

实验三：基于 UDP 服务设计可靠传输协议并编程实现

- 姓名：马永田
- 年级：2020 级
- 专业：计算机科学与技术
- 指导教师：张建忠 & 徐敬东

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、确认重传**等。流量控制采用**停等机制**，**完成给定测试文件的传输**。

- 数据报套接字：UDP；
- 建立连接：实现类似TCP的握手、挥手功能；
- 差错检测：计算校验和；
- 确认重传：rdt2.0、rdt2.1、rdt2.2、rdt3.0等，亦可自行设计协议；
- 单向传输：发送端、接收端；
- 有必要日志输出。

协议设计

报文格式

协议中设计的报文格式如下代码所示：

```
1  #pragma pack(1) //按字节对齐
2  struct Packet    //报文格式
3  {
4      //SYN=0x01  Ack=0x02  FIN=0x04  PACKET=0x08  StartFile=0x10  EndFile=0x20
      File=0x40
5      DWORD SendIP;        //发送端IP
6      DWORD RecvIP;        //接收端IP
7      u_short Port;        //端口
8      u_short Protocol;    //协议类型
9      int Flags = 0;        //标志位
10     int Seq;              //消息发送序号
11     int Ack;              //确认序号
12     int index;            //文件分片传输 第几片
13     int length;           //Data段长度
14     u_short checksum;     //校验和
15     char Data[BUF_SIZE]; //报文数据段
16 };
17 #pragma pack() //恢复4Byte编址格式
```

停等机制实现

停等协议是发送双方传输数据的一种协议方式，在停等协议下只有正确收到对方的ACK确认后才会继续发送新的数据包。实验中设置套接字为非阻塞模式，在客户端中(与服务端的停等接收差别不大)停等机制使用如下函数实现：

```
1 //收发包
2 void sendPacket(Packet* sendP) {
3     setChecksum(sendP); //设置校验和
4     if (sendto(sockClient, (char*)sendP, sizeof(Packet), 0, (struct
5 sockaddr*)&sockAddr, sizeof(sockaddr)) != SOCKET_ERROR) {
6     }
7 }
8 bool recvPacket(Packet* recvP) {
9     memset(recvP, 0, sizeof(sizeof(&recvP)));
10    int re = recvfrom(sockClient, (char*)recvP, sizeof(Packet), 0, (struct
11 sockaddr*)&sockAddr, &addr_len);
12    if (re != -1 && checkChecksum(recvP)) { //检查校验和
13        return true;
14    }
15    return false;
16 }
```

```
1 //停等发送函数
2 bool stopWaitSend(Packet* sendP, Packet* recvP) {
3     sendPacket(sendP);
4     if (checkFIN(sendP) && state != FIN_WAIT_1) {
5         state = FIN_WAIT_1;
6     }
7     clockStart = clock(); //开始计时
8     int retransNum = 0; //重发次数
9     while (1){
10         if (recvPacket(recvP)) {
11             if (checkACK(recvP) && recvP->Ack == (sendP->Seq + 1)){
12                 return 1; //收到对sendP的确认报文
13             }
14         }
15         clockEnd = clock();
16         if (retransNum == RETRANSMISSION_TIMES) { //到达最大重发次数
17             return 0;
18         }
19         if ((clockEnd - clockStart) >= WAIT_TIME) { //超时重发
20             retransNum++;
21             sendPacket(sendP);
22             clockStart = clock();
23         }
24     }
25     return 0; //发送失败
26 }
```

