计算机网络第三次作业第一部分

张昊星 2113419

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。

实验设计

协议设计

1. 报文格式

```
struct HEADER {
    u_short sum = 0;//校验和 16位
    u_short datasize = 0;//所包含数据长度 16位
    unsigned char flag = 0;//八位，使用后三位，排列是FIN ACK SYN
    unsigned char SEQ = 0;//八位，传输的序列号，0~255，超过后mod
    HEADER() {
        sum = 0;
        datasize = 0;
        flag = 0;
        SEQ = 0;
    }
};
```

报文头长度为48位，前16位为数据长度，用于记录数据区的大小，17-32位为校验和，用于检验传输的正确性，33-40位为标志位，只使用低3位，分别为FIN，ACK，SYN，40-48位为传输的数据包的序号（0~255循环使用）

2. 连接与断开

类似于TCP的握手与挥手功能：

- 三次握手进行连接：

  首先，客户端向服务端发送数据包，其中SYN=1，ACK=0，FIN=0

  服务端接受到数据包后，向客户端发送SYN=0，ACK=1，FIN=0

  客户端再次接收到数据包后，向服务端发送SYN=1，ACK=1，FIN=0

  服务端接收到数据包后，连接成功建立，可以进行数据传输

- 四次挥手断开连接：

  首先，客户端向服务端发送数据包，其中SYN=0，ACK=0，FIN=1

  服务端接受到数据包后，向客户端发送SYN=0，ACK=1，FIN=0

  客户端再次接收到数据包后，向服务端发送SYN=0，ACK=1，FIN=1

  服务端接收到数据包后，向客户端发送SYN=0，ACK=1，FIN=1

  客户端接收到数据包后，连接成功断开

3.数据传输

发送端和接收端的接收确认均采用rdt3.0，数据在传输时，将一个文件按照缓冲区的大小分为数个包进行分段传输，每个包的内容为数据头和数据。

在传输时，需要接受到上一个发送包序号的ACK=1才能发送下一个数据包；接收端接收到了一个数据包，先要进行校验，如果检查无误，则向发送放返回该序列号的ACK=1。

在特定的时间内（在程序里设定为0.5秒），如果没有收到该序列号的ACK=1，将会重新传输该包。

```
double MAX_TIME = 0.5 * CLOCKS_PER_SEC;
```

如果接收端收到了重复的包，则将其中一个丢弃，但仍需要向发送方发送该序列号的ACK=1。

在最后，发送方需要向接收端发送一个FIN=1， ACK=1，SYN=1的包，表示文件传输结束；接收端收到该包后，需要向发送方返回一个ACK=1，表示收到文件传输结束的信号。

代码实现

计算校验和

```
u_short cksum(u_short* mes, int size) {//计算校验和
    int count = (size + 1) / 2;
    u_short* buf = (u_short*)malloc(size + 1);
    if (buf != 0) {
        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);
}
```

数据准备：首先计算需要处理的 `u_short` 数组元素个数，每个 `u_short` 占用 2 个字节。然后分配内存`buf`以存储传入数据的副本。

数据复制：在检查内存分配是否成功，避免 `malloc` 返回空指针之后，将分配的内存初始化为0，再将传入的数据复制到分配的内存中，准备进行校验和计算。

校验和计算：初始化一个用于存储校验和的变量 `sum`，类型为 `u_long`，用于防止加法溢出。遍历 `buf` 中的每个 `u_short` 元素，将其值加到 `sum` 中。如果 `sum` 的高16位（超过 `0xffff`）不为 0，则进行循环进位操作，确保加法结果在16位以内。

返回校验和：返回计算得到的结果按位取反之后的值。

三次握手

- 客户端

```
int Connect(SOCKET& socketClient, SOCKADDR_IN& servAddr, int& servAddrlen)
{//三次握手建立连接
    HEADER header;
    char* Buffer = new char[sizeof(header)];
    //第一次握手
    header.flag = SYN;
    header.sum = 0;//校验和置0
    u_short temp = cksum((u_short*)&header, sizeof(header));
    header.sum = temp;//计算校验和
    memcpy(Buffer, &header, sizeof(header));//将首部放入缓冲区
    if (sendto(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen) == -1){
        return -1;
    }
    clock_t start = clock(); //记录发送第一次握手时间
    u_long mode = 1;
    ioctlsocket(socketClient, FIONBIO, &mode);
    //第二次握手
    while (recvfrom(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, &servAddrlen) <= 0){
        if (clock() - start > MAX_TIME){//超时，重新传输第一次握手
            cout << "第一次握手超时，正在进行超时重传" << endl;
            header.flag = SYN;
            header.sum = 0;
            header.sum = cksum((u_short*)&header, sizeof(header));
            memcpy(Buffer, &header, sizeof(header));
            sendto(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen);
            start = clock();
        }
    }
    //进行校验和检验
    memcpy(&header, Buffer, sizeof(header));
    if (header.flag != ACK || !cksum((u_short*)&header, sizeof(header) ==
0)){
        cout << "握手发生错误" << endl;
        return -1;
    }
    //进行第三次握手
    header.flag = ACK_SYN;
    header.sum = 0;
    header.sum = cksum((u_short*)&header, sizeof(header));//计算校验和
    if (sendto(socketClient, (char*)&header, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen) == -1)
    {
        return -1;//判断客户端是否打开，-1为未开启发送失败
    }
    cout << "服务器连接成功" << endl;
    return 1;
}
```

