南开大学
Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机网络实验报告

Lab 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

徐俊智

年级：2022 级

专业：计算机科学与技术

指导教师：吴英

2024 年 11 月 28 日

目录

一、 实验目的

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、接收确认、超时重传等。流量控制采用停等机制，完成给定测试文件的传输

二、 实验要求

1. 实现单向传输。
2. 对于每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、文件传输时延，给出图形结果并进行分析。
6. 完成详细的实验报告。
7. 编写的程序应该结构清晰，具有较好的可读性。
8. 现场演示。
9. 提交程序源码、可执行文件和实验报告。

三、 UDP 报文段格式

```
class Message {
public:
    u_long flag;          // 伪首部
    u_short seq;          // 序列号
    u_short ack;          // 确认号
    u_long len;           // 数据部分长度
    u_long num;           // 数据包个数
    u_short checksum;     // 校验和
    char data[1024];      // 数据

    Message() { memset(this, 0, sizeof(Message)); }

    bool isSYN() { return this->flag & 1; }

    bool isACK() { return this->flag & 2; }

    bool isFIN() { return this->flag & 4; }

    bool isSTART() { return this->flag & 8; }

    bool isEND() { return this->flag & 16; }
```

Message 的成员有伪首部 flag，发送号 seq 和确认号 ack，数据部分长度 len，校验和 checksum，数据部分 data[1024]，其长度设置为 1024。

其中，flag 包含的属性有：SYN，ACK，FIN，START，END。SYN 和 FIN 分别用于建立和关闭连接，ACK 表示响应，START 和 END 表示文件传输的开始和结束。

其次，定义了查询 flag 是否包含某个属性的函数。

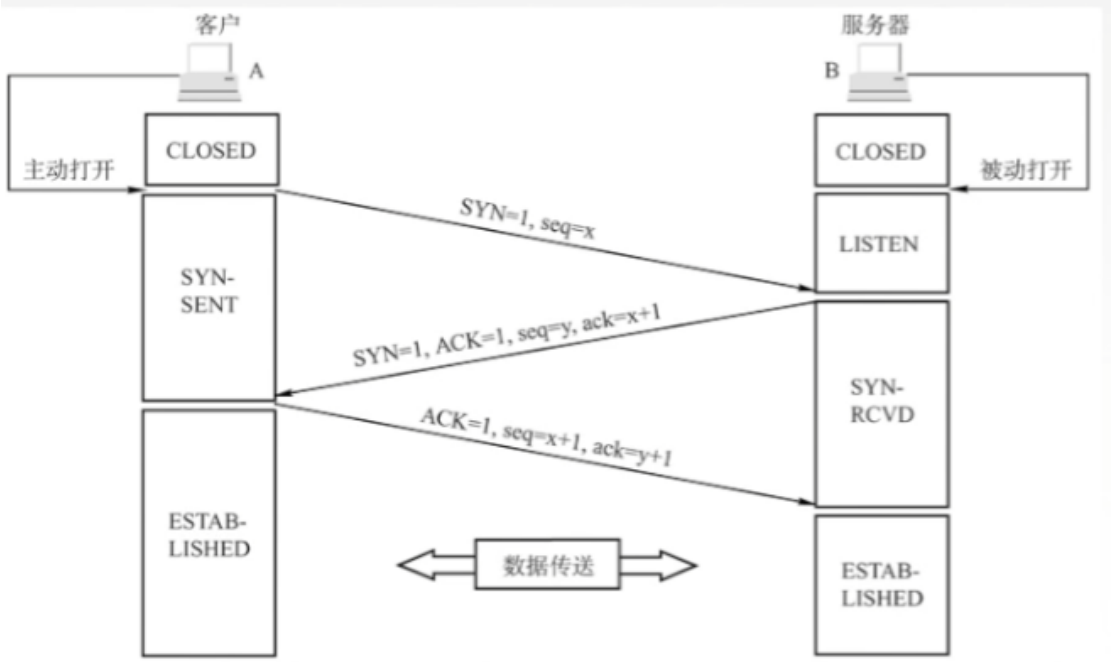
```
void setSYN() { this->flag |= 1; }
void setACK() { this->flag |= 2; }
void setFIN() { this->flag |= 4; }
void setSTART() { this->flag |= 8; }
void setEND() { this->flag |= 16; }
```

最后,定义了计算校验和的函数 `check_sum` 和检查数据包是否损坏的函数 packetCorruption。

```
    void setChecksum() {
        int sum = 0;
        u_char* temp = (u_char*)this;
        for (int i = 0; i < 8; i++) {
            sum += (temp[i << 1] << 8) + temp[i << 1 | 1];
            while (sum >= 0x10000) {
                // 溢出
                int t = sum >> 16;  // 将最高位回滚添加至最低位
                sum += t;
            }
        }
        this->checksum = ~(u_short)sum;  // 按位取反，方便校验计算
    }

    bool packetCorruption() {
        int sum = 0;
        u_char* temp = (u_char*)this;
        for (int i = 0; i < 8; i++) {
            sum += (temp[i << 1] << 8) + temp[i << 1 | 1];
            while (sum >= 0x10000) {
                // 溢出
                int t = sum >> 16; // 计算方法与设置校验和相同
                sum += t;
            }
        }
        // 把计算出来的校验和和报文中该字段的值相加，如果等于0xffff，则校验成功
        if (checksum + (u_short)sum == 65535)
            return false;
        return true;
    }
```

四、 建立连接 & 关闭连接

通过三次握手建立连接



- 第一次握手：客户端发出连接请求报文，SYN=1，seq=x=2000。
- 第二次握手：服务器端收到来自客户端的连接请求报文后，通过标志位 SYN=1 知道了客户端请求建立连接。然后服务器端向客户端发出确认报文，SYN=1，ACK=1，seq=y=1000，ack=x+1=2001。
- 第三次握手：客户端收到来自服务器端的确认报文后，检查 ACK 是否为 1、ack 是否为 x+1。如果正确，客户端向服务器端发出确认报文,ACK=1，seq=x+1=2001,ack=y+1=1001。

