



南开大学
Nankai University

南 开 大 学

计 算 机 学 院

计算机网络实验报告

大作业第一部分实验报告

汤清云 2013536

年级：2020 级

专业：计算机科学与技术

指导教师：张建忠 徐敬东

2022 年 11 月 19 日

摘要

本次实验为最终大作业的第一部分实现，在本次实验中，我利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、确认重传等。流量控制采用停等机制，完成给定测试文件的传输。

关键词：UDP 校验和 三次握手 四次挥手 超时重传

目录

一、 实验目的	1
二、 实验设计	1
(一) 报文格式	1
(二) 建立连接	1
(三) 断开连接	6
(四) 差错检测	13
(五) 确认重传	13
(六) 流量控制	14
(七) 协议设计	14
(八) 性能测试	25

一、 实验目的

使用数据报式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、断开连接、差错检测、确认重传等。

二、 实验设计

(一) 报文格式

据报由两部分组成，第一部分是 head，即数据报头，第二部分是所携带的数据部分，长度上限为 2048 字节。在数据报头 UDPpackethead 和数据报 UDPpacket 中包含以下内容：

```
1 struct UDPpackethead
2 {
3     unsigned int seq;//序列号 16位
4     u_short Check;//校验 16位
5     u_short len;//数据部分总长度
6     unsigned char flags;//标志位
7     UDPpackethead()
8     {
9         len = 0;
10        seq = 0;
11        Check = 0;
12        flags = 0;
13    }
14 };
15 struct UDPpacket
16 {
17     UDPpackethead head;
18     char data[2048];//每段data长度上限
19 };
```

(二) 建立连接

此处协议设计模拟了 TCP 的三次握手协议。

第一次握手：客户端向服务器发送头部标志位为 SYN 的数据包，表示想与服务器建立连接：

A. 此数据包校验和错误/中途丢失，服务器接收后直接丢弃，继续处于等待接收状态，不发送数据包——客户端在经过一段时间后收不到返回数据包，因此重发标志位为 SYN 的数据包。

B. 此数据包被服务器正确接收，进入第二次握手。

第二次握手：服务器向客户端发送头部标志位为 SYN_ACK 的数据包

A. 此数据包校验和错误/中途丢失，客户端接收后直接丢弃，继续处于等待接收状态，不发送数据包——服务器在经过一段时间后收不到返回数据包，因此重发标志位为 SYN_ACK 的数据包。

B. 此数据包被客户端正确接收，进入第三次握手。

第三次握手：客户端向服务器发送头部标志位为 ACK 的数据包

A. 此数据包校验和错误/中途丢失，服务器接收后直接丢弃，继续处于等待接收状态，不发送数据包——客户端在经过一段时间后收不到返回数据包，因此重发标志位为 ACK 的数据包。

B. 此数据包被服务器正确接收，握手完成。

客户端建立连接函数：

```

1  int connect(SOCKET& client, SOCKADDR_IN& ClientAddr, int &len)
2  {
3      //设置为非阻塞模式，避免卡在recvfrom
4      int iMode = 1; //0: 阻塞
5      ioctlsocket(client, FIONBIO, (u_long FAR*) & iMode); //非阻塞设置
6      //第一次握手，客户端发送SYN
7      UDPpacket pack1;
8      pack1.head.flags = SYN;
9      pack1.head.Check = packetcheck((u_short*)&(pack1), sizeof(pack1));
10     //发送缓冲区
11     char* buffer1 = new char[sizeof(pack1)];
12     memcpy(buffer1, &pack1, sizeof(pack1));
13     int flag = sendto(client, buffer1, sizeof(pack1), 0, (sockaddr*)&
        ClientAddr, len);
14     //第一次握手发送时间
15     clock_t pack1starttime = clock();
16     if (flag == -1)
17     {
18         now = time(nullptr);
19         curr_time = ctime(&now);
20         cout << curr_time << "[System]:";
21         cout << "————第一次握手发送失败————" << endl;
22     }
23     else
24     {
25         now = time(nullptr);
26         curr_time = ctime(&now);
27         cout << curr_time << "[System]:";
28         cout << "————第一次握手发送成功————" << endl;
29     }
30     //第二次握手，等待服务器发来的SYN_ACK
31     UDPpacket pack2;
32     //缓冲区
33     char* buffer2 = new char[sizeof(pack2)];
34     //一直没有收到包，说明之前的包出问题了，要重新发送第一次握手
35     while (1)
36     {
37         //如果超时则重传
38         if (clock() - pack1starttime >= 2 * CLOCKS_PER_SEC)
39         {
40             now = time(nullptr);
41             curr_time = ctime(&now);
42             cout << curr_time << "[System]:";
43             cout << "————第一次握手包超时重传————" <<
                endl;
44             sendto(client, buffer1, sizeof(pack1), 0, (sockaddr*)

```

```

45         &ClientAddr, len);
46         pack1starttime = clock();
47     }
48     flag = recvfrom(client, buffer2, sizeof(pack2), 0, (sockaddr
49         *)&ClientAddr, &len);
50     if (flag <= 0) //没收到包的话就继续recvfrom
51     {
52         continue;
53     }
54     //收到了包
55     memcpy(&pack2, buffer2, sizeof(pack2));
56     int res = packetcheck((u_short*)&pack2, sizeof(pack2));
57     //是第二次握手
58     if (pack2.head.flags == SYN_ACK && res == 0)
59     {
60         now = time(nullptr);
61         curr_time = ctime(&now);
62         cout << curr_time << "[System]:";
63         cout << "----- 第二次握手接收成功 -----" <<
64             endl;
65         break;
66     }
67     //否则一直处于等待状态
68 }
69 //第三次握手 客户端发送ACK
70 UDPpacket pack3;
71 pack3.head.flags = ACK;
72 pack3.head.Check = packetcheck((u_short*)&pack3, sizeof(pack3));
73 char* buffer3 = new char[sizeof(pack3)];
74 memcpy(buffer3, &pack3, sizeof(pack3));
75 flag = sendto(client, buffer3, sizeof(pack3), 0, (sockaddr*)&
76     ClientAddr, len);
77 if (flag == -1)
78 {
79     now = time(nullptr);
80     curr_time = ctime(&now);
81     cout << curr_time << "[System]:";
82     cout << "----- 第三次握手发送失败 -----" << endl;
83     return 0;
84 }
85 else
86 {
87     now = time(nullptr);
88     curr_time = ctime(&now);
89     cout << curr_time << "[System]:";
90     cout << "----- 第三次握手发送成功 -----" << endl;
91 }
92 clock_t checktime = clock(); //记录发送第三个包的时间

```

```

89     while ((clock() - checktime) <= 2* CLOCKS_PER_SEC)
90     {
91         //又把第二个包传过来了
92         //否则对面应该就结束传包
93         if (recvfrom(client, buffer2, sizeof(pack2), 0, (sockaddr*)&
94             ClientAddr, &len) <= 0)
95         {
96             continue;
97         }
98         //如果收到了包，那么就说明第三个包对面没收到，重传
99         sendto(client, buffer3, sizeof(pack3), 0, (sockaddr*)&
100             ClientAddr, len);
101         checktime = clock();
102     }
103     iMode = 0; //0: 阻塞
104     ioctlsocket(client, FIONBIO, (u_long FAR*) & iMode); //恢复成阻塞模式
105     return 1;
106 }

