

南 开 大 学

计算机网络

1811507 文静静

2020 年 12 月 8 日

基于 UDP 服务设计可靠传输协议并编程实现 3-1

一、实验内容

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。流量控制采用停等机制，完成给定测试文件的传输。

二、实验要求

- (1) 实现单向传输。
- (2) 给出详细的协议设计。
- (3) 完成详细的实验报告。
- (4) 完成给定测试文件的传输，显示传输时间和平均吞吐率。
- (4) 编写的程序应结构清晰，具有较好的可读性。
- (5) 提交程序源码和实验报告。

三、协议设计

协议特点：

- 发送方和接收方有握手过程，确认连接；
- 数据可以切包传输，保存序列号；
- 数据报有差错检测功能，正确接收返回确认；
- 数据报有确认重传，解决出错、丢失等情况；

- 数据报使用停等传输机制和序列号，不会出现失序情况；
- 发送方和接收方有挥手过程，断开连接。

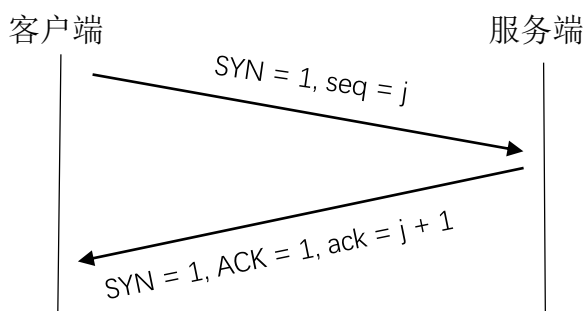
数据包格式：

- seq 和 ack：32 位，用于握手确认连接和发送切片时保存序列号；
- 标志位：ACK，SYN/FIN，用于握手确认连接和发包确认接收以及挥手断开连接；
- 长度：不包括头部，只保存数据长度。

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
发送序号(seq)															
确认序号(ack)															
ACK								SYN/FIN							
长度(length)															
校验和(checksum)															
数据(data)															

握手确认连接：

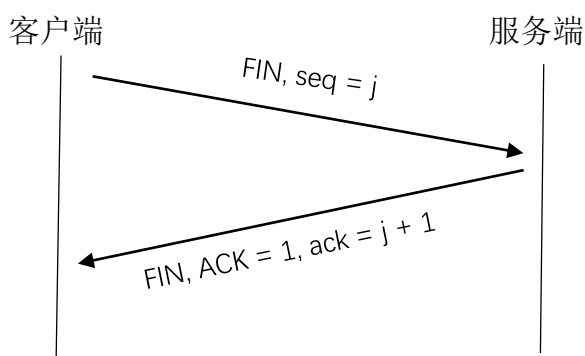
单向传输，二次握手确认连接



- 步骤一：客户端发送 SYN 段
- 步骤二：服务端接收 SYN 段，回送 SYN+ACK 段

✚ 挥手断开连接：

单向传输，二次挥手断开连接



- 步骤一：客户端发送 FIN 段
- 步骤二：服务端接收 FIN 段，回送 FIN+ACK 段

✚ 数据报的差错检测：

- 发送端：对要发送的 UDP 数据报，校验和域段清 0，将数据报用 0 补齐为 16 位整数倍，将数据报看成 16 位整数序列，进行 16 位二进制反码求和运算计算校验和，并将校验和结果取反写入校验和域段；
- 接收端：对接收到的 UDP 数据报，将数据报用 0 补齐为 16 位整数倍，将数据报看成 16 位整数序列，进行 16 位二进制反码求和运算计算校验和，并将校验和结果取反，如果取反结果为 0，则没有检测到错误，否则，数据报存在差错。

数据报切包发送:

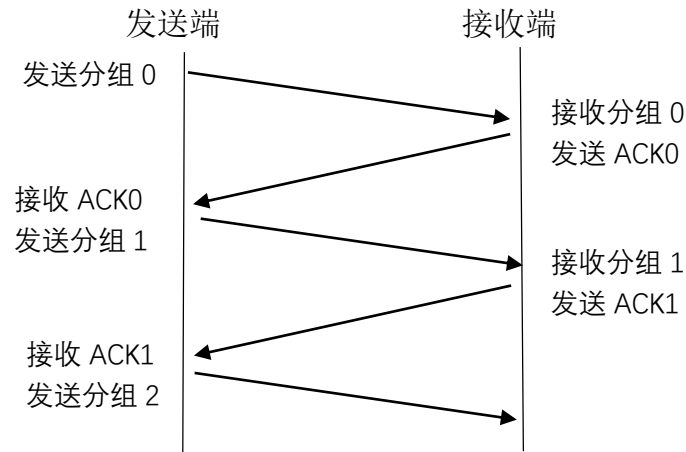
- 数据报大小上限为 `packet size` (测试时为 1500) 字节, 数据报头部为固定 10 个字节, 数据域段上限为 $(\text{packet size} - 10)$ 字节, 当要传输的数据长度大于 $(\text{packet size} - 10)$ 字节时, 需要进行切包发送。
- 发送端使用 `seq` 表示当前数据包的序列号, 使用 `ack` 表示下一个将要发的数据包的序列号, 如果 `ack` 为 0 表示现在发送的数据包为传输文件的最后一个数据包。
- 接收端接收数据包时检测收到的包的序列号是否为上一次收到包的序列号加一, 避免重复接收, 如果收到重复接受的数据包将直接丢弃。
- 接收端接收到 `ack` 为 0 的数据包, 表示为传输文件的最后一个包。

数据报确认重传:

- 接收端收到包之后, 计算校验和, 如果数据包检测无差错, 将 `ACK` 标志置为 1, 发送给发送端, 确认收到; 如果数据包校验和有差错, 将 `ACK` 标志置为 0, 发送给发送端, 表示出现包错误, 并丢弃错误的数据包;
- 发送端收到接收端发来的 `ACK` 之后, 如果 `ACK` 标志位为 1, 表示数据包发送成功, 发送端继续发送下一个数据包; 如果 `ACK` 标志位为 0, 表示数据包有差错, 发送端将重传数据包。

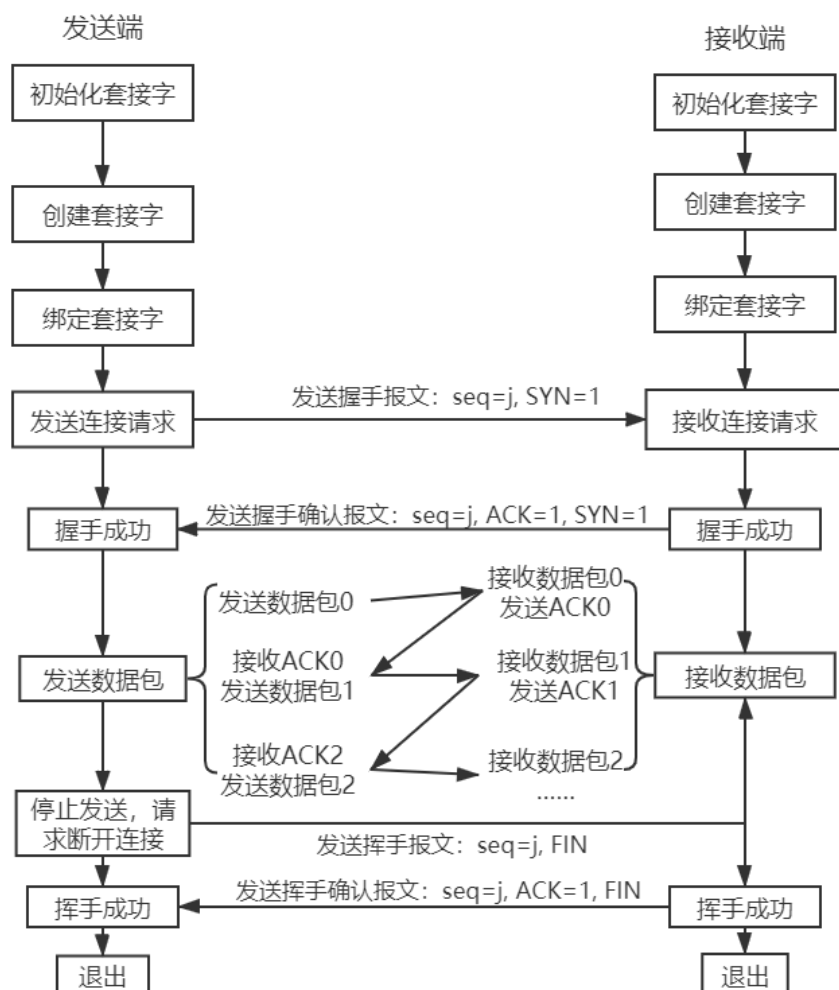
数据报停等机制:

发送端发送数据包之后, 只有收到 `ACK` 才会继续发送下一个数据包。避免失序, 实现可靠机制。



四、代码思路

✚ 发送端和接收端数据传输流程：



- ✚ 发送端和接收端数据报头：10 个字节，保存数据报序列号 seq、下一个数据报序列号 ack 和标志位 ACK, SYN/FIN、数据长度、校验和。

```
void setPacketHead(char* sendBuf) {
    sendBuf[0] = 0; //seq高8位
    sendBuf[1] = 0; //seq低8位
    sendBuf[2] = 0; //ack高8位
    sendBuf[3] = 0; //ack低8位
    sendBuf[4] = 0; //ACK
    sendBuf[5] = 0; //SYN/FIN(1 for SYN, 2 for FIN)
    sendBuf[6] = 0; //长度高8位
    sendBuf[7] = 0; //长度低8位
    sendBuf[8] = 0; //校验和高8位
    sendBuf[9] = 0; //校验和低8位
}
```

- ✚ 发送端握手请求：

```
char sendBuf[10] = { 0 };
char receiveBuf[10] = { 0 };
//设置数据报头
setPacketHead(sendBuf);
seq = rand() % 100; //得到随机数j, 并将j保存到seq对应位置
sendBuf[0] = (u_char)(seq >> 8);
sendBuf[1] = (u_char)(seq & 0xFF);
sendBuf[5] = 1; //SYN, 表示握手报文
//使用u_short数组将char型数据包两位两位拼接并进行校验和
u_short buf[6] = { 0 };
int i;
for (i = 0; i < 10; i += 2) {
    buf[i / 2] = ((u_char)sendBuf[i] << 8);
    if ((i + 1) < 10) {
        buf[i / 2] += (u_char)sendBuf[i + 1];
    }
}
u_short checks = checksum(buf, i / 2);
//将校验和填入对应位置
sendBuf[8] = (u_char)(checks >> 8); //校验和
sendBuf[9] = (u_char)(checks & 0xFF); //校验和
//发送握手请求报文
sendto(sendSocket, sendBuf, 10, 0, (SOCKADDR*)&receiveaddr, len);
```

接收端握手确认：

循环接收连接请求报文，通过判断数据报的 SYN 标志位是否为 1 来判断连接请求报文，收到请求报文之后，检验校验和，如果校验和为 0，则回送握手确认报文，将 ack 设置为 seq+1，并将标志位 SYN 和 ACK 都置 1。

如果校验和不对，或者捕获到不是 SYN 请求连接报文，将显示握手失败，退出。

```
while (true) {
    //接收报文
    res = recvfrom(receiveSocket, receiveBuf, 10, 0, (SOCKADDR*)&sendaddr, &len);
    if (res == SOCKET_ERROR) {
        Sleep(2000);
        continue;
    }
    if (receiveBuf[5] == 1) { //即数据报的SYN位为1, 标志握手请求报文
        //设置发送数据报头
        setPacketHead(sendBuf);
        //两位两位拼接进行校验和
        u_short buf[6] = { 0 };
        int i;
        for (i = 0; i < 10; i += 2) {
            buf[i / 2] = (u_char)receiveBuf[i] << 8;
            if ((i + 1) < 10) {
                buf[i / 2] += (u_char)receiveBuf[i + 1];
            }
        }
        u_short checks = checksum(buf, i / 2);
        if (checks == 0) { //校验和为0, 表示数据报传输正确
            //获取数据报中的seq
            seq = ((u_char)receiveBuf[0] << 8) + (u_char)receiveBuf[1];
            sendBuf[0] = (u_char)(seq >> 8);
            sendBuf[1] = (u_char)(seq & 0xFF);
            //确认序号ack=seq+1
            ack = seq + 1;
            sendBuf[2] = (u_char)(ack >> 8);
            sendBuf[3] = (u_char)(ack & 0xFF);
            //标志位SYN和ACK置1
            sendBuf[4] = 1; //ACK
            sendBuf[5] = 1; //SYN
            sendBuf[8] = checks >> 8; //校验和
            sendBuf[9] = checks & 0xFF; //校验和
            //发送握手确认报文
            sendto(receiveSocket, sendBuf, 10, 0, (SOCKADDR*)&sendaddr, len);
            cout << "shake hand successfully" << endl;
            return 1;
        }
    }
}
```


