南开大学
Nankai University

南　　开　　大　　学

计 算 机 学 院

计算机网络实验报告

基于 UDP 服务设计可靠传输协议 Part1

2011763 黄天昊

年级：2020 级

专业：计算机科学与技术

指导教师：徐敬东、张建忠

2022 年 11 月 19 日

摘要

在 UDP 实现可靠传输的 Part1 的设计中，主要工作是完成了可靠 UDP 的数据包类的自主设计和编程实现，实现了三次握手建立稳定连接和四次挥手断开连接；在传输文件的过程之中，完成了 rdt3.0 的超时重传机制和停等机制，并且在源代码中添加了对于数据包的差错检验和 Drop 丢包重传的测试。为后续的设计内容搭建好了整体的框架。

关键字：UDP 可靠传输; rdt3.0; 差错检验; 三次握手; 四次挥手

目录

一、 可靠 UDP 传输的报文格式设计

通过课程讲授的知识，我们都知道 UDP 是无连接的传输层协议，提供面向事务的简单不可靠信息传输服务。为了在网络层使得基于 UDP 服务设计可靠传输协议，我们需要设计有相应的字段实现对应的可靠传输功能。

下图展示了自主设计的 data packet 头的字段分布：

图 1: 自主设计的 data packet 头结构

上面展示的 data packet 头的结构中，为了实现 rdt3.0 的可靠传输协议，我首先是设置了标志位，其中 FIN 是标志该数据包为断连请求，SYN 标志该数据包为建连请求，ACK 为应答数据包，HEAD 和 TAIL 分别标志文件传输数据包的第一个和最后一个。基于这些标志位，服务器和客户端就可以知晓目前发送或接收到的数据包的类型，做出相应的应答。

代码实现：

Packet 类成员变量

```
class Packet {
public:
        uint32_t FLAG;  // 标志位，目前只用最后五位（二进制）分别是TAIL，HEAD，ACK，SYN，FIN
        uint32_t seq;  // 序列号，uint32_t相当于unsigned int，只不过是不管是多少位的系统他都是32位
        uint32_t ack;  // 确认号
        uint32_t len;  // 数据部分长度
        uint32_t checksum;  // 校验和
        uint32_t window;  // 窗口
        char data[1024];  // 数据长度
};
```

部分 Packet 类成员函数

```
void Packet::setHEAD(int seq, int fileSize, char* fileName) {
        // 设置HEAD位为1
```

```
        this->FLAG = 0x8;   // FLAG -> 00001000

        // 这里的len并不是data的len, 而是整个文件的size
        this->len = fileSize;
        this->seq = seq;
        memcpy(this->data, fileName, strlen(fileName) + 1);
}

void Packet::setTAIL() {
        // 设置TAIL位为1
        this->FLAG = 0x10;   // FLAG -> 00010000
}

void Packet::fillData(int seq, int size, char* data) {
        // 将文件数据填入数据包data变量
        this->len = size;
        this->seq = seq;
        memcpy(this->data, data, size);
}
```

这里简单说一下 uint32_t 和 unsigned int 的区别，我们都知道，C 语言的基本类型就 char, short, int 等。但是我们在看其他源码时经常碰到 int32_t, int8_t 这种形式的定义，他们是什么呢。其实他们就是基本类型的 typedef 重定义。也就是不同平台下，使用以下名称可以保证固定长度。

- 1 字节 int8_t ——char
- 2 字节 int16_t ——short
- 4 字节 int32_t ——int
- 8 字节 int64_t ——long long

那么为何要用重定义来代替基本类型呢?

有些数据类型的确切字节数依赖于程序是如何被编译的。比如数据类型 long 一般在 32 位程序中为 4 字节, 在 64 位程序中则为 8 字节。也就是说不同编译器下 long 的大小可能不同。

为了避免由于依赖“典型”大小和不同编译器设置带来的奇怪行为, ISO C99 引人了一类数据类型, 其数据大小是固定的, 不随编译器和机器设置而变化。其中就有数据类型比如 int32_t 和 int64_t，它们分别为 4 字节和 8 字节。使用确定大小的整数类型是我们准确控制数据表示的最佳途径。

这里引申另一个小知识，类成员在内存中是要对齐的，这主要是因为如果不对齐，比如在我的 Packet 类中 FLAG 只占 1Byte(char 类型)，会出现 seq 这个变量有 3Byte 出现在前一个 word 中，后 1Byte 在后面一个 word 不利于计算机去存取（需要两次存取，还要拼接操作），如果这里使用 char 类型，虽然 char 本身占 1Byte，但是为了和后面的几个成员变量对齐（最大的那个，在这里是 4Byte）这里 FLAG 也会占 4Byte，其中的 3Byte 是填充位。所以，既然无论如何都要消耗 3Byte，那么为什么我们不给他要过来?

