LAB3：基于UDP服务的可靠传输协议

- 姓名：卢麒萱
- 学号：2010519
- 专业：计算机科学与技术

实验要求

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。

协议设计

数据包及其首部

本协议定义了两种数据包，分别为携带标志信息的数据包 `SignPackage` 和携带文件的数据包 `DataPackage`。其中，携带标志信息的数据包 `SignPackage` 中包含一个 `label` 标志位和一个 `serialNumber` 序列号。该类数据包主要用于三次握手过程以及 `Server` 服务器端对于客户端的应答。同时，预设了7个标志位，分别为第一次握手、第二次握手、第三次握手、文件名传输、文件传输、传输出错、传输结束。携带文件的数据包。 `DataPackage` 主要用于文件数据的传输。它包括一个 `label` 标志位、一个 `serialNumber` 序列号、一个 `length` 长度位、一个用于承装数据的 `char` 型数组 `data` 以及 一个 `checkSum` 校验和。

建立连接过程

本协议模仿三次握手建立连接。具体流程为：客户端向服务器端发送一个 `SignPackage` 数据包，其 `label` 为第一次握手；服务器端收到数据包后回发一个 `SignPackage` 数据包，其 `label` 为第二次握手；客户端收到数据包后再回发一个 `SignPackage` 数据包，其 `label` 为第三次握手。当服务器端收到第三个数据包是代表三次握手成功，连接建立。其中，当客户端超时未收到第二个数据包，会重发第一个数据包；当服务器端超时未收到第三个数据包，会重发第二个数据包。

差错检测机制

本协议在传输文件，即发送 `DataPackage` 数据包时采用差错检测机制。 `DataPackage` 数据包首部有一个 `checkSum` 校验和位，在客户端发送数据包之前，会为其添加包头，然后计算这个数据包的校验和，其具体方法为进行 `16` 位二进制反码求和运算，若有进位则加到末位，计算结果取反写入校验和域段，之后发送该数据包。服务器端在接受到数据包后按照同样的方法计算除校验和位外的其他所有数据的校验和，并将计算结果与数据包中的校验和位进行比较，看是否一致，若一致，则可认为传输未出错，反之则证明传输出错，应重传。

确认重传机制

本协议在发送数据包的同时开启一个定时器，若是在一定时间内没有收到发送数据的 `ACK` 确认数据包，则对该数据包进行重传，在达到一定次数（此处设置为 `50`）还没有成功时放弃，退出程序并发送一个错误信号给对方，对方收到这个错误信号后也直接退出。

停等机制

本协议规定数据包的发送与接收遵循停等机制，即发送方发送数据包后，等待对方回送数据包，接收方在接收到一个数据包后，发送回一个应答信号给接收方，发送方如果没有收到应答信号则必须等待，超出一定时间后（此处设置为 `200ms`）启动重传机制。

代码分析

服务器端

宏定义

程序首先进行了一些宏定义以便于后续的编程，例如定义了端口号、发送数据的最大字节数、文件名长度的最大值、第一次握手标志位、第二次握手标志位、第三次握手标志位、以及一些文件传输过程中的标志位。

```
#define FIRST_HAND_SHAKE 1//第一次握手
#define SECOND_HAND_SHAKE 2//第二次握手
#define THIRD_HAND_SHAKE 3//第三次握手
#define FILE_NAME 4//文件名传输
#define SEND_DATA 5//文件传输
#define SEND_ERROR 6//传输出错
#define SEND_FINISH 7//传输结束
#define MAX_TEST_NUM 50//最大重发次数
#define MAX_DELAY_TIME 200//最大超时时间
#define PORT 1234
#define MAX_BUFFER 1024
#define MAX_NAME 1024
```

数据结构定义

网络中传输的数据包是经过封装的，每一次封装都会增加相应的首部。因此需要定义数据包的首部。在我设计的协议中，有两种数据包的格式，分别为携带标志信息的数据包和携带文件的数据包。