代码如下:

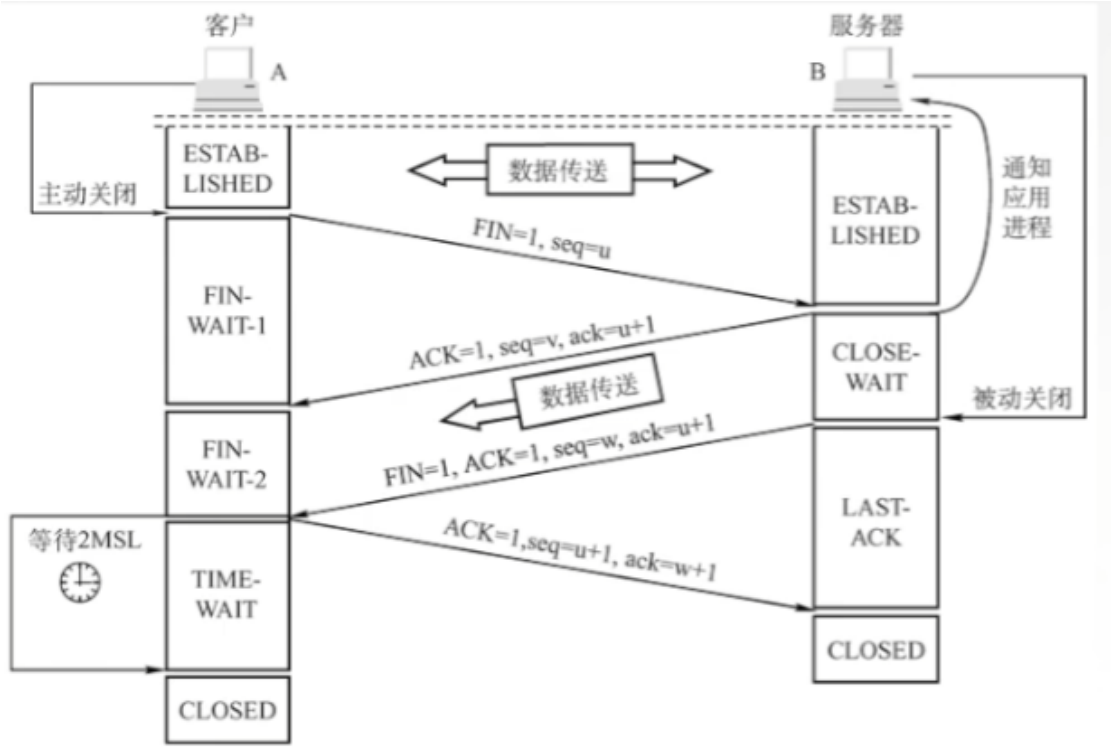
```
//Server.cpp
bool waitConnect() {
    // 设置套接字为非阻塞模式
    int mode = 1;
    ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);

    Message sendMsg, recvMsg;
    clock_t start;

    // 接收第一次握手消息
    while (1) {
        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            clientAddr, &len) != SOCKET_ERROR) {
            if (recvMsg.isSYN() && !recvMsg.packetCorruption()) {
                cout << "服务器端接收到第一次握手消息! 第一次握手成功!" <<
                    endl;
```

```
15                  break;
16              }
17          }
18      }
19
20      // 发送第二次握手消息
21      cout << "服务器端发送第二次握手消息! " << endl;
22      sendMsg.setSYN();
23      sendMsg.setACK();
24      sendMsg.seq = 1000;
25      sendMsg.ack = recvMsg.seq + 1;
26      sendMsg.setChecksum();
27      if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
            clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
28          cout << "服务器端发送第二次握手消息失败!" << endl;
29          cout << "当前网络状态不佳, 请稍后再试" << endl;
30          return 0;
31      }
32
33      // 接收第三次握手消息, 超时重传
34      start = clock();
35      while (1) {
36          if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
                clientAddr, &len) != SOCKET_ERROR) {
37              if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
                    .packetCorruption()) {
38                  cout << "服务器端接收到第三次握手消息! 第三次握手成功! " <<
                        endl;
39                  break;
40              }
41          }
42          if (clock() - start > RTO) {
43              cout << "第二次握手超时,服务器端重新发送第二次握手消息" << endl;
44              if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                    clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
45                  cout << "服务器端发送第二次握手消息失败!" << endl;
46                  cout << "当前网络状态不佳, 请稍后再试" << endl;
47                  return 0;
48              }
49              start = clock();
50          }
51      }
52
53      // 设置套接字为阻塞模式
54      mode = 0;
55      ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);
56
57      return 1;
```

```
}

//Client.cpp
bool waitConnect() {
    Message sendMsg, recvMsg;
    clock_t start;

    // 设置套接字为非阻塞模式
    int mode = 1;
    ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & mode);

    // 发送第一次握手消息
    cout << "尝试建立连接! 客户端发送第一次握手消息" << endl;
    sendMsg.setSYN();
    sendMsg.seq = 2000;
    sendMsg.setChecksum();
    if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        cout << "客户端发送第一次握手消息失败!" << endl;
        cout << "当前网络状态不佳, 请稍后再试" << endl;
        return 0;
    }

    // 接收第二次握手消息, 超时重传
    start = clock();
    while (1) {
        if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            serverAddr, &len) != SOCKET_ERROR) {
            if (recvMsg.isSYN() && recvMsg.isACK() && recvMsg.ack == sendMsg.
                seq + 1 && !recvMsg.packetCorruption()) {
                cout << "客户端接收到第二次握手消息! 第二次握手成功!" << endl
                    ;
                break;
            }
        }
        if (clock() - start > RTO) {
            cout << "第一次握手超时,客户端重新发送第一次握手消息" << endl;
            if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
                cout << "客户端发送第一次握手消息失败!" << endl;
                cout << "当前网络状态不佳, 请稍后再试" << endl;
                return 0;
            }
            start = clock();
        }
    }

    // 发送第三次握手消息
```

```
101     cout << "客户端发送第三次握手消息" << endl;
102     sendMsg.setACK();
103     sendMsg.seq = 2001;
104     sendMsg.ack = recvMsg.seq + 1;
105     sendMsg.setChecksum();
106     if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
            serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
107         cout << "客户端发送第三次握手消息失败!" << endl;
108         cout << "当前网络状态不佳，请稍后再试" << endl;
109         return 0;
110     }
111
112     return 1;
113 }
```