如果真的想用 char 去省空间，那么剩下的 3Byte 我们可以用于声明其他的成员变量，比如在 FLAG 后面加一个 short 类型的变量，这样只会浪费 1Byte，但是注意必须紧跟着 FLAG 之后声明。

二、 三次握手建立连接

既然是可靠传输，那么我们就需要一个稳定的传输通道，这里参考 TCP 的三次握手建立连接过程，设计了对应的可靠 UDP 建立连接流程，如图2。

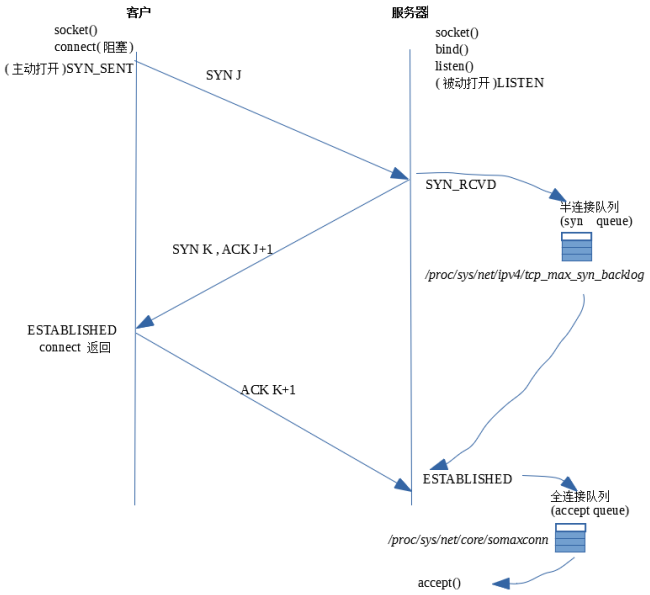

图 2: 自主设计可靠 UDP 建立连接流程图

源代码如下:

客户端 connect 函数实现三次握手

```
void connect() {
        int state = 0;  // 标识目前握手的状态
        bool flag = 1;
        Packet* sendPkt = new Packet;
        Packet* recvPkt = new Packet;

        u_long mode = 1;
        ioctlsocket(socketClient, FIONBIO, &mode); // 设置成阻塞模式等待ACK响应

        while (flag) {
                switch (state) {
                case 0:  // 发送SYN=1数据包状态
                        printTime();
                        cout << "开始第一次握手，向服务器发送SYN=1的数据包..." << endl;

                        // 第一次握手，向服务器发送SYN=1的数据包，服务器收到后会回应SYN，ACK=1的数据包
                        sendPkt->setSYN();
                        sendto(socketClient, (char*)sendPkt, BUFFER_SIZE, 0, (SOCKADDR*)&socketAddr, sizeof(SOCKADDR));
```

```
                        cout << socketAddr.sin_port << endl;
                        state = 1;  // 转状态1
                        break;

                case 1:  // 等待服务器回复
                        err = recvfrom(socketClient, (char*)recvPkt,
                            BUFFER_SIZE, 0, (SOCKADDR*)&(socketAddr), &
                            fromLen);
                        if (err >= 0) {
                                if ((recvPkt->FLAG & 0x2) && (recvPkt->FLAG &
                                    0x4)) {  // SYN=1, ACK=1
                                        printTime();
                                        cout << "开始第三次握手, 向服务器发送
                                            ACK=1的数据包..." << endl;

                                        // 第三次握手, 向服务器发送ACK=1的数
                                            据包, 告知服务器自己接收能力正常
                                        sendPkt->setACK();
                                        sendto(socketClient, (char*)sendPkt,
                                            BUFFER_SIZE, 0, (SOCKADDR*)&
                                            socketAddr, sizeof(SOCKADDR));
                                        cout << socketAddr.sin_port << endl;
                                        state = 2;  // 转状态2
                                }
                                else {
                                        printTime();
                                        cout << "第二次握手阶段收到的数据包有
                                            误, 重新开始第一次握手..." <<
                                            endl;
                                        state = 0;  // 转状态0
                                }
                        }
                        break;

                case 2: // 三次握手结束状态
                        printTime();
                        cout << "三次握手结束, 确认已建立连接, 开始文件传输
                            ..." << endl;
                        flag = 0;
                        break;
                }
        }
}
```

