# 实验3-1

**学号：2112066 姓名：于成俊**

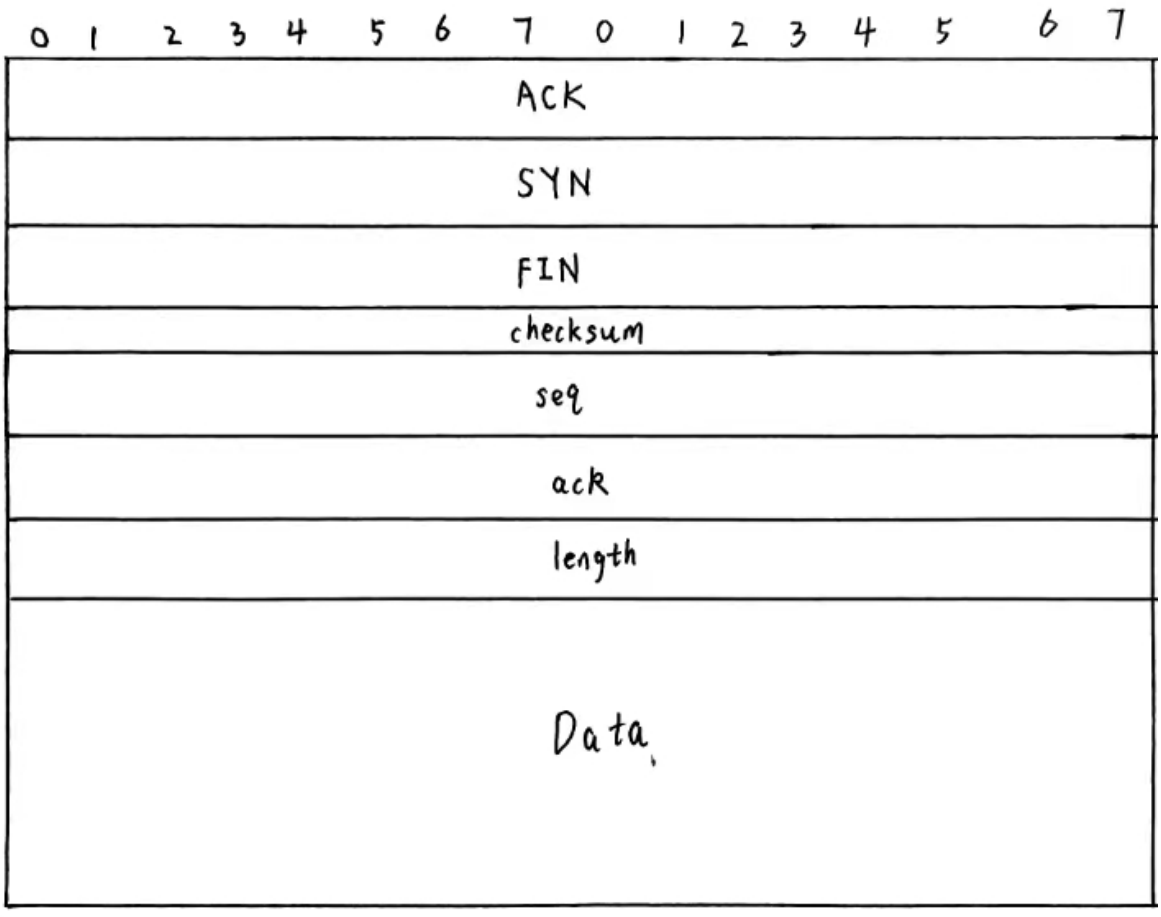
## 实验题目：

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**、**接收确认**、**超时重传**等。流量控制采用**停等机制**，完成给定测试文件的传输。

## 协议设计

### 报文格式：

所设计的报文格式如下图所示：

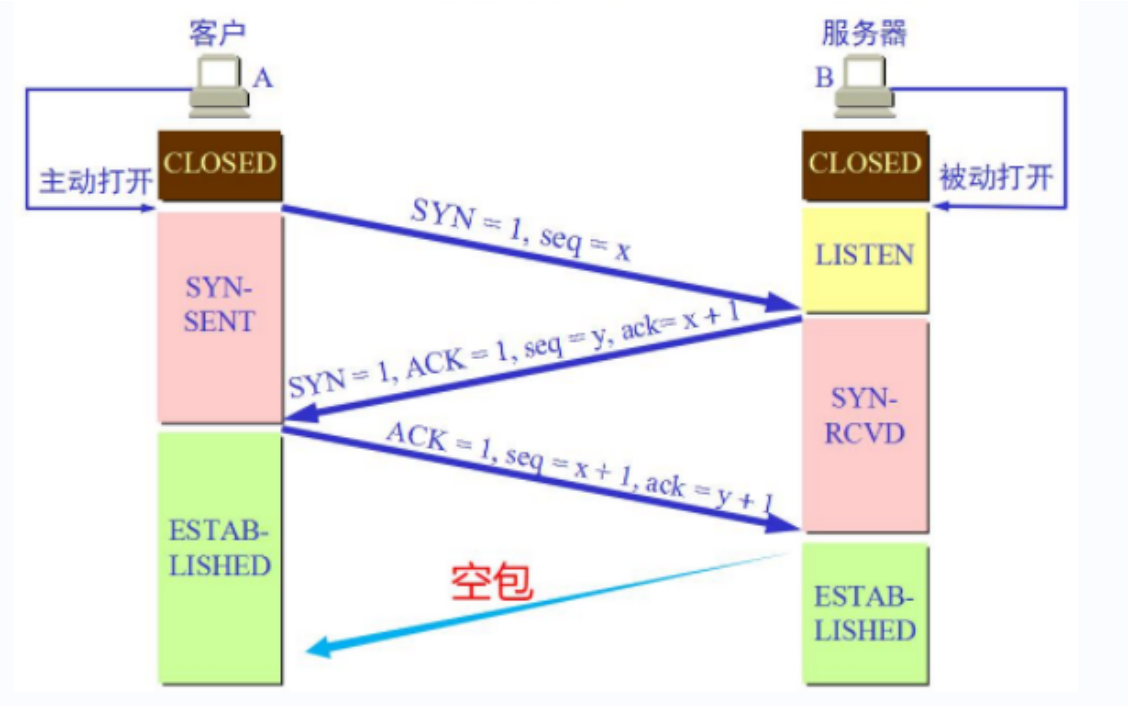



- 报文总长度为82128位，即10266字节。
- 前32位为ACK，用于确认序号有效。
- 33—64位为SYN，用于发起一个连接。
- 65—96位为FIN，用于释放一个连接；以及在数据传输过程中，用于告知数据已传输完毕。
- 97—112位为checksum，即校验和，用于差错检验。
- 113—144位为seq，为传输数据包的序列号。在建立连接和断开连接时，为随机取值；在数据传输时，只为0和1。

