高等学校本科应用型教材

十五

UNIX操作系统教程

——管理与编程

刘 循

高等教育出版社

高等学校本科应用型教材

UNIX 操作系统教程

——管理与编程

刘 循

高等教育出版社

内容提要

这是一本全面且实用的 UNIX 操作系统教材。该教材在介绍 UNIX 操作系统基本概念及基本使用的基础上，全面、深入地讲述了 UNIX 操作系统的系统管理和程序开发。在系统管理部分除了传统的管理内容外，还详细介绍了网络服务管理及配置。程序开发部分从操作系统的 Shell 编程到 C 语言编程（重点是系统调用），都作了理论和实例讲解。

本书既可以作为高等院校计算机及相关专业本科学生及研究生的教材，也可供从事 UNIX 平台的网络管理、网络服务及软件开发人员阅读参考。

出版发行　高等教育出版社　　购书热线　010 － 64054588
社　　址　北京市西城区德外大街 4 号　　免费咨询　800 － 810 － 0598
邮政编码　100011　　网　　址　http:// www.hep.edu.cn
总　　机　010 － 82028899　　http:// www.hep.com.cn

经　　销　新华书店北京发行所
印　　刷

开　　本　787×960　1/16　　版　　次　　年　月第 1 版
印　　张　26　　印　　次　　年　月第　次印刷
字　　数　490 000　　定　　价　32.30 元

策划编辑　刘建元
责任编辑　韩　飞
封面设计　刘晓翔
责任印制

前　言

UNIX 操作系统是一种多用户多任务的分时操作系统。该系统以其简洁方便、安全可靠、功能强大、开放性和可移植性等特点，成为小型机和工作站的主流操作系统，特别是在当前计算机网络和通信广泛应用的环境下，UNIX 系统承担着网络服务器和通信业务提供者的角色，得到了业界的高度重视。

作为操作系统，UNIX 的基本使用及基本命令已逐渐被大家所了解，而当前人们迫切需要的是如何更好地管理 UNIX 系统、更好地提供 UNIX 系统服务以及如何在 UNIX 系统上开发应用软件。基于此，本教材的主要内容包括三个部分：UNIX 操作系统基本概念及基本使用、UNIX 操作系统系统管理、UNIX 操作系统程序开发环境和程序开发。UNIX 操作系统基本概念及基本使用将在本书第一章中作简单明了的介绍，本书的重点是 UNIX 操作系统系统管理及系统程序开发，这两部分在书中占了大量篇幅。

UNIX 操作系统系统管理包括在第 2、3、4、8、10、11、12 章，分别就 UNIX 的文件系统管理、系统管理、设备管理、进程管理及进程通信、UNIX 窗口系统、UNIX 网络及其管理、UNIX 安全等内容，作了由浅入深、详细、全面地讲解，并附有操作实例，体现了该书系统管理的思想。

UNIX 操作系统程序开发部分包括第 5、6、7、9 章，讲述了 Shell 及其编程、UNIX 实用程序、UNIX 软件开发工具及 UNIX 系统调用等内容。每一部分除了从理论上讲解外，还通过编程实例深入说明，使读者更易于掌握。

作为教材，本书内容针对的是通用 UNIX 环境，部分实例以 IBM 公司的 AIX 系统和 SUN 公司的 Solaris 系统为环境。读者应具有一定的计算机知识和 C 语言程序设计知识。为了配合学习及上机，每章后附有上机练习和习题，可以提供上机内容和课后练习。另外在每章的开始部分还有每章主要内容，帮助读者更好地掌握每一章的重点。本书可作为高等院校计算机或相关专业本科生及研究生的教材，也可供在 UNIX 平台上从事网络管理、软件开发和系统程序设计人员阅读参考。

书中内容如有错误或不妥之处，恳请指正。

作者

2003 年 9 月

目　录

第 1 章　UNIX 系统概述及基本使用

UNIX 能够从一个最初的提供软件开发环境的文件系统，发展为今天计算机界首选的重要操作系统，其根本原因在于 UNIX 所具有的特性：开放、可扩展、多用户、方便、网络、安全、标准化。UNIX 系统中实现的一些思想和方法至今仍然是计算机操作系统中的典范。