✚ 发送端收到连接确认报文:

循环接收握手确认报文，如果报文的 SYN 和 ACK 位均为 1，表示握手确认报文，获取报文的 ack，如果 $ack=seq+1$ ，则握手成功，否则失败。

```
while (true) {
    res = recvfrom(sendSocket, receiveBuf, 10, 0, (SOCKADDR*)&receiveaddr, &len);
    if (res == SOCKET_ERROR) {
        Sleep(2000);
        continue;
    }
    if (receiveBuf[5] == 1 && receiveBuf[4] == 1) {
        //获取确认报文的ack
        ack = ((u_char)receiveBuf[2] << 8) + (u_char)receiveBuf[3];
        //如果ack=seq+1,则握手成功
        if (seq + 1 == ack) {
            cout << "shake hand successfully" << endl;
            return 1;
        }
        else {
            cout << "shake hand failed" << endl;
            break;
        }
    }
    else {
        cout << "shake hand failed" << endl;
        break;
    }
}
return 0;
```

✚ 发送端发送数据:

如果建立连接成功，那么发送端就可以开始发送数据报文。

➤ 读取文件

根据传入的文件路径，打开文件，获取大小并读取文件内容。

```
ifstream in(filename, ifstream::in | ios::binary);
in.seekg(0, in.end); //将指针定位在文件结束处
int length = in.tellg(); //获取文件大小
in.seekg(0, in.beg); //将指针定位在文件头
char* buffer = new char[length];
memset(buffer, 0, sizeof(char) * length);
in.read(buffer, length); //读取文件数据到缓冲区buffer中
in.close(); //关闭文件
```

➤ 切分数据

将缓冲区数据切分成大小为(packetsize-10)字节大小(packetsize 为数据报大小, 10 字节为报头大小), 使用 ll 标记当前将要放到数据报中的数据在缓冲区的起始下标, 使用 lk 标记结束下标, 当收到该数据包的 ACK 之后, 使 ll=lk, 开始新一轮发送。

```
//循环发送
while (true) {
    //使用resend标志确认是否为重传
    //如果是重传, 不需要处理数据包, 直接重新发送一遍
    //如果不是重传, 需要处理数据包
    if (!resend) {
        setPacketHead(sendBuf);
        //如果数据缓冲区剩下的数据不足一个数据包
        //那么只读取length-ll的数据, 到结束
        lk = min(length - ll, packetsize - 10) + ll;
        //将缓冲区对应段数据放入数据报对应位置
        for (int i = ll; i < lk; i++) {
            sendBuf[i - ll + 10] = (u_char)buffer[i];
        }
        //保存长度和数据包序列号
        sendBuf[6] = (lk - ll) >> 8; //长度
        sendBuf[7] = (lk - ll) & 0xFF; //长度
    }
}
```

➤ 打包数据包

✧ 设置数据包的序列号 seq, 下一个包的序列号 ack(如果之后还有包,

ack+seq+1, 如果是最后一个包, ack=0)

```
//保存数据包序列号
sendBuf[0] = (u_char)(seq >> 8);
sendBuf[1] = (u_char)(seq & 0xFF);
//如果之后还有包, 将下一个包的序列号保存在ack
ack = seq + 1;
//如果是最后一个包, 则将ack置0
if (length - ll <= (packetsize - 10)) {
    ack = 0;
}
sendBuf[2] = (u_char)(ack >> 8);
sendBuf[3] = (u_char)(ack & 0xFF);
```

