实验3-1报告

学号：2010764　姓名：高森森

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、确认重传等。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；
- 建立连接：实现类似TCP的握手、挥手功能；
- 差错检测：计算校验和；
- 确认重传：rdt2.0、rdt2.1、rdt2.2、rdt3.0等，亦可自行设计协议；
- 单向传输：发送端、接收端；
- 有必要日志输出。

实验设计

一、协议设计

我们设计了如下的报头，对于不传送文件的报文仅具有报头，对于传送文件的报文在报头后面具有1024B大小的数据区。

```
struct Header {
    //checksum
    u_short checksum = 0;//16 bits
    //group index
    u_short index=0;
    //file length
    unsigned int filelen = 0;
    //flags
    unsigned char flag = 0;
};
```

结构体Header的成员变量虽然只占据9B，但是结构体的大小必须是所有成员类型大小的整数倍，所以在编译之后，Header占据的实际大小为12B，因此我们给报头开辟的大小也为12B。

结构体Header的成员变量：

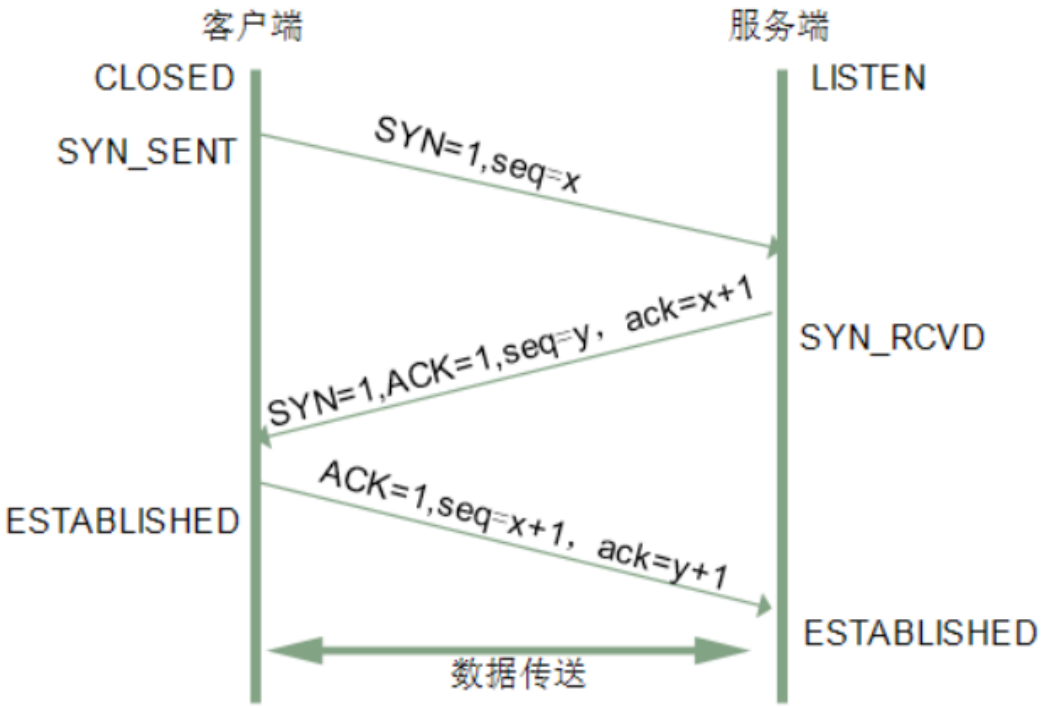
- checksum:占据2B，16位，记录校验和
- index:占据2B，16位，记录文件分组编号
- filelen:占据4B，32位，记录文件分组长度
- flag:占据1B，8位，记录报文标志信息

  报文的标志位，只占据一个字节的3位，从高到低依次为FIN,ACK,SYN，flag具体数值含义如下：

```
#define SYN 0x1 //001
#define ACK 0x2 //010
#define SYN_ACK 0x3 //011
#define FIN 0x4 //100
#define FIN_ACK 0x6 //110
#define OVER 0x7//111
```

