南開大學
Nankai University

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 (3-1)

姓名：杨馨仪

学号：2011440

专业：计算机科学与技术

2022 年 11 月 19 日

目录

1 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。

2 协议设计

本次实验中我使用数据报套接字实现了面向连接的可靠数据传输。在建立连接的过程中实现了类似于 TCP 的三次握手，实现了从客户端到服务器端以二进制单向传输文件，实现了对消息类型、校验和、序列号等成员的差错检验，实现了 rdt3.0 协议的确认重传功能，采用了停等机制的流量控制方法，断开连接实现了类似于 TCP 的四次挥手功能。

详细设计方式将在下一部分进行具体介绍。

3 模块功能介绍

3.1 UDP 协议的基本框架

在这里我们以客户端为例，介绍 UDP 协议的基本框架。

```
1  void main()
2  {
3    WSADATA wsaData;
4    WSAStartup(MAKEWORD(2, 2), &wsaData);
5
6    SOCKET sockClient = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
7
8    SOCKADDR_IN addrSrv = { 0 };
9    addrSrv.sin_family = AF_INET;//用AF_INET表示TCP/IP协议。
10   addrSrv.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");//设置为本地回环地址
11   addrSrv.sin_port = htons(4001);
12
13   SOCKADDR_IN addrClient;
14   int len = sizeof(SOCKADDR);
15
16   HandShake(sockClient, addrSrv);
17   while (1)
18   {
```

```
    ...// 以二进制方式打开文件
        SendMsg(data, sockClient, addrSrv, dataLen, Buf[0]);
  }
    WaveHand(sockClient, addrSrv) == true
  closesocket(sockClient);
  WSACleanup();
}
```

由上述代码我们可以分析出：首先，客户端（服务器端）开始工作，初始化 Socket DLL，协商使用的 Socket 版本，创建一个 socket，并绑定到 UDP 传输层服务。然后先对服务器端的地址信息（包括 IP 地址与端口号）进行初始化，服务器端使用 bind 函数将本地地址绑定到服务器 socket 上。接着进行三次握手操作，之后进入循环，对于用户输入的想要传输给服务器的文件进行传输。然后在用户的指示下，结束数据传输，主动断开连接，进行四次挥手操作。最后关闭服务器 socket，并且释放 socket DLL 资源。

客户端的流程与此类似，接下来我们对于实现可靠性传输过程中模块的介绍。

3.2 数据报格式介绍

```
struct HeadMsg {
  u_long dataLen;
  u_short len;
  u_short checkSum;
  unsigned char type;
  unsigned char seq;
  unsigned char fileNum;
  unsigned char fileTyp;
};

struct Package {
  HeadMsg hm;
  char data[8000];
};
```

本次实验中我定义了两个结构体，分别为 HeadMsg 用于存放首部，Package 用于存放整个数据报。

HeadMsg 中有七个成员：

- dataLen，4 个字节，用于存放整个文件的长度
- len，2 个字节，用于存放本数据报的数据部分长度（数据不能超过 8191 字节）
- checkSum，2 个字节，用于存放校验和
- type，1 个字节，用于存放标志位（包括 SYN，SYN_ACK，ACK，FIN_ACK，PSH，NAK 等）
- seq，1 个字节，用来存放序列号（0-255）
- fileNum，1 个字节，用来存放此次发送的文件编号
- fileTyp，1 个字节，用于存放文件类型

Package 有两个成员：

- hm，16 个字节，用于存放数据报头部
- data，8000 个字节，用于存放数据部分

3.3 数据报发送

接下来我以四个部分具体介绍数据报发送的流程。

3.3.1 读取文件

```
char Buf[BUFFER_SIZE] = {};
cin.getline(Buf, BUFFER_SIZE);
char* data;
if (strcmp(Buf, "1") == 0 || strcmp(Buf, "2") == 0 || strcmp(Buf, "3") == 0 ||
    strcmp(Buf, "4") == 0)
{
  char file[100] = "..\\test\\";
  if (Buf[0] == '1' || Buf[0] == '2' || Buf[0] == '3')
    sprintf(file, "%s%c.jpg", file, Buf[0]);
  else
    sprintf(file, "%s%s", file, "helloworld.txt");
  ifstream in(file, ifstream::in | ios::binary);
  int dataLen = 0;
  if (!in)
  {
    printf("%s [ INFO ] Client: can't open the file! Please retry\n", timei());
    continue;
  }
  // 文件读取到data
  BYTE t = in.get();
  char* data = new char[100000000];
  memset(data, 0, sizeof(data));
  while (in)
  {
    data[dataLen++] = t;
    t = in.get();
  }
  in.close();
  printf("read over\n");
```