- `SignPackage`**结构**

  `SignPackage`结构用于三次握手过程以及服务器端向客户端发送确认信息。它包括一个`char`类型的`label`标志位以及一个`int`类型的`serialNumber`序列号。为了便于数据包的发送和接收，使用了`union`联合类型将它们用一个`char`类型的数组进行表示。

  ```
  //用于三次握手以及ACK的数据包
  struct SignPackage {
      union {
          struct {
              char lable;
              int serialNumber;
          } pac;
          char str[5];
      };
  };
  ```

- `DataPackage`**结构**

  `DataPackage`结构用来发送文件数据。它包括一个2字节的`label`位，用来表示发送的数据包类型（例如文件名或文件结束等）；一个2字节的`serialNumber`位，用来表示序列号，在本实验中序列号只需要两个（0和1）即可；一个2字节的`length`位，用于表示传输数据的长度；一个`MAX_BUFFER`大小的`char`类型数组，用于承装数据；以及一个2字节的`checkSum`位，用来表示校验和。同样的，此处使用了`union`联合类型，以便于后续的文件发送、接收、以及校验和的赋值等。

  ```
  //发送文件数据的UDP包
  ```

```
struct DataPackage {
    union {
        struct {
            u_short lable;//标志位，储存部分上面的宏定义
            u_short serialNumber;//序列号
            u_short length;//数据段长度
            char data[MAX_BUFFER];//数据
            u_short checkSum;//校验和
        } pac;
        char str[4 * sizeof(u_short) + MAX_BUFFER];//用于sendto和recvfrom函数
        struct {
            u_short checkContent[3 + MAX_BUFFER / (sizeof(u_short))];//用于校验
            u_short checkSum;
        } useToCheck;
    };
};
```

校验和检测函数

在服务器端需要重新计算校验和并且将结果与数据包中的校验和进行比较，若相等，说明通过了校验和的检测。具体检验的过程就是借助于 `union` 联合定义的 `useToCheck` 结构，其中 `checkContent` 数组中的值为 `u_short` 类型，16位，每次取数组中的一个值，对齐反码做加法运算，若有进位则加到最低位上去，将最后得到的结果取反就是校验和的值。

```
#define CHECK_COUNT (sizeof(DataPackage) / 2 - 1)//用于校验和检验

//检测校验和函数
bool checkSum(DataPackage package) {
    register u_long sum = 0;
    for (int i = 0; i < CHECK_COUNT; i++) {
        sum += package.useToCheck.checkContent[i];
        if (sum & 0XFFFF0000)
        {
            sum &= 0XFFFF;
            sum++;
        }
    }
    return (package.useToCheck.checkSum == (u_short)~(sum & 0XFFFF));
}
```

启动 `winsock`

在 `Windows` 上进行 `Socket` 编程，首先必须启动 `winsock`，此处引入动态链接库 `ws2_32.lib`。

```
#include <WinSock2.h>//socket通信，系统头文件
#pragma comment(lib,"ws2_32.lib")//加载链接库文件，实现通信程序的管理
```

首先调用 `WSAStartup` 函数，这个函数是连接应用程序与 `ws2_32.dll` 的第一个调用。其中，第一个参数是 `winsock` 版本号，第二个参数是指向 `wsaData` 的指针。该函数返回一个 `INT` 型值，通过检查这个值来确定初始化是否成功。

在结束调用后，则可以通过 `WSACleanup()` 来中止 `ws2_32.lib` 的使用。

```
//初始化SOCKET DLL
WSADATA WsaData;//WSADATA变量
iResult = WSAStartup(MAKEWORD(2, 2), &WsaData);
if (iResult != 0)
{
    cout << "WSAStartup failed with error: " << iResult << endl;
    system("pause");
    return -1;
}
```

创建套接字、获取本机 IP 地址

首先初始化了一个 `Socket`，并且通过相关字段指明该 `Socket` 工作基于 `UDP （ AF_INET , SOCK_DGRAM ）` 。

```
//创建SOCKET
ServerSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (ServerSocket == INVALID_SOCKET)
{
    cout << "Socket failed with error:" << WSAGetLastError() << endl;
    system("pause");
    WSACleanup();
    return -1;
}
```