本章主要内容

- UNIX 系统介绍：主要介绍 UNIX 操作系统的起源、组成及特点。
- UNIX 操作系统基本使用及基本命令：介绍系统管理员与普通用户如何进入系统、退出系统、相互之间如何切换；在用户进入系统之后的一些基本命令的使用。
- vi 编辑器：介绍全屏幕编辑器 vi 的使用方法。

1.1　UNIX 操作系统概述

1.1.1　UNIX 操作系统简介

1970 年前后，AT&T 公司的贝尔实验室进行着一项“太空旅行实验”，该试验的目的是利用计算机模拟太阳系天体的运转，试验初期选用的是 GE 大型计算机。由于在大型计算机上进行该项实验价格非常昂贵，因此便将该项实验移植到 PDP-7 小型机环境中。

在 PDP-7 环境中，除了方便用户之间共享程序开发环境之外，更加看重用户文件的相互共享，所以最初的 UNIX 操作系统雏形又被看成是一个文件系统。

UNIX 操作系统“诞生”之初，正逢美国反垄断之风盛行，再加上它是以程序员应用需要为目的的，所以 UNIX 操作系统是公开的、开放的、不追逐利润的。只要需要，只要感兴趣，附上邮费和存储介质费就可以得到它。各大计算机公司和各高校纷纷加入 UNIX 系统应用、扩充及完善的行列，产生了 UNIX 系统的两大派别：SYSV 和 BSD 版本。SYSV 是由贝尔实验室推出的；BSD 由加州大学 Berkeley 分校

开发，网络协议首先在 BSD 版本上实现。

在 UNIX 的发展历程中，许多商家将 UNIX 系统与自身生产的硬件平台结合，形成了自己的 UNIX 系统，其中最有名的是 IBM 的 AIX 系统和 Sun 的 Solaris 系统。

当 UNIX 系统中实现了网络功能及网络服务后，UNIX 系统便随着 Internet 走向全世界。

1.1.2　UNIX 操作系统主要组成

UNIX 操作系统主要由内核、Shell 与系统调用、应用三部分组成，如图 1.1 所示。

图 1.1　UNIX 系统组成

内核是直接附在计算机硬件平台上的首层软件，是 UNIX 操作系统的核心和基础。它控制和管理系统的硬件和软件资源，为应用提供方便、有效、可靠和安全的环境。

在内核之上是 Shell 和系统调用。Shell 也称为外壳，是 UNIX 系统的命令解释器。终端用户通过 Shell 以命令方式或 Shell 程序方式使用内核提供的系统环境，与一般系统的命令解释器不同的是：UNIX 系统的 Shell 还具有程序语言能力，是一种结构化程序，用户可以利用 Shell 编制脚本程序，完成一些程序开发功能，这是 UNIX 系统一个最突出的优势。系统调用是 UNIX 提供给应用程序的使用接口，在用户应用程序中可以以函数调用方式使用系统调用，相当于在用户主程序中通过系统调用进入核心，直接使用系统资源。

在系统组成最上面的是应用。应用包括终端用户的应用和应用程序用户的应用。终端用户通过命令方式或以 Shell 脚本方式使用系统资源。应用程序用户通过系统调用方式使用系统资源。

与其他操作系统一样，UNIX 操作系统也提供了图形界面，如 X 窗口（X-Window）和公用桌面环境（CDE）。用户也可以通过图形窗口方便地使用系统。

1.1.3　UNIX 操作系统特点

UNIX 操作系统能够经历几十年长盛不衰，得益于其出色的特点，归纳起来主

要有如下几点：

1. 多任务系统

UNIX 提供后台及前台作业，使一个用户能同时处理多个任务。

2. 多用户共同使用环境

UNIX 允许多个用户通过各自的终端独立交互使用同一系统，共享系统中的各类资源。

3. 文件系统

在文件系统上最明显的特征是提供以下新的文件操作功能。

(1) 用户文件环境：每个系统用户有独立的文件目录环境，并有相应的安全保护措施。

(2) 符号链接（Symbolic Link）：跨过文件系统接口，在不同的文件系统内加以连接。

(3) 虚拟文件交换（Virtual File Switch）：使系统可以同时处理不同类型的文件系统。

(4) 内存映像文件（Memory-Mapped File）：改进系统的输入/输出缓冲及分页动作。

4. 系统调用

UNIX 操作系统通过一组系统调用语句为用户程序访问系统资源提供服务。对应于每个系统调用都有一个能完成特定功能的子程序，使得低级语言或高级语言编写的应用程序都能使用操作系统提供的系统调用。

