注意：可以直接运行exe文件（先receive.exe再send.exe）。如果要编译运行需要在g++编译时加入参数`-lws2_32`

整个程序大致流程

1. 建立socket，建立send端与receive端的连接
2. 双方执行`handshake()`函数进行握手
3. 开始收发文件，send端执行`sendfile(char* filename)`，receive端执行`recvfile()`，receive端收到每个packet后需要检查校验和
4. 收发文件完毕后，send端输入`q`告知程序，双方进入`wavehand()`函数
5. send端现实传输时间、传输数据大小和速率，双方程序结束

header协议设计

在send.cpp和receive.cpp中开一个全局数组，大小为14byte，每一位的作用如下：

```
// 伪首部14 byte，约定：
// 0 1 2 3--32位seq（0--低8位，3--高8位，下同）
// 4 5 6 7--32位ack
// 8--标志位，低三位分别代表ACK SYN FIN，第四位、第五位代表此次发送的是文件名、文件大小
// 9--空着，全0
// 10 11--数据部分长度
// 12 13--校验和
char header[HEADERSIZE] = {0};
```

每次发送packet需要修改header中的信息时修改该全局数组，再将其加入sendBuf即可。

checkSum()函数设计

接受input（也即要发过去的packet）和它的长度length。先用`u_short`类型把input转换成16位1组，然后按16位求和，最后取反即可。

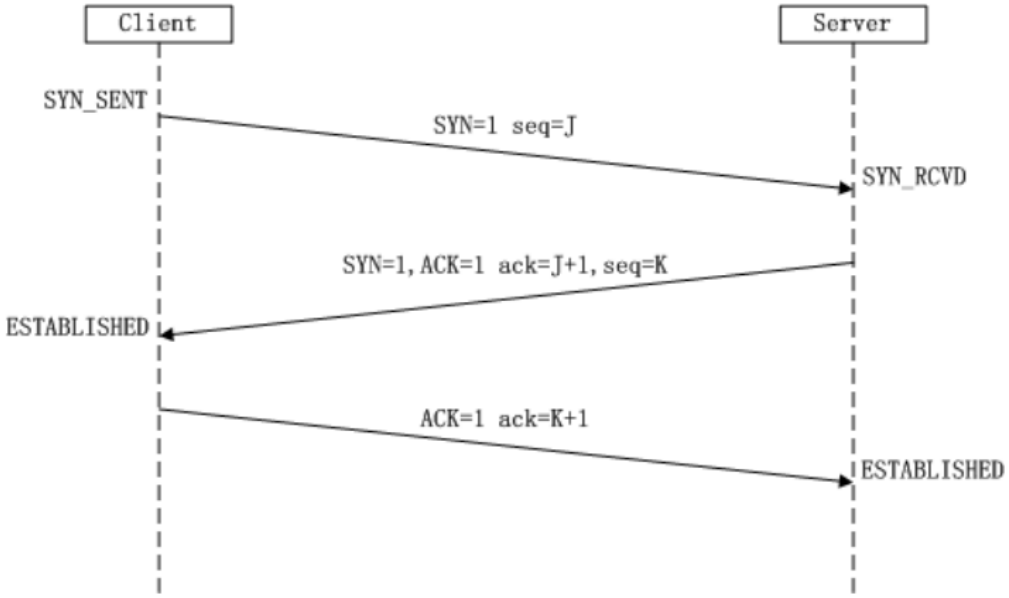
```
u_short checkSum(const char* input, int length) {
    int count = (length + 1) / 2; // 有多少组16 bits
    u_short* buf = new u_short[count]{0};
    for (int i = 0; i < count; i++) {
        buf[i] = (u_char)input[2 * i] + ((2 * i + 1 < length) ? (u_char)input[2
* i + 1] << 8 : 0);
        // 最后这个三元表达式是为了避免在计算buf最后一位时，出现input[length]的越界情况
    }

    register u_long sum = 0;
    while (count--) {
        sum += *buf++;
        // 如果sum有进位，则进位回滚
        if (sum & 0xFFFF0000) {
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
```

```
}
```

