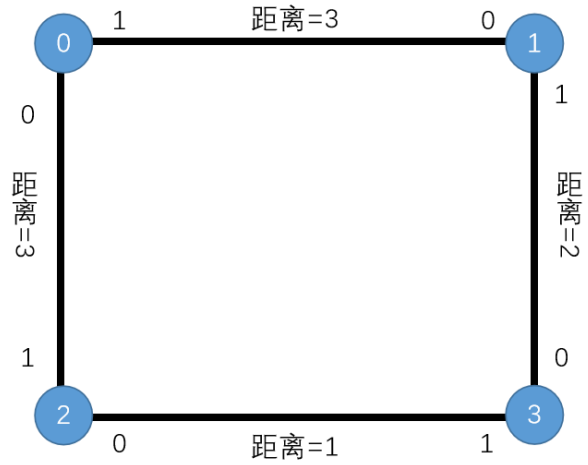
一.设计概览

1. 网络拓扑图：



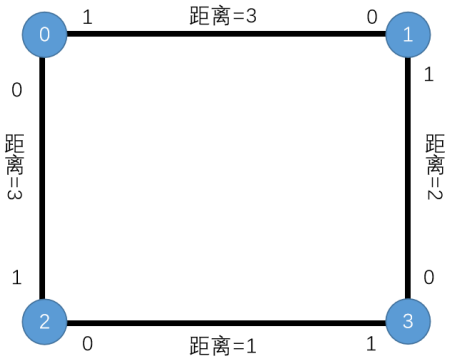
2. 预期路由表

0号主机路由表

目的	下一站	端口
0	99	0
1	1	1
2	2	0
3	2	0

1号主机路由表

目的	下一站	端口
0	0	0
1	99	0
2	3	1
3	3	1



2号主机路由表

目的	下一站	端口
0	0	1
1	3	0
2	99	0
3	3	0

3号主机路由表

目的	下一站	端口
0	2	1
1	1	0
2	2	1
3	99	0

3. 实现功能简述：

在本项目，我们实现了具有 4 个节点的环状拓扑网络的各节点路由表建立以及图片数据的跨节点传输。

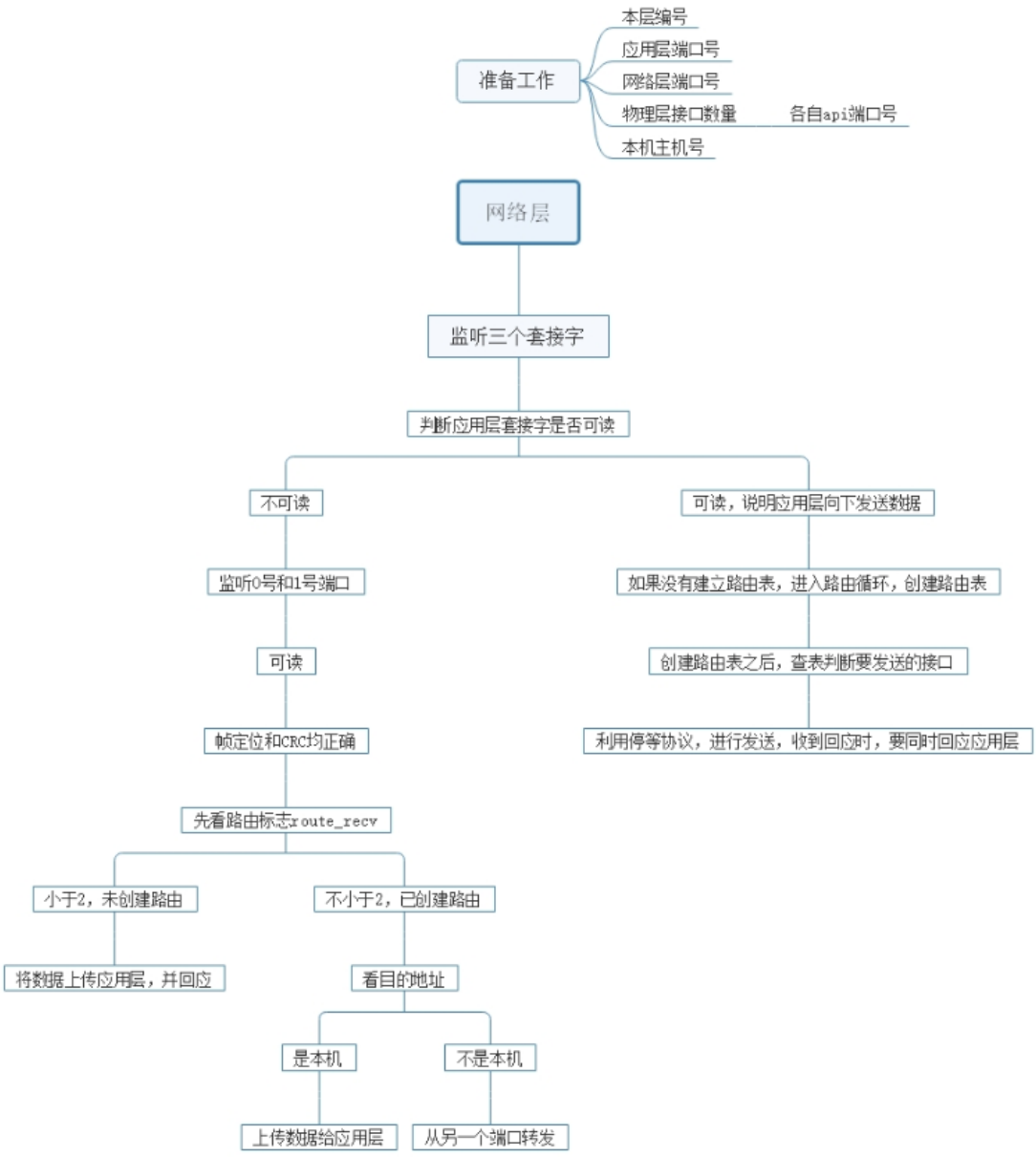
4. 网络层设计

4.1 准备工作：

输入本层编号
输入应用层端口号
输入网络层端口号
输入物理层接口数量
分别输入各自的 api 端口号

输入本地主机号

4.2 流程图：



4.3 实现逻辑：

需要监听三个套接字：应用层，0 号端口和 1 号端口

如果可读的话：

首先看是不是应用层的套接字可读：

如果是：（说明应用层向下发送数据）

接受数据

然后判断建立路由标志 route_choice 是否等于 0：

如果 route_choice=0，则说明还未创建路由，需要先创建路由表：

首先将收到的数据发给 0 号端口，然后进行停等，直到收到回应

然后将收到的数据发给 1 号端口，然后进行停等，直到收到回应
然后将 route_choice 置 1，说明结束发送初始路由阶段
如果 route_choice 不等于 0，则说明初始路由已经发送过，进入正常的数据发送：
首先判断收到数据的第 0 位是 0 还是 1，如果是 0 则选择 0 号端口发送数据，如果是 1 则选择 1 号端口发送数据。
具体发送数据的步骤：
先判断数据初始标志 count 是否等于-1：
如果等于-1，说明这是这段数据的第一帧（包含长度等信息)：
则需要提出去图片长度，比特流长度
然后用比特率长度除以每一数据块的长度（3160）再加 1
得到总共需要传输的帧数 endline
count++,表示收到了第一帧，并得到了长度
如果 count 不等于-1：
（假设发送端口为 0 号端口为例）
进入发送循环：
清空监听数组 readfds
将 0 号端口的套接字绑定
并将套接字数组 sock_list 放到监听数组 readfds 中
Select 查看监听数组 readfds：
如果超时，则重新将数据发送到 0 号端口
如果应用层发来数据，则接受并从 0 号端口发出，同时 endline--
如果是 0 号端口发来的收据，接受（说明对方已经成功收到）
然后向应用层发送回应信息 ack，指示它可以继续发送下一帧
然后判断 endline 是否等于 0：
如果 endline 等于 0
说明发送结束，将跳出发送循环，count=-1
如果 endline 不等于 0
说明发送未结束，继续发送
然后遍历套接字数组中的 0 号端口套接字和 1 号端口套接字：
当发现可读时，则设该端口号为 intf
接受数据，并进行查帧，CRC 的检验，
如果检验后发现错误，则丢弃
如果检验后发现正确：
先看接受路由标志 route_recv 是否小于 2：
如果 route_recv 小于 2，说明收到的是路由表数据：
然后上传到应用层，并回应 intf 端口
route_recv++
如果 route_recv 不小于 2，说明路由表已经建立，收到的是信息数据：
解出目的地址，看目的地址是否等于本机地址：
如果目的地址等于本机地址：（进行接受）
将数据上传给应用层，并回应 intf 端口
如果目的地址不等于本机地址：（进行转发）

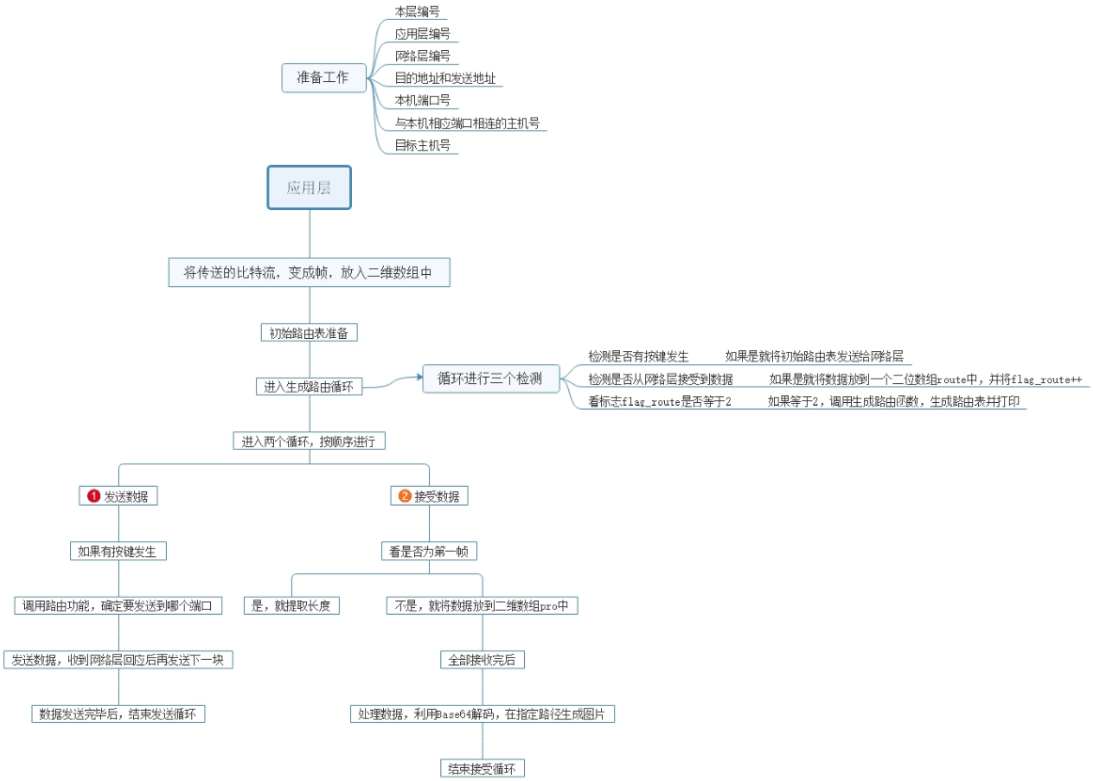
设 intf0=intf（传入端口），intf1=（intf+1）%2（传出端口）
将收到的数据发送到 intf1 口
然后看 intf0 口是否可读，如果可读进行转发；
然后看 intf1 口是否可读，如果可读对 inf0 口进行回应
接受

5. 应用层设计

5.1 准备工作：

输入本层编号
输入应用层端口号
输入网络层端口号
输入目的地址和发送地址：
输入本机端口号
输入与本机相应端口相连的主机号
输入目标主机号
按下任意键生产路由表
（生成路由表之后）
按下任意键发送数据

5.2 流程图：



5.3 实现逻辑：

首先是准备传输的帧：

通过图片路径打开图片，进行 Base64 编码，将生成的输入放到一个二维数组 binary_array 中，

其中每一块为 3168 位，第 0 块只包含图片大小，比特流长度作为控制信息发送

然后进行对 binary_array 数组的每一块进行 CRC，成帧

最后形成了 framebuff 数组中

然后是路由准备：

有个初始的四维数组 Initial_route[4][6]，里面放的是各个节点的初始路由信息

比如 0 号节点的初始路由信息是{0,3,1,0,3,2},

0,3,1:表示 0 到 1 的距离为 3

然后会通过输入的本机号 initial_choice 来进行选择初始路由信息（Initial_route[initial_choice])

发送工作：

首先进入的生成路由循环中：（利用 MMM=2 控制，当 MMM=0 时结束）

有三个检测：

第一个检测是否按键发生，如果是，就将初始路由信息发送给网络层，让其发送给相邻的两个主机

第二个检测是看是否从网络层接受到数据，如果接受到就放到二维数组 Route 中，然后让计数器 flag_route++，这样就可以将从两个端口收到的路由数据整合到一个二维数组 Route 中，然后生成路由表了，MMM--

第三个检测是看 flag_route 是否等于 2，如果是则调用生成路由函数（IP（））生成路由表，并打印出来,然后 flag_route++，MMM--

这样可以看出只有当路由表生成并且自己发送完自己的初始路由信息后才会结束生成路由循环

然后是发送数据：

如果有按键发生：

令 F=port_host[send[HOST]]查看要发送的端口号，（port_host[send[HOST]]是调用路由表的功能）

然后将 F 加到每次传输的数据的头部，在网络层判断后会去掉

通过获取 framebuff[0]中的比特流长度，用比特流长度除以 3168 再加 1，可以得到要发送的数据帧的总数 endline（用来进行控制）

然后开始发送 framebuff 从 1 到 endline 的每一块，

发送一块后，开始等待，

当收到网络层的回应时才会发送下一块，然后发送计数器 number++

当 number>=endline 的时候，表示这次数据已经发送完成了，跳出发送循环

最后是接受数据，当接受完路由的两个帧后收到的数据都是要传输的数据：

如果计数器 count==-1 说明是这次数据传输的第一帧，需要对接受的数据进

行处理：

提取图片长度，比特流长度，计算出这次要传输的帧的个数

然后 count++

如果计数器 count 不等于-1：

则开始接受数据，将每次收到的数据放到二维数组 pro 中，最后 pro 中的数据就是开始的 binary_array 中的数据

每收到一次，endline--

当 endline==0 时，结束接受循环

开始处理数据，利用 Base64 编码，解出 pro 中数据所代表的图片，并写到相应路径上。

二.课程知识应用

1. TCP/IP

1.1 垂直分层

TCP/IP 模型垂直分层从底层到高层依次为物理层、数据链路层、网络层、传输层、应用层共 5 层。

其中从物理层的主机到网络层之下的物理层和数据链路层属于网络内部的通信，内容不由 TCP/IP 定义。

网络层：IP 协议：核心协议，实现寻径和数据转发功能；ARP 协议：物理网地址和 IP 地址映射的关键协议；ICMP 协议：网络层控制信息协议，差错通知

传输层：TCP：提供面向连接，可靠通信服务的端到端协议；UDP：提供无连接的，不保证可靠性的端到端协议

应用层：FTP：文件传输协议，常用于文件下载；TELNET：远程虚拟终端协议，常用于 BBS；SMTP/POP3：常用于电子邮件；DNS：域名解析协议，用于域名和 IP 地址之间的映射

1.2 各层功能

联系第一单元的课程知识学习，分析各层功能。

物理层功能有 Bit 的信号表示、同步、收发、通信过程协调、物理介质驱动。

数据链路层功能有成帧：数据以块为单位、帧差错检测、帧差错处理（重传）、流量控制：控制线路两端速率一致。

网络层功能有路由计算、 路由选择、报文分段和重组、拥塞控制、网络互连和组网。

传输层功能有各种网络的适配，网络通信的分流和复用、多个并发通信的管理、流量控制等。

应用层：各种使用网络通信的应用都属于应用层，新的网络应用还在不断出现。

2. 路由表的建立

2.1 路由表

路由表类似道路指示牌，其建立方法是事先计算所有最优路由，形成路由表，也即 PDU 转发表，然后各节点在传输数据时根据路由表进行 PDU 的转发。路由表

表项一般包括目的节点、下一条和距离，一共 3 项。建立路由表的算法有静态路由算法和动态路由算法，其中包括距离矢量算法以及链路状态算法。我们选择最为优化的链路状态算法。