5. 良好的开放性和可移植性

良好的开放性和可移植性是 UNIX 迅速发展、持久使用的根本原因之一。

UNIX 无论在硬件还是软件上均具有良好的开放性。它可以安装在 PC 机上，也可以安装在超级计算机上，能与各种计算机的硬件环境兼容；不同厂家的应用软件在 UNIX 平台上运行良好，可以满足用户的应用需要。

UNIX 的内核非常简洁，功能结构实现了模块化，各模块可以单独编译，编译后即可以与其他模块装配在一起构成新的内核，内核代码绝大多数都是用易于掌握、易于移植的高级语言——C 语言编写的。这样的内核显然使得其操作系统具有可移植性。

6. 丰富的软件开发环境

与大多数商用操作系统不同，UNIX 诞生在从事通信软件开发的贝尔实验室，天生的、丰富的、独特的软件开发环境，提供给软件开发人员的不仅有文本编辑器、强有力的调试工具、操作系统资源的调用，还有独立的用户工作环境、安全的个人工作目录、相互之间公共文件的共享，这是其他操作系统所难以企及的。

7. 网络环境

有人说：“UNIX 使网络走向全世界。”又有人说：“网络使 UNIX 走到今天。”不管如何，UNIX 和网络之间的关系确实是密不可分的。

在 UNIX 环境下实现的 TCP/IP 协议，是系统与网络连接的优选环境。在 UNIX 环境下配置的网络服务器把信息带给 Internet 的每一个客户。除服务器之外，UNIX 系统中还配备有其他网络协议，如 APPC、IPX/SPX、ATM、ISDN、X.25、SNA 等。

8. 标准化

IEEE 标准化组织早在 1986 年就针对 UNIX 的核心提出了“1003.1 Portable Operating system Standard for Computer Environments”（POSIX）标准。

1.2　基本使用及基本命令

UNIX 是多用户操作系统，每个用户在操作系统中都有惟一的“姓名”，该“姓名”就是操作系统的用户帐号（又被称为用户名），它是用户使用系统的凭证。在用户进入系统后，通常情况下就可以通过系统提供的命令使用系统。

1.2.1　用户与系统管理员

UNIX 系统中主要的用户角色有系统管理员与普通用户。系统管理员（System Manager，也称为 System Administrator），主要负责系统的安装、维护、用户管理、系统文件管理等。

超级用户（用户帐号为 root）具有系统的最高权限，其主目录为根目录，因此用户帐号 root 不能修改。

通常情况下，系统管理员是以超级用户（root）的身份行使系统管理权力，即以 root 帐号进入系统。

除超级用户之外就是操作系统的普通用户，普通用户只能使用系统。

1.2.2 进入与退出系统

1. 硬件相连

UNIX 系统可以与多台 UNIX 终端相连，提供多个用户同时登录并使用系统，共享系统资源。目前终端与主机连接的方式主要有：

网络：通过 Internet 提供的 TCP 应用 telnet 登录到系统。

串口：本地终端与主机上的 RS-232 串口连接。

2. 进入系统

在硬件连接好之后，用户进入系统之前，必须知道用户帐号和用户密码。另外，通过网络与系统相连的用户还需知道所使用的系统主机名（或 IP 地址），这样才能登录到主机。

当终端打开之后或 telnet 登录到主机系统后，用户屏幕显示系统提示符：

login:

输入用户帐号后出现：

passwd:

输入用户密码。

如果用户帐号和用户密码准确无误，则成功进入系统，出现系统提示符“ $ ”或“ % ”，如果是超级用户则出现系统提示符“ # ”。

下面是用户 yaomim 进入 AIX 系统时的信息：

```
AIX Version 4
(C) Copyrights by IBM and by others 1982, 1996.
login: yaomim
yaomim's Password:
*******************************************************************
*                                                                 *
*   Welcome to AIX Version 4.3!                                   *
*                                                                 *
*   lease see the README file in /usr/lpp/bos for information pertinent to   *
*   this release of the AIX Operating System.                     *
*                                                                 *
*******************************************************************
```



```
Last unsuccessful login: Mon Jan  6 12:02:42 TAIST 2003 on /dev/pts/8 from 211.1
63.139.177
```

