基于UDP服务设计可靠传输协议并编程实现

第二次实验报告

1811379 李毅

日期：2020年12月13日

目录

摘要

本实验使用了基于UDP的非阻塞socket编程，完成了建立连接、差错检测、确认重传，累计确认等部分。流量控制采用滑动窗口，并完成了给定测试文件的传输。

关键字：UDP，可靠传输，滑动窗口，累计确认

一、引言

（1）实验环境

- os： windows10
- c++11
- mingw

（2）实验实现思路

在本机上同时运行服务器端和客户端的可执行代码，服务器端发送文件，接收端接收文件。

（3）实验完成情况

完成了建立连接、差错检测、确认重传，累计确认等部分。流量控制采用滑动窗口，完成了给定测试文件的传输。

二、协议

（1）报文头

```
struct DataPackage
{
    unsigned short sourcePort; // 2B 16bits
    unsigned short destPort;   // 2B 16bits
    unsigned short window;     // 窗口大小，字节数2B
    unsigned short checkSum;   // 校验和 2B
    unsigned short len;        // 数据部分长度
    unsigned short flag;       // 2B是否需要分段传输，不需要则为0，否则为分段传输的数量
    unsigned short offset; // 分段传输时的偏移号
    unsigned short timeLive; // 生存期
    unsigned int seqNum;  // 序列号 4B
    unsigned int ackNum;  // ACK号 4B
    bool ackflag = false; // 1B，判断该报文是否是ACK
    char message[0];      // 消息部分，柔性数组，可变长度
};
```

1、校验和计算

将数据报看成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

```
    // 计算checksum
    void CheckSum(unsigned short *buf)
    {
        int count = sizeof(this->message) / 2;
        register unsigned long sum = 0;
        while (count--)
        {
            sum += *buf++;
            if (sum & 0xFFFF0000)
            {
                sum &= 0xFFFF;
                sum++;
            }
        }
        this->checkSum = ~(sum & 0xFFFF);
    }
```

2、差错检测

将接收到的数据包的message段看成16位整数序列，进行16位二进制反码求和运算，计算结果与数据包的记录的checksum比较，如果形同则数据包没有损坏，否则数据包发生损坏

```
// 判断包是否损坏
bool corrupt(DataPackage *data)
{
    int count = sizeof(data->message) / 2;
    register unsigned long sum = 0;
    unsigned short *buf = (unsigned short *)(data->message);
    while (count--)
    {
        sum += *buf++;
        if (sum & 0xFFFF0000)
        {
            sum &= 0xFFFF;
            sum++;
        }
    }
    // 如果计算出的checksum与数据包的记录的checksum相同则数据包没有损坏
    if (data->checkSum == ~(sum & 0xFFFF))
        return true;
    return false;
}
```

（2）窗口的操作

1、窗口的初始化

初始化窗口，并发送初始化窗口中的内容

```
// 窗口初始化
void window_init()
{
    // 如果文件比较小，不能填满所有窗口
    if (sendFile->packageSum <= WINDOWSIZE)
    {
        // 打开文件
        is->open(sendFile->filePath, ifstream::in | ios::binary);
        // 将数据包加入到缓存当中
        for (int i = 0; i < sendFile->packageSum - 1; i++)
        {
            DataPackage *data = (DataPackage *)malloc(sizeof(DataPackage) +
(BUFFER - sizeof(DataPackage)) * sizeof(char));
            char *tmp = new char[BUFFER - sizeof(DataPackage)];
            is->read(tmp, BUFFER - sizeof(DataPackage) - 1);
            tmp[BUFFER - sizeof(DataPackage) - 1] = '\0';
            Strcpyn(data->message, tmp, BUFFER - sizeof(DataPackage));
            delete tmp;
            data->len = BUFFER - sizeof(DataPackage);
            data->make_pkt(SERVER_PORT, SERVER_PORT, nextseqnum, WINDOWSIZE);
            nextseqnum++;
            // 计算checksum
            data->CheckSum((unsigned short *)data->message);
            // 文件需要分段发送
            data->flag = sendFile->packageSum;
            // 该数据包在第几个分段
            data->offset = offset;
            offset++;
            // 拷贝在缓冲区中
            Strcpyn(sndpkt[i], (char *)data, sizeof(DataPackage) + (BUFFER -
sizeof(DataPackage)) * sizeof(char));

            delete data;
        }
        // 最后一部分
        DataPackage *data = (DataPackage *)malloc(sizeof(DataPackage) +
(sendFile->fileLenRemain + 1) * sizeof(char));
        char *tmp = new char[sendFile->fileLenRemain + 1];
        is->read(tmp, sendFile->fileLenRemain);
        is->close();
        tmp[sendFile->fileLenRemain] = '\0';
        Strcpyn(data->message, tmp, sendFile->fileLenRemain + 1);
        delete tmp;
        data->len = sendFile->fileLenRemain + 1;
        data->make_pkt(SERVER_PORT, SERVER_PORT, nextseqnum, WINDOWSIZE);
        nextseqnum++;

        data->CheckSum((unsigned short *)data->message);
        data->flag = sendFile->packageSum;
        data->offset = offset;
        offset++;
        Strcpyn(sndpkt[sendFile->packageSum - 1], (char *)data,
sizeof(DataPackage) + (sendFile->fileLenRemain + 1) * sizeof(char));
```

```
        delete data;
        // 其他地方都置为0
        for (int i = sendFile->packageSum; i < WINDOWSIZE; i++)
            ZeroMemory(sndpkt[i], BUFFER);
    }
    else
    { // 如果文件太大，一个窗口放不下
        is->open(sendFile->filePath, ifstream::in | ios::binary);
        for (int i = 0; i < WINDOWSIZE; i++)
        {
            DataPackage *data = (DataPackage *)malloc(sizeof(DataPackage) +
(BUFFER - sizeof(DataPackage)) * sizeof(char));
            char *tmp = new char[BUFFER - sizeof(DataPackage)];
            is->read(tmp, BUFFER - sizeof(DataPackage) - 1);
            tmp[BUFFER - sizeof(DataPackage) - 1] = '\0';
            Strcpyn(data->message, tmp, BUFFER - sizeof(DataPackage));
            delete tmp;

            data->len = BUFFER - sizeof(DataPackage);
            data->make_pkt(SERVER_PORT, SERVER_PORT, nextseqnum, WINDOWSIZE);
            nextseqnum++;
            // 计算checksum
            data->CheckSum((unsigned short *)data->message);
            // 文件需要分段发送
            data->flag = sendFile->packageSum;
            data->offset = offset;
            offset++;
            // 拷贝在缓冲区中
            Strcpyn(sndpkt[i], (char *)data, sizeof(DataPackage) + (BUFFER -
sizeof(DataPackage)) * sizeof(char));
            delete data;
        }
        // 记录现在的文件位置
        pos = is->tellg();
        // 关闭文件
        is->close();
    }
    // 开始发送现在的数据包
    int k = 0;
    while (k <= min_size)
    {
        DataPackage *tmp = (DataPackage *)malloc(sizeof(DataPackage) + (BUFFER -
sizeof(DataPackage)) * sizeof(char));
        extract_pkt(sndpkt[k], *tmp);
        // 发送窗口中的数据包
        if (tmp->seqNum < base + WINDOWSIZE)
            sendto(sockServer, (char *)(tmp), sizeof(DataPackage) + tmp->len, 0,
(SOCKADDR *)&addrClient, sizeof(SOCKADDR));
        // 打开计时器
        ack_timer[tmp->seqNum % WINDOWSIZE].Start();
        k++;
    }
}
```

