

# 计算机网络实验三

学号：2011273

姓名：姚翔载

## 实验原理

### 建立连接

通过模仿TCP三次握手四次挥手的原理来建立连接。

原理如下：

三次握手：

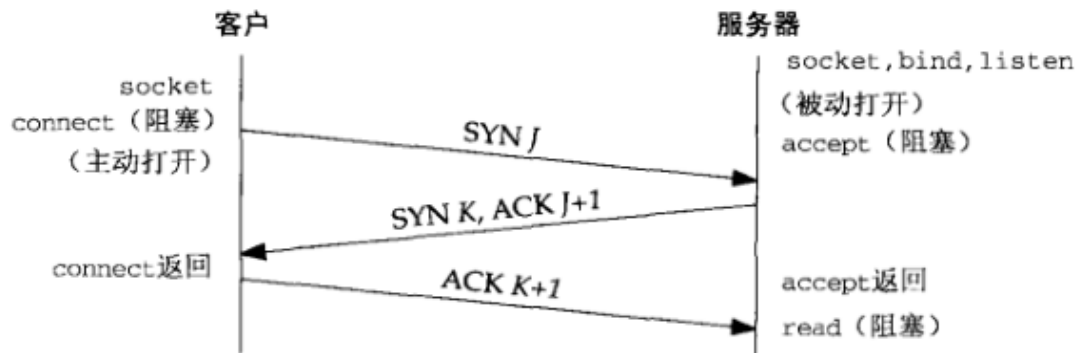
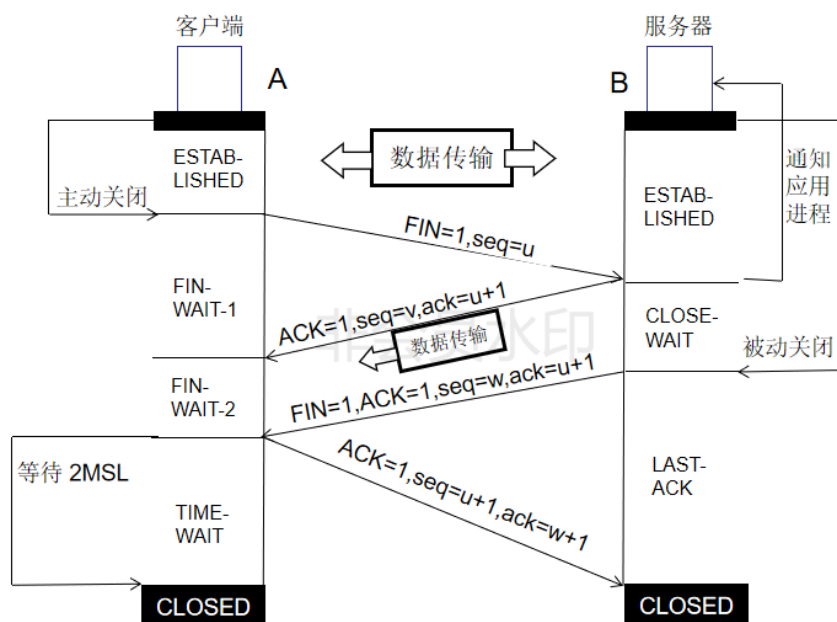


图2-2 TCP的三路握手 [log.csdn.net/jun2016425](http://log.csdn.net/jun2016425)

四次挥手：



<https://blog.csdn.net/C9A0MA>

将三次握手和四次挥手需要的各个标志位压缩在UDP包伪首部传输，具体见代码说明。三次握手四次挥手的详细原理在上次实验中已阐述，不再赘述。

## 差错检测

校验和的计算，与课上讲的相同。产生伪首部，用0补齐，与数据报（如果有）一起看成一整个序列，进行反码求和，并写入校验和域段。

## 3.3 用户数据报协议UDP



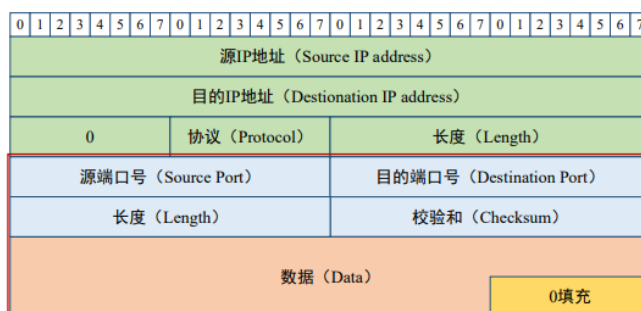
### ■ UDP校验和的计算方法

#### 发送端：

- 产生伪首部，校验和域段清0，将数据报用0补齐为16位整数倍
- 将伪首部和数据报一起看成16位整数序列
- 进行16位二进制反码求和运算，计算结果取反写入校验和域段

#### 接收端：

- 产生伪首部，将数据报用0补齐为16为整数倍
- 按16位整数序列，采用16位二进制反码求和运算
- 如果计算结果位全1，没有检测到错误；否则，说明数据报存在差错



```

u_short cksum(u_short *buf, int count)
{
    register u_long sum = 0;
    while (count-->0)
    {
        sum += *buf++;
        if (sum & 0xFFFF0000)
        {
            sum &= 0xFFFF;
            sum++;
        }
    }
    Return ~(sum & 0xFFFF);
}