- 服务端

```
int Connect(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int& ClientAddrLen){
    HEADER header;
    char* Buffer = new char[sizeof(header)];
    //第一次握手
    while (1){
        if (recvfrom(sockServ, Buffer, sizeof(header), 0,
(sockaddr*)&ClientAddr, &ClientAddrLen) == -1){
            return -1;
        }
        memcpy(&header, Buffer, sizeof(header));
        if (header.flag == SYN && cksum((u_short*)&header, sizeof(header))
== 0){
            break;
        }
    }
    //发送第二次握手信息
    header.flag = ACK;
    header.sum = 0;
    u_short temp = cksum((u_short*)&header, sizeof(header));
    header.sum = temp;
    memcpy(Buffer, &header, sizeof(header));
    if (sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1){
        return -1;
    }
    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, Buffer, sizeof(header), 0,
(sockaddr*)&ClientAddr, &ClientAddrLen) <= 0)
    {
        if (clock() - start > MAX_TIME)
        {
            header.flag = ACK;
            header.sum = 0;
            u_short temp = cksum((u_short*)&header, sizeof(header));
            header.flag = temp;
            memcpy(Buffer, &header, sizeof(header));
            if (sendto(sockServ, Buffer, sizeof(header), 0,
(sockaddr*)&ClientAddr, ClientAddrLen) == -1)
            {
                return -1;
            }
            cout << "握手超时，正在进行重传" << endl;
        }
    }
    HEADER temp1;
    memcpy(&temp1, Buffer, sizeof(header));
    if (temp1.flag != ACK_SYN || !cksum((u_short*)&temp1, sizeof(temp1) ==
0)){
        cout << "连接发生错误" << endl;
        return -1;
    }
    return 1;
}
```

传输数据

- 发送单个数据包

```
void send_package(SOCKET& socketClient, SOCKADDR_IN& servAddr, int&
servAddrlen, char* message, int len, int& order){
    HEADER header;
    char* buffer = new char[MAXSIZE + sizeof(header)];
    header.datasize = len;
    header.SEQ = unsigned char(order);//序列号
    memcpy(buffer, &header, sizeof(header));
    memcpy(buffer + sizeof(header), message, sizeof(header) + len);
    u_short check = cksum((u_short*)buffer, sizeof(header) + len);//计算校验和
    header.sum = check;
    memcpy(buffer, &header, sizeof(header));
    sendto(socketClient, buffer, len + sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen);
    cout << "Send message " << len << " bytes!" << " flag:" <<
int(header.flag) << " SEQ:" << int(header.SEQ) << " SUM:" << int(header.sum)
<< endl;
    clock_t start = clock();//记录发送时间
    //接收ack等信息
    while (1){
        u_long mode = 1;
        ioctlsocket(socketClient, FIONBIO, &mode);
        while (recvfrom(socketClient, buffer, MAXSIZE, 0,
(sockaddr*)&servAddr, &servAddrlen) <= 0){
            if (clock() - start > MAX_TIME){
                header.datasize = len;
                header.SEQ = u_char(order);
                header.flag = u_char(0x0);
                memcpy(buffer, &header, sizeof(header));
                memcpy(buffer + sizeof(header), message, sizeof(header) +
len);
                u_short check = cksum((u_short*)buffer, sizeof(header) +
len);
                header.sum = check;
                memcpy(buffer, &header, sizeof(header));
                sendto(socketClient, buffer, len + sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen);
                cout << "TIME OUT! ReSend message " << len << " bytes!
Flag:" << int(header.flag) << " SEQ:" << int(header.SEQ) << endl;
                start = clock();
            }
        }
        memcpy(&header, buffer, sizeof(header));//缓冲区接收到信息，读取
        u_short check = cksum((u_short*)&header, sizeof(header));
        if (header.SEQ == u_short(order) && header.flag == ACK){
            cout << "Send has been confirmed! Flag:" << int(header.flag) <<
" SEQ:" << int(header.SEQ) << endl;
            break;
        }
        else{
            continue;
        }
    }
    u_long mode = 0;
```

