

# 基于UDP服务设计可靠传输协议并编程实现

2013622 罗昕珂

## 一、实验内容

3-1：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。

## 二、实验原理

### 协议特点

- 发送方和接收方有握手和挥手过程，确认连接断开连接；
- 数据可以切包传输，保存序列号；
- 数据报有差错检测功能，通过序列号和标志位检查，正确接收返回确认；
- 数据报有出错重传，解决出错、超时、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

- 校验和，标志位，序列号，数据
- 如果是最后一个包，则增加一个字节来保存该数据段长度，即为校验和，标志位，序列号，长度，数据。

### 校验和：

计算校验和时，累加的长度可变

### 标志位：

- ACK,用于握手确认连接和发包确认接收以及确认挥手断开连接；
- SHAKE\_1,SHAKE\_2,SHAKE\_3,三次握手确认连接
- WAVE\_1,WAVE\_2,两次挥手断开连接
- last\_len：最后一个包的数据段长度

### 序列号：

8位，用于握手确认连接和发送切片时保存序列号；

### 长度限制：

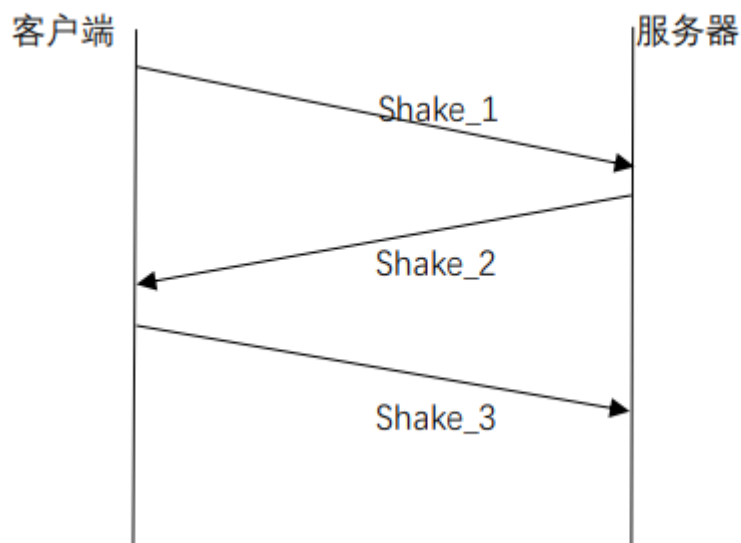
对于一个数据包的长度是有限制的，由于序列号只有8位，对于不是最后一个传输的数据包来说最多数据长度为253个字节，即数据报文为256个字节。对于传输段最后的不完整的数据来说则按具体长度发送。

### 结尾数据包标志位：

对于非结尾数据包，标志位的倒数第 4 位为 1；对于结尾数据包，标志位的倒数 4, 5 位为 1；因为文件传输期间，可能发送的整个数据段不止一个，因此序号位需要在连接未断开时不断递增，不会产生失序。

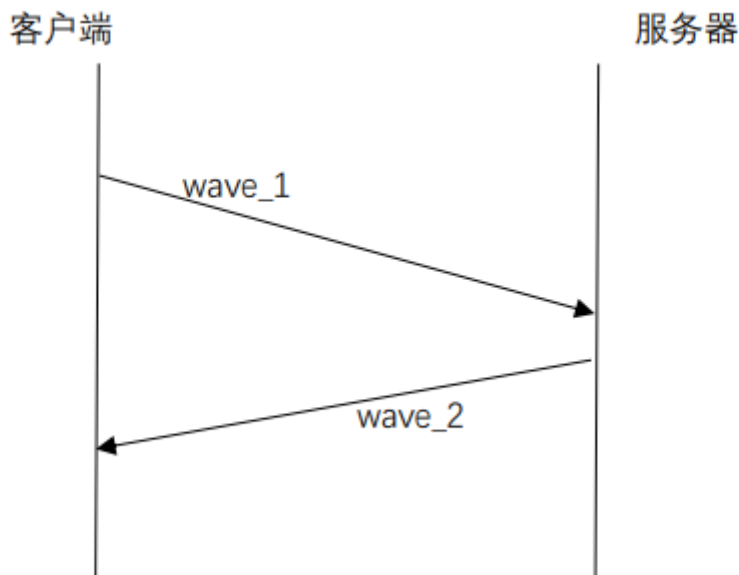
## 握手确认连接

- 单向传输，三次握手确认连接
- 步骤一：客户端发送SHAKE\_1段握手信号
- 步骤二：服务端接收SHAKE\_1段，计算校验和，若收到 SHAKE\_1 且校验和等于 0，则回送SHAKE\_2段.
- 步骤三：客户端接收SHAKE\_2段，并计算出校验和正确，开始回送SHAKE\_3段.
- 步骤四：服务器端接收到SHAKE\_3第三次握手信号，且计算出校验和正确，那么握手成功，建立连接。