通过四次挥手关闭连接



- 第一次挥手：客户端发出连接释放报文，FIN=1，seq=u=3000。
- 第二次挥手：服务器端收到来自客户端的连接请求报文后，通过标志位 FIN=1 知道了客户端请求释放连接。然后服务器端向客户端发出确认报文,ACK=1,seq=v=4000,ack=u+1=3001。
- 第三次挥手:当服务器端确认数据传输完毕后,向客户端发送连接释放报文,FIN=1,ACK=1,seq=w=5000，ack=u+1=3001。
- 第四次挥手：客户端收到来自服务器端的连接释放报文后，通过标志位 FIN=1 知道了服务器端请求释放连接,然后客户端向服务器发出确认报文,ACK=1,seq=u+1=3001,ack=w+1=5001。

代码如下:

```cpp
//Server.cpp
bool closeConnect(Message recvMsg) {
    Message sendMsg;
    clock_t start;

    /*
        第一次挥手在recv_file函数里面处理
    */

    // 设置套接字为非阻塞模式
    int mode = 1;
    ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);

    // 发送第二次挥手消息
    cout << "服务器端发送第二次挥手消息! " << endl;
    sendMsg.setACK();
    sendMsg.seq = 4000;
    sendMsg.ack = recvMsg.seq + 1;
    sendMsg.setChecksum();
    if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        cout << "服务器端发送第二次挥手消息失败!" << endl;
        cout << "当前网络状态不佳, 请稍后再试" << endl;
        return 0;
    }

    // 发送第三次挥手消息
    cout << "服务器端发送第三次挥手消息! " << endl;
    sendMsg.setFIN();
    sendMsg.setACK();
    sendMsg.seq = 5000;
    sendMsg.ack = recvMsg.seq + 1;
    sendMsg.setChecksum();
    if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        cout << "服务器端发送第三次挥手消息失败!" << endl;
        cout << "当前网络状态不佳, 请稍后再试" << endl;
        return 0;
    }

    // 接收第四次挥手消息, 超时重传
    start = clock();
    while (1) {
        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            clientAddr, &len) != SOCKET_ERROR) {
            if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
                .packetCorruption()) {
```

```
                cout << "服务器端接收到第四次挥手消息! 第四次挥手成功! " <<
                    endl;
                break;
            }
        }
        if (clock() - start > RTO) {
            cout << "第三次挥手超时,服务器端重新发送第三次挥手消息" << endl;
            if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
                cout << "服务器端发送第三次挥手消息失败!" << endl;
                cout << "当前网络状态不佳, 请稍后再试" << endl;
                return 0;
            }
            start = clock();
        }
    }
    return 0;
}


//Client.cpp
bool closeConnect() {
    Message sendMsg, recvMsg;
    clock_t start;

    // 发送第一次挥手消息
    cout << "尝试关闭连接! 客户端发送第一次挥手消息" << endl;
    sendMsg.setFIN();
    sendMsg.seq = 3000;
    sendMsg.setChecksum();
    if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        cout << "客户端发送第一次挥手消息失败!" << endl;
        cout << "当前网络状态不佳, 请稍后再试" << endl;
        return 0;
    }

    // 接收第二次挥手消息, 超时重传
    start = clock();
    while (1) {
        if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            serverAddr, &len) != SOCKET_ERROR) {
            if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
                .packetCorruption()) {
                cout << "客户端接收到第二次挥手消息! 第二次挥手成功! " <<
                    endl;
                break;
            }
```

```
86          }
87          if (clock() - start > RTO) {
88              cout << "第一次挥手超时,客户端重新发送第一次挥手消息" << endl;
89              if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                    serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
90                  cout << "客户端发送第一次挥手消息失败!" << endl;
91                  cout << "当前网络状态不佳, 请稍后再试" << endl;
92                  return 0;
93              }
94              start = clock();
95          }
96      }
97
98      // 接收第三次挥手消息, 超时重传
99      start = clock();
100     while (1) {
101         if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
                serverAddr, &len) != SOCKET_ERROR) {
102             if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
                    .packetCorruption()) {
103                 cout << "客户端接收到第三次挥手消息! 第三次挥手成功! " <<
                        endl;
104                 break;
105             }
106         }
107         if (clock() - start > RTO) {
108             cout << "第二次挥手超时,客户端重新发送第二次挥手消息" << endl;
109             if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                    serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
110                 cout << "客户端发送第二次挥手消息失败!" << endl;
111                 cout << "当前网络状态不佳, 请稍后再试" << endl;
112                 return 0;
113             }
114             start = clock();
115         }
116     }
117
118     // 发送第四次挥手消息
119     cout << "客户端发送第四次挥手消息! " << endl;
120     sendMsg.setACK();
121     sendMsg.seq = 3001;
122     sendMsg.ack = recvMsg.seq + 1;
123     sendMsg.setChecksum();
124     if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
            serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
125         cout << "客户端发送第四次挥手消息失败!" << endl;
126         cout << "当前网络状态不佳, 请稍后再试" << endl;
127         return 0;
```

```
    }

    return 1;
}
```

因为建立和关闭连接时都是客户先发起，服务器需要先阻塞在一个 while 循环里不断的接收消息，直到接收了客户发来的信息，才能继续往下执行。

五、 差错检测

计算检验和主要有三个步骤：

- 求和：把需要校验的数据看成以 16 位为单位的数字组成，依次进行二进制求和。
- 回卷：求和后超过 16 位的加到低 16 位。
- 取反：最后结果取反码就是检验和。