```

服务器端建立连接函数:

```

1  int connect(SOCKET& server, SOCKADDR_IN& ServerAddr, int& len)
2  {
3      //设置为非阻塞模式，避免卡在recvfrom
4      int iMode = 1; //0: 阻塞
5      ioctlsocket(server, FIONBIO, (u_long FAR*) & iMode); //非阻塞设置
6      UDPpacket pack1;
7      //缓冲区
8      char* buffer1 = new char[sizeof(pack1)];
9      //接收第一次握手请求信息
10     now = time(nullptr);
11     curr_time = ctime(&now);
12     cout << curr_time << "[System]:";
13     cout << "进入连接阶段" << endl;
14     int flag;
15     while (1)
16     {
17         //没收到包就一直循环
18         flag = recvfrom(server, buffer1, sizeof(pack1), 0, (sockaddr
19             *)&ServerAddr, &len);
20         if (flag <= 0)
21         {
22             continue;
23         }
24         //接收成功
25         memcpy(&(pack1), buffer1, sizeof(pack1.head));
26         u_short res = packetcheck((u_short*)&pack1, sizeof(pack1));
27         //只有当收到的数据包正确时才算是成功
28         if (pack1.head.flags == SYN && res == 0)

```

```

28         {
29             now = time(nullptr);
30             curr_time = ctime(&now);
31             cout << curr_time << "[System]:";
32             cout << "————— 第一次握手接收成功 ———" <<
33                 endl;
34             break;
35         }
36         //如果发来的包不正确的话, 继续这个循环, 等待客户端重新发送
37     //第二次握手, 服务器发送SYN_ACK
38     UDPpacket pack2;
39     pack2.head.flags = SYN_ACK;
40     pack2.head.Check = packetcheck((u_short*)&pack2, sizeof(pack2));
41     //发送缓冲区
42     char* buffer2 = new char[sizeof(pack2)];
43     memcpy(buffer2, &pack2, sizeof(pack2));
44     flag = sendto(server, buffer2, sizeof(pack2), 0, (sockaddr*)&
45         ServerAddr, len);
46     //超时重传机制
47     clock_t starttime = clock(); //第二次握手发送时间
48     if (flag == -1) //第二次握手发送失败, 重传
49     {
50         now = time(nullptr);
51         curr_time = ctime(&now);
52         cout << curr_time << "[System]:";
53         cout << "————— 第二次握手发送失败 ———" << endl;
54     }
55     else
56     {
57         now = time(nullptr);
58         curr_time = ctime(&now);
59         cout << curr_time << "[System]:";
60         cout << "————— 第二次握手发送成功 ———" << endl;
61     }
62     //第三次握手 服务器接收
63     UDPpacket pack3;
64     //缓冲区
65     char* buffer3 = new char[sizeof(pack3)];
66     //如果一直没有收到包的话, 说明上一个包需要重传
67     while (1)
68     {
69         //超时重传
70         if (clock_t() - starttime >= CLOCKS_PER_SEC)
71         {
72             now = time(nullptr);
73             curr_time = ctime(&now);
74             cout << curr_time << "[System]:";

```

```

74         cout << "----- 第二次握手包正在重传 -----" <<
75             endl;
76         sendto(server, buffer2, sizeof(pack2), 0, (sockaddr*)
77             &ServerAddr, len);
78         starttime = clock(); //更新发送时间
79     }
80     //没收到包就一直循环
81     flag = recvfrom(server, buffer3, sizeof(pack3), 0, (sockaddr
82         *)&ServerAddr, &len);
83     if (flag <= 0)
84     {
85         continue;
86     }
87     //第三次包接收成功
88     memcpy(&(pack3), buffer3, sizeof(pack3));
89     u_short res = packetcheck((u_short*)&pack3, sizeof(pack3));
90     if (pack3.head.flags == ACK && res == 0)
91     {
92         now = time(nullptr);
93         curr_time = ctime(&now);
94         cout << curr_time << "[System]:";
95         cout << "----- 第三次握手接收成功 -----" <<
96             endl;
97         break; //跳出循环
98     }
99     //否则的话就一直等到能接收正确的包
100 }
101 //等两个时钟周期
102 starttime = clock();
103 while ((clock() - starttime) <= 2 * CLOCKS_PER_SEC)
104 {
105 }
106 iMode = 0; //0: 阻塞
107 ioctlsocket(server, FIONBIO, (u_long FAR*) & iMode); //恢复成阻塞模式
108 return 1;
109 }

```

(三) 断开连接

此处协议设计模拟了 TCP 的四次挥手协议。

第一次挥手：客户端向服务器发送头部标志位为 FIN 的数据包，如果超时未收到来自服务器发送的 FIN_ACK 包，则重发第一次挥手包 FIN。如果收到了来自服务器发送的正确的 FIN_ACK 包，则进入第三次握手。

第二次挥手：服务器收到客户端发来的第一次挥手包后，确认无误后发送第二次挥手包，其标志位为 FIN_ACK。等待两个 CLOCKS_PER_SEC，如果在此期间没有收到客户端发来的包的话则说明客户端正确接收，第二次挥手结束。如果收到了来自客户端的包则需要重传第二次挥手。

第三、四次挥手机制与第一、二次相同，只是 FIN 发送端更改为服务器，FIN_ACK 为客户端。自此正式断开连接。

客户端断开连接函数：

```

1  int disconnect(SOCKET& client, SOCKADDR_IN& ClientAddr, int len)
2  {
3      // 设置为非阻塞模式，避免卡在recvfrom
4      int iMode = 1; // 0: 阻塞
5      ioctlsocket(client, FIONBIO, (u_long FAR*) & iMode); // 非阻塞设置
6      // 第一次挥手，客户端发送FIN
7      UDPpacket pack1;
8      pack1.head.flags = FIN;
9      pack1.head.Check = packetcheck((u_short*)&pack1, sizeof(pack1));
10     // 发送缓冲区
11     char* buffer1 = new char[sizeof(pack1)];
12     memcpy(buffer1, &pack1, sizeof(pack1));
13     int flag = sendto(client, buffer1, sizeof(pack1), 0, (sockaddr*)&
        ClientAddr, len);
14     if (flag == -1)
15     {
16         now = time(nullptr);
17         curr_time = ctime(&now);
18         cout << curr_time << "[System]:";
19         cout << "----- 第一次挥手发送失败 -----" << endl;
20     }
21     else
22     {
23         now = time(nullptr);
24         curr_time = ctime(&now);
25         cout << curr_time << "[System]:";
26         cout << "----- 第一次挥手发送成功 -----" << endl;
27     }
28     // 记录第一次挥手发送时间
29     clock_t starttime = clock();
30     // 第二次挥手，接收服务器发来的FIN_ACK
31     UDPpacket pack2;
32     // 缓冲区
33     char* buffer2 = new char[sizeof(pack2)];
34     // 接收第二次挥手请求信息
35     while (1)
36     {
37         if (clock() - starttime >= 2 * CLOCKS_PER_SEC)
38         {
39             now = time(nullptr);
40             curr_time = ctime(&now);
41             cout << curr_time << "[System]:";
42             // 超时了，重传
43             cout << "超时重发第一次挥手" << endl;
44             sendto(client, buffer1, sizeof(pack1), 0, (sockaddr*)&

