计算机网络实验报告

实验名称：Lab3-1 基于UDP服务设计可靠传输协议

学号：2012682　　姓名：韩佳迅

一、实验内容

利用**数据报套接字**（UDP）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。

二、协议设计

（一）报文格式

报文格式如下所示：

报文格式分为首部和数据部分：

首部：

- 1—4字节：源IP
- 5—8字节：目的IP
- 9—10字节：源端口号
- 11—12字节：目的端口号
- 13—16字节：序号
- 17—20字节：确认号
- 21—24字节：数据大小
- 25—26字节：标志
- 27—28字节：校验和

数据：

- 剩余字节流是数据部分

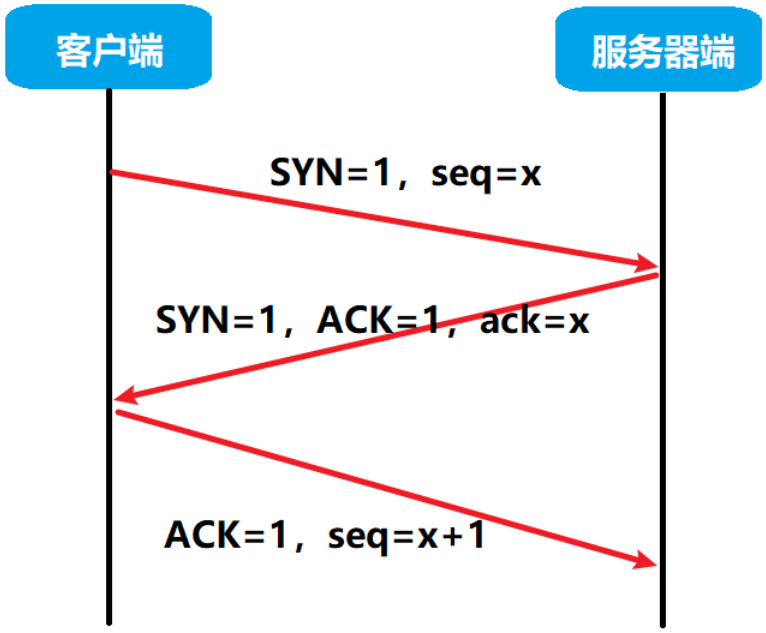
报文段的校验：

在报文段里设置校验和字段，发送时设置校验和，收到时检验校验和是否正确。若不正确，说明报文损坏。

（具体的校验方法见下面的代码实现部分）

（二）建立连接：三次握手

本程序在标准三次握手的基础上，进行了部分改进，最终实现的结构示意图如下：



1. 客户端向服务器端发送数据包：
    - SYN=1，seq=x
2. 服务器端向客户端回复数据包：
    - SYN=1，ACK=1
    - ack=x【这里对标准三次握手进行了更改：回复的ack = 上一个包发来的seq】
3. 客户端向服务器端发送数据包：
    - ACK=1
    - seq=x+1【序列号+1】

（三）数据传输：差错检测、确认重传

传输的总体流程

本实验实现了数据包的**差错检测**、**确认重传**，解决了数据包**丢失**、**失序**等问题。

将文件从客户端传输到服务器端，由于文件较大，需要分成多个报文传递。在本程序中，实现的总体流程如下：

1. 发送携带文件名和文件大小的报文
    - 文件名：在数据部分进行传输
    - 文件大小：在报文段的size域进行传输
2. 发送数据报文：根据文件大小分为最大装载报文和剩余部分报文
    - 最大装载报文：根据 $[fileSize/MaxMsgSize]$ 取整得到最大报文段个数
    - 剩余部分报文：根据 $[fileSize\%MaxMsgSize]$ 得到剩余部分报文段的大小

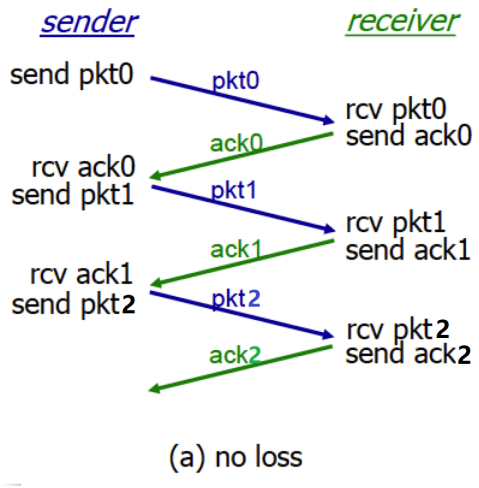
数据传输状态

- 发送端和接收端的状态机均采用 ***rdt 3.0*** 版本，并在其基础上进行了将原有的seq=0/1扩展到多位。
- 在没有丢包和失序的情况下，发送端发送一个seq=x的报文，接收端回复一个ack=x的报文

