**计算机网络编程**

**实验报告**

**班级：07111707**

**组长：1120171189 崔程远**

**成员：1120172149 吴沁璇**

**1120172153 张澈**

**1120172163 王晓媛**

**1120172736 张鉴昊**

**1120172765 曾煜瑾**

**1120173326 曾紫飞**

**北京理工大学**

**计算机学院**

**2020年5月**

**第三章 实验3 基于停止等待协议的可靠通信**

**1. 实验目的**

模拟数据链路层发送数据的停止等待协议，训练编程能力

1. **实验内容**

采用UDP Socket编程接口作为模拟物理层接口实现帧的发送和接收，协议采用单工方式进行数据通信。假设Host1要向Host2发送大文件，通过数据链路层的帧每次完成数据块的可靠传输，采用停止等待协议，差错编码采用CRC-CCITT标准。以教材协议3为基础，在帧末尾增加CRC校验字段。

发送程序配置文件关键要点：

数据传输目的UDP端口

UDPPort=8888

增添发送过滤程序，模拟传输出错或丢数据帧，下面两项指明每发送多少帧出现一次出错或丢帧，此例表示每10帧中一帧出错，每10帧中一帧丢失

FilterError=10

FilterLost=10

发送程序运行屏幕输出关键要点：

显示next\_frame\_to\_send变量的值，以及正在发送帧的编号

显示经过过滤器后是正确发送、模拟传输出错还是模拟帧丢失（实际没有发送）

显示接收到确认帧，确认帧的确认序号

或者显示超时

回到开始重复一直到文件发送完成

接收程序运行屏幕输出关键要点：

显示frame\_expected变量的值，

接收帧是否出错（CRC余数是否为零），正确则显示接收帧的发送帧序号

显示发送回确认帧，以及确认帧的确认序号

回到开始重复一直到文件接收完成

1. **实验原理**

这是单工传输，即只有一个方向的通信，一端负责发，另一端负责收。发送端发送的数据帧中加上seq，在停止等待协议中01交替。接收端每收到一个帧，把帧中seq的值和ack\_expected比较，如果相等，表示帧的序号没有问题，另外还需要根据数据和CRC码计算余数，如果余数为0，表示帧在传输中没有出错。以上两点若满足，接收端发送回确认帧，接收端在超时间隔内收到确认帧，继续发送下一帧，如果在超时间隔内没有收到确认帧，重复发送此帧。

1. **实验环境**

|  |  |  |
| --- | --- | --- |
| 语言 | 集成开发环境 | 编译器 |
| C++ | Visual Studio 2017 | gcc version 4.8.1 |
| Java | Eclipse 2019 | java version "1.8.0\_65" |
| Python | Pycharm 2017 | Python 3.7.0 |

1. **实验步骤**

三份代码的整体结构基本一致，输出格式完全一致，均采用面向对象的方法，下面以C++代码为例进行分析。

**一、发送端**

·定义全局变量

SOCKET Sock;

SOCKADDR\_IN receiverAddress;

int receiverPort = 8888; //接收端端口号

int dataLen = 20; //数据帧长度

int sendFrameLen = 40; //发送帧长度

int receiveFrameLen = 1; //接收帧长度

int serialPos = 0; //序列号位置

int dataStartPos = 1; //数据帧起始位置

int validDataLenPos = 21; //有效数据长度的位置

int crcStartPos = 22; //16位CRC校验码的起始位置

int crcLen = 16; //CRC检验码的长度

int isEndPos = 38; //文件是否发送完毕的标识位置

int nextFrameToSend = 0;

long seq = 0; //数据帧的编号

long filterSeq = 0; //发送帧的编号

int filterError = 10; //每10帧1帧出错

int filterLost = 10; //每10帧1帧丢失

int firstError = 3; //第一个出错发送帧的编号

int firstLost = 8; //第一个丢失发送帧的编号

//三种处理模式的代号

const int right = 0;

const int error = 1;

const int lost = 2;

·辅助函数

//获得单个字符的八位二进制码

string getSingleBinaryString(int a)

{

char s1[10];

\_itoa\_s(a, s1, 2);

string s2(s1);

while (s2.length() < 8)

{

s2 = "0" + s2;

}

return s2;

}

//获得字符串中每个字符的八位二进制码组合而成的二进制字符串

string getBinaryString(string source)

