

《计算机网络》课程实验手册

(2020-2021 学年第 2 学期)

供 2019 级卓越班和普通班使用

2021 年 3 月 20 日

目 录

1 简介	1
1.1 教学形式	1
1.2 课前要求	1
1.3 操作规范	1
1.4 作业提交	1
2 主要技术	4
3 利用可见光传输信息的软件	5
3.1 实验目的	5
3.2 用户需求	5
3.3 输入输出接口	5
3.4 技术建议	7
3.5 提交材料	7
3.6 附录一：学生作品	7
4 利用可见光传输帧的软件	9
4.1 实验目的	9
4.2 用户需求	9
4.3 输入输出接口	9
4.4 技术建议	10
4.5 提交材料	11
5 基于 PCAP 库侦听并分析网络流量	12
5.1 实验目的	12
5.2 用户需求	12
5.3 安全警告	13
5.4 附录一	14
5.5 附录二	14
5.6 附录三	25
5.7 附录四	30

6 CISCO IOS 路由器基本配置	31
6.1 实验目的	31
6.2 实验要求	31
6.3 操作步骤	31
6.4 附录一	32
7 利用 Socket API 实现许可认证软件	77
7.1 实验目的	77
7.2 需求说明	77
7.3 输入输出接口	77
7.4 技术建议	78
7.5 附录一：编写许可证服务器	78
8 应用层协议服务配置	120
8.1 实验要求	120
8.2 操作步骤	120
8.3 附录一：服务配置示例	120
9 代理服务器软件	156
9.1 实验目的	156
9.2 需求说明	156
9.3 输入输出接口	156
9.4 设计思路	157
9.5 特别说明	157
9.6 附录一：Socks 协议下载地址	157
9.7 附录二：SOCKS_PROXY 实现示例程序	158
10 RJ-45 接口网线的制作	175
10.1 实验要求	175
10.2 操作步骤	175
10.3 扩展实验	175
10.4 附录一 RJ-45 连接线实验	176
10.5 附录二 无线局域网的构建	180

11 RS-232 通信程序设计	181
11.1 实验要求	181
11.2 操作步骤	181
11.3 附录一 RS232 连线实验	182
11.4 附录二 .Net 或 Java 串口开发	185
11.5 附录三 利用 VMWare 虚拟机仿真	185
11.6 附录四 利用 VSPD 虚拟驱动器仿真	186
12 利用 Socket API 实现网上点对点通信	188
12.1 实验要求	188
12.2 操作步骤	188
12.3 附录一：UDP 和 TCP 编程介绍	190
12.4 附录二：网络多线程 2048 游戏学生作品	197

1 简介

1.1 教学形式

本课程的主讲教师为黄炜副教授（whuang@xmu.edu.cn）。每班配备一名教学助理。实验课采用教学助理主持，优秀学生带动和辅导其他学生的方式进行。

1.2 课前要求

学生应提前2周预习，观看教师提供的教程、教材和录像等。有条件的学生应在虚拟机上实现，掌握实验要领，总结疑问。将程序带到课堂，在实验课上与同学讨论。

1.3 操作规范

学生应稍提前到课，以免陆续进入影响他人听课。不得迟到早退。做实验时，应遵守校规校纪，按实验室规章制度规范操作。遇到损坏的耗材或无法正常使用的计算机，应向助教老师报告，以便及时补充更换，方便后续同学使用。

鼓励和同学探讨争论，允许离开座位。禁止进行与课程主题无关的活动，如闲聊、看电影和玩手机等。不得在实验室内用餐，实验中禁止打开机箱，注意用电安全。

实验结束后，属于重复利用的器材应当归还，设备应关机。如果实验需要拔出电源或网线等，实验后应恢复原状，方便后续同学使用。

1.4 作业提交

实验课每位学生应负责提交本人作业，在XMU Gitlab Gitee(码云)或GitHub等代码服务器上按各实验主题自建仓库（Repository）。每次实验课下课后2周内应提交实验报告，逾期为补交。

学生在实验准备阶段编写必要的程序或准备必要的数据，验证调试达到实验目的后。在实验时观察实验现象，得出实验结论。此时，对关键步骤应用“PrintScreen”键截全屏幕，或用“Alt-PrintScreen”键截活动窗口，再粘贴到画图板或者Word文档中。

实验结束后，学生继续完善程序和数据，并按实验报告的模板所要求事项书写报告，将截图和文字说明保存为 Word 文档（后缀为.docx），名为：report.docx。并将 Word 文档另存为 PDF 格式，形如：report.pdf。

对于有需要程序设计与实现的实验，建议按如下目录结构组织。

目录名	内容	注意事项
src	源代码工程	复制前先删除无关的缓存文件。 Visual Studio 请运行“清理解决方案”，并删除“.sln”所在的“.vs”文件夹； Java 请删除 build 文件夹。 以免引入数 MB 甚至 GB 的无关文件。
	代码库	使用第三方代码库或者包的，应在正确的位置下包含该包，方便编译。
bin	可执行文件	复制前先删除无关的调试文件。 C++ 工程请先删除 obj、pdb 文件，留下必需的 exe 和 dll 文件即可。 Java 或 Python 开发应提交 bat（Windows 下）或 sh（Linux 下）文件，其内容如：“python main.py”，以便一键运行。
/	说明文件	命名为“readme.txt”（文本文件格式）或者“readme.md”（学有余力的同学可以尝试 Markdown 格式），简单写入必要的说明。



- 1) 组长按各组组员建立项目库，名称形如：“E1234”。其中，1234 为组员学号后四位。
- 2) 组员在项目库下新建“E2_1234”文件夹，其中，2 为实验序号（意思是实验 2），1234 为组员学号后四位。
- 3) “E2_1234”文件夹下包含文件夹结构参考上图。
- 4) 在 commit 以后记得 push 到服务器，确认服务器上可以正确显示。新版可以覆盖旧版提交，历史信息是可以查询到的。

本课程结束前应自行保留实验报告和代码的副本，以免服务器故障需要重新传输。

2 主要技术

为了顺利完成以下实验，学生应该掌握以下技能：

- 1、安装虚拟机。
- 2、安装 Windows 系统，Windows 系统的使用。
- 3、安装 Linux 系统，Linux 系统的常用命令行操作。
- 4、C++程序设计。
- 5、Java 或 C#程序设计。
- 6、Web 程序设计，PHP、Java 或 .Net。

3 利用可见光传输信息的软件

3.1 实验目的

通过完成实验，理解物理层传输的基本原理。掌握传输过程中的编解码过程，熟悉传输中的噪声、分辨率、波特率、调制和误码等通信概念；了解奈氏定理和香农定理的含义。

3.2 用户需求

该实验分为发送方和接收方 2 个部分。

发送方将信息编码为一组信号，并通过可见光发送。接收方通过手机录制视频的形式接收，并解码得到原始消息。

由于大多数同学普遍尚无法完成手机编程，编码部分可简化为：

- 1、接受用户输入消息。
- 2、利用 FFMPEG 或 OPENCV 编码生成一系列图像。
- 3、利用 FFMPEG 命令行将图像编码为视频。
- 4、利用系统播放器播放在普通计算机液晶显示器上。

解码部分可简化为：

- 1、手机录制显示器播放的视频，通过 QQ 或 USB 线复制到计算机上。
- 2、利用 FFMPEG 命令行将视频解码为图像。
- 3、利用 FFMPEG 或 OPENCV 接受上述输入的一组图像。
- 4、解码得到消息，输出在界面上。

3.3 输入输出接口

学生应导出一套用于测评的程序，这里说明其输入输出接口。

编码器命令行接口：

1、用户提供一个文件名（ argv[1] ），内容为用户随机生成的一个二进制文件。

2、用户提供一个文件名（ argv[2] ），用于将编码后的视频输出到该文件。

3、用户提供一个整数（ argv[3] ），用于约束编码后的视频长度。单位：毫秒。

编码器命令行接口样例：用户通过“encode in.bin in.mp4 1000”调用程序。

解码器命令行接口：

1、用户提供一个文件名（ argv[1] ），内容为用户手机拍摄的视频；

2、用户提供一个文件名（ argv[2] ），用于将解码后的二进制数组输出到该文件。

3、用户提供一个文件名（ argv[3] ），用于将解码信息是否有效的二进制数组输出到该文件。

解码器命令行接口样例：用户通过“decode out.mp4 out.bin vout.bin”调用程序。

解码器输入接口：

1、用户复制到计算机的视频。

解码器输入接口样例：（无法提供）

解码器输出接口：

1、解码生成的一个二进制信息文件。

2、解码生成的一个二进制文件，指示信息文件的每一位是否有效。

解码器输出接口样例：

设文件“out.bin”内容为：（二进制文件而非文本文件）

13 2F 9B C7

设文件“vout.bin”内容为：（二进制文件而非文本文件）

FF	FC	FF	F0
----	----	----	----

解码器输出接口说明：程序应自动调用 ffmpeg 命令行。在程序运行结束时，程序最佳状态为原样恢复出编码前的文件。

文件 vout.val 对应于 out.bin，标记每个位是否有正确（1 表示正确，0 表示错误），以此类推。其中，各文件为二进制文件。

3.4 技术建议

- 1、本项目自选开发语言，但推荐 C 和 C++。
- 2、了解 ffmpeg 命令行的使用，调节帧率（fps）等参数。
- 3、了解如何调用命令行。
- 4、了解 OPENCV 库的使用。

3.5 提交材料

学生在实现软件以后，在报告中除了给出核心代码截图与运行界面截图外，说明以下问题，包括测算相关指标。

- 1、在实验中的，编解码算法是什么？
- 2、在实验中的，调制和解调算法是什么？其中，载体信号、调制信号是什么？使用的算法属于调频、调幅还是调相？
- 3、在实验中的，主要的噪声强度有多大，噪声来自哪些因素？
- 4、你的编码算法分辨率是多少？（作为实验，不要求分辨率太大）
- 5、你的编码波特率是多少？传输率是多少？
- 6、按奈氏定理和香农定理，通信率上限是多少？

此外，鼓励学生通过编程调用库而不是调用 ffmpeg 可执行程序的形式来操作视频。鼓励学生编辑图形用户界面。鼓励操作过程尽量流畅。

3.6 附录一：学生作品

以下代码来自厦门大学信息学院软件工程系 2018 级本科生李泽政同学。

<https://github.com/poor-circle/Visual-Net>

4 利用可见光传输帧的软件

4.1 实验目的

通过完成实验，理解数据链路层传输的基本原理。掌握传输过程中的帧格式设计理念；熟悉传输中的帧与成帧、帧定界符等通信概念，熟悉多方通信中的时分、频分、波分或码分多路复用与解复用等概念，熟悉多方通信中的编址的概念。

4.2 用户需求

在前项实验（第3章）的基础上，设计一个允许多方通信的程序，参照以太网机制，设计良好的帧格式，设计帧首尾定界符，在帧格式中体现编址概念。

4.3 输入输出接口

学生应导出一套用于测评的程序，这里说明其输入输出接口。

编码器命令行接口：

1、用户提供一个文件目录（`argv[1]`），其下有多个扩展名为“.bin”的文件，分别为形如“12.bin”，内容分别为用户随机生成的一个二进制文件。

2、用户提供一个整数（`argv[3]`），用于指示最大传输单元，单位：字节。
如：1500，表示参与通信的帧总长度最大为1500字节。

3、用户提供一个文件名（`argv[4]`），用于将编码后的视频输出到该文件。

4、用户提供一个整数（`argv[5]`），用于约束编码后的视频长度。单位：毫秒。

编码器命令行接口样例：用户通过“`encode ./ 150 in.mp4 1000`”调用程序。

解码器命令行接口：

1、用户提供一个文件名（`argv[1]`），内容为用户手机拍摄的视频；

2、用户提供一个文件目录（`argv[2]`），用于将解码后的二进制数组输出到该文件目录下的多个扩展名为“.bin”的文件，分别为形如“12.bin”，其中文件名的部分是参加复用多方的编址。同时，将解码信息是否有效的二进制数组输出到同名的扩展名为“.val”的文件，如“12.val”。

解码器命令行接口样例：用户通过“decode out.mp4 ./”调用程序。

解码器在指定文件目录下输出：

1、解码生成的多个二进制信息文件。

2、解码生成的信息文件对应的多个二进制文件，指示信息文件的每一位是否有效。

解码器输出接口样例：

设文件“12.bin”内容为：（二进制文件而非文本文件）

13 2F 9B C7

设文件“12.bin”内容为：（二进制文件而非文本文件）

FF FC FF F0

解码器输出接口说明：程序应自动调用 ffmpeg 命令行。在程序运行结束时，程序最佳状态为原样恢复出编码前的文件。

文件 12.val 对应于 12.bin，标记每个位是否有正确（1 表示正确，0 表示错误），以此类推。其中，各文件为二进制文件。

4.4 技术建议

1、帧格式设计可参考以太网，适当调节各部分的位数。编址的长度占比不应过大，应有 CRC 校验，例如：

帧首定界符	目的地址	源地址	长度	载荷	CRC
1 字节	1 字节	1 字节	2 字节	0~65535 字节	4 字节

2、帧首尾定界符应有设计，设帧首定界符是 0xCA，帧内所有 0xCA 的部分将用 0xCC、0xCA 代替，并且所有 0xCC 的部分将用 0xCC、0xCC 代替；

3、建议采用统计时分多路复用

4.5 提交材料

学生在实现软件以后，在报告中除了给出核心代码截图与运行界面截图外，说明以下问题，包括测算相关指标。

- 1、在实验中的，帧格式是什么？各个字段的作用是什么？长度范围是多少？
- 2、在实验中的，如何将数据成帧？其中，帧首定界符是什么？
- 3、在实验中的，采用的是何种多路复用算法？简要说明其基本原理。
- 4、在实验中，编址方案是什么？为什么这么设计？
- 5、你的编码数据载荷大小范围是什么？设定该上限值的依据是什么？

5 基于 PCAP 库侦听并分析网络流量

5.1 实验目的

通过完成实验，理解数据链路层、网络层、传输层和应用层的基本原理。掌握用 Wireshark 观察网络流量并辅助网络监听相关的编程；掌握用 Libpcap 或 WinPcap 库监听并处理以太网帧和 IP 报文的方法；熟悉以太网帧、IP 报文、TCP 段和 FTP 命令的格式概念，掌握 TCP 协议的基本机制；熟悉帧头部或 IP 报文头部各字段的含义。熟悉 TCP 段和 FTP 数据协议的概念，熟悉段头部各字段和 FTP 控制命令的指令和数据的含义。

5.2 用户需求

以下是本次实验的用户需求：

1、用监听解析软件观察数据格式

用 Wireshark 或 OmniPeek 等网络监听软件网络上的数据流，验证理论课讲授的网络协议层次嵌套，验证帧格式、IP 报文格式、TCP 段格式和 FTP 协议命令和响应的格式，验证 MAC 地址、IP 地址、TCP 端口等协议地址格式。

2、用监听解析软件观察 TCP 机制

用 Wireshark 监听并观察 TCP 数据段。观察其建立和撤除连接的过程，观察段 ID、窗口机制和拥塞控制机制等。将该过程截图在报告中。

3、用 Libpcap 或 WinPcap 库监听网络数据

用 Libpcap 或 WinPcap 库监听网络上的数据流，解析发送方与接收方的 MAC 和 IP 地址，并作记录与统计。程序在文件上输出形如下列 CSV 格式的日志：

时间、源 MAC、源 IP、目标 MAC、目标 IP、帧长度（以逗号间隔）

2015-03-14 13:05:16,60-36-DD-7D-D5-21,192.168.33.1,60-36-DD-7D-D5-72,192.168.33.2,1536

每隔一段时间（如 1 分钟），程序统计来自不同 MAC 和 IP 地址的通信数据长度，统计发至不同 MAC 和 IP 地址的通信数据长度。

4、解析侦听到的网络数据

用 Wireshark 倾听并观察 FTP 数据，分析其用户名密码所在报文的上下文特征，再总结出提取用户名密码的有效方法。解析协议内容，并作记录与统计。对用户登录行为进行记录。程序在文件上输出形如下列 CSV 格式的日志：

时间、源 MAC、源 IP、目标 MAC、目标 IP、登录名、口令、成功与否

2015-03-14 13:05:16,60-36-DD-7D-D5-21,192.168.33.1,60-36-DD-7D-D5-72,192.168.33.2,student,software,SUCCEED

2015-03-14 13:05:16,60-36-DD-7D-D5-21,192.168.33.1,60-36-DD-7D-D5-72,192.168.33.2,student,software1,FAILED

通过监听 FTP 通信时的网络分组，了解 FTP 的通信协议的过程。在此基础上，重点掌握 FTP 登录环节的通信过程。

通过实验 3 熟悉网络分组的格式，并对“数据区”进行分析。通过对分组中“数据区”的分析，提取用户名、密码、登陆是否成功的反馈信息。一般登录名以“USER”开头，口令以“PASS”开头，登录成功以“230”开头，失败以“530”开头。

如有条件，可以在虚拟机上访问数据，在主机倾听，用以模拟交换机上的“镜像端口”或者在局域网内倾听的过程。注意：由于多数网络交换机分割了广播域，因此你在局域网内侦听到的包可能有限。

建议学有余力的同学尝试将程序输出存入数据库中，并制作简单的网页展示实时倾听结果，对超过流量阈值的部分进行告警。

5.3 安全警告

请务必注意以下安全警告事项：

1、在测试过程中不要输入敏感的密码，如自己的银行卡密码、QQ 密码等，以防被写入表格。

2、在本机搭设 FTP 服务器，勿使用公开的 FTP 服务器，影响其他组织服务器的正常运行。

5.4 附录一

观看 Bilibili 站点上的《<计算机网络>实验 3 用 WinPCAP 倾听并解析分组》讲解视频。地址：<https://www.bilibili.com/video/BV1TE411s7W2>

5.5 附录二

第 3 章 网络协议分析

3.1 协议分析软件概述

目前局域网普遍使用以太网组网，其链路层采用载波监听多路访问/冲突检测（Carrier Sense Multiple Access Collision Detect, CSMA/CD）协议。数据帧的传送方式是将要发送的帧传输到域内所有主机。按照（SMA/CD）协议，只有网卡的 MAC 地址与帧头目标地址一致的那台主机才能接收数据帧，但是，当主机工作于混淆模式（监听模式）时，则无论到达网卡的数据帧目标地址是什么，主机都将接收下来。网络协议分析就是利用这种模式截获流经网卡的所有数据帧，并对帧内数据内容进行分层剖析。协议分析是学习网络协议的极好途径，也是开发网络软件的核心技术。纯软件的协议分析系统，在业内比较知名的有 Sniffer Pro、OmniPeek 等。它们都可以使用普通的网卡来进行数据采集。帧采集的范围依据交换设备的不同存在差异，一般有以下两种情况。

1. 共享式网络

使用集线器（Hub）作为交换设备的局域网，即为共享式网络。集线器以共享模式工作在 OSI 的物理层。协议分析软件可以安装在这种局域网中的任意一台主机上，此时软件可以捕获局域网中所有传输的数据帧。

2. 交换式网络

使用交换机（Switch）作为交换设备的网络，即为交换式网络。交换机工作在 OSI 模型的数据链路层，它的各端口之间能有效分隔冲突域。交换机依据端口与 MAC 地址的映射表转发数据帧，通常不会将帧广播到所有端口，所以接在交换机上的主机一般不能监听到其他端口上收发的数据帧。如果交换机具备镜像功能，则可在交换机上配置好端口镜像（通常是镜像接有路由器或服务器的端口），再将协议分析软件安装到连在镜像端口的主机上，此时软件可以捕获整个局域网进出口的所有数据帧，其网络拓扑如图 3.1 所示。

如果交换机没有端口镜像的功能，则可以将一个集线器连接在交换机的路由端口上，再将路由器和装有网络分析软件的计算机接到集线器上，利用集线器的广播功能，可获得与端口镜像同样的效果。其网络拓扑如图 3.2 所示。

OmniPeek Personal 是一种对网络进行协议分析的理想工具。人们可以将其作为软件研发辅助的测试、检验工具，也可以用来学习网络协议。这个产品向所有中国用户免费开放，OmniPeek Personal 包含有超过 1 000 种协议的解码，可以支持 TCP/IP、Microsoft、IBM、Cisco 等各类协议簇的分析，并且包含了基本的故障诊断分析功能。其局限性在于最多支持 250 个节点和 25

条专家系统分析条目，并且只能在一块网卡上工作。

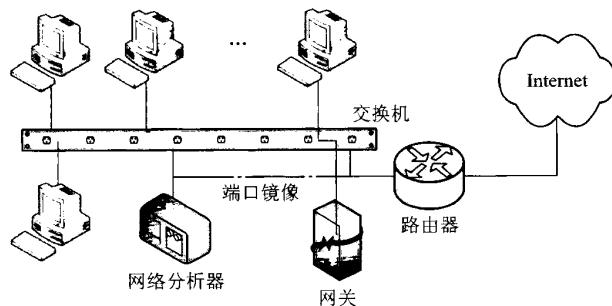


图 3.1 具备镜像功能的交换式网络

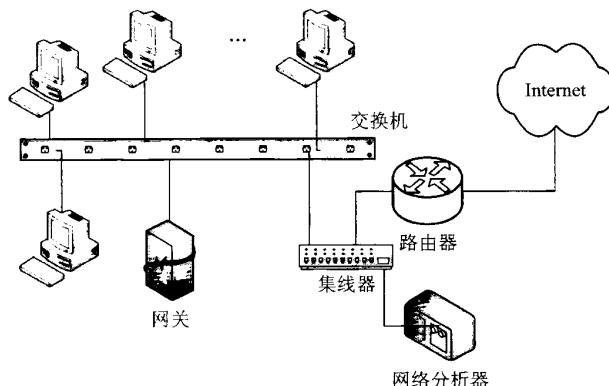


图 3.2 不具备镜像功能的交换式网络

使用 OmniPeek Personal 首先需要设置相关的协议过滤器，过滤条件可以包括协议类型、IP 地址范围、数据流方向、端口等选项。随后即可按照过滤条件进行数据捕获，并对捕获的数据包进行相应的分析。下面将通过几个基础实验来掌握协议分析的相关操作流程和分析技巧。

3.2 使用 OmniPeek 捕获链路数据帧

一、实验要求

熟悉网络监听软件 OmniPeek 的基本操作，并对其所捕获的链路数据帧进行分析。

二、实验条件

联网 PC, 装有 Windows XP 操作系统和网络监听软件 OmniPeek(实验指导使用 Personal 4.1 版本), 集线器。

三、实验指导

数据链路层 (Data Link Layer, DLL) 位于物理层与网络层之间, 它是 OSI 中比较重要的一层。数据链路层通常又分为逻辑链路控制 (Logical Link Control, LLC) 子层和介质访问控制 (Media Access Control, MAC) 子层。前者为物理上直接相连的两台计算机间数据帧的传输提供一定程度的错误控制, 后者则提供多个站点共享同一传输介质的分配策略。数据链路层的协议数据单元是帧。帧封装的数据信息中, 包含了帧头和数据段。前者含有发送节点和接收节点的 MAC 地址, 以及标识帧类型 (IP、ARP 或 RARP) 的字段, 后者包含实际要传输的数据。

下面用实例说明如何用 OmniPeek 捕获网络命令 arp 和 ping 产生的数据帧, 并分析帧结构。具体步骤如下:

① 启动 OmniPeek, 打开 File 菜单, 在下拉菜单中单击 New, 设置一个新的数据捕获器 (Capture), 如图 3.3 所示。

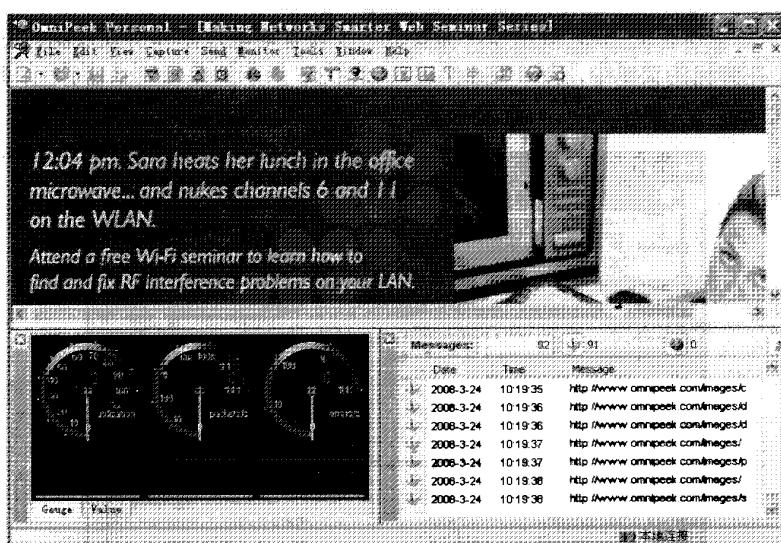


图 3.3 Omni Peek 启动界面

② 在 “Capture Options—本地连接” 对话框中, 单击左栏中的 Filters, 选择要过滤的网络协议类型。在本例中, 因为要捕获 ARP 数据, 所以选中 IP (包含了 ARP 和 RARP 协议), 如

图 3.4 所示。

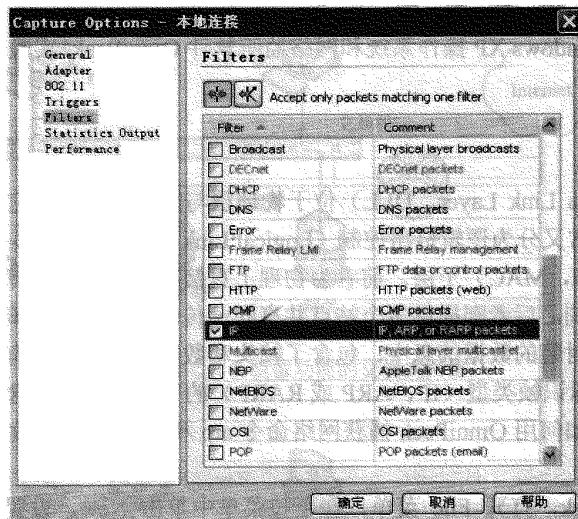


图 3.4 协议设置界面

③ 为了只捕获本机的 ARP 数据帧，需要进一步设置地址过滤和协议过滤。右键单击已选中的 IP 条目，在弹出的菜单中，选择“Edit”进入如图 3.5 所示的编辑过滤器界面，其中，“Type”表示编辑模式，可将其切换到 Advanced 模式，以便配置过滤条件的逻辑组合。

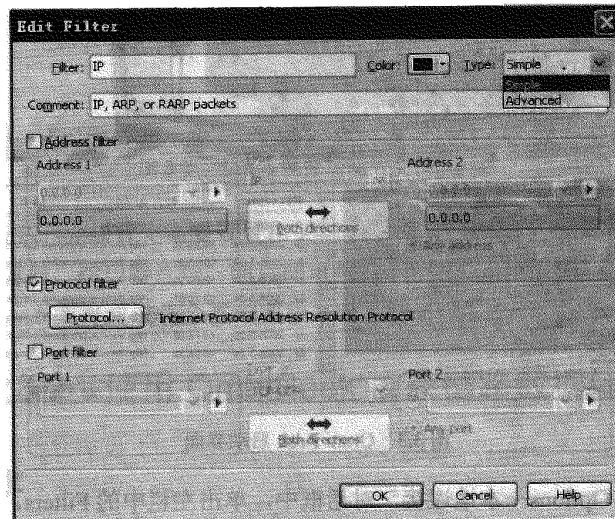


图 3.5 编辑过滤器界面

3.2 使用 OmniPeek 捕获链路数据帧

27

④ 进入过滤条件编辑界面（如图 3.6 所示），右键单击 Protocol IP 方块，在弹出的菜单中，选择“Delete”，删除 IP 协议过滤。类似方法可删除 RARP 协议过滤，即仅捕获 ARP 类型的数据帧。

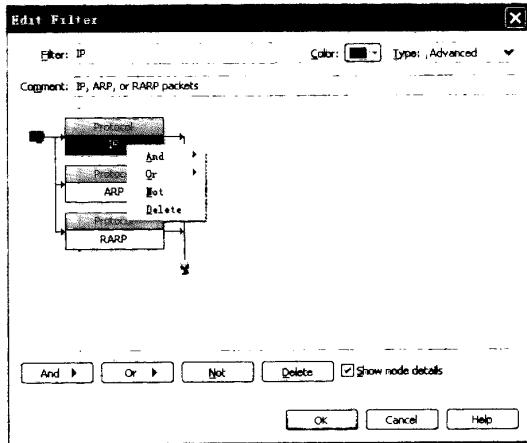


图 3.6 编辑协议过滤器

⑤ 同样在“Edit Filter”对话框中，右键点击 Protocol ARP 方块，在弹出的快捷菜单中，选择“And”→“Address”，进行地址过滤设置。在 Advanced 模式中，可以组合多种条件来定制一个过滤器，如图 3.7 所示。

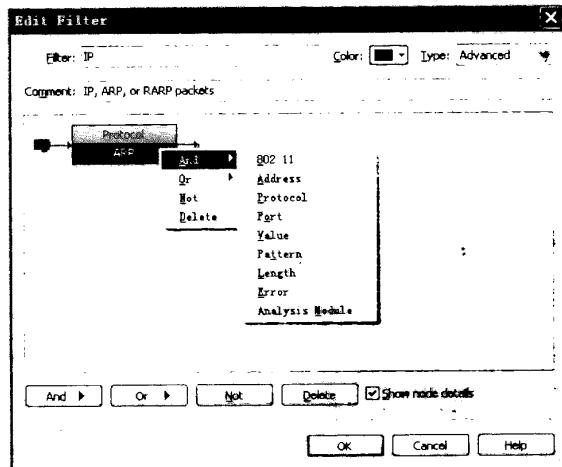


图 3.7 ARP 地址过滤设置

- ⑥ 通过 ipconfig 命令获取本机 IP 地址后，在 Address Filter 中，将 Address1 设置为本机的 IP 地址（本例中为 192.168.2.83），如图 3.8 所示。

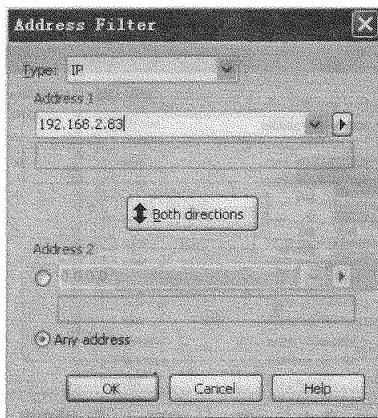


图 3.8 设置 IP 地址过滤

- ⑦ 完成数据捕获器的设置后，OmniPeek 进入如图 3.9 所示的待命状态。

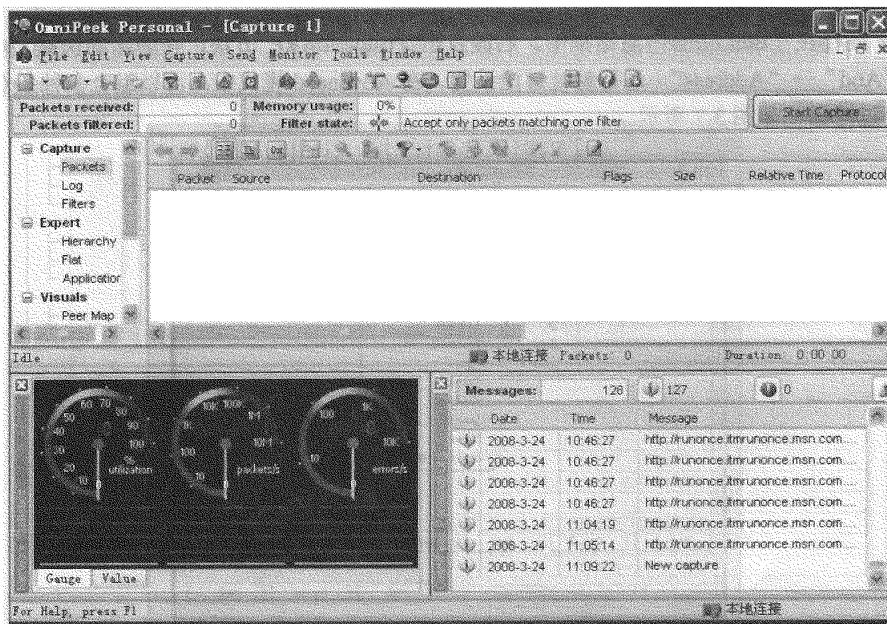


图 3.9 OmniPeek 待命状态

3.2 使用 OmniPeek 捕获链路数据帧

29

⑧ 单击绿色按钮 Start Capture, 使 OmniPeek 开始捕获本机的 ARP 数据包, 同时在 DOS 命令行中, 输入以下几条命令(如图 3.10 所示), 以产生相应的 ARP 数据包。然后, 单击 OmniPeek 中的 Stop Capture 红色按钮, 停止数据捕获。

```
C:\Documents and Settings\public>arp -d

C:\Documents and Settings\public>arp -a
No ARP Entries Found

C:\Documents and Settings\public>ping 192.168.2.1

Pinging 192.168.2.1 with 32 bytes of data:

Reply from 192.168.2.1: bytes=32 time=34ms TTL=128
Reply from 192.168.2.1: bytes=32 time=1ms TTL=128
Reply from 192.168.2.1: bytes=32 time=1ms TTL=128
Reply from 192.168.2.1: bytes=32 time<1ms TTL=128
```

图 3.10 输入命令以产生相应的 ARP 数据包

其中命令 “arp-d” 用于清除 ARP 地址缓存。这样, 当使用 ping 192.168.2.1 命令的时候, 就可以产生 ARP 请求数据包了。注意 192.168.2.1 必须是局域网里面真实存在的机器, 这样才能产生 ARP 响应数据包。

⑨ 查看捕获的 ARP 数据包。

按照 ARP 协议, 捕获的第一个数据包是一个 ARP 请求包, 由本机 (192.168.2.83) 发出的 ARP 请求, 旨在询问目标机器 (192.168.2.1) 的物理地址。可以看到在数据链路层对帧头的解码结果为:

接收节点 (Destination) 的 MAC 地址为: FF:FF:FF:FF:FF:FF (广播地址)。

发送节点 (Source) 的 MAC 地址为本机的 MAC 地址。

协议类型 (Protocol Type): 0x0806 (ARP 协议)。

在 OmniPeek 右侧的 Packets 详细列表中, 可以通过双击指定行来进行解码动作 (Decode), 得到详细的数据包情况。如图 3.11 所示。

四、思考题

1. 对照 ARP 协议, 详细分析捕获的 ARP 请求和 ARP 响应数据帧中各字段的内容。
2. 使用 OmniPeek 捕获由 ping 命令产生的 ICMP 数据包, 并对其进行解码。

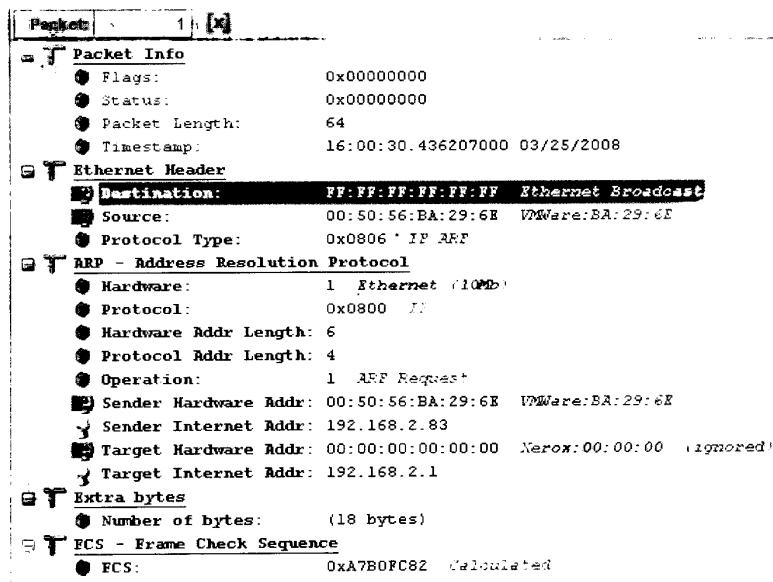


图 3.11 分析捕获的数据帧

3.3 分析由 FTP 命令产生的 TCP 数据包

一、实验要求

使用 OmniPeek 捕获由 FTP 命令产生的 TCP 数据包，并对其进行解码。

二、实验条件

安装 Windows XP 的 PC，装有网络监听软件 OmniPeek Personal 4.1，集线器，FTP 服务器。

三、实验指导

传输控制协议（Transmission Control Protocol, TCP）是一种面向连接的、可靠的、基于字节流的通信协议，由 IETF 的 RFC 793 说明。它与无连接的 UDP 同属于传输层。在因特网协议簇中，传输层是位于 IP 层之上，应用层之下的中间层。不同主机的应用层之间的通信经常需要像管道一样连接、可靠的通信保障，但是 IP 层不能提供这样的流机制，只是提供无连接的不可靠的包交换。这就需要借助传输层来弥补这种缺陷。另一方面，IP 层提供的是主机到主机的通信，而应用层需要进行进程之间的通信。同一主机可以同时运行多个通信进程，这也需要传输

3.3 分析由 FTP 命令产生的 TCP 数据包

31

层通过端口机制来提供这种复用。

应用层某些需要面向连接的通信实体（如 FTP、HTTP、SMTP 通信实体）向传输层的 TCP 通信实体发送通信报文，而后 TCP 把数据流分割成适当长度的报文段（通常受数据链路层的最大传送单元(MTU)的限制），之后 TCP 加入包括端口号在内的 TCP 报头，把数据单元传给 IP 层，由 IP 包承载传送给接收端的 TCP 通信实体。

FTP 是常用的应用层命令，采用 TCP 传输协议，可以采用与 3.2 节实验类似的方法捕获由 FTP 命令产生的本机 TCP 数据包。为了产生数据源，本实验在 DOS 命令行中，输入 ftp 的登录命令（本例为 `ftp://192.168.2.9/`），而后进入 FTP 服务器的某一目录，下载一个文本文件，最后退出 FTP 服务器。这一过程如图 3.12 所示。以下侧重说明如何对 OmniPeek 捕获的 TCP 数据包进行解码。

```
C:\Documents and Settings\public>ftp 192.168.2.9
Connected to 192.168.2.9.
220 Serv-U FTP Server v6.4 for WinSock ready...
User (192.168.2.9:(none)): anonymous
331 User name okay, please send complete E-mail address as password.
Password:
230 User logged in, proceed.
ftp> cd pub
250 Directory changed to /pub
ftp> get hello.txt
200 PORT Command successful.
150 Opening ASCII mode data connection for hello.txt (0 Bytes).
226 Transfer complete.
ftp> bye
221 Goodbye!

C:\Documents and Settings\public>
```

图 3.12 ftp 测试过程

对捕获的数据包进行分析，根据 FTP 和 TCP 协议，前面的 3 个数据包，是 TCP 连接的 3 次握手过程。在 OmniPeek 右侧的 Packets 详细列表中，可以通过双击指定行来进行解码动作（Decode），得到详细的数据包情况。如图 3.13 所示。

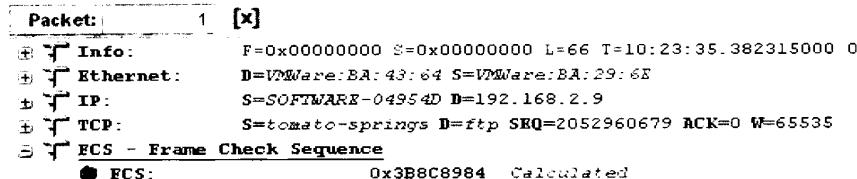


图 3.13 数据包解码

对数据包1，可以看出本机（192.168.2.83）向FTP服务器（192.168.2.9）发送第1次握手信号（SYN）；标志位里面只有个同步位（SYN），表示发出连接请求（如图3.14所示）。

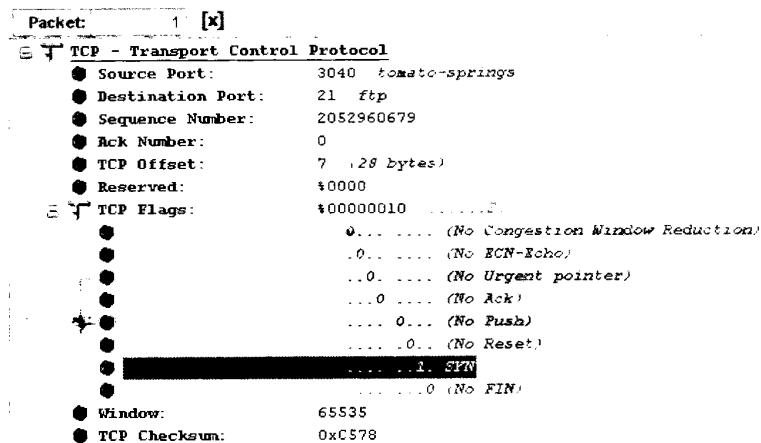


图3.14 TCP连接请求数据包解码

在数据包2中，可以看出FTP服务器（192.168.2.9）向本机（192.168.2.83）发送第2次握手信号（SYN+ACK），标志位里面有同步位和确认位（如图3.15所示）。

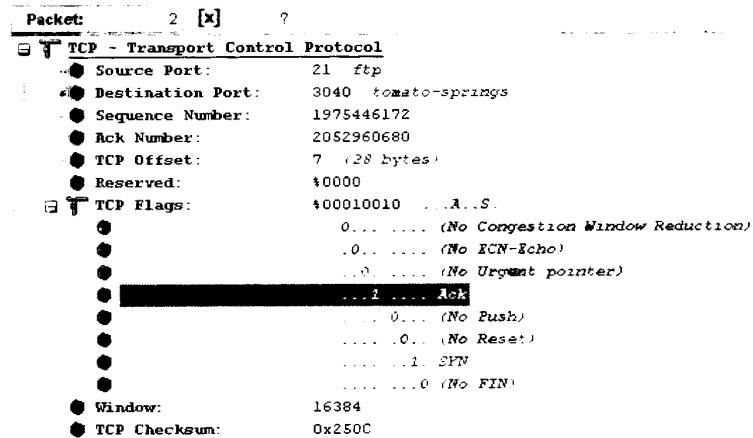


图3.15 TCP连接第二次握手数据包解码

在数据包3中，可以看出本机（192.168.2.83）向FTP服务器（192.168.2.9）发送第3次握手信号（ACK）；标志位里面只有个确认位，也就是做再次确认（图3.16）。经过3次握手后，本机与FTP服务器建立起连接，就可以下载文件了。

3.3 分析由 FTP 命令产生的 TCP 数据包

33

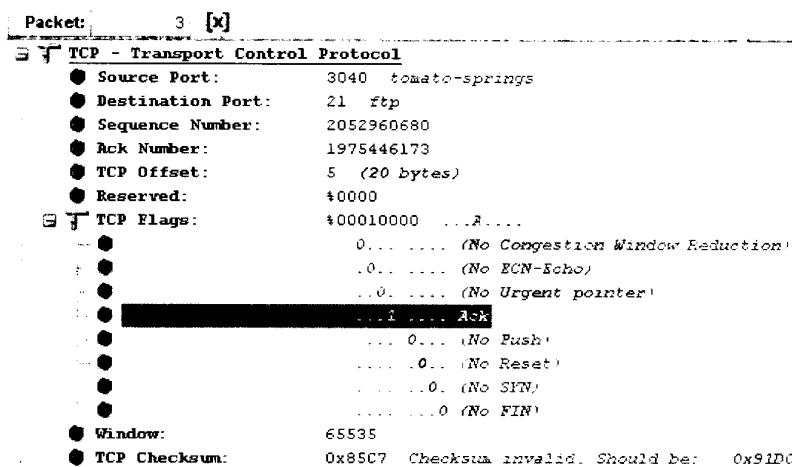


图 3.16 TCP 连接第 3 次握手数据包解码

在 TCP 的数据传送状态，很多重要的机制保证了 TCP 的可靠性和健壮性。包括使用序号对收到的 TCP 报文段进行排序以及检测重复的数据；使用校验和来检测报文段的错误；使用确认和计时器来检测和纠正丢包或延时。

在数据包 4 中，可以看出 FTP 服务器（192.168.2.9）向本机（192.168.2.83）发送第一批数据（FTP 登录的欢迎信息）。这时标志位里面有个发送位（Push），如图 3.17 所示。

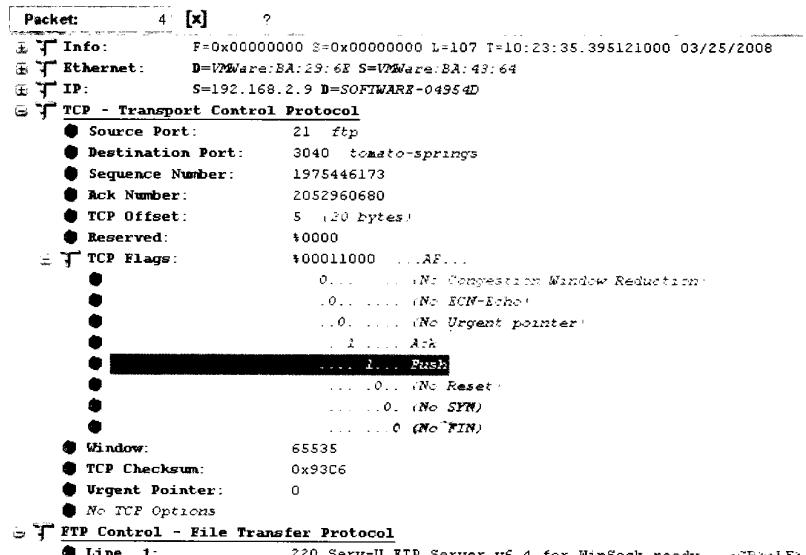


图 3.17 首批数据发送报文解码

在数据包 5 中，可以看到本机（192.168.2.83）向 FTP 服务器（192.168.2.9）发送数据接收确认信号（ACK）。这时标志位里面只有个确认位。确认报文的 Sequence Number（2052960680）采用此前所接收数据包的 Ack Number（2052960680）；确认报文的 Ack Number 则为期待接收的下批数据头一个字符的序号（1975446222），此即表示对由 1975446173 序号开始的首批数据的确认（如图 3.18 所示）。

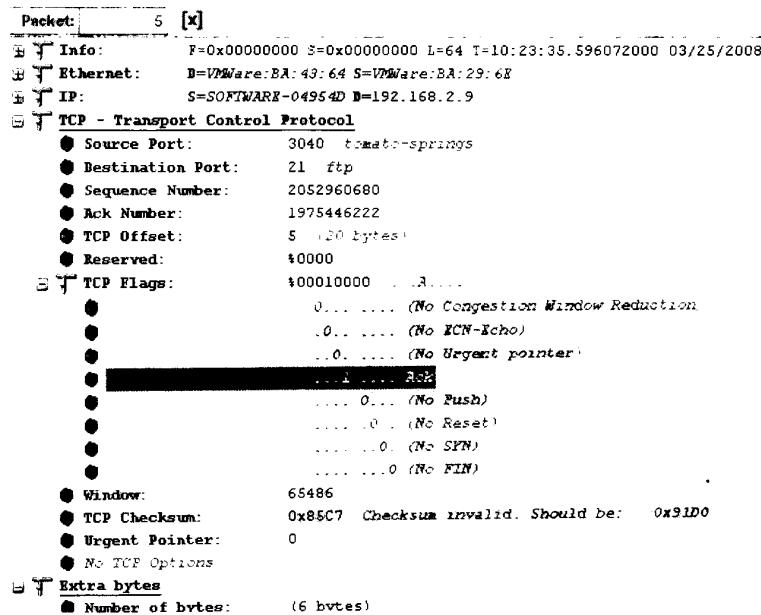


图 3.18 数据接收确认报文解码

TCP 连接的终止使用了 4 次握手过程，在这个过程中每个终端的连接都能独立地被终止。因此，一个典型的拆接过程需要每个终端都提供一对 FIN 和 ACK。对本例从数据包 1 开始的 5 个报文进行解码，可以清楚地看出拆除 TCP 连接的过程。

四、思考题

使用 OmniPeek 捕获由 HTTP 命令产生的 TCP 数据包，并对其进行解码。

5.6 附录三

3.4 使用 Linux C 设计数据帧捕获程序

一、实验要求

使用 Linux C 的基础函数，设计一个实时捕获和分析以太网数据帧的 C 程序，显示每一捕

获帧中发送端和接收端的 MAC 地址以及承载数据的协议类型，最后统计捕获的总帧数以及每一种类型数据帧所占比例。

二、实验条件

若干台微型计算机，连接在共享型或交换型以太网上。

三、实验指导

1. 数据帧在共享型和交换型以太网中的传输机制

对于采用集线器连接的共享型以太网，每一节点发送的数据帧都被广播到集线器上的每一个端口；而对于采用交换机连接的交换型以太网，交换机接收到数据帧后，除非广播帧，否则都将根据帧中的目标地址查找交换机上的端口——MAC 地址对照表，如检索到相应表项则直接转发到对应端口，检索不到时才广播至每一个端口。

2. 以太网数据帧格式

以太网数据帧的格式如图 3.19 所示。

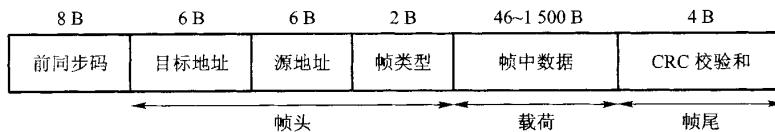


图 3.19 以太网数据帧的格式

目标地址和源地址都是 6 B 的 MAC 地址（网卡地址），而帧类型是帧中载荷的协议类型代码。当网络层运行 IP 协议时，链路层数据帧载荷的协议类型主要有：

- 0x0800: IP（网际协议）。
- 0x0806: ARP（地址解析协议）。
- 0x8035: RARP（反向地址解析协议）。

IP、ARP、RARP 协议数据单元的报头存放在帧承载的数据里。由于 10/100 Mbps 以太网最小帧长为 64 B，最大帧长为 1 518 B，而帧头加帧尾校验码为 18 B，故帧中数据为 46~1 500 B（若有效载荷少于 46 B，则通过填充字段使之达到 46 B）。

3. Linux C 编程要点

① 设置套接字以捕获链路帧。

在 Linux C 下编写网络监听程序，比较简单的方法是在超级用户模式下，利用类型为 SOCK_PACKET 的套接字（socket）函数来捕获链路帧。为此，需要引用如下头文件：

```
#include <sys/socket.h>
socket( )函数的原型是:
int socket(int domain, int type, int protocol);
```

本函数有以下3个输入参数：

- domain：表示所使用的协议簇。
- type：表示套接字的类型。
- protocol：表示所使用协议簇中的某个特定协议。

如果函数调用成功，所建立套接字的描述符（非负整数）就作为函数的返回值，假如返回值为-1则表明调用错误。

Linux C 扩展了 socket()函数，在 type 参数增加了 SOCK_PACKET 类型，使之可以用于捕获链路层的数据帧。为此，domain 参数应指定为 AF_INET，表示采用 Internet 协议簇；type 参数指定为 SOCK_PACKET，表示获取链路帧数据，不作处理；而 protocol 参数采用 htons(0x0003)，表示使用的特定协议是截取所有类型的数据帧。这里 htons()函数用于短整数的字节顺序转换。计算机数据存储有两种字节优先顺序：高位字节优先和低位字节优先。Internet 上数据以高位字节优先顺序在网络上传输，所以对于在内部是以低位字节优先方式存储数据的计算机，在 Internet 上传输数据时就需要进行转换，否则就会出现数据不一致。htons()函数表示把短整数从主机字节序转换成网络字节序。监听链路帧时 socket()函数调用形式为：

```
int fd; // fd 用于存放套接口的描述符
fd=socket(AF_INET, SOCK_PACKET, htons(0x0003));
```

