南开大学
Nankai University

南　　开　　大　　学

计算机网络实验报告

实验 3-1：基于 UDP 服务设计可靠传输协议并编程实现

姓名：李雅帆

学号：2213041

年级：2022 级

专业：信息安全

2024 年 11 月 30 日

目录

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、接收确认、超时重传等。流量控制采用停等机制，完成给定测试文件的传输。

- 数据报套接字: UDP
- 协议设计: 数据包格式，发送端和接收端交互，详细完整
- 建立连接、断开连接: 类似 TCP 的握手、挥手功能
- 差错检验: 校验和
- 接收确认、超时重传: rdt2.0、rdt2.1、rdt2.2、rtd3.0 等，亦可自行设计协议
- 单向传输: 发送端、接收端
- 日志输出: 收到/发送数据包的序号、ACK、校验和等，传输时间与吞吐率
- 测试文件：必须使用助教发的测试文件 (1.jpg、2.jpg、3.ipg，helloworld.txt)

二、 协议设计

(一) 数据包 Header 结构

字段大小总计 40 位（5 字节），用于控制和校验数据包，各字段用途如下。

字段名称	位数	描述
数据长度	16 位	表示数据部分的长度，以字节为单位，用于接收端解析数据。
校验和	16 位	用于检测数据包在传输过程中的完整性。
FIN	1 位	标志断开连接请求，若置为 1，表示请求断开连接。
ACK	1 位	确认标志，若置为 1，表示该包为确认包。
SYN	1 位	同步标志，若置为 1，表示请求建立连接。
序列号/确认号	8 位	若为序列号，用于标识当前数据包；若为确认号，表示对上一数据包的确认。

图 1: Header

```
class Header {
        /*
        根据数据长度选择类型:
        char-8位;short-16位;int-32位;long-32/64位;long long-64位
        */
        unsigned short datasize = 0;
        unsigned short sum = 0;
        unsigned char tag = 0;
        unsigned char ack = 0;
public:
        Header() :datasize(0), sum(0), tag(0), ack(0) {};
        void set_tag(unsigned char tag) {
                this->tag = tag;
        }
        void clear_sum() {
                sum = 0;
        }
        void set_sum(unsigned short sum) {
                this->sum = sum;
        }
        unsigned char get_tag() {
                return tag;
        }
        unsigned short get_sum() {
                return sum;
        }
        void set_datasize(unsigned short datasize) {
```

```
28                  this->datasize = datasize;
29          }
30          void set_ack(unsigned char ack) {
31                  this->ack = ack;
32          }
33          int get_datasize() {
34                  return datasize;
35          }
36          unsigned char get_ack() {
37                  return ack;
38          }
39          void print_header() {
40                  printf("datasize:%d, ack:%d, tag:%d\n", get_datasize(),
                        get_ack(), get_tag());
41          }
42  };
```

(二) 数据包 Packet 结构

- 数据包总大小计算
- 如果有要传输的数据的时候就放在 char 数组里面，同时对头部的数据长度进行赋值。
- 如果没有要传输的数据的时候（比如发送确认包 ACK 或者 SYN 包，FIN 包等，不带数据时可以只发送头部）

1. 数据包总大小计算, 数据包的总大小由头部大小和数据部分大小决定。
2. 数据内容操作: 设置数据内容并获取数据内容。
3. 数据长度控制: 通过头部字段 datasize 指定数据部分的长度。
4. 校验和设置与校验: 在发送数据包时，计算并设置校验和, 在接收数据包时，使用校验和验证完整性。
5. 打印数据包信息, 用于调试时输出数据包的详细内容。

```
1   class Packet {
2   private:
3           Header header;
4           char data_content[1024];
5   public:
6           Packet() :header() { memset(data_content, 0, 1024); }
7           Header get_header() {
8                   return header;
9           }
10          void set_datacontent(char* data_content) {
11                  memcpy(this->data_content, data_content, sizeof(data_content)
                        );
12          }
```

```
13          int get_size() {
14                  return sizeof(header) + header.get_datasize();
15          }
16          void print_pkt() {
17                  printf("bytes:%d\nseq:%d\n\n", header.get_datasize(), header.
                        get_ack());
18          }
19          char* get_data_content() {
20                  return data_content;
21          }
22          void set_datasize(int datasize) {
23                  header.set_datasize(datasize);
24          }
25          int get_datasize() {
26                  return header.get_datasize();
27          }
28          void set_ack(unsigned char ack) {
29                  header.set_ack(ack);
30          }
31          u_char get_ack() {
32                  return header.get_ack();
33          }
34          void set_tag(unsigned char tag) {
35                  header.set_tag(tag);
36          }
37          void clear_sum() {
38                  header.clear_sum();
39          }
40          void set_sum(unsigned short sum) {
41                  header.set_sum(sum);
42          }
43          unsigned char get_tag() {
44                  return header.get_tag();
45          }
46          unsigned short get_sum() {
47                  return header.get_sum();
48          }
49          void printpacketmessage() {
50                  printf("Packet size=%d bytes, tag=%d, seq=%d, sum=%d,
                        datasize=%d\n", get_size(), get_tag(), get_ack(), get_sum
                        (), get_datasize());
51          }
52  };
```

(三) 三次握手建立连接

1. 客户端发送 SYN：客户端向服务器发起连接请求。

2. 服务器返回 SYN+ACK：服务器接收到请求后进行响应，表示可以建立连接。

3. 客户端发送 ACK：客户端接收服务器响应后发送确认报文，完成连接的建立。

（四） 超时检测

1. 定时器机制

   - 每次发送报文后，记录当前时间 start。
   - 实验中设置最大等待时间为 MAX_TIME，取值为 $0.5 \times$ CLOCKS_PER_SEC（即 0.5 秒）。
   - 若在 MAX_TIME 内未收到期望的响应，触发超时事件。

```
clock_t start = clock();  // 记录发送时间
while (recvfrom(socket, buffer, buffer_size, 0, (sockaddr*)&addr, &addr_len)
    <= 0) {
    if (clock() - start > MAX_TIME) {
        // 超时事件触发
    }
}
```

2. 超时触发的处理

   - 在超时事件触发时，重新发送上一次的报文。
   - 同时，重置定时器，重新记录当前时间 start。

3. 超时场景

   - 三次握手:
     客户端发送 SYN 报文后等待服务器的 SYN+ACK。
     如果超时未收到 SYN+ACK，客户端重发 SYN 并继续等待。
     服务器在发送 SYN+ACK 报文后等待客户端的 ACK,若超时未收到,则重发 SYN+ACK。
   - 四次挥手:
     客户端发送 FIN 报文后等待服务器的 ACK，若超时未收到，则重发 FIN。
     服务器在发送 FIN+ACK 后等待客户端的 ACK，若超时未收到，则重发 FIN+ACK。
   - 数据传输:
     客户端发送数据包后等待服务器的 ACK，若超时未收到，则重发当前数据包。
     服务器在发送 ACK 后等待下一数据包，若超时未收到，则无需操作（由客户端负责重发）。

```
if (clock() - start > MAX_TIME) {
    // 超时，重新发送报文
    if (sendto(socket, packet, packet_size, 0, (sockaddr*)&addr, addr_len) ==
        -1) {
        cout << "[发送失败]" << endl;
    }
```

```
    // 重新记录发送时间
    start = clock();
    cout << "[超时重传]␣重新发送报文" << endl;
}
```