三次握手建立连接

我们采用和如下图的TCP三次握手连接相似的方式：



第一次握手：客户端向服务端发送SYN报文

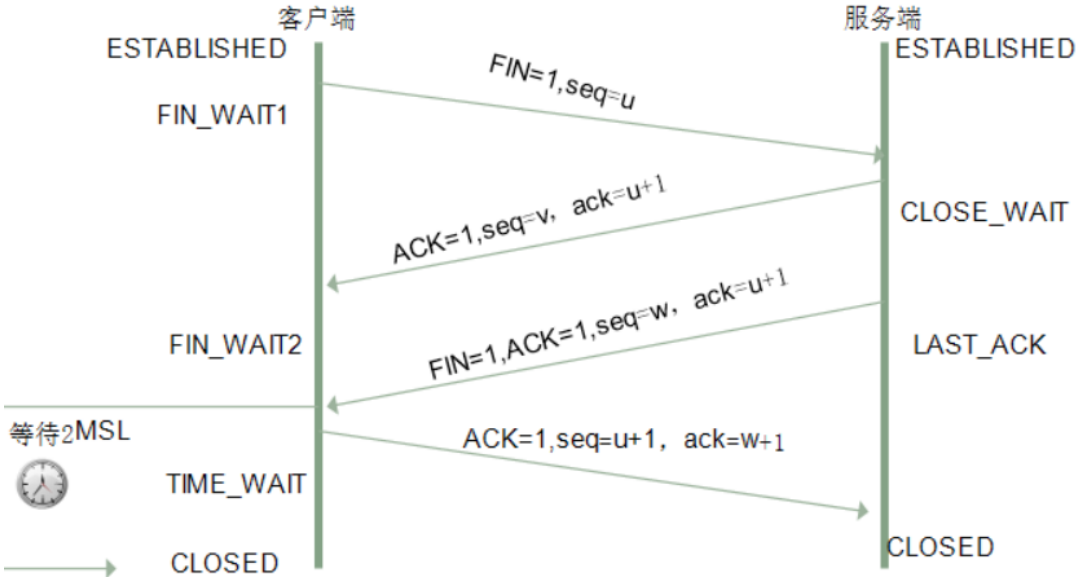
第二次握手：服务端接收到SYN报文后，进行校验和验证，之后向客户端发送SYN_ACK报文

第三次握手：客户端接收到SYN_ACK报文后，进行校验和验证，之后向服务端发送ACK报文

服务端接收到数据包后，连接成功建立，可以进行数据传输

四次挥手关闭连接

我们也采用和TCP的四次挥手相似的方式，TCP的四次挥手如下：



第一次挥手：客户端向服务端发送FIN报文

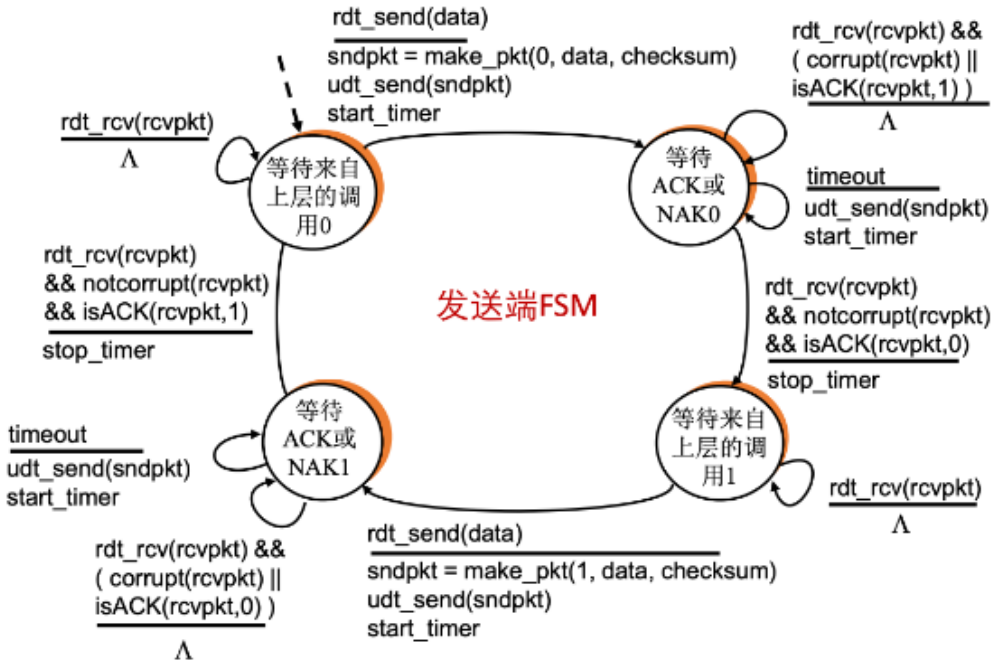
第二次挥手：服务端接收到FIN报文后，进行校验和检验，之后向客户端发送ACK确认报文