2、窗口的移动

窗口移动，每次向前移动一格，如果需要读取新的数据，就读取数据并发送，同时打开计时器

```
// 窗口移动
void window_shift()
{
    // 缓存区中往前移一个
    for (int i = 0; i < WINDOWSIZE - 1; i++)
        Strcpyn(sndpkt[i], sndpkt[i + 1], BUFFER);
    // 窗口移动时需要新增加数据
    if (base + WINDOWSIZE <= sendFile->packageSum)
    {
        // 还没有读到文件的最后一部分
        if (offset != sendFile->packageSum - 1)
        {
            DataPackage *data = (DataPackage *)malloc(sizeof(DataPackage) +
(BUFFER - sizeof(DataPackage)) * sizeof(char));

            char *tmp = new char[BUFFER - sizeof(DataPackage)];
            // 打开文件
            is->open(sendFile->filePath, ifstream::in | ios::binary);
            // 定位到上次读取的位置
            is->seekg(pos);
            is->read(tmp, BUFFER - sizeof(DataPackage) - 1);
            // 获取现在的位置
            pos = is->tellg();
            // 关闭文件
            is->close();

            tmp[BUFFER - sizeof(DataPackage) - 1] = '\0';

            Strcpyn(data->message, tmp, BUFFER - sizeof(DataPackage));
            delete tmp;
            data->len = BUFFER - sizeof(DataPackage);
            data->make_pkt(SERVER_PORT, SERVER_PORT, nextseqnum, WINDOWSIZE);
            nextseqnum++;
            // 计算checksum
            data->CheckSum((unsigned short *)data->message);
            // 标志文件分段发送
            data->flag = sendFile->packageSum;
            // 现在是第几段
            data->offset = offset;
            offset++;
            // 拷贝在缓冲区中
            Strcpyn(sndpkt[WINDOWSIZE - 1], (char *)data, sizeof(DataPackage) +
(BUFFER - sizeof(DataPackage)) * sizeof(char));
            // 发送数据
            if (data->seqNum < base + WINDOWSIZE)
                sendto(sockServer, (char *)(data), sizeof(DataPackage) + data-
>len, 0, (SOCKADDR *)&addrClient, sizeof(SOCKADDR));
            // 开启计时器
            ack_timer[data->seqNum % WINDOWSIZE].Start();
            delete data;
        }
        else
        { // 如果是文件的最后一部分
```

```
            DataPackage *data = (DataPackage *)malloc(sizeof(DataPackage) +
(sendFile->fileLenRemain + 1) * sizeof(char));
            char *tmp = new char[sendFile->fileLenRemain + 1];
            is->open(sendFile->filePath, ifstream::in | ios::binary);
            is->seekg(pos);
            is->read(tmp, sendFile->fileLenRemain);
            pos = is->tellg();
            is->close();

            tmp[sendFile->fileLenRemain] = '\0';
            Strcpyn(data->message, tmp, sendFile->fileLenRemain + 1);
            delete tmp;
            data->len = sendFile->fileLenRemain + 1;
            data->make_pkt(SERVER_PORT, SERVER_PORT, nextseqnum, WINDOWSIZE);
            nextseqnum++;
            data->CheckSum((unsigned short *)data->message);
            data->flag = sendFile->packageSum;
            data->offset = offset;
            offset++;
            Strcpyn(sndpkt[WINDOWSIZE - 1], (char *)data, sizeof(DataPackage) +
(sendFile->fileLenRemain + 1) * sizeof(char));
            // 发送数据
            if (data->seqNum < base + WINDOWSIZE)
                sendto(sockServer, (char *)(data), sizeof(DataPackage) + data-
>len, 0, (SOCKADDR *)&addrClient, sizeof(SOCKADDR));
            // 开启计时器
            ack_timer[data->seqNum % WINDOWSIZE].Start();
            delete data;
        }
    }
    else
    {
        // 不需要增加新数据时
        ZeroMemory(sndpkt[WINDOWSIZE - 1], BUFFER);
    }
}
```

