### 计算机网络第三次实验报告

网络空间安全学院 物联网工程 2111673 岳志鑫

1. **实验目的**

基于UDP服务设计可靠传输协议并编程实现（3-1）

1. **实验要求**

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、接收确认、超时重传等。流量控制采用停等机制，完成给定测试文件的传输。

➢ 数据报套接字：UDP

➢ 协议设计：数据包格式，发送端和接收端交互，详细完整

➢ 建立连接、断开连接：类似TCP的握手、挥手功能

➢ 差错检验：校验和

➢ 接收确认、超时重传：rdt2.0、rdt2.1、rdt2.2、rtd3.0等，亦可自行设计协议

➢ 单向传输：发送端、接收端

➢ 日志输出：收到/发送数据包的序号、ACK、校验和等，传输时间与吞吐率

➢ 测试文件：必须使用助教发的测试文件（1.jpg、2.jpg、3.jpg、helloworld.txt）

1. **实验内容**

**1.协议设计**

（1）建立连接：

通过握手过程建立连接。发送方发送连接请求数据包，接收方收到连接请求，发送连接确认数据包。

（2）差错检测：

在数据包中添加校验和字段，用于检测数据传输过程中的错误。发送方发送数据包，记录序列号和计算校验和。接收方收到数据包，进行序列号和校验和的检测，如果数据包正确就发送确认，如果数据包错误就重新发送请求。

（3）接收确认：

接收方收到数据后，向发送方发送确认，通知其数据已经接收。

（4）超时重传：

发送方发送数据包，启动定时器。如果发送方收到确认则停止定时器，如果发送方在规定时间内未收到确认，则进行超时重传，重新发送上一个数据包。

（5）流量控制：

发送方发送一个数据包后，等待接收到对应的确认后再发送下一个数据包。接收方收到数据后，发送确认，并等待下一个数据包。

**2.核心代码分析**

（1）数据报结构

struct Head

{

u\_short checksum;//校验和 16位

u\_short datasize;//所包含数据长度 16位

unsigned char flag;//八位，使用后三位表示FIN ACK SYN

unsigned char seq;//八位，传输的序列号

Head()

{

checksum = 0;

datasize = 0;

flag = 0;

seq = 0;

}

};

1. 三次握手（以发送端为例）

首先进行第一次握手，将数据包组装好后发送，标志位设定为SYN

Head head = Head(); //数据首部

head.flag = SYN; //标志设为SYN

head.checksum = check((u\_short\*)&head, sizeof(head)); //计算校验和

char\* buff = new char[sizeof(head)]; //缓冲数组

memcpy(buff, &head, sizeof(head));//将首部放入缓冲数组

if (sendto(socket, buff, sizeof(head), 0, (sockaddr\*)&addr, length) ==SOCKET\_ERROR)

{//发送失败

cout << "【第一次握手失败】" << endl;

return ;

}

cout << "第一次握手成功【SYN】" << endl;

发送成功后等待客户端的回复，用while循环持续接收数据包，如果超时未接收到确认则重新发送第一次握手的数据包，接收到回复则为第二次握手成功

clock\_t handstime = clock(); //记录发送第一次握手时间

u\_long mode = 1;

ioctlsocket(socket, FIONBIO, &mode); //设置非阻塞模式

int handscount1 = 0;//记录超时重传次数

//第二次握手

while (recvfrom(socket, buff, sizeof(head), 0, (sockaddr\*)&addr, &length) <=0)

{//等待接收

if (clock() - handstime > retime)//超时重传

{

memcpy(buff, &head, sizeof(head));//将首部放入缓冲区

sendto(socket, buff, sizeof(head), 0, (sockaddr\*)&addr, length); //再次发送

handstime = clock(); //计时

cout << "【连接超时！等待重传……】" << endl;

handscount1++;

if (handscount1 == handscount) {

cout << "【等待超时】" << endl;

return;

}

}

}

memcpy(&head, buff, sizeof(head)); //ACK正确且检查校验和无误

if (head.flag == ACK && check((u\_short\*)&head, sizeof(head) == 0))