首先根据交互获得用户想要发送的文件，然后将文件打开，将数据读取到 data，并存入文件数据的总长度，再将文件关闭。如此完成文件读取的过程。

3.3.2 发送信息

对于发送消息的过程：我们首先将得到的文件数据切分为一个一个的数据报，然后为每一个数据报设置头部信息，其中包括序列号，校验和，标志位等等。然后发送信息，并且开始计时，等待收到来

自于服务器端的 ACK 数据报。如果收到了 ACK 数据报并且验证序列号及校验和正确，则继续发下一个数据报；如果序列号，校验和或标志位错误则重新发送该数据报；如果超时，则重新发送该信息，并且重新计时。

接下来来看客户端具体的实现。

```
bool SendMsg(char* data, SOCKET sockClient, SOCKADDR_IN addrSrv, int dataLen, char
    fileNum)
{
  int sentLen = 0;
  for (int i = 0; i < dataLen / 8000 + 1; i++)
  {
    // 设置信息头
    Package p;
    p.hm.dataLen = dataLen;
    p.hm.seq = seq; //need to check
    p.hm.type = PSH;
    p.hm.checkSum = 0;
    p.hm.fileNum = fileNum;
    if (fileNum == '1' || fileNum == '2' || fileNum == '3')
      p.hm.fileTyp = JPG;
    else
      p.hm.fileTyp = TXT;

    if (i != dataLen / 8000)
      p.hm.len = 8000;
    else
      p.hm.len = dataLen % 8000;

    // data存放的是读入的二进制数据，sentLen是已发送的长度，作为分批次发送的偏移量
    memcpy(p.data, data + sentLen, p.hm.len); //把本个包的数据存进去
    sentLen += (int)p.hm.len; //发送长度加长
    // 计算校验和
    p.hm.checkSum = checkSumVerify((u_short*)&p, sizeof(p));
    SendPkg(p, sockClient, addrSrv);
    seq = (seq + 1) % 256;
  }
  return true;
}
```

客户端 SendPkg 函数

```
bool SendPkg(Package p, SOCKET sockClient, SOCKADDR_IN addrSrv)
{
  char Type[10];
  switch (p.hm.type) {
  case 1: strcpy(Type, "SYN"); break;
  case 4: strcpy(Type, "ACK"); break;
  case 8: strcpy(Type, "FIN_ACK"); break;
  case 16:strcpy(Type, "PSH"); break;
```

```
    }
    // 发送消息
    while (sendto(sockClient, (char*)&p, sizeof(p), 0, (SOCKADDR*)&addrSrv,
        sizeof(SOCKADDR)) == -1)
    {
      printf("%s [ ERR  ] Client: send [%s] ERROR:%s Seq=%d\n", timei(), Type,
          strerror(errno), p.hm.seq);
    }
    printf("%s [ INFO ] Client: [%s] Seq=%d\n", timei(), Type, p.hm.seq);

    if (!strcmp(Type, "ACK"))
      return true;
    // 开始计时
    clock_t start = clock();
    // 等待接收消息
    Package p1;
    int addrlen = sizeof(SOCKADDR);
    while (true) {
      if (recvfrom(sockClient, (char*)&p1, sizeof(p1), 0, (SOCKADDR*)&addrSrv,
          &addrlen) > 0 && clock() - start <= WAIT_TIME) {
        // 收到消息需要验证消息类型、序列号和校验和
        u_short ckSum = checkSumVerify((u_short*)&p1, sizeof(p1));
        if ((p1.hm.type == SYN_ACK && !strcmp(Type, "SYN")) && p1.hm.seq == seq &&
            ckSum == 0)
        {
          printf("%s [ GET  ] Client: receive [SYN, ACK] from Server\n", timei());
          return true;
        }
        else if ((p1.hm.type == ACK && (!strcmp(Type, "FIN_ACK") || !strcmp(Type,
            "PSH"))) && p1.hm.seq == seq && ckSum == 0)
        {
          printf("%s [ GET  ] Client: receive [ACK] from Server\n", timei());
          return true;
        }
        else {
          SendPkg(p, sockClient, addrSrv);
          return true;
          // 差错重传并重新计时
        }
      }
      else {
        SendPkg(p, sockClient, addrSrv);
        return true;
        // 超时重传并重新计时
      }
    }
}
```