- 当出现了丢包或失序，程序则需要进行超时重传和差错检测。

  下面我们依次对各种情况以及发送端接收端采取的状态进行分析：

（a）理想情况

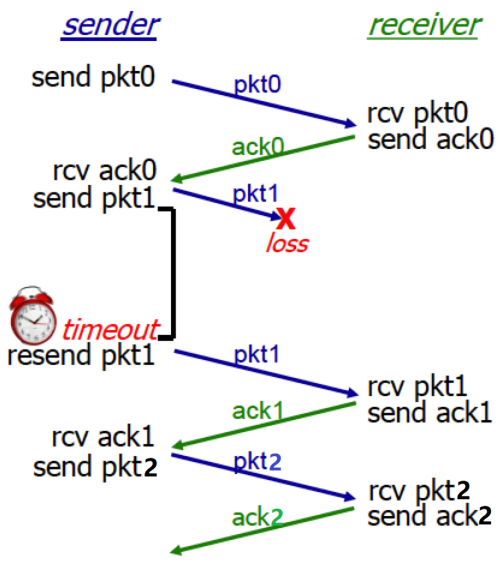


(a) no loss

当没有包丢失和失序时：

- 发送端：按序发送
- 接收端：按序接收，并返回一个确认包，且其确认号ack = 发来的序号 seq

（b）pkt丢失

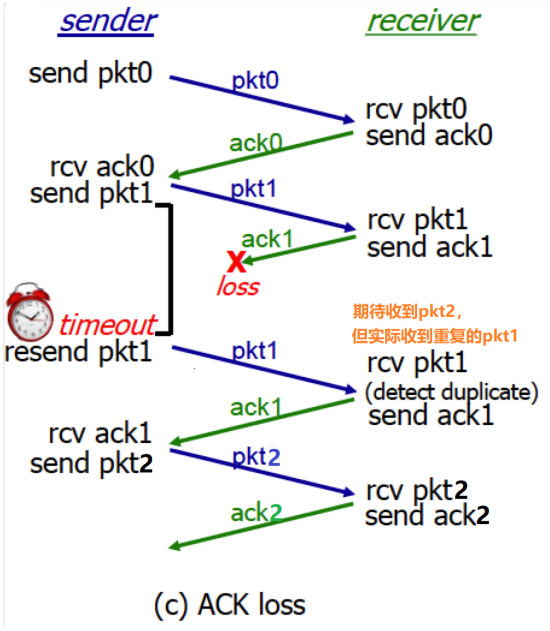


(b) packet loss

当发送的pkt包丢失时：

- 发送端：
  - 为每个包设置计时器，当超时未收到对应的ack时，重发该包
- 接收端：
  - 按序接收，并返回一个确认包，且其确认号ack = 发来的序号 seq

(c) ack丢失

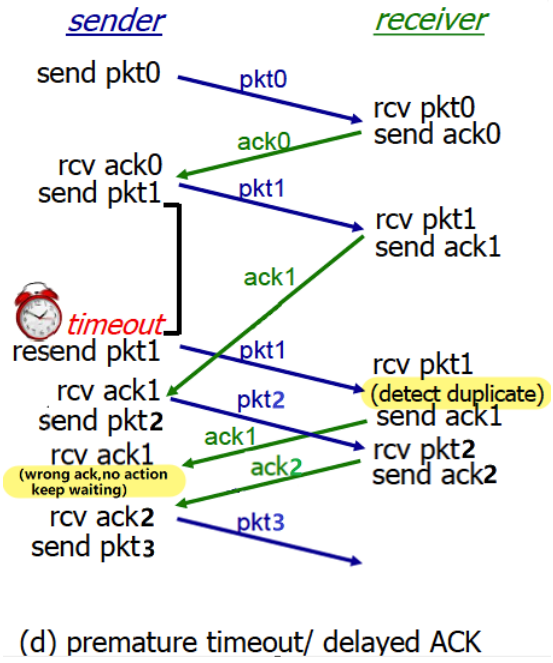


(c) ACK loss

当发送的ack包丢失时：

- 发送端：
  - 为每个包设置计时器，当超时未收到对应的ack时，重发该包
- 接收端：
  - 当收到了重复的pkt（比如上图中收到了重复的pkt1，但正在等待的是pkt2）
    - 丢掉这个pkt（不交付给上层应用）
    - 回复对应的ack

(d) 失序问题



(d) premature timeout/ delayed ACK

当发送的包失序 / ack迟到时：

- 发送端：
  - 为每个包设置计时器，当超时未收到对应的ack时，重发该包
  - 当收到的ack不等于期待收到的ack时，不采取动作，继续等待
- 接收端：

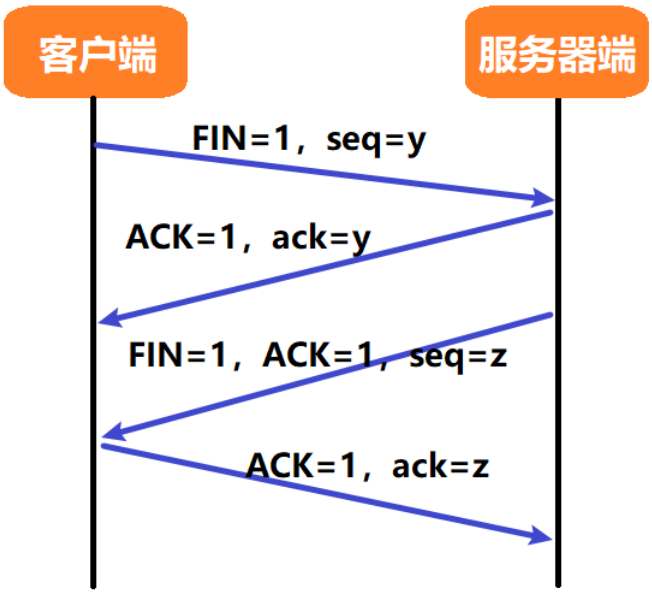
- 当收到了重复的pkt
  - 丢掉这个pkt（不交付给上层应用）
  - 回复对应的ack

状态机

- **发送端**
  - 发送 ***pkt n*** 并计时，开始等待回复
    - 超时未收到 ***ack n***，重发并重新计时
    - 收到 ***ack m （m != n）***，继续等待
  - 收到 ***ack n***
  - 继续发送下一个 ***pkt n+1***
- **接收端**
  - 等待 ***pkt n***
    - 当收到重复的pkt（***pkt m，m<n***），丢掉这个pkt，并回复对应的 ***ack m***
  - 收到 ***pkt n***，回复 ***ack n***
  - 继续等待下一个pkt