## 挥手断开连接

- 单向传输，二次握手断开连接
- 步骤一：客户端发送次挥手信号 wave\_1。
- 步骤二：服务端接收后计算校验和，若收到 wave\_1 且校验和等于 0，则开始 发送第二次挥手信号 wave\_2。
- 步骤三：客户端收到服务端发来的 wave\_2，并计算出校验和正确，则挥手成功，断开连接。



## 数据报的差错检测

在发送包（包括 ACK 和 NAK）还有握手挥手的时候都需要进行校验和的计算，来 确保发包是否出错

### 1、发送端：

- 对要发送的UDP数据报，校验和域段清0
- 产生伪首部将数据报用0补齐为16位整数倍
- 将伪首部和数据报看成16位整数倍序列
- 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。

### 2、接收端：

- 对接收到的UDP数据报，产生伪首部将数据报用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，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(packet size-10)字节，当要传输的数据长度大于(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+1，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：

### 1、超时重传：

在发送一个数据包之后，就开启一个定时器，若是在这个时间内没有收到发送数据的 ACK 确认报文，则对该报文进行重传。并且将发送端的计数器减一，选择适当的序列号。

在文件传输过程中，在服务端 recvfrom 是进行阻塞接受的，使用库函数将其阻塞时间进行设置，在我们的 TIMEOUT 范围之内。

### 2、出错重传：

- 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。

### 3、挥手握手重传：

- 握手挥手过程中的丢包因为无法确认和检测校验和，只能进行重新进行握手挥手实现。
- 并且在挥手中进行超时次数的设置，超过次数认为对方完全断网，自己则进行资源回收后断网。

### 数据报停等机制：

- 当服务器接收到数据包之后，将对于数据进行校验和计算，对于"校验和"和"数据 报文"进行统一的校验操作，标志位和序列号，序列号位为其对应的接收包的序号。如果校验和计算之后的结果是 0，那么证明没有错误产生，则缓存接收到的数据报 文。之后向客户端发送ACK数据包。
- 客户端只有收到 ACK 才会继续发送下一个数据包。

有效避免失序，实现可靠机制。

## 三、实验过程

### 计算校验和

```
unsigned char checksum(char* flag, int len) { //计算校验和
    if (len == 0) {
        return ~(0);
    }
    unsigned int ret = 0;
    int i = 0;
    while (len--) {
        ret += (unsigned char)flag[i++];
        if (ret & 0xFFFF0000) {
            ret &= 0xFFFF;
            ret++;
        }
    }
    return ~(ret & 0xFFFF);
}
```

### 三次握手过程

#### 客户端

```
void shake_hand() { //握手，建立连接
    while (1) {
        //发送shake_1
        char tmp[2]; //发送数据缓存区
        tmp[1] = SHAKE_1; //第二位记录握手的字段
        tmp[0] = checksum(tmp + 1, 1);
        sendto(client, tmp, 2, 0, (sockaddr*)&serverAddr, sizeof(serverAddr));
        int begintime = clock(); //开始计时
        char recv[2];
        int last_lenmp = sizeof(clientAddr);
        int fail_send = 0; //记录失败次数
        while (recvfrom(client, recv, 2, 0, (sockaddr*)&serverAddr, &last_lenmp))
```

```

== SOCKET_ERROR)
    if (clock() - begintime > TIMEOUT) { //已超时 退出重新shake_1
        fail_send = 1;
        break;
    }
//收到shake_2并校验
if (fail_send == 0 && checksum(recv, 2) == 0 && recv[1] == SHAKE_2) {
    {
        //发送shake_3
        tmp[1] = SHAKE_3;
        tmp[0] = checksum(tmp + 1, 1);
        sendto(client, tmp, 2, 0, (sockaddr*)&serverAddr,
sizeof(serverAddr));
        break;
    }
}
}
}
}