这里以客户端的 SendPkg 为例。首先这个函数的参数包括打包好的 Package p, 客户端的 socket

以及服务器端的地址。首先我们获取数据报中首部的标志位信息，并将其以字符串的形式存放在 Type 中。然后发送这个数据报给服务器端。如果这个数据报的标志位只有 ACK 置 1，那么直接返回；否则开始计时，使用 recvfrom 函数接收消息。如果当前数据报标志位为 SYN，并且接收到了 SYN_ACK 的数据报，在确认序列号与校验和无误后，直接返回；如果当前数据报标志位为 FIN_ACK，并且接收到了 ACK 的数据报，在确认序列号与校验和无误后，直接返回；如果当前数据报标志位为 PSH，并且接收到了 ACK 的数据报，在确认序列号与校验和无误后，直接返回；如果不是以上三种情况，若序列号或校验和错误则触发差错重传；如果超时没有接收到数据报，则超时重传。

以上是它的整个功能，部分还要在三次握手和四次挥手中用到的，在此介绍过之后，之后不再赘述。

3.3.3 差错检验

收到消息后，我们需要检查消息是否正确。具体为判断标志位正确，序列号正确，校验和正确。

接收端每次发送 ACK 消息时，序列号都为其最后正确收到的消息的序列号。例如发送端发送一条序列号 seq=n 的消息，接收端收到后，会发送 ACK 消息确认，序列号 seq=n；若发送端的消息损坏，接收端同样会发送 ACK 消息，但是这是序列号 seq=n-1，也就是最后正确收到的消息是 n-1 号消息，以此来告诉发送端你的消息损坏了。因此，在发送端，我们只需要正确设置序列号就可以了。本文章的消息头中，序列号有 8 位，可以表示 0-255。在发送端，我们只需要维护一个全局变量 seq 即可。

```
// 初始化8位序列号
unsigned char seq = 0;

// 每次成功发送消息后，序列号+1，但是要注意序列号空间有限
seq = (seq + 1) % 256;
```

校验和是消息头中的冗余字段，用来检测数据报传输过程中出现的差错。校验和的计算方法是：将消息头的校验和设置为 0，将消息头和数据看成 16 位整数序列，不足 16 位的最后补 0，每 16 位相加，溢出的部分加到最低位上，最后的结果取反。

接收端接收到数据时，需要用同样的方法计算校验和，但是不需要先将校验和清零。如果校验和结果全为 0，说明消息正确，否则，说明消息损坏。

实现校验和的具体代码如下：

```
u_short checkSumVerify(u_short* msg, int length) {
  int count = (length + 1) / 2;
  u_long checkSum = 0;//32bit
  while (count--) {
    checkSum += *msg++;
    if (checkSum & 0xffff0000) {
      checkSum &= 0xffff;
      checkSum++;
    }
  }
  return ~(checkSum & 0xffff);
}
```

3.3.4 超时重传

超时重传就是如果超出最大响应时间还没有收到 ACK 消息，则重新发送数据报。在这里我们使用了以下代码为接收信息设置超时时间：

```
struct timeval timeo = { 20,0 };
socklen_t lens = sizeof(timeo);

SOCKET sockClient = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
setsockopt(sockClient, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeo, lens);
```