（四）关闭连接：四次挥手

本程序在标准四次挥手的基础上，进行了部分改进，最终实现的结构示意图如下：



1. 客户端向服务器端发送数据包：
   - FIN=1，seq=y
2. 服务器端向客户端回复数据包：
   - ACK=1，ack=y【回复的ack = 上一个包发来的seq】
3. 服务器端向客户端发送数据包：
   - FIN=1，ACK=1
   - seq=z
4. 客户端向服务器端发送数据包：
   - ACK=1，ack=z【回复的ack = 上一个包发来的seq】

三、各模块功能与具体实现

（一）报文段

结构体实现报文段

- 首部和数据部分由结构体的属性实现
  - 注意：标志位字段使用一个unsigned short变量 ***flag*** 实现
  - 设置SYN、ACK、FIN字段分别为0x1、0x2、0x4，置位时直接加到 ***flag*** 上即可
- 功能实现：
  - 设置校验和
  - 检验校验和

UDP校验和

- **设置校验和：**

1. 校验和全部填充为0
2. 剩余数据部分填充为0
3. 按16bit为单位反码求和：
   - 【首部+数据】每16位求和得到一个32位数
   - 如果这个32位的数，高16位不为0，则高16位加低16位再得到一个32位的数
   - 重复直到高16位为0
   - 将低16位取反，得到校验和
4. 得到的16位结果填充到结构体的校验和字段

- **检验校验和：**

1. 按照设置校验和的方式，将接收到的结构体按16bit为单位反码求和
2. 如果得出的结果低16位全1，则校验成功

```
const unsigned short SYN = 0x1;
const unsigned short ACK = 0x2;
const unsigned short FIN = 0x4;
#pragma pack(1)//让编译器将结构体数据强制连续排列，禁止优化存储结构进行对齐
struct Message
{
    //================首部（28字节 = 224bit = 14*16bit）==============
    //源IP、目的IP
    unsigned int SrcIP, DestIP;//4字节、4字节
    //源端口号、目的端口号
    unsigned short SrcPort, DestPort;//2字节、2字节
    //序号 Seq num
    unsigned int SeqNum;//4字节
    //确认号 Ack num
    unsigned int AckNum;//4字节
    //数据大小
    unsigned int size;//4字节
    //标志
    unsigned short flag;//2字节
    //校验和
```

```
    unsigned short checkNum;//2字节
    //=================报文数据（MaxMsgSize字节）===============
    BYTE msgData[MaxMsgSize];

    Message();
    bool check();
    void setCheck();
};
#pragma pack()
Message::Message()
{
    SrcIP = 0;
    DestIP = 0;//由于测试时，一直是本地回环地址，故这两个字段没有用到
    SeqNum = 0;
    AckNum = 0;
    size = 0;
    flag = 0;
    memset(&msgData, 0, sizeof(msgData));
}
void Message::setCheck()
{
    this->checkNum = 0;
    int sum = 0;
    unsigned short* msgStream = (unsigned short*)this;
    for (int i = 0; i < sizeof(*this) / 2; i++)
    {
        sum += *msgStream++;
        if (sum & 0xFFFF0000)
        {
            sum &= 0xFFFF;
            sum++;
        }
    }
    this->checkNum = ~(sum & 0xFFFF);
}
bool Message::check()
{
    unsigned int sum = 0;
    unsigned short* msgStream = (unsigned short*)this;

    for (int i = 0; i < sizeof(*this) / 2; i++)
    {
        sum += *msgStream++;
        if (sum & 0xFFFF0000)
        {
            sum &= 0xFFFF;
            sum++;
        }
    }
    if ((sum & 0xFFFF) == 0xFFFF)
    {
        return true;
    }
    return false;
}
```

（二）建立连接

客户端

- 发送第一次握手的消息
    - 计时器开始计时
- 接收第二次握手的消息
    - 若超时没收到，则重传第一次的消息
- 发送第三次握手的消息

```
//实现client（发送端）的三次握手
bool ConnectWithServer(SOCKET clientSocket, SOCKADDR_IN serverAddr)
{
    int AddrLen = sizeof(serverAddr);
    Message buffer1;
    Message buffer2;
    Message buffer3;

    //==============发送第一次握手的消息（SYN=1，seq=x）============
    buffer1.SrcPort = ClientPORT;
    buffer1.DestPort = RouterPORT;
    buffer1.flag += SYN;//设置SYN
    buffer1.SeqNum = ++global_seq;//设置序号seq
    buffer1.setCheck();//设置校验和
    int sendByte = sendto(clientSocket, (char*)&buffer1, sizeof(buffer1), 0,
(sockaddr*)&serverAddr, AddrLen);
    clock_t buffer1start = clock();
    if (sendByte == 0)
    {
        cout << "连接失败......关闭连接！" << endl;
        return false;
    }
    cout << "client已发送第一次握手的消息！" << endl;

    //==============接收第二次握手的消息（SYN=1，ACK=1，ack=x）============
    while (1)
    {
        int recvByte = recvfrom(clientSocket, (char*)&buffer2, sizeof(buffer2),
0, (sockaddr*)&serverAddr, &AddrLen);
        if (recvByte == 0)
        {
            cout << "连接失败......关闭连接！" << endl;
            return false;
        }
        else if (recvByte > 0)
        {
            //成功收到消息，检查校验和、ACK、SYN、ack
            if ((buffer2.flag && ACK) && (buffer2.flag && SYN) &&
buffer2.check() && (buffer2.AckNum == buffer1.SeqNum))
            {
```

```
                cout << "client已收到第二次握手的消息！" << endl;
                break;
            }
            else
            {
                cout << "连接发生错误！" << endl;
                return false;
            }
        }
        //buffer1超时，重新发送并重新计时
        if (clock() - buffer1start > MAX_WAIT_TIME)
        {
            cout << "第一次握手超时，正在重传......" << endl;
            //============重传buffer1==============
            int sendByte = sendto(clientSocket, (char*)&buffer1,
sizeof(buffer1), 0, (sockaddr*)&serverAddr, AddrLen);
            buffer1start = clock();
            if (sendByte == 0)
            {
                cout << "连接失败......关闭连接！" << endl;
                return false;
            }
        }
    }
    //==============发送第三次握手的消息（ACK=1，seq=x+1）============
    buffer3.SrcPort = ClientPORT;
    buffer3.DestPort = RouterPORT;
    buffer3.flag += ACK;//设置ACK
    buffer3.SeqNum = ++global_seq;//设置序号seq=x+1
    buffer3.setCheck();//设置校验和
    sendByte = sendto(clientSocket, (char*)&buffer3, sizeof(buffer3), 0,
(sockaddr*)&serverAddr, AddrLen);
    if (sendByte == 0)
    {
        cout << "连接失败......关闭连接！" << endl;
        return false;
    }
    cout << "client已发送第三次握手的消息！" << endl;
    cout << "client连接成功！" << endl;
}
```