② 设置网卡工作于“混杂”模式。

按照以太网协议，对于流经的数据帧，网卡仅接收目标地址与自身 MAC 地址相符的帧。要使网卡捕获到所有到达帧，则必须首先设置网卡工作于“混杂”模式。这需要用到 ioctl()函数。其原型如下：

```
int ioctl(int d, int request,...)
```

ioctl()函数用于对文件进行 I/O 控制，这里的文件包含套接字、网卡、终端、存储等软、硬件设施，实际的操作来自各个设备自己提供的 ioctl 接口。设置网卡工作于“混杂”模式的 Linux C 参考程序段如下：

```
#include <sys/ioctl.h>
#include <netinet/if_ether.h>
#include <net/if.h>
struct ifreq ifr; int i;
strcpy(ifr.ifr_name,"eth0"); //将网卡设备名“eth0”传入 ifr_name 参数
i=ioctl(fd,SIOCGIFFLAGS,&ifr); //常量 SIOCGIFFLAGS 表示要获取网卡的工作模式
ifr.ifr_flags|=IFF_PROMISC; //在标志中加入“混杂”方式
i=ioctl(fd,SIOCSIFFLAGS,&ifr); //常量 SIOCSIFFLAGS 表示要设置网卡工作模式
if (i<0)
{ perror("can't set promiscuous\n");
exit(0); }
```

③ 分析所捕获链路帧的结构。

套接字建立以后，就可以从中循环读取链路层数据帧。帧中数据的分析主要针对帧头结构进行。而帧头结构类型 ethhdr 已在/usr/include/linux/if_ether.h 中定义：

```
struct ethhdr
{
    unsigned char h_dest[ETH_ALEN];      // 目标 MAC 地址
    unsigned char h_source[ETH_ALEN];    // 源 MAC 地址
    unsigned short h_proto;             // 帧中数据协议类型代码
}
```

为了分析帧结构，需要建立数据帧的缓冲区，并把帧头结构的指针指向这一缓冲区的首地址。

```
char ep[ETH_FRAME_LEN];           // 数据帧的缓冲区
struct ethhdr *eh;                // eh 指向帧头
int fl;                          // fl 用于存放捕获数据帧的帧长
eh=(struct ethhdr *)ep;           // eh 指向帧头
```

随后可类似从文件读取数据一样，从套接字循环捕获数据帧：

```
while(1)
{
    fl=read(fd,ep,sizeof(ep));        // 此语句应置入循环体内，以不断捕获数据帧
    switch(eh->h_proto)
    ...
}
```

一旦 ep 读入帧中数据，随即可以通过 eh->h_dest、eh->h_source、eh->h_proto 获取帧头的目标地址、源地址和帧载荷的协议类型等信息。

四、思考题

另外一种编写 Linux 下网络监听的常用方法是采用 libpcap（其 Windows 版本为 Winpcap）函数库，该函数库为不同的平台提供了一致的编程接口。libpcap 共有 20 多个 C 函数。请查找相关资料，探讨如何使用 libpcap 函数库设计网络监听程序。

3.5 使用 Linux C 设计 IP 包协议分析程序

一、实验要求

在完成使用 Linux C 设计数据帧捕获程序的基础上，设计一个捕获 IP 包的程序，使得当源端主机向目标主机发出 ping 连接测试时，目标主机可以立即捕获这类承载 ICMP ECHO 请求报文的 IP 包，显示 IP 包中的协议类型、源和目标 IP 地址等字段。

二、实验条件

若干台微型计算机，连接在共享型或交换型以太网上。

三、实验指导

1. IP v4 报头格式

根据 IP 协议中的数据报格式，相应地，在 Linux C 头文件 ip.h 中结构类型 iphdr 已经定义了 IP 报头默认部分(20 个字节)的各个字段如下：

```
struct iphdr {
    __u8    version:4,           // 版本号
            ihl:4;             // IP 报头长度
    __u8    tos;               // 服务类型
    __u16   tot_len;           // 总长度
    __u16   id;                // 标识域
    __u16   frag_off;          // 分片控制与分片偏移量
    __u8    ttl;                // 生命周期
    __u8    protocol;           // 协议
    __u16   check;              // 校验和
    __u32   saddr;              // 源 IP 地址
    __u32   daddr;              // 目标 IP 地址
};
```

2. IPv4 报头各字段的获取

3.4 节的实验指导已经给出：当链路帧帧头的协议类型代码为 0x0800 时，帧中载荷是 IP 分组。因此，一旦捕获的帧中 h_proto 字段的取值为 0x800，将类型为 iphdr 的结构指针指向帧头后面负载数据的起始位置（偏离帧头首地址 14B），则 IP 包基本报头的数据结构将一览无余。以下程序段表明这一定位过程：

```
#include <netinet/ip.h>
char ep[ETH_FRAME_LEN];                                //ep 是捕获帧的缓冲区
struct ethhdr *eh;                                     //eh 指向帧头首地址
struct iphdr *ih;                                     //ih 指向 IP 报头首地址
int fl;                                                 //fl 用于存放捕获数据帧的帧长
eh=(struct ethhdr *)ep;
ih=(struct iphdr *)((unsigned long)ep + ETH_HLEN)     //ETH_HLEN 为帧头长(14)
fl=read(fd, ep, sizeof(ep));                          //此语句应置入循环体内，以不断捕获数据帧
```

一旦 ep 读入帧中数据，随即可以通过 ih->protocol、ih->saddr 和 ih->daddr 获取 IP 报头的协议类型、源 IP 地址和目标 IP 地址等字段信息。其中 ih->protocol 表示 IP 信包载荷的协议类型代码，常见的有：

- 1: ICMP 报文。
- 6: UDP 报文。
- 8: TCP 报文。

当源端主机向目标主机发出 ping 连接测试时，实际上是发出一个 ICMP ECHO 请求报文，目标主机可以立即捕获这类承载 ICMP 报文的 IP 包，此时 ih->protocol 的值应为 1。

四、思考题

在捕获承载 ICMP ECHO 请求报文的 IP 包基础上，进一步设计对这类 ICMP 报头进行协议分析的 Linux C 程序。

5.7 附录四

参见 “Programming with Libpcap - Sniffing the network from our own application” ,链接地址为 “<http://www.programming-pcap.albablocking.com/>” 。

6 CISCO IOS 路由器基本配置

6.1 实验目的

通过完成实验，理解网络层和路由的基本原理。掌握路由器配置网络和组网的方法；掌握 IP 协议、IP 地址配置和路由的概念；掌握 IP 协议和路由的基本原理；了解在模拟器下根据教程配置网络的方法。

6.2 实验要求

使用 Router eSIM v1.1 模拟器来模拟路由器的配置环境；使用 CCNA Network Visualizer 6.0 配置静态路由、动态路由和交换机端口的 VLAN（虚拟局域网）。

6.3 操作步骤

请按以下步骤完成：

- 1、下载相关课件，阅读实验教程课本（附录 1），上网搜集相关资料，主要是 FTP 上“/教学课件 / 黄炜 /Computer_Network_2019/experiments/E05_CISCOConfig/”的视频和文字教程，记住实验要领；
- 2、按照课本描述使用 Router eSIM v1.1 模拟器来模拟路由器的配置环境；使用 CCNA Network Visualizer 6.0 配置静态路由、动态路由和交换机端口的 VLAN（虚拟局域网）。

6.4 附录一

第5章 IOS 路由与 VLAN 配置

5.1 交换设备和 IOS 简介

一、目的要求

本节提供学习 Cisco IOS 的预备知识，了解路由器的基本结构及其在网络中所起的作用，使得对 IOS 配置环境有一个初步的认识。

二、实验要求

- (1) 复习 OSI 七层模型中物理层、数据链路层和网络层（有时简称一、二、三层）的有关协议，了解这些层所关联的网络设备。
- (2) 回顾二层 MAC 地址、三层 IP 地址的概念和作用。
- (3) 重温网络上用户数据传输时的封装过程。
- (4) 了解直通线、交叉线和反转线的接线方式。
- (5) 了解常用的网络拓扑通用图标（如图 5.1 所示），会看网络拓扑图。

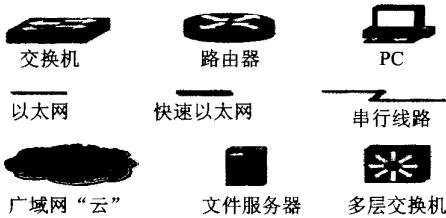


图 5.1 网络拓扑图中的部分通用图标

三、实验指导

1. 路由器的结构

路由器（Router）本质上是一台计算机，它包括中央处理器（CPU）、内存、接口、控制台端口和辅助端口等。

路由器的 CPU 负责执行处理数据包所需的工作，比如路由选择、协议转换、数据包过滤等。路由器处理数据包的速度在很大程度上取决于处理器的性能。

5.1 交换设备和 IOS 简介

87

路由器的内存一般包括 4 种，参见表 5.1：

表 5.1 路由器的 4 种内存

只读存储器 (ROM)	保存着路由器的自检和引导 (启动) 软件，负责让路由器进入正常工作状态
闪存 (flash Memory)	保存 IOS 镜像，维持路由器的正常工作
非易失性 RAM (NVRAM)	保存路由器的配置文件 startup-config
随机存储器 (RAM)	满足所有的常规存储需要，保存路由表、ARP 缓存等，运行配置 running-config 就是驻留在 RAM 里的。RAM 的内容会在路由器启动或供电间隙时丢失

接口是数据包进出路由器的门户，如图 5.2 所示。路由器支持的部分物理接口类型包括：以太网、快速以太网、令牌环、FDDI、低速/快速串行 (Serial Port)、HSSI 和 ISDN BRI 等。一个接口的全称由它的类型标识以及至少一个数字构成，编号自 0 开始。所有的路由器都配置了一个控制台端口 (Console Port)，控制台端口提供了一个 EIA/TIA-232 异步串行接口，采用 RJ-45 连接器，使网络管理员能与路由器通信。

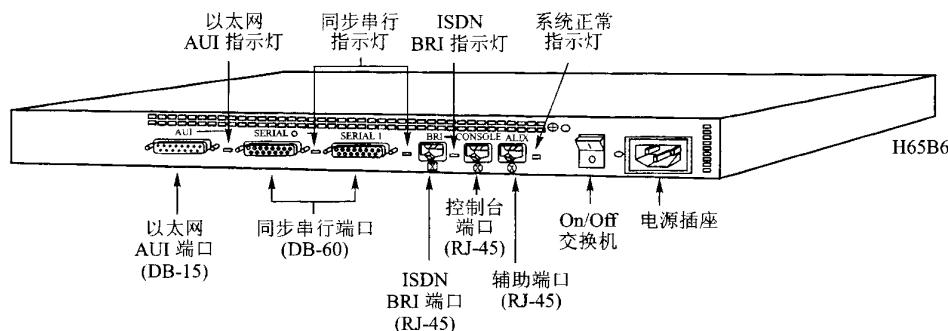


图 5.2 Cisco 2500 路由器的后面板

辅助端口 (Auxiliary Port, AUX) 通常用来连接 Modem，以实现对路由器的远程管理。

2. Cisco IOS 简介

Cisco 网间网操作系统 (Cisco Internet Work Operating System, IOS) 是一种特殊的软件。它为路由器和交换机提供了一个基础平台，用于配置路由器和交换机，令其将信息从一个网络路由交换至另一个网络。路由器和 IOS 的关系就好像计算机裸机和操作系统 (Windows、Linux 等) 一样，两者相辅相成，不可分割。

购买一个 Cisco 路由器时，必须购买运行 IOS 的一份许可证。Cisco 用一套编码方案来制定 IOS 的版本。IOS 的版本号由 3 部分组成，如 IOS 12.3 (10)，其中：12 是主版本号，3 是辅助版本号，(10) 表示维护版本，即第 10 次维护或补丁。

Cisco 用一套自己的命名方案标识其软件的可靠度：GD (General Deployment, 标准版)、

LD (Limited Deployment, 限制版) 以及 ED (Early Deployment, 早期版)。通常, 当一套 IOS 进入市场已有较长的时间, 使 Cisco 能够改正足够多的错误, 并认为其已经成熟, 可以令使用这套软件的大多数人满意, 就会为其冠以一个 GD 名称, 所以 GD 版是最可靠的。

Cisco 设备启动的顺序一般为: 首先对设备硬件进行自检, 接着查找并加载 (通常是加载到 flash 中) IOS 的镜像, 然后再将设备的配置信息 (start-up) 调出 (通常是加载到 NVRAM 中)。

3. 路由器的功能

路由器是工作在第 3 层 (网络层) 的设备, 它将各地区的各种不同类型的网络互连在一起, 形成现在所说的 Internet。路由器基于 3 层信息根据网络拓扑结构计算所有可达路径, 从中选取最优路径, 从而决定数据包下一跳的地址, 再将数据包送到相应的输出端口和网段, 即在源主机与目的主机之间跨越各种网络传递数据包, 如图 5.3 所示。

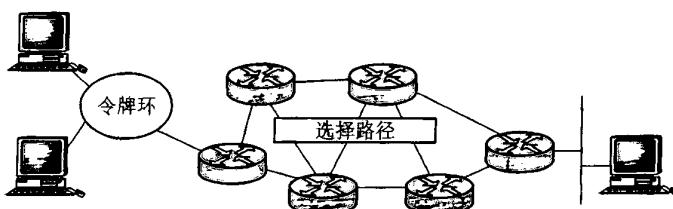


图 5.3 路由器决定路由路径

路由器划分广播域, 阻隔广播的继续传播; 还可以对多播 (Multicast) 进行控制管理。此外, 路由器还提供流量管理功能, 可以通过对访问列表的设置, 基于源地址、目标地址或端口对数据包进行过滤。

为了和其他网段的设备进行通信, 需要在计算机上配置网关。所谓网关其实就是路由器某个端口的 IP 地址。凡是不在本网段的数据包均先交给这个被称为网关的路由器的端口, 然后再由该路由器进行转发处理。

4. 路由器的配置方式

对路由器进行配置的方式有 5 种, 它们分别是 Console 端口配置、辅助端口配置 (利用 AUX 端口, 通过电话线远程配置)、Telnet 远程登录 (Virtual Terminals)、TFTP Server 和 Network Management Station, 如图 5.4 所示。但第一次配置只能选用 Console 端口这种方式。在完成对路由器的初始配置 (比如配好 IP 地址) 之后, 才可以支持其他 4 种配置手段。实际上, 网管人员最常用到的配置方式是远程登录到路由器上, 与路由器的一个终端 (VTY) 建立会话, 进而对该设备进行配置。

通过控制台端口对路由器进行配置, 需要一条控制线和一个 RJ-45 到 DB9 的转接头, 控制线的一端插在路由器的控制台端口上, 另一端插在转接头上, 然后把转接头与计算机的串口 (COM1、COM2 均可) 相连 (如图 5.5 所示), 继而可用“超级终端”方式进行配置。为此,

5.1 交换设备和 IOS 简介

89

按照以下步骤找到超级终端：单击“开始”→“程序”→“附件”→“通讯”→“超级终端”，出现如图 5.6 所示窗口，给本次连接起名字，单击“确定”按钮后，出现如图 5.7 所示对话框。如果控制线是连在串行端口 COM1，那么在图 5.7 下面的“Connect using”下拉列表框中选择“Direct to Com 1”，单击“OK”按钮后，在出现的图 5.8 所示的对话框中继续对 COM1 端口进行参数设置：常用的参数如图 5.8 所示。单击“OK”按钮后回车，如图 5.9 所示，这时候已经登录到路由器上了。

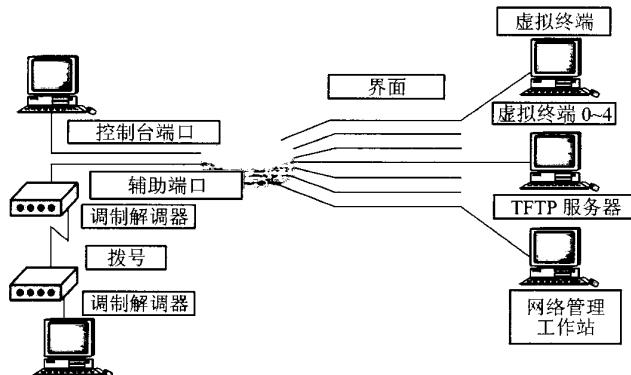


图 5.4 路由器的配置方式

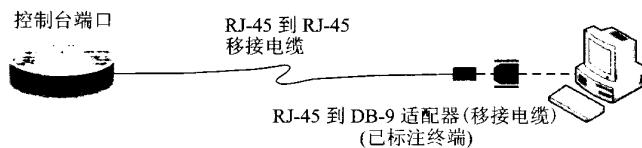


图 5.5 控口配置路由器

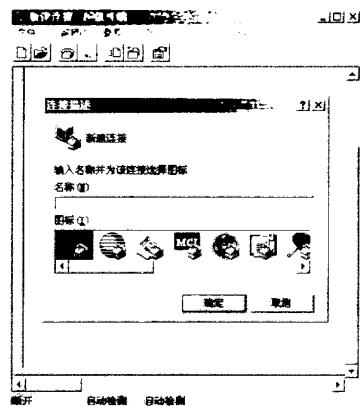


图 5.6 超级终端打开界面图

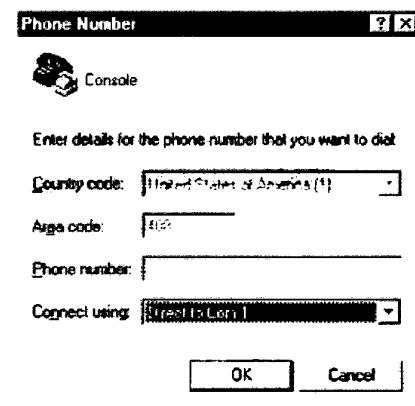


图 5.7 选择连接的串口

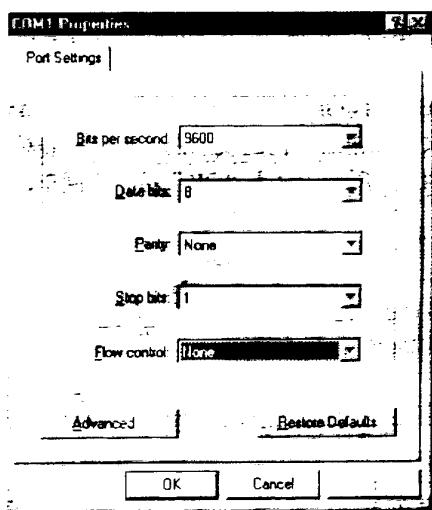


图 5.8 设置串口参数

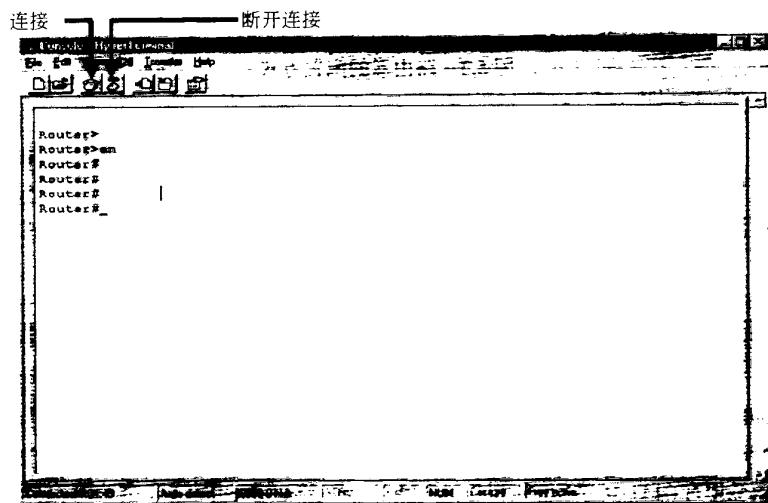


图 5.9 路由器的登录界面

5. 路由器、交换机和计算机之间的互连

以下互连使用交叉线 (Crossover): 计算机与计算机、路由器与路由器、交换机与交换机、集线器与集线器、交换机与集线器、交换机与第三层交换机 (第三层交换机就是带有路由功能的交换机, 包括路由模块和交换模块。因路由功能属第三层网络层功能故称第三层交换机)。

以下互连使用直通线 (Straight-Through): 计算机与集线器、计算机与交换机、交换机与路

由器。

以下互连使用反转线 (Rollover): 计算机串行接口连到路由器或交换机的 Console 端口对其进行配置。

6. 配置一台路由器之前要做的工作

在配置之前，应该先制订配置计划。如果可能，这个计划应以文档的形式确定下来，这样在实施过程中，既可随时加以参考，又可在出现问题时及时恢复原来的配置。

在启动路由器之前，如果是通过 Console 端口进行配置，请确认电源线和控制线连接正确。然后将路由器的开关打开，路由器进行自检，注意观察控制端机器上的启动显示。

5.2 Cisco IOS 的基本操作和路由器的常规配置

一、实验要求

通过本实验学习一些基本的 IOS 命令，了解 IOS 的使用环境。初步掌握路由器的一些常规配置，加深对路由器、网络层协议的理解。

二、实验条件

到网上搜索下载 Cisco 公司的“Router_eSIM_v1”模拟器来模拟路由器的配置环境。

三、实验指导

1. Cisco 网络实验室仿真软件的功能启用

首先，简要介绍 Cisco 公司开发的模拟路由器或交换机行为的软件 Router eSIM。这套软件可以在一台没有连接路由器的计算机上练习 Cisco IOS 命令，做一些路由器连接配置的实验，不过它不支持全部的 IOS 命令。Router_eSIM 模拟了按一定拓扑结构连接的 5 台路由器 A、B、C、D、E 的行为，用拓扑连接图来表示它所模拟的几个路由器的连接关系，还可测试所做的配置是否正确。

以下是对 Router eSIM 小实验模拟器的功能界面进行简单的使用介绍：

启动 Router eSIM 软件，进入启动界面（如图 5.10 所示），单击“*I Agree*”按钮进入实验界面（如图 5.11 所示）。

左窗格最左侧的“A”“B”“C”“D”“E”分别对应需要配置的 A、B、C、D、E 5 台路由器。单击相应的字母可以在这 5 台路由器间进行切换，右窗格就是对路由器的配置界面，在该窗格中输入命令行可以对相应路由器进行配置。单击左下角的“Show Topology”按钮可以显示 Router eSIM 模拟的网络拓扑图，如图 5.12 所示。单击“Show Done”可以显示出各个路由器都完成了哪些操作以及哪些操作还没有完成，如图 5.13 所示。操作完成会显示“Done”，操作没有完成或配置不正确就会显示“Not Done”。单击“Hide Done”按钮回到配置状态。实验要求

每个路由器都要配置端口的 IP 地址、DCE 端口的时钟速率，用 ip host 命令将主机名和 IP 地址映射起来，并配置简单的距离矢量路由协议（RIP）。当最后把 A、B、C、D、E 这 5 台路由器都配置好后，页面上就会显示提示信息“*This activity is completed*”，表示全部实验正确完成。

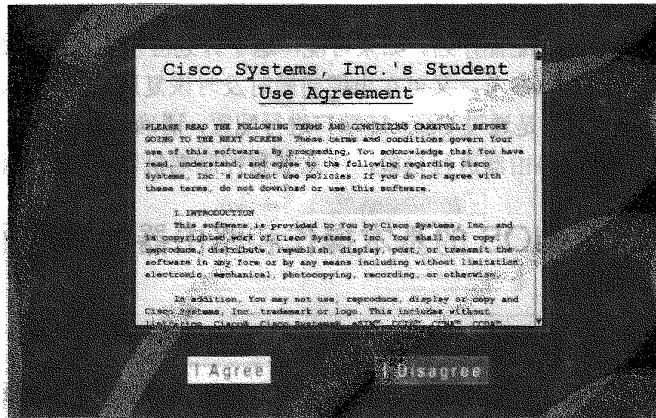


图 5.10 模拟软件 Router eSIM 启动界面

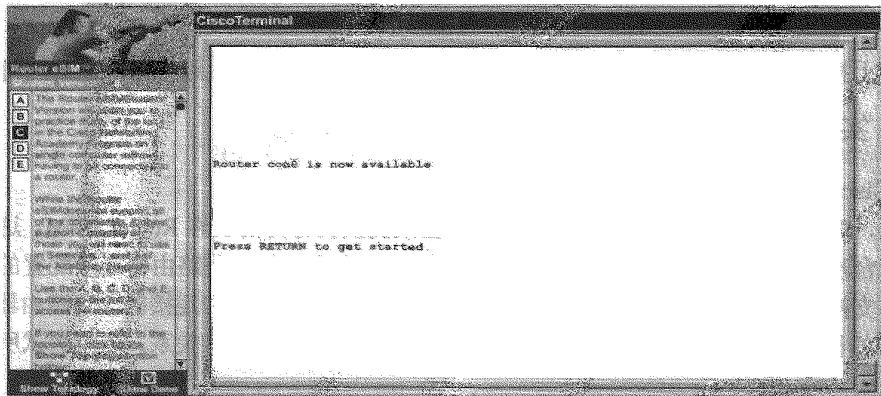


图 5.11 模拟软件 Router_eSIM_v1 实验界面

2. 路由器的配置模式切换

图 5.14 说明了 Cisco IOS 的几种配置模式，其中“Router>”中的 Router 是路由器的名字，可以通过 hostname 命令改变；有时路由器名字后紧跟着一个括号，括号内的字符串用来表明路由器的配置模式，如 Router (config) # 中的“(config)”表明路由器处在全局配置模式下；“(config-if)”表明路由器处于接口配置模式；“>”和“#”则表明路由器目前是处于普通用户模式下还是超级用户模式下；“Router (config-subif) #”表明了路由器 Router 目前处于子接口配置模式下。

每个路由器都要配置端口的 IP 地址、DCE 端口的时钟速率，用 ip host 命令将主机名和 IP 地址映射起来，并配置简单的距离矢量路由协议（RIP）。当最后把 A、B、C、D、E 这 5 台路由器都配置好后，页面上就会显示提示信息“*This activity is completed*”，表示全部实验正确完成。

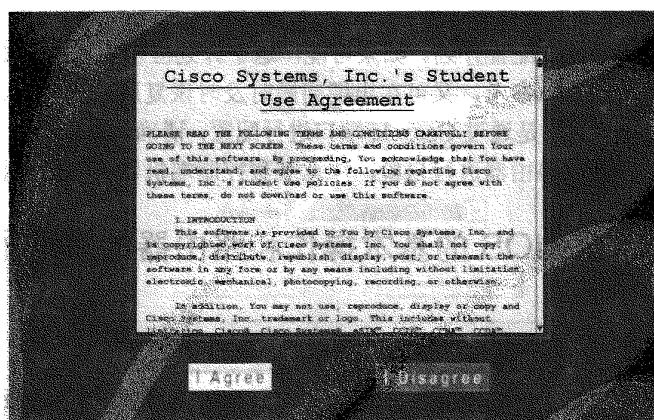


图 5.10 模拟软件 Router eSIM 启动界面

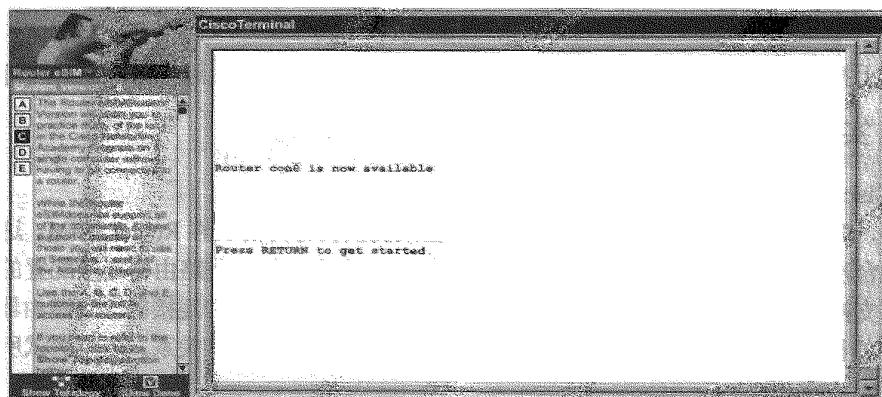


图 5.11 模拟软件 Router_eSIM_v1 实验界面

2. 路由器的配置模式切换

图 5.14 说明了 Cisco IOS 的几种配置模式，其中“Router>”中的 Router 是路由器的名字，可以通过 hostname 命令改变；有时路由器名字后紧跟着一个括号，括号内的字符串用来表明路由器的配置模式，如 Router (config) # 中的“(config)”表明路由器处在全局配置模式下；“(config-if)”表明路由器处于接口配置模式；“>”和“#”则表明路由器目前是处于普通用户模式下还是超级用户模式下；“Router (config-subif) #”表明了路由器 Router 目前处于子接口配置模式下。

① 在普通用户模式下（提示符为“>”），不能对路由器的配置做任何改动，只能对路由器的一些状态做有限的检查。在普通用户模式下可以使用的命令可以通过图 5.15 所示的方法获知。

```

Router>?
  access-profile  Apply user-profile to interface
  clear           Reset functions
  connect         Open a terminal connection
  disable         Turn off privileged commands
  disconnect      Disconnect an existing network connection
  enable          Turn on privileged commands
  exit            Exit from the EXEC
  help             Description of the interactive help system
  lock             Lock the terminal
  login            Log in as a particular user
  logout           Exit from the EXEC
  mrinfo          Request neighbor and version information from a multicast
                  router
  mstat            Show statistics after multiple multicast traceroutes
  mtrace           Trace reverse multicast path from destination to source
  name-connection Name an existing network connection
  pad              Open a X.29 PAD connection
  ping             Send echo messages
  ppp              Start IETF Point-to-Point Protocol (PPP)
  resume           Resume an active network connection
  rlogin            Open an rlogin connection
  set              Set system parameter (not config)
--More--

```

图 5.15 普通用户模式下可用命令

② 在普通用户模式下，输入“enable”，然后输入超级用户口令，就可以进入超级用户模式了，如图 5.16 所示。如果没有设置口令，则直接进入超级用户模式。在超级用户模式下（提示符为“#”），可以对路由器所有状态做全面的检查，也可以进入全局配置模式，对路由器进行事关全局的配置。

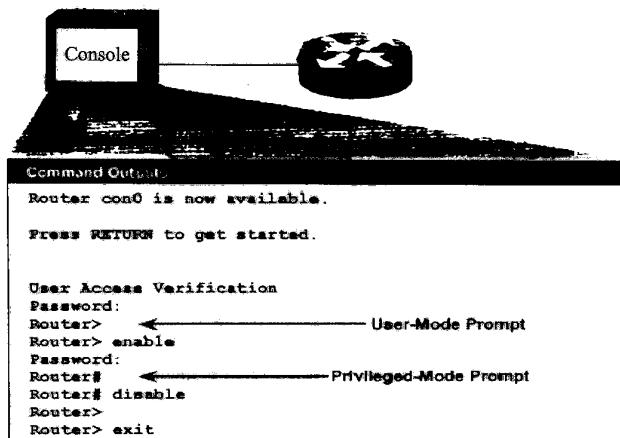


图 5.16 从普通用户进入超级用户

③ 在超级用户模式 Router#下输入“config terminal”，则进入全局配置模式 Router (config) #，比如给路由器改名字这样的对路由器整体起作用的配置必须要在全局模式下进行。如果是对路由器的各端口(如 Ethernet 端口)进行配置，则要在全局配置模式下输入相应命令(如 int eth0) 进入口配置模式 Router (config-if) #。

3. Cisco IOS 所提供的帮助功能

IOS 提供了命令记忆功能，可以用向上或向下方向键将用过的命令从缓存中调出来，重新使用。还可以自己设定命令缓存的大小，查看命令缓存的内容，具体命令可以参见图 5.17。

命令	说明
Ctrl-P 或向上箭头键	调出命令记忆缓存内当前命令的上一条命令
Ctrl-N 或向下箭头键	调出命令记忆缓存内当前命令的下一条命令
Router>show history	显示命令记忆缓存中的所有命令
Router>terminal history size lines	设置命令记忆缓存可能容纳内条命令

图 5.17 IOS 提供的命令记忆功能

Cisco IOS 也提供了基于上下文的帮助功能，除了上面提到的命令“？”之外，它还提供了诸如命令参数提示、错误提示以及输入命令语法检查等功能。以设置路由器的时钟为例（clock 并不为模拟软件 Router eSIM 支持，选用这条命令是因为它全面的显示了 IOS 的帮助功能），看看如何获得系统提供的其他帮助，如图 5.18 所示。另外对于一些命令，敲入前面部分，再按下“Tab”键，IOS 就可以完成整个命令的输入；或者干脆就不用输入完全，例如可以将 interface Ethernet 0/1 简化成 int eth 0/1。

4. 查看路由器的运行状态

了解路由器的状态、运行情况无疑是非常重要的，所以结合前面讲过的路由器的结构，介绍一下 IOS 提供的检查路由器各部分工作状态的命令 show。

① show running-config 显示当前运行状态的配置，如图 5.19 所示。如果不对 running-config 进行保存的话，那么随着路由器掉电，当前配置（running-config）也会丢失，所以要注意保存，保存命令为 copy running-config startup-config。

② show startup-config 显示保存在路由器 NVRAM 里的配置，如图 5.19 所示，路由器开机后会自动从 NVRAM 中将路由器的配置文件 startup-config 调出来。

③ show interfaces 命令显示了各接口的配置参数和工作数据。该命令对于差错检验和确定故障所在都很有帮助。也可以指定显示某个特定接口的参数，如 show int s0。

④ show version 显示当前运行在路由器上的 Cisco IOS 的版本号、路由器的型号。如图 5.20 所示，路由器所安装的 IOS 版本号 12.0，路由器的型号为 Cisco 2500。

Router# ctok

Translating "CLOK"
% Unknown command or computer name, or unable to find computer address

Router# cl?

clear clock

Router# clock

% Incomplete command.

Router# clock ?

set Set the time and date

Router# clock set

% Incomplete command.

Router# <CR>clock set? 7

hh:mm:ss Current Time

- 符号翻译(首先发现“clock”不是一条命令,然后试图将“clock”当作一台机器,查 ARP 表又找不到与之对应的 IP 地址,只好报错)
- 命令提示
- 调出上一条命令

Router# clk

Translating "clk"
% Unknown command

Router# clear

clear

Router# clock

% Incomplete command.

Router# clock ?

set Set the time and date

Router# clock set

% Incomplete command.

Router# clock set 19:56:00

MONTH Month of the year

Router# clock set 19:56:00 04 8

<1-31> Day of the month

Router# clock set 19:56:00 04 August

Year

Router# clock set 19:56:00 04 August ?

<1993-2035>

- 命令参数提示
- 命令语法检查
- 命令参数提示

图 5.18 IOS 提供的帮助功能

In RAM	In NVRAM
<pre>Router# show running-config Building configuration... [REDACTED] version 12.0 ! -- More --</pre>	<pre>Router# show startup-config [REDACTED] ! version 12.0 ! -- More --</pre>

图 5.19 查看路由器的配置文件

```

Router# show version
Cisco Internetwork Operating System Software
IOS (tm) 2500 Software (C2500-JS-L), Version 12.0(3), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-1999 by cisco Systems, Inc.
Compiled Mon 08-Feb-99 18:18 by phanguye
Image text-base: 0x03050C84, data-base: 0x00001000

ROM: System Bootstrap, Version 11.0(10c), SOFTWARE
BOOTFLASH: 3000 Bootstrap Software (IGS-BOOT-R), Version 11.0(10c), RELEASE SOFTWARE(fc1)

Router uptime is 20 minutes
System restarted by reload
System image file is "flash:c2500-is-l_120-3.bin"
(output omitted)
-- More --

```

图 5.20 查看 IOS 的版本信息

5.2 Cisco IOS 的基本操作和路由器的常规配置

97

这些常用 show 命令与路由器各组成部分的关系如图 5.21 所示。

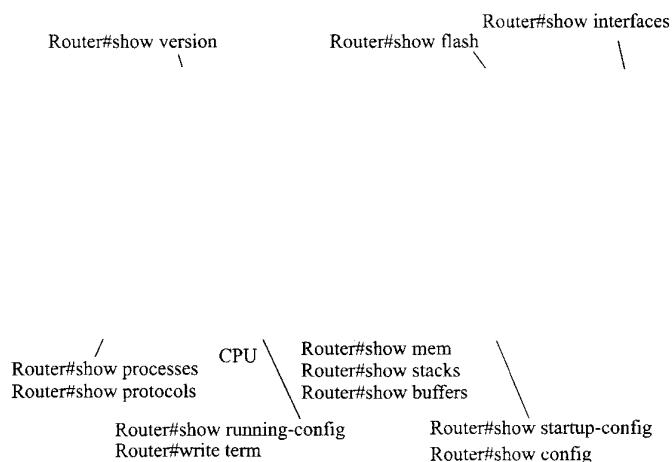


图 5.21 路由器运行状态查看命令

5. 路由器一些常规的配置

① 路由器出厂的名字都默认为 Router，为了区分网络中的各个路由器，要给路由器取名字，通常会将路由器的摆放地点表现到名字中。在全局配置模式下用 hostname 改变路由器的名字，IOS 会将这个名字显示在 IOS 的名字提示符中，如图 5.22 所示。

```

Router#
Router#config t
Enter configuration commands, one per line. End with END.
Router(config)#hostname lab_A
lab_A(config)#
  
```

图 5.22 改变路由器的名字

用于设置当日消息标题的命令 banner motd，是将#.....#之间的文本在各终端试图访问路由器时，在登录口令提示之前显示出来。可以用它来通知系统关闭等信息，如图 5.23 所示。

```

lab_A(config)#
lab_A(config)#banner motd #
Enter TEXT message. End with the character '#'.
Accounting Department
You have entered a secured system.
Authorized access only! #
lab_A(config)#
  
```

图 5.23 设置当日消息标题

Router# ctok

Translating "CLOK"
% Unknown command or computer name, or unable to find computer address

Router# cl?

clear clock

Router# clock

% Incomplete command.

Router# clock ?

set Set the time and date

Router# clock set

% Incomplete command.

Router# <CR>clock set? 7

hh:mm:ss Current Time

- 符号翻译(首先发现“clock”不是一条命令,然后试图将“clock”当作一台机器,查 ARP 表又找不到与之对应的 IP 地址,只好报错)
- 命令提示
- 调出上一条命令

Router# clk

Translating "clk"
% Unknown command

Router# clear

clear

Router# clock

% Incomplete command.

Router# clock set

set Set the time and date

Router# clock set 19:56:00

MONTH Month of the year

Router# clock set 19:56:00 04 8

<1-31> Day of the month

Router# clock set 19:56:00 04 August

Year Year

Router# clock set 19:56:00 04 August ?

<1993-2035> Year

- 命令参数提示
- 命令语法检查
- 命令参数提示

图 5.18 IOS 提供的帮助功能

In RAM	In NVRAM
<pre>Router# show running-config Building configuration... [REDACTED] version 12.0 ! -- More --</pre>	<pre>Router# show startup-config [REDACTED] ! version 12.0 ! -- More --</pre>

图 5.19 查看路由器的配置文件

```

Router# show version
Cisco Internetwork Operating System Software
IOS (tm) 2500 Software (C2500-JS-L), Version 12.0(3), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-1999 by cisco Systems, Inc.
Compiled Mon 08-Feb-99 18:18 by phanguye
Image text-base: 0x03050C84, data-base: 0x00001000

ROM: System Bootstrap, Version 11.0(10c), SOFTWARE
BOOTFLASH: 3000 Bootstrap Software (IGS-BOOT-R), Version 11.0(10c), RELEASE SOFTWARE(fc1)

Router uptime is 20 minutes
System restarted by reload
System image file is "flash:c2500-is-l_120-3.bin"
(output omitted)
-- More --

```

图 5.20 查看 IOS 的版本信息

5.2 Cisco IOS 的基本操作和路由器的常规配置

99

③ 可以在路由器内建立一个 IP 地址的映射表，静态指定机器名与 IP 地址的映射关系，这样可以通过机器名和 IP 地址两种方式指定计算机、交换机和路由器的接口。图 5.27 给出了建立映射表的方法和例子，在这个例子中在路由器 lab_A 上配置了名字解析的两个条目。如果在路由器 lab_A 上想和机器 lab_D 联系（lab_D 可能是计算机、交换机或路由器的接口），路由器将首先试图和 IP 地址为 210.93.105.1 的机器进行联系，如果联系不上，再与 IP 地址为 204.204.7.2 的机器联系，这样可以通过机器名和 IP 地址两种方式指定计算机、交换机和路由器的接口。

```
Router(config)#ip host name [tcp-port-number] address [address]
lab_A(config)#ip host lab_A 192.5.5.1 205.7.5.1 201.100.11.1
lab_A(config)#ip host lab_B 219.17.100.1 199.6.13.1 201.100.11.2
lab_A(config)#ip host lab_C 223.8.151.1 204.204.7.1 199.6.13.2
lab_A(config)#ip host lab_D 210.93.105.1 204.204.7.2
lab_A(config)#ip host lab_E 210.93.105.2
```

图 5.27 建立名字解析的映射表

④ 如果要对路由器的各个接口进行配置，必须首先进入路由器的接口配置模式。在全局配置模式 Router (config) #下，对于不带模块的路由器采用命令“interface type number”进入接口配置模式 Router (config-if)，而对于带模块的路由器则使用命令“interface type slot/port”。命令格式中的 type 可以是 serial、ethernet、fddi、hssi、loopback、atm、bri 等，根据要配置的路由器的接口而定。例如，在全局配置模式下，键入“int eth 0”则进入路由器接口 ethernet 0 的配置模式。一般来说，“exit”命令可以帮助回到上一级界面，例如可以使路由器从当前接口配置模式“Router (config-if) #”退回到全局配置模式“Router (config) #”下。

⑤ 为路由器的一个接口配置 IP 地址，在该接口上启动 IP 进程的方法，如图 5.28 所示。这个 IP 地址也是该接口所连接的子网的网关。

```
Router(config-if)#ip address {ip address} {subnet-mask}
lab_A(config)#int eth 0
lab_A(config-if)#ip address 192.5.5.1 255.255.255.0
lab_A(config-if)#int eth 1
lab_A(config-if)#ip address 205.7.5.1 255.255.255.0
lab_A(config-if)#int serial 0
lab_A(config-if)#ip address 201.100.11.1 255.255.255.0
```

图 5.28 给路由器接口配置 IP 地址

⑥ 在串行端口连接中，作为 DCE 的一端必须为连接的另一端 DTE 提供时钟信号。可以通过命令“show controller serial X”可查看串行端口是 DCE 还是 DTE（Cisco 公司的模拟器 Router eSIM 不支持这条命令，但在拓扑图中标注出了串行端口端是 DCE 端还是 DTE 端。后面介绍的 RouterSim 公司的 RouterSim Network Visualizer 支持这条命令），如图 5.29 所示。默认情

100

第5章 IOS 路由与 VLAN 配置

况下，Cisco 的路由器串行端口充当 DTE 设备，如果要配置成 DCE 端，必须用 `clock rate` 指定时钟频率，也只有 DCE 端口，才需要配置 `clock rate`，如图 5.30 所示。

```
Router#show controller serial 0
HD unit 0, idb = 0x121C04, driver structure at 0x127078
buffer size 1524 HD unit 0 V.35 DTE cable
.....
```

图 5.29 查看串行端口的类型

```
Router#config t ...
Enter configuration commands, one per line. End with END.
Router(config)#interface serial 0
Router(config-if)#clock rate 56000
```

图 5.30 配置充当 DCE 端的串行端口

可以用 `show` 命令来查看串口的配置情况，如图 5.31 所示。

```
lab_A#show interface serial 0
Serial0 is administratively down, line protocol is down
  Internet address is 201.100.11.1/24
  Hardware is HD64570
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
    reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation HDLC, loopback not set
  Keepalive set (10 sec)
  Last input never, output never, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0 (size/max/drops); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
    Conversations 0/0/256 (active/max active/max total)
    Reserved Conversations 0/0 (allocated/max allocated)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
    0 packets input, 0 bytes, 0 no buffer
    Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
    0 packets output, 0 bytes, 0 underruns
    0 output errors, 0 collisions, 1 interface resets
    0 output buffer failures, 0 output buffers swapped out
--More--
```

图 5.31 显示串口的配置情况和一些工作数据

输入命令之后回显的第一行显示了网络的工作状况：

`serial is up, line protocol is up`——接口工作正常。

`serial is up, line protocol is down`——数据链路出了问题，原因可能是串口没有设置时钟频率或封装了错误的协议等。

`serial is down, line protocol is down`——物理链路出了问题（检查线路、接口）。

serial is administratively, clown line protocol is down: ——管理员手动关闭该端口。

⑦ 当路由器接口被管理员手工关闭或路由器刚出厂时（各接口默认为管理员关闭状态），需要手工开启端口，此时只需要在该接口模式下键入 no shutdown 即可；管理员若想关闭接口，首先要进入到该接口的配置模式下，直接输入 shutdown 即可。图 5.32 给出的是在真正路由器键入“shut down”和“no shut down”后并且路由协议配置完成的情况下路由器的反应，在模拟软件输入这两条命令之后路由器是没有回显的。

```

Router#configure term
Router(config)#interface serial 0
Router(config-if)#no shutdown
%LINK-3-UPDOWN: Interface Serial0, changed state to up
%LINEPROTO-5-UPDOWN: Line Protocol on Interface Serial0, changed state to up

Router#configure term
Router(config)#interface serial 0
Router(config-if)#shutdown
%LINK-5-CHANGED: Interface Serial0, changed state to administratively down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0, changed state to down

```

图 5.32 手工开启和关闭端口

最后，通过模拟器左下角的“Show done”可以检查各个路由器的配置状态。操作完成会显示“Done”，操作没有完成或配置不正确就会显示“Not Done”。做到如图 5.31 所示就算完成本次实验任务，没有完成的命令将在 5.4 节介绍。

四、思考题

怎样对路由器设置密码保护？利用 show 命令如何判定网络故障？路由器有几种配置模式？如何在各个配置模式之间进行切换？如何判定路由器的串口是广域网的 DTE 端还是 DCE 端？

5.3 静态路由配置

一、实验要求

进一步了解路由器的工作过程，了解静态路由的优缺点及适用范围。掌握配置静态路由实现网络互通的方法，并能够通过查看路由表了解数据包在路由器中的转发过程。

二、实验条件

RouterSim 公司的模拟器 CCNA Network Visualizer 5.0。

三、实验指导

路由就是将数据包从一个网络的计算机跨越中间网络发送到另一个网络的目标机上。路由器的作用是根据网络拓扑结构和通信状况转发数据包，使其沿着一条最短最快的通路到达目的端。路由选择是第三层设备路由器最重要的功能，为了实现这个功能，路由器至少应该知道：目标地址、相邻的路由器（通过它可以获知不相邻网络的拓扑信息）、到达目标网络的所有可用的路径、到目标网络的最佳路径以及如何检验更新路由信息。

路由器可以通过网管人员，也可以由相邻的路由器来收集不相邻网络的拓扑信息，然后路由器会建立路由表来存放到达各网络的路由选择结果，并用它来指导数据包的路由转发。在转发数据包时，路由器会首先查看路由表，如果路由表中没有该数据包目标网络的转发声明，路由器不是像交换机那样发布广播来询问，而是直接丢弃这个数据包。

路由器将连接到它各接口的网络地址直接放入路由表中，所以它自己知道如何“到达”这些直连网络；而对于如何“到达”所谓“遥远”的目的网络，即非直连网络，路由器就要通过学习才能获知，如图 5.33 所示。

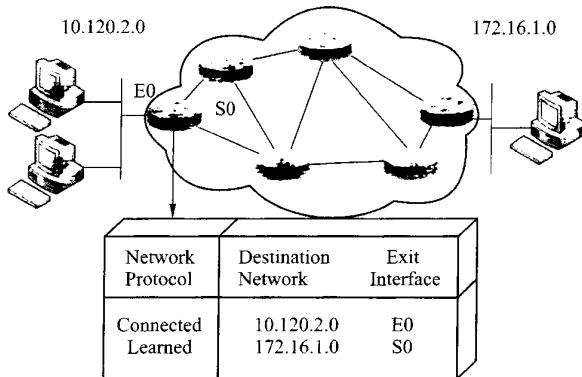


图 5.33 路由条目的获取

路由器可以通过静态路由和动态路由两种方式来获知怎样将数据包送达至非直连的目的地。所谓静态路由就是由网管人员定义的路由，是以人工方式将路由条目添加到路由表中，以指导数据包向目的端的转发。而动态路由则是路由器利用动态路由协议与相邻路由器进行路由通信，通过这个过程来获知到达目的网络的路径。配置了动态路由协议的各个路由器会将自己所知道的网络信息比如网络可达信息、拓扑变化信息等通知给彼此。

静态路由的优点在于它不会占用路由器 CPU 的资源，也不会占用路由器之间的带宽（启用动态路由协议的路由器之间需要定期进行路由更新、路由查询等路由通信，难免要消耗一些带宽）。最后就是安全，因为数据可以路由到哪个网络是由管理员自己指定的。

静态路由通常只用于网络路由相对简单、网络与网络之间只通过一条路径互连的情况。为了实现网络之间的互通，源端和目的端的路由器都要配置静态路由。静态路由不能对线路不通等路由变化做出反应，需要手工更新路由表。比如，如果有一个子网新加到网络中，为了令该子网对其他网络是可达的，管理员必须在网络中所有的路由器（除了与该子网直接相连的路由器）中加入相关的路由信息。另外，配置静态路由要求管理员对网络及各个路由器的连接情况比较了解，错误的配置会导致网络之间的连接中断。

1. RouterSim CCNA Network Visualizer 简介

RouterSim CCNA Network Visualizer 是 RouterSim 公司开发的模拟软件，功能要比上一节介绍的 Cisco 公司的模拟软件 Router eSIM 要强大，支持的命令更多，还可以根据需要自己设计网络。在安装 RouterSim CCNA Network Visualizer 之前，最好卸载已经安装的任何版本的 Network Visualizer。

现在要使用模拟器来模拟图 5.34 所示的网络拓扑。启动 RouterSim CCNA Network Visualizer，出现如图 5.35 所示的界面，单击按钮“Net Visualizer Screen”，出现如图 5.36 所示的设计界面。从设计界面工具栏上可以看到模拟器模拟的 Cisco 设备，包括一台 Cisco 2600 路由器和三台 Cisco 的交换机（型号分别是：1900、2950 和 3550）。鼠标单击工具栏中的设备快捷键，在设计界面的工具区就会出现该设备，再用鼠标拖动该设备到指定位置，以此类推，将拓扑图上的交换设备放在相应的位置上，如图 5.37 所示。当将鼠标指向设备图标时，会出现如图 5.38 所示的操作提示，如想连接设备可单击鼠标右键。

如果双击该设备图标则启动路由器的配置界面。因为现在需要连接各个交换设备，所以用鼠标右键单击 RouterA，出现如图 5.37 所示的 Cisco 2600 路由器的各个端口，选择要连接的端口。只需单击该端口就会有一条线从该端口引出。