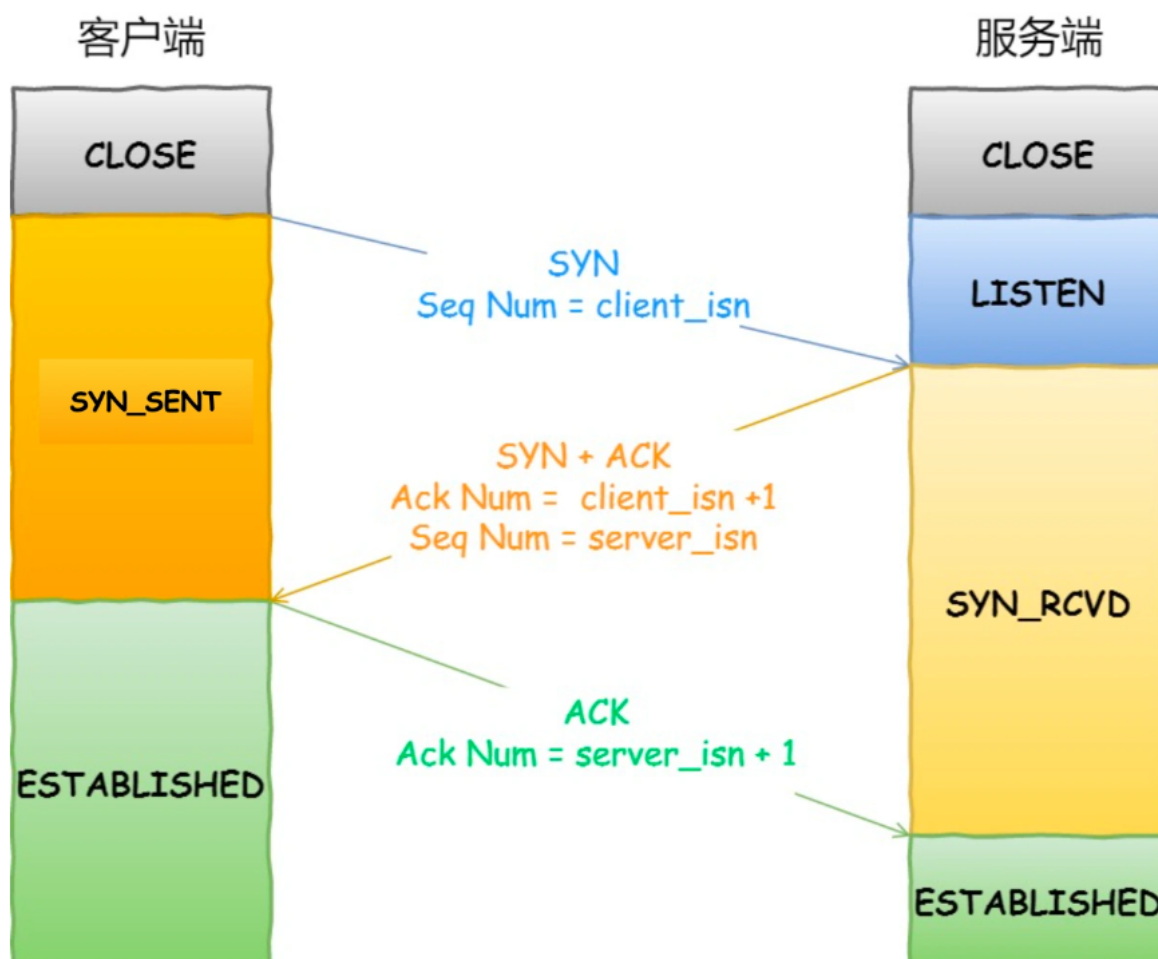
其工作流程如下：

1. 发送方发送前会为该数据帧设置消息序号Seq，并将当前数据帧暂时保留；
2. 发送方发送数据帧后，启动计时器；

3. 当接收方检测到一个含有差错的数据帧时，舍弃该帧；
4. 当接收方收到无差错的数据帧后，向发送方返回一个确认序号 $Ack=Seq+1$ 的ACK确认帧；
5. 若发送方在规定时间内收到想要的ACK确认帧，开始下一帧的发送；
6. 若发送方在规定时间内未收到ACK确认帧，则重新发送暂留的数据帧；
7. 当重发次数超过规定次数后，发送失败放弃发送。

建立连接

建立连接机制参考了TCP中的三次握手过程，其流程如图所示：



服务器端：在初始化完成后服务端会处于**LISTEN**状态，不断监听数据帧的到来，当接收到数据帧后检查其类型，若收到**SYN**同步请求，则执行**buildConnection**函数，进入**SYN_RCVD**状态。在该函数中，服务端会向客户端回复一个**SYN请求同步与ACK确认帧**，其Ack值为收到的SYN包的Seq+1，而Seq为服务端发送序号sendSeq。当服务端收到该帧的确认ACK数据帧(Ack=sendSeq+1)时说明服务端与客户端的收发能力均无问题，**连接成功建立**，服务端进入**ESTABLISHED**状态。

```
1 //服务端
2 bool buildConnection(Packet synPacket) {
3     state = SYN_RCVD;
4     Packet sendACK, recvACK;
5     setSYN(&sendACK); //发送请求同步的SYN信息帧
6     setACK(&sendACK, &synPacket); //回复对于消息synPacket的ACK
7     sendACK.Seq = sendSeq++; //设置消息sendACK的发送序号。
8
9     if(stopwaitSend(&sendSYN,&recvACK)){//为避免SYN攻击 使用停等发送 规定时间无响应则放弃
```

```

10     state = ESTABLISHED;
11     return 1;
12 }
13 state = CLOSE;
14 return 0;
15 }

```

客户端：在初始化完成后用户可以通过输入控制信号进行选择是否建立连接，若请求建立连接，输入服务器的IP地址，之后客户端就会执行**buildConnection**函数，使用**停等发送**功能向服务端发送一个**SYN**请求同步建立连接，进入**SYN_SENT**状态检查收到的ACK是否也为**SYN同步**请求，若是则回复一个ACK，其Ack值为收到的SYN的Seq值加一，之后**连接成功建立**，进入ESTABLISHED状态。

