



南开大学
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

计算机网络实验报告

实验 3-1

张政泽

年级：2022 级

专业：信息安全

学号：2213573

2024 年 11 月 30 日

目录

| | |
|------------------------------|-----------|
| 一、 实验要求 | 1 |
| 二、 协议设计 | 1 |
| (一) 数据包格式 | 1 |
| (二) 发送端与接收端完整的交互过程 | 2 |
| 1. 三次握手建立连接 | 2 |
| 2. 数据传输过程 | 4 |
| 3. 四次挥手关闭连接 | 5 |
| (三) 差错检测与恢复机制 | 7 |
| (四) 流量控制及超时重传 | 7 |
| 三、 代码总体结构及分析 | 7 |
| (一) 完整源代码 | 7 |
| (二) 函数说明 | 37 |
| 1. 发送端 | 37 |
| 2. 接收端 | 38 |
| (三) 建立连接过程 | 39 |
| (四) 数据传输过程 | 40 |
| (五) 断开连接过程 | 44 |
| 四、 结果展示 | 46 |
| 五、 遇到的问题、收获及心得体会 | 51 |

一、 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、接受确认、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、 协议设计

(一) 数据包格式

数据包结构体设计如下：

```
//数据包结构
struct Packet {
    char flag; // 标志位，用于控制连接建立和断开----0001:
    DWORD SendIP, RecvIP; // 发送和接收的IP地址
    u_short SendPort, RecvPort; // 发送和接收的端口号
    int msgseq; // 消息序列号，用于确认和重传
    int ackseq; // 确认序列号，确认收到的消息
    int filelength; // 文件长度
    u_short checksum; // 校验和，用于差错检测
    char msg[MAX_BUFFER_SIZE]; // 数据内容
};
```

在数据包的首部包括 flag 标志位、SendIP 发送方 IP 地址、RecvIP 接收方 IP 地址、SendPort 发送方端口号、RecvPort 接收方端口号、msgseq 消息序列号、ackseq 确认序列号、filelength 数据部分的长度、checksum 校验和字段。

其中 flag 标志位又详细地分为以下几种情况：

1. 0x01: 表示 SYN-> 建立连接请求
2. 0x02: 表示 ACK-> 确认数据
3. 0x04: 表示包内包含具体数据
4. 0x08: 表示 FIN-> 关闭连接请求
5. 0x10: 表示传输的内容为文件名，主要目的是通知对方文件的具体格式，每次在发送文件具体内容前都会先告知对方文件名及文件类型
6. 0x21: 表示空包，用于超时重传

```
#define MAX_BUFFER_SIZE 1024
#define MAX_TIMEOUT 1000
#define RouterPort 8088
```

数据字段 msg 存放具体的内容,数据段大小被设置为 1024 字节,超时时间设置为 1000ms(1s),目的端口设置为 8088(实际服务器端口使用的是 8078,在此处设置为 8088 旨在使用路由程序进行丢包和延时的测试).

(二) 发送端与接收端完整的交互过程

为了使用数据报套接字实现可靠数据传输,选择采用类似 TCP 中三次握手四次挥手的方式来建立连接和关闭连接。

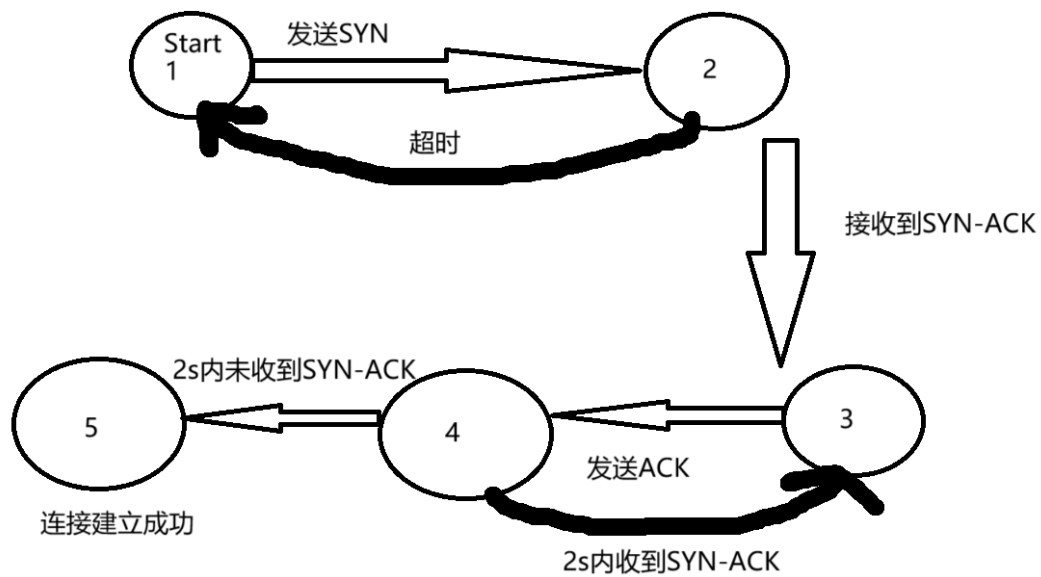
1. 三次握手建立连接

类似于 TCP 的三次握手过程,建立连接的具体流程如下:

客户端:

1. 客户端发送 SYN 包,提出建立连接的请求同时启动定时器,接着等待服务器回复 SYN-ACK 包,由于此 SYN 可能丢失,因此若等待接收 SYN-ACK 包的时间,超过了超时时间,那么就会再次发送一个 SYN(超时重传),直至接收到 SYN-ACK 包.
2. 在接收到服务器发送的 SYN-ACK 包后,这时客户端需要发送一个 ACK 包,在 TCP 三次握手中客户端发送完 ACK 包后客户端便认为连接建立完成,但实际上此 ACK 包也可能会丢失,为了解决这个问题,设计在客户端发送完 ACK 包后会等待两个超时时间 (2s),若在此期间,客户端没有收到服务端发来的重复 SYN-ACK 包,则说明此 ACK 包被对方成功接收,连接建立成功;相反,若在此期间收到了重复的 SYN-ACK 包,则说明发送的 ACK 包丢失需要重新发送 ACK 包.

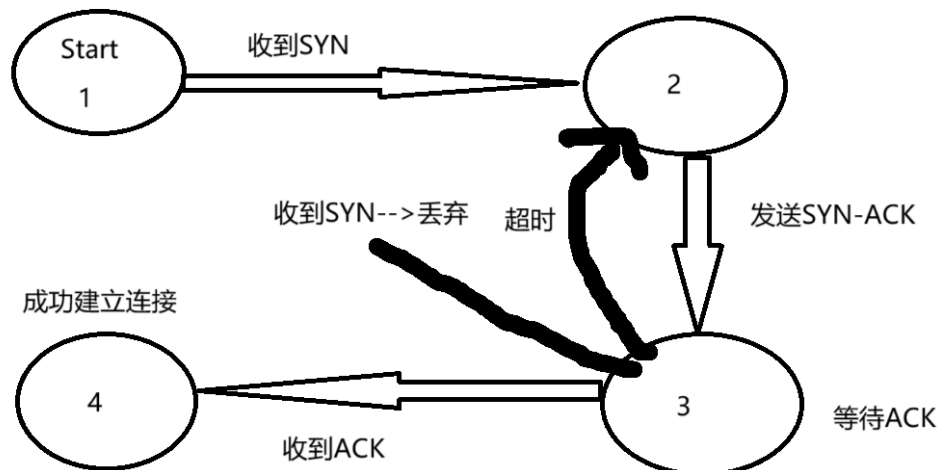
客户端对应的状态转换图如下图所示:



服务端:

1. 服务端启动后便进入等待状态，持续等待客户端发来的 SYN 包建立连接请求。
2. 服务端收到 SYN 包后需要回复对方 SYN-ACK 包同时启动定时器，紧接着等待接收对方发来的 ACK 包。在这里，SYN-ACK 包同样存在丢失的可能，若 SYN-ACK 包丢失，那么服务端就会不断地收到客户端发来的 SYN 包建立连接请求，而此时服务端处于等待接收 ACK 包的状态，因此对于客户端发来的 SYN 包一概选择丢弃，直至定时器超时，这时服务端便会重新发送 SYN-ACK 包并重置定时器，接着再次进入等待 ACK 的状态。
3. 服务端成功接收到客户端发来的 ACK 包，则三次握手过程完成，连接建立成功。上述提到客户端通过在发送完 ACK 包后 2s 内不再次收到 SYN-ACK 包来保证 ACK 包成功被接收到，因此服务端在接收到 ACK 包后会相应地休眠 2s。

服务端对应的状态转换图如下图所示:



2. 数据传输过程

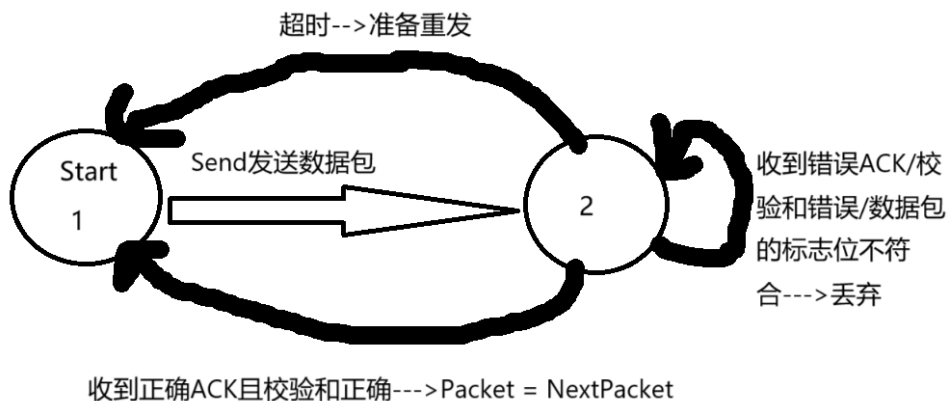
连接建立成功后，开始进行数据的传输。

由于在数据包中并未设计与传输文件类型相关的标志位，因此为了保证接收方能够正确地识别文件类型，需要在发送文件具体内容前额外发送一份数据包来告知对方文件的名称及类型信息，为此在数据包头部设计了标志位 (0x10) 来标识此时传输的数据包的目的是告知文件名称及类型信息。

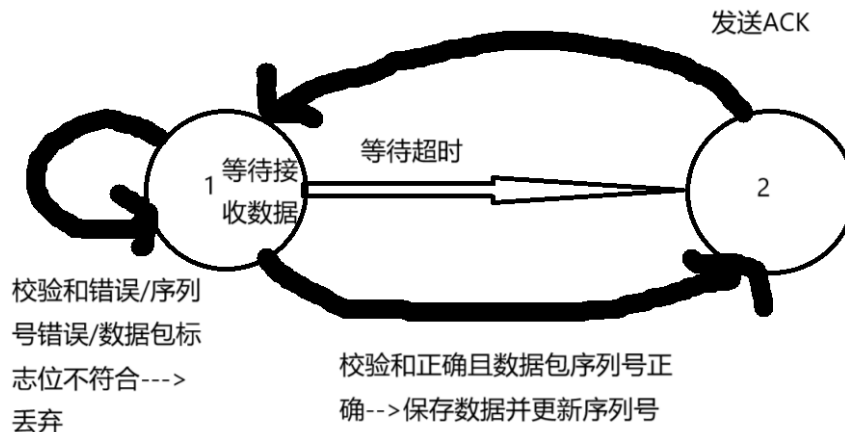
文件名及文件类型信息传递完毕后，开始进行文件内容的传输。

对于客户端，需要对传输的文件进行预处理工作，也就是将要传输的文件进行切分，切分的大小即数据包中 msg 字段的大小，为 1024 字节。

起初，客户端处于状态 1，发送数据包后，状态转换到状态 2，状态 2 实际上是一个等待状态。若在状态 2 下接收到错误 ACK/校验和错误/包的标志位错误，在这三种情况下均会选择将接收到的数据包丢弃，并继续等待接收；若在状态 2 下定时器超时，则回到状态 1，并准备重发数据包；若在状态 2 下收到正确 ACK 且校验和正确，则更新此时要发送的数据包为下一个数据包并回到状态 1，准备发送数据包。对应的状态转换图如下：



对于服务端，起初处于等待接收数据状态 1，若在状态 1 下收到数据包且序列号和校验和均正确，则此时将数据保存到文件中 (追加) 并更新此时的 ACK 序列号和 MSG 序列号；若在状态 1 下收到数据但校验和错误/序列号错误/标志位不符合，则丢弃此包并继续等待接收数据；若在状态 1 下定时器超时，这时会转到状态 2。在状态 2 下，根据此时的 ACK 序列号发送 ACK 包并启动定时器，接着转到状态 1。对应的状态转换图如下所示：



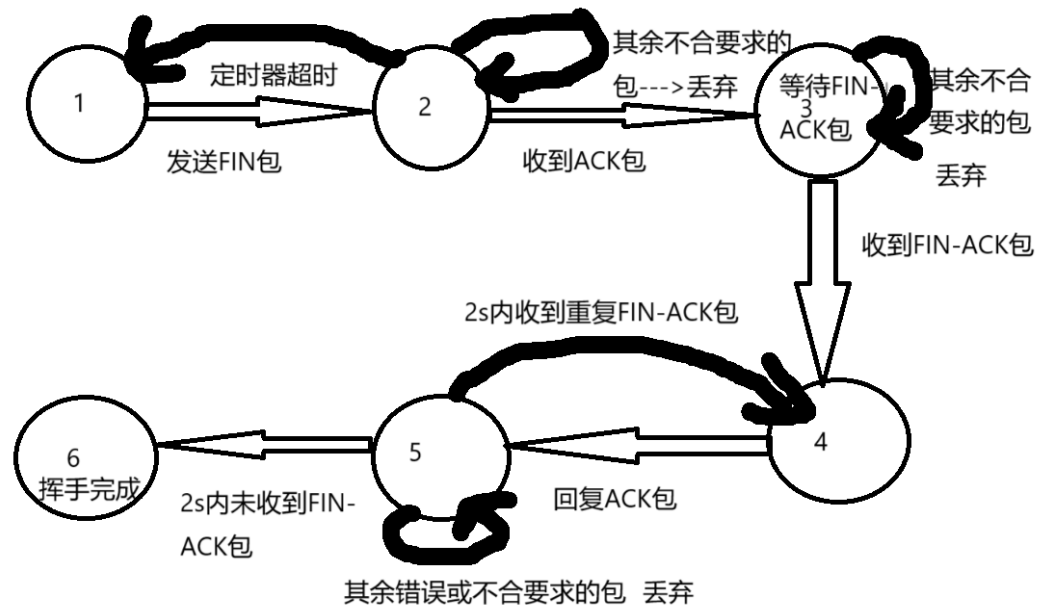
3. 四次挥手关闭连接

同样类似于 TCP 的四次挥手过程，但存在些许的差异。

客户端：

1. 在数据全部传输完毕后，客户端会发送一个 FIN 包并等待接收服务器发来的 ACK 包。在这个等待 ACK 包的状态下，对于其余接收到的所有不符合要求的数据包均会丢弃，直至收到符合要求的数据包/超时，超时后重新发送 FIN 包。同样存在 FIN 包丢失的问题，解决方式为：若 FIN 包丢失，则服务器端无法收到此 FIN 包，则回复的数据包并不符合客户端这一方希望接收的数据包的要求，因此会被丢弃，接着等到定时器超时，客户端便会重新发送 FIN 包。
2. 发送 FIN 包并收到 ACK 包后，客户端开始等待接收 FIN-ACK 包。同样地，处于该状态下对于其他不属于 FIN-ACK 的数据包或是存在错误的数据包均会丢弃，直至收到正确的 FIN-ACK 包或对方 (服务端) 定时器超时进而重新发送 FIN-ACK 包。
3. 收到 FIN-ACK 包后，客户端需要回复一个 ACK 包，但这个回复的 ACK 包同样存在丢失的可能性，因此，采用和三次握手时一样的处理方式，在发送完 ACK 包后，会再次尝试接收 FIN-ACK 包，若在 2s(2 倍超时时间) 内再次收到 FIN-ACK 包则说明发送的 ACK 包丢失，这时需要重新发送 FIN-ACK 包；否则则认为 ACK 包成功被对方接收到，四次挥手过程完成。