客户端发送数据包的时候，通过 setChecksum 函数设置校验和。服务器端接收数据包的时候，通过 packetCorruption 函数验证数据包是否正确。代码如下：

```
void setchecksum(){
    int sum = 0;
    u_char* temp = (u_char*)this;
    for (int i = 0; i < 8; i++)
    {
        sum += (temp[i<<1] << 8) + temp[i<<1|1];
        while (sum > 0xffff)
        {
            int t = sum >> 16;
            sum += t;
        }
    }
    checksum = ~(u_short)sum;
}

bool packetCorruption() {
    int sum = 0;
    u_char* temp = (u_char*)this;
    for (int i = 0; i < 8; i++) {
        sum += (temp[i << 1] << 8) + temp[i << 1 | 1];
        while (sum >= 0x10000) {
            // 溢出
            int t = sum >> 16; // 计算方法与设置校验和相同
            sum += t;
        }
    }
    // 把计算出来的校验和和报文中该字段的值相加，如果等于0xffff，则校
    if (checksum + (u_short)sum == 65535)
        return false;
```

```
    return true;
}
```

六、 发送文件

如果客户端发送的是带有 FIN 字段的报文，那么进入挥手模式。

如果发送的是带有 START 字段、内容是文件名的第一个数据包，那么以二进制方式打开文件（因为要发送的文件包含图片等非文本文件）。然后将文件拆成若干个 1024Bytes 大小的数据包，顺序发送出去，最后一个数据包要加上 END 字段。代码如下：

```
void send_file() {
    Message sendMsg, recvMsg;
    clock_t start, end;
    char filePath[20];
    ifstream in;
    int filePtrLoc;
    int dataAmount;
    int packetNum;
    int checksum;

    cout << "请输入要发送的文件名: ";
    memset(filePath, 0, 20);
    string temp;
    cin >> temp;
    string inputPath = "./input/" + temp;

    if (temp == "quit") {
        closeConnect();
        quit = true;
        return;
    }
    else if (temp == "1.jpg" || temp == "2.jpg" || temp == "3.jpg" || temp ==
        "helloworld.txt") {
        strcpy_s(filePath, sizeof(filePath), temp.c_str());
        in.open(inputPath, ifstream::in | ios::binary);// 以读取模式、二进制
            方式打开文件
        in.seekg(0, ios_base::end);// 将文件流指针移动到文件的末尾
        dataAmount = in.tellg();//文件大小（以字节为单位）
        filePtrLoc = dataAmount;
        packetNum = filePtrLoc / 1024 + 1;//数据包数量
        in.seekg(0, ios_base::beg);// 将文件流指针移回文件的开头
        cout << "文件" << temp << "有" << packetNum << "个数据包" << endl;
    }
    else {
        cout << "文件不存在，请重新输入您要传输的文件名！" << endl;
        return;
    }
```

```
36
37      // 发送第一个包，内容是文件名
38      cout << "客户端发送文件名" << endl;
39      memcpy(sendMsg.data, filePath, strlen(filePath));
40      sendMsg.setSTART();
41      sendMsg.seq = 0;
42      sendMsg.len = strlen(filePath);
43      sendMsg.num = packetNum;
44      sendMsg.setChecksum();
45      cout << "checksum: " << sendMsg.checksum << endl;
46      if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
            serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
47          cout << "客户端发送文件名失败!" << endl;
48          return;
49      }
50
51      start = clock();
52      while (1) {
53          if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
                serverAddr, &len) != SOCKET_ERROR) {
54              if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
                    .packetCorruption()) {
55                  cout << "客户端发送文件名成功!" << endl;
56                  break;
57              }
58          }
59          if (clock() - start > RTO) {
60              cout << "应答超时，客户端重新发送文件名" << endl;
61              sendMsg.setChecksum();
62              cout << "checksum: " << sendMsg.checksum << endl << endl;
63              if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                    serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
64                  cout << "客户端发送文件名失败!" << endl;
65                  cout << "当前网络状态不佳，请稍后再试" << endl;
66                  return;
67              }
68              start = clock();
69          }
70      }
71
72      // 开始发送文件内容
73      cout << "客户端开始发送文件内容! " << endl;
74      int seq = 1;
75      start = clock();
76      for (int i = 0; i < packetNum; i++) {
77          if (i == packetNum - 1) {
78              in.read(sendMsg.data, filePtrLoc);
79              sendMsg.seq = seq;
```

```
80              sendMsg.len = filePtrLoc;
81              sendMsg.setEND(); // 文件结束标志
82              filePtrLoc = 0;
83          }
84          else {
85              in.read(sendMsg.data, 1024);// 读取文件数据
86              sendMsg.seq = seq;
87              sendMsg.len = 1024;
88              filePtrLoc -= 1024;
89          }
90
91          // 发送数据包
92          sendMsg.seq = seq;
93          sendMsg.setChecksum();
94          cout << "checksum: " << sendMsg.checksum << endl << endl;
95          cout << "发送seq为" << seq << "的数据包" << endl;
96          if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
97              cout << "发送数据包失败!" << endl;
98          }
99
100         // 设置套接字为非阻塞模式
101         int mode = 1;
102         ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & mode);
103         int count = 0;
104
105         clock_t c = clock();
106         while (1) {
107             // 尝试接收ack
108             if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR
                    *)&serverAddr, &len)) {
109                 if (recvMsg.isACK() && recvMsg.ack == seq && !recvMsg.
                        packetCorruption()) {
110                     break;
111                 }
112             }
113
114             // 检查是否超时
115             if (clock() - c > RTO) {
116                 cout << "应答超时, 重新发送数据包" << endl;
117                 if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (
                        SOCKADDR*)&serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
                         {
118                     cout << "发送数据包失败!" << endl;
119                 }
120                 count++;
121                 cout << "尝试重新发送seq为" << seq << "的数据包第" << count
                        << "次, 最多5次" << endl;
```

```
122                 if (count >= 5) {
123                     cout << "尝试次数超过5次，退出发送" << endl;
124                     return;
125                 }
126                 c = clock();
127             }
128             count = 0;
129             // 为了避免CPU占用率过高，添加延迟
130             Sleep(2);
131         }
132         seq++;
133     }
134     end = clock();
135     cout << "成功发送文件！" << endl;
136 
137     double TotalTime = (double)(end - start) / CLOCKS_PER_SEC;
138     cout << "传输总时间:␣" << TotalTime << "s" << endl;
139     cout << "吞吐率:␣" << (double)dataAmount / TotalTime << "␣bytes/s" <<
        endl << endl;
140 
141     // 关闭文件并准备发送下一个文件
142     in.close();
143     in.clear();
144 }
```