Last login: Fri Mar 21 21:31:08 TAIST 2003 on /dev/pts/1 from 211.163.138.21

$

如果用户名或用户密码不正确，系统将出现如下信息：

login incorrect

表示用户帐号不能进入系统，仔细检查用户帐号或用户密码是否拼错。UNIX 系统要区分字母的大小写。

3. 退出系统

当用户完成所做的工作后离开系统，称为退出系统。退出系统的方法是在系统提示符后键入 exit、logout 或 Ctrl+D 键。如：

%exit↵

用户退出系统后出现：

login:

供用户再次进入系统使用。

用户退出系统后不仅可以让出终端给其他用户使用，节约系统资源；而且可以避免系统记帐日志继续记录及用户帐号被他人利用，用户文件遭到破坏。

注意　exit、logout、Ctrl+D 三种退出方式的区别是：logout 是用户这次使用环境注销；exit 和 Ctrl+D 是退出这次特定的 Shell 进程。用户可以在 C-Shell 运行环境参数设置文件.cshrc 中设定。

1.2.3　在用户之间切换

用户进入系统后，如果要切换到其他用户继续使用系统，可以用 su 操作。从切换用户退回到原用户用 exit 命令。

例　以用户帐号 stu01 的身份进入系统，之后切换到 stu02 用户帐号。

login:stu01

passwd:xxxxxx

$who am I

stu01　　tty0　　Dec 11 8:17

$su stu02

$who am I

stu02　　tty0　　Dec 11 8:17

$exit　

$who am I

```
stu01        tty0        Dec 11 8:18
$
```

从普通用户切换到超级用户 root：

```
%su↵
passwd:******↵  （输入 root 的密码）
```

如果密码正确，则成功切换到 root，出现系统提示符：

```
#
```

从超级用户 root 切换到普通用户：

```
#su jlwang↵        （切换到用户帐号为 jlwang，此时不需要用户密码）
```

出现普通用户系统提示符：

```
%
```

则成功切换到 jlwang。

1.2.4 基本命令

UNIX 系统终端用户基本上都是通过网络登录到系统的，在 UNIX 中用户应用最多的方式是操作系统的命令。与其他操作系统不同，掌握操作系统应用命令对 UNIX 用户尤其重要。

UNIX 操作系统给用户提供了丰富的命令供用户使用系统。这些命令也叫作 Shell 命令，在 Shell 提示符下输入。

命令通用格式如下：

```
command [-options] [arguments]
```

其中 command 是命令的名称，选项 options 是对命令作用的要求，选项 arguments 是描述命令作用的对象的命令参数。

下面逐一介绍主要的 Shell 命令。

1. who 命令

该命令用于查看当前登录到系统的用户信息。命令格式：

```
who [-ablqsu]
```

其中选项：

a：处理/etc/utmp 文件或者指定文件；

b：显示系统最近启动的时间和日期；

l：显示系统中登录的终端；

q：显示本地系统上的用户名称和用户总数；

s：显示登录用户名、终端号、日期和时间；

u：显示此时在系统中的用户。

例　$who

```
stu01        tty0        Dec 11 8:17
stu02        tty1        Dec 11 8:19
stu03        tty4        Dec 11 8:29
stu05        tty7        Dec 11 8:24
$
```

命令输出的第一列是用户名；第二列是用户连接的终端的名称（tty 是 teletype 简写，终端的一个称呼）；第三列是用户登录日期及时间。

例　显示本终端用户信息。

```
$who am I
stu01        tty0        Dec 11 8:17
$
```

2. uname 命令

该命令显示正在使用的 UNIX 系统信息。命令格式：

uname [-ranuv]

