实验 3：基于 UDP 服务设计可靠传输协议并编程实现

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

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。

(1)实现单向传输。

(2)对于每一个任务要求给出详细的协议设计。

(3)给出实现的拥塞控制算法的原理说明。

(4)完成给定测试文件的传输，显示传输时间和平均吞吐率。

(5)性能测试指标：吞吐率、时延，给出图形结果并进行分析。

(6)完成详细的实验报告（每个任务完成一份）。

(7)编写的程序应结构清晰，具有较好的可读性。

(8)提交程序源码和实验报告。

协议设计

面向连接

建立连接和断开连接的设计参考了 TCP 协议的三次握手和四次挥手。

在 TCP 协议中，三次握手过程如下：

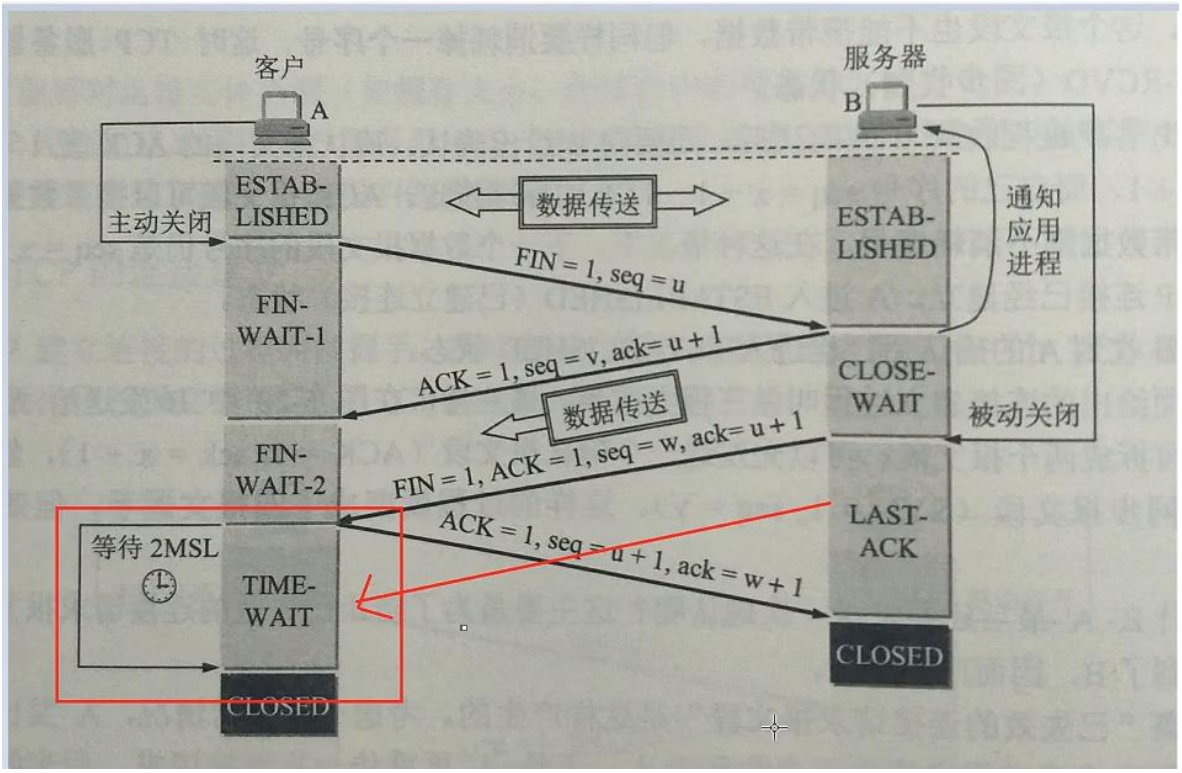
基于此，设计建立连接过程：

① 首先客户端向服务器端发送一个报文，其 SYN 标志位置 1，标志请求建立连接

② 服务器收到请求后，向客户端回复一个报文，SYN 和 ACK 标志位置 1，标志允许建立连接

③ 客户端收到服务器反馈后，向服务器发送一个报文，ACK 置 1，标志可以开始传输

TCP 中四次挥手过程如下：



基于此，设计断开连接过程：

①客户端向服务器端发送一个报文，将 FIN 标志位置 1，标识文件传输完毕请求断开连接

②服务器端收到断开请求后，回应一个报文，将 ACK 标志位置 1，标识接到断开请求

③服务器端向客户端发送一个报文，将 A=FIN 标志位置 1，标识请求断开连接

④客户端收到断开请求后，回应一个报文，将 ACK 标志位置 1，标识接到断开请求。之后客户端在等待两个 MSL 时间后，确保不再接收到服务端的数据包（即服务端已经收到客户端的回应，不会再进行重传操作）后，再关闭。