{

string s = "";

for (int i = 0; i < source.length(); i++)

{

s += getSingleBinaryString(int(source[i]));

}

return s;

}

·计算CRC码

//计算余数

string getRemainderStr(string dividendStr, string divisorStr)

{

int dividendLen = dividendStr.length();

int divisorLen = divisorStr.length();

for (int i = 0; i < divisorLen - 1; i++)

{

dividendStr += "0";

}

for (int i = 0; i < dividendLen; i++)

{

if (dividendStr[i] == '1')

{

dividendStr[i] = '0';

for (int j = 1; j < divisorLen; j++)

{

if (dividendStr[i + j] == divisorStr[j])

{

dividendStr[i + j] = '0';

}

else

{

dividendStr[i + j] = '1';

}

}

}

}

string remainderStr = dividendStr.substr(dividendLen, divisorLen);

return remainderStr;

}

string getCRCString(string s)

{

string gxStr = "10001000000100001";

return getRemainderStr(s, gxStr);

}

·打印函数

void printTime()

{

SYSTEMTIME time;

GetLocalTime(&time);

printf("Current time: %4d-%02d-%02d %02d:%02d:%02d\n", time.wYear, time.wMonth, time.wDay, time.wHour, time.wMinute, time.wSecond);

}

void Print(int method)

{

printTime();

cout << "next\_frame\_to\_send: " << nextFrameToSend << endl;

cout << "seq: " << seq << endl;

if (method == right)

{

cout << "Simulate sending right." << endl;

}

else if (method == error)

{

cout << "Simulate sending wrong." << endl;

}

else if (method == lost)

{

cout << "Simulate sending lost." << endl;

}

cout << endl;

}

·判断超时间隔内是否收到确认帧

//判断超时间隔内是否收到确认帧

bool waitForEvent()

{

char \*receiveFrame = new char[receiveFrameLen];

int len = sizeof(SOCKADDR);

int a = recvfrom(Sock, receiveFrame, 1024, 0, (SOCKADDR\*)&receiverAddress, &len);

if (a <= 0)

{

printTime();

cout << "Reveiving ack overtime, be ready to resend." << endl << endl;

return false;

}

printTime();

cout << "Received ack, ack is: " << receiveFrame[0] << endl << endl;

return true;

}

·Send函数

void Send()

{

//加载套接字库

WSADATA WSAdata;

WSAStartup(MAKEWORD(2, 2), &WSAdata);

//绑定端口

Sock = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

receiverAddress.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

receiverAddress.sin\_family = AF\_INET;

receiverAddress.sin\_port = htons(receiverPort);

int len = sizeof(SOCKADDR);

//设置recvfrom的接收超时为3秒

timeval tv\_out;

tv\_out.tv\_sec = 3000;

tv\_out.tv\_usec = 0;

setsockopt(Sock, SOL\_SOCKET, SO\_RCVTIMEO, (char \*)&tv\_out, sizeof(timeval));

ifstream in("D:\\desktop\\text.txt");

char \*data = new char[dataLen];

bool flag = true;

int pos = dataLen;

while (true)

{

for (int i = 0; i < dataLen; i++)

{

data[i] = '\a';

}

in.read(data, dataLen);

if (in.eof())//如果已读完文件，需要识别出读的字符个数

{

if (flag == false)//最后一个包含数据的帧收到确认帧，发送没有数据的帧，退出循环

{

char \*sendFrame = new char[sendFrameLen];

sendFrame[isEndPos] = '1';

sendto(Sock, sendFrame, strlen(sendFrame), 0, (SOCKADDR\*)&receiverAddress, len);

cout << "Send the file finished." << endl;

break;

}

pos = 0;

while (data[pos] != '\a')

{

pos++;

continue;

}

flag = false;

}

seq++;

//计算CRC校验码

string s = getBinaryString(string(data).substr(0, pos));

string crcStr = getCRCString(s);

//为发送帧赋值

char \*sendFrame = new char[sendFrameLen];

sendFrame[serialPos] = nextFrameToSend + '0';

for (int i = 0; i < dataLen; i++)

{

sendFrame[i + dataStartPos] = data[i];

}

sendFrame[validDataLenPos] = pos;

for (int i = 0; i < crcLen; i++)

{

sendFrame[i + crcStartPos] = crcStr[i];

