实验3-2 基于UDP服务设计可靠传输协议并编程实现

实验要求

实验3-1，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。

实验3-2，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，采用固定窗口大小，支持累积确认，完成给定测试文件的传输。

报告说明

本次实验报告说明**报文格式**，**交互流程**，**累计确认和超时重传**，**差错重传**，**指定文件**写入缓冲区，发送文件报文，**滑动窗口**，建立连接，断开连接方面的内容。

报文格式

本次实验在上次的基础上，适当更改了发送序号seq的宽度，将4字节的seq变成2字节seq和2字节ack，但报文长度仍然不变，报文头也不变，报文共0x4000字节

	0	4	8	12	16	20	24	28
0	ACK		SYN		FIN		SF	EF
4	源IP							
8	目的IP							
12	源端口				目的端口			
16	发送序号seq				确认序号ack			
20	报文长度filelength							
24	index				校验和checksum			
28	⋮				⋮			
⋮	⋮				⋮			
⋮	⋮				⋮			
⋮	⋮				⋮			

```
#define SEND_MAX 0x4000
#define SEND_TOP 0x1C
char sendBuf[SEND_MAX];
char recvBuf[SEND_MAX];
//五个flag标志位
sendBuf[0]=1;//ACK确认字符
sendBuf[1]=1;//SYN建立连接
sendBuf[2]=1;//FIN断开连接
sendBuf[3]=0x10//SF开始发送文件
sendBuf[3]=0x1//EF结束发送文件
```

交互流程

本次实验选择实现GBN，即实现重点为累计确认，超时重传，滑动窗口等。

首先client发送SYN报文和server建立连接，server回复ACK报文确认连接。

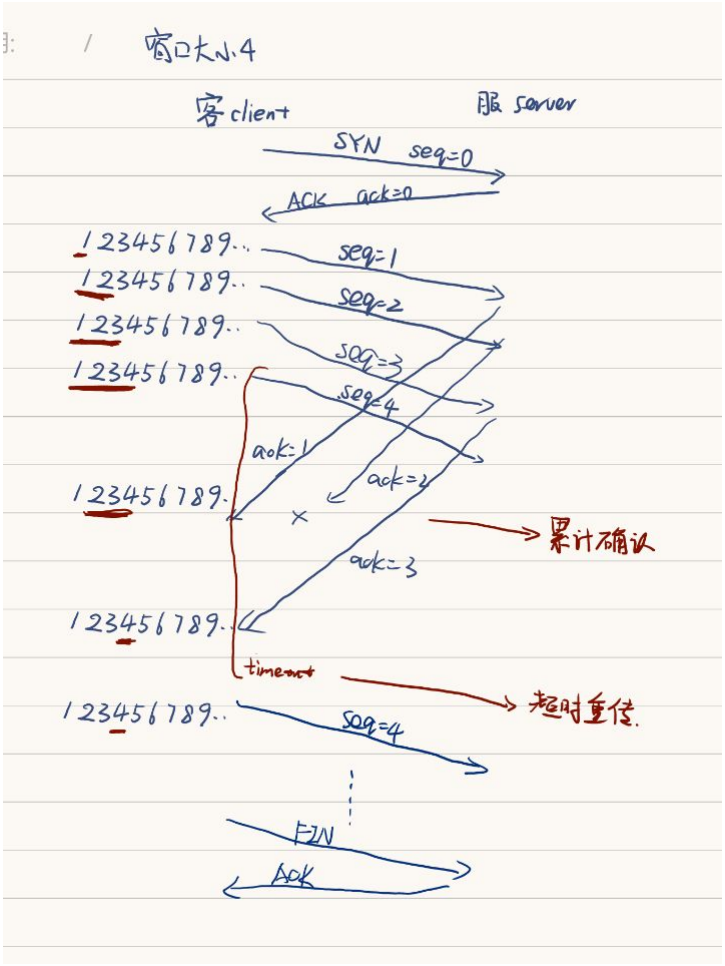
开始出传输文件，正常情况下seq随着报文个数升高而升高，server回复ACK，并且发送的ack=收到相应的seq，表示已经收到，窗口根据ack按照相应的大小向前滑动。使用队列记录每个已经发出并且没有收到回复报文ack的发送，判断队列中第一个记录时间是否超时，超时则清空队列，并从第一个开始重传。