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

准备发送的数据包：创建一个包含数据的缓冲区 `buffer`，大小为 `MAXSIZE + sizeof(header)`。将数据包的头信息填充到 `buffer` 的开头，并将实际数据复制到 `buffer` 中。

计算校验和：调用 `cksum()` 函数计算 `buffer` 中数据的校验和，并将结果保存到 `header.sum` 中。

发送数据包：使用 `sendto()` 函数将数据包发送到服务端。

等待确认：使用非阻塞模式进行接收，设置接收超时时间 `MAX_TIME`。进入一个 `while` 循环，循环中等待接收来自服务端的确认信息。如果接收到信息，检查接收到的确认信息是否正确，并确认接收到正确的序列号和确认标志。如果接收到正确的确认，结束循环，发送已被确认的信息。如果超时，则重新发送数据包，继续等待确认。

恢复阻塞模式：当确认信息已接收或超时后，将套接字设置回阻塞模式，以便后续的通信操作。

- 发送文件

```
void send(SOCKET& socketClient, SOCKADDR_IN& servAddr, int& servAddrlen,
char* message, int len){
    int packagenum = len / MAXSIZE + (len % MAXSIZE != 0);
    int seqnum = 0;
    for (int i = 0; i < packagenum; i++){
        send_package(socketClient, servAddr, servAddrlen, message + i *
MAXSIZE, i == packagenum - 1 ? len - (packagenum - 1) * MAXSIZE : MAXSIZE,
seqnum);
        seqnum++;
        if (seqnum > 255){
            seqnum = seqnum - 256;
        }
    }
    HEADER header;
    char* Buffer = new char[sizeof(header)];
    header.flag = OVER;
    header.sum = 0;
    header.sum = cksum((u_short*)&header, sizeof(header));
    memcpy(Buffer, &header, sizeof(header));
    sendto(socketClient, Buffer, sizeof(header), 0, (sockaddr*)&servAddr,
servAddrlen);
    cout << "发送结束" << endl;
    clock_t start = clock();
    while(1){
        u_long mode = 1;
        ioctlsocket(socketClient, FIONBIO, &mode);
        while (recvfrom(socketClient, Buffer, MAXSIZE, 0,
(sockaddr*)&servAddr, &servAddrlen) <= 0){
            if (clock() - start > MAX_TIME){
                char* Buffer = new char[sizeof(header)];
                header.flag = OVER;
                header.sum = 0;
                header.sum = cksum((u_short*)&header, sizeof(header));
                memcpy(Buffer, &header, sizeof(header));
                sendto(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen);
                cout << "Time Out! ReSend End!" << endl;
                start = clock();
            }
```

```
            }
            memcpy(&header, Buffer, sizeof(header));//缓冲区接收到信息，读取
            u_short check = cksum((u_short*)&header, sizeof(header));
            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
    }
```

数据分割和发送： `packagenum` 计算需要发送的数据包的数量，将数据分成大小为 `MAXSIZE` 的多个包进行发送。通过循环，对每个包调用 `send_package()` 函数进行发送，将数据分块发送给服务端。

发送结束标志：在发送完所有数据包后，创建一个结束标志的数据包，将包的头部信息中的标志位设置为结束标志，并发送这个结束标志给服务端。

等待接收端确认：进入一个循环，尝试从套接字接收数据，如果接收到了数据，则判断接收到的数据包的标志位是否为结束标志。如果接收到的标志位为结束标志，则表示接收端已成功接收完整文件，函数结束。如果超时或接收到的不是结束标志，则需要重新发送结束标志。