4. 非阻塞模式的辅助

   - 避免在 recvfrom() 函数中因阻塞造成的程序卡顿，允许在超时检测期间执行其他逻辑。
   - 使用 ioctlsocket() 将套接字设置为非阻塞模式。
   - 在非阻塞模式下，recvfrom() 返回 -1 时表示当前没有接收到数据，而不是等待数据到达。

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

5. 在完成超时检测后，可以将套接字恢复为阻塞模式以确保后续接收的稳定性。示例：

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

（五） 差错检验

本实验的差错检验采用经典的 16 位校验和算法（Checksum）。

1. 校验和计算的原理。

   - 将数据划分为若干个 16 位块（2 字节）。
   - 将所有 16 位块逐块相加，产生一个 16 位的累加和。
   - 如果累加和溢出（超过 16 位），则将高位的溢出部分加回到低位。
   - 对累加和取反，得到校验和。
   - 在数据传输时，将计算得到的校验和附加到报文中。
   - 接收方重新计算收到数据的校验和。
   - 将接收到的校验和与计算结果相加，验证总和是否为 0xFFFF。
   - 若结果为 0xFFFF，说明数据完整；否则认为数据损坏。

cksum 函数来实现 16 位校验和的计算：

```
u_short cksum(u_short* message, int size) {
    int count = (size + 1) / 2;
    u_long sum = 0;

    unsigned short* buf = (unsigned short*)malloc(size);
    memset(buf, 0, size);
    memcpy(buf, message, size);

```

```
    // 遍历每个 16 位块进行累加
    while (count--) {
        sum += *buf++;
        if (sum & 0xffff0000) {
            sum &= 0xffff;
            sum++;
        }
    }
    // 对累加和取反，得到最终的校验和
    u_short result = ~(sum & 0xffff);
    return result;
}
```

2. 差错检验的使用场景

- 三次握手和四次挥手中的校验: 在三次握手和四次挥手过程中，每个报文都附带校验和，用于检测控制报文的完整性。

```
if (cksum((u_short*)&header, sizeof(header)) == 0) {
    cout << "[接收]报文校验成功" << endl;
} else {
    cout << "[接收失败]报文校验错误" << endl;
}
```

- 数据传输中的校验: 在文件数据包的传输过程中，每个数据包都计算校验和并附加到报文头部, 接收方对收到的数据包进行校验，若校验失败则丢弃该包并等待重传。

```
sendpkt->set_sum(cksum((u_short*)sendpkt, sendpkt->get_size()));
```

```
if (cksum((u_short*)recvpkt, sizeof(recvpkt->get_header())) != 0) {
    cout << "[校验失败]数据包损坏，丢弃该包" << endl;
    continue;
}
```

3. 差错处理

- 控制报文的校验失败：若握手或挥手报文校验失败，接收方丢弃该报文并等待重传。例如，服务器未正确接收到客户端的 SYN 报文时：

```
if (header.get_tag() != SYN || cksum((u_short*)&header, sizeof(header)) != 0)
    {
    cout << "[接收失败]报文校验错误，丢弃" << endl;
    continue;
}
```

- 数据包的校验失败：若接收到的数据包校验和错误，接收方直接丢弃该包，发送方在超时后重新发送该数据包，直到接收方确认。例如，校验失败时的输出：

```
[校验失败] 数据包损坏，丢弃该包
```

（六） 四次挥手断开连接

1. 第一次挥手：客户端发送 FIN 请求断开连接。
2. 第二次挥手：服务器接收 FIN，并返回 ACK 确认。
3. 第三次挥手：服务器发送 FIN+ACK 请求断开连接。
4. 第四次挥手：客户端接收 FIN+ACK，并返回 ACK 确认。