其中选项：

r：显示操作系统的发行号（Release Number）；

a：打印出所有信息（All Information）；

n：显示网络上本机的节点名（Node Name）；

u：显示系统的序列号（Serial Number）；

v：显示操作系统的版本号（Version Number）。

例　显示操作系统的发行号。

```
$uname -r
5.8
$
$uname -a
AIX rs6000 3 4 000056437200
$
```

3. date 命令

该命令显示或设置此时系统的时间。显示系统时间的命令格式：

date [+%adDhHjmMrSTwWy]

其中选项：

a：以 Sun~Sat 表示星期几；

d：以 01~31 表示日期；

D：以 mm/dd/yy 表示日期；

h：以 Jan~Dec 表示月份；

H：以 00~23 表示小时；

j：指明是一年中的第几天；

m：以 01~12 表示月份；

M：以 00~59 表示分钟；

r：表示 AM/PM；

S：以 00~59 表示秒钟；

T：以 HH:MM:SS 表示输出时间；

w：以 0~6 表示星期几，星期天为 0；

W：指明是一年中的第几周（以星期一作为一周的第一天）；

y：以 00~99 表示年的后两位。

例 $date +%a

Sat

$

命令结果日期是：星期六。

例 $date

Sat Aug 23 11:36:27 ROC 1998

$

命令执行结果是：当前日期是 1998 年 8 月 23 日，星期六，时间是 11 点 36 分 27 秒。

系统管理员可以设置系统时间。将系统时间设置为现在时间的命令格式为：

date [current date]

例 用 date 命令设置系统现在时间为 10 月 20 日 12 点 23 分。

#date 10201223

4. cal 命令

该命令在屏幕上打印出万年历。命令格式：

cal [month] [year]

其中选项：

month：表示月份 1~12。

year：表示年份 1~9999。

例　打印出 2003 年 3 月的日历。

```
$cal 3 2003
          March 2003
Sun Mon Tue Wed Thu Fri Sat
                            1
 2   3   4   5   6   7   8
 9  10  11  12  13  14  15
16  17  18  19  20  21  22
23  24  25  26  27  28  29
30  31
$
```

打印出 2003 年的日历：

```
$cal 2003
                              2003

          January                        February
Sun Mon Tue Wed Thu Fri Sat    Sun Mon Tue Wed Thu Fri Sat
                1   2   3   4                          1
 5   6   7   8   9  10  11     2   3   4   5   6   7   8
12  13  14  15  16  17  18     9  10  11  12  13  14  15
19  20  21  22  23  24  25    16  17  18  19  20  21  22
26  27  28  29  30  31        23  24  25  26  27  28

                              ……

          November                       December
Sun Mon Tue Wed Thu Fri Sat    Sun Mon Tue Wed Thu Fri Sat
                            1        1   2   3   4   5   6
 2   3   4   5   6   7   8     7   8   9  10  11  12  13
 9  10  11  12  13  14  15    14  15  16  17  18  19  20
16  17  18  19  20  21  22    21  22  23  24  25  26  27
23  24  25  26  27  28  29    28  29  30  31
30
$
```

5. echo 命令

该命令用于回显（在屏幕上显示）输入内容。命令格式：

echo xyz

xyz 为要在屏幕上显示的内容。

例 $echo Welcome!

Welcome!

$

6. clear 命令

该命令清除 Shell 窗口中的内容。命令格式：

clear

7. cd 命令

该命令改变工作目录。命令格式：

cd [pathname]

其中 pathname 为要进入的目录。

例 要进入目录/export/home/lihui 用命令。

$cd /export/home/lihui

8. pwd 命令

该命令显示当前的工作目录。命令格式：

pwd

例 $pwd。

/export/home/lihui

$

命令显示当前的工作目录为/home/lihui。

9. passwd 命令

该命令用于修改用户密码。命令格式：

passwd username

如果省略 username，表示修改自己的密码。只有超级用户才能修改其他用户的密码。

例 用户修改自己的密码。

$passwd

```
Enter new passed:xxxxxx
Re-enter new passed:xxxxxx
$
```

例 超级用户修改一般用户 linli 的密码。

```
#passwd linli
Enter new passed:xxxxxx
Re-enter new passed:xxxxxx
#
```