```

## 服务端

```

void wait_shake_hand() { //等待建立连接
    while (1) {
        char recv[2];
        int reconnect = 0;
        int lentmp = sizeof(clientAddr);
        while (recvfrom(server, recv, 2, 0, (sockaddr*)&clientAddr, &lentmp) ==
SOCKET_ERROR);
        //接受了之后算出校验和不等于0或者接收到的不是shake1
        if (checksum(recv, 2) != 0 || recv[1] != SHAKE_1)
            continue;

        while (1) {

            //发送第二次握手
            recv[1] = SHAKE_2;
            recv[0] = checksum(recv + 1, 1);
            sendto(server, recv, 2, 0, (sockaddr*)&clientAddr,
sizeof(clientAddr));

            //接收shake_3
            while (recvfrom(server, recv, 2, 0, (sockaddr*)&clientAddr, &lentmp)
== SOCKET_ERROR);

            //先判断是否校验和为0和是否接受到正确的shake
            //客户端未收到shake_2, 重新发了shake_1
            if (checksum(recv, 2) == 0 && recv[1] == SHAKE_1)
                continue;

            //如果发送出错, 重新开启客户端
            if (checksum(recv, 2) != 0 || recv[1] != SHAKE_3) {

```

```

        printf("链接建立失败, 请重启客户端。");
        reconnect = 1;
    }
    break;
}
if (reconnect == 1)
    continue;
break;
}
}

```

## 两次挥手

### 客户端

```

void shake_hand() { //握手, 建立连接
    while (1) {
        //发送shake_1
        char tmp[2]; //发送数据缓存区
        tmp[1] = SHAKE_1; //第二位记录握手的字段
        tmp[0] = checksum(tmp + 1, 1);
        sendto(client, tmp, 2, 0, (sockaddr*)&serverAddr, sizeof(serverAddr));
        int begintime = clock(); //开始计时
        char recv[2];
        int last_lenmp = sizeof(clientAddr);
        int fail_send = 0; //记录失败次数
        while (recvfrom(client, recv, 2, 0, (sockaddr*)&serverAddr, &last_lenmp)
            == SOCKET_ERROR)
            if (clock() - begintime > TIMEOUT) { //已超时 退出重新shake_1
                fail_send = 1;
                break;
            }
        //收到shake_2并校验
        if (fail_send == 0 && checksum(recv, 2) == 0 && recv[1] == SHAKE_2) {
            {
                //发送shake_3
                tmp[1] = SHAKE_3;
                tmp[0] = checksum(tmp + 1, 1);
                sendto(client, tmp, 2, 0, (sockaddr*)&serverAddr,
                    sizeof(serverAddr));
                break;
            }
        }
    }
}
}

```

### 服务端

```

void wait_wave_hand() { //等待挥手
    while (1) {
        char recv[2];
        int lentmp = sizeof(clientAddr);
        //开始接收挥手信息
        while (recvfrom(server, recv, 2, 0, (sockaddr*)&clientAddr, &lentmp) ==
SOCKET_ERROR);

        //校验和不等于0或未接收到wave1
        if (checksum(recv, 2) != 0 || recv[1] != (char)WAVE_1)
            continue;

        //接收成功发送wave2
        recv[1] = WAVE_2;
        recv[0] = checksum(recv + 1, 1);
        sendto(server, recv, 2, 0, (sockaddr*)&clientAddr, sizeof(clientAddr));
        break;
    }
}

```

## 切分数据

```

//分片发包
//起始位置, 长度, 序列号, 是否是最后一个
bool send_package(char* message, int last_len, int seq, int last = 0) {
    //单片长度过大, 或不是最后一个包也不是单个片长度
    if (last_len > Mlenx && last == false && last_len != Mlenx) {
        return false;
    }

    //设置报文
    char* tmp;
    int tmp_len;
    //是最后一个包
    if (last) {
        tmp = new char[last_len + 4]; //分配缓冲区
        tmp[1] = LAST_PACK;
        tmp[2] = seq; //序列号
        tmp[3] = last_len; //比常规的片多一个长度的存储
        for (int i = 4; i < last_len + 4; i++)
            tmp[i] = message[i - 4]; //存入内容
        tmp[0] = checksum(tmp + 1, last_len + 3); //第一位存入校验和
        tmp_len = last_len + 4;
    }
    else {
        tmp = new char[last_len + 3];
        tmp[1] = NOTLAST_PACK; //不是最后一个包
        tmp[2] = seq;
        for (int i = 3; i < last_len + 3; i++)
            tmp[i] = message[i - 3];
    }
}

