哈尔滨工业大学计算机学院

《网络程序设计与实践》实验指导书

李全龙

2015 年 06 月

前　言

《网络程序设计与实践》课程是计算机科学与技术专业的重要实践类课程之一。随着计算机网络技术的迅速发展和在当今信息社会中的广泛应用，给《网络程序设计与实践》课程的教学提出了新的更高的要求。

《网络程序设计与实践》是一门实践性较强的课程，本课程不再以课堂教学为主，而将以实践教学为主，通过 4 个典型实验，驱动网络程序设计相关技术的学习，从而达到学以致用的目的。

希望同学们在使用本实验指导书及进行实验的过程中，能够帮助我们不断地发现问题，并提出建议，完善《网络程序设计与实践》课程的建设。

实验要求

计算机网络是现代信息社会最重要的基础设施之一。在过去十几年里得到了迅速的发展和应用。《网络程序设计与实践》课程实验的目的是通过典型的网络应用程序的设计与开发实现，使学生掌握网络应用程序开发过程、方法和技术，熟悉典型网络应用程序开发技能，培养网络应用程序的开发时间能力，增强网络化系统思维意识。总之，通过上述实验环节，使学生加深了解和更好地掌握《网络程序设计与实践》课程教学大纲要求的内容。

在《网络程序设计与实践》的课程实验过程中，要求学生做到：

（1）在各次实验之前提前预习实验指导书有关部分,认真做好实验准备, 就实验可能出现情况提前做出思考和分析。

（2）仔细观察上机和上网操作时出现的各种现象,记录主要情况,做出必要说明和分析。

（3）认真书写实验报告。实验报告包括实验目的和要求,实验情况及其分析。对需要编程的实验,写出程序设计说明,给出源程序框图和清单。

（4）遵守机房纪律,服从辅导教师指挥,爱护实验设备。

（5）实验课程不迟到。根据迟到时间长短扣除相应出勤分数。无故缺席, 当次实验按零分计,过后不补。

（6）实验采用当堂检查方式,每个实验都应当在规定的时间内完成并检查通过。检查指标包括对实验内容的操作完成情况和对指导老师提出的问题的回答情况。

（7）每次完成实验之后,应在一周内在软件学院教学系统上提交实验报告。 如本周一进行的实验,在下周一之前应提交到实验系统中。

（8）部分实验有加分内容,如果完成加分内容,则在操作分数上额外加 5-10 分,但最终全部实验总分数不超过原定满分。实验的验收将分为两个部分：

实验的验收将分为两个部分:

第一部分是上机操作,包括检查程序的运行或者相应实验操作的熟练程度,

以及能够即时回答实验指导老师提出的问题,对遇到的现象能给出合理的解答。

第二部分是提交电子版的实验报告。根据完成实验报告情况给予相应分数。

实验 1：多协议文件传输 C/S 网络应用设计与实现

1、实验目的

掌握 C/S 网络应用程序开发技术。

2、实验环境

- Windows 或 Linux；
- TCP/UDP 双协议；
- 任何你熟悉的编程语言。

3、实验内容

生活中的多协议服务器我们经常看到，本次实验中，我们将用 TCP/UDP 两个协议模拟多协议服务器。

4、实验方式

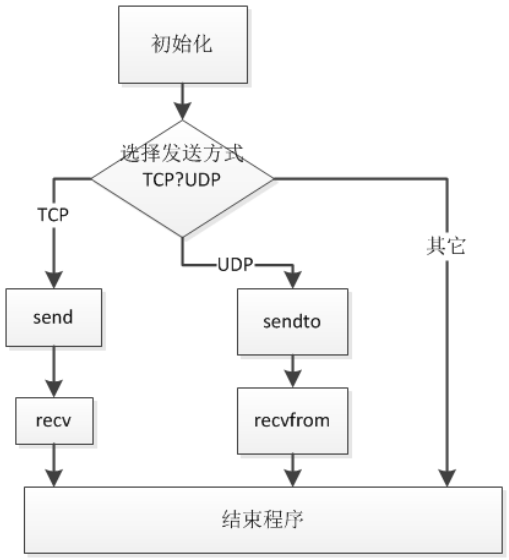
每位同学上机实验，实验指导教师现场指导。

5、实验过程

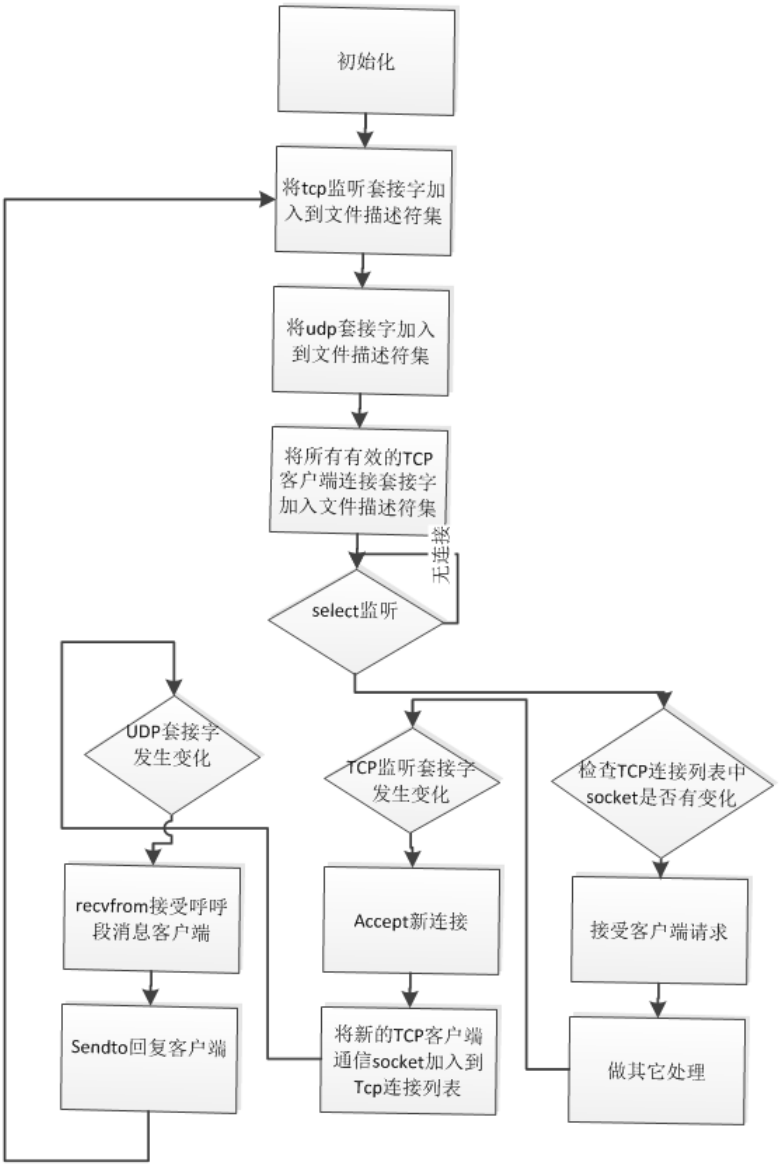
首先服务器端要将两种协议的套接字创建好，并且监听 TCP 端口的连接。必要时我们要用到 select 函数来进行多连接处理和非阻塞处理。而客户端我们直接可以按照发送的地址和端口号来进行通讯。

实验中，我们首先要运行服务器端，服务器会监听 TCP 端口。然后我们运行客户端，根据事先设定好的 IP 和端口，就可以进行连接传输数据了。

客户端程序逻辑较为简单，一个典型的流程图如下所示：



服务端的流程稍微复杂一些，如下图所示：



6、参考内容

（一）UDPService 服务器端参考代码

```
#include <WinSock2.h>
#include <stdio.h>
#include <iostream>

#pragma comment(lib, "ws2_32.lib")
```

```
#define SEND_PORT 8000
#define RECEIVE_PORT 8001
#define MAX_BUF_LEN 255
#define CLIENT_1_IP "192.168.4.107"
#define CLIENT_2_IP "192.168.4.109"

using namespace std;

int _tmain(int argc, _TCHAR* argv[]) {

	// 加载套接字库（必须）
	WORD wVersionRequested;
	WSADATA wsaData;
	// 套接字加载时错误提示
	int err;

	// 启动 socket api,版本 2.2
	wVersionRequested = MAKEWORD(2, 2);
	err = WSAStartup(wVersionRequested, &wsaData);
	if (err != 0) {
		//找不到 winsock.dll
		printf("WSAStartup failed with error: %d\n", err);
		return -1;
	}
	// 低字节，高字节
	if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {
		// 版本错误
		printf("Could not find a usable version of Winsock.dll\n");
		WSACleanup();
		return -1;
	}

	// 创建套接字
	SOCKET connectSocket = socket(AF_INET, SOCK_DGRAM, 0);
	if (INVALID_SOCKET == connectSocket) {
		err = WSAGetLastError();
		printf("\"socket\" error! error code is %d\n", err);
		return -1;
	}

	// 创建套接字
	SOCKET connectSocketTcp = socket(AF_INET, SOCK_STREAM, 0);
	if (INVALID_SOCKET == connectSocketTcp) {
```

```
        err = WSAGetLastError();
        printf("\"socket\" error! error code is %d\n", err);
        return -1;
    }

    // Service 套接字
    SOCKADDR_IN sSercice;
    sSercice.sin_family = AF_INET;
    sSercice.sin_port = htons(7999);
    sSercice.sin_addr.s_addr = 0;

    // Client_1 套接字
    SOCKADDR_IN sClient_1;
    sClient_1.sin_family = AF_INET;
    sClient_1.sin_port = htons(8000);
    sClient_1.sin_addr.s_addr = inet_addr(CLIENT_1_IP);

    // Client_1 套接字
    SOCKADDR_IN sClient_1_tcp;
    sClient_1_tcp.sin_family = AF_INET;
    sClient_1_tcp.sin_port = htons(8001);
    sClient_1_tcp.sin_addr.s_addr = inet_addr(CLIENT_1_IP);

    bool bOpt = true;
    //设置该套接字为广播类型
    setsockopt(connectSocket, SOL_SOCKET, SO_BROADCAST, (char*) &bOpt,
sizeof(bOpt));

    // 绑定本地端口套接字
    err = bind(connectSocket, (SOCKADDR*)&sSercice, sizeof(SOCKADDR));
    if (SOCKET_ERROR == err) {
        err = WSAGetLastError();
        printf("\"bind\" error! error code is %d/n", err);
        return -1;
    }
    connect(connectSocketTcp, (SOCKADDR*)&sClient_1_tcp, sizeof(sClient_1_tcp));

    // 发送缓存
    char sendBuff[MAX_BUF_LEN] = "";
    // 接受缓存
    char receiveBuff1[MAX_BUF_LEN] = "";
    char receiveBuff2[MAX_BUF_LEN] = "";
    int nAddrLen = sizeof(SOCKADDR);
    // 发送数据
```

```
    int nLoop = 0;
    while(nLoop < 100) {
        nLoop++;
        sprintf(sendBuff, "%8d", nLoop);

        // 发送数据用 UDP
        int nSendSize = sendto(connectSocket, sendBuff, strlen(sendBuff), 0,
(SOCKADDR*)&sClient_1, sizeof(SOCKADDR));
        if (SOCKET_ERROR == nSendSize) {
            err = WSAGetLastError();
            printf("\"sendto\" error!, error code is %d\n", err);
            return -1;
        }
        printf("Send: %s\n", sendBuff);

        // 接受数据用 TCP
        int nReceiveSize = recv(connectSocketTcp, receiveBuff1, MAX_BUF_LEN, 0);
        if (SOCKET_ERROR == nReceiveSize) {
            err = WSAGetLastError();
            printf("\"recv\" error! error code is %d\n", err);
            system("pause");
            return -1;
        }
        printf("Recieve1: %s\n", receiveBuff1);
        Sleep(500);
    }

    closesocket(connectSocket);
    closesocket(connectSocketTcp);

    WSACleanup();
    return 0;
}
```

（二）UDPClient 客户端参考代码

```
#include <WinSock2.h>
#include <stdio.h>
#include <iostream>

#pragma comment(lib, "ws2_32.lib")


#define SEND_PORT 8001
```

```
#define RECEIVE_PORT 8000
#define MAX_BUF_LEN 255
#define SERVICE_IP "192.168.4.108"

using namespace std;

int _tmain(int argc, _TCHAR* argv[]) {

	// 加载套接字库（必须）
	WORD wVersionRequested;
	WSADATA wsaData;
	// 套接字加载时错误提示
	int err;

	// 启动 socket api,版本 2.2
	wVersionRequested = MAKEWORD(2, 2);
	err = WSAStartup(wVersionRequested, &wsaData);
	if (err != 0) {
		//不到 winsock.dll
		printf("WSAStartup failed with error: %d\n", err);
		system("pause");
		return -1;
	}
	// 低字节，高字节
	if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {
		// 版本错误
		printf("Could not find a usable version of Winsock.dll\n");
		WSACleanup();
		system("pause");
		return -1;
	}

	// 创建 UDP 套接字
	SOCKET connectSocket = socket(AF_INET, SOCK_DGRAM, 0);
	if (INVALID_SOCKET == connectSocket) {
		err = WSAGetLastError();
		printf("\"socket\" error! error code is %d\n", err);
		system("pause");
		return -1;
	}

	// 创建 TCP 套接字
	SOCKET connectSocketTcp = socket(AF_INET, SOCK_STREAM, 0);
	if (INVALID_SOCKET == connectSocketTcp) {
```

```
        err = WSAGetLastError();
        printf("\"socket\" error! error code is %d\n", err);
        system("pause");
        return -1;
    }

    // 创建 TCP 链接套接字
    SOCKET connectSocketTcpClient;

    // Service 套接字
    SOCKADDR_IN sSercice;
    sSercice.sin_family = AF_INET;
    sSercice.sin_port = htons(7999);
    sSercice.sin_addr.s_addr = inet_addr(SERVICE_IP);

    // Client_1 本机套接字
    SOCKADDR_IN sClient_1;
    sClient_1.sin_family = AF_INET;
    sClient_1.sin_port = htons(8000);
    sClient_1.sin_addr.s_addr = 0;

    // Client_1 本机套接字
    SOCKADDR_IN sClient_1_tcp;
    sClient_1_tcp.sin_family = AF_INET;
    sClient_1_tcp.sin_port = htons(8001);
    sClient_1_tcp.sin_addr.s_addr = 0;

    // 绑定本地端口套接字
    err = bind(connectSocket, (SOCKADDR*)&sClient_1, sizeof(SOCKADDR));
    if (SOCKET_ERROR == err) {
        err = WSAGetLastError();
        printf("\"bind\" error! error code is %d/n", err);
        system("pause");
        return -1;
    }

    // 绑定本地端口套接字
    err = bind(connectSocketTcp, (SOCKADDR*)&sClient_1_tcp, sizeof(SOCKADDR));
    if (SOCKET_ERROR == err) {
        err = WSAGetLastError();
        printf("\"bind\" error! error code is %d/n", err);
        system("pause");
        return -1;
    }
```

```
    // 监听 TCP 连接
    listen(connectSocketTcp, 5);
    connectSocketTcpClient = accept(connectSocketTcp, NULL, NULL);
    printf("new connection");

    // 发送缓存
    char sendBuff[MAX_BUF_LEN] = "";
    // 接受缓存
    char receiveBuff[MAX_BUF_LEN] = "";
    int nAddrLen = sizeof(SOCKADDR);
    // 发送数据
    int nLoop = 100;
    while(nLoop > 0) {
        nLoop--;
        sprintf(sendBuff, "%8d", nLoop);

        // 接受数据通过 UDP
        int nReceiveSize = recvfrom(connectSocket, receiveBuff, MAX_BUF_LEN, 0,
(SOCKADDR*)&sSercice, &nAddrLen);
        if (SOCKET_ERROR == nReceiveSize) {
            err = WSAGetLastError();
            printf("\"recvfrom\" error! error code is %d\n", err);
            system("pause");
            return -1;
        }
        receiveBuff[nReceiveSize] = '\0';
        printf("Receive: %s\n", receiveBuff);

        // 发送数据通过 TCP
        int nSendSize = send(connectSocketTcpClient, sendBuff, strlen(sendBuff), 0);
        if (SOCKET_ERROR == nSendSize) {
            err = WSAGetLastError();
            printf("\"sendto\" error!, error code is %d\n", err);
            system("pause");
            return -1;
        }
        printf("Send: %s\n", sendBuff);
    }

    closesocket(connectSocket);
    closesocket(connectSocketTcp);
    closesocket(connectSocketTcpClient);
```

```
    WSACleanup();
    return 0;
}
```

7、实验报告

在实验报告中要说明实现 C/S 网络应用程序的关键步骤、程序运行的实验过程和实验结果（需要包含程序运行的截图）。

实验 2：基于原始套接字的 Tracert 程序设计与实现

1、实验目的

掌握基于原始套接字的网络应用程序开发技术。

2、实验环境

- Windows 或 Linux；
- ICMP 协议（注意：Linux 系统默认使用 UDP 协议实现 TraceRT，你可能需要添加参数-I 强制使用 ICMP 协议）；
- 任何你熟悉的编程语言。

3、实验内容

1）掌握常用网络命令的 TraceRT(TraceRoute)功能及使用方法。

2）实现基于原始套接字的 TraceRT 程序开发。本实验只需要实现一个简单的路由探测程序即可，目的为某个 IP 地址或者主机名，即：*tracert [ip_or_hostname]*

4、实验方式

每位同学上机实验，实验指导教师现场指导。

5、实验过程

（一）TraceRT 命令使用方法

用法: tracert [-d] [-h maximum_hops] [-j host-list] [-w timeout] [-R] [-S srcaddr] [-4] [-6] target_name

