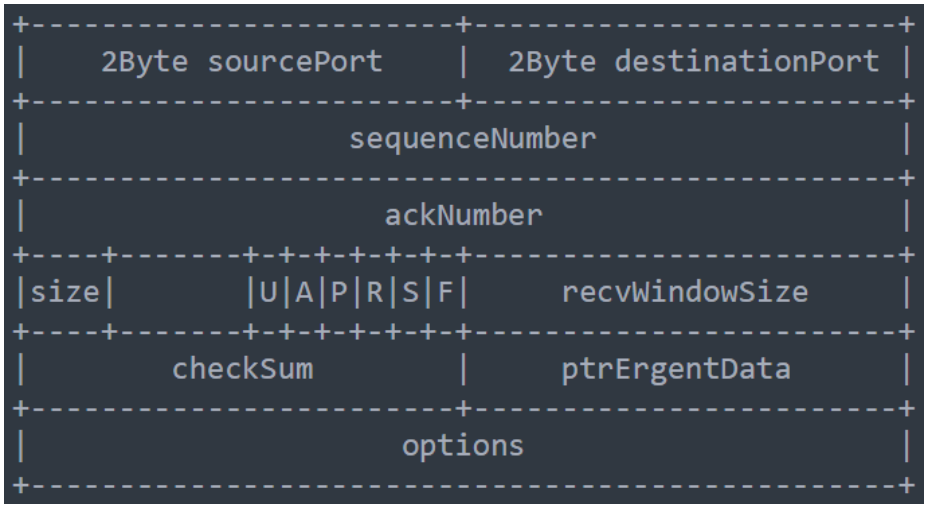
计算机网络实验3-1实验报告

> 1911590周安琪

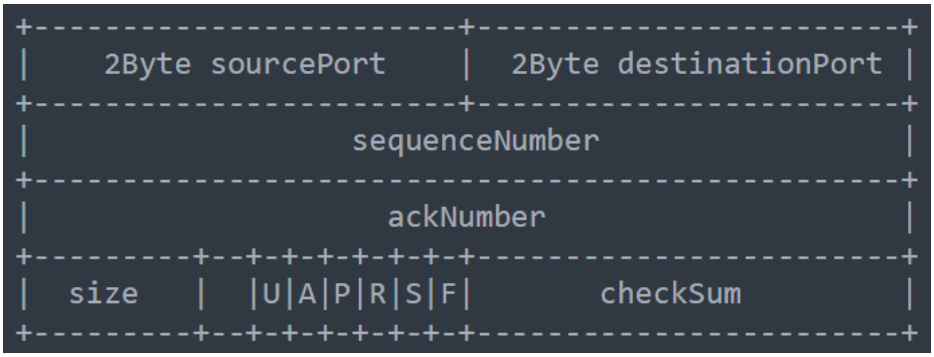
1 协议内容

我参考了TCP的报文头格式设计了我的报文头。

TCP格式如下，除去非定长的options共20Byte：



由于本次实验中用不上滑动窗口大小和紧急数据指针、options，所以我把这几项删去，并且把size扩充到8bit，用来存储文件名的长度，以便在传文件的第一个包里传文件名。



这里需要提到的一点是，为了方便debug（在计算校验和的时候输出遍历的某字节数值），在涉及数字的地方，包括Port, sequenceNumber, ackNumber, size, checkSum, 我是反着存储的。

举个例子，sourcePort为1234的时候，它的二进制表示是100 1101 0010。从我给出的头的结构来看，setSoucePort()的代码应该是这样的：

```
sendBuffer[0]=(1234>>8)&0xff;
sendBuffer[1]=1234&0xff;
```

但事实上我是这样写的：

```
sendBuffer[0]=1234&0xff;
sendBuffer[1]=(1234>>8)&0xff;
```

这是因为我在计算校验和的时候，是使用了unsigned short来从 `sendBuffer[0]` 开始遍历的，而unsigned short本身是两字节大小，它读入的方式似乎是认为 `sendBuffer[1]` 是高位而 `sendBuffer[0]` 为低位，所以输出的内容很奇怪，这令我困惑了很久。

