#include <stdio.h>

#include <string.h>

#include "protocol.h"

#include "datalink.h"

#define MAX\_SEQ 15

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

#define DATA\_TIMER 3000

#define ACK\_TIMER 240

#define DATA 1

#define ACK 2

#define NAK 3

struct frame{

unsigned char kind ;

unsigned char ack ;

unsigned char seq ;

unsigned char info[PKT\_LEN] ;

unsigned int padding;

} ;

static int no\_nak = 1 ;

static int phl\_ready = 0 ;

static int between (unsigned char a, unsigned char b, unsigned char c)

//窗口函数，判断帧号（ack号是否在窗口内）

{

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

return 1 ;

else

return 0 ;

}

static void put\_frame(unsigned char \*frame, int len)

//加入校验和CRC

{

\*(unsigned int \*)(frame + len) = crc32(frame, len);

send\_frame(frame, len + 4);

phl\_ready = 0 ;

}

static void send\_data\_frame( unsigned char fk, unsigned char frame\_nr, unsigned char frame\_expected, unsigned char buffer[NR\_BUFS][PKT\_LEN] )

//发送数据帧、ack、nak

{

struct frame s;

s.kind = fk;

s.seq = frame\_nr;

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

if ( fk == DATA )

{

memcpy( s.info, buffer[frame\_nr % NR\_BUFS], PKT\_LEN ) ;

//复制分组到帧中

dbg\_frame( "Send DATA %d %d, ID %d\n", s.seq, s.ack, \*(short \*)s.info ) ;

put\_frame( (unsigned char \*)&s, 3 + PKT\_LEN ) ;

//加入校验和

start\_timer( frame\_nr % NR\_BUFS, DATA\_TIMER ) ;

}

else if ( fk == NAK )

{

no\_nak = 0 ;

put\_frame( (unsigned char \*)&s, 3 ) ;

}

else if (fk == ACK )

{

dbg\_frame( "Send ACK %d\n", s.ack ) ;

put\_frame( (unsigned char \*)&s, 3 ) ;

}

phl\_ready = 0 ;

stop\_ack\_timer() ;

}

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

{

int event, arg;

struct frame f ;

int len = 0 ;

int i ;

static unsigned char ack\_expected = 0, next\_frame\_to\_send = 0 ;

static unsigned char frame\_expected = 0, too\_far = NR\_BUFS ;

static unsigned char nbuffered = 0 ;

int arrived[NR\_BUFS] ;

unsigned char in\_buf[NR\_BUFS][PKT\_LEN], out\_buf[NR\_BUFS][PKT\_LEN] ;

protocol\_init(argc, argv);

lprintf("Designed by dao, build: " \_\_DATE\_\_" "\_\_TIME\_\_"\n");

for ( i = 0 ; i < NR\_BUFS ; i ++ )

{

arrived[i] = 0 ;

}

//enable\_network\_layer();

for (;;)

{

event = wait\_for\_event(&arg) ;

switch (event)

{

case NETWORK\_LAYER\_READY :

get\_packet( out\_buf[next\_frame\_to\_send % NR\_BUFS] ) ;

nbuffered ++ ;

send\_data\_frame( DATA, next\_frame\_to\_send, frame\_expected, out\_buf ) ;

next\_frame\_to\_send = ( next\_frame\_to\_send + 1 ) % ( MAX\_SEQ + 1 ) ;

break;

case PHYSICAL\_LAYER\_READY :

phl\_ready = 1 ;

break ;

case FRAME\_RECEIVED :

len = recv\_frame( (unsigned char \*)&f, sizeof f ) ;

if ( len < 5 || crc32((unsigned char \*)&f, len) != 0 )

//校验和出错，发送nak请求重传

{

if ( no\_nak == 1 )

{

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

}

dbg\_event("\*\*\*\* Receiver Error, Bad CRC Checksum\n") ;

break;

}

if ( f.kind == DATA )

{

if ( (f.seq != frame\_expected ) && no\_nak == 1 )

//序列号错误，发送nak重传

{

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

}

else

{

start\_ack\_timer( ACK\_TIMER ) ;

}

if ( between( frame\_expected, f.seq, too\_far ) && arrived[f.seq % NR\_BUFS] == 0 )

{

//log\_printf("Received a frame, "); log\_printf("%d bytes\n", len);

dbg\_frame( "Recv DATA %d %d, ID %d\n", f.seq, f.ack, \*(short \*)f.info ) ;

arrived[f.seq % NR\_BUFS] = 1 ;

memcpy( in\_buf[f.seq % NR\_BUFS], f.info, len-7 ) ;

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

{

put\_packet( in\_buf[ frame\_expected % NR\_BUFS ], len-7 ) ;

no\_nak = 1 ;

arrived[ frame\_expected % NR\_BUFS ] = 0 ;

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

too\_far = ( too\_far + 1 ) % ( MAX\_SEQ + 1 ) ;

start\_ack\_timer( ACK\_TIMER ) ;

}

}

}

if ( (f.kind == NAK) && between( ack\_expected, ( f.ack + 1 ) % ( MAX\_SEQ + 1 ), next\_frame\_to\_send ) )

//发送方收到nak，重传

{

dbg\_frame( "Recv NAK %d\n", f.ack ) ;

send\_data\_frame( DATA, (f.ack + 1 ) % ( MAX\_SEQ + 1 ), frame\_expected, out\_buf ) ;

}

while ( between( ack\_expected, f.ack, next\_frame\_to\_send ) )

{

nbuffered -- ;

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

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

}

break ;

case DATA\_TIMEOUT :

//发送方超时重传

dbg\_event( "---- DATA %d timeout\n", arg ) ;

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

break ;

case ACK\_TIMEOUT :

dbg\_event( "---- ACK %d timeout\n", arg ) ;

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

break ;

}

if ( (nbuffered < NR\_BUFS) && phl\_ready )

enable\_network\_layer() ;

else

disable\_network\_layer() ;

}

}