1. 停等机制

客户端发送数据包后，要进入一个循环等待直至收到服务器端的 ack 响应后，才能 break 跳出循环，继续发送下一个数据包。

```
1  // 发送数据包
2  sendMsg.seq = seq;
3  sendMsg.setChecksum();
4  cout << "checksum: " << sendMsg.checksum << endl << endl;
5  cout << "发送seq为" << seq << "的数据包" << endl;
6  if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
7      cout << "发送数据包失败!" << endl;
8  }
9  // 设置套接字为非阻塞模式
10 int mode = 1;
11 ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & mode);
12 int count = 0;
13 clock_t c = clock();
14 while (1) {
15     // 尝试接收ack
16     if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKA
17         if (recvMsg.isACK() && recvMsg.ack == seq && !recvMsg.pac
18             break;
```

```
        }
    }
    // 检查是否超时
    if (clock() - c > RTO) {
        cout << "应答超时，重新发送数据包" << endl;
        if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOC
            cout << "发送数据包失败!" << endl;
        }
        count++;
        cout << "尝试重新发送seq为" << seq << "的数据包第" << cou
        if (count >= 5) {
            cout << "尝试次数超过5次，退出发送" << endl;
            return;
        }
        c = clock();
    }
    count = 0;
    // 为了避免CPU占用率过高，添加延迟
    Sleep(2);
}
```

2. 超时重传

如果客户端在 2 * CLOCKS_PER_SEC 的时间内没有收到服务器端回复的 ACK 报文，就重新发送该数据包。

```
const int RTO = 2 * CLOCKS_PER_SEC;//超时重传时间

// 检查是否超时
if (clock() - c > RTO) {
    cout << "应答超时，重新发送数据包" << endl;
    if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOC
        cout << "发送数据包失败!" << endl;
    }
    count++;
    cout << "尝试重新发送seq为" << seq << "的数据包第" << cou
    if (count >= 5) {
        cout << "尝试次数超过5次，退出发送" << endl;
        return;
    }
    c = clock();
}
```

发送消息后使用 clock() 函数计时，如果时间超过 PTO 还没有收到服务器返回的 ack 响应，就会重新发送消息。此处设置了最大的重发次数为 5 次，重发 5 次后还没有收到服务器返回的 ack 响应，就认为连接已断开，并直接 return，避免消耗网络资源。

七、 接收文件

如果服务器端接收到 FIN 报文，说明客户端准备断开连接，进入挥手模式（需要注意的是，接收第一次挥手的消息已经在 `recv_file` 函数中处理了，closeConnect 函数只需处理剩下的三次挥手即可）。

如果服务器端接收到 START 报文，说明客户端准备开始发送文件，服务器端以二进制方式打开文件，并开始写入接收到的数据。

```
void recv_file() {
    cout << "服务器正在等待接收文件中......" << endl;
    Message recvMsg, sendMsg;
    clock_t start, end;
    char filePath[20];
    string outputPath;
    ofstream out;
    int dataAmount = 0;
    int packetNum;

    // 接收文件名
    while (1) {
        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            clientAddr, &len) != SOCKET_ERROR) {
            // 接收第一次挥手信息
            if (recvMsg.isFIN() && !recvMsg.packetCorruption()) {
                cout << "客户端准备断开连接! 进入挥手模式! " << endl;
                cout << "服务器端接收到第一次挥手消息! 第一次挥手成功!" <<
                    endl;
                closeConnect(recvMsg);
                quit = true;
                return;
            }
            if (recvMsg.isSTART() && !recvMsg.packetCorruption()) {
                ZeroMemory(filePath, 20);
                memcpy(filePath, recvMsg.data, recvMsg.len);
                outputPath = "./output/" + string(filePath);
                out.open(outputPath, ios::out | ios::binary);//以写入模式、二
                    进制模式打开文件
                cout << "文件名为: " << filePath << endl;
                cout << "checksum: " << recvMsg.checksum << endl << endl;

                if (!out.is_open()) {
                    cout << "文件打开失败! ! ! " << endl;
                    exit(1);
                }

                packetNum = recvMsg.num;
                cout << "文件" << filePath << "有" << packetNum << "个数据包"
                    << endl;
```

```
37
38                  // 发送ack给客户端
39                  sendMsg.setACK();
40                  sendMsg.ack = recvMsg.seq + 1;
41                  sendMsg.setChecksum();
42                  if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
                        SOCKADDR*)&clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
                         {
43                      cout << "服务器端发送ack报文失败!" << endl;
44                      cout << "当前网络状态不佳, 请稍后再试" << endl;
45                      return;
46                  }
47                  break;
48              }
49          }
50      }
51
52      // 设置套接字为阻塞模式
53      int mode = 0;
54      ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);
55
56      // 开始接收文件内容
57      cout << "服务器端开始接收文件内容! " << endl;
58      int expectedSeq = 1;
59      start = clock();
60      for (int i = 0; i < packetNum; i++) {
61          while (1) {
62              if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR
                    *)&clientAddr, &len) != SOCKET_ERROR) {
63                  // 检查序列号是否正确
64                  if (recvMsg.seq == expectedSeq && !recvMsg.packetCorruption()
                        ) {
65                      out.write(recvMsg.data, recvMsg.len);// 写入数据到文件
66                      dataAmount += recvMsg.len;
67                      out.close();
68
69                      // 发送ack给客户端
70                      cout << "收到seq为" << recvMsg.seq << "的数据包" << endl;
71                      cout << "checksum: " << recvMsg.checksum << endl << endl;
72                      sendMsg.setACK();
73                      sendMsg.ack = recvMsg.seq;
74                      sendMsg.setChecksum();
75                      if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
                            SOCKADDR*)&clientAddr, sizeof(SOCKADDR)) ==
                            SOCKET_ERROR) {
76                          cout << "服务器端发送ack报文失败!" << endl;
77                          cout << "当前网络状态不佳, 请稍后再试" << endl;
78                          return;
```

```
79                      }
80                      expectedSeq++;
81                  }
82                  // 检查文件传输是否结束
83                  if (recvMsg.isEND() && !recvMsg.packetCorruption()) {
84                      end = clock();
85                      cout << "接收文件成功! " << endl;
86                      out.close();
87                      out.clear();
88
89
90                      double TotalTime = (double)(end - start) / CLOCKS_PER_SEC
                            ;
91                      cout << "传输总时间" << TotalTime << "s" << endl;
92                      cout << "吞吐率" << (double)dataAmount / TotalTime << "␣
                            bytes/s" << endl << endl;
93
94                      return;
95                  }
96
97              }
98          }
99      }
100 }
```