{

cout << "第二次握手成功【SYN ACK】" << endl;

handscount1 = 0;

}

else

{

cout << "【第二次握手失败】" << endl;

return;

}

继续发送第三次握手的数据包并等待回复，如果超时未接收到确认则重新发送第三次握手的数据包，接收到回复则为第三次握手成功

//第三次握手

head.flag = ACK\_SYN; //ACK=1 SYN=1

head.checksum = check((u\_short\*)&head, sizeof(head)); //计算校验和

sendto(socket, (char\*)&head, sizeof(head), 0, (sockaddr\*)&addr, length); //发送握手请求

bool win = 0; //检验是否连接成功的标志

while (clock() - handstime <= retime)

{//等待回应

if (recvfrom(socket, buff, sizeof(head), 0, (sockaddr\*)&addr, &length))

{//收到报文

win = 1;

break;

}

//选择重发

memcpy(buff, &head, sizeof(head));

sendto(socket, buff, sizeof(head), 0, (sockaddr\*)&addr, length);

handstime = clock();

handscount1++;

if (handscount1 == handscount) {

cout << "【等待超时】" << endl;

return;

}

}

if (!win)

{

cout << "【第三次握手失败】" << endl;

return;

}

cout << "第三次握手成功【ACK】" << endl;

1. 校验和计算

校验数据以16位为单位进行累加求和，如果累加和超过16位产生了进位，需将高16位置为0，低16位加一。循环步骤，直至计算完成为止，最后将所取得的结果取反。

u\_short check(u\_short\* head, int size)

{

int count = (size + 1) / 2;//计算循环次数，每次循环计算两个16位的数据

u\_short\* buf = (u\_short\*)malloc(size + 1);//动态分配字符串变量

memset(buf, 0, size + 1);//数组清空

memcpy(buf, head, size);//数组赋值

u\_long checkSum = 0;

while (count--) {

checkSum += \*buf++;//将2个16进制数相加

if (checkSum & 0xffff0000) {//如果相加结果的高十六位大于一，将十六位置零，并将最低位加一

checkSum &= 0xffff;

checkSum++;

}

}

return ~(checkSum & 0xffff);//对最后的结果取反

}

1. 发送数据包

将数据包组装后发送，还需要进行发送后的超时重传的检测以及接收数据包的检测

//数据段包传输

void sendbag(SOCKET& socket, SOCKADDR\_IN& addr, char\* data, int length, int&seq)

{

//头部初始化及校验和计算

sendsuccess = 0;

int addrlength = sizeof(addr);

Head head;

char\* buf = new char[maxlength + sizeof(head)];

head.datasize = length; //使用传入的data的长度定义头部datasize

head.seq = unsigned char(seq);//序列号

memcpy(buf, &head, sizeof(head)); //拷贝首部的数据

memcpy(buf + sizeof(head), data, sizeof(head) + length); //数据data拷贝到缓冲数组

head.checksum = check((u\_short\*)buf, sizeof(head) + length);//计算数据部分的校验和

memcpy(buf, &head, sizeof(head)); //更新后的头部再次拷贝到缓冲数组

//发送

sendto(socket, buf, length + sizeof(head), 0, (sockaddr\*)&addr, addrlength);//发送

cout << "【发送】标志位 = " <<head.flag << " 序列号 = " << int(head.seq)<< " 校验和 = " << int(head.checksum) << endl;

clock\_t starttime = clock();//记录发送时间

int sendcount1=0;

//处理超时重传

while (1)

{

sendcount1 = 0;

u\_long mode = 1;

ioctlsocket(socket, FIONBIO, &mode); //设置非阻塞模式

//等待接收消息

while (recvfrom(socket, buf, maxlength, 0, (sockaddr\*)&addr, &addrlength)<= 0)