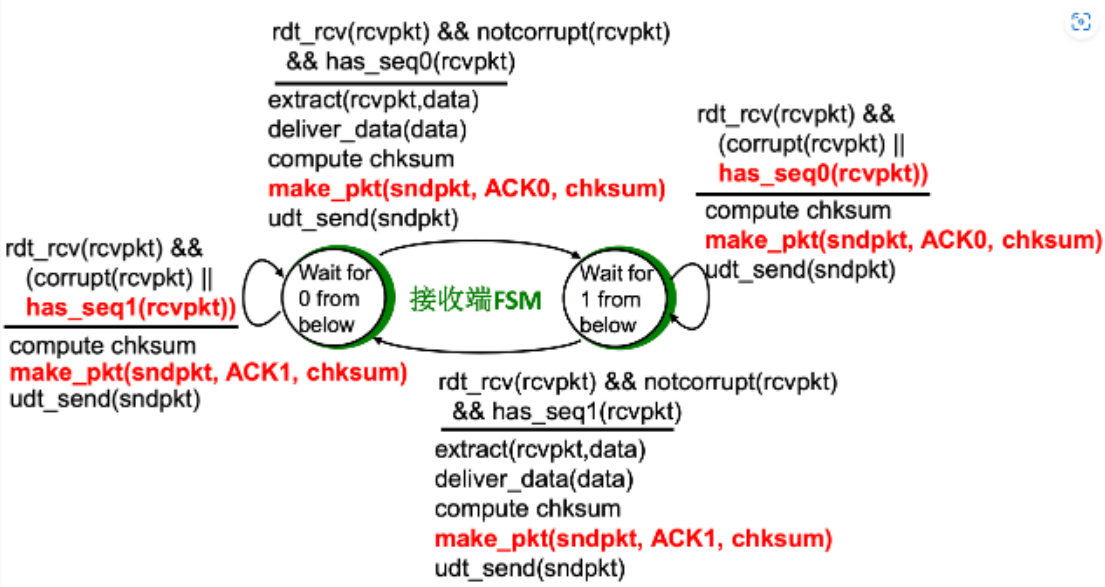
第三次挥手：服务端等待 2 MSL，随后发送FIN_ACK报文

第四次挥手：客户端接收到FIN_ACK报文后，进行校验和检验，之后向服务端发送ACK报文，随后等待 2 MSL后关闭连接，而服务端接收到ACK报文后立刻关闭连接。

数据传输

发送端和接收端均采用rdt3.0





数据在进行传输时，由于每个报文最多传输1024B，因此要进行分组传输，并在报头记录文件的分组编号。

在进行传输时，发送端只有接收到接收端对该编号分组文件的确认，才能继续传输下一个分组。当超过设置的时间上限，没有接收到确认报文，将重新发送该编号分组。

当所有分组文件发送完成后，发送端向接收端发送OVER报文，表示文件发送结束。接收端接收到OVER报文后，也发送OVER报文进行确认。

二、代码实现

计算校验和

发送端生成校验和

- 将发送的进行检验和运算的数据分成若干个16位的位串，每个位串看成一个二进制数
- 将首部中的检验和字段置为0，该字段也参与检验和运算。
- 对这些16位的二进制数进行1的补码和运算，累加的结果再取反码即生成了检验码。将检验码放入检验和字段中。其中1的补码和运算，即带循环进位的加法，最高位有进位应循环进到最低位

接收方校验检验和

- 接收方将接收的数据(包括检验和字段)按发送方的同样的方法进行1的补码和运算，累加的结果再取反码。
- 校验，如果上步的结果为0，表示传输正确；否则，说明传输有差错。

具体代码实现如下：

```
//calculate checksum
u_short cksum(u_short* mes, int size) {
    int count = (size + 1) / 2;
    u_short* buf = (u_short*)malloc(size + 1);
    memset(buf, 0, size + 1);
    memcpy(buf, mes, size);
    u_long sum = 0;
    while (count--) {
        sum += *buf++;
        if (sum & 0xffff0000) {
            sum &= 0xffff;
            sum++;
```

```
        }
    }
    return ~(sum & 0xffff);
}
```

在实现中，我们使用u_long32位进行存储，因为在计算校验和时会有进位。sum &= 0xffff;表示有进位时仅取低16位，同时低位+1。

三次握手

发送端

```
//3 way hand shake to create connection
int shakeHand(SOCKET& socketClient, SOCKADDR_IN& servAddr, int& servAddrlen)
{
    Header header;
    //set flags
    header.flag = SYN;
    header.checksum = 0;
    u_short temp = cksum((u_short*)&header, HEADER_LEN);
    header.checksum = temp;

    char* sendBuf = new char[HEADER_LEN];//12
    memcpy(sendBuf, &header, HEADER_LEN);//将首部放入缓冲区

    if (sendto(socketClient, sendBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
servAddrlen) == -1)
    {
        return -1;
    }

    cout << "第一次握手：flag=" << flag2str(int(header.flag)) << " 校验和=" <<
header.checksum << endl;

    clock_t start = clock(); //time of first shake hand

    u_long mode = 1;
    ioctlsocket(socketClient, FIONBIO, &mode);

    //receive second shake hand
    char* recvBuf = new char[HEADER_LEN];//12
    while (recvfrom(socketClient, recvBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
&servAddrlen) <= 0)
    {
        if (clock() - start > MAX_TIME)//overtime,repeat first shake hand
        {
            //sendBuf has not changed
            sendto(socketClient, sendBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
servAddrlen);
            start = clock();
            cout << "TIME OUT:正在进行重传第一次握手" << endl;
        }
    }

    //according to recvBuf to verify checksum
    memcpy(&header, recvBuf, HEADER_LEN);
    if (header.flag == SYN_ACK && cksum((u_short*)&header, HEADER_LEN) == 0)
```

```
    {
        cout << "收到第二次握手: flag=" << flag2str(int(header.flag))<<" 校验和="<<
header.checksum << endl;
    }
    else
    {
        cout << "连接发生错误，请重启客户端！" << endl;
        return -1;
    }

    //third shake hand
    header.flag = ACK;
    header.checksum = 0;
    header.checksum = cksum((u_short*)&header, HEADER_LEN);
    if (sendto(socketClient, (char*)&header, HEADER_LEN, 0,
(sockaddr*)&servAddr, servAddrlen) == -1)
    {
        return -1;
    }
    cout << "第三次握手：flag=" << flag2str(int(header.flag)) << " 校验和=" <<
header.checksum << endl;
    cout << "服务器成功连接！可以发送数据" << endl;
    return 1;
}
```