服务器端

- 接收第一次握手的消息并检验
- 发送第二次握手的消息
- 接收第三次握手的消息并检验
    - 若超时没收到，则重传第二次的消息

```
//实现server（接收端）的三次握手
bool ConnectWithClient(SOCKET serverSocket, SOCKADDR_IN clientAddr)
{
    int AddrLen = sizeof(clientAddr);
    Message buffer1;
    Message buffer2;
```

```
    Message buffer3;
    while (1)
    {
        //=============接收第一次握手的消息（SYN=1，seq=x）============
        int recvByte = recvfrom(serverSocket, (char*)&buffer1, sizeof(buffer1),
0, (sockaddr*)&clientAddr, &AddrLen);
        if (recvByte == 0)
        {
            cout << "连接失败......关闭连接！" << endl;
            return false;
        }

        else if (recvByte > 0)
        {
            //判断SYN、检验和
            if (!(buffer1.flag && SYN) || !buffer1.check() || !(buffer1.SeqNum
== global_seq+1))
            {
                cout << "连接发生错误！" << endl;
                return false;
            }
            global_seq++;
            cout << "server已收到第一次握手的消息！" << endl;
            //=============发送第二次握手的消息（SYN=1，ACK=1，ack=x）============
            buffer2.SrcPort = ServerPORT;
            buffer2.DestPort = RouterPORT;
            buffer2.AckNum = buffer1.SeqNum;//服务器回复的ack=客户端发来的seq
            buffer2.flag += SYN;
            buffer2.flag += ACK;
            buffer2.setCheck();//设置校验和
            int sendByte = sendto(serverSocket, (char*)&buffer2,
sizeof(buffer2), 0, (sockaddr*)&clientAddr, AddrLen);
            clock_t buffer2start = clock();
            if (sendByte == 0)
            {
                cout << "连接失败......关闭连接！" << endl;
                return false;
            }
            cout << "server已发送第二次握手的消息！" << endl;

            //=============接收第三次握手的消息（ACK=1，seq=x+1）============
            while (1)
            {
                int recvByte = recvfrom(serverSocket, (char*)&buffer3,
sizeof(buffer3), 0, (sockaddr*)&clientAddr, &AddrLen);
                if (recvByte == 0)
                {
                    cout << "连接失败......关闭连接！" << endl;
                    return false;
                }
                else if (recvByte > 0)
                {
                    //成功收到消息，检查校验和、seq
                    if ((buffer3.flag && ACK) && buffer3.check() &&
(buffer3.SeqNum == global_seq + 1))
```

```
                    {
                        global_seq++;
                        cout << "server已收到第三次握手的消息！" << endl;
                        cout << "server连接成功！" << endl;
                        return true;
                    }
                    else
                    {
                        cout << "连接发生错误！" << endl;
                        return false;
                    }
                }

                //buffer2超时，重新发送并重新计时
                if (clock() - buffer2start > MAX_WAIT_TIME)
                {
                    cout << "第二次握手超时，正在重传......" << endl;
                    //============重传buffer2==============
                    int sendByte = sendto(serverSocket, (char*)&buffer2,
sizeof(buffer2), 0, (sockaddr*)&clientAddr, AddrLen);
                    buffer2start = clock(); //重新设置buffer2的时间
                    if (sendByte == 0)
                    {
                        cout << "连接失败......关闭连接！" << endl;
                        return false;
                    }
                }
            }
        }
    }
    return false;
}
```

（三）数据传输：差错检测、确认重传、丢失及失序问题的解决

客户端

- **读取文件内容：**
  - 使用ifstream以二进制方式打开文件，将文件内容读到BYTE类型数组中

```
//=================打开文件，读成字节流=================
    ifstream fin(filename.c_str(), ifstream::binary);
    if (!fin) {
        printf("无法打开文件！\n");
        return;
    }
    //文件读取到fileBuffer
    BYTE* fileBuffer = new BYTE[MaxFileSize];
    unsigned int fileSize = 0;
    BYTE byte = fin.get();
    while (fin) {
        fileBuffer[fileSize++] = byte;
        byte = fin.get();
```

```
    }
    fin.close();
```