最后，client发送FIN报文表示结束传输，server回复ACK报文，二者成功断开连接。

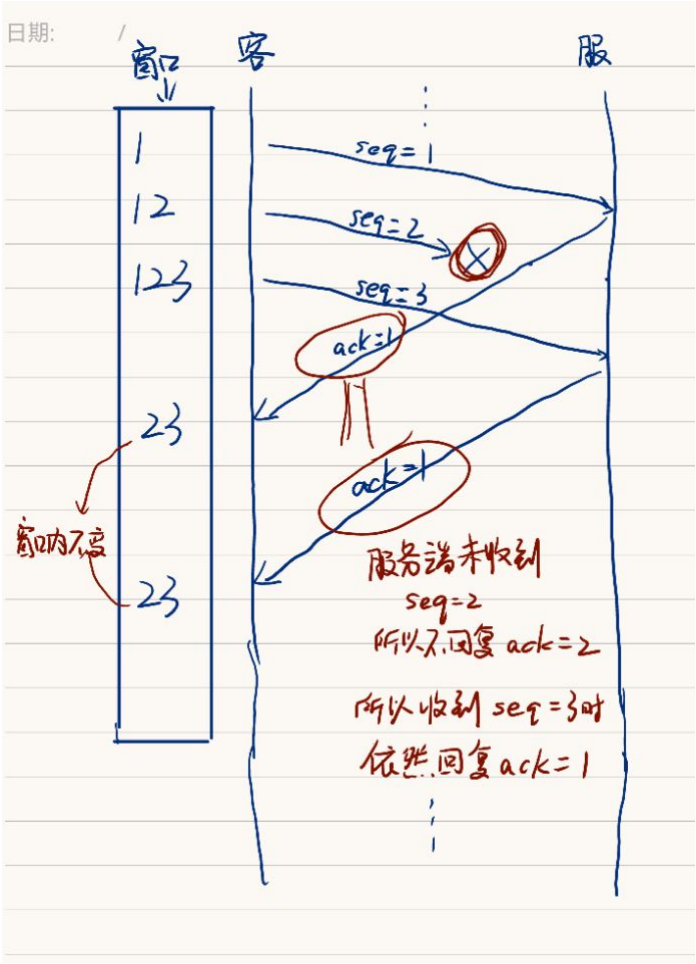
累计确认和超时重传

具体代码在下面的**发送文件报文、滑动窗口**有详细介绍。

下图是正常情况下窗口变化，如果出现某个ack丢失，但收到后续ack，窗口会直接滑动到后续收到ack的地方。下图中的超时重传是指当窗口首部等的时间过长，超过了规定的等待时间，将窗口中的内容清空后，重传报文。



下图是文件报文发送丢包情况，服务端没有收到想要的报文，回复上一次ack。窗口大小一直不变，根据记录的时间，可以判定超时重传。



差错重传

发送数据时计算校验和。

- 将校验和字段置为0
- 对除校验和位置外报文头部按2字节进行反码求和，这里就是对前26字节按2字节反码求和
    - 对两组数据进行求和时，将对应位置相加，如果出现进位则正常进位，如果出现最高位进位，则给结果加1。
    - 所有数据计算完成后按位取反就是校验和了
- 将计算所得校验和填入两个字节的校验和位置

```
u_short calchecksum;//计算校验和并填入
void calculatechecksum()
{
    int sum = 0;
    //计算前26字节
    for (int i = 0; i < 13; i += 2)
    {
        sum += (sendBuf[2 * i] << 8) + sendBuf[2 * i + 1];
        //出现进位
        if (sum >= 0x10000)
        {
            sum -= 0x10000;
            sum += 1;
        }
    }
    //checksum计算
    //取反
    calchecksum = ~(u_short)sum;
```

```
    sendBuf[26] = (char)(calchecksum >> 8);
    sendBuf[27] = (char)calchecksum;
}
```

收到报文检验校验和。

- 首先还是对前26字节按照2字节进行反码求和
- 取出报文中提供的校验和，和计算所得相加
  - 若结果为0xFFFF，在则校验成功
  - 否则校验失败

```
u_short checksum;
bool verifychecksum()
{
    int sum = 0;
    //计算前26字节
    for (int i = 0; i < 13; i += 2)
    {
        sum += (recvBuf[2 * i] << 8) + recvBuf[2 * i + 1];
        //出现进位
        if (sum >= 0x10000)
        {
            sum -= 0x10000;
            sum += 1;
        }
    }
    //checksum计算
    checksum = (recvBuf[26] << 8) + (unsigned char)recvBuf[27];
    if (checksum + (u_short)sum == 0xffff)return true;
    else return false;
}
```