(3) 累计确认

接收端

初始化累计确认的参数

```
    Timer *t = new Timer();
    t->timeout = 0.5;  // 500毫秒累计确认
    int multipkgs = 5; // 等待几个包传一次
    int getpkgs = 0; // 现在已经获得的包的个数
```

```
                    else if ((recvData.flag > 0) & hasseqnum(recvData,
expectedseqnum) & (!corrupt(&recvData)))
                    {

                        // 文件分段接收
                        cout << "pkg " << recvData.seqNum << " not bad!" <<
endl;
                        // 重新开始计时
```

```
                        if (getpkgs == 0)
                            t->Start();
                        // 如果没有超过500ms
                        if (!t->TimeOut())
                        {
                            getpkgs++;
                                // 期望值seq++
                            expectedseqnum++;
                            // 如果达到了累计的数据包就发送ack
                            if (getpkgs == multipkgs)
                            {
                                // 重置getpkgs
                                getpkgs = 0;
                                DataPackage sendData;
                                sendData.ackNum = recvData.seqNum;
                                sendData.make_pkt(CLIENT_PORT, CLIENT_PORT, 0,
WINDOWSIZE);
                                sendData.CheckSum((unsigned short
*)sendData.message);
                                sendData.ackflag = 1;
                                sendData.len = 0;
                                // 当前acknum
                                curACKnum = sendData.ackNum;
                                sendto(socketClient, (char *)(&sendData),
sizeof(DataPackage) + sendData.len, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                            }
                            // cout << "begin write to file: " << recvFile-
>filePath << endl;
                            printf("begin write %d segment ...\n",
recvData.offset);
                            if (recvData.offset < recvData.flag) // 直到所有分段发
完
                            {
                                // 打开文件在文件最后写入
                                outfile->open(recvFile->filePath, ios::out |
ios::binary | ios::app);
                                outfile->write(recvData.message, recvData.len -
1);
                                outfile->close();

                                if (recvData.offset == recvData.flag - 1)
                                {
                                    cout << "write file success!" << endl;
                                    // 发送最后一段的ack
                                    DataPackage sendData;
                                    sendData.ackNum = recvData.seqNum;
                                    sendData.make_pkt(CLIENT_PORT, CLIENT_PORT,
0, WINDOWSIZE);
                                    sendData.CheckSum((unsigned short
*)sendData.message);
                                    sendData.ackflag = 1;
                                    sendData.len = 0;
                                    // 当前acknum
                                    curACKnum = sendData.ackNum;
                                    sendto(socketClient, (char *)(&sendData),
sizeof(DataPackage) + sendData.len, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
```

```
                                stage = 10;
                                runFlag = false;
                                logout = true;
                            }
                        }
                    }
                    else // 如果超过了500就抛弃掉现在的数据包，重新开始计时
                    {
                        getpkgs = 0;
                        // 发送ACK
                        DataPackage sendData;
                        // 发送上一次的ack
                        sendData.ackNum = curACKnum;
                        sendData.make_pkt(CLIENT_PORT, CLIENT_PORT, 0,
WINDOWSIZE);
                        sendData.CheckSum((unsigned short
*)sendData.message);
                        sendData.ackflag = 1;
                        sendData.len = 0;
                        sendto(socketClient, (char *)(&sendData),
sizeof(DataPackage) + sendData.len, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                    }
                    stage = 2;
                }
                else if ((recvData.flag > 0) & (!hasseqnum(recvData,
expectedseqnum)) & (!corrupt(&recvData)))
                {
                    // 如果收到的不是对应期望的分组
                    cout << "get " << recvData.ackNum << " not get
expectedseqnum " << expectedseqnum << endl;
                    DataPackage sendData;
                    // 发送上一次的ack
                    sendData.ackNum = curACKnum;
                    sendData.make_pkt(CLIENT_PORT, CLIENT_PORT, 0,
WINDOWSIZE);
                    sendData.CheckSum((unsigned short *)sendData.message);
                    // 记为ack数据包
                    sendData.ackflag = 1;
                    sendData.len = 0;
                    sendto(socketClient, (char *)(&sendData),
sizeof(DataPackage) + sendData.len, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                }
            }

            break;
        }
```

发送端

```
                    ZeroMemory(buffer, sizeof(buffer));
                    recvSize = recvfrom(sockServer, buffer, BUFFER, 0,
((SOCKADDR *)&addrClient), &length);
                    if (recvSize < 0)
                    {
                        IfTimeout(ack_timer[expectACKnum % WINDOWSIZE]);
```

```
                            }
                            else
                            {
                                // cout << "get ack!!!" << endl;
                                DataPackage *tmpData = new DataPackage();
                                extract_pkt(buffer, *tmpData);
                                // 接收到ACK，ACK不是重复的ACK，且包没有损坏，且收到时没
有超时
                                if (tmpData->ackflag & (tmpData->ackNum !=
lastACKnum) & (!corrupt(tmpData)))
                                {
                                    cout << "ack success!!! ACK: " << tmpData-
>ackNum << endl;
                                    // 累计确认
                                    for (int i = base; i <= tmpData->ackNum;
i++)
                                        // 窗口移动
                                        window_shift();
                                    // base等于接收的ACK+1
                                    base = tmpData->ackNum + 1;
                                    expectACKnum = tmpData->ackNum + 1;
                                    lastACKnum = tmpData->ackNum;

                                }
                                else
                                {
                                    cout << "something error" << endl;
                                    IfTimeout(ack_timer[expectACKnum %
WINDOWSIZE]);
                                }
                                delete tmpData;
                            }
```

（4）建立连接过程

1、等待客户端发来连接请求 “0”

```
    while (!logout)
    {
        if (!is_connect)
            cout << "wait connect with client ..." << endl;
        ZeroMemory(buffer, sizeof(buffer));
        recvSize = recvfrom(sockServer, buffer, BUFFER, 0, ((SOCKADDR
*)&addrClient), &length);

        // 等待连接
        int i = 0;
        if (recvSize < 0)
        {
            i++;
            Sleep(500);
        }
        if (i > 20)
        {
            printf("can't connect\n exit program ...");
```

```
            exit(1);
        }
        int waitCount = 0;
        // 如果发来连接请求开始下面的步骤
        if (strcmp(buffer, "0") == 0)
        {

        }
```

2、服务端向客户端发送“S1”

服务端向客户端发送“S1”表示，服务器已收到连接请求可以正常连接

```
                    cout << "send S1 to client ..." << endl;
                    buffer[0] = S1;
                    sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR
*)&addrClient, sizeof(SOCKADDR));
                    Sleep(100);
                    stage = 1;
                    break;
```

3、客户端向服务端发送“S2”

客户端向服务端发送“S2”，表示已收到服务端的“S1”

```
                if ((unsigned char)buffer[0] == S1)
                {
                    cout << "get S1 form server!" << endl;
                    buffer[0] = S2;
                    buffer[1] = '\0';
                    cout << "send S2 to server ..." << endl;
                    sendto(socketClient, buffer, 2, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                    stage = 1;
                }
                break;
```