{

if (clock() - starttime > retime) //超时重传

{

head.datasize = length;

head.seq = u\_char(seq);//序列号

head.flag = u\_char(0x0); //清空发送栈

memcpy(buf, &head, sizeof(head)); //拷贝首部的数据

memcpy(buf + sizeof(head), data, sizeof(head) + length); //数据data拷贝到缓冲数组

head.checksum = check((u\_short\*)buf, sizeof(head) +length);//计算数据部分的校验和

memcpy(buf, &head, sizeof(head)); //更新后的头部再次拷贝到缓冲数组

cout << "【超时重传】【发送】标志位 = " <<head.flag << " 序列号 = "<< int(head.seq) << endl;

sendcount1++;

sendto(socket, buf, length + sizeof(head), 0, (sockaddr\*)&addr,addrlength);//重新发送

starttime = clock();//记录当前发送时间

if (sendcount1 == sendcount) {

return;

}

}

}

memcpy(&head, buf, sizeof(head));//缓冲区接收到信息，读取

//检验序列号和ACK均正确

u\_short checknum = check((u\_short\*)&head, sizeof(head));

if (head.seq == u\_short(seq) && head.flag == ACK)

{

cout << "【接收】标志位 = " <<head.flag<< " 序列号 = " << int(head.seq) << endl;

sendsuccess = 1;

break;

}

}

}

1. 发送文件

将文件拆分成多个固定大小为maxlength=2048的数据包，在循环中将所有数据包发送，也要对超时重传进行检测，如果所有数据包发送完毕，则最后组装一个END数据包发送给接收端，表示文件发送结束，可以断开连接。

//文件传输

void sendfile(SOCKET& socket, SOCKADDR\_IN& addr, char\* data, int data\_len)

{

int addrlength = sizeof(addr);//地址的长度

int bagsum = data\_len / maxlength; //数据包总数，等于数据长度/一次发送的字节数

if (data\_len % maxlength) {

bagsum++; //向上取整

}

int seq = 0; //序列号

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

{

int len;

if (i == bagsum - 1)

{//最后一个数据包是向上取整的结果，因此数据长度是剩余所有

len = data\_len - (bagsum - 1) \* maxlength;

}

else

{//非最后一个数据长度均为maxlength

len = maxlength;

}

sendbag(socket, addr, data + i \* maxlength, len, seq);

if (sendsuccess == 0) {

cout << "【重传失败】" << endl;

return;

}

seq++;

seq = seq % 256; //序列号在数据包中占8位，从0-255，超过则模256去除

}

//发送结束信息

Head head;

char\* buf = new char[sizeof(head)]; //缓冲数组

newbag(head, END, buf); //调用函数生成ACK=SYN=FIN=1的数据包，表示结束

sendto(socket, buf, sizeof(head), 0, (sockaddr\*)&addr, addrlength);

clock\_t starttime = clock();//计时

while (1)

{

u\_long mode = 1;

ioctlsocket(socket, FIONBIO, &mode);//设置为非阻塞模式

while (recvfrom(socket, buf, maxlength, 0, (sockaddr\*)&addr, &addrlength)<= 0)

{//等待接收

if (clock() - starttime > retime)

{//超过了设置的重传时间限制，重新传输数据包

char\* buf = new char[sizeof(head)]; //缓冲数组

newbag(head, END, buf); //调用函数生成ACK=SYN=FIN=1的数据包，表示结束

cout << "【超时等待重传】" << endl;

sendto(socket, buf, sizeof(head), 0, (sockaddr\*)&addr,addrlength); //继续发送相同的数据包

starttime = clock(); //新一轮计时

}

}

memcpy(&head, buf, sizeof(head));//缓冲区接收到信息，读取到首部

if (head.flag == END)

{//接收到END口令

cout << "【传输成功】" << endl;

break;

}

}

u\_long mode = 0;

ioctlsocket(socket, FIONBIO, &mode);//改回阻塞模式

}

（6）四次挥手

由于第二次挥手和第三次挥手可以重合在一次，因此代码只写了三次挥手。第一次挥手由发送端发起，组装数据包后进行发送。

//关闭连接 三次挥手

void fourbye(SOCKET& socket, SOCKADDR\_IN& addr)

{

int addrlength = sizeof(addr);

Head head;

char\* buff = new char[sizeof(head)];

//第一次挥手

head.flag = FIN;

