湖南科技大学计算机科学与工程学院

计算机网络 课程设计报告

**专业班级：** 20计算机科学与技术二班

**姓 名：** 黎丽薇

**学 号：** 2005010224

**指导教师：** 廖祝华

**时 间**： 2022.12.25

**地 点**： 逸夫楼401

|  |
| --- |
| **指导教师评语：**  **成绩： 等级：**  **签名：**  **年 月 日** |

1. **实验题目**

网络聊天程序的设计与实现

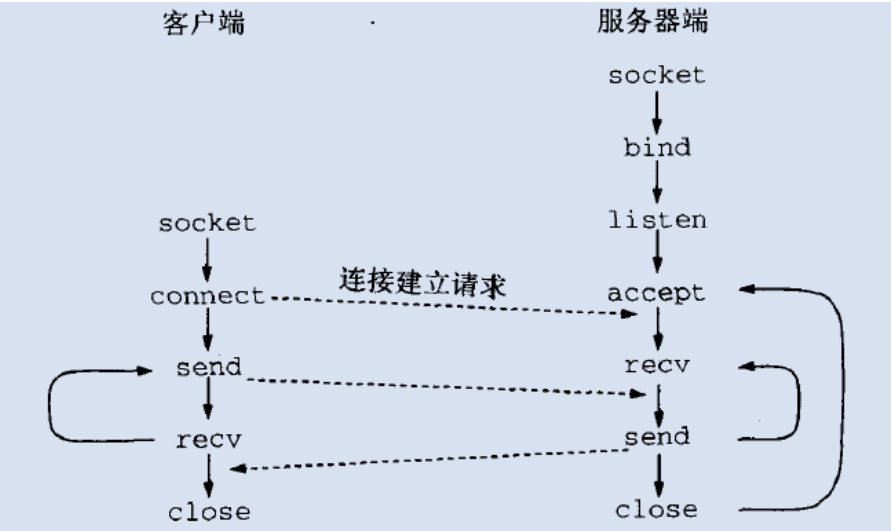
1. **实验目的**
2. 了解 Socket 通信的原理，在此基础上编写一个聊天程序。
3. 学习使用WinSock编程。

**三、总体设计**

**背景知识：**WinSock 并不是一种网络协议，它只是一个网络编程接口，也就是说，它不是协议，但是它可以访问很多种网络协议，你可以把它当作一些协议的封装。现在的 WinSock 已经基本上实现了与协议无关。你可以使用 WinSock 来调用多种协议的功能。那么，WinSock 和 TCP/IP 协议到底是什么关系呢？实际上，WinSock 就是 TCP/IP 协议的一种封装，你可以通过调用 WinSock 的接口函数来调用 TCP/IP的各种功能.例如我想用 TCP/IP 协议发送数据，你就可以使用 WinSock 的接口函数 Send()来调用TCP/IP 的发送数据功能，至于具体怎么发送数据，WinSock 已经帮你封装好了这种功能。

**基本原理与算法：**

多客户端通信：服务器要接受多个客户端的连接；多线程来解决多客户端通信问题，有客户端连接到服务器，启动一个线程来和这个客户端连接，线程里，接收客户端发来的数据，把数据广播给当前连接到服务器的所有客户端。

系统调用使用顺序：

**模块介绍：**

程序分为server(服务端)和client(客户端)，实现多线程客户端输入聊天内容，在服务端显示聊天情况。

Server:服务器端实现的功能是连接客户端，并且展示客户端的聊天内容。

Client:客户端是客户输入聊天内容发送至服务端，服务端展现客户端的编号以及聊天内容，从而构建成一个小的聊天室。

**设计步骤：**

程序分为server(服务端)和client(客户端)，实现多线程客户端输入聊天内容，在服务端显示聊天情况。

Server:此处用的是winsock编程的基础步骤，首先请求协议版本，然后创建SOCKET，之后创建协议地址族，然后绑定和监听，最后等待客户端连接，由于是要实现多线程连接，所以客户端连接是由客户端地址族进行连接，用循环，通讯时也需要根据不同的客户端记录在不同的内存中，以便显示客户端与输出内容。

Client:开始与Sever端一样，请求协议版本、创建SOCKET，之后与Server的创建协议地址族不同，要获取服务器协议地址族，连接服务器，然后进行通信。

**四、详细设计**

**主要的数据结构：**

//服务器协议地址族

SOCKADDR\_IN addr={0};

addr.sin\_family=AF\_INET;//协议版本

addr.sin\_addr.S\_un.S\_addr=inet\_addr("192.168.0.108");

addr.sin\_port=htons(10277); //0-65535 10000左右

**关键代码：**

Server:

int main(){

//6.等待客户端连接[前面代码略]

//客户端协议地址族

while(1){

clientSocket[countf]=accept(serverSocket,(sockaddr\*)&cAddr,&len);

if(SOCKET\_ERROR == clientSocket[countf]){

cout<<"服务器宕机了！"<<endl;

closesocket(serverSocket);

WSACleanup();

return -5;

