

Lab3-2 实验报告

UDP 可靠协议传输

1813265 李彦欣

目录

实验要求-----	3
协议设计-----	4
实验代码-----	10
实验结果-----	19

实验要求

一、具体要求：

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、确认重传。流量控制采用停等机制，完成给定测试文件的传输。

在以上的基础上，将停等机制改成基于滑动窗口的流量控制机制，采用固定窗口大小，支持累积确认，完成给定测试文件的传输。

二、开发环境：

- (1) 集成开发环境：Visual Studio 2019
- (2) 开发语言：c++
- (3) 操作系统：Windows10

协议设计

一、主要协议内容设计：

➤ 数据内容说明：

校验和	数据报文
-----	------

其中数据报文的格式：（序列号表示数据包的序号）

标志位	序列号	数据
-----	-----	----

对于最后一个包，增加一个字节来保存该数据段的长度。

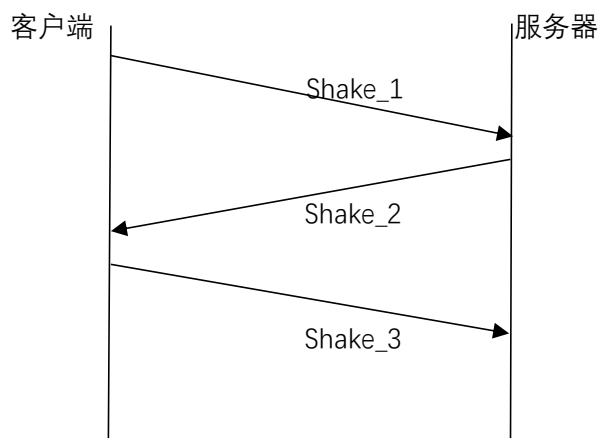
标志位	序列号	长度位	数据
-----	-----	-----	----

【数据包的具体长度说明】：

对于一个数据包的长度是有限制的，对于不是最后一个传输的数据包来说最多数据长度为 253 个字节，即数据报文为 256 个字节。对于传输段最后的不完整的数据来说则按具体长度发送。对于非结尾数据包，标志位的倒数第 4 位为 1；对于结尾数据包，标志位的倒数 4，5 位为 1；因为文件传输期间，可能发送的整个数据段不止一个，因此序号位需要在连接未断开时不断递增，否则接收方可能会混淆不同的阶段的数据。

➤ 握手过程：

三次握手过程的大致流程图如下：



步骤一：客户端发送第一次握手信号 shake_1。

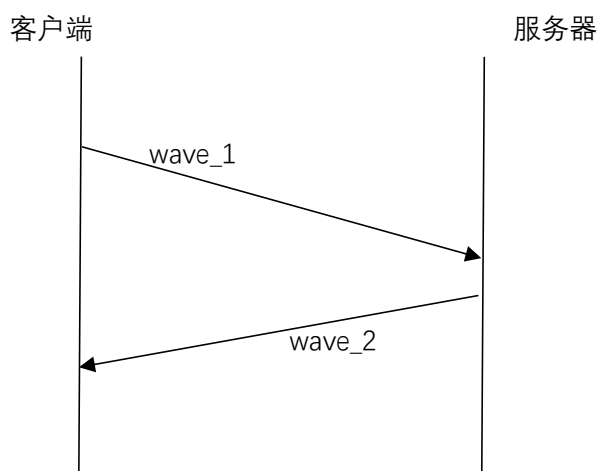
步骤二：服务器端接收后需计算校验和，若收到 shake_1 且校验和等于 0，则开始发送第二次握手信号 shake_2。

步骤三：客户端收到服务器端发来的 shake_2，并计算出校验和正确，开始发第三次握手 shake_3。

步骤四：服务器端接收到第三次握手信号，且计算出校验和正确，那么握手过程成功。

➤ 挥手过程：

两次挥手过程的大致流程图如下：



步骤一：客户端发送第一次挥手信号 wave_1。

步骤二：服务器端接收后需计算校验和，若收到 wave_1 且校验和等于 0，则开始发送第二次挥手信号 wave_2。

步骤三：客户端收到服务器端发来的 wave_2，并计算出校验和正确，则挥手过程成功。

➤ 校验和的计算：

使用上课 PPT 中讲到过的 checksum 函数进行差错检测：

【发送端】：

- 产生伪首部，校验和域段清 0，将数据报用 0 补齐为 16 位整数倍
- 将伪首部和数据报一起看成 16 位整数序列
- 进行 16 位二进制反码求和运算，计算结果取反写入校验和域段