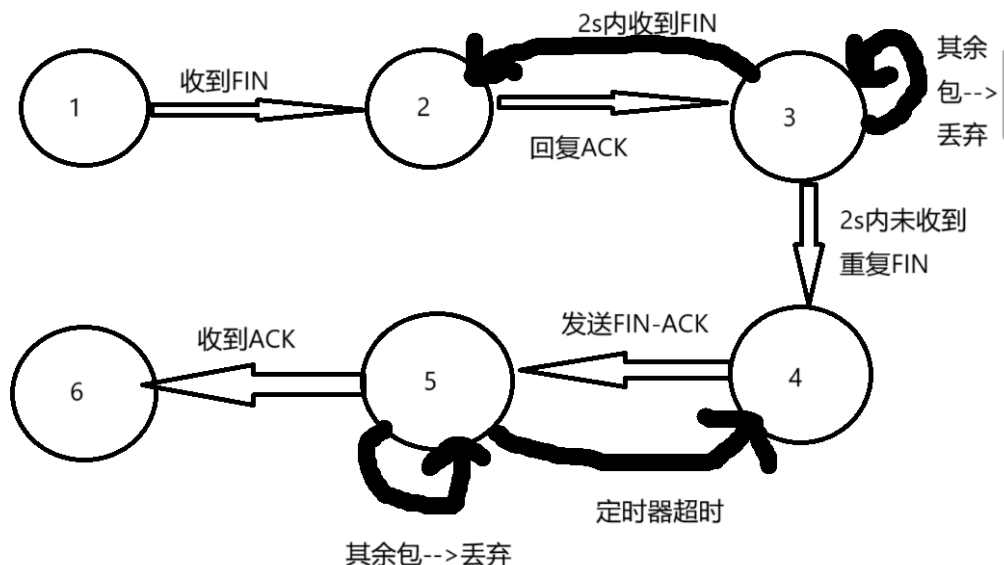
客户端对应的状态转换图如下图所示：



服务端:

1. 在服务端通过一个接收线程来接收数据包，当接收线程收到数据包并识别出是 FIN 数据包后，便会结束接收线程和发送线程，并回到主线程中处理挥手过程。
2. 在收到 FIN 包后，服务端需要回复一个 ACK 包，但同样的问题，这个 ACK 包可能会丢失，因此在发送完 ACK 包后，服务端会接着再次尝试接收 FIN 包，这个状态下只会接收 FIN 包，其余的数据包均会被丢弃。若再次接收到 FIN 数据包，意味着先前发送的 ACK 包丢失，紧接着会再次发送 ACK 数据包；若 2s(2 倍超时时间) 内未收到 FIN 包，则说明对方成功接收到 ACK 包。
3. 接着，服务端需要发送 FIN-ACK 包并等待接收 ACK. 在发送完 FIN-ACK 包后，进入等待 FinalACK 的状态，在此状态下，只会接收 ACK 包，对于其他的包均采用丢弃的方式，直至收到 ACK 包或定时器超时，超时后会重新发送 FIN-ACK 包。

服务端对应的状态转换图如下图所示:



(三) 差错检测与恢复机制

在上述数据包格式中指出存在校验和字段，校验和的计算方式如下：

对于发送端：首先将数据包首部信息补充完整，并将校验和字段清零，对于 msg 数据字段，不满足 1024 字节的数据包用 0 填充。接着将首部与数据段看成 16 位整数序列，进行 16 位二进制反码求和的运算，并将计算结果取反写入校验和域段。

对于接收端：接收端收到数据包后，采用类似的计算方式，将数据包看成 16 位整数序列，采用 16 位二进制反码求和运算，若计算结果为全 1，则通过查过检测；否则说明数据包存在差错。

当在差错检测过程中发现数据包存在数据错误时，将会丢弃该数据包，并接着接收下一组数据，直至收到没有差错且序列号正确的数据包或定时器超时，当定时器超时后，接收方便会重新发送上一次的 ACK 信息，当发送方收到此 ACK 信息后便可获知数据包传输存在问题，便会重新发送数据包。

(四) 流量控制及超时重传

流量控制机制用于控制和调节数据的发送速率以确保接收方能够有效地处理接收到的数据，防止网络拥塞。

在此次的设计中，流量控制采用停等机制，即发送方在发送完每个数据包后都会停止等待接收方确认，只有在接收到确认后才会发送下一个数据包。若在定时器超时后都没有收到确认信息/收到的确认信息与目标确认信息不一致，均会重新发送上一个数据包。

三、 代码总体结构及分析

(一) 完整源代码

数据发送

```
1 //实验3：基于UDP服务设计可靠传输协议并编程实现
2
```

```

3 //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
4 //实验要求:
5 //1.实现单向传输
6 //2.给出详细的协议设计
7 //3.完成给定测试文件的传输, 显示传输时间和平均吞吐率
8 //4.给出实现的拥塞控制算法的原理
9 //5.性能测试指标包括吞吐率、文件传输时延等, 给出图形结果并进行分析
10
11 #include <iostream>
12 #include <winsock2.h>
13 #include <Windows.h>
14 #include <stdio.h>
15 #include <cstdlib>
16 #include <cstring>
17 #include <vector>
18 #include <fstream>
19
20 using namespace std;
21
22 #define MAX_BUFFER_SIZE 1024
23 #define MAX_TIMEOUT 1000
24 #define RouterPort 8088
25
26
27 int msgseq; //全局变量, 用于记录下一个要发送的序列号
28 int expectedseq; //全局变量, 用于记录下一个期望接收的序列号
29 bool CanNext = false; //全局变量, 用于传递是否可以发送下一个数据包的信号
30 bool IsSendFINACK = false; //全局变量, 用于传递是否要发送FINACK的信号
31 bool IsTimeOut = false; //全局变量, 用于传递是否超时的信号
32 DWORD startTime = 99999999; // 记录开始时间
33 bool IsThreadOver = false;
34 DWORD FileSize;
35 HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); // 获取控制台句柄
36
37 //数据包结构
38 struct Packet {
39     char flag; // 标志位, 用于控制连接建立和断开——0001: 建立连接-SYN, 0010
        : 确认数据-ACK, 0100: 发送数据, 1000: 断开连接-FIN 0x21: 空包——!
        00010000: 文件名 11111111(0xff): 表示数据发送完毕
40     DWORD SendIP, RecvIP; // 发送和接收的IP地址
41     u_short SendPort, RecvPort; // 发送和接收的端口号
42     int msgseq; // 消息序列号, 用于确认和重传
43     int ackseq; // 确认序列号, 确认收到的消息
44     int filelength; // 文件长度
45     u_short checksum; // 校验和, 用于差错检测
46     char msg[MAX_BUFFER_SIZE]; // 数据内容
47 };

```

```

48 // 计算 Internet 校验和
49 uint16_t checksum(void *b, int len) {
50     uint32_t sum = 0;
51     uint16_t *buf = (uint16_t *)b;
52
53     while (len > 1) { //累加所有16位字
54         sum += *buf++;
55         len -= 2;
56     }
57     if (len == 1) { // 处理最后一个字节
58         sum += *(uint8_t *)buf;
59     }
60     while (sum >> 16) { // 处理进位
61         sum = (sum & 0xffff) + (sum >> 16);
62     }
63     return ~sum; // 返回反码
64 }
65
66 // 计算 Packet 的校验和
67 bool calculatePacketChecksum(Packet &packet) {
68     // 计算校验和
69     uint16_t cs = checksum(&packet, sizeof(Packet));
70     if(cs == 0x0000) return true;
71     else return false;
72 }
73
74 //三次握手建立连接——发送SYN
75 void ThreeHand_1(SOCKET socket, struct sockaddr_in serverAddr, int msgseq){
76     //模拟三次握手建立连接
77     Packet syn;
78     syn.flag = 0x01; // SYN
79     syn.msgseq = msgseq;
80     syn.ackseq = 0; //无Ack, 默认为0
81     syn.SendIP = inet_addr("127.0.0.1");
82     syn.RecvIP = inet_addr("127.0.0.1");
83     syn.SendPort = htons(0);
84     syn.RecvPort = htons(RouterPort);
85     syn.filelength = 0;
86     syn.checksum = 0;
87     memset(syn.msg, 0, sizeof(syn.msg));
88     syn.checksum = checksum(&syn, sizeof(syn));
89     sendto(socket, (char*)&syn, sizeof(syn), 0, (struct sockaddr*)&serverAddr,
90         sizeof(serverAddr));
91     SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
92     cout<<"[Checksum] ";
93     SetConsoleTextAttribute(hConsole, 0x07);
94     cout<<syn.checksum<<endl;
95 }

```

```

95  /// 三次握手建立连接——接收SYN-ACK
96  Packet ThreeHand_2(SOCKET socket, struct sockaddr_in serverAddr, int
    syn_msgSEQ){
97      bool SYN_ACK_Received = false;
98      Packet receivedsynAck;
99      int len = sizeof(serverAddr);
100     DWORD startTime = GetTickCount(); // 记录开始时间
101     while(!SYN_ACK_Received){
102         if(GetTickCount() - startTime > MAX_TIMEOUT){ // 超时了, 应该重传
            SYN
103             receivedsynAck.flag = 0x21; // 空包
104             return receivedsynAck;
105         }
106         int result = recvfrom(socket, (char*)&receivedsynAck, sizeof(
            receivedsynAck), 0, (struct sockaddr*)&serverAddr, &len);
107         if(result == SOCKET_ERROR){ // 没有数据可读
108             continue;
109         }
110         // 这里验证ACK和SYN是否正确
111         if(receivedsynAck.flag == 0x03 && receivedsynAck.ackseq == syn_msgSEQ
            ){
112             // 这里验证校验和是否正确
113             if(calculatePacketChecksum(receivedsynAck)==false){
114                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
115                 cout<<"[Error_Checksum]";
116                 SetConsoleTextAttribute(hConsole, 0x07);
117                 cout<<"ReceivedPacket_Checksum:"<<receivedsynAck.checksum<<
                    endl;
118                 continue;
119             }
120             else {
121                 SYN_ACK_Received = true;
122             }
123         }
124     }
125     else {
126         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
127         cout<<"[Error_Package]"<<endl;
128     }
129
130
131 }
132 return receivedsynAck;
133
134 }
135 /// 三次握手建立连接——发送ACK
136 void ThreeHand_3(SOCKET socket, struct sockaddr_in serverAddr, int syn_seq, int
    synAck_msgseq){

```

```

137
138     Packet ack;
139     ack.flag = 0x02; // ACK
140     ack.msgseq = syn_seq;
141     ack.ackseq = synAck_msgseq;
142     ack.SendIP = inet_addr("127.0.0.1");
143     ack.RecvIP = inet_addr("127.0.0.1");
144     ack.SendPort = htons(0);
145     ack.RecvPort = htons(RouterPort);
146     ack.filelength = 0;
147     ack.checksum = 0;
148     memset(ack.msg, 0, sizeof(ack.msg));
149     ack.checksum = checksum(&ack, sizeof(ack));
150     while(true){
151         int result = sendto(socket, (char*)&ack, sizeof(ack), 0, (struct
            sockaddr*)&serverAddr, sizeof(serverAddr));
152         if(result == SOCKET_ERROR){
153             SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
154             cout<<"[Error in sending ACK]"<<endl;
155             continue;
156         }
157         else{
158             break;
159         }
160     }
161 }
162
163
164 // 四次挥手断开连接——发送FIN
165 void FourHand_1(SOCKET socket, struct sockaddr_in serverAddr, int FIN_msgseq){
166
167     Packet finRequest;
168     finRequest.flag = 0x08; // FIN
169     finRequest.msgseq = FIN_msgseq;
170     finRequest.ackseq = 0; //无Ack, 默认为0
171     finRequest.SendIP = inet_addr("127.0.0.1");
172     finRequest.RecvIP = inet_addr("127.0.0.1");
173     finRequest.SendPort = htons(0);
174     finRequest.RecvPort = htons(RouterPort);
175     finRequest.filelength = 0;
176     finRequest.checksum = 0;
177     memset(finRequest.msg, 0, sizeof(finRequest.msg));
178     finRequest.checksum = checksum(&finRequest, sizeof(finRequest));
179
180     while(true){
181         int result = sendto(socket, (char*)&finRequest, sizeof(finRequest),
            0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
182         if(result == SOCKET_ERROR){

```

```

183         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
184         cout<<"[Error_in_sending_FIN]"<<endl;
185         continue;
186     }
187     else{
188         break;
189     }
190 }
191 }
192 // 四次挥手断开连接——接收ACK
193 Packet FourHand_2(SOCKET socket, struct sockaddr_in serverAddr, int ACK_seq){
194     bool ACK_Received = false;
195     Packet receivedACK;
196     int len = sizeof(serverAddr);
197     DWORD startTime = GetTickCount(); // 记录开始时间
198     while(!ACK_Received){
199         if(GetTickCount() - startTime > MAX_TIMEOUT){ // 超时
200             receivedACK.flag = 0x21;
201             return receivedACK;
202         }
203         int result = recvfrom(socket, (char*)&receivedACK, sizeof(receivedACK), 0, (struct sockaddr*)&serverAddr, &len);
204         if(result == SOCKET_ERROR){ // 没有数据可读
205             continue;
206         }
207         // 这里验证ACK及序列号是否正确
208         if(receivedACK.flag == 0x02 && receivedACK.ackseq == ACK_seq){
209             /// 验证校验和
210             if(calculatePacketChecksum(receivedACK) == true){
211                 ACK_Received = true;
212             }
213             else{
214                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
215                 cout<<"[Checksum_Error]";
216                 SetConsoleTextAttribute(hConsole, 0x07);
217                 cout<<"ReceivedACK_Checksum:"<<receivedACK.checksum<<endl;
218             }
219         }
220     }
221     else{
222         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
223         cout<<"[ERROR_PACKAGE]";
224         SetConsoleTextAttribute(hConsole, 0x07);
225         cout<<"ReceivedACK:"<<receivedACK.ackseq<<"ExpectedACK:"
226             <<ACK_seq<<"Receivedflag:"<<receivedACK.flag<<endl;
227         continue;
228     }