- ✧ 计算校验和，并将校验和填入对应位置

```
//将数据包两位两位拼接进行校验
u_short* buf = new u_short[packetsize / 2 + 1];
memset(buf, 0, packetsize / 2 + 1);
int i;
for (i = 0; i < 10 + 1k - 11; i += 2) {
    buf[i / 2] = ((u_char)sendBuf[i] << 8);
    if ((i + 1) < 10 + 1k - 11) {
        buf[i / 2] += (u_char)sendBuf[i + 1];
    }
}
u_short checks = checksum(buf, i / 2);
//将校验和字段保存
sendBuf[8] = checks >> 8; //校验和
sendBuf[9] = checks & 0xFF; //校验和
```

计算校验和函数：如果有进位，则取反加一，最后返回结果取反。

```
u_short checksum(u_short* buf, int count) {
    register u_long sum = 0;
    while (count--) {
        sum += *buf++;
        if (sum & 0xFFFF0000) {
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

- ✧ 发送数据包，并将重传位置 0

```
//发送数据包
sendto(sendSocket, sendBuf, packetsize, 0, (SOCKADDR*)&receiveaddr, len);
```

接收端接收数据：

➤ 接收数据

循环接收数据报，如果收到的数据报的第 6 个字节为 2 表示为 FIN 报文，即挥手申请断开连接，如果收到的数据报的第 6 个字节不为 1 (SYN)，2 (FIN)，并且数据报序列号连续，将对数据报进行处理，获取长度序列号，计算校验

和。

```
//数据报的第6个字节为2, 表示挥手报文
if (receiveBuf[5] == 2) { ... }
if (receiveBuf[5] != 2 && receiveBuf[5] != 1 && //数据报的第6个字节表示SYN(1)/FIN(2)
    seq + 1 == ((u_char)receiveBuf[0] << 8) + (u_char)receiveBuf[1]) { //确认数据包序号连续
    setPacketHead(sendBuf);
    u_short* buf = new u_short[packetSize / 2 + 1];
    memset(buf, 0, packetSize / 2 + 1);
    //获取数据的长度, 数据报的序列号和下一个序列号
    int length = ((u_char)receiveBuf[6] << 8) + (u_char)receiveBuf[7];
    seq = ((u_char)receiveBuf[0] << 8) + (u_char)receiveBuf[1];
    ack = ((u_char)receiveBuf[2] << 8) + (u_char)receiveBuf[3];
    //计算校验和
    int i;
    for (i = 0; i < 10 + length; i += 2) {
        buf[i / 2] = ((u_char)receiveBuf[i] << 8);
        if ((i + 1) < 10 + length) {
            buf[i / 2] += (u_char)receiveBuf[i + 1];
        }
    }
    u_short checks = checksum(buf, i / 2);
    sendBuf[0] = receiveBuf[0];
    sendBuf[1] = receiveBuf[1];
    sendBuf[2] = receiveBuf[2];
    sendBuf[3] = receiveBuf[3];
    sendBuf[4] = 0; //ACK
    sendBuf[8] = checks >> 8; //校验和
    sendBuf[9] = checks & 0xFF; //校验和
}
```

➤ 发送数据包确认报文

如果校验和为 0, 表示数据报传输正确, 并将数据报中的数据写入接收文件中, 并设置 ACK 为 1, 发送 ACK 确认报文; 如果校验和不为 0, 那么 ACK 位为 0, 表示 NAK, 然后发送 NAK 确认报文。

```
if (checks == 0) {
    sendBuf[4] = 1; //ACK
    char* buffer = new char[length];
    for (int j = 0; j < length; j++) {
        buffer[j] = receiveBuf[10 + j];
    }
    out.write(buffer, length);
}
sendto(receiveSocket, sendBuf, 10, 0, (SOCKADDR*)&sendaddr, len);
```

如果发送的包为最后一个包, 显示传输成功, 将初始序列 seq 设为-1, 关闭打开的文件, 标识文件个数的 k 自增, 然后跳出接收循环。