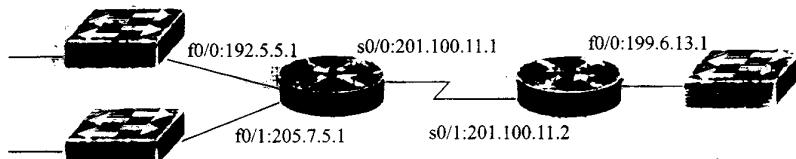


图 5.34 实验用网络拓扑图



图 5.35 RouterSim CCNA Network Visualizer 的启动界面

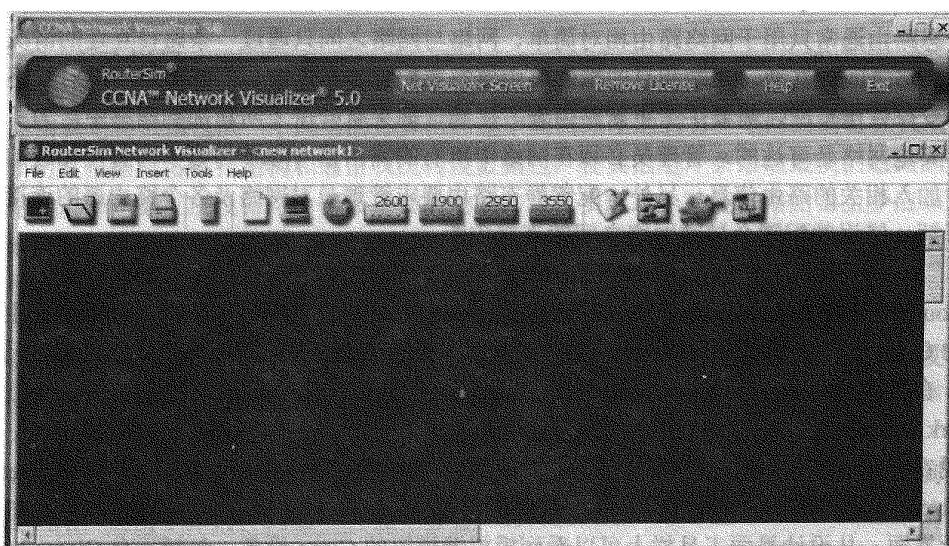


图 5.36 RouterSim CCNA Network Visualizer 的设计界面

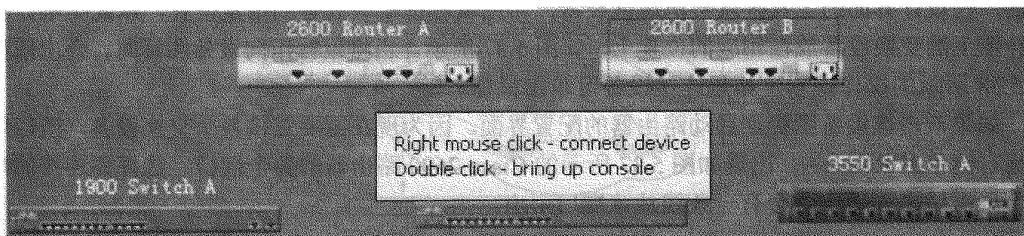


图 5.37 在模拟器中表达拓扑途中交换设备的布局

如果选择的是串行端口，比如 Serial0/0 (S0/0) ，会弹出对话框以便选择该串行端口是充当 DCE 端还是 DTE 端，如图 5.38 所示。



图 5.38 串行端口选择充当 DCE 端还是 DTE 端

然后右键单击另一端的交换设备，会出现该设备的接口界面，鼠标单击要连接的接口，则完成两个设备的连接。以此类推，将实验设备在模拟器的设计界面上按拓扑图连接完成，如图 5.39 所示。接下去要做的就是对各个交换设备进行配置。双击要配置的交换设备，配置界面就会被调出来。



图 5.39 模拟器上实验设备的连接图

2. 静态路由配置之前的工作

在配置静态路由之前，要配置路由器各个端口的 IP 地址，还要用命令 no shutdown 激活端口。串口如果充当 DCE 端，还需要配置时钟频率，如图 5.40 所示。

在准备工作做完之后，如果查看路由表 (show ip route)，会看到路由器直连网络的情况，如图 5.41 所示。在路由表中可以看到路由器直连了 3 个网络，在每个路由条目前都有一个字母 C，它所代表的含义可以通过路由条目前的 codes 得知。由“Code: C-connected,S-static,I...”，可知 C 是 connected 的第一个字母，代表直连。另外在路由条目中目标网络后都有一个“/24”，这是网络的子网掩码，即 255.255.255.0（二进制表示中正好是连续 24 个 1：11111111.11111111.11111111.0）。路由表中还将连接网络的路由器端口标识出来，如连接网络 201.100.11.0 的就是路由器的串行端口 Serial0。

```

RouterA#config t
Enter configuration commands, one per line. End with CNTL/Z
RouterA(config)#int f0/0
RouterA(config-if)#ip address 192.5.5.1 255.255.255.0
RouterA(config-if)#no shutdown
20:39:55 %LINK-3-UPDOWN: Interface Fastethernet0/0, changed state to up
20:39:55 %LINEPROTO-5-UPDOWN: Line protocol on Interface Fastethernet0/0, changed state to up

RouterA(config-if)#int f0/1
RouterA(config-if)#ip addr 205.7.5.1 255.255.255.0
RouterA(config-if)#no shutdown
20:40:16 %LINK-3-UPDOWN: Interface Fastethernet0/1, changed state to up
20:40:16 %LINEPROTO-5-UPDOWN: Line protocol on Interface Fastethernet0/1, changed state to up

RouterA(config-if)#int s0/0
RouterA(config-if)#ip addr 201.100.11.1 255.255.255.0
RouterA(config-if)#clock rate 56000
RouterA(config-if)#no shutdown
20:40:46 %LINK-3-UPDOWN: Interface Serial0/0, changed state to up
20:40:46 %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to up

RouterA(config-if)#exit
RouterA(config)#exit
RouterA#

```

图 5.40 静态路由配置之前的准备工作

```

RouterA#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
      D - EIGRP, EX - EIGRP external, O- OSPF, IA - OSPF inter area
      N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
      i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
      U - per-user static route, o - ODR, P - periodic downloaded static route
      T - traffic engineered route

Gateway of last resort is not set
C       201.100.11.0/24 is directly connected, Serial0/0
C       205.7.5.0/24 is directly connected, FastEthernet0/1
C       192.5.5.0/24 is directly connected, FastEthernet0/0

```

图 5.41 查看 RouterA 的路由表

同样的方法将 RouterB 各个端口的 IP 地址配好，用命令 no shutdown 激活。注意 RouterB 的串行接口充当 DTE 端，所以不需要配置时钟频率。配置完成后查看 RouterB 的路由表如图 5.42 所示。

```

RouterB#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
      D - EIGRP, EX - EIGRP external, O- OSPF, IA - OSPF inter area
      N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
      i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
      U - per-user static route, o - ODR, P - periodic downloaded static route
      T - traffic engineered route

Gateway of last resort is not set
C       201.100.11.0/24 is directly connected, Serial0/1
C       199.6.13.0/24 is directly connected, FastEthernet0/0

```

图 5.42 查看 RouterB 路由表

在 RouterA 上，通过 ping 命令测试到路由器 RouterB 的直连网络地址 199.6.13.1 是否连通，如图 5.43 所示。

```

RouterA#ping 199.6.13.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 199.6.13.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5), round-trip min/avg/max = 0/0/0 ms

```

图 5.43 查看是否连通

“Success rate is 0 percent” 表明 ping 不通，需要配置静态路由或动态路由协议，下面介绍静态路由的配制方法。

3. 配置静态路由

配置静态路由的命令格式为：

5.3 静态路由配置

107

```
Router (config) #ip route [destination_network] [mask] [next_hop_address or exitinterface]
[administrative_distance] [permanent]
```

在命令格式中，“destination_network”是所要到达的目的网络，“mask”为目的网络的子网掩码。而“next_hop_address”是指下一跳的IP地址。所谓下一跳是指数据包向目的地址前进的下一个路由器的端口，当然必须保证这个端口的IP地址可以ping得通。有时候在“next_hop_address”这个位置上用“exitinterface”，就是数据包离开路由器的接口，但这种配置方式只可以用于端到端的连接，比如说广域网，在以太网中就不可以使用这种配置方式。“administrative_distance”为管理距离（可选），静态路由默认的管理距离是1，可以通过这个参数修改这个权值。“permanent”指定此路由即使该端口关掉也不被移掉。如图5.44所示，路由器A如果打算向199.6.13.0网络转发数据包（199.6.13.0非路由器的直连网络），它必须配置一条关于199.6.13.0网络的路由条目。这样配置以后，再查看路由表，可以看见到达199.6.13.0的指导信息，如图5.44所示。

```
RouterA(config)#ip route 199.6.13.0 255.255.255.0 201.100.11.2
RouterA(config)#exit
RouterA#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
      D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
      N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
      L1 - IS-IS, L2 - IS-IS level-1, L3 - IS-IS level-2, * - candidate default
      U - per-user static route, o - ODR, P - periodic downloaded static route
      T - traffic engineered route

Gateway of last resort is not set
C   192.5.5.0/24 is directly connected, FastEthernet0/0
C   205.7.5.0/24 is directly connected, FastEthernet0/1
C   201.100.11.0/24 is directly connected, Serial0/0
S   199.6.13.0 [1/1] via 201.100.11.2
```

图5.44 配置查看路由表（静态路由）

从第一个路由条目中的第一个字母S，可知这条路由信息是通过静态配置得到的（S是Static的首字母），其次是到达目的网络199.6.13.0的下一跳的地址是201.100.11.2。注意在子网掩码“/24”后有一个[1/1]，其中分子位置的数字1表示静态路由的管理距离为1，关于管理距离的作用将在下一节动态路由中作详细讲解；分母位置的数字1则表示从该台路由器到达目标网络需要经过1跳，即1台路由器。

在图5.44中，在路由条目的前面都出现“Gateway of last resort is not set”，就是说如果目的子网没有在路由条目中出现，数据包就会被路由器丢掉，而不是发送到一个默认的端口。下面开始配置所谓的默认路由（Default Routing）。

4. 配置默认路由

当某个网络连接到“单一网络”（Stub Network）时最常使用的路由就是默认路由。“单一网络”是指一个网络只能由一个路径进出，而默认路由是指将目标网络不在路由表中的数据包

都将全部发送到出口路由的下一跳路由器上。

如图 5.45 所示的拓扑图中, 172.16.1.0 网络只能通过路由器 B 的串口 172.16.2.1 出入网络, 所以它是一个“stub network”, 可以使用网络地址和子网掩码的通配符 (即 0.0.0.0) 来配置默认路由。

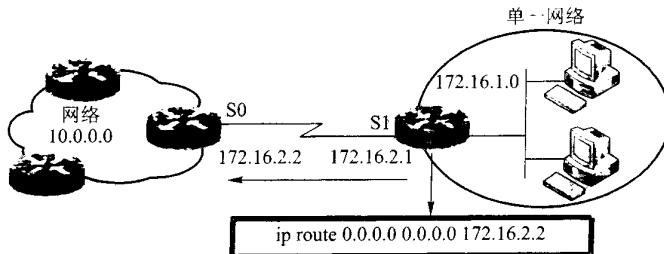


图 5.45 配置默认路由

查看路由表, 可以发现路由表中除了出现两个直连网络外, 还增加了一条 S*条目。这个 S* 表明该路由条目是默认路由条目, 如图 5.46 所示。其中路由表中的路由条目之前有一行这样的字符串 “Gateway of last resort is 172.16.2.2 to network 0.0.0.0”, 这是表明如果数据包的目标网络不在路由表中, 则将该数据包发往 172.16.2.2。S* 标识的默认路由条目中 0.0.0.0/0 是网络地址及子网掩码的通配符, 表示任意网络。

```

RouterA>sh ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
      D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
      N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
      E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
      i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
      U - per-user static route, o - ODR
      T - traffic engineered route

Gateway of last resort is 172.16.2.2 to network 0.0.0.0
172.16.0.0/24 is subnetted, 5 subnet

C    172.16.2.0      is directly connected, Serial0
C    172.16.1.0      is directly connected, Ethernet0
S*   0.0.0.0/0 [1/0] via 172.16.2.2

```

图 5.46 查看路由表 (默认路由)

配置默认网络还要注意有一条命令要特别配置, 就是在全局配置模式下输入 “ip classless”, 即 “Router (config) #ip classless”。“ip classless” 即 “无类型”, 这条命令允许用可变长掩码分配子网, 意思是现在的路由决策是基于整个 32 位 IP 地址的掩码操作, 而不管其 IP 地址是 A 类、B 类还是 C 类。关于有/无类别路由, 本次实验不做进一步探讨。总之, 如果不用命令 “ip classless”, 默认路由有时可以工作, 有时不能正常工作。所以, 在使用默认路由时, 要用 “ip

“classless”这条命令将无类别路由打开。

5. 检验连通性

当所有路由器的路由表都配置好后，需要对连通性进行检查，通常是使用 ping 命令。如果用图 5.34 所示的网络拓扑，为了测试路由器 A 与网络 199.6.13.0 的连通性（假设连接网络 199.6.13.0 的路由器的接口为 199.6.13.1），则可以在路由器 A 上（普通用户模式或超级用户模式下均可）运行 ping 199.6.13.1。

如果出现的是如图 5.47 所示内容，则说明网络连通性良好，路由配置正确；反之，如果路由配置有问题或其他原因导致目的网络不可达，就会出现如图 5.48 所示的内容。这时可以从离该路由器最近的路由器端口向目标网络方向逐跳 ping 过去，先确定故障位置，再想办法解决。

```
Router#ping 199.6.13.1
Type escape sequence to abort
Sending 5, 100-byte ICMP Echos to 199.6.13.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent(5/5), round-trip min/avg/max=4/4/4 ms
```

图 5.47 测试连通性（连通性良好）

```
Router#ping 199.6.13.1
Type escape sequence to abort
Sending 5, 100-byte ICMP Echos to 199.6.13.1, timeout is 2 seconds:
.....
Success rate is 0 percent(0/5)
```

图 5.48 测试连通性（网络不可到达）

四、思考题

如何实现如图 5.49 所示的网络 10.0.0.0 与 172.16.1.0 的互通，给出你的配置方案，其中路由器各端口的 IP 地址可以自己设定（在设计配置方案时，可将路由器 A、B、C 的常规配置也考虑进去，比如说路由器的口令、名字、各端口描述，等等）。

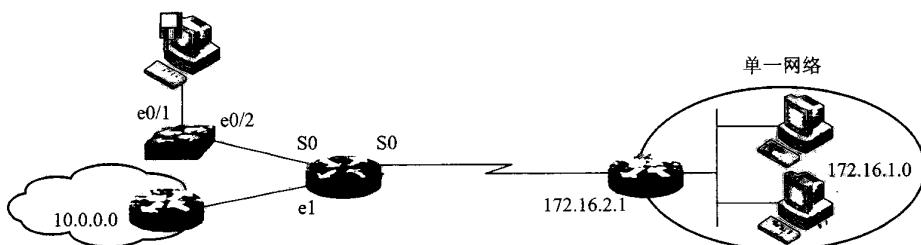


图 5.49 思考题的网络拓扑

5.4 动态路由协议 RIP 的配置

一、实验要求

给出比较简单的网络拓扑结构，能够选择并配置合适的动态路由协议，实现网络互连。掌握路由信息协议（Routing Information Protocol, RIP）协议的配置方法。

二、实验条件

模拟路由器功能的任意一个版本的 Cisco IOS 模拟软件（大多数模拟软件支持路由信息协议 RIP 的配置）。

三、实验指导

路由是把报文从一个网络转发到另一个网络的过程。在小规模的网络互连的情况下，可以采取手工静态建立路由表 Internet 的方法，人为地指定每一个可达目的网络的路由。在一个由成千上万个网络组成的互联网中，这种方式显然是不可行的。目前大多数的路由器都能够配置动态路由协议，通过与相邻的路由器交换网络信息而动态建立路由表。

路由信息协议（RIP）定义了路由器间相互交换网络信息的规范。路由器之间通过路由信息协议，相互交换网络的可达性信息，然后，每个路由器据此计算出到达各个目的网络的路由。路由信息协议能够用以下度量标准的几种或全部来决定到目的网络的最优路径：路径长度、可靠程度、延迟、带宽、负载和通信代价。

1. 内部及外部网关协议

首先，图 5.50 形象地给出了关于动态协议的几个概念。路由协议可以分为内部和外部两种路由协议。对于网络内部路由协议来说，最重要的是如何找到报文穿越网络到达目的地的最佳路径，像 RIP、IGRP 都属于内部网关协议（Internal Gateway Protocol, IGP）；而外部网关协议则主要用于在不同管理域的网络之间交换路由信息，比如说边界网关协议（Border Gateway Protocol, BGP）就是外部网关协议（Exterior Gateway Protocol, EGP）的一种。

2. 路由协议的管理距离

下面介绍一个重要参数——管理距离（Administrative Distance）。在 5.3 节，静态路由的管理距离是 1，动态路由协议也都有自己的管理距离。管理距离是衡量路由信息可信任程度的一个参数，管理距离越低，表明该协议提供的路由信息越可靠。如图 5.51 所示，路由器 A 上同时配置了 RIP 和 IGRP 两种动态网关协议。对于从路由器 A 发往目标网络 E 的数据包的转发路径，两种协议分别给路由器 A 两条路径：一条是 IGRP 协议建议从路由器 B 这条路径走；另一种是 RIP 建议取道路由器 C。路由器 A 就要根据管理距离来决定取舍。IGRP 的管理距离是 100，小

5.4 动态路由协议 RIP 的配置

111

于 RIP 协议的管理距离 120，所以路由器 A 就将采纳 IGRP 协议建议的路径。

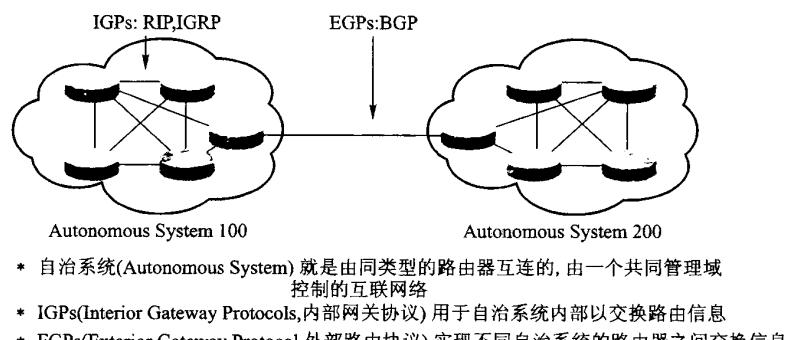


图 5.50 自治系统、内部路由和外部路由

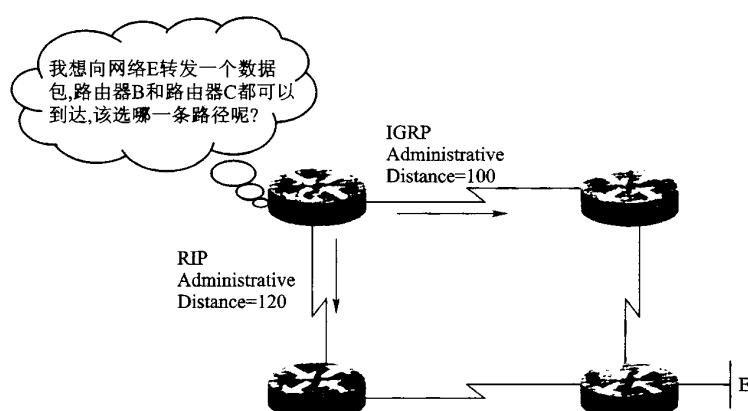


图 5.51 管理距离的作用

3. RIP 的特点

RIP 协议具有几个特点：首先 RIP 协议属于距离向量（Distance Vector）路由协议；其次，RIP 是根据跳数来做路由选择的。图 5.52 中的各路由器均配置了动态路由协议 RIP，两台计算机互通，既可以走 19.2 Kbps 这条路径，也可以走 T1 这条线路。T1 带宽可以达到 1.544 Mbps，所以尽管走 T1 这条路径过的跳数比较多，速度还是更快。但 RIP 协议却会选择 19.2 Kbps 这条路径。因为 RIP 协议做路由选择时只考虑跳数，走 19.2 Kbps 只需要一跳，而走 T1 这条路径则需要经过 3 跳。而且，RIP 协议最多可允许 15 跳，即 RIP 协议的作用半径在 15 个路由器之内。默认的情况下，RIP 协议每隔 30 秒进行一次路由更新，可以在 6 条代价相等的路由选择路径上

做负载均衡。最后一点就是，配置 RIP 起作用的网络时，网络地址必须使用 A 类、B 类或 C 类网络号（这种网络地址又称为可分类的网络）。在这里只讨论 RIPv1，事实上 RIPv2 允许无类别子网。

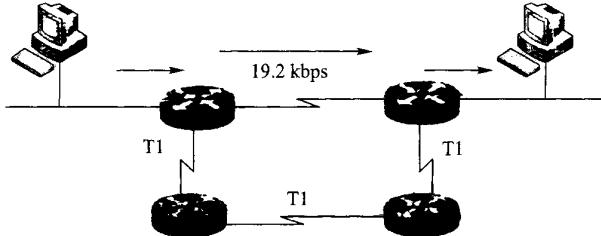


图 5.52 RIP 协议根据跳数做路由选择

4. RIP 的配置

配置 RIP 的两条命令分别是：

- Router (config) #router rip：用于启动 RIP 协议。
- Router (config-router) #network network-number：选择 RIP 协议起作用的网络（必须是路由器直连的可分类网络）。

RIP 协议的具体配置如图 5.53 所示，对于路由器 A，连接网络 172.16.1.0 的接口 E0 和连接网络 10.0.0.0 的接口 S2 对 RIP 路由更新信息进行发送和接收，这些更新信息使路由器 A 了解这个网络的拓扑结构（这个实验可以用 RouterSim CCNA Network Visualizer 来完成）。

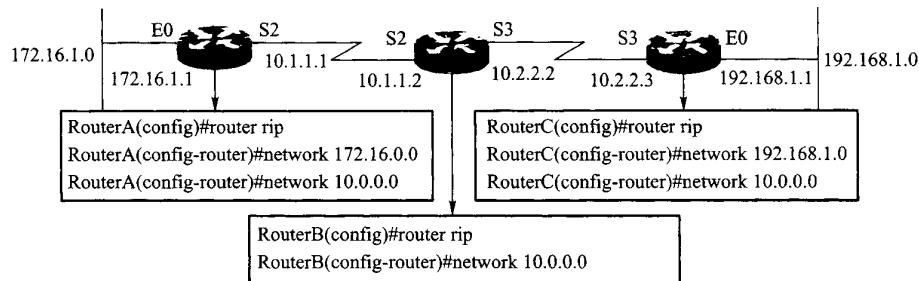


图 5.53 RIP 协议的配置方法

用“show ip protocols”可以显示路由协议 RIP 的工作情况，如图 5.54 所示。这条命令可以显示路由器定时器的信息及和路由器相连的一些网络信息，可以用这些信息来确定被怀疑发布错误路由信息的路由器。从图 5.54 的阴影部分可以看到路由器每隔 30 秒发布一次路由更新信息，同时可知路由协议 RIP 的管理距离是 120。

命令“show ip route”用来检查路由器 A 的路由表，路由表中包含了关于直连网络和非直连网络的网络信息条目。在 5.3 节的路由表中的路由条目可见，字母 C (connected) 表示直连网络，字母 S(static) 表示静态路由，以此类推。RIP 的第一个字母 R 用来表示通过 RIP 协议“学到”的网络。如果路由表中没有显示出通过 RIP 协议获知的路由条目，可以在超级用户模式下用命令“show running-config”或“show ip protocols”检查在配置路由协议的过程中是否配错。图 5.54 显示的路由表的输出中，阴影部分的条目就是通过 RIP 协议获得的。

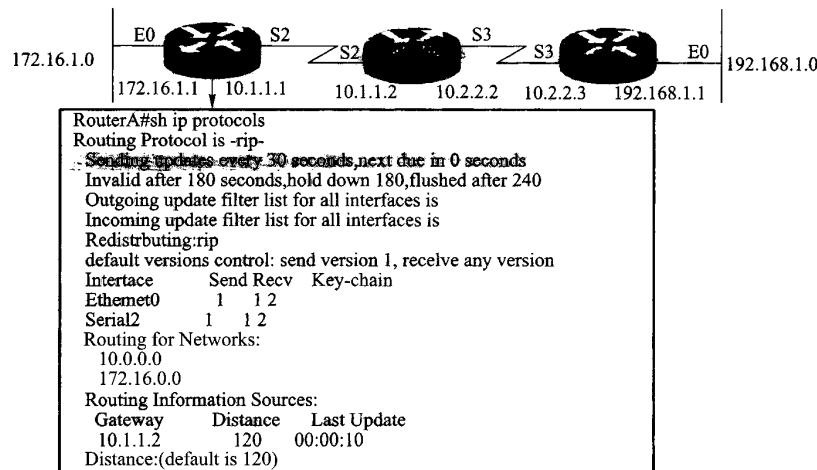


图 5.54 查看 RIP 协议的路由信息

例如，“R 10.2.2.0 [120/1] via 10.1.1.2,00:00:07,Serial2”，这里 R 表明了该条目是通过 RIP 协议学到的，到达目标网络 10.2.2.0 的数据包会从路由器的 Serial2 端口被转发到 IP 地址为 10.1.1.2 的下一跳路由器的端口上。[120/1]表明 RIP 协议的管理距离是 120，而到达目标网络 10.2.2.0 要经过 1 跳（即经过一个路由器）。

至此为止，也可以完成 5.2 节中未完成的配置命令了。

5.5 Cisco 路由器访问列表配置

一、实验要求

掌握标准访问控制列表和扩展访问列表的配置。

二、实验条件

RouterSim 公司的模拟器 CCNA Network Visualizer 5.0。

三、实验指导

访问列表（Access List）是一个有序的语句集，通过匹配数据包中信息与访问列表参数，并应用于路由器接口来允许通过或拒绝数据包通过此接口，从而增加了在路由器接口上过滤数据包出入的灵活性，帮助网络管理员限制网络流量。同时也控制用户和设备对网络的使用。访问列表根据网络中每个数据包所包含的信息内容决定是否允许该信息包通过接口，以期达到拒绝不期望访问的管理目的。

本节实验将配置两种访问列表，即标准访问列表和扩展访问列表。标准访问列表检查可被路由的数据包的源地址，根据源网络、子网、主机 IP 地址来决定对数据包的拒绝或允许，其序列号范围是 1~99；而扩展访问列表则能够检查可被路由的数据包的源地址和目的地址，同时还可以检查指定的协议、端口号和其他参数，配置更加灵活、控制也更加精确，其序列号的范围是 100~199。

1. 配置访问列表的步骤

首先要创建访问列表，无论是标准还是扩展的访问列表，命令格式均为：access-list access-list-number {deny|permit} {test conditions}

其中“access-list-number”为列表序列号或命名，建议使用命名来配置访问列表，这样在以后的修改中也方便。“test conditions”为过滤条件语句。然后开始应用访问列表，需要在接口模式下，应用“ip access-group access-list-number {in|out}”命令。

2. 配置访问列表之前对实验环境的配置

本次实验的拓扑图如图 5.55 所示，在模拟器上实验设备的连接图如图 5.56 所示，对于模拟器上计算机的 IP 地址的配置见图 5.57（以主机 A 为例）。在配置访问列表之前，两台路由器要进行基本的配置，分别如图 5.58 和图 5.59 所示。如果此时在主机 B 上 ping 同一个交换机上的主机 D，可以看到它们是连通的；在主机 B 上 ping 子网为 205.7.5.0 上的主机 A，可见它们也是连通的。此时虽然没有配置路由，但作为直连网络，205.7.5.0 这个条目已经出现在路由器 B 中的路由表中；在主机 B 上 ping 子网 199.6.13.0 上的主机 C，因为路由没有配置，所以是不通的。动态协议 RIP 的配置如图 5.60 和 5.61 所示。

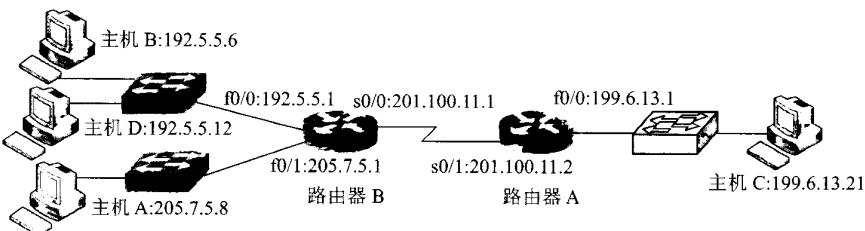


图 5.55 实验拓扑图

5.5 Cisco 路由器访问列表配置

115

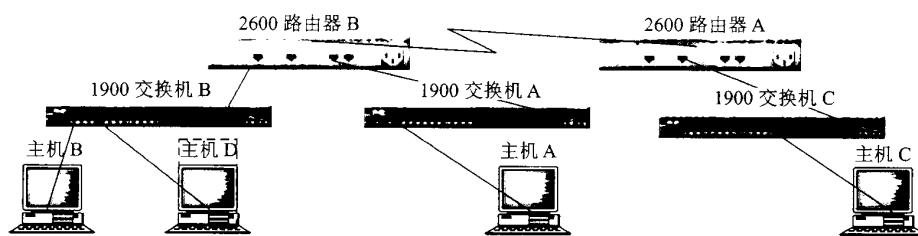


图 5.56 模拟器上的实验设备连接图

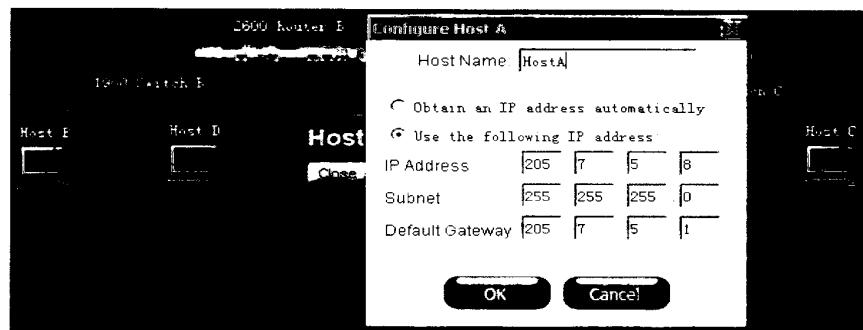


图 5.57 模拟器上计算机 IP 地址的配置

```

Router>en
Router#config t
Enter configuration commands, one per line. End with CNTL/Z
Router(config)#hostname RouterB
RouterB(config)#line console 0
RouterB(config-line)#password koalaB
RouterB(config-line)#login
RouterB(config-line)#exit
RouterB(config)#line vty 0 4
RouterB(config-line)#password tigerB
RouterB(config-line)#exit
RouterB(config)#enable secret ciscoB
RouterB(config)#int f0/0
RouterB(config-if)#ip addr 192.5.5.1 255.255.255.0
RouterB(config-if)#no shutdown
17:46:01 %LINK-3-UPDOWN: Interface Fastethernet0/0, changed state to up
17:46:01 %LINEPROTO-5-UPDOWN: Line protocol on Interface Fastethernet0/0, changed state to up

RouterB(config-if)#int f0/1
RouterB(config-if)#ip addr 205.7.5.1 255.255.255.0
RouterB(config-if)#no shutdown
17:46:44 %LINK-3-UPDOWN: Interface Fastethernet0/1, changed state to up
17:46:44 %LINEPROTO-5-UPDOWN: Line protocol on Interface Fastethernet0/1, changed state to up

RouterB(config-if)#int s0/0
RouterB(config-if)#ip addr 201.100.11.1 255.255.255.0
RouterB(config-if)#clock rate 56000
RouterB(config-if)#no shutdown
17:48:39 %LINK-3-UPDOWN: Interface Serial0/0, changed state to up
17:48:39 %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to up

```

图 5.58 路由器 RouterB 的基本配置

```

Router>en
Router#config t
Enter configuration commands, one per line. End with CNTL/Z
Router(config)#hostname RouterA
RouterA(config)#line console 0
RouterA(config-line)#password koalaA
RouterA(config-line)#login
RouterA(config-line)#exit
RouterA(config)#line vty 0 4
RouterA(config-line)#password tigerA
RouterA(config-line)#exit
RouterA(config)#enable secret ciscoA
RouterA(config)#int f0/0
RouterA(config-if)#ip addr 199.6.13.1 255.255.255.0
RouterA(config-if)#no shutdown
17:53:00 %LINK-3-UPDOWN: Interface Fastethernet0/0, changed state to up
17:53:00 %LINEPROTO-5-UPDOWN: Line protocol on Interface Fastethernet0/0, changed state to up

RouterA(config-if)#int s0/1
RouterA(config-if)#ip addr 201.100.11.2 255.255.255.0
RouterA(config-if)#no shutdown
17:53:32 %LINK-3-UPDOWN: Interface Serial0/1, changed state to up
17:53:32 %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1, changed state to up

```

图 5.59 路由器 RouterA 的基本配置

```

RouterA#config t
Enter configuration commands, one per line. End with CNTL/Z
RouterA(config)#router rip
RouterA(config-router)#network 201.100.11.0
RouterA(config-router)#network 199.6.13.0
RouterA(config-router)#exit
RouterA(config)#

```

图 5.60 路由器 RouterA 上路由协议 RIP 的配置

```

RouterB(config)#router rip
RouterB(config-router)#network 192.5.5.0
RouterB(config-router)#network 205.7.5.0
RouterB(config-router)#network 201.100.11.0
RouterB(config-router)#exit
RouterB(config)#

```

图 5.61 路由器 RouterB 上路由协议 RIP 的配置

3. 配置标准访问列表

建立标准访问控制列表的格式为：

access-list [list number] [permit|deny] [source address] [wildcard mask]

其中 [list number] 为列表序列号 (1~99) 或者命名, [source address] 为源 IP 地址, [wildcard mask] 为掩码, 如果不使用掩码, 则使用关键字 Host, 例: host 192.168.2.4。

① 利用标准访问列表限制主机 HostB 对 205.7.5.0 网络的访问。

访问列表在之前, 主机 HostB 是可以访问子网 205.7.5.0 上的主机的, 如图 5.62 所示。现在在路由器 RouterB 上设置访问列表, 如图 5.63 所示, 即首先在路由器 RouterB 上建立访问列表, 列表号 50, 然后把它应用到端口 f0/1 上(因为端口 f0/1 连接子网 205.7.5.0)。然后主机 HostB 与子网 205.7.5.0 的互通被阻隔了。但主机 HostB 对其他网络的连接访问仍是可以进行的, 如图

5.64 所示。

```
C:\>ping 205.7.5.8

Pinging 205.7.5.8 with 32 bytes of data:

Reply from 205.7.5.8 :bytes=32 time=23ms TTL=254

Ping Statistics for 205.7.5.8:
    Packets Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 23ms, Maximum = 23ms, Average = 23ms

C:\>
```

图 5.62 限制前 HostB 可以访问子网 205.7.5.0

```
RouterB#config t
Enter configuration commands, one per line. End with CNTL/Z
RouterB(config)#access-list 50 deny host 192.5.5.6
RouterB(config)#access-list 50 permit any
RouterB(config)#int f0/1
RouterB(config-if)#ip access-group 50 out
RouterB(config-if)#exit
RouterB(config)#
```

图 5.63 对主机的访问列表控制

```
C:\>ping 205.7.5.8

Pinging 205.7.5.8 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping Statistics for 205.7.5.8:
    Packets Sent = 4, Received = 0, Lost = 4 (100% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

图 5.64 限制后主机 HostB 已不能访问子网 205.7.5.0

② 利用访问列表限制子网对子网 199.6.13.0 的访问。

利用 IP 地址和子网掩码可以划分小的无类别子网。例如，IP 地址 192.5.5.8 和子网掩码 255.255.255.248 表示不分类子网的网络号为 192.5.5.8，所包含的主机为 192.5.5.9~192.5.5.14，192.5.5.15 是该子网的广播地址。现在对这个子网进行访问列表控制，如图 5.65 所示。访问列表配置完毕后，在模拟器上对该子网上的主机 HostD 用 ping 命令进行测试，可知该子网上的主机已经不能对子网 199.6.13.0 上的主机 HostC 进行访问了，但是不在该子网范围内的主机比如

HostB 对子网 199.6.13.0 子网的访问不受影响（同样可以用 ping 命令来验证）。

```
RouterB#config t
Enter configuration commands, one per line. End with CNTL/Z
RouterB(config)#access-list 51 deny 192.5.5.8 255.255.255.248
RouterB(config)#access-list 51 permit any
RouterB(config)#int s0/0
RouterB(config-if)#ip access-group 51 out
RouterB(config-if)#exit
RouterB(config)#_
```

图 5.65 对子网进行访问列表控制

③ 在 RouterA 上建立访问列表，使主机 HostA 不能远程登录 RouterA。

首先从主机 HostA 远程登录（telnet）到 RouterA，结果如图 5.66 所示。

```
C:\>telnet 201.100.11.2
Connecting To 201.100.11.2 ...

User Access Verification

Password:
RouterA>en
Password:
RouterA#exit

Connection to host lost.
```

图 5.66 在主机 HostA 上远程登录 RouterA

现在在 RouterA 上配置访问列表，拒绝主机 HostA 而允许其他主机访问，然后把这个访问列表应用到 VTY 虚拟终端线路上，如图 5.67 所示。配置完成后，验证如图 5.68 所示。

```
RouterA(config)#access-list 50 deny host 205.7.5.8
RouterA(config)#access-list 50 permit any
RouterA(config)#line vty 0 4
RouterA(config-line)#access-class 50 in
RouterA(config-line)#exit
RouterA(config)#
```

图 5.67 配置 RouterA，使 HostA 不能 telnet 到 RouterA 上

```
C:\>telnet 201.100.11.2
Connecting To 201.100.11.2 ... Could not open a connection to host: Connect failed
```

图 5.68 配置完成后验证结果

从上面的结果中可以看到，HostA 试图远程登录到路由器 RouterA 上的请求被拒绝了。通

过命令“show access-lists”可以查看路由器上已配置好的访问列表。

5.6 基于交换机端口的 VLAN 配置

一、实验要求

了解虚拟局域网（VLAN）的含义以及划分 VLAN 的目的。能够独立地在交换机上进行基于端口的 VLAN 配置，并且验证自己的配置是否正确。

交换机的一些常规配置命令（如设置名字、设置登录口令、配置模式之间的切换等）与路由器基本上是类似的。复习 5.2 节的实验指导，重温这些命令。

二、实验条件

RouterSim 公司的模拟器 CCNA Network Visualizer 5.0。

三、实验指导

虚拟局域网（virtual local-area network, VLAN）主要是在交换机上提供广播域分段化的一种技术。它允许一组不限物理地域的用户群共享一个独立的广播域，通过有效地划分用户群、控制广播范围等方式，VLAN 技术能够最大限度地控制广播域的影响范围以及减少由于共享介质所形成的安全隐患。

在规划上利用 VLAN 提供的网络分段性可以使一个部门的员工虽然分布在不同的楼层，但仍同处一个 VLAN，也就是共享一个逻辑上的广播域，从而有效地提高网络的灵活性、安全性和整体性能。传统的局域网（local-area network, LAN）通常是基于用户的物理位置划分子网，不同的子网连接在路由器的不同端口上（昂贵的选择）。一个子网内的各计算机处于同一个广播域中，而路由器阻止广播向其他子网继续传播。另外，由于不必用路由器的端口来划分子网和分割广播域，所以配置 VLAN 还有一个优点就是节省路由器的端口，图 5.69 使用了一个第三层交换机（带有路由功能的交换机）代替路由器连接各子网。

处于一个 VLAN 中的各个端口（即使它们处于不同的交换机）能够应用地址学习等机制互相转发数据包，工作起来就好像它们是一个独立的交换机。两个端口即使同属于一台交换机，如果它们分属于不同的 VLAN，那么它们之间进行通信也必须要经过路由层面。

为了让 VLAN 能跨越多台交换机，必须用 Trunk 将两个交换机连接起来。如图 5.70 所示，交换机 SwitchA 和 SwitchB 都包含 Red VLAN 的端口，而每一个逻辑上的 VLAN 就像一个独立的交换机那样工作，所以 SwitchA 上的属于 Red VLAN 端口会向 SwitchB 上的同属于 Red VLAN 的端口转发数据包。同样的情况在 Black VLAN 中也会发生，换言之，不同 VLAN 的数据包都会通过 SwitchA 和 SwitchB 之间的链路。这样就产生了两个要求：一个是对这种跨越交换机的数据包必须要封装一个特殊的标签，以声明它属于哪一个 VLAN；另一个要求是这条通路

必须可以承载不同 VLAN 的数据帧，这样的通路称为 Trunk。通路两端的交换机端口要配置成 Trunk 口。

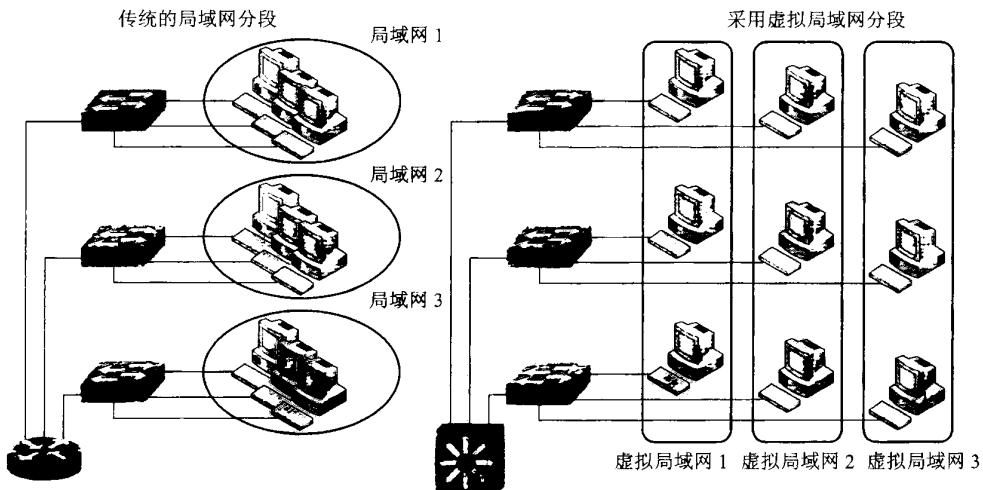


图 5.69 传统 VLAN 与 VLAN 的比较

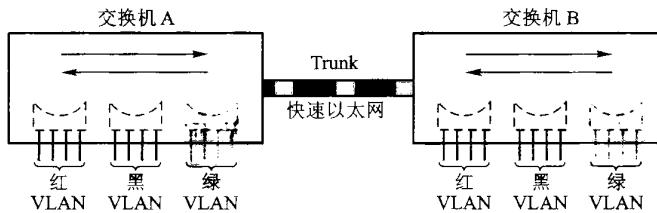


图 5.70 用 trunk 可以让 VLAN 跨越多个交换机

1. VLAN 成员的分配模式

指定交换机的某一个端口属于哪一个 VLAN 可以有两种方式：静态指定和动态分配。静态指定就是由网管人员手工配置交换机的端口，指定它属于哪一个 VLAN；而动态分配则是通过 VLAN 成员分配服务器(VLAN Membership Policy Server, VMPS)来动态分配 VLAN。在 VMPS 内建立一个数据库，将计算机网卡的 MAC 地址映射到某一个 VLAN 中。当数据帧抵达交换机的端口时，为了决定该端口所属的 VLAN，交换机会向 VMPS 查询这个数据帧中源 MAC 地址属于哪一个 VLAN，最终做出转发决定（根据所设定的时间参数，在之后的一段时间内，该端口属于这个 VLAN）。

2. 什么是 VTP (VLAN Trunk Protocol)

VTP 是第二层(数据链路层)协议，用来在一个交换网络中发布和同步关于添加、删除和

5.6 基于交换机端口的 VLAN 配置

121

改变 VLAN 的配置信息，以维护 VLAN 配置的一致性。VTP 能够减少一些配置上的错误，像一个局域网内的两个 VLAN 起了相同的名字，错误地指定了 VLAN 的类型参数等。

VTP 的管理域是指几个互连的交换机共享相同的 VTP 环境，一台交换机只能属于一个 VTP 的管理域，如图 5.71 所示。默认情况下，在交换机没有收到关于域的通告之前或没有配置管理域之前，Cisco 交换机是处在没有管理域的状态下，此时 VLAN 信息是不会在网络上发布的。

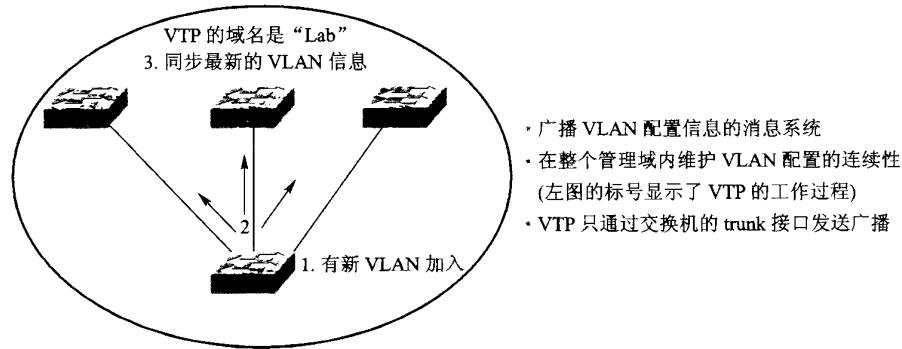


图 5.71 VTP 的管理域

3. VTP 的工作模式

VTP 有 3 种工作模式，分别是服务器模式、终端模式和透明模式，VTP 的默认模式是服务器模式。交换机处在不同的模式下，它们可以执行的操作是不同的，如图 5.72 所示。

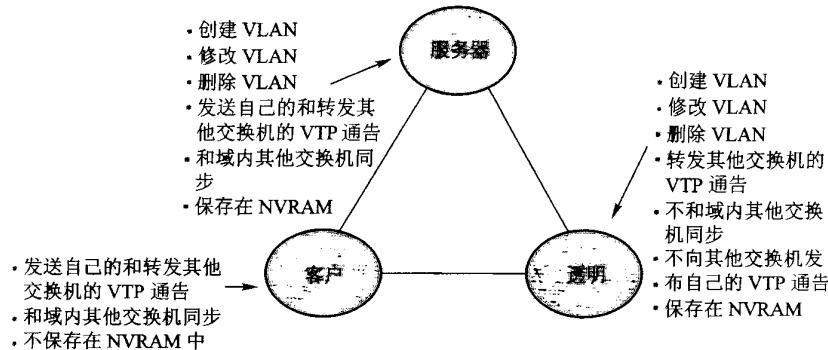


图 5.72 VTP 的工作模式

当一台交换机要加入一个已经开始运作的局域网时，一般是先将它的 VTP 配置成透明模式。因为在这种模式下，交换机不向网络发布自己的 VTP 通告，所以不会把这台交换机以前配置的一些 VLAN 信息和错误的 VTP 版本号发布到网络上，引起混乱。在透明模式下，可以创

建、修改和更改 VLAN 信息，在这些配置完成后，再将它的 VTP 模式改到客户模式。

4. VTP 工作过程

VTP 通告要“扩散”(Flooding)到整个 VTP 管理域中，它一般是通过出厂时默认的 VLAN 和多播帧来发布通告的。在 VTP 告示内有一个配置参数——版本号。版本号越高，暗示其 VLAN 的信息越新。收到 VTP 告示的交换机要检查几个参数，首先是管理域的名字和口令必须和本地配置的相符；其次是版本号是不是比本地的更高，只有更高才对本地的进行更新。每当服务器模式的 VTP 修改它的 VLAN 时，它都会在原有的版本号上加 1，然后再在 VTP 域内发布这个带有新版本号的 VTP 告示，具体过程用图 5.73 来表示。

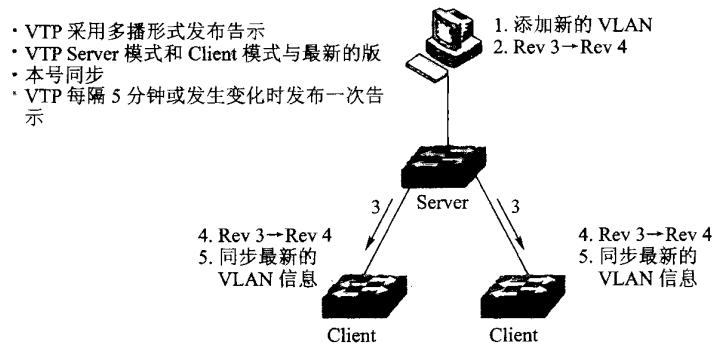


图 5.73 VTP 的工作过程

5. VTP 的修剪

启动 VTP 修剪模式可以通过减少不必要的“扩散”流量来增加可用的带宽，可以用图 5.74 中的例子来说明这个修剪过程。

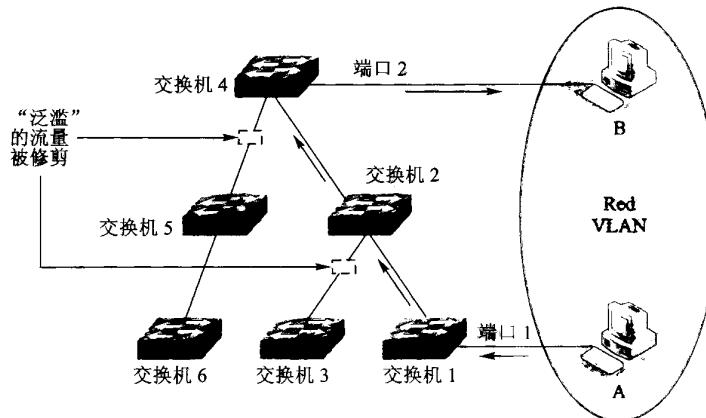


图 5.74 VTP 的修剪

计算机 A 发送了一个广播，广播经 Switch 2 扩散到 Switch 4，却没有被转发到 Switch 3 和 Switch 5。因为计算机 A 所连的端口 1 (Port 1) 是属于 Red VLAN 的，而无论是 Switch 3、Switch 5 还是 Switch 6 均没有属于 Red VLAN 的端口了，所以扩散到 Switch 3 和 Switch 5 的广播流量就被修剪了。因为 Switch 4 的端口 2 属于 Red VLAN，所以广播扩散到了 Switch 4。

6. VLAN 的配置

一般情况下，交换机有一个默认的 VLAN——VLAN1，VLAN1 是默认的管理域，简称域。VTP 告示是通过 VLAN1 发布的，交换机的 IP 地址也在 VLAN1 的广播域内（第二层设备交换机配置第三层的 IP 地址就是为了便于远程管理）。如果要创建 VLAN，交换机必须在 VTP Server 模式或 VTP Transparent 模式下，VLAN 的配置主要有 4 步：配置 VTP、启用 Trunk 端口、创建 VLAN 和指定 VLAN 到交换机的端口。

下面将给出两个实例来说明 VLAN 的具体配置。

实例 1，如何在一个典型的快速以太局域网中实现 VLAN。

所谓典型局域网就是指由一台具备第三层交换功能的核心交换机（Cisco 3550）接两台分支交换机（Cisco 2950），如图 5.75 所示。

需要做的主要工作：

- 设置 VTP 域（核心、分支交换机上都设置）。
- 配置中继（核心、分支交换机上都设置）。
- 创建 VLAN（在服务器上设置）。
- 将交换机端口划入 VLAN。
- 配置第三层交换机。

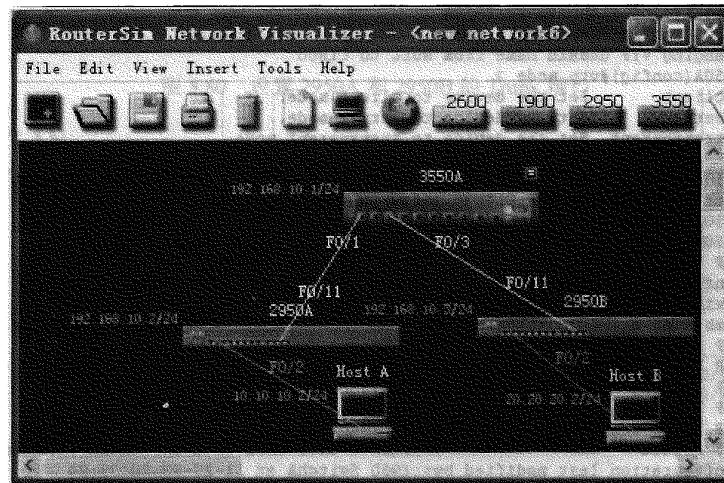


图 5.75 典型的快速以太局域网

(1) 设置 VTP 域。

在 Cisco 3550 交换机上将 VTP 管理域名称设置为“Cisco”，并使用命令“show vtp status”检查 VTP 配置，如图 5.76 所示。

```

switch>en
switch#conf t
Enter configuration commands, one per line. End with CNTL/Z
switch(config)#hostname 3550A
3550A(config)#vtp domain Cisco
Changing VTP domain name from null to Cisco
3550A(config)#exit
3550A#sh vtp status
VTP Version : 2
Configuration Revision : 1
Maximum VLANs supported locally : 64
Number of existing VLANs : 5
VTP Operating Mode : Server
VTP Domain Name : Cisco
VTP Pruning Mode : Disabled
VTP V2 Mode : Disabled
VTP Traps Generation : Disabled
MD5 digest : 0x70 0x01 0xF2 0x72 0x97 0xA1 0x35 0xEB
Configuration last modified by: 0.0.0.0 at 11-29-93 20:39:24
Local updater ID is 0.0.0.0 on interface V11 (lowest numbered VLAN interface
found)

```

图 5.76 配置 3550A 的 VTP

在交换机 Cisco 2950 将 VTP 管理域名称设置为“Cisco”，并设置为客户模式，如图 5.77 和图 5.78 所示。

```

switch>en
switch#conf t
Enter configuration commands, one per line. End with CNTL/Z
switch(config)#hostname 2950A
2950A(config)#vtp domain Cisco
Changing VTP domain name from NULL to Cisco
2950A(config)#vtp mode ?
    client      Set the device to client mode.
    server      Set the device to server mode.
    transparent Set the device to transparent mode.