选项:

-d	不将地址解析成主机名。
-h maximum_hops	搜索目标的最大跃点数。
-j host-list	与主机列表一起的松散源路由(仅适用于 IPv4)。
-w timeout	等待每个回复的超时时间(以毫秒为单位)。
-R	跟踪往返行程路径(仅适用于 IPv6)。
-4	强制使用 IPv4。
-6	强制使用 IPv6。

思考以下问题：

（1）查看本机到 www.sina.com.cn 所经过的路由的列表。总共经过了多少个路由？

（2）查看本机到 www.sina.com.cn 所经过的路由的列表，参数设成不解析 ip 地址到主机名。与第(1)问相比你看到了什么现象？解析其原因。

（3）将最大跳数设成 3，再查看本机到 www.sina.com.cn 所经过的路由的列表。看到了什么现象？解释其原因。

（二）TraceRT 工作原理

TraceRoute 通过设置 IP 首部中的寿命（TTL）字段来实现路由探测的功能。TTL 是一个 IP 分组的生存时间，IP 分组经过每个路由器的时候都会将 TTL 值减一。这样，TTL 值就

可以看成经过路由器跳数的计数器。每当路由器接受到一个 TTL 为 0 或者 1 的 IP 分组时，路由器就不再装发这个分组，而是直接丢弃，并且发送一个 ICMP“超时”报文给源主机。这个程序的关键就在于返回的超时错误的 ICMP 分组的 IP 首部的源地址就是这个路由器的入口 IP 地址。通过逐渐增大的 TTL 的值，可以得到该条路径上所有的路由器的入口 IP 地址，直到对目的主机发送一个 UDP 端口不可达报文，并收到 ICMP 的“端口不可达”的响应分组为止。当然，考虑实际的需求，一般会设定一个最大的跳数，比如 30，如果超过这个跳数依然没有到目的主机，也停止探测。

总结来说，整个工作过程如下。源端刚开始发送 UPD/IP 报文时，将报文 TTL 字段设置为 1，报文到达第一个路由器时将 TTL 减 1，这样由于报文的 TTL 字段为 0，报文通过的第一个路由器就将此报文丢弃，并向源端返回一个含有该路由器 IP 地址的 ICMP 超时报文。同样的到底，源端发送的报文的 TTL 依次加一，第二次时将 TTL 字段设置为 2，该报文在第二个路由器是被丢弃，第二个路由器向源端返回一个含有它的 IP 地址的 ICMP 超时报文。依次，通过这个过程就可一知道从源端到目的端的所有路由器的地址信息了。

（三）详细的软件设计

程序设计好的程序流程图如下：

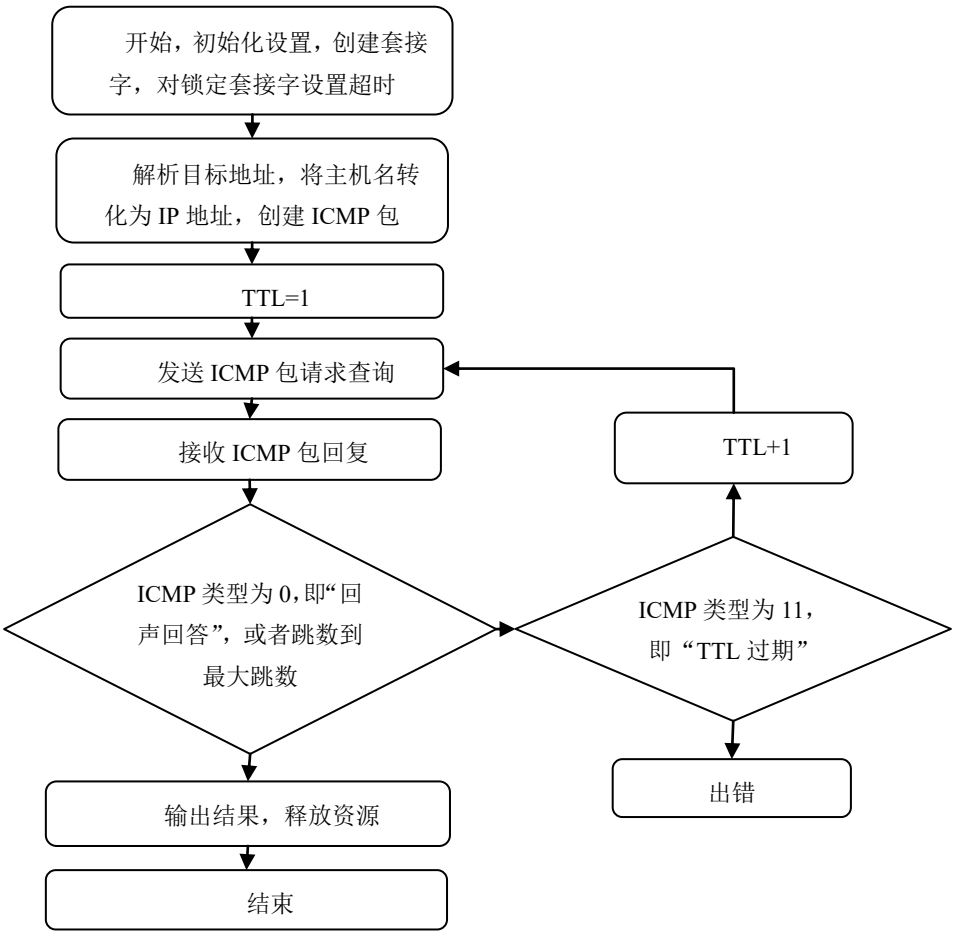


图2-1 程序流程图

程序的详细描述如下:

1. 初始化设置，创建socket，主机解析，创建ICMP包
2. 设置TTL值 n=1。设置TTL的最大值为30
3. 主机向目标主机发送TTL=n的数据包

4. 接收到数据包的路由器对接收到的数据包中的TTL字段减1
5. 对n值做判断，如果n=0，那么丢弃数据包，向源主机发送ICMP数据包
6. 源主机接收并且解析ICMP报文，类型为0则说明应经到达目的主机，跳转到8。类型为11则说明TTL过期。其他类型表示出错，也跳转到8.
7. 源主机TTL+1，跳转到3.
8. 输出结果，并且释放资源。

6、参考内容

参考代码：

```
#include <iostream>
#include <iomanip>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include "Tracert.h"
#pragma comment(lib,"ws2_32")

using namespace std;

int main(int argc, char* argv[])
{
    //检查命令行参数  第二个为目标主机 ip 或者域名
    if (argc != 2)
    {
        cerr << "用法: tracert [ip_or_hostname]\n";
        return -1;
    }
    //初始化 winsock2 环境
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    {
        cerr << "初始化  WinSock2 DLL  失败\n"
            << "错误代码: " << WSAGetLastError() << endl;
        return -1;
    }
    //将命令行参数转换为 IP 地址
    u_long ulDestIP = inet_addr(argv[1]);
    if (ulDestIP == INADDR_NONE)
    {
        //转换不成功时按域名解析
        hostent* pHostent = gethostbyname(argv[1]);
        if (pHostent)
        {
            ulDestIP = (*(in_addr*)pHostent->h_addr).s_addr;
```

```
            //输出屏幕信息
            cout << "通过最多 " << DEF_MAX_HOP << " 个跃点跟踪到\n" << argv[1]
            << " [" << inet_ntoa(*(in_addr*)(&ulDestIP)) << "] 的路由 \n"<<endl;
        }
        else //解析主机名失败
        {
            cerr << "无法解析目标系统名称 " << argv[1] << '\n'
                 << "错误代码: " << WSAGetLastError() << endl;
            WSACleanup();
            return -1;
        }
    }
    else
    {
        //输出屏幕信息
        cout << "通过最多 " << DEF_MAX_HOP << " 个跃点跟踪到 " << argv[1]
        << " 的路由 \n"<<endl;
    }
    //填充目的 Socket 地址
    sockaddr_in destSockAddr;
    ZeroMemory(&destSockAddr, sizeof(sockaddr_in));
    destSockAddr.sin_family = AF_INET;
    destSockAddr.sin_addr.s_addr = ulDestIP;
    //使用 ICMP 协议创建 Raw Socket
    //原始套接字必须用管理员权限创建
    SOCKET sockRaw = WSASocket(AF_INET, SOCK_RAW, IPPROTO_ICMP, NULL, 0, WSA_FLAG_OVERLAPPED);
    if (sockRaw == INVALID_SOCKET)
    {
        cerr << "创建原始套接字失败\n"
             << "错误代码: " << WSAGetLastError();
        if(WSAGetLastError() == 10013){
            cerr << "（请以管理员身份执行该命令）" <<endl;
        }
        WSACleanup();
        return -1;
    }
    //设置端口属性
    int iTimeout = DEF_ICMP_TIMEOUT;
    if (setsockopt(sockRaw, SOL_SOCKET, SO_RCVTIMEO, (char*)&iTimeout, sizeof(iTimeout)) == SOCKET_ERROR)
    {
        cerr << "设置超时时间失败\n"
```

```
                << "错误代码: " << WSAGetLastError() << endl;
            closesocket(sockRaw);
            WSACleanup();
            return -1;
        }
        //创建 ICMP 包发送缓冲区和接收缓冲区
        char IcmpSendBuf[sizeof(ICMP_HEADER)+DEF_ICMP_DATA_SIZE];
        memset(IcmpSendBuf, 0, sizeof(IcmpSendBuf));
        char IcmpRecvBuf[MAX_ICMP_PACKET_SIZE];
        memset(IcmpRecvBuf, 0, sizeof(IcmpRecvBuf));
        //填充待发送的 ICMP 包
        ICMP_HEADER* pIcmpHeader = (ICMP_HEADER*)IcmpSendBuf;
        pIcmpHeader->type = ICMP_ECHO_REQUEST;
        pIcmpHeader->code = 0;
        pIcmpHeader->id = (USHORT)GetCurrentProcessId();
        memset(IcmpSendBuf+sizeof(ICMP_HEADER), 'E', DEF_ICMP_DATA_SIZE);
        //开始探测路由
        DECODE_RESULT stDecodeResult;
        BOOL bReachDestHost = FALSE;
        USHORT usSeqNo = 0;
        int iTTL = 1;
        int iMaxHop = DEF_MAX_HOP;
        while (!bReachDestHost && iMaxHop--)
        {
            //设置 IP 数据报头的 ttl 字段
            setsockopt(sockRaw, IPPROTO_IP, IP_TTL, (char*)&iTTL, sizeof(iTTL));
            //输出当前跳站数作为路由信息序号
            cout << setw(3) << iTTL << flush;
            //填充 ICMP 数据报剩余字段
            ((ICMP_HEADER*)IcmpSendBuf)->cksum = 0;
            ((ICMP_HEADER*)IcmpSendBuf)->seq = htons(usSeqNo++);
            ((ICMP_HEADER*)IcmpSendBuf)->cksum                                    =
GenerateChecksum((USHORT*)IcmpSendBuf,
sizeof(ICMP_HEADER)+DEF_ICMP_DATA_SIZE);

            //记录序列号和当前时间
            stDecodeResult.usSeqNo = ((ICMP_HEADER*)IcmpSendBuf)->seq;
            stDecodeResult.dwRoundTripTime = GetTickCount();

            //发送 ICMP 的 EchoRequest 数据报
            if (sendto(sockRaw, IcmpSendBuf, sizeof(IcmpSendBuf), 0,
                (sockaddr*)&destSockAddr, sizeof(destSockAddr)) == SOCKET_ERROR)
            {
                //如果目的主机不可达则直接退出
```

```
            if (WSAGetLastError() == WSAEHOSTUNREACH)
                cout << '\t'<< "目标主机不可达\n"<< "\n 追踪完成。" << endl;
            closesocket(sockRaw);
            WSACleanup();
            return 0;
        }
        //接收 ICMP 的 EchoReply 数据报
        //因为收到的可能并非程序所期待的数据报，所以需要循环接收直到收到所要数据或超时
        sockaddr_in from;
        int iFromLen = sizeof(from);
        int iReadDataLen;
        while (1)
        {
            //等待数据到达
            iReadDataLen = recvfrom(sockRaw, IcmpRecvBuf, MAX_ICMP_PACKET_SIZE,
                0, (sockaddr*)&from, &iFromLen);
            if (iReadDataLen != SOCKET_ERROR) //有数据包到达
            {
                //解码得到的数据包，如果解码正确则跳出接收循环发送下一个 EchoRequest 包
                if (DecodeIcmpResponse(IcmpRecvBuf, iReadDataLen, stDecodeResult))
                {
                    if (stDecodeResult.dwIPaddr.s_addr == destSockAddr.sin_addr.s_addr)
                        bReachDestHost = TRUE;
                    cout << '\t' << inet_ntoa(stDecodeResult.dwIPaddr) << endl;
                    break;
                }
            }
            else if (WSAGetLastError() == WSAETIMEDOUT) //接收超时，打印星号
            {
                cout << setw(9) << '*' << '\t' << "请求超时" << endl;
                break;
            }
            else
            {
                cerr << "\n 接收失败 \n"
                    << "错误代码: " << WSAGetLastError() << endl;
                closesocket(sockRaw);
                WSACleanup();
                return -1;
```

```
                }
            }
            //TTL 值加 1
            iTTL++;
        }
        //输出屏幕信息
        cout << "\n 追踪完成。" << endl;
        closesocket(sockRaw);
        WSACleanup();
        return 0;
}

//产生网际校验和
USHORT GenerateChecksum(USHORT* pBuf, int iSize)
{
        unsigned long cksum = 0;
        while (iSize>1)
        {
            cksum += *pBuf++;
            iSize -= sizeof(USHORT);
        }
        if (iSize)
            cksum += *(UCHAR*)pBuf;
        cksum = (cksum >> 16) + (cksum & 0xffff);
        cksum += (cksum >> 16);
        return (USHORT)(~cksum);
}
//解码得到的数据报
BOOL DecodeIcmpResponse(char* pBuf, int iPacketSize, DECODE_RESULT&
stDecodeResult)
{
        //检查数据报大小的合法性
        IP_HEADER* pIpHdr = (IP_HEADER*)pBuf;
        int iIpHdrLen = pIpHdr->hdr_len * 4;
        if (iPacketSize < (int)(iIpHdrLen+sizeof(ICMP_HEADER)))
            return FALSE;
        //按照 ICMP 包类型检查 id 字段和序列号以确定是否是程序应接收的 Icmp 包
        ICMP_HEADER* pIcmpHdr = (ICMP_HEADER*)(pBuf+iIpHdrLen);
        USHORT usID, usSquNo;
        if (pIcmpHdr->type == ICMP_ECHO_REPLY)
        {
            usID = pIcmpHdr->id;
            usSquNo = pIcmpHdr->seq;
        }
```

```
    else if(pIcmpHdr->type == ICMP_TIMEOUT)
    {
        char* pInnerIpHdr = pBuf+iIpHdrLen+sizeof(ICMP_HEADER);  //载荷中的IP 头
        int iInnerIPHdrLen = ((IP_HEADER*)pInnerIpHdr)->hdr_len * 4;//载荷中的 IP 头长
        ICMP_HEADER* pInnerIcmpHdr = (ICMP_HEADER*)(pInnerIpHdr+iInnerIPHdrLen);//载荷中的 ICMP 头
        usID = pInnerIcmpHdr->id;
        usSquNo = pInnerIcmpHdr->seq;
    }
    else
        return FALSE;
    if (usID != (USHORT)GetCurrentProcessId() || usSquNo !=stDecodeResult.usSeqNo)
        return FALSE;
    //处理正确收到的 ICMP 数据报
    if (pIcmpHdr->type == ICMP_ECHO_REPLY ||
        pIcmpHdr->type == ICMP_TIMEOUT)
    {
        //返回解码结果
        stDecodeResult.dwIPaddr.s_addr = pIpHdr->sourceIP;
        stDecodeResult.dwRoundTripTime = GetTickCount()-stDecodeResult.dwRoundTripTime;
        //打印屏幕信息
        if (stDecodeResult.dwRoundTripTime)
            cout << setw(6) << stDecodeResult.dwRoundTripTime << " ms" << flush;
        else
            cout << setw(6) << "<1" << " ms" << flush;
        return TRUE;
    }
    return FALSE;
}
```

7、实验报告

要求撰写实验报告描述各个实验过程以及遇到的问题，回答报告中的问题，总结对相应内容的认识（需要包含程序运行的截图）。

实验 3：基于 Winpcap 的协议分析器程序设计与实现

1、实验目的

熟悉 Winpcap 并掌握基于 Winpcap 的网络应用程序开发技术。

2、实验环境

- Windows 9x/NT/2000/XP/2003；
- 与因特网连接的计算机网络系统；
- 任何你熟悉的编程语言。

3、实验内容

Winpcap 实现抓包软件。

4、实验方式

每位同学上机实验，并与指导教师讨论。

5、实验过程

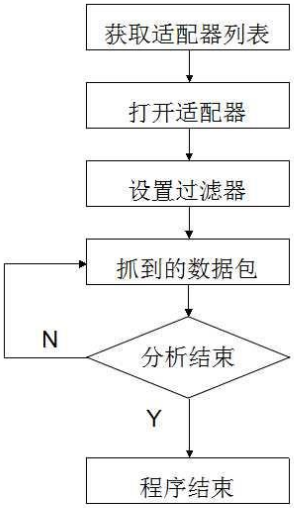
(1).使用 Winpcap 抓包首先使用 pcap_findalldevs()函数获取适配器列表，在程序的 CSelectAdapterDlg.cpp 文件中可以在 OnInitDialog()函数中可以看到使用方法。