```
if (checks == 0 && ack == 0) {
    cout << "receive the file" << filename << " successfully" << endl;
    seq = -1;
    out.close();
    k++;
    break;
}
```

使用 k 来标记接受的文件个数以及名字

```
filename = '0' + k;

ofstream out(filename, ifstream::out | ios::binary | ios::app);
```

✚ 发送端接收 ACK:

接收到对应序列号（即接收到的 ACK/NAK 确认报文序列号和发送的数据包的序列号相同），如果收到 ACK，那么序列号增加，进入下一个数据包发送；如果收到 NAK，那么将重传位 resend 置 1，以便进行重传。

```
while (true) {
    res = recvfrom(sendSocket, receiveBuf, 10, 0, (SOCKADDR*)&receiveaddr, &len);
    if (res == SOCKET_ERROR) {
        Sleep(2000);
        continue;
    }
    if (receiveBuf[0] == sendBuf[0] && receiveBuf[1] == sendBuf[1]) {
        if (receiveBuf[4] == 1) {
            seq++;
            break;
        }
        else { //ACK=0即NAK, 重传
            resend = 1;
            break;
        }
    }
}
```

✚ 发送端挥手请求:

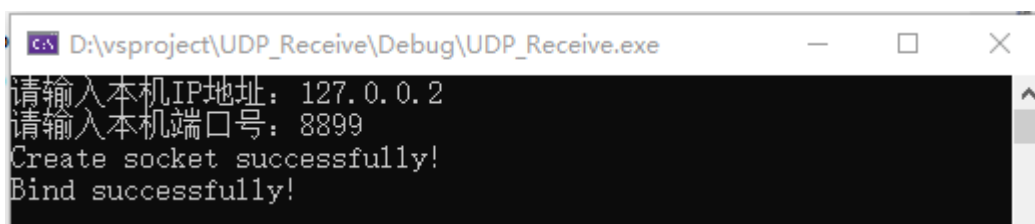
过程和握手相似，只将 SYN/FIN 标志位置 2，表示 FIN，即可发送请求。

接收端挥手确认：

接收端挥手确认和握手确认相识，只将 SYN/FIN 标志位置 2，表示 FIN，即可发送确认报文。

五、实验结果

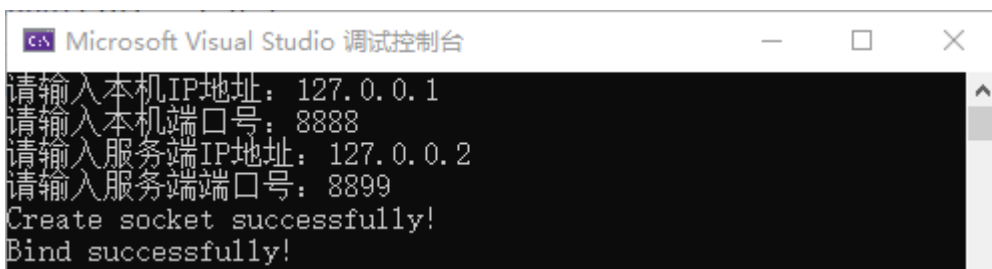
接收端开启：



```

D:\vsproject\UDP_Receive\Debug\UDP_Receive.exe
请输入本机IP地址: 127.0.0.2
请输入本机端口号: 8899
Create socket successfully!
Bind successfully!
    