- 145—176位为ack，为确认序列号，只在建立连接和断开连接时使用，在数据传输时不涉及，取值为seq+1。
- 177—208位为length，为数据长度。
- 209—82128位为数据。

## 四次握手建立连接

- 参考于TCP的三次握手，所设计的流程图如下：



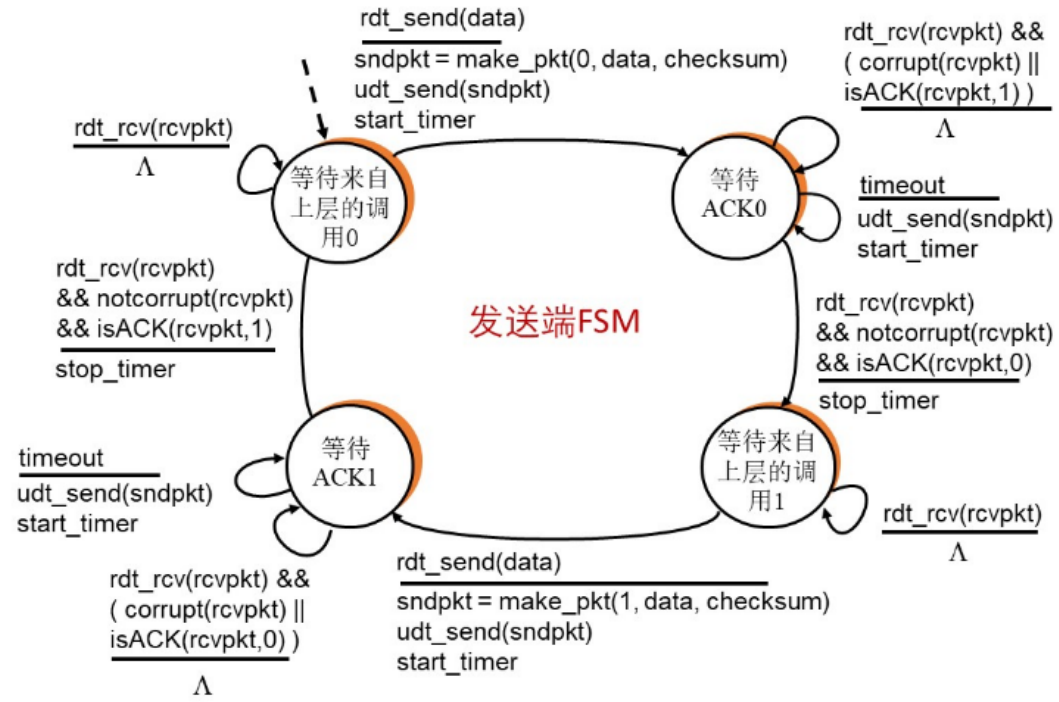

- 首先客户端向服务器端发送一个报文，其 SYN 标志位置 1，标志请求建立连接；并从1~100中随机选取一个整数x赋值给seq。
- 服务器收到请求后，向客户端回复一个报文，SYN 和 ACK 标志位置 1，标志允许建立连接；并从1~100中随机选取一个整数y赋值给seq。
- 客户端收到服务器反馈后，向服务器发送一个报文，ACK 置 1，标志将要开始传输；且将seq设为x+1，将ack设为y+1；
- 最后服务器再发送个空包（任意一个包也行）给客户端，这样，客户端收到一个空包就知道服务器端已经收到了第三次的报文。这就是第四次握手。

## 可靠数据传输

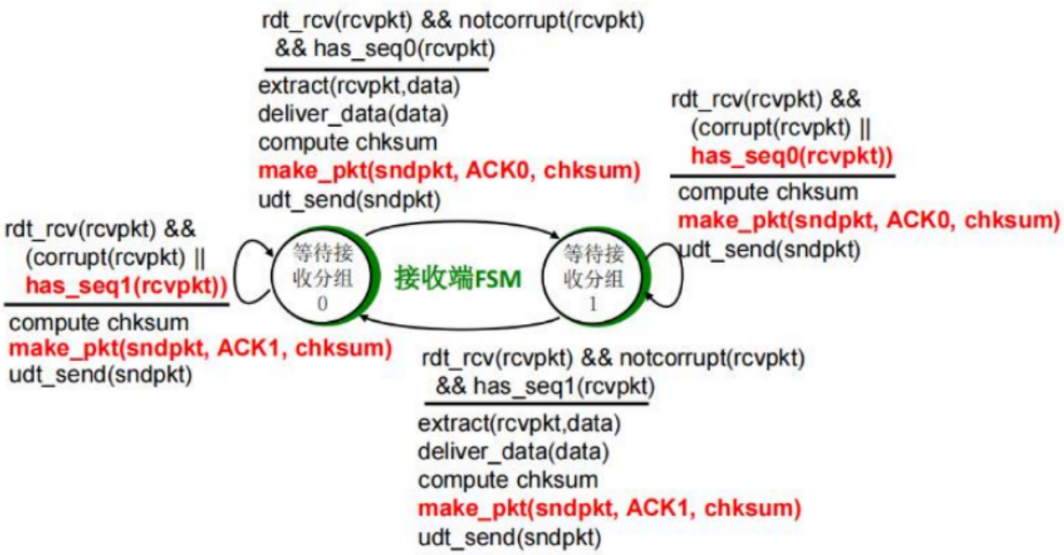
发送端和接收端均参考rdt3.0协议，采取停等协议，并加以简单修改。具体如下：

- seq的取值只为0和1。当ACK=seq时，说明服务器接收的数据报正常；当ACK!=seq时，说明服务器接收的数据报异常。
- 客户端发送 seq 号数据报时，需要计算**校验和**，并将校验和连同数据一起发送。然后等待对 seq号数据报的ACK分组，若接收到ACK分组，则**检查ACK分组的校验和是否为0且ACK是否等于seq**，若不符合条件，则重新发送数据报；若符合条件，则发送下一个数据报。若**超时**未能收到ACK分组，则重新发送数据报。若数据已发送完，则向服务器发送FIN=-1的数据报，告知数据已发送完。
- 接收端等待来自客户端发送的数据报，接收到后计算校验和。若校验和为0，说明无错误，则发送ACK=seq的数据报；若有错误，则发送ACK!=seq的数据报。若接收到的seq不是期望的序列号，则为重复的数据报，丢弃掉并发送ACK=seq的数据报。若收到FIN=-1的数据报，则不再等待接收该文件数据。