(2).在获取适配器列表之后再打开适配器进行抓包，这时应该使用 pcap_open_live()函数或者 pcap_open()函数。Winpcap 还可以将抓到的数据包储存在堆文件中，如果想把抓到的数据包储存在本地的.cap 文件中，可以使用 pcap_dump_open()函数先将堆文件打开。现在抓包的初始化工作基本上已经做完。

(3).使用 pcap_open_live()函数将适配器打开之后就可以使用 pcap_next_ex()函数进行抓包了，其中 pcap_next_ex(pcap_t * p, struct pcap_pkthdr ** pkt_header, const u_char ** pkt_data)函数有三个参数，第一个参数是一个句柄，第二个参数是数据包的部分信息，包括时间戳，数据包长度等，第三个参数为抓到的数据包，这时可以使用 pcap_dump()函数将数据包的部分信息和数据包储存在刚刚打开的堆文件中。同样 Winpcap 同样可是使用 pcap_open_offline()函数将脱机堆文件打开。

(4).在抓取数据包时 Winpcap 提供了过滤数据包的方法，首先使用 pcap_compile()编译过滤器，然后在使用 pcap_setfilter()设置过滤器。当然也可以抓取经过网卡的所有数据包，在分析数据包时进行相应的手工过滤。

实验程序设计好的程序流程图如下：



6、参考内容

MFC 程序，部分代码需要同学们自己补充。

（一）CapPackDlg.cpp

```
#include "stdafx.h"
#include "CapPack.h"
#include "CapPackDlg.h"
#include ".\cappackdlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// 用于应用程序“关于”菜单项的 CAboutDlg 对话框

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// 对话框数据
	enum { IDD = IDD_ABOUTBOX };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持
```

```
// 实现
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()


// CCapPackDlg 对话框

CCapPackDlg::CCapPackDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CCapPackDlg::IDD, pParent)
	, m_EditValue(_T(""))
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CCapPackDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_LIST1, m_List1);
	DDX_Text(pDX, IDC_EDIT1, m_EditValue);
}

BEGIN_MESSAGE_MAP(CCapPackDlg, CDialog)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_CLOSE()
	//}}AFX_MSG_MAP
	ON_WM_DESTROY()
	ON_COMMAND(ID_StartCap, OnStartcap)
	ON_COMMAND(ID_StopCap, OnStopcap)
	ON_COMMAND(ID_SetAdapter, OnSetAdapter)
	ON_NOTIFY(LVN_ITEMCHANGED, IDC_LIST1, OnLvnItemchangedList1)
```

```
    ON_MESSAGE(WM_UPDATE_LIST, OnUpdateList)
    ON_COMMAND(ID_SetFilter, OnSetfilter)
    ON_COMMAND(ID_SaveFile, OnSavefile)
    ON_COMMAND(ID_OpenFile, OnOpenfile)
    ON_COMMAND(ID_Exit, OnExit)
    ON_COMMAND(ID_Clear, OnClear)
END_MESSAGE_MAP()


// CCapPackDlg 消息处理程序
void CCapPackDlg::OnOK()
{

}
BOOL CCapPackDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // 将\“关于...\”菜单项添加到系统菜单中。

    // IDM_ABOUTBOX 必须在系统命令范围内。
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING,          IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动
    //  执行此操作
    SetIcon(m_hIcon, TRUE);          // 设置大图标
    SetIcon(m_hIcon, FALSE);        // 设置小图标

//-------------------------------------------------
    // TODO: 在此添加额外的初始化代码
```

```
    CRect rect;
    UINT ind[]={0,1};//运行状态  有效包数/总包数

    //加载菜单
    m_Menu.LoadMenu(IDR_MENU1);
    SetMenu(&m_Menu);
    //加载工具栏
    GetClientRect(&rect);
    m_ToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
    m_ToolBar.LoadToolBar(IDR_MENU1);
    m_ToolBar.SetDlgItemText(ID_StartCap,"Asfsaf");
    m_ToolBar.MoveWindow(rect.left,rect.top,rect.right-rect.left,20,1);
    //加载状态栏
    m_StatusBar.Create(this);
    m_StatusBar.SetIndicators(ind,sizeof(ind)/sizeof(UINT));
    m_StatusBar.MoveWindow(rect.left,rect.bottom-20,rect.right,20,1);
    m_StatusBar.SetPaneInfo(0,0,SBPS_STRETCH,(rect.right-rect.left)/5);
    m_StatusBar.SetPaneInfo(1,0,0,(rect.right-rect.left)*4/5);
    m_StatusBar.SetPaneText(0,"Ready...");
    m_StatusBar.SetPaneText(1,"Packets Count: Receive 0, Filter 0");

    //设置列表框
    m_List1.SetExtendedStyle( LVS_EX_GRIDLINES | LVS_EX_FULLROWSELECT | LVS_EX_HEADERDRAGDROP);
    m_List1.InsertColumn(0,"编号",LVCFMT_LEFT,50);
    m_List1.InsertColumn(1,"协议",LVCFMT_LEFT,40);
    m_List1.InsertColumn(2,"源 IP",LVCFMT_LEFT,90);
    m_List1.InsertColumn(3,"源 MAC",LVCFMT_LEFT,110);
    m_List1.InsertColumn(4,"源端口",LVCFMT_LEFT,40);
    m_List1.InsertColumn(5,"目的 IP",LVCFMT_LEFT,90);
    m_List1.InsertColumn(6,"目的 MAC",LVCFMT_LEFT,110);
    m_List1.InsertColumn(7,"目的端口",LVCFMT_LEFT,40);
    m_List1.InsertColumn(8,"长度",LVCFMT_LEFT,40);
    m_List1.InsertColumn(9,"内容",LVCFMT_LEFT,280);

    //设置菜单变灰
    Status=0;
    SetMenuStatus();

    InitializeCriticalSection(&csThreadStop);
    eThreadStart=CreateEvent(0,
        false,//自动 Reset
```

```
            false,//初始阻塞
            "ThreadStartCap");

        FilterSet.bAllProtocol=true;
        FilterSet.bArp=FilterSet.bIcmp=FilterSet.bUdp=FilterSet.bTcp=false;
        FilterSet.bAllIP=true;
        strcpy(FilterSet.IP,"");
        FilterSet.bAllPort=true;
        strcpy(FilterSet.Port,"");
        GetDlgItem(IDC_LIST3)->ShowWindow(0);

        return TRUE;  // 除非设置了控件的焦点，否则返回 TRUE
}

void CCapPackDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
        }
        else
        {
            CDialog::OnSysCommand(nID, lParam);
        }
}

// 如果向对话框添加最小化按钮，则需要下面的代码
//  来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，
//  这将由框架自动完成。

void CCapPackDlg::OnPaint()
{
        if (IsIconic())
        {
            CPaintDC dc(this); // 用于绘制的设备上下文

            SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

            // 使图标在工作矩形中居中
            int cxIcon = GetSystemMetrics(SM_CXICON);
            int cyIcon = GetSystemMetrics(SM_CYICON);
            CRect rect;
```

```
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;

            // 绘制图标
            dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
            CDialog::OnPaint();
        }
}

//当用户拖动最小化窗口时系统调用此函数取得光标显示。
HCURSOR CCapPackDlg::OnQueryDragIcon()
{
        return static_cast<HCURSOR>(m_hIcon);
}

void CCapPackDlg::OnStartcap()
{
        BOOL temp;
        char FilePath[MAX_PATH];
        m_List1.DeleteAllItems();

        GetModuleFileName(0,FilePath,MAX_PATH-1);
        CapFilePath=FilePath;
        CapFilePath=CapFilePath.Left(CapFilePath.ReverseFind('\\'));
        CapFilePath+="\\CapData.CAP";

        hCapThread=AfxBeginThread(CapThread,(LPVOID)this);

        WaitForSingleObject(eThreadStart, INFINITE);
        EnterCriticalSection(&csThreadStop);
        temp=bThreadStop;
        LeaveCriticalSection(&csThreadStop);
        if(temp)//线程自动退出
        {
            return;
        }
        else
        {
            Status=1;
            GetDlgItem(IDC_EDIT1)->SetWindowText("");
```

```
            GetDlgItem(IDC_EDIT2)->SetWindowText("");
            this->SetMenuStatus();
        }
}

void CCapPackDlg::OnStopcap()
{
        DWORD result;

        EnterCriticalSection(&csThreadStop);
        bThreadStop=true;
        LeaveCriticalSection(&csThreadStop);
        result=WaitForSingleObject(hCapThread->m_hThread,3000);//可能在这行执行前线
程就已退出
        if(result==WAIT_TIMEOUT )
        {
            MessageBox("工作线程超时不响应，强制结束！");
            TerminateThread(hCapThread->m_hThread,0);
        }

        Status=0;
        this->SetMenuStatus();

        //MessageBox("停止抓包");
}
void CCapPackDlg::SetMenuStatus()
{
        CMenu *pSubMenu;

        switch(Status)
        {
        case 0:
            //开始抓包菜单
            pSubMenu = m_Menu.GetSubMenu(1);
            pSubMenu->EnableMenuItem(0,MF_BYPOSITION| MF_ENABLED);
            //停止抓包菜单
            pSubMenu = m_Menu.GetSubMenu(1);
            pSubMenu->EnableMenuItem(1,MF_BYPOSITION|MF_GRAYED);
            //设置菜单
            pSubMenu = m_Menu.GetSubMenu(2);
            pSubMenu->EnableMenuItem(0,MF_BYPOSITION|MF_ENABLED);
            pSubMenu->EnableMenuItem(1,MF_BYPOSITION|MF_ENABLED);
            //工具栏
            m_ToolBar.GetToolBarCtrl().EnableButton(ID_StartCap,true);
```

```
            m_ToolBar.GetToolBarCtrl().EnableButton(ID_StopCap,false);
            m_ToolBar.GetToolBarCtrl().EnableButton(ID_SetFilter,true);
            m_ToolBar.GetToolBarCtrl().EnableButton(ID_SetAdapter,true);
            //状态栏
            m_StatusBar.SetPaneText(0,"Ready...");
            break;
        case 1:
            //开始抓包菜单
            pSubMenu = m_Menu.GetSubMenu(1);
            pSubMenu->EnableMenuItem(0,MF_BYPOSITION| MF_GRAYED);
            //停止抓包菜单
            pSubMenu = m_Menu.GetSubMenu(1);
            pSubMenu->EnableMenuItem(1,MF_BYPOSITION| MF_ENABLED);
            //设置菜单
            pSubMenu = m_Menu.GetSubMenu(2);
            pSubMenu->EnableMenuItem(0,MF_BYPOSITION|MF_GRAYED);
            pSubMenu->EnableMenuItem(1,MF_BYPOSITION|MF_GRAYED);
            //工具栏
            m_ToolBar.GetToolBarCtrl().EnableButton(ID_StartCap,false);
            m_ToolBar.GetToolBarCtrl().EnableButton(ID_StopCap,true);
            m_ToolBar.GetToolBarCtrl().EnableButton(ID_SetFilter,false);
            m_ToolBar.GetToolBarCtrl().EnableButton(ID_SetAdapter,false);
            //状态栏
            m_StatusBar.SetPaneText(0,"Cap...");
            break;
        default:
            MessageBox("错误的状态变量值");

        }

}


void CCapPackDlg::OnSetAdapter()
{
        CString temp;

        CSelectAdapterDlg SelectDlg;
        SelectDlg.pName=&temp;
        INT_PTR nResponse=SelectDlg.DoModal();
        AdapterName=temp;
}
LRESULT CCapPackDlg::OnUpdateList(WPARAM wParam, LPARAM lParam)
{
```

```
    ListData *list;
    list=(ListData *)wParam;
    char temp[50];
    int i,j;


    if(list->Falg)
    {
        i=atoi(list->ID);
        j=atoi(list->TotalPacket);
        sprintf(temp,"Packets Count: Receive %d, Filter %d",j,i);
        m_StatusBar.SetPaneText(1,temp);
        //设置状态栏
    }
    else
    {
        i=atoi(list->ID);
        j=atoi(list->TotalPacket);
        //设置状态栏
        sprintf(temp,"Packets Count: Receive %d, Filter %d",j,i);
        m_StatusBar.SetPaneText(1,temp);
        //
        i-=1;
        itoa(j-1,temp,10);
        i=m_List1.InsertItem(i,temp);
        m_List1.SetItemText(i,1,list->Protocol);
        m_List1.SetItemText(i,2,list->sIP);
        m_List1.SetItemText(i,3,list->sMac);
        m_List1.SetItemText(i,4,list->sPort);
        m_List1.SetItemText(i,5,list->dIP);
        m_List1.SetItemText(i,6,list->dMac);
        m_List1.SetItemText(i,7,list->dPort);
        m_List1.SetItemText(i,8,list->Len);
        m_List1.SetItemText(i,9,list->Text);
    }


    return 0;
}


UINT CCapPackDlg::CapThread(LPVOID lpParameter)
{
    CCapPackDlg *this2;
```

```
pcap_t *adhandle;
char errbuf[PCAP_ERRBUF_SIZE];
pcap_dumper_t *dumpfile;
struct pcap_pkthdr *header;
const u_char *data;
DWORD res;
bool bExit;
static long i=0,j=0;
bool bFilter=false;//true 表示过滤掉

ListData List;

struct ether_header *eth;
u_char* mac_string;
struct iphead *IPHead;
struct arphead *ARPHead;
in_addr ipaddr;

this2=(CCapPackDlg *)lpParameter;
i=0;
j=0;
if((adhandle= pcap_open_live(this2->AdapterName,65536,1 ,10,errbuf)) == NULL)
{
    ::MessageBox(0,"不能打开网络适配器,请在网卡设置中经行设置","错误",0);

    EnterCriticalSection(&(this2->csThreadStop));
    this2->bThreadStop=true;
    LeaveCriticalSection(&(this2->csThreadStop));
    SetEvent(this2->eThreadStart);

    return 0;
}

dumpfile=pcap_dump_open(adhandle,this2->CapFilePath);
if(dumpfile==NULL)
{
    ::MessageBox(0,"不能打开记录文件","错误",0);

    EnterCriticalSection(&(this2->csThreadStop));
    this2->bThreadStop=true;
    LeaveCriticalSection(&(this2->csThreadStop));
    SetEvent(this2->eThreadStart);
    return 0;
}
```

```
    EnterCriticalSection(&(this2->csThreadStop));
    this2->bThreadStop=false;
    LeaveCriticalSection(&(this2->csThreadStop));
    SetEvent(this2->eThreadStart);

    //::MessageBox(0,"开始抓包","ok",0);
    while(1)
    {
        EnterCriticalSection(&(this2->csThreadStop));
        bExit=this2->bThreadStop;
        LeaveCriticalSection(&(this2->csThreadStop));
        if(bExit)  return 0;

        res = pcap_next_ex(adhandle,&header,&data);
        if(res==0)
        {
            Sleep(100);
            continue;
        }
        else if(res<0)
        {
            break;
        }
        pcap_dump((u_char *)dumpfile, header, data);

        eth=(ether_header *)data;
        mac_string=eth->ether_shost;

    sprintf(List.sMac,"%02X:%02X:%02X:%02X:%02X:%02X",*mac_string,*(mac_stri
ng+1),*(mac_string+2),*(mac_string+3),*(mac_string+4),*(mac_string+5));
        mac_string=eth->ether_dhost;

    sprintf(List.dMac,"%02X:%02X:%02X:%02X:%02X:%02X",*mac_string,*(mac_stri
ng+1),*(mac_string+2),*(mac_string+3),*(mac_string+4),*(mac_string+5));
        ltoa(header->caplen,List.Len,10);
        memcpy(List.Text,data,45);//数据不含以太网头
        List.Text[45]='\0';
        this2->DecodeChar(List.Text,45);

        switch(ntohs(eth->ether_type))
        {
        case ETHERTYPE_ARP:
```

```
                if(!this2->FilterSet.bAllProtocol
&& !this2->FilterSet.bArp){bFilter=true;break;}
                strcpy(List.Protocol,"ARP");
                ARPHead=(arphead *)(data+14);


    sprintf(List.sIP,"%d.%d.%d.%d",ARPHead->arp_source_ip_address[0],ARPHead->ar
p_source_ip_address[1],ARPHead->arp_source_ip_address[2],ARPHead->arp_source_ip_
address[3]);

    sprintf(List.dIP,"%d.%d.%d.%d",ARPHead->arp_destination_ip_address[0],ARPHea
d->arp_destination_ip_address[1],ARPHead->arp_destination_ip_address[2],ARPHead->a
rp_destination_ip_address[3]);
                strcpy(List.sPort,"--");
                strcpy(List.dPort,"--");
                break;
            case ETHERTYPE_REVARP:
                strcpy(List.Protocol,"RARP");
                break;
            case ETHERTYPE_IP:
                IPHead=(iphead *)(data+14);
                ipaddr=IPHead->ip_souce_address;

    sprintf(List.sIP,"%d.%d.%d.%d",ipaddr.S_un.S_un_b.s_b1,ipaddr.S_un.S_un_b.s_b2,i
paddr.S_un.S_un_b.s_b3,ipaddr.S_un.S_un_b.s_b4);
                IPHead->ip_destination_address;

    sprintf(List.dIP,"%d.%d.%d.%d",ipaddr.S_un.S_un_b.s_b1,ipaddr.S_un.S_un_b.s_b2,i
paddr.S_un.S_un_b.s_b3,ipaddr.S_un.S_un_b.s_b4);
                switch(IPHead->ip_protocol)
                {
                case 1:
                    if(!this2->FilterSet.bAllProtocol
&& !this2->FilterSet.bIcmp){bFilter=true;break;}
                    strcpy(List.Protocol,"ICMP");
                    strcpy(List.sPort,"--");
                    strcpy(List.dPort,"--");
                    break;
                case 6:
                    if(!this2->FilterSet.bAllProtocol
&& !this2->FilterSet.bTcp){bFilter=true;break;}
                    strcpy(List.Protocol,"TCP");
                    sprintf(List.sPort,"%d",ntohs( ((tcphead *)(data+16+20))->th_sport ));
                    sprintf(List.dPort,"%d",ntohs( ((tcphead *)(data+16+20))->th_dport ));
```

```
                    break;
                case 17:
                    if(!this2->FilterSet.bAllProtocol
&& !this2->FilterSet.bUdp){bFilter=true;break;}
                    strcpy(List.Protocol,"UDP");
                    sprintf(List.sPort,"%d",ntohs(                    ((udphead