1. 接收确认

当服务器端接受到来自客户端的数据包后，就发送 ACK 报文给客户端。代码如下:

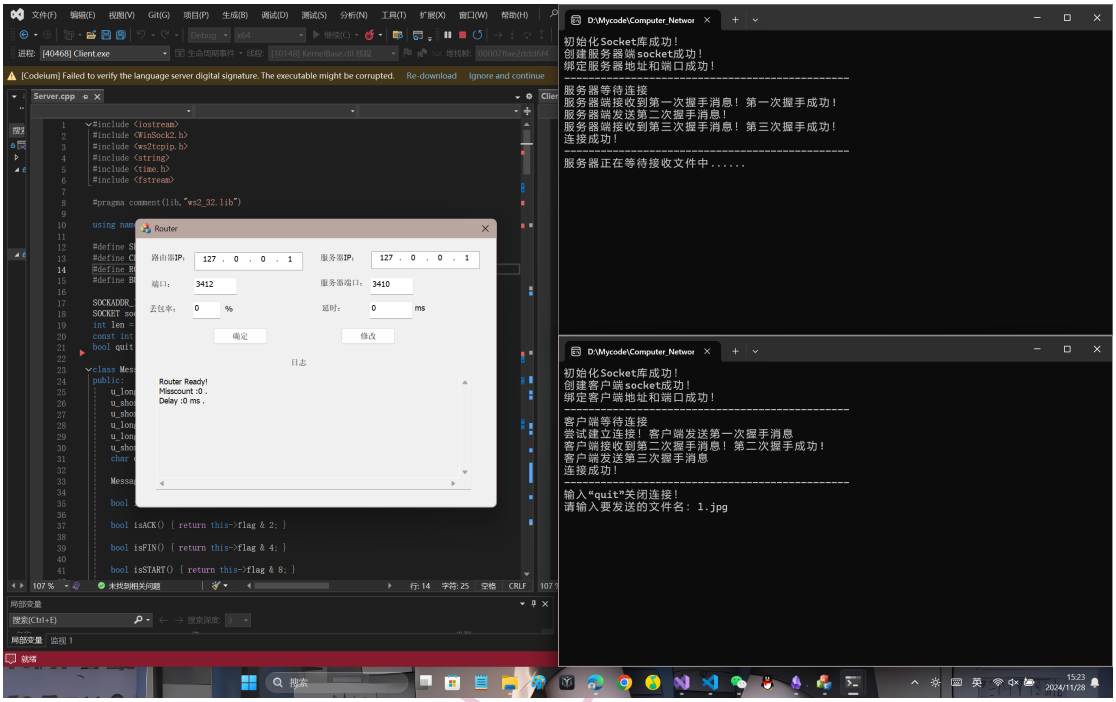
```
1  // 检查序列号是否正确
2  if (recvMsg.seq == expectedSeq && !recvMsg.packetCorrupti
3      out.write(recvMsg.data, recvMsg.len);// 写入数据到文
4      dataAmount += recvMsg.len;
5      out.close();
6      // 发送ack给客户端
7      cout << "收到seq为" << recvMsg.seq << "的数据包" << e
8      cout << "checksum: " << recvMsg.checksum << endl << e
9      sendMsg.setACK();
10     sendMsg.ack = recvMsg.seq;
11     sendMsg.setChecksum();
12     if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0,
13         cout << "服务器端发送ack报文失败!" << endl;
14         cout << "当前网络状态不佳, 请稍后再试" << endl;
15         return;
16     }
17     expectedSeq++;
18 }
```