```

```

229     }
230     return receivedACK;
231 }
232
233
234 // 四次挥手断开连接——接收FIN-ACK-1010
235 Packet FourHand_3(SOCKET socket, struct sockaddr_in serverAddr, int msg_seq,
236     int ACK_seq){
237     bool FINACK_Received = false;
238     Packet receivedFINACK;
239     int len = sizeof(serverAddr);
240     DWORD startTime = GetTickCount(); // 记录开始时间
241     while(!FINACK_Received){
242         if(GetTickCount() - startTime > MAX_TIMEOUT){ // 超时
243             receivedFINACK.flag = 0x21;
244             return receivedFINACK;
245         }
246         int result = recvfrom(socket, (char*)&receivedFINACK, sizeof(
247             receivedFINACK), 0, (struct sockaddr*)&serverAddr, &len);
248         if(result == SOCKET_ERROR){ // 没有数据可读
249             continue;
250         }
251         // 这里验证ACK及序列号是否正确
252         if(receivedFINACK.flag == 0x0A && receivedFINACK.ackseq == ACK_seq &&
253             receivedFINACK.msgseq == msg_seq){
254             /// 验证校验和
255             if(calculatePacketChecksum(receivedFINACK) == true){
256                 FINACK_Received = true;
257             }
258             else{
259                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
260                 cout<<"[Checksum_Error]";
261                 SetConsoleTextAttribute(hConsole, 0x07);
262                 cout<<"receivedFINACK_Checksum:_"<<receivedFINACK.checksum<<
263                     endl;
264             }
265         }
266         else{
267             SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
268             cout<<"[ERROR_PACKAGE]";
269             SetConsoleTextAttribute(hConsole, 0x07);
270             cout<<"_receivedFINACK:_"<<receivedFINACK.ackseq<<"_
271                 ExpectedACK:_"<<ACK_seq<<"_Receivedflag: "<<receivedFINACK.
272                 flag<<endl;
273             continue;
274         }
275     }

```

```

271     }
272 }
273 return receivedFINACK;
274 }
275 //四次挥手断开连接——发送ACK
276 void FourHand_4(SOCKET socket, struct sockaddr_in serverAddr, int msgseq, int
    FIN_ACK_seq){
277     Packet ACK;
278     ACK.flag = 0x02; // ACK
279     ACK.msgseq = msgseq;
280     ACK.ackseq = FIN_ACK_seq; //无Ack, 默认为0
281     ACK.SendIP = inet_addr("127.0.0.1");
282     ACK.RecvIP = inet_addr("127.0.0.1");
283     ACK.SendPort = htons(0);
284     ACK.RecvPort = htons(RouterPort);
285     ACK.filelength = 0;
286     ACK.checksum = 0;
287     memset(ACK.msg, 0, sizeof(ACK.msg));
288     ACK.checksum = checksum(&ACK, sizeof(ACK));
289
290     while(true){
291         int result = sendto(socket, (char*)&ACK, sizeof(ACK), 0, (struct
            sockaddr*)&serverAddr, sizeof(serverAddr));
292         if(result == SOCKET_ERROR){
293             SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
294             cout<<"[Error in sending FIN]"<<endl;
295             continue;
296         }
297         else{
298             break;
299         }
300     }
301 }
302
303
304 struct ThreadParams{
305     SOCKET socket;
306     struct sockaddr_in ServerAddr;
307 };
308 DWORD WINAPI ReceiveACKThread(LPVOID lpParam) { //用于不断地接收ACK
309     ThreadParams params = *(ThreadParams*)lpParam;
310     SOCKET socket = params.socket;
311     struct sockaddr_in ServerAddr = params.ServerAddr;
312     int serverAddrSize = sizeof(ServerAddr);
313
314     u_long mode = 1; // 非阻塞模式
315     // 设置非阻塞模式
316     ioctlsocket(socket, FIONBIO, &mode);

```



```

317
318     Packet receivedPacket;
319
320     while (true) {
321         if(IsThreadOver == true){
322             break;
323         }
324         if(GetTickCount() - startTime > MAX_TIMEOUT){ //超时
325             //需要重发数据包DataPacket
326             IsTimeOut = true;
327             startTime = GetTickCount(); //重置开始时间
328             continue;
329         }
330
331
332         int result = recvfrom(socket, (char*)&receivedPacket, sizeof(Packet),
333                               0, (struct sockaddr*)&ServerAddr, &serverAddrSize);
334         if (result == SOCKET_ERROR) { //表示没数据可以接收
335             //cout << "Receive failed with error: " << WSAGetLastError() <<
336                 endl;
337         } else {
338             // 处理接收到的数据
339             //首先计算校验和, 错了丢弃, 对了继续
340             if(calculatePacketChecksum(receivedPacket) == false){
341                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
342                 cout << "[Checksum Error]";
343                 SetConsoleTextAttribute(hConsole, 0x07);
344                 cout << "Checksum: " << receivedPacket.checksum << endl;
345                 continue;
346             }
347             //到这里的话, 标志位是对的, 根据标志位判断包的类型
348             switch(receivedPacket.flag){
349                 case 0x02: //ACK——
350                     //收到ACK包后, 可以更新seq
351                     //首先验证一下, 这个包ACK的是不是我们上一个发送的数据包
352                     if(receivedPacket.ackseq != msgseq+1){
353                         //表示不是, 需要重发
354                         //通过手动超时的方式
355                         IsTimeOut = true;
356                         continue;
357                     }
358                     //到这个位置, 表示是, 可以发下一个了
359                     startTime = GetTickCount(); //重置定时器
360
361                     //首先, 收到ACK包说明之前发的数据包已经到达, 现在可以发送
362                     //下一个数据包
363                     msgseq = receivedPacket.ackseq; //更新seq, receivedPacket.

```

```
ackseq表示的是对方期望收到的下一个数据包的序列号
362 CanNext = true; //可以发送下一个数据包了
363 expectedseq = receivedPacket.msgseq + 1; //更新期望收到的
    下一个数据包的序列号
364
365     break;
366
367
368     default:
369     break;
370 }
371
372 }
373 }
374
375 return 0;
376 }
377
378 void readAndSplitFile(const char* filePath, std::vector<Packet>& packets,
    ThreadParams params) {
379
380     FILE* file = fopen(filePath, "rb"); // 以二进制读取模式打开文件
381     if (!file) {
382         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
383         std::cerr << "[Failed to open file]" << std::endl;
384         return;
385     } else {
386         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
387         std::cout << "[File Opened]" << std::endl;
388     }
389
390     char buffer[MAX_BUFFER_SIZE];
391     int seqNum = msgseq + 1;
392
393     // 先放进去msgseq个空包
394     for (int i = 0; i < msgseq; i++) {
395         Packet Data;
396         Data.flag = 0x04; // Data数据
397         Data.msgseq = -1; // 序列号，发完就加
398         Data.ackseq = 0; // 确认序列号——这里不需要
399         packets[i] = Data;
400     }
401
402     // 读取文件并分割成数据包
403     while (true) {
404         size_t bytesRead = fread(buffer, 1, MAX_BUFFER_SIZE, file); // 使用
            fread读取
405         if (bytesRead == 0) break; // 如果读取0字节，表示到达文件末尾或发生错
```

```

    误
406
407     Packet Data;
408     Data.flag = 0x04; // Data数据
409     Data.msgseq = seqNum; // 序列号, 发完就加
410     Data.ackseq = 0; // 确认序列号——这里不需要
411     Data.SendIP = inet_addr("127.0.0.1"); // 发送者的IP地址
412     Data.RecvIP = inet_addr("127.0.0.1"); // 接收者的IP地址
413     Data.SendPort = htons(0); // 发送者的端口号
414     Data.RecvPort = htons(RouterPort); // 接收者的端口号
415     Data.filelength = bytesRead; // 文件长度
416     Data.checksum = 0; // 校验和, 根据实际情况计算
417     memset(Data.msg, 0, sizeof(Data.msg)); // 数据内容初始化为空
418     memcpy(Data.msg, buffer, bytesRead); // 复制数据内容
419     Data.checksum = checksum(&Data, sizeof(Data)); // 计算校验和
420     packets[seqNum++] = Data;
421
422     FileSize += bytesRead;
423 }
424
425     fclose(file); // 关闭文件
426 }
427
428 int main(){
429
430     //SetConsoleOutputCP(65001); // 设置控制台输出编码为UTF-8
431     cout<<"Client_Start!!!"<<endl;
432     WSADATA wsaData;
433     u_long mode = 1; // 非阻塞模式
434     int result = WSAStartup(MAKEWORD(2, 2), &wsaData);
435     if (result != 0) {
436         printf("WSAStartup_failed:%d\n", result);
437         return 1;
438     }
439     // 创建UDP套接字
440     SOCKET socket = WSASocket(AF_INET, SOCK_DGRAM, IPPROTO_UDP, NULL, 0,
        WSA_FLAG_OVERLAPPED);
441     if (socket == INVALID_SOCKET) {
442         printf("Could_not_create_socket:%d\n", WSAGetLastError());
443         WSACleanup();
444         return 1;
445     }
446     cout<<"UDP_Create_Success!!!"<<endl;
447
448
449
450     // 设置非阻塞模式
451     ioctlsocket(socket, FIONBIO, &mode);

```

```

452
453 //定义服务器地址
454 struct sockaddr_in serverAddr;
455 serverAddr.sin_family = AF_INET;
456 serverAddr.sin_port = htons(RouterPort); // 服务器端口
457 serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1"); // 服务器IP地址
458
459 //初始化序列号
460 msgseq = 0;
461
462
463 Packet synAck;///接收SYN-ACK
464 // 发送SYN
465 while(true){
466     ThreeHand_1(socket, serverAddr, msgseq);///序列号是0
467     SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
468     cout<<"[SYN_Send]"<<endl;
469     SetConsoleTextAttribute(hConsole, 0x07);
470     cout<<"Waiting_for_SYN-ACK_package..."<<endl;
471
472     // 等待SYN-ACK
473     synAck = ThreeHand_2(socket, serverAddr, msgseq+1);
474     if(synAck.flag == 0x21){ //空包, 表示超时, 需要重传
475         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
476         cout<<"[SYN-ACK_Time_Out]"<<endl;
477         continue;///超时重传
478     }
479     else {
480         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
481         cout<<"[SYN-ACK_Receive_Success]"<<endl;
482         expectedseq = synAck.msgseq+1;
483         break;
484     }
485 }
486 ///在成功建立连接后, 服务端会Sleep6s, 也就是说若超时后未收到重复的SYN-ACK
    包, 说明连接建立成功;若收到重复SYN-ACK包, 则说明需要重传ACK包
487
488
489 msgseq++;
490 while(true){
491     ThreeHand_3(socket, serverAddr, msgseq, expectedseq);
492     SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
493     cout<<"[ACK_Send_Success]"<<endl;
494     SetConsoleTextAttribute(hConsole, 0x07);
495     cout<<"Waiting_for_2_seconds_to_ensure_that_the_connection_is_
        established..."<<endl;
496     ///等待2s若未收到重复的SYN-ACK包, 则认为连接建立成功
497     // 等待SYN-ACK

```

```
498     int Count = 0;
499     while(true){
500         Packet synAck2 = ThreeHand_2(socket, serverAddr, msgseq);
501         if(synAck2.flag == 0x21){ //表示超时, 超时2次就是2秒
502             Count++;
503             if(Count == 2)break;
504         }
505         else { //在2s内收到了SYN-ACK包, 说明连接建立失败, 需要重传ACK包
506             break;
507         }
508     }
509     if(Count == 2)break;
510     else {
511         continue; //这种情况表示在2s内收到了SYN-ACK包, 说明连接建立失
                    败, 需要重传ACK包
512     }
513 }
514 cout<<"-----Connection Established!!!-----"<<endl;
515 msgseq++;///连接建立成功后, 序列号再加1
516 //
517
518
519 // 创建线程参数结构体
520 ThreadParams params;
521 params.socket = socket;
522 params.ServerAddr = serverAddr;
523
524
525
526 // 读取文件并分割成数据包
527 std::vector<Packet> packets;
528
529 while(true){
530     char FileName[256]; // 假设文件名不超过 255 个字符
531     cout << "Please Input the File Name:(Press './exit' to exit)" << endl;
532     cin.getline(FileName, 256); // 使用 getline 以允许空格
533
534     if (strcmp(FileName, "./exit") == 0) {
535         cout << "Exiting..." << endl;
536         break;
537     }
538     const char* FileName2 = FileName;
539     cout<<"FileName:"<<FileName2<<endl;
540
541     packets.clear();
542     packets.resize(30000);
```

```

543 readAndSplitFile(FileName2, packets, params);
544
545 ///packets[msgseq] 存放一个只有文件名的包
546 Packet filenamepacket;
547 filenamepacket.flag = 0x10; // 文件名
548 filenamepacket.msgseq = msgseq; // 序列号, 发完就加
549 filenamepacket.ackseq = 0; // 确认序列号——这里不需要
550 filenamepacket.SendIP = inet_addr("127.0.0.1"); // 发送者的IP地址
551 filenamepacket.RecvIP = inet_addr("127.0.0.1"); // 接收者的IP地址
552 filenamepacket.SendPort = htons(0); // 发送者的端口号
553 filenamepacket.RecvPort = htons(RouterPort); // 接收者的端口号
554 filenamepacket.checksum = 0; // 校验和, 根据实际情况计算
555 memset(filenamepacket.msg, 0, sizeof(filenamepacket.msg)); // 数据内容初
    始化为空
556 memcpy(filenamepacket.msg, FileName, sizeof(FileName)); // 复制数据内容
557 filenamepacket.checksum = checksum(&filenamepacket, sizeof(filenamepacket
    )); // 计算校验和
558 packets[msgseq] = (filenamepacket);
559
560 // 创建一个线程用于接收ACK
561 IsThreadOver = false;
562 HANDLE hThread = CreateThread(NULL, 0, ReceiveACKThread, &params, 0, NULL
    );
563
564
565 //接下来是不断地传输数据
566 Packet packet;
567 packet = packets[msgseq];
568
569
570 DWORD FileTransmitTime = clock();
571
572 while(true){
573
574     if(CanNext == true){
575         packet = packets[msgseq];
576         CanNext = false;
577         if(packet.flag == 0x00){
578             //数据发送完毕, 准备关闭连接
579             packets.clear();
580             SetConsoleTextAttribute(hConsole, 0x07);
581             cout<<"Data_Send_Finish!!!"<<endl;
582             cout<<"FileName:"<<FileName2<<"\t\t\tFileTransmitTime:"<<clock()
                -FileTransmitTime<<"ms"<<endl;
583             cout<<"InOut_Rate:"<<msgseq*1024*CLOCKS_PER_SEC/(clock()-
                FileTransmitTime)<<"byte/s"<<endl;
584             IsThreadOver = true;
585             break;