2.2 距离矢量算法

距离矢量算法以中继节点个数为度量。工作方式是每个节点自动找出相邻节点，形成初始路由表，距离为 1；每个节点定期和相邻节点交换路由信息——路由及距离；根据收到的路由信息，更新到其他节点的路径（最短距离）；通过不断扩散，逐渐形成到所有节点的路由。

但是距离矢量算法可能出现无穷计数问题，即，在某种情况下，距离矢量算法可能出现路由环路，其现象是路由表项随路由信息更新，不断增加。

2.3 链路状态算法

链路状态算法以链路状态作为链路度量，规避了距离矢量算法中可能产生的出现路由环路而无穷计数的问题，延时比节点数更能反映网络和信道的实际状况，线路的速率、当前负载下节点处理能力都会影响延时，能较好地防止网络拥塞现象、均匀分布网络流量，从发出 PDU 到收到应答来测量延时及变化

链路状态算法工作方式:从每个节点探询相邻节点,得到延时(链路状态)初始值；每个节点定期和所有节点交换路由信息；探询所得的相邻节点链路质量；根据收集到的路由信息，计算到其他节点的路径；最小延时的路径即为最短路径即为最佳路由。

3. 停等协议

停等协议的核心思想是发送方在完成一帧数据的发送后，等到接收方应答后再继续传输下一帧的方式。

关于差错控制探讨，根据接收方的应答，决定是否重传；如果接收拒绝，重传；如果接收方根本没有收到，或者应答丢了；发送方超时没有收到应答，重传。

4. CRC 循环冗余检验

CRC 的基本思想是给定生成式：G(x)，r+1 位；将原始码串左移 r 位后与生成式相“除”，得到余数码串；“除”为模 2 除，不借位进行异或；余数码串即为校验位码串；正常情况下，收方用相同的生成式去除传输码串；余数为 0 即为正确，余数不为 0 即为错误。

用数学表达为若传输中有错

$$T`(x)=T(x)+E(x)$$

$$T`(x)/G(x)=(2rM(x)+R(x)+E(x))/G(x)$$

$$=Q(x)+R(x)/G(x)+R(x)/G(x)+E(x)/G(x)$$

$$=Q(x)+E(x)/G(x)$$

余数不为 0。

5. 比特填充的字符界定法成帧

在帧的头之前和尾之后加一个特殊的字符 flag,只要读到这个字符帧就开始了，再次读到就认为这个帧结束了，在本实验中我们选择 flag 为 01111110。

为了避免我们所想要传输的数据流中本身就包含信息段为 6 个以上连续的 1，

我们在编码时当正文读取的时候一旦出现了 5 个连续的 1，那么在后面填充一个 0，避免出现 6 个 1 造成帧提前结束。

收端接收到之后，每读到 5 个连续的 1 之后，就把后面的 0 去掉一个，这样就得到了原文的数据了。

三.功能模块详细分析

1. 发收路由信息

1.1 模块名称或实现功能

发送与接收路由信息，并将接收的路由信息汇总。

1.2 模块所涉及知识

1.IP 协议寻径
2.路由表，下一跳，距离

1.3 模块代码及解释

模块伪代码展示:

```
int flag = 0 , i;
int initial[6] , route[2][6];
//输入这个路由器的路由信息
scanf ("%d" , &initial[6]);
while ( flag == 0)//flag=0 时进入路由，≠0 时跳出路由
{
   两边发 initial[6];
   for ( i = 0 ; i < 2; )
     if ( 收到相邻路由发的 initial[6] )
     {
       if ( i == 0 )
       {
         route[i][6] = 收到的 initial[6];
         i++;
       }
       else
         if ( 收到的 initial[6] != route[0][6] )
         {
           route[i][6] = 收到的 initial[6];
           i++;
           if ( i == 2 )
             flag = 1;
         }
     }
}
```

实现过程:

1.首先 scanf 输入本路由器的路由信息

2.flag 标志位为 0，进入路由状态

3.发送本路由的信息

4.接收到第一个路由信息，存到本路由的路由表中

5.接收到第二个路由信息，先验证是否是与第一个相同的路由发来的信息，若不是则保存到本路由的路由表中

1.4 模块亮点

避免了重复保存同一路由信息而造成的错误。

2. 建路由表

2.1 模块名称或实现功能

接收相邻端口的路由信息

通过每个结点的路由函数，建立特定的路由表。

2.2 模块所涉及知识

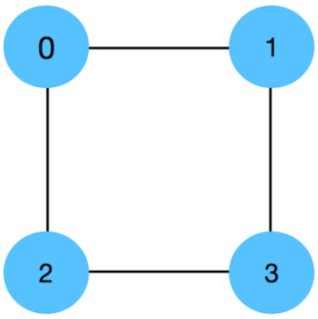
最短路径算法：计算每个路由器到其他端点的最短距离，及其传输方式。

路由表结构：目的地址、下一跳、距离。

泛射算法：整个网络先通过泛射法将每个结点的信息都发送给相邻的结点，从而知道所有的信息，并以此来计算路由。

得到路由表后的结点，在之后接收到消息时，可直接查找路由表每个地址所对应的下一跳，走最短的距离，花最短的时间来进行更有效率的传输。

2.3 模块代码及解释



每个端口标号为 0，1，2，3；这里 IP0 表示 0 号端口

输入为一个 2*6d 大小的数组，每行代表一个相邻接口，由于这里只有两个相邻接口所以设置为两行

每行的结构为:[相邻端口自身的端口号][到目标端口的距离][目标端口号]

输出为一维数组，send[1]表示数据想要传输到 1 号端口时的下一跳

对于与本路由器即 0 号结点相邻的结点路由需要进行一次判断：直接走 or 绕路走

对于不相邻的结点进行一次判断：左边走 or 右边走

对于本路由结点：距离设置为 99（大），接收到就会立即进行判断是否为本结点

具体代码如下：

```
int *IP0(int rec[2][6]){
```

static int send[4]; //输出的路由表，由于这里应用的是 y 同一套协议规则所以简化为一个数组

```
send[0]=99;//0-self
if(rec[0][2]==0){
    if(rec[0][1]<=rec[0][4]+rec[1][1]+rec[1][4]){
        send[rec[0][0]]=rec[0][0];
    }
    else{
        send[rec[0][0]]=rec[1][0];
    }
}
else{
    if(rec[0][4]<=rec[0][1]+rec[1][1]+rec[1][4]){
        send[rec[0][0]]=rec[0][0];
    }
    else
        send[rec[0][0]]=rec[1][0];
}
if(rec[1][2]==0){
    if(rec[1][1]<=rec[1][4]+rec[0][1]+rec[0][4]){
        send[rec[1][0]]=rec[1][0];
    }
    else
        send[rec[1][0]]=rec[0][0];
}
else{
    if(rec[1][4]<=rec[1][1]+rec[0][1]+rec[0][4]){
        send[rec[1][0]]=rec[1][0];
    }
    else
        send[rec[1][0]]=rec[0][0];
}
if(rec[0][1]+rec[0][4]>=rec[1][1]+rec[1][4]){
    if(rec[0][2]!=0){
        send[rec[0][2]]=rec[1][0];
    }
    else
        send[rec[0][5]]=rec[1][0];
}
else{
    if(rec[0][2]!=0){
        send[rec[0][2]]=rec[0][0];
    }
    else{
```

```
                send[rec[0][5]]=rec[0][0];
            }
        }
        return send;
    }
```

每个路由结点的函数只有略微差别，大体算法相同，这里不做赘述，只列出其中一个。

2.4 模块亮点

1. 由于本实验所涉及的路由结点较少，且考虑到其他的复杂性，本实验的路由算法只采用了简洁的判断方式。

2.通过建立路由，可以实现端到端的最短距离进行传输，大大节约了传输所耗费的时间，提高了传输的效率。

2.5 模块不足及改进

1.由于本实验简单可以只用判断来实现，当路由结点增加，需求增大时，需采用 Dijkstra 算法等来进行优化。

2.没有把传输的每个小部分进行清晰的命名，若以后要更改和增加路由，需要增加赋给每个小模块名字的过程。

3. 图片编码及解码

3.1 模块名称或实现功能

1、图片编码为 Base64 码；2、Base64 码编码为二进制码；3、二进制码转码为 Base64 码；4、Base64 码转码为图片；5、上述四个功能的整合和连接。

3.2 模块所涉及知识

Base64 码的优越性及 Base64 码与图片之间的互相转化。

Base64 码是一种在网络领域常用的编码方式，在 OSI 模型中，该协议位于表示层，定义了一种表示方式，使得各类文件都可以与其相应 Base64 码一一对应。一般的编码模式，在受到协议栈底层认为无关紧要的信息例如空格的影响后，解码出的文件与原文件“相似而不相同”，这对于有着严格定义的 jpg 或 png 文件来说是致命的损伤，而 Base64 编码很好地解决了这一矛盾。

Base64 编码采用 26 个英文字母的大小写和 10 个数字以及 2 个特殊字符共 64 个标识来对图片进行编码。这种编码有三个重要的原则：①把 3 个字符变成 4 个字符。②每 76 个字符加一个换行符。③最后的结束符也要处理。通过这些规则的约束，其他编码可能忽略掉的空格、换行、结尾得到了有效的处理。

二进制与 Base64 码之间的相互转换。

将任何一个文件转化为 Base64 编码后，其本质上就和一般的文档所类似，但要将其送往下一层次，还需要将其转化为二进制编码。我们编码时移位异或的方式，依此从高位到低位转出二进制码，解码时采用对比填充的方式，利用二分查找法查找二进制码对应得 Base64 码。

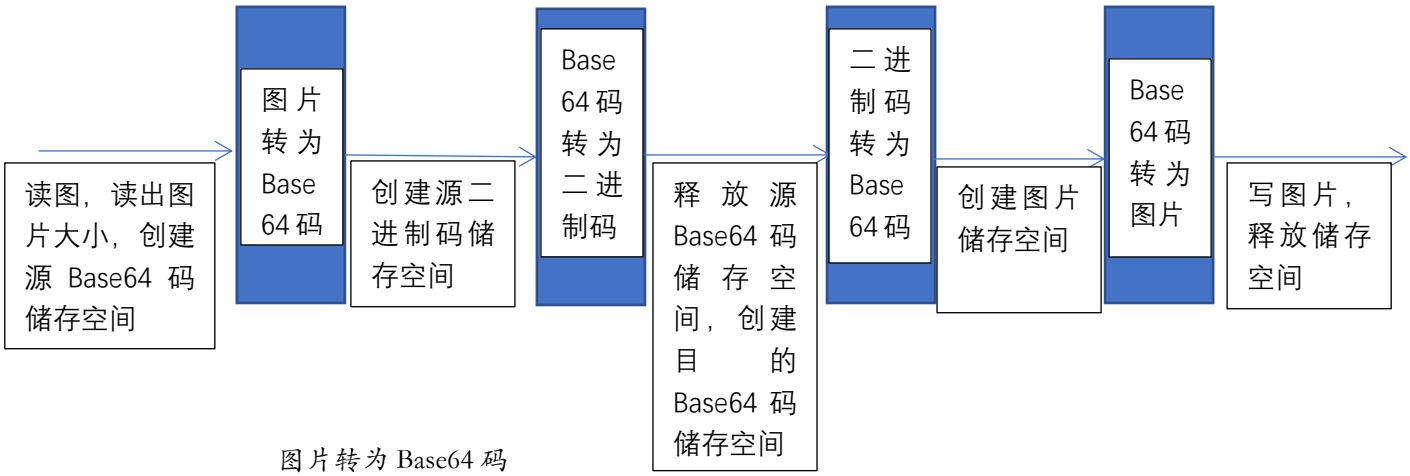
图片文件的相关操作和堆栈的相关管理。

图片相对于一般得文档而言，所占用得空间相对较大，由于通信系统要采用储存转发的魔术，这对于计算机的内存有了一定要求。我们采用的 VS 编译器默认的堆栈大小只有 8M，学生版更是只有 1M，由于我们对实际的每一位转换成了 8 位的 char 类型，所以我们实际可用的堆栈大小只有 128k。

所以为了节约空间，我们采用了动态分配内存的方式，分配的内存大小，显然是通过文件的相关操作得到的：只需知道文件的首尾地址（指针），即可获得图片的大小。

3.3 模块代码及解释

模块整体分四步完成，步与步之间有相应的调控。



图片转为 Base64 码

```
void base64_encode(char * bindata,char * base64,int binlength )
{
    int i, j;
    unsigned char current;
    for ( i = 0, j = 0 ; i < binlength ; i += 3 )
     {
          current = (bindata[i] >> 2) ;
          current &= (unsigned char)0x3F;
          base64[j++] = base64char[(int)current];

          current = ( (unsigned char)(bindata[i] << 4 ) ) & ( (unsigned char)0x30 ) ;
          if ( i + 1 >= binlength )
         {
               base64[j++] = base64char[(int)current];
               base64[j++] = '=';
               base64[j++] = '=';
               break;
          }
          current |= ( (unsigned char)(bindata[i+1] >> 4) ) & ( (unsigned char) 0x0F );
          base64[j++] = base64char[(int)current];
```

```
            current = ( (unsigned char)(bindata[i+1] << 2) ) & ( (unsigned char)0x3C ) ;
            if ( i + 2 >= binlength )
            {
                base64[j++] = base64char[(int)current];
                base64[j++] = '=';
                break;
            }
            current |= ( (unsigned char)(bindata[i+2] >> 6) ) & ( (unsigned char) 0x03 );
            base64[j++] = base64char[(int)current];

            current = ( (unsigned char)bindata[i+2] ) & ( (unsigned char)0x3F ) ;
            base64[j++] = base64char[(int)current];
        }
        base64[j] = '\0';

}
```

Base64 码转为二进制码

```
void base64tobit(char * base64,char * bit,int binlength)
{
  unsigned int test=1;
  test=test<<7;

  for(int i=0;i<binlength*2;i++){
        for(int j=0;j<8;j++){
              if(test&base64[i]){
                    bit[i*8+j]='1';
              }
              else{
                    bit[i*8+j]='0';
              }
              test=test>>1;

        }
        test=1<<7;
  }
}
```

二进制码转为 Base64 码

```
void bittobase64(char * base64,char * bit,int binlength)
{
  unsigned int test2=1<<7;
  for(int i=0;i<binlength*2;i++){
        base64[i]='0'-'0';
        for(int j=0;j<8;j++){
              if( bit[i*8+j] == '0' ){
                    test2 = test2 >> 1;
              }
              else{
                    base64[i]=base64[i]+(char)test2;
                    test2 = test2 >> 1;
              }
        }
        test2=1<<7;
  }
}
```