```

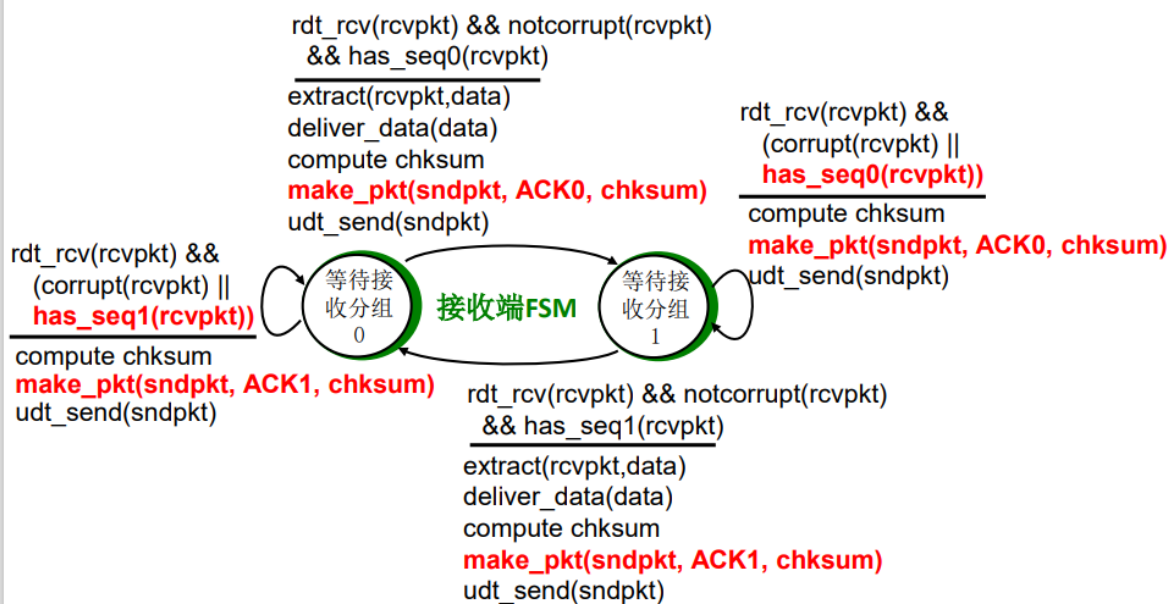
代码结构与伪代码相同，buf为存储的一整个序列指针，count实际上是要计算序列的位数。

这个过程的实现实操过程中，需要注意在计算除了校验和域段的校验和时，需要事先将校验和域段清零（用memset函数，或直接将结构体中的校验和部分清零），再计算校验和写入或比较。

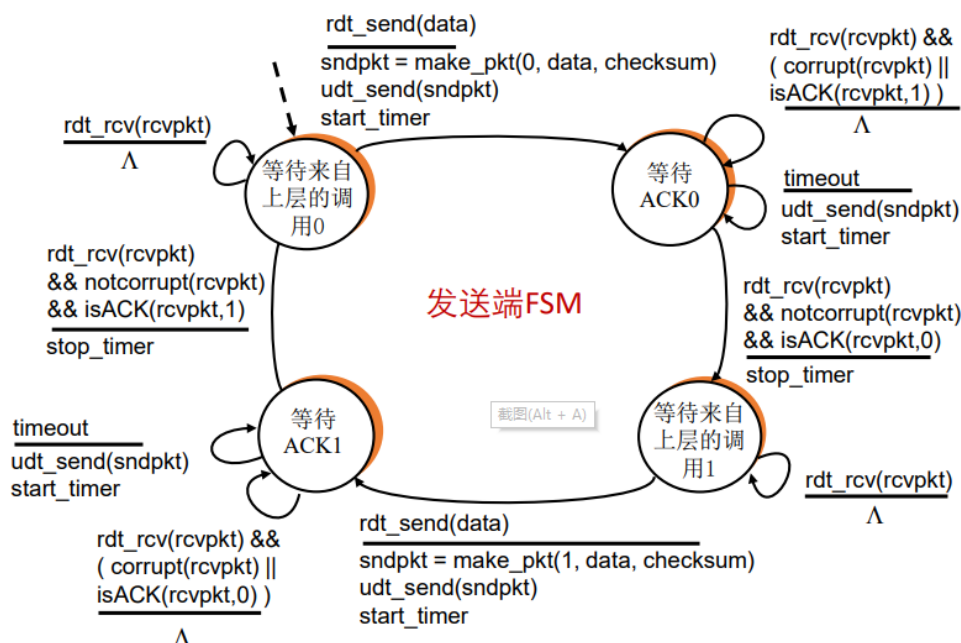
## 确认重传

模仿rdt3.0协议实现，但保留了序列号（方便观察与debug）而不是一位位的ACK号。

### ■ rdt2.2: 接收端状态机



## ■ rdt3.0：发送端状态机



(rdt3.0接收端状态机与2.2基本相同)