3.4 建立连接——三次握手

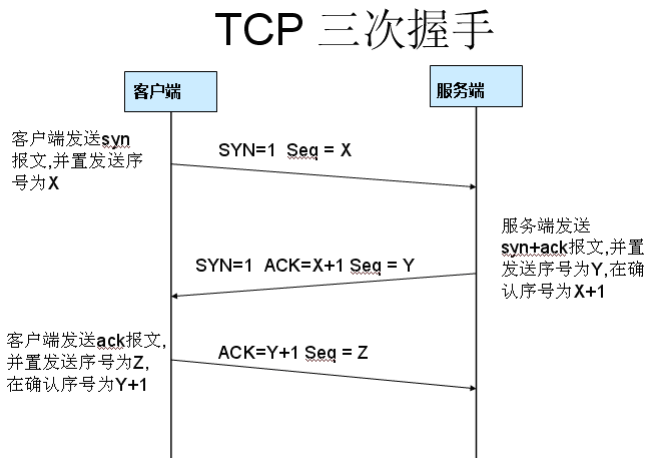
三次握手连接建立过程:

Step1： Client 将标志位 SYN 置为 1，产生 seq 值为 J，并将该数据包发送给 Server，Client 进入 SYN_SENT 状态，等待 Server 确认, 这是第一次握手。

Step2： Server 收到数据包后由标志位 SYN=1 知道 Client 请求建立连接，Server 将标志位 SYN 和 ACK 都置为 1，产生序列号 seq 为 J，代表确认收到客户端的建立连接请求。并将该数据包发送给 Client 以确认连接请求，Server 进入 SYN_RCVD 状态, 这是第二次握手。

Step3： Client 收到确认后，检查 seq 是否为 J，ACK,SYN 标志位是否为 1，如果正确则将标志位 ACK 置为 1，并将该数据包发送给 Server，Server 检查 ACK 是否为 1，如果正确则连接建立成功，Client 和 Server 进入 ESTABLISHED 状态，完成三次握手，随后 Client 与 Server 之间可以开始传输数据了。

TCP 连接建立，开始通讯。



然后我们来看一看客户端与服务器端的握手函数：

客户端 HandShake 函数

```
bool HandShake(SOCKET sockClient, SOCKADDR_IN addrSrv)
{
  char sendBuf[BUFFER_SIZE] = {};
  cin.getline(sendBuf, BUFFER_SIZE);
  if (strcmp(sendBuf, "connect") != 0)
    return false;
```

```
  Package p1;
  p1.hm.type = SYN;
  p1.hm.seq = seq;
  p1.hm.checkSum = 0;
  p1.hm.checkSum = checkSumVerify((u_short*)&p1, sizeof(p1));
  int len = sizeof(SOCKADDR);
  SendPkg(p1, sockClient, addrSrv);
  seq = (seq + 1) % 256;
  p1.hm.type = ACK;
  p1.hm.seq = seq;
  p1.hm.checkSum = 0;
  p1.hm.checkSum = checkSumVerify((u_short*)&p1, sizeof(p1));

  if (sendto(sockClient, (char*)&p1, sizeof(p1), 0, (SOCKADDR*)&addrSrv,
      sizeof(SOCKADDR)) != -1)
  {
    printf("%s [ INFO ] Client: [ACK] Seq=%d\n", timei(), seq);
    seq = (seq + 1) % 256;
    return true;
  }
  else
  {
    printf("%s [ ERR  ] Client: send [ACK] ERROR\n", timei());
    return false;
  }
}
```

服务器端 HandShake 函数

```
bool HandShake(SOCKET sockSrv, SOCKADDR_IN addrClient)
{
  Package p2;
  int len = sizeof(SOCKADDR);
  while (true)
  {
    if (recvfrom(sockSrv, (char*)&p2, sizeof(p2), 0, (SOCKADDR*)&addrClient, &len) >
        0)
    {
      int ck = checkSumVerify((u_short*)&p2, sizeof(p2));
      if (p2.hm.type == SYN && ck == 0)
      {
        printf("%s [ GET  ] Server: receive [SYN] from Client\n", timei());
        Package p3;
        p3.hm.type = SYN_ACK;
        p3.hm.seq = (lastAck + 1) % 256;
        lastAck = (lastAck + 2) % 256;
        p3.hm.checkSum = 0;
        p3.hm.checkSum = checkSumVerify((u_short*)&p3, sizeof(p3));
        SendPkg(p3, sockSrv, addrClient);
```

```
            break;
        }
        else
        {
            printf("%s [ ERR  ] Server: receive [SYN] ERROR\n", timei());
            return false;
        }
    }
  }
  return true;
}
```