2950A(config)#vtp mode client
Setting device to VTP CLIENT mode.
2950A(config)#exit
2950A#sh vtp status
VTP Version : 2
Configuration Revision : 1
Maximum VLANs supported locally : 64
Number of existing VLANs : 5
VTP Operating Mode : Client
VTP Domain Name : Cisco
VTP Pruning Mode : Disabled
VTP V2 Mode : Disabled
VTP Traps Generation : Disabled
MD5 digest : 0x70 0x01 0xF2 0x72 0x97 0xA1 0x35 0xE1
Configuration last modified by: 2950 SwitchA at 11-29-93 20:39:24
Local updater ID is 2950 SwitchA on interface V11 (lowest numbered VLAN :

```

图 5.77 配置 2950A 的 VTP

5.6 基于交换机端口的 VLAN 配置

125

```

switch#conf t
Enter configuration commands, one per line. End with CNTL/Z
switch(config)#hostname 2950B
2950B(config)#vtp domain Cisco
Changing VTP domain name from NULL to Cisco
2950B(config)#vtp mode client
Setting device to VTP CLIENT mode.
2950B(config)#exit
2950B#

```

图 5.78 配置 2950B 的 VTP

② 配置 Trunk。

将交换机 3550A 端口 fa0/1 和端口 fa0/3 配置为 Trunk 端口，并用 802.1q 封装，如图 5.79 所示。

```

3550A(config)#interface fa0/1
3550A(config-if)#switchport trunk encapsulation ?
    dot1q      Interface uses only 802.1q trunking encapsulation when trunking
    isl       Interface uses only ISL trunking encapsulation when trunking
    negotiate  Device will negotiate trunking encapsulation with peer on
               interface

3550A(config-if)#switchport trunk encapsulation dot1q
3550A(config-if)#switchport mode trunk
3550A(config-if)#interface fa0/3
3550A(config-if)#switchport trunk encapsulation dot1q
3550A(config-if)#switchport mode trunk

```

图 5.79 设置 3550A 的 Trunk 端口

分别将交换机 2950A 和 2950B 的端口 fa0/11 设置为 Trunk 端口，如图 5.80 所示。

```

2950A(config)#interface fa0/11
2950A(config-if)#switchport mode trunk
2950B(config)#interface fa0/11
2950B(config-if)#switchport mode trunk

```

图 5.80 设置 2950A 和 2950B 的 Trunk 端口

③ 创建 VLAN。

创建两个 VLAN：VLAN 10 和 VLAN 20，并用 show vlan 命令验证，如图 5.81 所示。

```

3550A(config)#vlan 10
3550A(config-vlan)#vlan 20
3550A(config-vlan)#exit
3550A(config)#exit
3550A#sh vlan

VLAN Name                               Status      Ports
-----+-----+-----+-----+
  1   default                            active     Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                            Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                            Fa0/9, Fa0/10
  10  VLAN0010                           active
  20  VLAN0020                           active
  1002 fddi-default                      active
  1003 token-ring-default                active
  1004 fddinet-default                  active

```

图 5.81 创建 VLAN 并验证

④ 分配交换机端口加入 VLAN。

分别将交换机 2950A 和 2950B 的端口 fa0/2 加入 vlan 10 和 vlan 20，如图 5.82 所示。

```
2950A(config)#interface fa0/2
2950A(config-if)#switchport access vlan 10
2950B(config)#interface fa0/2
2950B(config-if)#switchport access vlan 20
```

图 5.82 将交换机端口加入 VLAN

⑤ 配置第三层交换机。

(a) 在 3550 交换机上分别设置各 VLAN 的接口 IP 地址，如图 5.83 所示。3505 交换机将 VLAN 作为一种接口对待，就像路由器上的一样，提供 VLAN 10 和 VLAN 20 之间的路由。

```
3550A(config)#int vlan 10
3550A(config-if)#ip address 10.10.10.1 255.255.255.0
3550A(config-if)#no shut
3550A(config-if)#int vlan 20
3550A(config-if)#ip address 20.20.20.1 255.255.255.0
3550A(config-if)#no shut
3550A(config-if)#exit
3550A(config)#
```

图 5.83 设置 IP

(b) 启用路由，如图 5.84 所示。

```
3550A(config)#ip routing
```

图 5.84 启用路由

⑥ 配置各交换机的管理地址，如图 5.85 所示。

```
3550A(config)#int vlan 1
3550A(config-if)#ip address 192.168.10.1 255.255.255.0
3550A(config-if)#no shut
2950A(config)#int vlan 1
2950A(config-if)#ip address 192.168.10.2 255.255.255.0
2950A(config-if)#no shutdown
2950B(config)#int vlan 1
2950B(config-if)#ip address 192.168.10.3 255.255.255.0
2950B(config-if)#no shutdown
```

图 5.85 设置管理 IP

⑦ 配置主机 Host A 和 Host B，并进行测试。

Host A IP : 10.10.10.2/24 default gateway: 10.10.10.1

Host B: IP : 20.20.20.2/24 default gateway: 20.20.20.1

测试，在 3550 交换机上分别 ping 2950 交换机：

```
3550A#ping 192.168.10.2
```

5.6 基于交换机端口的 VLAN 配置

127

```
3550A#ping 192.168.10.3
```

在主机 Host A ping 主机 Host B。

实例 2, 以图 5.86 中的网络拓扑为例进行配置, 实现 VLAN 跨越多个交换机及不同 VLAN 之间的通信。

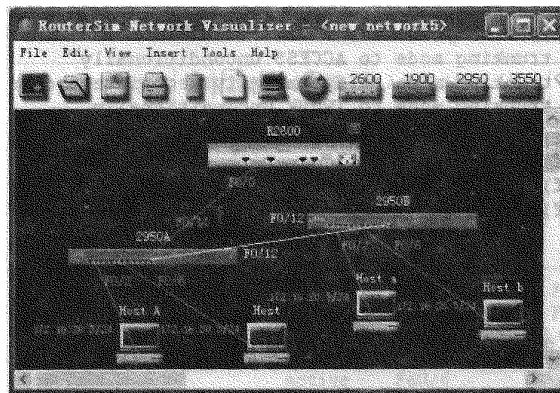


图 5.86 Vlan 跨越交换机 2950A 和 2950B

① 配置 VTP。

在 2950A 交换机设置 VTP 域的域名为 Test, 用命令 “vtp mode ?” 查看 VTP 工作模式, 并用命令 show vtp status 验证, 如图 5.87 所示。

```
switch#config t
Enter configuration commands, one per line. End with CNTL/Z
switch(config)#hostname 2950A
2950A(config)#vtp domain Test
Changing VTP domain name from NULL to Test
2950A(config)#vtp mode ?
    client      Set the device to client mode.
    server      Set the device to server mode.
    transparent Set the device to transparent mode.

2950A(config)#vtp mode server
Device mode already VTP SERVER.
2950A(config)#exit
2950A#show vtp status
VTP Version          : 2
Configuration Revision : 1
Maximum VLANs supported locally : 64
Number of existing VLANs   : 5
VTP Operating Mode       : Server
VTP Domain Name         : Test
VTP Pruning Mode        : Disabled
VTP V2 Mode              : Disabled
VTP Traps Generation    : Disabled
MD5 digest              : 0x70 0x01 0xF2 0x72 0x97 0xA1 0x35 0xEB
Configuration last modified by: 0.0.0.0 at 11-29-93 20:39:24
Local updater ID is 0.0.0.0 on interface V11 (lowest numbered VLAN interface
found)
```

图 5.87 设置 VTP 并验证

(2) 启动 Trunk。

设置 2950A 交换机的端口 fa0/12、fa0/11 为 Trunk 端口，如图 5.88 所示。

```
2950A#config t
Enter configuration commands, one per line. End with CNTL/Z
2950A(config)#interface fa0/12
2950A(config-if)#switchport mode ?
    access   Set trunking mode to ACCESS unconditionally
    dynamic   Set trunking mode to dynamically negotiate access or trunk mode
    trunk     Set trunking mode to TRUNK unconditionally
2950A(config-if)#switchport mode trunk
2950A(config-if)#interface fa0/11
2950A(config-if)#switchport mode trunk
2950A(config-if)#exit
2950A(config)#
```

图 5.88 在 2950A 设置 Trunk 端口

设置 2950B 交换机的端口 fa0/12 为 trunk，如图 5.89 所示。

```
switch>en
switch#config t
Enter configuration commands, one per line. End with CNTL/Z
switch(config)#hostname 2950B
switch(config)#interface fa0/12
2950B(config-if)#switchport mode trunk
2950B(config-if)#exit
2950B(config)#
```

图 5.89 在 2950B 设置 Trunk 端口

(3) 创建 VLAN。

在 2950A 交换机创建 VLAN，特权模式下使用 vlan database 命，如图 5.90 所示。

```
2950A#vlan database
2950A(vlan)#vlan 2 name vlan2
VLAN 2 added:
    Name: vlan2
2950A(vlan)#vlan 3 name vlan3
VLAN 3 added:
    Name: vlan3
2950A(vlan)#exit
APPLY completed.
Exiting....
```

图 5.90 创建 VLAN

(4) 分配端口到 VLAN。

将 2950A 交换机的端口 fa0/2 分配给 VLAN2，端口 fa0/6 分配给 VLAN3，如图 5.91 所示。

5.6 基于交换机端口的 VLAN 配置

129

```
2950A#config t
Enter configuration commands, one per line. End with CNTL/Z
2950A(config)#interface fastethernet 0/2
2950A(config-if)#switchport access vlan 2
2950A(config-if)#switchport mode access
2950A(config-if)#interface fastethernet 0/6
2950A(config-if)#switchport access vlan 3
2950A(config-if)#switchport mode access
2950A(config-if)#

```

图 5.91 将 2950A 的端口加入 VLAN

用 show vlan 命令验证，如图 5.92 所示。

```
2950A#show vlan
-----  

VLAN Name          Status    Ports  

-----  

1     default       active    Fa0/1, Fa0/3, Fa0/4, Fa0/5  

                           Fa0/7, Fa0/8, Fa0/9, Fa0/10  

                           Fa0/11, Fa0/12  

2     vlan2         active    Fa0/2  

3     vlan3         active    Fa0/6  

1002 fddi-default  active  

1003 token-ring-default  active  

1004 fdnet-default   active  

1005 trnet-default   active  

-----  

VLAN Type  SAID      MTU    Parent RingNo BridgeNo Stp  BrdgMode Transl  Trans:  

-----  

1   enet  100001  1500   -     -     -     -     -     0     0  

2   enet  100002  1500   -     -     -     -     -     0     0  

3   enet  100003  1500   -     -     -     -     -     0     0  

1002 fddi 101002  1500   -     -     -     -     -     0     0  

1003 tr   101003  1500   -     -     -     -     -     0     0  

1004 fdnet 101004  1500   -     -     -     ieee -     0     0  

1005 trnet 101005  1500   -     -     -     ibm -     0     0  

--More--
```

图 5.92 验证

将 2950B 设置成 VTP 客户模式，2950B 从 2950A 接收 VLAN 信息，如图 5.93 所示。

```
2950B#config t
Enter configuration commands, one per line. End with CNTL/Z
2950B(config)#vtp domain Test
Changing VTP domain name from NULL to Test
2950B(config)#vtp mode client
Setting device to VTP CLIENT mode.
2950B(config)#

```

图 5.93 设置 2950B 为 VTP CLIENT 模式

将 2950B 的端口 fa0/2 和 fa0/6 分别加入到 VLAN2 和 VLAN3，可以通过 show vlan 命令验证，如图 5.94 所示。

```

2950B(config)#interface fastethernet 0/2
2950B(config-if)#switchport access vlan 2
2950B(config-if)#switchport mode access
2950B(config-if)#
2950B(config-if)#interface fastethernet 0/6
2950B(config-if)#switchport access vlan 3
2950B(config-if)#switchport mode access
2950B(config-if)#exit
2950B(config)#

```

图 5.94 将 2950B 的端口加入 vlan

(5) 配置 VLAN 之间的路由。

实现 VLAN 之间的通信，就需要路由器或第三层交换机。

此示例中是以 2600 路由器来实现 VLAN 之间的通信。要在 2600 路由器的 FastEthernet 端口上支持 ISL 或 802.1q，需要将路由器的端口划分成许多逻辑接口（非物理），每一个逻辑接口对应一个 VLAN，这些逻辑接口就叫子接口（subinterfaces）。设置路由如图 5.95 所示。

```

2600#config t
Enter configuration commands, one per line. End with CNTL/Z
2600(config)#hostname R2600
R2600(config)#interface fastethernet 0/0
R2600(config-if)#no ip address
R2600(config-if)#no shutdown
R2600(config-if)#interface fastethernet 0/0.1
R2600(config-subif)#encapsulation dot1q 1
R2600(config-subif)#ip address 172.16.10.1 255.255.255.0
R2600(config-subif)#interface fastethernet 0/0.2
R2600(config-subif)#encapsulation dot1q 2
R2600(config-subif)#ip address 172.16.20.1 255.255.255.0
R2600(config-subif)#interface fastethernet 0/0.3
R2600(config-subif)#encapsulation dot1q 3
R2600(config-subif)#ip address 172.16.30.1 255.255.255.0
R2600(config-subif)#exit
R2600(config)#

```

图 5.95 设置路由

注：2950 系列交换机只支持 802.1q 中继，而不支持 ISL 中继，所以在 2600 路由器的子接口配置 802.1q。

(6) 配置主机 Host A、Host B、Host a、Host b。

Host A, IP Address : 172.16.20.3/24, default gateway : 172.16.20.1.

Host B, IP Address : 172.16.30.3/24, default gateway : 172.16.30.1.

Host a, IP Address : 172.16.20.5/24, default gateway : 172.16.20.1.

Host b, IP Address : 172.16.30.5/24, default gateway : 172.16.30.1.

(7) 验证连通性。

在属于 VLAN 2 的 Host A 上 ping 172.16.20.1。

在属于 VLAN 3 的 Host B 上 ping 172.16.30.1。

在 Host A 上 ping Host B。

7 利用 Socket API 实现许可认证软件

7.1 实验目的

通过完成实验，掌握应用层文件传输的原理；了解传输过程中传输层协议选用、应用层协议设计和协议开发等概念。

7.2 需求说明

某远程桌面连接软件 A 需要有一个许可证程序来保证合法运行。这样的许可证（License）是带有限制的。用户提供支付费用，获得一个若干人（如：10 人或 50 人）的许可证。规则如下：

- 1、某组织管理员在购买许可证时，输入用户名、口令和许可证类型，许可证程序返回一个由 10 个数字组成的序列号。
- 2、该组织的用户第一次使用软件 A 时，输入序列号。
- 3、该组织用户运行软件时，向许可证服务器发送验证。
- 4、许可证服务器查询得到该序列号的使用人数，如果未到达上限，则返回授权指令；否则，返回拒绝指令。
- 5、软件 A 得到授权指令，允许用户使用软件。否则，提示用户稍后再试，退出程序。

当软件 A 或非正常退出（崩溃被其它程序中止）时，许可证服务器应在扣除使用人数时剔除它。可以使软件 A 定期（如：30 分钟）向服务器报告其状态，超过一定时间没有收到报告时，则认定崩溃。

当许可证服务器崩溃时，软件 A 应能在重新启动时恢复。许可证服务器重启后，如果新的软件 A 前来连接，服务器不可以因接受其连接而拒绝已认证用户的连接。

7.3 输入输出接口

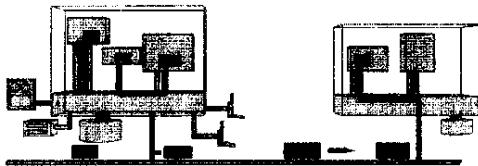
程序应提供 Windows 控制台可执行程序接口，以便于测试。

7.4 技术建议

- 1、本实验自选开发语言，但推荐 C 和 C++；
- 2、服务器的网页接口部分推荐 PHP，用于表达数据。
- 3、了解基本的动态文档网页开发。

7.5 附录一：编写许可证服务器

第 13 章 基于数据报(Datagram)的编程：编写许可证服务器^①



概念与技巧

- 基于数据报的编程, 数据报 socket
- TCP 与 UDP
- 许可证服务器
- 软件时间戳(Software ticket)
- 设计健壮系统
- 设计分布式系统
- Unix 域的 socket

相关的系统调用与函数

- socket
- sendto, recvfrom

13.1 软件控制

程序的运行需要内存、CPU 和一些系统资源。操作系统关心这类事情,但是一些程序还需要关心另一件事情:就是程序拥有者的允许。

从合法性的角度讲,需要有一个许可证(license)来保证程序的合法运行,但是有些许可证却是带有限制的。例如,有的许可证是限制同时运行程序的用户数。一个 10 人的许可证可能需要一定的花费,而 50 人的许可证可能需要更多的花费。有些厂商租赁软件许可证,当租赁期到了,程序就不能继续运行。当然除了合法性方面之外,软件也还受到其他一些因素的限制。学校里的计算机房就可能限制每天游戏程序运行的次数。

有些软件拥有者使用诚信机制来限制程序的使用,他们把许可证条款打印在屏幕上或纸上并要求用户遵守协约。其他的则使用特定的技术来实施许可证条款。

^① 本章的内容基于 Lawrence deLuca 在哈佛职业教育学院担任助教期间编写的讲稿,该讲稿取材于他参与开发的一个产品。

一种实施许可证技术是编写程序来执行许可证控制。通常的做法是设计从一个许可证服务器获得许可的应用程序。该服务器是一个进程，它授权应用程序的运行。

许可证服务器明确许可证条款并且强制执行该条款，如图 13.1 所示。

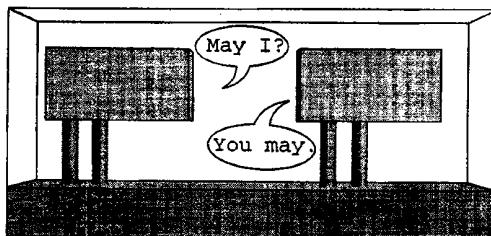


图 13.1 许可证服务器给予许可

请求许可和给予许可需要客户和许可证服务器之间的通信。

许可证服务器是如何工作的？本章将学习一个具体的客户/服务器许可证控制模型。通过该模型的学习，可以接触到另外一种类型的 socket——数据报 socket，另外的一个地址域——Unix 域，一个维持系统状态的网络协议，以及其他一些有关设计安全健壮的客户/服务器系统的知识。

13.2 许可证控制简史

软件控制技术的使用已经有多年的发展历史了。

在单机版的个人计算机时代，对软件的限制是靠特定的磁盘或由隐藏在特定磁道上的密码来实现的。磁盘上的密码很难被复制，并且只有磁盘在驱动器中的时候程序才能运行。如果磁盘丢失了或被损坏了，该程序将不能再运行。人们很快就破解并找到了如何来复制这种特殊磁盘的方法，所以软件厂商发明了硬件密钥。硬件密钥是一个适配器，它可以插在并口、串口或 USB 口上；被许可的程序只有在发现该适配器的时候才能运行。如果硬件密钥丢失，程序将不能运行。

然而网络计算机和多用户系统却引起了新的问题。如果 10 个用户同时想运行计算机或网络上的同一个程序，那么是不是每个用户都需要在服务器的端口上插一个硬件密钥呢？软件厂商们需要一种可靠的并且不需要给合法用户增加额外负担的方法来实施许可证条款。

网络和多用户系统提供了一种新的解决方法：许可证服务器。程序不是从磁盘或密钥取得许可，而是从服务器进程取得。许可证服务器被一台计算机上多个用户共享，服务器进程能够控制程序的使用人数、使用时间、使用地点甚至程序的使用方式。伴随更多的计算机加入因特网，服务器控制对软件和数据访问的需求也更加迫切。

本章中的许可证服务器将实施 n 用户的限制。也就是说，服务器只允许特定数量的程序实例同时运行。

13.3 一个非计算机系统实例：轿车管理系统

一个公司购买了许可证，该许可证限制了同时使用程序的用户数。公司可能拥有比许可证所允许的用户要多的雇员，但是并非所有的雇员要同时使用程序。怎样的一个设计才能满足上面的需求呢？

现实世界中有很多系统，通常由一大群人来共享其中的资源，而这些资源是有限的。这里将分析一个模型：雇员共享公司轿车的问题。一个公司拥有特定数量的轿车，同时还拥有更多数量的雇员。如何控制对轿车的使用呢？

13.3.1 轿车钥匙管理描述

控制轿车的使用是通过控制对轿车钥匙的访问。当想用轿车的时候，必须先得到一把钥匙。如果在钥匙盒中没有钥匙了，将不能使用轿车。如果有可用钥匙，可以先拿一把钥匙，签名，然后使用轿车。使用完毕后，把钥匙放回钥匙盒，在使用轿车名单中划去自己的名字。过程描述如图 13.2 所示。

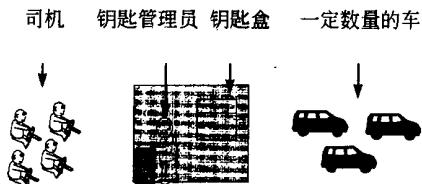


图 13.2 控制对轿车的使用

签名列表有何作用呢？该系统的目的是通过控制对轿车的使用，使得可用的钥匙一直很充足。人并非是完美的，有时司机会忘记归还钥匙。钥匙管理员可以根据签名列表找到该司机，确定他是否仍在使用车。

轿车使用管理系统是控制软件使用的一个现实模型。在将该系统转换成软件系统之前，需要更为详细地描述该系统。

钥匙管理系统的构成如下：

- (1) 钥匙管理中心的地点——到哪儿可以获得钥匙
- (2) 钥匙管理员——执行策略的人
- (3) 钥匙——需要得到的东西
- (4) 签名列表——保存钥匙和找回钥匙的记录

13.3.2 用客户/服务器方式管理轿车

在给出轿车钥匙管理系统的构成后，下面将用客户/服务器语言来描述该系统。

(1) 服务器和客户

谁是服务器和谁是客户？钥匙管理员拥有司机需要的钥匙。用网络术语来描述，钥匙

管理员是服务器，司机是客户。

(2) 协议

所用的协议是什么？交互的事务是什么？轿车钥匙管理协议包括两个主要的事务。

- 获得钥匙

客户：你好，我需要一把钥匙。

服务器：这里只有 5 号钥匙，没有别的钥匙了。

- 归还钥匙

客户：我已经用好 5 号钥匙。

服务器：谢谢。

(3) 通信系统

客户和服务器如何通信？在该系统中，人们通过对话来传递简单信息。

(4) 数据结构

司机和钥匙管理员需要什么样的数据结构呢？钥匙管理员保留一个用户签名列表，一把钥匙对应列表的一项。当一个用户取走一把钥匙，钥匙管理员在该项记下用户的名字，当用户归还钥匙时，管理员把司机的名字从列表中擦去。下表解释了用户签名列表：

签名列表	
钥匙 #	司 机
1	adam@sales
2	
3	carol@support
4	

如果在签名列表中没有司机与钥匙号对应，表示该钥匙是可用的。反之，表示不可用。

13.4 许可证管理

本节将钥匙管理系统的知识运用到许可证管理系统中。

13.4.1 许可证服务系统：它做些什么

图 13.3 描述了人们试图运行许可证程序的过程。工作如下：

- (1) 用户 U 运行被许可的程序 P；
- (2) 程序 P 向服务器 S 请求运行许可；
- (3) 服务器检查当前运行程序 P 的用户数；
- (4) 如果上限未达到，S 给予许可，程序 P 运行；
- (5) 如果达到上限，S 拒绝许可，程序 P 告诉 U 稍后再试。

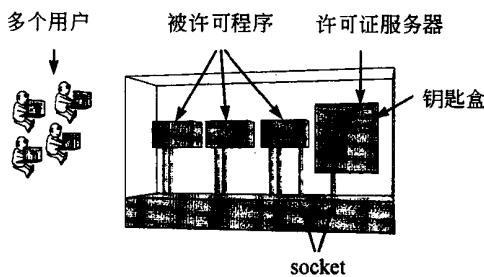


图 13.3 控制软件的使用

许可证服务系统与轿车钥匙服务系统稍有不同。在轿车系统中，司机向钥匙管理员请求许可；而在这里，程序向服务器请求许可。这就像司机请求轿车，而轿车向钥匙管理员请求钥匙^①。

程序的创建者要编写所有的程序：应用程序和服务器。这两个程序作为一个系统。服务器给予应用程序运行许可和实施许可证条款。如果许可证服务器不在运行，应用程序将得不到许可，不能运行。

这里以轿车钥匙服务系统作为模型进行了讨论。那么如何将这种思想运用到软件系统中呢？被许可程序如何从服务器得到许可？服务器如何给予许可？软件系统和轿车钥匙系统的等价之处在哪里呢？

13.4.2 许可证服务系统：如何工作

(1) 票据模型

钥匙管理员负责分发钥匙，许可证服务器发布什么呢？以电影院和棒球场为例，付钱获得进场许可，然后得到一张入场票据。与此类似，这里的许可证服务器发布的是数字票据。这种票据又是什么样子的呢？客户和服务器交换字符串，所以票据是字符串，其格式如下：

```
pid.ticketnumber 例如：6589.3
```

每张票据包括持有该票据进程的 PID 以及票据编号。在票据中包含 PID 的原因其实就和在飞机票上印上名字的道理是一样的。票据中的 PID 标识票据的使用者，在票据丢失时，可以帮助找回票据。进程可能丢失票据吗？

(2) 服务器和客户

谁是客户和谁是服务器？许可证服务器持有程序需要的资源：票据。用网络术语，许可证服务器是服务器，而应用程序是客户。

(3) 协议

协议是什么？交互的事务是什么？这里的票据管理协议包括下面的两个主要事务：

^① 这个概念并不难理解。随着设备越来越先进，连接越来越方便，很可能在不久的将来就可以通过你的收音机询问服务器是否有播放你最喜欢歌曲的许可。

- 获取钥匙

客户：HELO mypid

服务器：TICK ticketid or FAIL no tickets

- 归还钥匙

客户：GBYE ticketid

服务器：THNX message

这里定义了基于文本的协议，协议中使用了 4 个字符的简单命令，这与前面的 Web 服务器所用的命令类似。

(4) 通信系统

上述简短的文本信息如何在客户和服务器之间传送？在本文后面将讨论该问题。

(5) 数据结构

客户和服务器之间所用的数据结构是什么？这里使用整型数组作为签名列表。数组的每个入口对应一张票据。当一个客户取走了一张票，管理员将客户对应的 PID 写到该入口。如下表：

签名列表	
tick #	process
1	1234
2	0
3	6589
4	0

如果数组中的某个元素值是 0，表明该票据可用。否则，表明该票据正在被使用。

13.4.3 一个通信系统的例子

客户如何请求票据？服务器如何发布票据？这涉及到进程间的通信形式。客户和服务器之间通过短消息通信。服务器必须接收，处理和应答来自多个客户的请求。目前，哪些技术是可以使用的呢？本文前面学习的信号和管道机制可以考虑，但是信号太短了，而管道只连接相关联的进程。所以，使用 socket 是最明显的答案。而对于 socket，也有两种不同的选择。一种是基于流的 socket，它是用来连接不相关的进程的。另一种 socket 被称为数据报 socket，或称为 UDP，对于现在的项目，这种 socket 是更好的选择。

13.5 数据报 socket

流 socket 传送数据就跟电话网中传送声音一样，客户先建立连接，然后使用该连接进行单向、双向或类似管道的字节流传送。

数据报通信则与从一个邮箱到另一个邮箱发送包裹类似。客户不必建立连接，只要向特定的地址发送消息，而服务器进程在该地址接收消息。

流 socket 使用的网络协议叫 TCP 即传输控制协议(Transmission Control Protocol)。数据报 socket 叫 UDP 即用户数据报协议(User Datagram Protocol)。它们的区别是什么呢？何时选择何种 socket 是较好的呢？在程序中如何使用数据报 socket 呢？

13.5.1 流与数据报的比较

socket 是如何工作的？数据如何在 Internet 上传输的？当用户向流 socket 写数据时，内核要做些什么？这与向数据报 socket 写数据有何区别？

从一个流 socket 传输到另一个流 socket 的数据流，看上去是连续的、无缝的，实际上这是一种错觉。Internet 连接要把数据分割成独立的数据包。网络数据传输过程如图 13.4 所示。

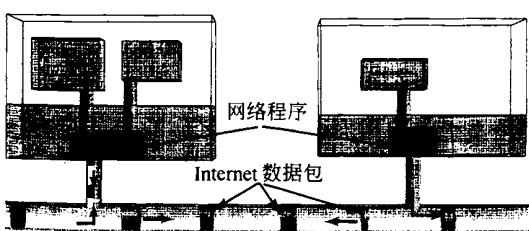


图 13.4 Internet 包转载数据

在现实世界中，有很多将一大块数据分割成若干较小的数据块的例子。假设要通过快递传送 100 页的文档。当快递公司要求你使用只能装 20 页的信封时，该怎么办？可以把文档分割成 5 个包裹，每个单独地打包和附上地址。然后把这 5 个独立的包裹放到邮箱中，运输系统将负责把它们送到目的地。在接收端，接受者打开这 5 个包裹，并把它们按顺序重组原来的文档。

Internet 就像上述运输系统一样，它上面传输的数据必须符合大小的限制。大块的数据被分割成小块的数据来传输，接收端必须以正确的顺序重组数据块。但通信过程可以被连接和中断，如图 13.5 所示。

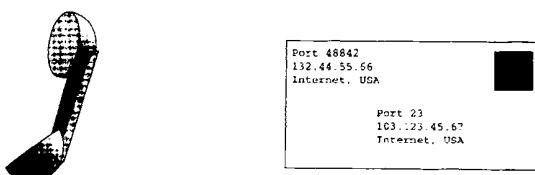


图 13.5 通信可以被连接和中断

流 socket 负责分割、排序、重组的所有工作。数据报 socket 则不会。下表给出了它们之间的区别。

TCP	UDP
流	数据报
分片/重组	否
排序	否
可靠的	可能未到达
连接的	多个发送者

在流 socket 中, 内核将大的数据块分割成带编号的数据片。位于接收机器上的内核按顺序接收数据片, 重组成发送者所发送的原始数据。在数据报 socket 中, 内核并不给数据加编号标签, 在目的地也不重组。

流 socket 对传送负责, 数据报 socket 则不。流 socket 的接收端检查数据片的顺序来确保数据完整到达。接收端提示发送端丢失的数据片的编号, 并等待丢失数据片的重传。数据报 socket 并不检查丢失数据报, 也不要求重传。如果某个包丢失在 Internet 中, 它只是不到达。TCP 比 UDP 做更多的工作。UDP 更快、更简单, 给网络较少的负载。

UDP 接收消息跟邮箱系统方式相同: 发送者将你的地址写在信件上, 邮件系统负责将信件送到你的邮箱中, 然后你从邮箱中取出信。TCP socket 需要明确调用 accept、read 和 close 来读取远端进程的消息。

UDP 正好适合这里的应用。客户发送短消息来获得许可, 服务器发送短消息给予或拒绝许可。客户和服务器的交互不需要建立连接, 不需要分片和重组。可靠性甚至是不要的。如果请求或票据丢失, 客户可以再次请求。在任何事件中, 服务器和客户就像在一台机器上或者一个网络的同一部分, 所以丢失数据报的风险较小。

UDP 对于 Web 服务器和 e-mail 服务是一个较差的选择。Web 服务器和 e-mail 信息可能是大的文件, 这些字节流必须完全和按序到达目的地。UDP 对于允许丢失帧的声音和视频流是较好的选择。

13.5.2 数据报编程

数据报与邮件网络系统类似, 包括 3 个主要部分: 目的地地址、返回地址和消息, 如图 13.6 所示。

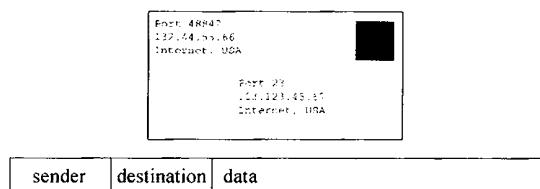
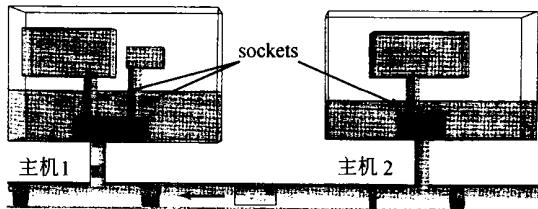


图 13.6 数据报的 3 部分

数据报 socket 可以被理解成一个内部有数据的盒子, 发送者给出目的 socket 的地址。网络将数据从发送者发送到目的 socket。接收进程从该 socket 中读取数据。程序使用 sendto 发送数据和 recvfrom 来读取数据, 如图 13.7 所示。



一台主机可能有多个接收 socket。每个 socket 被指派给一个特定的端口号。地址用主机上的端口来区分。

图 13.7 使用 sendto 和 recvfrom

1. 接收数据报

程序 dgrecv.c 是一个简单的基于数据报的服务器。dgrecv.c 使用命令行传过来的端口号建立 socket，然后进入循环，接收和打印从客户端发来的数据报：

```
*****  
* dgrecv.c - datagram receiver  
*           usage: dgrecv portnum  
*           action: listens at the specified port and reports messages  
*/  
  
# include <stdio.h>  
# include <stdlib.h>  
# include <sys/types.h>  
# include <sys/socket.h>  
# include <netinet/in.h>  
  
# define oops(m,x) { perror(m); exit(x); }  
  
int make_dgram_server_socket(int);  
int get_internet_address(char *, int, int *, struct sockaddr_in *);  
void say_who_called(struct sockaddr_in *);  
  
int main(int ac, char *av[])
{
    int      port;          /* use this port */  
    int      sock;          /* for this socket */  
    char    buf[BUFSIZ];    /* to receive data here */  
    size_t   msglen;        /* store its length here */  
    struct   sockaddr_in saddr;  /* put sender's address here */  
    socklen_t saddrlen;     /* and its length here */  
  
    if (ac == 1 || (port = atoi(av[1])) <= 0){
        fprintf(stderr, "usage: dgrecv portnumber\n");
        exit(1);
    }
}
```

• 390 •

Unix/Linux 编程实践教程

```

}

/* get a socket and assign it a port number */

if( (sock = make_dgram_server_socket(port)) == -1 )
    oops("cannot make socket",2);

/* receive messaages on that socket */

saddrlen = sizeof(saddr);
while( (msglen = recvfrom(sock,buf,BUFSIZ,0,&saddr,&saddrlen))>0 )
{
    buf[msglen] = '\0';
    printf("dgrecv: got a message: %s\n", buf);
    say_who_called(&saddr);
}
return 0;
}

void say_who_called(struct sockaddr_in *addrp)
{
    char      host[BUFSIZ];
    int       port;

    get_internet_address(host, BUFSIZ, &port, addrp);
    printf(" from: %s: %d\n", host, port);
}

```

辅助函数 `make_dgram_server_socket` 和 `get_internet_address` 将在后面给出的文件 `dgram.c` 中定义。在数据报 socket 中接收消息比从流 socket 接收简单。`recvfrom` 函数阻塞直到数据报到达。当数据报到达时，消息内容、返回地址和其长度将被复制到缓存中。

2. 发送数据报

程序 `dgsend.c` 发送数据报。`dgsend.c` 创建一个 socket，然后用它发送消息到以命令行参数传入的特定的主机和端口号。

```

*****
* dgsend.c - datagram sender
*
*           usage: dgsend hostname portnum "message"
*           action: sends message to hostname:portnum
*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define oops(m,x) { perror(m); exit(x); }

```

```

int make_dgram_client_socket();
int make_internet_address(char *, int, struct sockaddr_in *);

int main(int ac, char *av[])
{
    int      sock;           /* use this socket to send */
    char    *msg;            /* send this message */
    struct  sockaddr_in  saddr; /* put sender's address here */

    if (ac != 4){
        fprintf(stderr, "usage: dgsend host port 'message'\n");
        exit(1);
    }
    msg = av[3];

    /* get a datagram socket */

    if( (sock = make_dgram_client_socket()) == -1 )
        oops("cannot make socket",2);

    /* combine hostname and portnumber of destination into an address */

    make_internet_address(av[1], atoi(av[2]), &saddr)

    /* send a string through the socket to that address */

    if (sendto(sock, msg, strlen(msg), 0, &saddr, sizeof(saddr)) == -1)
        oops("sendto failed", 3);
    return 0;
}

```

sendto 函数将缓存中的内容发送到特定地址的 socket。

3. 辅助函数

创建 socket 和 socket 地址的细节被封装在 dgram.c 中：

```

*****
* dgram.c
* support functions for datagram based programs
*/
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string.h>

#define HOSTLEN 256

```

• 392 •

Unix/Linux 编程实践教程

```
int make_internet_address();

int make_dgram_server_socket(int portnum)
{
    struct sockaddr_in saddr; /* build our address here */
    char    hostname[HOSTLEN]; /* address */
    int     sock_id;          /* the socket */

    sock_id = socket(PF_INET, SOCK_DGRAM, 0); /* get a socket */
    if (sock_id == -1) return -1;

    /* build address and bind it to socket **/

    gethostname(hostname, HOSTLEN);           /* where am I ? */
    make_internet_address(hostname, portnum, &saddr);

    if( bind(sock_id,(struct sockaddr *) &saddr, sizeof(saddr)) != 0 )
        return -1;

    return sock_id;
}

int make_dgram_client_socket()
{
    return socket(PF_INET, SOCK_DGRAM, 0);
}

int make_internet_address(char * hostname, int port, struct sockaddr_in * addrp)
/*
 * constructor for an Internet socket address, uses hostname and port
 * (host,port) -> *addrp
 */
{
    struct hostent * hp;

    bzero((void *)addrp, sizeof(struct sockaddr_in));
    hp = gethostbyname(hostname);
    if (hp == NULL) return -1;
    bcopy((void *)hp->h_addr, (void *)&addrp->sin_addr, hp->h_length);
    addrp->sin_port = htons(port);
    addrp->sin_family = AF_INET;
    return 0;
}

int get_internet_address(char * host, int len, int * portp, struct sockaddr_in * addrp)
/*
 * extracts host and port from an internet socket address
 ** addrp -> (host,port)
 */
{
    strncpy(host, inet_ntoa(addrp->sin_addr), len);
}
```

```

    * portp = ntohs(addrp ->sin_port);
    return 0;
}

```

创建一个数据报 socket 与创建一个流 socket 类似。其不同点在于，这里设置 socket 的类型为 SOCK_DGRAM，而且不要调用 listen 函数。

4. 编译和测试

```

$ cc dgrecv.c dgram.c -o dgrecv
$ ./dgrecv 4444 &
[1] 19383
$ cc dgsend.c dgram.c -o desend
$ ./dgsend host2 4444 "testing 123"
dgrecv:got a message: testing 123
from:10.200.75.200:1041
$ ps
PID   TTY      TIME      CMD
14599 pts/3    00:00:00  bash
19383 pts/3    00:00:00  dgrecv
19393 pts/3    00:00:00  ps
$ 

```

编译服务器，并启动它，使得它监听在端口 4444。然后编译和运行客户，使客户发送字符串到端口 4444。服务器接收消息，打印消息，并且打印消息的返回地址。客户 socket 拥有主机地址和端口号，内核随机地给它分配了一个端口号 1041。ps 进程显示了服务器正在运行。

13.5.3 sendto 和 recvfrom 的小结

sendto		
目标	从 socket 发送消息	
头文件	# include <sys/types.h> # include <sys/socket.h>	
函数原型	nchars = sendto(int socket, const void * msg, size_t len, int flags, const struct sockaddr * dest,socklen_t dest_len);	
参数	socket	socket id
	msg	发送的字符类型的数组
	len	发送的字符数
	flags	比特的集合，设置发送属性，0 表示普通
	dest	指向远端 socket 地址的指针
	dest_len	地址长度
返回值	-1	遇到错误
	nchars	发送的字符数

• 394 •

Unix/Linux 编程实践教程

`sendto` 从源 socket 发送数据报到目的 socket。前 3 个参数与 `write` 的参数类似：要发送的 socket、保存要发送字符串的数组以及发送的字符数。与 `write` 类似，`sendto` 返回实际发送的字符数。`flags` 参数表明发送的各种属性；具体可参见 Unix 版本的帮助信息。最后两个参数给出发送目的地的 socket 地址。如果是 Internet 类型的地址，socket 地址包含目的主机的 IP 地址和端口号，而其他类型的地址则包含其他的成员。

recvfrom		
目标	从 socket 接收消息	
头文件	# include <sys/types.h> # include <sys/socket.h>	
函数原型	nchars = recvfrom(int socket, const void * msg, size_t len, int flags, const struct sockaddr * sender, socklen_t * sender_len);	
参数	socket	socket id
	msg	字符类型的数组
	len	接收的字符数
	flags	表示接收属性的比特的集合，0 表示普通
	sender	指向远端 socket 的地址的指针
	sender_len	地址长度
返回值	-1	遇到错误
	nchars	接收的字符数

`recvfrom` 从 socket 读取数据报。前 3 个参数与 `read` 类似：所要读取的 socket、存放字符的数组以及要读取的字符数。与 `read` 类似，`recvfrom` 返回实际接收的字符数。`flags` 指出了接收时所用的各种属性；具体可参见 Unix 系统的帮助信息。通过最后两个参数，可以获得发送者的地址。发送 socket 的地址将被存放在由第一个参数指向的结构中，地址长度存放在由第二个参数指向的整型值中。地址的长度必须提供给 `recvfrom` 函数；如果实际的地址是不同的长度，它将更改该值。如果第一个参数指针为空值，发送者地址将不被记录。

13.5.4 数据报应答

程序 `dgsend.c` 和 `dgrecv.c` 显示了如何从客户发送数据给服务器。服务器如何给客户发送应答呢？在现实世界中，假设有人给你发送了一封晚宴的邀请函，你将如何给出答复呢？这很简单：你只要给邀请函上的返回地址回信就行了。

程序 `dgrecv2.c` 从客户处接收消息并且发送谢谢作为应答：

```
*****  
* dgrecv2.c - datagram receiver  
*           usage: dgrecv portnum  
*           action: receives messages, prints them, sends reply  
*/  
  
# include <stdio.h>  
# include <stdlib.h>
```

第13章 基于数据报(Datagram)的编程：编写许可证服务器^①

• 395 •

```

#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define oops(m,x) { perror(m); exit(x);}

int make_dgram_server_socket(int);
int get_internet_address(char*, int, int* struct sockaddr_in*);
void say_who_called(struct sockaddr_in *);
void reply_to_sender(int, char *, struct sockaddr_in *, socklen_t);

int main(int ac, char *av[])
{
    int      port;           /* use this port */
    int      sock;           /* for this socket */
    char    buf[BUFSIZ];     /* to receive data here */
    size_t   msglen;         /* store its length here */
    struct   sockaddr_in saddr; /* put sender's address here */
    socklen_t saddrlen;      /* and its length here */

    if (ac == 1 || (port = atoi(av[1])) <= 0){
        fprintf(stderr,"usage: dgrecv portnumber\n");
        exit(1);
    }

    /* get a socket and assign it a port number */

    if( (sock = make_dgram_server_socket(port)) == -1 )
        oops("cannot make socket",2);

    /* receive messages on that socket */

    saddrlen = sizeof(saddr);
    while( (msglen = recvfrom(sock,buf,BUFSIZ,0,&saddr,&saddrlen))>0 ) {
        buf[msglen] = '\0';
        printf("dgrecv: got a message: %s\n", buf);
        say_who_called(&saddr);
        reply_to_sender(sock,buf,&saddr,saddrlen);
    }
    return 0;
}

void reply_to_sender(int sock,char *msg,struct sockaddr_in *addrp,socklen_t len)
{
    char reply[BUFSIZ + BUFSIZ];

    sprintf(reply, "Thanks for your %d char message\n", strlen(msg));
}

```

• 396 •

Unix/Linux 编程实践教程

```

        sendto(sock, reply, strlen(reply), 0, addrp, len);
    }
    void say_who_called(struct sockaddr_in * addrp)
    {
        char      host[BUFSIZ];
        int       port;
        get_internet_address(host,BUFSIZ,&port,addrp);
        printf(" from: %s: %d\n", host, port);
    }
}

```

发送者程序当然也要改变以接收应答。这作为练习由读者来完成。

13.5.5 数据报小结

数据报是从一个 socket 发送到另一个的短消息。发送者使用 sendto 来指定消息、长度和目的地。接收者使用 recvfrom 接收消息。数据报和带有地址的 Internet 上传输的数据包的基本结构接近。因此，数据报给内核网络功能和网络流量增加的负荷较少。由于数据报可能在传输中丢失，也有可能不按顺序地到达，所以它通常用对简单和高效的要求比完整性和一致性更为重要的应用中。

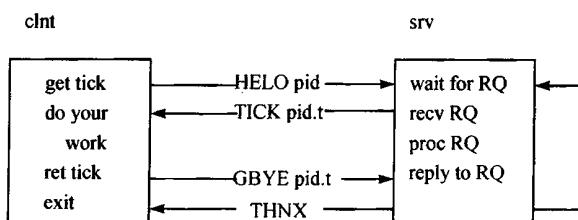
本章的许可证服务器是一种简单的消息传输，它用一个服务器来接收、处理短消息和发送请求，所以数据报是一个合适的选择。

13.6 许可证服务器版本 1.0

下面，回到许可证服务器项目上来。这里的服务器限制了程序同时运行的实例数目。当用户要运行受限制的程序时，该进程要向服务器请求运行许可。

如果并没有很多人正在用该程序，服务器发送票据给进程，给予许可证。如果达到了最大的程序实例数，服务器发送无可用票据的消息给客户，并且通知客户稍后再试或者购买支持更多用户版本的软件。被许可程序和服务器之间通过数据报来通信。

客户和服务器的运行流程及其交互过程如下。



客户和服务器分别由两个文件组成：短的文件包含 main 函数，长的文件包含票据管理函数。下面将分析客户和服务器程序。

13.6.1 客户端版本 1

```
*****  

* lclnt1.c  

* License server client version 1  

* link with lclnt_funcs1.o dgram.o  

*/  

#include <stdio.h>  

int main(int ac, char *av[])
{
    setup();
    if (get_ticket() != 0)
        exit(0);
    do_regular_work();
    release_ticket();
    shut_down();
}
*****  

* do_regular_work the main work of the application goes here
*/
do_regular_work()
{
    printf("SuperSleep version 1.0 Running - Licensed Software\n");
    sleep(10); /* our patented sleep algorithm */
}
```

客户端的最上层代码遵照了上一节中所小结的原则。客户得到票据，进行处理，释放票据，然后退出。该许可证例程是 Unix 中 sleep 程序的特殊版本，若不满意标准 sleep 版本的工作，也可以购买许可证来使用此版本的程序。当然还要运行许可证服务器，否则该 sleep 程序将拒绝运行。其中辅助函数在 lclnt_funcs1.c 中：

```
*****  

* lclnt_funcs1.c: functions for the client of the license server
*/  

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
```

```
/*
 * Important variables used throughout
 */
static int pid = -1;                      /* Our PID */
static int sd = -1;                        /* Our communications socket */
static struct sockaddr serv_addr;          /* Server address */
static socklen_t serv_alen;                /* length of address */
static char ticket_buf[128];               /* Buffer to hold our ticket */
static have_ticket = 0;                    /* Set when we have a ticket */

#define MSGLEN      128                  /* Size of our datagrams */
#define SERVER_PORTNUM 2020             /* Our server's port number */
#define HOSTLEN      512
#define oops(p) { perror(p); exit(1); }

char * do_transaction();

/*
 * setup: get pid, socket, and address of license server
 * IN no args
 * RET nothing, dies on error
 * notes: assumes server is on same host as client
 */
setup()
{
    char    hostname[BUFSIZ];

    pid = getpid();                     /* for ticks and msgs */
    sd = make_dgram_client_socket();    /* to talk to server */
    if (sd == -1)
        oops("Cannot create socket");
    gethostname(hostname, HOSTLEN);     /* server on same host */
    make_internet_address(hostname, SERVER_PORTNUM, &serv_addr);
    serv_alen = sizeof(serv_addr);
}

shut_down()
{
    close(sd);
}

***** * get_ticket
 * get a ticket from the license server
 * Results: 0 for success, -1 for failure
 */
int get_ticket()
```

```

{
    char * response;
    char buf[MSGLEN];

    if(have_ticket)                                /* don't be greedy */
        return(0);

    sprintf(buf, "HELO %d", pid);                  /* compose request */

    if ( (response = do_transaction(buf)) == NULL )
        return(-1);

    /* parse the response and see if we got a ticket.
     * on success, the message is: TICK ticket-string
     * on failure, the message is: FAIL failure-msg
     */
    if ( strncmp(response, "TICK", 4) == 0 ){
        strcpy(ticket_buf, response + 5);           /* grab ticket-id */
        have_ticket = 1;                            /* set this flag */
        narrate("got ticket", ticket_buf);
        return(0);
    }

    if ( strncmp(response, "FAIL", 4) == 0 )
        narrate("Could not get ticket", response);
    else
        narrate("Unknown message:", response);

    return(-1);
} /* get_ticket */

***** * release_ticket
* Give a ticket back to the server
* Results: 0 for success, -1 for failure
*/
int release_ticket()
{
    char buf[MSGLEN];
    char * response;

    if(! have_ticket)                            /* don't have a ticket */
        return(0);                             /* nothing to release */

    sprintf(buf, "GBYE %s", ticket_buf);        /* compose message */
    if ( (response = do_transaction(buf)) == NULL )
        return(-1);

    /* examine response

```

• 400 •

Unix/Linux 编程实践教程

```
* success: THNX info - string
* failure: FAIL error - string
*/
if ( strncmp(response, "THNX", 4) == 0 ){
    narrate("released ticket OK","");
    return 0;
}
if ( strncmp(response, "FAIL", 4) == 0 )
    narrate("release failed", response + 5);
else
    narrate("Unknown message:", response);
return(-1);
} /* release_ticket */

***** * do_transaction
* Send a request to the server and get a response back
* IN msg_p           message to send
* Results: pointer to message string, or NULL for error
*          NOTE: pointer returned is to static storage
*          overwritten by each successive call.
* note: for extra security, compare retaddr to serv_addr (why?)
*/
char * do_transaction(char * msg)
{
    static char buf[MSGLEN];
    struct sockaddr_in retaddr;
    socklen_t     addrlen = sizeof(retaddr);
    int ret;

    ret = sendto(sd, msg, strlen(msg), 0, &serv_addr, serv_alen);
    if ( ret == -1 ){
        syserr("sendto");
        return(NULL);
    }
    /* Get the response back */
    ret = recvfrom(sd, buf, MSGLEN, 0, &retaddr, &addrlen);
    if ( ret == -1 ){
        syserr("recvfrom");
        return(NULL);
    }

    /* Now return the message itself */
    return(buf);
```

第13章 基于数据报(Datagram)的编程：编写许可证服务器^①

• 401 •