- 发送端的有限状态机如下图所示：

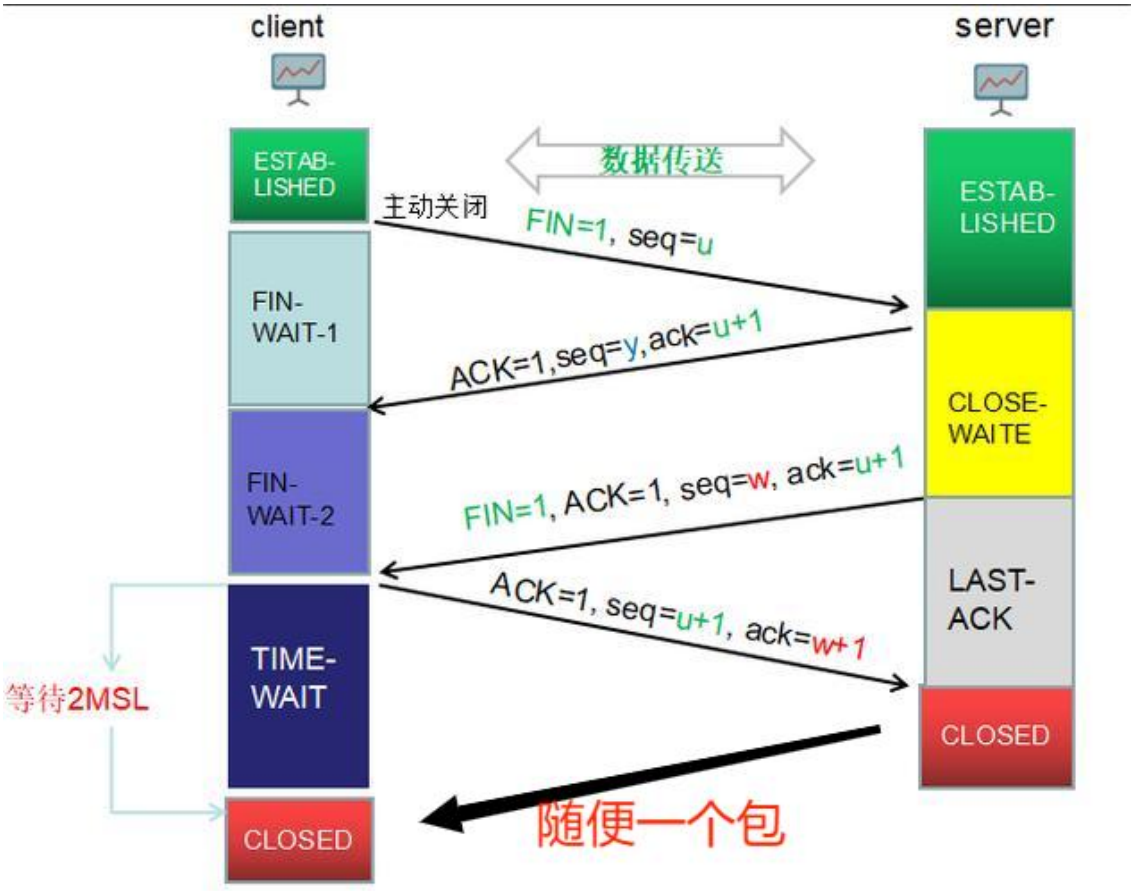



- 接收端的有限状态机如下图所示：




## 五次挥手断开连接

- 参考TCP的四次挥手，所设计的流程图如下：




- 客户端向服务器端发送一个报文，将 FIN 标志位置 1，并从1~100中随机选取一个整数u赋值给seq，标识请求断开连接。
- 服务器端收到断开请求后，回应一个报文，将 ACK 标志位置 1，并从1~100中随机选取一个整数y赋值给seq，且令ack=u+1，标识接到断开请求。
- 服务器端向客户端发送一个报文，将ACK和FIN 标志位置 1，并从1~100中随机选取一个整数w赋值给seq，且令ack=u+1，标识请求断开连接。
- 客户端收到断开请求后，回应一个报文，将 ACK 标志位置 1，令seq=u+1，ack=w+1。
- 最后，服务器随便发送一个包，我已关闭。客户端接收到后，关闭连接。

# 设计实现

### 报文格式：

```
const int BUFFER_SIZE = 10240;
//数据报
struct Datagram {
    //标志位
    int ACK, SYN, FIN;
    //校验和
    unsigned short int checksum; //unsigned short int是16位
    int seq, ack; //序列号和确认号
    int length;   //数据长度
    char data[BUFFER_SIZE];  //数据
};
```

### 差错检验机制实现（计算校验和）：

- 发送方生成校验和：
  - 将接受的数据报分成若干个16位的位串，每个位串看成一个二进制数。
  - 将校验和域段清零，该字段也参与校验和运算。
  - 对这些16位的二进制数进行反码求和。
  - 将累加的结果再取反，得到校验和，放入校验和域段

```
//发送时计算校验和
void send_calculate_Checksum(Datagram& datagram) {
    unsigned short int* buff = (unsigned short int*) & datagram;
    int num = sizeof(Datagram) / sizeof(unsigned short int);//有多少16位
    datagram.checksum = 0;//将校验和域段清0
    unsigned long int checksum = 0;//unsigned long int占四个字节
    while (num--)
    {
        checksum += *buff;
        buff++;//指向下一个16位
        //若超出16位，即有进位
        if (checksum & 0xffff0000)
        {
            //将超出16位的部分置0
            checksum &= 0xffff;
            //进位+1
            checksum++;
        }
    }
    //取反
    datagram.checksum = ~(checksum & 0xffff);
}
```

- 接收方生成校验和：

除了不用将校验和域段清零外，其他步骤一样。若结果为0，则没错误。

### 超时重传机制实现：

我使用了`setsockopt()`函数，函数原型如下：

```
int setsockopt(int sockfd , int level, int optname, void *optval, socklen_t
*optlen);
```

各参数定义：

- sockfd：要设置的套接字描述符。
- level：选项定义的层次。一般设为通用套接字代码（SOL_SOCKET）。
- optname：选项名。level对应的选项，一个level对应多个选项，不同选项对应不同功能。
- optval：指向某个变量的指针，该变量是要设置新值的缓冲区。
- optlen：optval的长度。

我将optname设为`SO_RCVTIMEO`，即接收超时。将超时时间设为timeout。

```
//接收数据时间限制5000毫秒
const int TIMEOUT = 5000;
int timeout = TIMEOUT;
```

使用方法如下：

```
    // 设置超时选项
    if (setsockopt(clientSocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout)) == SOCKET_ERROR) {
        cerr << "setsockopt failed" << endl;
        return -1;
    }
```