服务器端 connect 函数实现三次握手

```
void connect() {
        int state = 0;  // 标识目前握手的状态
        bool flag = 1;
```

```
4         Packet* sendPkt = new Packet;
5         Packet* recvPkt = new Packet;
6
7         while (flag) {
8             switch (state) {
9             case 0:  // 等待客户端发送数据包状态
10                err = recvfrom(socketServer, (char*)recvPkt,
                      BUFFER_SIZE, 0, (SOCKADDR*)&(socketAddr), &
                      fromLen);
11                if (err >= 0) {
12                    if (recvPkt->FLAG & 0x2) {  // SYN=1
13                        printTime();
14                        cout << "收到来自客户端的建连请求，开
                              始第二次握手，向客户端发送ACK,SYN
                              =1的数据包..." << endl;
15
16                        // 第二次握手，向客户端发送ACK,SYN=1
                              的数据包
17                        sendPkt->setSYNACK();
18                        sendto(socketServer, (char*)sendPkt,
                              BUFFER_SIZE, 0, (SOCKADDR*)&
                              socketAddr, sizeof(SOCKADDR));
19                        state = 1;  // 转状态1
20                    }
21                    else {
22                        printTime();
23                        cout << "第一次握手阶段收到的数据包有
                              误，重新开始第一次握手..." <<
                              endl;
24                    }
25                }
26                break;
27
28            case 1:  // 接收客户端的ACK=1数据包
29                err = recvfrom(socketServer, (char*)recvPkt,
                      BUFFER_SIZE, 0, (SOCKADDR*)&(socketAddr), &
                      fromLen);
30                if (err >= 0) {
31                    if (recvPkt->FLAG & 0x4) {  // ACK=1
32                        printTime();
33                        cout << "收到来自客户端第三次握手ACK
                              数据包..." << endl;
34
35                        state = 2;  // 转状态2
36                    }
37                    else {
38                        printTime();
39                        cout << "第三次握手阶段收到的数据包有
```

```
                                                误，重新开始第三次握手..." <<
                                                endl;
40                                          // 其实这里是有点问题的，这两边的
                                                sendto都没有单开一个状态，所以其
                                                实不会重复发送，但这里懒得写了
41                                  }
42                              }
43                              break;
44
45                      case 2: // 三次握手结束状态
46                              printTime();
47                              cout << "三次握手结束，确认已建立连接，开始文件传输
                                    ..." << endl;
48                              flag = 0;
49                              break;
50                      }
51              }
52      }
```

这里值得一提的是，我给客户端和服务器端各自设计了状态转换机，并通过 while 循环中套 switch 的机制编程实现，逻辑清晰，代码易懂。

三、 文件传输设计

在完成了三次握手之后，客户端开始进入发送文件的状态。我们需要对文件传输的过程来进行设计。

这里，我规定发送端发送的第一个数据包中携带的数据内容就是文件名内容，在 len 字段中，填入接下来要发送的整个文件的大小，同时在第一个数据包数据头之中，会将 FLAG 的 HEAD 置为 1 来标识这是一个包含文件名和文件总体信息的数据包。

发送完文件头数据包之后，发送端会进入循环监听服务器端的数据包的状态，直到接收到对应的 ACK 数据包，发送端会转为发送文件数据包的状态，将文件数据切分为最大为 1024 字节的数据包依次发送，每发送一个，就进入等待 ACK 状态，接收到 ACK 数据包之后会继续发送。