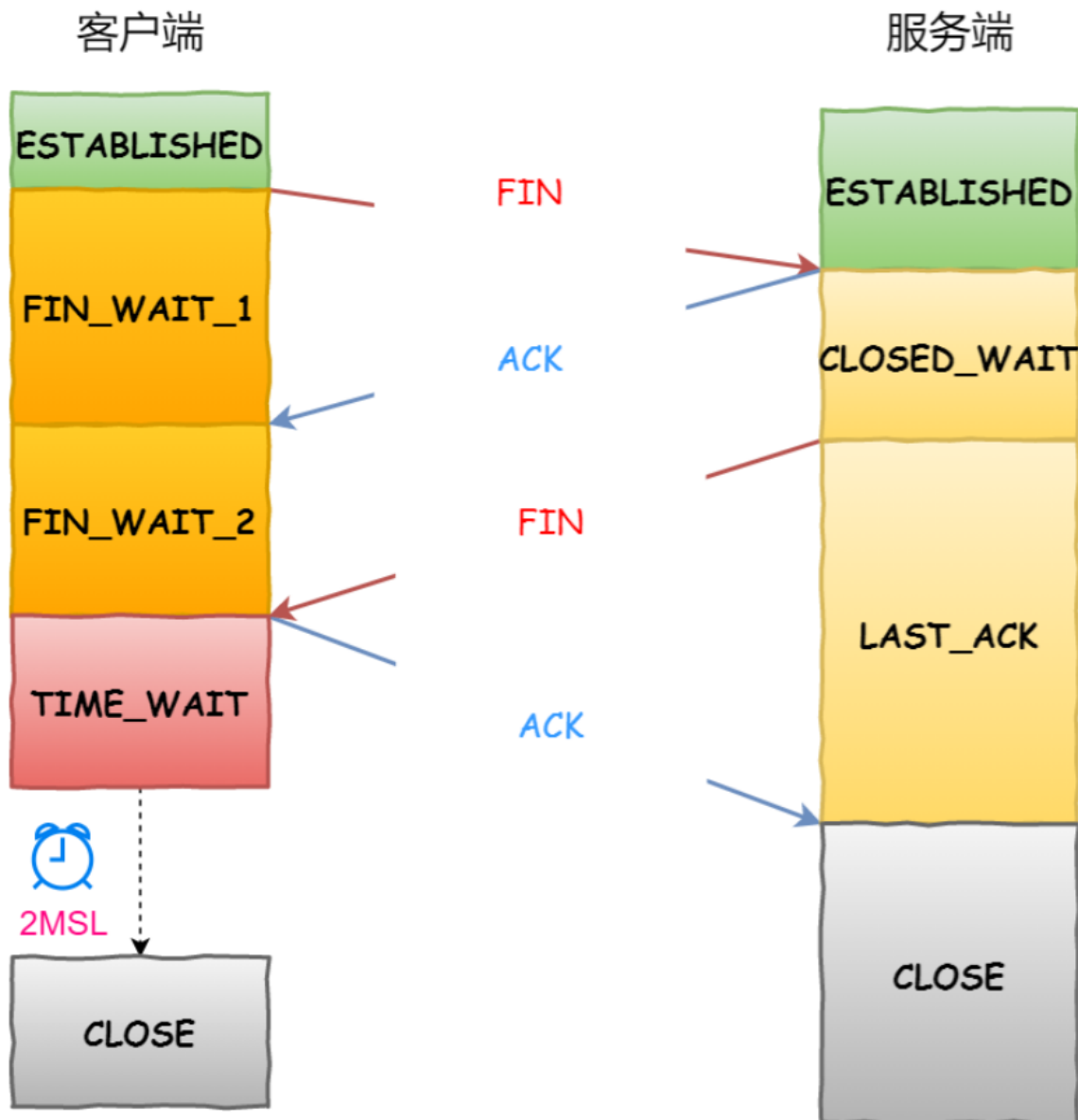
```

1  bool buildConnection() {
2      Packet sendSYN, recvACK, sendACK;
3      sendSYN.Seq = sendSeq++;
4      setSYN(&sendSYN);
5      state = SYN_SENT;
6
7      if(stopwaitSend(&sendSYN,&recvACK)){
8          if (checkSYN(&recvACK)) {
9              sendACK.Seq = sendSeq++;
10             setACK(&sendACK, &recvACK);
11             sendPacket(&sendACK);
12             state = ESTABLISHED;
13             return 1;
14         }
15     }
16     state = CLOSE;
17     return 0;
18 }

```

断开连接

断开连接机制也是参考了TCP协议中的四次挥手：其流程如下图所示：



服务端：服务端会不断监听数据帧的到来，当接收到数据帧后检查其类型，若收到FIN，则执行 `breakConnection` 函数，进入 `CLOSE_WAIT` 状态。在 `breakConnection` 函数中，服务端会向客户端回复一个 **ACK** 确认收到的FIN，之后服务端会使用**停等发送功能**向客户端发送一个 **FIN** 数据帧，在发送FIN后进入 `LAST_ACK` 状态，等待最后一个 **ACK** 确认，收到相应的ACK确认后**连接断开**，进入 `CLOSE` 状态。

```
1  bool breakConnection(Packet finPacket) {
2      state = CLOSE_WAIT;
3      Packet sendACK, sendFIN, recvACK;
4      sendACK.Seq = sendSeq++;
5      setACK(&sendACK, &finPacket);
6      sendPacket(&sendACK);
7
8      setFIN(&sendFIN); //FIN=1
9      sendFIN.Seq = sendSeq++;
10     if (stopwaitSend(&sendFIN, &recvACK)) {
11         state = CLOSE;
```

```

12         nowTime = getTime();
13         cout << nowTime << " [ BREAK ] " << "The connection is down! Enter
the " << state << " state!" << endl;
14         return 1;
15     }
16     return 0;
17 }