```

```

    &ClientAddr, len);
45     starttime = clock();
46 }
47 flag = recvfrom(client, buffer2, sizeof(pack2), 0, (sockaddr
    *)&ClientAddr, &len);
48 if (flag <= 0 )
49 {
50     continue;
51 }
52 //接收成功
53 memcpy(&(pack2), buffer2, sizeof(pack2));
54 u_short res = packetcheck((u_short*)&pack2, sizeof(pack2));
55 //是第二次挥手
56 if (pack2.head.flags == FIN_ACK && res == 0)
57 {
58     now = time(nullptr);
59     curr_time = ctime(&now);
60     cout << curr_time << "[System]:";
61     cout << "----- 第二次挥手接收成功 -----" <<
        endl;
62     break;
63 }
64 //传来的包不正确则一直等待
65 }
66 starttime = clock();
67 //等待两个时钟周期，跟server同步
68 while (clock() - starttime <= CLOCKS_PER_SEC)
69 {
70
71 }
72 //接收第三次挥手信息，头部为ACK
73 UDPpacket pack3;
74 //缓冲区
75 char* buffer3 = new char[sizeof(pack3)];
76 //接收第三次挥手请求信息
77 while (1)
78 {
79     flag = recvfrom(client, buffer3, sizeof(pack3), 0, (sockaddr
    *)&ClientAddr, &len);
80     if (flag <= 0)
81     {
82         continue;
83     }
84     //接收成功
85     memcpy(&(pack3), buffer3, sizeof(pack3));
86     u_short res = packetcheck((u_short*)&pack3, sizeof(pack3));
87     //是第三次挥手
88     if (pack3.head.flags == FIN && res == 0)
```

```

89         {
90             now = time(nullptr);
91             curr_time = ctime(&now);
92             cout << curr_time << "[System]:";
93             cout << "————第三次挥手接收成功————" <<
                endl;
94             break;
95         }
96         //否则一直处于等待状态
97     }
98     //第四次挥手，客户端发送FIN_ACK
99     UDPpacket pack4;
100     pack4.head.flags = FIN_ACK;
101     pack4.head.Check = packetcheck((u_short*)&pack4, sizeof(pack4));
102     //发送缓冲区
103     char* buffer4 = new char[sizeof(pack4)];
104     memcpy(buffer4, &pack4, sizeof(pack4));
105     flag = sendto(client, buffer4, sizeof(pack4), 0, (sockaddr*)&
        ClientAddr, len);
106     if (flag == -1)
107     {
108         now = time(nullptr);
109         curr_time = ctime(&now);
110         cout << curr_time << "[System]:";
111         cout << "————第四次挥手发送失败————" << endl;
112     }
113     else
114     {
115         now = time(nullptr);
116         curr_time = ctime(&now);
117         cout << curr_time << "[System]:";
118         cout << "————第四次挥手发送成功————" << endl;
119     }
120     starttime = clock(); //第四次挥手发送时间
121     while (clock() - starttime <= 2 * CLOCKS_PER_SEC)
122     {
123         //没收到
124         if (recvfrom(client, buffer3, sizeof(pack1), 0, (sockaddr*)&
            ClientAddr, &len) <= 0)
125         {
126             continue;
127         }
128         else //收到了，说明第四次挥手没发出去，需要重新发送
129         {
130             now = time(nullptr);
131             curr_time = ctime(&now);
132             cout << curr_time << "[System]:";
133             cout << "超时重发第四次挥手" << endl;

```

```

134         sendto(client, buffer4, sizeof(pack4), 0, (sockaddr*)
135             &ClientAddr, len);
136         //更新发送时间
137         starttime = clock();
138     }
139     return 1;
140 }

```

服务器端断开连接函数:

```

1  int disconnect(SOCKET& server, SOCKADDR_IN& ServerAddr, int& len)
2  {
3      //设置为非阻塞模式, 避免卡在recvfrom
4      int iMode = 1; //0: 阻塞
5      ioctlsocket(server, FIONBIO, (u_long FAR*) & iMode); //非阻塞设置
6      UDPpacket pack1;
7      //缓冲区
8      char* buffer1 = new char[sizeof(pack1)];
9      int flag;
10     //接收第一次挥手请求信息
11     while (1)
12     {
13         flag = recvfrom(server, buffer1, sizeof(pack1), 0, (sockaddr
14             *)&ServerAddr, &len);
15         //没有收到消息则一直循环
16         if ( flag <= 0)
17         {
18             continue;
19         }
20         //接收成功
21         memcpy(&(pack1), buffer1, sizeof(pack1));
22         u_short res = packetcheck((u_short*)&pack1, sizeof(pack1));
23         //是第一次握手
24         if (pack1.head.flags == FIN && res == 0)
25         {
26             now = time(nullptr);
27             curr_time = ctime(&now);
28             cout << curr_time << "[System]:";
29             cout << "----- 第一次挥手接收成功 -----" <<
30                 endl;
31             break;
32         }
33         //否则就一直等传送来正确的包
34     }
35     //第二次挥手, 服务器发送FIN_ACK
36     UDPpacket pack2;
37     pack2.head.flags = FIN_ACK;
38     pack2.head.Check = packetcheck((u_short*)&pack2, sizeof(pack2));

```

```
37 //发送缓冲区
38 char* buffer2 = new char[ sizeof(pack2)];
39 memcpy(buffer2, &pack2, sizeof(pack2));
40 flag = sendto(server, buffer2, sizeof(pack2), 0, (sockaddr*)&
    ServerAddr, len);
41 if (flag == -1)
42 {
43     now = time(nullptr);
44     curr_time = ctime(&now);
45     cout << curr_time << "[System]:";
46     cout << "----- 第二次挥手发送失败 -----" << endl;
47 }
48 else
49 {
50     now = time(nullptr);
51     curr_time = ctime(&now);
52     cout << curr_time << "[System]:";
53     cout << "----- 第二次挥手发送成功 -----" << endl;
54 }
55 clock_t starttime = clock(); //记录第二次挥手发送时间
56 while (clock() - starttime <= 2 * CLOCKS_PER_SEC)
57 {
58     //没收到
59     flag = recvfrom(server, buffer1, sizeof(pack1), 0, (sockaddr*)
        &ServerAddr, &len);
60     if (flag <= 0)
61     {
62         continue;
63     }
64     else //收到了, 说明第二次挥手没发出去, 需要重新发送
65     {
66         now = time(nullptr);
67         curr_time = ctime(&now);
68         cout << curr_time << "[System]:";
69         cout << "超时重发第二次挥手" << endl;
70         sendto(server, buffer2, sizeof(pack2), 0, (sockaddr*)
            &ServerAddr, len);
71         //更新发送时间
72         starttime = clock();
73     }
74 }
75
76 //第三次挥手, 服务器要发送FIN
77 UDPpacket pack3;
78 pack3.head.flags = FIN;
79 pack3.head.Check = packetcheck((u_short*)&pack3, sizeof(pack3));
80 //发送缓冲区
81 char* buffer3 = new char[ sizeof(pack3)];
```

```

82     memcpy(buffer3, &pack3, sizeof(pack3));
83     flag = sendto(server, buffer3, sizeof(pack3), 0, (sockaddr*)&
        ServerAddr, len);
84     if (flag == -1)
85     {
86         now = time(nullptr);
87         curr_time = ctime(&now);
88         cout << curr_time << "[System]:";
89         cout << "----- 第三次挥手发送失败 -----" << endl;
90     }
91     else
92     {
93         now = time(nullptr);
94         curr_time = ctime(&now);
95         cout << curr_time << "[System]:";
96         cout << "----- 第三次挥手发送成功 -----" << endl;
97     }
98     starttime = clock(); // 第三次挥手发送时间
99     // 接收第四次挥手请求信息
100    UDPpacket pack4;
101    // 缓冲区
102    char* buffer4 = new char[sizeof(pack4)];
103    while (1)
104    {
105        if (clock() - starttime >= 2 * CLOCKS_PER_SEC)
106        {
107            starttime = clock();
108            // 重新发送第三次挥手
109            now = time(nullptr);
110            curr_time = ctime(&now);
111            cout << curr_time << "[System]:";
112            cout << "超时重发第三次挥手" << endl;
113            sendto(server, buffer3, sizeof(pack3), 0, (sockaddr*)&
                ServerAddr, len);
114        }
115        flag = recvfrom(server, buffer4, sizeof(pack4), 0, (sockaddr*)
            &ServerAddr, &len);
116        if (flag <= 0)
117        {
118            continue;
119        }
120        // 接收成功
121        memcpy(&(pack4), buffer4, sizeof(pack4));
122        u_short res = packetcheck((u_short*)&pack4, sizeof(pack4));
123        // 是第四次挥手
124        if (pack4.head.flags == FIN_ACK && res == 0)
125        {
126            now = time(nullptr);