源代码如下：

sendPacket 函数

```
1  void sendPacket(Packet* sendPkt) {
2          Packet* recvPkt = new Packet;
3
4          // 检查一下文件的各个内容
5          cout << "Send Message " << sendPkt->len << " Bytes!";
6          cout << " Flag:" << sendPkt->FLAG << " SEQ:" << sendPkt->seq;
7          cout << " checksum:" << sendPkt->checksum << endl;
8          err = sendto(socketClient, (char*)sendPkt, BUFFER_SIZE, 0, (SOCKADDR
               *)&socketAddr, sizeof(SOCKADDR));
9          if (err == SOCKET_ERROR) {
10                 cout << "sendto failed with error: " << WSAGetLastError() <<
                       endl;
```

```
        }
        clock_t start = clock();  // 记录发送时间，超时重传

        // 等待接收ACK信息，验证acknowledge number
        while (true) {
                while (recvfrom(socketClient, (char*)recvPkt, BUFFER_SIZE, 0,
                    (SOCKADDR*)&(socketAddr), &fromLen) <= 0) {
                    if (clock() - start > MAX_TIME) {
                            printTime();
                            cout << "TIME OUT! Resend this data packet"
                                << endl;

                            err = sendto(socketClient, (char*)sendPkt,
                                BUFFER_SIZE, 0, (SOCKADDR*)&socketAddr,
                                sizeof(SOCKADDR));
                            if (err == SOCKET_ERROR) {
                                    cout << "sendto failed with error: "
                                        << WSAGetLastError() << endl;
                            }
                            start = clock(); // 重设开始时间
                    }
                }
                // cout << "Recieve Message " << recvPkt->len << " Bytes!";
                // cout << " Flag:" << recvPkt->FLAG << " SEQ:" << recvPkt->
                    seq << " ACK:" << recvPkt->ack;
                // cout << " checksum:" << recvPkt->checksum << endl;

                // 三个条件要同时满足，这里调试时判断用packet的seq
                if ((recvPkt->FLAG & 0x4) && (recvPkt->ack == sendPkt->seq))
                    {
                        sendSeq++;
                        break;
                }
        }

}
```

sendFile 函数

```
void sendFile() {
        sendSeq = 0;
        clock_t start = clock();

        // 先发一个记录文件名的数据包，并设置HEAD标志位为1，表示开始文件传输
        Packet* headPkt = new Packet;
        // headPkt->fillData(filePath, strlen(filePath));
        headPkt->setHEAD(sendSeq, fileSize, filePath);
        headPkt->checksum = checksum((uint32_t*)headPkt);
        sendPacket(headPkt);
```

```
11
12          // 开始发送装载文件的数据包
13          Packet* sendPkt = new Packet;
14          for (int i = 0; i < packetNum; i++) {
15                  if (i == packetNum - 1) {  // 最后一个包
16                          sendPkt->setTAIL();
17                          sendPkt->fillData(sendSeq, fileSize - i *
                                PACKET_LENGTH, fileBuffer + i * PACKET_LENGTH);
18                          sendPkt->checksum = checksum((uint32_t*)sendPkt);
19                  }
20                  else {
21                          sendPkt->fillData(sendSeq, PACKET_LENGTH, fileBuffer
                                + i * PACKET_LENGTH);
22                          sendPkt->checksum = checksum((uint32_t*)sendPkt);
23                  }
24                  sendPacket(sendPkt);
25                  Sleep(20);
26          }
27
28          clock_t end = clock();
29          printTime();
30          cout << "文件发送完毕, 传输时间为: " << (end - start) /
                CLOCKS_PER_SEC << "s" << endl;
31          cout << "吞吐率为: " << ((float)fileSize) / ((end - start) /
                CLOCKS_PER_SEC) << " Bytes/s " << endl << endl;
32  }
```

其中 sendPacket 函数只负责具体数据包的发送，sendFile 会调用 sendPacket，传入对应的 Packet 实例化对象进行发送。

在这里一定要使用 memcpy 函数进行深拷贝，避免出现浅拷贝中的指针的问题，使用该函数的过程之中一定要注意的就是按照 size 字节数的大小进行深拷贝，使用 memcpy 就不需要考虑字符数组末尾的问题了。

(一) 缺失重传机制

在 sendPacket 函数中，我实现了缺失重传机制，使用 clock() 函数记录发送完数据包之后经历的事件，一旦超时还没有收到期望的 ACK 数据包，则重新发送该数据包。

由于无法使用所提供的转发程序，这里手动设置了丢包率，如果服务器端收到的包不是目前等待的包，则直接丢弃，等待发送端重传。

具体源代码如下:

接收端丢包率设置

```
1   if (recvPkt->seq == waitSeq) {  // 收到了目前等待的包
2           double number = rand() % 10 + 1;  // 自己设置丢包率
3           if (number >= 1.5) {
4                   cout << "Recieve Message " << recvPkt->len << " Bytes!";
5                   cout << " Flag:" << recvPkt->FLAG << " SEQ:" << recvPkt->seq;
```

```
                cout << " checksum:" << recvPkt->checksum << endl;

                memcpy(fileBuffer + recvSize, recvPkt->data, recvPkt->len);
                recvSize += recvPkt->len;

                printTime();
                cout << "收到第 " << recvPkt->seq << " 号数据包，向发送端发送
                    ack=" << waitSeq << endl;

                waitSeq++;
                state = 2;
        }
}
```

(二) 差错检验机制

在协议的数据包结构设计好之后，首先需要完成是计算校验和的函数即 checksum 函数，实现的逻辑就是通过一个 16bit 大小的指针强制类型转换去遍历一个指向 class 类对象的地址，来计算校验和。(具体内容参考了课堂讲授的检验校验和伪代码)

计算校验和

```
uint16_t checksum(uint32_t* pkt){
        int size = sizeof(pkt);
        int count = (size + 1) / 2;
        uint16_t* buf = (uint16_t*)malloc(size);  // 可以+1也可以不+1
        memset(buf, 0, size);
        memcpy(buf, pkt, size);
        u_long sum = 0;
        while (count--) {
                sum += *buf++;
                if (sum & 0xFFFF0000) {
                        sum &= 0xFFFF;
                        sum++;
                }
        }
        return ~(sum & 0xFFFF);
}
```


四、 四次挥手断开连接

这里同样是参考 TCP 的四次挥手断连过程，如图3

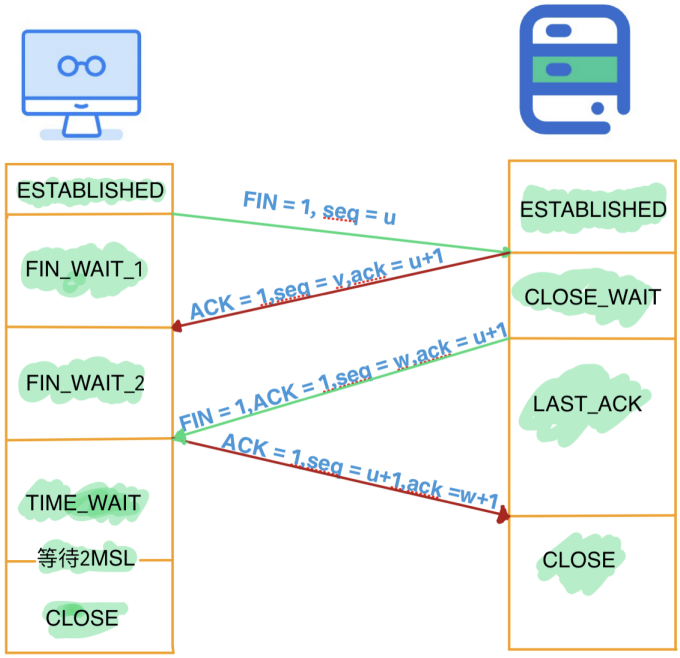

图 3: 自主设计可靠 UDP 断开连接流程图

客户端 disconnect 函数实现四次挥手

```
void disconnect() {  // 参考 <https://blog.csdn.net/LOOKTOMMER/article/details/121307137>
    int state = 0;  // 标识目前挥手的状态
    bool flag = 1;
    Packet* sendPkt = new Packet;
    Packet* recvPkt = new Packet;

    u_long mode = 1;
    ioctlsocket(socketClient, FIONBIO, &mode); // 设置成阻塞模式等待ACK响应

    while (flag) {
        switch (state) {
        case 0:  // 发送FIN=1数据包状态
            printTime();
            cout << "开始第一次挥手，向服务器发送FIN=1的数据包..." << endl;

            // 第一次挥手，向服务器发送FIN=1的数据包
            sendPkt->setFIN();
            sendto(socketClient, (char*)sendPkt, BUFFER_SIZE, 0, (SOCKADDR*)&socketAddr, sizeof(SOCKADDR));

            state = 1;  // 转状态1
            break;