三、 代码实现

(一) 服务器端

```
#include<iostream>
#include<WinSock2.h>
#include <WS2tcpip.h>
#include<time.h>
#include <string>
#include<fstream>
using namespace std;
#pragma comment(lib,"ws2_32.lib")

unsigned char FIN = 0b100;
unsigned char ACK = 0b10;
unsigned char SYN = 0b1;
unsigned char ACK_SYN = 0b11;
unsigned char ACK_FIN = 0b110;
unsigned char OVER = 0b111;
double MAX_TIME = 0.5 * CLOCKS_PER_SEC;
int MAXSIZE = 1024;

#include <iostream>
using namespace std;

//打印给定整数的二进制表示，按 4 位分隔。
void printBinary(unsigned int num) {
        // 48位整数，从最高位开始逐位输出
        for (int bit = 47; bit >= 0; --bit) {
                cout << ((num >> bit) & 1);
                if (bit % 4 == 0) {
                        cout << "␣";
                }
        }
        cout << endl;
}


//实现 16 位校验和。
//将数据分成 16 位块，逐块累加。
//若发生溢出，回卷至低 16 位，继续累加。
//最后取反返回，符合标准校验和的计算逻辑。

u_short cksum(u_short* message, int size) {
        int count = (size + 1) / 2;//16位是两个字节
        u_long sum = 0;

        unsigned short* buf = (unsigned short*)malloc(size);
```

```
45          memset(buf, 0, size);
46          memcpy(buf, message, size);
47          //防止message中的数据没有按照16位对其而导致的错误
48
49          while (count--) {
50                  sum += *buf++;
51                  if (sum & 0xffff0000) {
52                          sum &= 0xffff;
53                          sum++;
54                  }
55                  //处理反码溢出的情况
56          }
57          u_short result = ~(sum & 0xffff);
58          return result;
59  }
60
61  #pragma pack(2)
62  class Header {
63          /*
64          根据数据长度选择类型:
65          char-8位;short-16位;int-32位;long-32/64位;long long-64位
66          */
67          unsigned short datasize = 0;
68          unsigned short sum = 0;
69          unsigned char tag = 0;
70          unsigned char ack = 0;
71   public:
72          Header() :datasize(0), sum(0), tag(0), ack(0) {};
73          void set_tag(unsigned char tag) {
74                  this->tag = tag;
75          }
76          void clear_sum() {
77                  sum = 0;
78          }
79          void set_sum(unsigned short sum) {
80                  this->sum = sum;
81          }
82          unsigned char get_tag() {
83                  return tag;
84          }
85          unsigned short get_sum() {
86                  return sum;
87          }
88          void set_datasize(unsigned short datasize) {
89                  this->datasize = datasize;
90          }
91          void set_ack(unsigned char ack) {
92                  this->ack = ack;
```

```
93          }
94          int get_datasize() {
95                  return datasize;
96          }
97          unsigned char get_ack() {
98                  return ack;
99          }
100         void print_header() {
101                 printf("datasize:%d, ack:%d, tag:%d\n", get_datasize(),
                        get_ack(), get_tag());
102         }
103 };
104
105 class Packet {
106 private:
107         Header header;
108         char data_content[1024];
109 public:
110         Packet() :header() { memset(data_content, 0, 1024); }
111         Header get_header() {
112                 return header;
113         }
114         void set_datacontent(char* data_content) {
115                 memcpy(this->data_content, data_content, sizeof(data_content)
                        );
116         }
117         int get_size() {
118                 return sizeof(header) + header.get_datasize();
119         }
120         void print_pkt() {
121                 printf("bytes:%d\nseq:%d\n\n", header.get_datasize(), header.
                        get_ack());
122         }
123         char* get_data_content() {
124                 return data_content;
125         }
126         void set_datasize(int datasize) {
127                 header.set_datasize(datasize);
128         }
129         int get_datasize() {
130                 return header.get_datasize();
131         }
132         void set_ack(unsigned char ack) {
133                 header.set_ack(ack);
134         }
135         u_char get_ack() {
136                 return header.get_ack();
137         }
```

```
138         void set_tag(unsigned char tag) {
139                 header.set_tag(tag);
140         }
141         void clear_sum() {
142                 header.clear_sum();
143         }
144         void set_sum(unsigned short sum) {
145                 header.set_sum(sum);
146         }
147         unsigned char get_tag() {
148                 return header.get_tag();
149         }
150         unsigned short get_sum() {
151                 return header.get_sum();
152         }
153         void printpacketmessage() {
154                 printf("Packet size=%d bytes, tag=%d, seq=%d, sum=%d,
                        datasize=%d\n", get_size(), get_tag(), get_ack(), get_sum
                        (), get_datasize());
155         }
156 };
157
158
159 int connect(SOCKET& server, SOCKADDR_IN& client_addr, int& clientaddr_len) {
160         cout << "开始连接";
161         cout << endl;
162
163         Header header;
164         char* recv_buffer = new char[sizeof(header)];
165
166         //第一次握手: 接受SYN
167         while (true) {
168                 if (recvfrom(server, recv_buffer, sizeof(header), 0, (
                        sockaddr*)&client_addr, &clientaddr_len) == -1)
169                 {
170                         cout << "错误代码: " << WSAGetLastError() << endl;
171                         return -1;
172                 }
173
174                 memcpy(&header, recv_buffer, sizeof(header));
175                 if (header.get_tag() == SYN && cksum((u_short*)&header,
                        sizeof(header)) == 0)
176                 {
177                         cout << "[recv]" <<"SYN" << endl;
178                         break;
179                 }
180         }
181
```

```
182         char* send_buffer = new char[sizeof(header)];
183         //第二次握手: 发送SYN+ACK
184         header.set_tag(ACK_SYN);
185         header.clear_sum();
186         header.set_sum(cksum((u_short*)&header, sizeof(header)));
187         memcpy(send_buffer, &header, sizeof(header));
188         if (sendto(server, send_buffer, sizeof(header), 0, (sockaddr*)&
                client_addr, clientaddr_len) == -1)
189         {
190                 cout << "[Failed send]" << "ACK 、 SYN" << endl;
191                 return -1;
192         }
193         cout << "[send]" << "SYN 、 ACK" << endl;
194
195         //记录第二次握手的时间
196         clock_t start = clock();
197
198         //设置为非阻塞模式
199         u_long mode = 1;
200         ioctlsocket(server, FIONBIO, &mode);
201
202         //第三次握手: 接收ACK
203         while (recvfrom(server, recv_buffer, sizeof(header), 0, (sockaddr*)&
                client_addr, &clientaddr_len) <= 0) {
204                 //进行超时检测
205                 if (clock() - start > MAX_TIME) {
206                         //超时, 重新发送ACK+SYN
207                         cout << "[超时]" << "正在重新发送 ACK 和 SYN " <<
                                endl;
208
209                         header.set_tag(ACK_SYN);
210                         header.clear_sum();
211                         header.set_sum(cksum((u_short*)&header, sizeof(header
                                )));
212                         memcpy(send_buffer, &header, sizeof(header));
213                         if (sendto(server, send_buffer, sizeof(header), 0, (
                                sockaddr*)&client_addr, clientaddr_len) == -1)
214                         {
215                                 cout << "[发送失败]" << "ACK 、 SYN" << endl;
216                                 return -1;
217                         }
218                         //更新第二次握手的时间
219                         clock_t start = clock();
220                 }
221         }
222
223         //设置为阻塞模式
224         mode = 0;
```

```
225         ioctlsocket(server, FIONBIO, &mode);
226
227         //接收到数据后: 开始检验和检测和ACK的判断
228         memcpy(&header, recv_buffer, sizeof(header));
229         if (header.get_tag() == ACK && cksum((unsigned short*)&header, sizeof
                (header)) == 0) {
230             cout << "[接收]" << "ACK" << endl;
231
232             cout << "连接成功" << endl << endl;
233             return 1;
234         }
235         else {
236             cout << "[接收失败]" << "ACK" << endl;
237             return -1;
238         }
239 }
240
241 int disconnect(SOCKET& server, SOCKADDR_IN& client_addr, int clientaddr_len)
        {
242         cout << "开始断开连接" << endl
243             << endl;
244
245         Header header;
246
247         char* recv_buffer = new char[sizeof(header)];
248         //第一次挥手: 接收FIN
249         while (true) {
250             int length = recvfrom(server, recv_buffer, sizeof(header), 0,
                    (sockaddr*)&client_addr, &clientaddr_len);
251             memcpy(&header, recv_buffer, sizeof(header));
252             if (header.get_tag() == FIN && cksum((unsigned short*)&header
                    , sizeof(header)) == 0) {
253                 cout << "[接收]" << "FIN" << endl;
254                 break;
255             }
256         }
257
258         char* send_buffer = new char[sizeof(header)];
259
260         //开始第二次挥手: 发送ACK
261         header.set_tag(ACK);
262         header.clear_sum();
263         header.set_sum(cksum((unsigned short*)&header, sizeof(header)));
264         memcpy(send_buffer, &header, sizeof(header));
265         if (sendto(server, send_buffer, sizeof(header), 0, (sockaddr*)&
                client_addr, clientaddr_len) == -1) {
266             return -1;
267         }
```

```
268         cout << "[发送]" << "ACK" << endl;
269
270         //开始第三次挥手: 发送FIN+ACK
271         header.set_tag(ACK_FIN);
272         header.clear_sum();
273         header.set_sum(cksum((unsigned short*)&header, sizeof(header)));
274         memcpy(send_buffer, &header, sizeof(header));
275         if (sendto(server, send_buffer, sizeof(header), 0, (sockaddr*)&
                client_addr, clientaddr_len) == -1) {
276             return -1;
277         }
278         clock_t start = clock();
279         cout << "[发送]" << "FIN、ACK" << endl;
280
281         u_long mode = 1;
282         ioctlsocket(server, FIONBIO, &mode);
283
284         //开始第四次挥手: 等待ACK
285         while (recvfrom(server, recv_buffer, sizeof(header), 0, (sockaddr*)&
                client_addr, &clientaddr_len) <= 0) {
286             //检测超时
287             if (clock() - start > MAX_TIME) {
288                 cout << "[超时]" << "正在重新发送ACK和FIN" <<
                        endl;
289
290                 header.set_tag(ACK_FIN);
291                 header.clear_sum();
292                 header.set_sum(cksum((unsigned short*)&header, sizeof
                        (header)));
293                 memcpy(send_buffer, &header, sizeof(header));
294                 if (sendto(server, send_buffer, sizeof(header), 0, (
                        sockaddr*)&client_addr, clientaddr_len) == -1) {
295                     return -1;
296                 }
297                 clock_t start = clock();
298                 //记录第三次挥手的时间
299                 cout << "[发送]" << "FIN、ACK" << endl;
300             }
301         }
302
303         mode = 0;
304         ioctlsocket(server, FIONBIO, &mode);
305
306         //接收到信息, 开始判断是否为ack且是否校验和是否为0
307         memcpy(&header, recv_buffer, sizeof(header));
308         if (header.get_tag() == ACK && cksum((u_short*)&header, sizeof(header
                )) == 0) {
309             cout << "[接收]" << "ACK" << endl;
```

```
310
311                 cout << "断开连接" << endl;
312                 return 1;
313         }
314         else {
315                 //校验包出错
316                 cout << "[接收失败]" << "ACK" << endl;
317                 return -1;
318         }
319 }
320
321 int recvdata(SOCKET& server, SOCKADDR_IN& client_addr, int& clientaddr_len,
        char* data) {
322         cout << "开始接收当前文件" << endl;
323         Packet* recvpkt = new Packet();
324         Header header;
325
326         int seq_predict = 0; //确认号
327
328         int file_len = 0;
329
330         char* recv_buffer = new char[MAXSIZE + sizeof(recvpkt->get_header())
                ];
331         char* send_buffer = new char[sizeof(header)];
332
333         while (true) {
334                 //循环接受所有的数据包, 退出条件是数据包接收完毕
335                 int length = recvfrom(server, recv_buffer, sizeof(recvpkt->
                    get_header()) + MAXSIZE, 0, (SOCKADDR*)&client_addr, &
                    clientaddr_len);
336                 if (length == -1) {
337                         cout << "[接收失败]" << endl;
338                 }
339
340                 memcpy(recvpkt, recv_buffer, sizeof(recvpkt->get_header()) +
                    MAXSIZE);
341                 //判断是否结束, 如果已经是最后一个包, 则退出接收
342                 if (recvpkt->get_tag() == OVER && cksum((u_short*)recvpkt,
                    sizeof(recvpkt->get_header())) == 0) {
343                         cout << "[接收]" <<  "OVER" << endl
344                                 << endl;
345                         break;
346                 }
347
348                 //判断当前包是否是我们需要的包
349                 if (seq_predict != int(recvpkt->get_ack())) {
350                         cout << "该数据包重复发送, 不进行存储! " << endl;
351                         continue;
```

```
352                 }
353
354                 cout << "接收到目标数据包:" << endl;
355                 recvpkt->printpacketmessage();
356
357                 memcpy(data + file_len, recvpkt->get_data_content(), int(
                        recvpkt->get_datasize()));
358
359                 file_len += recvpkt->get_datasize();
360
361                 header.set_tag(ACK);
362                 header.set_datasize(0);
363                 header.clear_sum();
364                 header.set_ack((u_char)seq_predict);
365                 header.set_sum(cksum((u_short*)&header, sizeof(header)));
366
367                 memcpy(send_buffer, &header, sizeof(header));
368
369                 if (sendto(server, send_buffer, sizeof(header), 0, (SOCKADDR
                        *)&client_addr, clientaddr_len) == -1) {
370                     cout << "[接收失败]" << endl;
371                 }
372                 cout << "已发送确认: " << endl;
373                 header.print_header();
374                 cout << endl;
375
376                 seq_predict = (seq_predict + 1) % 256;
377
378         }
379         //文件接收完毕, 发送OVER
380         header.clear_sum();
381         header.set_tag(OVER);
382         header.set_datasize(0);
383         header.set_sum((cksum((u_short*)&header, sizeof(header))));
384         memcpy(send_buffer, &header, sizeof(header));
385         if (sendto(server, send_buffer, sizeof(header), 0, (SOCKADDR*)&
                client_addr, clientaddr_len) == -1) {
386             cout << "[接收失败]" << endl;
387             return -1;
388         }
389         cout << "[发送]" << "OVER" << endl;
390         cout << "成功接收当前文件␣" << endl;
391         return file_len;//返回读取的字节数, 为了之后的存储数据
392 }
393
394 int main() {
395         WSADATA wsadata;
396         int error = WSAStartup(MAKEWORD(2, 2), &wsadata);
```

```
397         if (error != 0) {
398                 perror("WSAStartup␣初始化失败!");
399                 exit(1);
400         }
401 
402         int port = 8888;
403         SOCKADDR_IN server_addr;
404         SOCKET server = socket(AF_INET, SOCK_DGRAM, 0);
405         if (server == SOCKET_ERROR) {
406                 perror("创建套接字失败!");
407                 WSACleanup();
408                 exit(1);
409         }
410 
411         server_addr.sin_family = AF_INET;
412         server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
413         server_addr.sin_port = htons(port);
414 
415         if (bind(server, (SOCKADDR*)&server_addr, sizeof(server_addr)) == -1)
                {
416                 perror("绑定失败!");
417                 closesocket(server);
418                 WSACleanup();
419                 exit(1);
420         }
421 
422         cout << "服务器正在监听端口␣8888" << endl;
423         int len = sizeof(server_addr);
424         connect(server, server_addr, len);
425 
426         char* name = new char[50];
427         char* data = new char[1000000000];
428         int namelen = recvdata(server, server_addr, len, name);
429         int datalen = recvdata(server, server_addr, len, data);
430         string a;
431         for (int i = 0; i < namelen; i++)
432         {
433                 a = a + name[i];
434         }
435         disconnect(server, server_addr, int(sizeof(server_addr)));
436         ofstream fout(a.c_str(), ofstream::binary);
437         for (int i = 0; i < datalen; i++)
438         {
439                 fout << data[i];
440         }
441         fout.close();
442         cout << "文件已成功下载到本地" << endl;
443 }
```