```

```

127         curr_time = ctime(&now);
128         cout << curr_time << "[System]:";
129         cout << "————第四次挥手接收成功————" <<
            endl;
130         break;
131     }
132     //否则的话一直循环等待
133 }
134 return 1;
135 }

```

(四) 差错检测

客户端在发送数据包时，先设置该数据包头部中的序列号 seq、数据长度 len、标志位 flags，将校验和 check 置 0，之后将数据部分用文件信息填充。之后调用 packetcheck 函数，其作用为将整个数据包按 16 位为一组进行划分，不足 16 位的添加 0 补齐，之后进行直接相加的运算（UDP 发送端真正校验和计算中是按照反码计算，但是由于数据本身就以反码形式计算，因此代码中直接相加），如果相加出现进位，则舍弃进位，所得结果 +1，最终返回 16 位计算结果的反码。

服务器在接收数据包时可以直接调用 packetcheck 函数计算校验结果是否为 0，如果为 0 则说明数据包无误（在 UDP 接收端中是将所有的数据以及包头部按照 16 位反码相加，其无误的结果应该为 1111 1111 1111 1111，但是由于 packetcheck 最终返回值取反，因此调用此函数后无误结果应为 0）。

具体 packetcheck 函数实现如下：

```

1  u_short packetcheck(u_short* packet, int packelength)
2  {
3      register u_long sum = 0;
4      int count = (packelength + 1) / 2; // 两个字节的计算
5      u_short* buf = (u_short*)malloc(packelength + 1);
6      memset(buf, 0, packelength + 1);
7      memcpy(buf, packet, packelength);
8      //cout << "count=" << packelength << endl;
9      while (count--)
10     {
11         sum += *buf++;
12         if (sum & 0xFFFF0000)
13         {
14             sum &= 0xFFFF;
15             sum++;
16         }
17     }
18     return ~(sum & 0xFFFF);
19 }

```

(五) 确认重传

此处重传分为两种情况：超时重传和差错重传。

对于超时重传，设定时间上限为 `2CLOCKS_PER_SEC`，如果超过这段时间时客户端仍未收到服务器发来的 ACK 包，说明之前发送的数据包或者服务器传来的确认包可能丢失了，因此此处理办法均为客户端重发上一个数据包。

而对于差错重传，服务器接收到的数据包若不正确 (`check! = 0`) 则返回上一个数据包的确认包，客户端接收到后重发上一个数据包；客户端接收到的确认包若不正确 (`check! = 0`) 则什么也不做，直到超时重传上一个数据包。

(六) 流量控制

使用停等机制，具体实现方式见【协议设计】

(七) 协议设计

本次实验基于 `rdt3.0` 以及停等机制设计了协议，在正式传输数据时大致流程如下：客户端向服务器发送 `seq=0` 的数据包：此时 `Cseq=0, SSeq=0`

A. `Seq=0` 数据包在传输过程中丢失，服务器一直收不到包，仍然处于等待状态（不发送 ACK 包），一直到客户端发现已经超时未收到 ACK 包，客户端重传 `seq=0` 的数据包。

B. 服务器收到数据包，假如校验和错误或者收到的包不是当前等待的包 (`SSeq! = seq`)，则应该返回 `Seq=SSeq (0)` 的 ACK 包。收到的数据包均正确，应该将 `SSeq+=1`，传回 `Seq=SSeq (1)` 的 ACK 包。

C. 返回的 `Seq=0` 或者 `Seq=1` 的 ACK 包丢失，则客户端会发现超时未收到 ACK 包，因此重传 `Seq=0` 的数据包。

D. 客户端收到返回的 `Seq=0` 的 ACK 包，如果校验和正确，此时 `Seq==Cseq`，因此重新传送 `Seq=0` 的数据包。如果校验和错误，仍然应该重新发送 `Seq=0` 的数据包。

E. 客户端收到返回的 `Seq=1` 的 ACK 包，如果校验和正确，此时 `Seq==Cseq+1`，因此将 `Cseq+1`，接下来传送 `Seq=Cseq=1` 的数据包。如果校验和错误，则重发 `Seq=0` 的数据包。

F. 重复以上循环即可

G. 当客户端发送到最后一个包时，将最后一个数据包的标志位设置为 `FINAL`，服务器由头部标志位判断出其为最后一个包，因此将对此数据包的返回确认包头部设置为 `FINAL_ACK`

H. 客户端若收到确认包头部为 ACK 或者超时未收到返回确认包或者收到的返回确认包校验和错误则重发最后一个数据包，若收到完全正确的 `FINAL_ACK` 确认包则代表文件已经全部正确传输，因此跳出循环。

I. 服务器发出最后一个确认包后等待 2 倍的 `CLOCKS_PER_SEC`，倘若此时间内没有收到任何包，则说明客户端已经完成数据传输并且全部确认，服务器结束接收消息；如果收到了数据包，说明之前发出的确认包可能出了问题，重发确认包。

客户端发送文件信息函数：

```

1  int sendmessage(int messagelength, char* message, int addrlength, SOCKET
    & client, SOCKADDR_IN & ServerAddr)
2  {
3      //设置为非阻塞模式，避免卡在recvfrom
4      int iMode = 1; //0: 阻塞
5      ioctlsocket(client, FIONBIO, (u_long FAR*) & iMode); //非阻塞设置
6      //计算要传多少个包
7      int packetnum = messagelength / 2048;
8      if (messagelength % 2048) //向上取整
9      {

```



```
10         packetnum += 1;
11     }
12     now = time(nullptr);
13     curr_time = ctime(&now);
14     cout << curr_time << "[System]:";
15     cout << "共有" << messagelength << "字节，因此共";
16     cout << "要传送" << packetnum << "个包" << endl;
17     int sequence = 0; // 记录要发送的数据包的序列号
18     clock_t starttime;
19     UDPpacket rcvpacket; // 来自服务器发回的数据包
20     UDPpacket sendpack; // 发送包
21     char* sendbuffer = new char[sizeof(sendpack)];
22     // 接收缓冲区
23     char* rcvbuffer = new char[sizeof(rcvpacket)];
24     while (1)
25     {
26         // 当前要发送的是最后一个包了
27         if (sequence == packetnum - 1)
28         {
29             sendpack.head.flags = FINAL;
30             sendpack.head.len = messagelength - 2048 * sequence
31                 ; // 数据部分长度
32             sendpack.head.seq = sequence; // 序列号
33             sendpack.head.Check = 0;
34             // 消息复制 报错——内存太小
35             memcpy(&(sendpack.data), message + 2048 * sequence,
36                 sendpack.head.len);
37             // 设置校验和
38             sendpack.head.Check = packetcheck((u_short*)&sendpack
39                 , sizeof(sendpack));
40             // 进行传输
41             memcpy(sendbuffer, &(sendpack), sizeof(sendpack));
42             int flag = sendto(client, sendbuffer, sizeof(sendpack)
43                 , 0, (sockaddr*)&ServerAddr, addrlen);
44             if (flag == -1)
45             {
46                 now = time(nullptr);
47                 curr_time = ctime(&now);
48                 cout << curr_time << "[System]:";
49                 cout << "包seq" << sendpack.head.seq << "发送
50                     失败" << endl;
51             }
52             else
53             {
54                 now = time(nullptr);
55                 curr_time = ctime(&now);
56                 cout << curr_time << "[System]:";
57                 cout << "成功发送包seq: " << sendpack.head.
```

```

53         seq << "    datalength: " << sendpack.head.
54         len << "    Check: " << sendpack.head.Check
55         << "    flags: " << (u_short)sendpack.head
56         .flags << endl;
57     }
58     //此时我需要收到来自服务器的FINAL_ACK包表示结束传送
59     starttime = clock();//发送时间
60     UDPpacket recvpacket;//来自服务器发回的数据包
61     //接收缓冲区
62     char* recvbuffer = new char[sizeof(recvpacket)];
63     while (recvfrom(client, recvbuffer, sizeof(recvpacket)
64         ), 0, (sockaddr*)&ServerAddr, &addrlength) <= 0)
65     {
66         if (clock() - starttime >= 2 * CLOCKS_PER_SEC
67             )
68         {
69             //超时重传
70             now = time(nullptr);
71             curr_time = ctime(&now);
72             cout << curr_time << "[System]:";
73             cout << "----- 超时重传包
74             -----" << sequence << endl
75             ;
76             //sendpacket(1, message + 2048 *
77                 sequence, messagelength - 2048 *
78                 sequence, sequence, addrlength,
79                 client, ServerAddr);
80             sendpack.head.flags = FINAL;
81             sendpack.head.len = messagelength -
                2048 * sequence;//数据部分长度
            sendpack.head.seq = sequence;//序列号
            sendpack.head.Check = 0;
            //消息复制 报错——内存太小
            memcpy(&(sendpack.data), message +
                2048 * sequence, sendpack.head.
                len);
            //设置校验和
            sendpack.head.Check = packetcheck((
                u_short*)&sendpack, sizeof(
                sendpack));
            //进行传输
            memcpy(sendbuffer, &(sendpack),
                sizeof(sendpack));
            int flag = sendto(client, sendbuffer,
                sizeof(sendpack), 0, (sockaddr*)
                &ServerAddr, addrlength);
            if (flag == -1)
            {