在某些情况下，我想让接收时间不受限制，我会**临时**将timeout设为很大，来近似不受限制。

```
timeout = INT_MAX;//INT_MAX为int类型的最大值
//重新设置
setsockopt(clientsocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout)) ;
```

## 四次握手建立连接：

- 握手过程中的差错检验是靠校验和实现的，若有错误，则等待对方重发。
- 握手过程还需检验ack、ACK、SYN等标志位是否等于预期值，若不等于，则等待对方重发。
- 超时重传机制。
  - 规定发出握手后，时间不能超过1800秒。若超过，则从头再来。

    ```
    const int maxtime = 1800; //1800秒
    ```

  - 规定请求建立连接后，不能超过3600秒。否则，连接失败。

    ```
    const int MAXTIME = 3600;//3600秒
    ```

  - 相关代码如下：

    ```
    /*----------发送端：主动握手----------*/
        clock_t startTime = clock();
        //主动发出握手
        while (!active_shakehands(clientSocket, routerAddr,
    sizeof(routerAddr))) {
            //超时
            if ((clock() - startTime) / CLOCKS_PER_SEC > MAXTIME) {
                cout << "连接不成功！" << endl;
                // 关闭套接字和清理Winsock
                closesocket(clientSocket);
                WSACleanup();
                return -1;
            }
        }
        cout << "连接成功！" << endl;
    /*---------接收端：被动握手----------*/
    ```

```
    clock_t startTime = clock();
    //被动接收握手
    while (!passive_shakehands(serverSocket, routerAddr, routerAddrLen))
{
        //超时
        if ((clock() - startTime) / CLOCKS_PER_SEC > MAXTIME) {
            cout << "连接不成功！" << endl;
            // 关闭套接字和清理Winsock
            closesocket(serverSocket);
            WSACleanup();
            return -1;
        }
    }
    cout << "连接成功！" << endl;
```

- 四次握手实现过程如下：（为了简洁，这里我只粘贴了关键代码，详见代码文件）
  - 客户端（发送端）：

```
//主动握手
bool active_shakehands(SOCKET& clientsocket, sockaddr_in& routerAddr,
int routerAddrLen) {
    clock_t starttime = clock();//记录握手开始时间
    /*----------第一次握手过程-----------*/
    while (true) {
        //超时连接不成功！
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要发送的数据清空
        memset(&SendData, 0, sizeof(Datagram));
        // 将SYN设置为1，表示要建立连接
        SendData.SYN = 1;
        //随机设定seq(1到100)
        SendData.seq = (rand() % 100) + 1;
        //计算校验和
        send_calculate_Checksum(SendData);
        //发送数据报
        sendto(clientsocket, (char*)&SendData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, routerAddrLen) ;
        //将接收数据的区域清空，以便接收数据
        memset(&ReceiveData, 0, sizeof(Datagram));
        /*-----------------第二次握手过程--------------------*/
        // 等待反馈（若超时则重新发送）
        if (recvfrom(clientsocket, (char*)&ReceiveData,
sizeof(Datagram), 0, (struct sockaddr*)&routerAddr, &routerAddrLen) ==
SOCKET_ERROR) {
            continue;
        }
        //若接受到反馈，则退出循环
        break;
    }
    //计算校验和
    receive_calculate_Checksum(ReceiveData);
    //将等待限制时间延长
    timeout = INT_MAX;
```

```
    setsockopt(clientsocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout)) ;
    //需满足条件，才不再等待，否则一直等待请求
    while (!(ReceiveData.ACK == 1 && ReceiveData.SYN == 1 &&
ReceiveData.ack == SendData.seq + 1 && ReceiveData.checksum == 0)) {
        //超时，握手失败!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        //接收数据
        recvfrom(clientsocket, (char*)&ReceiveData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, &routerAddrLen);
        //计算校验和
        receive_calculate_Checksum(ReceiveData);
    }
    /*-----------------第三次握手过程--------------------*/
    //将等待限制时间恢复正常
    timeout = TIMEOUT;
    setsockopt(clientsocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout)) ;
    while (true) {
        //超时连接不成功!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要发送的数据清空
        memset(&SendData, 0, sizeof(Datagram));
        //设置标志位
        SendData.ACK = 1;SendData.seq = ReceiveData.ack;
        SendData.ack = ReceiveData.seq + 1;
        //计算校验和
        send_calculate_Checksum(SendData);
        //发送数据报
        sendto(clientsocket, (char*)&SendData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, routerAddrLen) ;
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        /*-----------------第四次握手过程--------------------*/
        // 接收数据（接收到一个包，握手就结束）
        if (recvfrom(clientsocket, (char*)&ReceiveData,
sizeof(Datagram), 0, (struct sockaddr*)&routerAddr, &routerAddrLen) ==
SOCKET_ERROR) {
            continue;
        }
        break;
    }
    return true;
}
```

- 服务端（接收端）：

```
//被动握手
bool passive_shakehands(SOCKET& serversocket, sockaddr_in& routerAddr,
int routerAddrLen) {
    clock_t starttime = clock();
```

```
    //将等待限制时间延长
    timeout = INT_MAX;
    setsockopt(serversocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout));
    /*-----------------第一次握手过程--------------------*/
    while (true) {
        //超时连接不成功!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        // 接收数据
        if (recvfrom(serversocket, (char*)&ReceiveData,
sizeof(Datagram), 0, (struct sockaddr*)&routerAddr, &routerAddrLen) ==
SOCKET_ERROR) {
            continue;
        }
        //计算校验和
        receive_calculate_Checksum(ReceiveData);
        break;
    }
    /*-----------------第二次握手过程--------------------*/
    //将等待时间限制恢复正常
    timeout = TIMEOUT;
    setsockopt(serversocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout));
    while (true) {
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要发送的数据清空
        memset(&SendData, 0, sizeof(Datagram));
        //设置标志位
        SendData.SYN = 1;
        SendData.ACK = 1;
        SendData.seq = (rand() % 100) + 1;
        SendData.ack = ReceiveData.seq + 1;
        //计算校验和
        send_calculate_Checksum(SendData);
        sendto(serversocket, (char*)&SendData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, routerAddrLen);
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        /*-----------------第三次握手过程--------------------*/
        if (recvfrom(serversocket, (char*)&ReceiveData,
sizeof(Datagram), 0, (struct sockaddr*)&routerAddr, &routerAddrLen) ==
SOCKET_ERROR) {
            continue;
        }
        break;
    }
    //计算校验和
    receive_calculate_Checksum(ReceiveData);
    //将限制时间延长
    timeout = INT_MAX;
    setsockopt(serversocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout));
```

```
    while (!(ReceiveData.ACK == 1 && ReceiveData.seq == SendData.ack &&
ReceiveData.ack == SendData.seq + 1 && ReceiveData.checksum == 0)) {
        //超时握手失败!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        if (recvfrom(serversocket, (char*)&ReceiveData,
sizeof(Datagram), 0, (struct sockaddr*)&routerAddr, &routerAddrLen) ==
SOCKET_ERROR) {
            continue;
        }
        //计算校验和
        receive_calculate_Checksum(ReceiveData);
    }
    /*-----------------第四次握手过程--------------------*/
    //将要发送的数据清空
    memset(&SendData, 0, sizeof(Datagram));
    if (sendto(serversocket, (char*)&SendData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, routerAddrLen) == SOCKET_ERROR) {
        cerr << "发送错误! " << endl;
    }
    return true;
}
```

