计算机网络第五次实验报告

网络空间安全学院 物联网工程 2111673 岳志鑫

一、实验目的

基于 UDP 服务设计可靠传输协议并编程实现（3-3）

二、实验要求

在实验 3-1 的基础上，将停等机制改成基于滑动窗口的流量控制机制，发送窗口和接收窗口采用相同大小，支持选择确认，完成给定测试文件的传输。

- 协议设计：数据包格式，发送端和接收端交互，详细完整
- 流水线协议：多个序列号
- 发送缓冲区、接收缓冲区
- 选择确认：SR(Selective Repeat)
- 日志输出：收到/发送数据包的序号、ACK、校验和等，发送端和接收端的窗口大小等情况、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）

三、实验内容

1.协议设计

（1）建立连接：

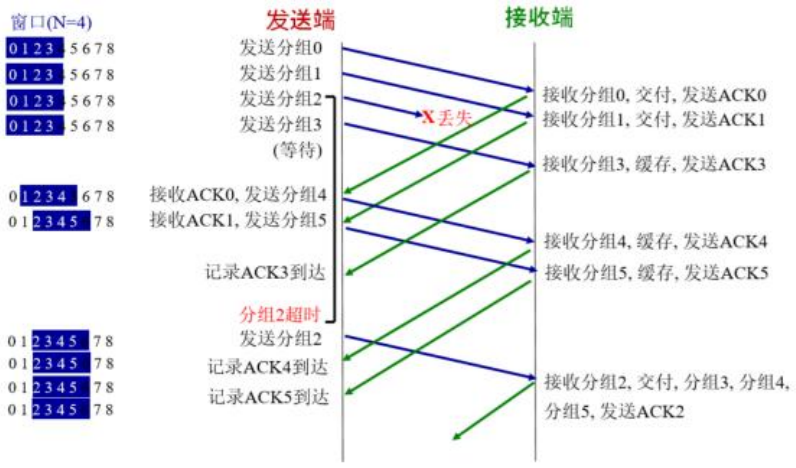
通过握手过程建立连接。发送方发送连接请求数据包，接收方收到连接请求，发送连接确认数据包。

（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）选择确认：

当接收方成功收到一个数据包时，它发送一个确认（ACK）给发送方。发送方接收到这个确认后就认为这个序号的数据包已传输成功，把这个数据包对应的计时器清除。如果发送方出现了丢包的问题，那么接收方就会接收这个传送过来的乱序的数据包并先缓存下来发送 ACK，发送方那边按照窗口继续发送数据包，当丢失的那个数据包的计时器超时后会重传该数据包，如果接收端接收到了这个丢失的包，就将缓存下来的所有正确顺序的数据包交付。



（4）超时重传：

发送方发送数据包，利用数组为每一个发送的数据包启动定时器。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，重新发送这个计时器对应的数据包。

（5）滑动窗口

一个窗口的大小是由一个后沿序号和一个前沿序号来控制的。避免发送大量数据而接收端缓冲区积累满后导致的丢包问题。发送窗口发送一个数据包后移动自己的nextseq，只有收到对方对于本段发送窗口内字节的 ACK 确认，才会移动发送窗口的前沿 base。当前面还有字节未接收但收到后面字节的情况下，窗口不会移动，超时后发送端会对丢失数据包进行重传，然后移动窗口。

（6）流量控制：

采用滑动窗口机制，传输的数据包会首先存储在滑动窗口对应的缓冲区中，如果缓冲区满了，则滑动窗口也会清零，发送端就不会发送数据，等待缓冲区中的所有数据都写入后清空缓冲区，再将滑动窗口开启，发送端就会继续发送数据包，保证了不会出现网络拥塞等问题。

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;
    }
};
```

（2）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;
```

（3）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
}
```

（4）发送文件（以发送端为例）

将文件拆分成多个固定大小为 maxlength=4096 的数据包，并输出总数据包个数，在循环中利用滑动窗口将所有数据包发送。