八、 测试

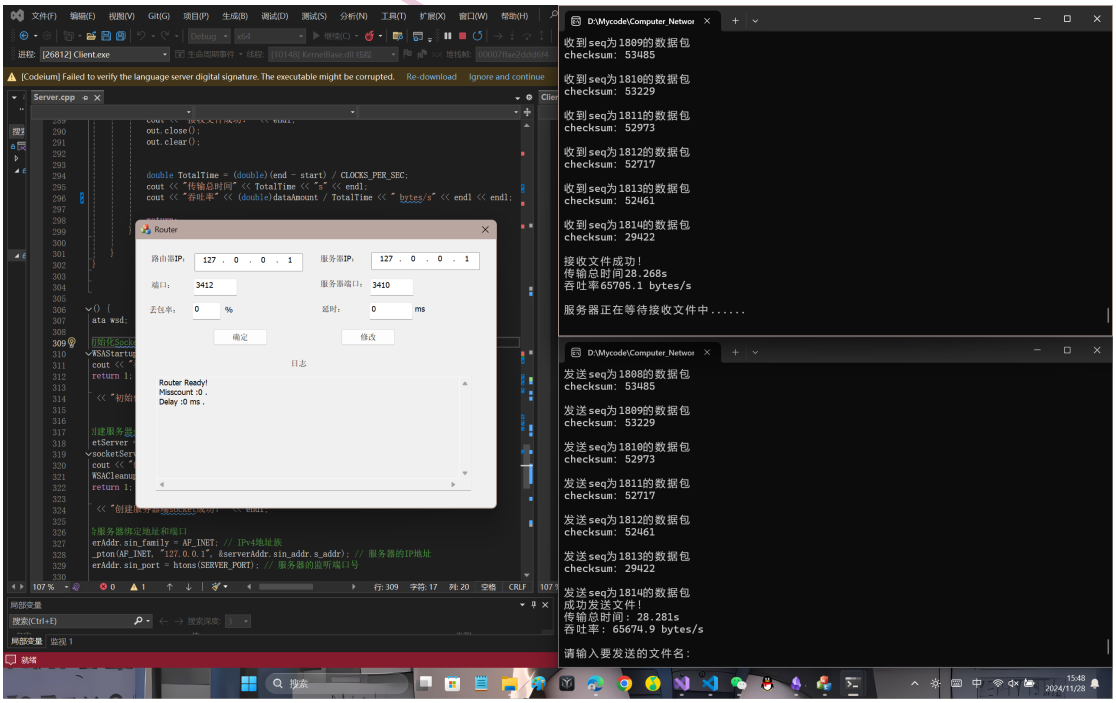
我们分别定义了服务器端、客户端、router 的端口号，IP 统一使用 127.0.0.1。

```
#define SERVER_PORT 3410
#define CLIENT_PORT 3411
#define ROUTER_PORT 3412
```