*)(data+16+20))->udp_source_port ));
                    sprintf(List.dPort,"%d",ntohs(                    ((udphead
*)(data+16+20))->udp_destinanion_port ));
                    break;
                default:
                    strcpy(List.Protocol,"未知 IP 包");
                    strcpy(List.sIP,"----------");
                    strcpy(List.dIP,"----------");
                    strcpy(List.sPort,"--");
                    strcpy(List.dPort,"--");
                    break;
                }

                break;
            case ETHERTYPE_PUP:
                strcpy(List.Protocol,"PUP");
                strcpy(List.sIP,"----------");
                strcpy(List.dIP,"----------");
                strcpy(List.sPort,"--");
                strcpy(List.dPort,"--");
                break;
            default:
                strcpy(List.Protocol,"未知以太包");
                strcpy(List.sIP,"----------");
                strcpy(List.dIP,"----------");
                strcpy(List.sPort,"--");
                strcpy(List.dPort,"--");
                break;
            }
            if(bFilter)
            {
                j++;
                List.Falg=true;
            }
            else
            {
                i++;j++;
                List.Falg=false;
```

```
            }
            ltoa(i,List.ID,10);
            ltoa(j,List.TotalPacket,10);

        SendMessageTimeout(this2->m_hWnd,WM_UPDATE_LIST,(WPARAM)&List,0,S
MTO_BLOCK,1000,&res);
            bFilter=false;
        }
        return 0;
}
void CCapPackDlg::OnLvnItemchangedList1(NMHDR *pNMHDR, LRESULT *pResult)
{
        LPNMLISTVIEW pNMLV = reinterpret_cast<LPNMLISTVIEW>(pNMHDR);
        // TODO: 在此添加控件通知处理程序代码
        if(     !(((NM_LISTVIEW*)pNMHDR)->uNewState     &     LVIS_SELECTED)     ||
(((NM_LISTVIEW*)pNMHDR)->uOldState & LVIS_SELECTED) )
        {
            *pResult = 0;
            return;
        }



        int i,j;
        pcap_t *fp;
        char errbuf[PCAP_ERRBUF_SIZE];
        struct pcap_pkthdr *header;
        const u_char *data;
        char *p;



        for(i =0;i<m_List1.GetItemCount();i++ )
        {
            if(m_List1.GetItemState(i,LVIS_SELECTED)==LVIS_SELECTED)// 选 中 状
态。。
            {
                break;
            }
        }

        //MessageBox(m_List1.GetItemText(i,0));
        i=atoi(m_List1.GetItemText(i,0));
```

```
    if ((fp = pcap_open_offline(CapFilePath,errbuf)) == NULL)
    {
        MessageBox("不能找到记录文件");
        return;
    }

    for(j=0;j<i;j++)//找到选中项
        pcap_next_ex(fp, &header, &data);


    if(pcap_next_ex(fp, &header, &data)>0)
    {
        AnalysePacket(header,data);
        p=(char *)malloc((header->caplen+1)*sizeof(u_char));
        memcpy(p,data,header->caplen);
        p[header->caplen]='\0';
        DecodeChar(p,header->caplen);
        GetDlgItem(IDC_EDIT1)->SetWindowText(p);
        free(p);
    }

    *pResult = 0;
}
void CCapPackDlg::DecodeChar(char *data,DWORD len)
{
    DWORD i;
    for(i=0;i<len;i++)
        if(data[i]=='\0')
            data[i]='.';
}
void CCapPackDlg::OnClose()
{
    if(Status)
    {
        MessageBox("请先停止抓包!");
        return;
    }
    DeleteCriticalSection(&csThreadStop);

    CDialog::OnClose();
}
void CCapPackDlg::OnSetfilter()
{
    Filter temp;
```

```
    CFilterDlg FilterDlg;
    memcpy(&temp,&FilterSet,sizeof(temp));
    FilterDlg.pFilterSet=&temp;
    INT_PTR nResponse=FilterDlg.DoModal();
    if(nResponse == IDOK)
    {
        memcpy(&FilterSet,&temp,sizeof(temp));
    }

}
void CCapPackDlg::AnalysePacket(const pcap_pkthdr *header,const u_char *data)
{
    CString AnalyseStr,temp;

    //--------------------------------------------以太网变量
    struct ether_header *eth; //以太网帧报头指针
    unsigned int ptype; //协议类型变量
    char mac_addr[19];
    u_char* mac_string;
    //-------------------------
    struct iphead *IPHead;
    //-------------------------
    struct arphead *ARPHead;

    AnalyseStr.Format("以太网帧长度:%d\r\n",header->caplen);

    eth=(struct ether_header *)data;
    mac_string=eth->ether_shost;
    sprintf(mac_addr,"%02X:%02X:%02X:%02X:%02X:%02X",*mac_string,*(mac_stri
ng+1),*(mac_string+2),*(mac_string+3),*(mac_string+4),*(mac_string+5));
    AnalyseStr+=(CString)"源 MAC 地址:"+mac_addr+(CString)"\r\n";
    mac_string=eth->ether_dhost;
    sprintf(mac_addr,"%02X:%02X:%02X:%02X:%02X:%02X",*mac_string,*(mac_stri
ng+1),*(mac_string+2),*(mac_string+3),*(mac_string+4),*(mac_string+5));
    AnalyseStr+=(CString)"目的 MAC 地址:"+mac_addr+(CString)"\r\n";

    AnalyseStr+="以太网帧类型:";
    ptype=ntohs(eth->ether_type);
    switch(ptype)
    {
    case ETHERTYPE_ARP:
        AnalyseStr+="ARP 包\r\n";
        AnalyseStr+="----------\r\n";
```

```
            ARPHead=(arphead *)(data+14);

            temp.Format("硬件类型:%d BYTE\r\n",ntohs(ARPHead->arp_hardware_type));
            AnalyseStr+=temp;
            temp.Format("ARP 包协议类型:%d\r\n",ntohs(ARPHead->arp_protocol_type));
            AnalyseStr+=temp;
            temp.Format("硬件长度:%d\r\n",ntohs(ARPHead->arp_hardware_length));
            AnalyseStr+=temp;
            temp.Format("协议长度:%d\r\n",ntohs(ARPHead->arp_protocol_length));
            AnalyseStr+=temp;
            temp.Format("ARP 操作码:%d (请求 1, 回答
2)\r\n",ntohs(ARPHead->arp_operation_code));
            AnalyseStr+=temp;
            mac_string=ARPHead->arp_source_ethernet_address;
            temp.Format("ARP 包发送方
MAC:%02X:%02X:%02X:%02X:%02X:%02X\r\n",*mac_string,*(mac_string+1),*(mac_
string+2),*(mac_string+3),*(mac_string+4),*(mac_string+5));
            AnalyseStr+=temp;
            temp.Format("ARP 包发送方
IP:%d.%d.%d.%d\r\n",ARPHead->arp_source_ip_address[0],ARPHead->arp_source_ip_a
ddress[1],ARPHead->arp_source_ip_address[2],ARPHead->arp_source_ip_address[3]);
            AnalyseStr+=temp;
            mac_string=ARPHead->arp_destination_ethernet_address;
            temp.Format("ARP 包接收方
MAC:%02X:%02X:%02X:%02X:%02X:%02X\r\n",*mac_string,*(mac_string+1),*(mac_
string+2),*(mac_string+3),*(mac_string+4),*(mac_string+5));
            AnalyseStr+=temp;
            temp.Format("ARP 包接收方
IP:%d.%d.%d.%d\r\n",ARPHead->arp_destination_ip_address[0],ARPHead->arp_destinati
on_ip_address[1],ARPHead->arp_destination_ip_address[2],ARPHead->arp_destination_i
p_address[3]);
            AnalyseStr+=temp;

            break;
        case ETHERTYPE_REVARP:
            AnalyseStr+="RARP 包\r\n";
            break;
        case ETHERTYPE_IP:
            AnalyseStr+="IP 包\r\n";
            IPHead=(iphead *)(data+14);
            AnalyseStr+="----------------------\r\n";

            temp.Format("IP 头长:%d BYTE\r\n",(IPHead->ip_header_length)*4);
            AnalyseStr+=temp;
```

```
            temp.Format("IP 版本号:%d\r\n",IPHead->ip_version);
            AnalyseStr+=temp;
            temp.Format("IP 服务类型:%d\r\n",ntohs(IPHead->ip_tos));
            AnalyseStr+=temp;
            temp.Format("IP 包总长度:%d\r\n",ntohs(IPHead->ip_length));////
            AnalyseStr+=temp;
            temp.Format("IP 包标识:%d\r\n",ntohs(IPHead->ip_id));//////
            AnalyseStr+=temp;

            temp.Format("IP 包分片标志(DF):%ld\r\n",(ntohs(IPHead->ip_off) &
0X4000)>>14);////////
            AnalyseStr+=temp;
            temp.Format("IP 包分片标志(MF):%ld\r\n",(ntohs(IPHead->ip_off) &
0X2000)>>13);////////
            AnalyseStr+=temp;
            temp.Format("IP 包分片偏移:%ld BYTE\r\n",8*(ntohs(IPHead->ip_off) &
0X1FFF));/////////
            AnalyseStr+=temp;

            temp.Format("IP 包生存时间:%d\r\n",(IPHead->ip_ttl));////////
            AnalyseStr+=temp;
            temp.Format("IP 包检验和:%0X\r\n",ntohs(IPHead->ip_checksum));/////////
            AnalyseStr+=temp;
            temp.Format("IP 包源
IP:%d.%d.%d.%d\r\n",IPHead->ip_souce_address.S_un.S_un_b.s_b1,IPHead->ip_souce_a
ddress.S_un.S_un_b.s_b2,IPHead->ip_souce_address.S_un.S_un_b.s_b3,IPHead->ip_souc
e_address.S_un.S_un_b.s_b4);
            AnalyseStr+=temp;
            temp.Format("IP 包目的
IP:%d.%d.%d.%d\r\n",IPHead->ip_destination_address.S_un.S_un_b.s_b1,IPHead->ip_de
stination_address.S_un.S_un_b.s_b2,IPHead->ip_destination_address.S_un.S_un_b.s_b3,I
PHead->ip_destination_address.S_un.S_un_b.s_b4);
            AnalyseStr+=temp;

            AnalyseStr+="IP 协议:";
            switch(IPHead->ip_protocol)
            {
            case 1:
                AnalyseStr+="ICMP\r\n";

      //Analyse_ICMPPacket(&(IPHead->ip_souce_address),&(IPHead->ip_destination_ad
dress),data+20);
                break;
            case 6:
```

```
                AnalyseStr+="TCP\r\n";

    //Analyse_TCPPacket(&(IPHead->ip_souce_address),&(IPHead->ip_destination_addr
ess),data+20);
                break;
            case 17:
                AnalyseStr+="UDP\r\n";

    //Analyse_UDPPacket(&(IPHead->ip_souce_address),&(IPHead->ip_destination_add
ress),data+20);
                break;
            default:
                temp.Format("%d(未知)\r\n",IPHead->ip_protocol);
                AnalyseStr+=temp;
                break;
            }
            break;
        case ETHERTYPE_PUP:
            AnalyseStr+="PUP\r\n";
            //printf("PUP\n");
            break;
        default:
            AnalyseStr+="未知\r\n";
            break;
        }


        GetDlgItem(IDC_EDIT2)->SetWindowText(AnalyseStr);
}

void CCapPackDlg::OnSavefile()
{
        //MessageBox("asdfsdaf");
}
void CCapPackDlg::OnOpenfile()
{
          CString path;
        pcap_t *fp;
        char errbuf[PCAP_ERRBUF_SIZE];
        struct pcap_pkthdr *header;
        const u_char *data;
        char filter[]="CAP File(*.CAP)|*.CAP|CAP File(*.*)|*.*||";

        struct ether_header *eth;
```

```
    u_char* mac_string;
    struct iphead *IPHead;
    struct arphead *ARPHead;
    in_addr ipaddr;
    ListData List;
    int i=0;

    m_List1.DeleteAllItems();
    CFileDialog file(true, NULL,"*.CAP",OFN_HIDEREADONLY,filter,NULL );
    file.DoModal();
    path=file.GetPathName();
    CapFilePath=path;

    if ((fp = pcap_open_offline(path,errbuf)) == NULL)
    {
        MessageBox("不能打开记录文件");
        return;
    }


    while(pcap_next_ex(fp, &header, &data)>0)
    {
        eth=(ether_header *)data;
        mac_string=eth->ether_shost;

    sprintf(List.sMac,"%02X:%02X:%02X:%02X:%02X:%02X",*mac_string,*(mac_stri
ng+1),*(mac_string+2),*(mac_string+3),*(mac_string+4),*(mac_string+5));
        mac_string=eth->ether_dhost;

    sprintf(List.dMac,"%02X:%02X:%02X:%02X:%02X:%02X",*mac_string,*(mac_stri
ng+1),*(mac_string+2),*(mac_string+3),*(mac_string+4),*(mac_string+5));
        ltoa(header->caplen,List.Len,10);
        memcpy(List.Text,data,45);//数据不含以太网头
        List.Text[45]='\0';
        this->DecodeChar(List.Text,45);

        switch(ntohs(eth->ether_type))
        {
        case ETHERTYPE_ARP:
            strcpy(List.Protocol,"ARP");
            ARPHead=(arphead *)(data+14);


    sprintf(List.sIP,"%d.%d.%d.%d",ARPHead->arp_source_ip_address[0],ARPHead->ar
```

```
p_source_ip_address[1],ARPHead->arp_source_ip_address[2],ARPHead->arp_source_ip_
address[3]);

    sprintf(List.dIP,"%d.%d.%d.%d",ARPHead->arp_destination_ip_address[0],ARPHea
d->arp_destination_ip_address[1],ARPHead->arp_destination_ip_address[2],ARPHead->a
rp_destination_ip_address[3]);
                strcpy(List.sPort,"--");
                strcpy(List.dPort,"--");
                break;
            case ETHERTYPE_REVARP:
                strcpy(List.Protocol,"RARP");
                break;
            case ETHERTYPE_IP:
                IPHead=(iphead *)(data+14);
                ipaddr=IPHead->ip_souce_address;

    sprintf(List.sIP,"%d.%d.%d.%d",ipaddr.S_un.S_un_b.s_b1,ipaddr.S_un.S_un_b.s_b2,i
paddr.S_un.S_un_b.s_b3,ipaddr.S_un.S_un_b.s_b4);
                IPHead->ip_destination_address;

    sprintf(List.dIP,"%d.%d.%d.%d",ipaddr.S_un.S_un_b.s_b1,ipaddr.S_un.S_un_b.s_b2,i
paddr.S_un.S_un_b.s_b3,ipaddr.S_un.S_un_b.s_b4);
                switch(IPHead->ip_protocol)
                {
                case 1:
                    strcpy(List.Protocol,"ICMP");
                    strcpy(List.sPort,"--");
                    strcpy(List.dPort,"--");
                    break;
                case 6:
                    strcpy(List.Protocol,"TCP");
                    sprintf(List.sPort,"%d",ntohs( ((tcphead *)(data+16+20))->th_sport ));
                    sprintf(List.dPort,"%d",ntohs( ((tcphead *)(data+16+20))->th_dport ));
                    break;
                case 17:
                    strcpy(List.Protocol,"UDP");
                    sprintf(List.sPort,"%d",ntohs(                    ((udphead
*)(data+16+20))->udp_source_port ));
                    sprintf(List.dPort,"%d",ntohs(                    ((udphead
*)(data+16+20))->udp_destinanion_port ));
                    break;
                default:
                    strcpy(List.Protocol,"未知 IP 包");
                    strcpy(List.sIP,"----------");
```

```
                    strcpy(List.dIP,"----------");
                    strcpy(List.sPort,"--");
                    strcpy(List.dPort,"--");
                    break;
                }

                break;
            case ETHERTYPE_PUP:
                strcpy(List.Protocol,"PUP");
                strcpy(List.sIP,"----------");
                strcpy(List.dIP,"----------");
                strcpy(List.sPort,"--");
                strcpy(List.dPort,"--");
                break;
            default:
                strcpy(List.Protocol,"未知以太包");
                strcpy(List.sIP,"----------");
                strcpy(List.dIP,"----------");
                strcpy(List.sPort,"--");
                strcpy(List.dPort,"--");
                break;
            }
            ltoa(i,List.ID,10);
            m_List1.InsertItem(i,List.ID);
            m_List1.SetItemText(i,1,List.Protocol);
            m_List1.SetItemText(i,2,List.sIP);
            m_List1.SetItemText(i,3,List.sMac);
            m_List1.SetItemText(i,4,List.sPort);
            m_List1.SetItemText(i,5,List.dIP);
            m_List1.SetItemText(i,6,List.dMac);
            m_List1.SetItemText(i,7,List.dPort);
            m_List1.SetItemText(i,8,List.Len);
            m_List1.SetItemText(i,9,List.Text);
            i++;
        }

}

void CCapPackDlg::OnExit()
{
        CDialog::OnOK();
}

void CCapPackDlg::OnClear()
```

```
{
	m_List1.DeleteAllItems();
	GetDlgItem(IDC_EDIT1)->SetWindowText("");
	GetDlgItem(IDC_EDIT2)->SetWindowText("");
}
```