因为采用了序列号SEQ而不是0或1的ACK，因此实际上实现的协议的FSM应该是一个很大的“圆圈”，从最小的SEQ到最大的SEQ。其余部分与上述相同，例如超时重传当前的pkt、数据包位反转、校验失败等操作都与上图的FSM相同。

## 停等机制

停等协议大致是指，发送端发送一个包，必须等待接收端的信号回应后才能继续发出下一个包；与此同时接收端必须在接收后发回一个应答信号。如果超过了计时的限制就采用超时重传机制。这一部分实际上已经包含在rdt3.0的设计实现当中与api的既有实现当中。例如一直循环调用recvfrom函数，直到对方sendto送到之后才执行下一步。

## 单向传输

两个cpp文件，两个可执行文件，分别绑定不同的ip以及端口（发送至对方的ip端口，或路由器的ip和端口），并根据FSM和握手挥手来分别进行实现。

## 日志输出

使用FILE的fopen方法来读取路径下的txt文件并进行输出即可。注意到fopen不是线程安全的，即在本实验中，如果server和client同时打开文件进行写操作，可能产生冲突（实际上实验过程中确实有这一点）。由于这部分并不是特别重要，除了命令行输出外，写文件的操作改为server和client分别写不同的txt文件。

## 代码思路

### 建立连接

### 初始化

使用UDP进行实现。

C语言的UDP编程需要先初始化DLL、

```
WSADATA wsaData;
WSAStartup(MAKEWORD(2, 2), &wsaData);
```

具体的创建过程实际上与第一次实验基本相同，不过是替换成了UDP。

```
SOCKET sock;
SOCKADDR_IN sockAddr;
sock = socket(AF_INET, SOCK_DGRAM, 0); //使用UDP
...
//*****本地socket*****
//绑定套接字
char cIntIP[20];
string mylocalip="127.4.5.6";
myippointer=cIntIP;
myippointer = strcpy(myippointer,mylocalip.c_str());

SOCKADDR_IN addrLocal; // 发送端IP和端口
memset(&addrLocal, 0, sizeof(SOCKADDR_IN));
addrLocal.sin_family = AF_INET;
addrLocal.sin_addr.s_addr = inet_addr(cIntIP);
addrLocal.sin_port = htons(9876);
if(bind(sock, (SOCKADDR*)&addrLocal, sizeof(addrLocal))==-1) //绑定
{
    perror("BIND FAILED!");
    exit(-1);
}
```

例如如上的代码初始化了一个socket和sockaddr，并将他绑定到本地，也即编译后的可执行文件运行后的ip和端口就是127.4.5.6:9876。需要转发到路由器，同时对路由器的sockaddr进行同样操作的初始化，不要绑定，并在之后的sendto和recvfrom中调用这个路由器的sockaddr即可。

程序最后使用WSACleanup函数终止DLL的使用即可。

## 三次握手&四次挥手

根据三次握手的原理来实现。

例如，客户端接收第一次握手：

```
while(1)
{ //第一次recv不重传，不用计时
    if(recvfrom(servSock,buffer,sizeof(header),0,
(sockaddr*)&cIntAddr,&cIntAddrLen)<0)
    {
        addtoservlog("第一次握手接收错误",myippointer);
        printf("第一次握手接收错误\n");
        return -1;
    }
    addtoservlog("第一次握手接收成功，进行校验",myippointer);
    printf("第一次握手接收了\n");
    memcpy(&header,buffer,sizeof(header));
    u_short checksum_from_recv=header.checksum;
    header.checksum=0;
    calc_chksum_rst=CalcChecksum((u_short*)&header,sizeof(header));
    if(header.flag==HSK10F3&&calc_chksum_rst==checksum_from_recv)
    {
```

```

        addtoservlog("第一次握手校验成功",myippointer);
        printf("第一次握手校验成功\n");
        break;
    }else{
        addtoservlog("第一次握手校验失败",myippointer);
        printf("校验码错误\n");
    }
}
printf("第一次握手结束\n");
addtoservlog("第一次握手完毕",myippointer);

```

循环调用recvfrom来接收发送端发来的第一次握手消息。使用memcpy将伪首部对应位按位复制到当前的一个header实例中，取出它的校验和，将收到的buffer的校验和位全部格式化为0，之后再计算校验和，并与取出的校验和进行比较。如果校验和比较成功，且经过flag的比较确实是第一次握手该有的flag（一个宏定义，见**伪首部**一节），即syn=1，ack=0，则校验成功，进入下一步，否则继续循环监听。addtoservlog是记录到日志文件的函数。