```

```
23                  case 1:  // FIN_WAIT_1
24                      err = recvfrom(socketClient, (char*)recvPkt,
                            BUFFER_SIZE, 0, (SOCKADDR*)&(socketAddr), &
                            fromLen);
25                      if (err >= 0) {
26                          cout << "Recieve Message " << recvPkt->len <<
                                " Bytes!";
27                          cout << " Flag:" << recvPkt->FLAG << " SEQ:"
                                << recvPkt->seq;
28                          cout << " checksum:" << recvPkt->checksum <<
                                endl;
29                          if (recvPkt->FLAG & 0x4) {  // ACK=1
30                              printTime();
31                              cout << "收到了来自服务器第二次挥手
                                    ACK数据包..." << endl;
32
33                              state = 2;  // 转状态2
34                          }
35                          else {
36                              printTime();
37                              cout << "第二次挥手阶段收到的数据包有
                                    误，重新开始第二次挥手..." <<
                                    endl;
38                          }
39                      }
40                      break;
41
42                  case 2:  // FIN_WAIT_2
43                      err = recvfrom(socketClient, (char*)recvPkt,
                            BUFFER_SIZE, 0, (SOCKADDR*)&(socketAddr), &
                            fromLen);
44                      if (err >= 0) {
45                          cout << "Recieve Message " << recvPkt->len <<
                                " Bytes!";
46                          cout << " Flag:" << recvPkt->FLAG << " SEQ:"
                                << recvPkt->seq;
47                          cout << " checksum:" << recvPkt->checksum <<
                                endl;
48                          if ((recvPkt->FLAG & 0x1) && (recvPkt->FLAG &
                                0x4)) {  // ACK=1, FIN=1
49                              printTime();
50                              cout << "收到了来自服务器第三次挥手
                                    FIN&ACK数据包，开始第四次挥手，向
                                    服务器发送ACK=1的数据包..." <<
                                    endl;
51
52                              // 第四次挥手，向服务器发送ACK=1的数
                                    据包，通知服务器确认断开连接
```

```
53                                      sendPkt->setFINACK();
54                                      sendto(socketClient, (char*)sendPkt,
                                            BUFFER_SIZE, 0, (SOCKADDR*)&
                                            socketAddr, sizeof(SOCKADDR));
55                                      state = 3;  // 转状态3
56                                  }
57                                  else {
58                                      printTime();
59                                      cout << "第三次挥手阶段收到的数据包有
                                            误，重新开始第三次挥手..." <<
                                            endl;
60                                  }
61                              }
62                              break;
63
64                  case 3:  // TIME_WAIT
65                              // 这里按照TCP的方法，需要等待2MSL，但是先不写了，直
                                    接退出
66                              state = 4;
67                              break;
68
69                  case 4:  // 四次挥手结束状态
70                              printTime();
71                              cout << "四次挥手结束，确认已断开连接，Bye-bye..." <<
                                    endl;
72                              flag = 0;
73                              break;
74                  }
75          }
76  }
```

服务器端 dsconnect 函数实现四次挥手

```
1   void disconnect() {  // 参考 <https://blog.csdn.net/LOOKTOMMER/article/
        details/121307137>
2           int state = 0;  // 标识目前挥手的状态
3           bool flag = 1;
4           Packet* sendPkt = new Packet;
5           Packet* recvPkt = new Packet;
6
7           u_long mode = 1;
8           ioctlsocket(socketServer, FIONBIO, &mode); // 设置成阻塞模式等待ACK响
                应
9
10          while (flag) {
11                  switch (state) {
12                  case 0:  // CLOSE_WAIT_1
13                              printTime();
14                              cout << "接收到客户端的断开连接请求，开始第二次挥手，
```

```
                向客户端发送ACK=1的数据包..." << endl;

            // 第二次挥手，向客户端发送ACK=1的数据包
            sendPkt->setACK();
            sendto(socketServer, (char*)sendPkt, BUFFER_SIZE, 0,
                (SOCKADDR*)&socketAddr, sizeof(SOCKADDR));
            cout << "Send Message " << sendPkt->len << " Bytes!";
            cout << " Flag:" << sendPkt->FLAG << " SEQ:" <<
                sendPkt->seq;
            cout << " checksum:" << sendPkt->checksum << endl;

            state = 1;  // 转状态1
            break;

        case 1:  // CLOSE_WAIT_2
            printTime();
            cout << "开始第三次挥手，向客户端发送FIN,ACK=1的数据
                包..." << endl;

            // 第三次挥手，向客户端发送FIN,ACK=1的数据包
            sendPkt->setFINACK();
            sendto(socketServer, (char*)sendPkt, BUFFER_SIZE, 0,
                (SOCKADDR*)&socketAddr, sizeof(SOCKADDR));
            cout << "Send Message " << sendPkt->len << " Bytes!";
            cout << " Flag:" << sendPkt->FLAG << " SEQ:" <<
                sendPkt->seq;
            cout << " checksum:" << sendPkt->checksum << endl;

            state = 2;  // 转状态2
            break;

        case 2:  // LAST_ACK
            err = recvfrom(socketServer, (char*)recvPkt,
                BUFFER_SIZE, 0, (SOCKADDR*)&(socketAddr), &
                fromLen);
            if (err >= 0) {
                if (recvPkt->FLAG & 0x4) {  // ACK=1
                    printTime();
                    cout << "收到了来自客户端第四次挥手的
                        ACK数据包..." << endl;

                    state = 3;  // 转状态3
                }
                else {
                    printTime();
                    cout << "第四次挥手阶段收到的数据包有
                        误，重新开始第四次挥手..." <<
                        endl;
```

```
52                                  }
53                              }
54                              break;
55
56                      case 3:  // CLOSE
57                              printTime();
58                              cout << "四次挥手结束，确认已断开连接，Bye-bye..." <<
                                    endl;
59                              flag = 0;
60                              break;
61                      }
62              }
63      }
```