```

```

        tmp[0] = checksum(tmp + 1, last_len + 2);
        tmp_len = last_len + 3;
    }
    //发送该单片的信息
    sendto(client, tmp, tmp_len, 0, (sockaddr*)&serverAddr, sizeof(serverAddr));
    return true;
}

```

## 客户端发送数据

```

void send_message(char* message, int last_len) {
    int begintime; //存发送出去的时间点
    //int leave_cnt = 0;
    int has_send = 0; //已发送未确认的片数
    int nextseqnum = 0;
    int has_send_succ = 0; //已确认的片数
    int tot_package = last_len / Mlenx + (last_len % Mlenx != 0); //确定发包数
    while(1) {
        //是否已全部发送成功
        if (has_send_succ == tot_package)
            break;
        //边发送边进行接收检测，查看对方的是否有确认状态码发送过来。
        if (has_send != tot_package) {
            send_package(message + has_send * Mlenx,
                has_send == tot_package - 1 ? last_len - (tot_package - 1) * Mlenx
                : Mlenx, //是最后一个包长度取前面那个
                nextseqnum, //序列号
                has_send == tot_package - 1); //是否是最后一个包
            begintime = clock(); //开始计时
            nextseqnum++;
            has_send++;
        }
        //使用了while循环来回调换recv和send进行实现，避免了使用线程导致的繁琐。
        //收到更新定时器，继续发送。
        char recv[3];
        int last_lenmp = sizeof(serverAddr);

        //未超时收到确认包，更新计时器。
        if (recvfrom(client, recv, 3, 0, (sockaddr*)&serverAddr, &last_lenmp) !=
            SOCKET_ERROR && checksum(recv, 3) == 0 &&
            recv[1] == ACK) { //如果接收到：校验和，标志位，序列号，且校验和为0，标志
            位为ack

            has_send_succ++; //已确认
        }
        else {
            //超时后进行回退一个，重新发送
            if (clock() - begintime > TIMEOUT) {
                nextseqnum--;
                has_send--;
            }
        }
    }
}

```



```

    }
}
}
if (has_send_succ % 100 == 0)
    printf("此文件已经发送第%d个数据包\n", has_send_succ);
}

```

## 超时重传

```

//超时后进行回退一个, 重新发送
if (clock() - begintime > TIMEOUT) {
    nextseqnum --;
    has_send --;
}

```

## 服务端接收数据

```

void recv_message(char* message, int& len_recv) {
    char recv[Mlenx + 4];
    int lentmp = sizeof(clientAddr);
    static unsigned char last_seq = 0; //记录序列号
    len_recv = 0; //记录文件长度
    while (1) {
        while (1) {
            memset(recv, 0, sizeof(recv));
            while (recvfrom(server, recv, Mlenx + 4, 0, (sockaddr*)&clientAddr,
&lentmp) == SOCKET_ERROR);
            int flag = 0;
            //检查接收的序列号和校验和是否正确
            if (checksum(recv, Mlenx + 4) == 0 && (unsigned char)recv[2] ==
last_seq) {
                last_seq++;
                flag = 1;
            }
            //发送ACK回去
            char send[3];
            send[1] = ACK; //标志位
            send[2] = last_seq - 1; //序列号
            send[0] = checksum(send + 1, 2); //校验和
            sendto(server, send, 3, 0, (sockaddr*)&clientAddr,
sizeof(clientAddr));
            //出错重传
            if (flag)
                break;
        }
    }
}

```

## 服务端整理合并数据

```
//整理收到的文件
if (LAST_PACK == recv[1]) { //最后一个包多了一个长度位
    if (recv[3] + 4 < 4 && recv[3] + 4 > -127) { //长度超过127, 长度是负值,
        需要加上256变为正
        for (int i = 4; i < (int)recv[3] + 4 + 256; i++) {
            message[len_recv++] = recv[i]; //顺序接收到数据
        }
    }
    else if (recv[3] + 4 < -127) {
        for (int i = 4; i < (int)recv[3] + 4 + 256 + 128; i++) {
            message[len_recv++] = recv[i]; //顺序接收到数据
        }
    }
    else {
        for (int i = 4; i < (int)recv[3] + 4; i++) {
            message[len_recv++] = recv[i]; //顺序接收到数据
        }
    }
    break;
}
else {
    for (int i = 3; i < Mlenx + 3; i++)
        message[len_recv++] = recv[i];
}
```