打开路由程序，将丢包率和延迟均设为 0%，开始运行，首先建立连接

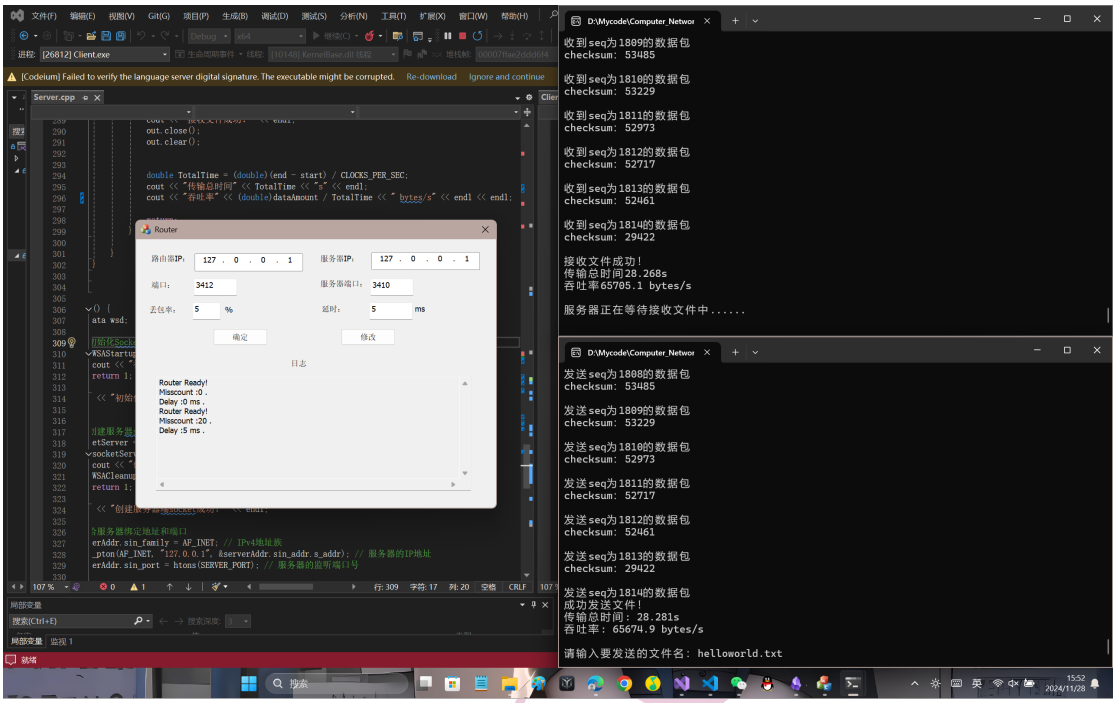


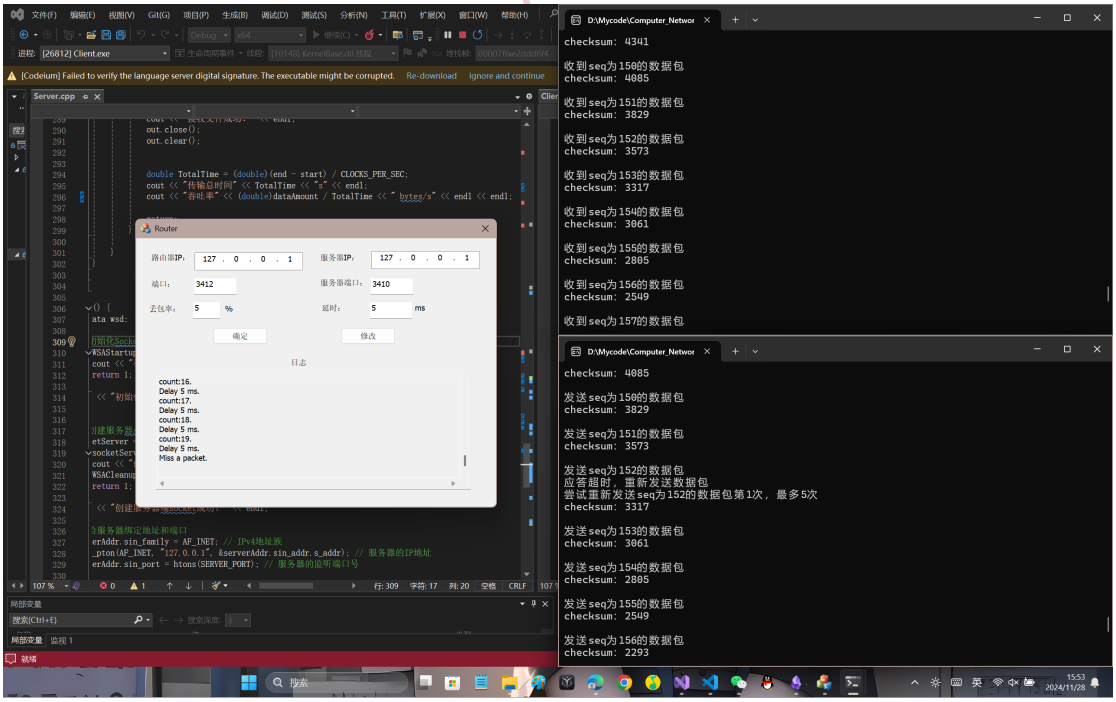
接下来发送文件“1.jpg”



文件“1.jpg”传输成功。

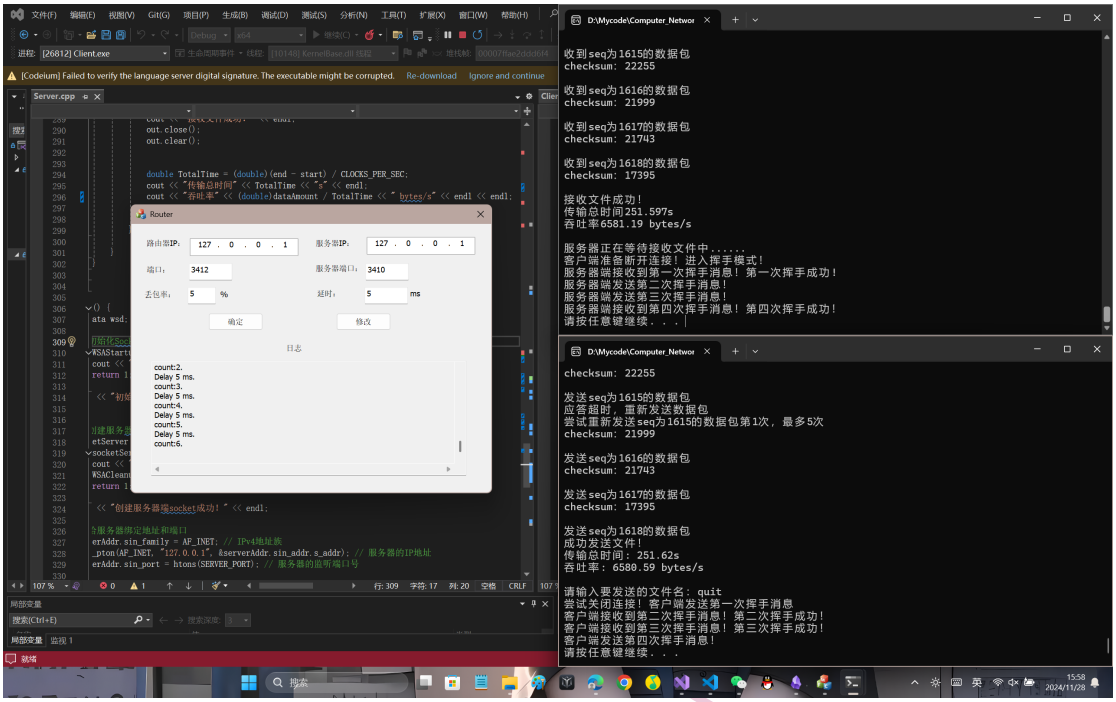
接下来测试超时重传，打开路由程序，将丢包率和延迟修改为为 5%, 开始运行





客户端发送的过程中，如果出现丢包，客户端会尝试重新发送报文。

断开连接：


Router
路由器IP: 127 . 0 . 0 . 1
服务器IP: 127 . 0 . 0 . 1
端口: 3412
服务器端口: 3410
丢包率: 5 %
延时: 5 ms
确定
修改
日志
count:2.
Delay 5 ms.
count:3.
Delay 5 ms.
count:4.
Delay 5 ms.
count:5.
Delay 5 ms.
count:6.
收到seq为1615的数据包
checksum: 22255
收到seq为1616的数据包
checksum: 21999
收到seq为1617的数据包
checksum: 21743
收到seq为1618的数据包
checksum: 17395
接收文件成功!
传输总时间251.597s
吞吐率6581.19 bytes/s
服务器正在等待接收文件中.....
客户端准备断开连接! 进入挥手模式!
服务器端接收到第一次挥手消息! 第一次挥手成功!
服务器端发送第二次挥手消息!
服务器端发送第三次挥手消息!
服务器端接收到第四次挥手消息! 第四次挥手成功!
请按任意键继续. . .
checksum: 22255
发送seq为1615的数据包
应答超时, 重新发送数据包
尝试重新发送seq为1615的数据包第1次, 最多5次
checksum: 21999
发送seq为1616的数据包
checksum: 21743
发送seq为1617的数据包
checksum: 17395
发送seq为1618的数据包
成功发送文件!
传输总时间: 251.62s
吞吐率: 6580.59 bytes/s
请输入要发送的文件名: quit
尝试关闭连接! 客户端发送第一次挥手消息
客户端接收到第二次挥手消息! 第二次挥手成功!
客户端接收到第三次挥手消息! 第三次挥手成功!
客户端发送第四次挥手消息!
请按任意键继续. . .