通过服务器端与客户端的多次交互，我们可以发现实现了开始所说的三次握手功能，并且其中也包含有差错检验与确认重传，超时重传的相关功能。

3.5 断开连接——四次挥手

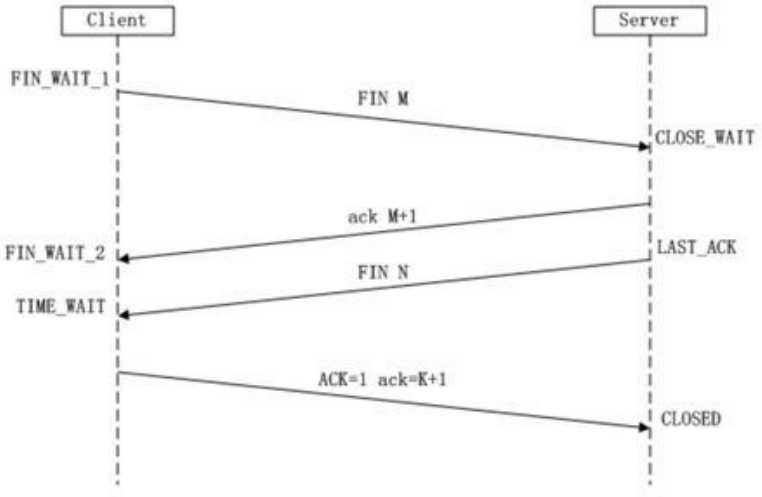
四次挥手即终止连接，就是指断开一个 TCP 连接时，需要客户端和服务端总共发送 4 个包以确认连接的断开。

Step1: Client 发送一个 FIN，用来关闭 Client 到 Server 的数据传送，Client 进入 FIN_WAIT_1 状态。

Step2: Server 收到 FIN 后，发送一个 ACK 给 Client，序号为收到序号，Server 进入 CLOSE_WAIT 状态。

Step3: Server 发送一个 FIN，用来关闭 Server 到 Client 的数据传送，Server 进入 LAST_ACK 状态。

Step4: Client 收到 FIN 后，Client 进入 TIME_WAIT 状态，接着发送一个 ACK 给 Server，序号为收到序号，Server 进入 CLOSED 状态，完成四次挥手。



实验中我们通过 WaveHand 函数来实现四次挥手功能，在其中依旧调用了 SendMsg 函数。

客户端 WaveHand 函数

```
bool WaveHand(SOCKET sockClient, SOCKADDR_IN addrSrv)
{
```

```
  Package p1;
  p1.hm.type = FIN_ACK;
  p1.hm.seq = seq;
  p1.hm.checkSum = 0;
  p1.hm.checkSum = checkSumVerify((u_short*)&p1, sizeof(p1));
  int len = sizeof(SOCKADDR);

  SendPkg(p1, sockClient, addrSrv);
  Package p4;

  while (true)
  {
    if (recvfrom(sockClient, (char*)&p4, sizeof(p4), 0, (SOCKADDR*)&addrSrv, &len) >
        0)
    {
      if (p4.hm.type == FIN_ACK)
      {
        printf("%s [ GET  ] Client: receive [FIN, ACK] from Server\n", timei());
        p4.hm.type = ACK;
        p4.hm.checkSum = 0;
        p4.hm.checkSum = checkSumVerify((u_short*)&p4, sizeof(p4));
        SendPkg(p4, sockClient, addrSrv);
        break;
      }
      else
      {
        printf("%s [ ERR  ] Server: receive [FIN, ACK] ERROR\n", timei());
        return false;
      }
    }
  }
  return true;
}
```