```

} /* do_transaction */

***** * narrate: print messages to stderr for debugging and demo purposes
* IN msg1, msg2 : strings to print along with pid and title
* RET nothing, dies on error
*/
narrate(char * msg1, char * msg2)
{
    fprintf(stderr, "CLIENT [ %d]: %s %s\n", pid, msg1, msg2);
}
syserr(char * msg1)
{
    char    buf[MSGLEN];
    sprintf(buf, "CLIENT [ %d]: %s", pid, msg1);
    perror(buf);
}

```

get_ticket 和 release_ticket 是程序中的主要函数。它们都遵循下面的规则：产生短请求、发送消息给服务器、等待服务器的应答，然后检查应答和根据应答采取行动。

get_ticket 通过发送命令 HELLO 以及紧跟其后的 PID 来请求票据。服务器通过发送 TICK ticket-id 接收请求。服务器发送 FAIL explanation 拒绝请求。

release-ticket 通过发送命令 GBYE ticket-id 返回票据。如果票据是合法的，服务器将发送 THNX greeting 消息作为应答。如果票据不合法，服务器发送 FAIL explanation 消息。

为何票据可能是不合法的？本文的后面将讨论该问题。

13.6.2 服务器端版本 1

```

***** * lserv1.c
* License server server program version 1
*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <sys/errno.h>
#define MSGLEN      128          /* Size of our datagrams */

int main(int ac, char * av[])
{
    struct sockaddr_in client_addr;
    socklen_t addrlen = sizeof(client_addr);

```

• 402 •

Unix/Linux 编程实践教程

```

char      buf[MSGLEN];
int       ret;
int       sock;

sock = setup();

while(1) {
    addrlen = sizeof(client_addr);
    ret = recvfrom(sock,buf,MSGLEN,0,&client_addr,&addrlen);
    if ( ret != -1 ) {
        buf[ret] = '\0';
        narrate("GOT:",buf,&client_addr);
        handle_request(buf,&client_addr,addrlen);
    }
    else if ( errno != EINTR )
        perror("recvfrom");
}
}

```

许可证服务器的主函数是一个循环，主要包括接收客户请求，处理请求和发送应答。其中处理请求的代码包含在 lserv_funcs1.c 文件中。

```

*****
* lsrv_funcs1.c
* functions for the license server
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <sys/errno.h>

#define SERVER_PORTNUM 2020          /* Our server's port number */
#define MSGLEN        128            /* Size of our datagrams */
#define TICKET_AVAIL  0              /* Slot is available for use */
#define MAXUSERS      3              /* Only 3 users for us */
#define oops(x) { perror(x); exit(-1); }

*****
* Important variables
*/
int ticket_array[MAXUSERS];      /* Our ticket array */
int sd = -1;                      /* Our socket */
int num_tickets_out = 0;           /* Number of tickets outstanding */

```

第13章 基于数据报(Datagram)的编程：编写许可证服务器^①

• 403 •

```

char * do_hello();
char * do_goodbye();

/*****************
 * setup() - initialize license server
 */
setup()
{
    sd = make_dgram_server_socket(SERVER_PORTNUM);
    if (sd == -1)
        oops("make socket");
    free_all_tickets();
    return sd;
}

free_all_tickets()
{
    int i;

    for(i = 0; i < MAXUSERS; i++)
        ticket_array[i] = TICKET_AVAIL;
}

/*****************
 * shut_down() - close down license server
 */
shut_down()
{
    close(sd);
}

/*****************
 * handle_request(request, clientaddr, addrlen)
 * branch on code in request
 */
handle_request(char * req, struct sockaddr_in * client, socklen_t addrlen)
{
    char * response;
    int ret;

    /* act and compose a response */
    if (strncmp(req, "HELO", 4) == 0)
        response = do_hello(req);
    else if (strncmp(req, "GBYE", 4) == 0)
        response = do_goodbye(req);
    else
        response = "FAIL invalid request";
}

```

• 404 •

Unix/Linux 编程实践教程

```
/* send the response to the client */
narrate("SAID:", response, client);
ret = sendto(sd, response, strlen(response), 0, client, addlen);
if (ret == -1)
    perror("SERVER sendto failed");
}

***** * do_hello
* Give out a ticket if any are available
* IN msg_p           message received from client
* Results: ptr to response
* Note: return is in static buffer overwritten by each call
* NOTE: return is in static buffer overwritten by each call
*/
char * do_hello(char * msg_p)
{
    int x;
    static char replybuf[MSGLEN];
    if(num_tickets_out >= MAXUSERS)
        return("FAIL no tickets available");
    /* else find a free ticket and give it to client */

    for(x = 0; x<MAXUSERS && ticket_array[x] != TICKET_AVAIL; x++) ;
    /* A sanity check - should never happen */
    if(x == MAXUSERS) {
        narrate("database corrupt","",NULL);
        return("FAIL database corrupt");
    }

    /* Found a free ticket. Record "name" of user (pid) in array.
     * generate ticket of form: pid.slot
     */
    ticket_array[x] = atoi(msg_p + 5); /* get pid in msg */
    sprintf(replybuf, "TICK %d.%d", ticket_array[x], x);
    num_tickets_out++;
    return(replybuf);
} /* do_hello */

***** * do_goodbye
* Take back ticket client is returning
* IN msg_p           message received from client
* Results: ptr to response
```

第13章 基于数据报(Datagram)的编程：编写许可证服务器^①

• 405 •

```

* Note: return is in static buffer over written by each call
*/
char * do_goodbye(char * msg_p)
{
    int pid, slot;           /* components of ticket */

    /* The user's giving us back a ticket. First we need to get
     * the ticket out of the message, which looks like:
     *
     * GBYE pid.slot
     */
    if((sscanf((msg_p + 5), "%d.%d", &pid, &slot) != 2) ||
       (ticket_array[slot] != pid)) {
        narrate("Bogus ticket", msg_p+5, NULL);
        return("FAIL invalid ticket");
    }

    /* The ticket is valid. Release it. */
    ticket_array[slot] = TICKET_AVAIL;
    num_tickets_out--;

    /* Return response */
    return("THNX See ya!");
} /* do_goodbye */

*****narrate() - chatty news for debugging and logging purposes
*/
narrate(char * msg1, char * msg2, struct sockaddr_in * clientp)
{
    fprintf(stderr, "\t\tSERVER: %s %s ", msg1, msg2);
    if (clientp)

        fprintf(stderr, "( %s: %d)", inet_ntoa(clientp->sin_addr),
                ntohs(clientp->sin_port));
    putc('\n', stderr);
}

```

3个重要的函数解释如下。

(1) handle_request

请求由4个字符的命令带一个参数构成。服务器先检查命令，然后调用对应的函数。即使命令不合法，服务器也必须发送应答，否则客户会一直阻塞下去。

(2) do_hello

HELO命令用来请求票据。服务器查找票据数组寻找空闲的项。如果某项的PID值为0，表示这是一个可用票据。服务器使用独立的变量num_tickets_out来节省时间。服务器

接收请求后,可以通过查找表来寻找空闲项,而该变量可以指明何时表已经满了,这样就不必做查找了。

(3) do_goodbye

GBYE 命令是返回票据的请求。票据是一个由 PID 和票据编号构成的字符串。服务器将票据的 PID 和票据编号与签出列表(sign-out list)中的值进行比较,如果数据一致,服务器从签出列表中清除该项的值并且致谢客户。如果不一致,则一定是什么地方发生了错误。

如果你是飞机场的检票员,如果某个客户给出存根的编号和名字在数据库中不存在,你就可能会问:“你是从哪里得到这张票的,你是谁?”后面将讨论伪造票据问题。下面来测试这个版本的程序。

13.6.3 测试版本 1

编译服务器端程序并在后台运行它。

```
$ cc lserv1.c lserv_funcs1.c dgram.c -o lserv1
$ ./lserv1&
[1] 25738
```

编译好客户端程序,然后同时运行 4 个实例:

```
$ cc lclnt1.c lclnt_funcs1.c dgram.c -o lclnt1
$ ./lclnt1 &./lclnt1 &./lclnt1 &./lclnt1 &

        SERVER:GOT:HELO 25912(10.200.75.200;1053)
        SERVER:SAID:TICK 25912.0(10.200.75.200;1053)
CLIENT[25912]:got ticket 25912.0
SuperSleep version 1.0 Running - Licensed Software
        SERVER:GOT:HELO 25913(10.200.75.200;1054)
        SERVER:SAID:TICK 25913.1(10.200.75.200;1054)
CLIENT[25913]:got ticket 25913.1
SuperSleep version 1.0 Running - Licensed Software
        SERVER:GOT:HELO 25915(10.200.75.200;1055)
        SERVER:SAID:TICK 25915.2(10.200.75.200;1055)
CLIENT[25915]:got ticket 25915.2
SuperSleep version 1.0 Running - Licensed Software
        SERVER:GOT:HELO 25914(10.200.75.200;1059)
        SERVER:SAID:FAIL no tickets available (10.200.75.200;1059)
CLIENT[25914]:Could not get ticket FAIL no tickets available (10.200.75.200;1059)
        SERVER:GOT:GBYE 25912.0(10.200.75.200;1053)
        SERVER: SAID:THNX See ya! (10.200.75.200;1053)
CLIENT[25912]:released ticket OK
        SERVER:GOT:GBYE 25913.1(10.200.75.200;1054)
        SERVER: SAID:THNK See ya! (10.200.75.200;1054)
CLIENT[25913]:released ticket OK
```

```

SERVER: GOT:GBYE 25915.2(10.200.75.200:1055)
SERVER: SAID:THNX See ya! (10.200.75.200:1055)
CLIENT[25915]:released ticket OK

```

实际运行的程序可能有非常不同的结果。尽管如此，从中可以看到服务器如何接收请求、给出票据以及客户端如何获取票据并开始工作的；可以看到进程 25914 没有得到票据。因为在该进程出现时，所有的票据都已经被占用了，而之前进程 25915 却得到了一张票据。如果运行该程序多次，可能会看到不同的结果。

13.6.4 进一步的工作

版本 1 的许可证服务器可以很好地工作了：服务器处理请求并且维持持有票据的进程列表。客户可以从服务器得到票据。现在为止一切都正常。看起来这非常理想。不过现实世界并不总这样美好，软件及其使用者并不完全是您所期望的那样。可能出现什么错误呢？该如何处理这些错误呢？

13.7 处理现实的问题

这里的许可证服务器能很好地工作，前提是所有的进程是正常工作的。有时，软件可能运行出错。如果 SuperSleep 程序被另外一个用户杀死了，或者程序发生段存取错误而被内核杀死了，该如何呢？对于它所占用的票据该如何处理呢？如果服务器崩溃了怎么办？在服务器重启后又将发生什么呢？

现实世界中的程序必须能够处理异常崩溃。这里考虑两种情形：客户端崩溃和服务器崩溃。

13.7.1 处理客户端崩溃

如果客户端崩溃，客户将不会归还票据，如图 13.8 所示。

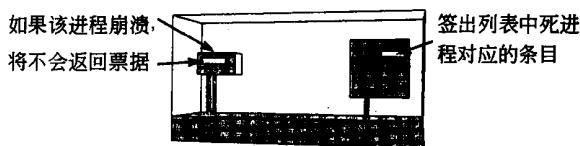


图 13.8 客户不归还票据

在出租车公司里，雇员可能辞职、被解聘、回家或死亡，但仍持有公司轿车的钥匙。这些对公司造成什么影响呢？签出列表指明票据仍被占用。其他进程就不能得到该票据。但是，如果有足够的进程崩溃，签出列表就可能虽然满了，但此时却没有运行的客户程序。

钥匙管理员通过给持有钥匙的人打电话，就可以收回钥匙。他可以定期地浏览签出列表，然后给每个司机打电话，“你仍在使用车吗？”如果无人响应，管理员把他的名字从签出列表中划去。管理员检查的频率越高，签出列表的精确性就越高。

许可证服务器可以使用相同的技术,定期检查票据数组,确认其中的每个进程是否还活着?如果某个进程已经不存在了,服务器可以把该进程从数组中去除,释放其占用的票据。检查程序运行的越频繁,数组的精确性越高。

1. 收回丢失的票据: 调度

服务器中如何增加收回票据的代码?如何调用这些代码?服务器必须实现两个独立的操作:等待客户的请求,同时周期性地收回丢失的票据。而调度行为是简单的:只要使用alarm 和 signal 技术来周期地调用一个函数。在前面章节的移动文字例子中,使用了这种技术。修订的程序流程如图 13.9 所示。

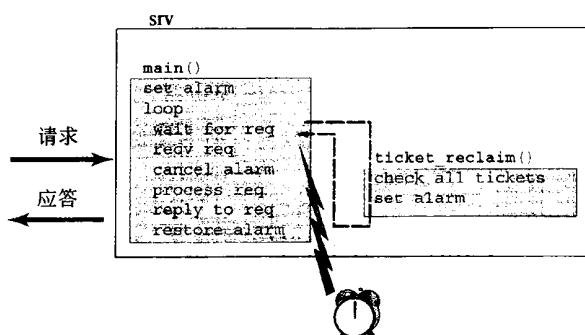


图 13.9 使用 alarm 来调度票据消除程序

在设计需同时处理两件事情的程序时,必定要考虑函数之间的冲突。如果服务器正在处理客户请求的同时,被 SIGALARM 信号触发调用回收丢失的票据的函数,会产生问题吗?这两个处理函数共享变量或者数据结构吗?显然是的,发放票据需要修改签出列表,而收回票据也需要修改签出列表。这种冲突可能会破坏数据的一致性吗?该问题留作课后练习。考虑到安全性,在处理请求的时候关闭 alarm。

2. 收回丢失的票据: 编程

服务器希望能回收已经不存在进程的票据。那么如何判断进程是否还活着呢?可以用 popen 来运行 ps,然后从 ps 的输出中查找 PID,以确定持有票据的 PID 是否存在。另一种快速简洁的方法是使用 kill 系统调用的特殊功能。

可以通过给进程发送编号为 0 的信号以确定它是否存在。如果进程不存在,内核将不会发送信号,而是返回错误并设置 errno 为 ESRCH。在 ticket_reclaim 中使用了该特征,该函数在 lserv_funcs2.c 文件中:

```

#define RECLAIM_INTERVAL 60 /* reclaim every 60 seconds */
*****
* ticket_reclaim
* go through all tickets and reclaim ones belonging to dead processes
* Results: none
*/
void ticket_reclaim()
{

```

第 13 章 基于数据报(Datagram)的编程：编写许可证服务器^①

• 409 •

```

int i;
char tick[BUFSIZ];

for(i = 0; i < MAXUSERS; i++) {
    if((ticket_array[i] != TICKET_AVAIL) &&
       (kill(ticket_array[i], 0) == -1) && (errno == ESRCH)) {
        /* Process is gone - free up slot */
        sprintf(tick, "%d %d", ticket_array[i], i);
        narrate("freeing", tick, NULL);
        ticket_array[i] = TICKET_AVAIL;
        num_tickets_out--;
    }
}
alarm(RECLAIM_INTERVAL);      /* reset alarm clock */
}

```

接下来，在 main 函数中增加调度回收票据的函数，并在正常操作中关闭 alarm。修改过的 main 在文件 lserv2.c 中：

```

int main(int ac, char *av[])
{
    struct sockaddr client_addr;
    socklen_t addrlen = sizeof(client_addr);
    char buf[MSGLEN];
    int ret, sock;
    void ticket_reclaim();           /* version 2 addition */
    unsigned time_left;

    sock = setup();
    signal(SIGALRM, ticket_reclaim); /* run ticket reclaimer */
    alarm(RECLAIM_INTERVAL);        /* after this delay */

    while(1) {
        addrlen = sizeof(client_addr);
        ret = recvfrom(sock, buf, MSGLEN, 0, &client_addr, &addrlen);
        if (ret != -1) {
            buf[ret] = '\0';
            narrate("GOT:", buf, &client_addr);
            time_left = alarm(0);
            handle_request(buf, &client_addr, addrlen);
            alarm(time_left);
        }
        else if (errno != EINTR)
            perror("recvfrom");
    }
}

```

通过上面的修改，许可证服务器就可以周期性地检查票据了。确实需要这样周期性地检查吗？为什么不只在票据列表满了并且有客户的请求被拒绝的时候检查呢？这样会更好吗？

13.7.2 处理服务器崩溃

服务器崩溃通常有两个严重的后果。首先，签出列表丢失，失去进程持有票据的记录。其次，新客户不可以再运行，因为分发许可证的程序已经不存在。最简单的解决方法是重新启动服务器，如图 13.10 所示。

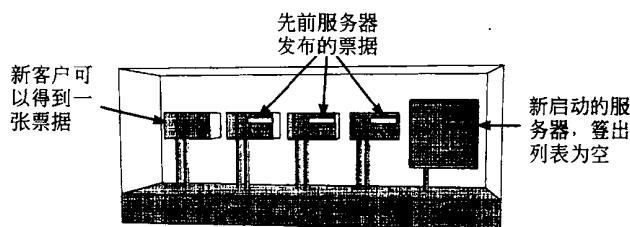


图 13.10 服务器在崩溃后重启

重启服务器使得新的客户可以运行，但会带来两个新的问题。

首先，重启服务器的票据数组是空的；服务器含有新的未被取走的票据列表。崩溃的服务器的票据数组可能已经满了，而重启服务器仍给其他客户发送许可。这样重复地关闭服务器，再重启服务器就像印钞机一样可以产生更多的票据。

其次，持有旧的服务器的票据的客户在归还票据时，将会被认为是伪造票据。

1. 票据验证

上述问题的一个解决方法是票据验证。票据验证表示每个客户周期性地向服务器发送票据的副本。客户发送数据包，对服务器说：“这是我的票据。是合法的吗？”，如图 13.11 所示。



图 13.11 客户验证票据

票据含有数组编号和 PID。服务器检查签出列表。如果该项为空，服务器可以认为该票据是自己先前的实例赋予的。服务器将会把该票据加到列表中。逐步地，客户提供票据来验证，签出列表被重新填入。

服务器重建签出列表解决了表丢失问题，但可能导致其他问题。如果一个新的客户在表重建好之前，请求票据，服务器可能分发一个已经给予其他客户的票据给该客户。当持有旧的票据的客户对其票据进行验证时，服务器会拒绝它。

第 13 章 基于数据报(Datagram)的编程：编写许可证服务器^①

• 411 •

另一种方法是服务器拒绝表中没有的所有的票据。持有被拒票据的客户尝试再申请新的。这种方法是否更好？

2. 协议中增加验证

票据验证是协议中的一个新的事务：

```
CLIENT:VALD tickid
SERVER:GOOD or FAIL invalid ticket
```

这里必须改变客户和服务器以支持验证。

3. 客户端增加验证

客户端增加验证，需要编写一个函数并在主函数中调用它，其流程如图 13.12 所示。

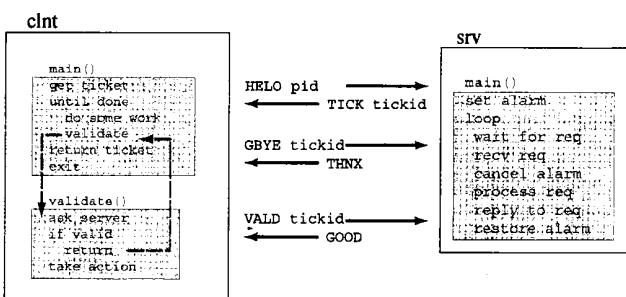


图 13.12 客户定期验证票据

客户可以根据系统的需要以一定的间隔定期验证票据，可以设置一个计时器来定期验证。如果客户是一个电子制表程序，验证可以在一定量的计算结束后进行。一个许可证服务器会响应验证请求。

SuperSleep 客户程序可以把 10 秒的睡眠时间分割成两个 5 秒，在这期间进行验证。这作为课后练习。

4. 服务器端增加票据验证

服务器端增加验证需要做两处改动。改动后的程序位于新的文件 lserv_funcs2.c 中。首先，增加一个函数来验证票据：

```
*****
* do_validate
* Validate client's ticket
* IN msg_p      message received from client
* Results: ptr to response
* NOTE: return is in static buffer overwritten by each call.
*/
static char * do_validate(char * msg)
{
    int pid, slot;          /* components of ticket */

    /* msg looks like VALD pid.slot - parse it and validate */
}
```

• 412 •

Unix/Linux 编程实践教程

```

if (sscanf(msg + 5, "%d %d", &pid, &slot) == 2 && ticket_array[slot] == pid)
    return("GOOD Valid ticket");

/* bad ticket */
narrate("Bogus ticket", msg + 5, NULL);
return("FAIL invalid ticket");
}

```

其次, handle_request 中增加更多的判断。

```

handle_request(char * req, struct sockaddr_in * client, socklen_t addlen)
{
    char * response;
    int ret;

    /* act and compose a response */
    if (strncmp(req, "HELO", 4) == 0)
        response = do_hello(req);
    else if (strncmp(req, "GBYE", 4) == 0)
        response = do_goodbye(req);
    else if (strncmp(req, "VALD", 4) == 0)
        response = do_validate(req);
    else
        response = "FAIL invalid request";

    /* send the response to the client */
    narrate("SAID:", response, client);
    ret = sendto(sd, response, strlen(response), 0, client, addlen);
    if (ret == -1)
        perror("SERVER sendto failed");
}

```

13.7.3 测试版本 2

现在可以编译和测试新版本的客户和服务器了。测试包含了杀死客户和服务器以及重启客户和服务器。试观察输出中的进程 ID 和消息。为了便于测试, 客户睡眠时间为两个 15 秒的间隔, 服务器每 5 秒尝试回收票据。结果如下:

```

$ cc lserv2.c lserv_funcs2.c dgram.c -o lserv2
$ cc lclnt2.c lclnt_funcs2.c dgram.c -o lclnt2
$ ./lserv2          # 启动 1 个服务器
[1]30804
$ ./lclnt2 & ./lclnt2 & ./lclnt2 &  # 启动 3 个客户端
[2]30805
[3]30806

```

第 13 章 基于数据报(Datagram)的编程：编写许可证服务器^①

• 413 •

```
[4]30807
$ SERVER: GOT; HELO 30805 (10.200.75.200;1085)
      SERVER:SAID:TICK 30805.0 (10.200.75.200;1085)
CLIENT [30805]:got ticket 30805.0
      SuperSleep version 1.0 Running - Licensed Software
          SERVER: GOT; HELO 30806 (10.200.75.200;1086)
              SERVER:SAID:TICK 30806.1 (10.200.75.200;1086)
CLIENT [30806]:got ticket 30806.1
      SuperSleep version 1.0 Running - Licensed Software
          SERVER: GOT; HELO 30807 (10.200.75.200;1087)
              SERVER:SAID:TICK 30807.2 (10.200.75.200;1087)
CLIENT [30807]:got ticket 30807.2
      SuperSleep version 1.0 Running - Licensed Software
$ kill 30806 #kill 一个客户
[3]- Terminated ./lclnt2
      SERVER:freeing 30806.1
      SERVER: GOT; VALD 30805.0 (10.200.75.200;1085)
          SERVER: SAID: GOOD Valid ticket (10.200.75.200;1085)
CLIENT [30805]:Validated ticket; GOOD Valid ticket
      SERVER: GOT : VALD 30807.2 (10.200.75.200;1087)
          SERVER: SAID: GOOD Valid ticket (10.200.75.200;1087)
CLIENT [30807]: Validated ticket; GOOD Valid ticket
$ kill 30804 #kill 服务器
[1]Terminated ./lserv2
$ ./lserv2 #启动新的服务器
[5]30808
$

      SERVER:GOT; GBYE 30805.0 (10.200.75.200;1085)
      SERVER: Bogus ticket 30805.0
      SERVER: SAID: FAIL invalid ticket (10.200.75.200;1085)
CLIENT[30805]:release failed invalid ticket
      SERVER:GOT; GBYE 30807.2 (10.200.75.200;1087)
      SERVER: Bogus ticket 30807.2
      SERVER: SAID: FAIL invalid ticket (10.200.75.200;1087)
CLIENT[30807]:release failed invalid ticket
$ ./lclnt2 #启动一个新的客户
      SERVER: GOT; HELO 30809 (10.200.75.200;10857
          SERVER:SAID:TICK 30809.0 (10.200.75.200;1087)
CLIENT [30809]:got ticket 30809.0
      SuperSleep version 1.0 Running - Licensed Software
          SERVER: GOT : VALD 30809.0 (10.200.75.200;1087)
              SERVER: SAID: GOOD Valid ticket (10.200.75.200;1087)
CLIENT [30809]: Validated ticket; GOOD Valid ticket
      SERVER:GOT; GBYE 30809.0 (10.200.75.200;1087)
```

• 414 •

Unix/Linux 编程实践教程

```

SERVER: SAID: THNX See ya! (10.200.75.200,1087)
CLIENT [30809]:release ticket OK
[2] Done          ./lclnt2
[4]- Done         ./lclnt2
$ ps
PID   TTY      TIME      CMD
23509 pts/3    00:00:00  bash
30808 pts/3    00:00:00  lserv2
30810 pts/3    00:00:00  ps
$
```

看起来还不错。不妨试一下上述程序，观察其中的交互过程。

13.8 分布式许可证服务器

许可证服务器和被许可程序通过 socket 进行通信，socket 可以连接不同主机上的进程。理论上，与 Web 服务器和客户运行在不同主机上类似，这里的客户可以运行在一台机器上，而服务器运行在另一台机器上。当它们运行在不同的机器上时，会有问题吗？是的。

- 问题 1：重复的进程 ID

进程 ID 在一台机器上是惟一的，但在不同主机上的进程可能拥有相同的进程 ID。图 13.13 说明的情况不含有任何错误且很常见。

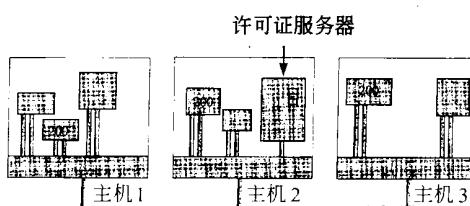


图 13.13 跨网络 PID 不惟一

存放票据的表中含有 PID 和票据编号。在图 13.13 的情况下，许可证服务将认为给同一个进程分配了 3 张票。每个进程只要一张票就可以运行，所以这是错误的。请求更多的票据，可以被认为是客户端的一个漏洞。

这里可以扩展票据表项的格式和内容，使其包含标识运行程序的主机从而解决重复 PID 问题。

- 问题 2：回收票据

服务器通过调用 kill(pid,0) 命令向客户回收票据。kill(pid, 0) 发送信号 0 给持有票据的进程。通过修订过的票据表，服务器现在可以知道客户运行在哪台主机上。

但是，服务器不能给其他机器上的进程发送信号，如图 13.14 所示。如果许可证服务器想给主机 3 上的进程发送信号，服务器必须产生主机 3 上的请求。

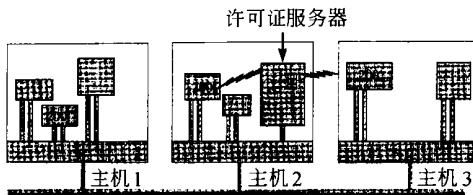


图 13.14 进程不能给其他主机发送信号

为什么不在每台机器上都运行一个服务器的实例？如图 13.15 所示，每个本地的服务器可以监控丢失的票据。

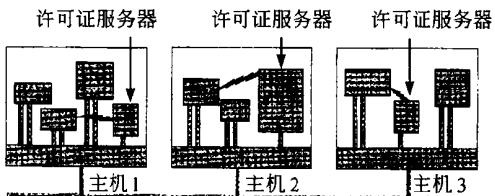


图 13.15 运行本地复制的 lserv

本地服务器解决了向主机发送信号的问题，但是又带来新的问题：哪个服务器发布票据？主服务器如何和本地服务器通信？客户把票据发给谁验证？

- 问题 3：主机崩溃

如果其中的一台机器停止运行，将会发生什么？主服务器如果还在运行的话，如何收回票据？客户端程序如何验证票据？如果运行主服务器的主机停止运行，谁来分发票据？

如何建立一个分布式许可证系统来同时支持多台机器？这里有 3 种方法。试考虑它们的设计细节以及优缺点，并考虑当客户、服务器、计算机或者网络崩溃时每个方案的后果。

- 方法 1：客户端服务器和中央服务器通信

每台机器都有一个本地服务器，就像本文编写的那样。每个客户跟本地的服务器通信。本地服务器把请求转发给中央服务器。中央服务器返回票据或者拒绝。本地服务器记录并把应答转发给客户。本地服务器也可以强制执行一些对本地客户的限制，例如该机器上可以运行的程序实例数，或者程序可以运行的时刻。

- 方法 2：每个客户都和中央服务器通信

客户直接给特定主机上的服务器发送请求。本地服务器运行在每个主机上，但这些服务器不和客户通信，它们在重新声明票据的时候作为中央服务器的代理。

- 方法 3：客户服务器和客户服务器通信

每台机器都有本地服务器，每个客户跟本地服务器通信。没有中央服务器。所有的本地服务器间互相通信。每次一个客户请求时，本地服务器询问其他所有的服务器目前已经用掉了多少张票。如果所用票据数小于所允许的总数，本地服务器分配一张票据给客户。

13.9 Unix 域 socket

本章的许可证服务器中的 socket 使用标准的主机 ID 和端口号地址系统。使用这些 Internet 地址，服务器可以接收本地的乃至更大网络上机器的客户请求。

在上述的两种分布式许可证服务器模型中，客户只需和同一台主机上的服务器通信。socket 只能用在仅限于主机内部的通信中吗？

13.9.1 文件名作为 socket 地址

有两种连接：流连接和数据报连接，也有两种 socket 地址：Internet 地址和本地地址。Internet 地址包含主机 ID 和端口号。本地地址通常叫做 Unix 域地址，它是一个文件名（例如 /dev/log、/dev/printer 或 /tmp/lserversock），没有主机和端口号。

两个 socket 名 /dev/log 和 /dev/printer 被用在很多 Unix 系统中。/dev/log 被 syslogd 服务器所使用。想要记录日志的程序只要给地址为 /dev/log 的 socket 发送数据报就行了。地址 /dev/printer 被一些打印系统使用。

13.9.2 使用 Unix 域 socket 编程

为了学习 Unix 域 socket 的客户/服务器编程，这里编写了一个日志系统。文件 wtmp 就是日志系统的一个实例。文件 wtmp 记录系统的所有登录、退出以及连接。日志系统被保证安全和系统维护的程序所使用，用来记录一些可疑行为。日志服务器是一个抄写员；客户发送信息给服务器，服务器将这些信息保存到只有自己可以修改的文件中。日志服务器可以用任何格式保存该文件，客户并不知道这些细节。

这里的日志服务器使用 Unix 域 socket 地址。只有同一台主机上的客户才能发消息给它。下面是客户和服务器的代码。服务器先创建 socket，然后绑定地址：

```
*****  
* logfiled.c - a simple logfile server using Unix Domain Datagram Sockets  
* usage: logfiled >>logfilename  
*/  
  
# include <stdio.h>  
# include <sys/types.h>  
# include <sys/socket.h>  
# include <sys/un.h>  
# include <time.h>  
  
# define MSGLEN 512  
# define oops(m,x) { perror(m); exit(x); }  
# define SOCKNAME "/tmp/logfilesock"  
  
int main(int ac, char * av[])  
{
```

第13章 基于数据报(Datagram)的编程：编写许可证服务器^①

• 417 •

```

int          sock;                      /* read messages here */
struct       sockaddr_un addr;           /* this is its address */
socklen_t    addrlen;
char         msg[MSGLEN];
int          l;
char         sockname[] = SOCKNAME ;
time_t       now;
int          msgnum = 0;
char         *timestr;

/* build an address */
addr.sun_family = AF_UNIX;             /* note AF_UNIX */
strcpy(addr.sun_path, sockname);       /* filename is address */
addrlen = strlen(sockname) + sizeof(addr.sun_family);

sock = socket(PF_UNIX, SOCK_DGRAM, 0);   /* note PF_UNIX */
if ( sock == -1 )
    oops("socket", 2);

/* bind the address */
if ( bind(sock, (struct sockaddr *) &addr, addrlen) == -1 )
    oops("bind", 3);

/* read and write */
while(1)
{
    l = read(sock, msg, MSGLEN);        /* read works for DGRAM */
    msg[l] = '\0';                     /* make it a string */
    time(&now);
    timestr = ctime(&now);
    timestr[strlen(timestr)-1] = '\0'; /* chop newline */
    printf("[ %5d] %s %s\n", msgnum++, timestr, msg);
    fflush(stdout);
}
}

```

这里仍使用 socket 和 bind 来创建服务器 socket。socket 的类型是 SOCK_DGRAM，地址族是 PF_UNIX^①。socket 地址是文件名。使用 read 而不是 recvfrom，因为不需要应答。下面是客户端程序：

```

*****
* logfilec.c - logfile client - send messages to the logfile server
* usage: logfilec "a message here"
*****
```

^① PF_LOCAL 也许能替代 PF_UNIX。

• 418 •

Unix/Linux 编程实践教程

```

/*
# include  <stdio.h>
# include  <sys/types.h>
# include  <sys/socket.h>
# include  <sys/un.h>

#define      SOCKET      "/tmp/logfilesock"
#define      oops(m,x)   { perror(m); exit(x); }

main(int ac, char * av[])
{
    int             sock;
    struct sockaddr_un addr;
    socklen_t       addrlen;
    char            sockname[] = SOCKET ;
    char            * msg = av[1];

    if ( ac != 2 ){
        fprintf(stderr,"usage: logfilec 'message'\n");
        exit(1);
    }
    sock = socket(PF_UNIX, SOCK_DGRAM, 0);
    if ( sock == -1 )
        oops("socket",2);

    addr.sun_family = AF_UNIX;
    strcpy(addr.sun_path, sockname);
    addrlen = strlen(sockname) + sizeof(addr.sun_family) ;

    if ( sendto(sock,msg,strlen(msg), 0, &addr, addrlen) == -1 )
        oops("sendto",3);
}

```

这里使用 socket 函数来创建 socket，并且使用 sendto 发送消息。服务器接收消息，然后打印消息。消息前面是消息编号和时间。

下面是测试的情形：

```

$ cc logfiled.c -o logfiled
$ ./logfiled >> vistorlog
1500
$ cc logfilec.c -o logfilec
$ ./logfilec 'Nice system. Swell software!'
$ ./logfilec "Testing this log thing."
$ ./logfilec "Can you read this?"
$ cat vistorlog
[ 0] Mon Aug 20 18:25:34 2001 Nice system. Swell software!

```

```
[ 1] Mon Aug 20 18:25:44 2001 Testing this log thing.  
[ 2] Mon Aug 20 18:25:48 2001 Can you read this?
```

上述两个程序展示了如何使用 Unix 域 socket 以及日志服务器的基本思想。程序的另一个特征是实现了自动追加功能，但没有使用 O_APPEND。服务器接收消息，然后把消息追加到文件末尾。即使有多个客户同时发送消息，底层的 socket 机制会负责消息的序列化。

13.10 小结：socket 和服务器

socket 对于进程间的数据通信是强大的、万能的工具。这里学习了两种 socket 和两种 socket 地址。

域		
socket	PF_INET	PF_UNIX
SOCK_STREAM	连接的, 跨机器	连接的, 本地
SOCK_DGRAM	数据报, 跨机器	数据报, 本地

在前面的几章中，已经学习了使用这 4 种组合中的 3 种 socket 的项目。当考虑 Unix 的网络编程和准备设计自己的项目时，可以考虑使用这张表格中的技术。根据发送消息的类型以及消息发送的距离来选择最佳的技术。

小 结

1. 主要内容

- 数据报是从一个 socket 发送到另一个 socket 的短消息。数据报 socket 是不连接的，每个消息包含有目的地址。数据报(UDP)socket 更加简单、快速，给系统增加的负荷更小。
- 许可证服务器是用来对被许可程序实施许可证验证规则的。许可证服务器发布许可，以短消息的形式发送给客户。
- 许可证服务器必须记住哪个进程使用了哪张票据，必须维持一个内部的数据库。因此，许可证服务器不同于本书的 Web 服务器。
- 记录系统状态的服务器必须设计成可以处理服务器和客户端的崩溃事件。
- 有些许可证服务器为一个网络上的多个机器提供服务。有几种设计方法，各有优缺点。
- socket 可以有两种类型的地址：网络或本地。本地的 socket 地址叫做 Unix 域 socket 或名字 socket。这种 socket 使用文件名作为地址，只能在一台机器上交互数据。

2. 下一步的工作

本章学习了两种服务器处理多个请求的方法。许可证服务器接收数据报请求，并且一次一条地返回消息。Web 服务器接收数据流消息，并且使用 fork 同时应答所有请求。服务

器有另外的选择：一个单一进程可以使用线程的技术同时运行几个函数。下一章将学习有关线程的思想和技术。

3. 习题

- 13.1 在使用流 socket 的例子中，服务器给客户应答时，并没有使用客户端的地址。服务器是如何知道给哪个地址的客户发消息的？
- 13.2 进程 25915 是如何战胜进程 25914 得到票据的？考虑每个客户进程的创建和服务器端请求到达的操作序列。在多任务系统中，进程依次运行。进程在哪儿可以被中断从而得到测试的结果？
- 13.3 如何使用一个许可证服务器处理 2 个或更多的程序的请求？一种方法是修改协议使得每个请求包含所要运行的程序名。描述可以支持多程序协议的数据结构和程序逻辑。
- 13.4 当服务器正在处理客户请求时，如果函数 ticket_reclaim 被调用，是否会对票据列表存在潜在的破坏？考虑函数中每个更改数组和计数器的地方。在哪一点上数组的状态和计数器的值不一致？处理函数是如何修改数组和计数器的？这些值的一个未预测的改变对常规处理函数有何影响？
- 13.5 当进程被创建时，进程 ID 被分配给进程。考虑下面的时间序列：一个进程 ID 为 777 的客户得到票据后崩溃了。不久，一个不同的用户运行程序创建了一个新的进程，进程 ID 恰好也是 777。当 ticket_reclaim 程序运行时，它发现了进程 777。即使现在编号为 777 的进程是一个不相关的程序，并且不持有票据，分配给原来进程 777 的票据也不能被回收。当这种情形出现时，该如何处理？如何避免这类事件的发生。
- 13.6 一种防止票据数组丢失的方法是服务器将数据写到磁盘文件中。如果要适应这种备份机制，该如何改变服务器？假设客户会故意杀死服务器以得到更多的票据，那么这里的文件备份机制在此种情形下该如何工作？
- 13.7 持有早期版本票据的客户在等待了较长的时间后来验证票据，可能发现已经没有更多的可用票据了。为这种情形设计一种应答，使得客户不允许继续运行，因为这将破坏最大同时运行进程的数目，但要求客户不能突然退出。
- 13.8 参照本章中列出的问题，比较三种分布式许可证服务器模型。
- 13.9 使用 socket 时，可以用 write 和 sendto 来发送数据。阅读 send 和 sendmsg 的帮助手册，后面两个函数与前面的有什么区别？
- 13.10 轿车管理系统与数据报不只是比喻。设想每部轿车都装有 GPS 设备。一个计算机可以通过 modem 连接到 Internet，使得可以定位轿车的位置。设想轿车的启动不是受钥匙控制，而是通过一个磁卡的登录来控制。设计一个允许雇员登

录到系统来使用轿车，并且管理员可以跟踪司机行驶路线以及定位轿车位置的系统。

4. 编程练习

- 13.11 更改程序 dgrecv.c，要求不仅打印出发送者的地址和消息接收的时间，还要打印消息编号。消息标号从 0 开始。希望得到的输出如下：

```
dgrecv:got a message : testing 123
from: 10.200.75.200:1041
at :Sun Aug 19 10:22:27 EDT 2001
msg #:23
```

- 13.12 为 dgssend.c 增加客户程序 dgrecv2.c。

- 13.13 许可证服务器在一张表中存有客户拥有票据的信息。如果想要服务器打印这些信息，该如何操作？看到表的信息有助于排错或测试服务器。一种标准的和服务器通信技术是使用信号。

更改程序 lserv1 使得它在接收信号 SIGHUP 时，打印出表的内容。可以通过命令 kill -HUP serverpid 来测试该特征。

- 13.14 更改许可证服务器使得它只在一个客户的请求被拒绝时，才调用 ticket_reclaim 程序。该方法的优缺点各是什么？

- 13.15 更改 lclnt2.c 程序，使它睡眠 5 秒钟后验证票据。如果票据合法，客户再睡眠 5 秒钟，然后退出并归还票据。如果票据不合法，客户尝试请求另一个票据。如果成功，继续正常操作。如果失败，告诉用户许可证服务器出错然后再退出。

- 13.16 更改前面章节中编写的 shell 程序和 bounce 程序，使用许可证服务器。在哪增加票据验证过程？当服务器崩溃时，票据不可用了，该和用户说什么？

- 13.17 更改客户服务器代码使得票据包含有主机 IP 地址。如何改变票据列表？确信对验证函数也做了改变。

- 13.18 实现三种分布式许可证控制模型的一种。

- 13.19 日志系统的一个问题是其中的消息都是匿名的，更改该系统使得消息包含发送消息用户的用户名。

- 13.20 在日志服务器中使用 read。编写两个新版本的服务器，一个使用 recvfrom，另一个使用 recv。这些获取数据的方法有何不同？更详细的信息请阅读帮助手册。

- 13.21 如果许可证服务器和客户需要使用 Unix 域 socket，需要做何改变？解释客户为什么要使用 bind。

13.22 在第 1 章中,讨论了一个 Internet 桥牌游戏。在任何的分布式扑克游戏中,软件必须模拟一个扑克牌栈,以确保两个客户不会持有相同的扑克。

编写一对程序 cardd 和 cardc,使用数据报来处理一桌的扑克牌。启动时服务先洗牌,接下来客户从命令行启动,就可以从服务器获取扑克牌。例程运行如下:

```
$ cardc get 5  
4D AH 2D TD KC
```

结果显示客户得到了 5 张牌,一张方块 4,一张红桃 A,一张方块 2,一张方块 10,一张王。确保程序不会将相同的扑克牌发给两个用户机,并且协议要有途径表明 dealer 何时发完牌。考虑一下还有其他什么有用的事物可以作为协议的补充?

5. 项目

基于本章的内容,可以学习编写下面的 Unix 程序:

talk、rwho、流媒体服务器

8 应用层协议服务配置

8.1 实验要求

配置以下服务：

操作系统	服务	建议软件
Windows Server	DNS	系统自带
	HTTP	系统自带 IIS
	HTTPS	系统自带证书服务器
	FTP	Serv-U FTP
	SMTP,POP3,IMAP	系统自带或第三方
Linux Server	SSH (远程桌面和文件服务)	OpenSSH
	HTTP	Nginx
	SMB	Samba

8.2 操作步骤

请按以下步骤完成：

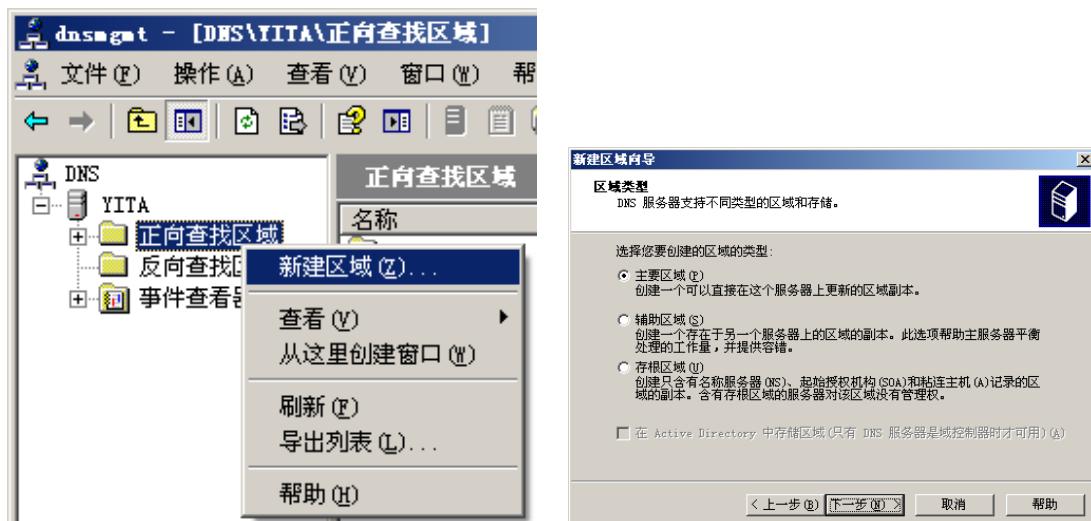
- 1、下载相关课件，阅读实验教程课本（附录1），上网搜集相关资料，视频和文字教程，记住实验要领；
- 2、按照描述在 Windows 或 Linux 操作系统下，配置服务。

8.3 附录一：服务配置示例

以下配置版本较为老旧，请搜索支持对应协议的新版本或新软件。

(1) DNS 服务器

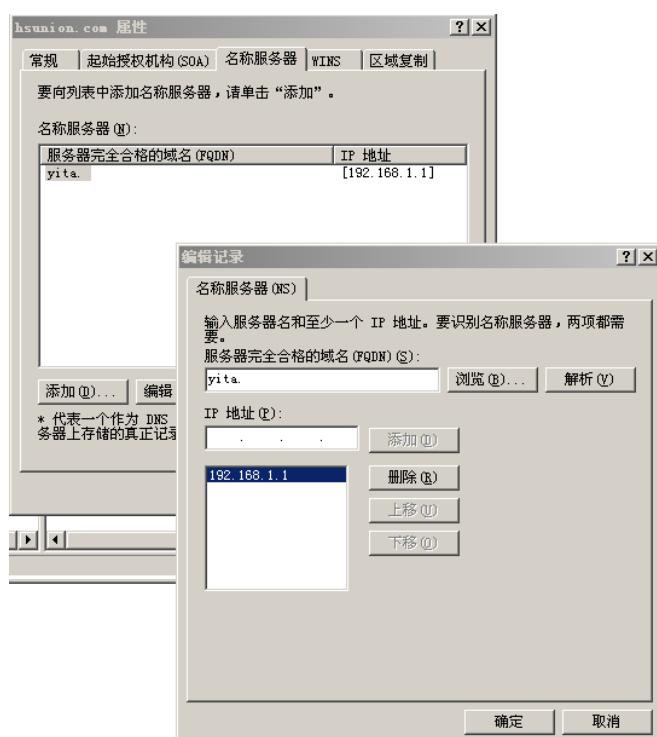
- 1、安装好 DNS 服务器：
- 2、“正向查找域”处单击鼠标右键：



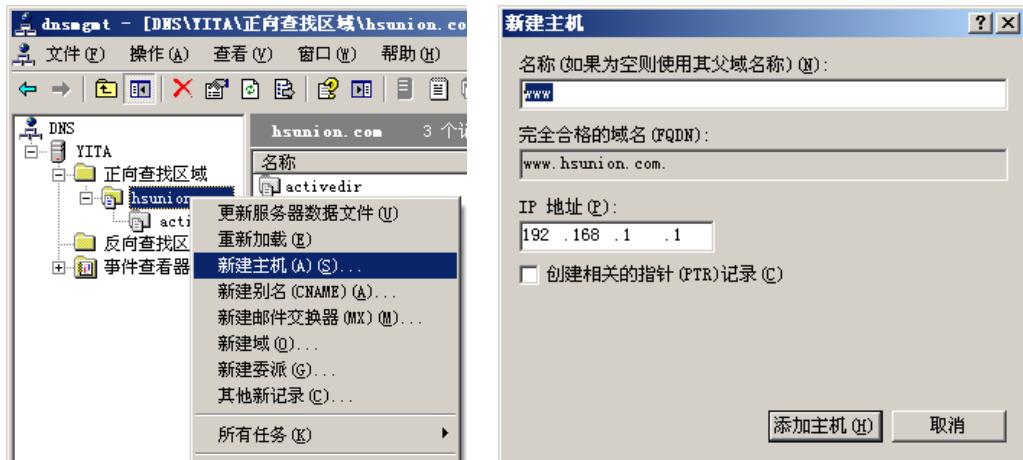
3、新建主要区域，输入域名“hsunion.com”



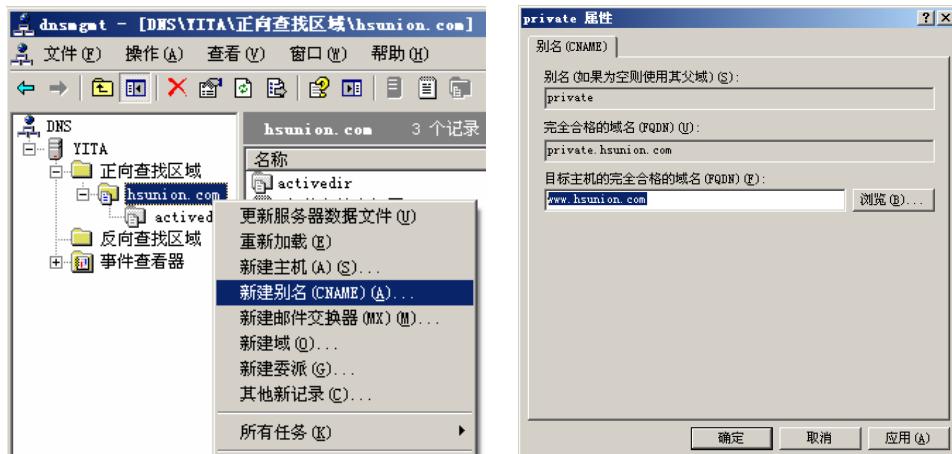
4、和 IP 对应起来，对应于 IP：192.168.1.1



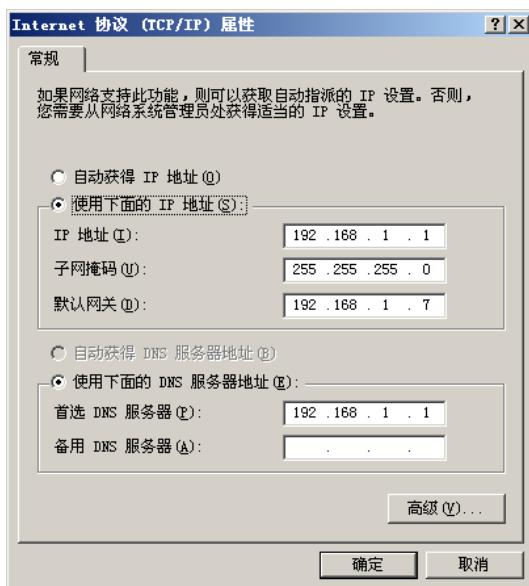
5、新建服务器主机



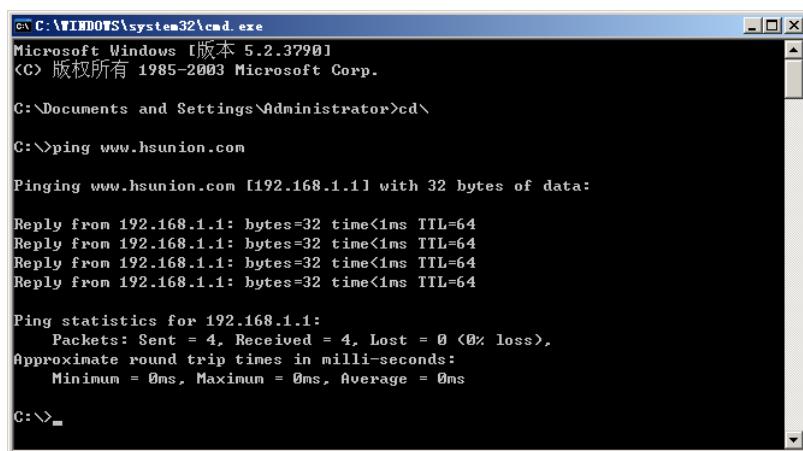
6、新建服务器别名



7、添加 DNS 服务器地址



8、测试该 DNS 是否配置成功



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 5.2.3790]
(C) 版权所有 1985-2003 Microsoft Corp.

C:\Documents and Settings\Administrator>cd\

C:\>ping www.hsunion.com

Pinging www.hsunion.com [192.168.1.1] with 32 bytes of data:

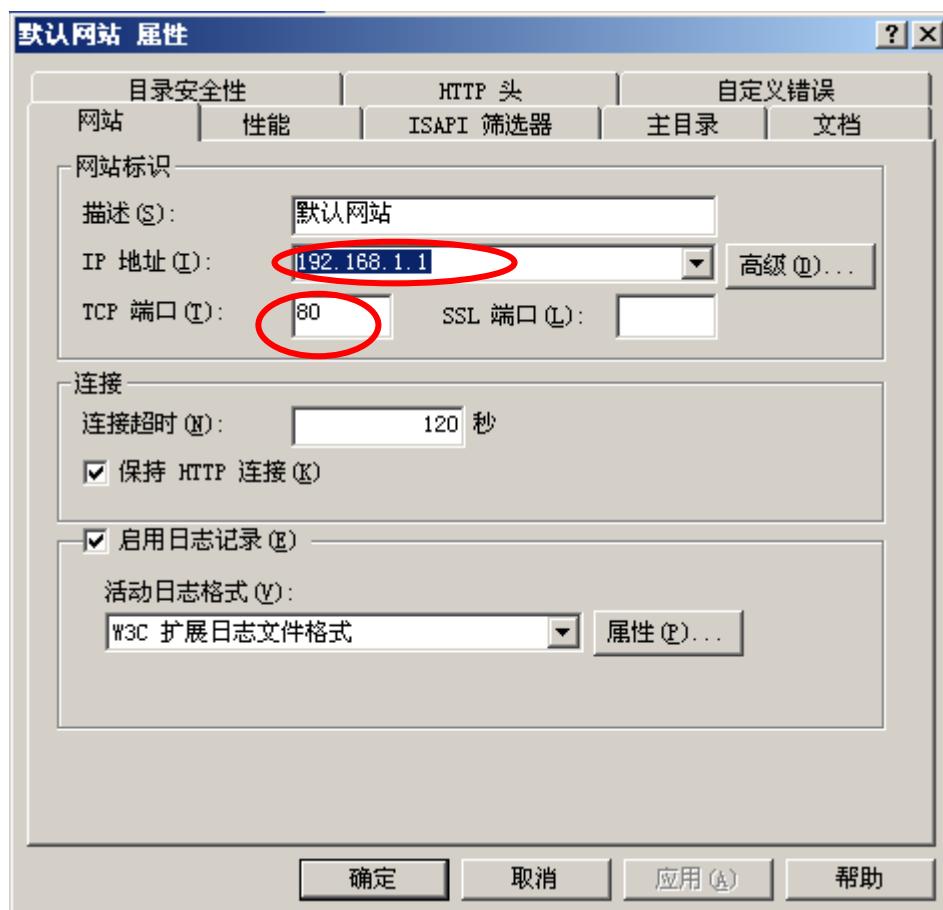
Reply from 192.168.1.1: bytes=32 time<1ms TTL=64

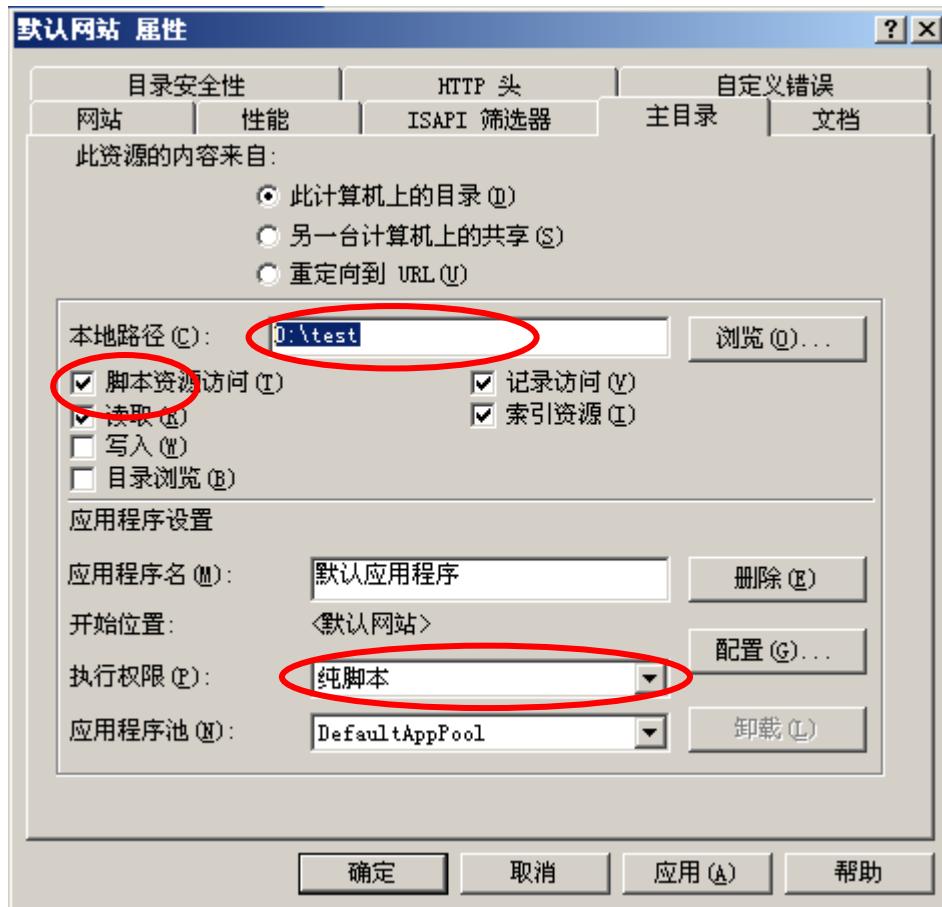
Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 <0% loss>
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>
```