```

```

82         now = time(nullptr);
83         curr_time = ctime(&now);
84         cout << curr_time << "[System
            ]:";
85         cout << "包seq" << sendpack.
            head.seq << "发送失败" <<
            endl;
86     }
87     else
88     {
89         now = time(nullptr);
90         curr_time = ctime(&now);
91         cout << curr_time << "[System
            ]:";
92         cout << "成功发送包seq: " <<
            sendpack.head.seq << "
            datalength: " << sendpack
            .head.len << " Check: "
            << sendpack.head.Check <<
            " flags: " << (u_short)
            sendpack.head.flags <<
            endl;
93     }
94
95     starttime = clock();
96 }
97 }
98 //收到了ACK包
99 u_short res = packetcheck((u_short*) &recvpacket,
    sizeof(recvpacket));
100 if (recvpacket.head.flags == FINAL_ACK && res == 0)
101 {
102     now = time(nullptr);
103     curr_time = ctime(&now);
104     cout << curr_time << "[System]:";
105     //文件全部正确传送, 结束
106     cout << "文件已经全部正确传送完毕" << endl;
107     //等待两个时钟周期以跟server同步
108     starttime = clock();
109     while (clock() - starttime <= 2 *
        CLOCKS_PER_SEC)
110     {
111     }
112     return 1;
113 }
114 else
115 {
116     //说明最后一个包传送不成功

```

```

117         //那就需要重传最后一个包
118         now = time(nullptr);
119         curr_time = ctime(&now);
120         cout << curr_time << "[System]:";
121         cout << "————— 差错重传包 —————"
122             << sequence << endl;
123         continue;
124     }
125     else//代表传的不是最后一个
126     {
127         //要传送的长度为2048
128         sendpack.head.flags = 0;
129         sendpack.head.len = 2048;//数据部分长度
130         sendpack.head.seq = sequence;//序列号
131         sendpack.head.Check = 0;
132         //消息复制 报错——内存太小
133         memcpy(&(sendpack.data), message + 2048 * sequence,
134             sendpack.head.len);
135         //设置校验和
136         sendpack.head.Check = packetcheck((u_short*)&sendpack,
137             sizeof(sendpack));
138         //进行传输
139         memcpy(sendbuffer, &(sendpack), sizeof(sendpack));
140         int flag = sendto(client, sendbuffer, sizeof(sendpack),
141             0, (sockaddr*)&ServerAddr, addrlen);
142         if (flag == -1)
143         {
144             now = time(nullptr);
145             curr_time = ctime(&now);
146             cout << curr_time << "[System]:";
147             cout << "包seq" << sendpack.head.seq << "发送失败" << endl;
148         }
149         else
150         {
151             now = time(nullptr);
152             curr_time = ctime(&now);
153             cout << curr_time << "[System]:";
154             cout << "成功发送包seq:" << sendpack.head.
155                 seq << " datalength:" << sendpack.head.

```

```

156         ), 0, (sockaddr*)&ServerAddr, &addrlength) <= 0)
157     {
158         if (clock() - starttime > 2 * CLOCKS_PER_SEC)
159         {
160             now = time(nullptr);
161             curr_time = ctime(&now);
162             cout << curr_time << "[System]:";
163             cout << "——— 超时重传 ———" << endl;
164             //走到这里说明需要重发上一个包
165             sendpack.head.flags = 0; //不设置标志位
166             sendpack.head.len = 2048; //数据部分长度
167             sendpack.head.seq = sequence; //序列号
168             sendpack.head.Check = 0;
169             //消息复制 报错——内存太小
170             memcpy(&(sendpack.data), message +
171                 2048 * sequence, sendpack.head.
172                 len);
173             //设置校验和
174             sendpack.head.Check = packetcheck((
175                 u_short*)&sendpack, sizeof(
176                 sendpack));
177             //进行传输
178             memcpy(sendbuffer, &(sendpack),
179                 sizeof(sendpack));
180             int flag = sendto(client, sendbuffer,
181                 sizeof(sendpack), 0, (sockaddr*)
182                 &ServerAddr, addrlength);
183             if (flag == -1)
184             {
185                 now = time(nullptr);
186                 curr_time = ctime(&now);
187                 cout << curr_time << "[System]:";
188                 cout << "包seq" << sendpack.
189                     head.seq << "发送失败" << endl;
190             }
191             else
192             {
193                 now = time(nullptr);
194                 curr_time = ctime(&now);
195                 cout << curr_time << "[System]:";
196                 cout << "成功发送包seq: " <<
197                     sendpack.head.seq << "

```

```

188                                     datalength: " << sendpack
189                                     .head.len << " Check: "
190                                     << sendpack.head.Check <<
191                                     " flags: " << (u_short)
192                                     sendpack.head.flags <<
193                                     endl;
194                                     }
195                                     starttime = clock();
196                                     }
197                                     }
198                                     //走到这里说明收到包
199                                     memcpy(&recvpacket, recvbuffer, sizeof(recvpacket));
200                                     //上一个包传送成功
201                                     u_short res = packetcheck((u_short*)&recvpacket,
202                                     sizeof(recvpacket));
203                                     if (recvpacket.head.flags == ACK && recvpacket.head.
204                                     seq == sequence + 1 && res == 0)
205                                     {
206                                         sequence += 1;
207                                         continue;//继续这个循环, 传下一个
208                                     }
209                                     else if (recvpacket.head.flags == ACK && recvpacket.
210                                     head.seq == sequence && res == 0)
211                                     {
212                                         now = time(nullptr);
213                                         curr_time = ctime(&now);
214                                         cout << curr_time << "[System]:";
215                                         cout << "客户端重传包seq" << sequence << endl
216                                         ;
217                                         //这种情况说明上一个包出问题了, 那就重新再传
218                                         这个。
219                                         continue;
220                                     }
221                                     }
222                                     }
223                                     }