```
    Head head;
    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
    }
    totalnum = bagsum;
    int base = 0;//发送窗口开始的位置
    int nextseq = 0;//发送窗口结束的位置
    int count = 0;//每个数据包的计时器的计数
    char* buff = new char[sizeof(head)]; //数据缓冲区
    clock_t starttime[10000] = {0}; //计时
    int flag[10000] = { 0 };
    int newseq = 0;//当前序列号，用于更新期待序列号
    int base2 = 0;
    while (base < bagsum - 1)
    {
        int len;
        if (nextseq == 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 部分、数据段包发送
        //头部初始化及校验和计算
        Head head;
        char* buf = new char[maxlength + sizeof(head)];
        for (int i = 0; i < slidewindows-1; i++) {
            if ( ((slidewindows - nextseq + base) > 0) && (nextseq<base+slidewindows) &&
nextseq != bagsum) {//没有超过窗口大小且未发送完，并且不是窗口的结尾
                head.datasize = len; //使用传入的 data 的长度定义头部 datasize
                head.seq = u_short(nextseq);//序列号
                memcpy(buf, &head, sizeof(head)); //拷贝首部的数据
                memcpy(buf + sizeof(head), data + nextseq * maxlength, sizeof(head) + len);
//数据 data 拷贝到缓冲数组
```

```
                head.checksum = check((u_short*)buf, sizeof(head) + len);//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
                sendto(socket, buf, len + sizeof(head), 0,
(sockaddr*)&addr,addrlength);//发送

                count = int(head.seq);
                cout << "--------发送第" << int(head.seq) << "/" << totalnum << "个数
据包--------" << endl;
                cout << "【发送】标志位 = 【SEND】" << " 序列号 = " << int(head.seq) <<
" 校验和 = " << int(head.checksum) << endl;
            }
            else if (nextseq==bagsum) {
                break;
            }
            nextseq++;
            starttime[count] = clock();//记录发送时间
        }
```

在接收回复中进行选择确认，对接收到的回复数据包进行检测，如果出现校验和不正确并且序列号乱序的情况，那么就是出现了丢包的问题，对错误信息进行输出。如果期望的序列号和收到的序列号相同则代表传输正常，将这个数据包对应的计时器清除。如果期望的序列号小于收到的序列号，就把收到的序列号对应的数据包的计时器清除。

```
    //接收消息
            for (int i = 0; i < slidewindows; i++) {
                if (recvfrom(socket, buf, maxlength, 0, (sockaddr*)&addr, &addrlength))
{
                    memcpy(&head, buf, sizeof(head));//缓冲区接收到信息，读取
                    u_short checksum1 = check((u_short*)&head, sizeof(head)); //计算校
验和
                    //如果出现丢包
                    if ((int(checksum1) != 0 ) && wishseq<int(head.seq))
                    {
                        cout << "--------接收到不正确的第" << int(head.seq) << "个数
据包的回复--------" << endl;
                        base = int(head.seq);
                        cout << "【接收】标志位 = 【RECEIVE】" << " 序列号 = " <<
int(head.seq) << " 校验和 = " << int(head.checksum) << endl;
                        cout << "【传输错误，" << "丢失" << int(head.seq) << "号数据
包】" << endl;
                        cout << "窗口前沿 = " << base << " 窗口后沿 = " << nextseq <<
endl;
                        newseq = int(head.seq);
                    }
```

```
                    //接收到正确的数据包，输出接收信息
                    else if (wishseq == int(head.seq))
                    {
                        base = int(head.seq);
                        flag[head.seq] = 1;//计时器标志位置为1
                        wishseq = (int(head.seq) + 1);
                        cout << "--------接收第" << int(head.seq) << "个数据包的回复
--------" << endl;
                        cout << "【接收】标志位 = 【RECEIVE】" << " 序列号 = " <<
int(head.seq) << " 校验和 = " << int(head.checksum) << endl;
                        cout << "窗口前沿 = " << base << " 窗口后沿 = " << nextseq <<
endl;

                    }
                    //接收到重传的数据包确认
                    else if (newseq = int(head.seq)) {
                        base++;
                        flag[head.seq] = 1;//计时器标志位置为1
                        wishseq = newseq + 1;

                        cout << "--------接收第" << int(head.seq) << "个数据包的重传
回复--------" << endl;
                        cout << "【接收】标志位 = 【RECEIVE】" << " 序列号 = " <<
int(head.seq) << " 校验和 = " << int(head.checksum) << endl;
                        cout << "窗口前沿 = " << base << " 窗口后沿 = " << nextseq <<
endl;
                    }
                    else if (wishseq<int(head.seq)) {
                        flag[head.seq] = 1;//计时器标志位置为1
                        wishseq = (int(head.seq) + 1);
                        cout << "--------接收第" << int(head.seq) << "个数据包的回复
--------" << endl;
                        cout << "【接收】标志位 = 【RECEIVE】" << " 序列号 = " <<
int(head.seq) << " 校验和 = " << int(head.checksum) << endl;
                        cout << "窗口前沿 = " << base << " 窗口后沿 = " << nextseq <<
endl;
                    }
                }
            }
```

对滑动窗口内的数据包进行遍历检查，发现谁的计时器超时了就对其进行重发，收到回复后再清除计时器。