Base64 码转为图片

```
void base64_decode(char * base64,char * bindata )
{
      int i, j;
      unsigned char k;
      unsigned char temp[4];
      for ( i = 0, j = 0; base64[i] != '\0' ; i += 4 )
      {
            memset( temp, 0xFF, sizeof(temp) );
            for ( k = 0 ; k < 64 ; k ++ )
            {
                  if ( base64char[k] == base64[i] )
                        temp[0]= k;
            }
            for ( k = 0 ; k < 64 ; k ++ )
            {
                  if ( base64char[k] == base64[i+1] )
                        temp[1]= k;
            }
            for ( k = 0 ; k < 64 ; k ++ )
            {
                  if ( base64char[k] == base64[i+2] )
                        temp[2]= k;
            }
```

```
            for ( k = 0 ; k < 64 ; k ++ )
            {
                if ( base64char[k] == base64[i+3] )
                    temp[3]= k;
            }

            bindata[j++] = ((unsigned char)(((unsigned char)(temp[0] << 2))&0xFC)) |
                    ((unsigned char)((unsigned char)(temp[1]>>4)&0x03));
            if ( base64[i+2] == '=' )
                break;

            bindata[j++] = ((unsigned char)(((unsigned char)(temp[1] << 4))&0xF0)) |
                    ((unsigned char)((unsigned char)(temp[2]>>2)&0x0F));
            if ( base64[i+3] == '=' )
                break;

            bindata[j++] = ((unsigned char)(((unsigned char)(temp[2] << 6))&0xF0)) |
                    ((unsigned char)(temp[3]&0x3F));
        }
    }
    整体调控
    int main()
    {
     FILE *fp = NULL;
     unsigned int imageSize,imageSize2,bitlong,bitlong2;
     char *imageBin,*imageOutput;
        char *imageBase64,*imageBase64_2;
        char *imagebit,*imagebit_2;

     //读图，输入为图片，输出为图片大小 imageSize 和图片原码 imageBin
     fp = fopen("C:\\Users\\Emptiness\\Desktop\\123.jpg","rb");    //待编码图片地
址
     fseek(fp, 0L, SEEK_END);
     imageSize = ftell(fp);
     fseek(fp, 0L, SEEK_SET);
     imageBin = (char *)malloc(sizeof(char)*imageSize);
     fread(imageBin, 1, imageSize, fp);
        fclose(fp);

        //编码，输入为图片大小 imageSize 和图片原码 imageBin，输出为图片二进
制码 imagebit
        imageBase64 = (char *)malloc(sizeof(char)*imageSize*2);
     imagebit= (char *)malloc(sizeof(char)*imageSize*16);
        base64_encode(imageBin, imageBase64, imageSize);
```

```
    base64tobit(imageBase64,imagebit,imageSize);
    bitlong=strlen(imagebit);

//编帧，输入为图片大小 imageSize，图片二进制码 imagebit，输出为发送帧
char pro[500][3168];
int dest=1,sour=1;
Input(imagebit,imageSize,bitlong, dest, sour, pro);

//解帧，输入为接收帧，输出为图片大小 imageSize2，图片二进制码 imagebit_2
imageSize2=Output_imagelong(imageSize2, pro);
bitlong2=Output_bitlong(bitlong2, pro);
imagebit_2 = (char *)malloc(sizeof(char)*imageSize2*16);
printf("%d",imageSize2);
Output(imagebit_2,bitlong2, pro);

    //解码，输入为图片大小 imageSize2 和图片二进制码 imagebit_2，输出为图片原码 imageOutput
imageOutput = (char *)malloc(sizeof(char)*imageSize2);
imageBase64_2 = (char *)malloc(sizeof(char)*imageSize2*2);
bittobase64(imageBase64_2,imagebit_2,imageSize2);
    base64_decode(imageBase64_2, imageOutput);

//写图，输入为图片大小 imageSize 和图片原码 imageOutput，输出为图片
fp = fopen("C:\\Users\\Emptiness\\Desktop\\789.jpg","wb");    //待写入图片地址
    fwrite(imageOutput, 1, imageSize2, fp);
    fclose(fp);

free(imageOutput);
      free(imageBase64_2);
      free(imagebit_2);
      free(imagebit);
      free(imageBin);
free(imageBase64);

return 0;
}
```

3.4 模块亮点

采用 Base64 码，该码技术成熟，应用广泛，有效地保证了 图片传输过程中地不失真。

采用动态分配内存，有效地节省了堆栈空间，提高了处理速度。

4. 帧处理

4.1 模块名称或实现功能

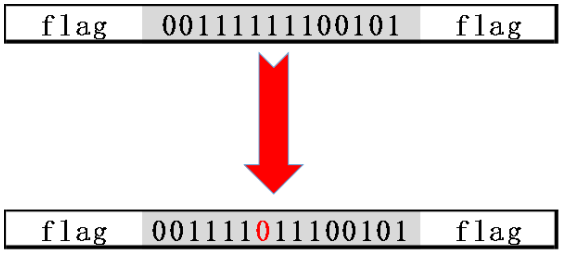
1、数据分块；2、比特填充的字符界定法成帧；3、帧定位

4.2 模块所涉及知识

数据分块

对输入的数据进行数组分组。

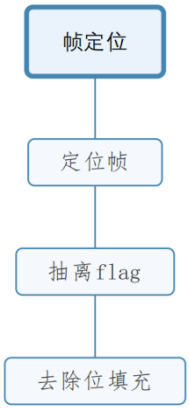
比特填充的字符界定法成帧



帧定位

目的：收端接收到之后，读到帧部分后，每读到 5 个连续的 1 之后，就把后面的 0 去掉一个，这样就得到了原文的数据了。

找到 flag-->去除帧头帧尾-->若发现剩余部分有连续 5 个 1,则去除后面加上的那一个 0



算法：定位到两个 flag（可认定是帧头帧尾）-->抽出帧头帧尾之间的数据-->检测到 5 个 1 则去除后面加的 0（去除位填充）

4.3 模块代码及解释

数据分块

```
int Input(char *ori, int imagelong, int bitlong, int dest, int sour, char pro[][3168]) //数据，图片大小，目的，源，输出分块数组
{
  int i, j, m = 1;
  int l;
  l = imagelong * 16;//l 为 ori 长度
```

```
  char source[2];
  char desti[2];
  char imagesize[24], bitsize[24];
  binlengthtobin(imagelong, imagesize);
  binlengthtobin(bitlong, bitsize);
  acktobin(sour, source);
  acktobin(dest, desti);
  pro[0][0] = 0;
  pro[0][1] = desti[0];
  pro[0][2] = desti[1];
  pro[0][3] = 0;
  pro[0][4] = 0;
  pro[0][5] = source[0];
  pro[0][6] = source[1];
  pro[0][7] = 0;

  for (int k = 0; k<24; k++){
        pro[0][k + 8] = imagesize[k];
  }
  for (int k = 0; k<24; k++){
        pro[0][k + 32] = bitsize[k];
  }
  for (j = 0; j < l; j++) {
        pro[m][0] = 0;
        pro[m][1] = desti[0];
        pro[m][2] = desti[1];
        pro[m][3] = 0;
        pro[m][4] = 0;
        pro[m][5] = source[0];
        pro[m][6] = source[1];
        pro[m][7] = 0;

        for (i = 8; (i < 3168)&(j<l); i++){
              pro[m][i] = ori[j];

              j++;
        }
        j--;
        m++;
  }
  return m;
}
```

比特填充的字符界定法成帧

```
void make_frame(char crc_data[], char frame[]) {
```

```
    int l, m, n, i;
    l = length(crc_data, 100);
    char a[8] = { 0, 1, 1, 1, 1, 1, 1, 0 };
    for (i = 0; i < 8; i++) {
        frame[i] = a[i];
    }
    for (i = 0, m = 0; i < l; i++) {
        if ((i>4) && (frame[i + 3 + m] == 1) && (frame[i + 4 + m] == 1) && (frame[i +
5 + m] == 1) && (frame[i + 6 + m] == 1) && (frame[i + 7 + m] == 1)) {
            frame[i + 8 + m] = 0;
            m++;
        }
        frame[i + 8 + m] = crc_data[i];
    }
    for (i = 0; i < 8; i++) {
        frame[i + 8 + m + l] = a[i];
    }
}
```

帧定位

```
int find(char rawdata[], char newdata[], int length, int* frame_length)
{
  int i, m, n, p, receival;
  p = 0;
  for (i = 0, n = 0, m = 0; i < 3160; i++) {
        if (rawdata[i] == 0 && rawdata[i + 1] == 1 && rawdata[i + 2] == 1 && rawdata[i
+ 3] == 1 && rawdata[i + 4] == 1 && rawdata[i + 5] == 1 && rawdata[i + 6] == 1 &&
rawdata[i + 7] == 0) {
            if (p == 0) {
                m = i;
                p = 1;
            }
            else if (p == 1) {
                n = i;
                p = 2;
            }
            else p = 3;
        }
    }
    if (m != 0 && n != 0 && p != 3) {
        for (i = 0; i < n - m + 8; i++) {
            newdata[i] = rawdata[i + m];
        }
        receival = 1;
        *frame_length = n - m + 8;
```

```
    }
    else receival = 0;

    return receival;//返回是否读到有效帧
}
```

4.4 模块亮点

比特填充的字符界定法相对与字填充方法提升了有效数据率
帧定位算法使用 flag 个数作为分界标志，结构简洁

5. 网络层和应用层的整体实现

网络层和应用层模块代码及解释由于过长放在附录并以记事本文件形式提交。

四. 心得与收获

1.在完成路由函数的过程中，熟悉了建立路由表的距离矢量算法。与运筹学的最优化问题同理，实现跨学科的理解，得到了思想的提升。

2.刚开始动手做时没有参考书上的路由表结构模型。自己在探索的过程中遇到了很多问题，如无法通过相邻路由发来的内容判断数据根源是哪里，很容易陷入死循环中。悟懂书上的路由表结构后，很好的理解了这样设置的原因，并以此来改进，最后进行简化得到了现在的路由计算函数。

3.本来一开始做这个时候，是想直接把文件转码为二进制，这样做的确可以传文件，但是传出地相似而不相同，用 txt 查看，也就是多了几个空格而已，但是计算机认不出这是图片了。后来查了很多资料，偶然翻到教材后面，看到了 Base64 编码，才了解到人们早就为此所扰，并创立了 Base64 码来解决。所以我们完成任何一个项目，都不是从 0 开始，都要汲取前人地智慧，多读书，多了解。

4.在实际的工程项目中，常常是将各个模块进行拆分，就像此项目中的各个辅助函数与主函数，如果我们在交接功能函数时没有进行严格的对接要求检查，那么对于汇总的人员来说是一种极其繁琐的过程。在我们做此项目的过程中，有些组员没有严格按照主操作人员的要求进行数组大小，数据类型等的初始化，这就使那一位同学的工作量增加了数倍。而这位同学又要将要求进行重新说明，并告知负责该函数的同学。最后，整个小组的任务量会无形的增大。这就要求我们在以后的项目中，将各个部分的要求进行尽可能详细的声名，以尽量避免我们这次出现的情况。