服务器端 WaveHand 函数

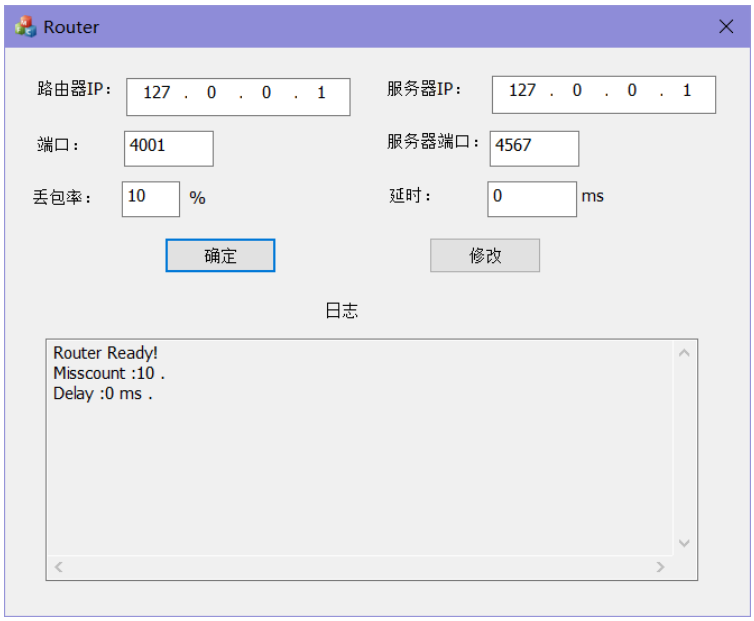
```
bool WaveHand(SOCKET sockSrv, SOCKADDR_IN addrClient, Package p2)
{
  int len = sizeof(SOCKADDR);
  u_short ckSum = checkSumVerify((u_short*)&p2, sizeof(p2));
  if (p2.hm.type == FIN_ACK && ckSum == 0)
  {
    printf("%s [ GET  ] Server: receive [FIN, ACK] from Client\n", timei());
    p2.hm.type = ACK;
    p2.hm.seq = (lastAck + 1) % 256;
    lastAck = (lastAck + 2) % 256;
    p2.hm.checkSum = 0;
    p2.hm.checkSum = checkSumVerify((u_short*)&p2, sizeof(p2));
    SendPkg(p2, sockSrv, addrClient);
```

```
14    }
15    else
16    {
17        printf("%s [ ERR  ] Server: receive [FIN, ACK] ERROR\n", timei());
18        return false;
19    }
20
21    Package p3;
22    p3.hm.type = FIN_ACK;
23    p3.hm.checkSum = 0;
24    p3.hm.checkSum = checkSumVerify((u_short*)&p2, sizeof(p2));
25    SendPkg(p3, sockSrv, addrClient);
26    return true;
27 }
```