client端和server端都对收到的报文进行检验，如果检验失败则发送上一个报文给对方，要求对方重新发送报文。这里也是以后面的建立连接为例，详细讲述重传代码。

给指定的文件传输

读取指定文件夹内的文件

这里就是提供的测试文件，输入想要传输的文件序号。将指定文件按二进制数据流存入全局二维数组content[10000][16356]，同时计算需要的数组行数，并记录下结束时最后一行数组的大小，即最后一个报文的长度。

```
void log()
{
    //下面开始检测文件夹下的测试文件
    cout << "下面开始检测任务1测试文件" << endl;
    vector<string> files;
    string path("C:\\web\\task_one_test_file");
    if (getFiles(path, files))
        cout << "读文件成功" << endl;
    int num = files.size();
    for (int i = 0; i < num; i++)cout << files[i].c_str() << endl;
    int choose = 0;
    cout << "输入文件序号" << endl;
    cin >> choose;
```

```
    ifstream fin(files[choose].c_str(),ifstream::binary);
    char t = fin.get();
    int length = 0;
    while (fin)
    {
        content[sum][length % 16356] = t;
        length++;
        if (length % 16356 == 0)
        {
            sum++;
            length = 0;
        }
        t = fin.get();
    }
    endfilesize = length;
    cout << "文件报文个数" << sum + 1 << endl;
    cout << "最后一个文件报文大小" << endfilesize << endl;
}
```

写入报文

设置每个报文的index，代表这是第几个文件报文。

第一个传输文件报文SF，中间报文的SF和EF为都为0，并对除了最后一个报文长度置16356(0x3FE4)。

最后一个传输文件报文EF，将上面算得的最后一行数组长度传到最后一个文件报文的filelength。

设置报文的seq，计算校验和并填入。

client端：

want_to_sendfile()

```
void want_to_sendfile()
{
    setflag();
    //对每一个都设置index
    sendBuf[24] = (char)(index >> 8);
    sendBuf[25] = (char)index;

    //将filelength所在初始化为0
    for (int i = 20; i < 24; i++) sendBuf[i] = 0;

    //最后一个设置filelength还有多少字节
    if (index == sum)
    {
        sendBuf[20] = (char)(endfilesize >> 24);
        sendBuf[21] = (char)(endfilesize >> 16);
        sendBuf[22] = (char)(endfilesize >> 8);
        sendBuf[23] = (char)endfilesize;
        sendBuf[3] = 0x1;
    }
    else
    {
        if (index == 0)
        {
            sendBuf[3] = 0x10;
        }
        else
```

```
        {
            sendBuf[3] = 0;
        }
        sendBuf[20] = (char)(16356 << 24);
        sendBuf[21] = (char)(16356 << 16);
        sendBuf[22] = (char)(16356 << 8);
        sendBuf[23] = (char)16356;
    }
    setseq();
    calculatechecksum();
}
```

发送文件报文、滑动窗口

发送文件报文，在发送报文前设置计时器，并记录下结束后的时间。

本次实验采用多线程接收发送，主线程为发送，子线程为接收。使用队列记录窗口，内容包括<报文序号，发送时间>。

主线程：

循环判断

- 判断是否成功完的个数，发完退出
- 判断窗口是否已满，是否已经发送全部的报文，没满并且没发送全部的则继续发送，并且在窗口队列里加入这个seq和发送时间，如果满了或者已经发送了所有的，就直接退出运行后面的内容或者运行子线程
- 判断发送时间是否超过规定时间，窗口清空，重新发送

子线程：

判断是否接收到消息，如果接收到消息，判断ack是否在窗口内，如果大于等于窗口最前端，窗口移动到ack+1。每次接收到消息时输出窗口变化。

client端：

主线程判断发送

```
int t_start = clock();
while (1)
{
    //判断是否成功发送所有报文
    WaitForSingleObject(hMutex, INFINITE);
    if (base == seq + sum+1)break;
    ReleaseMutex(hMutex);
    //判断窗口是否装满，是否在没装满的情况下已经发送（has_send-base<WINDOW_SIZE）
    WaitForSingleObject(hMutex, INFINITE);
    while (1)
    {
        if (((has_send - base) < WINDOW_SIZE) && has_send < seq + sum + 1)
        {
            want_to_sendfile(++has_send);
            sendto(sockSrv, sendBuf, sizeof(sendBuf),0,
(SOCKADDR*)&addrServer, len);
            timer_list.push(make_pair(has_send, clock()));
        }
        else break;
    }
    ReleaseMutex(hMutex);
```

```
    //判断发送时间
    WaitForSingleObject(hMutex, INFINITE);
    if (timer_list.size() != 0)
    {
        if ((clock() - timer_list.front().second) > TIMEOUT)
        {
            has_send = base - 1;
            while (timer_list.size()) timer_list.pop();
        }
    }
    ReleaseMutex(hMutex);
}
int t_end = clock();
```