```

```

586     }
587
588 }
589
590 // 发送数据包—可能需要重发
591 int sendResult = sendto(socket, (char*)&packet, sizeof(packet),
592     0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
593
594 IsTimeOut = false;
595 startTime = GetTickCount(); // 启动定时器
596 if (sendResult == SOCKET_ERROR) {
597     SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
598     cout<<"[Data_SendError]"<<endl;
599     continue;
600 }
601 else {
602     SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
603     cout<<"[Data_Send]";
604     SetConsoleTextAttribute(hConsole, 0x07);
605     cout<<"Packet_Seq:";
606     cout<<"msgseq:"<<msgseq<<endl;
607     cout<<packet.msgseq<<"Checksum:"<<packet.checksum<<endl;
608 }
609 while(1){
610     if(CanNext == true){
611         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
612         cout<<"[Preparing_for_Next_Data]"<<endl;
613         break;
614     }
615     else if(IsTimeOut == true){
616         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
617         cout<<"[Waiting_for_ACK_Time_Out]"<<endl;
618         break;
619     }
620 }
621 }
622 }
623 }
624 }
625
626 // 数据传输完毕，准备四次挥手关闭连接
627
628 // 发送FIN包并等待ACK包
629 Packet ACKPacket;
630 while(true){
631     FourHand_1(socket, serverAddr, msgseq);
632     // 等待接收ACK包

```

```

633     cout<<"[FIN_Send] ";
634     cout<<"msgseq: "<<msgseq<<endl;
635     cout<<"_Msg_Seq: "<<msgseq<<endl;
636     ACKPacket = FourHand_2(socket,serverAddr,msgseq+1);
637     if(ACKPacket.flag == 0x21){ ///空包, 表示超时, 需要重传
638         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
639         cout<<"[ACK_Recive_Time_Out]"<<endl;
640         continue;
641     }
642     else{
643         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
644         cout<<"[ACK_Recive_Success]"<<endl;
645         msgseq = msgseq+1;
646         expectedseq = ACKPacket.msgseq+1;
647         break;
648     }
649 }
650
651 //等待接收FIN-ACK包
652 Packet FINACKPacket;
653 cout<<"Waiting_for_FIN-ACK_Package!!!"<<endl;
654 while(true){
655     FINACKPacket = FourHand_3(socket,serverAddr,expectedseq,msgseq);
656     if(ACKPacket.flag == 0x21){ ///空包, 表示超时,这时候接着等待不需要重
        传
657         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
658         cout<<"[FIN-ACK_Recive_Time_Out]"<<endl;
659         continue;
660     }
661     else{
662         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
663         cout<<"[FIN-ACK_Recive_Success]"<<endl;
664         expectedseq = ACKPacket.msgseq+1;
665         break;
666     }
667 }
668
669 //回复一个ACK后关闭连接
670 while(true){
671     FourHand_4(socket,serverAddr,msgseq,expectedseq);
672     SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
673     cout<<"[ACK_Send] ";
674     cout<<"msgseq: "<<msgseq<<endl;
675     cout<<"_Msg_Seq: "<<msgseq<<endl;
676     SetConsoleTextAttribute(hConsole, 0x07);
677     cout<<"Waiting_for_2_seconds_to_ensure_that_the_ACK_package_has_been_
        received_by_the_server!!!"<<endl;
678     //等待2s, 若未收到重复的的FIN-ACK包, 则认为可以关闭连接

```



```

679     //等待FIN-ACK
680     int Count = 0;
681     while(true){
682         Packet FinAck = FourHand_3(socket,serverAddr,expectedseq-1,msgseq
        );
683         if(FinAck.flag == 0x21){ //表示超时
684             Count++;
685             if(Count == 2)break;
686         }
687         else{ //表示在2s内收到了FIN-ACK包,说明ACK包未接收到,需要重传
        ACK包
688             continue;
689         }
690     }
691     if(Count == 2)break;
692 }
693 SetConsoleTextAttribute(hConsole, 0x07);
694 cout<<"Client_Close_Successfully!!!"<<endl;
695 // 关闭套接字并清理
696 WSACleanup(); // 程序结束时清理WinSock
697 return 0;
698 }
699

```

数据接收端

```

1
2 #include <iostream>
3 #include <winsock2.h>
4 #include <Windows.h>
5 #include <stdio.h>
6 #include <cstdint>
7 #include <cstring>
8 #include <vector>
9 #include <fstream>
10
11
12 using namespace std;
13
14 #define MAX_BUFFER_SIZE 1024
15
16 #define TIMEOUT 1000 // 超时时间, 单位为毫秒
17
18
19 int msgseq; //全局变量, 用于记录下一个要发送的序列号
20 int expectedseq; //全局变量, 用于记录下一个期望接收的序列号
21 bool IsSendACK = false; //全局变量, 用于传递是否要发送ACK的信号
22 bool IsSendFINACK = false; //全局变量, 用于传递是否要发送FINACK的信号
23 DWORD startTime = 0; // 记录开始时间

```

```

24 string filename; // 文件名
25 bool IsThreadOver = false; // 线程是否结束
26 HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); // 获取控制台句柄
27
28
29 //数据包结构
30 struct Packet {
31     char flag; // 标志位, 用于控制连接建立和断开——0001: 建立连接-SYN,
        0010: 确认数据-ACK, 0100: 发送数据, 1000: 断开连接-FIN 0x21: 空包——!
        00010000(0x10): 文件名 11111111(0xff): 表示数据发送完毕
32     DWORD SendIP, RecvIP; // 发送和接收的IP地址
33     u_short SendPort, RecvPort; // 发送和接收的端口号
34     int msgseq; // 消息序列号, 用于确认和重传
35     int ackseq; // 确认序列号, 确认收到的消息
36     int filelength; // 文件长度
37     u_short checksum; // 校验和, 用于差错检测
38     char msg[MAX_BUFFER_SIZE]; // 数据内容
39 };
40 // 计算 Internet 校验和
41 uint16_t checksum(void *b, int len) {
42     uint32_t sum = 0;
43     uint16_t *buf = (uint16_t *)b;
44
45     while (len > 1) { // 累加所有16位字
46         sum += *buf++;
47         len -= 2;
48     }
49     if (len == 1) { // 处理最后一个字节
50         sum += *(uint8_t *)buf;
51     }
52     while (sum >> 16) { // 处理进位
53         sum = (sum & 0xffff) + (sum >> 16);
54     }
55     return ~sum; // 返回反码
56 }
57
58 // 计算 Packet 的校验和
59 bool calculatePacketChecksum(Packet &packet) {
60     // 计算校验和
61     uint16_t cs = checksum(&packet, sizeof(Packet));
62     if(cs == 0x0000) return true;
63     else return false;
64 }
65
66 //三次握手建立连接——接收SYN
67 Packet ThreeHand_1(SOCKET socket, struct sockaddr_in &clientAddr){
68     bool SYN_Received = false;
69     Packet receivedSYN;

```

```

70     int len = sizeof(clientAddr);
71     while(!SYN_Received){
72         int result = recvfrom(socket, (char*)&receivedSYN, sizeof(receivedSYN
73             ), 0, (struct sockaddr*)&clientAddr, &len);
74         if(result == SOCKET_ERROR){ //没有数据可读
75             continue;
76         }
77         //这里验证标志位SYN是否正确
78         if(receivedSYN.flag == 0x01){
79             //这里验证校验和是否正确
80             if(calculatePacketChecksum(receivedSYN)==false){
81                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
82                 cout<<"[Checksum_Error]";
83                 SetConsoleTextAttribute(hConsole, 0x07);
84                 cout<<"Waiting_for_a_new_SYN_packet."<<endl;
85                 continue;
86             }
87             else
88                 SYN_Received = true;
89         }
90     }
91     else {
92         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
93         cout<<"[Error_Package]";
94         SetConsoleTextAttribute(hConsole, 0x07);
95         cout<<"ReceivedSYN.flag"<<receivedSYN.flag<<endl;
96         continue;
97     }
98 }
99 return receivedSYN;
100 }
101 // 三次握手建立连接——发送SYN-ACK
102 void ThreeHand_2(SOCKET socket, struct sockaddr_in clientAddr, int syn_msgSEQ,
103     int msgseq){
104     Packet synAck;
105     synAck.flag = 0x03; // SYN-ACK
106     synAck.msgseq = msgseq;
107     synAck.ackseq = syn_msgSEQ;
108     synAck.SendIP = inet_addr("127.0.0.1");
109     synAck.RecvIP = inet_addr("127.0.0.1");
110     synAck.SendPort = htons(8078);
111     synAck.RecvPort = htons(clientAddr.sin_port);
112     synAck.filelength = 0;
113     synAck.checksum = 0;
114     memset(synAck.msg, 0, sizeof(synAck.msg));
115     synAck.checksum = checksum(&synAck, sizeof(synAck));
116     while(true){
117         int result = sendto(socket, (char*)&synAck, sizeof(synAck), 0, (

```

```

116         struct sockaddr*)&clientAddr, sizeof(clientAddr));
117     if(result == SOCKET_ERROR){
118         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
119         cout<<"[Send_SYN-ACK_Error]"<<endl;
120         continue;
121     }
122     else return;
123 }
124 }
125 /// 三次握手建立连接——接收ACK
126 Packet ThreeHand_3(SOCKET socket, struct sockaddr_in clientAddr, int ACK_SEQ){
127     bool ACK_Received = false;
128     Packet receivedACK;
129     int len = sizeof(clientAddr);
130     DWORD startTime = GetTickCount();
131     while(!ACK_Received){
132         if(GetTickCount() - startTime > TIMEOUT){
133             receivedACK.flag = 0x21; // 超时标志
134             SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
135             cout<<"[ACK_Received_Timeout]"<<endl;
136             return receivedACK;
137         }
138         int result = recvfrom(socket, (char*)&receivedACK, sizeof(receivedACK),
139                               0, (struct sockaddr*)&clientAddr, &len);
140         if(result == SOCKET_ERROR){ //没有数据可读
141             continue;
142         }
143         //这里验证标志位ACK及序列号是否正确
144         if(receivedACK.flag == 0x02 && receivedACK.ackseq == ACK_SEQ){
145             //这里验证校验和是否正确
146             if(calculatePacketChecksum(receivedACK)==false){
147                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
148                 cout<<"[Checksum_Error]";
149                 SetConsoleTextAttribute(hConsole, 0x07);
150                 cout<<"Waiting_for_a_new_ACK_packet."<<endl;
151                 continue;
152             }
153             else ACK_Received = true;
154         }
155         else {
156             SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
157             cout<<"[ERROR_Package]"<<endl;
158         }
159     }
160 }
161 }

```

```

162     return receivedACK;
163 }
164
165 // 四次挥手断开连接——接收FIN
166 Packet FourHand_1(SOCKET socket, struct sockaddr_in clientAddr, int
    expectedseq){
167
168     bool FIN_Received = false;
169     Packet receivedFIN;
170     int len = sizeof(clientAddr);
171     DWORD startTime = GetTickCount();
172     while(!FIN_Received){
173         if(GetTickCount() - startTime > TIMEOUT){
174             receivedFIN.flag = 0x21; // 超时标志
175
176             return receivedFIN;
177         }
178         int result = recvfrom(socket, (char*)&receivedFIN, sizeof(receivedFIN
            ), 0, (struct sockaddr*)&clientAddr, &len);
179         if(result == SOCKET_ERROR){ // 没有数据可读
180             continue;
181         }
182         // 这里验证标志位是否正确
183         if(receivedFIN.flag == 0x08 && receivedFIN.msgseq == expectedseq){
184             // 这里验证校验和是否正确
185             if(calculatePacketChecksum(receivedFIN) == false){
186                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
187                 cout<<"[Checksum_Error]";
188                 SetConsoleTextAttribute(hConsole, 0x07);
189                 cout<<"Waiting_for_a_new_FIN_packet."<<endl;
190                 continue;
191             }
192             else FIN_Received = true;
193         }
194     }
195     else {
196         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
197         cout<<"[ERROR_Package]";
198         SetConsoleTextAttribute(hConsole, 0x07);
199         cout<<"ReceivedPacket.msgseq"<<receivedFIN.msgseq<<"  "
            ExpectedPacket.msgseq"<<expectedseq<<endl;
200     }
201
202 }
203
204 return receivedFIN;
205 }
206 // 四次挥手断开连接——发送ACK

```

```

207 void FourHand_2(SOCKET socket, struct sockaddr_in clientAddr, int msgseq, int
    ACK_seq){
208     Packet ACK;
209     ACK.flag = 0x02; // ACK
210     ACK.msgseq = msgseq;
211     ACK.ackseq = ACK_seq; //无Ack, 默认为0
212     ACK.SendIP = inet_addr("127.0.0.1");
213     ACK.RecvIP = inet_addr("127.0.0.1");
214     ACK.SendPort = htons(8078);
215     ACK.RecvPort = htons(clientAddr.sin_port);
216     ACK.filelength = 0;
217     ACK.checksum = 0;
218     memset(ACK.msg, 0, sizeof(ACK.msg));
219     ACK.checksum = checksum(&ACK, sizeof(ACK));
220
221     while(true){
222         int result = sendto(socket, (char*)&ACK, sizeof(ACK), 0, (struct
            sockaddr*)&clientAddr, sizeof(clientAddr));
223         if(result == SOCKET_ERROR){
224             SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
225             cout<<"[Error_in_sending_FIN]"<<endl;
226             continue;
227         }
228         else{
229             break;
230         }
231     }
232 }
233
234 // 四次挥手断开连接——发送FIN-ACK
235 void FourHand_3(SOCKET socket, struct sockaddr_in clientAddr, int msgseq, int
    ACK_seq){
236     Packet FINACK;
237     FINACK.flag = 0x0A; // FINACK
238     FINACK.msgseq = msgseq;
239     FINACK.ackseq = ACK_seq; //无Ack, 默认为0
240     FINACK.SendIP = inet_addr("127.0.0.1");
241     FINACK.RecvIP = inet_addr("127.0.0.1");
242     FINACK.SendPort = htons(8078);
243     FINACK.RecvPort = htons(clientAddr.sin_port);
244     FINACK.filelength = 0;
245     FINACK.checksum = 0;
246     memset(FINACK.msg, 0, sizeof(FINACK.msg));
247     FINACK.checksum = checksum(&FINACK, sizeof(FINACK));
248
249     while(true){
250         int result = sendto(socket, (char*)&FINACK, sizeof(FINACK), 0, (
            struct sockaddr*)&clientAddr, sizeof(clientAddr));

```

```

251     if(result == SOCKET_ERROR){
252         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
253         cout<<"[Error_in_sending_FIN]"<<endl;
254         continue;
255     }
256     else{
257         break;
258     }
259 }
260 }
261 //四次挥手断开连接——接收ACK
262 Packet FourHand_4(SOCKET socket, struct sockaddr_in clientAddr, int msgseq, int
    ACK_seq){
263     bool ACK_Received = false;
264     Packet receivedACK;
265     int len = sizeof(clientAddr);
266     DWORD startTime = GetTickCount(); //记录开始时间
267     while(!ACK_Received){
268         if(GetTickCount() - startTime > TIMEOUT){ //超时
269             receivedACK.flag = 0x21;
270             return receivedACK;
271         }
272         int result = recvfrom(socket, (char*)&receivedACK, sizeof(receivedACK
            ), 0, (struct sockaddr*)&clientAddr, &len);
273         if(result == SOCKET_ERROR){ //没有数据可读
274             continue;
275         }
276         //这里验证ACK及序列号是否正确
277         if(receivedACK.flag == 0x02 && receivedACK.ackseq == ACK_seq &&
            receivedACK.msgseq == msgseq){
278             ///验证校验和
279             if(calculatePacketChecksum(receivedACK) == true){
280                 ACK_Received = true;
281             }
282             else{
283                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
284                 cout<<"[Checksum_Error]";
285                 SetConsoleTextAttribute(hConsole, 0x07);
286                 cout<<"Checksum:"<<receivedACK.checksum<<endl;
287             }
288         }
289     }
290     else{
291         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
292         cout<<"[ERROR_PACKAGE]";
293         SetConsoleTextAttribute(hConsole, 0x07);
294         cout<<"ReceivedPacket.msgseq"<<receivedACK.msgseq<<endl;
295         continue;