不过这并不重要，因为在set和get接口中已经隐藏了这一点。

1.1 服务端

当服务端被运行时，它将开始监听，一旦接收到SYN报文则发送一个SYN+ACK报文；如果报文损坏，返回空报文。

```
recvBuffer: SeqNum: 0, AckNum: 0, Size: 0, SYN: 1, ACK: 0, FIN: 0, CheckSum: 62554
Got an SYN datagram!
sendBuffer: SeqNum: 0, AckNum: 0, Size: 0, SYN: 1, ACK: 1, FIN: 0, CheckSum: 58458
Sent SYN ACK.
```

接着，将接收到普通文件报文。服务端接收报文，比对报文序列号是否是想要的序列号，如果是，则将expectedNum加一，返回ACK+expectedNum；如果不是或者损坏，那么不改变expectedNum，将上次的ACK报文再发一遍。

```
received.
recvBuffer: SeqNum: 0, AckNum: 0, Size: 5, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63061
Got an File datagram!
sendBuffer: SeqNum: 0, AckNum: 1, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58969
sent.
received.
recvBuffer: SeqNum: 1, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63065
Got an File datagram!
sendBuffer: SeqNum: 0, AckNum: 2, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58968
sent.
```

如果接收到的报文的FIN位置1，则将自己的FIN位也置一，发送回去。

```
received.
recvBuffer: SeqNum: 29, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 1, CheckSum: 62781
Got an File datagram!
我也挥手了
sendBuffer: SeqNum: 0, AckNum: 30, Size: 0, SYN: 0, ACK: 1, FIN: 1, CheckSum: 58684
sent.
```

1.2 客户端

当客户端被运行时，它将向服务器端发送SYN报文，直到接收到服务器的ACK才跳出循环，开始发送文件。

```
sendBuffer: SeqNum: 0, AckNum: 0, Size: 0, SYN: 1, ACK: 0, FIN: 0, CheckSum: 62554
sent.
recvBuffer: SeqNum: 0, AckNum: 0, Size: 0, SYN: 1, ACK: 1, FIN: 0, CheckSum: 58458
Got a SYN ACK!
Tell me which file you want to send.
1.jpg
The size of this file is 1857353 bytes.
We will split this file to 30 packages and send it.
```

发送文件时，客户端发送报文之后，接收服务端的ACK，如果对方的报文是完好的，则将自己的序列号设置为对方的ACKNum；如果对方报文是损坏的，仍然发送上一次的报文。

```
sendBuffer: SeqNum: 0, AckNum: 0, Size: 5, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63061
sent.
recvBuffer: SeqNum: 0, AckNum: 1, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58969
received.
sendBuffer: SeqNum: 1, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63065
sent.
recvBuffer: SeqNum: 0, AckNum: 2, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58968
received.
sendBuffer: SeqNum: 2, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63064
sent.
recvBuffer: SeqNum: 0, AckNum: 3, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58967
received.
```

在最后一个包发送出去的时候，给FIN置位，表示发送文件停止；接收到对方的FIN后跳出循环。

```
sendBuffer: SeqNum: 29, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 1, CheckSum: 62781
sent.
recvBuffer: SeqNum: 0, AckNum: 30, Size: 0, SYN: 0, ACK: 1, FIN: 1, CheckSum: 58684
received.
```

2 开发环境

- windows10
- g++ -std=c++11

将server.cpp和client.cpp放在两个文件夹下（这是为了方便传文件，避免同名覆盖），在两个目录各运行下列指令（windows命令行）可以从cpp文件编译出exe文件：

```
g++ -std=c++11 client.cpp -o client.exe -lws2_32
g++ -std=c++11 server.cpp -o server.exe -lws2_32
```

随后使用以下指令可以分别运行两个程序：

```
client.exe
server.exe
```

3 代码解释

3.1 client.cpp

在main函数中，先是设置了套接字。

3.1.1 发送SYN报文

随后开始发送SYN报文。这里用到的`packSynDatagram(0)`函数是为了处理SYN报文的头信息，意思是除了SYN和SeqNum之外头都置零。传的参数是序列号之所以要传参是因为SYN报文有协商起始序列号的作用。`printLogSendBuffer()`函数是为了输出日志，在后文详细展示。