10. bc 与 dc 命令

这两条命令是桌上计算器命令。命令格式：

$bc [-l]或$dc

bc 和 dc 命令均用 Ctrl+D 或 quit 退出。

它们提供的基本运算有：

+（加）、–（减）、*（乘）、/（除）、%（余数）、^（指数）。

若加上-l 选项则还提供以下运算：

s(x)（正弦函数）、c(x)（余弦函数）、e(x)（指数函数）、l(x)（log 函数）、a(x)（反正切函数）。

dc 是一个后缀式计算器，所有运算符均写在参与运算的数字之前，现在 dc 较少使用。

例 $bc

```
112 + 34
146
(345-11)*2
668
Ctrl+D
$
```

例 $dc

```
112 34  +p
146
345 11 - 2 * p
668
Ctrl+D
$
```

11. more 命令

该命令以“ 页 ”为单位在屏幕上显示文件内容，也可从文件头、指定行号、特定字组开始显示。

命令格式：more [-crs] [filename]

其中选项：

c：在显示每页数据之前清除屏幕。

s：有连续空白行时只显示一行。

filename：为要显示的文件名。

在屏幕最下方的：

----------——more——----------(xx%)

信息标明目前已显示文件内容的字符占总字符的百分比。

按 Space 键显示下一页；按 D 键显示下半页；按 Enter 键显示下一行；按 P 键则回到文件的首页。

例 $more /etc/passwd

```
passwd (12%)
daemon:!:1:1::/etc:
bin:!:2:2::/bin:
sys:!:3:3::/usr/sys:
adm:!:4:4::/var/adm:
uucp:!:5:5::/usr/lib/uucp:
guest:!:100:100::/home/guest:
nobody:!:4294967294:4294967294::/:
lpd:!:9:4294967294::/:
liuxn:!:227:1::/home/liuxn:/usr/bin/ksh
lxh:!:241:0::/home/lxh:/bin/ksh
blademan:!:246:1::/home/blademan:/usr/bin/ksh
ftp:*:247:1::/home/ftp:/usr/bin/faulse
anonymou:*:248:1::/home/ftp:/usr/bin/faulse
lmy:!:249:1::/home/lmy:/usr/bin/ksh
zwen:!:251:1::/home/zwen:/usr/bin/ksh
imnadm:*:254:201::/home/imnadm:/usr/bin/ksh
ldap:*:255:1::/home/ldap:/usr/bin/ksh
zhenxm:!:412:1::/home/zhenxm:/usr/bin/ksh
```

```
zhoul:!:413:1::/home/zhoul:/usr/bin/ksh


passwd (27%)
duj:!:415:1::/home/duj:/usr/bin/ksh
wangdg:!:416:1::/home/wangdg:/usr/bin/ksh
wangsl:!:417:1::/home/wangsl:/usr/bin/ksh
wongqb:!:418:1::/home/wongqb:/usr/bin/ksh
yaor:!:419:1::/home/yaor:/usr/bin/ksh
……（省去）
passwd: END
$
```

12. cat 命令

该命令显示文件内容、创建新文件或合并文件内容成一个文件。命令格式：

```
cat [filename]
```

其中 filename 为要显示的文件名。

· 如果文件 filename 已经存在，cat filename 就一次显示存在文件 filename 的全部内容。与 more 命令的不同之处是全部显示而不是分屏显示。如果文件 filename 很长，使用 more 命令更好。

例 由于 UNIX 系统将设备当作文件来管理，下面命令：

```
#cat music.au>/dev/sudio
```

可用于在 Sun 的工作站上播放音频文件 music.au。

· 如果文件 filename 不存在，则 cat filename 将创建新文件 filename，命令格式为：

```
$cat>filename
```

“ > ”为重定向符号，用 Ctrl+D 结束文件输入并保存。

例

```
$cat>file1
1111
22222
333333
4444444
<Ctrl+D>
$
$cat file1
```

```
1111
22222
333333
4444444
$
```

· 合并多个已存在文件的内容到新文件，命令格式为：

```
$cat filename1 filename2>newfile
```

将 filename1 和 filename2 中的内容写入 newfile 中，filename1 在前，filename2 在后。

例

```
$cat filename1
aaaaaaaaaaaaaaa
sssssssssssssss
ddddddddddddddd
fffffffffffff
$cat filename2
ggggggggggggggg
hhhhhhhhhhhhhhh
jjjjjjjjjjjjjj
kkkkkkkkkkkkkkk
$cat filename1 filename2>file
$cat file
aaaaaaaaaaaaaaa
sssssssssssssss
ddddddddddddddd
fffffffffffff
ggggggggggggggg
hhhhhhhhhhhhhhh
jjjjjjjjjjjjjj
kkkkkkkkkkkkkkk
$
```