4、服务端收到“S2”表示连接已正常建立

服务端收到“S2”表示连接已正常建立， 可以开始后面的操作

```
                    else if ((unsigned char)buffer[0] == S2)
                    {
                        printf("get S2 from client, connect with client !!!\n");
                        is_connect = true; // 客户端与服务器已经连接
                        stage = 2;
                    }
                    break;
```

（5）发送数据过程

服务端发送文件，客户端接收文件

1、服务端选择发送文件的路径和文件名

服务端选择发送文件的路径，并向服务器端通知，文件路径已经选择完比

```
                    cout << "==========file transport==========" << endl;
                    string filePath;
                    cout << "Please input file path which needs to be send: " <<
endl;
                    cin >> filePath;
                    if (filePath == "-1")
                    {
                        filePath = ".\\test\\helloworld.txt";
                        cout << "the default file path is " << filePath << endl;
                    }

                    sendFile->initFile(true, filePath);
                    sendFile->ReadFile();
                    first_open_file = true;

                    buffer[0] = S3;
                    sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR
*)&addrClient, sizeof(SOCKADDR));
                    stage = 10;
                    break;
```

服务器端向客户端发送下载的文件名

```
                case 10:
                {
                    string filename;
                    cout << "Please input file name: " << endl;
                    cin >> filename;
                    if (filename == "-1")
                    {
                        filename = "helloworld.txt";
                        cout << "the default file name is " << filename << endl;
                    }

                    for (int i = 0; i < filename.length(); i++)
                        buffer[i] = filename[i];
                    buffer[filename.length()] = '\0';
                    sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR
*)&addrClient, sizeof(SOCKADDR));
                    stage = 3;
                    break;
                }
```

2、客户端选择文件下载路径，并接收文件名

客户端收到服务器端发送的下载文件已经选择完毕的消息，并开始选择文件的下载路径

```
                if ((unsigned char)buffer[0] == S3)
                {
                    cout << "server has select file to transport!!!" << endl;
                    cout << "please select path to download this file ..." <<
endl;
                    filePath = ".\\";
                    cin >> filePath;
                    stage = 50;
                }
                break;
```

客户端接收下载的文件名，并向服务器端发送“S4”，表示已经准备好接收文件了

```
            case 50:
            {// 接收下载的文件名
                if (recvSize >= 0)
                {
                    string filename = buffer;
                    recvFile->initFile(false, filePath + filename);
                    recvFile->WriteFile();
                    ZeroMemory(buffer, sizeof(buffer));
                    buffer[0] = S4;
                    buffer[1] = '\0';
                    sendto(socketClient, buffer, 2, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                    stage = 2;
                    break;
                }
            }
```

3、服务器端收到发来的“S4”，开始传输文件

```
                    ZeroMemory(buffer, sizeof(buffer));
                    recvSize = recvfrom(sockServer, buffer, BUFFER, 0,
((SOCKADDR *)&addrClient), &length);
                    if ((unsigned char)buffer[0] == S4)
                    {
                        printf("get S4 from client !!!\n");
                        printf("this client has select download path begin
tansport this file !!!\n");
                        stage = 4;
                    }
                    break;
```

4、发送文件

初始化参数

```
char buffer[BUFFER];                        //数据发送接收缓冲区
Timer *ack_timer = new Timer[WINDOWSIZE]; // 标志每一个包的Timer

char sndpkt[WINDOWSIZE][BUFFER]; // 缓冲区
int nextseqnum = 0;              // 下一个seq的num
// 在接收到ACK之后才变化的值
int base = 0;          // 窗口的第一个序号
```

```
int lastACKnum = -1;  // 用于检测重复ACK，如果是重复的ACK就会被忽略
int expectACKnum = 0; // 期望的ACKnum
// 期望的ACK num = base
// 只有getACKnum == expecACKnum，窗口才发生移动，窗口内的数据才被发送，定时器才开始计时
streampos pos;                  //文件光标位置，用于记录上一次光标的位置
unsigned short offset = 0;      // 发送文件offset
ifstream *is = new ifstream(); // 读文件
File *sendFile = new File();    // 文件操作
int min_size = 0;               // 初始化中要发送的包的个数
```

```
    bool is_connect = false; // 判断是否连接
    Timer *t = new Timer();  // 计时器
    Timer *totalT = new Timer();

    bool logout = false;          // 是否登出
    bool runFlag = true;          //是否运行
    bool first_open_file = true; // 是否是第一次打开该文件
```