为了实现网络程序间的交互，服务器端首先需要获取自身的 IP 地址以填充 `Socket`，然后完成服务器端地址的填充，其中包括协议族，端口和 IP 地址三个内容。值得注意的是在 `sockaddr()` 结构体的赋值过程中， `sin_family` 字段是唯一没有进行网络字节顺序和主机字节顺序转换的，其原因在于，该字段仅是用于指导协议层和网络层进行填充而不同于 `sin_port` 和 `sin_addr` 字段需要写入消息和报文中进行传输，因此不必进行字节顺序的转换。

```
//服务器套接字地址
ServerAddr.sin_family = AF_INET;
ServerAddr.sin_port = htons(PORT);
ServerAddr.sin_addr.S_un.S_addr = INADDR_ANY;
```

绑定服务器

通过 `bind()` 函数实现。

```
iResult = bind(ServerSocket, (sockaddr*)& ServerAddr, sizeof(ServerAddr));
if (iResult == SOCKET_ERROR)
{
    cout << "Bind Failed With Error:" << WSAGetLastError() << endl;
    system("pause");
    closesocket(ServerSocket);//关闭套接字
    WSACleanup();//释放套接字资源;
    return -1;
}
```

创建连接

此时，服务器端就一直接收消息，若判断接收到的是第一次握手的消息，则创建一个线程，用于与该客户端交互。

```
sockaddr_in RemoteAddr;
int nAddrLen = sizeof(RemoteAddr);

DataPackage dataPackage;
SignPackage signPackage;

while (true)
{
    recvfrom(ServerSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, &nAddrLen);
    if (signPackage.pac.lable == FIRST_HAND_SHAKE)
    {
        cout << "receive first handshake." << endl;

        HANDLE hThread = CreateThread(NULL, 0, ClientThread, &RemoteAddr, 0, NULL);
        CloseHandle(hThread);
    }
}
```

三次握手

在新创建的线程中，首先创建服务器套接字、进行服务器端口绑定，并尝试绑定新的端口。由于我们在主线程已经正确接收了第一次握手数据包，此时需要回复一个第二次握手的数据包。将 `signPackage` 中的 `label` 设置为第二次握手的标志位 `SECOND_HAND_SHAKE`，并发送给客户端。接下来启动一个定时器，然后在一个循环里接收第三次握手，如果正确接收则跳出，如果超时未接收到，则重发第二次握手，并且重新设置定时器。当第三次握手的数据包已正确接收，此时可以认为连接建立成功。

```
int ret;
clock_t start, end;
timeval tv = { 5, 0 };
setsockopt(ClientSocket, SOL_SOCKET, SO_RCVTIMEO, (char*)& tv, sizeof(tv));

//第二次握手
signPackage.pac.lable = SECOND_HAND_SHAKE;
sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
int testNum = 1;
cout << "The second handshake has been sent." << endl;

    //第三次握手
start = clock();

while (true) {
    end = clock();
    ret = recvfrom(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, &nAddrLen);
    if (signPackage.pac.lable == THIRD_HAND_SHAKE)
        break;
    if (testNum > MAX_TEST_NUM) {
```

```
        cout << "Transmission error, press any key to exit." << endl;
        signPackage.pac.lable = SEND_ERROR;
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        system("pause");
        return -1;
    }
    if ((end - start) > 20) {
        signPackage.pac.lable = SECOND_HAND_SHAKE;
        signPackage.pac.serialNumber = 0;
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);//第二次握手重发
        cout << "The second handshake has been retransmitted." << endl;
        testNum++;
        start = clock();
    }
}

cout << "receive third handshake." << endl;

//连接建立成功
cout << "Client Connect Success!\n" << "Client:   Ip:" << inet_ntoa(RemoteAddr.sin_addr) << " ; Port:" << ntohs(RemoteAddr.sin_port) << endl;
```