- **发送文件**：
  - 先发送**文件名**和**文件大小**，文件名通过报文数据段传递，文件大小通过报文的size字段传递
  - 然后依次发送文件内的**数据部分**，分为最大装载报文和剩余部分的报文

    (超时重传机制、ACK失序机制见代码及注释)

```
bool sendMessage(Message& sendMsg, SOCKET clientSocket, SOCKADDR_IN serverAddr)
{
    sendto(clientSocket, (char*)&sendMsg, sizeof(sendMsg), 0,
(sockaddr*)&serverAddr, sizeof(SOCKADDR_IN));
    cout << "client已发送 Seq = " << sendMsg.SeqNum << " 的报文段！" << endl;
    int msgStart = clock();
    Message recvMsg;
    int AddrLen = sizeof(serverAddr);
    int timeOutTimes = 0;

    while (1)
    {
        int recvByte = recvfrom(clientSocket, (char*)&recvMsg, sizeof(recvMsg),
0, (sockaddr*)&serverAddr, &AddrLen);
        if (recvByte > 0)
        {
            //成功收到消息，检查校验和、ack
            if ((recvMsg.flag && ACK) && (recvMsg.AckNum == sendMsg.SeqNum))
            {
                cout << "client已收到 Ack = " << recvMsg.AckNum << "的确认报文" <<
endl;
                return true;
            }
            //若校验失败或ack不对，则忽略，继续等待
        }
        //超时，重新发送并重新计时
        if (clock() - msgStart > MAX_WAIT_TIME)
        {
            cout << "Seq = "<<sendMsg.SeqNum << "的报文段 第" << ++timeOutTimes <<
"次超时，正在重传......" << endl;
            sendto(clientSocket, (char*)&sendMsg, sizeof(sendMsg), 0,
(sockaddr*)&serverAddr, sizeof(SOCKADDR_IN));
            msgStart = clock();
        }
        if (timeOutTimes == MAX_SEND_TIMES)
        {
            cout << "超时重传超过" << MAX_SEND_TIMES << "次，传输失败！" << endl;
            break;
        }
    }
    return false;
}
```

服务器端

- 收到**文件名**和**文件大小**
- 根据文件大小计算一共将要收到几个报文，并依次接收这些报文

  （确认ACK机制、失序机制见代码及注释）

```
bool recvMessage(Message& recvMsg, SOCKET serverSocket, SOCKADDR_IN clientAddr)
{
    int AddrLen = sizeof(clientAddr);
    while (1)
    {
        int recvByte = recvfrom(serverSocket, (char*)&recvMsg, sizeof(recvMsg),
0, (sockaddr*)&clientAddr, &AddrLen);
        if (recvByte > 0)
        {
            //成功收到消息
            if (recvMsg.check() && (recvMsg.SeqNum == global_seq + 1))
            {
                global_seq++;
                //回复ACK
                Message replyMessage;
                replyMessage.SrcPort = ServerPORT;
                replyMessage.DestPort = RouterPORT;
                replyMessage.flag += ACK;
                replyMessage.AckNum = recvMsg.SeqNum;
                replyMessage.setCheck();
                sendto(serverSocket, (char*)&replyMessage, sizeof(replyMessage),
0, (sockaddr*)&clientAddr, sizeof(SOCKADDR_IN));
                cout << "server收到 Seq = " << recvMsg.SeqNum << "的报文段，并发送
Ack = " << replyMessage.AckNum << " 的回复报文段" << endl;
                return true;
            }
            //如果seq!=期待值，则传来了重复的报文：丢弃重复报文，重传该报文的回复ACK
            //（注：这种情况是由于上一个回复ACK报文段丢失，导致发送方超时重传所致，传了重复的
报文段）
            else if (recvMsg.check() && (recvMsg.SeqNum != global_seq + 1))
            {
                //回复ACK
                Message replyMessage;
                replyMessage.SrcPort = ServerPORT;
                replyMessage.DestPort = RouterPORT;
                replyMessage.flag += ACK;
                replyMessage.AckNum = recvMsg.SeqNum;
                replyMessage.setCheck();
                sendto(serverSocket, (char*)&replyMessage, sizeof(replyMessage),
0, (sockaddr*)&clientAddr, sizeof(SOCKADDR_IN));
                cout << "【重复接收报文段】server收到 Seq = " << recvMsg.SeqNum <<
"的报文段，并发送 Ack = " << replyMessage.AckNum << " 的回复报文段" << endl;
            }
        }
        else if (recvByte == 0)
        {
            return false;
        }
```

```
    }
}
```

（四）关闭连接

客户端

- 发送第一次挥手的消息
  - 计时器开始计时
- 接收第二次挥手的消息
  - 若超时没收到，则重传第一次的消息并重新计时
- 接收第三次挥手的消息
- 发送第四次挥手的消息
- 等待2MSL
  - 防止最后一个ACK丢失，处于半关闭
  - 若再次收到了消息，则回复第四次挥手的数据包