```
            //选择确认的超时重传
            for (int i = base2; i < nextseq; i++) {
```

```
                if (clock() - starttime[i] > retime && flag[i] == 0) {

                    cout << "【超时重新发送】" << endl;
                    Head head;
                    char* buf = new char[maxlength + sizeof(head)];
                    head.datasize = len; //使用传入的 data 的长度定义头部 datasize
                    head.seq = u_short(i);//序列号
                    base2 = i;
                    memcpy(buf, &head, sizeof(head)); //拷贝首部的数据
                    memcpy(buf + sizeof(head), data + i * maxlength, sizeof(head) + len);
//数据 data 拷贝到缓冲数组
                    head.checksum = check((u_short*)buf, sizeof(head) + len);//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, len + sizeof(head), 0, (sockaddr*)&addr,
addrlength);//发送
                    cout << "--------重新发送第" << int(head.seq) << "个数据包--------"
<< endl;

                    cout <<"【发送】标志位 = 【SEND】" <<" 序列号 =" << int(head.seq)
<< " 校验和 = " << int(head.checksum) << endl;
                    cout << "窗口前沿 = " << base << " 窗口后沿 = " << nextseq << endl;
                    continue;
                }
            }
            //Sleep(50); //休眠 50ms，防止频繁发送使得丢包
            mode = 0;
            ioctlsocket(socket, FIONBIO, &mode); //阻塞
```

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，同时进行超时重传的检测。

```
    //传输完毕
        buff = new char[sizeof(head)]; //缓冲数组
        newbag(head, END, buff); //调用函数生成 ACK=SYN=FIN=1 的数据包，表示结束
        sendto(socket, buff, sizeof(head), 0, (sockaddr*)&addr, addrlength);
        clock_t starttime2 = clock();//计时

        //处理超时重传
        while (1)
        {
            u_long mode = 1;
            ioctlsocket(socket, FIONBIO, &mode); //设置非阻塞模式
            //等待接收消息
```

```
            while (recvfrom(socket, buff, maxlength, 0, (sockaddr*)&addr, &addrlength) <=
0)
            {
                if (clock() - starttime2 > retime) //超时重传
                {
                    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
                    starttime2 = clock(); //新一轮计时
                }
            }
            memcpy(&head, buff, 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;
                //sendsuccess = 1;
                break;
            }
        }
        u_long mode = 0;
        ioctlsocket(socket, FIONBIO, &mode);//改回阻塞模式
}
```

（5）接收文件（以接收端为例）

在接收文件时，对数据包的序列号进行确认，如果期待的序列号和接收的序列号相同就交付并回复 ACK。如果期待序列号小于收到的序列号就代表发生了丢包，将收到的数据包先缓存在自己定义的一个二维数组中，然后回复 ACK，如果丢包的情况存在就一直将收到的数据包存在缓存中，直到丢失的数据包重传，将所有按顺序的缓存下的数据包交付。

```
//接收文件
int recvfile(SOCKET& socket, SOCKADDR_IN& addr, char* data)
{
    int addrlength = sizeof(addr);
    long int sum = 0;//文件长度，要返回的数据
    Head head;
    char* buf = new char[maxsize + sizeof(head)]; //缓冲数组长度是数据+头部的最大大小
    char** huanchong = new char* [3000];//缓冲储存的数组
    for (int i = 0; i < 3000; ++i) {
        huanchong[i] = new char[maxsize + sizeof(Head)];
    }
    int wishseq = 0; //期待序列号清 0
    int newseq = 0;//当前序列号，用于更新期待序列号
```