在receive端检查时，由于input中又加入了checkSum()的结果，所以求和的结果是16位1，但是又因为这个函数最后有一个取反，所以得到的结果应是0.

handShake()函数

1. 第一次握手：客户端发送报文：SYN = 1, seq = x，进入STN_SENT状态
2. 第二次握手：服务器端发送报文：ACK = 1, SYN = 1, seq = y, ack = x + 1，进入SYN_RCVD状态
3. 第三次握手：客户端发送报文：ACK = 1, seq = x + 1, ack = y + 1，进入ESTABLISHED状态，服务器端收到后，也进入ESTABLISHED状态



按照lab2中三次握手的过程依次设置send端和receive端的header中不同位即可。

send端的`handshake()`：

```
bool handshake() {
    u_short checksum = 0;

    // 发送第一次握手请求报文
    memset(header, 0, HEADERSIZE);
    // 设置seq位
    int seq = rand();
    header[SEQ_BITS_START] = (u_char)(seq & 0xFF);
    header[SEQ_BITS_START + 1] = (u_char)(seq >> 8);
    header[SEQ_BITS_START + 2] = (u_char)(seq >> 16);
    header[SEQ_BITS_START + 3] = (u_char)(seq >> 24);
    // 设置SYN位
    header[FLAG_BIT_POSITION] = 0b010; // SYN在header[FLAG_BIT_POSITION]的第二个
bit，所以这一行表示SYN == 1
    checksum = checkSum(header, HEADERSIZE);
    // 设置checksum位
    header[CHECKSUM_BITS_START] = (u_char)(checksum & 0xFF);
    header[CHECKSUM_BITS_START + 1] = (u_char)(checksum >> 8);
    sendto(sendSocket, header, HEADERSIZE, 0, (SOCKADDR*)&recvAddr,
sizeof(SOCKADDR));
    cout << "send the First Handshake message!" << endl;
```

```
    // 接受第二次握手应答报文
    char recvBuf[HEADERSIZE] = {0};
    int recvResult = 0;
    while(true) {
        recvResult = recvfrom(sendSocket, recvBuf, HEADERSIZE, 0,
(SOCKADDR*)&recvAddr, &len);
        // 接受ack
        int ack = recvBuf[ACK_BITS_START] + (recvBuf[ACK_BITS_START + 1] << 8)
                + (recvBuf[ACK_BITS_START + 2] << 16) + (recvBuf[ACK_BITS_START
+ 3] << 24);
        if ((ack == seq + 1) && (recvBuf[FLAG_BIT_POSITION] == 0b110)) { //
0b110代表ACK SYN FIN == 110
            cout << "successfully received the Second Handshake message!" <<
endl;
            break;
        } else {
            cout << "failed to received the correct Second Handshake message,
Handshake failed!" << endl;
            return false;
        }
    }

    // 发送第三次握手请求报文
    // 省略了，和第一次差不多

    cout << "Handshake successfully!" << endl;
    return true;

}
```

sendfile(char* filename)函数

握手完毕后开始发文件内容。发送端的函数要执行的步骤：

1. 读入文件，获得文件大小，并分配一个对应大小的缓冲区

```
ifstream is(filename, ifstream::in | ios::binary);
is.seekg(0, is.end);
int fileSize = is.tellg();
is.seekg(0, is.beg);
char* filebuf;
filebuf = (char*)calloc(fileSize, sizeof(char));
is.read(filebuf, fileSize);
is.close();
```

2. 发送文件名和文件大小给receive端

```
// 发送文件名
memset(sendBuf, 0, PACKETSIZE);
header[FLAG_BIT_POSITION] = 0b1000; // 代表“此次发送的是文件名”的标志值
strcat((char*)memcpy(sendBuf, header, HEADERSIZE) + HEADERSIZE, filename);
// 看上去有点复杂，但反正就是把filename放到header后面，然后一起放进sendBuf中发过去
sendto(sendSocket, sendBuf, PACKETSIZE, 0, (SOCKADDR*)&recvAddr,
sizeof(SOCKADDR));

// 发送文件大小
memset(sendBuf, 0, PACKETSIZE);
header[FLAG_BIT_POSITION] = 0b10000; // 代表“此次发送的是文件大小”的标志值
strcat((char*)memcpy(sendBuf, header, HEADERSIZE) + HEADERSIZE,
to_string(fileSize).c_str());
sendto(sendSocket, sendBuf, PACKETSIZE, 0, (SOCKADDR*)&recvAddr,
sizeof(SOCKADDR));
```