//head.checksum = 0;//校验和置0

head.checksum = check((u\_short\*)&head, sizeof(head));

memcpy(buff, &head, sizeof(head));

if (sendto(socket, buff, sizeof(head), 0, (sockaddr\*)&addr, addrlength) == SOCKET\_ERROR)

{

cout << "【第一次挥手失败】" << endl;

return;

}

cout << "第一次挥手【FIN ACK】" << endl;

clock\_t byetime = clock(); //记录发送第一次挥手时间

u\_long mode = 1;

ioctlsocket(socket, FIONBIO, &mode);

发送完数据包后需要利用while循环持续等待接收端发送的确认，从而判定是否需要超时重传，以及进行校验和的检验判断数据包是否正确，如果正确则第二次挥手成功。

//第二次挥手

while (recvfrom(socket, buff, sizeof(head), 0, (sockaddr\*)&addr, &addrlength) <= 0)

{//等待接收

if (clock() - byetime > retime)//超时重传

{

memcpy(buff, &head, sizeof(head));//将首部放入缓冲区

sendto(socket, buff, sizeof(head), 0, (sockaddr\*)&addr, addrlength);

byetime = clock();

}

}

//进行校验和检验

memcpy(&head, buff, sizeof(head));

if (head.flag == ACK && check((u\_short\*)&head, sizeof(head) == 0))

{

cout << "第二次挥手【FIN ACK】" << endl;

}

else

{

cout << "【第二次挥手失败】" << endl;

return;

}

组装第三次挥手的数据包，如果发送成功则判定第三次挥手成功，此时不需要等待客户端的响应，直接断开连接即可。

//第三次挥手

head.flag = ACK\_FIN;

head.checksum = check((u\_short\*)&head, sizeof(head));//计算校验和

memcpy(buff, &head, sizeof(head));

if (sendto(socket, (char\*)&head, sizeof(head), 0, (sockaddr\*)&addr, addrlength) == -1)

{

cout << "【第三次挥手失败】" << endl;

return;

}

cout << "第三次挥手【ACK】" << endl;

cout << "【结束连接】"<<endl;

cout << "---------------------------------------------------" << endl;