（二）FilterDlg.cpp

```
// FilterDlg.cpp : 实现文件
//

#include "stdafx.h"
#include "CapPack.h"
#include "FilterDlg.h"
#include ".\filterdlg.h"


// CFilterDlg 对话框


IMPLEMENT_DYNAMIC(CFilterDlg, CDialog)
CFilterDlg::CFilterDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CFilterDlg::IDD, pParent)
{
}

CFilterDlg::~CFilterDlg()
{
}

void CFilterDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}


BEGIN_MESSAGE_MAP(CFilterDlg, CDialog)
	ON_BN_CLICKED(IDC_RADIO2, OnBnClickedRadio2)
	ON_BN_CLICKED(IDC_RADIO1, OnBnClickedRadio1)
	ON_BN_CLICKED(IDC_RADIO4, OnBnClickedRadio4)
	ON_BN_CLICKED(IDC_RADIO3, OnBnClickedRadio3)
	ON_BN_CLICKED(IDC_RADIO6, OnBnClickedRadio6)
	ON_BN_CLICKED(IDC_RADIO5, OnBnClickedRadio5)
	ON_BN_CLICKED(IDOK, OnBnClickedOk)
	ON_BN_CLICKED(IDCANCEL, OnBnClickedCancel)
```

```
END_MESSAGE_MAP()


// CFilterDlg 消息处理程序
BOOL CFilterDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	if(pFilterSet->bAllProtocol)
	{
		CheckDlgButton(IDC_RADIO1,1);
		GetDlgItem(IDC_CHECK1)->EnableWindow(0);
		GetDlgItem(IDC_CHECK2)->EnableWindow(0);
		GetDlgItem(IDC_CHECK3)->EnableWindow(0);
		GetDlgItem(IDC_CHECK4)->EnableWindow(0);
	}
	else
	{
		CheckDlgButton(IDC_RADIO2,1);
		if(pFilterSet->bArp)
			CheckDlgButton(IDC_CHECK1,1);
		if(pFilterSet->bIcmp)
			CheckDlgButton(IDC_CHECK2,1);
		if(pFilterSet->bUdp)
			CheckDlgButton(IDC_CHECK3,1);
		if(pFilterSet->bTcp)
			CheckDlgButton(IDC_CHECK4,1);
	}
	if(pFilterSet->bAllIP)
	{
		CheckDlgButton(IDC_RADIO3,1);
		GetDlgItem(IDC_IPADDRESS1)->EnableWindow(0);
	}
	else
	{
		CheckDlgButton(IDC_RADIO4,1);
		GetDlgItem(IDC_IPADDRESS1)->SetWindowText(pFilterSet->IP);
	}
	if(pFilterSet->bAllPort)
	{
		CheckDlgButton(IDC_RADIO5,1);
		GetDlgItem(IDC_EDIT1)->EnableWindow(0);
	}
	else
	{
```

```
            CheckDlgButton(IDC_RADIO6,1);
            GetDlgItem(IDC_EDIT1)->SetWindowText(pFilterSet->Port);
        }
        return true;
}
void CFilterDlg::OnBnClickedRadio2()
{
        pFilterSet->bAllProtocol=false;
        GetDlgItem(IDC_CHECK1)->EnableWindow(1);
        GetDlgItem(IDC_CHECK2)->EnableWindow(1);
        GetDlgItem(IDC_CHECK3)->EnableWindow(1);
        GetDlgItem(IDC_CHECK4)->EnableWindow(1);
}

void CFilterDlg::OnBnClickedRadio1()
{
        pFilterSet->bAllProtocol=true;
        GetDlgItem(IDC_CHECK1)->EnableWindow(0);
        GetDlgItem(IDC_CHECK2)->EnableWindow(0);
        GetDlgItem(IDC_CHECK3)->EnableWindow(0);
        GetDlgItem(IDC_CHECK4)->EnableWindow(0);
}

void CFilterDlg::OnBnClickedRadio4()
{
        pFilterSet->bAllIP=false;
        GetDlgItem(IDC_IPADDRESS1)->EnableWindow(1);
}

void CFilterDlg::OnBnClickedRadio3()
{
        pFilterSet->bAllIP=true;
        GetDlgItem(IDC_IPADDRESS1)->EnableWindow(0);
}

void CFilterDlg::OnBnClickedRadio6()
{
        pFilterSet->bAllPort=false;
        GetDlgItem(IDC_EDIT1)->EnableWindow(1);
}

void CFilterDlg::OnBnClickedRadio5()
{
        pFilterSet->bAllPort=true;
```

```
    GetDlgItem(IDC_EDIT1)->EnableWindow(0);
}

void CFilterDlg::OnBnClickedOk()
{
    // TODO: 在此添加控件通知处理程序代码
    //UpdateData(true);
    CString temp;

    if(((CButton *)GetDlgItem(IDC_RADIO1))->GetCheck())//1 选上
    {
        pFilterSet->bAllProtocol=true;
    }
    else
    {
        pFilterSet->bAllProtocol=false;
        if(((CButton *)GetDlgItem(IDC_CHECK1))->GetCheck())
            pFilterSet->bArp=true;
        else
            pFilterSet->bArp=false;

        if(((CButton *)GetDlgItem(IDC_CHECK2))->GetCheck())
            pFilterSet->bIcmp=true;
        else
            pFilterSet->bIcmp=false;

        if(((CButton *)GetDlgItem(IDC_CHECK3))->GetCheck())
            pFilterSet->bUdp=true;
        else
            pFilterSet->bUdp=false;

        if(((CButton *)GetDlgItem(IDC_CHECK4))->GetCheck())
            pFilterSet->bTcp=true;
        else
            pFilterSet->bTcp=false;

    }

    if(((CButton *)GetDlgItem(IDC_RADIO3))->GetCheck())//1 选上
    {
        pFilterSet->bAllIP=true;
    }
    else
    {
```

```
            pFilterSet->bAllIP=false;
            GetDlgItem(IDC_IPADDRESS1)->GetWindowText(temp);
            strcpy(pFilterSet->IP,temp);
        }
        if(((CButton *)GetDlgItem(IDC_RADIO5))->GetCheck())//1 选上
        {
            pFilterSet->bAllPort=true;
        }
        else
        {
            pFilterSet->bAllPort=false;
            GetDlgItem(IDC_EDIT1)->GetWindowText(temp);
            if(temp.GetLength()>5 || atol(temp)>65535)
            {
                MessageBox("端口有误");
                return;
            }
            else
            {
                strcpy(pFilterSet->Port,temp);
            }
        }


        OnOK();
}

void CFilterDlg::OnBnClickedCancel()
{
        // TODO: 在此添加控件通知处理程序代码
        OnCancel();
}
```

（三）SelectAdapterDlg.cpp

```
#include "stdafx.h"
#include "CapPack.h"
#include "SelectAdapterDlg.h"
#include ".\selectadapterdlg.h"


// CSelectAdapterDlg 对话框
IMPLEMENT_DYNAMIC(CSelectAdapterDlg, CDialog)
CSelectAdapterDlg::CSelectAdapterDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CSelectAdapterDlg::IDD, pParent)
```

```
{
}

CSelectAdapterDlg::~CSelectAdapterDlg()
{
}

void CSelectAdapterDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_COMBO1, m_Combo);
}

BEGIN_MESSAGE_MAP(CSelectAdapterDlg, CDialog)
	ON_BN_CLICKED(IDC_BUTTON1, OnBnClickedButton1)
	ON_CBN_SELENDCANCEL(IDC_COMBO1, OnCbnSelendcancelCombo1)
END_MESSAGE_MAP()


// CSelectAdapterDlg 消息处理程序
BOOL CSelectAdapterDlg::OnInitDialog()
{
	BOOL result;
	result=CDialog::OnInitDialog();

	if(result)
	{
		pcap_if_t *d;
		char errbuf[PCAP_ERRBUF_SIZE];
		// 获取设备列表
		if (pcap_findalldevs(&AllDevs, errbuf) == -1)
		{
			MessageBox("不能得到网络适配器列表");

		}
		// 数据列表
		for(d=AllDevs; d; d=d->next)
		{
			m_Combo.AddString(d->name);
		}
	}

	return result;
```

```
}
void CSelectAdapterDlg::OnBnClickedButton1()
{
	CString temp;

	int i=m_Combo.GetCurSel();
	if(i<0)return;

	m_Combo.GetLBText(i,temp);
	(*pName)=temp;


	pcap_freealldevs(AllDevs);

	CDialog::OnOK();
}

void CSelectAdapterDlg::OnCbnSelendcancelCombo1()
{
	pcap_if_t *d;
	CString temp;
	int i;
	i=m_Combo.GetCurSel();
	if(i<0)return;
	m_Combo.GetLBText(i,temp);
	(*pName)=temp;
	for(d=AllDevs; d; d=d->next)
	{
		if(temp==d->name)
			GetDlgItem(IDC_EDIT1)->SetWindowText(d->description);
	}
}
```

7、实验报告

要求撰写实验报告，在实验报告中要说明实现程序的关键步骤、程序运行的实验过程和实验结果（需要包含程序运行的截图）。

实验 4：文件分发 P2P 应用设计与实现

1、实验目的

掌握 P2P 网络应用程序结构及其开发技术。

2、实验环境

- Windows 或 Linux；
- TCP 协议；
- 任何你熟悉的编程语言。

3、实验内容

本实验要实现一个集中式索引的 P2P 服务器。集中式索引的 P2P 服务器是指由一台大型服务器（或服务器场）来提供索引服务。当用户启动 P2P 文件共享应用程序时，该应用程序将它的 IP 地址以及可供共享的文件名称通知索引服务器。索引服务器收集可共享的对象，建立集中式的动态数据库（对象名称到 IP 地址的映射）。这种索引方式的特点是：文件传输是分散的（P2P 的），但定位内容的过程是高度集中的（客户机/服务器）。这种模式的代表软件是 Napster、QQ。

Napster 网站是一个服务器集群，是世界上第一个大型的 p2p 应用网络，主要用于查找 mp3，它有一个服务器用于存储 mp3 文件的链接位置并提供检索，而真正的 mp3 文件则存放在千千万万的个人电脑上，搜索到的文件通过 p2p 方式直接在个人电脑间传播共享。每一个服务器保存一部分用户的共享文件索引信息，所有的服务器互连、整合起来对网站外面的 Napster 用户提供统一的访问接口，在每个用户看来他们访问的都是同一个服务器。

在 mp3 文件版权之争火热的年代，Napster 很快就成为众矢之的，被众多唱片公司诉讼侵犯版权而被迫关闭。中心服务器被迫关闭后对整个 P2P 网络的打击是毁灭性的，拥有 8000 万用户的整个 P2P 网络一夜间全部消失。

Napster 倒闭后，将自己的 P2P 协议开源了[1]。本次实验的协议规范就是参考了该开源协议。

4、实验方式

每位同学亲自动手实验，实验指导教师现场指导。

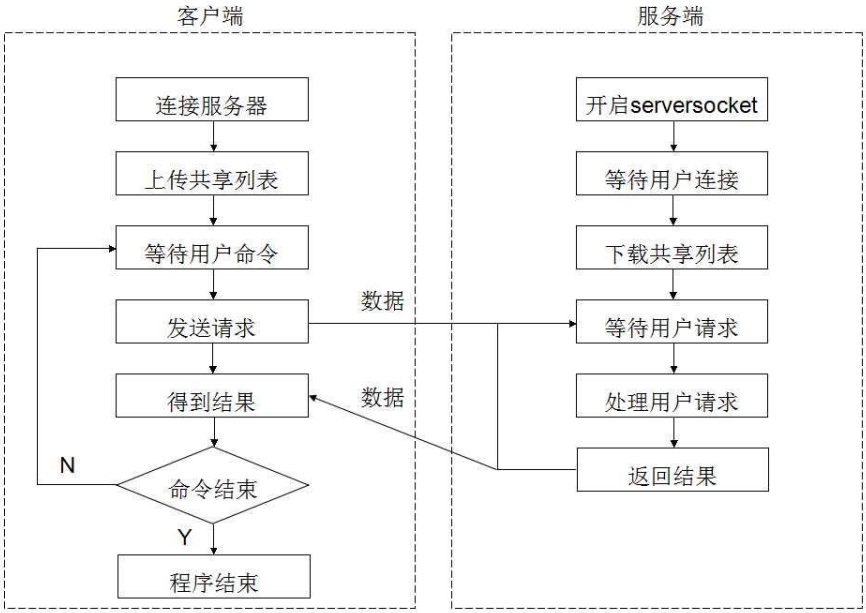
5、实验过程

（一）P2P 协议规范

本次实验分组完成，每组两个人，一个实现服务器端程序，一个实现客户端程序。服务端工程生成的可执行文件命名为 Napd，客户端工程生成的可执行文件命名为 Nap。由于两人同时开发，为了提高开发效率，下面给出客户端与服务器之间通信的协议规范，尤其是通信消息的格式标准，便于双方开发者同时开展开发工作。

[1]开源协议下载地址：http://opennap.sourceforge.net/napster.txt

通信的流程如下图所示：



（二）技术规范

1. TCP连接建立后的确认报文

与大多基于TCP的协议类似，Napster在生命周期中，P2P服务端会话有多个状态。一旦TCP连接被打开，服务器需要向客户端发送确认信息，此时服务器该会话进入“确认”状态。在确认状态中，客户端必须向服务器确认自己是它的客户。一旦确认成功，服务器就进入了“操作”状态。双方才可以真正进行资源操作。处于“操作”状态。

整个项目中，需要验证身份的一共两种情况：第一种是Nap客户端建立TCP连接到Napd服务器后，服务端需要确认身份；另一种是Nap客户端向另一个peer请求一个文件时，会向这个peer建立TCP连接，之后他们也需要确认身份。

由于本实验是对Napster的精简版，服务端并不维护用户信息，因此对于第一种情况，在Nap客户端与Napd服务器建立TCP连接后，只需要客户端发送的报文内容为“CONNECT”就可以认为是合法用户，服务端发送确认报文“ACCEPT”。 对第二种情况，在客户端Nap1与另一个客户端Nap2建立TCP连接后，Nap2发送的内容为“HELLO” 的报文，Nap1就可以确认身份，给出确认报文”ACCEPT”。

2. 服务端可以接收的消息格式

- 添加当前客户端本地参与共享的文件的信息
  - ✓ 接收的消息格式 ：ADD <文件名> <Hash 值> <文件大小>
  - ✓ 执行成功返回消息：OK
  - ✓ 执行失败返回消息：ERROR + 失败原因
- 删除当前客户端本地参与共享的文件的信息
  - ✓ 接收的消息格式 ：DELETE <文件名><HASH 值>
  - ✓ 执行成功返回消息：OK
  - ✓ 执行失败返回消息：ERROR + 失败原因
- 展示服务端可供下载的所有文件信息
  - ✓ 接收的消息格式 ：LIST

- ✓ 执行成功返回消息：OK
- ✓ 执行失败返回消息：ERROR + 失败原因

➢ 结束会话

- ✓ 接收的消息格式 ：QUIT

➢ 获取指定文件

- ✓ 接收的消息格式 ：REQUEST〈文件名〉
- ✓ 执行成功返回消息：该文件对应的 peer 地址+文件大小
- ✓ 执行失败返回消息：ERROR + 失败原因

3. 两个peer之间通信的消息格式

两个peer之间传输文件时，一个peer需要作为本地文件服务器，另一个peer作为客户端去请求文件，这两个功能全部集成在Nap客户端中。

因为两个peer之间传输消息全部由客户端开发的同学完成，不会在两个开发者之间产生歧义，所以本协议中不做规定。

（三）通信流程

程序使用TCP协议实现客户端到服务器的通信，在默认情况下使用7777作为通信端口。两个peer之间通信，收发消息使用7701端口，传输文件使用7702端口。

首先运行Napd服务端程序，此时服务端应该允许用户对服务器的以下参数进行配置：

➢ 端口号：默认值为 7777；

➢ 线程池可分配线程数 ：默认值为 64[1]；

➢ Socket 连接等待队列长度 ：默认值为 32[2]

服务端一旦启动，就会开始工作。Napster 的工作过程如下图 1 所示。

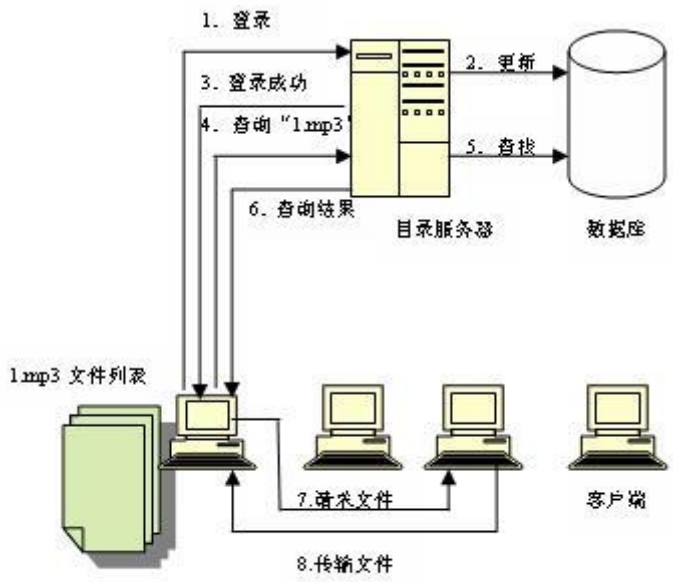


图1 Napster的工作过程

[1]可选参数。当使用到线程池技术时添加此参数，该参数实质上代表了最大可支持同时在线客户端数。下发提供的代码使用了线程池技术，并参考 Johan 的开源实现：https://github.com/Pithikos/C-Thread-Pool 。