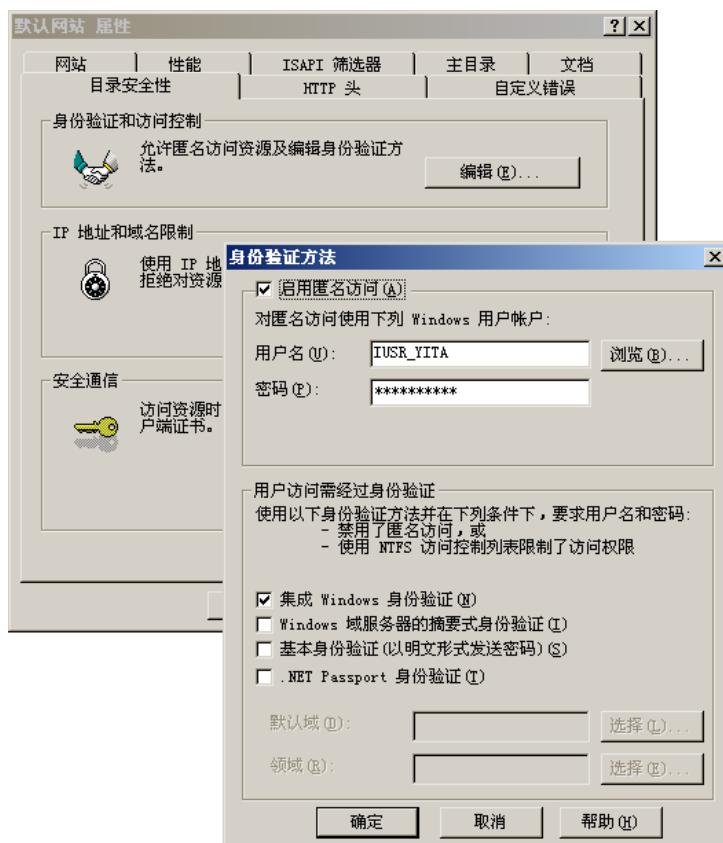
(2) Web 服务器

1、安装 Web 服务器，设定 IP 和端口，以及主目录

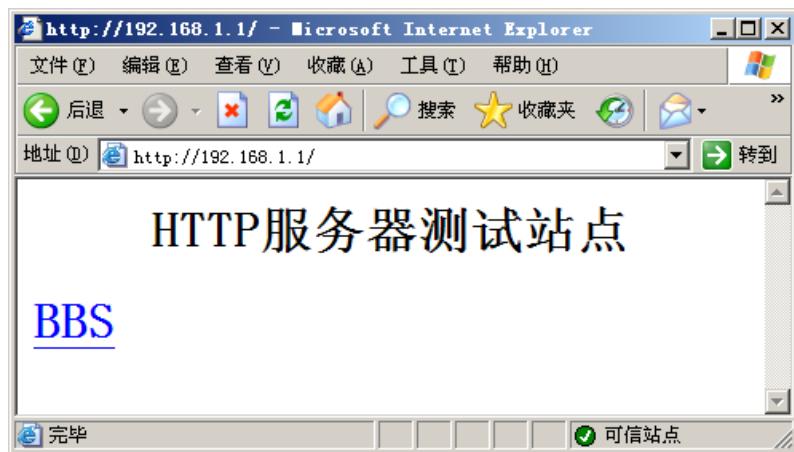




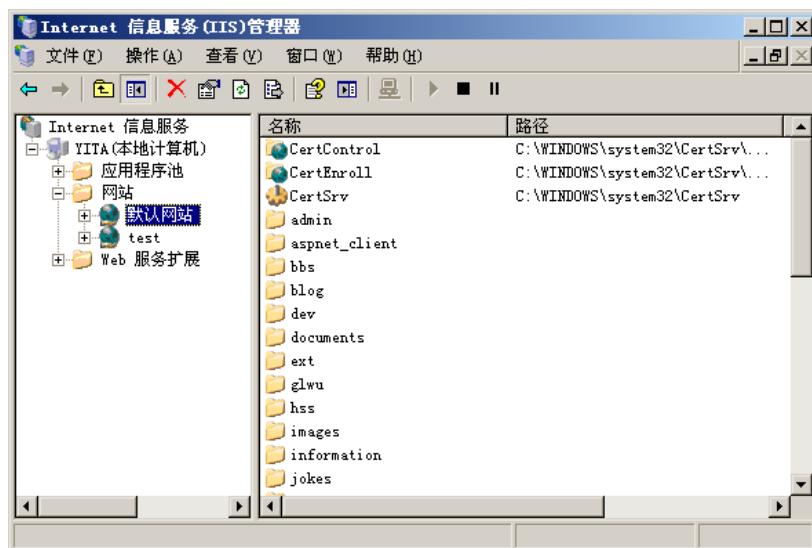
2、启用 IE 浏览器匿名访问：



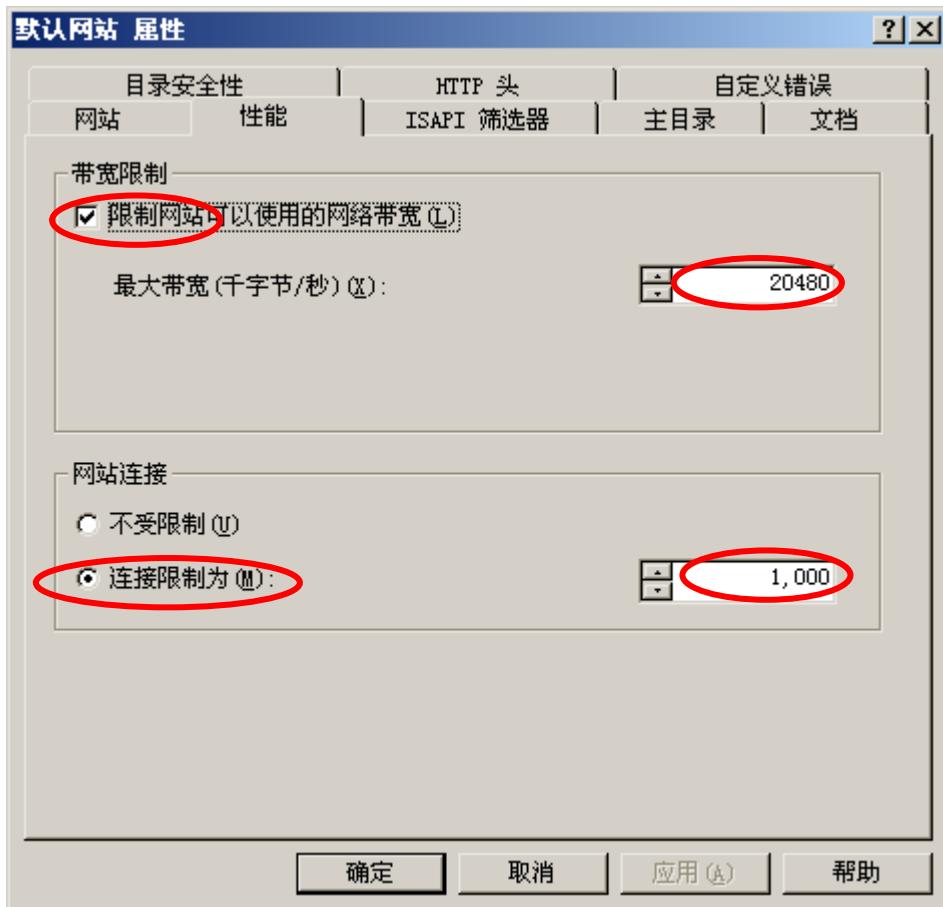
3、测试是否架构成功



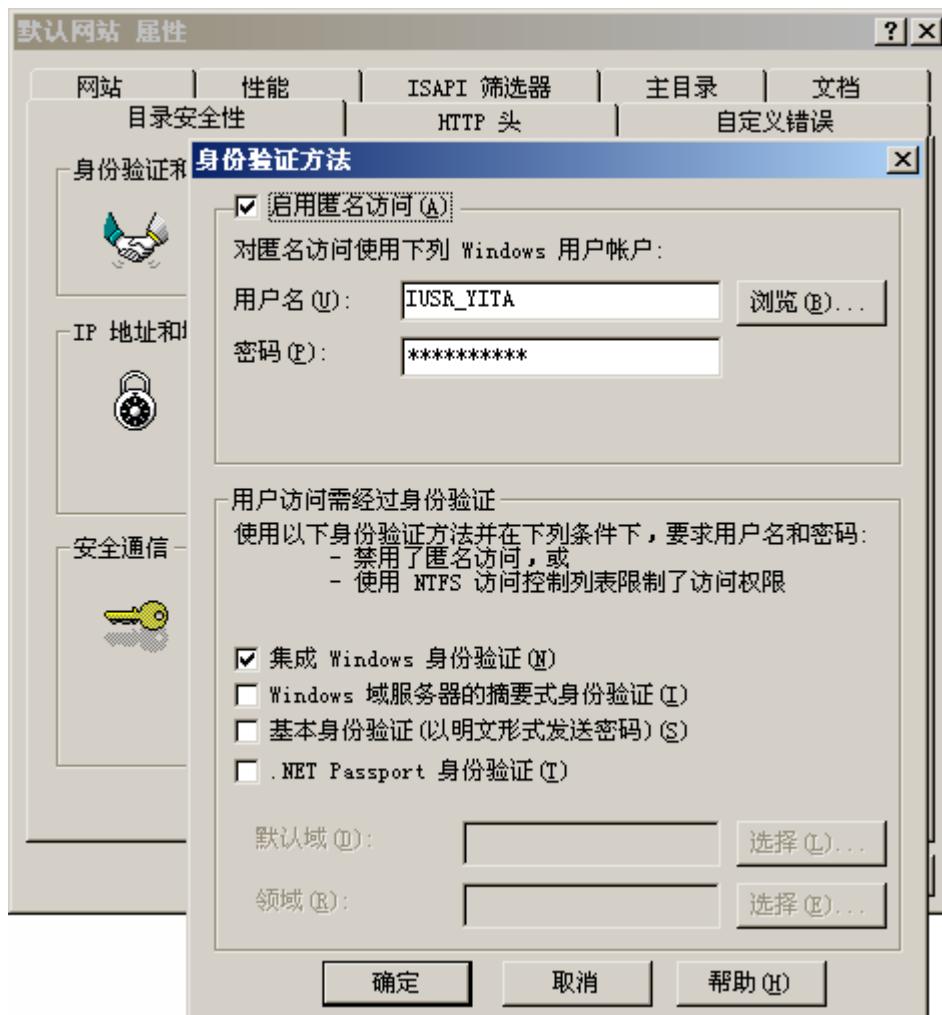
4、启动停止重启服务器



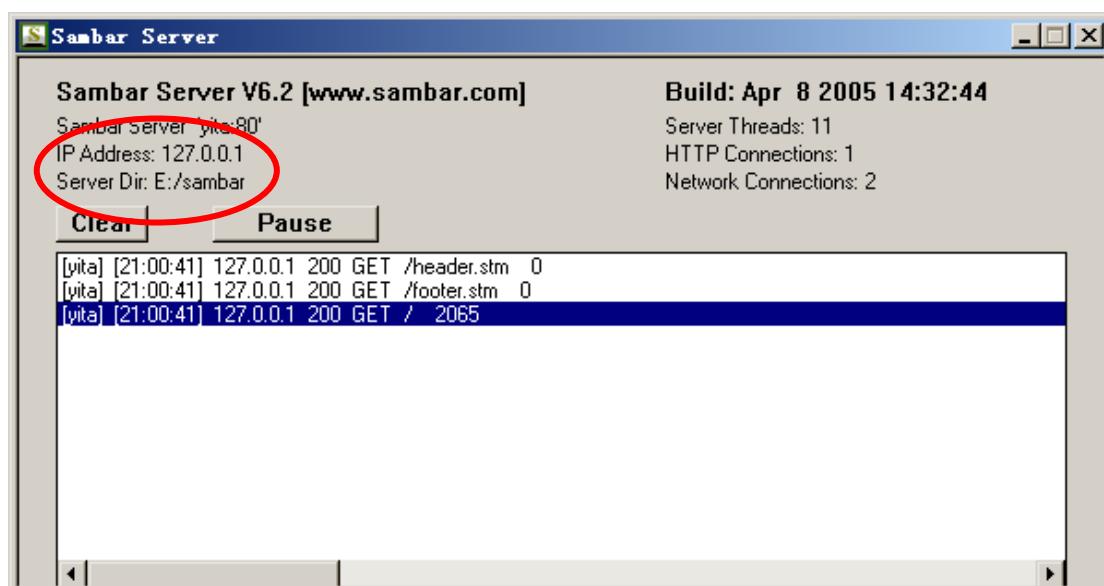
5、控制流量



6、安全性之凭密码访问



7、用 Sambar Server 做 HTTP 服务器



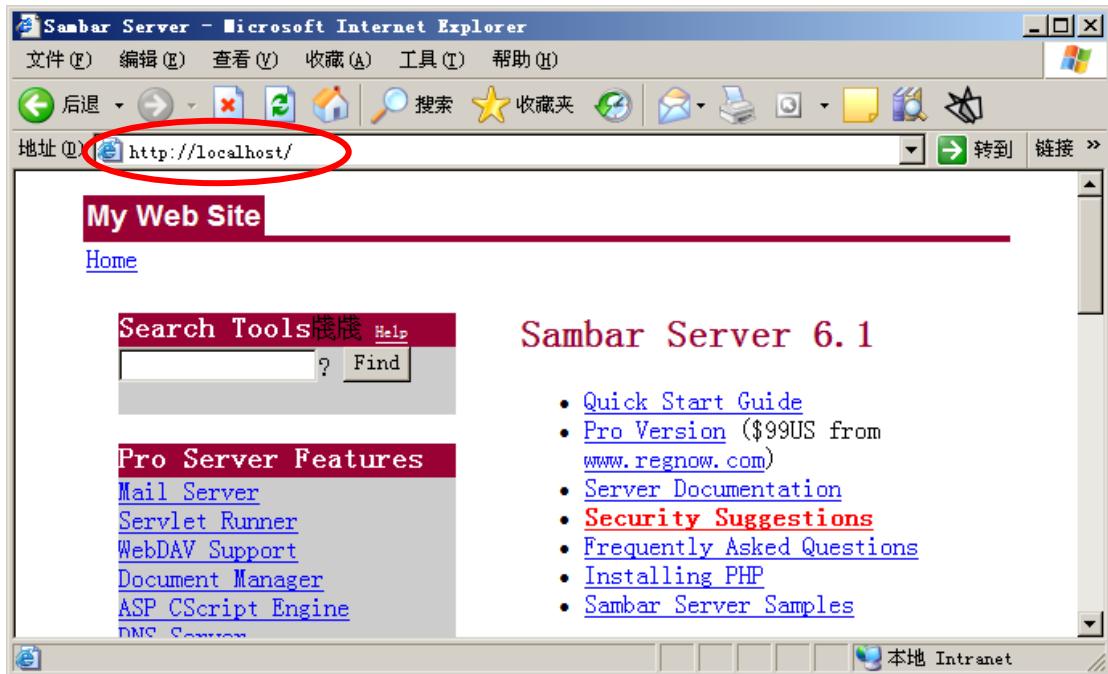
图中可以看出：

服务器 IP 地址：127.0.0.1

端口：80

服务器目录：E:\Sambar

8、验证启动成功

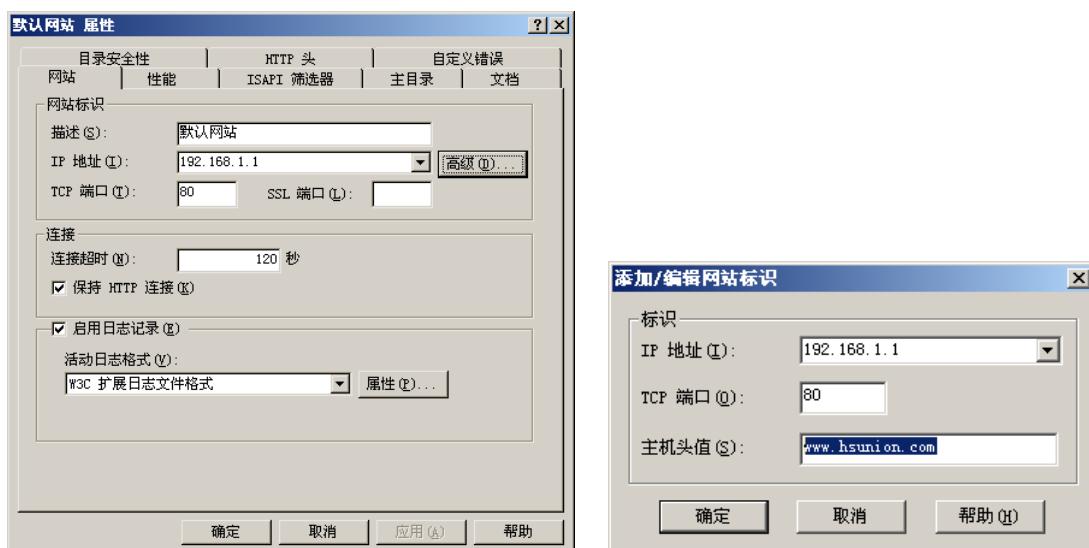


9、服务器的关闭和重启



(3) 虚拟主机技术

1、在“网站”标签页处单击“高级”。



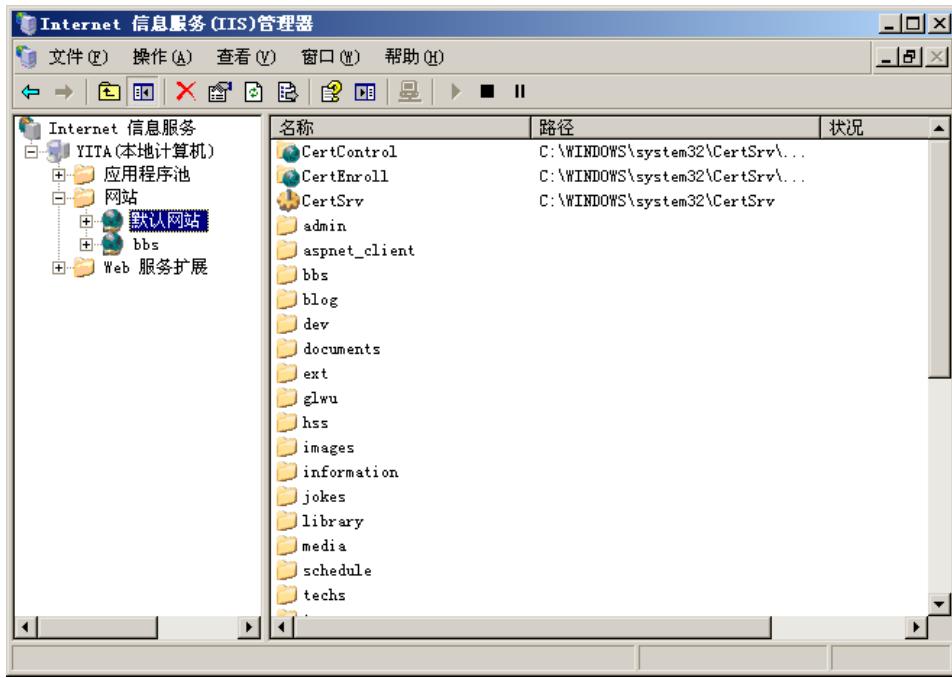
2、添加其他的站点，设置相同的 IP 和端口号，不同的主机头值



3、设置不同路径



4、设置成功



5、测试站点



结果：IP 和端口号相同（192.168.1.1:80），主机头不同，显示的页面不同。

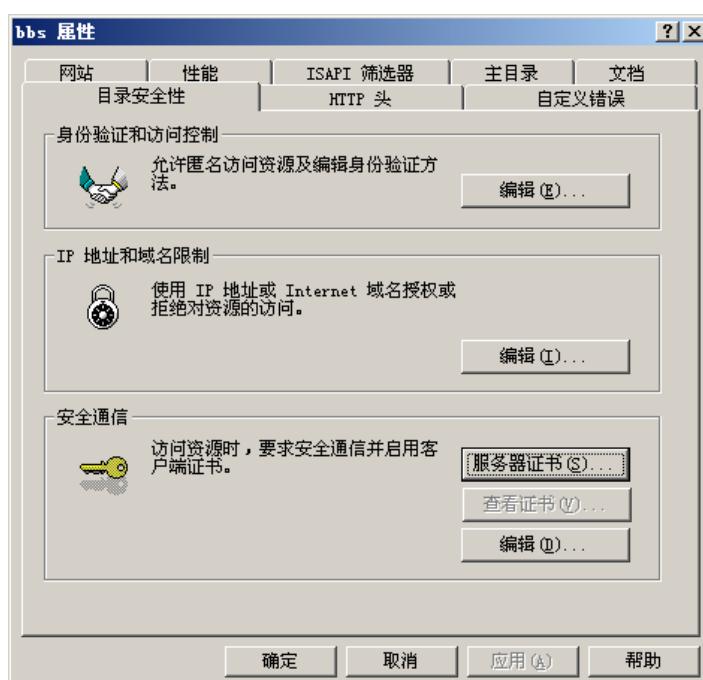
(4) 安全站点

1、打开站点，设置端口

打开 private.hsunion.com 的站点，设置端口为 443（443 是默认 https 端口）



2、申请服务器证书



3、申请证书



4、输入正确的信息



5、通过证书服务器介绍（看下一章所示）签发服务器证书

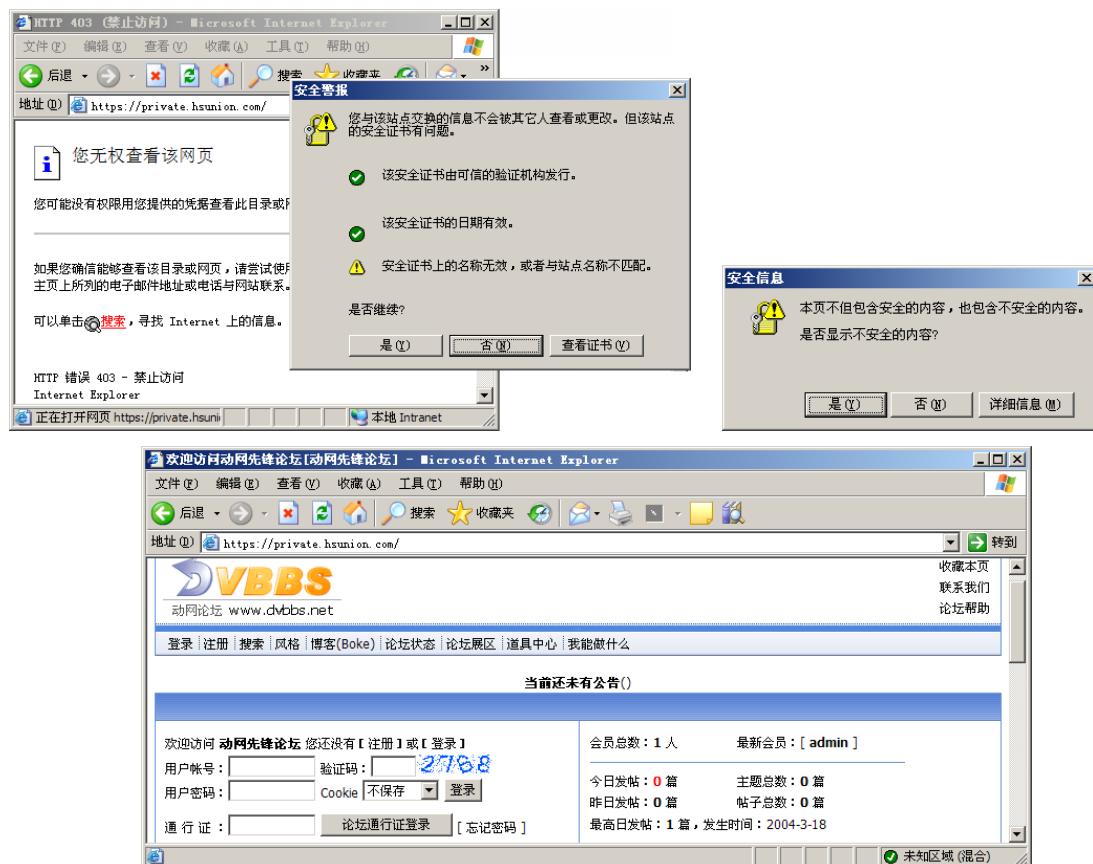
6、导入证书



7、通过“查看证书”可以看到该证书

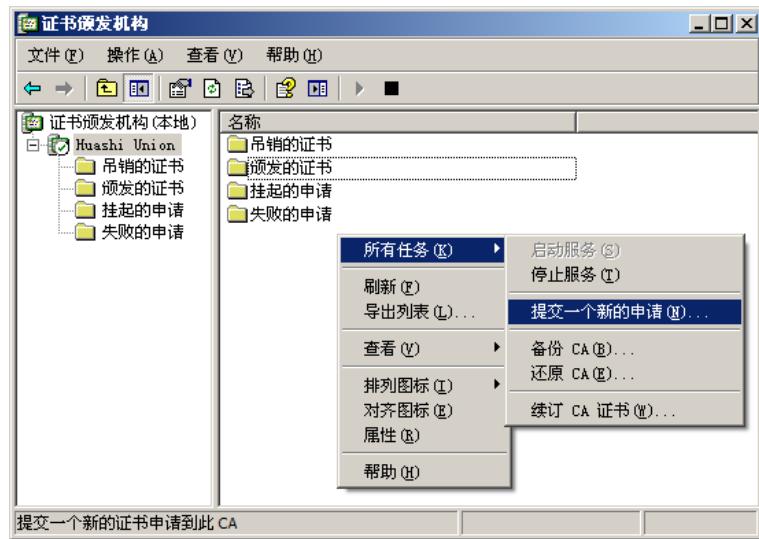


8、测试该站点



(5) 证书服务器

1、提交一个新的申请



2、查看到新的申请



3、颁发该申请的证书



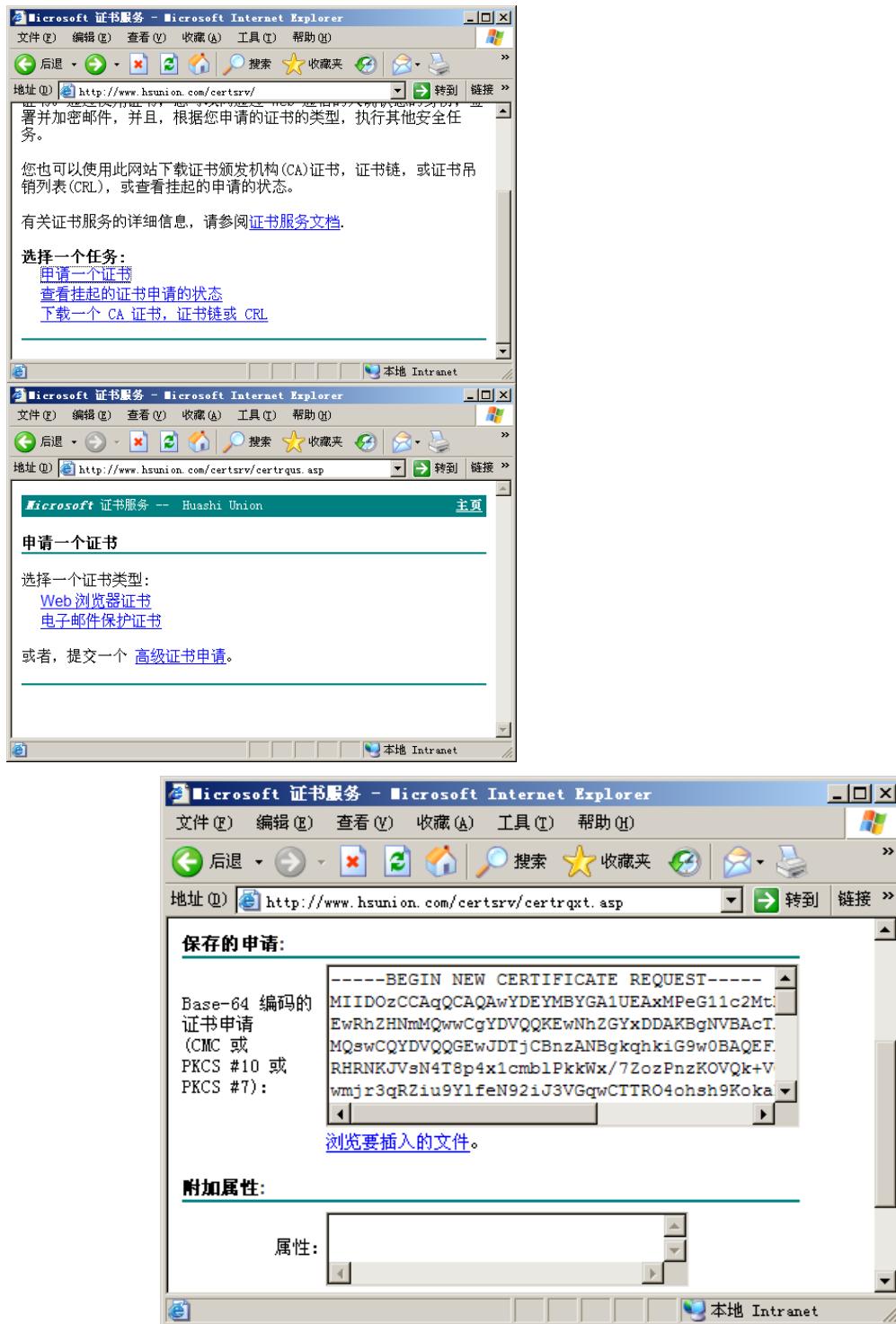
4、查看该证书



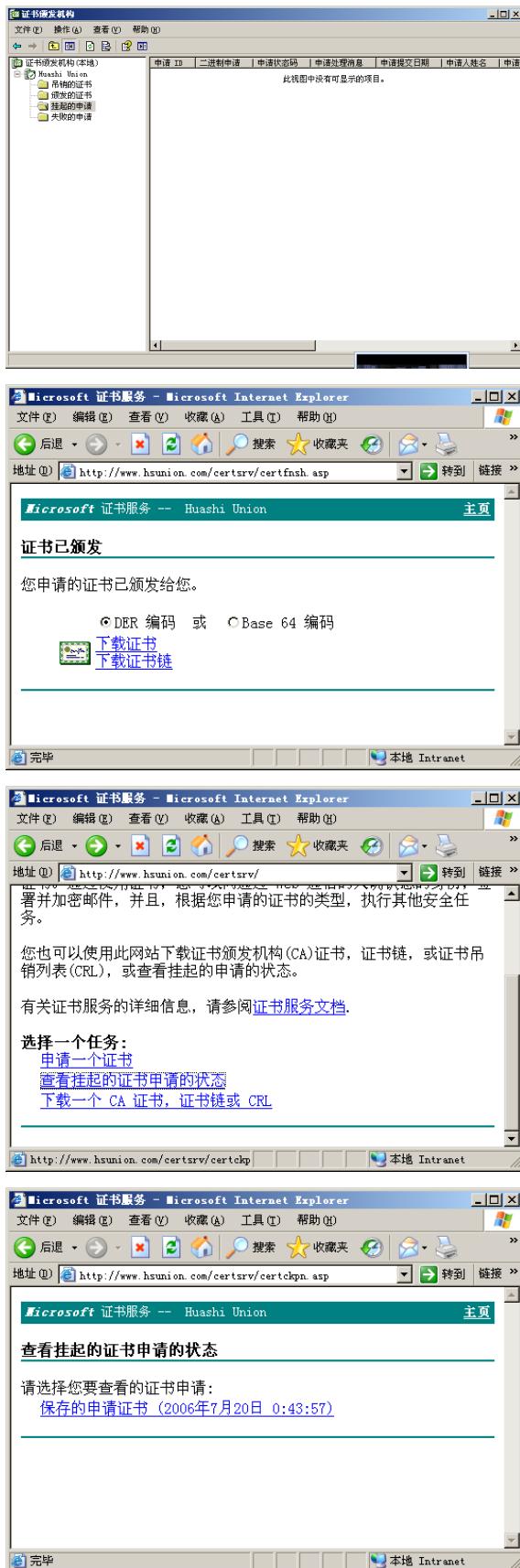
5、导出该证书

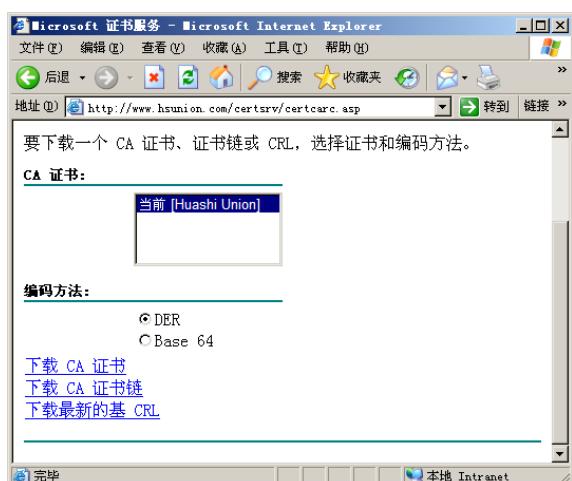
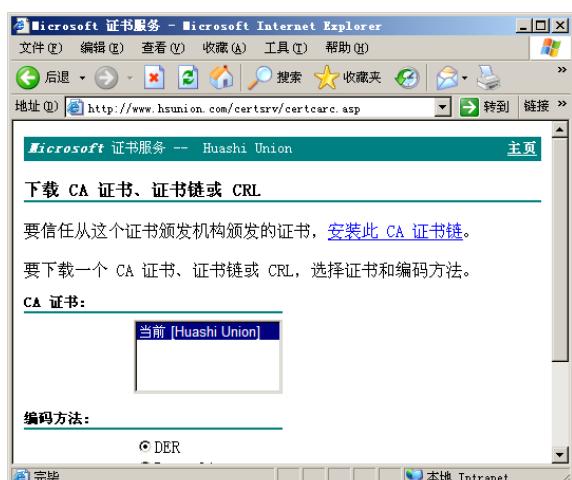
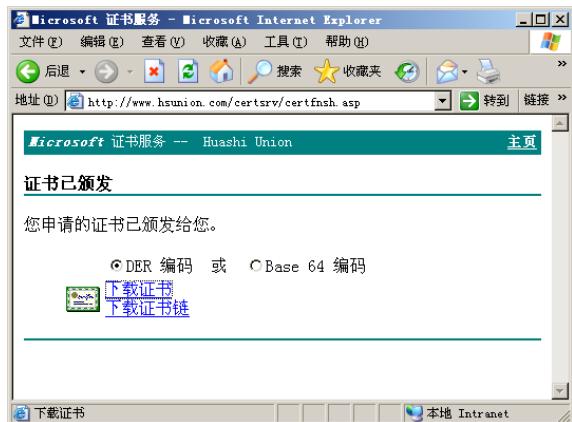


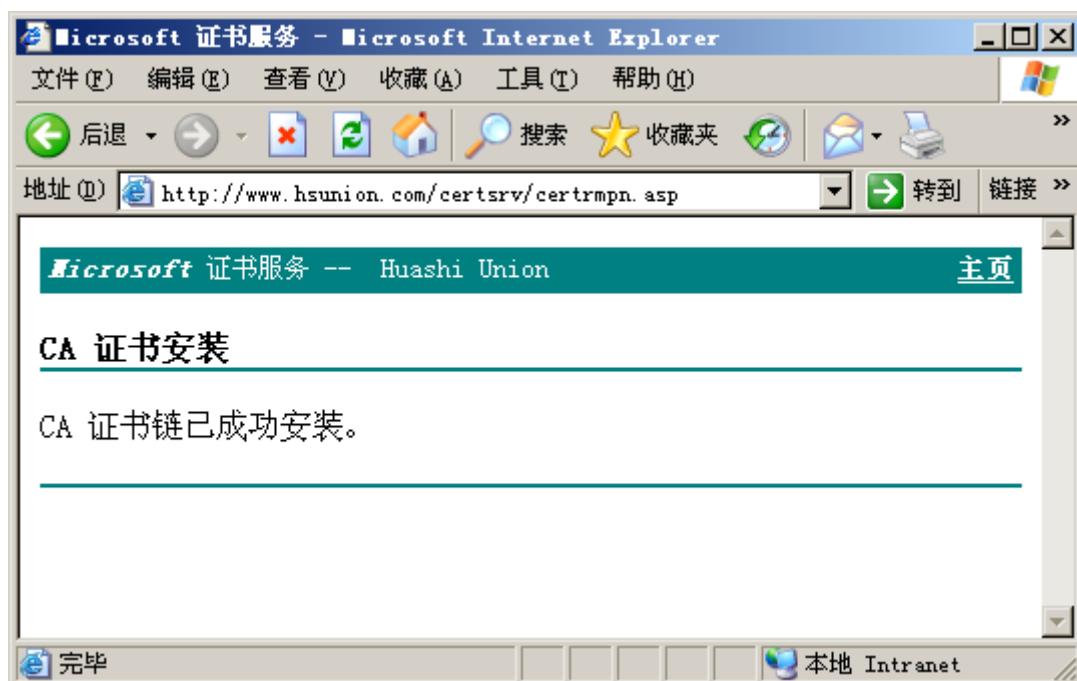
6、通过 Web 方式申请并颁发证书



通过申请的办法同 CA 方式，即：







(6) FTP 服务器

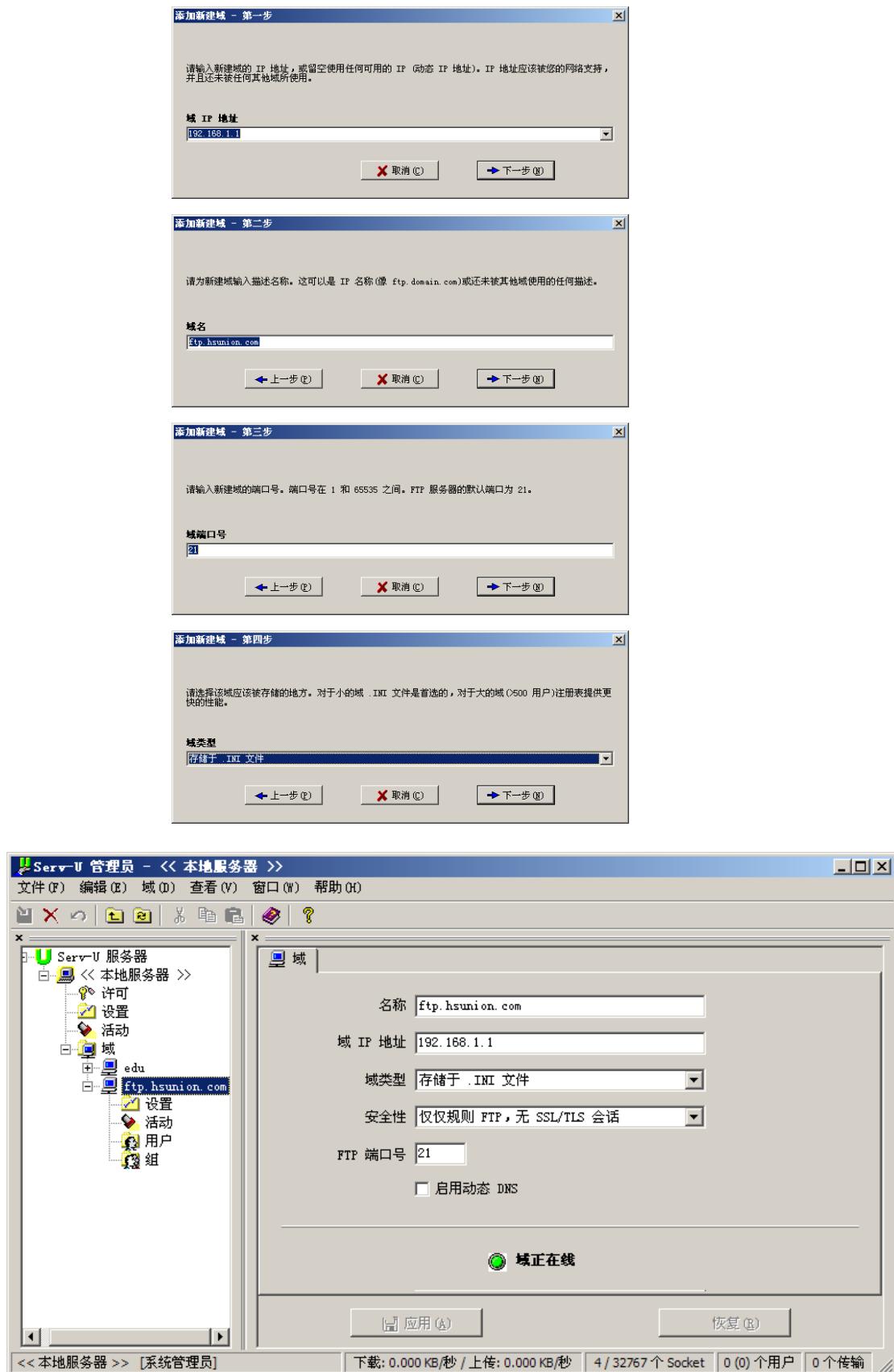
1、安装 Serv-U 服务器

2、启动 Serv-U



3、新建域

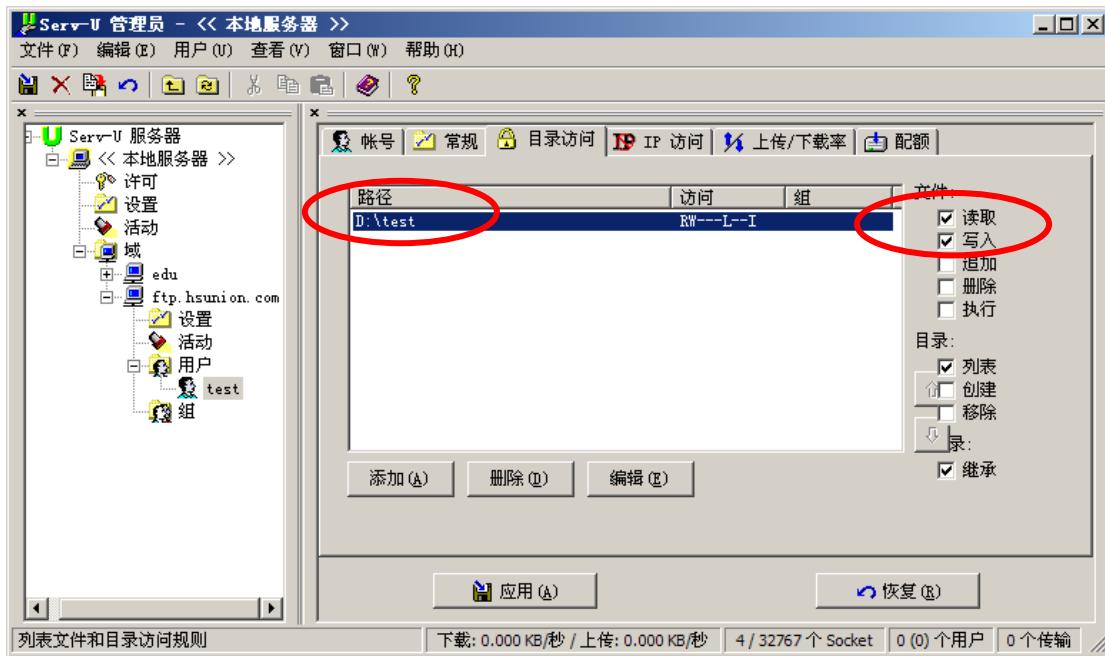




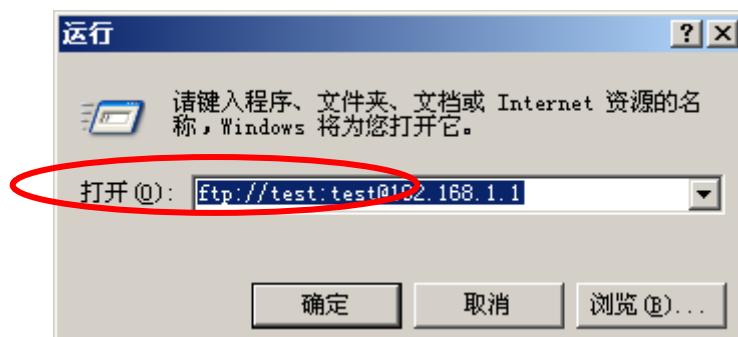
4、添加帐号



5、设置访问权限

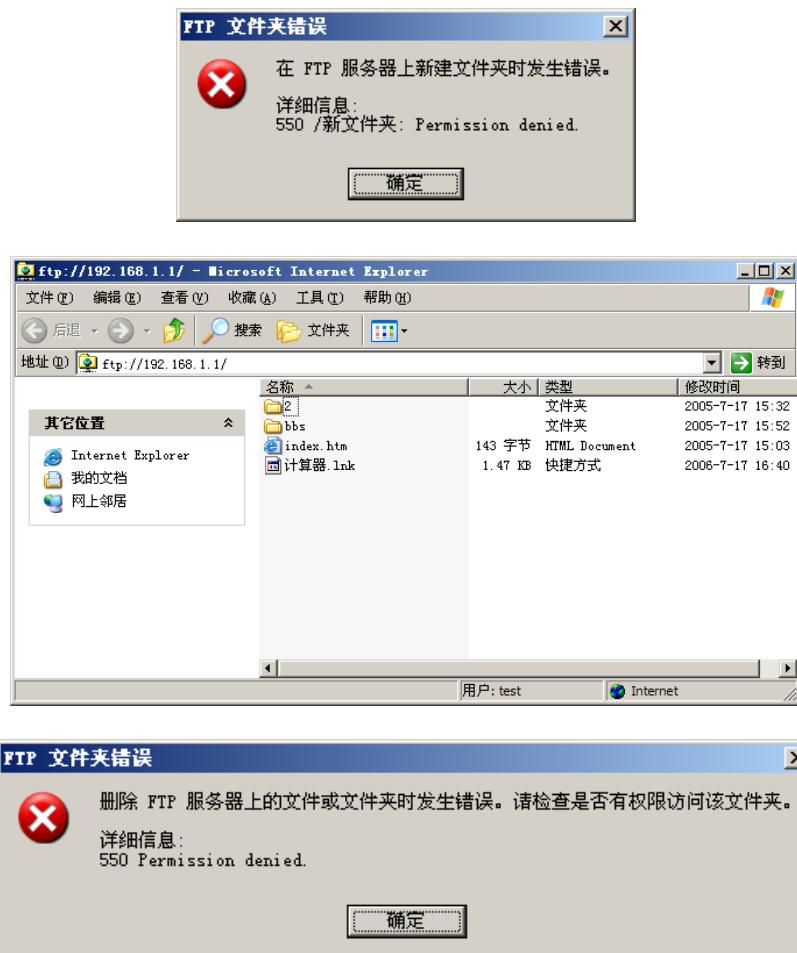


6、测试站点是否可以访问



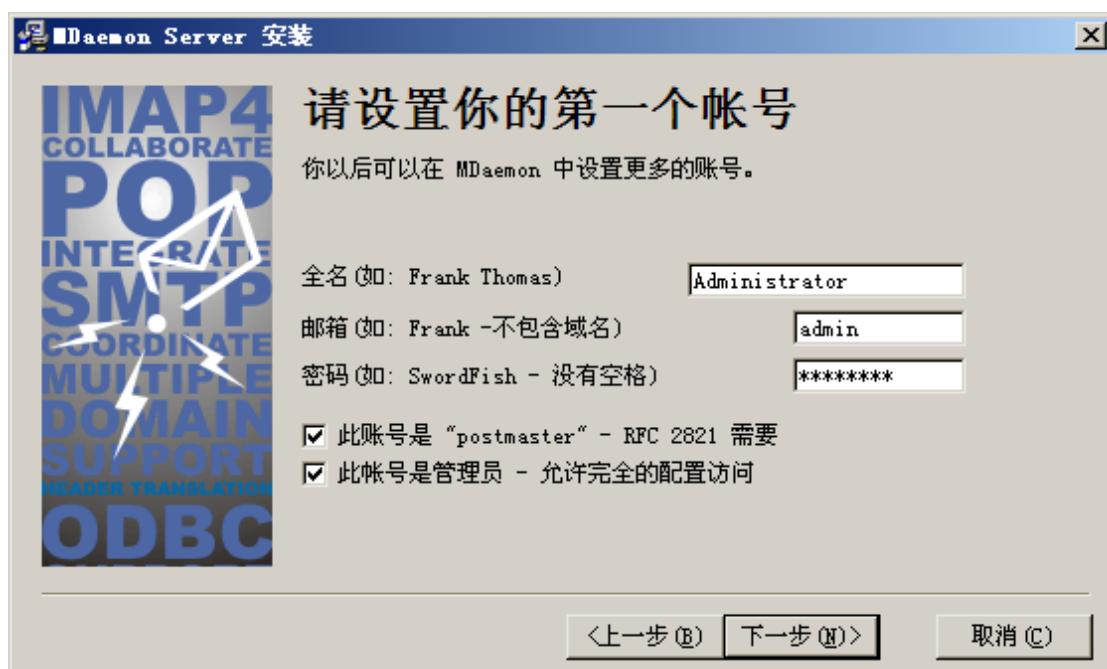
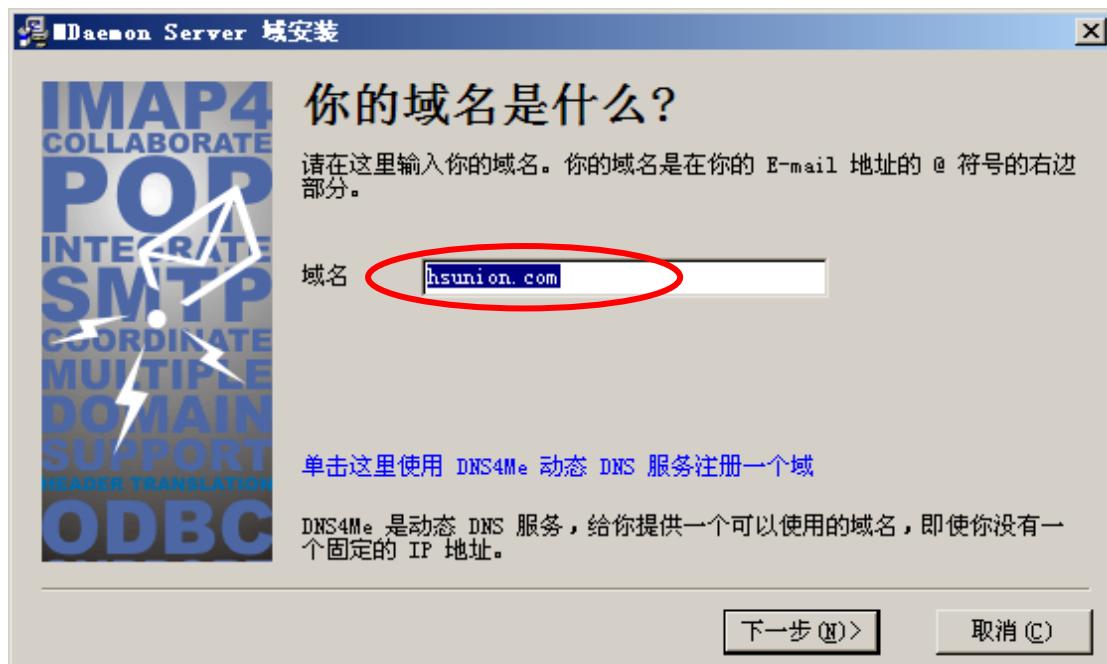
7、测试权限