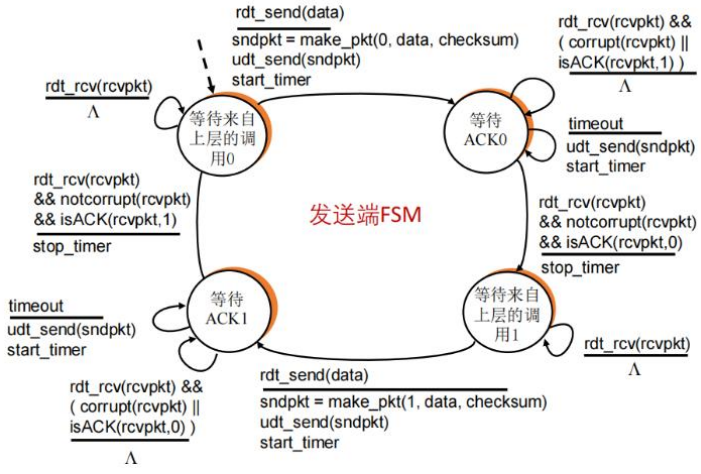
拥塞设置：阻塞模式与非阻塞模式

由于重传机制的实现需要对数据报的发送和接收进行计时，但是初始化 socket 时默认是阻塞态的 socket，调用 recvfrom 函数后线程被阻塞，计时函数也不能正常运行。如果我们在阻塞态调用 recvfrom 那么计时函数就需要新开一个线程，为了避免这种麻烦，我们需要计时重传的阶段调用以下代码，将 socket 切换为非阻塞态。

可靠数据传输

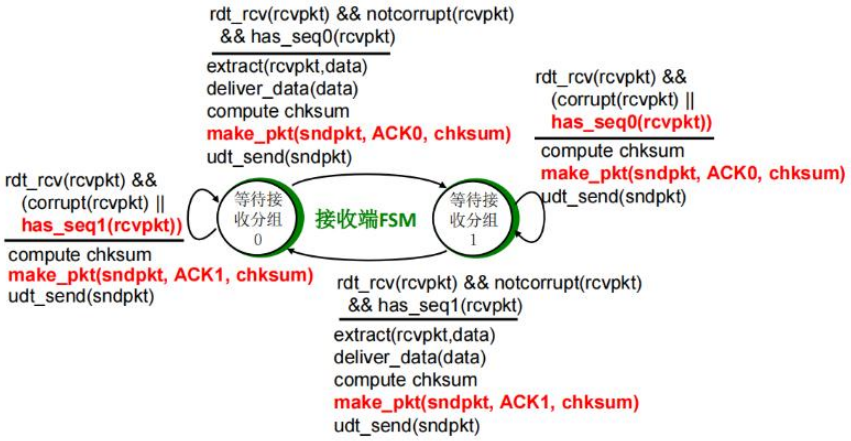
传输协议参考 rdt3.0，并加以简单修改，来保障数据传输过程的可靠性。rdt3.0 实现中，客户端发送 seq 号数据报后，需要等待对 seq 数据报的确认号 ack=seq，并且检查数据报传输时无损，才能继续发送下一个数据报；如果超时未能收到正确无误的数据报，客户端就会重传一次 seq 数据报。服务器则是在等待 seq 数据报，如果等来的不是目标序列号数据报，服务端就会重新发送一次 ack 报文，直到服务端收到期望的序列号数据报。具体实现参考下图。

客户端有限状态机：



由于四个状态分别对应 seq=0 的发送和接收确认、seq=1 的发送和接收确认，所以在实验中，我设置了全局变量 curseq 和 curack，代替 0 和 1，将 4 个状态整合为 2 个状态，分别为：等待来自上层的调用 curseq、等待 ACKcurseq。