【接收端】:

- 产生伪首部, 将数据报用 0 补齐为 16 为整数倍
- 按 16 位整数序列, 采用 16 位二进制反码求和运算
- 如果计算结果位全 1, 没有检测到错误; 否则, 说明数据报存在差错

在发送包 (包括 ACK 和 NAK) 还有握手挥手的时候都需要进行校验和的计算, 来确保发包是否出错。

➤ 确认重传:

具体的机制意思是在发送一个数据包之后, 就开启一个定时器, 若是在这个时间内没有收到发送数据的 ACK 确认报文, 则对该报文进行重传, 在达到一定次数还没有成功时放弃并发送一个复位信号。主要包括累计确认、超时时间计算、快速重传等几个方面。在 3-1 中我采用的是超时重传。

【超时重传】:

对于当前数据包开启一个计时器, 如果超时则进行重传, 并且选择适当的序列号。实现出来就体现在当数据包丢包时就会进行超时重传; 并且在 ACK/NAK 进行丢包的时候也能进行数据包的超时重传。

在文件传输过程中的超时的设置:

因为在服务端 `recvfrom` 是进行阻塞接受的, 我们先使用库函数将其阻塞时间进行设置, 在我们的 `TIMEOUT` 范围之内。

【出错重传】:

出错重传分为两种:

- 自身的数据包重传, 当对方发来的 NAK 数据包被接收到的时候, 我们需要进行重传;
- 接收到的 ACK 或者 NAK 数据包校验有问题, 无法恢复, 则进行重传之前的数据包。

(为了和之前保持一致, 重传数据包的标志位也和之前的标志位相同。)

(如果在挥手和握手期间传递的数据包出错, 则从头进行整个挥手握手过程, 保证连接信息完整性。)

(握手挥手过程中的丢包因为无法确认和检测, 只能进行重新进行挥手实现,

并且在挥手中进行超时次数的设置，超过次数认为对方完全断网，自己则进行资源回收后断网。)

➤ **ACK 和 NAK 数据包: (3-2 不涉及 NAK, 原因见累计确认)**

当服务器接收到数据包之后，将对于数据进行校验和计算，对于"校验和"和"数据报文"进行统一的校验操作，其中的 ACK 和 NAK 数据包长度均为 3 字节（包含差错检测位，标志位和序列号，序列号位为其对应的接收包的序号。

如果校验和计算之后的结果是 0，那么证明没有错误产生，则缓存接收到的数据报文。之后向客户端发送"ACK"数据包，ACK 数据包的标志位的后两位为 1，其他位都为 0(即 0x03);

如果校验和计算之后的结果不是 0，那么证明出错了，那么发送"NAK"数据包，NAK 数据包的标志位后 3 位为 1，其他位都为 0(即 0x07)，表示接受的数据包出错。

➤ **传输的具体内容:**

首先我们会将文件名作为一个单独的数据段发送过去，之后发送整个文件的数据段，在接收端接收到两端数据之后，就可以根据文件名进行数据解析工作，或者在本地存储数据。

lab3-2 的补充协议部分

➤ **滑动窗口:**

该协议允许发送方在停止并等待确认前可以连续发送多个分组。由于发送方不必每发一个分组就停下来等待确认，因此该协议可以加速数据的传输。但它也受限在流水线中未确认的分组不能超过最大窗口数。不同于 3-1 实现的停等协议的是，之前是一个一个连续地发送，相当于滑动窗口的窗口大小等于 1。