恢复阻塞模式：当确认接收完整文件后，将套接字设置回阻塞模式，以便后续的通信操作。

接收数据

```
int RecvMessage(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int& ClientAddrLen,
char* message){
    long int file_length = 0;//文件长度
    HEADER header;
    char* Buffer = new char[MAXSIZE + sizeof(header)];
    int seq = 0;
    int index = 0;

    while (1){
        int length = recvfrom(sockServ, Buffer, sizeof(header) + MAXSIZE, 0,
(sockaddr*)&ClientAddr, &ClientAddrLen);//接收报文长度
        memcpy(&header, Buffer, sizeof(header));
        //判断是否是结束
        if (header.flag == OVER && cksum((u_short*)&header, sizeof(header)) ==
0){
            cout << "文件接收完毕" << endl;
            break;
        }
        if (header.flag == unsigned char(0) && cksum((u_short*)Buffer, length -
sizeof(header))){
            if (seq != int(header.SEQ)){
                header.flag = ACK;
                header.datasize = 0;
                header.sum = 0;
                u_short temp = cksum((u_short*)&header, sizeof(header));
                header.sum = temp;
                memcpy(Buffer, &header, sizeof(header));
                //重发该包的ACK
```

```
                sendto(sockServ, Buffer, sizeof(header), 0,
(sockaddr*)&ClientAddr, ClientAddrLen);
                cout << "Send to Clinet ACK:" << (int)header.flag << " SEQ:" <<
(int)header.SEQ << endl;
                continue;//丢弃该数据包
            }
            seq = int(header.SEQ);
            if (seq > 255){
                seq = seq - 256;
            }
            //取出buffer中的内容
            cout << "Recv message " << length - sizeof(header) << " bytes!Flag:"
<< int(header.flag) << " SEQ : " << int(header.SEQ) << " SUM:" <<
int(header.sum) << endl;
            char* temp = new char[length - sizeof(header)];
            memcpy(temp, Buffer + sizeof(header), length - sizeof(header));
            memcpy(message + file_length, temp, length - sizeof(header));
            file_length = file_length + int(header.datasize);
            //返回ACK
            header.flag = ACK;
            header.datasize = 0;
            header.SEQ = (unsigned char)seq;
            header.sum = 0;
            u_short temp1 = cksum((u_short*)&header, sizeof(header));
            header.sum = temp1;
            memcpy(Buffer, &header, sizeof(header));
            sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr*)&ClientAddr,
ClientAddrLen);
            cout << "Send to Clinet ACK:" << (int)header.flag << " SEQ:" <<
(int)header.SEQ << endl;
            seq++;
            if (seq > 255){
                seq = seq - 256;
            }
        }
    }
    header.flag = OVER;
    header.sum = 0;
    u_short temp = cksum((u_short*)&header, sizeof(header));
    header.sum = temp;
    memcpy(Buffer, &header, sizeof(header));
    if (sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1){
        return -1;
    }
    return file_length;
}
```

循环接收数据报文：使用 `recvfrom()` 从套接字接收数据报文，并将其存储在 `Buffer` 中。提取报文头部信息 `HEADER`，判断接收到的报文是否是结束标志。

确认序列号和校验和：对接收到的数据报文进行序列号和校验和的检查。如果序列号错误或者校验和不匹配，则发送一个确认（ACK）并丢弃该数据包。如果接收到的序列号与当前期望的序列号一致，则将数据从报文中提取出来，并存储到 `message` 缓冲区中。

发送确认（ACK）：发送确认消息给客户端，确认接收到的数据包。

循环接收直到收到结束标志：循环接收直到接收到结束标志的数据包。

发送结束标志的确认：发送结束标志的确认消息给客户端。

四次挥手

- 客户端

```
int disConnect(SOCKET& socketClient, SOCKADDR_IN& servAddr, int&
servAddrlen){//四次挥手断开连接
    HEADER header;
    char* Buffer = new char[sizeof(header)];
    //进行第一次挥手
    header.flag = FIN;
    header.sum = 0;//校验和置0
    header.sum = cksum((u_short*)&header, sizeof(header));
    memcpy(Buffer, &header, sizeof(header));//将首部放入缓冲区
    if (sendto(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen) == -1){
        return -1;
    }
    clock_t start = clock(); //记录发送第一次挥手时间
    u_long mode = 1;
    ioctlsocket(socketClient, FIONBIO, &mode);
    //第二次挥手
    while (recvfrom(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, &servAddrlen) <= 0){
        if (clock() - start > MAX_TIME){//超时，重新传输第一次挥手
            cout << "第一次挥手超时，正在进行重传" << endl;
            header.flag = FIN;
            header.sum = 0;
            header.sum = cksum((u_short*)&header, sizeof(header));
            memcpy(Buffer, &header, sizeof(header));//将首部放入缓冲区
            sendto(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen);
            start = clock();
        }
    }
    //进行校验和检验
    memcpy(&header, Buffer, sizeof(header));
    if (header.flag != ACK || !cksum((u_short*)&header, sizeof(header) ==
0)){
        cout << "连接发生错误，程序直接退出！" << endl;
        return -1;
    }
    //第三次挥手
    header.flag = FIN_ACK;
    header.sum = 0;
    header.sum = cksum((u_short*)&header, sizeof(header));//计算校验和
    if (sendto(socketClient, (char*)&header, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen) == -1){
        return -1;
    }
    start = clock();
    //第四次挥手
    while (recvfrom(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, &servAddrlen) <= 0){
        if (clock() - start > MAX_TIME){//超时，重新传输第三次挥手
            cout << "第三次握手超时，正在进行重传" << endl;
```

```
            header.flag = FIN;
            header.sum = 0;//校验和置0
            header.sum = cksum((u_short*)&header, sizeof(header));//计算校验和
            memcpy(Buffer, &header, sizeof(header));//将首部放入缓冲区
            sendto(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrlen);
            start = clock();
        }
    }
    cout << "四次挥手结束，连接断开！" << endl;
    return 1;
}
```