}

cout<<"客户端连接到服务器了:";

cout<<inet\_ntoa(cAddr.sin\_addr);

cout<<"！"<<endl; CreateThread(NULL,NULL,(LPTHREAD\_START\_ROUTINE)tongxing,(char\*)countf,NULL,NULL);

countf++;

}

}

void tongxing(int idx)

{

char buff[1024];

int r;

while(1)

{

r=recv(clientSocket[idx],buff,1023,NULL);

if(r>0){

buff[r]=0;

cout<<idx<<">>"<<buff<<endl;

//广播数据

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

{

send(clientSocket[i],buff,strlen(buff),NULL);

}

}

}

}

Client:

int main(){

//3.获取服务器协议地址族[前面代码略]

SOCKADDR\_IN addr={0};

addr.sin\_family=AF\_INET;//协议版本

addr.sin\_addr.S\_un.S\_addr=inet\_addr("192.168.0.108");

addr.sin\_port=htons(10277); //0-65535 10000左右

//4.连接服务器

int r=connect(clientSocket,(sockaddr\*)&addr,sizeof addr);

if(r==-1)

{

cout<<"连接服务器失败！"<<endl;

return -3;

}

cout<<"连接服务器成功！"<<endl;

//5.通信

char buff[1024];

CreateThread(NULL,NULL,

(LPTHREAD\_START\_ROUTINE)jieshou,

NULL,NULL,NULL);

while(1){

cout<<"聊天内容："<<endl;

cin>>buff;

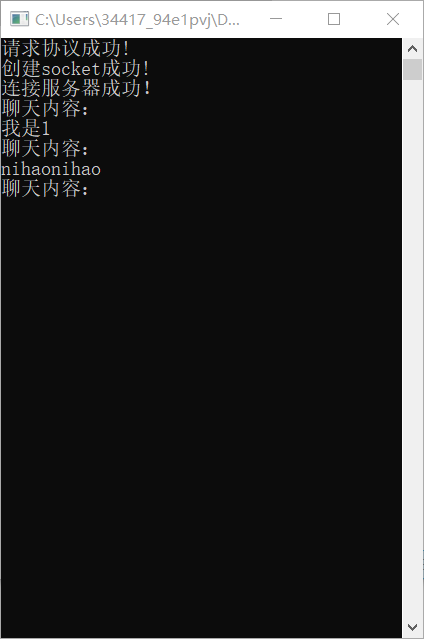
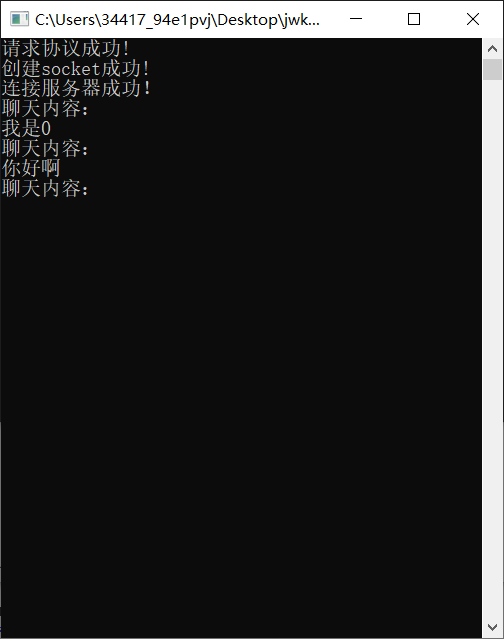
send(clientSocket,buff,strlen(buff),NULL);

}

return 0;

}

1. **实验结果与分析**





**六、小结与心得体会**

该题目让我初步了解了socket编程的一些基本步骤，以及用C语言写多线程如何写，实现多线程客户端连接服务端，为完成本题目，我通过看书以及上网查找socket编程的教程视频学习完成编程，因为是第一次接触所以操作还有步骤很多不熟悉，再加上由于visual C++闪退的原因，选择使用codeblocks编程，由于编译器的差异性，所以有很多关于编译器配置的问题出现，解决也花了很长一段时间，虽然功能实现没有实现我预想的结果，但是已经完成了题目要求，学习到了很多，同样也学到了codeblocks开发工具的更多功能，受益匪浅。

1. **实验题目**

Tracert 与 Ping 程序设计与实现

1. **实验目的**

了解 Tracert 程序的实现原理，并调试通过。然后参考 Tracert 程序和教材 4.4.2 节，编写一个 Ping 程序，并能测试本局域网的所有机器是否在线。

**三、总体设计**

**背景知识：**Tracert（跟踪路由）是路由跟踪实用程序，用于确定 IP数据包访问目标所采取的路径。Tracert 命令使用用 IP 生存时间 (TTL) 字段和 ICMP 错误消息来确定从一个主机到网络上其他主机的路由。

ping （Packet Internet Groper）是一种因特网包探索器，用于测试网络连接量的程序。Ping是工作在 TCP/IP网络体系结构中应用层的一个服务命令， 主要是向特定的目的主机发送 ICMP（Internet Control Message Protocol 因特网报文控制协议）Echo 请求报文，测试目的站是否可达及了解其有关状态。