在发送端，窗口内的分组序号对应的数据分组是可以连续发送的。窗口内的数据分组有：

- 已发送但尚未得到确认
- 未发送但可以连续发送
- 已发送且已得到确认，但窗口中本序号的前面还有未得到确认的数据分组

滑动窗口法要求各数据分组按顺序发送，但并不要求确认按序返回。一旦窗口前面部分的数据分组得到确认，则窗口向前滑动相应的位置，落入窗口中的后续分组又可以连续发送。

在本次程序中，对于发送方我们设定一个大小为 WINDOW_SIZE 的序号窗口，其中的数据为已经发送确还未确认的。如果窗口未满，则我们可以继续发送数据包；如果窗口满了，则需要等待 ACK 确认后窗口的减小，之后才能发送数据包。

对于一个数据段的结束，因为窗口此时不能继续扩大，因此我们需要做特殊的标记处理，此时只允许缩小窗口等待全部数据包确认，则可以退出发送函数。

➤ GO——BACKN

整体的 GBN 协议设计，采用 while 循环来回调换 recv 和 send 进行实现，避免了使用线程导致的繁琐。具体含义是对于每一个发送的数据包，记录了他发送出去的时间，存储在一个队列里 timer_list 中，之后根据当前 clock()-最初发送的时间来判断队首的包是否发送超时，若超时则需要进行回退重发，并且清空队列；如果未超时收到包，则需要弹出队列队首，即更新计时器。

➤ 累计确认：

由于使用累计确认策略，我们将不会在接收方进行 NAK 包的设定。只有当当前序号之前的所有包都接收到了，发送这个序号对应的 ACK 包，否则即使接受到了之后的包，也只会发送之前连续接受到的正确的包的 ACK。

➤ 整体实现思路：

对于上层应用程序发送来的数据段，首先进行各个发送包的封装，开始流水线的式的进行数据的发送，在发送窗口未填满时，边发送边进行接收检测，查看对方的是否有累计确认状态码发送过来。对于一个累计确认状态的序号高于当前的 base 序号的包，则进行发送窗口的滑动，并且更新定时器，继续数据包发送。在对方 ACK 包丢失的情况下，我们需要处理队列 timer_list 的出队个数，我们通过创建一个 bool 型的数组来实现在队列中的序号这样可以快速地进行查找。

其中窗口大小和 ip 地址和文件名都可以进行输入。

➤ 发送窗口是否已满的判断:

因为我们设定的序列号只有一个 char 型大小(8 比特), 因此存在序号重复的可能, 使得 ACK 序号和发送方窗口是否已满的判断很难通过 base 和 nextpacknum 判断, 因此我们可以利用前面提到的 timer_list 的长度和变换进行判断, 使得此问题可以得到处理。

➤ 传输时间和平均吞吐率:

从调用开始传包的函数开始计时, 到调用结束也结束计时, 定义 clock_t start, end,time;按照 $time = (end - start) / \text{CLOCKS_PER_SEC}$ 计算出 time 得到具体的时间。

平均吞吐率根据公式: $x = \frac{C}{t}$

其中其中 C 为完成的任务总量, 而 t 为完成这些任务所需要的总时间。

所以在本次程序中, 平均吞吐率=length *8/ 1000 / time 单位是: "kbps" (其中 length 为文件的长度)

实验代码

【客户端】:

- 先定义一些常量，比如数据段的长度和 ACK 等一些标志的字符。

```

1  #pragma comment(lib, "Ws2_32.lib")
2
3  #include <iostream>
4  #include <winsock2.h>
5  #include <stdio.h>
6  #include <fstream>
7  #include <vector>
8  #include <string>
9  #include <time.h>
10
11  // 对于不是整个传输数据结尾的数据包来说最多数据长度为253个字节，
12  //即数据报文为256个字节。
13
14  using namespace std;
15  const int Mlenx = 253;
16  //如果最后一个数据包长度为253字节时，整个报文的长度可能达到257字节，
17  //因此再接受的时候需要统一按照257字节接收
18  const unsigned char ACK = 0x03; //都是不可变的无符号字符
19  const unsigned char NAK = 0x07;
20  const unsigned char LAST_PACK = 0x18;
21  const unsigned char NOTLAST_PACK = 0x08;
22  const unsigned char SHAKE_1 = 0x01;
23  const unsigned char SHAKE_2 = 0x02;
24  const unsigned char SHAKE_3 = 0x04;
25  const unsigned char WAVE_1 = 0x80;
26  const unsigned char WAVE_2 = 0x40;
27  const int TIMEOUT = 1000;
28  char buffer[200000000];
29  int len;
30
31  SOCKET client;
32  SOCKADDR_IN serverAddr, clientAddr;