服务器有限状态机：



同理，我设置全局变量 curseq 和 curack，将两个状态整合为一个状态：等待接收分组 curseq。

确认应答

参考 TCP 的 seq/ack 机制，在对 UDP 封装是增加 seq 和 ack 字段，以实现确认应答

①服务器端给客户端发送 seq=J 的报文

②客户端收到后确认，发送 ack=J+1.为期待接收的下一报文段

在本次实验中，采用超时重传机制，seq/ack 只需要 0 和 1 两个值。

超时重传

服务器端每发送一个报文时，启动一个计时器，当超时时，重发该数据报。

差错检测

模仿 tcp，发送方发送报文前先计算 checksum 并封装到包内，接收方收到包进行校验，如果正确则正确接收。

功能实现

报文结构定义和一些宏定义

报文结构

```
struct packetHead {
    u_int seq;
    u_int ack;
    u_short checkSum;
    u_short bufSize;
    char flag;

    packetHead() {
        seq = ack = 0;
        checkSum = bufSize = 0;
        flag = 0;
    }
};

struct packet {
    packetHead head;
    char data[MAX_DATA_SIZE];
};
```

宏定义

```
#define SYN 0x1
#define ACK 0x2
#define FIN 0x4
#define END 0x8
#define PORT 7878
#define ADDRSRV "127.0.0.1"
#define MAX_DATA_SIZE 2048

double MAX_TIME = CLOCKS_PER_SEC;
int seqsize = 2;
int curseq = 0;
int curack = 1;
```

计算校验和

利用 u_short 类型为 8 位，将 temp 的位数声明为 16 的倍数，并用 0 填充，然后将 packet 的数据填充到 temp 中去，从而实现补 0 到 16 位的倍数。

```
u_short checkPacketSum(u_short* packet, int packetLen) {

    u_long sum = 0;
    int count = (packetLen + 1) / 2;

    u_short* temp = new u_short[count];
    memset(temp, 0, 2 * count);
    memcpy(temp, packet, packetLen);

    while (count--) {
        sum += *temp++;
        if (sum & 0xFFFF0000) {
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

主函数

服务端：

```
int main() {
    WSAData wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        //加载失败
        cout << "加载DLL失败" << endl;
        return -1;
    }
    SOCKET sockSrv = socket(AF_INET, SOCK_DGRAM, 0);

    SOCKADDR_IN addrSrv;
    addrSrv.sin_family = AF_INET;
    addrSrv.sin_port = htons(PORT);
    addrSrv.sin_addr.S_un.S_addr = inet_addr(ADDRSRV);
    bind(sockSrv, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR));

    SOCKADDR_IN addrClient;

    //三次握手建立连接
    if (!acceptClient(sockSrv, addrClient)) {
        cout << "连接失败" << endl;
        return 0;
    }

    char* filename = new char;
    u_long r = recvFSM(filename, sockSrv, addrSrv);
    if (r > 0) {
        printf("文件路径为:%s\n", filename);
    }
    else {
        cout << "没有接收到文件路径" << endl;
    }
```

```
    //char fileBuffer[MAX_FILE_SIZE];
    //可靠数据传输过程
    u_long fileLen = recvFSM(fileBuffer, sockSrv, addrClient);
    //四次挥手断开连接
    if (!disConnect(sockSrv, addrClient)) {
        cout << "断开失败" << endl;
        return 0;
    }

    //写入复制文件
    ofstream outfile(filename, ofstream::binary);
    if (!outfile.is_open()) {
        cout << "打开文件出错" << endl;
        return 0;
    }
    //cout << fileLen << endl;
    outfile.write(fileBuffer, fileLen);
    outfile.close();

    cout << "文件复制完毕" << endl;
    return 1;
}
```

客户端：

```
int main() {
    WSAData wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        //加载失败
        cout << "加载DLL失败" << endl;
        return -1;
    }
    SOCKET sockClient = socket(AF_INET, SOCK_DGRAM, 0);

    u_long imode = 1;
    ioctlsocket(sockClient, FIONBIO, &imode);//非阻塞

    SOCKADDR_IN addrSrv;
    addrSrv.sin_family = AF_INET;
    addrSrv.sin_port = htons(PORT);
    addrSrv.sin_addr.S_un.S_addr = inet_addr(ADDRSRV);

    if (!connectToServer(sockClient, addrSrv)) {
        cout << "连接失败" << endl;
        return 0;
    }

    string filename;
    cout << "请输入需要传输的文件名" << endl;
    cin >> filename;

    sendFSM(filename.length(), (char *)filename.c_str(), sockClient, addrSrv);

    ifstream infile(filename, ifstream::binary);

    if (!infile.is_open()) {
        cout << "无法打开文件" << endl;
        return 0;
    }
```