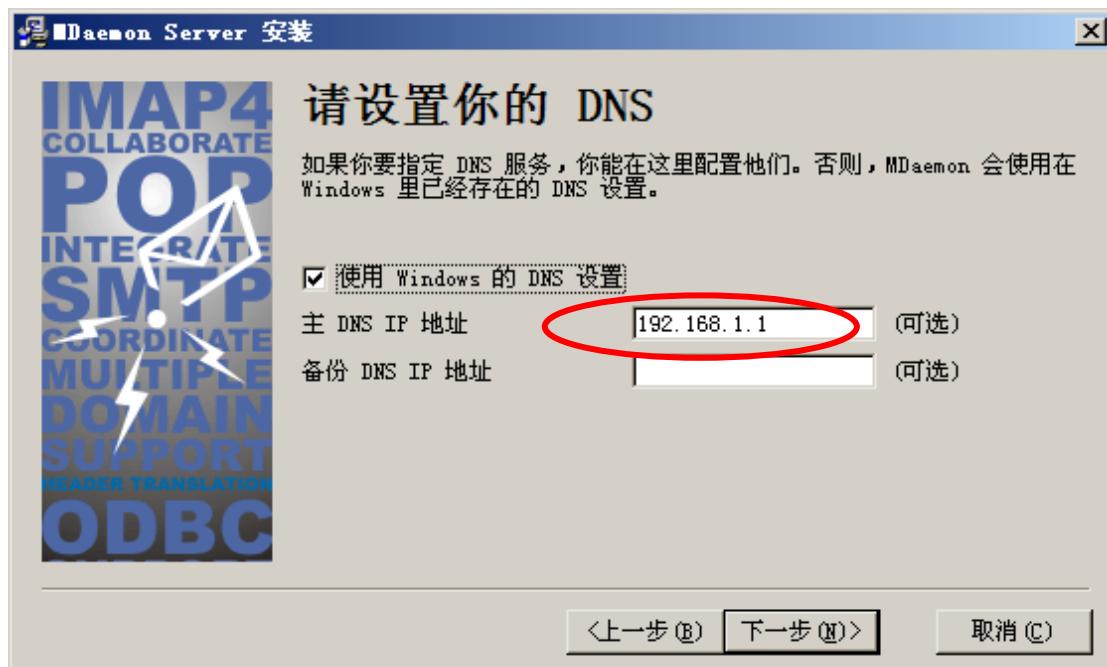
不能新建文件夹，复制文件可以通过，不能删除。



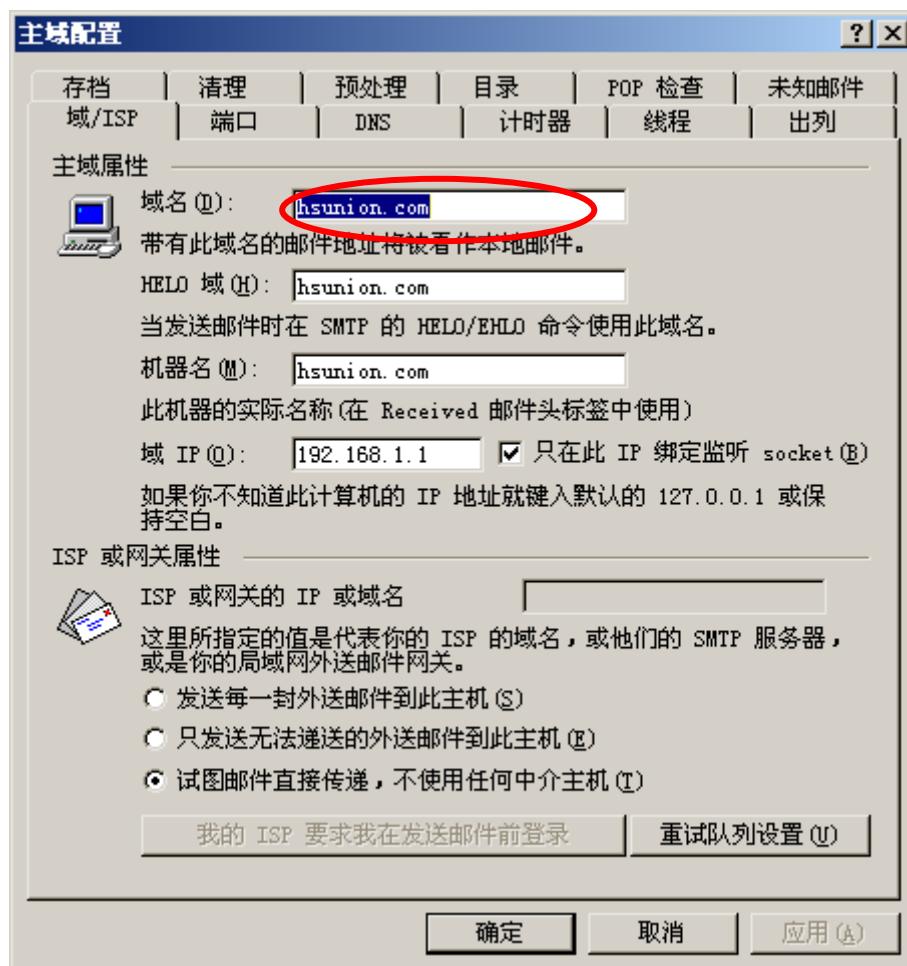
(7) SMTP 和 POP 服务器

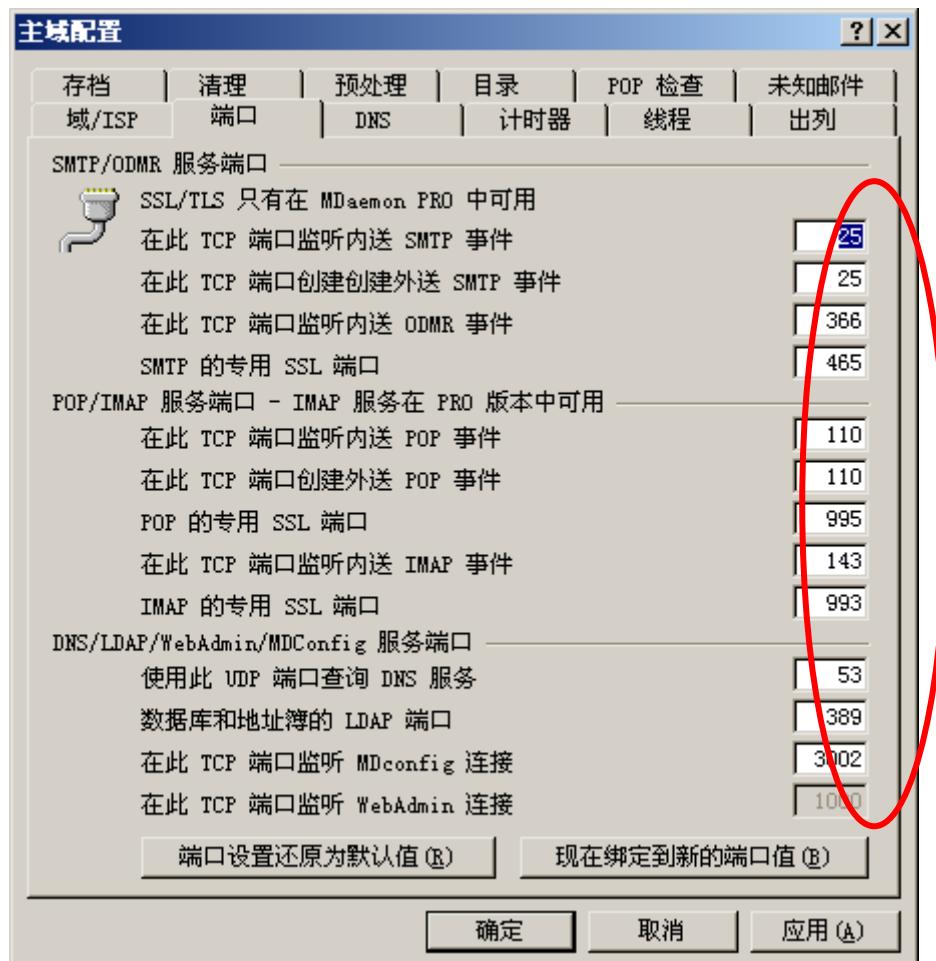
1、安装、启动并配置 MDaemon。



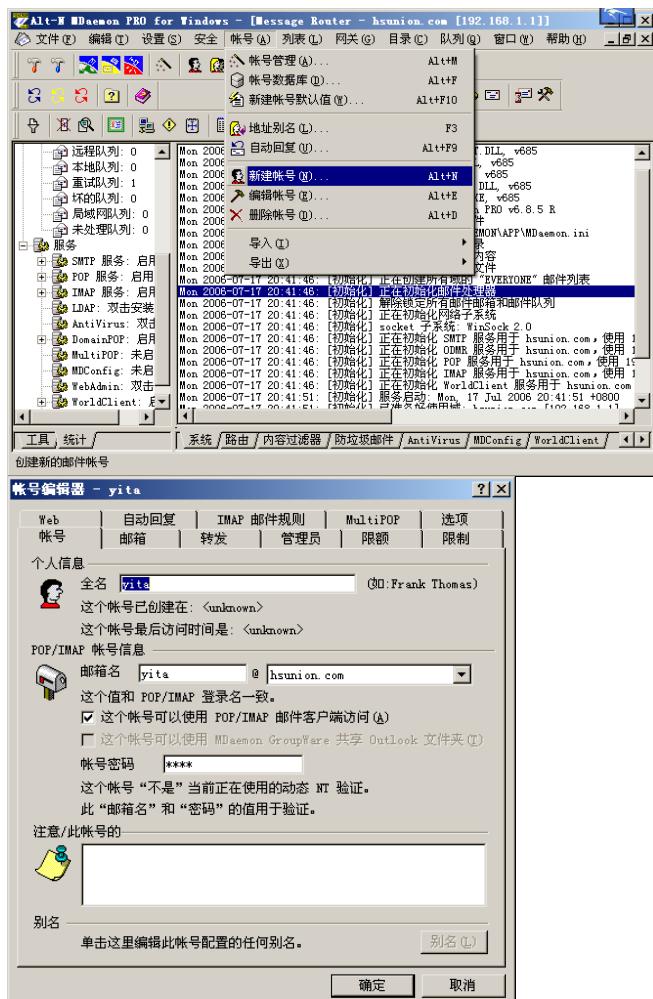


2、配置 IP 和端口号



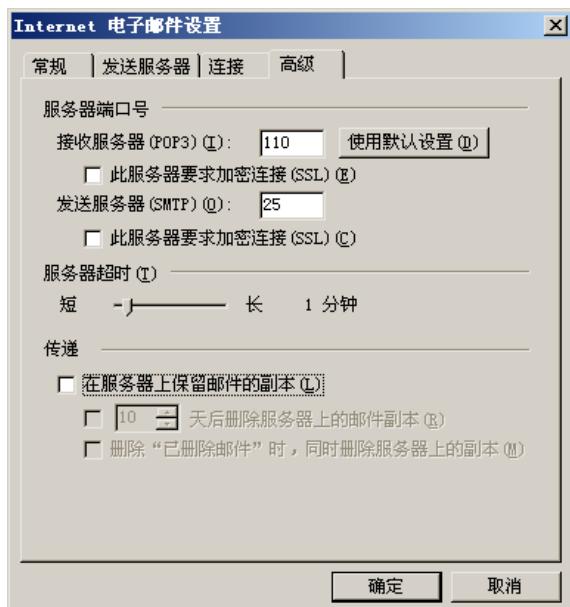
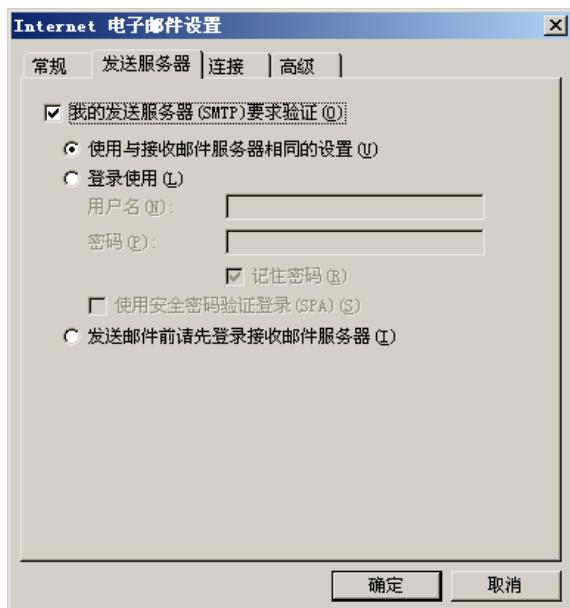


3、创建帐号

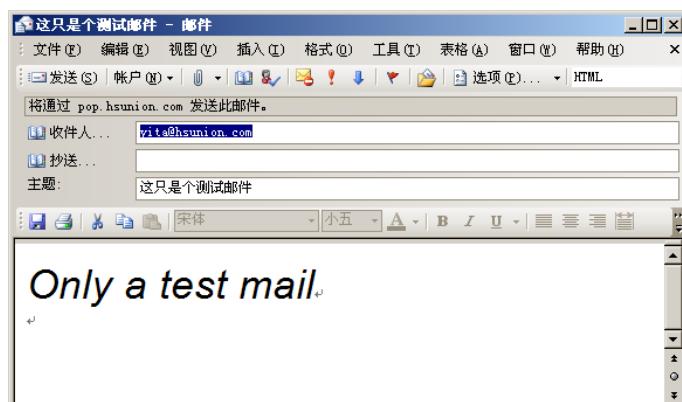


4、配置客户端（如 Microsoft Office Outlook 2003）

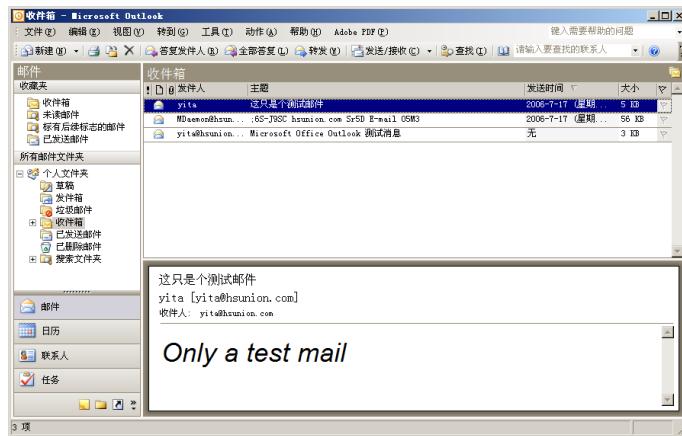




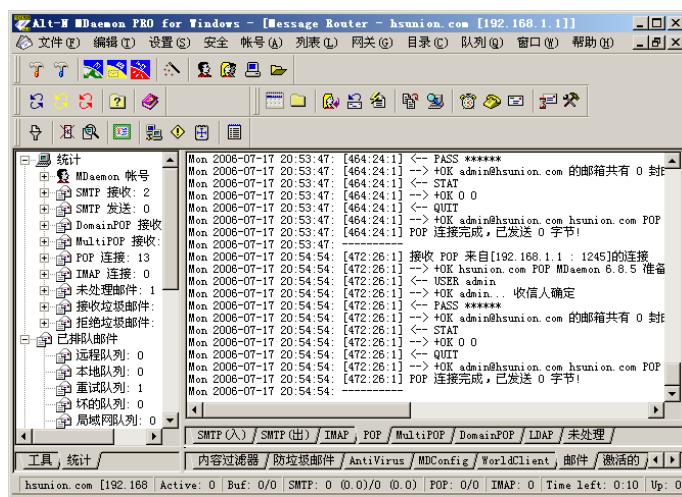
5、测试邮件是否能发送、配置是否正确



6、收取成功



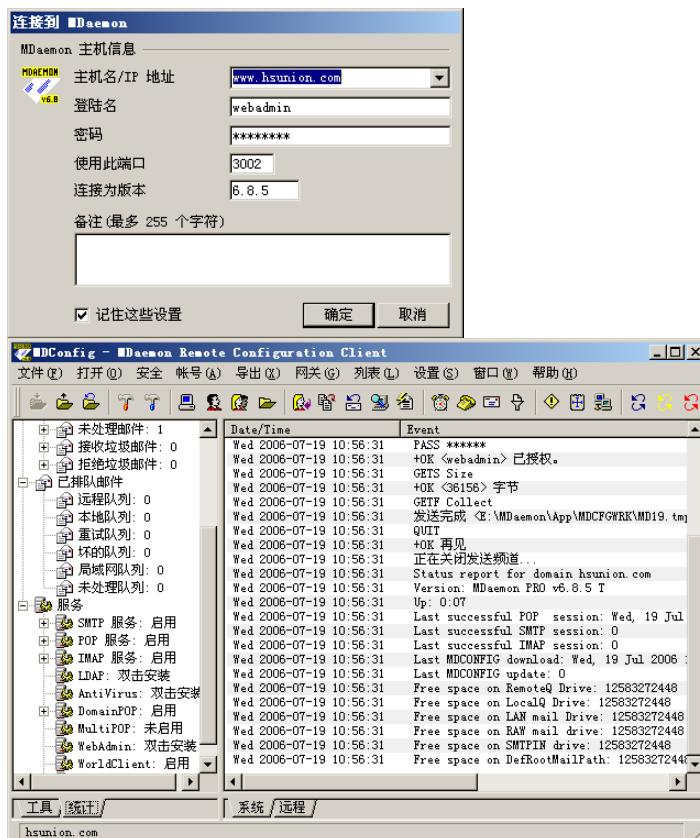
7、邮件服务器上的记录



8、远程管理 MDaemon 邮件服务器



从开始菜单启动 MDConfig



9、Web 方式管理 MDaemon



写邮件，并发送：



收到邮件：



9 代理服务器软件

9.1 实验目的

通过完成实验，掌握基于 RFC 应用层协议规约文档传输的原理，实现符合接口且能和已有知名软件协同运作的软件。

9.2 需求说明

某公司为了保障网络安全，规范网络出入的数据，只允许少数计算机连接外网。内网计算机偶尔需要上网时，应通过这些可以连接外网的计算机作为代理服务器。

该公司委托你设计并实现一个 SOCK5 和 SOCK4 协议的代理服务器，代理服务器运行时，允许 Chrome 浏览器、Internet Explorer 等软件通过设置代理连接外网。

9.3 输入输出接口

通过运行命令行 “`./proxy -n 1080`” ，将该软件服务器启动，监听 1080 端口。

安装 Windows 虚拟机，设置网卡运行模式为 Host Only，确保该机器无法连接外网。

在虚拟机中通过运行 Internet Explorer，通过输入外网 IP 地址访问网站，得到无法连接的错误提示。在 Internet 选项中，找到“连接”标签页，单击“局域网设置”按钮，选中“为 LAN 使用代理服务器”，并单击“高级”按钮。取消选中“对所有协议均使用相同的代理”，并在“套接字”一行的标签中填入服务器端的 IP 地址与端口号，如：192.168.147.129 和 1080。刷新页面，如果成功则可以访问网站。此处勿使用域名，因为 IE 默认将 DNS 解析交给本地服务器，而虚拟机处于断外网状态，无法成功解析。

通过以下命令行运行 Chrome（其中 Chrome 路径和 IP 地址的部分因各人机器不同而异）：

```
"C:\Users\***\AppData\Local\Google\Chrome\Application\chrome.exe" --show-app-list --proxy-server="SOCKS5://192.168.14.7.129:1080"
```

通过域名访问外网，成功访问。

9.4 设计思路

服务器端主要实现以下功能：

- 1、通过 socket()函数，建立套接字；
- 2、通过 bind()函数，绑定本地地址与端口；
- 3、通过 listen()函数，监听，并设置并发数；
- 4、通过 accept()接受连入的套接字，并通过 pthread()新建线程处理请求；
- 5、在处理请求时，读取 1 个字节，判断 SOCKS 版本号是 4 还是 5，交由相应模块处理；
- 6、以 SOCKS5 为例，判断命令是针对 IP 还是域名的处理；
- 7、建立对目标 IP 和端口号的连接，读取内容，并发送到客户端上。

9.5 特别说明

在实现时，可以自行学习附录二示例程序后尝试设计实现新的代理软件。如无法完成，也可以在学习之后对附录二程序做注解，分析其程序结构，做适当的笔记。

9.6 附录一：Socks 协议下载地址

Socks4：<https://www.openssh.com/txt/socks4.protocol>

Socks4a：<https://www.openssh.com/txt/socks4a.protocol>

Socks5 用户认证：<https://tools.ietf.org/html/rfc1929>

Socks5：<https://tools.ietf.org/html/rfc1928>

9.7 附录二：SOCKS_PROXY 实现示例程序

本文件下载地址为：https://github.com/fgssfgss/socks_proxy/

在 Cygwin 或 Linux 系统下，参考使用以下命令编译。

```
gcc main.c -pthread -Wno-unused-result -g -std=gnu99 -Wall -o proxy.exe
```

具体地，源代码如下：

```
#define __GNU_SOURCE
#include <sys/types.h>
#include <stdio.h>
#include <stdarg.h>
#include <time.h>
#include <errno.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/fcntl.h>
#include <sys/stat.h>
#include <netdb.h>
#include <sys/select.h>
#include <arpa/inet.h>
#include <netinet/tcp.h>
#include <pthread.h>

#define BUFSIZE 65536
#define IPSIZE 4
#define ARRAY_SIZE(x) (sizeof(x) / sizeof(x[0]))
#define ARRAY_INIT {0}

unsigned short int port = 1080;
int daemon_mode = 0;
int auth_type;
char *arg_username;
char *arg_password;
FILE *log_file;
pthread_mutex_t lock;
```

```
enum socks {
    RESERVED = 0x00,
    VERSION4 = 0x04,
    VERSION5 = 0x05
};

enum socks_auth_methods {
    NOAUTH = 0x00,
    USERPASS = 0x02,
    NOMETHOD = 0xff
};

enum socks_auth_userpass {
    AUTH_OK = 0x00,
    AUTH_VERSION = 0x01,
    AUTH_FAIL = 0xff
};

enum socks_command {
    CONNECT = 0x01
};

enum socks_command_type {
    IP = 0x01,
    DOMAIN = 0x03
};

enum socks_status {
    OK = 0x00,
    FAILED = 0x05
};

void log_message(const char *message, ...)
{
    if (daemon_mode) {
        return;
    }

    char vbuffer[255];
    va_list args;
    va_start(args, message);
    vsnprintf(vbuffer, ARRAY_SIZE(vbuffer), message, args);
}
```

```
va_end(args);

time_t now;
time(&now);
char *date = ctime(&now);
date[strlen(date) - 1] = '\0';

unsigned long int self = (unsigned long
int)pthread_self();

if (errno != 0) {
    pthread_mutex_lock(&lock);
    fprintf(log_file, "[%s][%lu] Critical: %s - %s\n",
date, self,
            vbuffer, strerror(errno));
    errno = 0;
    pthread_mutex_unlock(&lock);
} else {
    fprintf(log_file, "[%s][%lu] Info: %s\n", date, self,
vbuffer);
}
fflush(log_file);
}

int readn(int fd, void *buf, int n)
{
    int nread, left = n;
    while (left > 0) {
        if ((nread = read(fd, buf, left)) == -1) {
            if (errno == EINTR || errno == EAGAIN) {
                continue;
            }
        } else {
            if (nread == 0) {
                return 0;
            } else {
                left -= nread;
                buf += nread;
            }
        }
    }
    return n;
}
```

```
int writen(int fd, void *buf, int n)
{
    int nwrite, left = n;
    while (left > 0) {
        if ((nwrite = write(fd, buf, left)) == -1) {
            if (errno == EINTR || errno == EAGAIN) {
                continue;
            }
        } else {
            if (nwrite == n) {
                return 0;
            } else {
                left -= nwrite;
                buf += nwrite;
            }
        }
    }
    return n;
}

void app_thread_exit(int ret, int fd)
{
    close(fd);
    pthread_exit((void *)&ret);
}

int app_connect(int type, void *buf, unsigned short int
portnum)
{
    int fd;
    struct sockaddr_in remote;
    char address[16];

    memset(address, 0, ARRAY_SIZE(address));

    if (type == IP) {
        char *ip = (char *)buf;
        snprintf(address, ARRAY_SIZE(address),
"%hu.%hu.%hu.%hu",
ip[0], ip[1], ip[2], ip[3]);
        memset(&remote, 0, sizeof(remote));
        remote.sin_family = AF_INET;
    }
}
```

```
remote.sin_addr.s_addr = inet_addr(address);
remote.sin_port = htons(portnum);

fd = socket(AF_INET, SOCK_STREAM, 0);
if (connect(fd, (struct sockaddr *)&remote,
sizeof(remote)) < 0) {
    log_message("connect() in app_connect");
    close(fd);
    return -1;
}

return fd;
} else if (type == DOMAIN) {
    char portaddr[6];
    struct addrinfo *res;
    snprintf(portaddr, ARRAY_SIZE(portaddr), "%d",
portnum);
    log_message("getaddrinfo: %s %s", (char *)buf,
portaddr);
    int ret = getaddrinfo((char *)buf, portaddr, NULL,
&res);
    if (ret == EAI_NODATA) {
        return -1;
    } else if (ret == 0) {
        struct addrinfo *r;
        for (r = res; r != NULL; r = r->ai_next) {
            fd = socket(r->ai_family, r->ai_socktype,
r->ai_protocol);
            if (fd == -1) {
                continue;
            }
            ret = connect(fd, r->ai_addr, r->ai_addrlen);
            if (ret == 0) {
                freeaddrinfo(res);
                return fd;
            } else {
                close(fd);
            }
        }
    }
    freeaddrinfo(res);
    return -1;
}
```

```
    return -1;
}

int socks_invitation(int fd, int *version)
{
    char init[2];
    int nread = readn(fd, (void *)init, ARRAY_SIZE(init));
    if (nread == 2 && init[0] != VERSION5 && init[0] != VERSION4) {
        log_message("They send us %hhX %hhX", init[0], init[1]);
        log_message("Incompatible version!");
        app_thread_exit(0, fd);
    }
    log_message("Initial %hhX %hhX", init[0], init[1]);
    *version = init[0];
    return init[1];
}

char *socks5_auth_get_user(int fd)
{
    unsigned char size;
    readn(fd, (void *)&size, sizeof(size));

    char *user = (char *)malloc(sizeof(char) * size + 1);
    readn(fd, (void *)user, (int)size);
    user[size] = 0;

    return user;
}

char *socks5_auth_get_pass(int fd)
{
    unsigned char size;
    readn(fd, (void *)&size, sizeof(size));

    char *pass = (char *)malloc(sizeof(char) * size + 1);
    readn(fd, (void *)pass, (int)size);
    pass[size] = 0;

    return pass;
}
```

```
int socks5_auth_userpass(int fd)
{
    char answer[2] = { VERSION5, USERPASS };
    writen(fd, (void *)answer, ARRAY_SIZE(answer));
    char resp;
    readn(fd, (void *)&resp, sizeof(resp));
    log_message("auth %hhX", resp);
    char *username = socks5_auth_get_user(fd);
    char *password = socks5_auth_get_pass(fd);
    log_message("l: %s p: %s", username, password);
    if (strcmp(arg_username, username) == 0
        && strcmp(arg_password, password) == 0) {
        char answer[2] = { AUTH_VERSION, AUTH_OK };
        writen(fd, (void *)answer, ARRAY_SIZE(answer));
        free(username);
        free(password);
        return 0;
    } else {
        char answer[2] = { AUTH_VERSION, AUTH_FAIL };
        writen(fd, (void *)answer, ARRAY_SIZE(answer));
        free(username);
        free(password);
        return 1;
    }
}

int socks5_auth_noauth(int fd)
{
    char answer[2] = { VERSION5, NOAUTH };
    writen(fd, (void *)answer, ARRAY_SIZE(answer));
    return 0;
}

void socks5_auth_notsupported(int fd)
{
    char answer[2] = { VERSION5, NOMETHOD };
    writen(fd, (void *)answer, ARRAY_SIZE(answer));
}

void socks5_auth(int fd, int methods_count)
{
    int supported = 0;
```

```
int num = methods_count;
for (int i = 0; i < num; i++) {
    char type;
    readn(fd, (void *)&type, 1);
    log_message("Method AUTH %hhX", type);
    if (type == auth_type) {
        supported = 1;
    }
}
if (supported == 0) {
    socks5_auth_notsupported(fd);
    app_thread_exit(1, fd);
}
int ret = 0;
switch (auth_type) {
case NOAUTH:
    ret = socks5_auth_noauth(fd);
    break;
case USERPASS:
    ret = socks5_auth_userpass(fd);
    break;
}
if (ret == 0) {
    return;
} else {
    app_thread_exit(1, fd);
}
}

int socks5_command(int fd)
{
    char command[4];
    readn(fd, (void *)command, ARRAY_SIZE(command));
    log_message("Command %hhX %hhX %hhX %hhX", command[0],
    command[1],
        command[2], command[3]);
    return command[3];
}

unsigned short int socks_read_port(int fd)
{
    unsigned short int p;
    readn(fd, (void *)&p, sizeof(p));
```

```
log_message("Port %hu", ntohs(p));
return p;
}

char *socks_ip_read(int fd)
{
    char *ip = (char *)malloc(sizeof(char) * IPSIZE);
    readn(fd, (void *)ip, IPSIZE);
    log_message("IP %hu.%hu.%hu.%hu", ip[0], ip[1], ip[2],
ip[3]);
    return ip;
}

void socks5_ip_send_response(int fd, char *ip, unsigned short
int port)
{
    char response[4] = { VERSION5, OK, RESERVED, IP };
    writen(fd, (void *)response, ARRAY_SIZE(response));
    writen(fd, (void *)ip, IPSIZE);
    writen(fd, (void *)&port, sizeof(port));
}

char *socks5_domain_read(int fd, unsigned char *size)
{
    unsigned char s;
    readn(fd, (void *)&s, sizeof(s));
    char *address = (char *)malloc((sizeof(char) * s) + 1);
    readn(fd, (void *)address, (int)s);
    address[s] = 0;
    log_message("Address %s", address);
    *size = s;
    return address;
}

void socks5_domain_send_response(int fd, char *domain,
unsigned char size,
        unsigned short int port)
{
    char response[4] = { VERSION5, OK, RESERVED, DOMAIN };
    writen(fd, (void *)response, ARRAY_SIZE(response));
    writen(fd, (void *)&size, sizeof(size));
    writen(fd, (void *)domain, size * sizeof(char));
    writen(fd, (void *)&port, sizeof(port));
}
```

```
}

int socks4_is_4a(char *ip)
{
    return (ip[0] == 0 && ip[1] == 0 && ip[2] == 0 && ip[3] != 0);
}

int socks4_read_nstring(int fd, char *buf, int size)
{
    char sym = 0;
    int nread = 0;
    int i = 0;

    while (i < size) {
        nread = recv(fd, &sym, sizeof(char), 0);

        if (nread <= 0) {
            break;
        } else {
            buf[i] = sym;
            i++;
        }

        if (sym == 0) {
            break;
        }
    }

    return i;
}

void socks4_send_response(int fd, int status)
{
    char resp[8] = {0x00, (char)status, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
    writen(fd, (void *)resp, ARRAY_SIZE(resp));
}

void app_socket_pipe(int fd0, int fd1)
{
    int maxfd, ret;
    fd_set rd_set;
```

```
size_t nread;
char buffer_r[BUFSIZE];

log_message("Connecting two sockets");

maxfd = (fd0 > fd1) ? fd0 : fd1;
while (1) {
    FD_ZERO(&rd_set);
    FD_SET(fd0, &rd_set);
    FD_SET(fd1, &rd_set);
    ret = select(maxfd + 1, &rd_set, NULL, NULL, NULL);

    if (ret < 0 && errno == EINTR) {
        continue;
    }

    if (FD_ISSET(fd0, &rd_set)) {
        nread = recv(fd0, buffer_r, BUFSIZE, 0);
        if (nread <= 0)
            break;
        send(fd1, (const void *)buffer_r, nread, 0);
    }

    if (FD_ISSET(fd1, &rd_set)) {
        nread = recv(fd1, buffer_r, BUFSIZE, 0);
        if (nread <= 0)
            break;
        send(fd0, (const void *)buffer_r, nread, 0);
    }
}

void *app_thread_process(void *fd)
{
    int net_fd = *(int *)fd;
    int version = 0;
    int inet_fd = -1;
    char methods = socks_invitation(net_fd, &version);

    switch (version) {
    case VERSION5: {
        socks5_auth(net_fd, methods);
        int command = socks5_command(net_fd);
    }
}
```

```
if (command == IP) {
    char *ip = socks_ip_read(net_fd);
    unsigned short int p = socks_read_port(net_fd);

    inet_fd = app_connect(IP, (void *)ip, ntohs(p));
    if (inet_fd == -1) {
        app_thread_exit(1, net_fd);
    }
    socks5_ip_send_response(net_fd, ip, p);
    free(ip);
    break;
} else if (command == DOMAIN) {
    unsigned char size;
    char *address = socks5_domain_read(net_fd,
&size);
    unsigned short int p = socks_read_port(net_fd);

    inet_fd = app_connect(DOMAIN, (void *)address,
ntohs(p));
    if (inet_fd == -1) {
        app_thread_exit(1, net_fd);
    }
    socks5_domain_send_response(net_fd, address,
size, p);
    free(address);
    break;
} else {
    app_thread_exit(1, net_fd);
}
}

case VERSION4: {
    if (methods == 1) {
        char ident[255];
        unsigned short int p = socks_read_port(net_fd);
        char *ip = socks_ip_read(net_fd);
        socks4_read_nstring(net_fd, ident,
sizeof(ident));

        if (socks4_is_4a(ip)) {
            char domain[255];
            socks4_read_nstring(net_fd, domain,
sizeof(domain));
        }
    }
}
```

```
        log_message("Socks4A: ident:%s; domain:%s;",  
ident, domain);  
        inet_fd = app_connect(DOMAIN, (void *)domain,  
ntohs(p));  
    } else {  
        log_message("Socks4: connect by ip & port");  
        inet_fd = app_connect(IP, (void *)ip,  
ntohs(p));  
    }  
  
    if (inet_fd != -1) {  
        socks4_send_response(net_fd, 0x5a);  
    } else {  
        socks4_send_response(net_fd, 0x5b);  
        free(ip);  
        app_thread_exit(1, net_fd);  
    }  
  
    free(ip);  
} else {  
    log_message("Unsupported mode");  
}  
break;  
}  
}  
  
app_socket_pipe(inet_fd, net_fd);  
close(inet_fd);  
app_thread_exit(0, net_fd);  
  
return NULL;  
}  
  
int app_loop()  
{  
    int sock_fd, net_fd;  
    int optval = 1;  
    struct sockaddr_in local, remote;  
    socklen_t remotelen;  
    if ((sock_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {  
        log_message("socket()");  
        exit(1);  
    }
```

```
if (setsockopt
    (sock_fd, SOL_SOCKET, SO_REUSEADDR, (char *)&optval,
     sizeof(optval)) < 0) {
    log_message("setsockopt()");
    exit(1);
}

memset(&local, 0, sizeof(local));
local.sin_family = AF_INET;
local.sin_addr.s_addr = htonl(INADDR_ANY);
local.sin_port = htons(port);

if (bind(sock_fd, (struct sockaddr *)&local,
sizeof(local)) < 0) {
    log_message("bind()");
    exit(1);
}

if (listen(sock_fd, 25) < 0) {
    log_message("listen()");
    exit(1);
}

remotelen = sizeof(remote);
memset(&remote, 0, sizeof(remote));

log_message("Listening port %d...", port);

pthread_t worker;
while (1) {
    if ((net_fd =
        accept(sock_fd, (struct sockaddr *)&remote,
               &remotelen)) < 0) {
        log_message("accept()");
        exit(1);
    }
    int one = 1;
    setsockopt(sock_fd, SOL_TCP, TCP_NODELAY, &one,
sizeof(one));
    if (pthread_create
        (&worker, NULL, &app_thread_process,
         (void *)&net_fd) == 0) {
```

```
        pthread_detach(worker);
    } else {
        log_message("pthread_create()");
    }
}

void daemonize()
{
    pid_t pid;
    int x;

    pid = fork();

    if (pid < 0) {
        exit(EXIT_FAILURE);
    }

    if (pid > 0) {
        exit(EXIT_SUCCESS);
    }

    if (setsid() < 0) {
        exit(EXIT_FAILURE);
    }

    signal(SIGCHLD, SIG_IGN);
    signal(SIGHUP, SIG_IGN);

    pid = fork();

    if (pid < 0) {
        exit(EXIT_FAILURE);
    }

    if (pid > 0) {
        exit(EXIT_SUCCESS);
    }

    umask(0);
    chdir("/");

    for (x = sysconf(_SC_OPEN_MAX); x >= 0; x--) {
```

```
        close(x);
    }
}

void usage(char *app)
{
    printf
        ("USAGE: %s [-h][-n PORT][-a AUTHTYPE][-u USERNAME][-p
PASSWORD][-l LOGFILE]\n",
         app);
    printf("AUTHTYPE: 0 for NOAUTH, 2 for USERPASS\n");
    printf
        ("By default: port is 1080, authtype is no auth,
logfile is stdout\n");
    exit(1);
}

int main(int argc, char *argv[])
{
    int ret;
    log_file = stdout;
    auth_type = NOAUTH;
    arg_username = "user";
    arg_password = "pass";
    pthread_mutex_init(&lock, NULL);

    signal(SIGPIPE, SIG_IGN);

    while ((ret = getopt(argc, argv, "n:u:p:l:a:hd")) != -1) {
        switch (ret) {
        case 'd':{
            daemon_mode = 1;
            daemonize();
            break;
        }
        case 'n':{
            port = atoi(optarg) & 0xffff;
            break;
        }
        case 'u':{
            arg_username = strdup(optarg);
            break;
        }
    }
}
```

```
case 'p':{
    arg_password = strdup(optarg);
    break;
}
case 'l':{
    freopen(optarg, "wa", log_file);
    break;
}
case 'a':{
    auth_type = atoi(optarg);
    break;
}
case 'h':
default:
    usage(argv[0]);
}
}
log_message("Starting with authtype %X", auth_type);
if (auth_type != NOAUTH) {
    log_message("Username is %s, password is %s",
arg_username,
                arg_password);
}
app_loop();
return 0;
}
```

10 RJ-45 接口网线的制作

10.1 实验要求

掌握使用双绞线作为传输介质实现以太网连接的方法，学会制作 RJ-45 接头。通过双机互联验证有效性，并完成实验报告。

10.2 操作步骤

请按以下步骤完成：

- 1、阅读实验教程课本（附录 1），上网搜集相关资料，观看群文件中的解说视频，或上网搜索 RJ-45 网线的知识，记住网线颜色顺序和实验要领；
- 2、实验课上每组向助教老师领取网线 4 条（每条约 1 米）、水晶头 16 个，每人单独使用；压线钳 1 个、测线器 1 个轮流使用。水晶头压制后不可拆卸，请节约使用。
- 3、制作时应注意水晶头金属片与每一条导线充分接触，再用压线钳压线。
- 4、在测线器测试成功后，拔出两台主机的网线，用所制作的网线互联，设置 IP 地址（192.168.1.1 和 192.168.1.2）和子网掩码（255.255.255.0），相互拷贝文件，验证连接成功，截图。
- 5、注意收集整理废弃物，实验完毕后将主机插线恢复原状。压线钳和网线测试仪应归还助教老师，网线可以带走留念。
- 6、提交的实验报告应包括制作网线的照片和网络连通的截图。请适当压缩图像，控制文件大小。

10.3 扩展实验

学有余力的同学可以参考附录二，完成无线局域网的构建实验。

10.4 附录一 RJ-45 连接线实验

2.2 RJ-45 接口连线实验

一、实验要求

掌握使用双绞线作为传输介质实现以太网连接的方法，学会制作两种类型的 RJ-45 接头。

二、实验条件

以太网集线器、微型计算机（带 RJ-45 接口的网卡）、五类双绞线、水晶头、卡线钳、测线器或万用电表。

三、实验指导

IEEE 于 1990 年批准双绞线可用于网络互联，目前它已经成为一种非常流行的通信介质。双绞线的结构如图 2.5 所示，分为屏蔽双绞线（Shielded Twisted-Pair, STP）和非屏蔽双绞线（Unshielded Twisted-Pair, UTP）两种。双绞线是指两根绝缘导线互相绞结在一起的通用传输介质。它可减少线间电磁干扰，适用于模拟或数字数据通信。屏蔽双绞线是为了提高双绞线的抗干扰能力，在外层增加铝箔屏蔽层而形成的。非屏蔽双绞线抗干扰能力虽然不强，但价格相对便宜，而且易于安装，是现在局域网上最常用到的网络电缆。

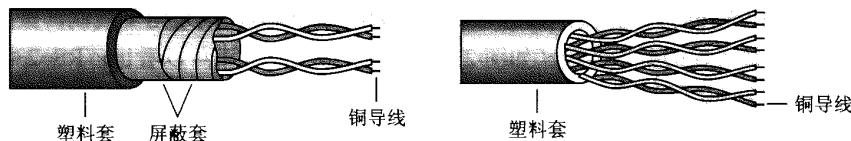


图 2.5 屏蔽和非屏蔽双绞线

非屏蔽双绞线有 6 种类型，如表 2.3 所示，其中 10BASE-T 的定义是传输速率是 10 Mbps，信号采用基带方式，T 表示双绞线。

表 2.3 非屏蔽双绞线的 6 种类型

类别	应 用	
Cat 1	可传输语音，不用于传输数据，常见于早期的电话线路	电信系统
Cat 2	可传输语音和数据，常见于 ISDN 和 T1 线路	
Cat 3	带宽 16 MHz，用于 10BASE-T，制作质量严格的 3 类线缆，也可用于 100BASE-T	计算机网络
Cat 4	带宽 20 MHz，用于 10BASE-T 或 100BASE-T	
Cat 5	带宽 100 MHz，用于 10BASE-T 或 100BASE-T。制作质量严格的 5 类线缆，可用于 1000BASE-T	计算机网络
Cat 6	带宽高达 200 MHz，可稳定运行千兆以太网 1000BASE-T	

1. RJ-45 连接器和双绞线线序

双绞线和以太网接口的连接是通过 RJ-45 连接器实现的，RJ-45 连接器俗称水晶头，结构如图 2.6 所示，它有 8 个金属针脚，以太网接口与之相对应也有 8 个针脚。根据这 8 个针脚的定义不同可将以太网接口分为两类，如表 2.4 和表 2.5 所示。

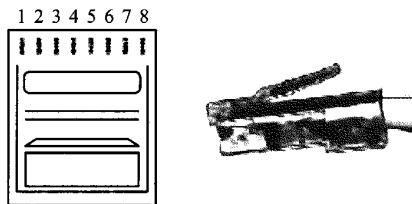


图 2.6 水晶头

表 2.4 第一类以太网接口针脚信号定义

第一类以太网接口（计算机和路由器上的以太网接口）

针脚序号	1	2	3	4	5	6	7	8
信号定义	TD+	TD-	RD+	NC	NC	RD-	NC	NC

2.2 RJ-45 接口连线实验

11

表 2.5 第二类以太网接口针脚信号定义

第二类以太网接口（交换机、集线器上的以太网接口）								
针脚序号	1	2	3	4	5	6	7	8
信号定义	RD+	RD-	TD+	NC	NC	TD-	NC	NC

计算机和路由器上的以太网接口属于第一类；交换机、集线器上的以太网接口属于第二类。一根 5 类双绞线由 8 根线组成，这 8 根线分成 4 对互绞在一起，颜色分别为：

[橙白、橙]、[绿白、绿]、[蓝白、蓝]、[棕白、棕]。

在以太网络中，用来连接网络设备的双绞线有两种：一种是直通线，用于连接数据终端设备（DTE）与数据通信设备（DCE）。例如用于微型计算机与交换机、交换机与路由器之间的连接线。直通线两端水晶头的导线排列顺序如图 2.7 所示，即两端水晶头的导线排列顺序必须一样。另一类是交叉线，用于连接网络中的相同设备，比如 PC 之间、交换机之间、路由器之间的连线。交叉线的导线排列顺序如图 2.8 所示，即一端水晶头第 1 根线的颜色要与另一端的第 3 根线一样，而第 2 根的要与第 6 根的一样。其他的导线的排列顺序与直通线一样。

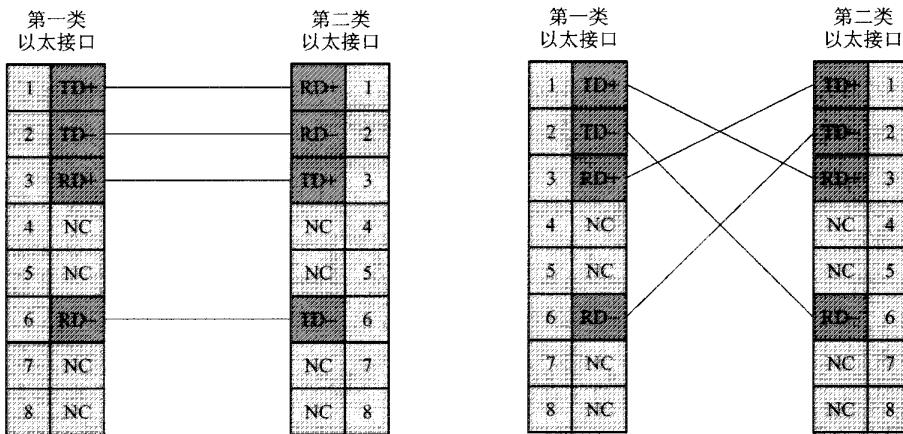


图 2.7 直通线连接示意图

图 2.8 交叉线连接示意图

2. 双绞线线头的制作步骤

压制水晶头需要使用专用卡线钳，按照下述步骤制作：

① 剥线：用卡线钳剪线刀口将线头剪齐，再将双绞线线头伸入剥线刀口，使线头触及前挡板，然后适度握紧卡线钳同时慢慢旋转双绞线，让刀口划开双绞线的保护胶皮，剥下保护胶皮取出线头。

② 理线：双绞线由 8 根有色导线两两绞合而成，将其按照橙白、橙、绿白、绿、蓝白、蓝、棕白、棕顺序平行排列，整理完毕用剪线刀将前端修齐。

③ 插线：右手捏住水晶头（有弹片一面向下），左手捏平双绞线，用力将排好的线平行插

入水晶头的线槽顶端。

④ 压线：确认所有导线都到位后，将水晶头放入卡线钳夹槽中，用力压下卡线钳，压紧线头即可。

如果要做的是一条直通线，重复上述方法制作双绞线的另一端即可（理线时按相同的线序）；而如果要制作的是交叉线，则理线时另一端的导线排列顺序是：绿白、绿、橙白、蓝、蓝白、橙、棕白、棕（即两端针脚1与3、2与6交叉互连）。

使用前要用测线器检查连接情况（没有条件也可以用万用电表检验），断路会导致无法通信，短路还可能损坏网卡或集线器。

3. 建立以太局域网

如图2.9所示，利用直通线把各台PC与集线器相连，用交叉线把集线器与交换机相连，这样一个小型以太局域网就建立好了。接下来只要对PC、集线器和交换机进行适当的配置，各PC就可以相互访问了。