五.附录

1. 网络层代码及解释

```
// server.cpp :
//

#ifndef _CRT_SECURE_NO_WARNINGS

#define _CRT_SECURE_NO_WARNINGS
```

```
#endif

#define feclong 6


#define framelong_send 3173
#define datalong_send 3168


#define framelong_receive 3173
#define datalong_receive 3168
#define length_A 3500


#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <sys/types.h>

#include <winsock2.h>

#include <WS2tcpip.h>

#include <iostream>

#include "winsock.h"

#include "conio.h"

#include <fcntl.h>

typedef unsigned int uint;
char p[feclong] = { 1, 0, 0, 1, 1, 1 };//CRC 用
#pragma  comment(lib,"ws2_32.lib")//静态库
const  char  *  base64char  =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

void binlengthtobin(int binlength, char imagesize[])//将 binlength 转为二进制
```

```
{
  unsigned long test;
  int i;
  test = 1;
  test = test << 23;
  for (i = 0; i < 24; i++) {
        if (test & binlength) {
              imagesize[i] = 1;
        }
        else {
              imagesize[i] = 0;
        }
        test = test >> 1;
  }
}
void base64_encode(char * bindata, char * base64, int binlength)
{
  int i, j;
  unsigned char current;
  for (i = 0, j = 0; i < binlength; i += 3)
  {
        current = (bindata[i] >> 2);
        current &= (unsigned char)0x3F;
        base64[j++] = base64char[(int)current];

        current = ((unsigned char)(bindata[i] << 4)) & ((unsigned char)0x30);
        if (i + 1 >= binlength)
        {
              base64[j++] = base64char[(int)current];
              base64[j++] = '=';
              base64[j++] = '=';
              break;
        }
        current |= ((unsigned char)(bindata[i + 1] >> 4)) & ((unsigned char)0x0F);
        base64[j++] = base64char[(int)current];

        current = ((unsigned char)(bindata[i + 1] << 2)) & ((unsigned char)0x3C);
        if (i + 2 >= binlength)
        {
              base64[j++] = base64char[(int)current];
              base64[j++] = '=';
              break;
        }
        current |= ((unsigned char)(bindata[i + 2] >> 6)) & ((unsigned char)0x03);
```

```
        base64[j++] = base64char[(int)current];

        current = ((unsigned char)bindata[i + 2]) & ((unsigned char)0x3F);
        base64[j++] = base64char[(int)current];
  }
  base64[j] = '\0';

}
int base64tobit(char * base64, char * bit, int binlength)
{
  unsigned int test = 1;
  test = test << 7;
  int bitlong = 0;
  for (int i = 0; i<binlength * 2; i++) {
        for (int j = 0; j<8; j++) {
              if (test&base64[i]) {
                    bit[i * 8 + j] = 1;
              }
              else {
                    bit[i * 8 + j] = 0;
              }
              bitlong = i * 8 + j + 1;
              test = test >> 1;

        }
        test = 1 << 7;

  }
  return bitlong;
}
void base64_decode(char * base64, char * bindata)
{
  int i, j;
  unsigned char k;
  unsigned char temp[4];
  for (i = 0, j = 0; base64[i] != '\0'; i += 4)
  {
        memset(temp, 0xFF, sizeof(temp));
        for (k = 0; k < 64; k++)
        {
              if (base64char[k] == base64[i])
                    temp[0] = k;
        }
        for (k = 0; k < 64; k++)
```

```
        {
            if (base64char[k] == base64[i + 1])
                temp[1] = k;
        }
        for (k = 0; k < 64; k++)
        {
            if (base64char[k] == base64[i + 2])
                temp[2] = k;
        }
        for (k = 0; k < 64; k++)
        {
            if (base64char[k] == base64[i + 3])
                temp[3] = k;
        }

        bindata[j++] = ((unsigned char)(((unsigned char)(temp[0] << 2)) & 0xFC)) |
            ((unsigned char)((unsigned char)(temp[1] >> 4) & 0x03));
        if (base64[i + 2] == '=')
            break;

        bindata[j++] = ((unsigned char)(((unsigned char)(temp[1] << 4)) & 0xF0)) |
            ((unsigned char)((unsigned char)(temp[2] >> 2) & 0x0F));
        if (base64[i + 3] == '=')
            break;

        bindata[j++] = ((unsigned char)(((unsigned char)(temp[2] << 6)) & 0xF0)) |
            ((unsigned char)(temp[3] & 0x3F));
    }
}
void bittobase64(char * base64, char * bit, int binlength)
{
    unsigned int test2 = 1 << 7;
    for (int i = 0; i<binlength * 2; i++) {
        base64[i] = '0' - '0';
        for (int j = 0; j<8; j++) {
            if (bit[i * 8 + j] == 0) {
                test2 = test2 >> 1;
            }
            else {
                base64[i] = base64[i] + (char)test2;
                test2 = test2 >> 1;
            }
        }
        test2 = 1 << 7;
```

```
  }
}

int Output_imagelong(char pro[3168]) //图片大小，输入分块数组
{
  int imagelong = 0;
  char imagesize[24];
  for (int k = 0; k<24; k++) {
        imagesize[k] = pro[k + 8];
  }
  unsigned int test = 1 << 23;
  for (int k = 0; k<23; k++) {
        if (imagesize[k] == 1) {
              imagelong = (test | 0) + imagelong;
        }
        test = test >> 1;
  }
  return imagelong;
}
int Output_bitlong(char pro[3168]) //图片大小，输入分块数组
{
  int bitlong = 0;
  char bitsize[24];
  for (int k = 0; k<24; k++) {
        bitsize[k] = pro[k + 32];
  }
  unsigned int test = 1 << 23;
  for (int k = 0; k<23; k++) {
        if (bitsize[k] == 1) {
              bitlong = (test | 0) + bitlong;
        }
        test = test >> 1;
  }
  return bitlong;
}
void Output(char *ori, int bitlong, char pro[][3168]) //数据，输入图片大小，分块数
组
{
  int j = 1;
  for (int k = 0; k<bitlong;) {

        for (int i = 8; i<3168; i++) {
              ori[k] = pro[j][(k % 3160) + 8];
              k++;
```

```
        }
        j++;

  }

}

void Add(char array_A[], char array_B[], int flag)
{
  int i;
  array_B[0] = flag;
  for (i = 0; i < length_A; i++)
  {
        array_B[i + 1] = array_A[i];
  }

}
void Sub(char B[], char A[])
{
  int i = 0;
  for (i; i < length_A; i++)
  {
        A[i] = B[i + 1];
  }
}

void acktobin(int ack, char Ack[])//将 ack 转为二进制
{
  unsigned long test;
  int i;
  test = 1;
  test = test << 1;
  for (i = 0; i < 2; i++) {
        if (test & ack) {
              Ack[i] = 1;
        }
        else {
              Ack[i] = 0;
        }
        test = test >> 1;
  }

}
int length(char a[], int maxlength) {
```

```
  int i, m;
  for (i = 0, m = 0; i < maxlength; i++) {
        if ((a[i] != 0) && (a[i] != 1)) {
              break;
        }
        else if (a[i] == '/0') {
              break;
        }
        else {
              m++;
        }
  }
  return m;
}//计算数组已填充长度

void outkernel(char raw[], char b[]) {
  int len, m;
  len = length(raw, 1100);
  m = 0;
  for (int i = 0; i < len; i++) {
        if (i % 32 < 8) {
              m++;
        }
        else b[i - m] = raw[i];
  }
}//去除 seqack
void tranlate(char rec[], char output[]) {
  int test, i;
  char yong[800];
  outkernel(rec, yong);
  int len = length(output, 150);
  for (int j = 0; j < 100; j++) {
        output[j + len] = 0;
        for (i = 0; i < 8; i++) {
              test = 1;
              test = test << (7 - i);
              test = test | output[j + len];
              if (yong[i + 8 * j] == 1) {
                    output[j + len] = output[j + len] | test;
              }
              else {
                    output[j + len] = output[j + len] & test;
              }
        }
```

```
  }
}//解码
void CRC(char data[], char crc_data[], char p[])
{

  char sub[feclong];
  char data_long[framelong_send];

  for (int i = 0; i < datalong_send; i++)
  {
        data_long[i] = data[i];
  }

  for (int i = datalong_send; i < framelong_send; i++)
  {
        data_long[i] = 0;
  }
  for (int i = 0; i < feclong; i++)
  {
        sub[i] = data_long[i];
  }


  for (int i = 0; i < datalong_send; i++)
  {

        for (int j = 1; j < feclong; j++)
        {
              if (sub[j] == p[j])
              {
                    sub[j - 1] = 0;
              }
              else
              {
                    sub[j - 1] = 1;
              }
        }
        sub[feclong - 1] = data_long[i + feclong];

        while (sub[0] == 0 && i< (datalong_send - 1))
        {

              for (int k = 0; k < (feclong - 1); k++)
```

```
            {
                sub[k] = sub[k + 1];
            }
            i++;
            sub[(feclong - 1)] = data_long[i + feclong];
        }
    }




    for (int i = 0; i < datalong_send; i++)
        crc_data[i] = data[i];

    for (int i = datalong_send; i < framelong_send; i++)
        crc_data[i] = sub[i - datalong_send];




}
int check_CRC(char data[], char crc_data[], char p[])
{

    char sub[feclong];
    for (int i = 0; i < feclong; i++)
        sub[i] = crc_data[i];
    char data_long[framelong_receive];
    for (int i = 0; i < datalong_receive; i++)
        data_long[i] = data[i];
    for (int i = datalong_receive; i < framelong_receive; i++)
        data_long[i] = 0;

    for (int i = 0; i < datalong_receive; i++)
    {

        for (int j = 1; j < feclong; j++)
        {
            if (sub[j] == p[j])
                sub[j - 1] = 0;
            else sub[j - 1] = 1;

        }
```

```
        sub[feclong - 1] = crc_data[i + feclong];

        while (sub[0] == 0 && i < (datalong_receive - 1))
        {
            if (sub[0] == 0)
            {
                for (int k = 0; k < (feclong - 1); k++)
                    sub[k] = sub[k + 1];
                i++;
                sub[(feclong - 1)] = crc_data[i + feclong];
            }
        }
    }
    int check = 0;
    for (int i = 0; i < feclong - 1; i++)
        check = check + sub[i];
    if (check == 0)
    {
        for (int i = 0; i < datalong_receive; i++)
        {
            data[i] = crc_data[i];
        }
        return 0;
    }
    else
        return 1;
}
int length_Input(char a[], int maxlength) {
    int i, m;
    for (i = 0, m = 0; i < maxlength; i++) {
        if (a[i] == 0) {
            break;
        }
        else {
            m++;
        }
    }
    return m;
}//计算数组已填充长度
int check(char *frame, char restore_data[], int l)//l 为 frame 的长度，可用 sizeof
{
    //先去帧
    //再去填充
    //再 CRC
```

```
    int i = 0;
    int m = 0;
    for (i = 0; i < l - 16 - m; i++)
    {
        if ((i>4) && (frame[i + 7 + m] == 1) && (frame[i + 6 + m] == 1) && (frame[i +
5 + m] == 1) && (frame[i + 4 + m] == 1) && (frame[i + 3 + m] == 1))
        {
            m++;
        }
        restore_data[i] = frame[i + 8 + m];
        //          printf("%d", restore_data[i]);
    }
    //去 CRC
    if (check_CRC(restore_data, restore_data, p) == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
    //正确返回 1,正确数据就是 restore_data

    //反之返回 0
}
int event(char newdata[], char restore_data[], int receival, int length)
{
    int back = -1;
    if (receival == 1)
    {
        if (check(newdata, restore_data, length))
        {
            back = 0;
        }
        else {
            back = 1;
        }
    }
    else { back = 1; }
    return back;
}

void make_frame(char crc_data[], char frame[]) {
    int l, m, n, i;
```

```
    l = framelong_send;
    char a[8] = { 0, 1, 1, 1, 1, 1, 1, 0 };
    for (i = 0; i < 8; i++) {
        frame[i] = a[i];
    }
    for (i = 0, m = 0; i < l; i++) {
        if ((i>4) && (frame[i + 3 + m] == 1) && (frame[i + 4 + m] == 1) && (frame[i +
5 + m] == 1) && (frame[i + 6 + m] == 1) && (frame[i + 7 + m] == 1)) {
            frame[i + 8 + m] = 0;
            m++;
        }
        frame[i + 8 + m] = crc_data[i];
    }
    for (i = 0; i < 8; i++) {
        frame[i + 8 + m + l] = a[i];
    }
}
void prepare_ack(char frame[], int *ack)//发送
{
    char Ack[2];
    char crc_ack[framelong_send] = { 0 };
    acktobin(*ack, Ack);
    CRC(Ack, crc_ack, p);
    make_frame(crc_ack, frame);
}
void uncode_goal(char restore_data[], int* seq)
{
    if (restore_data[1] == 0 && restore_data[2] == 0) {
        *seq = 0;
    }
    else if (restore_data[1] == 0 && restore_data[2] == 1) {
        *seq = 1;
    }
    else if (restore_data[1] == 1 && restore_data[2] == 0) {
        *seq = 2;
    }
    else if (restore_data[1] == 1 && restore_data[2] == 1) {
        *seq = 3;
    }
}
void uncode(char restore_data[], int* seq)
{

    /*    if (restore_data[0] == 0 && restore_data[1] == 0) {
```

```
*seq = 0;
}
else if (restore_data[0] == 0 && restore_data[1] == 1) {
*seq = 1;
}
else if (restore_data[0] == 1 && restore_data[1] == 0) {
*seq = 2;
}
else if (restore_data[0] == 1 && restore_data[1] == 1) {
*seq = 3;
}
*/
/*  if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0) {
*seq = 8;
}*/
/*
if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0) {
*seq = 0;
}
if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1) {
*seq = 1;
}
if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0) {
*seq = 2;
}
if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1) {
*seq = 3;
}
if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0) {
*seq = 4;
}
if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1) {
*seq = 5;
}
if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0) {
*seq = 6;
```

```
        }
        if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1) {
        *seq = 7;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0) {
        *seq = 8;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1) {
        *seq = 9;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0) {
        *seq = 10;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1) {
        *seq = 11;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0) {
        *seq = 12;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1) {
        *seq = 13;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0) {
        *seq = 14;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1) {
        *seq = 15;
        }
        */
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 0) {
                *seq = 0;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 1) {
```

```
            *seq = 1;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 0) {
            *seq = 2;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 1) {
            *seq = 3;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 0) {
            *seq = 4;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 1) {
            *seq = 5;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 0) {
            *seq = 6;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 1) {
            *seq = 7;
        }
        if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 0) {
            *seq = 8;
        }
        if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 1) {
            *seq = 9;
        }
        if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 0) {
            *seq = 10;
        }
        if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 1) {
            *seq = 11;
        }
        if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 0) {
```

```
            *seq = 12;
        }
        if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 1) {
            *seq = 13;
        }
        if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 0) {
            *seq = 14;
        }
        if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 1) {
            *seq = 15;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 0) {
            *seq = 16;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 1) {
            *seq = 17;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 0) {
            *seq = 18;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 1) {
            *seq = 19;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 0) {
            *seq = 20;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 1) {
            *seq = 21;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 0) {
            *seq = 22;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 1) {
```

```
            *seq = 23;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 0) {
            *seq = 24;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 1) {
            *seq = 25;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 0) {
            *seq = 26;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 1) {
            *seq = 27;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 0) {
            *seq = 28;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 1) {
            *seq = 29;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 0) {
            *seq = 30;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 1) {
            *seq = 31;
        }
    }

    int find(char rawdata[], char newdata[], int length, int* frame_length)
    {
      int i, m, n, p, receival;
      p = 0;
      for (i = 0, n = 0, m = 0; i < 3592; i++) {
            if (rawdata[i] == 0 && rawdata[i + 1] == 1 && rawdata[i + 2] == 1 && rawdata[i
+ 3] == 1 && rawdata[i + 4] == 1 && rawdata[i + 5] == 1 && rawdata[i + 6] == 1 &&
rawdata[i + 7] == 0) {
```

```
                if (p == 0) {
                    m = i;
                    p = 1;
                }
                else if (p == 1) {
                    n = i;
                    p = 2;
                }
                else p = 3;
            }
        }
        if (n != 0 && p != 3)
        {
            for (i = 0; i < n - m + 8; i++) {
                newdata[i] = rawdata[i + m];
            }
            receival = 1;
            *frame_length = n - m + 8;
        }
        else receival = 0;

        return receival;//返回是否读到有效帧
}
int uncode_send(char restore_data[], int seq, int ack)
{
        //解码解出 seq,ack 再判断

        if (seq == ack)
        {
            return 1;
        }
        else
        {
            return 0;
        }
}


/*****************************************************************/
struct socket_list//socket 数组
{
        SOCKET MainSock;
        unsigned short MainPort;//应用层的端口号
        int num; //接口数量
```

```
  SOCKET sock_array[64];
  unsigned short port_array[64];//各接口模块的端口号
};
void init_list(socket_list *list)//初始化 socket 数组
{
  int i;
  list->MainSock = 0;
  list->num = 0;
  for (i = 0; i < 64; i++) {
        list->sock_array[i] = 0;
  }
}
void insert_list(SOCKET s, unsigned short port, socket_list *list)//将指定套接字放入
socket 数组空位置
{
  int i;
  for (i = 0; i < 64; i++) {
        if (list->sock_array[i] == 0) {
              list->sock_array[i] = s;
              list->port_array[i] = port;
              list->num += 1;
              break;
        }
  }
}
void delete_list(SOCKET s, socket_list *list)//从 socket 数组中删除指令套接字
{
  int i;
  for (i = 0; i < 64; i++) {
        if (list->sock_array[i] == s) {
              list->sock_array[i] = 0;
              list->num -= 1;
              break;
        }
  }
}
void make_fdlist(socket_list *list, fd_set *fd_list)//将 socket 数组放入 fd 数组中，准
备进入 select
{
  int i;
  FD_SET(list->MainSock, fd_list);
  for (i = 0; i < 64; i++) {
        if (list->sock_array[i] > 0) {
              FD_SET(list->sock_array[i], fd_list);
```

```
        }
    }
}
void code(unsigned long x, char A[], int length)
{
    unsigned long test;
    int i;
    x = x << (length - 5);
    test = 1;
    test = test << (length - 1);
    printf("this is");
    for (i = 0; i < length; i++) {
        if (test & x) {
            A[i] = 1;
        }
        else {
            A[i] = 0;
        }
        printf("%d", A[i]);
        test = test >> 1; //本算法利用了移位操作和"与"计算，逐位测出 x 的每一
位是 0 还是 1.
    }
    printf("\n");
}
unsigned long decode(char A[], int length)//解码函数
{
    unsigned long x;
    int i;

    x = 0;
    for (i = 0; i < length; i++) {
        if (A[i] == 0) {
            x = x << 1;;
        }
        else {
            x = x << 1;
            x = x | 1;
        }
    }
    return x;
}
int Input(char *ori, int imagelong, int bitlong, int dest, int sour, char pro[][3168]) //数
据，图片大小，目的，源，输出分块数组
{
```

```
int i, j, m = 1;
int l;
l = imagelong * 16;//l 为 ori 长度

char source[2];
char desti[2];
char imagesize[24], bitsize[24];
binlengthtobin(imagelong, imagesize);
binlengthtobin(bitlong, bitsize);
acktobin(sour, source);
acktobin(dest, desti);
pro[0][0] = 0;
pro[0][1] = desti[0];
pro[0][2] = desti[1];
pro[0][3] = 0;
pro[0][4] = 0;
pro[0][5] = source[0];
pro[0][6] = source[1];
pro[0][7] = 0;

for (int k = 0; k<24; k++) {
	pro[0][k + 8] = imagesize[k];
}
for (int k = 0; k<24; k++) {
	pro[0][k + 32] = bitsize[k];
}



for (j = 0; j < l; j++) {
	pro[m][0] = 0;
	pro[m][1] = desti[0];
	pro[m][2] = desti[1];
	pro[m][3] = 0;
	pro[m][4] = 0;
	pro[m][5] = source[0];
	pro[m][6] = source[1];
	pro[m][7] = 0;

	for (i = 8; (i < 3168)&(j<l); i++) {
		pro[m][i] = ori[j];

		j++;
```

```
        }
        j--;
        m++;
  }
  m--;
  return m;
}
#define MAX_BUFFSIZE 3600
#define SEND_L 3500
/*********************************************************************/
int main(int argc, char* argv[])
{
  SOCKET s, sock;//创建两个套接字
  struct sockaddr_in remote_addr;//一个地址
  int len;
  char buf[MAX_BUFFSIZE] = { 0 };//接收缓存
  char sendbuf[SEND_L] = { 1, 0, 0, 1, 0 };//测试数据
  WSAData wsa;
  int retval, selretval;
  struct socket_list sock_list;//创建 socket 数组
  //准备 select
  fd_set readfds, writefds, exceptfds;
  timeval timeout;

  int intf, i;
  unsigned long arg;
  int linecount = 0;
  unsigned short* port;
  int portNum;
  int ID;
  unsigned short myPort;
  int portLen;

  WSAStartup(0x101, &wsa);


  //buf = (char *)malloc(MAX_BUFFSIZE);//动态创建 4000 缓存区
  //sendbuf = (char *)malloc(MAX_BUFFSIZE);//动态创建 4000 发送区

  init_list(&sock_list);//初始化 socket 数组
  //从键盘输入，需要连接的 API 端口号
  printf("本模块是网络层,\n");
  printf("请输入本模块的设备编号：");//主要用来区分不同的设备
  scanf_s("%d", &ID);
```

```
s = socket(AF_INET, SOCK_DGRAM, 0);
if (s == SOCKET_ERROR) {
	return 0;
}
//设置应用层端口号，以向应用层发送数据
printf("请输入应用层模块的端口号：");
scanf_s("%d", &sock_list.MainPort);

//绑定本模块端口号，以接收应用层数据
printf("请设置本模块的端口号：");
scanf_s("%d", &myPort);
remote_addr.sin_family = AF_INET;
remote_addr.sin_port = htons(myPort);
remote_addr.sin_addr.S_un.S_addr = htonl(INADDR_LOOPBACK);
while (bind(s, (sockaddr*)&remote_addr, sizeof(remote_addr)) != 0) {
	printf("本模块的端口号已被占用，请换一个再试:");
	scanf_s("%d", &myPort);
	remote_addr.sin_port = htons(myPort);
}
sock_list.MainSock = s;

//设置物理层接口端口号，以向接口发送数据
printf("请输入接口模块的数量：");
scanf_s("%d", &portNum);
port = (unsigned short *)malloc(portNum * sizeof(unsigned short));
for (i = 0; i < portNum; i++) {
	printf("请输入接口%d 的 api 端口号(从物理层接口模块软件上获取): ", i);
	scanf_s("%d", &(port[i]));
	//可以生成多个套接字，每个套接字对应一个接口模块
	s = socket(AF_INET, SOCK_DGRAM, 0);
	if (s == SOCKET_ERROR) {
		return 0;
	}

	remote_addr.sin_family = AF_INET;
	remote_addr.sin_addr.S_un.S_addr = htonl(INADDR_LOOPBACK);
	remote_addr.sin_port = htons(port[i]);
	//向 api 接口发送一个测试命令
	sendto(s, "connect", 8, 0, (sockaddr*)&remote_addr, sizeof(remote_addr));
	len = sizeof(remote_addr);
	retval = recvfrom(s, buf, MAX_BUFFSIZE, 0, (sockaddr*)&remote_addr, &len);
	if (retval == SOCKET_ERROR) {
		printf("接口 API 关联失败\n");
```

```
            return 0;
        }
        //有接收，就可以认为关联成功
        printf("接口 API 连接成功！\n");
        //设置套接字的状态为阻塞态，也可以不设置