子线程判断接收

```
HANDLE hMutex = NULL;//互斥量
DWORD WINAPI recvthread(LPVOID lpParamter)
{
    SOCKET sockSrv = (SOCKET)(LPVOID)lpParamter;
    while (1)
    {
    if(recvfrom(sockSrv,recvBuf,sizeof(recvBuf),0,
(SOCKADDR*)&addrServer,&len) !=-1
            && recvBuf[0]
            && verifychecksum())
        {
            WaitForSingleObject(hMutex, INFINITE);
            int ack = (((unsigned int)((unsigned char)recvBuf[18]) << 8)) +
                      (((unsigned int)(unsigned char)recvBuf[19]));
            if (ack >= base && ack <= has_send)
            {
                while (timer_list.size()!=0&&(timer_list.front().first <=
ack))
                {
                    timer_list.pop();
                    base++;
                }
            }
            ReleaseMutex(hMutex);
        }
    }
    return 0L;
}
```

server端：

服务端如果接收到文件报文，判断校验和

- 如果校验和正确
  - 计算本次报文的序列号，如果序列号比上次大1，说明接收正确
    - 如果该报文为文件报文开始，即recvBuf[3] == 0x10，将名字写入name，设置ack发送报文。
    - 如果该报文为文件报文结束，即recvBuf[3] == 0x01，将内容写入content后，设置ack发送报文，结束循环。

        - 如果该报文为文件报文内容，即recvBuf[3] == 0x00，根据index，将内容写入相应的content，设置ack发送报文。
    - 如果序列号不正确，则发送上次已经发送过的报文，让client端窗口不能滑动，直到超时重传
- 如果校验和不正确，直接发送ACK=0

```
while (1)
{
    if (recvfrom(sockSrv, recvBuf, sizeof(recvBuf), 0, (SOCKADDR*)&addrClient, &len)!=-1)
    {
        if (verifychecksum())
        {
            thisseq = (((unsigned int)((unsigned char)recvBuf[16]) << 8))
                + (((unsigned int)(unsigned char)recvBuf[17]));
            if (recvseq == thisseq - 1)
            {
                if (recvBuf[3] == 0x10)
                {
                    setname();
                }
                if (recvBuf[3] == 0x00)
                {
                    setcontent(calculateindex());
                }
                if (recvBuf[3] == 0x01)
                {
                    size = calculateindex();
                    setcontentend(size);
                    sendBuf[0] = 1;//ACK=1
                    getseqsetack();
                    calculatechecksum();
                    sendto(sockSrv, sendBuf, sizeof(sendBuf), 0, (SOCKADDR*)&addrClient, len);
                    break;
                }
                sendBuf[0] = 1;//ACK=1
                getseqsetack();
                calculatechecksum();
                recvseq = thisseq;
                sendto(sockSrv, sendBuf, sizeof(sendBuf), 0, (SOCKADDR*)&addrClient, len);
            }
            else
            //如果判断本次收到的报文不是希望得到的下一个报文，即出现丢包的情况，还是发送上一个报文给客户端，直到超时重传
            sendto(sockSrv,sendBuf, sizeof(sendBuf), 0, (SOCKADDR*)&addrClient, len);
        }
        else
        {
            sendBuf[0] = 0;//ack=0
            calculatechecksum();
            sendto(sockSrv, sendBuf, sizeof(sendBuf), 0, (SOCKADDR*)&addrClient, len);
        }
    }
```

```
}
```

setcontent():

```
void setcontent(int index)
{
    for (int i = 0; i < SEND_MAX-SEND_TOP; i++)
    {
        content[index - 1][i] = recvBuf[i + 28];
    }
}
```

setcontentend():

```
void setcontentend(int index)
{
    endfilesize = (((unsigned int)((unsigned char)recvBuf[20]) << 24))
        + (((unsigned int)((unsigned char)recvBuf[21]) << 16))
        + (((unsigned int)((unsigned char)recvBuf[22]) << 8))
        + (((unsigned int)(unsigned char)recvBuf[23]));
    for (int i = 0; i < endfilesize; i++)
    {
        content[index - 1][i] = recvBuf[i + 28];
    }
}
```