```
    infile.seekg(0, infile.end);
    u_long fileLen = infile.tellg();
    infile.seekg(0, infile.beg);
    //cout << fileLen << endl;

    char* fileBuffer = new char[fileLen];
    infile.read(fileBuffer, fileLen);
    infile.close();
    //cout.write(fileBuffer,fileLen);
    cout << "开始传输" << endl;

    clock_t start = clock();
    sendFSM(fileLen, fileBuffer, sockClient, addrSrv);
    clock_t end = clock();
    cout << "传输总时间为:" << (end - start) / CLOCKS_PER_SEC << "s" << endl;
    cout << "吞吐率为:" << ((float)fileLen) / ((end - start) / CLOCKS_PER_SEC) << "byte/s" << endl;

    if (!disConnect(sockClient, addrSrv)) {
        cout << "断开失败" << endl;
        return 0;
    }
    cout << "文件传输完成" << endl;
    return 1;
}
```

建立连接

服务端：

```
bool acceptClient(SOCKET& socket, SOCKADDR_IN& addr) {

    char* buffer = new char[sizeof(packetHead)];
    int len = sizeof(addr);
    recvfrom(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, &len);

    if ((((packetHead*)buffer)->flag & SYN) && (checkPacketSum((u_short*)buffer, sizeof(packetHead)) == 0))
        cout << "第一次握手成功" << endl;
    else {
        cout << "不是第一次握手的数据包" << endl;
        return false;
    }

    packetHead head;
    head.flag |= ACK;
    head.flag |= SYN;
    head.checkSum = checkPacketSum((u_short*)&head, sizeof(packetHead));
    memcpy(buffer, &head, sizeof(packetHead));
    if (sendto(socket, buffer, sizeof(packetHead), 0, (sockaddr*)&addr, len) == -1) {
        cout << "第二次握手数据包发送失败" << endl;
        return false;
    }
    cout << "第二次握手成功" << endl;//[SYN_ACK_SEND]
```

```
    u_long mode = 1;//mode=0为阻塞，mode=1为非阻塞
    ioctlsocket(socket, FIONBIO, &mode);//非阻塞

    clock_t start = clock(); //开始计时
    while (recvfrom(socket, buffer, sizeof(head), 0, (sockaddr*)&addr, &len) <= 0) {
        if (clock() - start >= MAX_TIME) {
            cout << "未接收到第三次握手信息，超时重传" << endl;
            sendto(socket, buffer, sizeof(buffer), 0, (sockaddr*)&addr, len);
            start = clock();
        }
    }

    if ((((packetHead*)buffer)->flag & ACK) && (checkPacketSum((u_short*)buffer, sizeof(packetHead)) == 0)) {
        cout << "第三次握手成功" << endl;//[ACK_RECV]
    }
    else {
        return false;
    }
    mode = 0;
    ioctlsocket(socket, FIONBIO, &mode);//阻塞

    cout << "与用户端成功建立连接，准备接收文件" << endl;//[CONNECTED]
    return true;
}
```

客户端：

```cpp
bool connectToServer(SOCKET& socket, SOCKADDR_IN& addr) {
    int len = sizeof(addr);

    packetHead head;
    head.flag |= SYN;
    head.seq = 0;
    head.checkSum = checkPacketSum((u_short*)&head, sizeof(head));

    char* buffer = new char[sizeof(head)];
    memcpy(buffer, &head, sizeof(head));
    sendto(socket, buffer, sizeof(head), 0, (sockaddr*)&addr, len);
    cout << "第一次握手成功" << endl;

    clock_t start = clock(); //开始计时
    while (recvfrom(socket, buffer, sizeof(head), 0, (sockaddr*)&addr, &len) <= 0) {
        if (clock() - start >= MAX_TIME) {
            memcpy(buffer, &head, sizeof(head));
            sendto(socket, buffer, sizeof(buffer), 0, (sockaddr*)&addr, len);
            start = clock();
        }
    }

    memcpy(&head, buffer, sizeof(head));
    if ((head.flag & ACK) && (checkPacketSum((u_short*)&head, sizeof(head)) == 0)) {
        cout << "第二次握手成功" << endl;
    }
    else {
        return false;
    }
```

```cpp
    //服务器建立连接
    if (head.flag & SYN) {
        head.flag = 0;
        head.flag |= ACK;
        head.checkSum = 0;
        head.checkSum = (checkPacketSum((u_short*)&head, sizeof(head)));
    }
    else {
        return false;
    }
    memcpy(buffer, &head, sizeof(head));
    sendto(socket, buffer, sizeof(head), 0, (sockaddr*)&addr, len);