这里和三次握手类似，客户端和服务器端各自设计了状态转换机，在不同的状态做出相应的操作，并通过 while 循环中套 switch 的机制编程实现。

五、 可靠 UDP 传输流程展示

本节进行上述可靠 UDP 传输流程的展示。

首先是三次握手的 log 信息：

```
[2022/11/19 13:7:21:570]服务器启动成功，等待客户端建立连接
[2022/11/19 13:7:28:737]收到来自客户端的建连请求，开始第二次握手，向客户端发送ACK,SYN=1的数据包...
[2022/11/19 13:7:28:738]收到来自客户端第三次握手ACK数据包...
[2022/11/19 13:7:28:739]三次握手结束，确认已建立连接，开始文件传输...

********************************************************************************************************
```

图 4: 接收端三次握手阶段

```
[2022/11/19 13:7:28:736]客户端初始化成功，准备与服务器建立连接
[2022/11/19 13:7:28:737]开始第一次握手，向服务器发送SYN=1的数据包...
38915
[2022/11/19 13:7:28:738]开始第三次握手，向服务器发送ACK=1的数据包...
38915
[2022/11/19 13:7:28:739]三次握手结束，确认已建立连接，开始文件传输...

********************************************************************************************************
```

图 5: 发送端三次握手

接下来是文件传输的过程：

```
请输入您要传输的文件名：C:\Users\new\Desktop\GRE填空机经1100题难度分级版（第二版）.pdf
文件大小为 1156876 Bytes，总共要发送 1130 个数据包
Send Message 1156876 Bytes! Flag:8 SEQ:0 checksum:65527
Send Message 1024 Bytes! Flag:0 SEQ:1 checksum:65534
Send Message 1024 Bytes! Flag:0 SEQ:2 checksum:65533
Send Message 1024 Bytes! Flag:0 SEQ:3 checksum:65532
Send Message 1024 Bytes! Flag:0 SEQ:4 checksum:65531
[2022/11/19 13:8:43:962]TIME OUT! Resend this data packet
Send Message 1024 Bytes! Flag:0 SEQ:5 checksum:65530
Send Message 1024 Bytes! Flag:0 SEQ:6 checksum:65529
Send Message 1024 Bytes! Flag:0 SEQ:7 checksum:65528
Send Message 1024 Bytes! Flag:0 SEQ:8 checksum:65527
Send Message 1024 Bytes! Flag:0 SEQ:9 checksum:65526
Send Message 1024 Bytes! Flag:0 SEQ:10 checksum:65525
Send Message 1024 Bytes! Flag:0 SEQ:11 checksum:65524
Send Message 1024 Bytes! Flag:0 SEQ:12 checksum:65523
Send Message 1024 Bytes! Flag:0 SEQ:13 checksum:65522
```