· 给已存在的文件增加内容，命令格式为：

```
$cat>>filename
```

用 Ctrl+D 结束数据输入并保存 filename。

例

```
$cat>>file1
55555555
```

```
666666666
<Ctrl+D>
$
$cat file1
1111
22222
333333
4444444
55555555
666666666
$
```

13. head 与 tail 命令

head 与 tail 命令分别用于显示文件头部信息和文件尾部信息。命令格式：

```
head [-n] filename
```

参数 n 为显示的行数，默认为 10，表示从头开始显示前 10 行。

例　显示文件头部信息。

```
$head /etc/passwd
root:!:0:0::/:/usr/bin/ksh
daemon:!:1:1::/etc:
bin:!:2:2::/bin:
sys:!:3:3::/usr/sys:
adm:!:4:4::/var/adm:
uucp:!:5:5::/usr/lib/uucp:
guest:!:100:100::/home/guest:
nobody:!:4294967294:4294967294::/:
lpd:!:9:4294967294::/:
jixi:!:227:1::/home/jixi:/usr/bin/ksh
$
```

tail 命令格式为：

```
tail [+/- number] [lbc] filename
```

number 为整数，默认为 10。

+表示从文件头部起 number 单位（行、块、字符）后开始显示；-表示从文件尾部起 number 单位（行、块、字符）后开始显示。

其中选项：

l：表示行（line）；

b：表示块（block）；

c：表示字符（character）。

例　看文件尾部倒数 10 行。

```
$tail /etc/passwd
wud:!:552:1::/home/wud:/usr/bin/ksh
liuk:!:553:1::/home/liuk:/usr/bin/ksh
hek:!:555:1::/home/hek:/usr/bin/ksh
tangyj:!:556:1::/home/tangyj:/usr/bin/ksh
starsoul:!:557:1::/home/starsoul:/usr/bin/ksh
youmer:!:558:1::/home/youmer:/usr/bin/ksh
limhai:!:8:0::/home/limhai:/usr/bin/ksh
liwt:!:228:1::/home/liwt:/usr/bin/ksh
hancl:!:229:1::/home/hancl:/usr/bin/ksh
vivian:*:230:1::/home/vivian:/usr/bin/ksh
$
```