剩余部分的实现基本用了这样的计算校验和并比较的逻辑，并用了相同的一些api（sendto、recvfrom、memcpy等）。

对于需要**超时重传**的部分，开启调用clock()，计算两个clock()返回值的差，与一个阈值比较作为计时器，在循环调用recvfrom监听的过程中，循环体时刻计算是否超时，超时则重传需要重传的数据包并更新计时器即可。例如服务端接收第三次握手：

```

while(recvfrom(servSock,buffer,sizeof(header),0,
(sockaddr*)&cIntAddr,&cIntAddrLen)<=0){
    if(clock()-now_clocktime>MAX_TIME)
    {//超时重传
        header.flag=HSK2OF3;
        header.checksum=0;
        calc_chksum_rst=CalcChecksum((u_short*)&header,sizeof(header));
        memcpy(&header,buffer,sizeof(header));
        if (sendto(servSock,buffer,sizeof(header),0,(sockaddr*)&cIntAddr,
cIntAddrLen) == -1)
        {
            addtoservlog("第二次握手超时重传失败",myippointer);
            return -1;
        }
        now_clocktime=clock();
        printf("第三次握手超时，第二次握手重传发送成功\n");
        addtoservlog("第三次握手超时，重传发送成功",myippointer);
    }
}
}

```

制作伪首部，计算校验和并写入，sendto目标socket，更新计时器。

客户端和服务端的其余部分实现的api和逻辑基本与上述相同，按照三次握手四次挥手来实现即可。具体的代码和一些特殊说明具体可见打包的代码和注释，这里不再进行赘述。

## 差错检测

原理如实验原理一节所述，参考伪代码实现即可。

```

u_short CalcChecksum(u_short* bufin,int size)
{
    int count = (size + 1) / 2;

```

```

//u_short* buf = new u_short[size+1];
u_short* buf = (u_short*)malloc(size + 1);
memset(buf, 0, size + 1);
memcpy(buf, bufin, size);
u_long sum = 0;
while (count--) {
    sum += *buf++;
    if (sum & 0xffff0000) {
        sum &= 0xffff;
        sum++;
    }
}
return ~(sum & 0xffff);
}

```

输入一个缓冲区char型指针和长度，返回一个16位（u\_short）的反码和。

## 伪首部

```

struct HEADER
{
    u_short checksum=0;
    u_short datasize=0;
    //为把四个用到的flag“压缩”成四位
    //用一个unsigned char记录
    //使用低4位
    //分别是OVER FIN ACK SYN
    //其中over是指一个文件发送完毕后发送的标志位
    unsigned char flag=0;
    unsigned char SEQ=0;
    HEADER()
    {
        checksum=0;
        datasize=0;
        flag=0;
        SEQ=0;
    }
};

```

使用两个16位的u\_short，分别存储校验和和数据的长度。再用三个8位的unsigned char存储flag、ack和SEQ，flag的说明如上。并声明了构造函数。

```

#define HSK10F3 0x1 //SYN=1 ACK=0
#define HSK20F3 0x2 //SYN=0,ACK=1
#define HSK30F3 0x3 //SYN=1 ACK=1
#define WAV10F4 0x4 //SYN=0 ACK=0 FIN=1
#define WAV20F4 0x2 //SYN=0 ACK=1 FIN=0
#define WAV30F4 0x6 //SYN=0 ACK=1 FIN=1
#define WAV40F4 0x2 //SYN=0 ACK=1 FIN=0
#define OVERFLAG 0x8 //1000,OVER=1
#define INITFLAG 0x0 //发送数据包，全部为0

```

并宏定义了一些参数，都是flag来用。例如第三次握手时SYN=1，ACK=1，那么“翻译”到unsigned char的对应位上就是16进制的3（00000011）。

## 确认重传

按照rdt3.0实现，但保留了SEQ（在实验中具体为8位，能表示0-255的seq号）而不是ack。

### 接收端

#### 初始化

首先对需要用到的一些伪首部以及seq、指针等进行初始化。

```
long int tailpointer=0;//由于按包来传输，记录每次拼接后末尾的指针便于下一次拼接
HEADER header;
HEADER temp1;
u_short calc_chksum_rst;
```

```
char* buffer = new char[MAX_BUFFER_SIZE+sizeof(header)];
```

一个缓冲区的大小最大是数据的大小+伪首部的大小

#### 接收

基本的api实际上与三次握手四次挥手相同。

首先循环中先阻塞地调用recvfrom，试图从发送端接收消息。它的返回值可以直接作为整个udp包接收的缓冲区大小（即数据的大小+伪首部的大小）。返回值小于0为错误，猜想按丢包一样的方法处理即可，但这里直接返回错误并不再接收（函数直接返回）。

```
int recvLen=recvfrom(servSock,buffer,sizeof(header)+MAX_BUFFER_SIZE,0,
(sockaddr*)&clntAddr,&clntAddrLen);
if(recvLen<0)
{
    addtoservlog("udp接收错误",myippointer);
    printf("接收错误\n");
    return -1;
}
```