接收文件名

在一个 `while` 循环中，可以不断地先接收文件名，再接收文件。

首先接收文件名，接收文件名时需要确认 `dataPackage` 中的 `lable` 标签是 `FILE_NAME` ，而且要确认校验和是否正确，若正确接收到了文件名，则回发一个数据包，该数据包的 `lable` 标签同样是 `FILE_NAME`。

```
char file_name[MAX_NAME];
int ret = 0;
int testNum = 0;
memset(dataPackage.str, 0x00, sizeof(dataPackage.str));
//接收文件名
while (true) {
    ret = recvfrom(ClientSocket, dataPackage.str, sizeof(dataPackage.str), 0, (sockaddr*)& RemoteAddr, &nAddrLen);
    if (ret != SOCKET_ERROR && dataPackage.pac.lable == FILE_NAME && checkSum(dataPackage)) {
        strcpy(file_name, dataPackage.pac.data);
        file_name[dataPackage.pac.length] = '\0';
        cout << "Received file name:" << file_name << endl;
        cout << "Send confirmation message.\n";
        signPackage.pac.lable = FILE_NAME;
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        testNum++;
```

```
        break;
    }
    if (testNum == 100 || dataPackage.pac.lable == SEND_ERROR) {
        cout << "Transmission error, press any key to exit." << endl;
        signPackage.pac.lable = SEND_ERROR;
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        system("pause");
        return -1;
    }
    Sleep(200);
}
```

接收文件

首先尝试在本地对应的目录下以可写的方式打开文件，如果文件打开失败，返回错误信息，并发送一个标志位为 `SEND_ERROR` 的数据包，告诉客户端出错。如果正确打开文件，就在一个 `while` 循环中不断接收数据包，在接收的时候需要判断序列号是否正确，标志位是否正确，校验和是否正确，如果都正确则回发一个数据包告诉客户端发送成功，然后将读到的内容写入文件。

```
//打开文件
FILE* fp;
if (!(fp = fopen(file_name, "wb")))
{
    cout << "File: " << file_name << " Can't Open" << endl;
    signPackage.pac.lable = SEND_ERROR;
    sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
    sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
    sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
    return -1;
}
memset(RecvBuffer, 0, MAX_BUFFER);
//接收文件
int ret1;
int ret2;
memset(dataPackage.str, 0, sizeof(dataPackage.str));
testNum = 0;
signPackage.pac.serialNumber = 0;
signPackage.pac.lable = SEND_DATA;
while (true)
{
    ret1 = recvfrom(ClientSocket, dataPackage.str, sizeof(dataPackage.str), 0, (sockaddr*)& RemoteAddr, &nAddrLen);
    if (testNum == 100 || dataPackage.pac.lable == SEND_ERROR) {//传送出错
        cout << "Transmission error, press any key to exit." << endl;
        signPackage.pac.lable = SEND_ERROR;
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
```

```
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        system("pause");
        return -1;
    }
    if (ret1 != SOCKET_ERROR && dataPackage.pac.lable == SEND_FINISH) {//传送完毕
        signPackage.pac.lable = SEND_FINISH;
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        cout << "Receive: " << file_name << " Successful!" << endl;
        cout << "File received successfully.\n";
        break;
    }
    if (ret1 == SOCKET_ERROR || dataPackage.pac.serialNumber != signPackage.pac.serialNumber || !checkSum(dataPackage)) {//传输出问题
        ret2 = sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        testNum++;
        Sleep(200);
        continue;
    }
    while (dataPackage.pac.lable == SEND_DATA) {
        ret2 = sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
        ret = fwrite(dataPackage.pac.data, sizeof(char), dataPackage.pac.length, fp);//写数据
        signPackage.pac.serialNumber = signPackage.pac.serialNumber ? 0 : 1;
        if (ret < dataPackage.pac.length) {
            cout << file_name << "Write failed, press any key to exit." << endl;
            signPackage.pac.lable = SEND_ERROR;
            sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
            sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
            sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
            system("pause");
            return -1;
        }
        if (ret2 != SOCKET_ERROR) {
            break;
        }
        if (testNum++ == 100) {
            cout << "Transmission error, press any key to exit." << endl;
            signPackage.pac.lable = SEND_ERROR;
            sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
```

```
            sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
            sendto(ClientSocket, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& RemoteAddr, nAddrLen);
            system("pause");
            return -1;
        }
    }
    memset(dataPackage.str, 0, sizeof(dataPackage.str));
}
```