```

```

296     }
297
298 }
299 return receivedACK;
300 }
301
302 // 保存数据包到文件
303 void SavePacketToFile(const Packet& packet, const std::string& filename) {
304     FILE* file = fopen(filename.c_str(), "ab"); // 以追加和二进制模式打开文件
305     if (!file) {
306         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
307         std::cerr << "[Failed to open file for writing.]" << std::endl;
308         return;
309     }
310     size_t bytesWritten = fwrite(packet.msg, sizeof(char), packet.filelength,
311                                   file); // strlen(packet.msg)
312     if (bytesWritten < strlen(packet.msg)) {
313         // 如果写入的字节数小于要写入的字节数, 可能发生了错误
314         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
315         std::cerr << "[Error writing to file.]" << std::endl;
316     }
317     fclose(file); // 关闭文件
318 }
319
320 struct ThreadParams{
321     SOCKET socket;
322     struct sockaddr_in clientAddr;
323 };
324 DWORD WINAPI ReceiveDataThread(LPVOID lpParam) { //用于接收数据
325     ThreadParams params = *(ThreadParams*)lpParam;
326     SOCKET socket = params.socket;
327     struct sockaddr_in clientAddr = params.clientAddr;
328     int clientAddrSize = sizeof(clientAddr);
329
330     u_long mode = 1; // 非阻塞模式
331     // 设置非阻塞模式
332     ioctlsocket(socket, FIONBIO, &mode);
333
334     Packet receivedPacket;
335     char buffer[MAX_BUFFER_SIZE];
336
337     while (true) {
338         if(IsThreadOver == true) break;
339         if(GetTickCount() - startTime > TIMEOUT){ //超时
340             IsSendACK = true; //IsSendACK设置为true后会重发ACK
341             startTime = GetTickCount(); //重置开始时间
342         }

```



```
343
344     int result = recvfrom(socket, (char*)&receivedPacket, sizeof(Packet),
345                             0, (struct sockaddr*)&clientAddr, &clientAddrSize);
346
347     if (result == SOCKET_ERROR) { //表示没数据可以接收
348
349     } else {
350         // 处理接收到的数据
351         //首先计算校验和, 错了丢弃, 对了继续
352         if (calculatePacketChecksum(receivedPacket) == false){
353             SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
354             cout<<"[Checksum_Error]";
355             SetConsoleTextAttribute(hConsole, 0x07);
356             cout<<"Checksum:<<receivedPacket.checksum<<endl;
357             continue;
358         }
359         //到这里的话, 标志位是对的, 根据标志位判断包的类型——
360         //同时, 收到数据的话还需要取消定时器——在这里就通过将starttime设
361         //置为负数的方式来实现
362         switch(receivedPacket.flag){
363
364         case 0x04: //Data
365             SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
366             cout<<"[Data_Received]"<<endl;
367             //接收到了数据包, 说明服务端的上一个发的ACK成功收到了, 因
368             //此发送序号可以增加
369             //这里需要更新序列号, 但在更新序列号前, 需要判断是否需要
370             //重传——
371             if(receivedPacket.msgseq != expectedseq){ //真正收到的
372                 //与期望收到的不一致, 需要重发ACK
373                 IsSendACK = true; //IsSendACK设置为true后会重发ACK
374                 continue; //接着等待
375             }
376             //否则不需要重传
377
378             //这一部分为处理收到的数据的代码
379             SavePacketToFile(receivedPacket, "./COPY-"+filename); //
380             //保存数据包到文件
381
382             expectedseq = receivedPacket.msgseq + 1; //更新期望的序列
383             //号
384             //更新msgseq
385             msgseq = msgseq + 1;//////因为收到了数据包说明上一个ACK已
386             //经发送成功了
387             startTime = -1; //取消定时器
388             IsSendACK = true;
389             break;
390
391         case 0x08: //FIN
```

```
383 //收到FIN包，结束接收和发送线程，回到主线程完成四次挥手
384
385 expectedseq = receivedPacket.msgseq + 1; //更新期望的序列
    号
386
387 //更新msgseq
388 msgseq = msgseq + 1;/////因为收到了数据包说明上一个ACK已
    经发送成功了
389 IsThreadOver = true;
390
391 break;
392
393 case 0x10: //文件名
394     SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
395     cout<<"[Filename_Received]"<<endl;
396     //接收到了数据包，说明服务端的上一个发的ACK成功收到了，因
    此发送序号可以增加
397     //这里需要更新序列号，但在更新序列号前，需要判断是否需要
    重传
398     if(receivedPacket.msgseq != expectedseq){ //真正收到的
        与期望收到的不一致，需要重发ACK
399         IsSendACK = true; //IsSendACK设置为true后会重发ACK
400         continue; //接着等待
401     }
402     //否则不需要重传
403
404     //这一部分为处理收到的数据的代码
405     filename = receivedPacket.msg; // 保存文件名
406
407     expectedseq = receivedPacket.msgseq + 1; //更新期望的序列
        号
408     //更新msgseq
409     msgseq = msgseq + 1;/////因为收到了数据包说明上一个ACK已
        经发送成功了
410     startTime = -1; //取消定时器
411     IsSendACK = true;
412
413     break;
414
415     default:
416         break;
417 }
418
419 }
420
421 }
422
423 return 0;
```

```

424 }
425 Packet CreateACKPacket(int msgseq, int ackseq, ThreadParams* params){
426     SOCKET socket = params->socket;
427     struct sockaddr_in clientAddr = params->clientAddr;
428
429     Packet ACK;
430     ACK.flag = 0x02; // ACK标志位
431     ACK.msgseq = msgseq; // 序列号, 发完就加
432     ACK.ackseq = ackseq; // 确认序列号
433     ACK.SendIP = inet_addr("127.0.0.1"); // 发送者的IP地址
434     ACK.RecvIP = inet_addr("127.0.0.1"); // 接收者的IP地址
435     ACK.SendPort = htons(8078); // 发送者的端口号
436     ACK.RecvPort = clientAddr.sin_port; // 接收者的端口号
437     ACK.checksum = 0; // 校验和, 根据实际情况计算
438     memset(ACK.msg, 0, sizeof(ACK.msg)); // 数据内容初始化为空
439     ACK.checksum = checksum(&ACK, sizeof(ACK)); // 计算校验和
440
441     return ACK;
442 }
443 }
444
445 DWORD WINAPI SendDataThread(LPVOID lpParam) {
446     // 从 lpParam 中取出参数
447     ThreadParams* params = (ThreadParams*)lpParam;
448     SOCKET socket = params->socket;
449     struct sockaddr_in clientAddr = params->clientAddr;
450     while(true){ //ACK需要超时重传
451         if(IsThreadOver == true)break;
452         if(IsSendACK == true){
453             // 准备要发送的数据包
454             Packet packet = CreateACKPacket(msgseq, expectedseq, params);
455             // 发送数据
456             int result = sendto(socket, (char*)&packet, sizeof(packet), 0, (
                struct sockaddr*)&clientAddr, sizeof(clientAddr));
457             startTime = GetTickCount(); // 记录开始时间
458             if (result == SOCKET_ERROR) {
459                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
460                 cout << "[Send failed]error: ";
461                 SetConsoleTextAttribute(hConsole, 0x07);
462                 cout << WSAGetLastError() << endl;
463             } else {
464                 SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
465                 cout << "[ACK Send] ";
466                 SetConsoleTextAttribute(hConsole, 0x07);
467                 cout<<"\t\t\t\tmsgseq:"<<msgseq<< "\t\t\tACKSeq:"<<expectedseq<<endl
                    ;
468             }
469             IsSendACK = false;

```

```
470     }
471
472 }
473 return 0;
474 }
475
476 int main() {
477
478     //SetConsoleOutputCP(65001); // 设置控制台输出为UTF-8编码
479
480     WSADATA wsaData;
481     SOCKET socket;
482     struct sockaddr_in serverAddr, clientAddr;
483     int result;
484     u_long mode = 1; // 非阻塞模式
485     int seq;
486
487     // 初始化WinSock
488     result = WSASStartup(MAKEWORD(2, 2), &wsaData);
489     if (result != 0) {
490         printf("WSASStartup failed: %d\n", result);
491         return 1;
492     }
493
494     // 创建UDP套接字
495     socket = WSASocket(AF_INET, SOCK_DGRAM, IPPROTO_UDP, NULL, 0,
496         WSA_FLAG_OVERLAPPED);
497     if (socket == INVALID_SOCKET) {
498         printf("Could not create socket: %d\n", WSAGetLastError());
499         WSACleanup();
500         return 1;
501     }
502
503     // 设置非阻塞模式
504     ioctlsocket(socket, FIONBIO, &mode);
505
506     // 设置服务器地址信息
507     serverAddr.sin_family = AF_INET;
508     serverAddr.sin_port = htons(8078); // 绑定到端口8078
509     serverAddr.sin_addr.s_addr = INADDR_ANY; // 绑定到所有可用接口
510
511     // 绑定套接字
512     result = bind(socket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
513     if (result == SOCKET_ERROR) {
514         printf("Bind failed with error: %d\n", WSAGetLastError());
515         closesocket(socket);
516         WSACleanup();
517         return 1;
518     }
```

```
517     }
518
519     printf("UDP_server_bound_to_port%d\n", ntohs(serverAddr.sin_port));
520
521     cout<<"Waiting_for_connection..."<<endl;
522
523     //初始化序列号
524     msgseq = 0;////这里会一直等直到收到SYN包
525     Packet synPacket = ThreeHand_1(socket, clientAddr);
526     expectedseq = synPacket.msgseq+1;////ACK序列号
527     SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
528     cout<<"[SYN_Received]"<<endl;
529
530     Packet AckPacket;
531     //等待接收SYN包
532     while(true){
533         //发送SYN-ACK
534         ThreeHand_2(socket, clientAddr, expectedseq, msgseq);
535         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
536         cout<<"[SYN-ACK_Send]"<<endl;
537
538         //等待接收ACK
539         AckPacket = ThreeHand_3(socket, clientAddr, msgseq+1);
540         if(AckPacket.flag == 0x21){ //空包, 表示超时, 需要重传——超时会重
            新发送SYN-ACK
541             continue;
542         }
543         else {
544             SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
545             cout<<"[ACK_Received]"<<endl;
546             break;
547         }
548     }
549     Sleep(4000); //睡眠4s
550     msgseq++;
551     expectedseq = AckPacket.msgseq+1; //期望收到的下一个数据包的序列号
552     cout<<"-----Connection_Established!!!-----"<<endl;
553     //Ack序列号, 每次在成功接收到数据后+1
554     //seq序列号, 每次在收到ACK后+1
555
556     //从这里开始应该新建一个线程, 用于接收数据
557
558     // 创建线程参数结构体
559     ThreadParams params;
560     params.socket = socket;
561     params.clientAddr = clientAddr;
562
563     // 创建线程
```

```

564 HANDLE hThread = CreateThread(NULL, 0, ReceiveDataThread, &params, 0,
565 NULL);
566
567 //创建另一个线程用于发送ACK
568 HANDLE hThread2 = CreateThread(NULL, 0, SendDataThread, &params, 0, NULL)
569 ;
570
571 cout<<"Thread created Success!!!"<<endl;
572 while(true){
573     if(IsThreadOver == true)break;
574 }
575
576 //四次挥手关闭连接
577 SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
578 cout<<"[FIN Received]"<<endl;
579
580 while(true){
581     //发送ACK包
582     FourHand_2(socket, clientAddr, msgseq, expectedseq);
583     SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
584     cout<<"[ACK Send]"<<endl;
585     cout<<"Waiting for 2 seconds to ensure that the ACK package has been
586         received!!!"<<endl;
587
588     //等待接收FIN包
589     Packet finPacket = FourHand_1(socket, clientAddr, expectedseq-1);
590     if(finPacket.flag == 0x21){ //空包, 表示超时——表示没有收到重复FIN
591         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
592         cout<<"[Ensure->ACK Received]"<<endl;
593         break;
594     }
595     else {
596         //2s内收到了FIN包, 需要重传ACK包
597         SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
598         cout<<"[Receive FIN Package Again] Preparing to resend ACK Package
599             !!!"<<endl;
600         continue;
601     }
602 }
603
604 Packet finackPacket;
605 //发送FIN-ACK包并等待接收ACK包
606 while(true){
607     //发送FIN-ACK包
608     FourHand_3(socket, clientAddr, msgseq, expectedseq);
609     SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
610     cout<<"[FIN-ACK Send]"<<endl;
611

```

```

608     //等待接收ACK包
609     finackPacket = FourHand_4(socket, clientAddr, expectedseq, msgseq+1);
610     if(finackPacket.flag == 0x21){ //空包, 表示超时
611         continue;
612     }
613     else {
614         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
615         cout<<"[FinalACK_Received]"<<endl;
616         break;
617     }
618 }
619 SetConsoleTextAttribute(hConsole, 0x07);
620 cout<<"Connection_Closed_Successfully!!!"<<endl;
621
622 // 清理
623 closesocket(socket);
624 WSACleanup();
625 return 0;
626 }

```

(二) 函数说明

1. 发送端

```

//数据包结构
> struct Packet { ...
// 计算 Internet 校验和
> uint16_t checksum(void *b, int len) { ...

// 计算 Packet 的校验和
> bool calculatePacketChecksum(Packet &packet) { ...

//三次握手建立连接---发送SYN
> void ThreeHand_1(SOCKET socket, struct sockaddr_in serverAddr, int msgseq) { ...
/// 三次握手建立连接---接收SYN-ACK
> Packet ThreeHand_2(SOCKET socket, struct sockaddr_in serverAddr, int syn_msgseq) { ...
/// 三次握手建立连接---发送ACK
> void ThreeHand_3(SOCKET socket, struct sockaddr_in serverAddr, int syn_seq, int synAck_msgseq) { ...

// 四次挥手断开连接---发送FIN
> void FourHand_1(SOCKET socket, struct sockaddr_in serverAddr, int FIN_msgseq) { ...
// 四次挥手断开连接---接收ACK
> Packet FourHand_2(SOCKET socket, struct sockaddr_in serverAddr, int ACK_seq) { ...

// 四次挥手断开连接---接收FIN-ACK-1010
> Packet FourHand_3(SOCKET socket, struct sockaddr_in serverAddr, int msg_seq, int ACK_seq) { ...
//四次挥手断开连接---发送ACK
> void FourHand_4(SOCKET socket, struct sockaddr_in serverAddr, int msgseq, int FIN_ACK_seq) { ...

> struct ThreadParams { ...
> DWORD WINAPI ReceiveACKThread(LPVOID lpParam) { //用于不断地接收ACK ...

> void readAndSplitFile(const char* filePath, std::vector<Packet>& packets, ThreadParams params) { ...

> int main() { ...