}

sendFrame[isEndPos] = '0';

bool mark = false;

while (mark == false)

{

//模拟传输出错

if ((filterSeq - firstError) % filterError == 0)

{

char pre = sendFrame[crcStartPos];

sendFrame[crcStartPos] = '1' - pre + '0';

sendto(Sock, sendFrame, strlen(sendFrame), 0, (SOCKADDR\*)&receiverAddress, len);

Print(error);

sendFrame[crcStartPos] = pre;

filterSeq++;

}

//模拟传输丢失

else if ((filterSeq - firstLost) % filterLost == 0)

{

Print(lost);

filterSeq++;

}

//模拟传输正确

else

{

sendto(Sock, sendFrame, strlen(sendFrame), 0, (SOCKADDR\*)&receiverAddress, len);

Print(right);

filterSeq++;

}

// 调节传输速度

Sleep(1000);

mark = waitForEvent();

if (mark == true)

{

nextFrameToSend = (nextFrameToSend + 1) % 2;

}

}

}

in.close();

closesocket(Sock);

WSACleanup();

}

·主函数

int main()

{

//需要先打开接收端，否则recefrom不阻塞

string confirm;

cout << "Please open UDPReceiver.exe and input yes!" << endl;

cin >> confirm;

if (confirm.compare("yes"))

{

return 0;

}

cout << endl;

cout << "Be ready to send file..." << endl;

UDPSender operation;

operation.Send();

system("pause");

}

1. **接收端**

**·**定义全局变量

SOCKET Sock;

SOCKADDR\_IN senderAddress;

SOCKADDR\_IN receiverAddress;

int receiverPort = 8888; //接收端端口号

int dataLen = 20; //数据帧长度

int sendFrameLen = 1; //发送帧长度

int receiveFrameLen = 40; //接收帧长度

int serialPos = 0; //序列号位置

int dataStartPos = 1; //数据帧其实位置

int validDataLenPos = 21; //有效数据长度的位置

int crcStartPos = 22; //16为CRC校验码的起始位置

int crcLen = 16; //CRC检验码的长度

int isEndPos = 38; //文件是否发送完毕的标识位置

int frameExpected = 0;

·辅助函数和计算CRC函数和发送端一致

·Receive函数

void Receive()

{

//加载套接字库

WSADATA WSAdata;

WSAStartup(MAKEWORD(2, 2), &WSAdata);

//设置UDP通信的相关属性

Sock = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

receiverAddress.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

receiverAddress.sin\_family = AF\_INET;

receiverAddress.sin\_port = htons(receiverPort);

bind(Sock, (SOCKADDR\*)&receiverAddress, sizeof(SOCKADDR));

int len = sizeof(SOCKADDR);

ofstream out("D:\\desktop\\copyText.txt");

while (true)

{

char \*receiveFrame = new char[receiveFrameLen];

memset(receiveFrame, 0, sizeof(receiveFrame));

recvfrom(Sock, receiveFrame, 1024, 0, (SOCKADDR\*)&senderAddress, &len);

if (receiveFrame[isEndPos] == '1') //如果收到没有数据的帧，表示文件全部发送完毕，退出循环

{

cout << "Receive the file finished." << endl;

break;

}

int serial = receiveFrame[serialPos] - '0';

int validDataLen = receiveFrame[validDataLenPos];

//计算余数

string dataStr = getBinaryString(string(receiveFrame).substr(dataStartPos, validDataLen));

string CRC = string(receiveFrame).substr(crcStartPos, crcLen);

string crcStr = getCRCString(dataStr + CRC);

if (atoi(crcStr.c\_str()) == 0)//数据正确

{

printTime();

cout << "frame\_expected: " << frameExpected << endl;

cout << "Data of the frame is right, serial is: " << serial << endl;

char \*sendFrame = new char[sendFrameLen];

sendFrame[0] = 1 - frameExpected + '0';

sendto(Sock, sendFrame, strlen(sendFrame), 0, (SOCKADDR\*)&senderAddress, len);

cout << "Sending ack, ack is: " << (1 - frameExpected) << endl << endl;

if (serial - 0 == frameExpected)

{

char \*dataFrame = new char[validDataLen];

for (int i = 0; i < validDataLen; i++)

{

dataFrame[i] = receiveFrame[dataStartPos + i];