## 可靠数据传输

- 客户端（发送端）（包括相关日志输出、传输时间、吞吐量）

  完全发送一个文件后，再调用该函数，发送另一个文件。

```
//传输文件
int transform_file(char filepath[], SOCKET& clientSocket, sockaddr_in&
routerAddr, int& routerAddrLen) {
    cout << "现在开始发送  " << filepath << endl;
    // 以二进制模式打开文件（图片或文本文件）
    ifstream file(filepath, ios::binary);
    if (!file) {
        std::cerr << "Failed to open the file." << std::endl;
        closesocket(clientSocket);
        WSACleanup();
        return 0;
    }
    //文件总字节
    int totalBytes = 0;
    // 读取文件内容并分割成数据包
    int sequenceNumber = 0;
    //开始传输时间
    clock_t startTime = clock();
    //执行到到文件末尾（end of file，EOF）
    while (!file.eof()) {
        memset(&SendData, 0, sizeof(SendData));
        SendData.seq = sequenceNumber++;
```

```
        sequenceNumber = sequenceNumber % 2;
        //每次从文件中读取最多BUFFER_SIZE 字节的数据
        file.read(SendData.data, BUFFER_SIZE);
        int bytesToSend = static_cast<int>(file.gcount());
        totalBytes += bytesToSend;
        SendData.length = bytesToSend;
        //计算校验和
        send_calculate_Checksum(SendData);
        // 发送数据包
        while (true) {
            //打印发送的数据报
            cout << "发送数据报： ";
            printDatagram(SendData);
            if (sendto(clientSocket, (char*)&SendData, sizeof(SendData), 0,
(SOCKADDR*)&routerAddr, sizeof(routerAddr)) == SOCKET_ERROR) {
                cerr << "发送错误!" << endl;
            }
            // 等待反馈
            memset(&ReceiveData, 0, sizeof(ReceiveData));
            // 接收数据
            if (recvfrom(clientSocket, (char*)&ReceiveData,
sizeof(Datagram), 0, (struct sockaddr*)&routerAddr, &routerAddrLen) ==
SOCKET_ERROR) {
                continue;
            }
            receive_calculate_Checksum(ReceiveData);
            //打印接收的数据报
            cout << "接收数据报： ";
            printDatagram(ReceiveData);
            //因为sequenceNumber已经变了，而它只有两个值，所以不等于这个，就等于之前那
个
            if (ReceiveData.ACK != sequenceNumber && ReceiveData.checksum ==
0) {
                break;
            }
        }
    }
    file.close();
    clock_t endTime = clock();
    double transferTime = static_cast<double>(endTime - startTime) /
CLOCKS_PER_SEC;
    double throughput = (totalBytes / 1024) / transferTime; // 计算吞吐量（单
位：KB/s）
    cout << filepath << "传输完毕" << endl;
    cout << "传输文件大小为： " << totalBytes <<"B"<<endl;
    cout << "传输时间为： " << transferTime <<"s"<< endl;
    cout << "吞吐量为： " << throughput << "KB/s"<<endl;
    //告诉对方文件传输完了
    while (true) {
        memset(&SendData, 0, sizeof(SendData));
        SendData.FIN = -1;
        send_calculate_Checksum(SendData);
        //打印发送的数据报
        cout << "发送数据报： ";
        printDatagram(SendData);
        sendto(clientSocket, (char*)&SendData, sizeof(SendData), 0,
(SOCKADDR*)&routerAddr, sizeof(routerAddr));
        // 接收数据
```

```
        if (recvfrom(clientSocket, (char*)&ReceiveData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, &routerAddrLen) == SOCKET_ERROR) {
            continue;
        }
        break;
    }
    return 1;
}
```

- 服务器（接收端）：（只粘贴了关键代码）

  接收完一个文件，再调用该函数，接收另一个文件。

```
//接收文件
int receive_file(char filepath[], SOCKET& serverSocket, sockaddr_in&
routerAddr,int& routerAddrLen) {
    cout << "开始接收"<<filepath << endl;
    // 打开文件用于写入接收的数据
    ofstream outputFile(filepath, ios::binary);
    if (!outputFile) {
        cerr << "Failed to open the output file." << endl;
        closesocket(serverSocket);
        WSACleanup();
        return 0;
    }
    // 接收和重组数据包
    int sequenceNumber = 0;
    while (true) {
        memset(&ReceiveData, 0, sizeof(Datagram));
        int receivedBytes = recvfrom(serverSocket, (char*)&ReceiveData,
sizeof(ReceiveData), 0, (SOCKADDR*)&routerAddr, &routerAddrLen);
        if (receivedBytes == SOCKET_ERROR) {
            continue;
        }
        receive_calculate_Checksum(ReceiveData);
        if (ReceiveData.FIN == -1 && ReceiveData.checksum == 0) {
            break;
        }
        //接收到重复数据报
        if (ReceiveData.seq != sequenceNumber && ReceiveData.checksum == 0)
{
            //将要发送的数据清空
            memset(&SendData, 0, sizeof(Datagram));
            SendData.ACK = sequenceNumber;
            send_calculate_Checksum(SendData);
            if (sendto(serverSocket, (char*)&SendData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, routerAddrLen) == SOCKET_ERROR) {
                cerr << "发送错误! " << endl;
            }
            continue;
        }
        if (ReceiveData.seq == sequenceNumber && !(ReceiveData.checksum ==
0)) {
            //将要发送的数据清空
            memset(&SendData, 0, sizeof(Datagram));
            SendData.ACK = (sequenceNumber + 1) % 2;
```

```
            send_calculate_Checksum(SendData);
            if (sendto(serverSocket, (char*)&SendData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, routerAddrLen) == SOCKET_ERROR) {
                cerr << "发送错误！" << endl;
            }
            continue;
        }
        outputFile.write(ReceiveData.data, ReceiveData.length);
        //将要发送的数据清空
        memset(&SendData, 0, sizeof(Datagram));
        //设置ACK
        SendData.ACK = sequenceNumber;
        send_calculate_Checksum(SendData);
        if (sendto(serverSocket, (char*)&SendData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, routerAddrLen) == SOCKET_ERROR) {
            cerr << "发送错误！" << endl;
        }
        sequenceNumber = ++sequenceNumber % 2;
    }
    //随便发送个消息，告诉他我知道文件传输完了
    sendto(serverSocket, (char*)&SendData, sizeof(Datagram), 0, (struct
sockaddr*)&routerAddr, routerAddrLen);
    outputFile.close();
    return 1;
}
```