1. 初始化网络环境

服务器启动时需要完成网络环境的初始化：

- 调用 `WSAStartup()` 初始化 Windows 套接字环境。
- 创建一个 UDP 套接字，用于接收和发送数据。
- 将服务器的套接字绑定到指定的端口（本实验中为 8888），使服务器可以监听客户端的请求。
- 打印服务器启动成功的信息。

2. 建立连接（三次握手）

服务器与客户端使用三次握手建立可靠连接，具体步骤如下：

- 第一次握手：
  - 服务器等待接收客户端的 `SYN` 包。
  - 验证接收到的包是否具有正确的标志位 `SYN`，并通过校验和检查确保数据完整性。
- **第二次握手：**
  - 服务器发送 `SYN+ACK` 包，表示接收了客户端的连接请求，并要求客户端确认。
- **第三次握手：**
  - 服务器等待接收客户端的 `ACK` 包。
  - 如果超时未接收到客户端的 `ACK`，服务器会重新发送 `SYN+ACK`。
  - 在成功接收并验证 `ACK` 后，服务器完成连接建立。

3. 接收文件数据

服务器通过循环接收客户端发送的文件数据包，具体流程如下：

- **每次接收数据包时：**
  - 提取数据包的头部和数据部分。
  - 验证数据包的校验和是否正确。
  - 检查数据包的序列号是否与服务器的预期值一致（防止重复包或丢包）。
  - 如果数据包正确且符合预期，提取数据并存储到内存中。
- **每次成功接收数据包后：**
  - 服务器发送 `ACK` 确认包，告知客户端数据已成功接收。
  - 更新预期的序列号。
- 如果服务器接收到标志为 `OVER` 的数据包，表示客户端已发送完全部数据，服务器退出接收循环。

4. 断开连接（四次挥手）

服务器与客户端使用四次挥手断开连接，具体步骤如下：

- **第一次挥手：**
  - 服务器接收客户端发送的 FIN 包，表示客户端请求断开连接。
  - 验证 FIN 包的标志位和校验和。
- **第二次挥手：**
  - 服务器发送 ACK 包，表示确认客户端的断开请求。
- **第三次挥手：**
  - 服务器主动发送 FIN+ACK 包，向客户端请求断开连接。
- **第四次挥手：**
  - 服务器等待接收客户端的 ACK 包。
  - 验证 ACK 的校验和正确后，确认断开连接。