客户端

客户端的宏定义和数据结构的定义和服务器端类似，其启动 `Winsock`、初始化套接字的过程也与服务器端类似，此处不再赘述。

三次握手

在创建完 `Socket` 之后，客户端向服务器端发送一个数据包，这个数据包的标志位为 `FIRST_HAND_SHAKE` ，表示第一次握手，然后启动一个定时器，在 `while` 循环中接收服务器端发送的第二次握手的数据包。如果正确接收，则跳出循环，继续发送第三次握手的数据包，如果超时未接收到正确的数据包，则重新发送第一次握手的数据包，并且重新开启定时器。当第三次握手成功发送之后，表示客户端已经准备好，可以发送文件了。

```
//第一次握手
signPackage.pac.lable = FIRST_HAND_SHAKE;
signPackage.pac.serialNumber = 0;
sendto(sclient, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& sin, len);
int testNum = 1;
cout << "The first handshake has been sent." << endl;

start = clock();
//第二次握手
while (true) {
    end = clock();
    recvfrom(sclient, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& sin, &len);
    if (signPackage.pac.lable == SECOND_HAND_SHAKE) {
        cout << "The second handshake has been received." << endl;
        break;
    }
    if (testNum > MAX_TEST_NUM) {
        cout << "Transmission error, press any key to exit." << endl;
        signPackage.pac.lable = SEND_ERROR;
        sendto(sclient, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& sin, len);
        sendto(sclient, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& sin, len);
        sendto(sclient, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)& sin, len);
        system("pause");
        return -1;
    }
    if ((end - start) > 20) {
```

```
        signPackage.pac.lable = FIRST_HAND_SHAKE;
        signPackage.pac.serialNumber = 0;
        sendto(sclient, signPackage.str, sizeof(signPackage.str), 0,
(sockaddr*)& sin, len);
        cout << "The first handshake has been retransmitted." << endl;
        testNum++;
        start = clock();
    }
}

//第三次握手
signPackage.pac.lable = THIRD_HAND_SHAKE;
sendto(sclient, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)&
sin, len);
sendto(sclient, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)&
sin, len);
sendto(sclient, signPackage.str, sizeof(signPackage.str), 0, (sockaddr*)&
sin, len);
cout << "The third handshake has been sent." << endl;

cout << "Client OK!" << endl;
```

发送文件名

在建立连接之后，先发送文件名。用户输入文件名，若在对应的目录下不存在这个文件，则报错，重新输入文件名，当文件名合法，就给服务器端发送数据包，这个数据包的 `label` 为标志位 `FILE_NAME` ，然后客户端就在一个循环中等待服务器端返回确认数据包。若超时未收到，则重发文件名的数据包。

```
FILE* fp;
char file_name[MAX_PATH];

cout << "Please Input The Filename:" << endl;
cin >> file_name;
//传送文件名
if (!(fp = fopen(file_name, "rb")))
{
    cout << "File " << file_name << " Can't Open" << endl;
    continue;
}

memset(dataPackage.str, 0x00, sizeof(dataPackage.str));
dataPackage.pac.lable = FILE_NAME;
dataPackage.pac.length = strlen(file_name);
strcpy(dataPackage.pac.data, file_name);
checkSum(&dataPackage);
int testNum = 0;//记录失败次数

sendto(sclient, dataPackage.str, sizeof(dataPackage.str), 0, (sockaddr*)&
sin, len);
testNum++;
cout << "File name has been sent!" << endl;
start = clock();

while (true) {
    end = clock();
```

```
    ret = recvfrom(sclient, signPackage.str, sizeof(signPackage), 0,
(sockaddr*)& sin, &len);
    if (testNum == 100 || signPackage.pac.lable == SEND_ERROR) {
        cout << "Transmission error, press any key to exit." << endl;
        signPackage.pac.lable = SEND_ERROR;
        sendto(sclient, signPackage.str, sizeof(signPackage.str), 0,
(sockaddr*)& sin, len);
        sendto(sclient, signPackage.str, sizeof(signPackage.str), 0,
(sockaddr*)& sin, len);
        sendto(sclient, signPackage.str, sizeof(signPackage.str), 0,
(sockaddr*)& sin, len);
        system("pause");
        return -1;
    }
    if (ret != SOCKET_ERROR && signPackage.pac.lable == FILE_NAME) {
        cout << "server has confirmed to receive the file name." << endl;
        break;
    }
    if ((end - start) > 200) {
        sendto(sclient, dataPackage.str, sizeof(dataPackage.str), 0,
(sockaddr*)& sin, len);
        testNum++;
        cout << "File name has been sent!" << endl;
        start = clock();
    }
}
```