接收后校验。需要对校验码进行校验，成功后再进入下一步（不成功则continue外层循环，继续调用recvfrom等待发送端发回这个错误的数据包即可。校验码无误则可以找出伪首部header中的flag以及序列号SEQ。

flag中over一位为1，则代表对方已经发送来所有数据包传输完毕的消息，再经过一些后续处理就可以直接退出了。如果flag全0，则表示只是正常的数据包传输，那么对数据包的序列号进行检查，看是否是当前记录的“应该”收到的那个序列号。如果相等，则返回确认的消息（rdt3.0中是ack，这里直接设置seq，可以达到一样的效果。）如果不相等，这说明传输的序列有误，此时根据有限状态机，需要重发“应该”收到的那个序列号（即rdt3.0中相反的那个ack）。实际上在代码中可以直接通过一个结构体的实例来记录这个“应该”的seq所在的伪首部，重新计算校验和，发回这个伪首部（数据部分没有即可）。

```
if(servSeq!=(int)header.SEQ)
{
    //has_seq1(),重传，当前记录的应该的seq与收到的seq不同
    //buffer不变，checksum不变，传回当前的seq
    unsigned char waitSEQ=header.SEQ;
    header=HEADER();
    header.SEQ=waitSEQ;
    header.checksum=0;
    //只传回一个header，不用传回数据
    //也不用数据计算校验和，否则那边无法校验
}
```



```

//只需要seq
calc_chksum_rst=CalcChecksum((u_short*)&header,sizeof(header));
header.checksum=calc_chksum_rst;
memcpy(buffer,&header,sizeof(header));
//发回
if (sendto(servSock,buffer,sizeof(header),0,
(sockaddr*)&clntAddr, clntAddrLen) == -1)
{
    printf("seq值不正确, 重新发送当前seq值错误\n");
    return -1;
}
addtoservlog("SEQ值不正确, 重新发送当前SEQ值成功",myippointer);
printf("%d!=%d,seq不对信息发送成功\n",servSeq,waitSEQ);
}
else
{
    //has_seq0,发回正确的seq, 更新服务端的seq
    ...
    这里发送回正确的seq, 也即rdt3.0中的ack值
    servSeq++;
    if(servSeq>255)//SEQ只有八位
    {
        servSeq=0;
    }
    //更新“应该的”seq。用八位记录seq, 那么大于255时需要回0
}

```

在校验码和seq校验均通过之后, 就是fsm中的extract and deliver data的部分。除了记录日志之外, 将接受到的包通过指针“接”到整个数据序列之后, 最后接受所有包完毕后就能组成发来的文件的为二进制序列的char数组。

```

char* tail=new char[recvLen-sizeof(header)];
//把除了头部的缓冲区（及数据）解码记录
memcpy(tail,buffer+sizeof(header),recvLen-sizeof(header));
//将这部分作为尾部, 接到已接收的包后面
memcpy(fullData+tailpointer,tail,recvLen-sizeof(header));
//更新尾部指针
tailpointer=tailpointer+recvLen-sizeof(header);

```

将上述部分封装为一个函数, 最后返回-1则代表整个有一些错误, 返回一个整数值, 为接收到所有数据的长度, 则代表接受正确 (在当前层面的rdt3.0可以处理的范围内)

通过一个函数调用上述封装好的函数, 根据读取的消息、存储二进制序列的char数组指针以及长度等, 通过ofstream的方法来写入文件。最后能在同级目录下找到接收好的文件。

```

void RecvFileHelper()
{
    char* filename=new char[10000];
    char* filebuff=new char[INT_MAX];
    int lenn=sizeof(sockAddr);
    //发送端会发回文件名和文件的char数组
    int nameLen=RecvFile(servSock,sockAddr, lenn,filename);
    memset(message,0,sizeof(message));
    sprintf(message,"文件名:%s接收完毕",filename);
    addtoservlog((const char*)message,myippointer);
    printf("%s\n",message);
    int fileLen=RecvFile(servSock,sockAddr, lenn,filebuff);
}

```

```

memset(message,0,sizeof(message));
sprintf(message,"文件:%s接收完毕,长度为%dbytes",filename,fileLen);
addtoservlog((const char*)message,myippointer);
printf("%s\n",message);
string namestr=filename;
// ofstream fout(namestr.c_str(),ofstream::binary);
ofstream fout(namestr.c_str(),ios::out|ios::binary);
for(int i=0;i<fileLen;i++){fout<<filebuff[i];}
fout.close();
addtoservlog("文件写入完毕",myippointer);
printf("文件写入完毕\n");
}

```

## 发送端

客户端发送，首先根据读取文件为二进制的结果，将文件存入到表示二进制的char数组中，以及记录出它的长度。分成数据包是通过对这个数组的指针操作进行的：设置一个最大的数据序列大小，对数组大小进行整除操作，能够得到包的个数和包的大小（前面的包即为最大的数据序列大小，最后一个为整除的余数大小，如下：

```

int pktnum=dataLen%MAX_BUFFER_SIZE==0?
dataLen/MAX_BUFFER_SIZE:dataLen/MAX_BUFFER_SIZE+1;
//计算包的个数

```

```

char* nowPktPointer=fullData+nowpkt*MAX_BUFFER_SIZE;
memcpy(buffer,&header,sizeof(header));
memcpy(buffer+sizeof(header),nowPktPointer,pktlen);
calc_chksum_rst=CalcChecksum((u_short*)buffer,sizeof(header)+pktlen);
//通过对指针的处理来分配每个包的char型的部分序列。

```

最后计算校验和，调用sendto发出。注意包的长度是伪首部+数据部分的总长度，由于参数命名相同，在写这部分时盲目ctrlcv握手挥手部分的sendto形参表，导致debug用了很久。