建立连接

1. client发送SYN报文给server请求连接
2. server收到报文后首先对校验和进行检验
    - 如果校验和不正确，则设置ACK为0，设置seq为收到seq，计算自己的校验和然后发送给server。
    - 如果校验和正确，则设置ACK为1，其余同上
3. client收到回应报文后判断ACK和校验和，如果ACK=0或校验和不正确，则继续发送SYN报文给server

client端：

want_to_connect():

```
void want_to_connect()
{
    for (int i = 0; i < 4; i++)
        sendBuf[i] = 0;
    //syn设为1其余设置为0
    sendBuf[1] = 1;
    sendBuf[20] = sendBuf[21] = sendBuf[22] = sendBuf[23] = sendBuf[24] = sendBuf[25] = 0;
    sendBuf[16] = (char)(seq >> 8);
    sendBuf[17] = (char)(seq);
    calculatechecksum();
}
```

发送报文建立连接代码：

```
want_to_connect();
while (1)
{
    sendto(sockSrv, sendBuf, sizeof(sendBuf), 0, (SOCKADDR*)&addrServer, len);
    cout << "想要连接" << endl;
    if(recvfrom(sockSrv, recvBuf, sizeof(recvBuf), 0, (SOCKADDR*)&addrServer, &len) != -1)
    {
        if (verifychecksum() && recvBuf[0] == 1)
        {
            cout << "成功连接" << endl;
            seq++;
            break;
        }
    }
}
```

server端：

回应报文建立连接

```
while (1)
{
    if (recvfrom(sockSrv, recvBuf, sizeof(recvBuf), 0, (SOCKADDR*)&addrClient, &len) != -1)
    {
        cout << "接收连接" << endl;
        if (verifychecksum())
        {
            cout << "连接成功" << endl;
            sendBuf[0] = 1;
            getseqsetack();
            calculatechecksum();
            sendto(sockSrv, sendBuf, sizeof(sendBuf), 0, (SOCKADDR*)&addrClient, len);
            break;
        }
        else
        {
            sendBuf[0] = 0;
            getseqsetack();
            calculatechecksum();
            sendto(sockSrv, sendBuf, sizeof(sendBuf), 0, (SOCKADDR*)&addrClient, len);
        }
    }
}
```

断开连接

1. client端发送断开报文给server，设置FIN报文，设置seq，计算校验和并填入
2. server端判断收到的是否为FIN报文
    - 如果是FIN报文并且校验和正确，则发送确认报文给client
    - 否则发送ACK=0

3. client端收到断开回应报文，判断ACK是否为1，并计算校验和是否正确，若满足，则成功断开连接。否则继续发送FIN报文。

client端：

want_to_break()

```
void want_to_break()
{
    for (int i = 0; i < 4; i++)
        sendBuf[i] = 0;
    sendBuf[2] = 1;
    sendBuf[16] = (char)((seq) >> 8);
    sendBuf[17] = (char)(seq);
    calculatechecksum();
}
```

发送请求断开连接：

```
want_to_break();
while (1)
{
    sendto(sockSrv, sendBuf, sizeof(sendBuf), 0, (SOCKADDR*)&addrServer, len);
    cout << "想要断开" << endl;
    if (recvfrom(sockSrv, recvBuf, sizeof(recvBuf), 0, (SOCKADDR*)&addrServer, &len) != -1)
    {
        if (verifychecksum() && recvBuf[0] == 1)
        {
            cout << "成功断开" << endl;
            break;
        }
    }
}
```

server端：

```
//断开连接
while (1)
{
    recvfrom(sockSrv, recvBuf, sizeof(recvBuf), 0, (SOCKADDR*)&addrClient, &len);
    if ((recvBuf[2] == 1)&&(verifychecksum()))
    {
        sendBuf[0] = 1;//ack=1
        get_and_set_seq();
        calculatechecksum();
        sendto(sockSrv, sendBuf, sizeof(sendBuf),0,(SOCKADDR*)&addrClient, len);
        break;
    }
    else
    {
        sendBuf[0] = 0;//ack=0
        get_and_set_seq();
```

```
        calculatechecksum();
        sendto(sockSrv, sendBuf, sizeof(sendBuf),0,(SOCKADDR*)&addrClient,
len);
    }
}
```