```

客户端：客户端在ESTABLISHED状态下可以通过输入控制信号选择**断开连接**，执行breakConnection函数，在该函数中客户端会先使用**停等发送功能**发送一个FIN请求，进入FIN_WAIT_1状态，当收到ACK确认报文后进入FIN_WAIT_2状态，之后使用**停等接收功能**，等待一个FIN数据包并回复ACK确认，表示可以断开连接，此时客户端进入TIME_WAIT状态，等待2MSL后**成功断开连接**，进入CLOSE状态。

```

1
2 bool breakConnection() {
3     Packet sendFIN, recvACK, recvFIN, sendACK;
4     sendFIN.Seq = sendSeq++;
5     setFIN(&sendFIN);
6     if (stopwaitSend(&sendFIN, &recvACK)) {
7         state = FIN_WAIT_2;
8     }
9     else {
10        state = ESTABLISHED;
11        return 0;
12    }
13    if (stopwaitRecv(&recvFIN, &sendACK)) {
14        state = TIME_WAIT;
15    }
16    else {
17        state = ESTABLISHED;
18        return 0;
19    }
20    _sleep(2*MSL);
21    state = CLOSE;
22    return 1;
23 }

```

差错检测

使用**校验和**检测是否出现差错，其计算与校验的过程如下所示：

```

1 //设置校验和
2 void setChecksum(Packet* t){
3     int sum = 0;
4     u_char* packet = (u_char*)t;
5     for (int i = 0; i < 16; i++){//对报文的前32字节 即256位进行校验计算
6         sum += packet[2 * i] << 8 + packet[2 * i + 1];
7         while (sum > 0xFFFF) {//溢出后将高十六位加到低十六位上
8             int sumh = sum >> 16;
9             int suml = sum & 0xFFFF;
10            sum = sumh + suml;
11        }
12    }

```

```

13     t->checksum = ~(u_short)sum; //按位取反
14 }

```

```

1  //校验
2  bool checkChecksum(Packet* t){
3      int sum = 0;
4      u_char* packet = (u_char*)t;
5      for (int i = 0; i < 16; i++) { //对报文的前32字节 即256位进行校验计算
6          sum += packet[2 * i] << 8 + packet[2 * i + 1];
7          while (sum > 0xFFFF) { //溢出后将高十六位加到低十六位上
8              int sumh = sum >> 16;
9              int suml = sum & 0xFFFF;
10             sum = sumh + suml;
11         }
12     }
13     //校验和与报文中该字段相加，等于0xFFFF则校验成功
14     if (t->checksum + (u_short)sum == 0xFFFF) {
15         return true;
16     }
17     return false;
18 }

```

确认重传

该部分的实现主要在**停等机制**中提到的代码里，在发送数据后会打开**计时器**并进入循环中不断接收数据，对于收到的数据报文会先进行**校验和的计算与检验**，确定数据帧没有问题后，检查是否为**ACK确认**以及其确认序号**Ack的值**是否为发出数据的序号**Seq值加一**，均无问题后表示数据发送成功；而若在**规定时间内**未收到正确的ACK确认，则进行重发，当**重发次数超过最大重传次数**后，**发送失败**放弃发送。

```

1  //发包函数
2  bool recvPacket(Packet* recvP) {
3      memset(recvP, 0, sizeof(sizeof(&recvP)));
4      int re = recvfrom(sockClient, (char*)recvP, sizeof(Packet), 0, (struct
sockaddr*)&sockAddr, &addr_len);
5      if (re != -1 && checkChecksum(recvP)) { //检查校验和
6          return true;
7      }
8      return false;
9  }
10 //停等发送函数
11 bool stopWaitSend(Packet* sendP, Packet* recvP) {
12     sendPacket(sendP);
13     if (checkFIN(sendP) && state != FIN_WAIT_1) {
14         state = FIN_WAIT_1;
15     }
16     clockStart = clock(); //开始计时
17     int retransNum = 0; //重发次数
18     while (1){
19         if (recvPacket(recvP)) {
20             if (checkACK(recvP) && recvP->Ack == (sendP->Seq + 1)){
21                 return 1; //收到对sendP的确认报文
22             }
23         }
24         clockEnd = clock();

```

```

25         if (retransNum == RETRANSMISSION_TIMES) { //到达最大重发次数
26             return 0;
27         }
28         if ((clockEnd - clockStart) >= WAIT_TIME) { //超时重发
29             retransNum++;
30             sendPacket(sendP);
31             clockStart = clock();
32         }
33     }
34     return 0; //发送失败
35 }

```

文件传输

发送端可以通过输入控制信号决定是否要发送文件，输入文件名后执行**sendFile**函数，首先发送端会将文件读入**缓冲区**并根据设置好的**最大数据长度**对文件进行分段，并记录长度；之后会使用**停等发送功能**发送一个设置**SF标志位**的数据包，其中包含文件名，文件分片数目等信息，表示开始发送文件；之后将**分段的文件按序打包**，设置**FILE标志**，其中文件分片序号、分片数据长度等信息，使用**停等发送功能**依次发送，当发送到最后一个包时设置其标志位为**EF**表示文件**发送结束**，计算吞吐率。

```

1  int sendFile(char* fileName) {
2      if (state != ESTABLISHED) { //连接建立后才可发送
3          return 0;
4      }
5      if (!readFile(fileName)) {
6          return 0;
7      }
8      clock_t timeStart = clock(); //timer
9      int nameLength = strlen(fileName);
10     Packet sendFileName, recvACK;
11     setStart(&sendFileName);
12     sendFileName.Seq = sendSeq++;
13     sendFileName.index = PacketNum;
14     sendFileName.length = nameLength;
15     for (int i = 0; i < nameLength; i++) {
16         sendFileName.Data[i] = fileName[i];
17     }
18     if (!stopwaitSend(&sendFileName, &recvACK)) {
19         return 0;
20     }
21     for (int i = 0; i <= PacketNum; i++) {
22         Packet sendFile;
23         sendFile.Seq = sendSeq++;
24         sendFile.index = i;
25         setFile(&sendFile);
26         if (i == PacketNum) {
27             setEnd(&sendFile);
28             sendFile.length = data_p;
29         }
30         else {
31             sendFile.length = BUF_SIZE;
32         }
33         for (int j = 0; j < sendFile.length; j++) {
34             sendFile.Data[j] = dataToSend[i][j];