上述代码实现中需要注意的是超时重传机制，当接收第二次握手的确认超时时，要重传第一次握手重新连接。

接收端

```
int shakeHand(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int& ClientAddrLen)
{

    Header header;
    char* recvBuf = new char[HEADER_LEN];
    char* sendBuf = new char[HEADER_LEN];

    //接收第一次握手信息
    while (1)
    {
        if (recvfrom(sockServ, recvBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
&ClientAddrLen) == -1)
        {
            return -1;
        }
        memcpy(&header, recvBuf, HEADER_LEN);
        if (header.flag == SYN && cksum((u_short*)&header, HEADER_LEN) == 0)
        {
            cout << "收到第一次握手: flag=" << flag2str(int(header.flag)) << " 校验
和=" << header.checksum << endl;
            break;
        }
    }

    //发送第二次握手信息
    header.flag = SYN_ACK;
    header.checksum = 0;
```

```
    u_short temp = cksum((u_short*)&header, HEADER_LEN);
    header.checksum = temp;
    memcpy(sendBuf, &header, HEADER_LEN);
    if (sendto(sockServ, sendBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1)
    {
        return -1;
    }
    cout << "第二次握手：flag=" << flag2str(int(header.flag)) << " 校验和=" <<
header.checksum << endl;
    clock_t start = clock();//记录第二次握手发送时间

    //接收第三次握手
    while (recvfrom(sockServ, recvBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
&ClientAddrLen) <= 0)
    {
        if (clock() - start > MAX_TIME)
        {
            if (sendto(sockServ, sendBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1)
            {
                return -1;
            }
            cout << "TIME OUT:正在进行重传第二次握手" << endl;
        }
    }

    Header temp1;
    memcpy(&temp1, recvBuf, HEADER_LEN);
    if (temp1.flag == ACK && cksum((u_short*)&temp1, sizeof(temp1) == 0))
    {
        cout << "收到第三次握手: flag=" << flag2str(int(temp1.flag)) << " 校验和="
<< temp1.checksum << endl;
        cout << "成功建立通信！可以接收数据" << endl;
    }
    else
    {
        cout << "serve连接发生错误，请重启客户端！" << endl;
        return -1;
    }
    return 1;
}
```

传输数据

发送单个分组

```
/send single data package
//index:group index
void send_package(SOCKET& socketClient, SOCKADDR_IN& servAddr, int& servAddrlen,
char* message, int len, int& index)
{
    char* sendBuf = new char[MESSAGE_LEN];
    Header header;
    header.filelen = len;
    header.index = (u_short)index;
```

```
    memcpy(sendBuf, &header, HEADER_LEN);

    memcpy(sendBuf + HEADER_LEN, message, len);
    u_short checksum = cksum((u_short*)sendBuf, HEADER_LEN);//计算校验和
    header.checksum = checksum;
    memcpy(sendBuf, &header, HEADER_LEN);
    sendto(socketClient, sendBuf, len + HEADER_LEN, 0, (sockaddr*)&servAddr,
servAddrlen);//发送
    cout << "发送分组 " << len << " bytes" << " flag=" <<
flag2str(int(header.flag)) << " 分组序号="
        << int(header.index) << " 校验和=" << int(header.checksum) << endl;
    clock_t start = clock();//记录发送时间

     //接收确认信息
    char* recvBuf = new char[HEADER_LEN];
    while (1)
    {
        u_long mode = 1;
        ioctlsocket(socketClient, FIONBIO, &mode);
        while (recvfrom(socketClient, recvBuf, HEADER_LEN, 0,
(sockaddr*)&servAddr, &servAddrlen) <= 0)
        {
            if (clock() - start > MAX_TIME)
            {
                sendto(socketClient, sendBuf, len + HEADER_LEN, 0,
(sockaddr*)&servAddr, servAddrlen);//发送
                cout << "TIME OUT:重新发送分组 " << len << " bytes" << " flag = "
<< flag2str(int(header.flag)) <<
                    " 分组序号 ="<< int(header.index) << " 校验和=" <<
int(header.checksum) << endl;
                start = clock();//记录发送时间
            }
        }
        Header recvHeader;
        memcpy(&recvHeader, recvBuf, sizeof(recvHeader));
        u_short checksum = cksum((u_short*)&recvHeader, sizeof(recvHeader));
        if (recvHeader.index == u_short(index) && recvHeader.flag == ACK)
        {
            cout << "分组已被确认 flag=" << flag2str(int(recvHeader.flag)) << " 分
组序号=" << int(recvHeader.index) << endl;
            break;
        }
        else//直到接收到确认信息，否则继续等待接收，继续重传分组
        {
            continue;
        }
    }
    u_long mode = 0;
    ioctlsocket(socketClient, FIONBIO, &mode);//改回阻塞模式
}
```

上述代码实现中需要注意的是，如果超出设定时间没有接收到该分组的确认报文，我们会重传该分组；如果接收到的确认报文序号与该分组序号不同，也会重传该分组。直到确认了该分组，才能进行下一个分组的传输。