```
    //int huanchongbase = 0;//缓冲的数据包序号
    while (1)
    {//接收数据包
        int recvlength = recvfrom(socket, buf, sizeof(head) + maxsize, 0, (sockaddr*)&addr,
&addrlength);//接收报文长度
        memcpy(&head, buf, sizeof(head));

        if (head.flag == END && check((u_short*)&head, sizeof(head)) == 0)//END 标志位，
校验和为 0，结束
        {
            cout << "【传输成功】" << endl;
            break; //结束跳出 while 循环
        }

        if (head.flag == unsigned char(0) && check((u_short*)buf, recvlength -
sizeof(head)))//校验和不为 0 且 flag 是无符号字符
        {
            //判断收到的数据包是否正确
            if (wishseq<int(head.seq))
            {//期望 seq 小于收到的 seq，代表有丢包产生
                cout << "【接收错误，" <<"丢失"<<wishseq<<"号数据包】" <<endl;
                //缓存这个接收到的数据包并发送 ack
                cout << "--------接收第" << int(head.seq) << "个数据包--------" << endl;
                cout <<"【接收】标志位 = 【RECEIVE】" <<" 序列号 =" << int(head.seq) <<
" 校验和 = " << int(head.checksum) << endl;
                // 使用二维数组，拷贝整个数据包
                //数组的大小是接收到的报文长度减去头部大小
                memcpy(huanchong[int(head.seq)], buf + sizeof(head), recvlength -
sizeof(head)); //从头部后面开始拷贝，把数据拷贝到缓冲数组
                cout << "数据包已缓存至" << "缓冲[" << int(head.seq) << "]"<<endl;

                //初始化首部
                newbag2(head, ACK, buf, head.seq);
                //发送 ACK
                sendto(socket, buf, sizeof(head), 0, (sockaddr*)&addr, addrlength);
                cout << "--------发送第" << int(head.seq) << "个数据包的回复--------" <<
endl;
                cout <<"【发送】标志位 = 【SEND】" <<" 序列号 =" << (int)head.seq << "
校验和 = " << int(head.checksum) << endl;
                newseq = head.seq;

            }
            else if(newseq>int(head.seq)){
                //接收到重传的数据包
```

```
                cout << "--------重新接收第" << int(head.seq) << "个数据包--------" <<
endl;
                cout << "【接收】标志位 =【RECEIVE】 "  << " 序列号 = " << int(head.seq)
<< " 校验和 = " << int(head.checksum) << endl;

                memcpy(huanchong[int(head.seq)], buf + sizeof(head), recvlength -
sizeof(head)); //从头部后面开始拷贝，把数据拷贝到缓冲数组

                for (int i = int(head.seq); i <= newseq; i++) {
                    cout << "数据包已从" << "缓冲[" << i << "]加载" << endl;
                    memcpy(data + sum, huanchong[i], recvlength - sizeof(head));
                    sum = sum + int(head.datasize);
                }
                //初始化首部
                newbag2(head, ACK, buf, head.seq);
                //发送 ACK
                sendto(socket, buf, sizeof(head), 0, (sockaddr*)&addr, addrlength);

                cout << "--------发送第" << int(head.seq) << "个数据包的回复--------" <<
endl;
                cout << "【发送】标志位 =【SEND】 "  << " 序列号 = " << (int)head.seq <<
" 校验和 = " << int(head.checksum) << endl;
                wishseq =( newseq+1);
            }
            else if (wishseq == int(head.seq)) {
                //接收到数据包正确
                cout << "--------接收第" << int(head.seq) << "个数据包--------" << endl;
                cout << "【接收】标志位 = 【RECEIVE】" << " 序列号 =" << int(head.seq) <<
" 校验和 = " << int(head.checksum) << endl;
                char* bufdata = new char[recvlength - sizeof(head)]; //数组的大小是接收
到的报文长度减去头部大小
                memcpy(bufdata, buf + sizeof(head), recvlength - sizeof(head)); //从头部
后面开始拷贝，把数据拷贝到缓冲数组
                memcpy(data + sum, bufdata, recvlength - sizeof(head));
                sum = sum + int(head.datasize);
                wishseq = head.seq;
                //初始化首部
                newbag2(head, ACK, buf, wishseq);
                //发送 ACK
                sendto(socket, buf, sizeof(head), 0, (sockaddr*)&addr, addrlength);
                cout << "--------发送第" << int(head.seq) << "个数据包的回复--------" <<
endl;
                cout << "【发送】标志位 = 【SEND】" << " 序列号 = " << (int)head.seq << "
校验和 = " << int(head.checksum) << endl;
```

```
                wishseq++;//序列号加
            }
        }
    }
    //发送 END 信息，结束
    newbag(head, END, buf);
    if (sendto(socket, buf, sizeof(head), 0, (sockaddr*)&addr, addrlength) == SOCKET_ERROR)
    {
        cout << "【发送错误】" << endl;
        return -1; //发送错误
    }
    // 释放子数组
    for (int i = 0; i < 3000; ++i) {
        delete[] huanchong[i];
    }