[2]这个参数涉及到一些网络的细节。在进程正理一个一个连接请求的时候，可能还存在其它的连接请求。因为 TCP 连接是一个过程，所以可能存在一种半连接的状态，有时由于同时尝试连接的用户过多，使得服务器进程无法快速地完成连接请求。如果这个情况出现了，服务器进程希望内核如何处理呢？内核会在自己的进程空间里维护一个队列以跟踪这些完成的连接但服务器进程还没有接手处理或正在进行的连接，这样的一个队列内核不可能让其任意大，所以必须有一个大小的上限。

6、参考内容

注意：参考代码基于 Linux 完成，在 windows 下无法正常运行。

（一）服务端 代码 main.c

```
#include <arpa/inet.h>
#include <fcntl.h>
#include <netdb.h>
#include <netinet/in.h>
#include <pthread.h>
#include <signal.h>
#include <sqlite3.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#include <ctype.h>

#include "config.h"
#include "thpool.h"

extern sqlite3 *db;
typedef struct
{
    int fd;
    char ipaddr[128];
} p2p_t;

int loc_fd, inc_fd;
struct sockaddr_storage inc_addr;
socklen_t inc_len = sizeof(inc_addr);

threadpool thpool;
pthread_t net_thread;

int num_threads = NUM_THREADS;
int pidfile;
char *port = (char *)DEFAULT_PORT;
int queue_length = QUEUE_LENGTH;
```

```
char clientaddr[128] = { '\0' };
sqlite3 *db;
time_t start_time;
char *term;
static int c_count = 0;

void clean_string(char *);//字符串处理，去除诸如\b 之类的转义符
int client_count(int);//自加一，计算客户端数量
void console_help();//打印帮助信息
void *get_in_addr(struct sockaddr *);//获取 IP 地址
int recv_msg(int, char *);
int send_msg(int, char *);
int validate_int(char *);
void print_stats();
void stat_handler();
void shutdown_handler();
void *p2p(void *);
void *tcp_listen();

int main(int argc, char *argv[])
{
    //======系统环境设置======
    struct addrinfo hints, *result;
    int yes = 1;
    char command[512] = { '\0' };
    int i = 0;
    sqlite3_stmt *stmt;
    char query[256] = { '\0' };

    //调用 shutdown_handler 处理三种情况
    signal(SIGHUP, shutdown_handler);//关闭终端
    signal(SIGINT, shutdown_handler);//按下 CTRL+C
    signal(SIGTERM, shutdown_handler);//kill 命令

    // 注册自定义信号
    signal(SIGUSR1, stat_handler);
    signal(SIGUSR2, stat_handler);

    term = strdup(ttyname(1));

    fprintf(stdout, "%s: %s 正在初始化%s... \n", SERVER_NAME, INFO_MSG,
SERVER_NAME);

    start_time = time(NULL);//开始计时
```

```
    //======处理执行参数======

    for(i = 1; i < argc; i++)
    {
        if(strcmp("-h", argv[i]) == 0 || strcmp("--help", argv[i]) == 0)
        {
            fprintf(stdout, "usage: %s [-h | --help] [-p | --port port] [-q | --queue queue_length] [-t | --threads thread_count]\n\n", SERVER_NAME);
            fprintf(stdout, "%s 参数说明:\n", SERVER_NAME);
            fprintf(stdout, "\t-h | --help:                    help - 展示帮助信息\n");
            fprintf(stdout, "\t-p | --port:                    port - 为服务器指定一个端口号(默认: %s)\n", DEFAULT_PORT);
            fprintf(stdout, "\t-q | --queue:     queue_length - 为服务器指定连接队列的长度(默认: %d)\n", QUEUE_LENGTH);
            fprintf(stdout, "\t-t | --threads: thread_count - 为服务器指定连接池的长度(也就是最大支持的客户端数量) (默认: %d)\n", NUM_THREADS);
            fprintf(stdout, "\n");
            //退出
            exit(0);
        }
        else if(strcmp("-p", argv[i]) == 0 || strcmp("--port", argv[i]) == 0)
        {
            if(argv[i+1] != NULL)
            {
                if(validate_int(argv[i+1]))
                {
                    if(atoi(argv[i+1]) >= 0 && atoi(argv[i+1]) <= MAX_PORT)
                    {
                        port = argv[i+1];
                        i++;
                    }
                    else
                        fprintf(stderr, "%s: %s 端口号不在范围内(0-%d), 恢复默认端口号 %s\n", SERVER_NAME, ERROR_MSG, MAX_PORT, DEFAULT_PORT);
                }
                else
                {
                    fprintf(stderr, "%s: %s 指定的端口号非法, 恢复默认端口号 %s\n", SERVER_NAME, ERROR_MSG, DEFAULT_PORT);
                }
            }
            else
            {
```

```
                fprintf(stderr, "%s: %s 没有在 port 参数后找到端口值，恢复默认端口号 %s\n", SERVER_NAME, ERROR_MSG, DEFAULT_PORT);
            }
        }
        else if(strcmp("-q", argv[i]) == 0 || strcmp("--queue", argv[i]) == 0)
        {
            if(argv[i+1] != NULL)
            {
                if(validate_int(argv[i+1]))
                {
                    if(atoi(argv[i+1]) >= 1)
                    {
                        queue_length = atoi(argv[i+1]);
                        i++;
                    }
                    else
                        fprintf(stderr, "%s: %s 队列不能为非正数，恢复默认队列长度 %d\n", SERVER_NAME, ERROR_MSG, QUEUE_LENGTH);
                }
                else
                {
                    fprintf(stderr, "%s: %s 队列长度参数非法，恢复默认队列长度 %d\n", SERVER_NAME, ERROR_MSG, QUEUE_LENGTH);
                }
            }
            else
            {
                // Print error and use default queue length if no length was specified after the flag
                fprintf(stderr, "%s: %s 没有在 queue 参数后找到队列长度，恢复默认队列长度 %d\n", SERVER_NAME, ERROR_MSG, QUEUE_LENGTH);
            }
        }
        else if(strcmp("-t", argv[i]) == 0 || strcmp("--threads", argv[i]) == 0)
        {
            if(argv[i+1] != NULL)
            {
                if(validate_int(argv[i+1]))
                {
                    if(atoi(argv[i+1]) >= 1)
                    {
                        num_threads = atoi(argv[i+1]);
                        i++;
                    }
```

```
                    else
                        fprintf(stderr, "%s: %s 线程数不能为非正数，恢复默认 %d 线程数\n", SERVER_NAME, ERROR_MSG, NUM_THREADS);
                }
                else
                {
                    fprintf(stderr, "%s: %s 线程数参数非法，恢复默认 %d 线程数\n", SERVER_NAME, ERROR_MSG, NUM_THREADS);
                }
            }
            else
            {
                fprintf(stderr, "%s: %s 没有在 thread 参数后找到线程数，恢复默认 %d 线程数\n", SERVER_NAME, ERROR_MSG, NUM_THREADS);
            }
        }
        else
        {
            fprintf(stderr, "%s: %s 检测到未知参数'%s'，输入 '%s -h' 查看 usage\n", SERVER_NAME, ERROR_MSG, argv[i], SERVER_NAME);
            exit(-1);
        }
    }

    //======准备数据库======

    sqlite3_open(DB_FILE, &db);
    if(db == NULL)
    {
        fprintf(stderr, "%s: %s sqlite: 不能打开 SQLite %s\n", SERVER_NAME, ERROR_MSG, DB_FILE);
        exit(-1);
    }
    sprintf(query, "DELETE FROM files");
    sqlite3_prepare_v2(db, query, strlen(query) + 1, &stmt, NULL);
    if(sqlite3_step(stmt) != SQLITE_DONE)
    {
        fprintf(stderr, "%s: %s sqlite: 操作失败！ \n", SERVER_NAME, ERROR_MSG);
        exit(-1);
    }
    sqlite3_finalize(stmt);

    //======初始化 TCP 连接======
```

```
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    if((getaddrinfo(NULL, port, &hints, &result)) != 0)
    {
        fprintf(stderr, "%s: %s 调用 getaddrinfo() 失败，程序中断 \n", SERVER_NAME, ERROR_MSG);
        exit(-1);
    }
    if((loc_fd = socket(result->ai_family, result->ai_socktype, result->ai_protocol)) == -1)
    {
        fprintf(stderr, "%s: %s socket 创建失败，程序中断 \n", SERVER_NAME, ERROR_MSG);
        exit(-1);
    }
    if(setsockopt(loc_fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
    {
        fprintf(stderr, "%s: %s 不能允许 socket 重新绑定(SO_REUSEADDR)，程序中断 \n", SERVER_NAME, ERROR_MSG);
        exit(-1);
    }

    //绑定 socket
    if((bind(loc_fd, result->ai_addr, result->ai_addrlen)) == -1)
    {
        if(atoi(port) < PRIVILEGED_PORT)
            fprintf(stderr, "%s: %s 绑定 socket 失败，权限不足 \n", SERVER_NAME, ERROR_MSG);
        else
            fprintf(stderr, "%s: %s 绑定 socket 失败，请检查当前端口是否被占用 \n", SERVER_NAME, ERROR_MSG);

        // Exit on failure
        exit(-1);
    }
    freeaddrinfo(result);
    listen(loc_fd, queue_length);//设置 socket 为 listen 模式

    //初始化一个线程池
    thpool = thpool_init(num_threads);
    pthread_create(&net_thread, NULL, &tcp_listen, NULL);
```

```
    fprintf(stdout, "%s: %s 服务器初始化成功 配置信息如下： [PID: %d] [端口号: %s] [队列长度: %d] [线程数: %d]\n", SERVER_NAME, OK_MSG, getpid(), port, queue_length, num_threads);
    fprintf(stdout, "%s: %s 你可以通过输入'help' 获取帮助信息 \n", SERVER_NAME, INFO_MSG);
    fprintf(stdout, "%s: %s 你可以通过输入'stop' 或者使用快捷键 Ctrl+C 来停止运行 \n", SERVER_NAME, INFO_MSG);

    //======用户输入处理======

    while(1)
    {
        fgets(command, sizeof(command), stdin);
        clean_string((char *)&command);
        if(strcmp(command, "clear") == 0)
            system("clear");
        else if(strcmp(command, "help") == 0)
            console_help();
        else if(strcmp(command, "stat") == 0)
            print_stats();
        else if(strcmp(command, "stop") == 0)
            break;
        else
            fprintf(stderr, "%s: %s 命令'%s'未知，输入'help'获取帮助 \n", SERVER_NAME, ERROR_MSG, command);
    }
    kill(getpid(), SIGINT);
}

void clean_string(char *str)
{
    int i = 0;
    int index = 0;
    char buffer[1024];
    for(i = 0; i < strlen(str); i++)
    {
        if(str[i] != '\b' && str[i] != '\n' && str[i] != '\r')
            buffer[index++] = str[i];
    }
    memset(str, 0, sizeof(str));
    buffer[index] = '\0';
    strcpy(str, buffer);
}
```

```
int client_count(int change)
{
	c_count += change;
	return c_count;
}

void console_help()
{
	fprintf(stdout, "%s  帮助:\n", SERVER_NAME);
	fprintf(stdout, "\tclear -  清除终端信息\n");
	fprintf(stdout, "\t help -  获取帮助信息\n");
	fprintf(stdout, "\t stat -  获取当前状态\n");
	fprintf(stdout, "\t stop -  停止服务器\n");
}

void *get_in_addr(struct sockaddr *sa)
{
		if (sa->sa_family == AF_INET)
			return &(((struct sockaddr_in*)sa)->sin_addr);
		else
			return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int recv_msg(int fd, char *message)
{
	int b_received = 0;
	int b_total = 0;
	char buffer[1024];
	memset(buffer, '\0', sizeof(buffer));

	b_received = recv(fd, buffer, sizeof(buffer), 0);
	b_total += b_received;
	strcpy(message, buffer);
	return b_total;
}

int send_msg(int fd, char *message)
{
	return send(fd, message, strlen(message), 0);
}

int validate_int(char *string)
{
```

```
    int isInt = 1;
    int j = 0;
        for(j = 0; j < strlen(string); j++)
        {
            if(isInt == 1)
            {
                if(!isdigit(string[j]))
                        isInt = 0;
                }
        }
    return isInt;
}

void print_stats()
{
    //打印运行时间
    int hours, minutes, seconds;
    char runtime[32] = { '\0' };
    char tpusage[32] = { '\0' };
    seconds = (int)difftime(time(NULL), start_time);
    minutes = seconds / 60;
    hours = minutes / 60;
    minutes = minutes % 60;
    seconds = seconds % 60;
    sprintf(runtime, "%02d:%02d:%02d", hours, minutes, seconds);

    //打印连接池状态

    //连接池容量绰绰有余时
    if(client_count(0) < (num_threads * TP_UTIL))
    {
        fprintf(stdout, "%s: %s ", SERVER_NAME, OK_MSG);
        sprintf(tpusage, "[在线用户数: %d/%d]", client_count(0), num_threads);
    }
    // 连接池快满了或者已经饱和时
    else if(((double)client_count(0) >= ((double)num_threads * TP_UTIL)) &&
client_count(0) <= num_threads)
    {
        //转为警告
        fprintf(stdout, "%s: %s ", SERVER_NAME, WARN_MSG);
        sprintf(tpusage, "\033[1;33m[在线用户数: %d/%d]\033[0m", client_count(0),
num_threads);
    }
    // 连接池已经超负荷时
```

```
    else
    {
        // 转为错误
        fprintf(stdout, "%s: %s ", SERVER_NAME, ERROR_MSG);
        sprintf(tpusage, "\033[1;31m[在线用户数: %d/%d]\033[0m", client_count(0), num_threads);
    }
    fprintf(stdout, "服务器运行中： [PID: %d] [运行时间: %s] [运行端口: %s] [queue: %d] %s\n", getpid(), runtime, port, queue_length, tpusage);
}


// 当产生 SIGUSR1/SIGUSR2 信号时，向客户端报告服务器状态
void stat_handler()
{
    freopen(term, "w", stdout);

    // 打印服务器状态
    print_stats();

    // Return stdout to /dev/null
    freopen("/dev/null", "w", stdout);
}

void shutdown_handler()
{
    // 关闭 net_thread，停止接收新的请求
    pthread_cancel(net_thread);
    fprintf(stdout, "\n");

    // 关闭 SQLite 数据库
    if(sqlite3_close(db) != SQLITE_OK)
    {
        // 失败时
        fprintf(stderr, "%s: %s sqlite: 未能关闭 SQLite 数据库.\n", SERVER_NAME, ERROR_MSG);
        exit(-1);
    }

    // 尝试从容关闭 socket
    if(shutdown(loc_fd, 2) == -1)
    {
        // 失败时
        fprintf(stderr, "%s: %s 未能成功 shutdown 本机的 socket.\n", SERVER_NAME,
```

```
ERROR_MSG);
            exit(-1);
        }

        // 尝试暴力关闭 socket
        if(close(loc_fd) == -1)
        {
            // 失败时
            fprintf(stderr, "%s: %s 未能成功 close 本机的 socket.\n", SERVER_NAME,
ERROR_MSG);
            exit(-1);
        }

        // 关闭所有创建的连接池
        thpool_destroy(thpool);

        fprintf(stdout, "%s: %s 成功剔除 %d 台客户端设备，服务器中断。\n",
SERVER_NAME, OK_MSG, client_count(0));

        exit(0);
}

void *p2p(void *args)
{
        char in[512],out[512] = { '\0' };
        p2p_t params = *((p2p_t *)(args));
        char *filename, *filehash, *filesize;
        long int f_size = 0;
        char peeraddr[128] = { '\0' };
        strcpy(peeraddr, params.ipaddr);
        int user_fd = params.fd;
        char query[256];
        int status;
        int flag=0;
        sqlite3_stmt *stmt;

        sprintf(out, "%s: %s \n", SERVER_NAME, USER_MSG);
        send_msg(user_fd, out);

        // 等待客户端发来消息
        while((strcmp(in, "CONNECT")) != 0 && (strcmp(in, "QUIT") != 0))
        {
            //获取消息
            recv_msg(user_fd, (char *)&in);
```

```
        clean_string((char *)&in);

        //如果发来的是握手消息 CONNECT，返回确认信息 ACCEPT
        if(strcmp(in, "CONNECT") == 0)
        {
            fprintf(stdout, "%s: %s 检测到 %s 向服务器发送了一个握手消息，返回确认消息 [句柄: %d]\n", SERVER_NAME, OK_MSG,peeraddr, user_fd);

            sprintf(out, "ACCEPT\n");
            send_msg(user_fd, out);
        }
    }

    //服务端已经发送确认信息，等待客户端发来进一步的消息
    while(strcmp(in, "QUIT") != 0)
    {
        memset(in, 0, sizeof(in));
        memset(out, 0, sizeof(out));
        memset(query, 0, sizeof(query));

        //获取消息
        recv_msg(user_fd, (char *)&in);
        clean_string((char *)&in);

        // 格式: ADD <文件名> <Hash 值> <文件大小>
        if(strncmp(in, "ADD", 3) == 0)
        {
            strtok(in, " ");
            filename = strtok(NULL, " ");
            flag=0;

            if(filename != NULL)
            {
                filehash = strtok(NULL, " ");
                if(filehash != NULL)
                {
                    filesize = strtok(NULL, " ");
                    if((filesize != NULL) && (validate_int(filesize) == 1))
                    {
                        f_size = atoi(filesize);
                        sprintf(query, "INSERT INTO files VALUES('%s', '%s', '%ld', '%s')", filename, filehash, f_size, peeraddr);
                        sqlite3_prepare_v2(db, query, strlen(query) + 1, &stmt, NULL);
```

```
                        if((status = sqlite3_step(stmt)) != SQLITE_DONE)
                        {
                            if(status == SQLITE_CONSTRAINT)
                            {
                                fprintf(stderr, "%s: %s sqlite: 添加文件失败，服务器数据库中已经存在当前文件\n", SERVER_NAME, ERROR_MSG);
                                sprintf(out, "ERROR 添加文件失败，服务器数据库中已经存在当前文件\n");
                                send_msg(user_fd, out);
                            }
                            else
                            {
                                fprintf(stderr, "%s: %s sqlite: 添加文件失败 \n", SERVER_NAME, ERROR_MSG);
                                sprintf(out, "ERROR 添加文件信息到数据库失败，原因未知\n");
                                send_msg(user_fd, out);
                            }
                        }
                        sqlite3_finalize(stmt);

                        if(status == SQLITE_DONE)
                        {
                            fprintf(stdout, "%s: %s 客户端%s 向服务器添加了文件 %20s [hash 值: %20s] [大小: %10ld]\n", SERVER_NAME, INFO_MSG, peeraddr, filename, filehash, f_size);

                            //返回 OK
                            sprintf(out, "OK\n");
                            send_msg(user_fd, out);
                        }
                    }
                    else
                        flag=1;
                }
                else
                    flag=1;
            }
            else
                flag=1;

            //传入参数的格式错误
            if(flag)
            {
```

```
                    fprintf(stderr, "%s: %s 添加文件失败，传入参数的格式错误 \n",
SERVER_NAME, ERROR_MSG);
                    sprintf(out, "ERROR 添加文件失败，传入参数的格式错误\n");
                    send_msg(user_fd, out);
                }

            }

            // 格式: DELETE [文件名] [HASH 值]
            else if(strncmp(in, "DELETE", 6) == 0)
            {
                strtok(in, " ");
                filename = strtok(NULL, " ");
                flag=0;

                if(filename != NULL)
                {
                    filehash = strtok(NULL, " ");
                    if(filehash != NULL)
                    {
                        sprintf(query, "DELETE FROM files WHERE file='%s' AND
hash='%s' AND peer='%s'", filename, filehash, peeraddr);
                        sqlite3_prepare_v2(db, query, strlen(query) + 1, &stmt, NULL);
                        if(sqlite3_step(stmt) != SQLITE_DONE)
                        {
                            fprintf(stderr, "%s: %s sqlite: 删除文件失败 \n",
SERVER_NAME, ERROR_MSG);
                            sprintf(out, "ERROR 从数据库中删除文件失败，原因未知
\n");
                            send_msg(user_fd, out);
                        }
                        sqlite3_finalize(stmt);

                        fprintf(stdout, "%s: %s 客户端%s 向服务器删除了文件
'%s'('%s') \n", SERVER_NAME, OK_MSG, peeraddr, filename, filehash);
                        sprintf(out, "OK\n");
                        send_msg(user_fd, out);
                    }
                    else
                        flag=1;
                }
                else
                    flag=1;
                //传入参数的格式错误
```

```
                if(flag)
                {
                    fprintf(stderr, "%s: %s 删除文件失败，传入参数的格式错误 \n",
SERVER_NAME, ERROR_MSG);
                    sprintf(out, "ERROR 删除文件失败，传入参数的格式错误\n");
                    send_msg(user_fd, out);
                }
            }

            // LIST
            else if(strcmp(in, "LIST") == 0)
            {
                sprintf(query, "SELECT DISTINCT file,size,peer FROM files ORDER BY
file ASC");
                sqlite3_prepare_v2(db, query, strlen(query) + 1, &stmt, NULL);
                while((status = sqlite3_step(stmt)) != SQLITE_DONE)
                {
                    if(status == SQLITE_ERROR)
                    {
                        fprintf(stderr, "%s: %s sqlite: 未能获得所有记录，数据库错误
\n", SERVER_NAME, ERROR_MSG);
                        sprintf(out, "ERROR 未能获得所有记录，服务端数据库错误
\n");
                        send_msg(user_fd, out);
                    }
                    else if(strcmp(peeraddr,(char *) sqlite3_column_text(stmt, 2)))
                    {
                        sprintf(out, "%s %d\n", sqlite3_column_text(stmt, 0),
sqlite3_column_int(stmt, 1));
                        send_msg(user_fd, out);
                    }
                }
            sqlite3_finalize(stmt);
            sprintf(out, "OK\n");
            send_msg(user_fd, out);
            }

            // QUIT
            else if(strcmp(in, "QUIT") == 0)
            {
                continue;
            }

            // syntax: REQUEST [文件名]
```

```
        else if(strncmp(in, "REQUEST", 7) == 0)
        {
            strtok(in, " ");
            filename = strtok(NULL, " ");
            if(filename != NULL)
            {
                sprintf(query, "SELECT peer,size FROM files WHERE file='%s' ORDER BY peer ASC", filename);
                sqlite3_prepare_v2(db, query, strlen(query) + 1, &stmt, NULL);
                while((status = sqlite3_step(stmt)) != SQLITE_DONE)
                {
                    if(status == SQLITE_ERROR)
                    {
                        fprintf(stderr, "%s: %s sqlite: 未能成功获取文件信息，数据库错误 '%s'\n", SERVER_NAME, ERROR_MSG, filename);
                        sprintf(out, "ERROR 未能成功获取文件信息，数据库错误\n");
                        send_msg(user_fd, out);
                    }
                    else
                    {
                        sprintf(out, "%s %ld\n", sqlite3_column_text(stmt, 0), (long int)sqlite3_column_int(stmt, 1));
                        send_msg(user_fd, out);
                    }
                }
                sqlite3_finalize(stmt);

                sprintf(out, "OK\n");
                send_msg(user_fd, out);
            }
            else
            {
                sprintf(out, "ERROR 没能成功获得请求的文件名 \n");
                send_msg(user_fd, out);
            }
        }
        else
        {
            sprintf(out, "ERROR 参数错误\n");
            send_msg(user_fd, out);
        }
    }
```

```
    memset(out, 0, sizeof(out));

    sprintf(out, "GOODBYE\n");
    send_msg(user_fd, out);

    fprintf(stdout, "%s: %s 客户端 %s 已经从服务器注销登录 [在线用户数 : %d/%d]\n", SERVER_NAME, OK_MSG, peeraddr, client_count(-1), NUM_THREADS);

    sprintf(query, "DELETE FROM files WHERE peer='%s'", peeraddr);
    sqlite3_prepare_v2(db, query, strlen(query) + 1, &stmt, NULL);
    if(sqlite3_step(stmt) != SQLITE_DONE)
    {
        fprintf(stderr, "%s: %s 客户端 %s 剔除失败 [句柄: %d]\n", SERVER_NAME, ERROR_MSG, peeraddr, user_fd);
        return (void *)-1;
    }
    sqlite3_finalize(stmt);

    if(close(user_fd) == -1)
    {
        fprintf(stderr, "%s: %s 关闭套接字失败 [句柄: %d]\n", SERVER_NAME, ERROR_MSG, user_fd);
        return (void *)-1;
    }

    return (void *)0;
}

//建立 TCP 连接
void *tcp_listen()
{
    p2p_t params;
    char out[512] = { '\0' };

    while(1)
    {
        if((inc_fd = accept(loc_fd, (struct sockaddr *)&inc_addr, &inc_len)) == -1)
        {
            fprintf(stderr, "%s: %s 未能成功接收连接 \n", SERVER_NAME, ERROR_MSG);
            return (void *)-1;
        }
        else
```

```
        {
            inet_ntop(inc_addr.ss_family, get_in_addr((struct sockaddr *)&inc_addr), clientaddr, sizeof(clientaddr));

            fprintf(stdout, "%s: %s 监测到 %s 正在尝试连接到服务器 [socket 编号: %d] [在线用户数: %d/%d]\n", SERVER_NAME, INFO_MSG, clientaddr, inc_fd, client_count(1), num_threads);

            if(((double)client_count(0) >= ((double)num_threads * TP_UTIL)) && (client_count(0) <= num_threads))
            {
                if(client_count(0) == num_threads)
                    fprintf(stdout, "%s: %s 连接池资源耗尽 [在线用户数: %d/%d]\n", SERVER_NAME, WARN_MSG, client_count(0), num_threads);
                else
                    fprintf(stdout, "%s: %s 连接池资源即将耗尽 [在线用户数: %d/%d]\n", SERVER_NAME, WARN_MSG, client_count(0), num_threads);
            }
            else if((client_count(0)) > num_threads)
            {
                fprintf(stderr, "%s: %s 连接池资源耗尽，仍然有新用户尝试连接 [在线用户数: %d/%d]\n", SERVER_NAME, ERROR_MSG, client_count(0), num_threads);
                sprintf(out, "%s: %s 服务器负载过大，请稍后再试 \n", SERVER_NAME, USER_MSG);
                send_msg(inc_fd, out);
            }
            params.fd = inc_fd;
            strcpy(params.ipaddr, clientaddr);
            thpool_add_work(thpool, &p2p, (void*)&params);//添加到线程池
        }
    }
}
```