**基本原理与算法：**

Tracert 程序关键是对 IP 头部生存时间(time to live)TTL 字段的使用,程序实现时是向目地主机发送一个 ICMP 回显请求消息，初始时 TTL 等于 1，这样当该数据报抵达途中的第一个路由器时，TTL 的值就被减为 0，导致发生超时错误，因此该路由生成一份 ICMP 超时差错报文返回给源主机。随后，主机将数据报的 TTL 值递增 1，以便 IP 报能传送到下一个路由器，并由下一个路由器生成ICMP 超时差错报文返回给源主机。不断重复这个过程，直到数据报达到最终的目地主机，此时目地主机将返回 ICMP 回显应答消息。这样，源主机只需对返回的每一份 ICMP 报文进行解析处理，就可以掌握数据报从源主机到达目地主机途中所经过的路由信息。

模块介绍：该题主要是通过给的tracert的例子编写一个ping程序，我写的简易的ping程序，可以通过网址查ip，也可以查询某一段ip下的在线情况。

设计步骤：输入ip地址查询，填充目的端socket地址，创建原始套接字，构造ICMP回显请求消息，并以TTL递增的顺序发送报文，填充ICMP报文中每次发送时不变的字段，初始化发送缓冲区，初始化接收缓冲区，设置IP报头的TTL字段，填充ICMP报文中每次发送变化的字段，记录序列号和当前时间，发送TCP回显请求信息，接收ICMP差错报文并进行解析处理。

**四、详细设计**

**主要的数据结构：**

//IP 报头

typedef struct

{

unsigned char hdr\_len:4; //4 位头部长度

unsigned char version:4; //4 位版本号

unsigned char tos; //8 位服务类型

unsigned short total\_len; //16 位总长度

unsigned short identifier; //16 位标识符

unsigned short frag\_and\_flags; //3 位标志加13位片偏移

unsigned char ttl; //8 位生存时间

unsigned char protocol; //8 位上层协议号

unsigned short checksum; //16 位校验和

unsigned long sourceIP; //32 位源 IP 地址

unsigned long destIP; //32 位目的 IP 地址

} IP\_HEADER;

//ICMP 报头

typedef struct

{

BYTE type; //8 位类型字段

BYTE code; //8 位代码字段

USHORT cksum; //16 位校验和

USHORT id; //16 位标识符

USHORT seq; //16 位序列号

} ICMP\_HEADER;

//报文解码结构

typedef struct

{

USHORT usSeqNo; //序列号

DWORD dwRoundTripTime; //往返时间

in\_addr dwIPaddr; //返回报文的 IP 地址

} DECODE\_RESULT;

**关键代码：**

int main(){

//其余代码略

//填充目地端 socket 地址

sockaddr\_in destSockAddr;

ZeroMemory(&destSockAddr, sizeof(sockaddr\_in)); destSockAddr.sin\_family = AF\_INET;

destSockAddr.sin\_addr.s\_addr = ulDestIP;

//创建原始套接字

SOCKET sockRaw = WSASocket(AF\_INET, SOCK\_RAW, IPPROTO\_ICMP, NULL, 0,WSA\_FLAG\_OVERLAPPED);

//超时时间

int iTimeout = 3000;

//接收超时

setsockopt(sockRaw, SOL\_SOCKET, SO\_RCVTIMEO, (char \*) &iTimeout, sizeof(iTimeout));

//发送超时

setsockopt(sockRaw, SOL\_SOCKET, SO\_SNDTIMEO, (char \*) &iTimeout, sizeof(iTimeout));

//构造 ICMP 回显请求消息，并以 TTL 递增的顺序发送报文

//ICMP 类型字段

const BYTE ICMP\_ECHO\_REQUEST = 8; //请求回显

const BYTE ICMP\_ECHO\_REPLY = 0; //回显应答

const BYTE ICMP\_TIMEOUT = 11; //传输超时

//其他常量定义

const int DEF\_ICMP\_DATA\_SIZE = 32; //ICMP 报文默认数据字段长度

const int MAX\_ICMP\_PACKET\_SIZE = 1024;//ICMP 报文最大长度（包括报头）

const DWORD DEF\_ICMP\_TIMEOUT = 3000; //回显应答超时时间

const int DEF\_MAX\_HOP = 50; //最大跳站数

//填充 ICMP 报文中每次发送时不变的字段

char IcmpSendBuf[sizeof(ICMP\_HEADER) + DEF\_ICMP\_DATA\_SIZE];//发送缓冲区

memset(IcmpSendBuf, 0, sizeof(IcmpSendBuf)); //初始化发送缓冲区

char IcmpRecvBuf[MAX\_ICMP\_PACKET\_SIZE]; //接收缓冲区

memset(IcmpRecvBuf, 0, sizeof(IcmpRecvBuf)); //初始化接收缓冲区

ICMP\_HEADER \*pIcmpHeader = (ICMP\_HEADER \*) IcmpSendBuf;

pIcmpHeader->type = ICMP\_ECHO\_REQUEST; //类型为请求回显

pIcmpHeader->code = 0; //代码字段为 0

pIcmpHeader->id = (USHORT) GetCurrentProcessId(); //ID 字段为当前进程号

memset(IcmpSendBuf + sizeof(ICMP\_HEADER), 'E', DEF\_ICMP\_DATA\_SIZE);//数据字段

USHORT usSeqNo = 0; //ICMP 报文序列号

int iTTL = 30; //TTL 初始值为 1

BOOL bReachDestHost = FALSE; //循环退出标志

int iMaxHot = DEF\_MAX\_HOP; //循环的最大次数

DECODE\_RESULT DecodeResult; //传递给报文解码函数的结构化参数

while (!bReachDestHost && iMaxHot--)