5. 保存文件

在成功接收文件数据后，服务器会将文件数据保存到本地：

- 服务器提取客户端发送的文件名。
- 使用 ofstream 以二进制模式将数据写入本地文件。
- 打印提示，告知用户文件已成功下载到本地。

6. 退出程序

服务器完成以下任务后退出程序：

- 关闭套接字，释放资源。
- 调用 WSACleanup() 清理套接字环境。

(二) 客户端

```
#include<iostream>
#include <WINSOCK2.h>
#include <WS2tcpip.h>
#include <time.h>
#include <string>
#include<fstream>
using namespace std;
#pragma comment(lib,"ws2_32.lib")

const int MAXSIZE = 1024;
unsigned char FIN = 0b100;
```

```
12 unsigned char ACK = 0b10;
13 unsigned char SYN = 0b1;
14 unsigned char ACK_SYN = 0b11;
15 unsigned char ACK_FIN = 0b110;
16 unsigned char OVER = 0b111;
17 double MAX_TIME = 0.5 * CLOCKS_PER_SEC;
18
19 void printBinary(unsigned int num) {
20         // 48位整数, 从最高位开始逐位输出
21         for (int bit = 47; bit >= 0; --bit) {
22                 cout << ((num >> bit) & 1);
23                 if (bit % 4 == 0) {
24                         printf("␣");
25                 }
26         }
27         cout << endl;
28 }
29
30 u_short cksum(u_short* message, int size) {
31         int count = (size + 1) / 2;
32         u_long sum = 0;
33
34         unsigned short* buf = (unsigned short*)malloc(size);
35         memset(buf, 0, size);
36         memcpy(buf, message, size);
37
38         while (count--) {
39                 sum += *buf++;
40                 if (sum & 0xffff0000) {
41                         sum &= 0xffff;
42                         sum++;
43                 }
44
45         }
46         u_short result = ~(sum & 0xffff);
47         return result;
48 }
49
50 #pragma pack(2)
51 class Header {
52         unsigned short datasize = 0;
53         unsigned short sum = 0;
54         unsigned char tag = 0;
55         unsigned char ack = 0;
56  public:
57         Header() :datasize((u_short)0), sum((u_short)0), tag((u_char)0), ack
               ((u_char)0) {};
58         void set_tag(unsigned char tag) {
```

```
59                  this->tag = tag;
60          }
61          void clear_sum() {
62                  sum = 0;
63          }
64          void set_sum(unsigned short sum) {
65                  this->sum = sum;
66          }
67          unsigned char get_tag() {
68                  return tag;
69          }
70          unsigned short get_sum() {
71                  return sum;
72          }
73          void set_datasize(unsigned short datasize) {
74                  this->datasize = datasize;
75          }
76          void set_ack(unsigned char ack) {
77                  this->ack = ack;
78          }
79          int get_datasize() {
80                  return datasize;
81          }
82          unsigned char get_ack() {
83                  return ack;
84          }
85          void print_header() {
86                  printf("datasize:%d, ack:%d\n", get_datasize(), get_ack());
87          }
88  };
89
90  class Packet {
91  private:
92          Header header;
93          char data_content[MAXSIZE];
94   public:
95          Packet() :header() {
96                  memset(data_content, 0, MAXSIZE);
97          }
98          Header get_header() {
99                  return header;
100         }
101         void set_datacontent(char* data_content) {
102                 memcpy(this->data_content, data_content, header.get_datasize
                        ());
103         }
104         int get_size() {
105                 return sizeof(header) + get_datasize();
```

```
106         }
107         void set_datasize(int datasize) {
108                 header.set_datasize(datasize);
109         }
110         int get_datasize() {
111                 return header.get_datasize();
112         }
113         void set_ack(unsigned char ack) {
114                 header.set_ack(ack);
115         }
116         u_char get_ack() {
117                 return header.get_ack();
118         }
119         void set_tag(unsigned char tag) {
120                 header.set_tag(tag);
121         }
122         void clear_sum() {
123                 header.clear_sum();
124         }
125         void set_sum(unsigned short sum) {
126                 header.set_sum(sum);
127         }
128         unsigned char get_tag() {
129                 return header.get_tag();
130         }
131         unsigned short get_sum() {
132                 return header.get_sum();
133         }
134         void printpacketmessage() {
135                 printf("Packet size=%d bytes, tag=%d, seq=%d, sum=%d,
                        datasize=%d\n", get_size(), get_tag(), get_ack(), get_sum
                        (), get_datasize());
136         }
137 };
138
139
140
141 int connect(SOCKET& client, SOCKADDR_IN& serv_addr, int& servaddr_len) {
142         cout << "开始连接" << endl;
143         //三次握手建立连接
144         Header header;
145         //进行第一次握手: 发送SYN
146         header.set_tag(SYN);
147         header.clear_sum();
148         header.set_sum(cksum((unsigned short*)&header, sizeof(header)));
149
150         char* send_buffer = new char[sizeof(header)];
151         memcpy(send_buffer, &header, sizeof(header));
```

```
152         if (sendto(client, send_buffer, sizeof(header), 0, (sockaddr*)&
                serv_addr, servaddr_len) == -1)
153         {
154             cout << "[发送失败]" << "SYN" << endl;
155             return -1;
156         }
157         cout << "[发送]" << "SYN" << endl;
158
159
160         clock_t start = clock();
161
162         u_long mode = 1;
163         ioctlsocket(client, FIONBIO, &mode);
164
165         //第二次握手: 接收ACK+SYN
166         char* recv_buffer = new char[sizeof(header)];
167         while (recvfrom(client, recv_buffer, sizeof(header), 0, (sockaddr*)&
                serv_addr, &servaddr_len) <= 0) {
168             if (clock() - start > MAX_TIME) {
169                 //超时重传, 重新进行第一次握手
170                 header.set_tag(SYN);
171                 header.clear_sum();
172                 header.set_sum(cksum((unsigned short*)&header, sizeof
                        (header)));
173
174                 memcpy(send_buffer, &header, sizeof(header));
175                 if (sendto(client, send_buffer, sizeof(header), 0, (
                        sockaddr*)&serv_addr, servaddr_len) == -1)
176                 {
177                     cout << "[发送失败]" << "SYN" << endl;
178                     return -1;
179                 }
180                 start = clock();
181                 cout << "[超时]" << "SYN" << endl;
182             }
183         }
184
185         mode = 0;
186         ioctlsocket(client, FIONBIO, &mode);
187
188         memcpy(&header, recv_buffer, sizeof(header));
189         if (header.get_tag() == ACK_SYN && cksum((unsigned short*)&header,
                sizeof(header)) == 0) {
190             cout << "[接收]" << "SYN␣、ACK" << endl;
191
192             //如果接收成功, 进行第三次握手: 发送ACK
193             header.set_tag(ACK);
194             header.clear_sum();
```

```
195                 header.set_sum(cksum((unsigned short*)&header, sizeof(header)
                        ));
196                 memcpy(send_buffer, &header, sizeof(header));//将header中的内
                        容复制给发送缓存区
197                 if (sendto(client, send_buffer, sizeof(header), 0, (sockaddr
                        *)&serv_addr, servaddr_len) == -1)
198                 {
199                         cout << "[发送失败]" << "ACK" << endl;
200                         return -1;
201                 }
202                 cout << "[发送]" << "ACK" << endl;
203
204                 cout << "连接成功" << endl;
205         }
206         else {
207                 //收到的数据包有误
208                 cout << "[数据包损坏]" << "正在重新发送␣SYN" << endl << endl;
209                 return -1;
210         }
211
212 }
213
214 int disconnect(SOCKET& client, SOCKADDR_IN& serv_addr, int servaddr_len) {
215         cout << "开始断开连接" << endl;
216
217         Header header;
218         char* send_buffer = new char[sizeof(header)];
219
220         //进行第一次挥手: 发送FIN
221         header.set_tag(FIN);
222         header.clear_sum();
223         header.set_sum(cksum((u_short*)&header, sizeof(header)));
224         memcpy(send_buffer, &header, sizeof(header));
225         if (sendto(client, send_buffer, sizeof(header), 0, (sockaddr*)&
                serv_addr, servaddr_len) == -1)
226         {
227                 cout << "[发送失败]" << "FIN" << endl;
228                 return -1;
229         }
230         cout << "[发送]" << "FIN" << endl;
231
232         clock_t start = clock();
233         //记录第一次挥手时间
234
235         u_long mode = 1;
236         ioctlsocket(client, FIONBIO, &mode);
237
238         //进行第二次挥手: 接受ACK
```

```
239         char* recv_buffer = new char[sizeof(header)];
240         while (recvfrom(client, recv_buffer, sizeof(header), 0, (sockaddr*)&
                serv_addr, &servaddr_len) <= 0)
241         {
242             if (clock() - start > MAX_TIME)
243             {
244                 header.set_tag(FIN);
245                 header.clear_sum();
246                 header.set_sum(cksum((u_short*)&header, sizeof(header
                        )));
247                 memcpy(send_buffer, &header, sizeof(header));
248                 if (sendto(client, send_buffer, sizeof(header), 0, (
                        sockaddr*)&serv_addr, servaddr_len))
249                 {
250                     cout << "[发送失败]" << "FIN" << endl;
251                     return -1;
252                 }
253                 start = clock();
254                 //更新时间
255                 cout << "[超时]" << "正在重新发送FIN" << endl;
256             }
257         }
258
259         //进行校验和检验以及ACK
260         memcpy(&header, recv_buffer, sizeof(header));
261         if (header.get_tag() == ACK && cksum((unsigned short*)&header, sizeof