}

out.write(dataFrame, validDataLen);

frameExpected = (frameExpected + 1) % 2;

}

}

else //数据出错

{

printTime();

cout << "Data of the frame is wrong, doesn't send ack." << endl << endl;

}

}

out.close();

closesocket(Sock);

WSACleanup();

}

·主函数

int main()

{

cout << "Be ready to receive file..." << endl;

UDPReceiver operation;

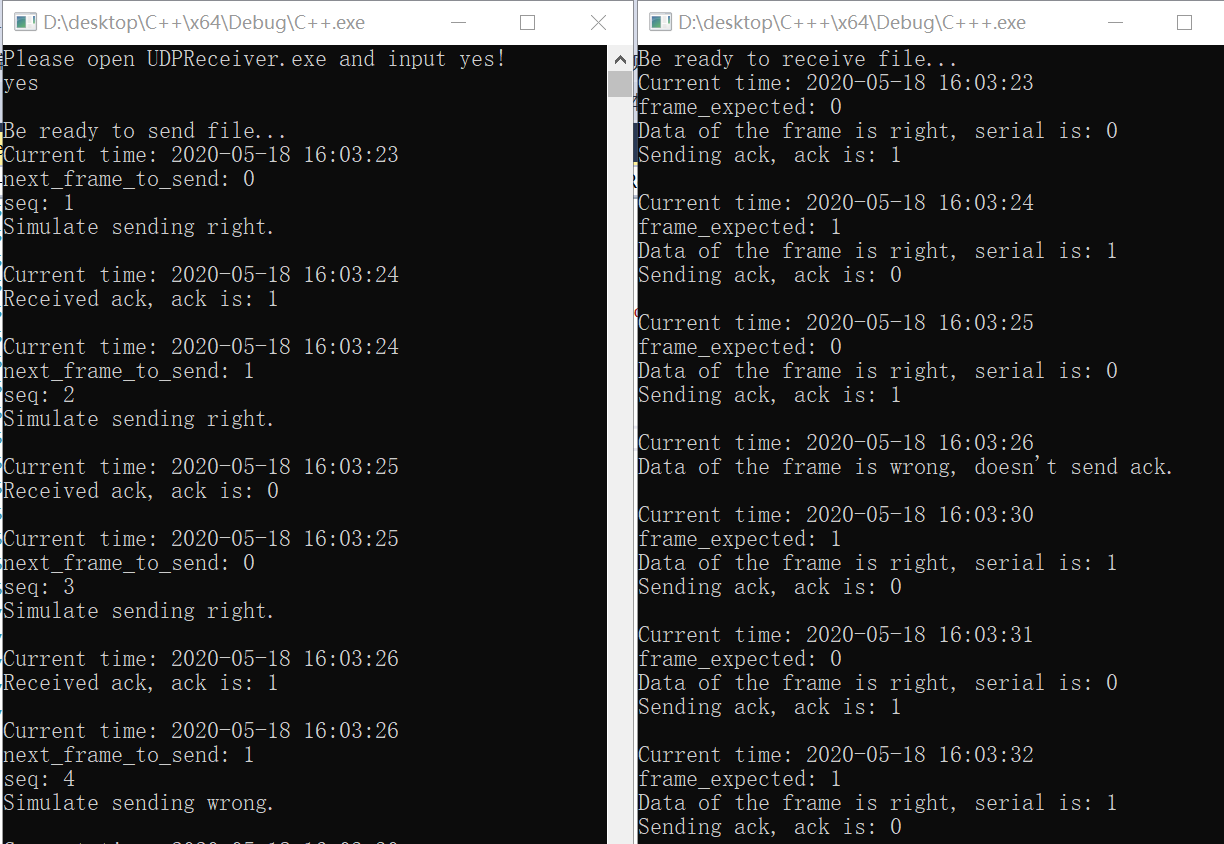
operation.Receive();