```

在发送端的代码中编写的函数主要有

1. checksum: 计算校验和, 计算方式如协议中所述, 16 位为一组, 求和后计算反码.
2. calculatePacketChecksum: 验证接收到的包的校验和, 最后通过将校验和的计算结果与 0 比较, 若为全零则认为无差错, 函数返回 true; 否则返回 false.
3. 三次握手的函数: 分别用于发送 SYN、接收 SYN-ACK、发送 ACK
4. 四次挥手的函数: 分别用于发送 FIN、接收 ACK、接收 FIN-ACK、发送 ACK
5. 线程函数 ReceiveACKThread: 发送端需要接收接收端发来的 ACK, 通过此函数来处理。
6. readAndSplitFile: 协议中提到数据包中数据段的大小为 1024 字节, 由于发送的文件一般远远大于这个大小, 因此需要将其分割为多个数据包分别传输。

2. 接收端

```

28
29 //数据包结构
30 > struct Packet { ...
31 // 计算 Internet 校验和
32 > uint16_t checksum(void *b, int len) { ...
33
34 // 计算 Packet 的校验和
35 > bool calculatePacketChecksum(Packet &packet) { ...
36
37 //三次握手建立连接---接收SYN
38 > Packet ThreeHand_1(SOCKET socket, struct sockaddr_in &clientAddr){ ...
39 // 三次握手建立连接---发送SYN-ACK
40 > void ThreeHand_2(SOCKET socket, struct sockaddr_in clientAddr,int syn_msgSEQ,int msgseq){ ...
41 // 三次握手建立连接---接收ACK
42 > Packet ThreeHand_3(SOCKET socket, struct sockaddr_in clientAddr,int ACK_SEQ){ ...
43
44 // 四次挥手断开连接---接收FIN
45 > Packet FourHand_1(SOCKET socket, struct sockaddr_in clientAddr,int expectedseq){ ...
46 // 四次挥手断开连接---发送ACK
47 > void FourHand_2(SOCKET socket, struct sockaddr_in clientAddr,int msgseq,int ACK_seq){ ...
48
49 // 四次挥手断开连接---发送FIN-ACK
50 > void FourHand_3(SOCKET socket, struct sockaddr_in clientAddr,int msgseq,int ACK_seq){ ...
51 //四次挥手断开连接---接收ACK
52 > Packet FourHand_4(SOCKET socket, struct sockaddr_in clientAddr,int msgseq,int ACK_seq){ ...
53
54 // 保存数据包到文件
55 > void SavePacketToFile(const Packet& packet, const std::string& filename) { ...
56
57
58
59
60 > struct ThreadParams{ ...
61 > DWORD WINAPI ReceiveDataThread(LPVOID lpParam) { //用于接收数据 ...
62 > Packet CreateACKPacket(int msgseq, int ackseq,ThreadParams* params){ ...
63
64
65 > DWORD WINAPI SendDataThread(LPVOID lpParam) { ...
66
67
68
69
70
71
72
73
74
75
76 int main() {

```

在发送端的代码中编写的函数主要有

1. checksum: 计算校验和, 计算方式如协议中所述, 16 位为一组, 求和后计算反码.
2. calculatePacketChecksum: 验证接收到的包的校验和, 最后通过将校验和的计算结果与 0 比较, 若为全零则认为无差错, 函数返回 true; 否则返回 false.

3. 三次握手的函数：这里分别用于接收 SYN、发送 SYN-ACK、接收 ACK
4. 四次挥手的函数：分别用于接收 FIN、发送 ACK、发送 FIN-ACK、接收 ACK
5. 线程函数 ReceiveACKThread: 接收端接收发送端发来的数据包，可能是多种类型的数据包，集中在这个线程函数中处理。
6. 线程函数 SendDataThread: 接收端收到发送端的数据包后需要回复 ACK，通过这个线程函数发送。
7. SavePacketToFile: 接收端在收到数据后通过该函数将数据包中的数据保存到指定文件中。

(三) 建立连接过程


建立连接的过程采用模拟三次握手的过程，但存在些许差别，具体的过程已经在协议说明中给出，在这里就不再重复说明，但同时列出发送端和接收端的连接建立过程代码以便理解：

```

Packet synAck; //接收SYN-ACK
// 发送SYN
while(true){
    ThreeHand_1(socket, serverAddr, msgseq); //序列号是0
    SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
    cout<<"[SYN Send]"<<endl;
    SetConsoleTextAttribute(hConsole, 0x07);
    cout<<"Waiting for SYN-ACK package..."<<endl;

    // 等待SYN-ACK
    synAck = ThreeHand_2(socket, serverAddr, msgseq+1);
    if(synAck.flag == 0x21){ //空包，表示超时，需要重传
        SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
        cout<<"[SYN-ACK Time Out]"<<endl;
        continue; //超时重传
    }
    else {
        SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
        cout<<"[SYN-ACK Receive Success]"<<endl;
        expectedseq = synAck.msgseq+1;
        break;
    }
}
//在成功建立连接后，服务端会sleep6s，也就是说若超时后未收到重复的SYN-ACK包，说明连接建立成功；若收到重复SYN-ACK包，则说明需要重传ACK包

```



```

msgseq++;
while(true){
    ThreeHand_3(socket, serverAddr, msgseq, expectedseq);
    SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
    cout<<"[ACK Send Success]"<<endl;
    SetConsoleTextAttribute(hConsole, 0x07);
    cout<<"Waiting for 2 seconds to ensure that the connection is established..."<<endl;
    ///等待2s若未收到重复的SYN-ACK包，则认为连接建立成功
    // 等待SYN-ACK
    int Count = 0;
    while(true){
        Packet synAck2 = ThreeHand_2(socket, serverAddr, msgseq);
        if(synAck2.flag == 0x21){ //表示超时，超时2次就是2秒
            Count++;
            if(Count == 2)break;
        }
        else { //在2s内收到了SYN-ACK包，说明连接建立失败，需要重传ACK包
            break;
        }
    }
    if(Count == 2)break;
    else {
        continue; //这种情况表示在2s内收到了SYN-ACK包，说明连接建立失败，需要重传ACK包
    }
}
cout<<"-----Connection Established!!!-----"<<endl;
msgseq++; //连接建立成功后，序列号再加1
//-----

```

```

//初始化序列号
msgseq = 0; ///这里会一直等直到收到SYN包
Packet synPacket = ThreeHand_1(socket, clientAddr);
expectedseq = synPacket.msgseq+1; //ACK序列号
SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
cout<<"[SYN Received]"<<endl;

Packet AckPacket;
//等待接收SYN包
while(true){
    //发送SYN-ACK
    ThreeHand_2(socket, clientAddr, expectedseq, msgseq);
    SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
    cout<<"[SYN-ACK Send]"<<endl;

    //等待接收ACK
    AckPacket = ThreeHand_3(socket, clientAddr, msgseq+1);
    if(AckPacket.flag == 0x21){ //空包, 表示超时, 需要重传----超时会重新发送SYN-ACK
        continue;
    }
    else {
        SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
        cout<<"[ACK Received]"<<endl;
        break;
    }
}
Sleep(4000); //睡眠4s
msgseq++;
expectedseq = AckPacket.msgseq+1; //期望收到的下一个数据包的序列号
cout<<"----Connection Established!!!-----"<<endl;
/////Ack序列号, 每次在成功接收到数据后+1
/////seq序列号, 每次在收到ACK后+1

```

(四) 数据传输过程

发送端发送数据部分代码:

```

while(true){
    char FileName[256]; // 假设文件名不超过 255 个字符
    cout << "Please Input the File Name:(Press './exit' to exit)" << endl;
    cin.getline(FileName, 256); // 使用 getline 以允许空格

    if (strcmp(FileName, "./exit") == 0) { ...
        const char* FileName2 = FileName;
        cout<<"FileName:"<<FileName2<<endl;

        packets.clear();
        packets.resize(30000);
        readAndSplitFile(FileName2, packets, params);

        ///packets[msgseq]存放一个只有文件名的包
        Packet filenamepacket;
        filenamepacket.flag = 0x10; // 文件名
        filenamepacket.msgseq = msgseq; // 序列号, 发完就加
        filenamepacket.ackseq = 0; // 确认序列号---这里不需要
        filenamepacket.SendIP = inet_addr("127.0.0.1"); // 发送者的IP地址
        filenamepacket.RecvIP = inet_addr("127.0.0.1"); // 接收者的IP地址
        filenamepacket.SendPort = htons(0); // 发送者的端口号
        filenamepacket.RecvPort = htons(RouterPort); // 接收者的端口号
        filenamepacket.checksum = 0; // 校验和, 根据实际情况计算
        memset(filenamepacket.msg, 0, sizeof(filenamepacket.msg)); // 数据内容初始化为空
        memcpy(filenamepacket.msg, FileName, sizeof(FileName)); // 复制数据内容
        filenamepacket.checksum = checksum(&filenamepacket, sizeof(filenamepacket)); // 计算校验和
        packets[msgseq]=(filenamepacket);

        // 创建一个线程用于接收ACK
        IsThreadOver = false;
        HANDLE hThread = CreateThread(NULL, 0, ReceiveACKThread, &params, 0, NULL);
    }
}

```

```

//接下来是不断地传输数据
Packet packet;
packet = packets[msgseq];

DWORD FileTransmitTime = clock();

while(true){
    if(CanNext == true){
        packet = packets[msgseq];
        CanNext = false;
        if(packet.flag == 0x00){
            //数据发送完毕, 准备关闭连接
            packets.clear();
            SetConsoleTextAttribute(hConsole, 0x07);
            cout<<"Data Send Finish!!!"<<endl;
            cout<<"FileName:"<<FileName2<<"    FileTransmitTime:"<<clock()-FileTransmitTime<<"ms"<<endl;
            cout<<"InOut Rate:"<<msgseq*1024*CLOCKS_PER_SEC/(clock()-FileTransmitTime)<<"byte/s"<<endl;
            IsThreadOver = true;
            break;
        }
    }

    // 发送数据包--可能需要重发
    int sendResult = sendto(socket, (char*)&packet, sizeof(packet), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));

    IsTimeOut = false;
    startTime = GetTickCount(); //启动定时器
    if (sendResult == SOCKET_ERROR) {
        SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
    }
}

// 发送数据包--可能需要重发
int sendResult = sendto(socket, (char*)&packet, sizeof(packet), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));

IsTimeOut = false;
startTime = GetTickCount(); //启动定时器
if (sendResult == SOCKET_ERROR) {
    SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
    cout<<"[Data SendError]"<<endl;
    continue;
}
else {
    SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
    cout<<"[Data Send]";
    SetConsoleTextAttribute(hConsole, 0x07);
    cout<<"Packet Seq:";
    cout<<"msgseq:"<<msgseq<<endl;
    cout<<packet.msgseq<<"Checksum:"<<packet.checksum<<endl;
}

while(1){
    if(CanNext == true){
        SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
        cout<<"[Preparing for Next Data]"<<endl;
        break;
    }
    else if(IsTimeOut == true){
        SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
        cout<<"[Waiting for ACK Time Out]"<<endl;
        break;
    }
}
}

```

这部分代码的逻辑是：首先要求用户输入要传输的文件名，接着分割此文件并保存在名为 Packets 的 Vector 容器中，接着进入一个 while 循环开始不断发送数据，每次发送完一个数据包后都会进入另一个 while 死循环，只有当定时器超时（对应重发数据包）或收到 ACK（对应发送下一个数据包）时才会 break 退出循环。

接下来需要具体说明一下发送端的接收线程函数以及接收端的发送线程函数和接收线程函数：

发送端的接受线程函数，这个函数用于接收 ACK 数据包：

```

320 while (true) {
321     if(IsThreadOver == true){
322         break;
323     }
324     if(GetTickCount() - startTime > MAX_TIMEOUT){ //超时
325         //需要重发数据包DataPacket
326         IsTimeOut = true;
327         startTime = GetTickCount();//重置开始时间
328         continue;
329     }
330     //接收ACK
331     int result = recvfrom(socket, (char*)&receivedPacket, sizeof(Packet), 0, (struct sockaddr*)&ServerAddr, &serverAddrSize);
332     if (result == SOCKET_ERROR) { //表示没数据可以接收
333         //cout << "Receive failed with error: " << WSAGetLastError() << endl;
334     } else {
335         // 处理接收到的数据
336         //首先计算校验和, 错了丢弃, 对了继续
337         if(calculatePacketChecksum(receivedPacket) == false){
338             SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
339             cout << "[Checksum Error]";
340             SetConsoleTextAttribute(hConsole, 0x07);
341             cout << "Checksum: " << receivedPacket.checksum << endl;
342             continue;
343         }
344         //到这里的话, 标志位是对的, 根据标志位判断包的类型
345         switch(receivedPacket.flag){
346             case 0x02: //ACK-----
347                 //收到ACK包后, 可以更新seq
348                 //首先验证一下, 这个包ACK的是不是我们上一个发送的数据包
349                 if(receivedPacket.ackseq != msgseq+1){
350                     //表示不是, 需要重发
351                     //通过手动超时的方式
352                     IsTimeOut = true;
353                     continue;
354                 }
355                 //到这个位置, 表示是, 可以发下一个了
356                 startTime = GetTickCount();//重置定时器
357                 //首先, 收到ACK包说明之前发的数据包已经到达, 现在可以发送下一个数据包
358                 msgseq = receivedPacket.ackseq; //更新seq, receivedPacket.ackseq表示的是对方期望收到的下一个数据包的序列号
359                 CanNext = true; //可以发送下一个数据包了
360                 expectedseq = receivedPacket.msgseq + 1; //更新期望收到的下一个数据包的序列号
361             break;
362             default:
363                 break;
364         }
365     }
366 }
367 }
368 }
369 return 0;
370 }
371 }

```

在这个线程函数中, 主体部分是一个 While(true) 的循环, 只有在特定情况下才会 break 结束。

可以看到, 这个 while 循环会不断地调用 recvfrom 函数接收数据包, 需要指明 socket 被设置为非阻塞状态, 因此每接收到一个数据包都要先判断是否属于无数据可接受的情况。接着会首先计算校验和, 若校验和错误说明数据包发生错误, 可以直接丢弃该包; 校验和正确后, 根据其数据包标志位字段来判断数据包的类型, 实际上在这里目前只处理 ACK 的情况, 当属于 ACK 的情况时会进一步判断其 ACK 序列号是否与期望的一致, 不一致则丢弃, 一致则说明之前的数据包被成功接收, 可以更新序列号。