```



```

35     }
36     if (!stopwaitSend(&sendFile, &recvACK)) {
37         return 0;
38     }
39 }
40 //吞吐率计算
41 clock_t timeEnd = clock();
42 double totalTime = (double)(timeEnd - timeStart) / CLOCKS_PER_SEC;
43 double RPS = (double)(PacketNum + 2) * sizeof(Packet) * 8 / totalTime /
1024 / 1024;
44 return 1;
45 }

```

接收端在接收到**SF**数据包后会执行**recvFile**函数，先将SF数据包进行解析，得到文件名、文件分片数量等信息，之后进入**停等接收状态**，依次接收后续的文件分片内容，直到收到最后一块时检查**EF**位，文件接收结束，将文件导出。

```

1  int recvFile(Packet recvFileName){
2      Packet sendACK;
3      sendACK.Seq = sendSeq++;
4      setACK(&sendACK, &recvFileName);
5      sendPacket(&sendACK);
6      //获取文件名
7      int PacketNum = recvFileName.index;
8      int nameLength = recvFileName.length;
9      char* fileName = new char[nameLength + 1];
10     memset(fileName, 0, nameLength + 1);
11     for (int i = 0; i < nameLength; i++) {
12         fileName[i] = recvFileName.Data[i];
13     }
14     fileName[nameLength] = '\0';
15
16     int lastLength = 0;
17     for (int i = 0; i <= PacketNum; i++) {
18         Packet recvFile, sendACK;
19         memset(recvData[i], 0, BUF_SIZE);
20         if (stopwaitRecv(&recvFile, &sendACK)) {
21             if (recvFile.index != i) {
22                 return 0;
23             }
24             for (int j = 0; j < recvFile.length; j++) {
25                 recvData[i][j] = recvFile.Data[j];
26             }
27         }
28         else {
29             return 0;
30         }
31         if (i == PacketNum) {
32             lastLength = recvFile.length;
33             if (!checkEnd(&recvFile)) {
34                 return 0;
35             }
36         }
37     }
38     outFile(fileName, PacketNum, BUF_SIZE, lastLength);

```

```
39     return 1;
40 }
```

日志输出

对于客户端与服务端运行时的不同状态，均会在相应的地方进行日志的输出打印，此外对于报文的内容也会进行打印，该部分主要由如下代码实现：

```
1  string PacketLog(Packet* toLog) {
2      string log = "[ ";
3      if (checkSYN(toLog)) {
4          log += "SYN ";
5      }
6      if (checkACK(toLog)) {
7          log += "ACK ";
8      }
9      if (checkFIN(toLog)) {
10         log += "FIN ";
11     }
12     if (checkStart(toLog)) {
13         log += "SF ";
14     }
15     if (checkFile(toLog)) {
16         log += "FILE ";
17     }
18     if (checkEnd(toLog)) {
19         log += "EF ";
20     }
21     log += "] ";
22     log += "Seq=";
23     log += to_string(toLog->Seq);
24     if (checkACK(toLog)) {
25         log += " Ack=";
26         log += to_string(toLog->Ack);
27     }
28     if (checkFile(toLog)) {
29         log += " Index=";
30         log += to_string(toLog->index);
31     }
32     log += " CheckNum=";
33     log += to_string(toLog->checksum);
34     return log;
35 }
```

程序执行演示

超时重传

在客户端对未启动的服务端发起连接，检验超时重传功能：

```
D:\大三上课程\计算机网络\Computer-Network\Lab3-1\UDP_Client\UDP_Client.exe
2022/11/19 20:10:03 [ INFO ] Client is created successfully
2022/11/19 20:10:03 [ RINC ] Connect(1) Or Exit(0) : 1
2022/11/19 20:10:05 [ RINC ] Please enter the Server IP address: 127.0.0.1
2022/11/19 20:10:07 [ CONN ] Start building connections!
2022/11/19 20:10:07 [ CONN ] Starting synchronization!
2022/11/19 20:10:07 [ STATE ] Enter the 2 state!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 1 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 2 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 3 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 4 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 5 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 6 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 7 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 8 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 9 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] Too long waiting time, retrans the 10 time!
2022/11/19 20:10:07 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:10:07 [ TIMEO ] The number of retransmission times is too many, fail to send!
2022/11/19 20:10:07 [ STATE ] Enter the 0 state!
2022/11/19 20:10:07 [ ERRO ] Connection establishment failure!
```