```

服务器接收消息函数:

```

1      u_long ReceiveMessage(char* message, int& addrlength, SOCKET& server,
2      SOCKADDR_IN& ClientAddr)
3      {
4          long int filelength = 0;//记录文件长度
5          UDPpacket recvpacket, sendpacket;
6          //接收缓冲区
7          char *recvbuffer = new char[sizeof(recvpacket)];
8          //发送缓冲区
9          char* sendbuffer = new char[sizeof(sendpacket)];

```

```

9      unsigned int Ssequence = 0;//所期望得到的下一个数据包的编号
10     while (1)
11     {
12         //接收报文长度
13         int recvlength = recvfrom(server, recvbuffer, sizeof(
14             recvpacket), 0, (sockaddr*)&ClientAddr, &addrlength);
15         memcpy(&recvpacket, recvbuffer, sizeof(recvpacket));
16         u_short res = packetcheck((u_short*)&recvpacket, sizeof(
17             recvpacket));
18         if (res != 0)//出错了
19         {
20             cout << "信息传输校验码错误!" << endl;
21             sendpacket.head.flags = ACK;
22             //此时传回的sequence仍然为上一次传送的值
23             //表示客户端之前传来的包错误
24             sendpacket.head.seq = Ssequence;
25             //计算校验和
26             sendpacket.head.Check = packetcheck((u_short*)&
27                 sendpacket, sizeof(sendpacket));
28             memcpy(sendbuffer, &sendpacket, sizeof(sendpacket));
29             //期待重传
30             int flag = sendto(server, sendbuffer, sizeof(
31                 sendpacket), 0, (sockaddr*)&ClientAddr,
32                 addrlength);
33             if (flag == -1)
34             {
35                 cout << "发送失败" << endl;
36             }
37             continue;
38         }
39     }
40     else//至少校验码是对的
41     {
42         //判断一下这是不是最后一个包
43         if (recvpacket.head.flags == FINAL)
44         {
45             //放进消息里面
46             memcpy(message + filelength, recvpacket.data,
47                 recvpacket.head.len);
48             filelength += recvpacket.head.len;
49             now = time(nullptr);
50             curr_time = ctime(&now);
51             cout << curr_time << "[System]:";
52             cout << "——— 最后一个包seq:"<<recvpacket.
53                 head.seq<<"已经接收完毕———" << endl;
54             sendpacket.head.flags = FINAL_ACK;
55             sendpacket.head.Check = 0;
56             sendpacket.head.Check = packetcheck((u_short

```

```

    *)&sendpacket, sizeof(sendpacket));
50 memcpy(sendbuffer, &sendpacket, sizeof(
    sendpacket));
51 int flag = sendto(server, sendbuffer, sizeof(
    sendpacket), 0, (sockaddr*)&ClientAddr,
    addrlen);
52 clock_t starttime = clock();//最后一个ACK的发
    送时间
53 now = time(nullptr);
54 curr_time = ctime(&now);
55 cout << curr_time << "[System]:";
56 cout << "发送最后一个确认包seq:" <<
    sendpacket.head.seq << " datalength:" <<
    sendpacket.head.len << " Check:" <<
    sendpacket.head.Check << " flags:" << (
    u_short)sendpacket.head.flags << endl;
57 //为了防止最后一个包的ACK发送丢失, 需要等2MSL
58 //设置为非阻塞模式, 避免卡在recvfrom
59 int iMode = 1; //0: 阻塞
60 ioctlsocket(server, FIONBIO, (u_long FAR*) &
    iMode);//非阻塞设置
61 while (clock() - starttime <= 2 *
    CLOCKS_PER_SEC)
62 {
63     if (recvfrom(server, recvbuffer,
        sizeof(recvpacket), 0, (sockaddr
        *)&ClientAddr, &addrlen) <= 0)
64     {
65         //一直没有收到包, 说明对面正
        确接收了ACK
66         continue;
67     }
68     //否则需要重传最后一个包的ACK
69     else
70     {
71         now = time(nullptr);
72         curr_time = ctime(&now);
73         cout << curr_time << "[System
            ]:";
74         cout << "重传最后一个确认包
            seq:" << sendpacket.head.
            seq << " datalength:" <<
            sendpacket.head.len << "
            Check:" << sendpacket.
            head.Check << " flags:"
            << (u_short)sendpacket.
            head.flags << endl;
75         sendto(server, sendbuffer,

```



```

76         sizeof(sendpacket), 0, (
77             sockaddr*)&ClientAddr,
78             addrlen);
79     starttime = clock();
80 }
81 //恢复阻塞模式
82 iMode = 0; //0: 阻塞
83 ioctlsocket(server, FIONBIO, (u_long FAR*) &
84     iMode); //非阻塞设置
85 return filelength;
86 }
87 else //后面还有别的包
88 {
89     //如果收到的包不是想要的包的话
90     if (Ssequence != recvpacket.head.seq)
91     {
92         //说明出了问题, 返回ACK
93         sendpacket.head.flags = ACK;
94         sendpacket.head.len = 0;
95         sendpacket.head.Check = 0;
96         //此时仍然应该返回之前的seq
97         sendpacket.head.seq = Ssequence;
98         //计算校验和
99         sendpacket.head.Check = packetcheck((
100             u_short*)&sendpacket, sizeof(
101                 sendpacket));
102         memcpy(sendbuffer, &sendpacket,
103             sizeof(sendpacket));
104         //重发该包的ACK
105         int flag = sendto(server, sendbuffer,
106             sizeof(sendpacket), 0, (sockaddr
107                 *)&ClientAddr, addrlen);
108         if (flag == -1)
109         {
110             //int i=WSAGetLastError();
111             //cout << "wrong code:" << i
112                 << endl;
113             //错误代码10038
114             now = time(nullptr);
115             curr_time = ctime(&now);
116             cout << curr_time << "[System
117                 ]:";
118             cout << "———— server 发
119                 送序列号为: " << (int)
120                     sendpacket.head.seq << "的
121                     ACK失败 ————" << endl;
122             continue; //丢弃该数据包
123         }
124     }
125 }
```

```
110     }
111     else
112     {
113         now = time(nullptr);
114         curr_time = ctime(&now);
115         cout << curr_time << "[System
116             ]:";
117         cout << "----- server发
118             送序列号为: " << (int)
119             sendpacket.head.seq << "
120             的ACK成功-----" <<
121             endl;
122         continue;
123     }
124 }
125 else//收到的数据包是此时服务器想要的
126 {
127     //放进消息里面
128     int cur_length = recvpacket.head.len;
129     memcpy(message+filelength, recvpacket
130         .data, cur_length);
131     filelength += int(recvpacket.head.len
132         );
133     //接下来发送ACK
134     Ssequence += 1;
135     sendpacket.head.flags = ACK;
136     sendpacket.head.seq = Ssequence;
137     sendpacket.head.Check = packetcheck((
138         u_short*)&sendpacket, sizeof(
139         sendpacket));
140     memcpy(sendbuffer, &sendpacket,
141         sizeof(sendpacket));
142     int flag = sendto(server, sendbuffer,
143         sizeof(sendpacket), 0, (SOCKADDR
144             *) &ClientAddr, addrlength);
145     //为什么第三个发送失败? -- buffer太
146     小了.....
147     if (flag == -1)
148     {
149         now = time(nullptr);
150         curr_time = ctime(&now);
151         cout << curr_time << "[System
152             ]:";
153         cout << "-----ACK发送
154             失败-----" << endl;
155     }
156     else
```

```

143     {
144         now = time(nullptr);
145         curr_time = ctime(&now);
146         cout << curr_time << "[System
147             ]:";
148         cout << "发送确认包seq:" <<
149             sendpacket.head.seq << "
150             datalength:" <<
151             sendpacket.head.len << "
152             Check:" << sendpacket.
153             head.Check << " flags:"
154             << (u_short)sendpacket.
155             head.flags << endl;
156     }

```

(八) 性能测试

helloworld.txt 文件传输结果如下:

The image displays two side-by-side screenshots of the Microsoft Visual Studio debug console, showing the results of a file transfer for 'helloworld.txt'.