{

int flag=0;

//设置 IP 报头的 TTL 字段

setsockopt(sockRaw, IPPROTO\_IP, IP\_TTL, (char \*) &iTTL, sizeof(iTTL));

//cout<<iTTL<<flush; //输出当前序号

//填充 ICMP 报文中每次发送变化的字段

((ICMP\_HEADER \*) IcmpSendBuf)->cksum = 0; //校验和先置为 0

((ICMP\_HEADER \*) IcmpSendBuf)->seq = htons(usSeqNo++); //填充序列号

((ICMP\_HEADER \*) IcmpSendBuf)->cksum = checksum((USHORT \*) IcmpSendBuf, sizeof(ICMP\_HEADER) + DEF\_ICMP\_DATA\_SIZE); //计算校验和

//记录序列号和当前时间

DecodeResult.usSeqNo = ((ICMP\_HEADER \*) IcmpSendBuf)->seq; //当前序号

DecodeResult.dwRoundTripTime = GetTickCount(); //当前时间

//发送 TCP 回显请求信息

sendto(sockRaw, IcmpSendBuf, sizeof(IcmpSendBuf), 0, (sockaddr \*) &destSockAddr, sizeof(destSockAddr));

//接收 ICMP 差错报文并进行解析处理

sockaddr\_in from; //对端 socket 地址

int iFromLen = sizeof(from); //地址结构大小

int iReadDataLen; //接收数据长度

while (1)

{

//接收数据

iReadDataLen = recvfrom(sockRaw, IcmpRecvBuf, MAX\_ICMP\_PACKET\_SIZE, 0, (sockaddr \*) &from, &iFromLen);

if (iReadDataLen != SOCKET\_ERROR)//有数据到达

{

//对数据包进行解码

if (DecodeIcmpResponse(IcmpRecvBuf, iReadDataLen, DecodeResult, ICMP\_ECHO\_REPLY, ICMP\_TIMEOUT))

{

//到达目的地，退出循环

if (DecodeResult.dwIPaddr.s\_addr ==destSockAddr.sin\_addr.s\_addr)

bReachDestHost = true;

//输出 IP 地址

cout << "\t 目的IP为：" << inet\_ntoa(DecodeResult.dwIPaddr) << endl;

break;

}

}

else if (WSAGetLastError() == WSAETIMEDOUT)

//接收超时，输出\*号

{

flag=1;

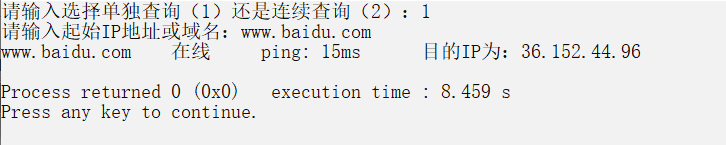
cout << " 不在线" << endl;

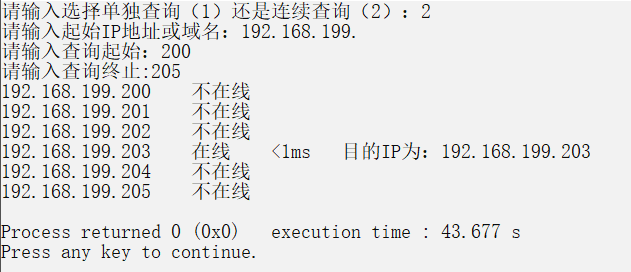
break;

}

}

**五、实验结果与分析**





**六、小结与心得体会**

该题目主要了解了tracert以及ping的相关知识，通过看tracert的源代码写出ping程序，能查找出ip地址上设备的在线情况，对于tracert和ping有了更深的理解，对报文传输等方面知识也有更多的学习，同时也学习了利用socket原始套接字实现发送/监听ICMP报文，实现基本的Ping功能，向目标主机发送ICMP回应请求,然后等待目标主机的ICMP回应答复。

1. **实验题目**

电子邮件客户端程序设计与实现

1. **实验目的**

参照教材 6.5 节原理，设计一个电子邮件客户端程序。

**三、总体设计**

**背景知识：**

SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）是由源地址到目的地址传送邮件的一组规则，用来控制信件的中转方式。SMTP协议属于TCP/IP协议族，它使每台计算机在发送或中转信件时能找到下一个目的地。通过使用指定的服务器，把Email寄到收信人的服务器上。