                (header) == 0)) {
262             cout << "[接收]" << "ACK" << endl;
263
264         }
265         else {
266             //检验包出错
267             cout << "[接收失败]" << "ACK" << endl;
268             return -1;
269         }
270
271         mode = 0;
272         ioctlsocket(client, FIONBIO, &mode);
273
274         //进行第三次挥手: 等待FIN+ACK
275         while (recvfrom(client, recv_buffer, sizeof(header), 0, (sockaddr*)&
                serv_addr, &servaddr_len) != SOCKET_ERROR) {
276             //进行校验和检验, 且对ACK标志位进行检测
277             memcpy(&header, recv_buffer, sizeof(header));
278             if (header.get_tag() == ACK_FIN && cksum((unsigned short*)&
                    header, sizeof(header) == 0)) {
279                 //检测成功
280                 cout << "[接收]" << "FIN、ACK" << endl;
```

```
281
282                         //第四次挥手: 发送ACK
283                         header.set_tag(ACK);
284                         header.clear_sum();
285                         header.set_sum(cksum((u_short*)&header, sizeof(header
                                )));
286                         memcpy(send_buffer, &header, sizeof(header));
287                         if (sendto(client, send_buffer, sizeof(header), 0, (
                                sockaddr*)&serv_addr, servaddr_len) == -1)
288                         {
289                                 cout << "[发送失败]" << "ACK" << endl;
290
291                                 return -1;
292                         }
293                         cout << "[发送]" << "ACK" << endl;
294                         start = clock();
295                         cout << "准备退出" << endl;
296                         break;
297                 }
298         }
299         cout << "已退出连接" << endl;
300         return 1;
301 }
302
303 int send_package(SOCKET& client, SOCKADDR_IN& server_addr, int&
        serveraddr_len, char* data_content, int datasize, int& seq) {
304         Packet* sendpkt = new Packet();
305         sendpkt->set_datasize(datasize);
306         sendpkt->clear_sum();
307         sendpkt->set_ack((unsigned char)seq);
308
309         sendpkt->set_datacontent(data_content);
310
311         sendpkt->set_sum(cksum((u_short*)sendpkt, sendpkt->get_size()));
312
313         cout << "检查数据包内容: " << endl;
314         sendpkt->printpacketmessage();
315
316         if (sendto(client, (char*)sendpkt, sendpkt->get_size(), 0, (SOCKADDR
                *)&server_addr, serveraddr_len) == -1) {
317                 cout << "[发送失败]" << "数据包" << endl;
318                 return -1;
319         }
320         cout << "成功发送数据包: " << endl;
321         sendpkt->printpacketmessage();
322
323         //记录当前时间
324         clock_t start = clock();
```

```
325
326         Header* header = new Header();
327         //等待接收ACK等信息, 同时验证seq
328         while (true) {
329                 u_long mode = 1;
330                 ioctlsocket(client, FIONBIO, &mode);
331
332                 //超时检测
333                 while (recvfrom(client, (char*)header, MAXSIZE, 0, (SOCKADDR
                        *)&server_addr, &serveraddr_len) <= 0) {
334                     if (clock() - start > MAX_TIME) {
335                             cout << "[超时]" << "正在重新发送数据包" <<
                                endl;
336
337                             if (sendto(client, (char*)sendpkt, sendpkt->
                                get_size(), 0, (SOCKADDR*)&server_addr,
                                serveraddr_len) == -1) {
338                                 cout << "[失败]" << "数据包" << endl;
339                                 return -1;
340                             }
341                             //重置发送时间
342                             start = clock();
343                     }
344                 }
345                 //接收到数据, 进行序列号的检测和ACK的确认
346                 if (header->get_ack() == (u_char)seq && header->get_tag() ==
                        ACK) {
347                     cout << "对方已接受到数据包并发送确认:" << endl;
348                     header->print_header();
349                     break;
350                 }
351                 else {
352                     continue;
353                 }
354         }
355         u_long mode = 0;
356         ioctlsocket(client, FIONBIO, &mode);
357
358         return 1;
359 }
360 int send(SOCKET& client, SOCKADDR_IN& server_addr, int& serveraddr_len, char*
        data_content, int datasize) {
361         int package_num = datasize / MAXSIZE + (datasize % MAXSIZE == 0 ? 0 :
                1);
362
363         int seqnum = 0;
364
365         cout << "准备发送当前文件" << endl;
```

```
366
367         for (int i = 0; i < package_num; i++) {
368                 //循环发送所有分开后的数据包
369                 cout << "发送当前文件中的第" << i << "号数据包: " << endl;
370                 int len = 0;
371                 if (i == package_num - 1)
372                         len = datasize - (package_num - 1) * MAXSIZE;
373                 else
374                         len = MAXSIZE;
375
376                 if (send_package(client, server_addr, serveraddr_len,
                        data_content + i * MAXSIZE, len, seqnum) == -1) {
377                         cout << "[发送数据包失败]" << endl;
378                         i--;
379                         continue;
380                 }
381                 seqnum = (seqnum + 1) % 256;
382                 cout << "第" << i << "号数据包发送成功" << endl << endl;
383         }
384
385         Header header;
386         header.set_tag(OVER);
387         header.set_datasize((u_short)0);
388         header.set_ack((u_char)0);
389         header.clear_sum();
390         header.set_sum(cksum((u_short*)&header, sizeof(header)));
391
392         char* send_buffer = new char[sizeof(header)];
393         memcpy(send_buffer, &header, sizeof(header));
394         if (sendto(client, send_buffer, sizeof(header), 0, (SOCKADDR*)&
                server_addr, serveraddr_len) == -1) {
395                 cout << "[发送失败]" << "OVER" << endl << endl;
396                 return -1;
397         }
398         cout << "[发送]" << "OVER" << endl;
399
400         clock_t start = clock();
401
402         u_long mode = 1;
403         ioctlsocket(client, FIONBIO, &mode);
404
405         char* recv_buffer = new char[sizeof(header)];
406
407         while (true)
408         {
409                 while (recvfrom(client, recv_buffer, sizeof(header), 0, (
                        SOCKADDR*)&server_addr, &serveraddr_len) <= 0) {
410                         if (clock() - start > MAX_TIME) {
```

```
411                                 cout << "[超时]" << "重新发送OVER" << endl;
412                                 if (sendto(client, send_buffer, sizeof(header
                                        ), 0, (SOCKADDR*)&server_addr,
                                        serveraddr_len) == -1) {
413                                         cout << "[发送失败]" << "OVER" <<
                                                endl;
414                                 }
415                                 start = clock();
416                         }
417                 }
418
419                 mode = 0;
420                 ioctlsocket(client, FIONBIO, &mode);
421
422                 memcpy(&header, recv_buffer, sizeof(header));
423                 if (header.get_tag() == OVER && cksum((u_short*)&header,
                        sizeof(header)) == 0) {
424                         cout << "[接收]" << "OVER" << endl;
425                         cout << "对方已接受到文件" << endl;
426                         break;
427                 }
428                 else
429                         continue;
430         }
431         return 1;
432 }
433
434 int main() {
435         SOCKADDR_IN server_addr;
436
437         WSADATA wsadata;
438         int err = WSAStartup(MAKEWORD(2, 2), &wsadata);
439         if (err != 0) {
440                 perror("WSAStartup初始化失败!");
441                 exit(1);
442         }
443
444         SOCKET server = socket(AF_INET, SOCK_DGRAM, 0);
445         // 建立套接字
446         if (server == SOCKET_ERROR) {
447                 perror("创建客户端失败!");
448                 WSACleanup();
449                 exit(1);
450         }
451
452         if (InetPton(AF_INET, TEXT("127.0.0.1"), &server_addr.sin_addr.s_addr
                ) != 1) {
453                 perror("地址无效!");
```

```
454         }
455         server_addr.sin_family = AF_INET;
456         server_addr.sin_port = htons(6666);
457 
458         int len = sizeof(server_addr);
459         if (connect(server, server_addr, len) == -1) {
460                 perror("连接失败!");
461                 exit(1);
462         }
463 
464         string file;
465         cout << "请输入要传输的文件名称:" << endl;
466         cin >> file;
467         ifstream fin(file.c_str(), ifstream::binary);
468         int ptr = 0;
469         unsigned char temp = fin.get();
470         int index = 0;
471         char* buffer = new char[1000000000];
472         while (fin) {
473                 buffer[index] = temp;
474                 temp = fin.get();
475                 index++;
476         }
477         fin.close();
478 
479         send(server, server_addr, len, (char*)(file.c_str()), file.length());
480 
481         clock_t start_data = clock();
482         send(server, server_addr, len, buffer, index);
483         clock_t end_data = clock();
484 
485         cout << "传输时间: " << (end_data - start_data) / CLOCKS_PER_SEC << "
                秒" << endl;
486         cout << "吞吐率: " << ((float)index) / ((end_data - start_data) /
               CLOCKS_PER_SEC) << "字节/秒" << endl << endl;
487         disconnect(server, server_addr, int(sizeof(server_addr)));
488 }
```