图 6: 发送端文件传输 log

```
Recieve Message 1024 Bytes! Flag:0 SEQ:117 checksum:65418
[2022/11/19 13:9:21:625]收到第 117 号数据包，向发送端发送 ack=117
Recieve Message 1024 Bytes! Flag:0 SEQ:118 checksum:65417
[2022/11/19 13:9:21:656]收到第 118 号数据包，向发送端发送 ack=118
Recieve Message 1024 Bytes! Flag:0 SEQ:119 checksum:65416
[2022/11/19 13:9:21:687]收到第 119 号数据包，向发送端发送 ack=119
Recieve Message 1024 Bytes! Flag:0 SEQ:120 checksum:65415
[2022/11/19 13:9:23:196]收到第 120 号数据包，向发送端发送 ack=120
Recieve Message 1024 Bytes! Flag:0 SEQ:121 checksum:65414
[2022/11/19 13:9:27:229]收到第 121 号数据包，向发送端发送 ack=121
Recieve Message 1024 Bytes! Flag:0 SEQ:122 checksum:65413
[2022/11/19 13:9:27:268]收到第 122 号数据包，向发送端发送 ack=122
Recieve Message 1024 Bytes! Flag:0 SEQ:123 checksum:65412
[2022/11/19 13:9:27:298]收到第 123 号数据包，向发送端发送 ack=123
```

图 7: 接收端文件传输 log

可以看到，如果出现丢包，发送端会打印超时信息，并进行重传。
文件传输完成后，客户端发送断连请求，开始四次挥手并退出程序：

```
[2022/11/19 13:13:31:356]收到第 1128 号数据包，向发送端发送 ack=1128
Recieve Message 1024 Bytes! Flag:0 SEQ:1129 checksum:64406
[2022/11/19 13:13:31:387]收到第 1129 号数据包，向发送端发送 ack=1129
Recieve Message 780 Bytes! Flag:16 SEQ:1130 checksum:64389
[2022/11/19 13:13:31:417]收到第 1130 号数据包，向发送端发送 ack=1130
[2022/11/19 13:13:31:418]文件接收完毕...

********************************************************************************************************************

[2022/11/19 13:13:31:459]接收到客户端的断开连接请求，开始第二次挥手，向客户端发送ACK=1的数据包...
Send Message 0 Bytes! Flag:4 SEQ:0 checksum:0
[2022/11/19 13:13:31:465]开始第三次挥手，向客户端发送FIN,ACK=1的数据包...
Send Message 0 Bytes! Flag:5 SEQ:0 checksum:0
[2022/11/19 13:13:31:485]收到了来自客户端第四次挥手的ACK数据包...
[2022/11/19 13:13:31:485]四次挥手结束，确认已断开连接，Bye-bye...
[2022/11/19 13:13:31:487]程序退出...

F:\All assignments during college\Junior(first semester)\Computer Network\Computer-Network\lab3\3.
ver.exe (进程 49872)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. . .
```

图 8: 接收端四次挥手 log

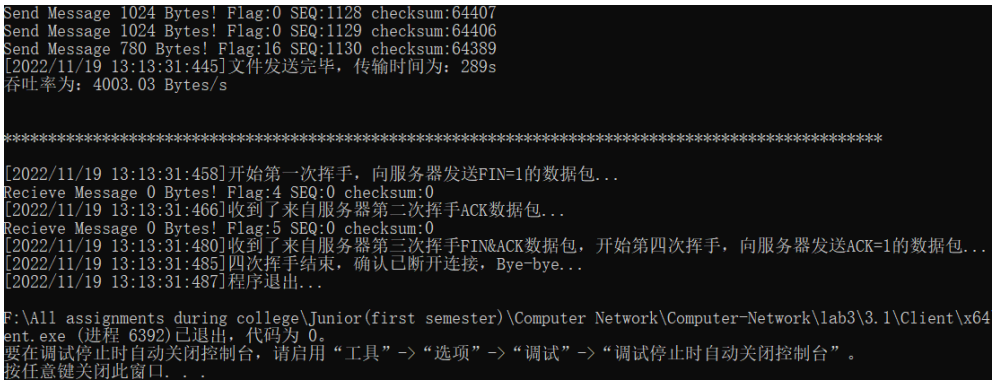

图 9: 发送端四次挥手 log

至此，整个过程结束。

六、 总结

在本次实验之中，深入地了解了可靠传输 rdt 从 2.0 到 3.0 的具体的状态机转换内容，熟悉了 C++ 之中关于文件的操作，对于使用 int32_t 等类型的变量有了深入的理解，进一步体会了 socket 编程。

个人 GitHub 仓库链接：https://github.com/Skyyyy0920/Computer-Network

参考文献

1、https://blog.csdn.net/LOOKTOMMER/article/details/121307137
2、03-计算机网络-第三章-2022.pdf
3、上机作业 3 讲解-2022.pdf