发送完后等待接收端的确认消息。同样循环阻塞调用recvfrom接收，接收返回的udp包（实际上只有一个头部），并进行校验。在recv所处while调用的外层还有一个while，是为了handle接收接收端返回的消息后是否因为有误继续recv的处理。具体可见代码以及注释。

memcpy时同样需要注意整个udp包的长度是伪首部+数据部分的长度。同样的方法处理超时重传

```

mode = 1;
ioctlsocket(cIntSock, FIONBIO, &mode);
while(recvfrom(cIntSock, buffer, sizeof(header), 0,
(sockaddr*)&servAddr, &servAddrLen)<0){
    if(clock()-now_clocktime>MAX_TIME){

        memcpy(buffer,nowbuffer,sizeof(header)+pktlen);
        if(sendto(cIntSock, buffer, sizeof(header)+pktlen, 0,
(sockaddr*)&servAddr, servAddrLen)<0){
            addtocIntlog("确认消息超时, udp包重传发送错误",myippointer);
            printf("超时重传错误\n");
            return -1;
        }
        addtocIntlog("确认消息超时, udp包重传发送成功",myippointer);
        printf("超时重传\n");
        now_clocktime=clock();
    }
}

```

```
    }  
}
```

注意需要用`ioctlsocket`来设置非阻塞模式（并在处理完毕后设置回阻塞模式）

同样的方法对校验码和seq进行校验。同样要比较发回的包的seq和当前记录“应该”的seq。实际上在rdt3.0中校验码不通过（比特位反转）和序列号（ack）错误都是和超时重传一块处理即可，即continue到中层循环的开始，调用`recvfrom`进行接收。

如果没有continue，会跳出中层循环，此时可以设置socket回阻塞模式，并更新当前的pkt序号（便于移动指针）以及更新记录的“应该的”seq号。

如果遍历的当前pkt序号（从0开始）等于计算出的包个数-1时，表示所有包都发送完毕了，这时发回一个flag的over位为1的header的包，经过后续处理退出即可。

将上述封装为一个函数，外层一个函数调用它。外层的函数通过`ifstream`进行读文件，记录到char型数组，并记录指针和数据长度，传到上述函数的参数中方便分包处理。由于`ofstream`需要一个文件名来打开一个文件，因此可以在发送端先后发送文件名和文件本身，方便`ofstream`操作。

## 停等机制

实际上在上述过程的超时重传等方面就有体现，这里不再赘述。

## 日志输出

由于FILE的fopen不是线程安全的，即发送端和接收端同时fopen一个文件并写入时可能会混乱或乱码，为避免在不必要的方面花费太多时间，分别用两个文件进行日志记录。

日志记录的函数大概如下：

```
void addtoservlog(const char* initmsg,char* ip)
{
    FILE *logs = fopen("../servlog.txt", "a+");
    if(logs== NULL)
    {
        printf("open file error: \n");
    }else{
        char buf[MAX_MSG_SIZE];
        memset(buf,0,sizeof(buf));
        sprintf(buf, "%s",initmsg);
        //为消息添加时间戳和用户ip
        addtimestamp(buf);
        adduserip(buf,ip);
        fputs(buf,logs);
        fputs("\n",logs);
        fclose(logs);
    }
}
```

打开同级文件夹下的一个txt文件并在尾部写入（a+）。addtimestamp和adduserip是向消息的string头部添加时间戳和ip信息，直接照搬了第一次实验中的函数。

## 结果展示

工作区

loghdh.h
E:\workplace> c:\space> h: loghdh.h @ addtoservlog(const char \*, char \*)
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
void addtoservlog(const char\* initmsg, char\* ip)
{
FILE \*logs = fopen("../servlog.txt", "a+");
if(logs== NULL)
{
printf("open file error: \n");
}
else{
char buf[MAX MSG SIZE];
}
}

myserver.cpp
E:\workplace> c:\space> C- myserver.cpp @ init0
452 //
453 // if (MAX\_CLIENT\_NUM == i-1){
454 // char const \*msg\_room\_full = "对不起, 聊天室已经满了!";
455 // send(cIntSock, msg\_room\_full, strlen(msg\_room\_full), 0);
456 // closesocket(cIntSock);
457 //
458 // }
459 //
460 // // closesocket(cIntSock);
461 // closesocket(servSock);
462 //
463 int main()
464 {
465 Init();
466 //start();
467 //防止使用 DLL
468 WSACleanup();
469 system("pause");
470 return 0;
471 }

myclient.cpp
E:\workplace> c:\space> C- myclient.cpp @ SendFileABinary(SOCKET &, SOCKADDR\_IN &, int &, char \*, int)
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
286