1. 初始化网络环境

客户端启动时需要完成网络环境的初始化：

- 调用 `WSAStartup()` 初始化 Windows 套接字环境。
- 创建一个 UDP 套接字，用于发送和接收数据。
- 配置服务器的 IP 地址（本实验中为 `127.0.0.1`）和端口号（`6666`）。
- 打印客户端启动成功的信息。

2. 建立连接（三次握手）

客户端与服务器使用三次握手建立可靠连接，具体步骤如下：

- **第一次握手：**
  - 客户端发送 SYN 包，表示请求与服务器建立连接。
  - 构造 SYN 包（设置标志位并计算校验和）。
  - 调用 sendto() 将 SYN 包发送给服务器。
- **第二次握手：**
  - 客户端等待接收服务器的 SYN+ACK 包。
  - 设置套接字为非阻塞模式，处理超时重传的逻辑。
  - 如果接收到 SYN+ACK 包，则通过校验和验证其完整性。
- **第三次握手：**
  - 客户端发送 ACK 包给服务器，表示确认连接成功。
  - 设置标志位为 ACK，计算校验和，并通过 sendto() 将包发送至服务器。
  - 如果成功，则打印连接成功的信息。

3. 读取文件数据

客户端需要将本地文件读取到内存中，以便分块发送：

- 提示用户输入文件名。
- 使用文件输入流 ifstream 以二进制模式打开文件。
- 将文件内容逐字节读取到缓冲区 buffer 中，并记录文件大小。

4. 文件数据发送

客户端通过循环发送文件数据包，具体流程如下：

- 将文件数据分块，每块大小为 1024 字节（MAXSIZE）。
- 逐个发送数据包：
  - 构造数据包，包括头部和数据内容。
  - 设置序列号 seq，并计算校验和。
  - 调用 sendto() 将数据包发送到服务器。
  - 等待服务器返回的 ACK 包：
    - 如果未收到 ACK 或超时，则重新发送当前数据包。
    - 如果收到的 ACK 包序列号正确，更新序列号并继续发送下一个数据包。
- 在所有数据包发送完成后，发送标志为 OVER 的包，通知服务器文件传输结束。
- 等待服务器确认接收到 OVER 包。

5. 断开连接（四次挥手）

客户端与服务器使用四次挥手断开连接，具体步骤如下：

- **第一次挥手：**
  - 客户端发送 FIN 包，表示请求断开连接。
  - 构造 FIN 包（设置标志位并计算校验和）。
  - 调用 sendto() 将 FIN 包发送给服务器。
- **第二次挥手：**
  - 客户端等待接收服务器的 ACK 包，确认服务器已收到断开请求。
  - 设置套接字为非阻塞模式，处理超时重传的逻辑。
- **第三次挥手：**
  - 客户端等待接收服务器的 FIN+ACK 包，表示服务器请求断开连接。
  - 验证 FIN+ACK 包的校验和，确认包的完整性。
- **第四次挥手：**
  - 客户端发送 ACK 包，表示确认服务器的断开请求。
  - 调用 sendto() 将 ACK 包发送至服务器。
  - 打印断开连接成功的信息，并释放资源。