建立连接与断开连接

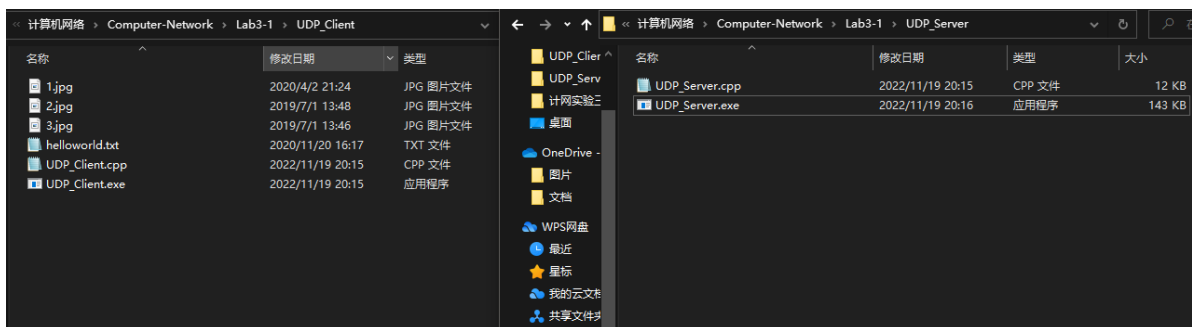
测试请求建立连接与断开连接，如下可见在三次握手后成功建立连接，四次挥手后成功断开连接。

```
D:\大三上课程\计算机网络\Computer-Network\Lab3-1\UDP_Server\UDP_Server.exe
2022/11/19 20:11:48 [ INFO ] Server is created successfully!
2022/11/19 20:11:48 [ STATE ] Enter the 0 state!
2022/11/19 20:11:48 [ STATE ] Enter the 1 state!
2022/11/19 20:11:55 [ RECV ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:11:55 [ STATE ] Enter the 3 state!
2022/11/19 20:11:55 [ CONN ] Start building connections!
2022/11/19 20:11:55 [ CONN ] Starting synchronization!
2022/11/19 20:11:55 [ SEND ] [ SYN ACK ] Seq=0 Ack=1 CheckNum=29823
2022/11/19 20:11:55 [ CONN ] Waiting . . .
2022/11/19 20:11:55 [ RECV ] [ ACK ] Seq=1 Ack=1 CheckNum=29823
2022/11/19 20:11:55 [ CONN ] The connection has been established successfully!
2022/11/19 20:11:57 [ RECV ] [ FIN ] Seq=2 CheckNum=22783
2022/11/19 20:11:57 [ STATE ] Start disconnecting, Enter the 6 state!
2022/11/19 20:11:57 [ SEND ] [ ACK ] Seq=1 Ack=3 CheckNum=29311
2022/11/19 20:11:57 [ SEND ] [ FIN ] Seq=2 CheckNum=22783
2022/11/19 20:11:57 [ RECV ] [ ACK ] Seq=3 Ack=3 CheckNum=28799
2022/11/19 20:11:57 [ BREAK ] The connection is down! Enter the 0 state!

D:\大三上课程\计算机网络\Computer-Network\Lab3-1\UDP_Client\UDP_Client.exe
2022/11/19 20:11:44 [ INFO ] Client is created successfully
2022/11/19 20:11:44 [ RINC ] Connect(1) Or Exit(0) : 1
2022/11/19 20:11:44 [ RINC ] Please enter the Server IP address: 127.0.0.1
2022/11/19 20:11:55 [ CONN ] Start building connections!
2022/11/19 20:11:55 [ CONN ] Starting synchronization!
2022/11/19 20:11:55 [ STATE ] Enter the 2 state!
2022/11/19 20:11:55 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:11:55 [ RECV ] [ SYN ACK ] Seq=0 Ack=1 CheckNum=29823
2022/11/19 20:11:55 [ SEND ] [ ACK ] Seq=1 Ack=1 CheckNum=29823
2022/11/19 20:11:55 [ CONN ] The connection has been established successfully!
2022/11/19 20:11:55 [ RINC ] Send File(1) Or Disconnect(0) : 0
2022/11/19 20:11:57 [ BREAK ] Start disconnecting!
2022/11/19 20:11:57 [ SEND ] [ FIN ] Seq=2 CheckNum=22783
2022/11/19 20:11:57 [ STATE ] Enter the 5 state!
2022/11/19 20:11:57 [ RECV ] [ ACK ] Seq=1 Ack=3 CheckNum=29311
2022/11/19 20:11:57 [ STATE ] Enter the 7 state!
2022/11/19 20:11:57 [ RECV ] [ FIN ] Seq=2 CheckNum=22783
2022/11/19 20:11:57 [ SEND ] [ ACK ] Seq=3 Ack=3 CheckNum=28799
2022/11/19 20:11:57 [ STATE ] Enter the 9 state!
2022/11/19 20:11:57 [ BREAK ] The connection is down! Enter the 0 state!
2022/11/19 20:11:57 [ RINC ] Connect(1) Or Exit(0) :
```