    // 释放主数组
    delete[] huanchong;
    huanchong = nullptr;//避免指针悬空
    return sum; //返回接收到的数据包字节总数
}
```

（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;
    }
```

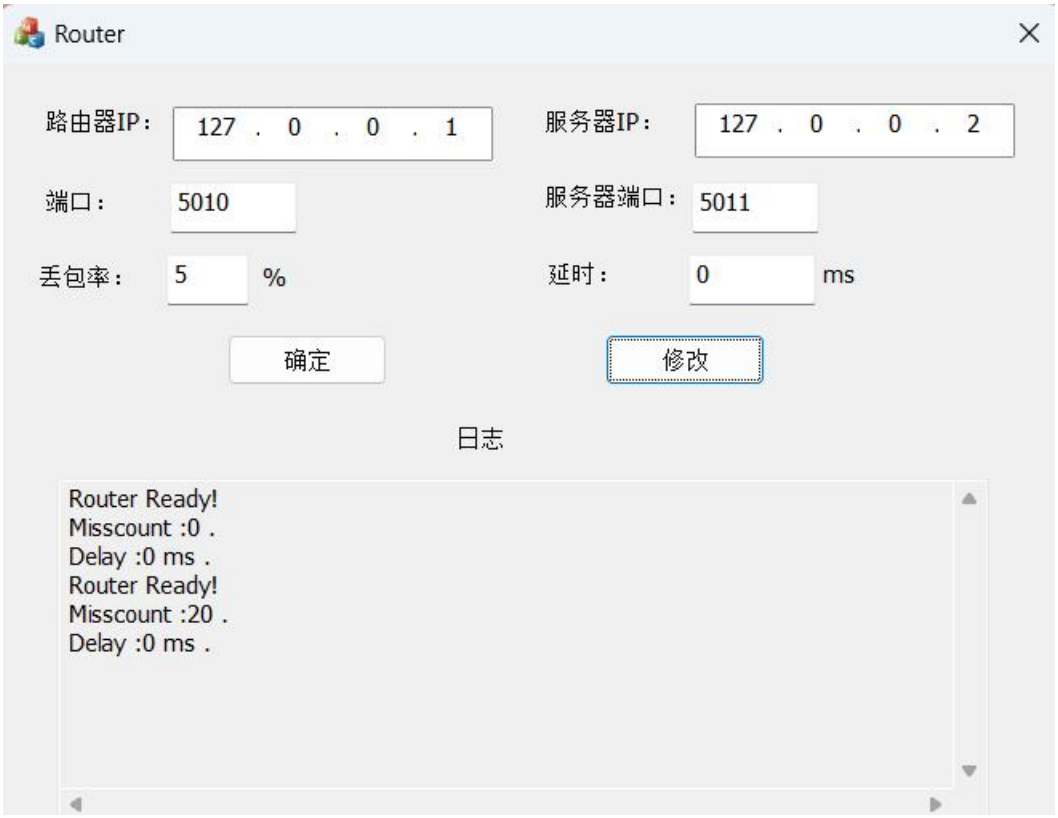
组装第三次挥手的数据包，如果发送成功则判定第三次挥手成功，此时不需要等待客户端的响应，直接断开连接即可。

```
    //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;
}
```