## 必要的日志输出

```
if (has_send_succ % 100 == 0)
    printf("此文件已经发送第%d个数据包\n", has_send_succ);

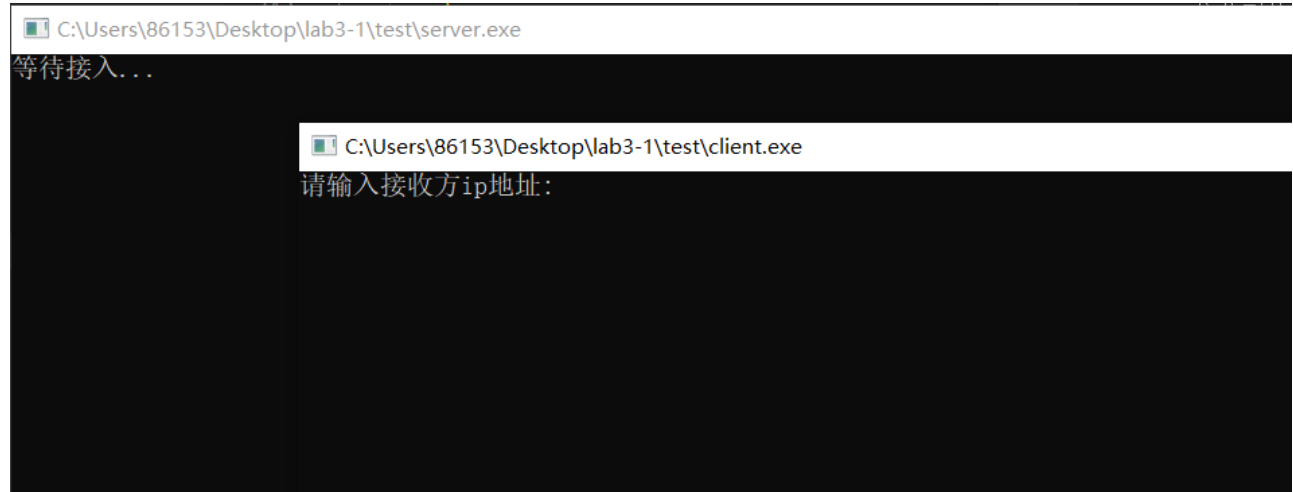
printf("连接建立中...\n");
shake_hand();
printf("握手完成, 连接已建立。 \n正在发送信息...\n");
clock_t start, end, time;
send_message((char*)(filename.c_str()), filename.length());
printf("文件名发送完毕。 \n正在发送文件内容...\n");
start = clock();
send_message(buffer, len);
end = clock();
time = (end - start) / CLOCKS_PER_SEC;
printf("发送的文件共:%dbytes\n", len);
cout << "传输时间为: " << setw(10) << (double)time << "s" << endl;
cout << "平均吞吐率为: " << len*8/1000 / (double)time << "kbps" << endl;
printf("文件内容发送完毕。 \n 开始断开连接...\n");
```

```
wave_hand();  
printf("挥手完成, 连接已断开.\n");  
printf("输入0退出\n");
```