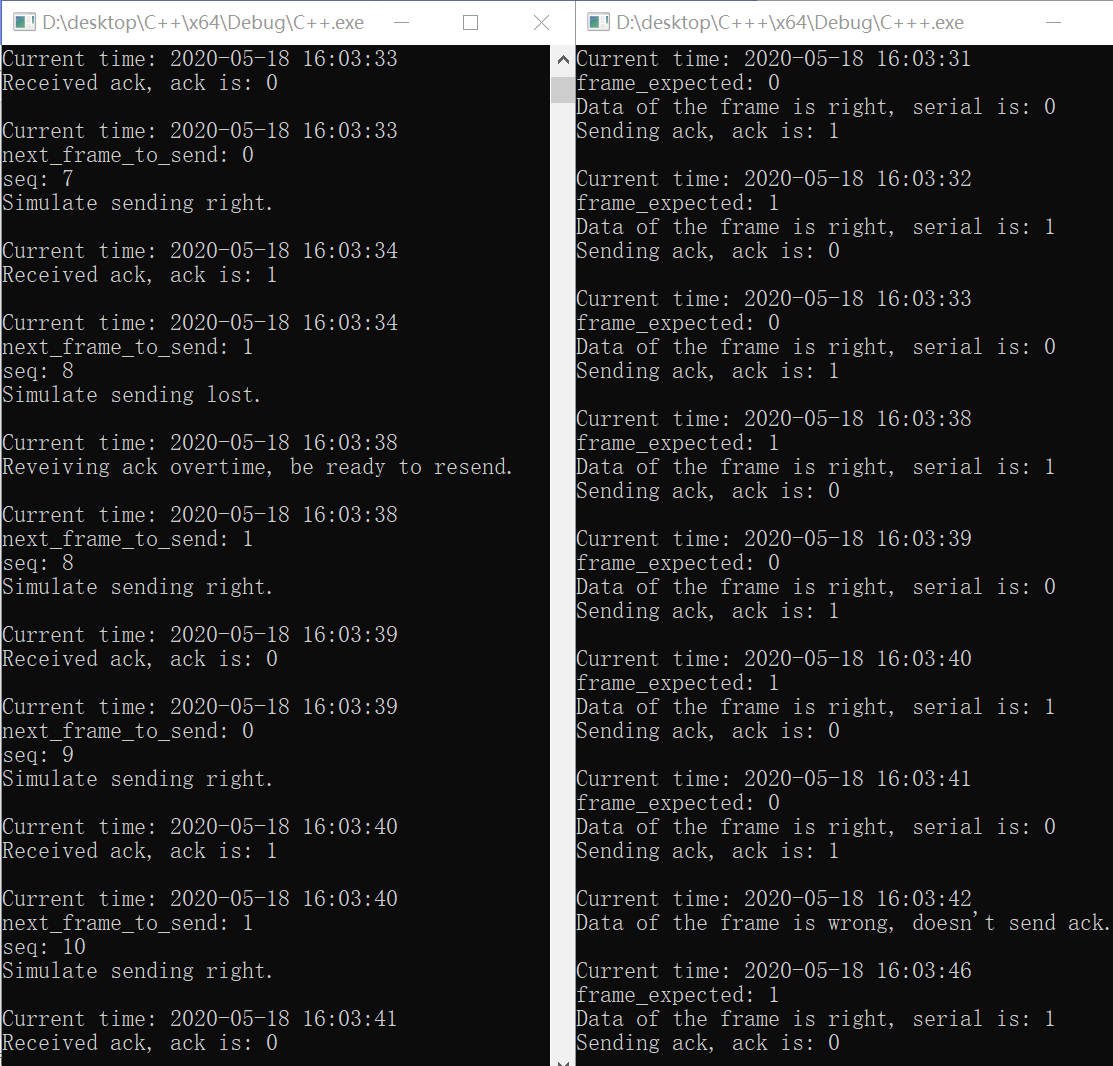
system("pause");