先根据文件长度，判断是否需要多次发送

如果只需要发送一次的话，将报文头中的

`flag` 标志位置为0，表示该文件不需要多次传输

序列号 sunNum = 0，

将文件中的内容，发送给客户端，等待客户端的ACK 0，如果超时就会重新发送

直到收到对方的正确的，没有损坏的ACK0的话就结束文件传输

```
                    // 文件小，直接一个包发过去，文件大需要多次发
                    // 文件长度只有package时
                    cout << "begin transport file ..." << endl;
                    if (sendFile->packageSum == 1)
                    {
                        // 总的文件发送定时器打开
                        totalT->Start();
                        // 读取文件中内容到 message里， dataPackage发过去，然后检测是否
会收到ACK，如果有就说明，发送完毕，
                        // 如果没有就超时重传
                        char *tmp = new char[sendFile->fileLen + 1];
                        // 打开文件
                        is->open(sendFile->filePath, ifstream::in |
ios::binary);
                        // 读取文件内容
                        is->read(tmp, sendFile->fileLen);
                        // 关闭文件
                        is->close();

                        tmp[sendFile->fileLen] = '\0';

                        DataPackage *data = (DataPackage
*)malloc(sizeof(DataPackage) + (sendFile->fileLen + 1) * sizeof(char));
                        Strcpyn(data->message, tmp, sendFile->fileLen + 1);
```

```
                        data->make_pkt(SERVER_PORT, SERVER_PORT, nextseqnum,
WINDOWSIZE);
                        // 计算checksum
                        data->CheckSum((unsigned short *)data->message);
                        data->len = sendFile->fileLen + 1;
                        data->flag = 0;
                        data->offset = 0;
                        cout << "msg: " << data->message << endl;
                        Strcpyn(sndpkt, (char *)data, sizeof(DataPackage) +
data->len);
                        // 计算CheckSum，并开始定时器t
                        rdt_send(data, t);
                        delete data;
                        cout << "send success!!!" << endl;
                        // 等待接收ACK
                        while (true)
                        {
                            ZeroMemory(buffer, sizeof(buffer));
                            recvSize = recvfrom(sockServer, buffer, BUFFER, 0,
((SOCKADDR *)&addrClient), &length);
                            // 如果没有收到，就等待，并判断是否超时
                            if (recvSize < 0)
                            {
                                IfTimeout(t);
                            }
                            else
                            {
                                // 如果收到了客户端发来的消息
                                cout << "get ack!!!" << endl;
                                DataPackage *tmpData = new DataPackage();
                                // 解析数据包，将char* 转为DataPackage *类型
                                extract_pkt(buffer, *tmpData);
                                // 发过来的是ACK的数据，发过来ackNum是所期望的ackNum，
并且数据包没有损坏
                                if (tmpData->ackflag & (tmpData->ackNum ==
nextseqnum - 1) & (!corrupt(tmpData)))
                                {
                                    cout << "ack success!!!" << endl;
                                    totalT->Show();
                                    stage = 5;
                                    ack[nextseqnum - 1 - base] = 1;
                                    base = nextseqnum;
                                    break;
                                }
                                else
                                {// 否则说明收到的数据包有问题，判断收到正确的数据包是否超
时
                                    cout << "something error" << endl;
                                    IfTimeout(t);
                                }
                                delete tmpData;
                            }
                        }
                    }
```

如果需要分多次发送的话，需要将报文中的

flag 设为需要发送的数据次数

`flag = sendFile->packageSum`

```
                    else // 文件需要多次传输
                    {

                        // 如果是第一次打开文件
                        if (first_open_file)
                        {
                            // 总的计时器打开
                            totalT->Start();
                            first_open_file = false;
                            // 窗口初始化参数，mini_size赋值
                            if (sendFile->packageSum - 1 <= WINDOWSIZE - 1)
                                min_size = sendFile->packageSum - 1;
                            else
                            {
                                min_size = WINDOWSIZE - 1;
                            }
                        }
                        // 窗口初始化
                        window_init();

                        // 发完现在所有窗口中的内容了，等待接收ACK
                        while (true)
                        {
                            // 如果文件发完了就退出循环
                            if (lastACKnum == sendFile->packageSum - 1)
                            {
                                totalT->Show();
                                stage = 5;
                                offset = 0;
                                break;
                            }
                            ZeroMemory(buffer, sizeof(buffer));
                            recvSize = recvfrom(sockServer, buffer, BUFFER, 0,
((SOCKADDR *)&addrClient), &length);
                            if (recvSize < 0)
                            {
                                IfTimeout(ack_timer[expectACKnum % WINDOWSIZE]);
                            }
                            else
                            {
                                // cout << "get ack!!!" << endl;
                                DataPackage *tmpData = new DataPackage();
                                extract_pkt(buffer, *tmpData);
                                // 接收到ACK，ACK不是重复的ACK，且包没有损坏，且收到时没
有超时
                                if (tmpData->ackflag & (tmpData->ackNum !=
lastACKnum) & (!corrupt(tmpData)))
                                {
                                    cout << "ack success!!! ACK: " << tmpData-
>ackNum << endl;
                                    // 累计确认
                                    for (int i = base; i <= tmpData->ackNum;
i++)
                                        // 窗口移动
                                        window_shift();
```

```
                                    // 等于接收的ACK+1
                                    base = tmpData->ackNum + 1;
                                    expectACKnum = tmpData->ackNum + 1;
                                    lastACKnum = tmpData->ackNum;

                                }
                                else
                                {
                                    cout << "something error" << endl;
                                    IfTimeout(ack_timer[expectACKnum %
WINDOWSIZE]);
                                }
                                delete tmpData;
                            }
                        }
                    }
                    break;
                }
```

(4) 接收文件

初始化参数

```
int expectedseqnum = 1; // 期望的seqnum
```

```
    ofstream *outfile = new ofstream(); // 写文件
    File *recvFile = new File();        // 文件操作
    bool logout = false;                // 登出
    bool runFlag = true;                // 是否run
    unsigned int curACKnum = 0; //当前的ack
    string filePath = "";//文件路径
    Timer *t = new Timer(); //累计确认计时器
    t->timeout = 0.5;  // 500毫秒累计确认
    int multipkgs = 5; //等待几个包传一次
    int getpkgs = 0;   // 获得的包的个数
```

文件长度小于`BUFFER-sizeof(DataPackage)`时

只需要接收一次文件

```
                if (recvSize >= 0)
                {
                    cout << "get pkg from server ..." << endl;

                    DataPackage recvData;
                    extract_pkt(buffer, recvData);
                    // 如果是不分段的文件，所期望的包，且包中的数据没有损坏的话
                    if ((recvData.flag == 0) & hasseqnum(recvData,
expectedseqnum) & (!corrupt(&recvData)))
                    { // 文件不分段的接收
                        cout << "pkg not bad!" << endl;
                        DataPackage *data = (DataPackage
*)malloc(sizeof(DataPackage));
                        // acknum等于接收的seqnum
                        data->ackNum = recvData.seqNum;
                        data->make_pkt(CLIENT_PORT, CLIENT_PORT, 0, WINDOWSIZE);
```

```
                        // 计算校验和
                        data->CheckSum((unsigned short *)data->message);
                        // 标志该数据包为ack包
                        data->ackflag = 1;
                        // 数据部分长度为0
                        data->len = 0;
                        //发送数据包
                        sendto(socketClient, (char *)data, sizeof(DataPackage) +
data->len, 0, (SOCKADDR *)&addrServer, sizeof(SOCKADDR));
                        // 期望的seq++
                        expectedseqnum++;

                        cout << "begin write to file: " << recvFile->filePath <<
endl;
                        // 打开文件开始写入
                        outfile->open(recvFile->filePath, ios::out | ios::binary
| ios::app);
                        outfile->write(recvData.message, recvData.len - 1);
                        outfile->close();
                        //关闭套接字
                        cout << "write file success!" << endl;
                        // 结束文件传输
                        runFlag = false;
                        logout = true;
                        stage = 10;
                    }
```