接收端的接收线程函数, 用于接收数据包:

```

337 while (true) {
338     if(IsThreadOver == true) break;
339     if(GetTickCount() - startTime > TIMEOUT){ //超时
340         IsSendACK = true; //IsSendACK设置为true后会重发ACK
341         startTime = GetTickCount(); //重置开始时间
342     }
343
344     int result = recvfrom(socket, (char*)&receivedPacket, sizeof(Packet), 0, (struct sockaddr*)&clientAddr, &clientAddrSize);
345     if (result == SOCKET_ERROR) { //表示没数据可以接收
346
347     } else {
348         // 处理接收到的数据
349         // 首先计算校验和, 错了丢弃, 对了继续
350         if(calculatePacketChecksum(receivedPacket) == false){
351             SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
352             cout<<"[Checksum Error]";
353             SetConsoleTextAttribute(hConsole, 0x07);
354             cout<<"Checksum: "<<receivedPacket.checksum<<endl;
355             continue;
356         }
357         // 到这里的话, 标志位是对的, 根据标志位判断包的类型---
358         // 同时, 收到数据的话还需要取消定时器---在这里就通过将starttime设置为负数的方式来实现
359         switch(receivedPacket.flag){
360
361             case 0x04: //Data
362                 SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
363                 cout<<"[Data Received]"<<endl;
364                 //接收到了数据包, 说明服务端的上一个发的ACK成功收到了, 因此发送序号可以增加
365                 //这里需要更新序列号, 但在更新序列号前, 需要判断是否需要重传-----
366                 if(receivedPacket.msgseq != expectedseq){ //真正收到的与期望收到的不一致, 需要重发ACK
367                     IsSendACK = true; //IsSendACK设置为true后会重发ACK
368                     continue; //接着等待
369                 }

```

```

359 switch(receivedPacket.flag){
370     //否则不需要重传
371     SavePacketToFile(receivedPacket, ".\\COPY-"+filename); // 保存数据包到文件
372     expectedseq = receivedPacket.msgseq + 1; //更新期望的序列号
373     //更新msgseq
374     msgseq = msgseq + 1; //因为收到了数据包说明上一个ACK已经发送成功了
375     startTime = -1; //取消定时器
376     IsSendACK = true;
377     break;
378     case 0x08: //FIN
379         //收到FIN包, 结束接收和发送线程, 回到主线程完成四次挥手
380         expectedseq = receivedPacket.msgseq + 1; //更新期望的序列号
381         //更新msgseq
382         msgseq = msgseq + 1; //因为收到了数据包说明上一个ACK已经发送成功了
383         IsThreadOver = true;
384         break;
385     case 0x10: //文件名
386         SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
387         cout<<"[Filename Received]"<<endl;
388         //接收到了数据包, 说明服务端的上一个发的ACK成功收到了, 因此发送序号可以增加
389         //这里需要更新序列号, 但在更新序列号前, 需要判断是否需要重传-----
390         if(receivedPacket.msgseq != expectedseq){ //真正收到的与期望收到的不一致, 需要重发ACK
391             IsSendACK = true; //IsSendACK设置为true后会重发ACK
392             continue; //接着等待
393         }
394         //否则不需要重传
395         //这一部分为处理接收到的数据的代码
396         filename = receivedPacket.msg; // 保存文件名
397
398         expectedseq = receivedPacket.msgseq + 1; //更新期望的序列号
399         //更新msgseq
400         msgseq = msgseq + 1; //因为收到了数据包说明上一个ACK已经发送成功了

```

这个线程函数的逻辑大致与上述的线程函数一致, 差别在于多增加了几种处理情况: 数据包为 FIN 包和特用于传输文件名时的情况。

在这里还需要注意一个全局 Bool 变量 IsSendACK, 这个全局变量用于控制发送线程函数是否发送 ACK 包。

接收端的发送线程函数, 用于发送 ACK 数据包:

```

DWORD WINAPI SendDataThread(LPVOID lpParam) {
    // 从 lpParam 中取出参数
    ThreadParams* params = (ThreadParams*)lpParam;
    SOCKET socket = params->socket;
    struct sockaddr_in clientAddr = params->clientAddr;
    while(true){ //ACK需要超时重传
        if(IsThreadOver == true)break;
        if(IsSendACK == true){
            // 准备要发送的数据包
            Packet packet = CreateACKPacket(msgseq, expectedseq,params);
            // 发送数据
            int result = sendto(socket, (char*)&packet, sizeof(packet), 0, (struct sockaddr*)&clientAddr, sizeof(clientAddr));
            startTime = GetTickCount(); // 记录开始时间
            if (result == SOCKET_ERROR) {
                SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
                cout << "[Send failed]error: ";
                SetConsoleTextAttribute(hConsole, 0x07);
                cout << WSAGetLastError() << endl;
            } else {
                SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
                cout << "[ACK Send]" ;
                SetConsoleTextAttribute(hConsole, 0x07);
                cout<<"    msgseq:"<<msgseq<< "    ACKSeq:"<<expectedseq<<endl;
            }
            IsSendACK = false;
        }
    }
    return 0;
}

```

这个发送线程函数同样是一个 while(true) 的循环，在每次循环都会通过变量 IsSendACK 来决定是否要发送 ACK 数据包，如上所述这个变量会在每次接收到正确数据包或超时后被置为 true，且每次发送完 ACK 后置为 false。

(五) 断开连接过程

断开连接的过程采用模拟四次挥手的方式，但存在些许差别，具体的过程已经在协议说明中给出，在这里就不再重复说明，但同时列出发送端和接收端的连接关闭过程代码以便理解：

发送方：

```

// 发送FIN包并等待ACK包
Packet ACKPacket;
while(true){
    FourHand_1(socket,serverAddr,msgseq);
    //等待接收ACK包
    cout<<"[FIN Send]";
    cout<<"msgseq:"<<msgseq<<endl;
    cout<<"    Msg Seq:"<<msgseq<<endl;
    ACKPacket = FourHand_2(socket,serverAddr,msgseq+1);
    if(ACKPacket.flag == 0x21){ ///空包，表示超时，需要重传
        SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
        cout<<"[ACK Recive Time Out]"<<endl;
        continue;
    }
    else{
        SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
        cout<<"[ACK Recive Success]"<<endl;
        msgseq = msgseq+1;
        expectedseq = ACKPacket.msgseq+1;
        break;
    }
}
}

```

```

//等待接收FIN-ACK包
Packet FINACKPacket;
cout<<"Waiting for FIN-ACK Package!!!"<<endl;
while(true){
    FINACKPacket = FourHand_3(socket,serverAddr,expectedseq,msgseq);
    if(ACKPacket.flag == 0x21){ ///空包, 表示超时,这时候接着等待不需要重传
        SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
        cout<<"[FIN-ACK Recive Time Out]"<<endl;
        continue;
    }
    else{
        SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
        cout<<"[FIN-ACK Recive Success]"<<endl;
        expectedseq = ACKPacket.msgseq+1;
        break;
    }
}
}

```

```

//回复一个ACK后关闭连接
while(true){
    FourHand_4(socket,serverAddr,msgseq,expectedseq);
    SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
    cout<<"[ACK Send]";
    cout<<"msgseq:"<<msgseq<<endl;
    cout<<"Msg Seq:"<<msgseq<<endl;
    SetConsoleTextAttribute(hConsole, 0x07);
    cout<<"Waiting for 2 seconds to ensure that the ACK package has been received by the server!!!"<<endl;
    //等待2s, 若未收到重复的FIN-ACK包, 则认为可以关闭连接
    //等待FIN-ACK
    int Count = 0;
    while(true){
        Packet FinAck = FourHand_3(socket,serverAddr,expectedseq-1,msgseq);
        if(finAck.flag == 0x21){ //表示超时
            Count++;
            if(Count == 2)break;
        }
        else{ //表示在2s内收到了FIN-ACK包, 说明ACK包未接收到, 需要重传ACK包
            continue;
        }
    }
    if(Count == 2)break;
}
}

```

接收方:

```

//四次挥手关闭连接
SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
cout<<"[FIN Received]"<<endl;

while(true){
    //发送ACK包
    FourHand_2(socket, clientAddr,msgseq,expectedseq);
    SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
    cout<<"[ACK Send]"<<endl;
    cout<<"Waiting for 2 seconds to ensure that the ACK package has been received!!!"<<endl;

    //等待接收FIN包
    Packet finPacket = FourHand_1(socket, clientAddr,expectedseq-1);
    if(finPacket.flag == 0x21){ ///空包, 表示超时---表示没有收到重复FIN
        SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
        cout<<"[Ensure->ACK Received]"<<endl;
        break;
    }
    else {
        //2s内收到了FIN包, 需要重传ACK包
        SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
        cout<<"[Receive FIN Package Again]Preparing to resend ACK Package!!!"<<endl;
        continue;
    }
}
}

```

```
Packet finackPacket;
//发送FIN-ACK包并等待接收ACK包
while(true){
    //发送FIN-ACK包
    FourHand_3(socket, clientAddr, msgseq, expectedseq);
    SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
    cout<<"[FIN-ACK Send]"<<endl;

    //等待接收ACK包
    finackPacket = FourHand_4(socket, clientAddr, expectedseq, msgseq+1);
    if(finackPacket.flag == 0x21){ //空包, 表示超时
        continue;
    }
    else {
        SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
        cout<<"[FinalACK Received]"<<endl;
        break;
    }
}
SetConsoleTextAttribute(hConsole, 0x07);
cout<<"Connection Closed Successfully!!!"<<endl;
```

四、 结果展示

使用提供的 Router 路由, 并设置丢包率为 3%, 延时为 3%, 在此条件下进行测试。

The image shows a 'Router' configuration window with the following fields and values:

- 路由器IP: 127 . 0 . 0 . 1
- 服务器IP: 127 . 0 . 0 . 1
- 端口: 8088
- 服务器端口: 8078
- 丢包率: 3 %
- 延时: 3 ms

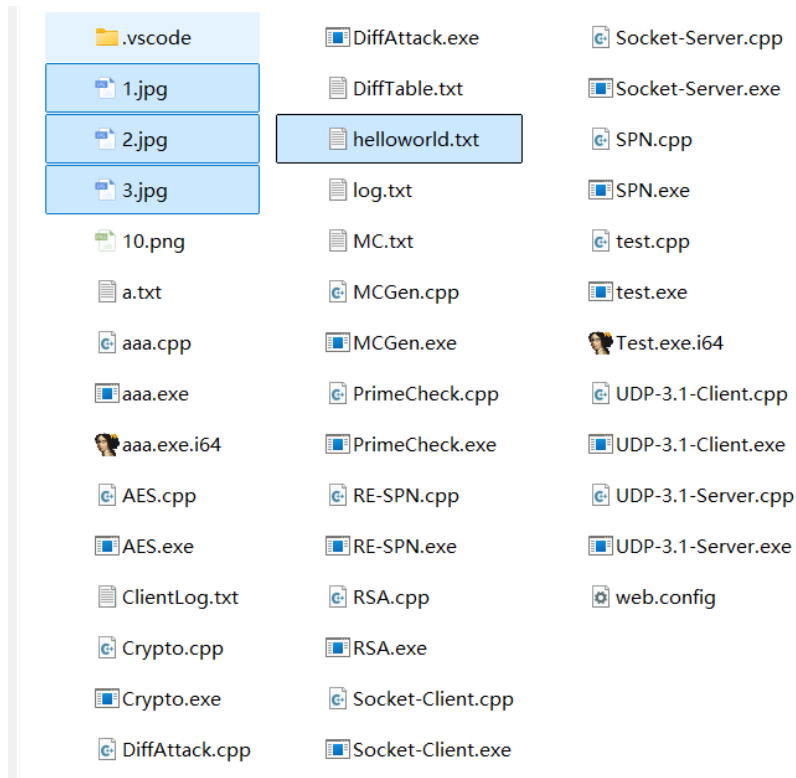
Buttons: 确定 (OK), 修改 (Modify)

日志 (Log):

```
count:5.
Delay 3 ms.
count:6.
Delay 3 ms.
count:7.
Delay 3 ms.
count:8.
Delay 3 ms.
count:9.
```

注: 此时的数据传输过程为: 发送端→Router(端口 8088)→ 接收端 (8078)

传输提供的测试文件, 传输前的文件夹下的文件结构如下, 要传输的文件均已标注出:



注：在传输后，接收到的文件也被放在该目录，同时文件名被设置为”COPY-”+ 传输的文件的文件名

运行两个程序，三次握手过程如下：


```
PS D:\Code-Cpp> ./UDP-3.1-Server.exe
UDP server bound to port 8078
Waiting for connection...
[SYN Received]
[SYN-ACK Send]
[ACK Received]
[]

PS D:\Code-Cpp> ./UDP-3.1-Client.exe
Client Start!!!
UDP Create Success!!!
[Checksum]9601
[SYN Send]
Waiting for SYN-ACK package...
[SYN-ACK Receive Success]
[ACK Send Success]
Waiting for 2 seconds to ensure that the connection is established...
----Connection Established!!!-----
Please Input the File Name:(Press './exit' to exit)
```

接下来传输 helloworld.txt 文件和 1.jpg 文件，传输过程的部分截图如下：

```
[ACK Send] msgseq:76 ACKSeq:77
[Data Received]
[ACK Send] msgseq:77 ACKSeq:78
[Data Received]
[ACK Send] msgseq:78 ACKSeq:79
[Data Received]
[ACK Send] msgseq:79 ACKSeq:80
[Data Received]
[ACK Send] msgseq:80 ACKSeq:81
[Data Received]
[ACK Send] msgseq:81 ACKSeq:82
[Data Received]
[ACK Send] msgseq:82 ACKSeq:83
[Data Received]
[ACK Send] msgseq:83 ACKSeq:84
[Data Received]
[ACK Send] msgseq:84 ACKSeq:85
[Data Received]
[ACK Send] msgseq:85 ACKSeq:86
[ACK Send] msgseq:85 ACKSeq:86
[Data Received]
[ACK Send] msgseq:86 ACKSeq:87
[Data Received]
[ACK Send] msgseq:87 ACKSeq:88
[Data Received]
[ACK Send] msgseq:88 ACKSeq:89
[Data Received]
[ACK Send] msgseq:89 ACKSeq:90
[Data Received]
[ACK Send] msgseq:90 ACKSeq:91
[Data Received]
[ACK Send] msgseq:91 ACKSeq:92
[Data Received]
[ACK Send] msgseq:92 ACKSeq:93

[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:83
83Checksum:4459
[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:84
84Checksum:4458
[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:85
85Checksum:4457
[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:86
86Checksum:4456
[Waiting for ACK Time Out]
[Data Send]Packet Seq:msgseq:86
86Checksum:4456
[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:87
87Checksum:4455
[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:88
88Checksum:4454
[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:89
89Checksum:4453
[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:90
90Checksum:4452
[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:91
91Checksum:4451
[Preparing for Next Data]
[Data Send]Packet Seq:msgseq:92
92Checksum:4450
```

 Router ×

路由器IP:

服务器IP:

端口:

服务器端口:

丢包率: %

延时: ms

确定

修改

日志

count:29.
Delay 3 ms.
count:30.
Delay 3 ms.
count:31.
Delay 3 ms.
count:32.
Delay 3 ms.
Miss a packet.

```

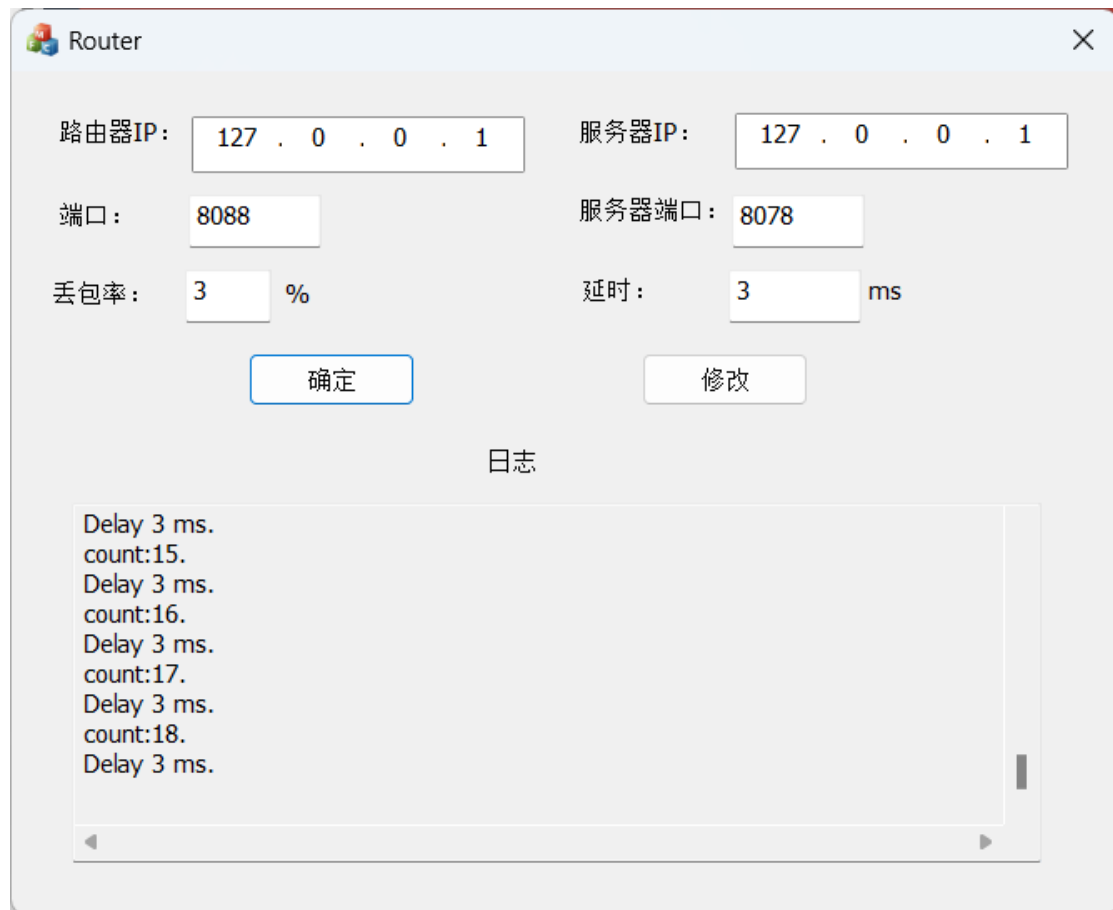
[ACK Send] msgseq:1619 ACKSeq:1620 1610Checksum:2932
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 [Data Send]Packet Seq:msgseq:1611
[ACK Send] msgseq:1619 ACKSeq:1620 1611Checksum:2931
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 [Data Send]Packet Seq:msgseq:1612
[ACK Send] msgseq:1619 ACKSeq:1620 1612Checksum:2930
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 [Data Send]Packet Seq:msgseq:1613
[ACK Send] msgseq:1619 ACKSeq:1620 1613Checksum:2929
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 [Data Send]Packet Seq:msgseq:1614
[ACK Send] msgseq:1619 ACKSeq:1620 1614Checksum:2928
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 [Data Send]Packet Seq:msgseq:1615
[ACK Send] msgseq:1619 ACKSeq:1620 1615Checksum:2927
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 [Data Send]Packet Seq:msgseq:1616
[ACK Send] msgseq:1619 ACKSeq:1620 1616Checksum:2926
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 [Data Send]Packet Seq:msgseq:1617
[ACK Send] msgseq:1619 ACKSeq:1620 1617Checksum:2925
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 [Data Send]Packet Seq:msgseq:1618
[ACK Send] msgseq:1619 ACKSeq:1620 1618Checksum:2924
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 [Data Send]Packet Seq:msgseq:1619
[ACK Send] msgseq:1619 ACKSeq:1620 1619Checksum:2923
[ACK Send] msgseq:1619 ACKSeq:1620 [Preparing for Next Data]
[ACK Send] msgseq:1619 ACKSeq:1620 Data Send Finish!!!
[ACK Send] msgseq:1619 ACKSeq:1620 FileName:hellworld.txt FileTransmitTime:178589ms
[ACK Send] msgseq:1619 ACKSeq:1620 InOut Rate:9288byte/s
[ACK Send] msgseq:1619 ACKSeq:1620 Please Input the File Name:(Press './exit' to exit)

```

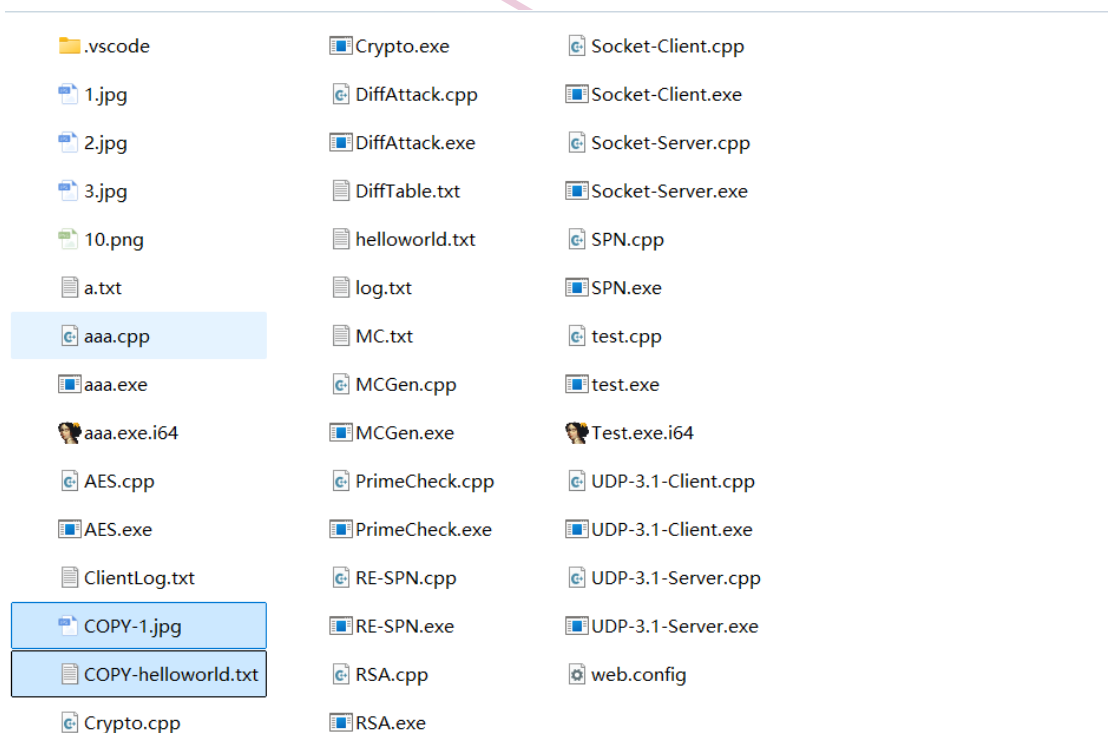
```

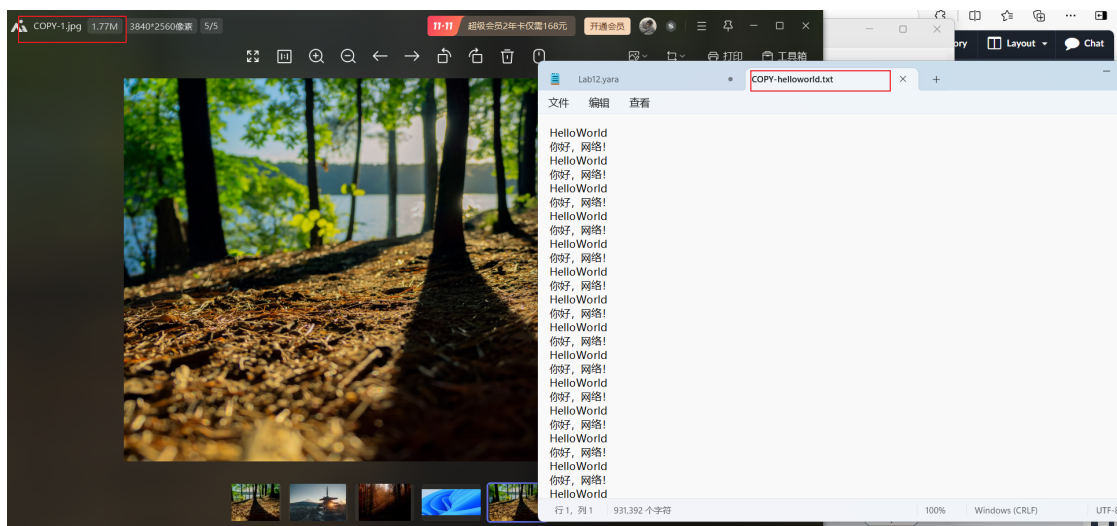
[ACK Send] msgseq:2404 ACKSeq:2405 [Preparing for Next Data]
[Data Received] [Data Send]Packet Seq:msgseq:2411
[ACK Send] msgseq:2405 ACKSeq:2406 2411Checksum:56566
[Data Received] [Preparing for Next Data]
[ACK Send] msgseq:2406 ACKSeq:2407 [Data Send]Packet Seq:msgseq:2412
[Data Received] 2412Checksum:40866
[ACK Send] msgseq:2407 ACKSeq:2408 [Preparing for Next Data]
[Data Received] [Data Send]Packet Seq:msgseq:2413
[ACK Send] msgseq:2408 ACKSeq:2409 2413Checksum:35696
[Data Received] [Preparing for Next Data]
[ACK Send] msgseq:2409 ACKSeq:2410 [Data Send]Packet Seq:msgseq:2414
[Data Received] 2414Checksum:29817
[ACK Send] msgseq:2410 ACKSeq:2411 [Preparing for Next Data]
[Data Received] [Data Send]Packet Seq:msgseq:2415
[ACK Send] msgseq:2411 ACKSeq:2412 2415Checksum:6788
[Data Received] [Preparing for Next Data]
[ACK Send] msgseq:2412 ACKSeq:2413 [Data Send]Packet Seq:msgseq:2416
[Data Received] 2416Checksum:28233
[ACK Send] msgseq:2413 ACKSeq:2414 [Preparing for Next Data]
[Data Received] [Data Send]Packet Seq:msgseq:2417
[ACK Send] msgseq:2414 ACKSeq:2415 2417Checksum:47431
[Data Received] [Preparing for Next Data]
[ACK Send] msgseq:2415 ACKSeq:2416 [Data Send]Packet Seq:msgseq:2418
[Data Received] 2418Checksum:12863
[ACK Send] msgseq:2416 ACKSeq:2417 [Preparing for Next Data]
[Data Received] [Data Send]Packet Seq:msgseq:2419
[ACK Send] msgseq:2417 ACKSeq:2418 2419Checksum:3103
[Data Received] [Preparing for Next Data]
[ACK Send] msgseq:2418 ACKSeq:2419 [Data Send]Packet Seq:msgseq:2420
[Data Received] 2420Checksum:38524
[ACK Send] msgseq:2419 ACKSeq:2420 [Preparing for Next Data]
[Data Received] [Data Send]Packet Seq:msgseq:2421
[ACK Send] msgseq:2420 ACKSeq:2421 2421Checksum:39798

```

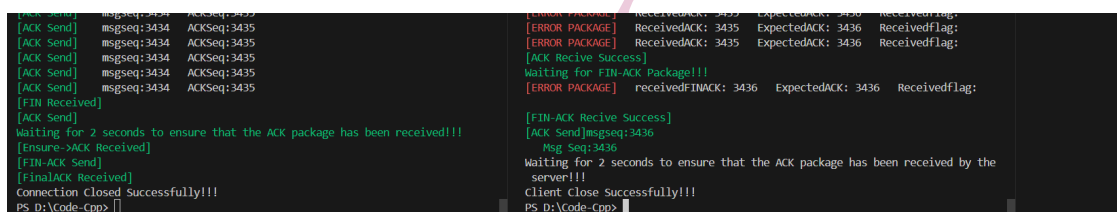


文件传输完毕后查看原文件夹，发现了传输得到的新文件 COPY-helloworld.txt 和 COPY-1.jpg:

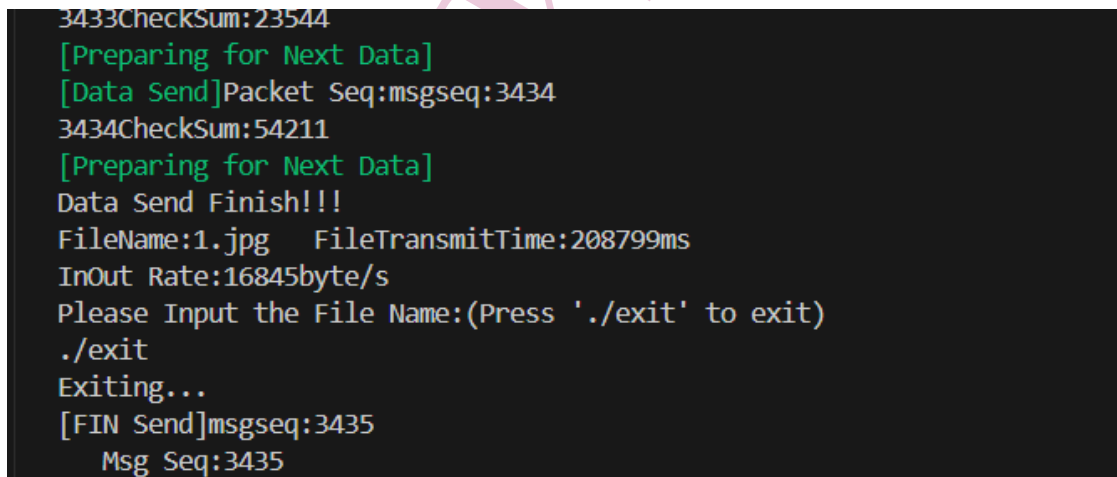




传输完毕后，发送端输入指令‘exit’以进入四次挥手过程，最后关闭连接：



在每个文件传输完成后还会输出本次传输过程中的传输速率及传输时间的信息：



五、 遇到的问题、收获及心得体会

在这次实验中，实现了一个基于 UDP 数据报的可靠数据传输程序，UDP 本身是不可靠的，但是通过协议的设计，可以保证数据传输的可靠性。这次实验从协议设计开始，到三次握手、四次挥手，以及最后的数据传输过程，对各个阶段的协议设计和代码实现，使我对可靠数据传输的相关知识有了更加深入的了解，特别是对序列号的使用印象非常深刻。

这次实验过程中我认为比较困难的一个点是，为了保证数据包的正确性，需要通过传输序列号及 ACK 序列号来判断当前数据包的处理情况，除此之外还需要同时考虑发送方和接收方两

方，这就使得我在编写序列号的判断代码时非常容易把顺序弄乱，尤其是在编写三次握手和四次挥手的代码时。

最后很明显这次程序存在许多可以改进的地方，例如使用停等机制虽然实现简单，但却会影响传输效率；以及对序列号的使用其实可以更加精简，例如只使用 0 和 1。这两点都有待提升。

NIKU