    //等待两个MAX_TIME，如果收到消息说明ACK没有丢包
    start = clock();
    while (clock() - start <= 2 * MAX_TIME) {
        if (recvfrom(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, &len) <= 0)
            continue;
        //说明这个ACK丢了
        memcpy(buffer, &head, sizeof(head));
        sendto(socket, buffer, sizeof(head), 0, (sockaddr*)&addr, len);
        start = clock();
    }
    cout << "三次握手成功" << endl;
    cout << "成功与服务器建立连接，准备发送数据" << endl;
    return true;
}
```

客户端有限状态机

```
void sendFSM(u_long len, char* fileBuffer, SOCKET& socket, SOCKADDR_IN& addr) {

    int packetNum = int(len / MAX_DATA_SIZE) + (len % MAX_DATA_SIZE ? 1 : 0);
    int index = 0;
    int packetDataLen = min(MAX_DATA_SIZE, len - index * MAX_DATA_SIZE);
    int stage = 0;

    int addrLen = sizeof(addr);
    clock_t start;

    char* data_buffer = new char[packetDataLen], * pkt_buffer = new char[sizeof(packet)];
    packet sendPkt, pkt;

    cout << "本次文件数据长度为" << len << "Bytes,需要传输" << packetNum << "个数据包" << endl;
```

```
    while (true) {
        if (index == packetNum) {
            packetHead endPacket;
            endPacket.flag |= END;
            endPacket.checkSum = checkPacketSum((u_short*)&endPacket, sizeof(packetHead));
            memcpy(pkt_buffer, &endPacket, sizeof(packetHead));
            sendto(socket, pkt_buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, addrLen);

            while (recvfrom(socket, pkt_buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, &addrLen) <= 0) {
                if (clock() - start >= MAX_TIME) {
                    memcpy(pkt_buffer, &endPacket, sizeof(packetHead));
                    sendto(socket, pkt_buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, addrLen);
                    start = clock();
                }
            }

            if (((packetHead*)(pkt_buffer))->flag & ACK) {
                cout << "文件传输完成" << endl;
            }

            return;
        }
        packetDataLen = min(MAX_DATA_SIZE, len - index * MAX_DATA_SIZE);
```

```
        switch (stage) {
        case 0:
            memcpy(data_buffer, fileBuffer + index * MAX_DATA_SIZE, packetDataLen);
            sendPkt = makePacket(curseq, data_buffer, packetDataLen);

            memcpy(pkt_buffer, &sendPkt, sizeof(packet));
            sendto(socket, pkt_buffer, sizeof(packet), 0, (SOCKADDR*)&addr, addrLen);

            start = clock();//计时
            stage = 1;
            curseq = (curseq + 1) % seqsize;
            break;
        case 1:
            //time_out
            while (recvfrom(socket, pkt_buffer, sizeof(packet), 0, (SOCKADDR*)&addr, &addrLen) <= 0) {
                if (clock() - start >= MAX_TIME) {
                    sendto(socket, pkt_buffer, sizeof(packet), 0, (SOCKADDR*)&addr, addrLen);
                    cout << "第" << index << "号数据包超时重传" << endl;
                    start = clock();
                }
            }

            memcpy(&pkt, pkt_buffer, sizeof(packet));
            if (pkt.head.ack == curack || checkPacketSum((u_short*)&pkt, sizeof(packet)) != 0) {
                stage = 1;
                break;
            }
            //cout << "成功发送第" << index << "号数据包" << endl;
            stage = 0;
            curack = (curack + 1) % seqsize;
            index++;
            break;

        default:
            cout << "error" << endl;
            return;
        }
    }
```

服务端有限状态机

```
u_long recvFSM(char* fileBuffer, SOCKET& socket, SOCKADDR_IN& addr) {
    u_long fileLen = 0;

    int addrLen = sizeof(addr);
    char* pkt_buffer = new char[sizeof(packet)];
    packet pkt, sendPkt;
    int index = 0;
    int dataLen;
    while (true) {
        memset(pkt_buffer, '0', sizeof(packet));
        recvfrom(socket, pkt_buffer, sizeof(packet), 0, (SOCKADDR*)&addr, &addrLen);

        memcpy(&pkt, pkt_buffer, sizeof(packetHead));

        if (pkt.head.flag & END) {
            cout << "文件传输完毕" << endl;
            packetHead endPacket;
            endPacket.flag |= ACK;
            endPacket.checkSum = checkPacketSum((u_short*)&endPacket, sizeof(packetHead));
            memcpy(pkt_buffer, &endPacket, sizeof(packetHead));
            sendto(socket, pkt_buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, addrLen);
            return fileLen;
        }

        memcpy(&pkt, pkt_buffer, sizeof(packet));
```

```
        if (pkt.head.seq == curseq || checkPacketSum((u_short*)&pkt, sizeof(packet)) != 0) {
            sendPkt = makePacket(1);
            memcpy(pkt_buffer, &sendPkt, sizeof(packet));
            sendto(socket, pkt_buffer, sizeof(packet), 0, (SOCKADDR*)&addr, addrLen);
            cout << "收到重复的" << index - 1 << "号数据包，将其抛弃" << endl;
            break;
        }