3. 开始发送文件数据

   大致过程：

   - 对每个packet设置header中的seq, ack，并把filebuf中对应的数据段拷贝到packet中
   - 等待响应报文
   - 收到正确的响应报文后再发下一个packet（如果不正确就重传）。在本协议中，“正确”的判断方法是收到响应报文，且响应报文中的seq == 发送报文的seq
   - 重复，直到hasSent = filesize->发送完毕

```
int hasSent = 0; // 已发送的文件大小
int sendResult = 0; // 每次sendto函数的返回结果
int sendSize = 0; // 每次实际发送的报文总长度
int seq = 0, ack = 0; // 发送包时的seq, ack
int seq_opp = 0, ack_opp = 0; // 收到的对面的seq, ack
int dataLength = 0; // 每次实际发送的数据部分长度(= sendSize - HEADERSIZE)
u_short checksum = 0; // 校验和
bool resend = false; // 重传标志
char recvBuf[HEADERSIZE] = {0}; // 接受响应报文的缓冲区
int recvResult = 0; // 接受响应报文的返回值

// 发送文件
while(true) {
    // 初始化头部和数据段
    memset(header, 0, HEADERSIZE);
    memset(dataSegment, 0, DATASIZE);
    memset(sendBuf, 0, PACKETSIZE);
    // 设置本次发送长度
    sendSize = min(PACKETSIZE, fileSize - hasSent + HEADERSIZE);

    if (!resend) {
        // 如果不是重传，需要设置header
        // seq = 收到的包的ack，表示接下来要发的字节位置
        // ack = 收到的包的seq + 收到的data length（然而receive方的data length永
远为0）
        // 设置seq位
        seq = ack_opp;
        header[SEQ_BITS_START] = (u_char)(seq & 0xFF);
        header[SEQ_BITS_START + 1] = (u_char)(seq >> 8);
```

```
        header[SEQ_BITS_START + 2] = (u_char)(seq >> 16);
        header[SEQ_BITS_START + 3] = (u_char)(seq >> 24);
        // 设置ack位
        ack = seq_opp;
        header[ACK_BITS_START] = (u_char)(ack & 0xFF);
        header[ACK_BITS_START + 1] = (u_char)(ack >> 8);
        header[ACK_BITS_START + 2] = (u_char)(ack >> 16);
        header[ACK_BITS_START + 3] = (u_char)(ack >> 24);
        // 设置ACK位
        header[FLAG_BIT_POSITION] = 0b100;
        // 设置data length位
        dataLength = sendSize - HEADERSIZE;
        header[DATA_LENGTH_BITS_START] = dataLength & 0xFF;
        header[DATA_LENGTH_BITS_START + 1] = dataLength >> 8;

        // file中此次要被发送的数据->dataSegment
        memcpy(dataSegment, filebuf + hasSent, sendSize - HEADERSIZE);
        // header->sendBuf
        memcpy(sendBuf, header, HEADERSIZE);
        // dataSegment->sendBuf（从sendBuf[10]开始）
        memcpy(sendBuf + HEADERSIZE, dataSegment, sendSize - HEADERSIZE);
        // 设置checksum位
        checksum = checkSum(sendBuf, sendSize);
        header[CHECKSUM_BITS_START] = sendBuf[CHECKSUM_BITS_START] =
checksum & 0xFF;
        header[CHECKSUM_BITS_START + 1] = sendBuf[CHECKSUM_BITS_START + 1] =
checksum >> 8;

        sendResult = sendto(sendSocket, sendBuf, sendSize, 0,
(SOCKADDR*)&recvAddr, sizeof(SOCKADDR));
    } else {
        // 如果是重传，不需要设置header，再发一次即可
        sendResult = sendto(sendSocket, sendBuf, sendSize, 0,
(SOCKADDR*)&recvAddr, sizeof(SOCKADDR));
    }

    // 发完packet后接受响应报文
    while (true) {
        // TODO: 如果超时还没收到响应报文，则break并重传
        recvResult = recvfrom(sendSocket, recvBuf, HEADERSIZE, 0,
(SOCKADDR*)&recvAddr, &len);
        if (recvResult == SOCKET_ERROR) {
            cout << "receive error! sleep!" << endl;
            std::this_thread::sleep_for(std::chrono::milliseconds(2000));
            continue;
        }
        seq_opp = (u_char)recvBuf[SEQ_BITS_START] +
((u_char)recvBuf[SEQ_BITS_START + 1] << 8)
            + ((u_char)recvBuf[SEQ_BITS_START + 2] << 16) +
((u_char)recvBuf[SEQ_BITS_START + 3] << 24);
        ack_opp = (u_char)recvBuf[ACK_BITS_START] +
((u_char)recvBuf[ACK_BITS_START + 1] << 8)
            + ((u_char)recvBuf[ACK_BITS_START + 2] << 16) +
((u_char)recvBuf[ACK_BITS_START + 3] << 24);
```

```
        if (recvBuf[FLAG_BIT_POSITION] == 0b100 && seq == seq_opp) {
            // 因为接收方没有要发送的数据，所以响应报文里的seq按传统TCP/UDP协议来说一直为0，但在我的协议里把响应报文的seq置成发送报文的seq，方便确认
            // 对方正确收到了这个packet
            resend = false;
            hasSent += sendSize - HEADERSIZE;
            // cout << "send seq = " << seq << " packet successfully!" << endl;
            break;
        } else {
            resend = true;
            cout << "failed to send seq = " << seq << " packet! This packet will be resent." << endl;
            break;
        }
    }

    if (hasSent == fileSize) {
        cout << "send successfully, send " << fileSize << " bytes." << endl;
        totalLength += fileSize;
        break;
    }
}
```