```

- 计算校验和：返回为 0 则校验正确。

```

49  unsigned char cksum(char* flag, int len) { //计算校验和
50  if (len == 0) {
51      return ~0;
52  }
53  unsigned int ret = 0;
54  int i = 0;
55  while (len--) {
56      ret += (unsigned char)flag[i++];
57      if (ret & 0xFF00) {
58          ret &= 0x00FF;
59          ret++;
60      }
61  }
62  //cout << ~(ret & 0x00FF);
63  return ~(ret & 0x00FF);
64  }
65

```

➤ 握手函数:

```

66 void shake_hand() { //开始握手
67     while (1) {
68         //发送shake_1
69         char tmp[2]; //发送数据的缓冲区, 第一位仍然是存入校验和, 进行校验和的计算
70         tmp[1] = SHAKE_1; //发送第一次握手
71         tmp[0] = cksum(tmp + 1, 1);
72         sendto(client, tmp, 2, 0, (sockaddr*)&serverAddr, sizeof(serverAddr));
73         int begintime = clock(); //记录时间
74         char recv[2];
75         int lentmp = sizeof(clientAddr);
76         int fail_send = 0;
77         while (recvfrom(client, recv, 2, 0, (sockaddr*)&serverAddr, &lentmp) == SOCKET_ERROR)
78             if (clock() - begintime > TIMEOUT) {
79                 fail_send = 1;
80                 break;
81             }
82         //接受shake_2并校验
83         if (fail_send == 0 && cksum(recv, 2) == 0 && recv[1] == SHAKE_2) {
84             //发送shake_3
85             tmp[1] = SHAKE_3;
86             tmp[0] = cksum(tmp + 1, 1);
87             sendto(client, tmp, 2, 0, (sockaddr*)&serverAddr, sizeof(serverAddr));
88             break;
89         }
90     }
91 }
92
93

```

➤ 挥手函数:

```

95 void wave_hand() {
96     int tot_fail = 0; //记录总的失败传递数
97     while (1) {
98         //发送wave_1
99         char tmp[2];
100         tmp[1] = WAVE_1;
101         tmp[0] = cksum(tmp + 1, 1); //发送第一次挥手
102         sendto(client, tmp, 2, 0, (sockaddr*)&serverAddr, sizeof(serverAddr));
103         int begintime = clock();
104         char recv[2];
105         int lentmp = sizeof(serverAddr);
106         int fail_send = 0;
107         while (recvfrom(client, recv, 2, 0, (sockaddr*)&serverAddr, &lentmp) == SOCKET_ERROR)
108             if (clock() - begintime > TIMEOUT) {
109                 fail_send = 1;
110                 tot_fail++;
111                 break;
112             }
113         //接受wave_2并校验
114         if (fail_send == 0 && cksum(recv, 2) == 0 && recv[1] == WAVE_2)
115             break;
116         else {
117             if (tot_fail == 3) { //三次失败
118                 printf("挥手失败...");
119                 break;
120             }
121             continue;
122         }
123     }
124 }
125