文件长度大于`BUFFER-sizeof(DataPackage)`时

数据包多次接收

```
                    else if ((recvData.flag > 0) & hasseqnum(recvData,
expectedseqnum) & (!corrupt(&recvData)))
                    {   //是所期望的包，且包没有损坏

                        // 文件分段接收
                        cout << "pkg " << recvData.seqNum << " not bad!" <<
endl;
                        // 重新开始计时
                        if (getpkgs == 0)
                            t->Start();
                        // 如果没有超过500ms
                        if (!t->TimeOut())
                        {
                            getpkgs++;
                            // 期望值seq++
                            expectedseqnum++;
                            // 如果达到了累计的数据包就发送ack
                            if (getpkgs == multipkgs)
                            {
                                // 重置getpkgs
                                getpkgs = 0;
                                DataPackage sendData;
                                sendData.ackNum = recvData.seqNum;
                                sendData.make_pkt(CLIENT_PORT, CLIENT_PORT, 0,
WINDOWSIZE);
```

```
                                sendData.CheckSum((unsigned short
*)sendData.message);
                                sendData.ackflag = 1;
                                sendData.len = 0;
                                // 当前acknum
                                curACKnum = sendData.ackNum;
                                sendto(socketClient, (char *)(&sendData),
sizeof(DataPackage) + sendData.len, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                            }
                            // cout << "begin write to file: " << recvFile-
>filePath << endl;
                            printf("begin write %d segment ...\n",
recvData.offset);
                            if (recvData.offset < recvData.flag) // 直到所有分段发
完
                            {
                                // 打开文件在文件最后写入
                                outfile->open(recvFile->filePath, ios::out |
ios::binary | ios::app);
                                outfile->write(recvData.message, recvData.len -
1);
                                outfile->close();

                                if (recvData.offset == recvData.flag - 1)
                                {
                                    cout << "write file success!" << endl;
                                    // 发送最后一段的ack
                                    DataPackage sendData;
                                    sendData.ackNum = recvData.seqNum;
                                    sendData.make_pkt(CLIENT_PORT, CLIENT_PORT,
0, WINDOWSIZE);
                                    sendData.CheckSum((unsigned short
*)sendData.message);
                                    sendData.ackflag = 1;
                                    sendData.len = 0;
                                    // 当前acknum
                                    curACKnum = sendData.ackNum;
                                    sendto(socketClient, (char *)(&sendData),
sizeof(DataPackage) + sendData.len, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                                    stage = 10;
                                    runFlag = false;
                                    logout = true;
                                }
                            }
                        }
                        else // 如果超过了500就抛弃掉现在的数据包，重新开始计时
                        {
                            getpkgs = 0;
                            // 发送ACK
                            DataPackage sendData;
                            // 发送上一次的ack
                            sendData.ackNum = curACKnum;
                            sendData.make_pkt(CLIENT_PORT, CLIENT_PORT, 0,
WINDOWSIZE);
                            sendData.CheckSum((unsigned short
*)sendData.message);
```

```
                            sendData.ackflag = 1;
                            sendData.len = 0;
                            sendto(socketClient, (char *)(&sendData),
sizeof(DataPackage) + sendData.len, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                        }
                        stage = 2;
                    }
                    else if ((recvData.flag > 0) & (!hasseqnum(recvData,
expectedseqnum)) & (!corrupt(&recvData)))
                    {
                        // 如果收到的不是对应期望的分组
                        cout << "get " << recvData.ackNum << " not get
expectedseqnum " << expectedseqnum << endl;
                        DataPackage sendData;
                        // 发送上一次的ack
                        sendData.ackNum = curACKnum;
                        sendData.make_pkt(CLIENT_PORT, CLIENT_PORT, 0,
WINDOWSIZE);
                        sendData.CheckSum((unsigned short *)sendData.message);
                        // 记为ack数据包
                        sendData.ackflag = 1;
                        sendData.len = 0;
                        sendto(socketClient, (char *)(&sendData),
sizeof(DataPackage) + sendData.len, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                    }
```

(3) 多文件发送和接收

添加多个文件的发送和接收部分

server端

```
                case 5: // 多个文件传输部分
                {
                    bool multifile = false;
                    cout << "this file has send success!!!\n If you want to
transport other file cin [1] else [0]: \n";
                    cin >> multifile;
                    if (multifile)
                        stage = 2;
                    else
                    {

                        ZeroMemory(buffer, sizeof(buffer));
                        recvSize = recvfrom(sockServer, buffer, BUFFER, 0,
((SOCKADDR *)&addrClient), &length);
                        if ((unsigned char)buffer[0] == DISCONNECT)
                        {
                            cout << "the client disconnet with server...";
                        }
                        stage = 6;
                        break;
                    }
                }
```

client端

```
            case 3:
            {
                cout << "file transport end!" << endl;
                cout << "if you what to get other file input [1] else [0]\n";
                bool multiRev = 0;
                cin >> multiRev;
                // 如果是多次接收
                if (multiRev)
                    stage = 1;
                else
                {// 关闭连接
                    buffer[0] = DISCONNECT;
                    buffer[1] = '\0';
                    sendto(socketClient, buffer, 2, 0, (SOCKADDR *)&addrServer,
sizeof(SOCKADDR));
                    runFlag = false;
                    logout = true;
                }
                break;
            }
```

三、代码其他细节

（1）基本类和定义

```
#define BUFFER 1024    // int类型 //缓冲区大小（以太网中 UDP 的数据帧中包长度应小于 1480
字节）
#define WINDOWSIZE 10 //滑动窗口大小为 10，当改为1时即为停等协议
#define TIMEOUT 5     // 超时
#define S1 1 // 不同的状态
#define S2 2
#define S3 3
#define S4 4
#define S5 5
#define DISCONNECT 0x001
```