        arg = 1;
        ioctlsocket(s, FIONBIO, &arg);
        //把 s 放入套接字队列中，今后该套接字在 sock_list 里的下标可以用于表
示接口的编号
        insert_list(s, port[i], &sock_list);
    }

    timeout.tv_sec = 2; //设置发送数据的时间间隔为 10 微秒
    timeout.tv_usec = 0;
    FD_ZERO(&readfds);
    FD_ZERO(&writefds);
    FD_ZERO(&exceptfds);

    /*    for(i = 0 ; i < 4000; i ++){
    sendbuf[i] = 1; //初始化数据全为 1,以备测试
    }
    */
    /*************************/
    /*********************************/
    char receive_endline[MAX_BUFFSIZE] = { 0 };
    char restore_data[SEND_L] = { 0 };
    int back = -1;
    char newdata[SEND_L] = { 0 };
    int receival = 0;
    int length1 = 0;
    int ACK = 0;
    /*********************************/
    char Sendbuf[MAX_BUFFSIZE] = { 0 };
    int _length = SEND_L;
    char rawdata[MAX_BUFFSIZE] = { 0 };
    int Ack = 0;
    char send_ack[64] = { 0 };
    int k = 1;
    int K = 1;
    int count = -1;
    int endline = 0;
    int flag;
    int host = 0;
    int intf0 = 0;
```

```
int intf1 = 0;
int Eng = 0;
char blank_buf[MAX_BUFFSIZE] = { 0 };
int rec_Imag_length = 0;
int rec_bit_length = 0;
/*************************/
/*****************************/
int Host;
printf("\n 请输入本地主机号:");
scanf_s("%d", &Host);
printf("\n");
int route_choice = 0;
int route_recv = 0;
//int flag = 0;
/*****************************/
while (1)
{
    make_fdlist(&sock_list, &readfds);
    //本例程采用了基于 select 机制，不断发送测试数据，和接收测试数据，也可以采用多线程，一线专发送，一线专接收的方案
    selretval = select(0, &readfds, &writefds, &exceptfds, &timeout);
    if (selretval == SOCKET_ERROR)
    {
        retval = WSAGetLastError();
        break;
    }
    else if (selretval == 0)
    {
        //定时器触发
        continue;
    }
    if (FD_ISSET(sock_list.MainSock, &readfds))
    {
        //从应用层收到数据
        portLen = sizeof(remote_addr);
        len = sizeof(remote_addr);
        retval = recvfrom(sock_list.MainSock, buf, MAX_BUFFSIZE, 0, (sockaddr*)&remote_addr, &len);

        //  retval = recvfrom(sock_list.MainSock,buf,MAX_BUFFSIZE,0,(sockaddr*)&remote_addr,&portLen);
        if (retval > 0) {
            //将数据编码为"比特流"形式
            /*for (i = 0; i < retval; i++){
```

```
                    //每次编码 8 位
                    //code(buf[i], sendbuf + i * 8, 8);
                    }*/
                    //判断应用层发来数据的第 0 位
                    if (route_choice == 0)
                    {
                        remote_addr.sin_port = htons(sock_list.port_array[0]);
                        retval = sendto(sock_list.sock_array[0], buf, _length, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                        while (k)
                        {
                            FD_ZERO(&readfds);
                            FD_ZERO(&writefds);
                            FD_ZERO(&exceptfds);
                            remote_addr.sin_port = htons(sock_list.port_array[0]);
                            make_fdlist(&sock_list, &readfds);
                            selretval = select(0, &readfds, &writefds, &exceptfds,
&timeout);
                            if (selretval == SOCKET_ERROR)
                            {
                                retval = WSAGetLastError();
                                break;
                            }
                            else if (selretval == 0)
                            {
                                //定时器触发
                                sendto(sock_list.sock_array[0], buf, _length, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                            }
                            if (FD_ISSET(sock_list.sock_array[0], &readfds))
                            {
                                retval = recv(sock_list.sock_array[0], rawdata,
MAX_BUFFSIZE, 0);
                                if (retval > 0)
                                {

                                    printf("初始路由表已经发送到 0 号接口\n");
                                    k = 0;
                                }
                                else if (retval == 0)
                                {
                                    system("pause");
                                }
                            }
```

```
                    }
                    k = 1;
                    remote_addr.sin_port = htons(sock_list.port_array[1]);
                    retval = sendto(sock_list.sock_array[1], buf, _length, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                    while (k)
                    {
                        FD_ZERO(&readfds);
                        FD_ZERO(&writefds);
                        FD_ZERO(&exceptfds);
                        remote_addr.sin_port = htons(sock_list.port_array[1]);
                        make_fdlist(&sock_list, &readfds);
                        selretval = select(0, &readfds, &writefds, &exceptfds,
&timeout);
                        if (selretval == SOCKET_ERROR)
                        {
                            retval = WSAGetLastError();
                            break;
                        }
                        else if (selretval == 0)
                        {
                            //定时器触发
                            sendto(sock_list.sock_array[1], buf, _length, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                        }
                        if (FD_ISSET(sock_list.sock_array[1], &readfds))
                        {
                            retval = recv(sock_list.sock_array[1], rawdata,
MAX_BUFFSIZE, 0);
                            if (retval > 0)
                            {

                                printf("初始路由表已经发送到 1 号接口\n");
                                k = 0;
                            }
                            else if (retval == 0)
                            {
                                system("pause");
                            }
                        }
                    }
                    route_choice = 1;
                    k = 1;
```

```
                    }
                    else
                    {
                        if (buf[0] == 0)
                        {
                            Sub(buf, Sendbuf);
                            remote_addr.sin_port = htons(sock_list.port_array[0]);
                            retval = sendto(sock_list.sock_array[0], Sendbuf, _length, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                            while (k)
                            {
                                if (count == -1)
                                {
                                    memcpy(receive_endline,          Sendbuf,
MAX_BUFFSIZE);
                                    receival    =    find(receive_endline,    newdata,
SEND_L, &length1);
                                    back = event(newdata, restore_data, receival,
length1);
                                    if (back == 0)
                                    {
                                        //uncode(restore_data, &endline);
                                        rec_Imag_length                          =
Output_imagelong(restore_data);
                                        rec_bit_length                           =
Output_bitlong(restore_data);
                                        endline = rec_bit_length / 3160 + 1;
                                        printf("send endline %d", endline);
                                        printf("\n");
                                        count++;
                                    }
                                }
                                FD_ZERO(&readfds);
                                FD_ZERO(&writefds);
                                FD_ZERO(&exceptfds);
                                remote_addr.sin_port = htons(sock_list.port_array[0]);
                                make_fdlist(&sock_list, &readfds);
                                selretval = select(0, &readfds, &writefds, &exceptfds,
&timeout);
                                if (selretval == SOCKET_ERROR)
                                {
                                    retval = WSAGetLastError();
                                    break;
                                }
```

```
                                else if (selretval == 0)
                                {
                                    //定时器触发
                                    sendto(sock_list.sock_array[0], Sendbuf, _length, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                                }
                                if (FD_ISSET(sock_list.MainSock, &readfds))
                                {
                                    remote_addr.sin_port                    =
htons(sock_list.port_array[0]);

                                    len = sizeof(remote_addr);
                                    retval = recvfrom(sock_list.MainSock, buf,
MAX_BUFFSIZE, 0, (sockaddr*)&remote_addr, &len);
                                    Sub(buf, Sendbuf);
                                    remote_addr.sin_port                    =
htons(sock_list.port_array[0]);
                                    retval = sendto(sock_list.sock_array[0], Sendbuf,
_length, 0, (sockaddr*)&remote_addr, sizeof(remote_addr));

                                    endline--;


                                }
                                if (FD_ISSET(sock_list.sock_array[0], &readfds))
                                {
                                    retval = recv(sock_list.sock_array[0], rawdata,
MAX_BUFFSIZE, 0);
                                    if (retval > 0)
                                    {

                                        printf("将应用层数据已经发送到 0 号接
口\n");

                                        printf("还剩下 endline:%d", endline);
                                        //将 Ack 编码到 send_ack
                                        acktobin(Ack, send_ack);
                                        remote_addr.sin_port                    =
htons(sock_list.MainPort);
                                        sendto(sock_list.MainSock, send_ack, 64, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));

                                        if (endline == 0)
                                        {
                                            k = 0;
```

```
                                        count = -1;

                                    }

                                }
                                else if (retval == 0)
                                {
                                    system("pause");
                                }

                            }

                            /*
                            char sendbuf1[100] = { 1, 0, 0, 1, 0 };//测试数据
                            remote_addr.sin_port = htons(sock_list.port_array[0]);
                            sendto(sock_list.sock_array[0], sendbuf1, 5, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));

                            printf("将应用层数据发送到 0 号接口\n");
                            for (i == 0; i < 5; i++)
                            {
                            printf("%d", buf[i]);
                            }

                            */

                        }
                    }
                    else
                    {
                        Sub(buf, Sendbuf);
                        remote_addr.sin_port = htons(sock_list.port_array[1]);
                        retval = sendto(sock_list.sock_array[1], Sendbuf, _length, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                        while (k)
                        {
                            if (count == -1)
                            {
                                memcpy(receive_endline, Sendbuf,
MAX_BUFFSIZE);
                                receival = find(receive_endline, newdata,
SEND_L, &length1);
                                back = event(newdata, restore_data, receival,
length1);
```

```
                    if (back == 0)
                    {
                        //uncode(restore_data, &endline);
                        rec_Imag_length = Output_imagelong(restore_data);
                        rec_bit_length = Output_bitlong(restore_data);
                        endline = rec_bit_length / 3160 + 1;
                        printf("send endline %d", endline);
                        printf("\n");
                        count++;
                    }
                }
                FD_ZERO(&readfds);
                FD_ZERO(&writefds);
                FD_ZERO(&exceptfds);
                remote_addr.sin_port = htons(sock_list.port_array[1]);
                make_fdlist(&sock_list, &readfds);
                selretval = select(0, &readfds, &writefds, &exceptfds, &timeout);
                if (selretval == SOCKET_ERROR)
                {
                    retval = WSAGetLastError();
                    break;
                }
                else if (selretval == 0)
                {
                    //定时器触发
                    sendto(sock_list.sock_array[1], Sendbuf, _length, 0, (sockaddr*)&remote_addr, sizeof(remote_addr));
                }
                if (FD_ISSET(sock_list.MainSock, &readfds))
                {
                    remote_addr.sin_port = htons(sock_list.port_array[1]);

                    len = sizeof(remote_addr);
                    retval = recvfrom(sock_list.MainSock, buf, MAX_BUFFSIZE, 0, (sockaddr*)&remote_addr, &len);
                    Sub(buf, Sendbuf);
                    remote_addr.sin_port = htons(sock_list.port_array[1]);
                    retval = sendto(sock_list.sock_array[1], Sendbuf, _length, 0, (sockaddr*)&remote_addr, sizeof(remote_addr));
```

```
                endline--;


            }
            if (FD_ISSET(sock_list.sock_array[1], &readfds))
            {
                retval = recv(sock_list.sock_array[1], rawdata,
MAX_BUFFSIZE, 0);
                if (retval > 0)
                {

                    printf("将应用层数据已经发送到 0 号接
口\n");
                    printf("还剩下 endline:%d", endline);
                    //将 Ack 编码到 send_ack
                    acktobin(Ack, send_ack);
                    remote_addr.sin_port                    =
htons(sock_list.MainPort);
                    sendto(sock_list.MainSock, send_ack, 64, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));

                    if (endline == 0)
                    {
                        k = 0;
                        count = -1;

                    }

                }
                else if (retval == 0)
                {
                    system("pause");
                }

            }

            /*
            char sendbuf1[100] = { 1, 0, 0, 1, 0 };//测试数据
            remote_addr.sin_port = htons(sock_list.port_array[0]);
            sendto(sock_list.sock_array[0], sendbuf1, 5, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));

            printf("将应用层数据发送到 0 号接口\n");
```

```
                        for (i == 0; i < 5; i++)
                        {
                        printf("%d", buf[i]);
                        }

                        */

                    }
                }
            }


        }
}
/*

*/
for (intf = 0; intf < 64; intf++)
{
    if (sock_list.sock_array[intf] == 0)
        continue;
    sock = sock_list.sock_array[intf];

    if (FD_ISSET(sock, &readfds)) {

        retval = recv(sock, buf, MAX_BUFFSIZE, 0);
        if (retval == 0) {
            closesocket(sock);
            printf("失去接口%d 的关联\n", i);
            delete_list(sock, &sock_list);
            continue;
        }
        else if (retval == -1) {
            retval = WSAGetLastError();
            if (retval == WSAEWOULDBLOCK)
                continue;
            closesocket(sock);
            printf("失去接口%d 的关联\n", i);
            delete_list(sock, &sock_list);
            continue;
        }
        //收到数据后，打印
        printf("%d 号接口收到比特流:", intf);
        printf("\n");
```

```
                    //处理程序
                    //查帧，去 CRC
                    memcpy(receive_endline, buf, MAX_BUFFSIZE);
                    receival = find(receive_endline, newdata, SEND_L, &length1);
                    back = event(newdata, restore_data, receival, length1);
                    if (back == 0)
                    {
                        //给 flag 赋值为目的地址

                        if (route_recv<2)
                        {
                            route_recv++;
                            remote_addr.sin_port = htons(sock_list.MainPort);
                            sendto(sock_list.MainSock, restore_data, _length, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                            printf("从%d 号口收到的初始路由表上交\n", intf);
                            //将 Ack 编码到 send_ack,Ack=0;
                            prepare_ack(send_ack, &ACK);
                            remote_addr.sin_port = htons(sock_list.port_array[intf]);
                            printf("回应%d 号接口\n", intf);

                            retval = sendto(sock_list.sock_array[intf], send_ack, 2, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                        }
                        else
                        {
                            uncode_goal(restore_data, &flag);
                            if (flag == Host)//本机
                            {
                                remote_addr.sin_port = htons(sock_list.MainPort);
                                sendto(sock_list.MainSock, restore_data, _length, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                                printf("从%d 号口收到的数据上交\n", intf);
                                //将 Ack 编码到 send_ack,Ack=0;
                                prepare_ack(send_ack, &ACK);
                                remote_addr.sin_port                                =
htons(sock_list.port_array[intf]);
                                printf("回应%d 号接口\n", intf);

                                retval = sendto(sock_list.sock_array[intf], send_ack, 2, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                            }
                            else//转发
                            {
```

```
                                intf0 = intf;
                                intf1 = (intf + 1) % 2;
                                remote_addr.sin_port                                =
htons(sock_list.port_array[intf1]);
                                printf("转发接口");
                                memcpy(sendbuf, newdata, SEND_L);
                                retval = sendto(sock_list.sock_array[intf1], sendbuf,
SEND_L, 0, (sockaddr*)&remote_addr, sizeof(remote_addr));
                                //定义一个控制位 K=1
                                /*
                                FD_ZERO(&readfds);
                                FD_ZERO(&writefds);
                                FD_ZERO(&exceptfds);
                                FD_SET(sock_list.sock_array[intf0], &readfds);
                                FD_SET(sock_list.sock_array[intf1], &readfds);
                                selretval = select(0, &readfds, &writefds, &exceptfds,
&timeout);
                                if (selretval == SOCKET_ERROR)
                                {
                                retval = WSAGetLastError();
                                break;
                                }*/
                                if (FD_ISSET(sock_list.sock_array[intf0], &readfds))
                                {
                                    retval = recv(sock_list.sock_array[intf0], buf,
MAX_BUFFSIZE, 0);
                                    if (retval > 0)
                                    {
                                        memcpy(receive_endline,                buf,
MAX_BUFFSIZE);
                                        receival = find(receive_endline, newdata,
SEND_L, &length1);
                                        back = event(newdata, restore_data, receival,
length1);
                                        remote_addr.sin_port                        =
htons(sock_list.port_array[intf1]);
                                        printf("转发接口");
                                        memcpy(sendbuf, newdata, SEND_L);
                                        retval = sendto(sock_list.sock_array[intf1],
sendbuf, SEND_L, 0, (sockaddr*)&remote_addr, sizeof(remote_addr));
                                    }

                                }
```

```
                    if (FD_ISSET(sock_list.sock_array[intf1], &readfds))
                    {
                        retval = recv(sock_list.sock_array[intf1], blank_buf, MAX_BUFFSIZE, 0);
                        printf("收到转发回应\n");
                        if (retval > 0)
                        {
                            remote_addr.sin_port = htons(sock_list.port_array[intf0]);
                            //将 Ack 编码到 send_ack,Ack=0;
                            prepare_ack(send_ack, &ACK);
                            //retval = sendto(sock_list.sock_array[intf0], send_ack, 64, 0, (sockaddr*)&remote_addr, sizeof(remote_addr));
                            retval = sendto(sock_list.sock_array[intf0], send_ack, 2, 0, (sockaddr*)&remote_addr, sizeof(remote_addr));
                        }
                    }


                }
            }


            /*
            //为了测试，把 0 号接口的数据转发到 1 号接口，如果有 1 号接口的话
            if (intf == 0) {
            if (sock_list.sock_array[1] != 0) {
            remote_addr.sin_port = htons(sock_list.port_array[1]);
            printf("从 0 号转发到 1 号接口\n");
            retval = sendto(sock_list.sock_array[1], buf, 5, 0, (sockaddr*)&remote_addr, sizeof(remote_addr));
            }
            }
            //为了测试，从 1 号接口收到的数据，上交给应用层模块
            if (intf == 1) {
            //先把"比特流"解码成普通数据形式，每次解码 8 位
            for (i = 0; i < retval; i += 8){
            sendbuf[i / 8] = (char)decode(buf + i, 8);
            }

            if (back == 0)
```

```
                    {
                    remote_addr.sin_port = htons(sock_list.MainPort);
                    sendto(sock_list.MainSock, restore_data, 64, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));
                    printf("从 1 号口收到的数据上交\n");
                    prepare_ack(sendbuf, &ACK, s);
                    remote_addr.sin_port = htons(sock_list.port_array[1]);
                    printf("回应 1 号接口\n");
                    retval = sendto(sock_list.sock_array[1], sendbuf, 64, 0,
(sockaddr*)&remote_addr, sizeof(remote_addr));

                    }
                    }
                    */
                }

            }
        }
        FD_ZERO(&readfds);
        FD_ZERO(&writefds);
        FD_ZERO(&exceptfds);
    }

    closesocket(sock_list.MainSock);
    WSACleanup();
    return 0;
}
```