```

- 具体发包函数：进行滑动窗口的实现，用一个 `in_list` 数组来实现对应序号的数据包是否在窗口中，用一个队列 `timer_list` 存入窗口中的数据包发出去的时间点和序列号。其中具体的每次发送数据还是需要去调用 `send_package()` 函数，最后打印出发送过程的进度条。

```

151     bool in_list[UCHAR_MAX + 1]; // 序列号编号256
152     void send_message(char* message, int lent) { // lab3-2
153         queue<pair<int, int>> timer_list; // 存入timer记录发送出去的时间点, order
154         int leave_cnt = 0;
155         static int base = 0;
156         int has_send = 0; // 已发送未确认
157         int nextseqnum = base;
158         int has_send_succ = 0; // 已确认
159         int tot_package = lent / Mlenx + (lent % Mlenx != 0); // 确定发包数, 是否加最后一个包
160         while (1) {
161             if (has_send_succ == tot_package) // 先判断是否已全部发送成功
162                 break;
163             // 在发送窗口未填满时, 边发送边进行接收检测, 查看对方的是否有累计确认状态码发送过来。
164             if (timer_list.size() < WINDOW_SIZE && has_send != tot_package) {
165                 send_package(message + has_send * Mlenx,
166                             has_send == tot_package - 1 ? lent - (tot_package - 1) * Mlenx : Mlenx, // 是最
167                             nextseqnum % ((int)UCHAR_MAX + 1), // 序列号
168                             has_send == tot_package - 1); // 是否是最后一个包
169                 timer_list.push(make_pair(clock(), nextseqnum % ((int)UCHAR_MAX + 1))); // 开始计时
170                 in_list[nextseqnum % ((int)UCHAR_MAX + 1)] = 1; // 并记录序列号, 进入窗口的记为1
171                 nextseqnum++;
172                 has_send++;
173             }
174             // 使用了while循环来回调换recv和send进行实现, 避免了使用线程导致的繁琐。
175             // 一个累计确认状态的序号高于当前的base序号的包, 则进行发送窗口的滑动,
176             // 并且更新定时器, 继续数据包发送。
177             char recv[3];
178             int lentmp = sizeof(serverAddr);
179             // 窗口满了就等待ack确认后窗口减小, 才能继续发送

```

之后等待 ACK 的确认，采用的是累计确认的方法。同时也需要检查是否超时，超时则需要进行回退重发，并且更新一下队列；若未超时，则弹出队列中的第一个元素表示将计时器进行一下更新。

```

180     if (recvfrom(client, recv, 3, 0, (sockaddr*)&serverAddr, &lenmp) != SOCKET_ERROR &&
181         recv[1] == ACK && in_list[(unsigned char)recv[2]]) { //recv2是序列号, 接收ACK
182         //累计确认
183         while (timer_list.front().second != (unsigned char)recv[2]) { //队列中第一个元素的
184             has_send_succ++; //已确认
185             base++; //窗口移动
186             in_list[timer_list.front().second] = 0; //存入第一个元素的序列号, 不在窗口, 记
187             timer_list.pop(); //删除第一个元素
188         }
189         //未超时收到包, 则需要弹出队列队首, 即更新计时器。
190         in_list[timer_list.front().second] = 0; //对应的序列号, 不在窗口内, 记为0
191         has_send_succ++;
192         base++;
193         leave_cnt = 0;
194         timer_list.pop(); //删除第一个元素
195     }
196     else {
197         //超时后进行回退重发, 并且清空队列
198         if (clock() - timer_list.front().first > TIMEOUT) { //当前clock()判断队首的包是否?
199             nextseqnum = base; //回退到base
200             leave_cnt++;
201             has_send -= timer_list.size(); //减去现在队列中的元素个数, 缩小窗口等待的数据
202             //未超时收到包, 则需要弹出队列队首, 即更新计时器。
203             while (!timer_list.empty()) //不为空
204                 timer_list.pop(); //删除第一个元素
205         }
206     }
207     //cout << leave_cnt;
208     //...
209     if (base % 100 == 0)
210         printf("此文件已经发送%.2f%%\n", (float)base / tot_package * 100);
211 }
212
213
214
215
216