1、文件类

```
// 文件处理
class File
{
public:
    string filePath; // 文件的路径
    int fileLen;     // 文件大小 bits
    int packageSum;  // 需要分多少次发
    bool read; // 是否读取文件
    int fileLenRemain; //文件最后一次发送的长度
    // 初始化文件
    void initFile(bool read, string fp)
    {
        this->read = read;
```

```
        this->filePath = fp;
    }
    void WriteFile()
    {
        if (!this->read)
        {
            ofstream writeos(this->filePath, std::ios::out | std::ios::binary);
            if (!writeos.is_open())
            {
                printf("file open fail ! ! ! \n");
                exit(1);
            }
        }
    }
    void ReadFile()
    {
        if (this->read)
        {
            ifstream readis(this->filePath, ifstream::in | ios::binary); //以二进
制方式打开文件
            // 如果文件没有打开
            cout << this->filePath << endl;
            if (!readis.is_open())
                cout << "this file can't be opened!" << endl;

            streampos pos = readis.tellg(); // 获得文件开始的位置
            readis.seekg(0, ios::end);      //将文件流指针定位到流的末尾
            this->fileLen = readis.tellg(); // 文件的总长度
            readis.seekg(pos);              //将文件流指针重新定位到流的开始
            this->fileLenRemain = this->fileLen % (BUFFER - sizeof(DataPackage)
- 1);
            if (this->fileLenRemain)
                this->packageSum = int(this->fileLen / (BUFFER -
sizeof(DataPackage) - 1)) + 1;
            else
            {
                this->packageSum = int(this->fileLen / (BUFFER -
sizeof(DataPackage) - 1));
            }
            cout << "file pk: " << this->packageSum << endl;
        }
    }
};
```

2、计时器

```
// 计时操作
class Timer
{
public:
    clock_t start_time;
    bool is_stop;  //是否处于停止状态
    double timeout;

public:
    Timer();
    //计时器的三种动作（功能）
```

```
    void Start();
    double GetTime() { return (double)(clock() - this->start_time) /
CLOCKS_PER_SEC; }
    void Show();
    // 判断是否超时
    bool TimeOut()
    {
        if (this->GetTime() >= this->timeout)
            return true;
        else
            return false;
    }
};

Timer::Timer()
{
    //初始化计时器状态
    this->is_stop = true;
    // 初始化超时间隔
    this->timeout = TIMEOUT;
}

void Timer::Start() //开始
{
    this->start_time = clock();
    this->is_stop = false;
}

void Timer::Show()
{
    cout<<"====================="<<endl;
    cout << "t: " << this->GetTime() << "s" << endl;
    cout<<"====================="<<endl;

}
```

3、`extract_pkt`

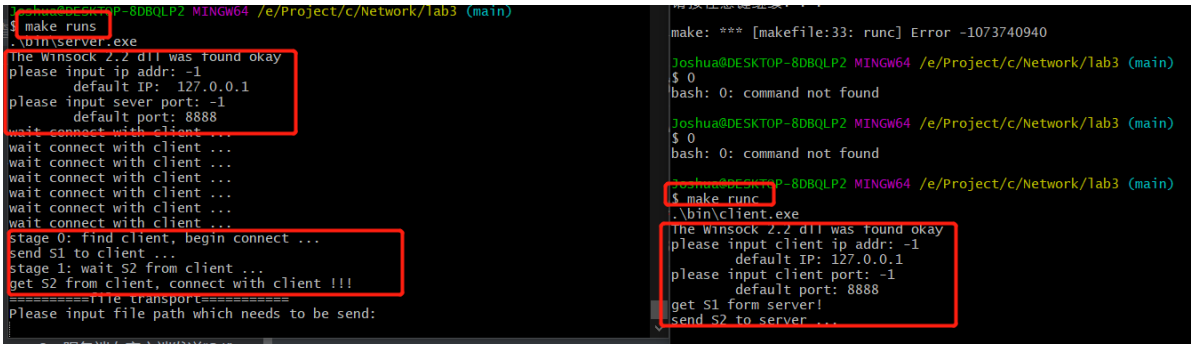
```
// 将char*转为DataPackage
void extract_pkt(char *message, DataPackage &resultdata)
{
    resultdata.sourcePort = *((unsigned short *)&(message[0]));
    resultdata.destPort = *((unsigned short *)&(message[2]));
    resultdata.window = *((unsigned short *)&(message[4]));
    resultdata.checkSum = *((unsigned short *)&(message[6]));
    resultdata.len = *((unsigned short *)&(message[8]));
    resultdata.flag = *((unsigned short *)&(message[10]));
    resultdata.offset = *((unsigned short *)&(message[12]));
    resultdata.timeLive = *((unsigned short *)&(message[14]));
    resultdata.seqNum = *((unsigned int *)&(message[16]));
    resultdata.ackNum = *((unsigned int *)&(message[20]));
    resultdata.ackflag = *((bool *)&(message[24]));
    Strcpyn(resultdata.message, (char *)&(message[25]), resultdata.len);
}
```

4、IfTimeout(Timer &t)

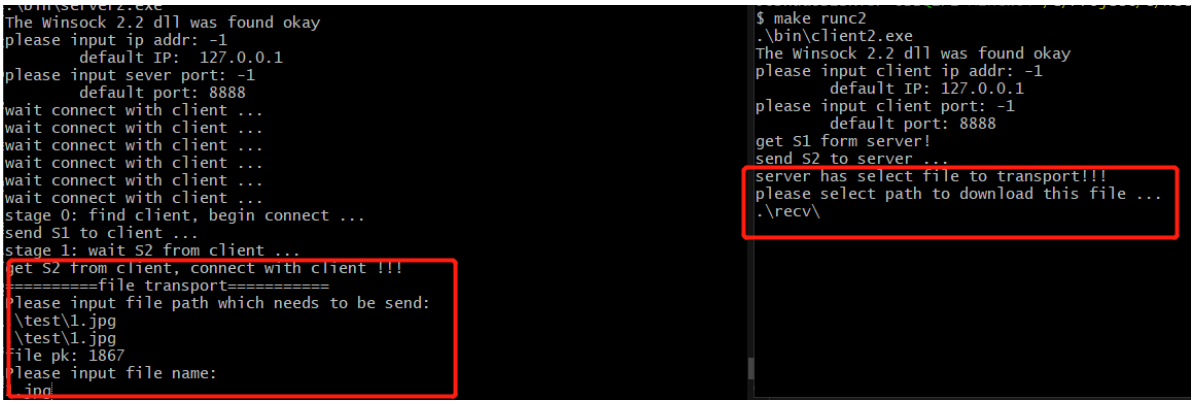
```
// 超时重传处理函数 重传现在缓冲区中所有的数据包
void IfTimeout(Timer &t)
{
    if (t.TimeOut())
    {
        printf("Timer out error.\n");
        t.Start();
        for (int i = 0; i < WINDOWSIZE; ++i)
        {
            DataPackage *tmp = (DataPackage *)malloc(sizeof(DataPackage) +
(BUFFER - sizeof(DataPackage)) * sizeof(char));;
            extract_pkt(sndpkt[i], *tmp);
            sendto(sockServer, (char*)tmp, sizeof(DataPackage) + tmp->len, 0,
(SOCKADDR *)&addrClient, sizeof(SOCKADDR));
            delete tmp;
        }
    }
}
```