recvfile()函数

receive端接受文件的函数。大致要做：

1. 接受文件名（用于在这边创建同名文件）和文件大小

```
// 接收文件名
recvResult = recvfrom(recvSocket, recvBuf, PACKETSIZE, 0, (SOCKADDR*)&sendAddr, &len);
// 提取文件名
if (recvBuf[FLAG_BIT_POSITION] == 0b1000) {
    memcpy(filename, recvBuf + HEADERSIZE, FILE_NAME_MAX_LENGTH);
}

// 接收文件大小
recvResult = recvfrom(recvSocket, recvBuf, PACKETSIZE, 0, (SOCKADDR*)&sendAddr, &len);
// 提取文件大小
if (recvBuf[FLAG_BIT_POSITION] == 0b10000) {
    filesize = atoi(recvBuf + HEADERSIZE);
}
cout << "begin to receive a file, filename: " << filename << ", filesize: " << filesize << " bytes." << endl;
```

2. 根据filename创建一个文件，用于写入收到的数据

```
ofstream out;
out.open(filename, ios::out | ios::binary | ios::app);
while (true) {
    // 接受数据...
    out.write(dataSegment, dataLength);
    // 发送响应报文...
}
out.close();
```

3. 接收数据+写入文件+给响应报文

   大致过程：

   - 检查checksum of packet 和 ACK bit
   - 如果正确，检查本地的ack是否与收到的包中的seq相等
   - 如果相等，说明正确接收了这个包，则：
     - 把对应部分的数据写入文件
     - 重新设置seq和ack（ack往后移收到的dataLength字节），发送响应报文
   - 如果不想等，则：
     - 再次发送上一次的ack响应报文

```
int hasReceived = 0; // 已接收字节数
int seq_opp = 0, ack_opp = 0; // 对方发送报文中的seq, ack
int seq = 0, ack = 0; // 要发送的响应报文中的seq, ack
int dataLength = 0; // 接收到的数据段长度(= recvResult - HEADERSIZE)
u_short checksum = 0; // 校验和（为0时正确）

while (true) {
    memset(recvBuf, 0, PACKETSIZE);
    memset(header, 0, HEADERSIZE);
    recvResult = recvfrom(recvSocket, recvBuf, PACKETSIZE, 0,
(SOCKADDR*)&sendAddr, &len);
    if (recvResult == SOCKET_ERROR) {
        cout << "receive error! sleep!" << endl;
        std::this_thread::sleep_for(std::chrono::milliseconds(2000));
        continue;
    }

    // 检查校验和 and ACK位
    checksum = checkSum(recvBuf, recvResult);
    if (checksum == 0 && recvBuf[FLAG_BIT_POSITION] == 0b100) {
        seq_opp = (u_char)recvBuf[SEQ_BITS_START] +
((u_char)recvBuf[SEQ_BITS_START + 1] << 8)
            + ((u_char)recvBuf[SEQ_BITS_START + 2] << 16) +
((u_char)recvBuf[SEQ_BITS_START + 3] << 24);
        ack_opp = (u_char)recvBuf[ACK_BITS_START] +
((u_char)recvBuf[ACK_BITS_START + 1] << 8)