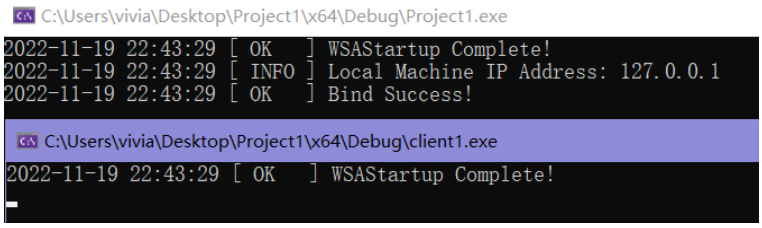
局部代码不能体现完整流程，但大致的框架如上所示，具体实现详见源码。

4 程序界面展示及运行说明

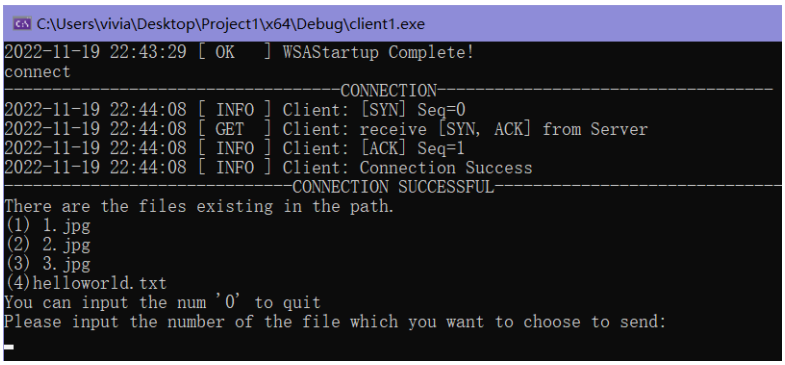
step1: 首先设置丢包率如下所示：



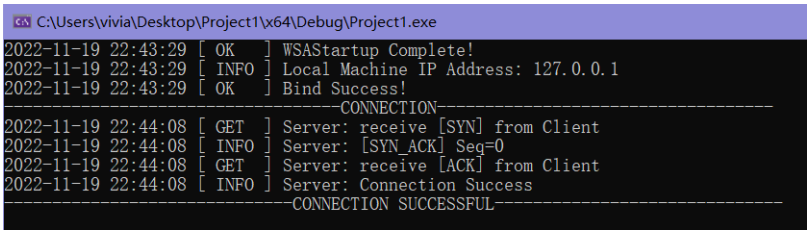
step2: 通过 visual studio 运行程序，如下是客户端与服务器端的一些初始打印信息



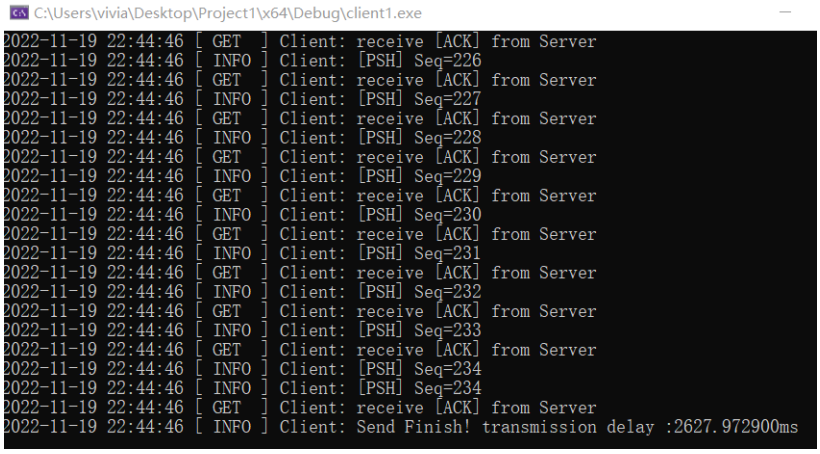
step3: 在客户端输入 connect，进行三次握手，建立连接，如下为客户端的截图