### 五次挥手断开连接：

- 客户端（发送端）：（只粘贴关键代码）

```
//主动挥手
bool active_wakehands(SOCKET& clientsocket, sockaddr_in& routerAddr, int
routerAddrLen) {
    clock_t starttime = clock();
    /*--------------第一次挥手过程------------*/
    while (true) {
        //超时挥手失败！
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要发送的数据清空
        memset(&SendData, 0, sizeof(Datagram));
        // 将FIN设置为1，表示要d断开连接
        SendData.FIN = 1;
        //随机设定seq(1到100)
        SendData.seq = (rand() % 100) + 1;
        //计算校验和
        send_calculate_Checksum(SendData);
        sendto(clientsocket, (char*)&SendData, sizeof(Datagram), 0, (struct
sockaddr*)&routerAddr, routerAddrLen);
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        /*------------第二次挥手过程---------------*/
```

```
        // 接收数据
        if (recvfrom(clientsocket, (char*)&ReceiveData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, &routerAddrLen) == SOCKET_ERROR) {
            continue;
        }
        break;
    }
    //计算校验和
    receive_calculate_Checksum(ReceiveData);
    //将限制时间延长
    timeout = INT_MAX;
    setsockopt(clientsocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout));
    while (!(ReceiveData.ACK == 1 && ReceiveData.ack == SendData.seq + 1 &&
ReceiveData.checksum == 0)) {
        //超时挥手失败!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        recvfrom(clientsocket, (char*)&ReceiveData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, &routerAddrLen);
        //计算校验和
        receive_calculate_Checksum(ReceiveData);
    }
    /*--------------------第三次挥手过程--------------*/
    //将要接受的数据清空
    memset(&ReceiveData, 0, sizeof(Datagram));
    while (!(ReceiveData.ACK == 1 && ReceiveData.FIN == 1 && ReceiveData.ack
== SendData.seq + 1 && ReceiveData.checksum == 0)) {
        //超时挥手失败!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        if (recvfrom(clientsocket, (char*)&ReceiveData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, &routerAddrLen) == SOCKET_ERROR) {
            continue;
        }
        //计算校验和
        receive_calculate_Checksum(ReceiveData);
    }
    /*--------------------第四次挥手过程--------------*/
    //将限制时间恢复正常
    timeout = TIMEOUT;
    setsockopt(clientsocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout));
    while (true) {
        //超时挥手失败!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要发送的数据清空
        memset(&SendData, 0, sizeof(Datagram));
        SendData.ACK = 1;
        SendData.seq = ReceiveData.ack;
```

```
        SendData.ack = ReceiveData.seq + 1;
        //计算校验和
        send_calculate_Checksum(SendData);
        sendto(clientsocket, (char*)&SendData, sizeof(Datagram), 0, (struct
sockaddr*)&routerAddr, routerAddrLen);
        /*--------------------第五次挥手过程--------------*/
        // 接收数据
        if (recvfrom(clientsocket, (char*)&ReceiveData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, &routerAddrLen) == SOCKET_ERROR) {
            continue;
        }
        break;
    }
    return true;
}
```

- 服务器（接收端）：（只粘贴关键代码）

```
//被动挥手
bool passive_wakehands(SOCKET& serversocket, sockaddr_in& routerAddr, int
routerAddrLen) {
    clock_t starttime = clock();
    //将限制时间延长
    timeout = INT_MAX;
    setsockopt(serversocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout));
    /*--------------------第一次挥手过程--------------*/
    while (true) {
        //超时挥手失败!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        // 接收数据
        if (recvfrom(serversocket, (char*)&ReceiveData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, &routerAddrLen) == SOCKET_ERROR) {
            continue;
        }
        //计算校验和
        receive_calculate_Checksum(ReceiveData);
        if (!(ReceiveData.checksum == 0 && ReceiveData.FIN == 1)) {
            continue;
        }
        break;
    }
    //将时间限制恢复正常
    timeout = TIMEOUT;
    setsockopt(serversocket, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout,
sizeof(timeout)) ;
    /*--------------------第二次挥手过程--------------*/
    //将要发送的数据清空
    memset(&SendData, 0, sizeof(Datagram));
    SendData.ACK = 1;
    SendData.seq = (rand() % 100) + 1;
    SendData.ack = ReceiveData.seq + 1;
    //计算校验和
```

```
    send_calculate_Checksum(SendData);
    sendto(serversocket, (char*)&SendData, sizeof(Datagram), 0, (struct
sockaddr*)&routerAddr, routerAddrLen);
    /*--------------------第三次挥手过程--------------*/
    while (true) {
        //超时挥手失败!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要发送的数据清空
        memset(&SendData, 0, sizeof(Datagram));
        SendData.ACK = 1;
        SendData.FIN = 1;
        SendData.seq = (rand() % 100) + 1;
        SendData.ack = ReceiveData.seq + 1;
        //计算校验和
        send_calculate_Checksum(SendData);
        sendto(serversocket, (char*)&SendData, sizeof(Datagram), 0, (struct
sockaddr*)&routerAddr, routerAddrLen);
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        /*--------------------第四次挥手过程--------------*/
        if (recvfrom(serversocket, (char*)&ReceiveData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, &routerAddrLen) == SOCKET_ERROR) {
            continue;
        }
        break;
    }
    //计算校验和
    receive_calculate_Checksum(ReceiveData);
    while (!(ReceiveData.ACK == 1 && ReceiveData.seq == SendData.ack &&
ReceiveData.ack == SendData.seq + 1 && ReceiveData.checksum == 0)) {
        //超时挥手失败!
        if ((clock() - starttime) / CLOCKS_PER_SEC > maxtime) {
            return false;
        }
        //将要接受的数据清空
        memset(&ReceiveData, 0, sizeof(Datagram));
        recvfrom(serversocket, (char*)&ReceiveData, sizeof(Datagram), 0,
(struct sockaddr*)&routerAddr, &routerAddrLen);
        //计算校验和
        receive_calculate_Checksum(ReceiveData);
    }
    /*--------------------第五次挥手过程--------------*/
    //随便发送个消息，告诉他挥手结束
    sendto(serversocket, (char*)&SendData, sizeof(Datagram), 0, (struct
sockaddr*)&routerAddr, routerAddrLen);
    return true;
}
```