发送文件

在收到文件名成功发送的确认信息后，便循环读入文件，每个读入`1024`个字节并封装成数据包，将该数据包的`label`设置为`SEND_DATA`标志位。然后设置序列号`serialNumber`，使之在0和1之间反复。在发送该数据包后，开启定时器，然后循环接收确认数据包，接收时要检查校验和以及序列号。若超时未接收到数据包，则重新发送。在接收到确认数据包后，客户端才进行下一个数据包的发送。如此循环，直到文件读取结束。这时，客户端给服务器端发送一个空包，其`label`设置为`SEND_FINISH`标志位。 同样的，开启定时器，循环接收服务器端发送的确认数据包，若超时未收到，则重发数据包，若正确收到，则代表这个文件发送完毕。即可进行下一个文件的发送。

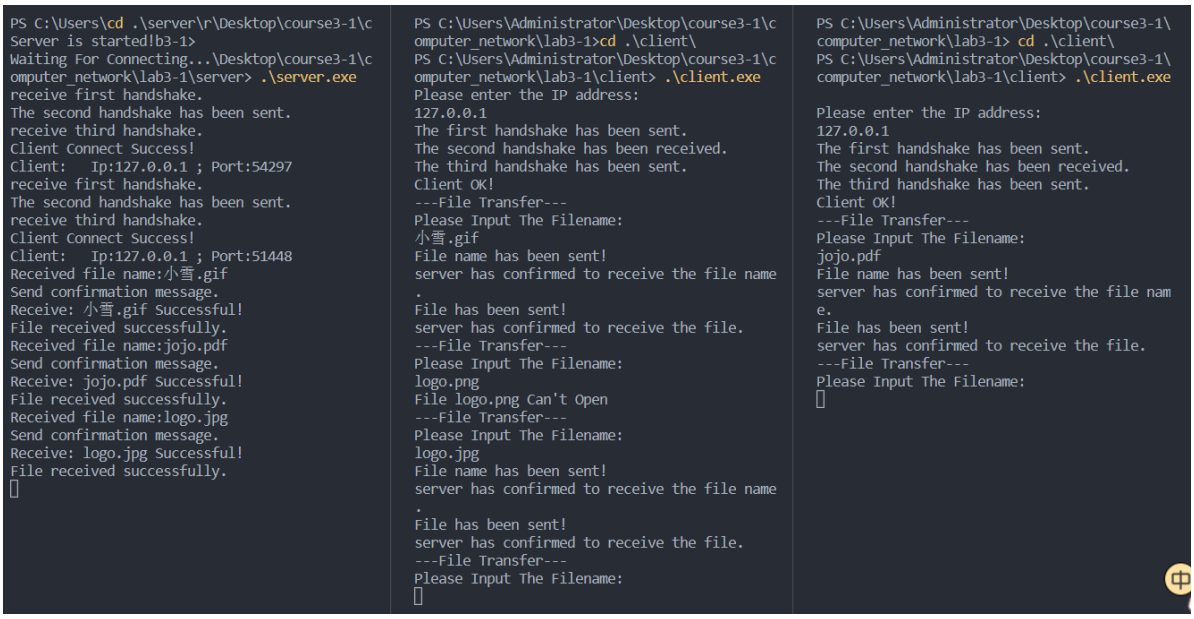
```
//传送文件
int length;
testNum = 0;
memset(dataPackage.str, 0x00, sizeof(dataPackage.str));
int serialNumber = 1;
dataPackage.pac.serialNumber = serialNumber;
while ((length = fread(dataPackage.pac.data, sizeof(char),
sizeof(dataPackage.pac.data), fp)) > 0)
{
    testNum = 0;
    dataPackage.pac.length = length;
    dataPackage.pac.lable = SEND_DATA;
    serialNumber = serialNumber ? 0 : 1;
    dataPackage.pac.serialNumber = serialNumber;
    checkSum(&dataPackage);