发送文件

```
void send(SOCKET& socketClient, SOCKADDR_IN& servAddr, int& servAddrlen, char*
message, int len)
{
    int packagenum = len / FILE_LEN + (len % FILE_LEN != 0);
    int index = 0;//分组编号
    for (int i = 0; i < packagenum; i++)
    {
        send_package(socketClient, servAddr, servAddrlen, message + i *
FILE_LEN,
            i == packagenum - 1 ? len - (packagenum - 1) * FILE_LEN : FILE_LEN,
index);
        index++;
        index %= 65536;
    }
    //发送结束信息
    Header header;
    char* sendBuf = new char[HEADER_LEN];
    header.flag = OVER;
    header.checksum = 0;
    u_short temp = cksum((u_short*)&header, HEADER_LEN);
    header.checksum= temp;
    memcpy(sendBuf, &header, HEADER_LEN);
    sendto(socketClient, sendBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
servAddrlen);
    cout << "发送END信息!" << endl;
    clock_t start = clock();

    //接收反馈
    char* recvBuf = new char[HEADER_LEN];
    while (1)
    {
        u_long mode = 1;
        ioctlsocket(socketClient, FIONBIO, &mode);
        while (recvfrom(socketClient, recvBuf, HEADER_LEN, 0,
(sockaddr*)&servAddr, &servAddrlen) <= 0)
        {
            if (clock() - start > MAX_TIME)
            {
                sendto(socketClient, sendBuf, HEADER_LEN, 0,
(sockaddr*)&servAddr, servAddrlen);
                cout << "TIME OUT:重新发送END信息" << endl;
                start = clock();
            }
        }
        memcpy(&header, recvBuf, HEADER_LEN);
        u_short checksum = cksum((u_short*)&header, HEADER_LEN);
        if (header.flag == OVER)
        {
            cout << "对方已成功接收文件!" << endl;
            break;
        }
        else
        {
            continue;
        }
    }
    u_long mode = 0;
```

```
    ioctlsocket(socketClient, FIONBIO, &mode);//改回阻塞模式
}
```

我们首先按照每个分组最多1024B，计算文件需要多少分组。随后调用send_package发送所有文件分组。在所有分组发送完成后，发送OVER报文，告知接收端所有文件均已发送。当接收到接收端的OVER报文，即可退出函数。

接收文件

```
int RecvMessage(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int& ClientAddrLen,
char* message)
{
    int fileLen= 0;//文件长度
    Header header;
    char* recvBuf = new char[MESSAGE_LEN];
    char* sendBuf = new char[MESSAGE_LEN];
    int index = 0;//当前确认到的分组序号

    while (1)
    {
        int length = recvfrom(sockServ, recvBuf, MESSAGE_LEN, 0,
(sockaddr*)&ClientAddr, &ClientAddrLen);//接收报文长度
        memcpy(&header, recvBuf, HEADER_LEN);
        //判断是否是结束
        if (header.flag == OVER && cksum((u_short*)&header, HEADER_LEN) == 0)
        {
            cout << "文件接收完毕" << endl;
            break;
        }
        if(header.flag == unsigned char(0)&& cksum((u_short*)&header,
HEADER_LEN) == 0)
        {
            if (index != int(header.index)&& cksum((u_short*)recvBuf, length -
sizeof(header)))
            {
                //依然返回index的确认，而不是header.index的确认
                header.flag = ACK;
                header.filelen = 0;
                header.index = (u_short)index;
                header.checksum = 0;
                u_short temp = cksum((u_short*)&header, HEADER_LEN);
                header.checksum = temp;
                memcpy(sendBuf, &header, HEADER_LEN);
                //重发该包的ACK
                sendto(sockServ, sendBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
ClientAddrLen);
                cout << "接收数据包的确认 flag=" << flag2str(int(header.flag)) << "
分组序号=" << (int)header.index << endl;
                continue;//丢弃该数据包
            }
            index = int(header.index);
            index %= 65536;
            //取出recvBuf中的内容
            cout << "接收分组 " << length - HEADER_LEN << " bytes flag=" <<
flag2str(int(header.flag)) << "分组序号 = "
                << (int)header.index << " 校验和=" << int(header.checksum) <<
endl;
```

```
            char* temp = new char[length - HEADER_LEN];
            memcpy(temp, recvBuf + HEADER_LEN, length - HEADER_LEN);
            memcpy(message + fileLen, temp, length - HEADER_LEN);
            fileLen = fileLen + int(header.filelen);

            //返回ACK
            header.flag = ACK;
            header.filelen = 0;
            header.index = (u_short)index;
            header.checksum = 0;
            u_short temp1 = cksum((u_short*)&header, HEADER_LEN);
            header.checksum = temp1;
            memcpy(sendBuf, &header, HEADER_LEN);
            //重发该包的ACK
            sendto(sockServ, sendBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
ClientAddrLen);
            cout << "接收数据包的确认 flag=" << flag2str(int(header.flag)) << " 分
组序号=" << (int)header.index << endl;
            index++;//确认该包之后，才能下一个包
            index %= 65536;
        }
    }
    //发送OVER信息
    header.flag = OVER;
    header.checksum = 0;
    u_short temp = cksum((u_short*)&header, HEADER_LEN);
    header.checksum = temp;
    memcpy(sendBuf, &header, HEADER_LEN);
    if (sendto(sockServ, sendBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1)
    {
        return -1;
    }
    return fileLen;
}
```

index是函数中设定待确认的分组序号，如果接收到报文序号与index不同，仍然只会确认index序号的报文。当所有分组接收完成后，并接收到来自发送端的OVER报文，接收端发送OVER报文进行确认，随后即可退出函数。