}

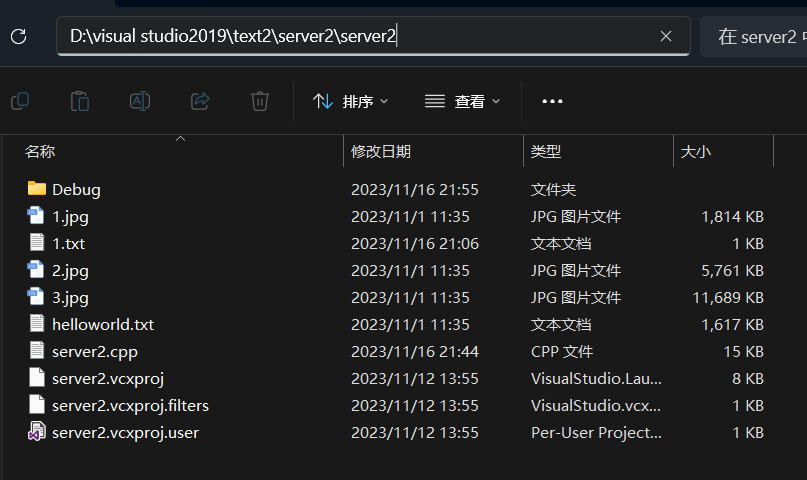
1. **实验结果**

**1.运行截图**

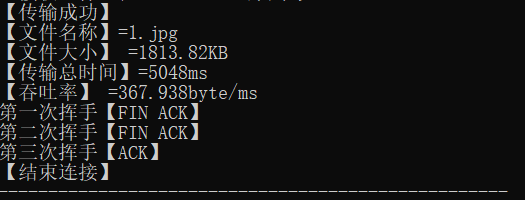
（1）将路由器设置端口号与IP地址，并设置丢包率为1%，延迟1ms

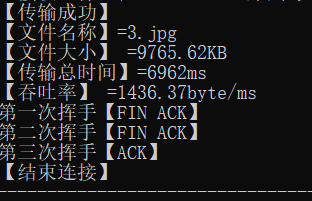


1. 将测试文件放置于发送端的程序目录下

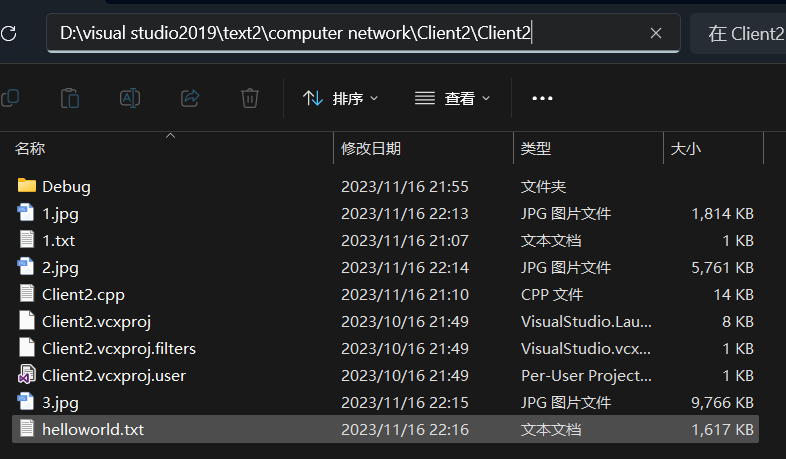


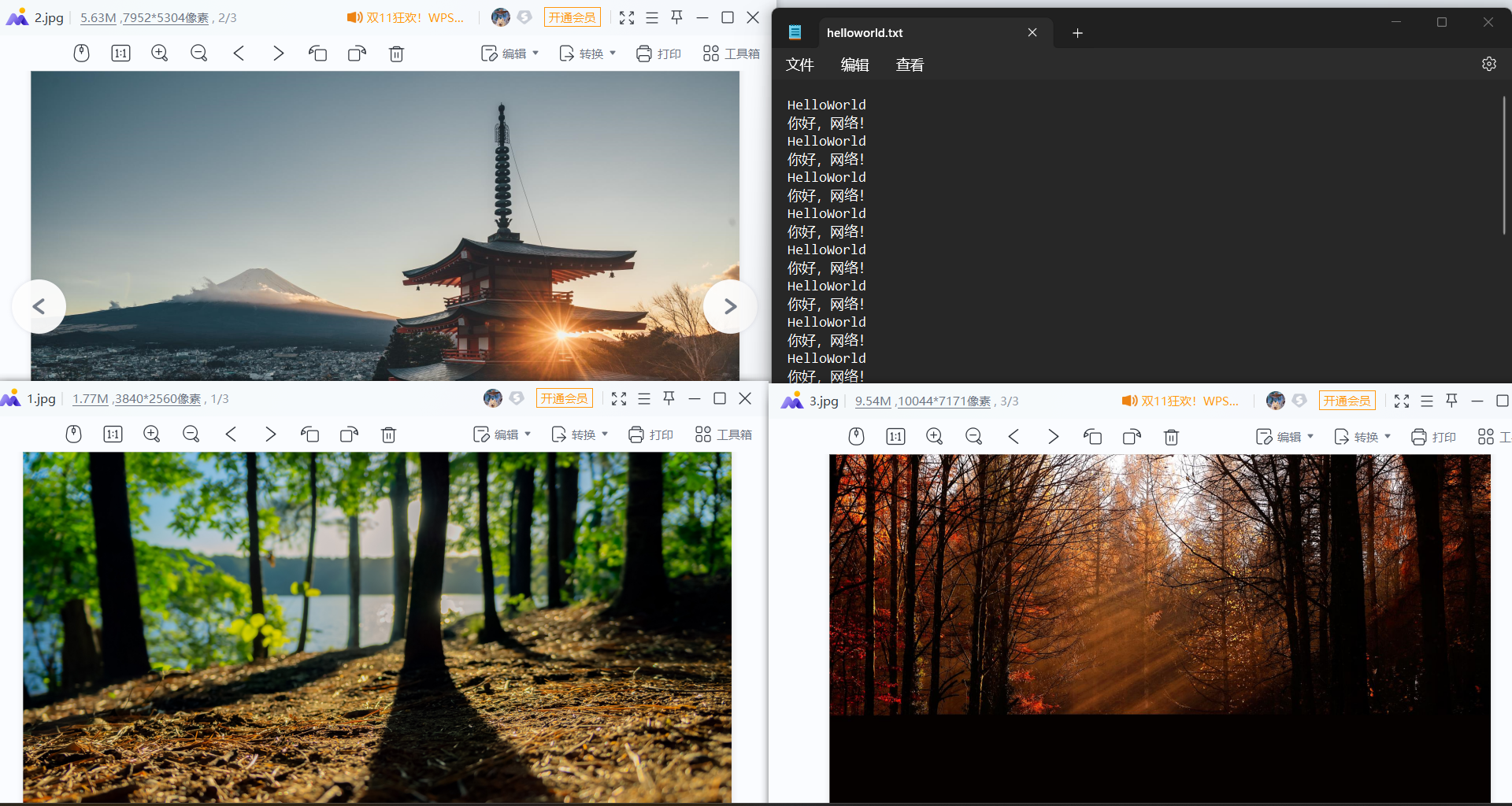
1. 发送1.jpg,2.jpg,3.jpg,helloworld.txt文件测试