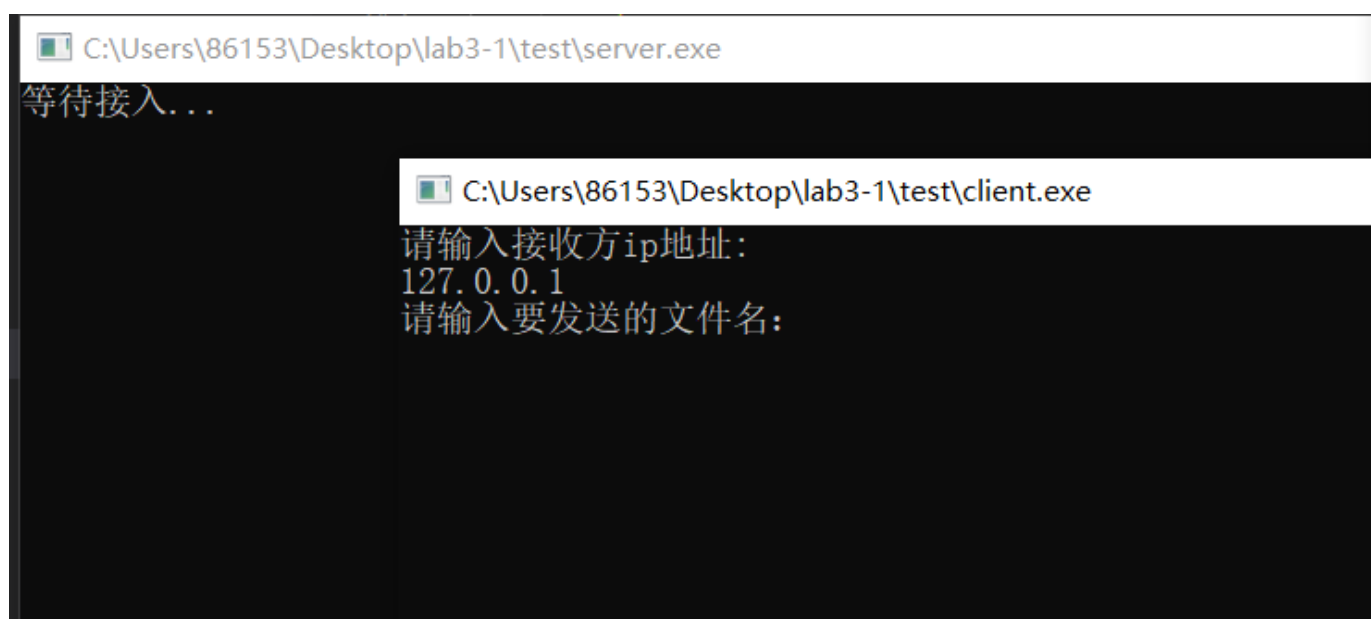
## 四、实验结果

先打开服务端再打开客户端：

- 首先能输入接收方的IP地址



- 然后输入要发送的文件名



传输第一个文件：1.jpg

C:\Users\86153\Desktop\lab3-1\test\server.exe

```
等待接入...
握手完成，用户已接入。
正在接收文件：1.jpg
接收到的文件共1857353 bytes
传输时间为：0s
平均吞吐率为：infkbps
挥手完成，连接已断开
输入0退出
```

C:\Users\86153\Desktop\lab3-1\test\client.exe

```
请输入接收方ip地址：
127.0.0.1
请输入要发送的文件名：1.jpg
连接建立中...
握手完成，连接已建立。
正在发送信息...
文件名发送完毕。
正在发送文件内容...
此文件已经发送第100个数据包
此文件已经发送第200个数据包
此文件已经发送第300个数据包
此文件已经发送第400个数据包
此文件已经发送第500个数据包
此文件已经发送第600个数据包
此文件已经发送第700个数据包
此文件已经发送第800个数据包
此文件已经发送第900个数据包
此文件已经发送第1000个数据包
此文件已经发送第1100个数据包
此文件已经发送第1200个数据包
此文件已经发送第1300个数据包
此文件已经发送第1400个数据包
此文件已经发送第1500个数据包
此文件已经发送第1600个数据包
此文件已经发送第1700个数据包
此文件已经发送第1800个数据包
此文件已经发送第1900个数据包
此文件已经发送第2000个数据包
此文件已经发送第2100个数据包
此文件已经发送第2200个数据包
此文件已经发送第2300个数据包
此文件已经发送第2400个数据包
此文件已经发送第2500个数据包
此文件已经发送第2600个数据包
发送的文件共:1857353 bytes
传输时间为：0s
平均吞吐率为：infkbps
文件内容发送完毕。
开始断开连接...
挥手完成，连接已断开。
输入0退出
```

发送的文件共:1857353 bytes

传输时间为：0s

平均吞吐率为：infkbps

传输第二个文件：2.jpg

C:\Users\86153\Desktop\lab3-1\test\server.exe

```
等待接入...
握手完成，用户已接入。
正在接收文件：2.jpg
接收到的文件共5898249 bytes
传输时间为：      36s
平均吞吐率为：1310.69kbps
挥手完成，连接已断开
输入0退出
```