（2）过程截图

1、建立连接



2、发送文件



```
something error                                    get 0 not get expectedseqnum 1845
something error                                    get 0 not get expectedseqnum 1845
something error                                    get 0 not get expectedseqnum 1845
something error                                    get 0 not get expectedseqnum 1845
something error                                    get 0 not get expectedseqnum 1845
something error                                    pkg 1845 not bad!
something error                                    begin write 1845 segment ...
something error                                    pkg 1846 not bad!
ack success!!! ACK: 1824                           begin write 1846 segment ...
Timer out error.                                   pkg 1847 not bad!
ack success!!! ACK: 1829                           begin write 1847 segment ...
Timer out error.                                   pkg 1848 not bad!
ack success!!! ACK: 1834                           begin write 1848 segment ...
Timer out error.                                   pkg 1849 not bad!
ack success!!! ACK: 1839                           begin write 1849 segment ...
Timer out error.                                   get 0 not get expectedseqnum 1850
ack success!!! ACK: 1844                           get 0 not get expectedseqnum 1850
something error                                    get 0 not get expectedseqnum 1850
something error                                    get 0 not get expectedseqnum 1850
something error                                    get 0 not get expectedseqnum 1850
something error                                    get 0 not get expectedseqnum 1850
something error                                    get 0 not get expectedseqnum 1850
something error                                    get 0 not get expectedseqnum 1850
something error                                    get 0 not get expectedseqnum 1850
```

3、发送完毕

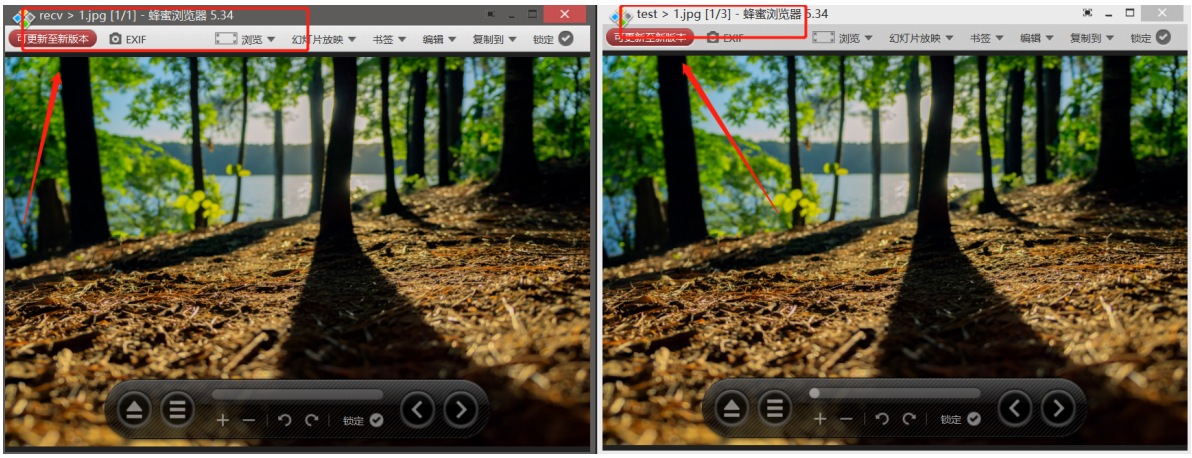
```
something error                                    begin write 1857 segment ...
something error                                    pkg 1858 not bad!
something error                                    begin write 1858 segment ...
something error                                    pkg 1859 not bad!
something error                                    begin write 1859 segment ...
something error                                    pkg 1860 not bad!
something error                                    begin write 1860 segment ...
something error                                    pkg 1861 not bad!
something error                                    begin write 1861 segment ...
something error                                    pkg 1862 not bad!
something error                                    begin write 1862 segment ...
ack success!!! ACK: 1859                           pkg 1863 not bad!
Timer out error.                                   begin write 1863 segment ...
ack success!!! ACK: 1864                           pkg 1864 not bad!
Timer out error.                                   begin write 1864 segment ...
ack success!!! ACK: 1866                           pkg 1865 not bad!
====================                               begin write 1865 segment ...
t: 118.543s                                        pkg 1866 not bad!
====================                               begin write 1866 segment ...
this file has send success!!!                      write file success!
 If you want to transport other file cin [1] else [0]: 请按任意键继续. . .
```

4、发送结果

```
Joshua@DESKTOP-8DBQLP2 MINGW64 /e/Project/c/Network/lab3 (main)
$ diff.exe ./test/helloworld.txt ./recv/helloworld.txt

Joshua@DESKTOP-8DBQLP2 MINGW64 /e/Project/c/Network/lab3 (main)
$
```

没有错误



(3) 结果

文件	发送时间/s
helloworld.txt	104.566s
1.jpg	118.543s
2.jpg	365.045s
3.jpg	745.763s

问题

1、undefined reference to `__imp_WSAStartup'

编译的时候一定要加-lwsock32,不然就会出现undefined reference to `__imp_WSAStartup'等等等等的错误

2、打印时候的乱码问题

setlocale(LC_CTYPE, "");
[https://blog.csdn.net/xie_star/article/details/32162567](

参考文献

https://blog.csdn.net/xie_star/article/details/32162567)