Telnet协议是TCP/IP协议族的其中之一，是Internet远程登录服务的标准协议和主要方式，常用于网页服务器的远程控制。它为用户提供了在本地计算机上完成远程主机工作的能力。在终端使用者的电脑上使用telnet程序，用它连接到服务器。终端使用者可以在telnet程序中输入命令，这些命令会在服务器上运行，就像直接在服务器的控制台上输入一样。可以在本地就能控制服务器。要开始一个telnet会话，必须输入用户名和密码来登录服务器。Telnet是常用的远程控制Web服务器的方法。

**基本原理与算法：**

在SMTP握手阶段，客户端向SMTP服务器分别指定发件人和收件人的电子邮件地址。握手阶段完毕，SMTP服务器把客户端发出的邮件消息添加到发信队列中，通过TCP提供的可靠数据传输服务把该消息准确地传送到收件人的服务器。

连接和发送过程如下：

（1）建立TCP连接。

（2）客户端发送 HELO 命令以标识发件人自己的身份，客户端发送 MAIL 命令。服务器以OK作为响应，表明准备接收。

（3）使用 AUTH 命令登录SMTP服务器，输入用户名和密码（注意，用户名和密码都需要base64加密）。

（4）客户端发送 RCPT 命令，标识该电子邮件的计划接收人，可以有多个RCPT行。服务器以OK作为响应，表示愿意为收件人发送邮件。

（5）协商结束后，使用 DATA 命令发送。

（6）以 . 号表示结束，输入内容一起发送出去，结束此次发送，用 QUIT 命令退出。

**四、详细设计**

**主要的数据结构：**

WSADATA wsaData;

WORD wVersionRequested = MAKEWORD(2, 2);

//WSAStarup，即WSA(Windows SocKNDs Asynchronous，Windows套接字异步)的启动命令

int err = WSAStartup(wVersionRequested, &wsaData);

cout << "WSAStartup:" << err << endl;

SOCKET sockClient; //客户端的套接字

sockClient = socket(AF\_INET, SOCK\_STREAM, 0); //建立socket对象

HOSTENT\* pHostent;//hostent是host entry的缩写，该结构记录主机的信息，包括主机名、别名、地址类型、地址长度和地址列表

pHostent = gethostbyname("smtp.qq.com"); //得到有关于域名的信息,链接到qq邮箱服务器

SOCKADDR\_IN addrServer; //服务端地址

addrServer.sin\_addr.S\_un.S\_addr = \*((DWORD \*)pHostent->h\_addr\_list[0]); //得到smtp服务器的网络字节序的ip地址

addrServer.sin\_family = AF\_INET;

addrServer.sin\_port = htons(25); //连接端口25

**关键代码：**

err = connect(sockClient, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR)); //向服务器发送请求

//代码略

static string uername = base64加密的邮箱

send(sockClient, uername.c\_str(), uername.length(), 0);

buff[recv(sockClient, buff, 1500, 0)] = '\0';

cout << "usrname:" << buff << endl;

static string psw = "cnpicXRuaGVvcXBjZmhiaQ==\r\n";//此处填base64加密的授权码

send(sockClient, psw.c\_str(), psw.length(), 0);

buff[recv(sockClient, buff, 1500, 0)] = '\0';

cout << "password:" << buff << endl;

string mail;

//代码略

message.append(mail);

message.append("> \r\n");

cout << "message=" << message;

send(sockClient, message.c\_str(), message.length(), 0);

buff[recv(sockClient, buff, 1500, 0)] = '\0';

cout << "mail from的状态码: " << buff << endl;

buff[recv(sockClient, buff, 1500, 0)] = '\0';

cout << "rcpt to的状态码: " << buff << endl;

message = "DATA\r\n";

send(sockClient, message.c\_str(), message.length(), 0);

buff[recv(sockClient, buff, 1500, 0)] = '\0';

cout<< "data命令返回的状态码: " << buff << endl;

static string form= 邮箱地址

cout<<"主题：";

cin>>subject;

form.append(subject);

form.append("\r\n\r\n");

cout<<"内容：";

cin>>info;

form.append(info);

form.append("\r\n.\r\n");

send(sockClient, form.c\_str(), form.length(), 0);

message = "quit\r\n";

send(sockClient, message.c\_str(), message.length(), 0);