四次挥手关闭连接

发送端

int wave_hand(SOCKET& socketClient, SOCKADDR_IN& servAddr, int& servAddrlen)
{
  Header header;
  char* sendBuf = new char[HEADER_LEN];
  char* recvBuf = new char[HEADER_LEN];

```
//第一次挥手
header.flag = FIN;
header.checksum = 0;//校验和置0
u_short temp = cksum((u_short*)&header, HEADER_LEN);
header.checksum = temp;//计算校验和
memcpy(sendBuf, &header, HEADER_LEN);//将首部放入缓冲区
```

```
if (sendto(socketClient, sendBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
servAddrlen) == -1)
{
    return -1;
}
cout << "第一次挥手：flag=" << flag2str(int(header.flag)) << " 校验和=" <<
header.checksum << endl;
clock_t start = clock(); //记录发送第一次挥手时间

u_long mode = 1;
ioctlsocket(socketClient, FIONBIO, &mode);

//接收第二次挥手
while (recvfrom(socketClient, recvBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
&servAddrlen) <= 0)
{
    if (clock() - start > MAX_TIME)//超时，重新传输第一次挥手
    {
        sendto(socketClient, sendBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
servAddrlen);
        start = clock();
        cout << "第一次挥手超时，正在进行重传" << endl;
    }
}

//进行校验和检验
memcpy(&header, recvBuf, HEADER_LEN);
if (header.flag == ACK && cksum((u_short*)&header, HEADER_LEN) == 0)
{
    cout << "收到第二次挥手信息" << endl;
}
else
{
    cout << "连接发生错误，程序直接退出！" << endl;
    return -1;
}

//接收第三次挥手
while (1)
{
    int length = recvfrom(socketClient, recvBuf, HEADER_LEN, 0,
(sockaddr*)&servAddr, &servAddrlen);//接收报文长度
    memcpy(&header, recvBuf, HEADER_LEN);
    if (header.flag == FIN_ACK && cksum((u_short*)&header, HEADER_LEN) == 0)
    {
        cout << "成功接收第三次挥手信息" << endl;
        break;
    }
}
//发送第四次挥手
header.flag = ACK;
header.checksum = 0;
temp = cksum((u_short*)&header, HEADER_LEN);
header.checksum = temp;
memcpy(sendBuf, &header, HEADER_LEN);
if (sendto(socketClient, sendBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
servAddrlen) == -1)
{
```

```
    return -1;
}
cout << "第四次挥手：flag=" << flag2str(int(header.flag)) << " 校验和=" <<
header.checksum << endl;

start = clock();
while (1)
{
    if (clock() - start > MAX_TIME)//等待MAX_TIME
        break;
    if (recvfrom(socketClient, recvBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
&servAddrlen)>0)//接收到报文
    {
        memcpy(&header, recvBuf, HEADER_LEN);
        if (header.flag == FIN_ACK && cksum((u_short*)&header, HEADER_LEN) == 0)
        {
            cout << "ACK报文丢失，收到客户端第三次挥手重传" << endl;
            break;
        }
        //发送第四次挥手信息
        header.flag = ACK;
        header.checksum = 0;
        temp = cksum((u_short*)&header, HEADER_LEN);
        header.checksum = temp;
        memcpy(sendBuf, &header, HEADER_LEN);
        if (sendto(socketClient, sendBuf, HEADER_LEN, 0, (sockaddr*)&servAddr,
servAddrlen) == -1)
        {
            return -1;
        }
        cout << "第四次挥手：flag=" << flag2str(int(header.flag)) << " 校验和=" <<
header.checksum << endl;
        start = clock();//再次进入等待2MSL

    }
}

cout << "四次挥手结束，连接断开！" << endl;
return 1;
}
```

四次挥手的具体过程可见上面的协议设计。需要注意的是，在进行第四次挥手时，发送端向接收端发送ACK报文，接收端在接收到该报文后即可关闭连接。如果发送ACK报文后，发送端直接关闭连接，那么ACK报文丢失后，接收端就无法关闭连接。因此发送端在发送ACK报文后，等待2MSL，当没有收到接收端的重传第三次挥手报文，说明接收端已经接收到ACK确认报文，这时发送端才可以关闭连接。

接收端

```
int waveHand(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int& ClientAddrLen)
{
    Header header;
    char* recvBuf = new char[HEADER_LEN];
    char* sendBuf = new char[HEADER_LEN];
    while (1)
    {
        int length = recvfrom(sockServ, recvBuf, HEADER_LEN, 0,
(sockaddr*)&ClientAddr, &ClientAddrLen);//接收报文长度
```