2. 应用层代码及解释

```
// server.cpp :
//

#ifndef _CRT_SECURE_NO_WARNINGS

#define _CRT_SECURE_NO_WARNINGS

#endif

#define feclong 6


#define framelong_send 3173
#define datalong_send 3168
```

```
#define framelong_receive 3173
#define datalong_receive 3168
#define length_A 3500


#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <sys/types.h>

#include <winsock2.h>

#include <WS2tcpip.h>

#include <iostream>

#include "winsock.h"

#include "conio.h"

#include <fcntl.h>

typedef unsigned int uint;
char p[feclong] = { 1, 0, 0, 1, 1, 1 };//CRC 用
#pragma  comment(lib,"ws2_32.lib")//静态库
const char * base64char =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

void binlengthtobin(int binlength, char imagesize[])//将 binlength 转为二进制
{
  unsigned long test;
  int i;
  test = 1;
  test = test << 23;
  for (i = 0; i < 24; i++) {
        if (test & binlength) {
              imagesize[i] = 1;
        }
```

```
        else {
            imagesize[i] = 0;
        }
        test = test >> 1;
    }
}
void base64_encode(char * bindata, char * base64, int binlength)
{
    int i, j;
    unsigned char current;
    for (i = 0, j = 0; i < binlength; i += 3)
    {
        current = (bindata[i] >> 2);
        current &= (unsigned char)0x3F;
        base64[j++] = base64char[(int)current];

        current = ((unsigned char)(bindata[i] << 4)) & ((unsigned char)0x30);
        if (i + 1 >= binlength)
        {
            base64[j++] = base64char[(int)current];
            base64[j++] = '=';
            base64[j++] = '=';
            break;
        }
        current |= ((unsigned char)(bindata[i + 1] >> 4)) & ((unsigned char)0x0F);
        base64[j++] = base64char[(int)current];

        current = ((unsigned char)(bindata[i + 1] << 2)) & ((unsigned char)0x3C);
        if (i + 2 >= binlength)
        {
            base64[j++] = base64char[(int)current];
            base64[j++] = '=';
            break;
        }
        current |= ((unsigned char)(bindata[i + 2] >> 6)) & ((unsigned char)0x03);
        base64[j++] = base64char[(int)current];

        current = ((unsigned char)bindata[i + 2]) & ((unsigned char)0x3F);
        base64[j++] = base64char[(int)current];
    }
    base64[j] = '\0';

}
int base64tobit(char * base64, char * bit, int binlength)
```

```
{
  unsigned int test = 1;
  test = test << 7;
  int bitlong = 0;
  for (int i = 0; i<binlength * 2; i++) {
        for (int j = 0; j<8; j++) {
              if (test&base64[i]) {
                    bit[i * 8 + j] = 1;
              }
              else {
                    bit[i * 8 + j] = 0;
              }
              bitlong = i * 8 + j + 1;
              test = test >> 1;

        }
        test = 1 << 7;

  }
  return bitlong;
}
void base64_decode(char * base64, char * bindata)
{
  int i, j;
  unsigned char k;
  unsigned char temp[4];
  for (i = 0, j = 0; base64[i] != '\0'; i += 4)
  {
        memset(temp, 0xFF, sizeof(temp));
        for (k = 0; k < 64; k++)
        {
              if (base64char[k] == base64[i])
                    temp[0] = k;
        }
        for (k = 0; k < 64; k++)
        {
              if (base64char[k] == base64[i + 1])
                    temp[1] = k;
        }
        for (k = 0; k < 64; k++)
        {
              if (base64char[k] == base64[i + 2])
                    temp[2] = k;
        }
```

```
        for (k = 0; k < 64; k++)
        {
            if (base64char[k] == base64[i + 3])
                temp[3] = k;
        }

        bindata[j++] = ((unsigned char)(((unsigned char)(temp[0] << 2)) & 0xFC)) |
            ((unsigned char)((unsigned char)(temp[1] >> 4) & 0x03));
        if (base64[i + 2] == '=')
            break;

        bindata[j++] = ((unsigned char)(((unsigned char)(temp[1] << 4)) & 0xF0)) |
            ((unsigned char)((unsigned char)(temp[2] >> 2) & 0x0F));
        if (base64[i + 3] == '=')
            break;

        bindata[j++] = ((unsigned char)(((unsigned char)(temp[2] << 6)) & 0xF0)) |
            ((unsigned char)(temp[3] & 0x3F));
    }
}
void bittobase64(char * base64, char * bit, int binlength)
{
    unsigned int test2 = 1 << 7;
    for (int i = 0; i<binlength * 2; i++) {
        base64[i] = '0' - '0';
        for (int j = 0; j<8; j++) {
            if (bit[i * 8 + j] == 0) {
                test2 = test2 >> 1;
            }
            else {
                base64[i] = base64[i] + (char)test2;
                test2 = test2 >> 1;
            }
        }
        test2 = 1 << 7;
    }
}

int Output_imagelong(char pro[3168]) //图片大小，输入分块数组
{
    int imagelong = 0;
    char imagesize[24];
    for (int k = 0; k<24; k++) {
        imagesize[k] = pro[k + 8];
```

```
  }
  unsigned int test = 1 << 23;
  for (int k = 0; k<23; k++) {
        if (imagesize[k] == 1) {
              imagelong = (test | 0) + imagelong;
        }
        test = test >> 1;
  }
  return imagelong;
}
int Output_bitlong(char pro[3168]) //图片大小，输入分块数组
{
  int bitlong = 0;
  char bitsize[24];
  for (int k = 0; k<24; k++) {
        bitsize[k] = pro[k + 32];
  }
  unsigned int test = 1 << 23;
  for (int k = 0; k<23; k++) {
        if (bitsize[k] == 1) {
              bitlong = (test | 0) + bitlong;
        }
        test = test >> 1;
  }
  return bitlong;
}
void Output(char *ori, int bitlong, char pro[][3168]) //数据，输入图片大小，分块数
组
{
  int j = 1;
  for (int k = 0; k<bitlong;) {

        for (int i = 8; i<3168; i++) {
              ori[k] = pro[j][(k % 3160) + 8];
              k++;
        }
        j++;

  }

}

void Add(char array_A[], char array_B[], int flag)
{
```

```
  int i;
  array_B[0] = flag;
  for (i = 0; i < length_A; i++)
  {
        array_B[i + 1] = array_A[i];
  }

}
void Sub(char B[], char A[])
{
  int i = 0;
  for (i; i < length_A; i++)
  {
        A[i] = B[i + 1];
  }
}

void acktobin(int ack, char Ack[])//将 ack 转为二进制
{
  unsigned long test;
  int i;
  test = 1;
  test = test << 1;
  for (i = 0; i < 2; i++) {
        if (test & ack) {
              Ack[i] = 1;
        }
        else {
              Ack[i] = 0;
        }
        test = test >> 1;
  }

}
int length(char a[], int maxlength) {
  int i, m;
  for (i = 0, m = 0; i < maxlength; i++) {
        if ((a[i] != 0) && (a[i] != 1)) {
              break;
        }
        else if (a[i] == '/0') {
              break;
        }
        else {
```

```
            m++;
        }
    }
    return m;
}//计算数组已填充长度

void outkernel(char raw[], char b[]) {
    int len, m;
    len = length(raw, 1100);
    m = 0;
    for (int i = 0; i < len; i++) {
        if (i % 32 < 8) {
            m++;
        }
        else b[i - m] = raw[i];
    }
}//去除 seqack
void tranlate(char rec[], char output[]) {
    int test, i;
    char yong[800];
    outkernel(rec, yong);
    int len = length(output, 150);
    for (int j = 0; j < 100; j++) {
        output[j + len] = 0;
        for (i = 0; i < 8; i++) {
            test = 1;
            test = test << (7 - i);
            test = test | output[j + len];
            if (yong[i + 8 * j] == 1) {
                output[j + len] = output[j + len] | test;
            }
            else {
                output[j + len] = output[j + len] & test;
            }
        }

    }
}//解码
void CRC(char data[], char crc_data[], char p[])
{

    char sub[feclong];
    char data_long[framelong_send];
```

```
for (int i = 0; i < datalong_send; i++)
{
	data_long[i] = data[i];
}

for (int i = datalong_send; i < framelong_send; i++)
{
	data_long[i] = 0;
}
for (int i = 0; i < feclong; i++)
{
	sub[i] = data_long[i];
}


for (int i = 0; i < datalong_send; i++)
{

	for (int j = 1; j < feclong; j++)
	{
		if (sub[j] == p[j])
		{
			sub[j - 1] = 0;
		}
		else
		{
			sub[j - 1] = 1;
		}
	}
	sub[feclong - 1] = data_long[i + feclong];

	while (sub[0] == 0 && i< (datalong_send - 1))
	{

		for (int k = 0; k < (feclong - 1); k++)
		{
			sub[k] = sub[k + 1];
		}
		i++;
		sub[(feclong - 1)] = data_long[i + feclong];
	}
}
```

```
  for (int i = 0; i < datalong_send; i++)
        crc_data[i] = data[i];

  for (int i = datalong_send; i < framelong_send; i++)
        crc_data[i] = sub[i - datalong_send];




}
int check_CRC(char data[], char crc_data[], char p[])
{

  char sub[feclong];
  for (int i = 0; i < feclong; i++)
        sub[i] = crc_data[i];
  char data_long[framelong_receive];
  for (int i = 0; i < datalong_receive; i++)
        data_long[i] = data[i];
  for (int i = datalong_receive; i < framelong_receive; i++)
        data_long[i] = 0;

  for (int i = 0; i < datalong_receive; i++)
  {

        for (int j = 1; j < feclong; j++)
        {
              if (sub[j] == p[j])
                    sub[j - 1] = 0;
              else sub[j - 1] = 1;

        }
        sub[feclong - 1] = crc_data[i + feclong];

        while (sub[0] == 0 && i < (datalong_receive - 1))
        {
              if (sub[0] == 0)
              {
                    for (int k = 0; k < (feclong - 1); k++)
                          sub[k] = sub[k + 1];
                    i++;
```

```
                    sub[(feclong - 1)] = crc_data[i + feclong];
                }
            }
        }
        int check = 0;
        for (int i = 0; i < feclong - 1; i++)
            check = check + sub[i];
        if (check == 0)
        {
            for (int i = 0; i < datalong_receive; i++)
            {
                data[i] = crc_data[i];
            }
            return 0;
        }
        else
            return 1;
    }
    int length_Input(char a[], int maxlength) {
        int i, m;
        for (i = 0, m = 0; i < maxlength; i++) {
            if (a[i] == 0) {
                break;
            }
            else {
                m++;
            }
        }
        return m;
    }//计算数组已填充长度
    int check(char *frame, char restore_data[], int l)//l 为 frame 的长度，可用 sizeof
    {
        //先去帧
        //再去填充
        //再 CRC
        int i = 0;
        int m = 0;
        for (i = 0; i < l - 16 - m; i++)
        {
            if ((i>4) && (frame[i + 7 + m] == 1) && (frame[i + 6 + m] == 1) && (frame[i +
5 + m] == 1) && (frame[i + 4 + m] == 1) && (frame[i + 3 + m] == 1))
            {
                m++;
            }
```

```
            restore_data[i] = frame[i + 8 + m];
            //            printf("%d", restore_data[i]);
        }
        //去 CRC
        if (check_CRC(restore_data, restore_data, p) == 0)
        {
            return 1;
        }
        else
        {
            return 0;
        }
        //正确返回 1,正确数据就是 restore_data

        //反之返回 0
    }
    int event(char newdata[], char restore_data[], int receival, int length)
    {
        int back = -1;
        if (receival == 1)
        {
            if (check(newdata, restore_data, length))
            {
                back = 0;
            }
            else {
                back = 1;
            }
        }
        else { back = 1; }
        return back;
    }

    void make_frame(char crc_data[], char frame[]) {
        int l, m, n, i;
        l = framelong_send;
        char a[8] = { 0, 1, 1, 1, 1, 1, 1, 0 };
        for (i = 0; i < 8; i++) {
            frame[i] = a[i];
        }
        for (i = 0, m = 0; i < l; i++) {
            if ((i>4) && (frame[i + 3 + m] == 1) && (frame[i + 4 + m] == 1) && (frame[i +
5 + m] == 1) && (frame[i + 6 + m] == 1) && (frame[i + 7 + m] == 1)) {
                frame[i + 8 + m] = 0;
```

```
            m++;
        }
        frame[i + 8 + m] = crc_data[i];
  }
  for (i = 0; i < 8; i++) {
        frame[i + 8 + m + l] = a[i];
  }
}
void prepare_ack(char frame[], int *ack)//发送
{
  char Ack[2];
  char crc_ack[framelong_send] = { 0 };
  acktobin(*ack, Ack);
  CRC(Ack, crc_ack, p);
  make_frame(crc_ack, frame);
}
void uncode_goal(char restore_data[], int* seq)
{
  if (restore_data[0] == 0 && restore_data[1] == 0) {
        *seq = 0;
  }
  else if (restore_data[0] == 0 && restore_data[1] == 1) {
        *seq = 1;
  }
  else if (restore_data[0] == 1 && restore_data[1] == 0) {
        *seq = 2;
  }
  else if (restore_data[0] == 1 && restore_data[1] == 1) {
        *seq = 3;
  }
}
void uncode(char restore_data[], int* seq)
{

  /*    if (restore_data[0] == 0 && restore_data[1] == 0) {
  *seq = 0;
  }
  else if (restore_data[0] == 0 && restore_data[1] == 1) {
  *seq = 1;
  }
  else if (restore_data[0] == 1 && restore_data[1] == 0) {
  *seq = 2;
  }
  else if (restore_data[0] == 1 && restore_data[1] == 1) {
```

```
*seq = 3;
}
*/
/*  if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0) {
*seq = 8;
}*/
/*
if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0) {
*seq = 0;
}
if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1) {
*seq = 1;
}
if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0) {
*seq = 2;
}
if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1) {
*seq = 3;
}
if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0) {
*seq = 4;
}
if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1) {
*seq = 5;
}
if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0) {
*seq = 6;
}
if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1) {
*seq = 7;
}
if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0) {
*seq = 8;
}
```

```
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1) {
        *seq = 9;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0) {
        *seq = 10;
        }
        if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1) {
        *seq = 11;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0) {
        *seq = 12;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1) {
        *seq = 13;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0) {
        *seq = 14;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1) {