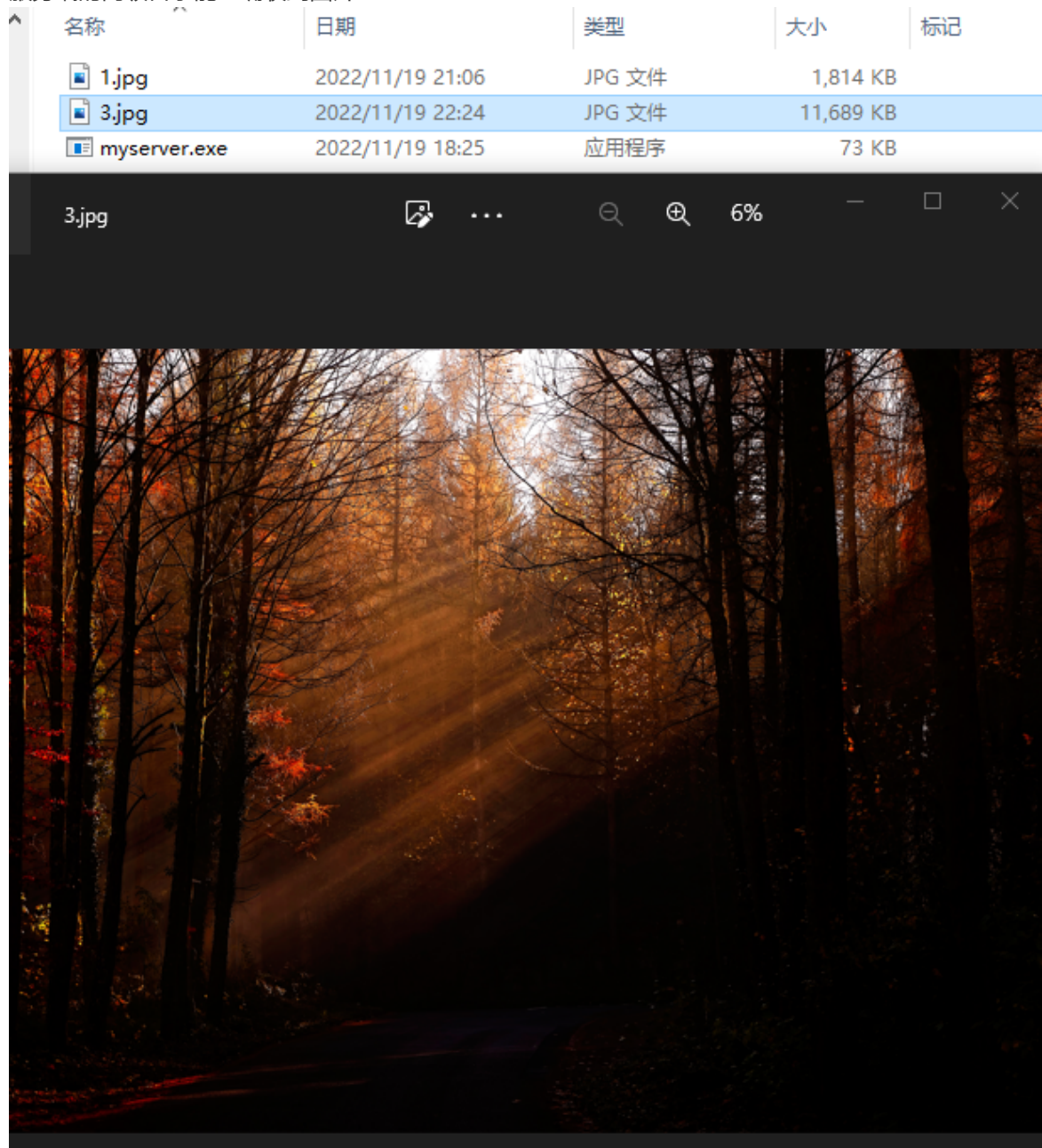


## 日志输出，包括四次挥手和文件的传输结果

```
[127.4.5.6][Sat Nov 19 22:24:53 2022]接收到确认信息，进行校验
[127.4.5.6][Sat Nov 19 22:24:53 2022]udpb包确认信息校验通过，传输下一个包
[127.4.5.6][Sat Nov 19 22:24:53 2022]成功发送 1024 bytes数据，序列号为167
[127.4.5.6][Sat Nov 19 22:24:53 2022]udpb包发送成功
[127.4.5.6][Sat Nov 19 22:24:53 2022]接收到确认信息，进行校验
[127.4.5.6][Sat Nov 19 22:24:53 2022]udpb包确认信息校验通过，传输下一个包
[127.4.5.6][Sat Nov 19 22:24:53 2022]成功发送 482 bytes数据，序列号为168
[127.4.5.6][Sat Nov 19 22:24:53 2022]udpb包发送成功
[127.4.5.6][Sat Nov 19 22:24:53 2022]接收到确认信息，进行校验
[127.4.5.6][Sat Nov 19 22:24:53 2022]udpb包确认信息校验通过，传输下一个包
[127.4.5.6][Sat Nov 19 22:24:53 2022]所有包发送完成，发送结束标志
[127.4.5.6][Sat Nov 19 22:24:53 2022]文件:3.jpg接收完毕,长度为11968994bytes
[127.4.5.6][Sat Nov 19 22:24:53 2022]第一次挥手发送成功，开启计时器
[127.4.5.6][Sat Nov 19 22:24:53 2022]第二次挥手接收超时，重传第一次挥手成功
[127.4.5.6][Sat Nov 19 22:24:54 2022]第二次挥手接收成功，进行校验，关闭计时器
[127.4.5.6][Sat Nov 19 22:24:54 2022]第三次挥手接收成功，进行校验
[127.4.5.6][Sat Nov 19 22:24:54 2022]第三次挥手接收成功
[127.4.5.6][Sat Nov 19 22:24:54 2022]第四次挥手发送成功，开启计时器
[127.4.5.6][Sat Nov 19 22:24:55 2022]第四次挥手后经2*MSL未收到数据，成功断开
[127.4.5.6][Sat Nov 19 22:24:55 2022]四次挥手断连成功，返回值为1
```

```
[127.1.2.3][Sat Nov 19 22:24:52 2022]进入监听状态，等待连接
[127.1.2.3][Sat Nov 19 22:06:32 2022]第一次握手接收成功，进行校验
[127.1.2.3][Sat Nov 19 22:06:32 2022]第一次握手校验成功
[127.1.2.3][Sat Nov 19 22:06:32 2022]第一次挥手完毕
[127.1.2.3][Sat Nov 19 22:06:32 2022]第二次挥手发送成功，开启计时器
[127.1.2.3][Sat Nov 19 22:06:32 2022]第三次握手接收成功，关闭计时器
[127.1.2.3][Sat Nov 19 22:06:32 2022]三次握手完成，已建立连接
[127.1.2.3][Sat Nov 19 22:06:32 2022]三次握手连接成功，返回值为1
[127.1.2.3][Sat Nov 19 22:06:32 2022]准备接收文件
[127.1.2.3][Sat Nov 19 22:06:42 2022]udpb包接收成功
[127.1.2.3][Sat Nov 19 22:06:42 2022]udpb包校验码校验成功
[127.1.2.3][Sat Nov 19 22:06:42 2022]SEQ值验证正确
[127.1.2.3][Sat Nov 19 22:06:42 2022]校验码和SEQ均校验成功，确认消息发送成功
[127.1.2.3][Sat Nov 19 22:06:42 2022]成功收到 5 bytes数据，序列号为0
[127.1.2.3][Sat Nov 19 22:06:42 2022]udpb包接收成功
[127.1.2.3][Sat Nov 19 22:06:42 2022]所有文件接收完毕
[127.1.2.3][Sat Nov 19 22:06:42 2022]文件名:1.jpg接收完毕