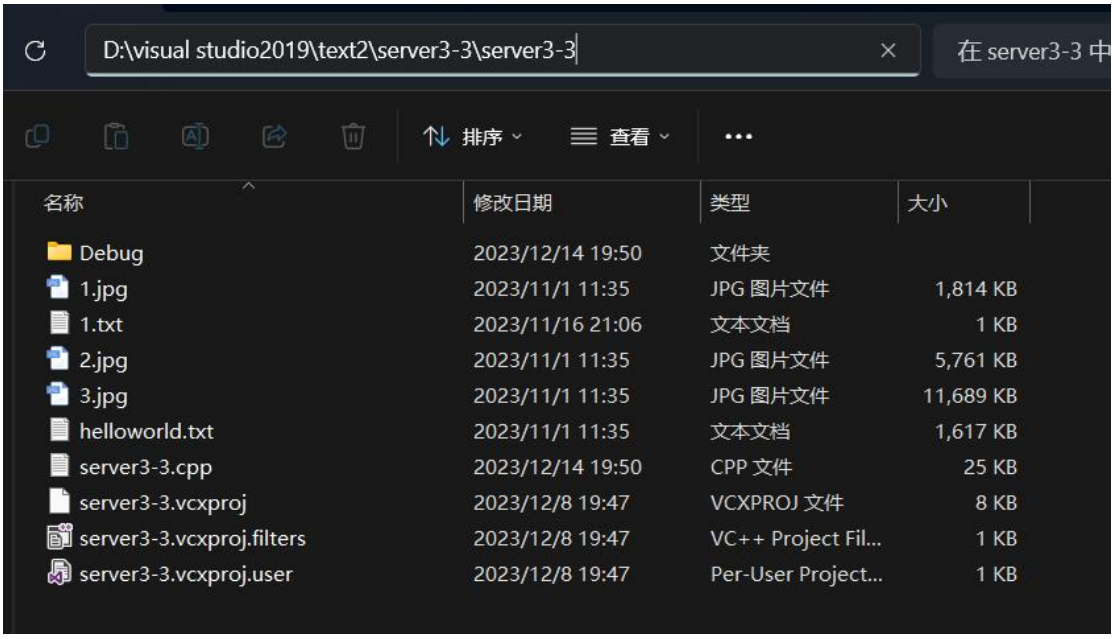
四、实验结果

1. 运行截图

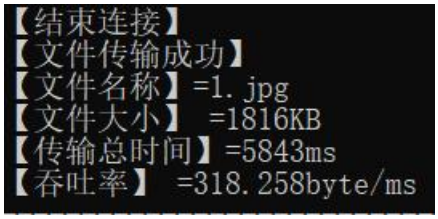
（1）设置路由器

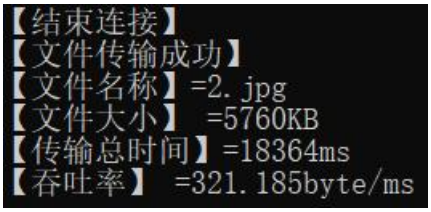


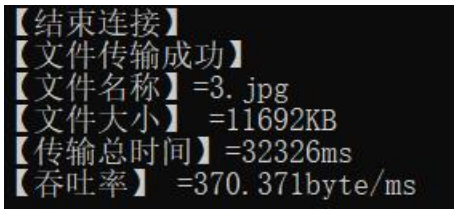
（2）将测试文件放置于发送端的程序目录下

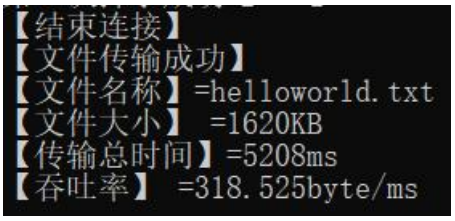


（3）发送窗口设置为 10，发送 1.jpg,2.jpg,3.jpg,helloworld.txt 文件测试

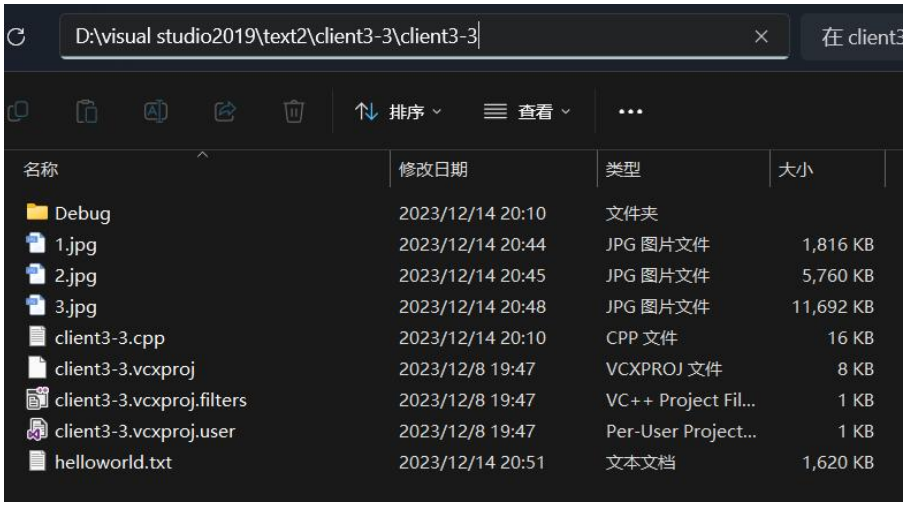


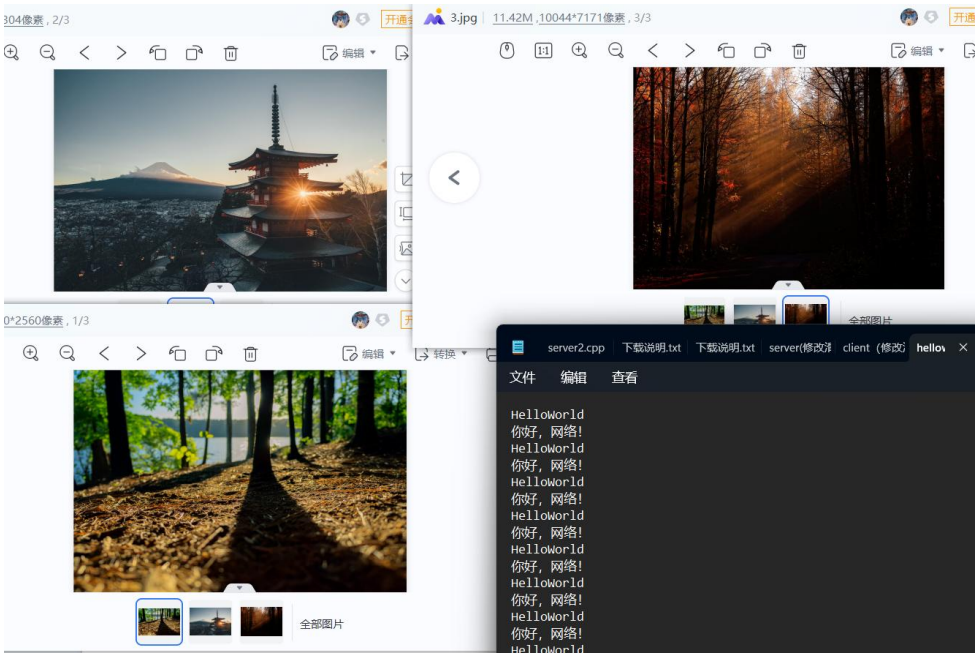






发现四个文件都传输完成，并且文件大小和系统显示的都相同，经检测也都能正常打开文件，传输成功。