（二）服务端 代码 采用开源的线程池--thpool.c

```
/* ********************************
 * Author:          Johan Hanssen Seferidis
 * License:         MIT
 * Description:   Library providing a threading pool where you can add
 *                  work. For usage, check the thpool.h file or README.md
 *
 *//** @file thpool.h *//*
 *
 ********************************/
```

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <errno.h>
#include <time.h>

#include "thpool.h"

#ifdef THPOOL_DEBUG
#define THPOOL_DEBUG 1
#else
#define THPOOL_DEBUG 0
#endif

#define MAX_NANOSEC 999999999
#define CEIL(X) ((X-(int)(X)) > 0 ? (int)(X+1) : (int)(X))

static volatile int threads_keepalive;
static volatile int threads_on_hold;




/*                      ============================                      STRUCTURES
============================== */


/* Binary semaphore */
typedef struct bsem {
        pthread_mutex_t mutex;
        pthread_cond_t      cond;
        int v;
} bsem;


/* Job */
typedef struct job{
        struct job*    prev;                          /* pointer to previous job      */
        void*    (*function)(void* arg);            /* function pointer             */
```

```
    void*    arg;                                  /* function's argument           */
} job;


/* Job queue */
typedef struct jobqueue{
    pthread_mutex_t rwmutex;                  /* used for queue r/w access */
    job    *front;                            /* pointer to front of queue */
    job    *rear;                             /* pointer to rear    of queue */
    bsem *has_jobs;                           /* flag as binary semaphore    */
    int     len;                              /* number of jobs in queue      */
} jobqueue;


/* Thread */
typedef struct thread{
    int            id;                        /* friendly id                  */
    pthread_t pthread;                       /* pointer to actual thread    */
    struct thpool_* thpool_p;                /* access to thpool             */
} thread;


/* Threadpool */
typedef struct thpool_{
    thread**      threads;                    /* pointer to threads           */
    volatile int num_threads_alive;          /* threads currently alive      */
    volatile int num_threads_working;        /* threads currently working */
    pthread_mutex_t    thcount_lock;         /* used for thread count etc */
    jobqueue*    jobqueue_p;                 /* pointer to the job queue    */
} thpool_;




/*                    ============================                    PROTOTYPES
============================== */


static void    thread_init(thpool_* thpool_p, struct thread** thread_p, int id);
static void* thread_do(struct thread* thread_p);
static void    thread_hold();
static void    thread_destroy(struct thread* thread_p);
```

```
static int      jobqueue_init(thpool_* thpool_p);
static void     jobqueue_clear(thpool_* thpool_p);
static void     jobqueue_push(thpool_* thpool_p, struct job* newjob_p);
static struct job* jobqueue_pull(thpool_* thpool_p);
static void     jobqueue_destroy(thpool_* thpool_p);

static void     bsem_init(struct bsem *bsem_p, int value);
static void     bsem_reset(struct bsem *bsem_p);
static void     bsem_post(struct bsem *bsem_p);
static void     bsem_post_all(struct bsem *bsem_p);
static void     bsem_wait(struct bsem *bsem_p);




/*                         ==========================                         THREADPOOL
============================== */


/* Initialise thread pool */
struct thpool_* thpool_init(int num_threads){

    threads_on_hold      = 0;
    threads_keepalive = 1;

    if ( num_threads < 0){
        num_threads = 0;
    }

    /* Make new thread pool */
    thpool_* thpool_p;
    thpool_p = (struct thpool_*)malloc(sizeof(struct thpool_));
    if (thpool_p==NULL){
        fprintf(stderr, "thpool_init(): Could not allocate memory for thread pool\n");
        exit(1);
    }
    pthread_mutex_init(&(thpool_p->thcount_lock), NULL);
    thpool_p->num_threads_alive      = 0;
    thpool_p->num_threads_working = 0;

    /* Initialise the job queue */
    if (jobqueue_init(thpool_p)==-1){
        fprintf(stderr, "thpool_init(): Could not allocate memory for job queue\n");
```

```
            exit(1);
        }

        /* Make threads in pool */
        thpool_p->threads = (struct thread**)malloc(num_threads * sizeof(struct thread));
        if (thpool_p->threads==NULL){
            fprintf(stderr, "thpool_init(): Could not allocate memory for threads\n");
            exit(1);
        }

        /* Thread init */
        int n;
        for (n=0; n<num_threads; n++){
            thread_init(thpool_p, &thpool_p->threads[n], n);
            if (THPOOL_DEBUG)
                printf("THPOOL_DEBUG: Created thread %d in pool \n", n);
        }

        /* Wait for threads to initialize */
        while (thpool_p->num_threads_alive != num_threads) {}

        return thpool_p;
}


/* Add work to the thread pool */
int thpool_add_work(thpool_* thpool_p, void *(*function_p)(void*), void* arg_p){
        job* newjob;

        newjob=(struct job*)malloc(sizeof(struct job));
        if (newjob==NULL){
            fprintf(stderr, "thpool_add_work(): Could not allocate memory for new job\n");
            return -1;
        }

        /* add function and argument */
        newjob->function=function_p;
        newjob->arg=arg_p;

        /* add job to queue */
        pthread_mutex_lock(&thpool_p->jobqueue_p->rwmutex);
        jobqueue_push(thpool_p, newjob);
        pthread_mutex_unlock(&thpool_p->jobqueue_p->rwmutex);
```

```
    return 0;
}


/* Wait until all jobs have finished */
void thpool_wait(thpool_* thpool_p){

    /* Continuous polling */
    double timeout = 1.0;
    time_t start, end;
    double tpassed = 0.0;
    time (&start);
    while (tpassed < timeout &&
            (thpool_p->jobqueue_p->len || thpool_p->num_threads_working))
    {
        time (&end);
        tpassed = difftime(end,start);
    }

    /* Exponential polling */
    long init_nano =    1; /* MUST be above 0 */
    long new_nano;
    double multiplier =    1.01;
    int    max_secs      = 20;

    struct timespec polling_interval;
    polling_interval.tv_sec    = 0;
    polling_interval.tv_nsec = init_nano;

    while (thpool_p->jobqueue_p->len || thpool_p->num_threads_working)
    {
        nanosleep(&polling_interval, NULL);
        if ( polling_interval.tv_sec < max_secs ){
            new_nano = CEIL(polling_interval.tv_nsec * multiplier);
            polling_interval.tv_nsec = new_nano % MAX_NANOSEC;
            if ( new_nano > MAX_NANOSEC ) {
                polling_interval.tv_sec ++;
            }
        }
        else break;
    }

    /* Fall back to max polling */
    while (thpool_p->jobqueue_p->len || thpool_p->num_threads_working){
```

```
            sleep(max_secs);
        }
}


/* Destroy the threadpool */
void thpool_destroy(thpool_* thpool_p){

        volatile int threads_total = thpool_p->num_threads_alive;

        /* End each thread 's infinite loop */
        threads_keepalive = 0;

        /* Give one second to kill idle threads */
        double TIMEOUT = 1.0;
        time_t start, end;
        double tpassed = 0.0;
        time (&start);
        while (tpassed < TIMEOUT && thpool_p->num_threads_alive){
                bsem_post_all(thpool_p->jobqueue_p->has_jobs);
                time (&end);
                tpassed = difftime(end,start);
        }

        /* Poll remaining threads */
        while (thpool_p->num_threads_alive){
                bsem_post_all(thpool_p->jobqueue_p->has_jobs);
                sleep(1);
        }

        /* Job queue cleanup */
        jobqueue_destroy(thpool_p);
        free(thpool_p->jobqueue_p);

        /* Deallocs */
        int n;
        for (n=0; n < threads_total; n++){
                thread_destroy(thpool_p->threads[n]);
        }
        free(thpool_p->threads);
        free(thpool_p);
}
```

```
/* Pause all threads in threadpool */
void thpool_pause(thpool_* thpool_p) {
	int n;
	for (n=0; n < thpool_p->num_threads_alive; n++){
		pthread_kill(thpool_p->threads[n]->pthread, SIGUSR1);
	}
}


/* Resume all threads in threadpool */
void thpool_resume(thpool_* thpool_p) {
	threads_on_hold = 0;
}




/*                          ============================                          THREAD
================================ */


/* Initialize a thread in the thread pool
 *
 * @param thread		address to the pointer of the thread to be created
 * @param id			id to be given to the thread
 *
 */
static void thread_init (thpool_* thpool_p, struct thread** thread_p, int id){

