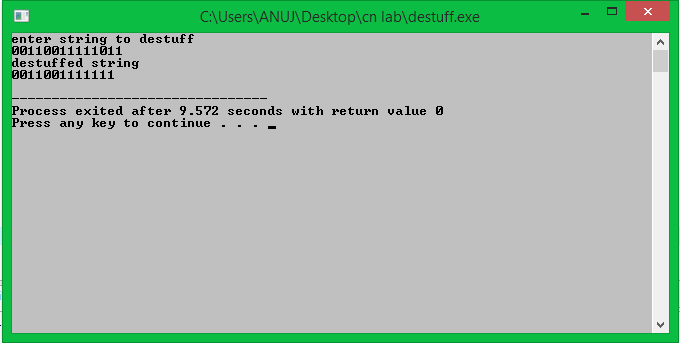
m+=s[i];

}

}

cout<<"destuffed string"<<m<<endl;

}



**//code for bit stuffing**

#include<iostream>

using namespace std;

int main()

{

string s,m="";

cout<<"enter string to stuff";

cin>>s;

int count=0,k=0;

for(int i=0;i<s.length();i++)

{

if(s[i]=='0')

{

count=0;m+=s[i];

}

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

{

count++;m+=s[i];

}

if(count==5)

{

count=0;

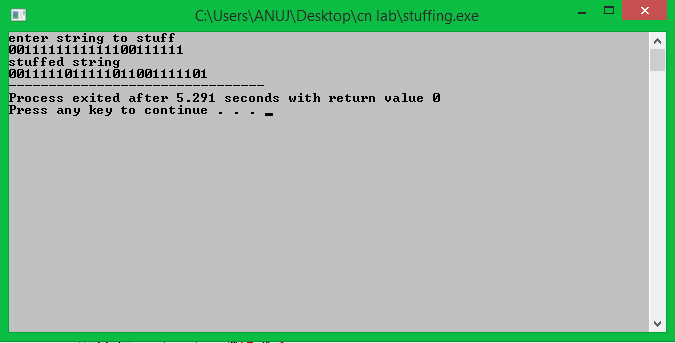
m+='0';

}

}

cout<<"stuffed string"<<m;

}



**go back n protocol**

# include <iostream>

# include <conio.h>

# include <stdlib.h>

# include <time.h>

# include <math.h>

# define TOT\_FRAMES 500

# define FRAMES\_SEND 10

using namespace std;

class gobkn

{

private:

int fr\_send\_at\_instance;

int arr[TOT\_FRAMES];

int arr1[FRAMES\_SEND];

int sw;

int rw; // tells expected frame

public:

gobkn();

void input();

void sender(int);

void reciever(int);

};

gobkn :: gobkn()

{

sw = 0;

rw = 0;

}

void gobkn :: input()

{

int n; // no of bits for the frame

int m; // no of frames from n bits

cout << "Enter the no of bits for the sequence no ";

cin >> n;

m = pow (2 , n);

int t = 0;

fr\_send\_at\_instance = (m / 2);

for (int i = 0 ; i < TOT\_FRAMES ; i++)

{

arr[i] = t;

t = (t + 1) % m;

}

sender(m);

}

void gobkn :: sender(int m)

{

int j = 0;

for (int i = sw ; i < sw + fr\_send\_at\_instance ; i++)

{

arr1[j] = arr[i];

j++;

}

for (int i = 0 ; i < j ; i++)

cout << " SENDER : Frame " << arr1[i] << " is sent\n";

reciever (m);

}

void gobkn :: reciever(int m)

{

time\_t t;

int f;

int f1;

int a1;

char ch;

srand((unsigned) time(&t));

f = rand() % 10;

// if = 5 frame is discarded for some reason

// else they are correctly recieved

if (f != 5)

{

for (int i = 0 ; i < fr\_send\_at\_instance ; i++)

{

if (rw == arr1[i])

{

cout << "RECIEVER : Frame " << arr1[i] << " recieved correctly\n";

rw = (rw + 1) % m;

}

else

cout << "RECIEVER : Duplicate frame " << arr1[i] << " discarded\n";

}

a1 = rand() % 15;

// if a1 belongs to 0 to 3 then

// all ack after this (incl this one) lost

// else

// all recieved

if (a1 >= 0 && a1 <= 3)

{

cout << "(Acknowledgement " << arr1[a1] << " & all after this lost)\n";

sw = arr1[a1];

}

else

sw = (sw + fr\_send\_at\_instance) % m;

}

else

{

f1 = rand() % fr\_send\_at\_instance;

// f1 gives index of the frame being lost

for (int i = 0 ; i < f1 ; i++)

{

if (rw == arr1[i])

{

cout << " RECIEVER : Frame " << arr1[i] << " recieved correctly\n";

rw = (rw + 1) % m;

}

else

cout << " RECIEVER : Duplicate frame " << arr1[i] << " discarded\n";

}

int ld = rand() % 2;