Left Screenshot (Client Side):

- Sat Nov 19 18:24:04 2022 [System]:正确接收包: 807
- Sat Nov 19 18:24:04 2022 发送确认包seq:808 datalength:0 Check:65
- Sat Nov 19 18:24:04 2022 [System]:server发送序列号为:
- Sat Nov 19 18:24:04 2022 [System]:最后一个包seq:808已经接收
- Sat Nov 19 18:24:04 2022 [System]:发送最后一个确认包seq:808 datal
- Sat Nov 19 18:24:06 2022 [System]:文件内容接收成功
- Sat Nov 19 18:24:06 2022 [System]:文件名为: "helloworld.txt"
- Sat Nov 19 18:24:08 2022 [System]:文件 "helloworld.txt" 已经收到并写
- Sat Nov 19 18:24:08 2022 [System]:第一次挥手接收成功
- Sat Nov 19 18:24:08 2022 [System]:第二次挥手接收成功
- Sat Nov 19 18:24:10 2022 [System]:第三次挥手接收成功
- Sat Nov 19 18:24:11 2022 [System]:第四次挥手接收成功
- Sat Nov 19 18:24:11 2022 [System]:断开连接成功

Right Screenshot (Server Side):

- Sat Nov 19 18:24:04 2022 [System]:成功发送包seq: 805 datalength: 2048 Check: 11223 flags: 0
- Sat Nov 19 18:24:04 2022 [System]:成功发送包seq: 806 datalength: 2048 Check: 11222 flags: 0
- Sat Nov 19 18:24:04 2022 [System]:成功发送包seq: 806 datalength: 2048 Check: 11222 flags: 0
- Sat Nov 19 18:24:04 2022 [System]:成功发送包seq: 807 datalength: 2048 Check: 11221 flags: 0
- Sat Nov 19 18:24:04 2022 [System]:成功发送包seq: 807 datalength: 2048 Check: 11221 flags: 0
- Sat Nov 19 18:24:04 2022 [System]:成功发送包seq: 808 datalength: 1024 Check: 44622 flags: 7
- Sat Nov 19 18:24:04 2022 [System]:文件已经全部正确传送完毕
- Sat Nov 19 18:24:06 2022 [System]:文件大小为: 1655808bytes, 总共传输时间为: 21.585s, 平均吞吐率为: 613688bps
- Sat Nov 19 18:24:08 2022 [System]:第一次挥手发送成功
- Sat Nov 19 18:24:08 2022 [System]:第二次挥手发送成功
- Sat Nov 19 18:24:10 2022 [System]:第三次挥手发送成功
- Sat Nov 19 18:24:11 2022 [System]:第四次挥手发送成功
- Sat Nov 19 18:24:13 2022 [System]:断开连接成功

At the bottom of the right screenshot, a message indicates: "E:\junior1\ComputerNet\final_project\Project3_1Client\Debug\Project3_1Client.exe (进程 19228) 要在调试停止时自动关闭控制台, 请启用 '工具' -> '选项' -> '调试' -> '调试停止时自动关闭控制台' 按任意键关闭此窗口. . ."

图 1: txt 文件传输结果

第一张图片传输结果如下:

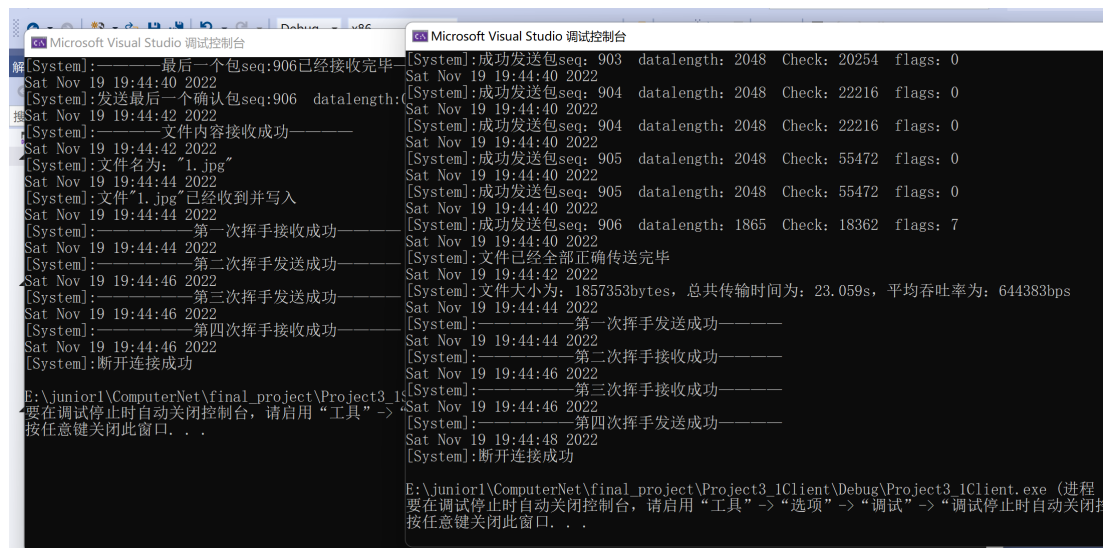


图 2: 图片一传输结果

第二张图片传输结果如下:

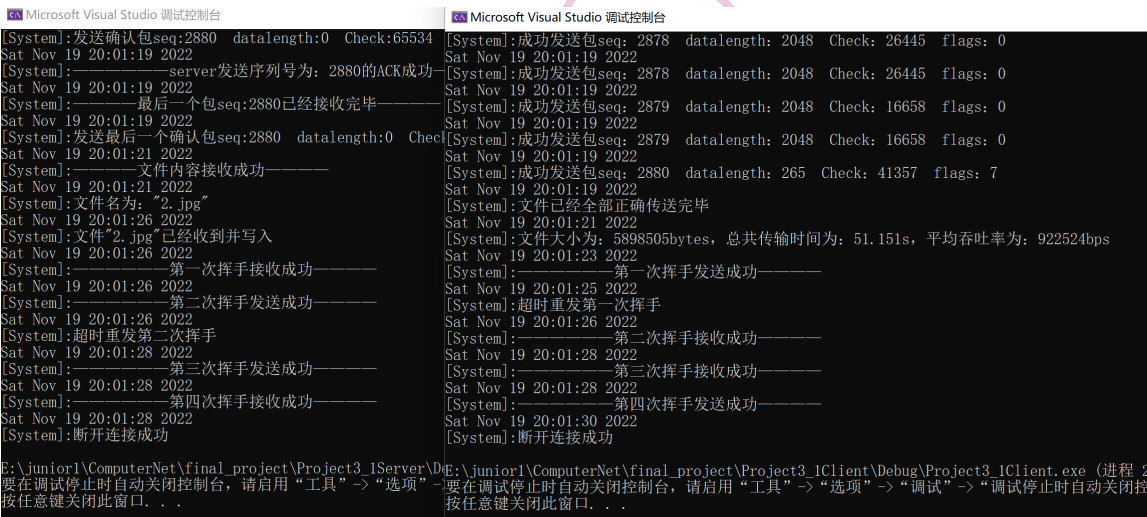
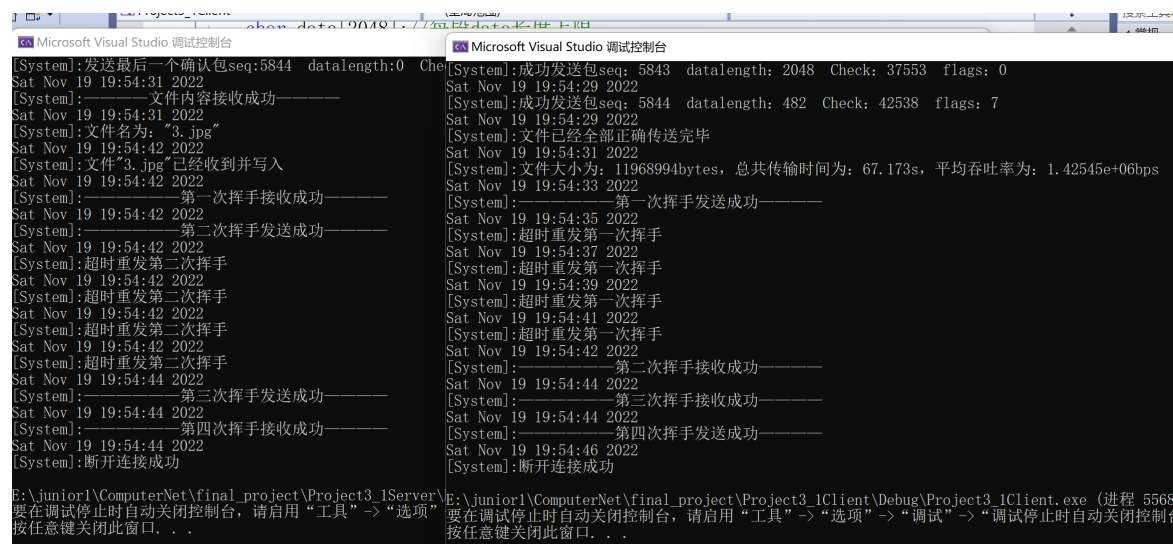