C:\Users\86153\Desktop\lab3-1\test\client.exe

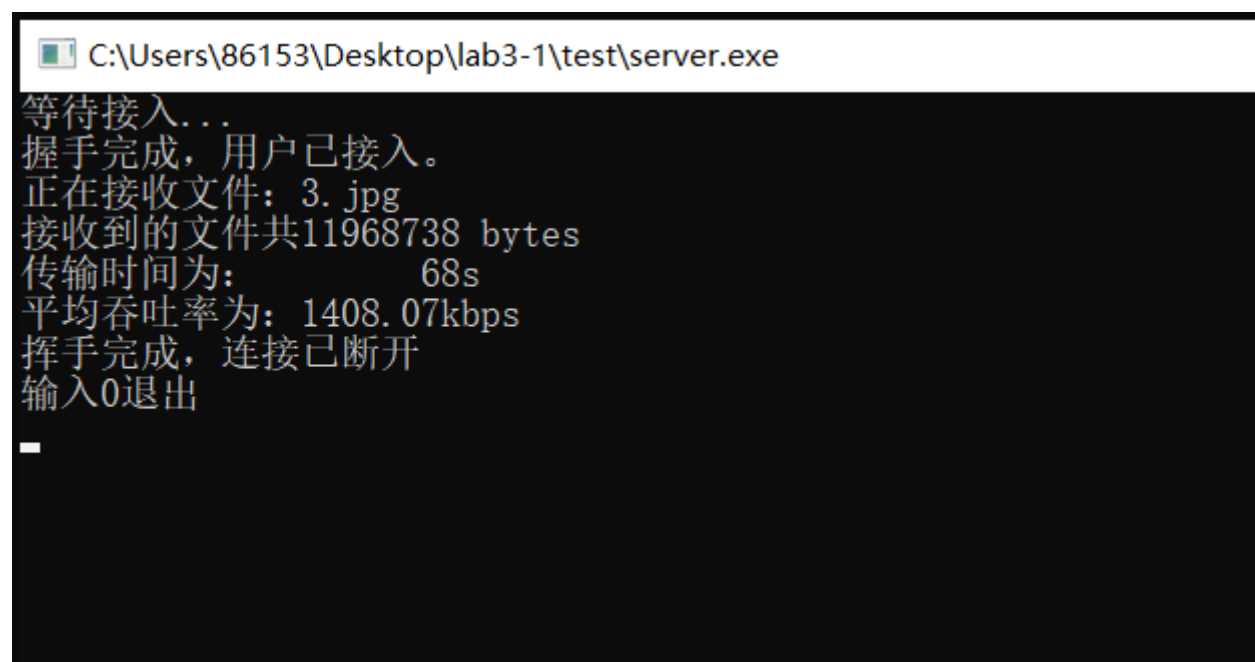
```
此文件已经发送第5100个数据包
此文件已经发送第5200个数据包
此文件已经发送第5300个数据包
此文件已经发送第5400个数据包
此文件已经发送第5500个数据包
此文件已经发送第5600个数据包
此文件已经发送第5700个数据包
此文件已经发送第5800个数据包
此文件已经发送第5900个数据包
此文件已经发送第6000个数据包
此文件已经发送第6100个数据包
此文件已经发送第6200个数据包
此文件已经发送第6300个数据包
此文件已经发送第6400个数据包
此文件已经发送第6500个数据包
此文件已经发送第6600个数据包
此文件已经发送第6700个数据包
此文件已经发送第6800个数据包
此文件已经发送第6900个数据包
此文件已经发送第7000个数据包
此文件已经发送第7100个数据包
此文件已经发送第7200个数据包
此文件已经发送第7300个数据包
此文件已经发送第7400个数据包
此文件已经发送第7500个数据包
此文件已经发送第7600个数据包
此文件已经发送第7700个数据包
此文件已经发送第7800个数据包
此文件已经发送第7900个数据包
此文件已经发送第8000个数据包
此文件已经发送第8100个数据包
此文件已经发送第8200个数据包
此文件已经发送第8300个数据包
此文件已经发送第8400个数据包
发送的文件共:5898505 bytes
传输时间为：      36s
平均吞吐率为：1310.78kbps
文件内容发送完毕。
开始断开连接...
挥手完成，连接已断开。
输入0退出
```

接收到的文件共5898249 bytes

传输时间为： 36s

平均吞吐率为：1310.69kbps

传输第三个文件：3.jpg



```
C:\Users\86153\Desktop\lab3-1\test\server.exe
等待接入...
握手完成，用户已接入。
正在接收文件：3. jpg
接收到的文件共11968738 bytes
传输时间为： 68s
平均吞吐率为：1408.07kbps
挥手完成，连接已断开
输入0退出
```