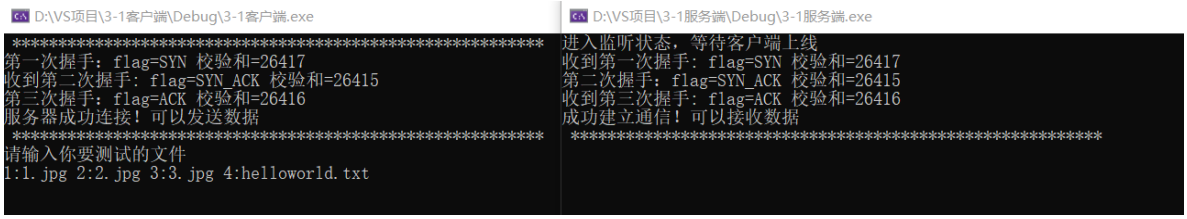
```
        memcpy(&header, recvBuf, HEADER_LEN);
        if (header.flag == FIN && cksum((u_short*)&header, HEADER_LEN) == 0)
        {
            cout << "成功接收第一次挥手信息" << endl;
            break;
        }
    }
    //发送第二次挥手信息
    header.flag = ACK;
    header.checksum = 0;
    u_short temp = cksum((u_short*)&header, HEADER_LEN);
    header.checksum = temp;
    memcpy(sendBuf, &header, HEADER_LEN);
    if (sendto(sockServ, sendBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1)
    {
        return -1;
    }
    cout << "第二次挥手：flag=" << flag2str(int(header.flag)) << " 校验和=" <<
header.checksum << endl;

    //第二次发送的ACK包可能会丢失，我们进行如下处理
    clock_t start = clock();//记录第二次挥手发送时间
    u_long mode = 1;
    ioctlsocket(sockServ, FIONBIO, &mode);
    while (1)
    {
        if (int(clock() - start) > int(MAX_TIME))//等待MAX_TIME
            break;
        if (recvfrom(sockServ, recvBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
&ClientAddrLen) > 0)//接收到报文
        {
            cout << "进入判断1" << endl;
            memcpy(&header, recvBuf, HEADER_LEN);
            if (header.flag == FIN && cksum((u_short*)&header, HEADER_LEN) == 0)
            {
                cout << "ACK报文丢失，收到客户端第一次挥手重传" << endl;
                break;
            }
            //发送第二次挥手信息
            header.flag = ACK;
            header.checksum = 0;
            u_short temp = cksum((u_short*)&header, HEADER_LEN);
            header.checksum = temp;
            memcpy(sendBuf, &header, HEADER_LEN);
            if (sendto(sockServ, sendBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1)
            {
                return -1;
            }
            cout << "第二次挥手：flag=" << flag2str(int(header.flag)) << " 校验和="
<< header.checksum << endl;
            start = clock();//再次进入等待2MSL
        }
    }

    //第三次挥手
    header.flag = FIN_ACK;
```

```
    header.checksum = 0;
    header.checksum = cksum((u_short*)&header, HEADER_LEN);//计算校验和
    if (sendto(sockServ, (char*)&header, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1)
    {
        return -1;
    }
    cout << "第三次挥手：flag=" << flag2str(int(header.flag)) << " 校验和=" <<
header.checksum << endl;
    start = clock();//如果超时要进行重传

    //接收第四次挥手
    while (recvfrom(sockServ, recvBuf, HEADER_LEN, 0, (sockaddr*)&ClientAddr,
&ClientAddrLen) <= 0)
    {
        if (clock() - start > MAX_TIME)
        {
            if (sendto(sockServ, (char*)&header, HEADER_LEN, 0,
(sockaddr*)&ClientAddr, ClientAddrLen) == -1)
            {
                return -1;
            }
            cout << "第三次挥手超时，正在进行重传" << endl;
        }
    }

    Header temp1;
    memcpy(&temp1, recvBuf, HEADER_LEN);
    if (temp1.flag == ACK && cksum((u_short*)&temp1, sizeof(temp1)) == 0)
    {
        cout << "成功接收第四次挥手" << endl;
    }
    else
    {
        cout << "发生错误,客户端关闭！" << endl;
        return -1;
    }
    cout << "四次挥手结束，连接断开！" << endl;
    return 1;
}
```

与第四次挥手的ACK报文相同，接收端在发送第二次挥手ACK报文时，也需要启动计时器等待 2 MSL，避免ACK报文丢失，无法断开连接。

三、实验结果展示

三次挥手建立连接



文件传输

D:\VS项目\3-1客户端\Debug\3-1客户端.exe

```
分组已被确认 flag=ACK 分组序号=0
发送分组 1024 bytes flag=NULL 分组序号=1 校验和=25393
分组已被确认 flag=ACK 分组序号=1
发送分组 1024 bytes flag=NULL 分组序号=2 校验和=25392
分组已被确认 flag=ACK 分组序号=2
发送分组 1024 bytes flag=NULL 分组序号=3 校验和=25391
分组已被确认 flag=ACK 分组序号=3
发送分组 1024 bytes flag=NULL 分组序号=4 校验和=25390
分组已被确认 flag=ACK 分组序号=4
发送分组 1024 bytes flag=NULL 分组序号=5 校验和=25389
分组已被确认 flag=ACK 分组序号=5
发送分组 1024 bytes flag=NULL 分组序号=6 校验和=25388
分组已被确认 flag=ACK 分组序号=6
发送分组 1024 bytes flag=NULL 分组序号=7 校验和=25387
分组已被确认 flag=ACK 分组序号=7
发送分组 1024 bytes flag=NULL 分组序号=8 校验和=25386
分组已被确认 flag=ACK 分组序号=8
发送分组 1024 bytes flag=NULL 分组序号=9 校验和=25385
分组已被确认 flag=ACK 分组序号=9
发送分组 1024 bytes flag=NULL 分组序号=10 校验和=25384
分组已被确认 flag=ACK 分组序号=10
发送分组 1024 bytes flag=NULL 分组序号=11 校验和=25383
分组已被确认 flag=ACK 分组序号=11
发送分组 1024 bytes flag=NULL 分组序号=12 校验和=25382
分组已被确认 flag=ACK 分组序号=12
发送分组 1024 bytes flag=NULL 分组序号=13 校验和=25381
分组已被确认 flag=ACK 分组序号=13
发送分组 1024 bytes flag=NULL 分组序号=14 校验和=25380
分组已被确认 flag=ACK 分组序号=14
发送分组 1024 bytes flag=NULL 分组序号=15 校验和=25379
```