        //correctly receive the seq
        dataLen = pkt.head.bufSize;
        memcpy(fileBuffer + fileLen, pkt.data, dataLen);
        fileLen += dataLen;

        //give back ack
        sendPkt = makePacket(curack);
        memcpy(pkt_buffer, &sendPkt, sizeof(packet));
        sendto(socket, pkt_buffer, sizeof(packet), 0, (SOCKADDR*)&addr, addrLen);

        //cout<<"成功收到"<<index<<"号数据包，其长度是"<<dataLen<<endl;
        curseq = (curseq + 1) % seqsize;
        curack = (curack + 1) % seqsize;
        index++;
    }
}
```

断开连接

服务端：

```
bool disConnect(SOCKET& socket, SOCKADDR_IN& addr) {
    int addrLen = sizeof(addr);
    char* buffer = new char[sizeof(packetHead)];

    recvfrom(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, &addrLen);
    if ((((packetHead*)buffer)->flag & FIN) && (checkPacketSum((u_short*)buffer, sizeof(packetHead) == 0))) {
        cout << "用户端断开" << endl;
    }
    else {
        cout << "错误发生，程序中断" << endl;
        return false;
    }

    packetHead closeHead;
    closeHead.flag = 0;
    closeHead.flag |= ACK;
    closeHead.checkSum = checkPacketSum((u_short*)&closeHead, sizeof(packetHead));
    memcpy(buffer, &closeHead, sizeof(packetHead));
    sendto(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, addrLen);

    closeHead.flag |= FIN;
    closeHead.checkSum = checkPacketSum((u_short*)&closeHead, sizeof(packetHead));
    memcpy(buffer, &closeHead, sizeof(packetHead));
    sendto(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, addrLen);
```

```
    u_long mode = 1;
    ioctlsocket(socket, FIONBIO, &mode);
    clock_t start = clock();
    while (recvfrom(socket, buffer, sizeof(packetHead), 0, (sockaddr*)&addr, &addrLen) <= 0) {
        if (clock() - start >= MAX_TIME) {
            memcpy(buffer, &closeHead, sizeof(packetHead));
            sendto(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, addrLen);
            start = clock();
        }
    }

    if ((((packetHead*)buffer)->flag & ACK) && (checkPacketSum((u_short*)buffer, sizeof(packetHead) == 0))) {
        cout << "链接关闭" << endl;
    }
    else {
        cout << "错误发生，程序中断" << endl;
        return false;
    }
    closesocket(socket);
    return true;
}
```

客户端：

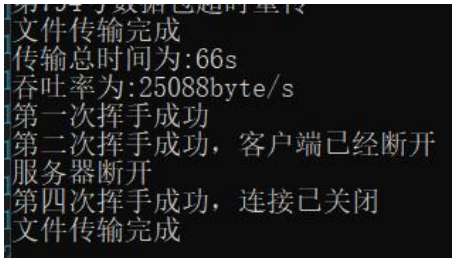
```
bool disConnect(SOCKET& socket, SOCKADDR_IN& addr) {

    int addrLen = sizeof(addr);
    char* buffer = new char[sizeof(packetHead)];
    packetHead closeHead;
    closeHead.flag |= FIN;
    closeHead.checkSum = checkPacketSum((u_short*)&closeHead, sizeof(packetHead));

    memcpy(buffer, &closeHead, sizeof(packetHead));
    if (sendto(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, addrLen) != SOCKET_ERROR)
        cout << "第一次挥手成功" << endl;
    else
        return false;

    clock_t start = clock();
    while (recvfrom(socket, buffer, sizeof(packetHead), 0, (sockaddr*)&addr, &addrLen) <= 0) {
        if (clock() - start >= MAX_TIME) {
            memcpy(buffer, &closeHead, sizeof(packetHead));
            sendto(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, addrLen);
            start = clock();
        }
    }

    if ((((packetHead*)buffer)->flag & ACK) && (checkPacketSum((u_short*)buffer, sizeof(packetHead) == 0))) {
        cout << "第二次挥手成功，客户端已经断开" << endl;
    }
    else {
        return false;
    }
```