（详见下方代码及注释）

```
bool CloseConnectWithServer(SOCKET clientSocket, SOCKADDR_IN serverAddr)
{
    int AddrLen = sizeof(serverAddr);
    Message buffer1;
    Message buffer2;
    Message buffer3;
    Message buffer4;

    //==============发送第一次挥手的消息（FIN=1，seq=y）============
    buffer1.SrcPort = ClientPORT;
    buffer1.DestPort = RouterPORT;
    buffer1.flag += FIN;//设置FIN
    buffer1.SeqNum = ++global_seq;//设置序号seq
    buffer1.setCheck();//设置校验和
    int sendByte = sendto(clientSocket, (char*)&buffer1, sizeof(buffer1), 0,
(sockaddr*)&serverAddr, AddrLen);
    clock_t buffer1start = clock();
    if (sendByte == 0)
    {
        cout << "连接失败.....关闭连接！" << endl;
        return false;
    }
    cout << "client已发送第一次挥手的消息！" << endl;

    //==============接收第二次挥手的消息（ACK=1，ack=y）============
    while (1)
    {
        int recvByte = recvfrom(clientSocket, (char*)&buffer2, sizeof(buffer2),
0, (sockaddr*)&serverAddr, &AddrLen);
        if (recvByte == 0)
        {
            cout << "关闭连接error！" << endl;
```

```
            return false;
        }
        else if (recvByte > 0)
        {
            //成功收到消息，检查校验和、ACK、ack
            if ((buffer2.flag && ACK) && buffer2.check() && (buffer2.AckNum ==
buffer1.SeqNum))
            {
                cout << "client已收到第二次挥手的消息！" << endl;
                break;
            }
            else
            {
                cout << "连接发生错误！" << endl;
                return false;
            }
        }
        //buffer1超时，重新发送并重新计时
        if (clock() - buffer1start > MAX_WAIT_TIME)
        {
            cout << "第一次挥手超时，正在重传......" << endl;
            //============重传buffer1==============
            int sendByte = sendto(clientSocket, (char*)&buffer1,
sizeof(buffer1), 0, (sockaddr*)&serverAddr, AddrLen);
            buffer1start = clock();
            if (sendByte == 0)
            {
                cout << "关闭连接error！" << endl;
                return false;
            }
        }
    }


    //=============接收第三次挥手的消息（FIN=1，ACK=1，seq=z）============
    while (1)
    {
        int recvByte = recvfrom(clientSocket, (char*)&buffer3, sizeof(buffer3),
0, (sockaddr*)&serverAddr, &AddrLen);
        if (recvByte == 0)
        {
            cout << "关闭连接error！" << endl;
            return false;
        }
        else if (recvByte > 0)
        {
            //成功收到消息，检查校验和、ACK、ack
            if ((buffer3.flag && ACK)&& (buffer3.flag && FIN) &&
buffer3.check())
            {
                cout << "client已收到第三次挥手的消息！" << endl;
                break;
            }
            else
            {
```

```
                cout << "连接发生错误! " << endl;
                return false;
            }
        }
    }

    //==============发送第四次挥手的消息（ACK=1，ack=z）============
    buffer4.SrcPort = ClientPORT;
    buffer4.DestPort = RouterPORT;
    buffer4.flag += ACK;//设置ACK
    buffer4.AckNum = buffer3.SeqNum;//设置序号seq
    buffer4.setCheck();//设置校验和
    sendByte = sendto(clientSocket, (char*)&buffer4, sizeof(buffer4), 0,
(sockaddr*)&serverAddr, AddrLen);
    if (sendByte == 0)
    {
        cout << "关闭连接error! " << endl;
        return false;
    }
    cout << "client已发送第四次挥手的消息! " << endl;

    //=============第四次挥手之后还需等待2MSL，防止最后一个ACK丢失，处于半关闭=======
    int tempclock = clock();
    cout << "client端2MSL等待..." << endl;
    Message tmp;
    while (clock() - tempclock < 2 * MAX_WAIT_TIME)
    {
        int recvByte = recvfrom(clientSocket, (char*)&tmp, sizeof(tmp), 0,
(sockaddr*)&serverAddr, &AddrLen);
        if (recvByte == 0)
        {
            cout << "关闭连接error! " << endl;
            return false;
        }
        else if (recvByte > 0)
        {
            //回复丢失的ack
            sendByte = sendto(clientSocket, (char*)&buffer4, sizeof(buffer4), 0,
(sockaddr*)&serverAddr, AddrLen);
            cout << "回复" << endl;
        }
    }
    cout << "\n关闭连接成功! " << endl;
}
```

服务器端

- 接收第一次挥手的消息并检验
- 发送第二次挥手的消息
- 发送第三次挥手的消息
- 接收第四次挥手的消息并检验
    - 若超时没收到，则重传第三次的消息

```
//实现server（接收端）的四次挥手
bool CloseConnectWithClient(SOCKET serverSocket, SOCKADDR_IN clientAddr)
{
    int AddrLen = sizeof(clientAddr);
    Message buffer1;
    Message buffer2;
    Message buffer3;
    Message buffer4;
    while (1)
    {
        //==============接收第一次挥手的消息（FIN=1，seq=y）============
        int recvByte = recvfrom(serverSocket, (char*)&buffer1, sizeof(buffer1),
0, (sockaddr*)&clientAddr, &AddrLen);
        if (recvByte == 0)
        {
            cout << "连接失败.....关闭连接！" << endl;
            return false;
        }

        else if (recvByte > 0)
        {
            //判断SYN、检验和
            if (!(buffer1.flag && FIN) || !buffer1.check() || !(buffer1.SeqNum
== global_seq + 1))
            {
                cout << "连接发生错误！" << endl;
                return false;
            }
            global_seq++;
            cout << "server已收到第一次挥手的消息！" << endl;

            //==============发送第二次挥手的消息（ACK=1，ack=y）============
            buffer2.SrcPort = ServerPORT;
            buffer2.DestPort = RouterPORT;
            buffer2.AckNum = buffer1.SeqNum;
            buffer2.flag += ACK;
            buffer2.setCheck();//设置校验和
            int sendByte = sendto(serverSocket, (char*)&buffer2,
sizeof(buffer2), 0, (sockaddr*)&clientAddr, AddrLen);
            clock_t buffer2start = clock();
            if (sendByte == 0)
            {
                cout << "连接失败.....关闭连接！" << endl;
                return false;
            }
            cout << "server已发送第二次挥手的消息！" << endl;
            break;
        }
    }

    //==============发送第三次挥手的消息（FIN=1，ACK=1，seq=z）============
    buffer3.SrcPort = ServerPORT;
    buffer3.DestPort = RouterPORT;
    buffer3.flag += FIN;//设置FIN
    buffer3.flag += ACK;//设置ACK
```