        *seq = 15;
        }
        */
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 0) {
                *seq = 0;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 1) {
                *seq = 1;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 0) {
                *seq = 2;
        }
        if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 1) {
                *seq = 3;
```

```
    }
    if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 0) {
        *seq = 4;
    }
    if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 1) {
        *seq = 5;
    }
    if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 0) {
        *seq = 6;
    }
    if (restore_data[0] == 0 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 1) {
        *seq = 7;
    }
    if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 0) {
        *seq = 8;
    }
    if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 1) {
        *seq = 9;
    }
    if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 0) {
        *seq = 10;
    }
    if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 1) {
        *seq = 11;
    }
    if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 0) {
        *seq = 12;
    }
    if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 1) {
        *seq = 13;
    }
    if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 0) {
        *seq = 14;
```

```
    }
    if (restore_data[0] == 0 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 1) {
        *seq = 15;
    }
    if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 0) {
        *seq = 16;
    }
    if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 1) {
        *seq = 17;
    }
    if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 0) {
        *seq = 18;
    }
    if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 1) {
        *seq = 19;
    }
    if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 0) {
        *seq = 20;
    }
    if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 1) {
        *seq = 21;
    }
    if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 0) {
        *seq = 22;
    }
    if (restore_data[0] == 1 && restore_data[1] == 0 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 1) {
        *seq = 23;
    }
    if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 0) {
        *seq = 24;
    }
    if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 0 && restore_data[4] == 1) {
        *seq = 25;
```

```
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 0) {
                *seq = 26;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 0 &&
restore_data[3] == 1 && restore_data[4] == 1) {
                *seq = 27;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 0) {
                *seq = 28;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 0 && restore_data[4] == 1) {
                *seq = 29;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 0) {
                *seq = 30;
        }
        if (restore_data[0] == 1 && restore_data[1] == 1 && restore_data[2] == 1 &&
restore_data[3] == 1 && restore_data[4] == 1) {
                *seq = 31;
        }
      }
      int find(char rawdata[], char newdata[], int length, int* frame_length)
      {
        int i, m, n, p, receival;
        p = 0;
        for (i = 0, n = 0, m = 0; i < 3592; i++) {
                if (rawdata[i] == 0 && rawdata[i + 1] == 1 && rawdata[i + 2] == 1 && rawdata[i
+ 3] == 1 && rawdata[i + 4] == 1 && rawdata[i + 5] == 1 && rawdata[i + 6] == 1 &&
rawdata[i + 7] == 0) {
                        if (p == 0) {
                                m = i;
                                p = 1;
                        }
                        else if (p == 1) {
                                n = i;
                                p = 2;
                        }
                        else p = 3;
                }
```

```
  }
  if (m != 0 && n != 0 && p != 3) {
        for (i = 0; i < n - m + 8; i++) {
              newdata[i] = rawdata[i + m];
        }
        receival = 1;
        *frame_length = n - m + 8;
  }
  else receival = 0;

  return receival;//返回是否读到有效帧
}
int uncode_send(char restore_data[], int seq, int ack)
{
  //解码解出 seq,ack 再判断

  if (seq == ack)
  {
        return 1;
  }
  else
  {
        return 0;
  }
}
/*********************************************************************/
struct socket_list//socket 数组
{
  SOCKET MainSock;
  unsigned short MainPort;//应用层的端口号
  int num; //接口数量
  SOCKET sock_array[64];
  unsigned short port_array[64];//各接口模块的端口号
};
void init_list(socket_list *list)//初始化 socket 数组
{
  int i;
  list->MainSock = 0;
  list->num = 0;
  for (i = 0; i < 64; i++) {
        list->sock_array[i] = 0;
  }
}
void insert_list(SOCKET s, unsigned short port, socket_list *list)//将指定套接字放入
```

```
socket 数组空位置
    {
     int i;
     for (i = 0; i < 64; i++) {
          if (list->sock_array[i] == 0) {
               list->sock_array[i] = s;
               list->port_array[i] = port;
               list->num += 1;
               break;
          }
     }
    }
    void delete_list(SOCKET s, socket_list *list)//从 socket 数组中删除指令套接字
    {
     int i;
     for (i = 0; i < 64; i++) {
          if (list->sock_array[i] == s) {
               list->sock_array[i] = 0;
               list->num -= 1;
               break;
          }
     }
    }
    void make_fdlist(socket_list *list, fd_set *fd_list)//将 socket 数组放入 fd 数组中，准
备进入 select
    {
     int i;
     FD_SET(list->MainSock, fd_list);
     for (i = 0; i < 64; i++) {
          if (list->sock_array[i] > 0) {
               FD_SET(list->sock_array[i], fd_list);
          }
     }
    }
    void code(unsigned long x, char A[], int length)
    {
     unsigned long test;
     int i;
     x = x << (length - 5);
     test = 1;
     test = test << (length - 1);
     printf("this is");
     for (i = 0; i < length; i++) {
          if (test & x) {
```

```
                A[i] = 1;
            }
            else {
                A[i] = 0;
            }
            printf("%d", A[i]);
            test = test >> 1; //本算法利用了移位操作和"与"计算，逐位测出 x 的每一位是 0 还是 1.
        }
        printf("\n");
    }
    unsigned long decode(char A[], int length)//解码函数
    {
        unsigned long x;
        int i;

        x = 0;
        for (i = 0; i < length; i++) {
            if (A[i] == 0) {
                x = x << 1;;
            }
            else {
                x = x << 1;
                x = x | 1;
            }
        }
        return x;
    }
    int Input(char *ori, int imagelong, int bitlong, int dest, int sour, char pro[][3168]) //数据，图片大小，目的，源，输出分块数组
    {
        int i, j, m = 1;
        int l;
        l = imagelong * 16;//l 为 ori 长度

        char source[2];
        char desti[2];
        char imagesize[24], bitsize[24];
        binlengthtobin(imagelong, imagesize);
        binlengthtobin(bitlong, bitsize);
        acktobin(sour, source);
        acktobin(dest, desti);
        pro[0][0] = 0;
        pro[0][1] = desti[0];
```

```
  pro[0][2] = desti[1];
  pro[0][3] = 0;
  pro[0][4] = 0;
  pro[0][5] = source[0];
  pro[0][6] = source[1];
  pro[0][7] = 0;

  for (int k = 0; k<24; k++) {
        pro[0][k + 8] = imagesize[k];
  }
  for (int k = 0; k<24; k++) {
        pro[0][k + 32] = bitsize[k];
  }



  for (j = 0; j < l; j++) {
        pro[m][0] = 0;
        pro[m][1] = desti[0];
        pro[m][2] = desti[1];
        pro[m][3] = 0;
        pro[m][4] = 0;
        pro[m][5] = source[0];
        pro[m][6] = source[1];
        pro[m][7] = 0;

        for (i = 8; (i < 3168)&(j<l); i++) {
              pro[m][i] = ori[j];

              j++;

        }
        j--;
        m++;
  }
  m--;
  return m;
}
int *IP0(char rec[2][6]) {
  static int send[4];
  send[0] = 99;//0-self
  if (rec[0][2] == 0) {
        if (rec[0][1] <= rec[0][4] + rec[1][1] + rec[1][4]) {
              send[rec[0][0]] = rec[0][0];
```

```
        }
        else {
            send[rec[0][0]] = rec[1][0];
        }
    }
    else {
        if (rec[0][4] <= rec[0][1] + rec[1][1] + rec[1][4]) {
            send[rec[0][0]] = rec[0][0];
        }
        else
            send[rec[0][0]] = rec[1][0];
    }
    if (rec[1][2] == 0) {
        if (rec[1][1] <= rec[1][4] + rec[0][1] + rec[0][4]) {
            send[rec[1][0]] = rec[1][0];
        }
        else
            send[rec[1][0]] = rec[0][0];
    }
    else {
        if (rec[1][4] <= rec[1][1] + rec[0][1] + rec[0][4]) {
            send[rec[1][0]] = rec[1][0];
        }
        else
            send[rec[1][0]] = rec[0][0];
    }
    if (rec[0][1] + rec[0][4] >= rec[1][1] + rec[1][4]) {
        if (rec[0][2] != 0) {
            send[rec[0][2]] = rec[1][0];
        }
        else
            send[rec[0][5]] = rec[1][0];
    }
    else {
        if (rec[0][2] != 0) {
            send[rec[0][2]] = rec[0][0];
        }
        else {
            send[rec[0][5]] = rec[0][0];
        }
    }
    return send;
}
/*计算机顺序:
```

```
0 1
2 3
当数据传给自身时，出口为 99;
出口按距离计算;
*/
int *IP1(char rec[2][6]) {
  static int send[4];
  send[1] = 99;//0-self
  if (rec[0][2] == 1) {
        if (rec[0][1] <= rec[0][4] + rec[1][1] + rec[1][4]) {
              send[rec[0][0]] = rec[0][0];
        }
        else {
              send[rec[0][0]] = rec[1][0];
        }
  }
  else {
        if (rec[0][4] <= rec[0][1] + rec[1][1] + rec[1][4]) {
              send[rec[0][0]] = rec[0][0];
        }
        else
              send[rec[0][0]] = rec[1][0];
  }
  if (rec[1][2] == 1) {
        if (rec[1][1] <= rec[1][4] + rec[0][1] + rec[0][4]) {
              send[rec[1][0]] = rec[1][0];
        }
        else
              send[rec[1][0]] = rec[0][0];
  }
  else {
        if (rec[1][4] <= rec[1][1] + rec[0][1] + rec[0][4]) {
              send[rec[1][0]] = rec[1][0];
        }
        else
              send[rec[1][0]] = rec[0][0];
  }
  if (rec[0][1] + rec[0][4] >= rec[1][1] + rec[1][4]) {
        if (rec[0][2] != 1) {
              send[rec[0][2]] = rec[1][0];
        }
        else
              send[rec[0][5]] = rec[1][0];
  }
```

```
  else {
        if (rec[0][2] != 1) {
              send[rec[0][2]] = rec[0][0];
        }
        else {
              send[rec[0][5]] = rec[0][0];
        }
  }
  return send;
}
/*计算机顺序:
0 1
2 3
当数据传给自身时，出口为 9;
2->0 出口为 0;
2->3 出口为 1;
2->1 按距离计算;
*/
int *IP2(char rec[2][6]) {
  static int send[4];
  send[2] = 99;//0-self
  if (rec[0][2] == 2) {
        if (rec[0][1] <= rec[0][4] + rec[1][1] + rec[1][4]) {
              send[rec[0][0]] = rec[0][0];
        }
        else {
              send[rec[0][0]] = rec[1][0];
        }
  }
  else {
        if (rec[0][4] <= rec[0][1] + rec[1][1] + rec[1][4]) {
              send[rec[0][0]] = rec[0][0];
        }
        else
              send[rec[0][0]] = rec[1][0];
  }
  if (rec[1][2] == 2) {
        if (rec[1][1] <= rec[1][4] + rec[0][1] + rec[0][4]) {
              send[rec[1][0]] = rec[1][0];
        }
        else
              send[rec[1][0]] = rec[0][0];
  }
  else {
```

```
        if (rec[1][4] <= rec[1][1] + rec[0][1] + rec[0][4]) {
            send[rec[1][0]] = rec[1][0];
        }
        else
            send[rec[1][0]] = rec[0][0];
    }
    if (rec[0][1] + rec[0][4] >= rec[1][1] + rec[1][4]) {
        if (rec[0][2] != 2) {
            send[rec[0][2]] = rec[1][0];
        }
        else
            send[rec[0][5]] = rec[1][0];
    }
    else {
        if (rec[0][2] != 2) {
            send[rec[0][2]] = rec[0][0];
        }
        else {
            send[rec[0][5]] = rec[0][0];
        }
    }
    return send;
}
/*计算机顺序:
0 1
2 3
当数据传给自身时，出口为 9;
3->1 出口为 0;
3->2 出口为 1;
3->0 按距离计算;
*/
int *IP3(char rec[2][6]) {
    static int send[4];
    send[3] = 99;//0-self
    if (rec[0][2] == 3) {
        if (rec[0][1] <= rec[0][4] + rec[1][1] + rec[1][4]) {
            send[rec[0][0]] = rec[0][0];
        }
        else {
            send[rec[0][0]] = rec[1][0];
        }
    }
    else {
        if (rec[0][4] <= rec[0][1] + rec[1][1] + rec[1][4]) {
```

```
            send[rec[0][0]] = rec[0][0];
        }
        else
            send[rec[0][0]] = rec[1][0];
    }
    if (rec[1][2] == 3) {
        if (rec[1][1] <= rec[1][4] + rec[0][1] + rec[0][4]) {
            send[rec[1][0]] = rec[1][0];
        }
        else
            send[rec[1][0]] = rec[0][0];
    }
    else {
        if (rec[1][4] <= rec[1][1] + rec[0][1] + rec[0][4]) {
            send[rec[1][0]] = rec[1][0];
        }
        else
            send[rec[1][0]] = rec[0][0];
    }
    if (rec[0][1] + rec[0][4] >= rec[1][1] + rec[1][4]) {
        if (rec[0][2] != 3) {
            send[rec[0][2]] = rec[1][0];
        }
        else
            send[rec[0][5]] = rec[1][0];
    }
    else {
        if (rec[0][2] != 3) {
            send[rec[0][2]] = rec[0][0];
        }
        else {
            send[rec[0][5]] = rec[0][0];
        }
    }
    return send;
}
int *IP(int choice, char rec[2][6])
{
    int *send = NULL;
    switch (choice)
    {
    case 0:
        send = IP0(rec);
        break;
```

```
  case 1:
        send = IP1(rec);
        break;
  case 2:
        send = IP2(rec);
        break;
  case 3:
        send = IP3(rec);
        break;

  }
  return send;
}
void transferA(char a[], char b[]) {//a[6]-->b[12]
  char c[2];
  for (int i = 0; i < 6; i++) {
        acktobin(a[i], c);
        b[i * 2] = c[0];
        b[i * 2 + 1] = c[1];
  }

}
void transferB(char m[], char n[]) {//m[12]-->n[6]
  char d[2];
  int s = 1;
  for (int i = 0; i < 6; i++) {
        d[0] = m[i * 2];
        d[1] = m[i * 2 + 1];
        if (d[0] == 0 && d[1] == 0) {