- 服务端

```
int disConnect(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int&
ClientAddrLen){
    HEADER header;
    char* Buffer = new char[sizeof(header)];
    while (1){
        int length = recvfrom(sockServ, Buffer, sizeof(header) + MAXSIZE, 0,
(sockaddr*)&ClientAddr, &ClientAddrLen);//接收报文长度
        memcpy(&header, Buffer, sizeof(header));
        if (header.flag == FIN && cksum((u_short*)&header, sizeof(header))
== 0){
            break;
        }
    }
    //第二次挥手
    header.flag = ACK;
    header.sum = 0;
    u_short temp = cksum((u_short*)&header, sizeof(header));
    header.sum = temp;
    memcpy(Buffer, &header, sizeof(header));
    if (sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1)
    {
        return -1;
    }
    clock_t start = clock();//记录第二次挥手发送时间

    //第三次挥手
    while (recvfrom(sockServ, Buffer, sizeof(header), 0,
(sockaddr*)&ClientAddr, &ClientAddrLen) <= 0){
        if (clock() - start > MAX_TIME){
            cout << "第二次挥手超时，正在进行重传" << endl;
            header.flag = ACK;
            header.sum = 0;
            u_short temp = cksum((u_short*)&header, sizeof(header));
            header.flag = temp;
            memcpy(Buffer, &header, sizeof(header));
            if (sendto(sockServ, Buffer, sizeof(header), 0,
(sockaddr*)&ClientAddr, ClientAddrLen) == -1){
                return -1;
            }
        }
    }
```

```
    HEADER temp1;
    memcpy(&temp1, Buffer, sizeof(header));
    if (temp1.flag != FIN_ACK || !cksum((u_short*)&temp1, sizeof(temp1) ==
0)){
        cout << "发生错误" << endl;
        return -1;
    }
    //发送第四次挥手信息
    header.flag = FIN_ACK;
    header.sum = 0;
    temp = cksum((u_short*)&header, sizeof(header));
    header.sum = temp;
    memcpy(Buffer, &header, sizeof(header));
    if (sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1){
        cout << "发生错误" << endl;
        return -1;
    }
    cout << "四次挥手结束，连接断开！" << endl;
    return 1;
}
```

客户端

```
int main(){
    WSADATA wsadata;
    WSAStartup(MAKEWORD(2, 2), &wsadata);
    SOCKADDR_IN server_addr;
    SOCKADDR_IN client_addr;
    SOCKET Client;
    char serverIP[50];
    int server_port;
    char clientIP[50];
    int client_port;
    cout << "请输入目标IP: ";
    cin.getline(serverIP, sizeof(serverIP));
    cout << "请输入目标端口: ";
    cin >> server_port;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(server_port);
    if (inet_pton(AF_INET, serverIP, &server_addr.sin_addr) <= 0) {
        cerr << "目标IP地址不可用" << endl;
        WSACleanup();
        return 1;
    }
    cin.ignore();
    cout << "请输入本机IP: ";
    cin.getline(clientIP, sizeof(clientIP));
    cout << "请输入本机端口: ";
    cin >> client_port;
    client_addr.sin_family = AF_INET;
    client_addr.sin_port = htons(client_port);
    if (inet_pton(AF_INET, clientIP, &client_addr.sin_addr) <= 0) {
        cerr << "本机IP地址不可用" << endl;
        WSACleanup();
        return 1;
    }
```

```
    Client = socket(AF_INET, SOCK_DGRAM, 0);
    bind(Client, (SOCKADDR*)&client_addr, sizeof(client_addr));
    int len = sizeof(server_addr);
    if (Connect(Client, server_addr, len) == -1){
        return 0;
    }
    string filename;
    cout << "请输入文件名称:";
    cin >> filename;
    ifstream fin(filename.c_str(), ifstream::binary);
    char* buffer = new char[100000000];
    int index = 0;
    unsigned char temp = fin.get();
    while (fin){
        buffer[index++] = temp;
        temp = fin.get();
    }
    fin.close();
    send(Client, server_addr, len, (char*)(filename.c_str()),
filename.length());
    clock_t start = clock();
    send(Client, server_addr, len, buffer, index);
    clock_t end = clock();
    cout << "传输总时间为:" << (end - start) / CLOCKS_PER_SEC << "s" << endl;
    cout << "吞吐率为:" << ((float)index) / ((end - start) / CLOCKS_PER_SEC) <<
"byte/s" << endl;
    disConnect(Client, server_addr, len);
    delete[] buffer;
    WSACleanup();
    system("pause");
    return 0;
}
```