```

send_package () 单次发包的函数:


```
84 bool send_package(char* message, int lent, int order, int last = 0) { //分片发包
85     if (lent > Mlenx) { //单片长度过大
86         return false;
87     }
88     if (last == false && lent != Mlenx) { //不是最后一个包也不是单片长度
89         return false;
90     }
91     char* tmp;
92     int tmp_len;
93
94     if (last) { //是最后一个包
95         tmp = new char[lent + 4]; //分配缓冲区
96         tmp[1] = LAST_PACK;
97         tmp[2] = order; //序列号
98         tmp[3] = lent; //多一个长度的存储
99         for (int i = 4; i < lent + 4; i++)
100             tmp[i] = message[i - 4]; //存入内容
101         tmp[0] = cksum(tmp + 1, lent + 3); //第一位存入校验和
102         //cout << "lastbao校验和为: " << int(tmp[0]);
103         tmp_len = lent + 4; //还是记录一下长度
104     }
105     else {
106         tmp = new char[lent + 3];
107         tmp[1] = NOTLAST_PACK; //不是最后一个包
108         tmp[2] = order;
109         for (int i = 3; i < lent + 3; i++)
110             tmp[i] = message[i - 3];
111         tmp[0] = cksum(tmp + 1, lent + 2);
112         //cout << "bushilastbao校验和为: " << int(tmp[0]);
113         tmp_len = lent + 3;
114     }
115     sendto(client, tmp, tmp_len, 0, (sockaddr*)&serverAddr, sizeof(serverAddr)); //将单片的信息
116     return true;
117 }
```

- 主函数部分：包含 socket 套接字的创建和连接建立和发包的调用主要过程。

```
204 int main() {
205     //string filename[4] = { "E:\\1.jpg", "E:\\helloworld.txt", "E:\\2.jpg", "E:\\3.jpg" };
206     WSADATA wsadata;
207     int error = WSStartup(MAKEWORD(2, 2), &wsadata);
208     if (error) {
209         printf("init error");
210         return 0;
211     }
212     string serverip;
213     while (1) {
214         printf("请输入接收方ip地址:\n");
215         getline(cin, serverip);
216
217         if (inet_addr(serverip.c_str()) == INADDR_NONE) { //c_str()可以得到ip的一个char*参数
218             printf("ip地址不合法! \n");
219             continue;
220         }
221         break;
222     }
223
224     int port = 11451;
225
226     serverAddr.sin_family = AF_INET; //使用ipv4
227     serverAddr.sin_port = htons(port);
228     serverAddr.sin_addr.s_addr = inet_addr(serverip.c_str()); //得到输入的ip
229     client = socket(AF_INET, SOCK_DGRAM, 0);
230
231     int time_out = 1; //1ms超时
232     setsockopt(client, SOL_SOCKET, SO_RCVTIMEO, (char*)&time_out, sizeof(time_out));
233
234     if (client == INVALID_SOCKET) {
235         printf("creat udp socket error");
236         return 0;
237     }
```

实验 3-2 需要输出传输时间和平均吞吐率，具体计算说明见协议设计，计算过程见以下的代码中。

```
255 while (1) {
256     printf("请输入要发送的文件名: ");
257     cin >> filename;
258     ifstream fin(filename.c_str(), ifstream::binary);
259     if (!fin) { ... }
260     unsigned char t = fin.get();
261     while (fin) {
262         buffer[len++] = t;
263         t = fin.get();
264     }
265     fin.close();
266     break;
267 }
268
269 printf("请输入发送窗口大小: \n");
270 cin >> WINDOW_SIZE;
271 WINDOW_SIZE %= UCHAR_MAX; //防止窗口大小大于序号域长度
272 printf("连接建立中...\n");
273 shake_hand();
274 printf("连接建立完成。 \n正在发送信息...\n");
275 clock_t start, end, time;
276 send_message((char*)(filename.c_str()), filename.length());
277 printf("文件名发送完毕, 正在发送文件内容...\n");
278 start = clock();
279 send_message(buffer, len);
280 end = clock();
281 time = (end - start) / CLOCKS_PER_SEC;
282 printf("发送的文件bytes:%d\n", len);
283 cout << "传输时间为: " << (double)time << "s" << endl;
284 cout << "平均吞吐率为: " << len*8 / 1000 / (double)time << "kbps" << endl;
285 printf("文件内容发送完毕。 \n 开始断开连接...\n");
286 wave_hand();
287 printf("连接已断开。");
288 closesocket(client);
289 WSACleanup();
290
291 return 0;
```


【服务器端】:

其中校验和的计算函数、主函数和程序头部定义的一些常量，比如数据段的长度和 ACK 等一些标志字符和客户端完全相同。因此不再阐述。

- 等待握手函数：与客户端那边的 shake_hand 函数相对应。

```

48 void wait_shake_hand() { //等待握手的函数
49     while (1) {
50         char recv[2]; //接收数据缓冲区
51         int connect = 0;
52         int lentmp = sizeof(clientAddr);
53         while (recvfrom(server, recv, 2, 0, (sockaddr*)&clientAddr, &lentmp) == SOCKET_ERROR)
54             if (cksum(recv, 2) != 0 || recv[1] != SHAKE_1)
55                 continue; //数据校验和不为0, 也没有收到第一次握手
56
57         while (1) { //发送第二次握手
58             recv[1] = SHAKE_2;
59             recv[0] = cksum(recv + 1, 1);
60             sendto(server, recv, 2, 0, (sockaddr*)&clientAddr, sizeof(clientAddr));
61             while (recvfrom(server, recv, 2, 0, (sockaddr*)&clientAddr, &lentmp) == SOCKET_ERROR)
62                 if (cksum(recv, 2) == 0 && recv[1] == SHAKE_1)
63                     continue;
64             //是否接收到第三次握手
65             if (cksum(recv, 2) != 0 || recv[1] != SHAKE_3) {
66                 printf("链接建立失败, 请重启客户端。");
67                 connect = 1; //握手失败
68             }
69             break;
70         }
71         if (connect == 1)
72             continue;
73         break;
74     }
}

```

- 等待挥手函数：与客户端那边的 wave_hand 函数相对应。

```

77 void wait_wave_hand() { //等待挥手
78     while (1) {
79         char recv[2]; //接收数据缓冲区
80         int lentmp = sizeof(clientAddr);
81         while (recvfrom(server, recv, 2, 0, (sockaddr*)&clientAddr, &lentmp) == SOCKET_ERROR)
82             if (cksum(recv, 2) != 0 || recv[1] != (char)WAVE_1) //校验和不为0且未收到第一次挥手
83                 continue;
84         recv[1] = WAVE_2;
85         recv[0] = cksum(recv + 1, 1);
86         sendto(server, recv, 2, 0, (sockaddr*)&clientAddr, sizeof(clientAddr)); //发送第二次挥手
87         break;
88     }
89 }

```

- 具体接收数据的函数：由于使用累计确认策略，我们将不会在接收方进行 NAK 包的设定。只有当当前序号之前的所有包都接收到了，发送这个序号对应的 ack 包，否则即使接受到了之后的包，也只会发送之前连续接受到的正确的包

的 ACK。

```

88 void recv_message(char* message, int& len_recv) {
89     char recv[Mlenx + 4];
90     int lentmp = sizeof(clientAddr);
91     static unsigned char last_order = 0;
92     len_recv = 0; //记录文件长度
93     while (1) {
94         while (1) {
95             memset(recv, 0, sizeof(recv));
96             while (recvfrom(server, recv, Mlenx + 4, 0, (sockaddr*)&clientAddr, &lentmp) == S
97                 char send[3];
98                 int flag = 0;
99                 if (cksum(recv, Mlenx + 4) == 0 && (unsigned char)recv[2] == last_order) {
100                     last_order++;
101                     flag = 1;
102                 }
103
104                 send[1] = ACK; //标志位
105                 send[2] = last_order - 1; //序列号
106                 send[0] = cksum(send + 1, 2); //校验和
107                 sendto(server, send, 3, 0, (sockaddr*)&clientAddr, sizeof(clientAddr)); //发送ACK
108                 if (flag)
109                     break;
110             }
111     }