测试文件传输

首先将测试文件移到客户端目录下，并确保服务端目录下无其他文件：



之后打开客户端与服务端开始传输测试文件 1.jpg：

```

D:\大三上课程\计算机网络\Computer-Network\Lab3-1\UDP_Server\UDP_Server.exe
2022/11/19 20:17:34 [ INFO ] Server is created successfully!
2022/11/19 20:17:34 [ STATE ] Enter the 0 state!
2022/11/19 20:17:34 [ STATE ] Enter the 1 state!
2022/11/19 20:17:40 [ RECV ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:17:40 [ STATE ] Enter the 3 state!
2022/11/19 20:17:40 [ CONN ] Start building connections!
2022/11/19 20:17:40 [ CONN ] Starting synchronization!
2022/11/19 20:17:40 [ SEND ] [ SYN ACK ] Seq=0 Ack=1 CheckNum=29823
2022/11/19 20:17:40 [ RECV ] Waiting . . .
2022/11/19 20:17:40 [ ACK ] Seq=1 Ack=1 CheckNum=29823
2022/11/19 20:17:40 [ CONN ] The connection has been established successfully!
2022/11/19 20:20:07 [ RECV ] [ SF ] Seq=2 CheckNum=64244
2022/11/19 20:20:07 [ SEND ] [ ACK ] Seq=1 Ack=3 CheckNum=29311
2022/11/19 20:20:07 [ PRECV ] Start to receive file: 1.jpg
2022/11/19 20:20:07 [ RECV ] [ FILE ] Seq=3 Index=0 CheckNum=20223
2022/11/19 20:20:07 [ SEND ] [ ACK ] Seq=2 Ack=4 CheckNum=28799
2022/11/19 20:20:07 [ RECV ] [ FILE ] Seq=4 Index=1 CheckNum=19711
2022/11/19 20:20:07 [ SEND ] [ ACK ] Seq=3 Ack=5 CheckNum=60927
2022/11/19 20:20:07 [ RECV ] [ FILE ] Seq=5 Index=2 CheckNum=19199
2022/11/19 20:20:07 [ SEND ] [ ACK ] Seq=4 Ack=6 CheckNum=60415
2022/11/19 20:20:07 [ RECV ] [ FILE ] Seq=6 Index=3 CheckNum=18687
2022/11/19 20:20:07 [ SEND ] [ ACK ] Seq=5 Ack=7 CheckNum=59903
2022/11/19 20:20:07 [ RECV ] [ FILE ] Seq=7 Index=4 CheckNum=18175

D:\大三上课程\计算机网络\Computer-Network\Lab3-1\UDP_Client\UDP_Client.exe
2022/11/19 20:17:30 [ INFO ] Client is created successfully
2022/11/19 20:17:30 [ RINC ] Connect(1) Or Exit(0) : 1
2022/11/19 20:17:33 [ RINC ] Please enter the Server IP address: 127.0.0.1
2022/11/19 20:17:40 [ CONN ] Start building connections!
2022/11/19 20:17:40 [ CONN ] Starting synchronization!
2022/11/19 20:17:40 [ STATE ] Enter the 2 state!
2022/11/19 20:17:40 [ SEND ] [ SYN ] Seq=0 CheckNum=24063
2022/11/19 20:17:40 [ RECV ] [ SYN ACK ] Seq=0 Ack=1 CheckNum=29823
2022/11/19 20:17:40 [ SEND ] [ ACK ] Seq=1 Ack=1 CheckNum=29823
2022/11/19 20:17:40 [ CONN ] The connection has been established successfully!
2022/11/19 20:17:40 [ RINC ] Send File(1) Or Disconnect(0) : 1
2022/11/19 20:19:53 [ RINC ] Please enter the name of file to send: 1.jpg
2022/11/19 20:20:06 [ FIN ] Start to input file: 1.jpg
2022/11/19 20:20:07 [ FIN ] File 1.jpg input succeeded!
2022/11/19 20:20:07 [ FSEND ] Start to send file: 1.jpg
2022/11/19 20:20:07 [ SEND ] [ SF ] Seq=2 CheckNum=64244
2022/11/19 20:20:07 [ RECV ] [ ACK ] Seq=1 Ack=3 CheckNum=29311
2022/11/19 20:20:07 [ SEND ] [ FILE ] Seq=3 Index=0 CheckNum=20223
2022/11/19 20:20:07 [ RECV ] [ ACK ] Seq=2 Ack=4 CheckNum=28799
2022/11/19 20:20:07 [ SEND ] [ FILE ] Seq=4 Index=1 CheckNum=19711
2022/11/19 20:20:07 [ RECV ] [ ACK ] Seq=3 Ack=5 CheckNum=60927
2022/11/19 20:20:07 [ SEND ] [ FILE ] Seq=5 Index=2 CheckNum=19199
2022/11/19 20:20:07 [ RECV ] [ ACK ] Seq=4 Ack=6 CheckNum=60415

D:\大三上课程\计算机网络\Computer-Network\Lab3-1\UDP_Server\UDP_Server.exe
2022/11/19 20:20:10 [ RECV ] [ FILE ] Seq=1807 Index=1804 CheckNum=53745
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1806 Ack=1808 CheckNum=62960
2022/11/19 20:20:10 [ RECV ] [ FILE ] Seq=1808 Index=1806 CheckNum=53744
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1807 Ack=1809 CheckNum=62959
2022/11/19 20:20:10 [ RECV ] [ FILE ] Seq=1809 Index=1806 CheckNum=53743
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1808 Ack=1810 CheckNum=62958
2022/11/19 20:20:10 [ RECV ] [ FILE ] Seq=1810 Index=1807 CheckNum=53742