	*thread_p = (struct thread*)malloc(sizeof(struct thread));
	if (thread_p == NULL){
		fprintf(stderr, "thpool_init(): Could not allocate memory for thread\n");
		exit(1);
	}

	(*thread_p)->thpool_p = thpool_p;
	(*thread_p)->id		= id;

	pthread_create(&(*thread_p)->pthread, NULL, (void * (*)(void *))thread_do,
(*thread_p));
	pthread_detach((*thread_p)->pthread);

}
```

```
/* Sets the calling thread on hold */
static void thread_hold () {
	threads_on_hold = 1;
	while (threads_on_hold){
		sleep(1);
	}
}


/* What each thread is doing
*
* In principle this is an endless loop. The only time this loop gets interuppted is once
* thpool_destroy() is invoked or the program exits.
*
* @param	thread		thread that will run this function
* @return nothing
*/
static void* thread_do(struct thread* thread_p){

	/* Assure all threads have been created before starting serving */
	thpool_* thpool_p = thread_p->thpool_p;

	/* Register signal handler */
	struct sigaction act;
	act.sa_handler = thread_hold;
	if (sigaction(SIGUSR1, &act, NULL) == -1) {
		fprintf(stderr, "thread_do(): cannot handle SIGUSR1");
	}

	/* Mark thread as alive (initialized) */
	pthread_mutex_lock(&thpool_p->thcount_lock);
	thpool_p->num_threads_alive += 1;
	pthread_mutex_unlock(&thpool_p->thcount_lock);

	while(threads_keepalive){

		bsem_wait(thpool_p->jobqueue_p->has_jobs);

		if (threads_keepalive){

			pthread_mutex_lock(&thpool_p->thcount_lock);
			thpool_p->num_threads_working++;
```

```
                pthread_mutex_unlock(&thpool_p->thcount_lock);

                /* Read job from queue and execute it */
                void*(*func_buff)(void* arg);
                void*    arg_buff;
                job* job_p;
                pthread_mutex_lock(&thpool_p->jobqueue_p->rwmutex);
                job_p = jobqueue_pull(thpool_p);
                pthread_mutex_unlock(&thpool_p->jobqueue_p->rwmutex);
                if (job_p) {
                        func_buff = job_p->function;
                        arg_buff    = job_p->arg;
                        func_buff(arg_buff);
                        free(job_p);
                }

                pthread_mutex_lock(&thpool_p->thcount_lock);
                thpool_p->num_threads_working--;
                pthread_mutex_unlock(&thpool_p->thcount_lock);

            }
        }
        pthread_mutex_lock(&thpool_p->thcount_lock);
        thpool_p->num_threads_alive --;
        pthread_mutex_unlock(&thpool_p->thcount_lock);

        return NULL;
}


/* Frees a thread    */
static void thread_destroy (thread* thread_p){
        free(thread_p);
}




/*              ============================          JOB          QUEUE
============================ */


/* Initialize queue */
```

```
static int jobqueue_init(thpool_* thpool_p){

	thpool_p->jobqueue_p = (struct jobqueue*)malloc(sizeof(struct jobqueue));
	pthread_mutex_init(&(thpool_p->jobqueue_p->rwmutex), NULL);
	if (thpool_p->jobqueue_p == NULL){
		return -1;
	}

	thpool_p->jobqueue_p->has_jobs = (struct bsem*)malloc(sizeof(struct bsem));
	if (thpool_p->jobqueue_p->has_jobs == NULL){
		return -1;
	}
	bsem_init(thpool_p->jobqueue_p->has_jobs, 0);

	jobqueue_clear(thpool_p);
	return 0;
}


/* Clear the queue */
static void jobqueue_clear(thpool_* thpool_p){

	while(thpool_p->jobqueue_p->len){
		free(jobqueue_pull(thpool_p));
	}

	thpool_p->jobqueue_p->front = NULL;
	thpool_p->jobqueue_p->rear  = NULL;
	bsem_reset(thpool_p->jobqueue_p->has_jobs);
	thpool_p->jobqueue_p->len = 0;

}


/* Add (allocated) job to queue
 *
 * Notice: Caller MUST hold a mutex
 */
static void jobqueue_push(thpool_* thpool_p, struct job* newjob){

	newjob->prev = NULL;

	switch(thpool_p->jobqueue_p->len){
```

```
            case 0:    /* if no jobs in queue */
                            thpool_p->jobqueue_p->front = newjob;
                            thpool_p->jobqueue_p->rear    = newjob;
                            break;

            default: /* if jobs in queue */
                            thpool_p->jobqueue_p->rear->prev = newjob;
                            thpool_p->jobqueue_p->rear = newjob;

        }
        thpool_p->jobqueue_p->len++;

        bsem_post(thpool_p->jobqueue_p->has_jobs);
}


/* Get first job from queue(removes it from queue)
 *
 * Notice: Caller MUST hold a mutex
 */
static struct job* jobqueue_pull(thpool_* thpool_p){

        job* job_p;
        job_p = thpool_p->jobqueue_p->front;

        switch(thpool_p->jobqueue_p->len){

            case 0:    /* if no jobs in queue */
                            return NULL;

            case 1:    /* if one job in queue */
                            thpool_p->jobqueue_p->front = NULL;
                            thpool_p->jobqueue_p->rear    = NULL;
                            break;

            default: /* if >1 jobs in queue */
                            thpool_p->jobqueue_p->front = job_p->prev;

        }
        thpool_p->jobqueue_p->len--;

        /* Make sure has_jobs has right value */
        if (thpool_p->jobqueue_p->len > 0) {
            bsem_post(thpool_p->jobqueue_p->has_jobs);
```

```
        }

        return job_p;
}


/* Free all queue resources back to the system */
static void jobqueue_destroy(thpool_* thpool_p){
        jobqueue_clear(thpool_p);
        free(thpool_p->jobqueue_p->has_jobs);
}




/*                  ========================                    SYNCHRONISATION
========================== */


/* Init semaphore to 1 or 0 */
static void bsem_init(bsem *bsem_p, int value) {
        if (value < 0 || value > 1) {
                fprintf(stderr, "bsem_init(): Binary semaphore can take only values 1 or 0");
                exit(1);
        }
        pthread_mutex_init(&(bsem_p->mutex), NULL);
        pthread_cond_init(&(bsem_p->cond), NULL);
        bsem_p->v = value;
}


/* Reset semaphore to 0 */
static void bsem_reset(bsem *bsem_p) {
        bsem_init(bsem_p, 0);
}


/* Post to at least one thread */
static void bsem_post(bsem *bsem_p) {
        pthread_mutex_lock(&bsem_p->mutex);
        bsem_p->v = 1;
        pthread_cond_signal(&bsem_p->cond);
        pthread_mutex_unlock(&bsem_p->mutex);
```

```
}


/* Post to all threads */
static void bsem_post_all(bsem *bsem_p) {
	pthread_mutex_lock(&bsem_p->mutex);
	bsem_p->v = 1;
	pthread_cond_broadcast(&bsem_p->cond);
	pthread_mutex_unlock(&bsem_p->mutex);
}


/* Wait on semaphore until semaphore has value 0 */
static void bsem_wait(bsem* bsem_p) {
	pthread_mutex_lock(&bsem_p->mutex);
	while (bsem_p->v != 1) {
		pthread_cond_wait(&bsem_p->cond, &bsem_p->mutex);
	}
	bsem_p->v = 0;
	pthread_mutex_unlock(&bsem_p->mutex);
}
```

（三）客户端 代码

```
import java.io.*;
import java.net.*;

//用于计算 hash 值
import org.apache.commons.codec.digest.DigestUtils;

class Global
{
	public static String path = "";
}

class peer_server implements Runnable
{
	// 实现一个基本的文件服务器
	public void run()
	{
		try
		{
			// 预定义
			String path = Global.path;
```

```
            // 建立用于接收服务器消息的 ServerSocket
            ServerSocket comServSock = new ServerSocket(7701);

            // 建立用于接收另一个 peer 消息、传输文件的 ServerSocket
            ServerSocket fileServSock = new ServerSocket(7702);

            while(true)
            {
                Socket socket = comServSock.accept();

                //创建输入输出流
                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                PrintWriter out = new PrintWriter(socket.getOutputStream(), false);

                String response = "";
                String[] respArray;

                // 循环监听连接请求，直到接收到"HELLO"或者"QUIT"握手消息
                while(!response.equals("HELLO") && !response.equals("QUIT"))
                {
                    response = in.readLine();
                    // 如果接收到的握手消息是 OPEN，回复确认消息 HELLO
                    if(response.equals("HELLO"))
                    {
                        out.println("ACCEPT");
                        out.flush();
                    }
                }

                // 循环监听连接请求，直到收到 QUIT 握手消息
                while(!response.equals("QUIT"))
                {
                    response = in.readLine();

                    //请求参数分段处理
                    respArray = response.split(" ");

                    // syntax: GET [filename]
                    if(respArray[0].equals("GET"))
                    {
                        try
                        {
                            // 请求的文件名不为空
```

```
                                    if(!respArray[1].isEmpty())
                                    {
                                        // 新建一个用于文件传输的 socket
                                        Socket fileSocket = fileServSock.accept();

                                        File peerfile = new File(path + File.separator +
respArray[1]);
                                        byte[] buffer = new byte[(int)peerfile.length()];
                                        BufferedInputStream fileIn = new
BufferedInputStream(new FileInputStream(peerfile));
                                        fileIn.read(buffer, 0, buffer.length);
                                        BufferedOutputStream fileOut = new
BufferedOutputStream(fileSocket.getOutputStream());
                                        fileOut.write(buffer, 0, buffer.length);
                                        fileOut.flush();
                                        fileIn.close();
                                        fileOut.close();
                                        fileSocket.close();

                                        out.println("OK");
                                        out.flush();
                                    }
                                }
                                catch (Exception e)
                                {
                                    out.print("ERROR "+e);
                                    out.flush();
                                }
                            }
                            else if(response.equals("CLOSE"))
                            {
                                continue;
                            }
                        }
                        out.print("GOODBYE");
                        out.flush();
                        socket.close();
                    }
                }
                catch (Exception e)
                {
                    System.out.println("\033[1;31m[错误] >>\033[0m "+e);
                    System.exit(-1);
                }
```

```
    }
}

public class Nap
{
    public static void error_handler(String err)
    {
        System.out.println("\033[1;31m[错误] >>\033[0m " + err.substring(6));
        System.exit(-1);
    }

    // Main method
    public static void main(String[] args)
    {
        try
        {
            System.out.println("Nap 客户端");

            Socket socket;
            BufferedReader in;
            PrintWriter out;

            // 初始化用于接收用户输入的 stdin
            BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

            String server;//P2P 服务器 IP
            int port;//P2P 服务器端口
            String path;//本地 P2P 工作目录

            String request = "";
            String[] reqArray;
            String response;
            String[] respArray;

            //获取几个必要信息
            System.out.print("服务器的 IP 地址 >> ");
            server = stdin.readLine();
            System.out.print("服务器的端口号 >> ");
            port = Integer.parseInt(stdin.readLine());
            System.out.print("本机的工作目录 >> ");
            path = stdin.readLine();
            Global.path = path;
```

```
            socket = new Socket(server, port);
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));//服务器返回的消息
            out = new PrintWriter(socket.getOutputStream(), false);//发送给服务器的消息

            // 打印服务器返回的消息
            System.out.println(in.readLine());

            // 发送握手消息"CONNECT"，开始握手
            out.print("CONNECT");
            out.flush();
            response = in.readLine();

            // 接收确认信息
            if(!response.equals("ACCEPT"))
            {
                System.out.println("\033[1;31m[错误] >>\033[0m 向服务端发送的握手信息未能接收到正确的确认包");
                System.exit(-1);
            }
            else
            {
                System.out.println("\033[1;32m[成功] >>\033[0m 成功连接到 Napd 服务器 " + server + ":" + port);
            }

            File folder = new File(path);
            File[] files = folder.listFiles();
            FileInputStream f_stream;
            String filename;
            String filehash;
            String filesize;
            System.out.println("[信息] 正在为工作目录 " + path + " 建立文件索引...");
            int index_total = 0;

            for(int i = 0; i < files.length; i++)
            {
                if(files[i].isFile())
                {
                    filename = files[i].getName();
                    f_stream = new FileInputStream(files[i]);
                    filehash = DigestUtils.md5Hex(f_stream);
```

```
                    f_stream.close();
                    filesize = String.valueOf(files[i].length());

                    out.print("ADD " + filename + " " + filehash + " " + filesize);
                    out.flush();
                    response = in.readLine();

                    if(!response.equals("OK"))
                        error_handler(response);
                    else
                    {
                        System.out.print(". ");
                        index_total++;
                    }
                }
            }

            System.out.println("\n\033[1;32m[ 成 功 ]  >>\033[0m  成 功 添 加  "  +
index_total + "  个文件信息到服务器");

            // 开启文件服务器线程
            Runnable run = new peer_server();
            Thread thread = new Thread(run);
            thread.start();

            System.out.println("[信息]  等待用户输入");

            do
            {
                System.out.print(">> ");
                request = stdin.readLine();
                reqArray = request.split(" ");

                if(request.equals("list"))
                {
                    System.out.println("[信息]  正在向服务器请求文件列表...");

                    // 发送 LIST 命令
                    out.print("LIST");
                    out.flush();

                    int list_total = 0;
                    response = in.readLine();
                    respArray = response.split(" ");
```

```
                    while((!respArray[0].equals("OK"))                    &&
(!respArray[0].equals("ERROR")))
                    {
                        list_total++;
                        System.out.println(String.format("[%2d] : %20s [文件大
小: %10s]", new Object[] { new Integer(list_total), respArray[0], respArray[1] }));


                        response = in.readLine();
                        respArray = response.split(" ");
                    }
                    System.out.println("[信息] 一共获取到 " + list_total + " 个文件
");

                    if(!response.equals("OK"))
                        error_handler(response);
                }
                else if(reqArray[0].equals("request"))
                {
                    try
                    {
                        if(!reqArray[1].isEmpty())
                        {
                            //发送 REQUEST
                            out.print("REQUEST " + reqArray[1]);
                            out.flush();

                            response = in.readLine();
                            respArray = response.split(" ");
                            if(respArray[0].equals("OK"))
                                System.out.println("\033[1;31m[错误] >>\033[0m
在服务器上并未找到文件'" + reqArray[1]);

                            while((!respArray[0].equals("OK"))                    &&
(!respArray[0].equals("ERROR")))
                            {
                                //respArray 格式：peer 的 IP+文件大小
                                Socket comSocket = new Socket(respArray[0],
7701);

                                String comResponse;
                                BufferedReader comIn = new BufferedReader(new
InputStreamReader(comSocket.getInputStream()));
                                PrintWriter comOut = new
```

```
PrintWriter(comSocket.getOutputStream(), false);

                                        //验证身份
                                        comOut.println("HELLO");
                                        comOut.flush();
                                        comResponse = comIn.readLine();

                                        //确认
                                        if(!comResponse.equals("ACCEPT"))
                                        {
                                            System.out.println("\033[1;31m[                错
误] >>\033[0m 客户端握手消息验证失败");
                                            System.exit(-1);
                                        }

                                        Socket fileSocket = new Socket(respArray[0],
7702);
                                        comOut.println("GET " + reqArray[1]);
                                        comOut.flush();
                                        InputStream fileIn = fileSocket.getInputStream();

                                        File f = new File(path+File.separator+"recv");
                                         if (!f.exists())
                                         {
                                             f.mkdirs();
                                         }
                                        BufferedOutputStream fileOut = new
BufferedOutputStream(new FileOutputStream(path+File.separator+"recv"+File.separator +
reqArray[1]));
                                        int bytesRead,current = 0;

                                        byte[] buffer = new
byte[Integer.parseInt(respArray[1])];
                                        bytesRead = fileIn.read(buffer, 0, buffer.length);
                                        current = bytesRead;

                                        System.out.println("[信息] 开始传输文件...");
                                        do
                                        {
                                            System.out.print(". ");

                                            bytesRead = fileIn.read(buffer, current,
(buffer.length - current));
                                            if(bytesRead >= 0)
```

```
                                        current += bytesRead;
                                    } while(bytesRead > -1 && buffer.length != current);

                                    fileOut.write(buffer, 0, current);
                                    fileOut.flush();

                                    System.out.println("\n\033[1;32m[                成功] >>\033[0m 文件传输成功");

                                    fileIn.close();
                                    fileOut.close();
                                    fileSocket.close();

                                    respArray[0] = "OK";

                                    response = in.readLine();
                                    respArray = response.split(" ");

                                }


                                if(!respArray[0].equals("OK"))
                                    error_handler(response);
                            }
                        }
                        catch (Exception e)
                        {
                            System.out.println("\033[1;31m[错误] >>\033[0m "+e);
                        }
                    }
                } while(!request.equals("quit"));

                out.print("QUIT");
                out.flush();

                response = in.readLine();
                if(!response.equals("GOODBYE"))
                {
                    System.out.println("\033[1;31m[错误] >>\033[0m 程序未正常退出：" + response);
                    System.exit(-1);
                }
                else
```

```
                {
                    System.out.println("\033[1;32m[成功] >>\033[0m  成功关闭连接");
                }

                in.close();
                out.close();
                socket.close();
            }
            catch (Exception e)
            {
                System.out.println("\033[1;31m[错误] >>\033[0m "+e);
            }
        }
}
```

7、实验报告

按实验步骤的内容作详细记录、分析。在实验报告中写出网络配置过程以及分析网络中的二层交换、三层交换机和路由器的作用。回答报告中提出的问题。