这个循环跳出的唯一条件是接收到了来自服务端的SYN+ACK报文（不可以是损坏的）。这里用到的`checkSumIsRight()`函数作用是计算校验和是否正确，即判断信息是否被损坏。

```
while (1){
    packSynDatagram(0);
    sendto(sockSrv, sendBuffer, sizeof(sendBuffer),
           0, (sockaddr*)&addrServer, len);
    printLogSendBuffer();//输出sendBuffer的头信息
    cout << "sent." << endl;

    recvfrom(sockSrv, recvBuffer, sizeof(recvBuffer),
             0, (SOCKADDR*)&addrServer, &len);
    printLogRecvBuffer();//输出recvBuffer的头信息

    // 不断发送SYN，直到收到SYN+ACK为止。
    if(getter.getAckBit(recvBuffer) && getter.getSynBit(recvBuffer)
       && checkSumIsRight()){
        cout<<"Got a SYN ACK!"<<endl;
        break;
    }
}
```

3.1.2 发送文件

然后开始发送文件。

首先获取文件名、计算文件的长度和需要发送的次数。

```
getFileName();

// 读入文件，计算文件长度
ifstream fin(fileName, ifstream::in | ios::binary);
fin.seekg(0, fin.end);                  //把指针定位到末尾
fileLength = fin.tellg();               //获取指针
fin.seekg(0, fin.beg);                  //将文件流指针重新定位到流的开始

if (fileLength ≤ 0) {
    printFileErr();
    return 0;
}
cout≪"The size of this file is "≪fileLength≪" bytes.\n";

int sendTimes = ceil( fileLength / DATA_SIZE );     //需要发送这么多次
cout ≪ "We will split this file to " ≪ sendTimes ≪ " packages and
send it." ≪ endl;
```

然后进入发送的循环。

如果是第一个包，那么需要在数据的头部写入文件名称，在头部的size中填入文件名的长度以便解包。

```
if(isFirstPackage){
    packFirst();//加上Size

    // 数据段的前length个位置放文件名
    for(int j=0;j<fileName.length();j++){
        sendBuffer[HEAD_SIZE+j]=fileName[j];
    }

    isFirstPackage=false;
}
else{
    packData();//没有Size的普通头，内含序列号。
}
```

如果是最后一个包，那么将Fin位置位并且重算校验和。我在这里使用了一个独立的nowTime来计算当前的包是第几个包，这个数字仅仅在受到来自服务器端的ACK+ACKNum且通过校验的时候才会自增1。

（起初我是直接用的sequenceNum来计数，但是不知为什么我虽然给序列号设置了16位的空间，但是它还是在超过八位即到达128的时候翻转，很奇怪。所以我干脆把序列号的最大值设置成127了。）

```
if(nowTime==sendTimes-1){
    setFinBit(1);//这里需要重算校验和
    setCheckSum();
}
```

开始读数据（如果是第一个文件的话从文件名后开始读数据）：

```
fin.seekg(bytesHaveRead,fin.beg);
int sendSize = min(leftDataSize, fileLength-bytesHaveRead); //如果是最后一个包的话可能会不满。
fin.read(&sendBuffer[HEAD_SIZE + (DATA_SIZE - leftDataSize)], sendSize);// sendBuffer从什么地方开始读起，读多少
bytesHaveRead += sendSize;
bytesHaveWritten += sendSize;
```

发送并接收回复。

```
sendto(sockSrv, sendBuffer, sizeof(sendBuffer), 0, (sockaddr*)&addrServer, len);
printLogSendBuffer();
cout<<"sent."<<endl;
memset(sendBuffer, 0, sizeof(sendBuffer));

recvfrom(sockSrv, recvBuffer, sizeof(recvBuffer), 0, (SOCKADDR*)&addrServer, &len);
printLogRecvBuffer();
cout<<"received."<<endl;
```

接收的回复如果没通过校验，说明已经被损坏了，则把当前的包再发一遍。

```
if(!checkSumIsRight()){
    bytesHaveRead-=sendSize;
    bytesHaveWritten -= sendSize;
    continue;
}
```

如果校验和是符合的，而且是Ack包，那么将序列号设置为对方的期望序列号。如果对方希望停止传输（FIN==1）那么跳出发送文件的循环。

```
if(getter.getAckBit(recvBuffer)){
    if(sequenceNumber+1==getter.getAckNum(recvBuffer)
       ||sequenceNumber+1+getter.getAckNum(recvBuffer)==128){
        nowTime++;
        cout<<"nowTime: "<<nowTime<<endl;
    }
    sequenceNumber=getter.getAckNum(recvBuffer);

    if(getter.getFinBit(recvBuffer)){
        break;
    }
    continue;
}
```