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1809 Ack=1811 CheckNum=62957
2022/11/19 20:20:10 [ RECV ] [ FILE ] Seq=1812 Index=1808 CheckNum=53741
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1810 Ack=1812 CheckNum=62956
2022/11/19 20:20:10 [ RECV ] [ FILE ] Seq=1812 Index=1809 CheckNum=53740
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1811 Ack=1813 CheckNum=62955
2022/11/19 20:20:10 [ RECV ] [ FILE ] Seq=1813 Index=1810 CheckNum=53739
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1812 Ack=1814 CheckNum=62954
2022/11/19 20:20:10 [ RECV ] [ FILE ] Seq=1814 Index=1811 CheckNum=53738
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1813 Ack=1815 CheckNum=62953
2022/11/19 20:20:10 [ RECV ] [ FILE ] Seq=1815 Index=1812 CheckNum=53737
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1814 Ack=1816 CheckNum=62952
2022/11/19 20:20:10 [ RECV ] [ FILE EF ] Seq=1816 Index=1813 CheckNum=27110
2022/11/19 20:20:10 [ SEND ] [ ACK ] Seq=1815 Ack=1817 CheckNum=62951
2022/11/19 20:20:10 [ PRECV ] File 1.jpg received successfully!
2022/11/19 20:20:10 [ FPREV ] Start to output file: 1.jpg
2022/11/19 20:20:10 [ FOUT ] File 1.jpg output succeeded!
2022/11/19 20:20:17 [ RECV ] [ FIN ] Seq=1817 CheckNum=56050
2022/11/19 20:20:17 [ STATE ] Start disconnecting, Enter the 6 state!
2022/11/19 20:20:17 [ SEND ] [ ACK ] Seq=1816 Ack=1818 CheckNum=30310
2022/11/19 20:20:17 [ RECV ] [ FIN ] Seq=1817 CheckNum=56050
2022/11/19 20:20:17 [ SEND ] [ ACK ] Seq=1818 Ack=1818 CheckNum=30309
2022/11/19 20:20:17 [ RECV ] [ ACK ] Seq=1818 Ack=1818 CheckNum=30309
2022/11/19 20:20:17 [ BREAK ] The connection is down! Enter the 0 state!

D:\大三上课程\计算机网络\Computer-Network\Lab3-1\UDP_Client\UDP_Client.exe
2022/11/19 20:20:10 [ RECV ] [ ACK ] Seq=1807 Ack=1809 CheckNum=62959
2022/11/19 20:20:10 [ SEND ] [ FILE ] Seq=1809 Index=1806 CheckNum=53743
2022/11/19 20:20:10 [ RECV ] [ ACK ] Seq=1808 Ack=1810 CheckNum=62958
2022/11/19 20:20:10 [ SEND ] [ FILE ] Seq=1810 Index=1807 CheckNum=53742
2022/11/19 20:20:10 [ RECV ] [ ACK ] Seq=1809 Ack=1811 CheckNum=62957
2022/11/19 20:20:10 [ SEND ] [ FILE ] Seq=1812 Index=1808 CheckNum=53741
2022/11/19 20:20:10 [ RECV ] [ ACK ] Seq=1810 Ack=1812 CheckNum=62956
2022/11/19 20:20:10 [ SEND ] [ FILE ] Seq=1812 Index=1809 CheckNum=53739
2022/11/19 20:20:10 [ RECV ] [ ACK ] Seq=1812 Ack=1814 CheckNum=62954
2022/11/19 20:20:10 [ SEND ] [ FILE ] Seq=1814 Index=1811 CheckNum=53738
2022/11/19 20:20:10 [ RECV ] [ ACK ] Seq=1813 Ack=1815 CheckNum=62953
2022/11/19 20:20:10 [ SEND ] [ FILE ] Seq=1815 Index=1812 CheckNum=53737
2022/11/19 20:20:10 [ RECV ] [ ACK ] Seq=1814 Ack=1816 CheckNum=62952
2022/11/19 20:20:10 [ SEND ] [ FILE EF ] Seq=1816 Index=1813 CheckNum=27110
2022/11/19 20:20:10 [ RECV ] [ ACK ] Seq=1815 Ack=1817 CheckNum=62951
2022/11/19 20:20:10 [ FSEND ] File 1.jpg sent successfully!
2022/11/19 20:20:10 [ INFO ] Total time: 3.063s RPS: 4.78305 Mbps
2022/11/19 20:20:10 [ RINC ] Send File(1) Or Disconnect(0) : 0
2022/11/19 20:20:17 [ BREAK ] Start disconnecting!
2022/11/19 20:20:17 [ SEND ] [ FIN ] Seq=1817 CheckNum=56050
2022/11/19 20:20:17 [ STATE ] Enter the 5 state!
2022/11/19 20:20:17 [ RECV ] [ ACK ] Seq=1816 Ack=1818 CheckNum=30310
2022/11/19 20:20:17 [ SEND ] [ FIN ] Seq=1817 CheckNum=56050
2022/11/19 20:20:17 [ RECV ] [ ACK ] Seq=1818 Ack=1818 CheckNum=30309
2022/11/19 20:20:17 [ STATE ] Enter the 9 state!
2022/11/19 20:20:18 [ BREAK ] The connection is down! Enter the 0 state!
2022/11/19 20:20:18 [ RINC ] Connect(1) Or Exit(0) :

```

如下图可见对应目录下出现了刚刚传输的图片1.jpg，且能够正常的打开，说明文件成功传输，其中由于I/O占用较长时间，因此吞吐率RPS较低。