D:\VS项目\3-1服务端\Debug\3-1服务端.exe

```
接收数据包的确认 flag=ACK 分组序号=0
接收分组 1024 bytes flag=NULL分组序号 = 1 校验和=25393
接收数据包的确认 flag=ACK 分组序号=1
接收分组 1024 bytes flag=NULL分组序号 = 2 校验和=25392
接收数据包的确认 flag=ACK 分组序号=2
接收分组 1024 bytes flag=NULL分组序号 = 3 校验和=25391
接收数据包的确认 flag=ACK 分组序号=3
接收分组 1024 bytes flag=NULL分组序号 = 4 校验和=25390
接收数据包的确认 flag=ACK 分组序号=4
接收分组 1024 bytes flag=NULL分组序号 = 5 校验和=25389
接收数据包的确认 flag=ACK 分组序号=5
接收分组 1024 bytes flag=NULL分组序号 = 6 校验和=25388
接收数据包的确认 flag=ACK 分组序号=6
接收分组 1024 bytes flag=NULL分组序号 = 7 校验和=25387
接收数据包的确认 flag=ACK 分组序号=7
接收分组 1024 bytes flag=NULL分组序号 = 8 校验和=25386
接收数据包的确认 flag=ACK 分组序号=8
接收分组 1024 bytes flag=NULL分组序号 = 9 校验和=25385
接收数据包的确认 flag=ACK 分组序号=9
接收分组 1024 bytes flag=NULL分组序号 = 10 校验和=25384
接收数据包的确认 flag=ACK 分组序号=10
接收分组 1024 bytes flag=NULL分组序号 = 11 校验和=25383
接收数据包的确认 flag=ACK 分组序号=11
接收分组 1024 bytes flag=NULL分组序号 = 12 校验和=25382
接收数据包的确认 flag=ACK 分组序号=12
接收分组 1024 bytes flag=NULL分组序号 = 13 校验和=25381
接收数据包的确认 flag=ACK 分组序号=13
接收分组 1024 bytes flag=NULL分组序号 = 14 校验和=25380
接收数据包的确认 flag=ACK 分组序号=14
接收分组 1024 bytes flag=NULL分组序号 = 15 校验和=25379
```

四次挥手关闭连接

D:\VS项目\3-1客户端\Debug\3-1客户端.exe

```
分组已被确认 flag=ACK 分组序号=1813
发送END信息!
对方已成功接收文件!
传输总时间为:10s
吞吐率为:185735byte/s
 ************************************************************
第一次挥手：flag=FIN 校验和=26414
收到第二次挥手信息
成功接收第三次挥手信息
第四次挥手：flag=ACK 校验和=26416
四次挥手结束，连接断开!
请按任意键继续. . .
```

选择 D:\VS项目\3-1服务端\Debug\3-1服务端.exe

```
接收数据包的确认 flag=ACK 分组序号=1809
接收分组 1024 bytes flag=NULL分组序号 = 1810 校验和=23584
接收数据包的确认 flag=ACK 分组序号=1810
接收分组 1024 bytes flag=NULL分组序号 = 1811 校验和=23583
接收数据包的确认 flag=ACK 分组序号=1811
接收分组 1024 bytes flag=NULL分组序号 = 1812 校验和=23582
接收数据包的确认 flag=ACK 分组序号=1812
接收分组 841 bytes flag=NULL分组序号 = 1813 校验和=23764
接收数据包的确认 flag=ACK 分组序号=1813
文件接收完毕
 ************************************************************
成功接收第一次挥手信息
第二次挥手：flag=ACK 校验和=26416
第三次挥手：flag=FIN_ACK 校验和=26412
成功接收第四次挥手
四次挥手结束，连接断开!
1.jpg文件已成功下载到本地
请按任意键继续. . .
```

文件传输结果

名称	修改日期	类型	大小
Debug	2022/11/19 18:38	文件夹	
1.jpg	2022/11/19 20:49	JPG 图片文件	1,814 KB
3-1服务端.vcxproj	2022/11/19 9:19	VC++ Project	8 KB
3-1服务端.vcxproj.filters	2022/11/19 9:19	FILTERS 文件	2 KB
3-1服务端.vcxproj.user	2022/11/16 15:42	USER 文件	1 KB
server.cpp	2022/11/19 18:40	CPP 文件	12 KB
test.cpp	2022/11/19 9:21	CPP 文件	11 KB

在接收端目录下多了1.jpg。打开如下：

经过对比，该图片与发送端的图片大小、信息完全相同，说明协议完成了可靠传输。