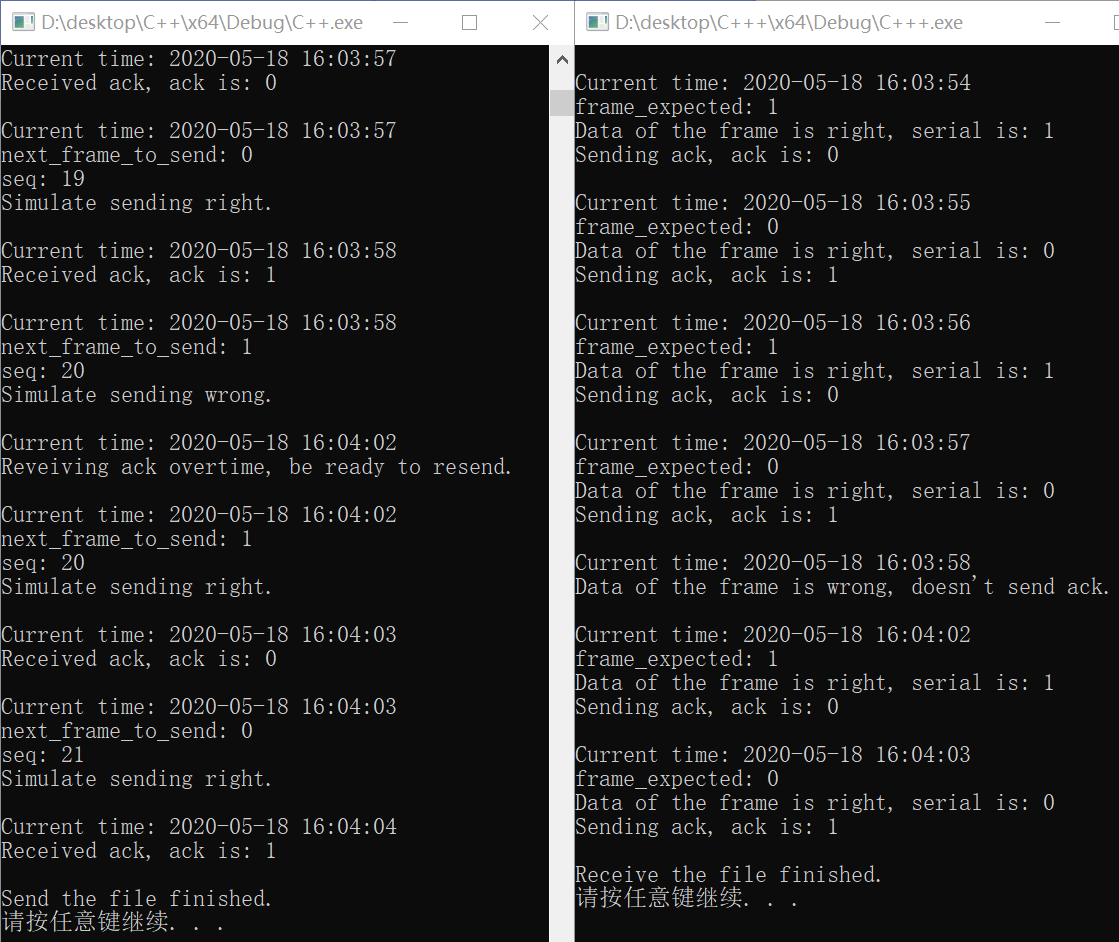
}

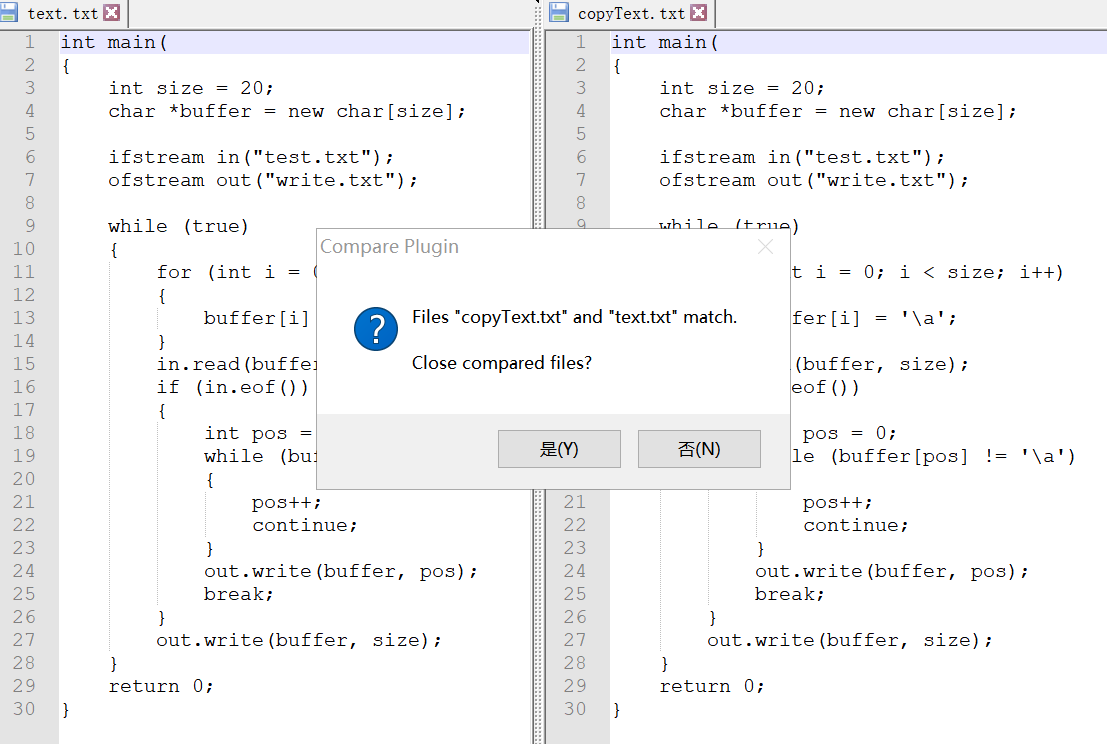
1. **实验结果**

三份代码的输出格式完全一致，下面以C++代码输出为例





**比较发送文件和接收文件**



1. **实验总结**

这个实验的三份代码太折腾了，难点有以下几点：

1. 读文件，发送端每发送完一帧收到确认帧后，从文件中读取一定长度的数据，最后一次读取的数据长度不是之前的固定长度，所以需要判断，接收端同样需要识别出有效数据。
2. UDP传输，Java和python的UDP是传输字节数据，而C++的UDP是传输字符串，所以在Java和Python中的CRC校验是两个字节，而在C++中是长度为16的01字符串，这需要一系列不同的转换函数。
3. 模拟三种模式，题目要求每10帧1帧出错和丢失，其他发送帧就是模拟正确传输。模拟出错通过把计算后的CRC校验码中的某一位更改，模拟丢失就不发送。
4. 处理超时间隔，三种语言的UDP传输都可以设置接收函数的阻塞时间，如果超时退出阻塞，Java和Python中是接收异常，C++中是返回值为-1。
5. 传输完毕退出，发送端收到最后一个数据帧的确认帧后，发送没有数据的标识帧，打印并退出；接收端收到一帧，识别出是标识帧，打印并退出。
6. 调试时间过长，UDP传输和不同类型数据之间的转换很多坑点，用到了很多奇怪的函数，每份代码都需要调试很长时间，远超过写代码的时间，经过数天调试，最终成功将三份代码的输出格式统一。
7. 更复杂更心累的体验的请见下一个实验——GBN。
8. **参考博客**

·C++

1. C++ udp实现简单的通信

<https://blog.csdn.net/qq_39200996/article/details/89314881>

1. 关于recvfrom接收超时

<https://blog.csdn.net/u010951692/article/details/38657173>

·Java

# UDP发送数据和接收数据

<https://blog.csdn.net/weixin_44706512/article/details/89955175>

1. Java 网络编程 之 UDP 文件传输

<https://blog.csdn.net/verejava/article/details/80620553?depth_1-utm_source=distribute.pc_relevant.none-task&utm_source=distribute.pc_relevant.none-task>

1. Java UDP实现文件传输

<https://blog.csdn.net/Ramer42/article/details/83582240>

1. java基础知识之FileInputStream流

<https://blog.csdn.net/ai_bao_zi/article/details/81097898>

1. Java UDP通信：Java DatagramSocket类和DatagramPacket类

<http://c.biancheng.net/view/1203.html>

·Python

1. Python实现socket——udp的传输与接收

<https://blog.csdn.net/weixin_44321116/article/details/96475120>

1. Python中的socket网络编程(TCP/IP，UDP)讲解

<https://blog.csdn.net/csdn15698845876/article/details/78311576>

1. python - udp socket通信循环发送和接收数据

<https://blog.csdn.net/xuezhangjun0121/article/details/88786590>

1. python socket编程详细介绍

<https://blog.csdn.net/weixin_34112900/article/details/85070202?depth_1-utm_source=distribute.pc_relevant.none-task&utm_source=distribute.pc_relevant.none-task>

1. Python实现UDP协议下的文件传输

<https://blog.csdn.net/qq_38898129/article/details/89319767>