代码逻辑结束，计算一些信息输出，包括一共传输的字节数、使用的时间总数和吞吐率。这只在客户端输出，不在服务端输出。

```
cout<<"Sent "<<bytesHaveSent<<" bytes, ";
cout<<"Cost time: "<< t_end - t_start << "(ms), ";
// numOfBit/time
cout <<"Throughput rate: " << bytesHaveSent * 8 / (t_end - t_start) *
CLOCKS_PER_SEC << " bps" << endl;
```

3.2 server.cpp

在main函数中，先是设置了套接字。

随后开始监听。

在进入循环之前，先设置一个期望序列号。

```
int expectedNum=0;
```

接收到一个报文之后，首先判断是SYN报文还是普通的数据报文。

如果是SYN报文，判断包是否损坏。如果没有损坏，则获取对方的起始序列号，设置为自己的expectedNum。

```
if(getter.getSynBit(recvBuffer)){
    cout<<"Got an SYN datagram!"<<endl;

    //如果校验和没问题就返回一个SYN ACK报文，否则返回空报文
    if(checkSumIsRight()){
        // SYN报文协商起始的序列号。
        expectedNum=getter.getSeqNum(recvBuffer);
        setAckNum(getter.getSeqNum(recvBuffer));
        packSynAckDatagram();
```

```
            sendto(sockSrv, sendBuffer, sizeof(sendBuffer),
                   0, (sockaddr*)&addrClient, len);
            printLogSendBuffer();
            cout<<"Sent SYN ACK."<<endl;
        }
    }
```

如果校验和出现问题，则返回一个空的报文。

```
else{
    packEmptyDatagram();
    sendto(sockSrv, sendBuffer, sizeof(sendBuffer), 0,
           (sockaddr*)&addrClient, len);
    printLogSendBuffer();
    cout<<"sent."<<endl;
}
```

如果是普通的数据报文，判断是否是损坏的报文。

如果不是损坏的，判断是不是第一个数据报文，如果是的话需要读入文件名并且把输出流和该文件名绑定。

```
if(checkSumIsRight()){
    // 如果是第一个，那么头部的size段不为0
    int fileNameLength=0;
    if(getter.getSize(recvBuffer)≠0){
        fileName="";
        fileNameLength=getter.getSize(recvBuffer);

        for(int i=0;i<getter.getSize(recvBuffer);i++){
            fileName+=recvBuffer[HEAD_SIZE+i];
        }
        cout<<"fileName: "<<fileName<<endl;
        fout.open(fileName,ios_base::out | ios_base::app
                  | ios_base::binary);
    }
    //......
}
```

判断发过来的报文的序列号是不是期望的序列号（也包括了第一个报文）。

如果是，expectedNum加一，如果不是，不改变expectedNum，再次返回上次的ACK报文。同时把收到的信息写入文件。

如果收到的报文中FIN被置位了说明是最后一个报文，将自己的报文FIN也置位，并且关闭输出流。

```
if(checkSumIsRight()){
```

```
        // 如果是第一个
        // ...

        // 如果发过来的序列号正是想要的，期望序列号++，否则再发一遍之前的。
        if(expectedNum==getter.getSeqNum(recvBuffer)){
            expectedNum++;
            if(expectedNum>=128){
                expectedNum%=128;
            }
            int len = 0;

            fout.write(&recvBuffer[HEAD_SIZE + fileNameLength],
                       DATA_SIZE);
        }
        else{
            cout<<"Not expectedNum!"<<endl;
        }

        packAckDatagram(expectedNum);

        if(getter.getFinBit(recvBuffer)){//如果收到了fin，挥手。
            setFinBit(1);
            setCheckSum();
            fout.close();
            cout<<"file receiving ends."<<endl;
        }

        sendto(sockSrv, sendBuffer, sizeof(sendBuffer), 0,
               (sockaddr*)&addrClient, len);
        printLogSendBuffer();
        cout<<"sent."<<endl;
    }
```

如果报文是损坏的，再次返回上次的ACK报文。

3.3 握手的实现

在上文中已经提到过，在一开始的时候客户端会先给服务端发送一个SYN报文，随后服务端会给服务端发送SYN+ACK报文表明已经接收到了。

客户端：

```
    // 发送SYN报文
    while (1){
        packSynDatagram(0);
        sendto(sockSrv, sendBuffer, sizeof(sendBuffer), 0,
               (sockaddr*)&addrServer, len);
        printLogSendBuffer();
```

```
        cout << "sent." << endl;

        recvfrom(sockSrv, recvBuffer, sizeof(recvBuffer), 0,
                 (SOCKADDR*)&addrServer, &len);
        printLogRecvBuffer();

        // 不断发送SYN，直到收到SYN+ACK为止。
        if(getter.getAckBit(recvBuffer) &&
           getter.getSynBit(recvBuffer) && checkSumIsRight()){
            cout<<"Got a SYN ACK!"<<endl;
            break;
        }
    }
    // 发送SYN报文
```