选择 C:\Users\86153\Desktop\lab3-1\test\client.exe

```
此文件已经发送第13500个数据包
此文件已经发送第13600个数据包
此文件已经发送第13700个数据包
此文件已经发送第13800个数据包
此文件已经发送第13900个数据包
此文件已经发送第14000个数据包
此文件已经发送第14100个数据包
此文件已经发送第14200个数据包
此文件已经发送第14300个数据包
此文件已经发送第14400个数据包
此文件已经发送第14500个数据包
此文件已经发送第14600个数据包
此文件已经发送第14700个数据包
此文件已经发送第14800个数据包
此文件已经发送第14900个数据包
此文件已经发送第15000个数据包
此文件已经发送第15100个数据包
此文件已经发送第15200个数据包
此文件已经发送第15300个数据包
此文件已经发送第15400个数据包
此文件已经发送第15500个数据包
此文件已经发送第15600个数据包
此文件已经发送第15700个数据包
此文件已经发送第15800个数据包
此文件已经发送第15900个数据包
此文件已经发送第16000个数据包
此文件已经发送第16100个数据包
此文件已经发送第16200个数据包
此文件已经发送第16300个数据包
此文件已经发送第16400个数据包
此文件已经发送第16500个数据包
此文件已经发送第16600个数据包
此文件已经发送第16700个数据包
此文件已经发送第16800个数据包
此文件已经发送第16900个数据包
此文件已经发送第17000个数据包
此文件已经发送第17100个数据包
发送的文件共:11968994 bytes
传输时间为: 68s
平均吞吐率为: 1408.1kbps
文件内容发送完毕。
开始断开连接...
挥手完成, 连接已断开。
输入0退出
```

发送的文件共:11968994 bytes

传输时间为: 68s

平均吞吐率为: 1408.1kbps

传输第四个文件: helloworld.txt



C:\Users\86153\Desktop\lab3-1\test\server.exe

```
等待接入...
握手完成，用户已接入。
正在接收文件：helloworld.txt
接收到的文件共1655552 bytes
传输时间为：      15s
平均吞吐率为：882.933kbps
挥手完成，连接已断开
输入0退出
```

C:\Users\86153\Desktop\lab3-1\test\client.exe

```
请输入接收方ip地址：
127.0.0.1
请输入要发送的文件名：helloworld.txt
连接建立中...
握手完成，连接已建立。
正在发送信息...
文件名发送完毕。
正在发送文件内容...
此文件已经发送第100个数据包
此文件已经发送第200个数据包
此文件已经发送第300个数据包
此文件已经发送第400个数据包
此文件已经发送第500个数据包
此文件已经发送第600个数据包
此文件已经发送第700个数据包
此文件已经发送第800个数据包
此文件已经发送第900个数据包
此文件已经发送第1000个数据包
此文件已经发送第1100个数据包
此文件已经发送第1200个数据包
此文件已经发送第1300个数据包
此文件已经发送第1400个数据包
此文件已经发送第1500个数据包
此文件已经发送第1600个数据包
此文件已经发送第1700个数据包
此文件已经发送第1800个数据包
此文件已经发送第1900个数据包
此文件已经发送第2000个数据包
此文件已经发送第2100个数据包
此文件已经发送第2200个数据包
此文件已经发送第2300个数据包
发送的文件共:1655808 bytes
传输时间为：      15s
平均吞吐率为：883.067kbps
文件内容发送完毕。
开始断开连接...
挥手完成，连接已断开。
输入0退出
```















发送的文件共:1655808 bytes

传输时间为: 15s

平均吞吐率为: 883.067kbps

传输结果文件都能1: 1还原

 1.jpg	2022/11/19 19:40	JPG 文件	1,814 KB
 1received.jpg	2022/11/19 23:27	JPG 文件	1,814 KB
 2.jpg	2019/7/1 13:48	JPG 文件	5,761 KB
 2received.jpg	2022/11/19 23:29	JPG 文件	5,761 KB
 3.jpg	2019/7/1 13:46	JPG 文件	11,689 KB
 3received.jpg	2022/11/19 23:39	JPG 文件	11,689 KB
 client.cpp	2022/11/19 23:25	C++ 源文件	11 KB
 client.exe	2022/11/19 23:23	应用程序	2,879 KB
 helloworld.txt	2020/11/20 16:17	文本文档	1,617 KB
 received.txt	2022/11/19 23:49	文本文档	1,617 KB
 server.cpp	2022/11/19 20:38	C++ 源文件	7 KB
 server.exe	2022/11/19 23:22	应用程序	2,833 KB