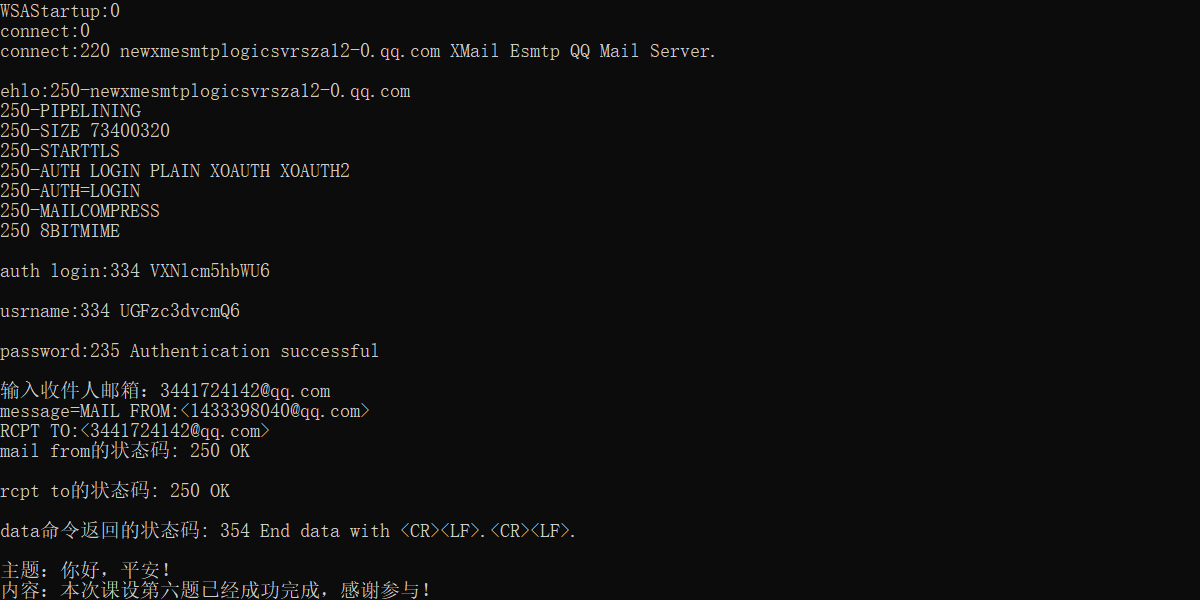
buff[recv(sockClient, buff, 1500, 0)] = '\0';

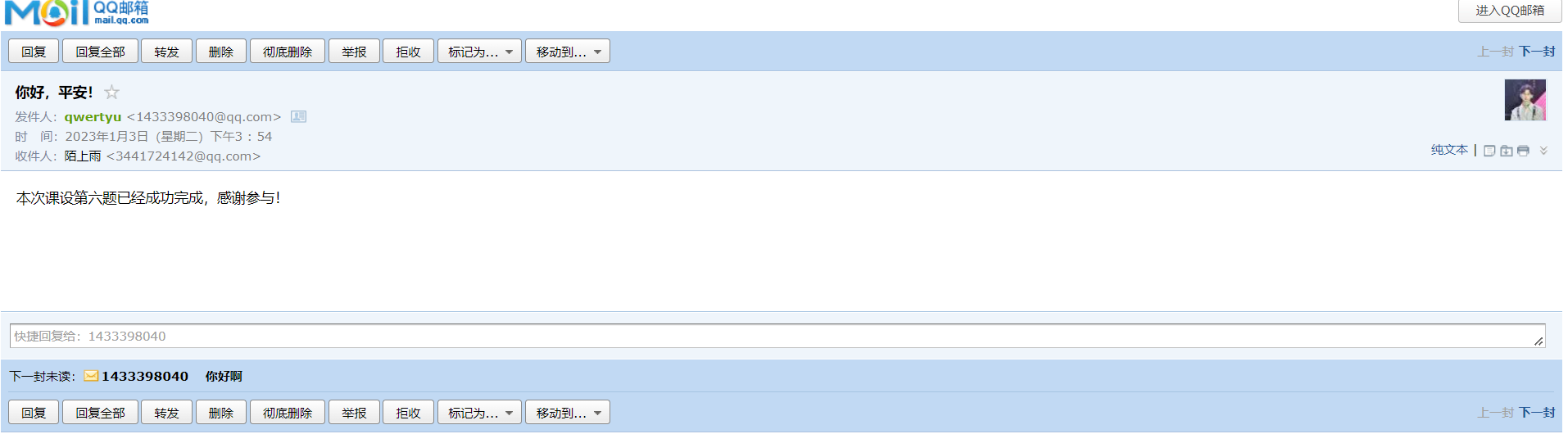
cout << "返回状态码：" << buff << endl;

cout << "发送成功！"<<endl;

system("pause");

**五、实验结果与分析**





**六、小结与心得体会**

通过该题了解了邮件发送在终端的流程，对发送邮件有了更好的理解，尤其是书上的SMTP协议的格式，同时也有了一定的能力用代码发送邮件，对于邮箱这种常用的工具，对它的实现原理有了更好的理解，获益匪浅。

1. **实验题目**

简单 Web Server 程序的设计与实现

1. **实验目的**

编写简单的 Web Server 有助于读者了解 Web Server 的工作流程，掌握超文本传送协议( HTTP)基本原理，掌握 Windows 环境中用 socket 实现 C/S 结构程序的编程方法。附录 5 介绍了一个简单 Web Server 的程序设计过程。

**三、总体设计**

背景知识：Web 服务是 Internet 最方便与受用户欢迎的服务类型，它的影响力也远远超出了专业技术范畴，已广泛应用于电子商务、远程教育、远程医疗与信息服务等领域，并且有继续扩大的趋势。目前很多的 Internet 应用都是基于 Web 技术的，因此掌握 Web 环境的软件编程技术对软件人员是至关重要的。