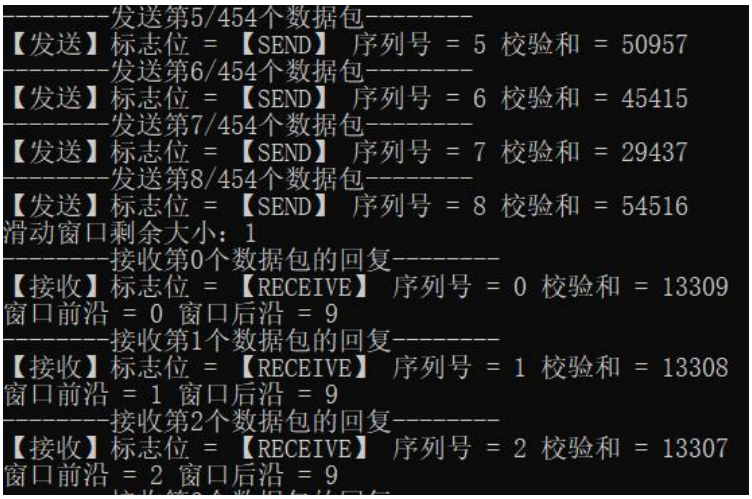
2.传输结果分析

（1）使用刚才设定的 router 程序来模拟丢包

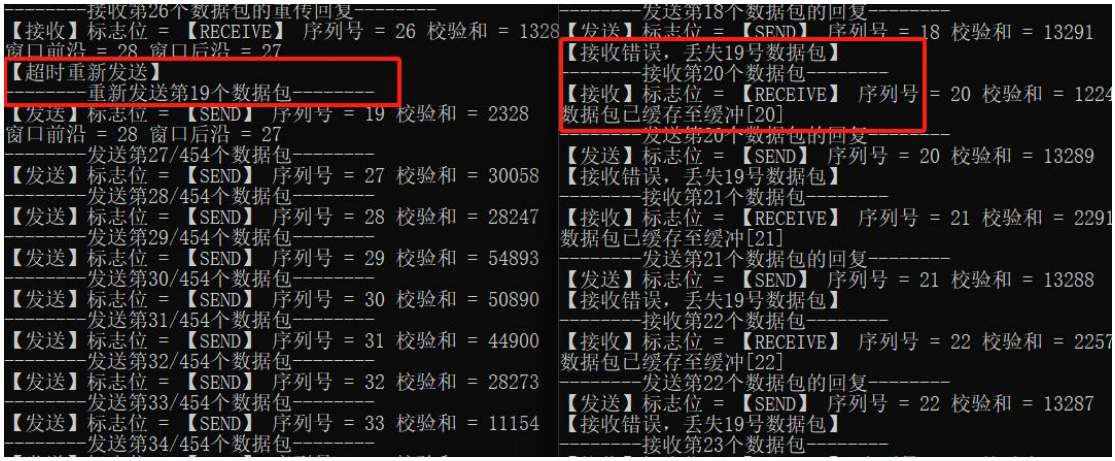
（2）输入滑动窗口大小、对应的路由器 IP 和服务器 IP 等信息，实现连接

（3）传输测试文件 1.jpg

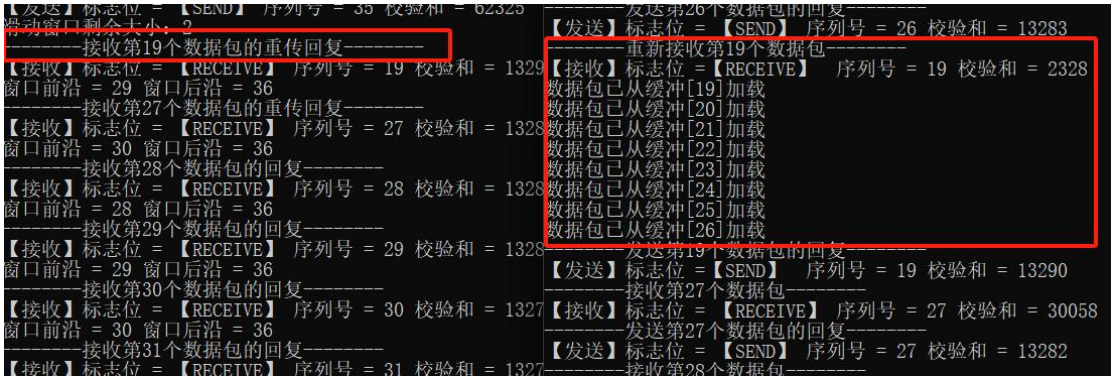
可以发现开始传输时会打印输出标志位、序列号、校验和、滑动窗口剩余大小、窗口前沿、窗口后沿等信息



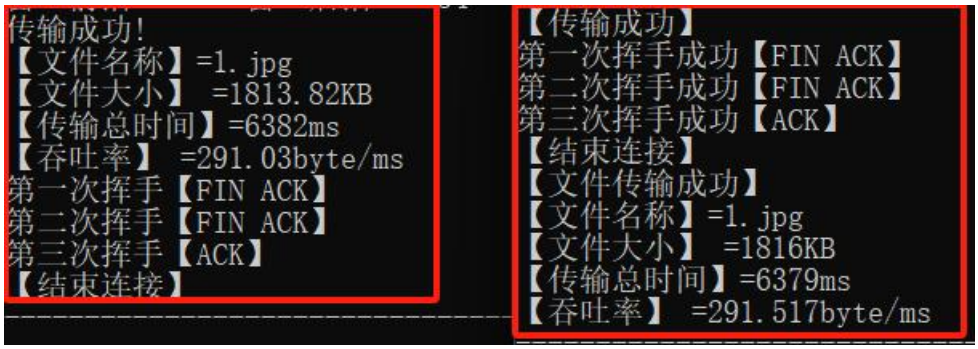
如果产生丢包的情况，则会在发送端计时器超时后显示超时重新发送，接收端会将接收到的数据包先缓存在缓存区数组中，并显示出丢失的数据包序号。



如果超时重传接收后，会将缓存区中的所有数据包交付，并回复 ACK



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，由于发送端和接收端的计时节点不同，所以也会导致出现一些细微的差异。



五、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、选择确认、超时重传、流量控制的滑动窗口等知识有了深入的了解，巩固了所学的知识。

也发现了许多问题，实验所使用的单线程也有一些问题，比如数据包超时后不能做到立即重传，而是需要等待单线程的代码块执行到发送数据包的部分才能重传，所以比多线程效率低，也有可能会引发一些问题。滑动窗口过大时传送大文件时也会出现问题，也可能是单线程或代码的问题。

所以发送窗口设置的区域范围为 4-25，不能随意输入，保证了文件传输的成功率。

六、附录

完整代码参照 GitHub:

https://github.com/Q-qiuqiu/Computer-Networks/tree/main/lab3-3