初始化和配置服务器地址：通过 `WSAStartup` 函数初始化 Winsock 库，创建服务器地址 `server_addr` 和客户端地址 `client_addr`。

从用户输入获取目标服务器的 IP 地址和端口号，以及本地客户端的 IP 地址和端口号。

创建 UDP 套接字并绑定到本地端口：使用 `socket()` 函数创建 UDP 套接字 `Client`，然后使用 `bind()` 函数将其绑定到本地客户端地址。

连接到服务器：使用 `Connect()` 函数连接到目标服务器，实际上在 UDP 中 `Connect()` 并不会真正建立连接，而是为后续的发送操作指定默认目标地址。

读取文件数据并发送：从用户输入获取要传输的文件名，并使用 `ifstream` 以二进制方式打开文件。读取文件内容到名为 `buffer` 的字符数组中。使用 `send()` 函数向服务器发送文件名和文件内容。

计算传输时间和吞吐率：使用 `clock()` 记录发送文件的起始时间和结束时间，以计算总传输时间。根据传输时间和发送的字节数计算吞吐率。

关闭连接并清理资源：使用 `disConnect()` 断开与服务器的连接。释放动态分配的内存空间并关闭 Winsock 库。

服务端

```
int main(){
    WSADATA wsadata;
    WSAStartup(MAKEWORD(2, 2), &wsadata);
    SOCKADDR_IN server_addr;
    SOCKET Server;
    char serverIP[50];
    int port;
    cout << "请输入本服务器IP: ";
    cin.getline(serverIP, sizeof(serverIP));
    cout << "请输入本服务器端口: ";
    cin >> port;
    server_addr.sin_family = AF_INET;//使用IPV4
    server_addr.sin_port = htons(port);
    if (inet_pton(AF_INET, serverIP, &server_addr.sin_addr) <= 0) {
        cerr << "本服务器IP地址不可用" << endl;
        WSACleanup();
        return 1;
    }
    Server = socket(AF_INET, SOCK_DGRAM, 0);
    bind(Server, (SOCKADDR*)&server_addr, sizeof(server_addr));//绑定套接字，进入监
听状态
    cout << "进入监听状态，等待客户端连接...." << endl;
    int len = sizeof(server_addr);
    Connect(Server, server_addr, len);
    char* name = new char[20];
    char* data = new char[100000000];
    int namelen = RecvMessage(Server, server_addr, len, name);
    int datalen = RecvMessage(Server, server_addr, len, data);
    string file;
    for (int i = 0; i < namelen; i++){
        file = file + name[i];
    }
    disConnect(Server, server_addr, len);
    ofstream fout(file.c_str(), ofstream::binary);
    for (int i = 0; i < datalen; i++){
        fout << data[i];
    }
    fout.close();
    cout << file <<"已成功下载到本地" << endl;
    delete[] name;
    delete[] data;
    WSACleanup();
    system("pause");
    return 0;
}
```

初始化和配置服务器地址：使用 `WSAStartup` 初始化 Winsock 库，创建服务器地址 `server_addr`。从用户输入获取本地服务器的 IP 地址和端口号。

创建 UDP 套接字并绑定到本地端口：使用 `socket()` 函数创建 UDP 套接字 `Server`，然后使用 `bind()` 函数将其绑定到本地服务器地址。进入监听状态，等待客户端连接。