基本原理与算法：最简单的web服务器就是通过TCP三次握手建立连接后，服务器直接返回一个结果给浏览器。浏览器和服务器是通过TCP三路握手建立连接的。浏览器在通过URL（统一资源定位符，就是我们俗称的网络地址）去请求服务器的连接，并且通过URL中的路径请求服务器上的资源。

设计步骤：定义并且初始化一个服务器套接字，然后初始化，之后创建套接字，接着绑定服务器套接字和监听，然后服务器端建立连接与接收数据，再读取文件路径发送数据。

**四、详细设计**

**主要的数据结构：**

#define BUFFER\_SIZE 1024

#define HOST "192.168.56.1"

#define PORT 9100

#define HEADER "\

HTTP/1.1 200 OK\r\n\

Content-Type: text/html; charset=UTF-8\r\n\

Server: ZYhttp\_v1.0.1\r\n\

Content-Length: %ld\r\n\r\n\

**关键代码：**

int main(int argc, char \*\*argv)

{

// 定义并且初始化一个服务器套接字

sockaddr\_in addrServer;

addrServer.sin\_family = AF\_INET;

addrServer.sin\_addr.S\_un.S\_addr = INADDR\_ANY;

addrServer.sin\_port = htons(PORT);

// 初始化

WSADATA wsaData;

WORD socketVersion = MAKEWORD(2, 2);

if (WSAStartup(socketVersion, &wsaData) != 0)

{exit(1);}

// 创建套接字

SOCKET socketServer = socket(AF\_INET, SOCK\_STREAM, 0);

if (socketServer == SOCKET\_ERROR)

{exit(1);}

// 绑定服务器套接字

if (bind(socketServer, (LPSOCKADDR)&addrServer, sizeof(addrServer)) == SOCKET\_ERROR)

{ exit(1);}

// 监听

if (listen(socketServer, 10) == SOCKET\_ERROR)

{exit(1);}

while (true)

{

cout<<"Listening ... "<<endl;

sockaddr\_in addrClient;

int nClientAddrLen = sizeof(addrClient);

//服务器端建立连接

SOCKET socketClient = accept(socketServer, (sockaddr\*)&addrClient, &nClientAddrLen);

if (SOCKET\_ERROR == socketClient)

{break;}

char buffer[BUFFER\_SIZE];

memset(buffer, 0, BUFFER\_SIZE);

//接收数据

if (recv(socketClient, buffer, BUFFER\_SIZE, 0) < 0)

{break; }

cout<<"接收到的数据 : "<<endl<<buffer;

memset(buffer, 0, BUFFER\_SIZE);

sprintf(buffer, HEADER, GetFileLength(strPath));//把文件和头文件合并然后发送数据

if (send(socketClient, buffer, strlen(buffer), 0) < 0)

{break;}

//读取文本

ifstream fin(strPath.c\_str(), ios::in | ios::binary);

//读取文件长度

if (fin.is\_open())

{

memset(buffer, 0, BUFFER\_SIZE);

//每次读取一个字节

while (fin.read(buffer, BUFFER\_SIZE - 1))

{

if (send(socketClient, buffer, strlen(buffer), 0) < 0)

{break;}

memset(buffer, 0, BUFFER\_SIZE);

}

if (send(socketClient, buffer, strlen(buffer), 0) < 0)

{break;}

}

//关闭，结束

fin.close();

closesocket(socketClient);

}

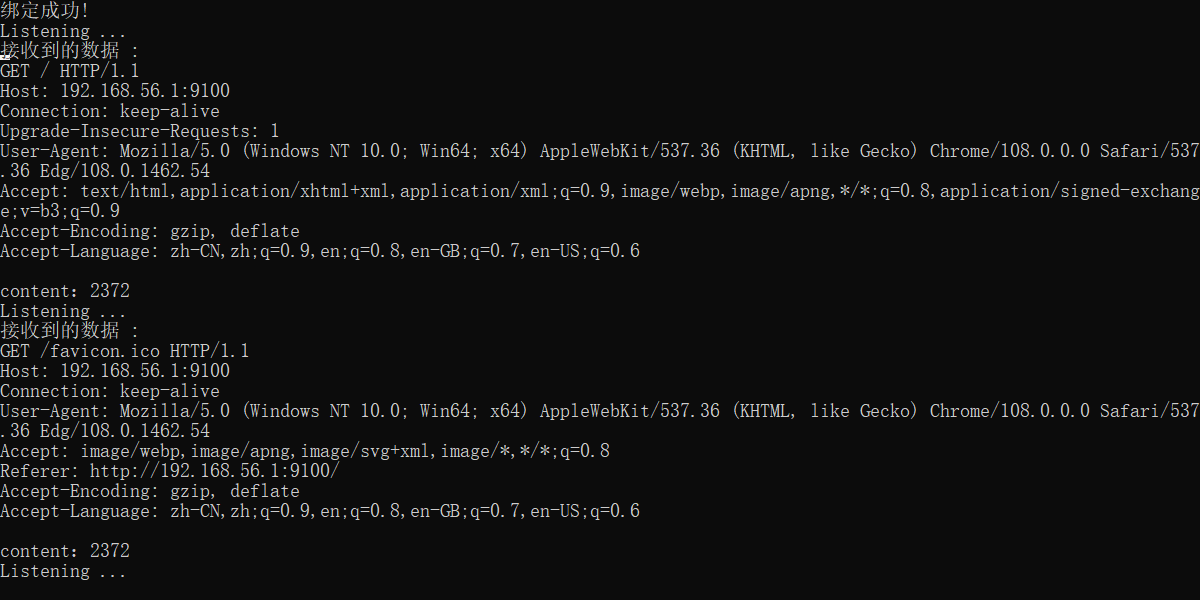
closesocket(socketServer);