服务端：

```
    // 如果是SYN报文
    if(getter.getSynBit(recvBuffer)){
        cout<<"Got an SYN datagram!"<<endl;

        //如果校验和没问题就返回一个SYN ACK报文，否则返回空报文
        if(checkSumIsRight()){
            // SYN报文协商起始的序列号。
            expectedNum=getter.getSeqNum(recvBuffer);
            setAckNum(getter.getSeqNum(recvBuffer));
            packSynAckDatagram();
            sendto(sockSrv, sendBuffer, sizeof(sendBuffer), 0,
                   (sockaddr*)&addrClient, len);
            printLogSendBuffer();
            cout<<"Sent SYN ACK."<<endl;
        }
        else{
            packEmptyDatagram();
            sendto(sockSrv, sendBuffer, sizeof(sendBuffer), 0,
                   (sockaddr*)&addrClient, len);
            printLogSendBuffer();
            cout<<"sent."<<endl;
        }
    }
```

3.4 挥手的实现

当当前发送的已经是最后一个数据包的时候，客户端将会把自己的FIN置位，服务端收到FIN置位之后会把自己的FIN也置位，当客户端收到FIN置位的报文时会跳出循环结束传输。

客户端发送FIN：

```
if(nowTime==sendTimes-1){
    setFinBit(1);
    setCheckSum();
}
```

服务端接收处理FIN：

```
if(getter.getFinBit(recvBuffer)){//如果收到了fin，挥手。
    setFinBit(1);
    setCheckSum();
    fout.close();
    cout<<"file receiving ends."<<endl;
}
```

客户端接受服务端的FIN：

```
// 没有损坏
if(getter.getAckBit(recvBuffer)){
    if(sequenceNumber+1==getter.getAckNum(recvBuffer)||
       sequenceNumber+1+getter.getAckNum(recvBuffer)==128){
        nowTime++;
        cout<<"nowTime: "<<nowTime<<endl;
    }
    sequenceNumber=getter.getAckNum(recvBuffer);

    if(getter.getFinBit(recvBuffer)){
        break;
    }
    continue;
}
```

3.5 日志的输出

日志的输出用两个函数来实现，一个输出sendBuffer的信息，一个输出recvBuffer的信息。

```
void printLogSendBuffer(){
    cout<<"sendBuffer: ";
    cout<<"SeqNum: "<<getter.getSeqNum(sendBuffer)<<", ";
    cout<<"AckNum: "<<getter.getAckNum(sendBuffer)<<", ";
    cout<<"Size: "<<getter.getSize(sendBuffer)<<", ";
    cout<<"SYN: "<<getter.getSynBit(sendBuffer)<<", ";
    cout<<"ACK: "<<getter.getAckBit(sendBuffer)<<", ";
    cout<<"FIN: "<<getter.getFinBit(sendBuffer)<<", ";
```

```
    cout<<"CheckSum: "<<getter.getCheckSum(sendBuffer)<<endl;
}

void printLogRecvBuffer(){
    cout<<"recvBuffer: ";
    cout<<"SeqNum: "<<getter.getSeqNum(recvBuffer)<<", ";
    cout<<"AckNum: "<<getter.getAckNum(recvBuffer)<<", ";
    cout<<"Size: "<<getter.getSize(recvBuffer)<<", ";
    cout<<"SYN: "<<getter.getSynBit(recvBuffer)<<", ";
    cout<<"ACK: "<<getter.getAckBit(recvBuffer)<<", ";
    cout<<"FIN: "<<getter.getFinBit(recvBuffer)<<", ";
    cout<<"CheckSum: "<<getter.getCheckSum(recvBuffer)<<endl;
}
```