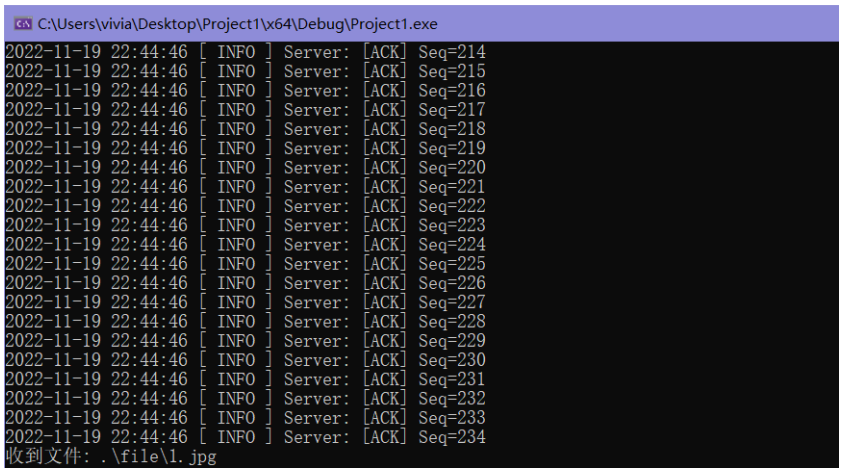
如下为服务器端的截图



step4: 在客户端输入 1，开始对第一张图片进行可靠性传输；传输完毕显示发送时延

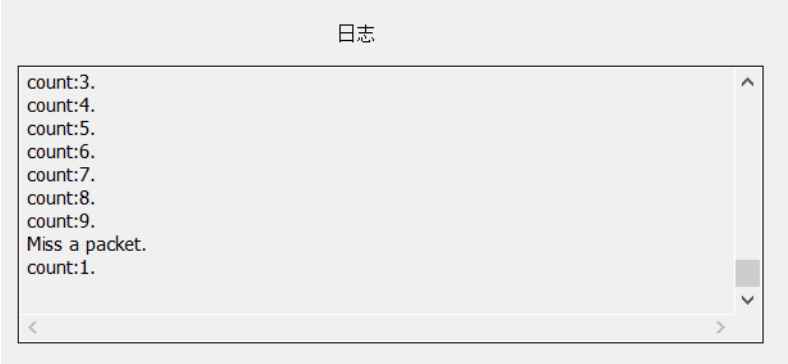


服务器端显示收到文件



我们可以看到在传输下面这个数据报时发生了丢包，所以客户端发送了两次该数据报

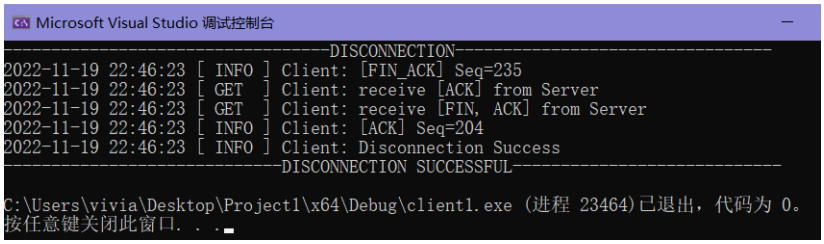
```
2022-11-19 22:44:46 [ INFO ] Client: [PSH] Seq=234
2022-11-19 22:44:46 [ INFO ] Client: [PSH] Seq=234
2022-11-19 22:44:46 [ GET  ] Client: receive [ACK] from Server
```



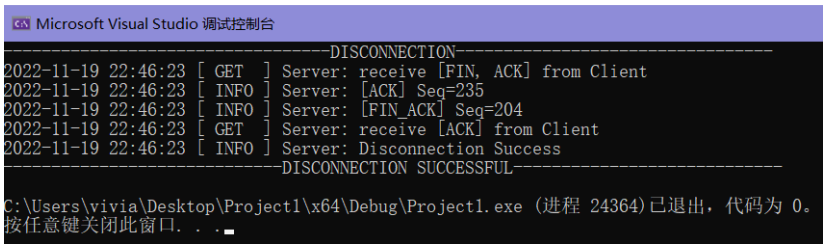
step5: 输入 5，提示没有第五个文件

```
5
2022-11-19 22:46:05 [ ERR  ] Client: Invalidate Input
```

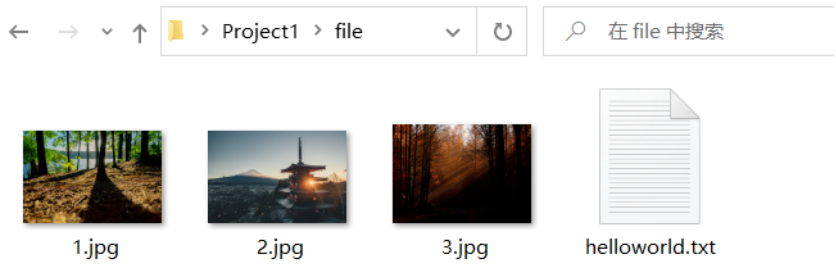
step6: 输入 0，进行四次挥手，断开连接，如下是客户端截图



如下是服务器端截图



step7: 打开文件夹，可以看到成功传输的图片与文本文件



5 实验反思

在本次实验中，经过反思我认为有以下几点可以进一步改进：

- 实验过程中在校验和上出现了很多的错误。其中有一次是没有先把首部校验和位置上填充 0 就直接计算了校验和；第二次是由于发送端与接收端计算校验和的缓冲区大小不统一导致没有通过差错检验。
- 在进行超时重传时，最开始使用了 clock 函数，但是后来发现由于 recvfrom 默认为堵塞传输，导致 recvfrom 在没有接收到数据报时一直不返回，无法计算 clock 的时间，使超时重传部分失效。在经过一系列查阅资料的过程，知道了如何设置 socket 接受数据报的超时时间。
- 本次实验只设计了序列号，没有设计确认号，再进一步丰富协议设计时，可以添加上去。