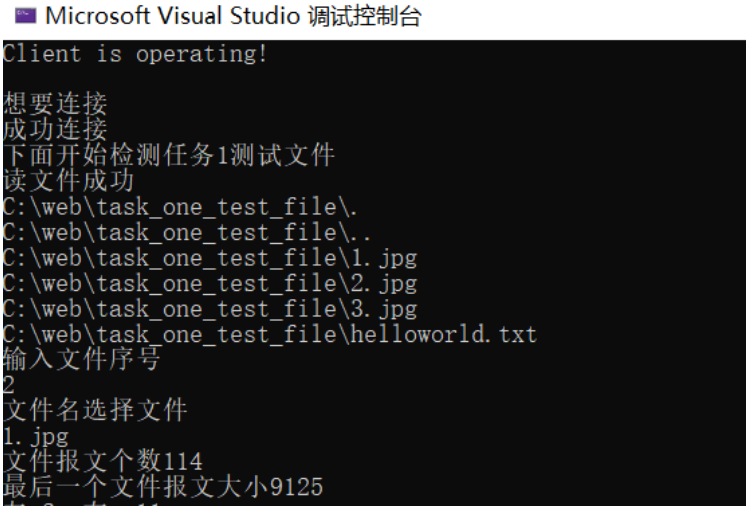
计算平均吞吐率

根据计算得到的文件文件报文个数计算平均吞吐率。

```
cout << "发送" << (sum * 16356 + endfilesize) << "字节" << (t_end - t_start)
<< "毫秒" << endl;
cout << "平均吞吐率" << (sum * 16356 + endfilesize) * 8 * 1.0 / (t_end -
t_start) * CLOCKS_PER_SEC << " bps" << endl;
```

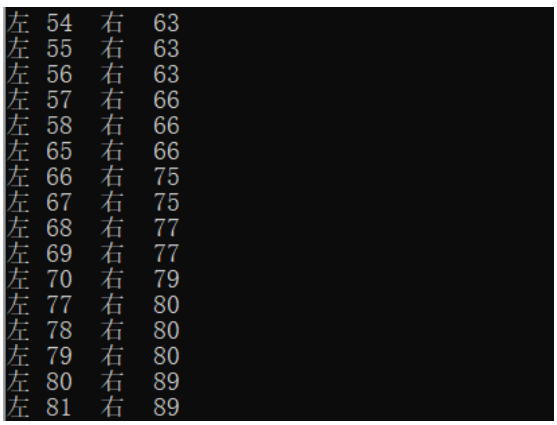
执行界面输出日志

建立连接

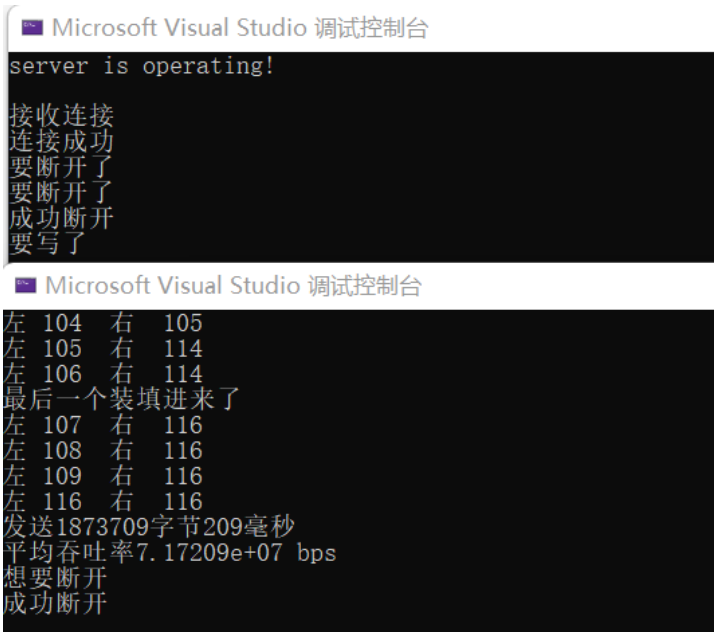


发送文件

client端



断开连接


Microsoft Visual Studio 调试控制台
server is operating!
接收连接
连接成功
要断开了
要断开了
成功断开
要写了
Microsoft Visual Studio 调试控制台
左 104 右 105
左 105 右 114
左 106 右 114
最后一个装填进来了
左 107 右 116
左 108 右 116
左 109 右 116
左 116 右 116
发送1873709字节209毫秒
平均吞吐率7.17209e+07 bps
想要断开
成功断开