发现四个文件都传输完成，并且文件大小和系统显示的都相同，经检测也都能正常打开文件，传输成功。





**2.传输结果分析**

（1）使用刚才设定的router程序来模拟丢包和延迟

（2）输入对应的路由器IP和服务器IP等信息，实现连接



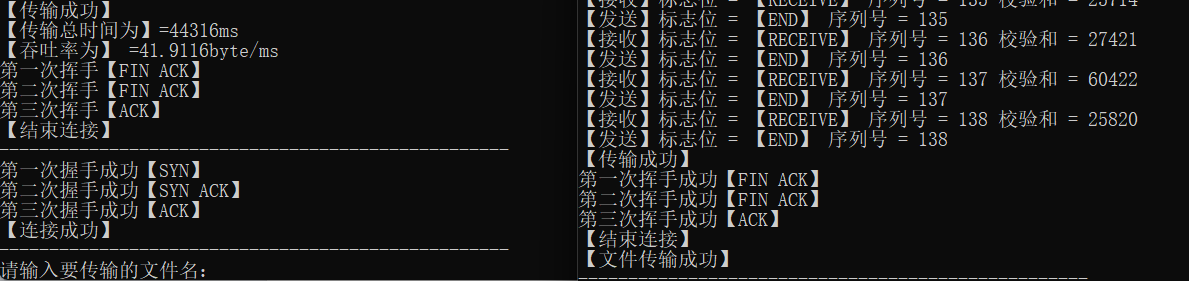
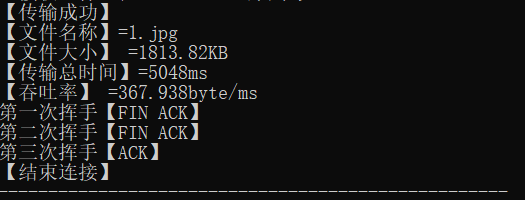
（3）传输测试文件1.jpg

可以发现开始传输时会打印输出标志位、序列号、校验和等信息，

如果产生丢包的情况，则会在发送端显示出超时重传，接收端也会发现有校验和为0的情况出现



传输成功后会打印文件名称、文件大小、传输总时间、吞吐率等信息



1. **心得体会**

通过此次实验，学习到了UDP连接等相关的知识，对于建立连接、差错检测、接收确认、超时重传协议有了深入的了解，巩固了所学的知识。

也发现了许多问题，例如网络延迟较高时或者丢包率较大时发送效率就会显著下降，卡顿明显，可能是代码还有待完善，还要继续学习。

**六、附录**

完整代码参照GitHub:

https://github.com/Q-qiuqiu/Computer-Networks/tree/main/lab3-1