图 3: 图片二传输结果

第三张图片传输结果如下:



The image shows two side-by-side Visual Studio debug console windows. The left window shows the server-side logs for Project3_1Server, and the right window shows the client-side logs for Project3_1Client.exe. Both logs show the successful transmission of a file named '3.jpg' with a size of 11,968,994 bytes. The logs include timestamps, sequence numbers, data lengths, and check values. The client log also shows the file size and total transmission time (67.173s) and average throughput (1.42545e+06bps).

```
[System]:发送最后一个确认包seq:5844 datalength:0 Check:37553 flags:0
Sat Nov 19 19:54:31 2022
[System]:文件内容接收成功
Sat Nov 19 19:54:31 2022
[System]:文件名为: "3.jpg"
Sat Nov 19 19:54:42 2022
[System]:文件"3.jpg"已经收到并写入
Sat Nov 19 19:54:42 2022
[System]:第一次挥手接收成功
Sat Nov 19 19:54:42 2022
[System]:第二次挥手发送成功
Sat Nov 19 19:54:42 2022
[System]:超时重发第二次挥手
Sat Nov 19 19:54:42 2022
[System]:超时重发第二次挥手
Sat Nov 19 19:54:42 2022
[System]:超时重发第二次挥手
Sat Nov 19 19:54:42 2022
[System]:超时重发第二次挥手
Sat Nov 19 19:54:42 2022
[System]:超时重发第二次挥手
Sat Nov 19 19:54:44 2022
[System]:第三次挥手发送成功
Sat Nov 19 19:54:44 2022
[System]:第四次挥手接收成功
Sat Nov 19 19:54:44 2022
[System]:断开连接成功

E:\junior1\ComputerNet\final_project\Project3_1Server\
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”
按任意键关闭此窗口...
```

```
[System]:成功发送包seq: 5843 datalength: 2048 Check: 37553 flags: 0
Sat Nov 19 19:54:29 2022
[System]:成功发送包seq: 5844 datalength: 482 Check: 42538 flags: 7
Sat Nov 19 19:54:29 2022
[System]:文件已经全部正确传送完毕
Sat Nov 19 19:54:31 2022
[System]:文件大小为: 11968994bytes, 总共传输时间为: 67.173s, 平均吞吐率为: 1.42545e+06bps
Sat Nov 19 19:54:33 2022
[System]:第一次挥手发送成功
Sat Nov 19 19:54:35 2022
[System]:超时重发第一次挥手
Sat Nov 19 19:54:37 2022
[System]:超时重发第一次挥手
Sat Nov 19 19:54:39 2022
[System]:超时重发第一次挥手
Sat Nov 19 19:54:41 2022
[System]:超时重发第一次挥手
Sat Nov 19 19:54:42 2022
[System]:第二次挥手接收成功
Sat Nov 19 19:54:44 2022
[System]:第三次挥手接收成功
Sat Nov 19 19:54:44 2022
[System]:第四次挥手发送成功
Sat Nov 19 19:54:46 2022
[System]:断开连接成功

E:\junior1\ComputerNet\final_project\Project3_1Client\Debug\Project3_1Client.exe (进程 5568)
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”
按任意键关闭此窗口...
```

图 4: 图片三传输结果

最终传输结果如下, 可以看到图片均正确传输, 而传输结果图片打开后与源文件相同, 可知传送成功。

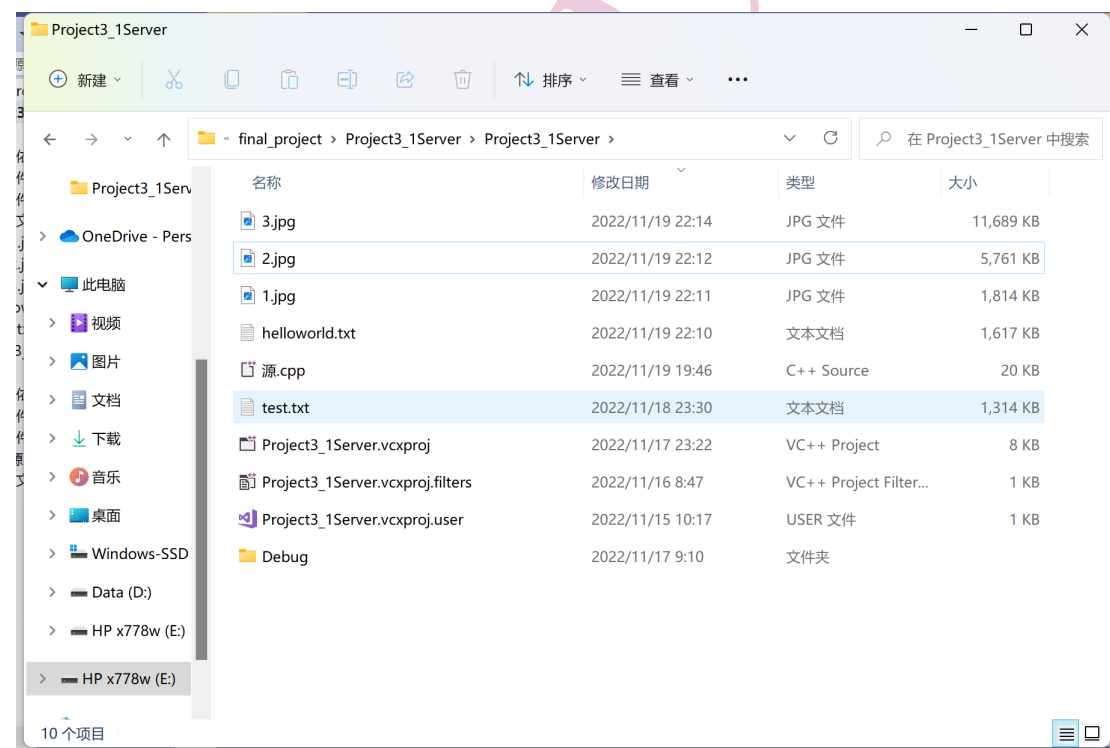


图 5: 本地接收结果