## 实验结果：

## 四次握手建立连接:

发送端:

```
发送数据报:  Seq:  8   ACK:  0  FIN:   0
  校验和:  65526    数据长度:   0
第一次握手成功
接收数据报:  Seq:  82   ACK:  1  FIN:   0
  校验和:  0    数据长度:   0
第二次握手成功!
发送数据报:  Seq:  9   ACK:  1  FIN:   0
  校验和:  65442    数据长度:   0
第三次握手成功
第四次握手成功
连接成功!
```

接收端:

```
UDP server is listening on port 8080...
接收数据报:  Seq:  8   ACK:  0  FIN:   0
  校验和:  0    数据长度:   0
第一次握手成功
发送数据包:  Seq:  82   ACK:  1  FIN:   0
  校验和:  65442    数据长度:   0
第二次握手成功
接收数据报:  Seq:  9   ACK:  1  FIN:   0
  校验和:  0    数据长度:   0
第三次握手成功
第四次握手成功
连接成功!
```

## 文件传输:

发送端:

```
现在开始发送  helloworld.txt
发送数据报:  Seq:  0   ACK:  0  FIN:   0
  校验和:  59120    数据长度:   10240
接收数据报:  Seq:  0   ACK:  0  FIN:   0
  校验和:  0    数据长度:   0
发送数据报:  Seq:  1   ACK:  0  FIN:   0
  校验和:  59119    数据长度:   10240
接收数据报:  Seq:  0   ACK:  1  FIN:   0
  校验和:  0    数据长度:   0
发送数据报:  Seq:  0   ACK:  0  FIN:   0
  校验和:  59120    数据长度:   10240
```

传输时间和吞吐量显示:

```
helloworld.txt传输完毕
传输文件大小为:  1655808B
传输时间为:  9.643s
吞吐量为:  167.686KB/s
发送数据报:  Seq:  0   ACK:  0  FIN:   -1
  校验和:  0    数据长度:   0
```

接收端:

```
开始接收./helloworld.txt
接收数据报:  Seq:  0   ACK:  0  FIN:   0
  校验和:  0    数据长度:   10240
发送数据报:  Seq:  0   ACK:  0  FIN:   0
  校验和:  65535    数据长度:   0
接收数据报:  Seq:  1   ACK:  0  FIN:   0
  校验和:  0    数据长度:   10240
发送数据报:  Seq:  0   ACK:  1  FIN:   0
  校验和:  65534    数据长度:   0
接收数据报:  Seq:  0   ACK:  0  FIN:   0
  校验和:  0    数据长度:   10240
```

```
接收数据报:  Seq:  0   ACK:  0  FIN:   -1
  校验和:  0    数据长度:   0
./3.jpg 文件接收完毕!
开始接收./helloworld.txt
```