            + ((u_char)recvBuf[ACK_BITS_START + 2] << 16) +
((u_char)recvBuf[ACK_BITS_START + 3] << 24);
        if (seq_opp == ack) { // 检查收到的包的seq(即seq_opp)是否等于上一个包发过去
的ack
            // 如果收到了正确的包，那就提取内容 + 回复
            dataLength = recvResult - HEADERSIZE;
```

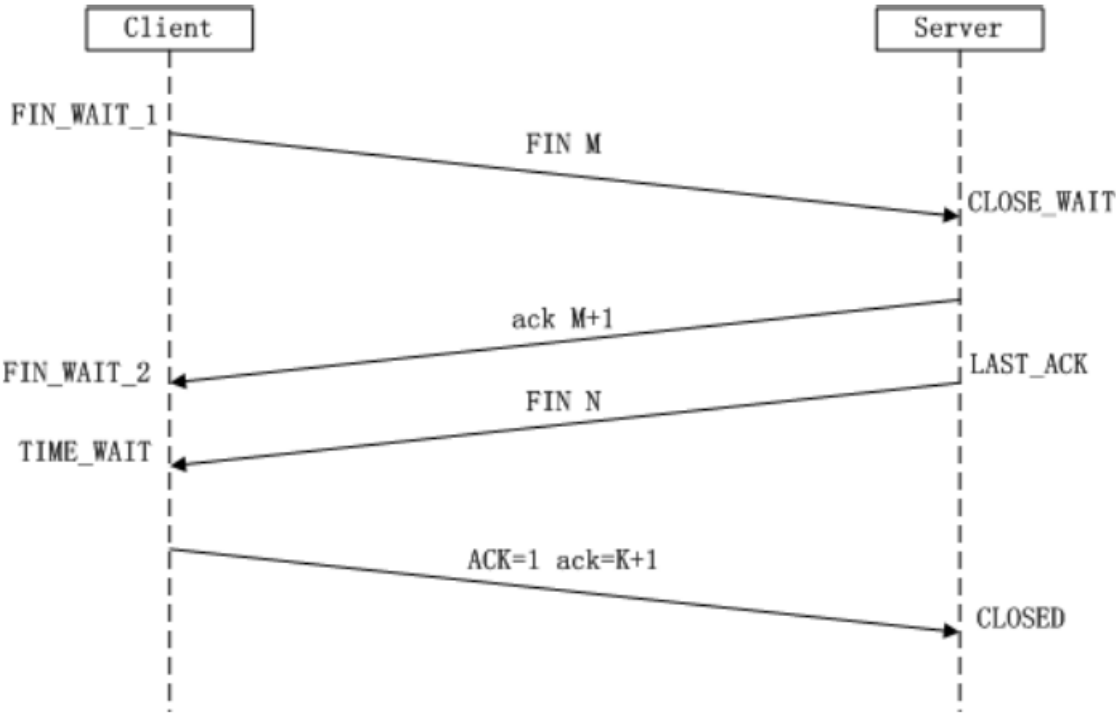
```
            // 提取数据
            memcpy(dataSegment, recvBuf + HEADERSIZE, dataLength);
            out.write(dataSegment, dataLength);

            // 设置seq位，本协议中为了确认方便，就把响应报文的seq置为收到报文的seq
            seq = seq_opp;
            header[SEQ_BITS_START] = (u_char)(seq & 0xFF);
            header[SEQ_BITS_START + 1] = (u_char)(seq >> 8);
            header[SEQ_BITS_START + 2] = (u_char)(seq >> 16);
            header[SEQ_BITS_START + 3] = (u_char)(seq >> 24);
            // 设置ack位，= seq_opp + dataLength，表示确认接收到了这之前的全部内
容，并期待收到这之后的内容
            ack = seq_opp + dataLength;
            header[ACK_BITS_START] = (u_char)(ack & 0xFF);
            header[ACK_BITS_START + 1] = (u_char)(ack >> 8);
            header[ACK_BITS_START + 2] = (u_char)(ack >> 16);
            header[ACK_BITS_START + 3] = (u_char)(ack >> 24);
            // 设置ACK位
            header[FLAG_BIT_POSITION] = 0b100;
            // 响应报文中的data length为0，就不用设置了

            hasReceived += recvResult - HEADERSIZE;
            // cout << "has received " << hasReceived << " bytes, ack = " <<
ack << endl;
            sendto(recvSocket, header, HEADERSIZE, 0, (SOCKADDR*)&sendAddr,
sizeof(SOCKADDR));
        } else {
            // 说明网络异常，丢了包，所以不用更改，直接重发上一个包的ack即可
            // TODO: 再用函数封装一下
            sendto(recvSocket, header, HEADERSIZE, 0, (SOCKADDR*)&sendAddr,
sizeof(SOCKADDR));
            cout << "seq_opp != ack." << endl;
        }
    } else {
        // 校验和或ACK位异常，重发上一个包的ack
        // TODO: 再用函数封装一下
        cout << "checksum ERROR or ACK ERROR!" << endl;
        continue;
    }

    if (hasReceived == filesize) {
        cout << "receive file " << filename << " successfully! total " <<
hasReceived << " bytes." << endl;
        break;
    }
}
```