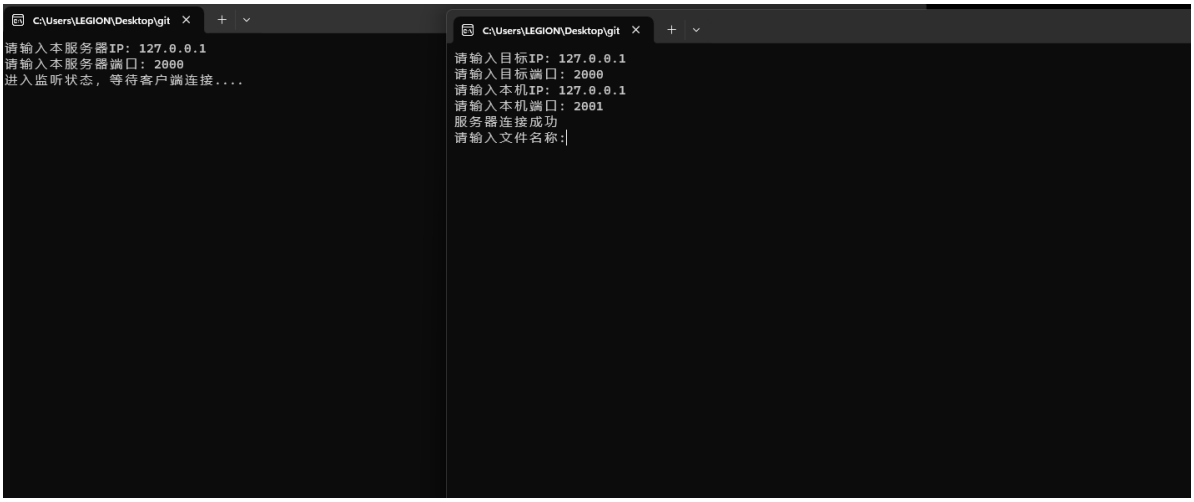
接收文件名和文件数据：创建缓冲区并接收文件名和文件数据，通过调用 `RecvMessage()` 函数两次来分别接收文件名和文件数据。将接收到的文件名组合成字符串 `file`。

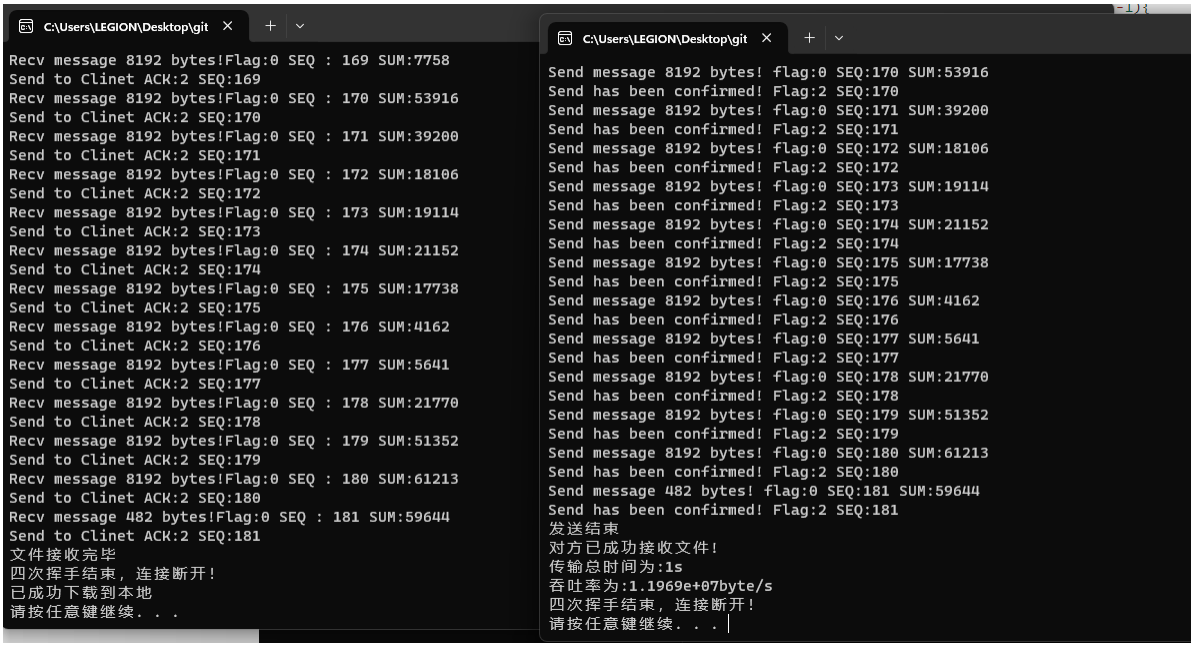
断开连接并保存文件：调用 `disConnect()` 函数来断开连接。将接收到的文件数据写入文件中。