3.6 统计数据的输出

首先统计了发送报文的字节数，其次计算了发送文件所用的时间数，然后利用两者计算了吞吐率。只在客户端输出。

```
int t_start=clock();
int bytesHaveSent=0;//这在源代码中是全局变量，在此处为了方便展示放在这里。
while(1){
    // ...
}
int t_end=clock();

cout<<"Sent "<<bytesHaveSent<<" bytes, ";
cout<<"Cost time: "<< t_end - t_start << "(ms), ";
// numOfBit/time
cout <<"Throughput rate: " << bytesHaveSent * 8 / (t_end - t_start)
* CLOCKS_PER_SEC << " bps" << endl;
```

输出结果：

```
Sent 5898510 bytes, Cost time: 2409(ms), Throughput rate: 195880
00 bps
```

4 log输出结果

以2.jpg的传输为例：

4.1 客户端

握手：

```
C:\Users\16834\Desktop\client>client.exe
sendBuffer: SeqNum: 0, AckNum: 0, Size: 0, SYN: 1, ACK: 0, FIN: 0, CheckSum: 62554
sent.
recvBuffer: SeqNum: 0, AckNum: 0, Size: 0, SYN: 1, ACK: 1, FIN: 0, CheckSum: 58458
Got a SYN ACK!
```

获取文件：

```
Tell me which file you want to send.
2.jpg
The size of this file is 5898505 bytes.
We will split this file to 97 packages and send it.
```

文件传输：

```
bytesHaveSent: 61424
sendBuffer: SeqNum: 0, AckNum: 0, Size: 5, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63061
sent.
recvBuffer: SeqNum: 0, AckNum: 1, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58969
received.
nowTime: 1
bytesHaveSent: 122848
sendBuffer: SeqNum: 1, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63065
sent.
recvBuffer: SeqNum: 0, AckNum: 2, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58968
received.
nowTime: 2
bytesHaveSent: 184272
sendBuffer: SeqNum: 2, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63064
sent.
recvBuffer: SeqNum: 0, AckNum: 3, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58967
received.
nowTime: 3
```

挥手：

```
bytesHaveSent: 5898510
sendBuffer: SeqNum: 96, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 1, CheckSum: 62458
sent.
recvBuffer: SeqNum: 0, AckNum: 97, Size: 0, SYN: 0, ACK: 1, FIN: 1, CheckSum: 58361
received.
nowTime: 97
Sent 5898510 bytes, Cost time: 2266(ms), Throughput rate: 20824000 bps
```

4.2 服务端

握手：

```
recvBuffer: SeqNum: 0, AckNum: 0, Size: 0, SYN: 1, ACK: 0, FIN: 0, CheckSum: 62554
Got an SYN datagram!
sendBuffer: SeqNum: 0, AckNum: 0, Size: 0, SYN: 1, ACK: 1, FIN: 0, CheckSum: 58458
Sent SYN ACK.
```

文件传输（注意此处第一次收到的时候会解析出文件名并且和fout绑定）：

```
recvBuffer: SeqNum: 0, AckNum: 0, Size: 5, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63061
Got an File datagram!
fileName: 2.jpg
sendBuffer: SeqNum: 0, AckNum: 1, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58969
sent.
received.
recvBuffer: SeqNum: 1, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63065
Got an File datagram!
sendBuffer: SeqNum: 0, AckNum: 2, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58968
sent.
received.
recvBuffer: SeqNum: 2, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63064
Got an File datagram!
sendBuffer: SeqNum: 0, AckNum: 3, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58967
sent.
received.
recvBuffer: SeqNum: 3, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 0, CheckSum: 63063
Got an File datagram!
```

挥手：

```
sent.
received.
recvBuffer: SeqNum: 95, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 0, CheckSum: 62715
Got an File datagram!
sendBuffer: SeqNum: 0, AckNum: 96, Size: 0, SYN: 0, ACK: 1, FIN: 0, CheckSum: 58618
sent.
received.
recvBuffer: SeqNum: 96, AckNum: 0, Size: 0, SYN: 0, ACK: 0, FIN: 1, CheckSum: 62458
Got an File datagram!
file receiving ends.
sendBuffer: SeqNum: 0, AckNum: 97, Size: 0, SYN: 0, ACK: 1, FIN: 1, CheckSum: 58361
sent.
```

5 代码的不足之处

- 没有实现超时重传

6 一些发现

一件很奇怪的事情是，我一开始把BUFFER_SIZE设置成0xffff，这导致我在长时间内都无法将客户端和服务端连接，但是当我改成0xf000的时候，它们很快就连上了。我猜测这是套接字本身有长度限制。