4. 对于每次收到的`recvBuf`，需要先要检查是否是挥手报文（FIN位是否被置为1），如果是则进入`wavehand()`函数

```
// 检查是否是挥手报文
if (recvBuf[FLAG_BIT_POSITION] == 0b101) {
    // 记录一下seq
    for (int i = 0; i < 4; i++) {
        header[SEQ_BITS_START + i] = recvBuf[SEQ_BITS_START + i];
    }
    cout << "successfully received the Fisrt Wavehand message!" << endl;
    wavehand();
    return;
}
```

wavehand()函数

1. 第一次挥手：客户端发送报文：FIN = 1, ACK = 1, seq = x，进入FIN_WAIT_1状态
2. 第二次挥手：服务器端收到报文，发送报文：ACK = 1, seq = y, ack = x + 1，告诉客户端，我知道你要关闭了，等我处理一下数据，之后进入CLOSE_WAIT状态；客户端收到报文后进入FIN_WAIT_2状态
3. 第三次挥手：服务器端发送报文：FIN = 1, ACK = 1, seq = z, ack = x + 1，进入LAST_ACK状态
4. 第四次挥手：客户端收到报文，发送报文：ACK = 1, seq = x + 1, ack = z + 1之后进入TIME_WAIT状态；服务器端受到后进入CLOSED状态，客户端进入TIME_WAIT状态等待2MSL（报文最大生存时间）后也进入CLOSED状态



和握手一样，按握手过程以此 `设置发送的header/检查接收的header` 中不同的位就行了。代码可以直接看文件吧。

但是在这个程序中由于receive端不需要向send端发数据，所以第二、三次挥手显然是可以合到一起的。