```

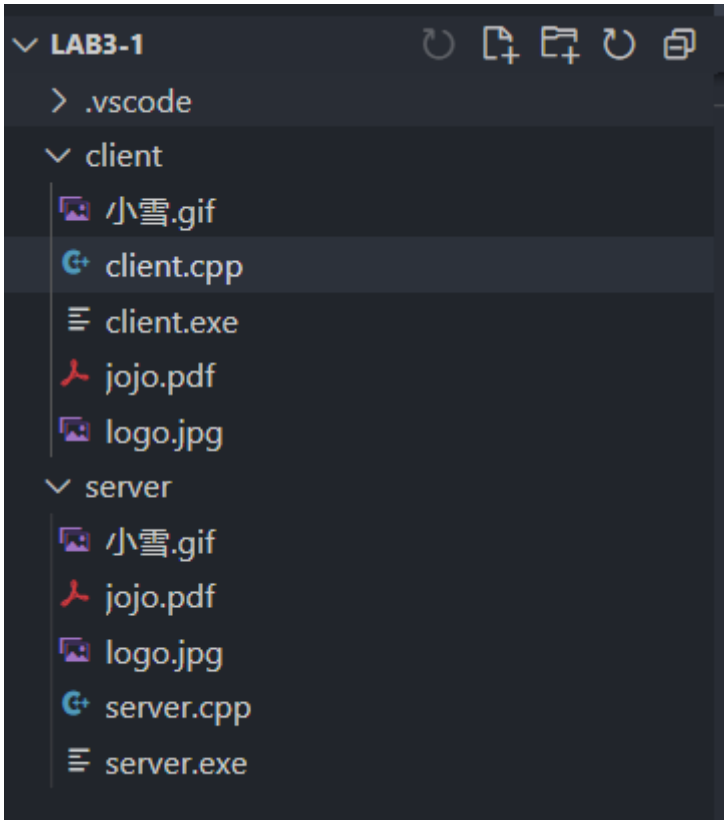
```
    ret = sendto(sclient, dataPackage.str, sizeof(dataPackage), 0,
(sockaddr*)& sin, len);
    testNum++;
    start = clock();

    while (true) {
        end = clock();
        ret = recvfrom(sclient, signPackage.str, sizeof(signPackage), 0,
(sockaddr*)& sin, &len);
        if (testNum == 100 || signPackage.pac.lable == SEND_ERROR) {
            cout << "Transmission error, press any key to exit." << endl;
            signPackage.pac.lable = SEND_ERROR;
            sendto(sclient, signPackage.str, sizeof(signPackage), 0,
(sockaddr*)& sin, len);
            sendto(sclient, signPackage.str, sizeof(signPackage), 0,
(sockaddr*)& sin, len);
            sendto(sclient, signPackage.str, sizeof(signPackage), 0,
(sockaddr*)& sin, len);
            system("pause");
            return -1;
        }
        if (ret != SOCKET_ERROR && signPackage.pac.lable == SEND_DATA &&
signPackage.pac.serialNumber == dataPackage.pac.serialNumber) {
            break;
        }
        if ((end - start) > 200) {
            ret = sendto(sclient, dataPackage.str, sizeof(dataPackage), 0,
(sockaddr*)& sin, len);
            testNum++;
            start = clock();
        }
    }
    memset(dataPackage.str, 0x00, sizeof(dataPackage.str));
}

testNum = 0;
memset(dataPackage.str, 0x00, sizeof(dataPackage.str));
dataPackage.pac.lable = SEND_FINISH;
checkSum(&dataPackage);

ret = sendto(sclient, dataPackage.str, sizeof(dataPackage), 0, (sockaddr*)&
sin, len);
testNum++;
cout << "File has been sent!" << endl;
start = clock();

while (true) {
    end = clock();
    ret = recvfrom(sclient, signPackage.str, sizeof(signPackage), 0,
(sockaddr*)& sin, &len);
    if (testNum == 100 || signPackage.pac.lable == SEND_ERROR) {
        cout << "Transmission error, press any key to exit." << endl;
        signPackage.pac.lable = SEND_ERROR;
        sendto(sclient, signPackage.str, sizeof(signPackage), 0,
(sockaddr*)& sin, len);
```

