// SELECTIVE REPEAT PROTOCOL

#include <conio.h>

#include <iostream.h>

#include <dos.h>

#include <stdio.h>

#include <stdlib.h>

#include<time.h>

#define MAX\_SEQ 3

#define NR\_BUFS ((MAX\_SEQ+1)/2)

#define TOT\_PACKETS 5

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

typedef enum{true,false} boolean;

boolean no\_nak=true;

typedef struct

{

int data;

}packet;

typedef enum{ack,data,nak} frame\_kind;

typedef struct

{

int seq;

int ack;

frame\_kind kind;

packet info;

}frame;

frame DATA,DATA1;

typedef enum{frame\_arrival,timeout,error,network\_layer\_ready,ack\_timeout} event\_type;

class station

{

public:

int i; //Data to be sent by sender

int DISCONNECT;

frame r,s;

packet buffer[MAX\_SEQ+1];

int frame\_to\_send,frame\_expected,ack\_expected;

int nbuffered,too\_far;

packet out\_buf[NR\_BUFS];

packet in\_buf[NR\_BUFS];

boolean arrived[NR\_BUFS];

event\_type event;

station()

{

DISCONNECT=0;

i=1;

frame\_to\_send=0;

frame\_expected=0;

ack\_expected=0;

nbuffered=0;

enable\_network\_layer();

too\_far=NR\_BUFS;

}

void protocol();

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

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

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

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

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

void enable\_network\_layer();

void disable\_network\_layer();

boolean between(int a,int b,int c);

void send\_data(frame\_kind fk,int frame\_nr,int frame\_expected,packet buffer[]);

};

void main()

{

clrscr();

station s1;

int x=0;

while(x<NR\_BUFS)

{

s1.arrived[x]=true;

x++;

}

while(1)

{

s1.protocol();

delay(400);

if((s1.DISCONNECT==1))

break;

}

getch();

}

void station::protocol()

{

wait\_for\_event(&event);

switch(event)

{

case network\_layer\_ready:

from\_network\_layer(&out\_buf[frame\_to\_send%NR\_BUFS]);

nbuffered++;

//send\_data(frame\_to\_send,frame\_expected,buffer);

send\_data(data,frame\_to\_send,frame\_expected,out\_buf);

cout<<" frame send:"<<frame\_to\_send;

inc(frame\_to\_send);

break;

case frame\_arrival:

from\_physical\_layer(&r);

if(r.kind==data)

{

if((r.seq!=frame\_expected)&& no\_nak)

send\_data(nak,0,frame\_expected,out\_buf);

if(between(frame\_expected,r.seq,too\_far)&&(arrived[r.seq%NR\_BUFS]==false))

{

arrived[r.seq%NR\_BUFS]=true;

in\_buf[r.seq%NR\_BUFS]=r.info;

while(arrived[frame\_expected%NR\_BUFS])

{

to\_network\_layer(&in\_buf[frame\_expected%NR\_BUFS]);

no\_nak=true;

arrived[frame\_expected%NR\_BUFS]=false;

inc(frame\_expected);

//start\_timer();

}

}

}

if((r.kind==nak) && between(ack\_expected,((r.ack+1)%(MAX\_SEQ+1)),frame\_to\_send))

send\_data(data,(r.ack+1)%(MAX\_SEQ+1),frame\_expected,out\_buf);

while(between(ack\_expected,r.ack,frame\_to\_send));

{

nbuffered--;

//stop\_timer(ack\_expected%NR\_BUFS);

i++;

inc(ack\_expected);

}

break;

case error:

if(no\_nak)

send\_data(nak,0,frame\_expected,out\_buf);

break;

/\* case timeout:

send\_data(data,oldest\_frame,frame\_expected,out\_buf);

break;\*/

case ack\_timeout:

{

cout<<"\nacknowledgement timeout...";

send\_data(ack,0,frame\_expected,out\_buf);

}

}

/\*if(nbuffered<NR\_BUFS)

enable\_network\_layer();

else

disable\_network\_layer();\*/

/\* if(r.ack==frame\_to\_send)

{

cout<<"\nstation:"<<st\_no<<"..ack received";

from\_network\_layer(&buffer);

inc(frame\_to\_send);

i++;

}\*/

/\* while(between(ack\_expected,r.ack,frame\_to\_send))

{

nbuffered--;

i++;

cout<<"\nacknowledgement number:"<<ack\_expected<<" received";

inc(ack\_expected);

}

break;

case error:

break;

case timeout:

frame\_to\_send=ack\_expected;

cout<<"\ntimeout...resending frames..\n";

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

{

cout<<" sending frame:"<<frame\_to\_send;

send\_data(frame\_to\_send,frame\_expected,buffer);

inc(frame\_to\_send);

}

} \*/

if(nbuffered<MAX\_SEQ)

enable\_network\_layer();

else

disable\_network\_layer();

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

{

DISCONNECT = 1;

cout<<"\nDISCONNECTED:";

return;

}

}

boolean station::between(int a,int b,int c)

{

if( ((a<=b)&&(b<c)) || ((c<a)&&(a<=b)) || ((b<c)&&(c<a)) )

return true;

else

return false;

}

void station::enable\_network\_layer()

{

event=network\_layer\_ready;

}

void station::disable\_network\_layer()

{

event=frame\_arrival;

}

void station::send\_data(frame\_kind fk,int frame\_nr,int frame\_expected,packet buffer[])

{

frame s;

s.kind=fk;

if(fk==data)

s.info=buffer[frame\_nr%NR\_BUFS];

s.seq=frame\_nr;

s.ack=(frame\_expected+MAX\_SEQ)%(MAX\_SEQ+1);

if(fk==nak)

no\_nak=false;

to\_physical\_layer(&s);

//if(fk==data)

//start\_timer(frame\_nr%NR\_BUFFS);

//stop\_ack\_timer();

}

/\*void station::send\_data(int frame\_nr,int frame\_expected,packet buffer[])

{

frame s;

s.info=buffer[frame\_nr];

s.seq=frame\_nr;

s.ack=(frame\_expected+MAX\_SEQ)%(MAX\_SEQ+1);

to\_physical\_layer(&s);

}\*/

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

{

(\*buffer).data = i;

}

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

{

DATA1=\*s;

}

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

{

if((\*buffer).data==0)

cout<<" new packet received";

else

cout<<" Packet "<<(\*buffer).data<<" recieved , Ack Sent";

}

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

{

\*buffer = DATA;

(\*buffer).seq=frame\_expected;

(\*buffer).ack=ack\_expected;//ack\_expected;

}

void station::wait\_for\_event(event\_type \* e)

{

int r;

if(\*e==network\_layer\_ready)

return;

else

{

int x=random(5);

if(x%2==0)

\*e=frame\_arrival;

else

\*e=ack\_timeout;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

frame send:0 frame send:1 frame send:2 frame send:3 frame send:0

acknowledgement timeout... frame send:1

acknowledgement timeout...

acknowledgement timeout... frame send:2

DISCONNECTED:

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