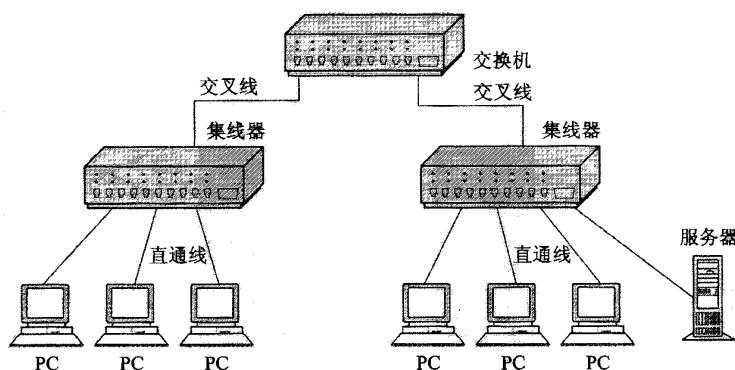


图2.9 用RJ-45连接的以太局域网结构图

如图2.10所示，建立10 Mbps以太局域网的另一种连线方式是使用同轴电缆和刺刀螺母连接器（Bayonet Nut Connector, BNC）接头，这种方式目前较少使用。

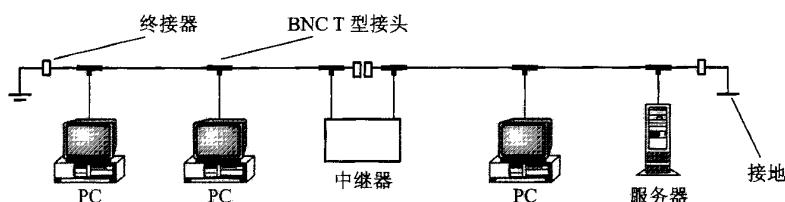


图2.10 用BNC连接的以太局域网结构图

四、思考题

RJ-45接口两种连接法的区别是什么？各自在什么场合下应用？

10.5 附录二 无线局域网的构建

2.3 无线局域网的构建

一、实验要求

掌握使用无线网卡和无线接入器构建无线局域网的方法。

二、实验条件

配有无线网卡的计算机 2 台、无线接入器 1 台（其中的以太网接口接入网络交换机）。

三、实验指导

通常计算机组网的传输介质主要依赖铜缆或光缆，构成有线局域网。但有线网络在某些场合存在许多不便：布线、改线工程量大；网中各节点难以移动；线路可能发生故障。这对正在迅速扩大的联网需求形成了瓶颈阻塞。无线局域网（Wireless Local Area Networks, WLAN）就是为了解决有线网络的这些问题而出现的。近年来，WLAN 已经在许多不适合网络布线的场合得到了广泛的应用。

目前常见的无线局域网基本上采用 IEEE 802.11 协议标准。在一个典型的 WLAN 环境中，有一些类似有线集线器的设备，称为接入点（Access Point, AP）。通常，一个 AP 能够在几十米至上百米的范围内连接多个无线用户。在同时具有有线和无线网络的情况下，AP 可以通过标准的以太网电缆与传统的有线网络相连，作为无线网络和有线网络的连接点。WLAN 的终端用户可通过无线网卡等访问网络。本实验利用无线路由器（又称无线 Hub、无线路由器）和无线网卡来组建一个典型的无线局域网，其网络拓扑如图 2.11 所示。

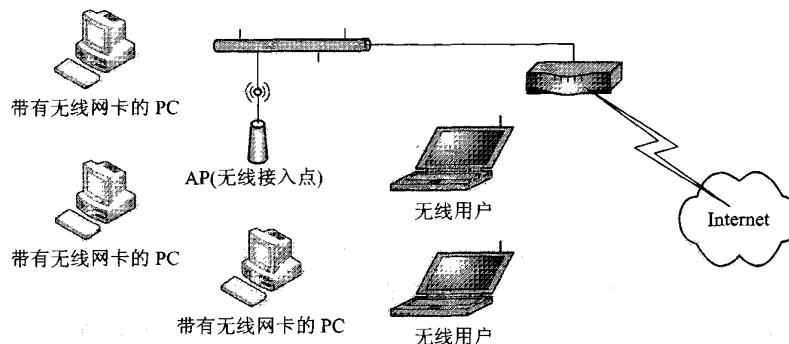


图 2.11 用无线路由器和无线网卡组建的无线局域网结构图

四、思考题

利用网络搜索，说明无线加密协议 WPA 的基本内容，指出这一协议与其他一些无线安全协议的区别。

11 RS-232 通信程序设计

11.1 实验要求

按附录 1 要求制作双机通信程序，实现两台计算机通过 RS-232 串口相互连接。实现发送和接收字符串的程序，支持互发信息，支持多次发送。

由甲方向乙方发送字符串(如 “Hello World!”)，如格式如 “[SENT 2015-03-26 08:01:15] Hello World!”，并在甲方界面上显示该字符串。同时，在乙方机器上显示 “[RECV 2015-03-26 08:01:33] [SENT 2015-03-26 08:01:15] Hello World!”。

11.2 操作步骤

请按以下步骤完成：

- 1、阅读实验教程课本（见附录），上网搜集相关资料；
- 2、安装串行端口驱动器（附录 4），在主机上安装开发环境；
- 3、编写 RS-232 串口互联程序并在本机调试成功后，安装虚拟机（附录 3）进一步测试。
- 4、实验课上向助教老师领取 RS-232 线一条，连接到相邻两台机箱上。
- 5、运行和调试程序，实现实验需求。
- 6、将 RS-232 连接线交还老师，完成实验报告。

11.3 附录一 RS232 连线实验

第 2 章 物理连接基础实验

2.1 RS-232 连线实验

一、实验要求

- 掌握 RS-232 接口的标准及其特点；
- 学会 RS-232 接口的基本接线方法；
- 能够使用 ComTools 串口测试软件测试接口的连通性。

二、实验条件

- 带有 RS-232 接口的计算机、双绞线、电烙铁、焊锡等；
- ComTools 串口测试软件（可从网上搜索下载）。

三、实验指导

1. RS-232 串口标准

RS-232C 标准是美国电子工业联合会（Electronic Industries Association, EIA）与 BELL 等公司一起开发并于 1969 年公布的通信协议。它允许一个发送设备连接到一个接收设备以传送数据，其原始规范的最大传输速率为 20 Kbps，直接连接的最大物理距离为 15 m。但事实上，现在的应用早已超出这个范围。它常用于连接到一个 Modem，或其他带 RS-232 接口的设备上，也可以直接应用于各种类型计算机之间的简单连接。

由于 RS-232 并未定义连接器的物理特性，因此，目前出现了 DB-25、DB-15 和 DB-9 几种类型的连接器，其针脚的定义各不相同。下面主要介绍目前 PC 上常用的 DB-9 类型的连接器。DB-9 提供异步通信的 9 个信号线，其针脚的排列如图 2.1 所示，针脚的说明如表 2.1 所示。EIA 对 RS-232 的电器特性、逻辑电平和各种信号线功能都作了规定。

在 TXD 和 RXD 上：

- 逻辑 1 (MARK) = -3~-15 V;
逻辑 0 (SPACE) = +3~+15 V。

在 RTS、CTS、DSR、DTR 和 DCD 等控制线上：

- 信号有效 (ON 状态, 高电平) = +3~+15 V;
信号无效 (OFF 状态, 低电平) = -3~-15 V。

2.1 RS-232 连线实验

7

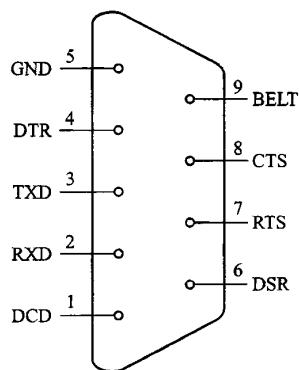


表 2.1 DB-9 的常用信号脚说明

针号	功能说明	缩写
1	数据载波检测	DCD
2	接收数据	RXD
3	发送数据	TXD
4	数据终端准备	DTR
5	信号地	GND
6	数据设备准备好	DSR
7	请求发送	RTS
8	清除发送	CTS
9	振铃指示	BELL

图 2.1 DB-9 连接器示意图

RS-232 串行口信号分为 3 类：传送信号、联络信号和信号地。

传送信号（TXD 和 RXD）：经由 TXD 传送和 RXD 接收的信息格式。一个传送单位由起始位、数据位、奇偶校验位和停止位组成。

联络信号有以下几种：

- RTS：PC 向 Modem 发出的联络信号。ON 表示 PC 请求向 Modem 传送数据。
- CTS：Modem 向 PC 发出的联络信号。ON 表示 Modem 响应 PC 发出的 RTS 信号，且准备向远程 Modem 发送数据。
- DTR：PC 向 Modem 发出的联络信号。ON 表示 PC 处于就绪状态，本地 Modem 和远程 Modem 之间可以建立通信信道；若为 OFF，则强迫 Modem 终止通信。
- DSR：Modem 向 PC 发出的联络信号。它指出本地 Modem 的工作状态，ON 表示 Modem 没有处于测试通话状态，可以和远程 Modem 建立通道。
- DCD：Modem 向 PC 发出的状态信号。ON 表示本地数据通信设备（DCE），接收到远程 Modem 发来的载波信号。
- BELL：Modem 向 PC 发出的状态信号。ON 表示本地 Modem 收到远程 Modem 发来的振铃信号。

信号地（GND）：为相连的 PC 和 Modem 提供同一电势参考点。

2. RS-232 串口通信基本接线方法

通信距离较近时（<15 m），可以用电缆直接连接标准 RS-232 端口，最为简单且常用的是三线制接法，即两个 RS-232 端口的信号地、接收数据和发送数据 3 脚分别相连，接线方法如图 2.2 所示。其他类型的串口，如 DB-25 之间或 DB-9 与 DB-25 之间的连线，如表 2.2 所示，只要记住“接收数据针脚与发送数据针脚相连，彼此交叉，信号地对应相接”原则即可。

另外，为了保证串口物理上可以正常工作，可以制作堵头测试该口是否正常。堵头的制作

很简单，只需将堵头（DB-9 母头）的 2, 3 针短接即可。测试方法为：将堵头插入要测试的串口上，启动 ComTools 串口测试软件，连接以后，在 ComTools 的“发送文本框”中输入字符，若在“接收文本框”中显示出输入的字符，则说明该串口物理功能正常。测试界面如图 2.3 所示。ComTools 同样可以用于测试双机串口互联接线的连通状况。

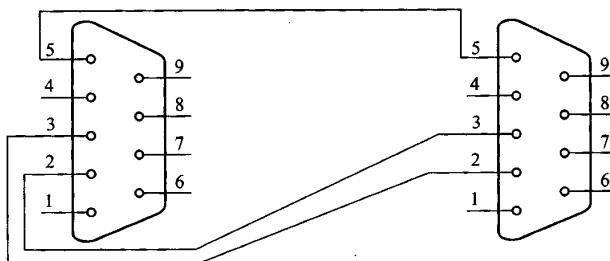


图 2.2 DB-9 基本连线图

表 2.2 不同类型连接器的接线方法

9 针—9 针		25 针—25 针		9 针—25 针	
2	3	3	2	2	2
3	2	2	3	3	3
5	5	7	7	5	7

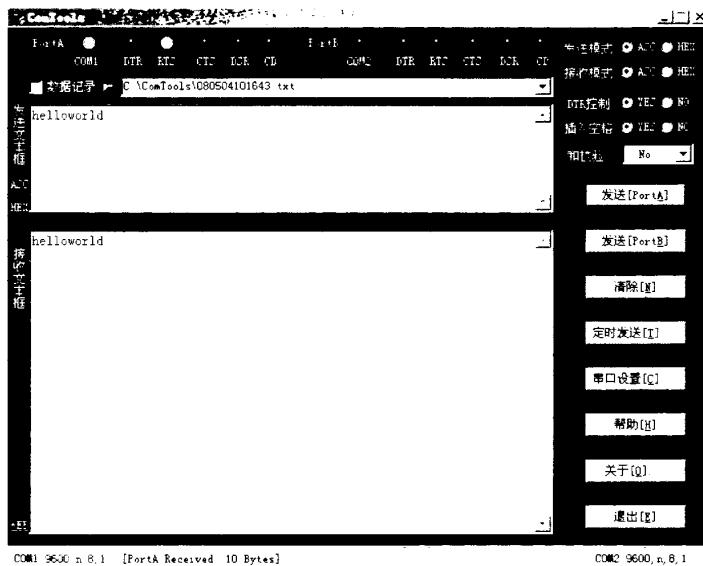


图 2.3 ComTools 测试界面

2.2 RJ-45 接口连线实验

9

如果想在直接连接时，同时启用 RS-232C 的联络控制信号，则采用 7 线制，按照 DTE 和 DCE 之间信息交换协议的要求连接（如图 2.4 所示）。从图 2.4 中可以看出，RS-232C 接口标准定义的所有信号线都用到了。

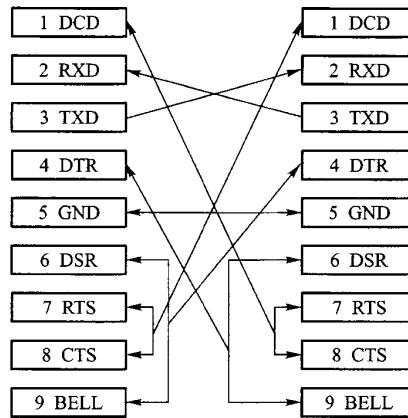


图 2.4 DB-9 7 线制连线图

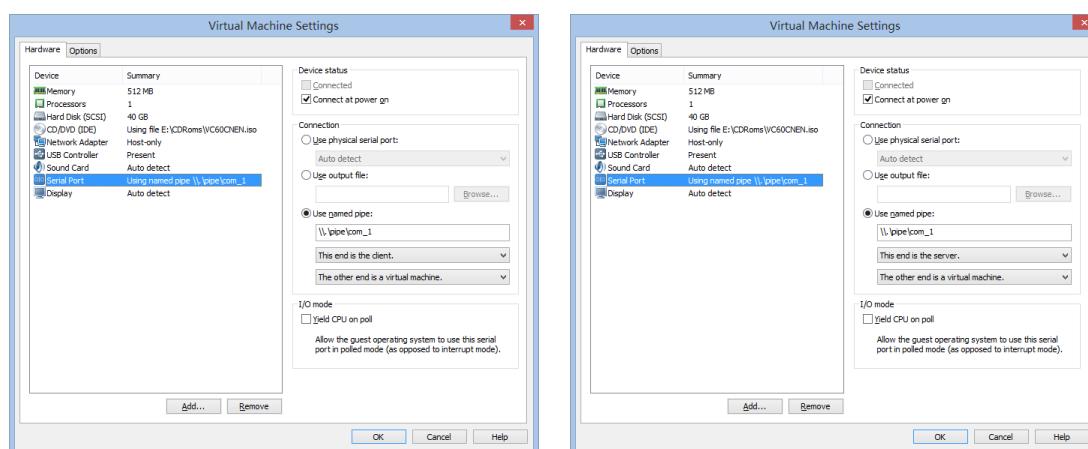
11.4 附录二 .Net 或 Java 串口开发

建议使用 .Net Framework 的 System.IO.Ports.SerialPort 类进行开发。注意：直接使用示例程序不得分。

详见：<https://msdn.microsoft.com/zh-cn/library/system.io.ports.serialport>

11.5 附录三 利用 VMWare 虚拟机仿真

新建两台 Windows 虚拟机（安装一台再复制为另一台），新建串口（Serial Port），选择命名管道模式：Use Named Pipe。选择相同的命名管道路径：一台机器配置成服务器端，另一台配成客户端（This end is the server. This end is the client.）。选择对方端点为虚拟机（The other end is a virtual machine.）。

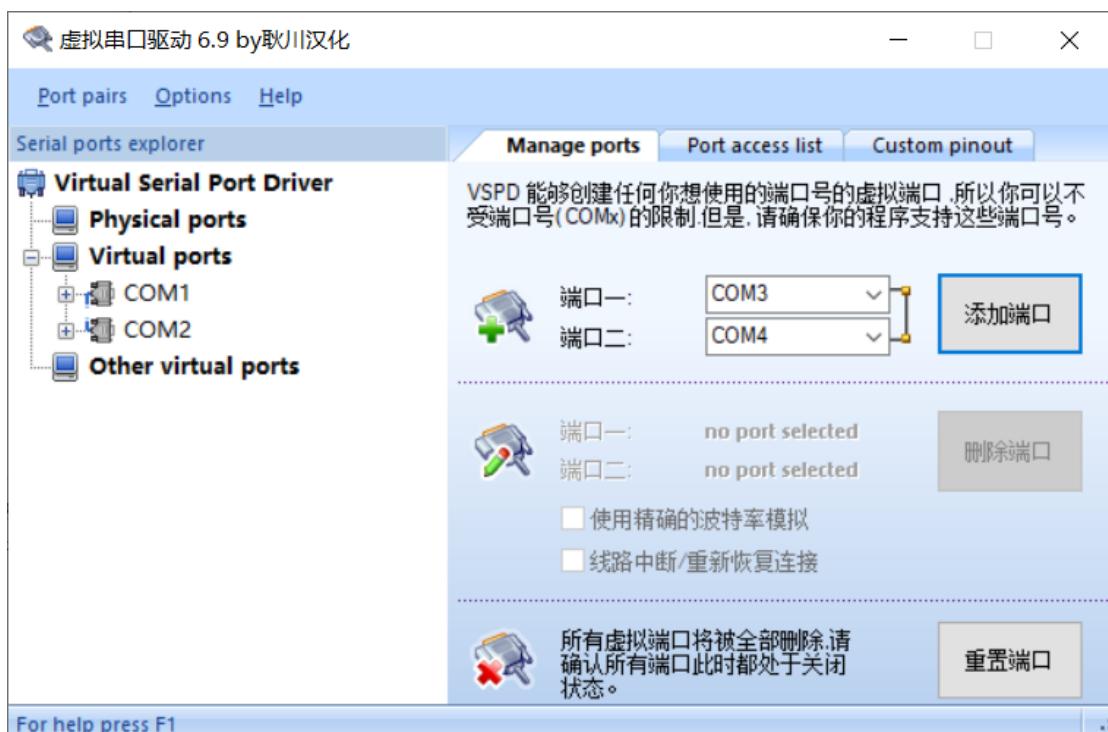


在两台机器上安装 Windows XP（因为该系统的硬件要求较小）或其它操作系统。安装运行程序必须的.NET 或 Java 环境（视实际需要）。建议完全装好一台后拷贝为另外一台以节省时间（记得修改串口配置）。

11.6 附录四 利用 VSPD 虚拟驱动器仿真

对于在一台计算机上同时运行 2 台虚拟机有困难的同学，可以使用 Virtual Serial Port Driver 完成实验的仿真开发。

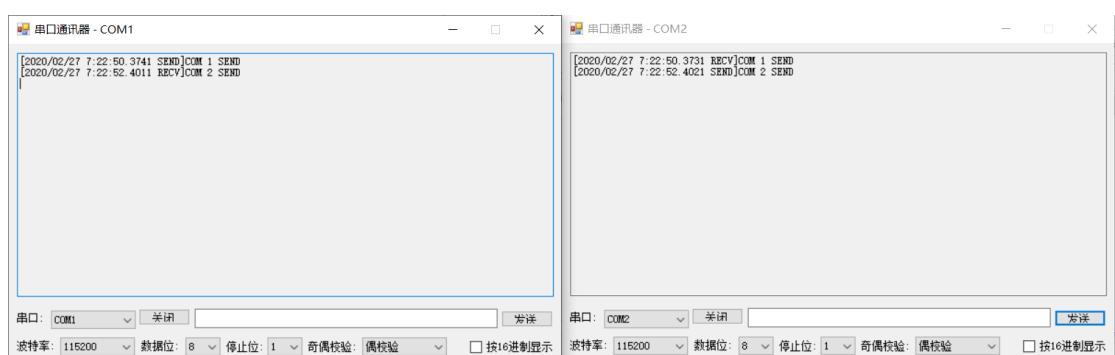
安装完成后，在虚拟串口驱动配套的软件添加端口，软件会自动创建一对互相连接的虚拟串口。即：模拟一条 COM1 和 COM2 都接在自己的电脑上的 RS-232 线。给 COM1 发消息，可以从 COM2 接收到，反之亦然。



开发人员还可以下载串口调试软件 SSCom，来对这两个串口进行发送和接收测试。修改串口参数，对比试验结果。



本次实验即是写一个简易版的串口调试软件。其运行结果，如下图所示。尝试修改串行波特率、每个字节的标准停止位数等参数，观察传输是否能够正确进行。



感谢厦门大学软件工程系 2018 级本科生陈泊霖同学为本实验提出开发和测试环境建议。

12 利用 Socket API 实现网上点对点通信

12.1 实验要求

在 Windows 或 Linux 操作系统（也可以将客户端部署在 Android、iOS 或 WinPhone 手机）下，分别基于 TCP 和 UDP 协议，利用 Socket API 实现网上点对点通信。

程序一“基于 TCP 的可靠文件传输”，功能包括：

在客户端，用户选择本地的某个文件，并发送到服务器端。

在服务器端，接收客户端传输的数据流，并按 IP 地址保存在服务器端（文件名重复的，可以覆盖）。

如果传输过程中服务器端发现客户端断开，服务器端应删除文件，并在屏幕上提示，如“IP：1.2.3.4 发来 abcd.txt 文件过程中失去连接。”。如果客户端发现服务器端不工作，客户端应有提示“服务器 1.2.3.5:62345 失去连接”。

程序二“基于 UDP 的不可靠文件传输”，功能同上，但不能使用 TCP 协议进行传输。考虑如果传输过程中服务器端、客户端如何发现断开。

要求：

(1) 开发环境自选。

(2) 不要选用公用端口号(0~49151)。

(3) 注意对出错进行处理，在传输过程中可以使用任务管理器终止服务器或客户端模拟实现。

用 C 语言实现，开发环境和操作系统自选。建议不要选用公用端口号(0~49151)。

12.2 操作步骤

请按以下步骤完成：

- 1、下载相关课件，阅读实验教程课本（附录 1），上网搜集相关资料，示例代码和文字教程，记住实验要领；

2、按照课本描述课前准备好可以运行的代码，分别基于 TCP 和 UDP 协议，实现双机（或多机）互联。

12.3 附录一：UDP 和 TCP 编程介绍

一、概述

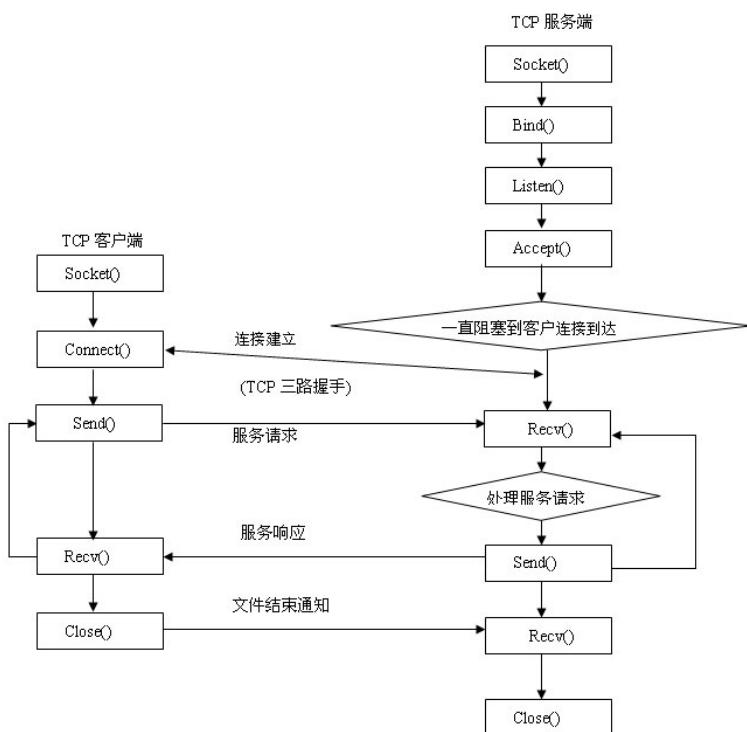
TCP（传输控制协议）和 UDP（用户数据报协议是网络体系结构 TCP/IP 模型中传输层一层中的两个不同的通信协议。

TCP：传输控制协议，一种面向连接的协议，给用户进程提供可靠的全双工的字节流，TCP 套接口是字节流套接口(stream socket)的一种。

UDP：用户数据报协议。UDP 是一种无连接协议。UDP 套接口是数据报套接口(datagram socket)的一种。

二、TCP 和 UDP 介绍

1) 基本 TCP 客户—服务器程序设计基本框架

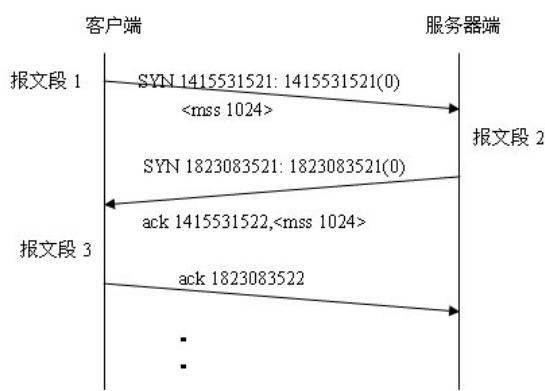


说明：(三路握手)

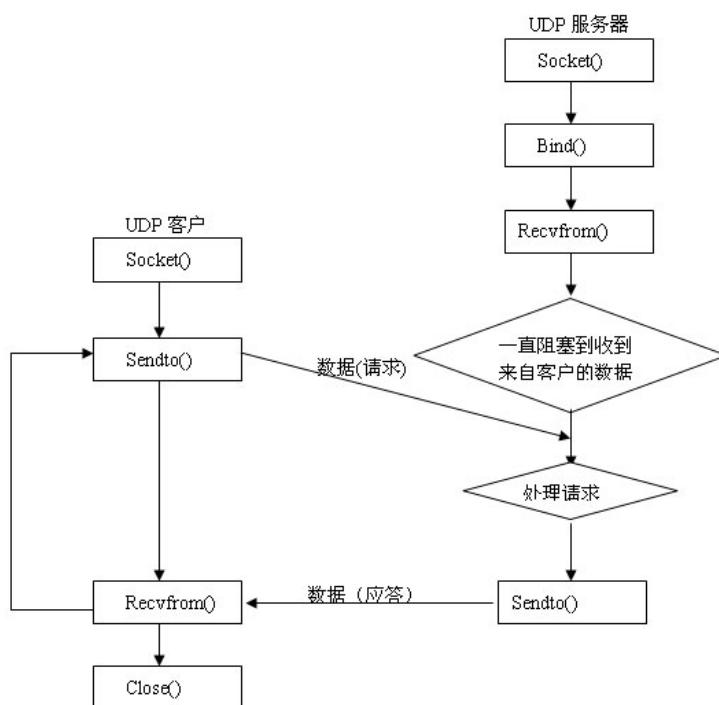
1. 客户端发送一个 SYN 段(同步序号)指明客户打算连接的服务器端口，以及初始化序号(ISN) 。

2. 服务器发回包含服务器的初始序号的 SYN 报文段作为应答。同时，将确认序号(ACK)设置为客户的 ISN 加 1 以对客户的 SYN 报文段进行确认。一个 SYN 将占用一个序号。

3. 客户必须将确认序号设置为服务器的 ISN 加 1 以对服务器的 SYN 报文段进行确认。



2) 基本 TCP 客户—服务器程序设计基本框架流程图



3) UDP 和 TCP 的对比：

从上面的流程图比较我们可以很明显的看出 UDP 没有三次握手过程。

简单点说。UDP 处理的细节比 TCP 少。UDP 不能保证消息被传送到（它也报告消息没有传送到）目的地。UDP 也不保证数据包的传送顺序。UDP 把数据发出去后只能希望它能够抵达目的地。

TCP 优缺点：

优点：

- 1 · TCP 提供以认可的方式显式地创建和终止连接。
- 2 · TCP 保证可靠的、顺序的（数据包以发送的顺序接收）以及不会重复的数据传输。
- 3 · TCP 处理流控制。
- 4 · 允许数据优先
- 5 · 如果数据没有传送到，则 TCP 套接口返回一个出错状态条件。
- 6 · TCP 通过保持连续并将数据块分成更小的分片来处理大数据块。—无需程序员知道

缺点：TCP 在转移数据时必须创建（并保持）一个连接。这个连接给通信进程增加了开销，让它比 UDP 速度要慢。

UDP 优缺点：

- 1 · UDP 不要求保持一个连接
- 2 · UDP 没有因接收方认可收到数据包（或者当数据包没有正确抵达而自动重传）而带来的开销。
- 3 · 设计 UDP 的目的是用于短应用和控制消息
- 4 · 在一个数据包连接一个数据包的基础上，UDP 要求的网络带宽比 TCP 更小。

三、Socket 编程

Socket 接口是 TCP/IP 网络的 API，Socket 接口定义了许多函数或例程，程序员可以用它们来开发 TCP/IP 网络上的应用程序。要学 Internet 上的 TCP/IP 网络编程，必须理解 Socket 接口。

Socket 接口设计者最先是将接口放在 Unix 操作系统里面的。如果了解 Unix 系统的输入和输出的话，就很容易了解 Socket 了。网络的 Socket 数据传输是一种特殊的 I/O，Socket 也是一种文件描述符。Socket 也具有一个类似于打开文件的函数调用 `Socket()`，该函数返回一个整型的 Socket 描述符，随后的连接建立、数据传输等操作都是通过该 Socket 实现的。常用的 Socket 类型有两种：流式 Socket (`SOCK_STREAM`) 和数据报式 Socket (`SOCK_DGRAM`) 。流式是一种面向连接的 Socket，针对于面向连接的 TCP 服务应用；数据报式 Socket 是一种无连接的 Socket，对应于无连接的 UDP 服务应用。

1、socket 调用库函数主要有：

创建套接字

`Socket(af,type@protocol)`

建立地址和套接字的联系

`bind(sockid, local addr, addrlen)`

服务器端侦听客户端的请求

`listen(Sockid ,quenlen)`

建立服务器/客户端的连接（面向连接 TCP）

客户端请求连接

`Connect(sockid, destaddr, addrlen)`

服务器端等待从编号为 Sockid 的 Socket 上接收客户连接请求

`newsockid=accept(Sockid , Clientaddr, paddrlen)`

发送/接收数据

面向连接：`send(sockid, buff, bufflen)`

`recv()`

面向无连接：`sendto(sockid,buff,...,addrlen)`

`recvfrom()`

释放套接字

```
close(sockid)
```

2、TCP/IP 应用编程接口（API）

服务器的工作流程：首先调用 socket 函数创建一个 Socket，然后调用 bind 函数将其与本机地址以及一个本地端口号绑定，然后调用 listen 在相应的 socket 上监听，当 accpet 接收到一个连接服务请求时，将生成一个新的 socket。服务器显示该客户机的 IP 地址，并通过新的 socket 向客户端发送字符串"hi,I am server!"。最后关闭该 socket。

```
main()
{
    int sock_fd, client_fd; /*sock_fd：监听socket；client_fd：数据传输socket */
    struct sockaddr_in ser_addr; /* 本机地址信息 */
    struct sockaddr_in cli_addr; /* 客户端地址信息 */
    char msg[MAX_MSG_SIZE];/* 缓冲区*/
    ser_sockfd = socket(AF_INET, SOCK_STREAM, 0);/*创建连接的SOCKET */
    if (ser_sockfd < 0)
    {/*创建失败 */
        fprintf(stderr, "socket Error:%s\n", strerror(errno));
        exit(1);
    }
    /* 初始化服务器地址*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&ser_addr, addrlen);
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    ser_addr.sin_port = htons(SERVER_PORT);
    if (bind(ser_sockfd, (struct sockaddr*)&ser_addr, sizeof(struct
    sockaddr_in)) < 0)
    { /*绑定失败 */
        fprintf(stderr, "Bind Error:%s\n", strerror(errno));
        exit(1);
    }
    /*侦听客户端请求*/
    if (listen(ser_sockfd, BACKLOG) < 0)
    {
        fprintf(stderr, "Listen Error:%s\n", strerror(errno));
        close(ser_sockfd);
        exit(1);
    }
    while (1)
    {/* 等待接收客户连接请求*/
        cli_sockfd = accept(ser_sockfd, (struct sockaddr*) &cli_addr,
        &addrlen);
        if (cli_sockfd <= 0)
        {
            fprintf(stderr, "Accept Error:%s\n", strerror(errno));
        }
        else
        {/*开始服务*/
            recv(cli_addr, msg, MAX_MSG_SIZE, 0); /* 接受数据*/
            printf("received a connection from %sn",
            inet_ntoa(cli_addr.sin_addr));
            printf("%s\n", msg);/*在屏幕上打印出来 */
        }
    }
}
```

```

        strcpy(msg, "hi,I am server!");
        send(cli_addr, msg, sizeof(msg), 0); /*发送的数据*/
        close(cli_addr);
    }
}
close(ser_sockfd);
}

```

客户端的工作流程：首先调用 socket 函数创建一个 Socket，然后调用 bind 函数将其与本机地址以及一个本地端口号绑定，请求连接服务器，通过新的 socket 向客户端发送字符串" hi,I am client!"。最后关闭该 socket。

```

main()
{
    int cli_sockfd; /*客户端SOCKET */
    int addrlen;
    char seraddr[14];
    struct sockaddr_in ser_addr,/* 服务器的地址*/
        cli_addr; /* 客户端的地址*/
    char msg[MAX_MSG_SIZE]; /* 缓冲区*/
    GetServerAddr(seraddr);
    cli_sockfd = socket(AF_INET, SOCK_STREAM, 0); /*创建连接的SOCKET */
    if (ser_sockfd < 0)
    {/*创建失败 */
        fprintf(stderr, "socket Error:%s\n", strerror(errno));
        exit(1);
    }
    /* 初始化客户端地址*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&ser_addr, addrlen);
    cli_addr.sin_family = AF_INET;
    cli_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    cli_addr.sin_port = 0;
    if (bind(cli_sockfd, (struct sockaddr*)&cli_addr, addrlen) < 0)
    {
        /*绑定失败 */
        fprintf(stderr, "Bind Error:%s\n", strerror(errno));
        exit(1);
    }
    /* 初始化服务器地址*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&ser_addr, addrlen);
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_addr.s_addr = inet_addr(seraddr);
    ser_addr.sin_port = htons(SERVER_PORT);
    if (connect(cli_sockfd, (struct sockaddr*)&ser_addr, &addrlen) != 0)/*请求连
接*/
    {
        /*连接失败 */
        fprintf(stderr, "Connect Error:%s\n", strerror(errno));
        close(cli_sockfd);
        exit(1);
    }
    strcpy(msg, "hi,I am client!");
    send(sockfd, msg, sizeof(msg), 0); /*发送数据*/
    recv(sockfd, msg, MAX_MSG_SIZE, 0); /* 接受数据*/
    printf("%s\n", msg); /*在屏幕上打印出来 */
    close(cli_sockfd);
}

```

```
}
```

3、UDP/IP 应用编程接口（ API ）

服务器的工作流程：首先调用 socket 函数创建一个 Socket，然后调用 bind 函数将其与本机地址以及一个本地端口号绑定，接收到一个客户端时，服务器显示该客户端的 IP 地址，并将字串返回给客户端。

```
int main(int argc, char **argv)
{
    int ser_sockfd;
    int len;
    //int addrlen;
    socklen_t addrlen;
    char seraddr[100];
    struct sockaddr_in ser_addr;
    /*建立socket*/
    ser_sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (ser_sockfd < 0)
    {
        printf("I cannot socket success\n");
        return 1;
    }
    /*填写sockaddr_in 结构*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&ser_addr, addrlen);
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    ser_addr.sin_port = htons(SERVER_PORT);
    /*绑定客户端*/
    if (bind(ser_sockfd, (struct sockaddr *)&ser_addr, addrlen) < 0)
    {
        printf("connect");
        return 1;
    }
    while (1)
    {
        bzero(seraddr, sizeof(seraddr));
        len = recvfrom(ser_sockfd, seraddr, sizeof(seraddr), 0, (struct sockaddr*)&ser_addr, &addrlen);
        /*显示client端的网络地址*/
        printf("receive from %s\n", inet_ntoa(ser_addr.sin_addr));
        /*显示客户端发来的字串*/
        printf("recevce:%s", seraddr);
        /*将字串返回给client端*/
        sendto(ser_sockfd, seraddr, len, 0, (struct sockaddr*)&ser_addr,
               addrlen);
    }
}
```

客户端的工作流程：首先调用 socket 函数创建一个 Socket，填写服务器地址及端口号，从标准输入设备中取得字符串，将字符串传送给服务器端，并接收服务器端返回的字符串。最后关闭该 socket。

```
int GetServerAddr(char * addrname)
{
```

```

    printf("please input server addr:");
    scanf("%s", addrname);
    return 1;
}
int main(int argc, char **argv)
{
    int cli_sockfd;
    int len;
    socklen_t addrlen;
    char seraddr[14];
    struct sockaddr_in cli_addr;
    char buffer[256];
    GetServerAddr(seraddr);
    /* 建立socket*/
    cli_sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (cli_sockfd < 0)
    {
        printf("I cannot socket success\n");
        return 1;
    }
    /* 填写sockaddr_in*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&cli_addr, addrlen);
    cli_addr.sin_family = AF_INET;
    cli_addr.sin_addr.s_addr = inet_addr(seraddr);
    //cli_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    cli_addr.sin_port = htons(SERVER_PORT);
    bzero(buffer, sizeof(buffer));
    /* 从标准输入设备取得字符串*/
    len = read(STDIN_FILENO, buffer, sizeof(buffer));
    /* 将字符串传送给server端*/
    sendto(cli_sockfd, buffer, len, 0, (struct sockaddr*)&cli_addr, addrlen);
    /* 接收server端返回的字符串*/
    len = recvfrom(cli_sockfd, buffer, sizeof(buffer), 0, (struct
    sockaddr*)&cli_addr, &addrlen);
    //printf("receive from %s\n",inet_ntoa(cli_addr.sin_addr));
    printf("receive: %s", buffer);
    close(cli_sockfd);
}

```

12.4 附录二：网络多线程 2048 游戏学生作品

以下代码来自厦门大学软件学院 2014 级本科生黄健明和朱宽同学。

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#define PROTOPORT      5194          /* default protocol
port number */
#define QUENUM 2

```

```
#define INITIALIZE closemark = 0; \
    outputmark = 0; \
    outputmarkc = 0; \
    jud = 0; \
    Rjud = 0;
struct position
{
    int i, j;
};
// 
struct hostent *ptrh; /* pointer to a host table entry */
*/
struct protoent *ptrp; /* pointer to a protocol table entry */
*/
struct sockaddr_in sad; /* structure to hold server's
address */
struct sockaddr_in cad; /* structure to hold client's
address */
int sd, sd2; /* socket descriptors
*/
int port; /* protocol port number
*/
int alen;
char host[100];
int option;
int table[4][4];
int Rtable[4][4];
char closemark, outputmark, outputmarkc;
char jud, Rjud;
// 

int OutputTable(int table[][4])
{
    int i, j;
    for (i = 0; i <= 3; i++)
    {
        for (j = 0; j <= 3; j++)
        {
            if (table[i][j] < 0)
                table[i][j] = -table[i][j];
            printf("%5d ", table[i][j]);
        }
        printf("\n");
    }
}
```

```
    printf("\n");
}
return 0;
}
int RanInsert(int table[][4], int race4)
{
    int k = 0, re = 0, i, j, bloNum;
    struct position blaBlo[16];
    for (i = 0; i <= 15; i++)
    {
        blaBlo[i].i = 0;
        blaBlo[i].j = 0;
    }
    for (i = 0; i <= 3; i++)
        for (j = 0; j <= 3; j++)
            if (table[i][j] == 0)
            {
                blaBlo[k].i = i;
                blaBlo[k].j = j;
                k++;
            }
    if (k > 1)
    {
        re = randomer(k--);
        if (race4 != 0 && randomer(race4) == race4) bloNum =
4;
        else bloNum = 2;
        if (table[blaBlo[re].i][blaBlo[re].j] != 0)
        {
            if (re + 1 < k)
                table[blaBlo[re + 1].i][blaBlo[re + 1].j] =
bloNum;
            else table[blaBlo[re - 1].i][blaBlo[re - 1].j] =
bloNum;
        }
        if (table[blaBlo[re].i][blaBlo[re].j] == 0)
            table[blaBlo[re].i][blaBlo[re].j] = bloNum;
    }
    if (k == 1)
        table[blaBlo[0].i][blaBlo[0].j] = 2;
    if (k < 1)
        printf("No position to insert.\n");
    return 0;
}
```

```
}

int randomer(int n)
{
    int i, j;
    srand((unsigned)(time(NULL)));
    i = ((rand() % 1000) / 100 + (rand() % 100) / 10 +
(rand() % 10)) % n;
    return i;
}

int Rightmove(int table[][4])
{
    int k, i, j;
    for (k = 0; k <= 3; k++) //check loop.
        for (j = 2; j >= 0; j--)
            for (i = 3; i >= 0; i--) //upmove every available
block;
            if (table[i][j] != 0)
            {
                if (table[i][j + 1] == 0)
                {
                    table[i][j + 1] = table[i][j];
                    table[i][j] =
0;/**debug:*/printf("upmove:%d %d\n",i,j);
                }
                if ((table[i][j + 1] > 0) && (table[i][j + 1] == table[i][j]) && (table[i][j] > 0))
                {
                    table[i][j + 1] = -(table[i][j + 1] +
table[i][j]);
                    table[i][j] =
0;/**debug:*/printf("absorb:%d %d\n",i,j);
                }
            }
        return 0;
}

int Leftmove(int table[][4])
{
    int k, i, j;
    for (k = 0; k <= 3; k++) //check loop.
        for (j = 1; j <= 3; j++)
            for (i = 3; i >= 0; i--) //upmove every available
block;
            if (table[i][j] != 0)
```

```
{  
    if (table[i][j - 1] == 0)  
    {  
        table[i][j - 1] = table[i][j];  
        table[i][j] =  
0;//debug:printf("upmove:%d %d\n",i,j);  
    }  
    if ((table[i][j - 1] > 0) && (table[i][j -  
1] == table[i][j]) && (table[i][j] > 0))  
    {  
        table[i][j - 1] = -(table[i][j - 1] +  
table[i][j]);  
        table[i][j] =  
0;//debug:printf("absorb:%d %d\n",i,j);  
    }  
}  
return 0;  
}  
int Downmove(int table[][4])  
{  
    int k, i, j;  
    for (k = 0; k <= 3; k++) //check loop.  
        for (i = 2; i >= 0; i--)  
            for (j = 3; j >= 0; j--) //downmove every  
available block;  
            if (table[i][j] != 0)  
            {  
                //debug:printf("check:%d %d\n",i,j);  
                if (table[i + 1][j] == 0)  
                {  
                    table[i + 1][j] = table[i][j];  
                    table[i][j] =  
0;//debug:printf("upmove:%d %d\n",i,j);  
                }  
                if ((table[i + 1][j] > 0) && (table[i +  
1][j] == table[i][j]) && (table[i][j] > 0))  
                {  
                    table[i + 1][j] = -(table[i + 1][j] +  
table[i][j]);  
                    table[i][j] =  
0;//debug:printf("absorb:%d %d\n",i,j);  
                }  
            }  
}
```

```
    return 0;
}
int Upmove(int table[][4])
{
    int k, i, j;
    for (k = 0; k <= 3; k++) //check loop.
        for (i = 1; i <= 3; i++)
            for (j = 3; j >= 0; j--) //upmove every available
block;
            if (table[i][j] != 0)
            {
                if (table[i - 1][j] == 0)
                {
                    table[i - 1][j] = table[i][j];
                    table[i][j] =
0;//debug:printf("upmove:%d %d\n",i,j);
                }
                if ((table[i - 1][j] > 0) && (table[i -
1][j] == table[i][j]) && (table[i][j] > 0))
                {
                    table[i - 1][j] = -(table[i - 1][j] +
table[i][j]);
                    table[i][j] =
0;//debug:printf("absorb:%d %d\n",i,j);
                }
            }
    return 0;
}
int Judgement(int table[][4])
{
    int jud = 0, i, j;
    for (i = 0; i <= 3; i++)
        for (j = 0; j <= 3; j++)
    {
        if (table[i][j] == 2048)
        {
            jud = 2;//Seek for 2048 block.
            break;
        }
        if (table[i][j] == 0)
        {
            jud = 1;
            break;//Seek for blank block.
        }
    }
}
```

```
    }

    if (((table[i][j] == table[i - 1][j]) && (i - 1 >= 0)) || ((table[i][j] == table[i + 1][j]) && (i + 1 <= 3)) ||
    ((table[i][j] == table[i][j - 1]) && (j - 1 >= 0)) ||
    ((table[i][j] == table[i][j + 1]) && (j + 1 <= 3)))
    {
        jud = 1;
        break; //Seek for possible absorbing.
    }
}

return jud;
}

// Socket.
int Server()
{
    memset((char *)&sad, 0, sizeof(sad)); /* clear sockaddr
structure */
    sad.sin_family = AF_INET;           /* set family to
Internet */
    sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP
address */
    port = PROTOPORT;
    printf("Port:(Can be ignored):");
    if (scanf("%d", &port) == 1)
    {
        if (port <= 0)                  /* test for illegal
value */
        {
            /* print error message and
exit */
            fprintf(stderr, "bad port number %d\n", port);
            return -1;
        }
    }
    else fflush(stdin);
    sad.sin_port = htons((u_short)port);
/*if ( ((int)(ptrp = getprotobynumber("tcp")))) == 0)
{
    fprintf(stderr, "cannot map \"tcp\" to protocol number");
    return -2;
}*/
```

```
ptrp = getprotobyname("tcp");
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0)
{
    fprintf(stderr, "socket creation failed\n");
    return -3;
}
if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0)
{
    fprintf(stderr, "bind failed\n");
    return -4;
}
if (listen(sd, QUENUM) < 0)
{
    fprintf(stderr, "listen failed\n");
    return -5;
}

while (1)
{
    alen = sizeof(cad);
    if ((sd2 = accept(sd, (struct sockaddr *)&cad,
&alen)) < 0)
    {
        fprintf(stderr, "accept failed\n");
        return -6;
    }
    else break;
    /*           pthread_t ID;

pthread_create(&ID,NULL,sendmessage,sd2);*/
}
}

void TableRecv()
{
    do{
        if (recv(option == 1 ? sd2 : sd, Rtable,
sizeof(Rtable), 0) > 0)
        {
            outputmarkc = 1;
        }
        if (recv(option == 1 ? sd2 : sd, &Rjud, sizeof(Rjud),
0) > 0)
```

```
{  
    outputmarkc = 1;  
}  
if (closemark)  
    break;  
} while (1);  
}  
void TableOut()  
{  
    int i, j;  
    char co[100];  
    while (1)  
    {  
        if (outputmark || outputmarkc)  
        {  
            TableSend();  
            system("clear");  
            printf("localplayer           remoteplayer\\n");  
            for (i = 0; i < 4; ++i)  
            {  
                if (jud == 0)  
                {  
                    for (j = 0; j < 4; ++j)  
                    {  
                        if (table[i][j] < 0)  
                            table[i][j] = -table[i][j];  
                        printf("%5d", table[i][j]);  
                    }  
                    printf("      ");  
                }  
                else if (jud == -1)  
                {  
                    printf("      You lose!  
");  
                    break;  
                }  
                else  
                {  
                    printf("      You win!  
");  
                    break;  
                }  
                if (Rjud == 0)
```

```
{  
    for (j = 0; j < 4; ++j)  
    {  
        printf("%5d", Rtable[i][j] > 0 ?  
Rtable[i][j] : (-Rtable[i][j]));  
    }  
}  
else if (jud == -1)  
{  
    printf("Opponent lose!");  
    break;  
}  
else  
{  
    printf("Opponent win!");  
    break;  
}  
printf("\n\n");  
if (jud || Rjud)  
    closemark = 1;  
}  
outputmark = outputmarkc = 0;  
printf("Please enter 'w' to upmove,'s'to  
downmove,'a'to leftmove,'d'to rightmove,'q'to quit.\n");  
//printf("Command:");  
co[0] = 0;  
/*if (fscanf(stdin,"%s",co) == 1)  
    printf("%s",co);*/  
}  
if (closemark)  
    break;  
}  
}  
int Client()  
{  
    memset((char *)&sad, 0, sizeof(sad)); /* clear sockaddr  
structure */  
    sad.sin_family = AF_INET; /* set family to  
Internet */  
    sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP  
address */  
    port = PROTOPORT;  
    printf("Port:(Can be ignored):");
```

```
if (scanf("%d", &port) == 1)
{
    if (port <= 0) /* test for illegal
value */           /* print error message and
exit */
        fprintf(stderr, "bad port number %d\n", port);
        return -1;
}
else fflush(stdin);
sad.sin_port = htons((u_short)port);
printf("IP:");
scanf("%s", host);
ptrh = gethostbyname(host);
if (((char *)ptrh) == NULL) {
    fprintf(stderr, "invalid host: %s\n", host);
    return -2;
}
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
/*if ( ((int)(ptrp = getprotobynumber("tcp")) == 0) {
    fprintf(stderr, "cannot map \"tcp\" to
protocol number");
    return -3;
} */
ptrp = getprotobynumber("tcp");
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
    fprintf(stderr, "socket creation failed\n");
    return -4;
}

/* Connect the socket to the specified server. */

if (connect(sd, (struct sockaddr *)&sad, sizeof(sad)) <
0) {
    fprintf(stderr, "connect failed\n");
    return -5;
}

/* Repeatedly read data from socket and write to user's
screen. */
/*      pthread_t ID;
```

```
    pthread_create(&ID, NULL, sendmessage, sd);*/
/* Close the socket. */
}

int game2048()
{
    char command = 0; int i, j, k = 0, re = 0, loop, race4 =
5, temp;
    for (i = 0; i <= 3; ++i)
        for (j = 0; j <= 3; j++)
            table[i][j] = 0;
//Give numbers to two blocks randomly.
    for (i = 1; i <= 2; ++i)
        RanInsert(table, 0);
//output the table.
//OutputTable(table);
    outputmark = 1;
//Ask to enter command.
//printf("Please enter 'w' to upmove,'s'to downmove,'a'to
leftmove,'d'to rightmove,'q'to quit.\n");
//Game.
do
{
    //printf("Command:");
    command = getchar();
    if (command == '\n') command = getchar();
    if ((command != 'w') && (command != 'a') &&
(command != 's') && (command != 'd') && (command != 'q') &&
(command != '4'))
    {
        printf("invalid instruction\n");
        continue;
    }
//Movements.
    if (command == 'w')//upmove.
        Upmove(table);
    if (command == 'a')//leftmove.
        Leftmove(table);
    if (command == 'd')//rightmove.
        Rightmove(table);
    if (command == 's')//downmove.
        Downmove(table);
    if (command == 'q') return 0;
    if (command == '4')
```

```
{  
    scanf("%d", &race4);  
    if (race4 < 2)  
        race4 = 5;  
}  
//insert a new block.  
RanInsert(table, race4);  
//output the table.  
//PrintHead();  
outputmark = 1;  
TableSend();  
//W or L judgement  
if ((temp = Judgement(table)) == 0)  
{  
    jud = -1;  
    outputmark = 1;  
    return 0;  
}  
if (temp == 2)  
{  
    jud = 1;  
    outputmark = 1;  
    return 0;  
}  
} while (command != 'q');  
while (1)  
    if (closemark)  
        return 0;  
}  
  
int TableSend()  
{  
    if (option == 1)  
    {  
        send(sd2, table, sizeof(table), 0);  
        send(sd2, &jud, sizeof(jud), 0);  
    }  
    else  
    {  
        send(sd, table, sizeof(table), 0);  
        send(sd, &jud, sizeof(jud), 0);  
    }  
}  
int main()
```

```
{  
    printf("1 - server.\n2 - client.\nTo choose a mode.\n");  
    scanf("%d", &option);  
    switch (option)  
    {  
        case 1:Server();  
            break;  
        case 2:Client();  
    }  
    pthread_t TID[2];  
    pthread_create(&TID[0], NULL, TableRecv, NULL);  
    pthread_create(&TID[1], NULL, TableOut, NULL);  
    INITIALIZE  
    game2048();  
    close(sd);  
    if (option == 1)  
        close(sd2);  
}
```