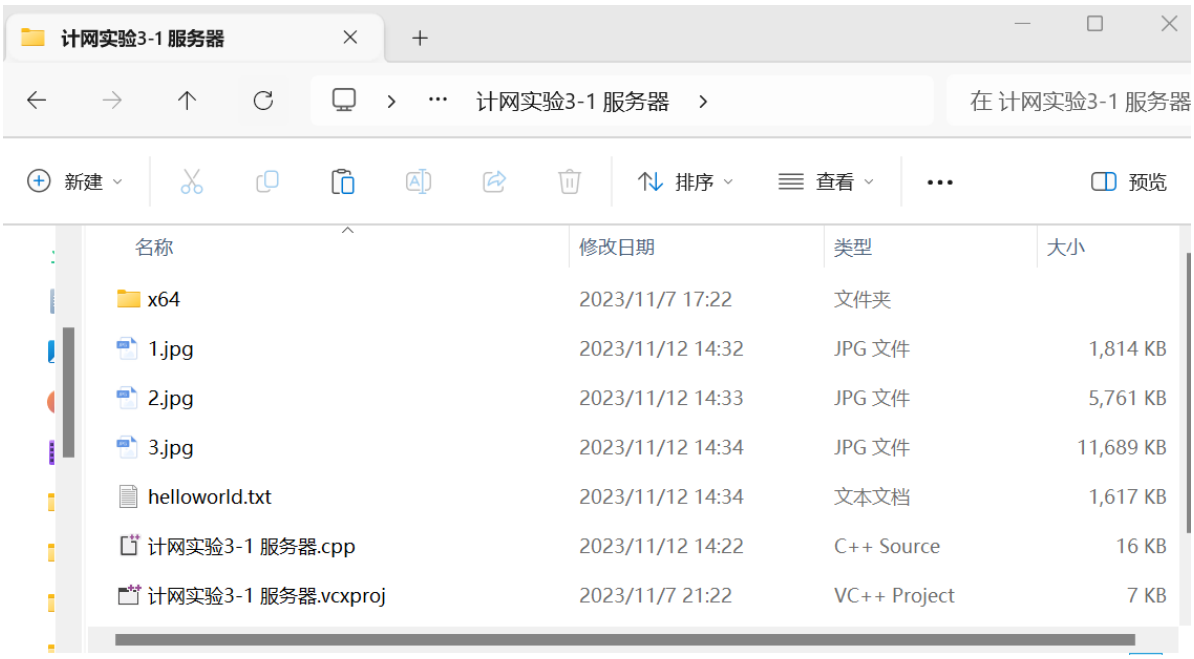
**路由器日志：**



**文件传输过程中**：

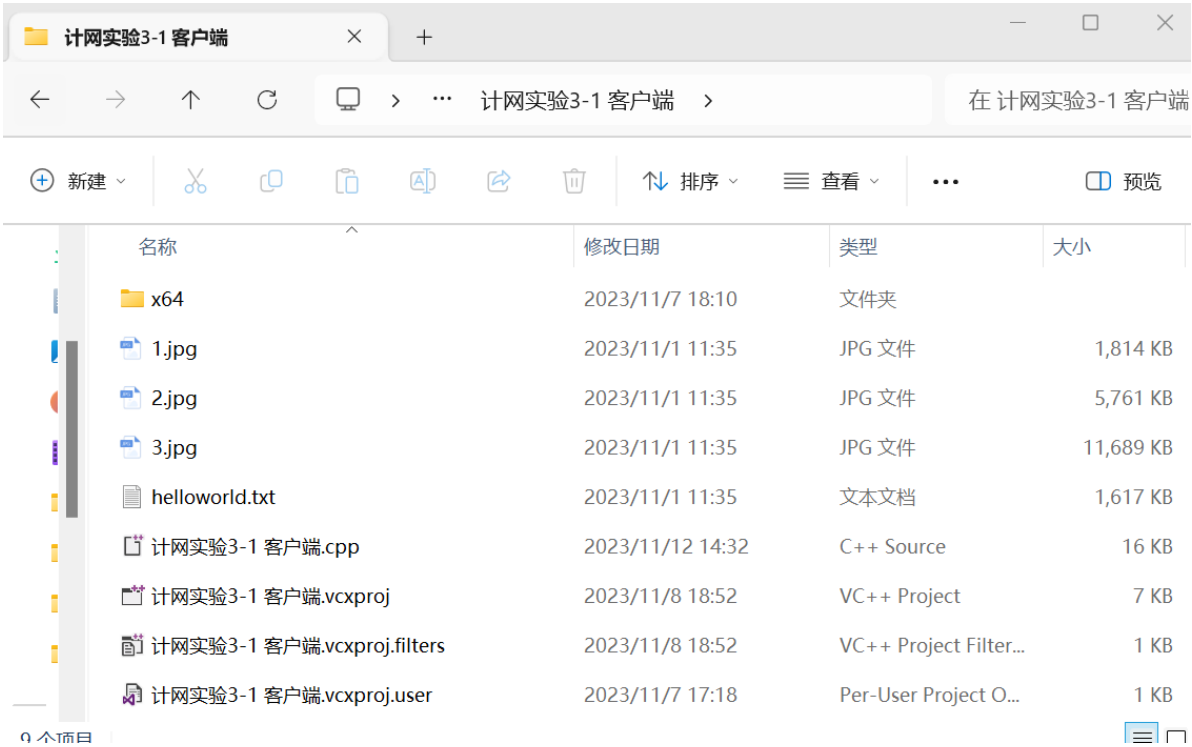
| | | | |
|---|---|---|---|
| x64 | 2023/11/7 17:22 | 文件夹 | |
| 计网实验3-1 服务器.cpp | 2023/11/10 17:58 | C++ Source | 16 KB |
| 计网实验3-1 服务器.vcxproj | 2023/11/7 21:22 | VC++ Project | 7 KB |
| 计网实验3-1 服务器.vcxproj.filters | 2023/11/7 21:22 | VC++ Project Filter... | 1 KB |
| 计网实验3-1 服务器.vcxproj.user | 2023/11/7 17:17 | Per-User Project O... | 1 KB |
| 1.jpg | 2023/11/12 14:22 | JPG 文件 | 1,814 KB |
| 2.jpg | 2023/11/12 14:22 | JPG 文件 | 0 KB |

**文件传输完毕**：




| 名称 | 修改日期 | 类型 | 大小 |
|---|---|---|---|
| x64 | 2023/11/7 17:22 | 文件夹 | |
| 1.jpg | 2023/11/12 14:32 | JPG 文件 | 1,814 KB |
| 2.jpg | 2023/11/12 14:33 | JPG 文件 | 5,761 KB |
| 3.jpg | 2023/11/12 14:34 | JPG 文件 | 11,689 KB |
| helloworld.txt | 2023/11/12 14:34 | 文本文档 | 1,617 KB |
| 计网实验3-1 服务器.cpp | 2023/11/12 14:22 | C++ Source | 16 KB |
| 计网实验3-1 服务器.vcxproj | 2023/11/7 21:22 | VC++ Project | 7 KB |

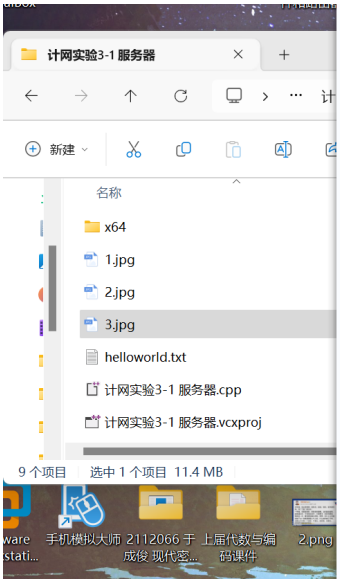
和客户端的文件对比：




| 名称 | 修改日期 | 类型 | 大小 |
|---|---|---|---|
| x64 | 2023/11/7 18:10 | 文件夹 | |
| 1.jpg | 2023/11/1 11:35 | JPG 文件 | 1,814 KB |
| 2.jpg | 2023/11/1 11:35 | JPG 文件 | 5,761 KB |
| 3.jpg | 2023/11/1 11:35 | JPG 文件 | 11,689 KB |
| helloworld.txt | 2023/11/1 11:35 | 文本文档 | 1,617 KB |
| 计网实验3-1 客户端.cpp | 2023/11/12 14:32 | C++ Source | 16 KB |
| 计网实验3-1 客户端.vcxproj | 2023/11/8 18:52 | VC++ Project | 7 KB |
| 计网实验3-1 客户端.vcxproj.filters | 2023/11/8 18:52 | VC++ Project Filter... | 1 KB |
| 计网实验3-1 客户端.vcxproj.user | 2023/11/7 17:18 | Per-User Project O... | 1 KB |

9 个项目

**文件大小一致，且可以正常打开：**




## 五次挥手断开连接：

发送端：

```
发送数据报：  Seq:  30   ACK:  0  FIN:   1
  校验和:  65504    数据长度:   0
第一次挥手成功
接收数据报：  Seq:  27   ACK:  1  FIN:   0
  校验和:  0    数据长度:   0
第二次挥手成功!
接收数据报：  Seq:  97   ACK:  1  FIN:   1
  校验和:  0    数据长度:   0
第三次挥手成功
发送数据报：  Seq:  31   ACK:  1  FIN:   0
  校验和:  65405    数据长度:   0
第四次挥手成功
第五次挥手成功
已断开连接
```

接收端：

```
接收数据报:  Seq:  30   ACK:  0  FIN:   1
  校验和:  0    数据长度:   0
第一次挥手成功
发送数据包:  Seq:  27   ACK:  1  FIN:   0
  校验和:  65476    数据长度:   0
第二次挥手成功
发送数据包:  Seq:  97   ACK:  1  FIN:   1
  校验和:  65405    数据长度:   0
第三次挥手成功
接收数据报:  Seq:  31   ACK:  1  FIN:   0
  校验和:  0    数据长度:   0
第四次挥手成功
第五次挥手成功
已断开连接
```