              s = 0;
        }
        else if (d[0] == 0 && d[1] == 1) {
              s = 1;
        }
        else if (d[0] == 1 && d[1] == 0) {
              s = 2;
        }
        else if (d[0] == 1 && d[1] == 1) {
              s = 3;
        }
        n[i] = s;
        printf("%d", n[i]);
  }
}
```

```
#define MAX_BUFFSIZE 3600
#define SEND_L 3500
int main(int argc, char* argv[])
{
  SOCKET s;
  struct sockaddr_in remote_addr;
  char buf[MAX_BUFFSIZE] = { 0 };//接收缓存
  char sendbuf[MAX_BUFFSIZE] = { 0, 1, 1, 0, 1 };//测试数据
  WSAData wsa;
  int retval, selretval;
  struct socket_list sock_list;
  fd_set readfds, writefds, exceptfds;
  timeval timeout;
  fd_set read;
  unsigned long arg;
  int linecount = 0;
  unsigned short myPort;
  int ID;
  int i;

  /***********************/
  int count = -1;
  int endline = 0;
  int length_restore = 3168;
  //char right_store[1000] = { 0 };
  int right_number = 0;
  char out[100];
  int k = -1;
  int K = 1;
  int flag = -1;
  int F;

  //char Pre_route[2][length_restoredata] = { 0 };
  //char Route[4] = { 5, 5, 5, 5 };
  /***********************/

  WSAStartup(0x101, &wsa);

  //buf = (char *)malloc(MAX_BUFFSIZE);

  init_list(&sock_list);
  //从键盘输入，需要连接的 API 端口号
  printf("本模块是应用层,\n");
  printf("请输入本模块的设备编号：");//主要用来区分不同的设备
```

```
scanf_s("%d", &ID);
//绑定本模块端口号，以接收应用层数据
printf("请设置本模块的端口号：");
scanf_s("%d", &myPort);

s = socket(AF_INET, SOCK_DGRAM, 0);
if (s == SOCKET_ERROR) {
	return 0;
}

remote_addr.sin_family = AF_INET;
remote_addr.sin_port = htons(myPort);
remote_addr.sin_addr.S_un.S_addr = htonl(INADDR_LOOPBACK);
while (bind(s, (sockaddr*)&remote_addr, sizeof(remote_addr)) != 0) {
	printf("本模块的端口号已被占用，请换一个再试:");
	scanf_s("%d", &myPort);
	remote_addr.sin_port = htons(myPort);
}
sock_list.MainSock = s;

printf("请输入所连接的网络层模块端口号：");
scanf_s("%d", &sock_list.MainPort);

printf("请输入测试数据(小于 20 字符的字符串):");
printf("请注意，按任意键将发送一份测试数据\n");


//设置套接字的状态为阻塞态，也可以不设置
arg = 1;
ioctlsocket(s, FIONBIO, &arg);

timeout.tv_sec = 0; //设置发送数据的时间间隔为 10 微秒
timeout.tv_usec = 10;
FD_ZERO(&readfds);
FD_ZERO(&writefds);
FD_ZERO(&exceptfds);

/*	for(i = 0 ; i < 4000; i ++){
sendbuf[i] = 1; //初始化数据全为 1,以备测试
}
*/
/*************************/
int receival;//看是否接受到帧
int distinc;
```

```
int back;
int Seq = 0;
int frame_length = 0;
//int retval;//接受数据长度
//int s;//套接字
//存放帧的数组
char framebuff[30][3500] = { 0 };
/*******************************/
FILE *fp = NULL;
unsigned int imageSize, bitlong;
char *imageBin;
char *imageBase64;
char *imagebit;
char *imageOutput, *imageBase64_2;
int rec_Imag_length = 0;
int rec_bit_length = 0;
char *right_store;
char pro[30][3168] = { 0 };
int j = 1;
fp = fopen("C:\\Users\\hp\\Desktop\\123.jpg", "rb");    //待编码图片地址
fseek(fp, 0L, SEEK_END);
imageSize = ftell(fp);
fseek(fp, 0L, SEEK_SET);
imageBin = (char *)malloc(sizeof(char)*imageSize);
fread(imageBin, 1, imageSize, fp);
fclose(fp);
imageBase64 = (char *)malloc(sizeof(char)*imageSize * 2);
imagebit = (char *)malloc(sizeof(char)*imageSize * 16);
base64_encode(imageBin, imageBase64, imageSize);
bitlong = base64tobit(imageBase64, imagebit, imageSize);
int dest = 1, sour = 1;
char binary_array[30][3168] = { 0 };//初始数组
printf("请输入目的地址和发送地址:\n");
printf("dest=");
scanf_s("%d", &dest);
printf("sour=");
scanf_s("%d", &sour);
endline = Input(imagebit, imageSize, bitlong, dest, sour, binary_array);

printf("endline:%d\n", endline);

char crc_data[30][3500] = { 0 };
char Crc_data[30][framelong_send] = { 0 };
//给第一帧成
```

```
char Crcend_line[framelong_send] = { 0 };
char frameend_line[3500] = { 0 };
CRC(binary_array[0], Crcend_line, p);
make_frame(Crcend_line, frameend_line);
//成其他帧
for (i = 1; i <= endline; i++)
{
    CRC(binary_array[i], Crc_data[i], p);
    make_frame(Crc_data[i], crc_data[i]);
}
for (i = 1; i <= endline; i++)
{
    memcpy(framebuff[i], crc_data[i], 3500);
}
/*********************************/
/*
char ori[101] = { "abcdefghij0123456789~!@#$%^&*(水中月揽天上月，眼前人即意中人)余大师" };
char binary_array[32][32] = { 0 };
endline = Input(ori, binary_array) + 1;
printf("endline:%d\n", endline);
char end_line[datalong_send] = { 0 };
code(endline, end_line, 32);
*/




//窗口
char arraybuff[4][64] = { 0 };
int number = 0;//指向 framebuff
//循环队列标志
int Select = 0;
int begin = 0;
int finish = 3;
//接受数字
char restore_data[3500] = { 0 };
//测试参数
int test;
//测试数据
char newdata[3500] = { 0 };
char rawdata[MAX_BUFFSIZE] = { 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0 };
```

```
//初始化窗口

/**************************/
/**************************/
char Initial_route[4][6] = { { 0, 3, 1, 0, 3, 2 }, { 1, 3, 0, 1, 2, 3 }, { 2, 3, 0, 2, 1, 3 }, { 3,
2, 1, 3, 1, 2 } };
char Route[2][6] = { 0 };
char Send_route[3168] = { 0 };
int port_host[4] = { 0 };
int host1;
int host2;
int port_choice = 0;
int flag_route = 0;
int initial_choice = 0;
printf("请输入本机端口号: ");
scanf_s("%d", &initial_choice);
printf("请输入与本机对应端口相连的主机号:\n");
printf("0 号端口对应主机号: ");
scanf_s("%d", &host1);
printf("1 号端口对应主机号: ");
scanf_s("%d", &host2);
int HOST;
printf("输入目标主机号: ");
scanf_s("%d", &HOST);
port_host[host1] = 0;
port_host[host2] = 1;
for (i = 0; i < 4; i++)
{
    if ((i != host1) && (i != host2))
    {
        port_host[i] = 100;
    }
}
int *send;
int lll = 0;
char route_crc[3173] = { 0 };
char route_frame[3500] = { 0 };
int MMM = 2;
/**************************/
while (1) {
    make_fdlist(&sock_list, &readfds);
    //本例程采用了基于 select 机制，不断发送测试数据，和接收测试数据，
也可以采用多线程，一线专发送，一线专接收的方案
    selretval = select(0, &readfds, &writefds, &exceptfds, &timeout);
```

```
if (selretval == SOCKET_ERROR) {
    retval = WSAGetLastError();
    break;
}
/*创建路由信息*/
if ((flag_route == 0) && (lll == 0))
{
    printf("按下任意键进行生成路由表\n");
    lll++;
}
while (MMM)
{
    make_fdlist(&sock_list, &readfds);
    //本例程采用了基于 select 机制，不断发送测试数据，和接收测试数据，也可以采用多线程，一线专发送，一线专接收的方案
    selretval = select(0, &readfds, &writefds, &exceptfds, &timeout);
    if (selretval == SOCKET_ERROR) {
        retval = WSAGetLastError();
        break;
    }
    if (flag_route<2)
    {
        if (FD_ISSET(sock_list.MainSock, &readfds)) {
            //从网络层收到数据
            retval = recv(sock_list.MainSock, buf, MAX_BUFFSIZE, 0);
            if (retval > 0)
            {
                for (i = 0; i < 12; i++)
                {
                    Send_route[i] = buf[i];


                }
                transferB(Send_route, Route[flag_route]);
                flag_route++;
            }
        }
    }
    if (flag_route == 2)
    {
        printf("已经接受其他路由表，开始计算并生成路由表");
        //char route[2][6] = { { 3,2,1,3,4,2 },{ 0,3,1,0,10,2 } };//第一行表示：3->1 距离为 2 ； 3->2 距离为 4
```

```
                    //int i;
                    send = IP(initial_choice, Route);
                    printf("路由表信息：\n");
                    printf("目的主机　下一站主机　　本地所连端口号\n");
                    for (i = 0; i < 4; i++)
                    {
                        printf("%d                %d                %d\n", i, send[i],
port_host[send[i]]);
                    }
                    flag_route++;
                    MMM--;
                }

                if (_kbhit()) {

                    _getch();
                    printf("开始创建路由\n");
                    remote_addr.sin_port = htons(sock_list.MainPort);
                    s = sock_list.MainSock;
                    transferA(Initial_route[initial_choice], Send_route);
                    CRC(Send_route, route_crc, p);
                    make_frame(route_crc, route_frame);
                    retval = sendto(sock_list.MainSock, route_frame, SEND_L, 0,
(sockaddr *)&remote_addr, sizeof(remote_addr));
                    if (retval > 0)
                        printf("已发送初始路由表，开始等待接受其他路由表\n");
                    MMM--;

                }
                FD_ZERO(&readfds);
                FD_ZERO(&writefds);
                FD_ZERO(&exceptfds);
            }


            if (_kbhit()) {
                _getch();
                remote_addr.sin_port = htons(sock_list.MainPort);
                //发送一份测试数据到接口 0
                /*
                发数据的时候需要路由表
                获取 Goal
                制造端口选择参数 F
                每次发送数据之前先将 F 放到 sendbuf 的第 0 位
```

```
                retval = sendto(sock_list.MainSock, sendbuf, 5, 0, (sockaddr
*)&remote_addr, sizeof(remote_addr));
                if (retval > 0)
                printf("发送一份测试数据\n");

                */
                s = sock_list.MainSock;
                F = port_host[send[HOST]];
                Add(frameend_line, sendbuf, F);
                retval = sendto(sock_list.MainSock, sendbuf, SEND_L, 0, (sockaddr
*)&remote_addr, sizeof(remote_addr));
                if (retval > 0)
                    printf("send endline\n");
                while (K)
                {
                    FD_ZERO(&readfds);

                    FD_SET(sock_list.MainSock, &readfds);
                    //进入 Selcet 等待，5s

                    Select = select(s + 1, &readfds, NULL, NULL, NULL);

                    //判读是否可读
                    if (FD_ISSET(s, &readfds) != 0)
                    {
                        //接受数据，retval=recv(s,rawdata,256,0);
                        retval = recv(s, rawdata, MAX_BUFFSIZE, 0);
                        //    retval = recv(s, buf, 256, 0);//recv 返回的是接受数据长度
                        //如果数据非空进行操作
                        if (retval)
                        {
                            if (number < endline)
                            {
                                number++;
                                //Seq++;
                                //    Seq = Seq % 4;
                                Add(framebuff[number], sendbuf, F);
                                retval = sendto(sock_list.MainSock, sendbuf, SEND_L,
0, (sockaddr *)&remote_addr, sizeof(remote_addr));
                                printf("sendone\n");
                                printf("updated and send,number=%d", number);
                                printf("\n");

                                if (number >= (endline))
```

```
                    {
                        K = 0;
                        printf("这次数据已发送完成\n");
                    }
                }
            }
            //无错误发生;
            //进行正确发送，操作



        }

    }


}

if (flag_route >= 2)
{
    if (FD_ISSET(sock_list.MainSock, &readfds)) {
        //从网络层收到数据
        retval = recv(sock_list.MainSock, buf, MAX_BUFFSIZE, 0);
        if (retval > 0) {
            printf("收到数据\n");
            /*
            if(flag<2)
            {
            for(i=0;i<length_restoredata;i++)
            {
            Pre_route[flag][i]=buf[i];
            }
            flag++;
            }
            成路由表
            if(flag==2)
            {
            make_route(Pre_route,Route);
            flag++;
            }
            if(flag==3)
            {
            //接受数据
            if (count == -1)
```

```
{
uncode(buf, &endline);
printf("send endline %d", endline);
printf("\n");
count++;
}
else
{
for (i = 0; i < length_restore; i++) {
right_store[i + right_number] = buf[i];
}
right_number = right_number + i;
endline--;
if (endline == 0)
{
k = 0;
}
//将 ack+1 后经编码成帧后发送;
//send_ack(seq, &ACK, s);
printf("send endline %d", endline);
printf("\n");

}

}

}
if (k == 0)
{
tranlate(right_store, out);
printf("接收到如下信息： ");
for (int i = 0; i<100; i++) {
printf("%c", out[i]);

}
k = -1;
count = -1;
}

}
*/
if (count == -1)
{
	//uncode(buf, &endline);
```

```
                rec_Imag_length = Output_imagelong(buf);
                rec_bit_length = Output_bitlong(buf);
                endline = rec_bit_length / 3160 + 1;
                printf("send endline %d", endline);
                printf("\n");
                count++;
            }
            else
            {
                for (i = 0; i < length_restore; i++) {
                    pro[j][i] = buf[i];
                }
                j++;
                endline--;
                if (endline == 0)
                {
                    k = 0;
                }
                //将 ack+1 后经编码成帧后发送;
                //send_ack(seq, &ACK, s);
                printf("send endline %d", endline);
                printf("\n");

            }

        }

    }
    if (k == 0)
    {
        /*
        tranlate(right_store, out);
        printf("接收到如下信息: ");
        for (int i = 0; i<100; i++) {
        printf("%c", out[i]);

        }

        printf("hello world\n");
        system("pause");
        k = -1;
        count = -1;
        j = 1;
        */
```

```
                    right_store = (char *)malloc(sizeof(char)*rec_Imag_length * 16);
                    imageOutput = (char *)malloc(sizeof(char)*rec_Imag_length);
                    imageBase64_2 = (char *)malloc(sizeof(char)*rec_Imag_length * 2);
                    Output(right_store, rec_bit_length, pro);
                    //    tranlate(right_store, out);

                    bittobase64(imageBase64_2, right_store, rec_Imag_length);
                    base64_decode(imageBase64_2, imageOutput);

                    //写图，输入为图片大小 imageSize 和图片原码 imageOutput，输出为图片
                    fp = fopen("C:\\Users\\hp\\Desktop\\789.jpg", "wb");     //待写入图片地址
                    fwrite(imageOutput, 1, rec_Imag_length, fp);
                    fclose(fp);
                    printf("图片已接受完成\n");
                    k = -1;
                    count = -1;
                    j = 1;
                }
            }


            FD_ZERO(&readfds);
            FD_ZERO(&writefds);
            FD_ZERO(&exceptfds);
        }
        closesocket(sock_list.MainSock);
        WSACleanup();
        return 0;
    }
```