```
    u_long mode = 0;
    ioctlsocket(socket, FIONBIO, &mode);//阻塞

    recvfrom(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, &addrLen);

    if ((((packetHead*)buffer)->flag & FIN) && (checkPacketSum((u_short*)buffer, sizeof(packetHead) == 0))) {
        cout << "服务器断开" << endl;
    }
    else {
        return false;
    }

    mode = 1;
    ioctlsocket(socket, FIONBIO, &mode);

    closeHead.flag = 0;
    closeHead.flag |= ACK;
    closeHead.checkSum = checkPacketSum((u_short*)&closeHead, sizeof(packetHead));

    memcpy(buffer, &closeHead, sizeof(packetHead));
    sendto(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, addrLen);
    start = clock();
    while (clock() - start <= 2 * MAX_TIME) {
        if (recvfrom(socket, buffer, sizeof(packetHead), 0, (SOCKADDR*)&addr, &addrLen) <= 0)
            continue;
        //说明这个ACK丢了
        memcpy(buffer, &closeHead, sizeof(packetHead));
        sendto(socket, buffer, sizeof(packetHead), 0, (sockaddr*)&addr, addrLen);
        start = clock();
    }

    cout << "第四次挥手成功，连接已关闭" << endl;
    closesocket(socket);
    return true;
}
```

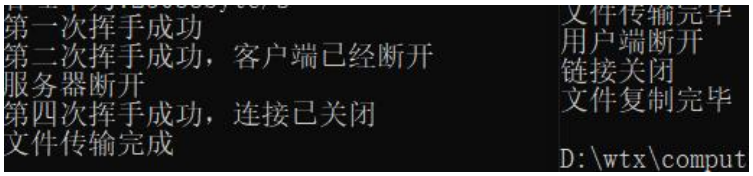
输出结果

建立连接

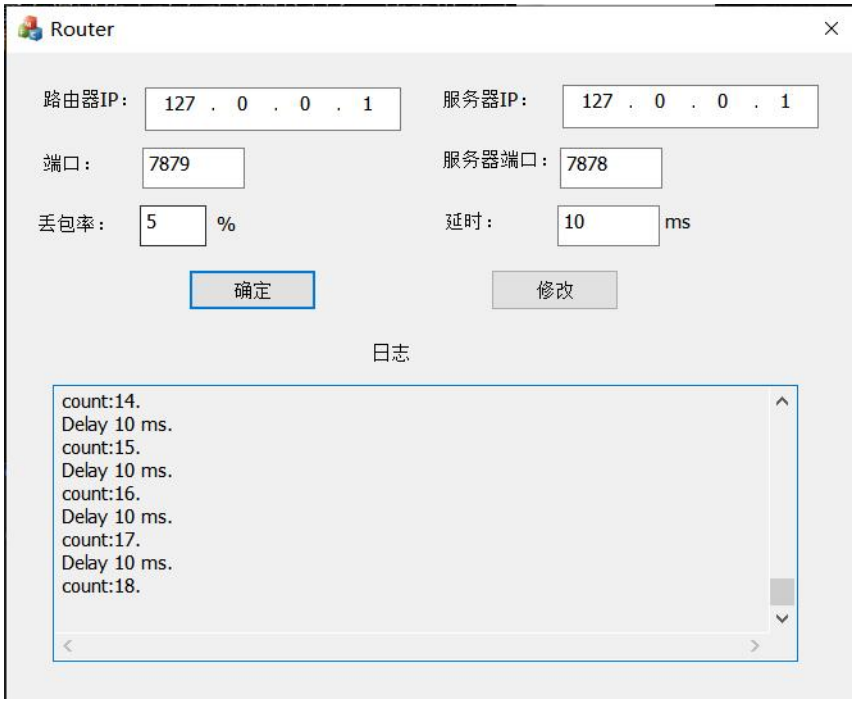
传输时间、吞吐率



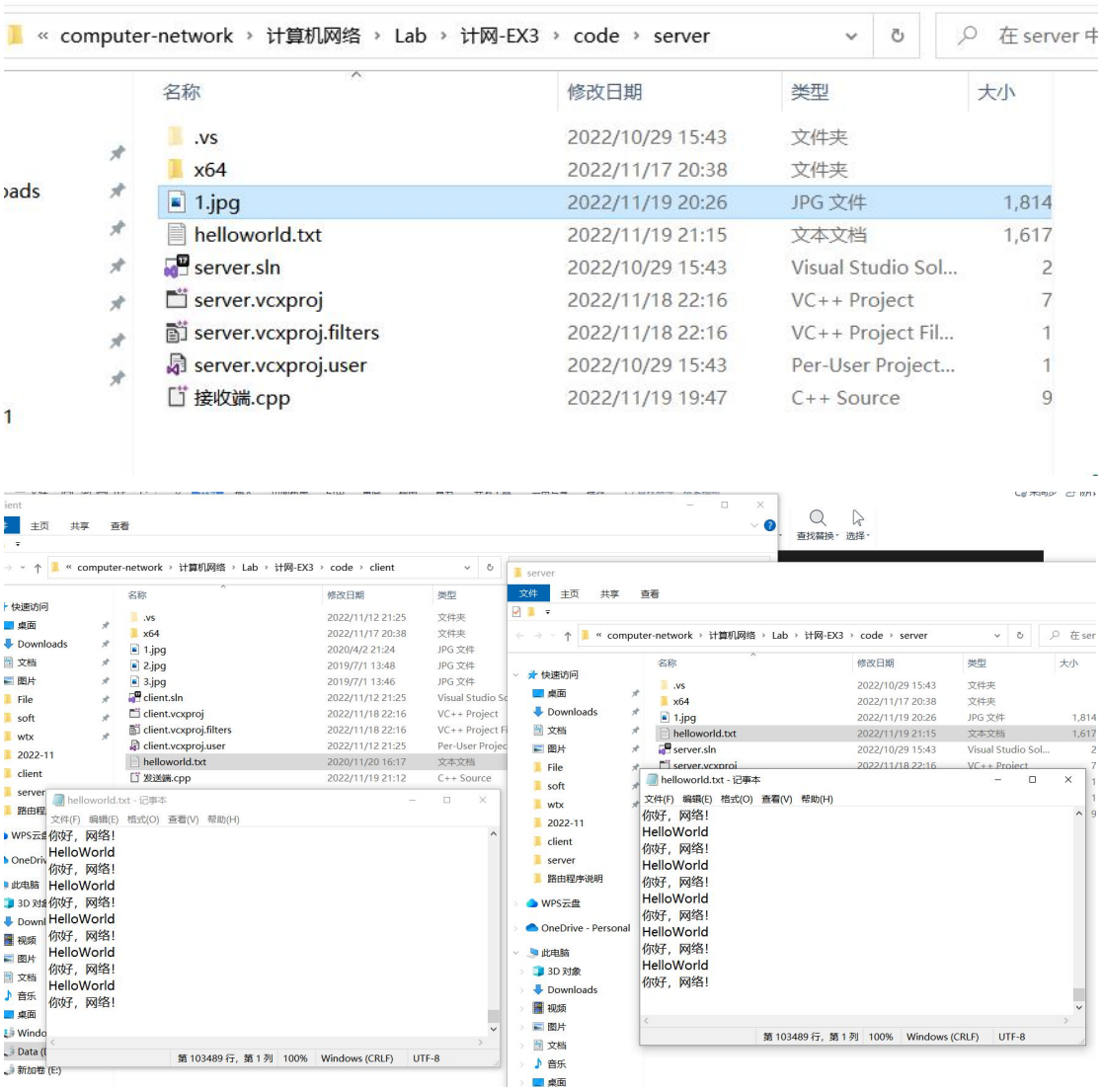
断开连接



路由程序日志输出



输出文件展示


« computer-network › 计算机网络 › Lab › 计网-EX3 › code › server
在 server 中
名称
修改日期
类型
大小
.vs 2022/10/29 15:43 文件夹
x64 2022/11/17 20:38 文件夹
1.jpg 2022/11/19 20:26 JPG 文件 1,814
helloworld.txt 2022/11/19 21:15 文本文档 1,617
server.sln 2022/10/29 15:43 Visual Studio Sol... 2
server.vcxproj 2022/11/18 22:16 VC++ Project 7
server.vcxproj.filters 2022/11/18 22:16 VC++ Project Fil... 1
server.vcxproj.user 2022/10/29 15:43 Per-User Project... 1
接收端.cpp 2022/11/19 19:47 C++ Source 9
« computer-network › 计算机网络 › Lab › 计网-EX3 › code › client
helloworld.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
你好，网络!
HelloWorld
第 103489 行，第 1 列 100% Windows (CRLF) UTF-8
« computer-network › 计算机网络 › Lab › 计网-EX3 › code › server
helloworld.txt - 记事本
第 103489 行，第 1 列 100% Windows (CRLF) UTF-8