清理资源：释放动态分配的内存空间，关闭 Winsock 库。

实验结果

正常情况

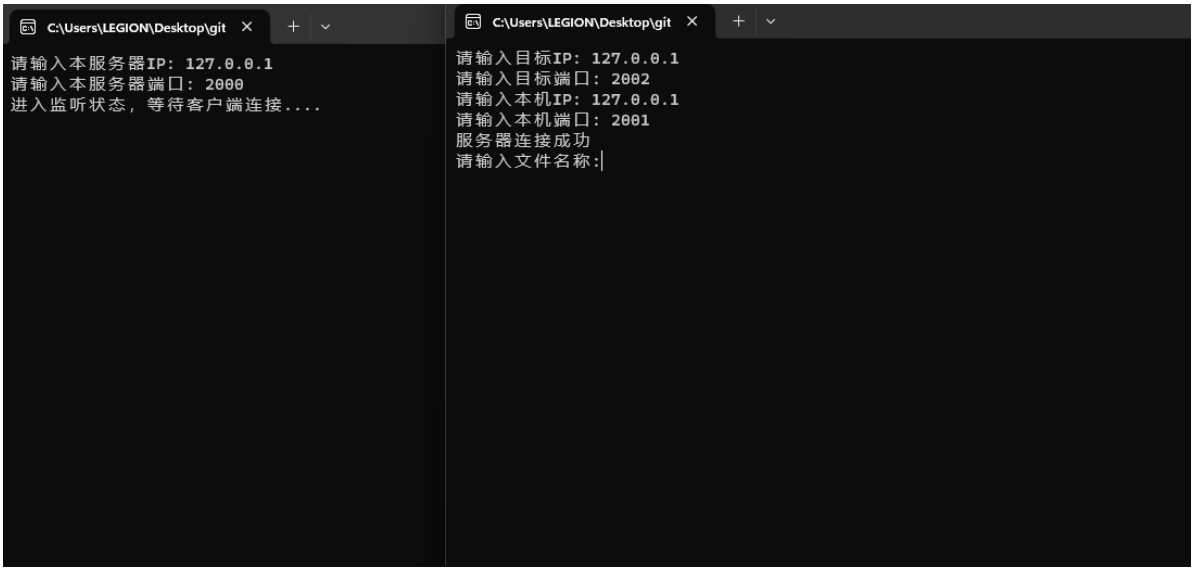


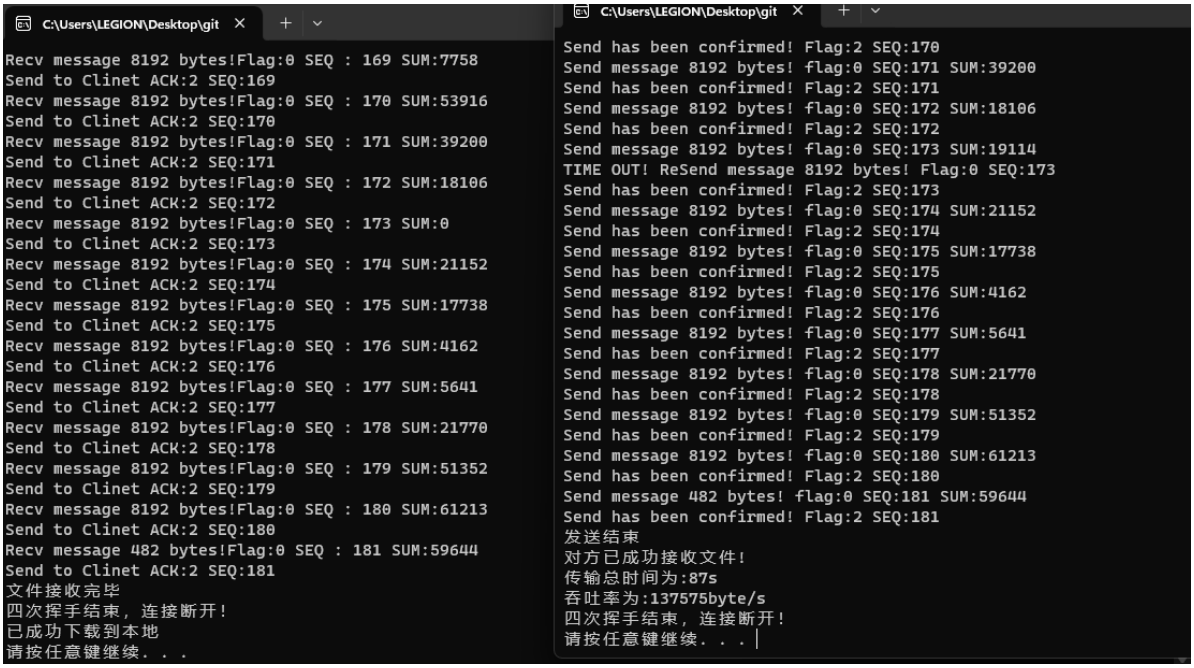


接收到的图片大小与原图片一致，传输成功。

路由情况

路由器设置如下：





同样成功传输并在客户端运行界面可以看到“TIME OUT”的信号，说明超时重传成功。