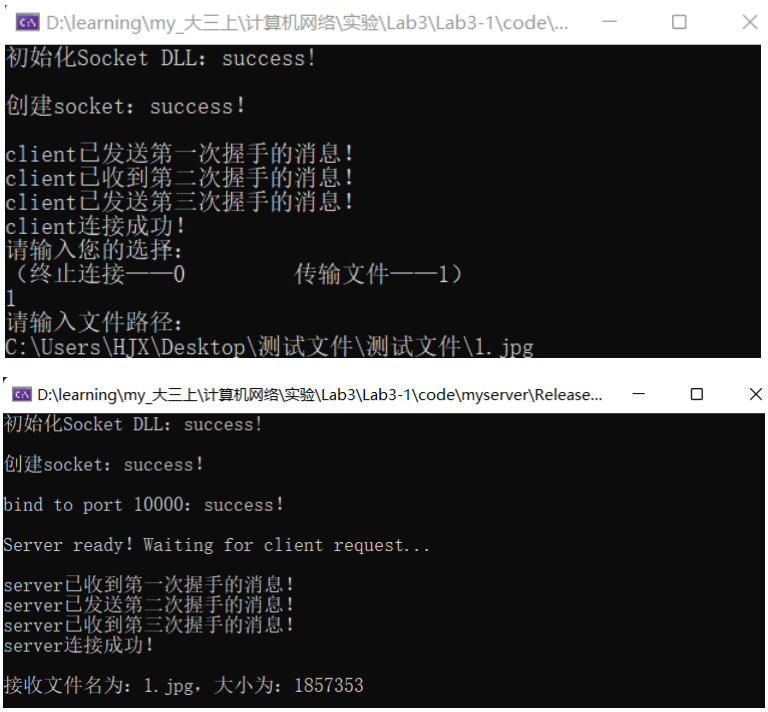
```
    buffer3.SeqNum = global_seq++;//设置序号seq
    buffer3.setCheck();//设置校验和
    int sendByte = sendto(serverSocket, (char*)&buffer3, sizeof(buffer3), 0,
(sockaddr*)&clientAddr, AddrLen);
    clock_t buffer3start = clock();
    if (sendByte == 0)
    {
        cout << "连接失败......关闭连接！" << endl;
        return false;
    }
    cout << "server已发送第三次挥手的消息！" << endl;

    //==============接收第四次挥手的消息（ACK=1，ack=z）============
    while (1)
    {
        int recvByte = recvfrom(serverSocket, (char*)&buffer4, sizeof(buffer4),
0, (sockaddr*)&clientAddr, &AddrLen);
        if (recvByte == 0)
        {
            cout << "关闭连接error！" << endl;
            return false;
        }
        else if (recvByte > 0)
        {
            //成功收到消息，检查校验和、ACK、ack
            if ((buffer4.flag && ACK) && buffer4.check() && (buffer4.AckNum ==
buffer3.SeqNum))
            {
                cout << "server已收到第四次挥手的消息！" << endl;
                break;
            }
            else
            {
                cout << "连接发生错误！" << endl;
                return false;
            }
        }
        //buffer3超时，重新发送并重新计时
        if (clock() - buffer3start > MAX_WAIT_TIME)
        {
            cout << "第三次挥手超时，正在重传......" << endl;
            //============重传buffer3==============
            int sendByte = sendto(serverSocket, (char*)&buffer3,
sizeof(buffer3), 0, (sockaddr*)&clientAddr, AddrLen);
            buffer3start = clock();
            if (sendByte == 0)
            {
                cout << "关闭连接error！" << endl;
                return false;
            }
        }
    }
    cout << "\n关闭连接成功！" << endl;
    return true;
}
```