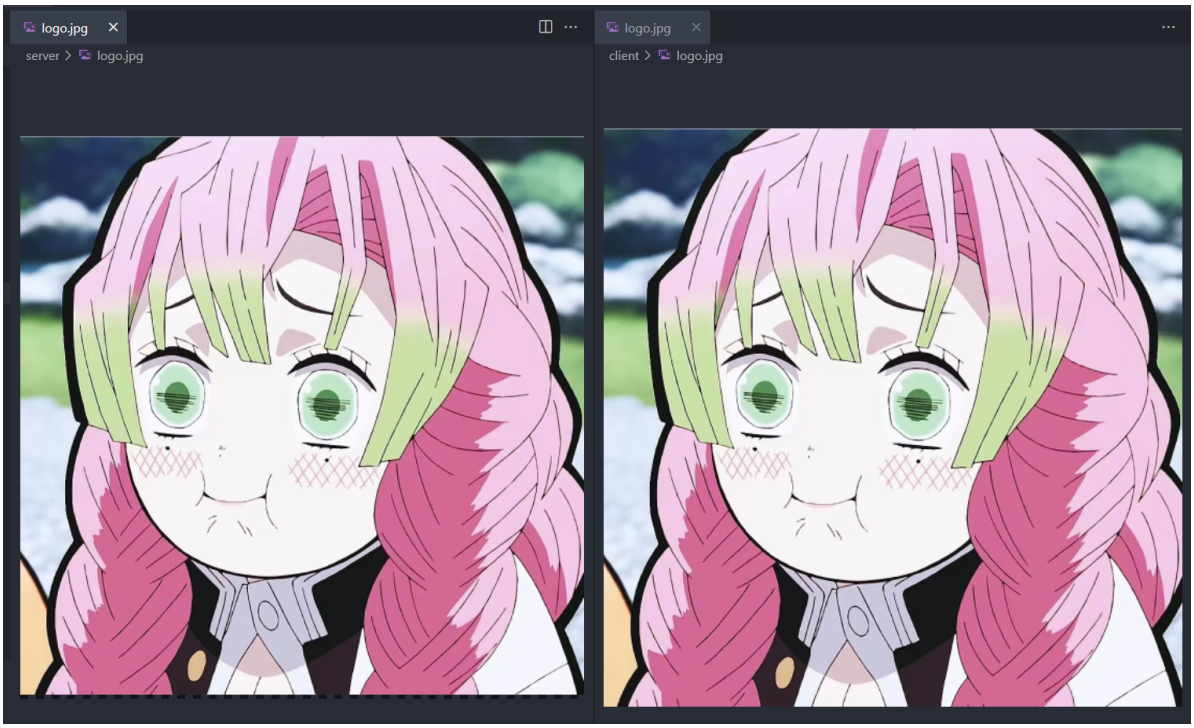
```
        sendto(sclient, signPackage.str, sizeof(signPackage), 0, (sockaddr*)& sin, len);
        sendto(sclient, signPackage.str, sizeof(signPackage), 0, (sockaddr*)& sin, len);
        system("pause");
        return -1;
    }
    if (ret != SOCKET_ERROR && signPackage.pac.lable == SEND_FINISH) {
        cout << "server has confirmed to receive the file." << endl;
        break;
    }
    if ((end - start) > 200) {
        ret = sendto(sclient, dataPackage.str, sizeof(dataPackage), 0, (sockaddr*)& sin, len);
        testNum++;
        cout << "File has been sent!" << endl;
        start = clock();
    }
}
fclose(fp);
```

实验结果

同时启动服务器端与客户端程序，然后依次发送4个测试文件，可以发现服务器端正确接收到了这4个文件，通过观察文件大小以及文件内容，可以看到未发生数据丢失。




LAB3-1
.vscode
client
小雪.gif
client.cpp
client.exe
jojo.pdf
logo.jpg
server
小雪.gif
jojo.pdf
logo.jpg
server.cpp
server.exe


logo.jpg
server > logo.jpg
logo.jpg
client > logo.jpg