```

接收时也需要考虑是否是最后一个包的情况，最后一个包的长度需要注意，因为长度的值是用 char 型来存储的，所以需要进行转换。

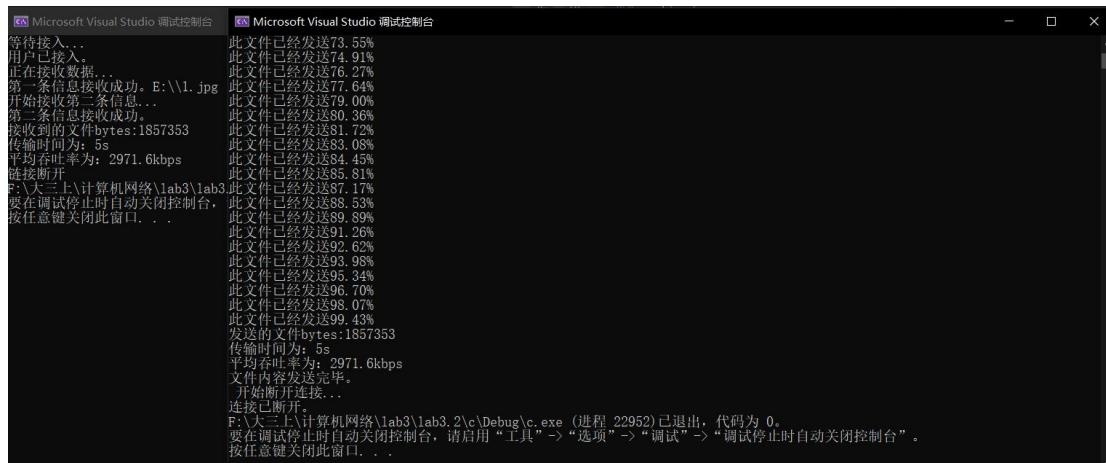
```

121 if (LAST_PACK == recv[1]) {
122     // ...
129     if (recv[3] + 4 < 4 && recv[3] + 4 > -127) { //长度超过127，长度是负值，需要加上256变为正
130         for (int i = 4; i < (int)recv[3] + 4 + 256; i++) {
131             message[len_recv++] = recv[i]; //顺序接收到数据
132         }
133     }
134     else if (recv[3] + 4 < -127) {
135         for (int i = 4; i < (int)recv[3] + 4 + 256 + 128; i++) {
136             message[len_recv++] = recv[i]; //顺序接收到数据
137         }
138     }
139     else {
140         for (int i = 4; i < (int)recv[3] + 4; i++) {
141             message[len_recv++] = recv[i]; //顺序接收到数据
142         }
143     }
144     //printf("最后! 接收的bytes为: %d\n", len_recv);
145     break;
146 }
147 else { //非结尾数据包
148     for (int i = 3; i < Mlenx + 3; i++)
149         message[len_recv++] = recv[i];
150 }
151 //printf("本次接收的bytes为: %d\n", len_recv);

```

实验结果

手动输入 ip 地址进行绑定, 输入需要进行传输的文件路径, 再输入窗口的大小, 回车之后便开始传输, 先传文件名, 再传具体的文件内容。



```
Microsoft Visual Studio 调试控制台
等待接入...
用户已接入,
正在接收数据...
第一条信息接收成功。E:\1.jpg
开始接收第二条信息...
第二条信息接收成功,
接收到的文件bytes:1857353
传输时间为: 5s
平均吞吐率为: 2971.6kbps
连接断开
F:\大三上\计算机网络\lab3\lab3
要在调试停止时自动关闭控制台,
按任意键关闭此窗口。

Microsoft Visual Studio 调试控制台
此文件已经发送73.55%
此文件已经发送74.91%
此文件已经发送76.27%
此文件已经发送77.64%
此文件已经发送79.00%
此文件已经发送80.36%
此文件已经发送81.72%
此文件已经发送83.08%
此文件已经发送84.45%
此文件已经发送85.81%
此文件已经发送87.17%
此文件已经发送88.53%
此文件已经发送89.89%
此文件已经发送91.26%
此文件已经发送92.62%
此文件已经发送93.98%
此文件已经发送95.34%
此文件已经发送96.70%
此文件已经发送98.07%
此文件已经发送99.43%
发送的文件bytes:1857353
传输时间为: 5s
平均吞吐率为: 2971.6kbps
文件内容发送完毕。
开始断开连接...
连接已断开。
F:\大三上\计算机网络\lab3\lab3.2\c\Debug\c.exe (进程 22952) 已退出, 代码为 0。
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。...
```

通过在两端的文件 bytes 数来确认是否传输成功, 另外打开接收到的文件在本地的路径, 也可以进行检查。在最后会显示出传输时间和平均吞吐率, 进行一下传输性能的测试。