6. 打印传输统计信息

客户端在文件传输完成后，打印相关统计信息：

- **传输时间：**
  - 使用 clock() 记录文件发送开始和结束时间。
  - 计算总传输时间并打印。
- **吞吐率：**
  - 根据文件大小和传输时间，计算吞吐率（单位：字节/秒）。
  - 打印吞吐率，展示文件传输性能。

7. 退出程序

客户端完成以下任务后退出程序：

- 关闭套接字，释放资源。
- 调用 WSACleanup() 清理套接字环境。

四、 程序界面展示

(一) 丢包率和时延

- 如图2所示，服务器端口设为 8888，路由器端口设为 6666。
- 丢包率设为 3%。
- 时延设为 3ms。

图 2: Router

(二) 启动服务器端和客户端

- 如图3所示，服务器端成功启动后会出现“服务器正在监听端口 8888”的字样。
- 客户端启动后进行三次握手连接。

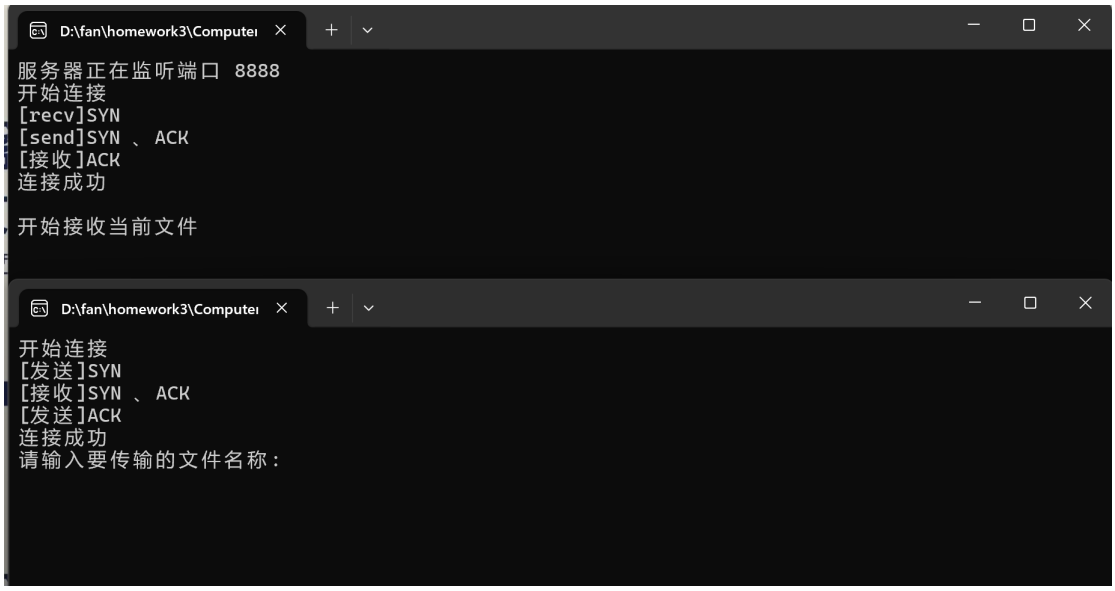

图 3: 启动

(三) 发送文件

- 如图4所示，发送 3.jpg 图片文件。

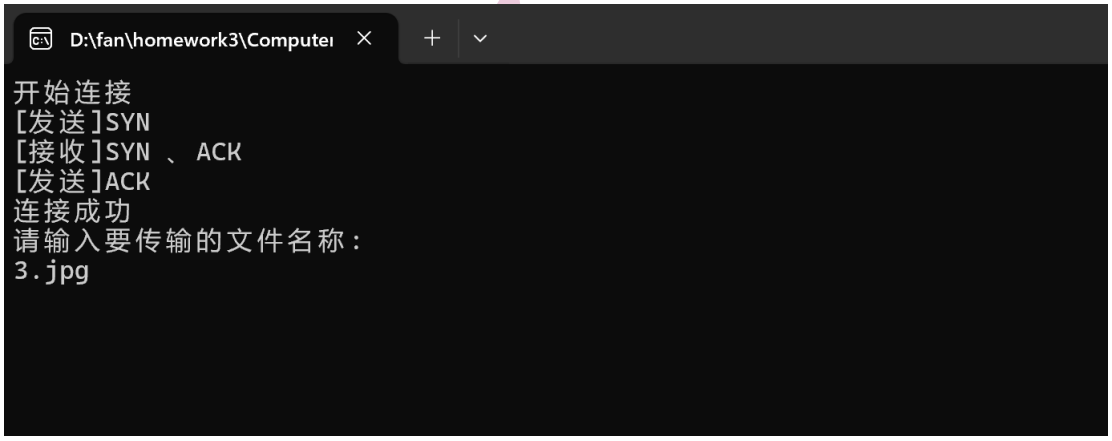

图 4: 发送文件

- 客户端发送成功。

```
发送当前文件中的第11688号数据包：
检查数据包内容：
Packet size=488 bytes, tag=0, seq=168, sum=62972, datasize=482
成功发送数据包：
Packet size=488 bytes, tag=0, seq=168, sum=62972, datasize=482
对方已接受到数据包并发送确认：
datasize:0, ack:168
第11688号数据包发送成功

[发送]OVER
[接收]OVER
对方已接受到文件
传输时间：784秒
吞吐率：15266.6字节/秒

开始断开连接
[发送]FIN
[接收]ACK
[接收]FIN 、 ACK
[发送]ACK
准备退出
已退出连接

D:\fan\homework3\ComputerNetwork\ClientUDP\x64\Debug\ClientUDP.exe (进程 25508)已退出，代码为
 0。
按任意键关闭此窗口．．．
```

图 5: 发送成功

- 服务器端接收成功。

```
Packet size=1030 bytes, tag=0, seq=166, sum=29999, datasize=1024
已发送确认：
datasize:0, ack:166, tag:2

接收到目标数据包：
Packet size=1030 bytes, tag=0, seq=167, sum=32801, datasize=1024
已发送确认：
datasize:0, ack:167, tag:2

接收到目标数据包：
Packet size=488 bytes, tag=0, seq=168, sum=62972, datasize=482
已发送确认：
datasize:0, ack:168, tag:2

[接收]OVER

[发送]OVER
成功接收当前文件
开始断开连接

[接收]FIN
[发送]ACK
[发送]FIN 、 ACK
[接收]ACK
断开连接
文件已成功下载到本地

D:\fan\homework3\ComputerNetwork\ServerUDP\x64\Debug\ServerUDP.exe (进程 27824)已退出，代码为
 0。
按任意键关闭此窗口．．．
```

图 6: 接收成功

- 成功下载到本地。

名称	修改日期	类型	大小
x64	2024/11/20 19:39	文件夹	
2	2024/11/29 21:00	JPG 文件	5,761 KB
3	2024/11/30 22:12	JPG 文件	11,689 KB
ServerUDP.cpp	2024/11/30 19:30	C++ Source	13 KB
ServerUDP.vcxproj	2024/11/20 19:39	VCXPROJ 文件	7 KB
ServerUDP.vcxproj.filters	2024/11/20 19:39	VC++ Project Filter...	1 KB
ServerUDP.vcxproj.user	2024/11/20 18:56	Per-User Project O...	1 KB

图 7: 成功下载到本地

（四） 四次挥手断开连接

- 客户端:

图 8: 客户端四次挥手

- 服务器端:

图 9: 服务器端四次挥手

（五） 传输时间和吞吐率

图 10