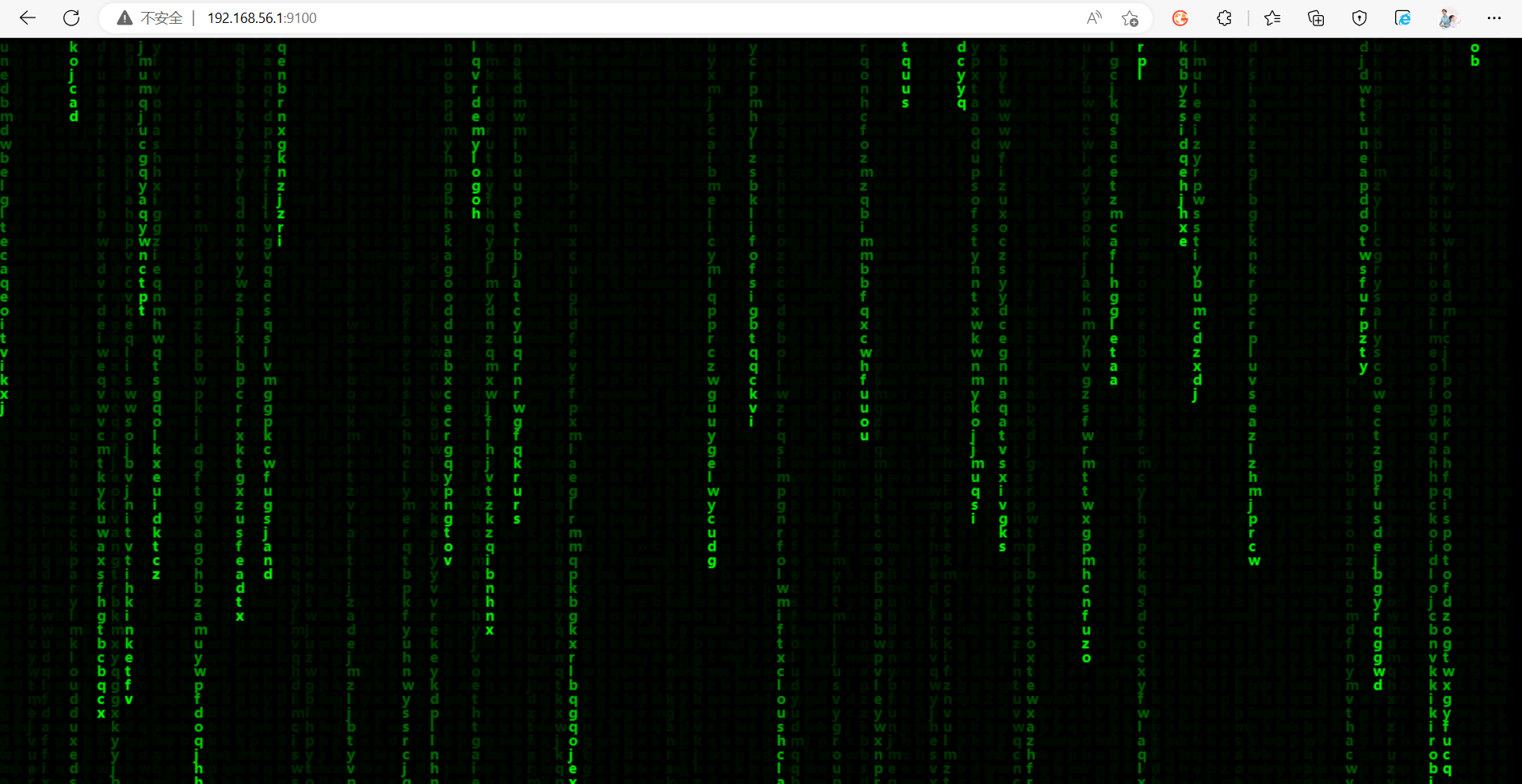
WSACleanup();

return 0;

}

**五、实验结果与分析**





**六、小结与心得体会**

经过两周的计算机网路课设的学习，我学习了解了C/C++的SOCKET编程，通过查找资料等操作，增强了自学能力，在该题目中更是对于web server有了一个更深的了解，虽然最开始对于文件路径有一定的问题，也找了很多资料，最后终于明白了问题所在，解决了遇到的问题，学到了很多。