// ld == 0 frame damaged

// else frame lost

if (ld == 0)

cout << " RECIEVER : Frame " << arr1[f1] << " damaged\n";

else

cout << " (Frame " << arr1[f1] << " lost)\n";

for (int i = f1 + 1 ; i < fr\_send\_at\_instance ; i++)

cout << " RECIEVER : Frame " << arr1[i] << " discarded\n";

cout << " (SENDER TIMEOUTS --> RESEND THE FRAME)\n";

sw = arr1[f1];

}

cout << "Want to continue...";

cin >> ch;

if (ch == 'y')

sender(m);

else

exit(0);

}

int main()

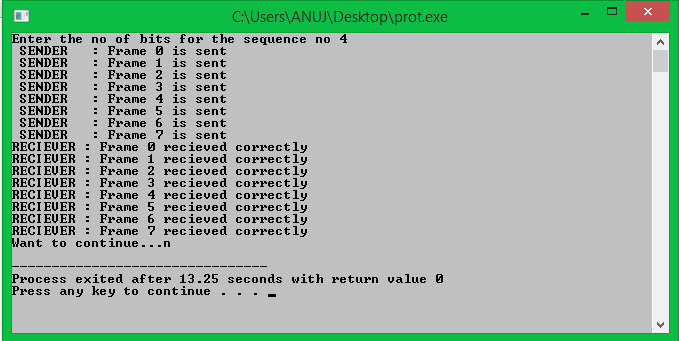
{

gobkn gb;

gb.input();

getch();

}



***program for wait and stop***

#include <conio.h>

#include <dos.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <bits/stdc++.h>

#define TIMEOUT 5

#define MAX\_SEQ 1

#define TOT\_PACKETS 8

#define inc(k) if(k<MAX\_SEQ) k++; else k=0;

using namespace std;

typedef struct

{

int data;

}packet;

typedef struct

{

int kind;

int seq;

int ack;

packet info;

int err;

}frame;

frame DATA;

typedef enum{frame\_arrival,err,timeout,no\_event} event\_type;

void from\_network\_layer(packet \*);

void to\_network\_layer(packet \*);

void to\_physical\_layer(frame \*);

void from\_physical\_layer(frame \*);

void wait\_for\_event\_sender(event\_type \*);

void wait\_for\_event\_reciever(event\_type \*);

void reciever();

void sender();

int i=1; //Data to be sent by sender

char turn; //r , s

int DISCONNECT=0;

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

int main()

{

rand();

while(!DISCONNECT)

{

sender();

sleep(1);

reciever();

}

getch();

}

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

void sender()

{

static int frame\_to\_send=0;

static frame s;

packet buffer;

event\_type event;

static int flag=0;

if(flag==0)

{

from\_network\_layer(&buffer);

s.info = buffer;

s.seq = frame\_to\_send;

printf("SENDER : Info = %d Seq No = %d ",s.info,s.seq);

turn = 'r';

to\_physical\_layer(&s);

flag = 1;

}

wait\_for\_event\_sender(&event);

if(turn=='s')

{

if(event==frame\_arrival)

{

from\_network\_layer(&buffer);

inc(frame\_to\_send);

s.info = buffer;

s.seq = frame\_to\_send;

printf("SENDER : Info = %d Seq No = %d ",s.info,s.seq);

turn = 'r';

to\_physical\_layer(&s);

}

if(event==timeout)

{

printf("SENDER : Resending Frame ");

turn = 'r';

to\_physical\_layer(&s);

}

}

}

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

void reciever()

{

static int frame\_expected=0;

frame r,s;

event\_type event;

wait\_for\_event\_reciever(&event);

if(turn=='r')

{

if(event==frame\_arrival)

{

from\_physical\_layer(&r);

if(r.seq==frame\_expected)

{

to\_network\_layer(&r.info);

inc(frame\_expected);

}

else

printf("RECIEVER : Acknowledgement Resent\n");

turn = 's';

to\_physical\_layer(&s);

}

if(event==err)

{

printf("RECIEVER : Garbled Frame\n");

turn = 's'; //if frame not recieved

} //sender shold send it again

}

}

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

void from\_network\_layer(packet \*buffer)

{

(\*buffer).data = i;

i++;

}

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

void to\_physical\_layer(frame \*s)

{ // 0 means error

s->err = rand(); //non zero means no error

DATA = \*s; //probability of error = 1/4

}

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

void to\_network\_layer(packet \*buffer)

{

printf("RECIEVER :Packet %d recieved , Ack Sent\n",(\*buffer).data);

if(i>TOT\_PACKETS) //if all packets recieved then disconnect

{

DISCONNECT = 1;

printf("\nDISCONNECTED");

}

}

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

void from\_physical\_layer(frame \*buffer)

{

\*buffer = DATA;

}

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

void wait\_for\_event\_sender(event\_type \* e)

{

static int timer=0;

if(turn=='s')

{

timer++;

if(timer==TIMEOUT)

{

\*e = timeout;

printf("SENDER : Ack not recieved=> TIMEOUT\n");

timer = 0;

return;

}

if(DATA.err==0)

\*e = err;

else

{

timer = 0;

\*e = frame\_arrival;

}

}

}

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

void wait\_for\_event\_reciever(event\_type \* e)

{

if(turn=='r')

{

if(DATA.err==0)

\*e = err;

else

\*e = frame\_arrival;

}

}