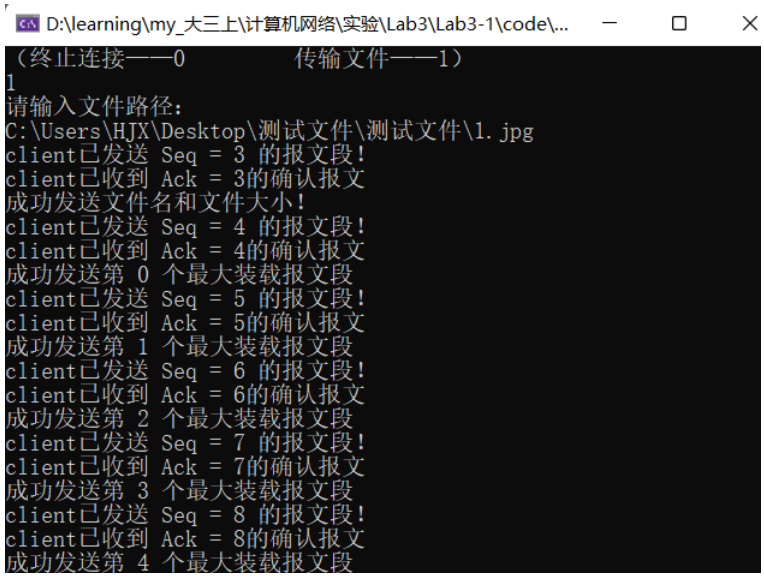
三、程序界面与运行

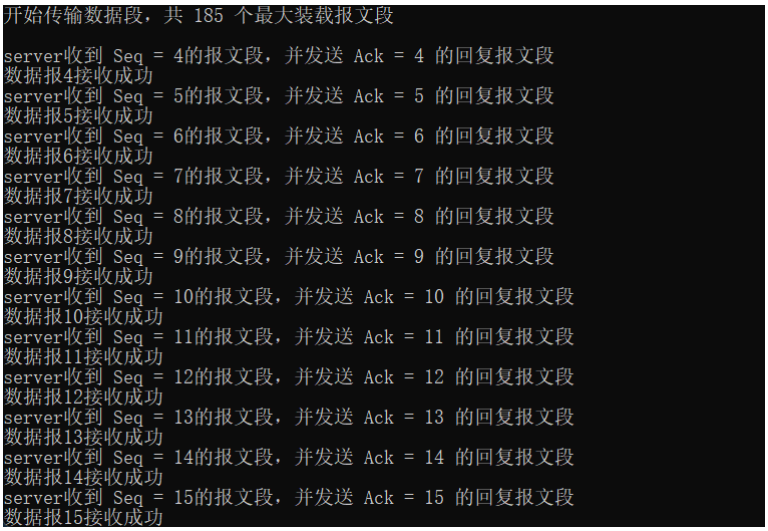
- 设置路由器如下：
  - 3%的丢包率和1ms延时

- **建立连接**



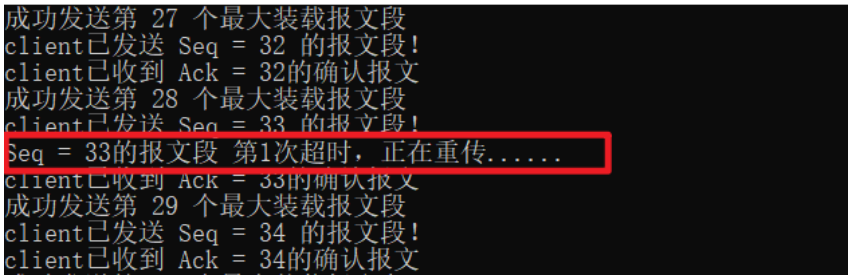
- **传输数据**

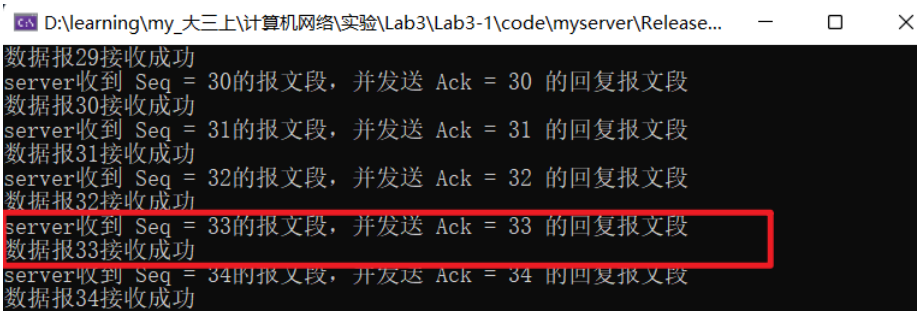




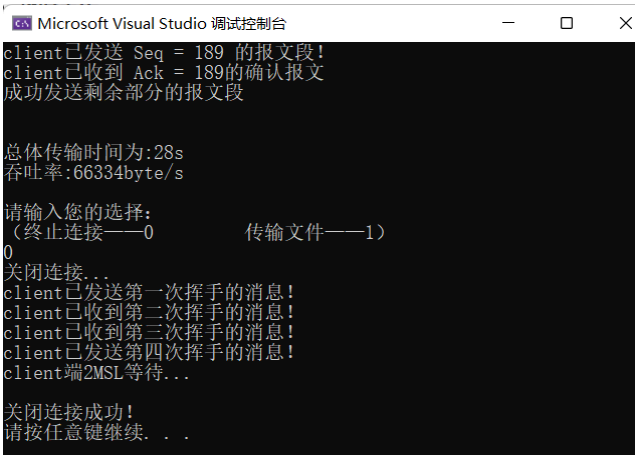
- **超时重传**

当数据包丢失，报文段超时，则进行重传：

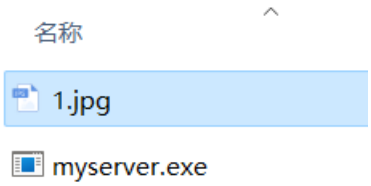




- **关闭连接**



- 最终在sever端程序的目录下，成功得到了传输的文件，且文件大小和内容完全一致。



四、实验过程遇到的问题及分析

1. recvfrom的阻塞问题：

需要在初始化socket的时候将其设置为非阻塞，这样就可以在while循环里等候消息的同时判断是否延时，而非一直被阻塞到接收消息的地方。

```
SOCKET serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
unsigned long on = 1;
ioctlsocket(serverSocket, FIONBIO, &on);//设置非阻塞
```

2. 文件路径问题：

在客户端传递文件时，使用绝对路径，可以传输本机上任意的文件。

传输文件名时，程序只会传输完整的文件名，例如（xxx.txt），而无需传递路径。

在服务器端，将接收到的文件保存时，默认保存到与程序同路径的目录下。

3. 如何检验client和server的seq和ack是否一致：

在两端都设置一个全局变量global_seq，都初始化为0。每当client发送数据包，global_seq++，当server接收数据包，global_seq也++，并判断二者是否相等。