```

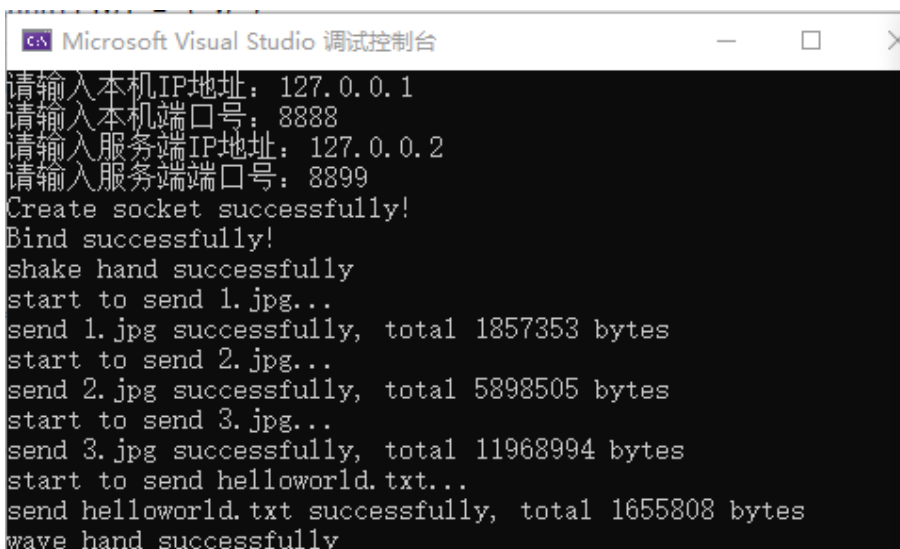
发送端开启：



```

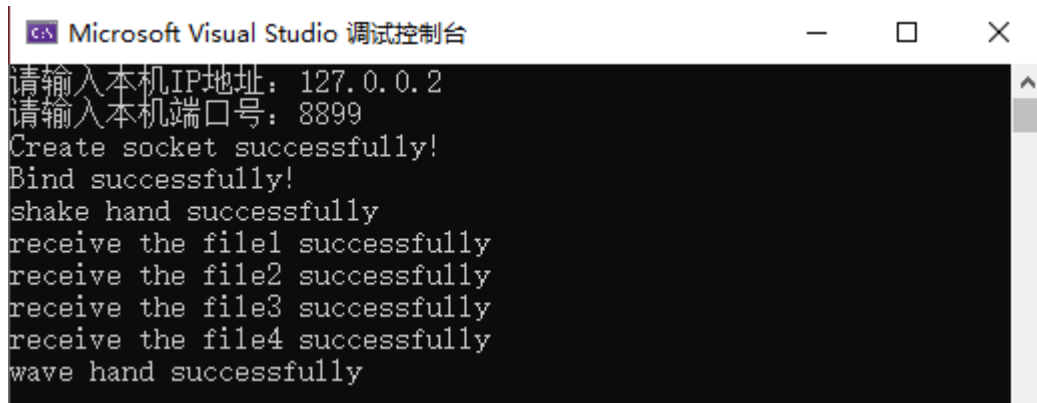
Microsoft Visual Studio 调试控制台
请输入本机IP地址: 127.0.0.1
请输入本机端口号: 8888
请输入服务端IP地址: 127.0.0.2
请输入服务端端口号: 8899
Create socket successfully!
Bind successfully!
    
```

握手成功之后发送数据，完成测试文件传输发送完毕之后挥手断开连接



```

Microsoft Visual Studio 调试控制台
请输入本机IP地址: 127.0.0.1
请输入本机端口号: 8888
请输入服务端IP地址: 127.0.0.2
请输入服务端端口号: 8899
Create socket successfully!
Bind successfully!
shake hand successfully
start to send 1.jpg...
send 1.jpg successfully, total 1857353 bytes
start to send 2.jpg...
send 2.jpg successfully, total 5898505 bytes
start to send 3.jpg...
send 3.jpg successfully, total 11968994 bytes
start to send helloworld.txt...
send helloworld.txt successfully, total 1655808 bytes
wave hand successfully
    
```



```

Microsoft Visual Studio 调试控制台
请输入本机IP地址: 127.0.0.2
请输入本机端口号: 8899
Create socket successfully!
Bind successfully!
shake hand successfully
receive the file1 successfully
receive the file2 successfully
receive the file3 successfully
receive the file4 successfully
wave hand successfully
    
```

✚ 显示传输时间和传输数据大小，以及平均吞吐率



```

sendtime:13.417 s
totallength:21380660 Bytes
吞吐率:12748.4 Kbps
    
```

如图，传输四个测试文件的时间为 13.417 s，

总传输数据大小为 21380660 Bytes，

算得的平均吞吐率为 12748.4 Kbps。