[127.1.2.3][Sat Nov 19 22:06:42 2022]准备接收文件
[127.1.2.3][Sat Nov 19 22:06:42 2022]udpb包接收成功
[127.1.2.3][Sat Nov 19 22:06:42 2022]udpb包校验码校验成功
[127.1.2.3][Sat Nov 19 22:06:42 2022]SEQ值验证正确
[127.1.2.3][Sat Nov 19 22:06:42 2022]校验码和SEQ均校验成功，确认消息发送成功
[127.1.2.3][Sat Nov 19 22:06:42 2022]成功收到 1024 bytes数据，序列号为197
[127.1.2.3][Sat Nov 19 22:24:52 2022]udpb包接收成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]udpb包校验码校验成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]SEQ值验证正确
[127.1.2.3][Sat Nov 19 22:24:52 2022]校验码和SEQ均校验成功，确认消息发送成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]成功收到 1024 bytes数据，序列号为198
[127.1.2.3][Sat Nov 19 22:24:52 2022]udpb包接收成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]udpb包校验码校验成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]SEQ值验证正确
[127.1.2.3][Sat Nov 19 22:24:52 2022]校验码和SEQ均校验成功，确认消息发送成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]成功收到 1024 bytes数据，序列号为199
[127.1.2.3][Sat Nov 19 22:24:52 2022]udpb包接收成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]udpb包校验码校验成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]SEQ值验证正确
[127.1.2.3][Sat Nov 19 22:24:52 2022]校验码和SEQ均校验成功，确认消息发送成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]成功收到 1024 bytes数据，序列号为200
[127.1.2.3][Sat Nov 19 22:24:52 2022]udpb包接收成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]udpb包校验码校验成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]SEQ值验证正确
[127.1.2.3][Sat Nov 19 22:24:52 2022]校验码和SEQ均校验成功，确认消息发送成功
[127.1.2.3][Sat Nov 19 22:24:52 2022]成功收到 1024 bytes数据，序列号为201
[127.1.2.3][Sat Nov 19 22:24:52 2022]udpb包接收成功
```

服务端的同级目录能正确收到图片



大小与发送端的测试文件完全相同

位置: E:\workplace\network\_ex\ex3\test\serv

大小: 11.4 MB (11,968,994 字节)

占用空间: 11.4 MB (11,972,608 字节)

(必须完全相同, 例如txt文件最后如果多写入了制表符等等可能整个文件都会显示乱码, 实验中也遇到了这种情况并正确debug)

## 实验心得

网络课的实验并不简单, 不仅需要对课上知识有很深刻的了解, 并且极其需要注重细节。做完实验后真正地对校验码、rdt、停等协议等重要知识有了更加细致的认识。并且实验过程中也因为细节吃了不少亏, 例如: 接收数据包只拿出了伪首部来处理、读文件时index多加了1没退回、没有讲校验码置零就计算新的校验码.....这些过程都是痛并快乐着的过程, 提高了debug能力, 也为后续实验的细节方面、debug方法等提供了很好的思路, 在深夜都debug完之后感觉灵魂得到了升华