也可以用：$tail –10 /etc/passwd

14. ls 命令

ls 命令列出文件和文件的属性，默认输出为标准输出（屏幕）。命令格式：

```
ls [-option] [name]
```

name 为所需列出的目录或文件名称。

其中选项：

a：即 all，列出所有文件和目录的属性，包括隐含的文件和目录；

d：仅仅显示指定目录属性；

i：显示文件的 inode；

l：打印文件属性，包括文件类型、文件拥有者、文件名称、存取权限、链接数目、最后修改日期；

t：按文件修改时间的先后顺序显示；

x：以一行多个文件方式显示；

F：显示的文件中可执行文件后加“ * ”标记，目录后加“ / ”标记。

例　显示当前目录下的文件。

```
$ls
```

```
db2             ifconfig.txt    mbox            smit.script
netgate         jobbegin        nfsmount        v7-dbacert.pdf
db2red.pdf      lliu            project         x
hlliu           logoffs         smit.log        y1
$
$ls -x
db2             netgate         db2red.pdf      hlliu           ifconfig.txt
jobbegin        lliu            logoffs         mbox            nfsmount
project         smit.log        smit.script     v7-dbacert.pdf  x
y1
$
$ls -l
total 42521
-rw-r-----      1 gixi      staff       8886060 Jan 06 12:04 db2
-rw-r--r--      1 gixi      staff       98 Jan 06 13:30 netgate
-rw-r-----      1 gixi      staff       8886060 May 02 17:02 db2red.pdf
-rw-r--r--      1 gixi      staff       4 Mar 21 21:24 hlliu
-rw-r--r--      1 gixi      staff       13957 Apr 18 08:09 ifconfig.txt
-rwxr--r--      1 gixi      staff       89 Mar 28 17:09 jobbegin
-rw-r--r--      1 gixi      staff       12 Mar 21 21:24 lliu
-rw-r--r--      1 gixi      staff       44 Apr 01 16:38 logoffs
-rw-------      1 gixi      staff       3424 Apr 12 13:03 mbox
drwxr-xr-x      2 gixi      staff       512 Apr 29 18:39 nfsmount
drwxr-xr-x      2 gixi      staff       512 Apr 01 22:41 project
-rw-r--r--      1 gixi      staff       29899 Apr 29 18:37 smit.log
-rw-r--r--      1 gixi      staff       756 Apr 19 11:52 smit.script
-rw-r-----      1 gixi      staff       3937716 Jan 03 21:43 v7-dbacert.pdf
-rw-r--r--      1 gixi      staff       10 Dec 24 14:36 x
-rw-r--r--      1 gixi      staff       10 Dec 24 14:51 y1
$
$ls -t
db2red.pdf      ifconfig.txt    jobbegin        db2
nfsmount        mbox            hlliu           v7-dbacert.pdf
smit.log        project         lliu            y1
```

```
smit.script     logoffs     netgate     x
$
$ls –dl
drwxr-xr-x   4 gixi   staff        512 Jul 03 16:10 .
$
$ls -a
.               netgate     jobbegin    nfsmount    v7-dbacert.pdf
..              db2red.pdf  lliu        project     x
.sh_history     hlliu       logoffs     smit.log    y1
db2             ifconfig.txt mbox       smit.script
$
$ls -i
4250 db2            4224 lliu       4230 smit.script
4251 netgate        4228 logoffs    4246 v7-dbacert.pdf
4248 db2red.pdf     4252 mbox       4272 x
4225 hlliu          2158 nfsmount   4273 y1
4231 ifconfig.txt   2085 project
4226 jobbegin       4227 smit.log
$
```

15. cp 命令

cp 命令用于拷贝文件或目录。命令格式：

cp [-fhip] [-r|-R] src1 src2

src1 为源文件或目录，src2 为目标文件或目录。

其中选项：

f：如果目标文件不能打开，则在拷贝之前删除目标文件；

h：强制复制符号链接；

i：交互式选项，提示将覆盖所有文件；

p：文件属性一同复制，即保持文件的修改时间和存取权限。如果没有该选项，则时间被设置成文件的复制时间，权限为默认权限；

r：递归复制源文件或者目录；

R：递归复制源文件或者目录，特殊文件重新创建而非复制。

如果没有指定目标路径，则为当前目录；如果源为多个文件或目录，则目标必须为一个目录；如果目标为文件，则可以用通配符。

例　跨目录拷贝：将文件/export/home/stu01/exer1 拷贝到文件/export/stu03/exer1。

```
#cp /export/home/stu01/exer1 /export/stu03/exer1
#
```

例　将文件 file1 拷贝到文件 file2，并要求 file2 具有与 file1 相同的权限。

```
$ls –l file*
-rwxrw-r--        1 gixi     staff          98 Jan 06 13:30 file1
-rw----r--        1 gixi     staff          98 Jan 06 12:50 file2
$cp –p    file1 file2
$ls –l file*
-rwxrw-r--        1 gixi     staff          98 Jan 06 13:30 file1
-rwxrw-r--        1 gixi     staff          98 Jan 06 13:30 file2
```

例　$cp /export/home/stu01/f* /export/home/stu02

```
$ls /export/home/stu02/f*
file1    file2
```

16. mv 命令

mv 命令用于重命名文件、目录和移动文件或目录。mv 之后，源文件名或源目录名不复存在，这是 mv 命令与 cp 命令不同的地方。命令格式：

mv [-if] src1 src2

src1 为源文件或目录，src2 为目标文件或目录。

其中选项：

f：覆盖现有文件不提示；

i：移动之前提示。

例　$ls –i f*

```
12540       file1        2267    file2
$mv file1 file3
$ls –i f*
2267        file2        12540   file3
$
```

17. rm 命令

rm 命令用于删除文件或目录。要删除文件或目录必须要求文件或目录具有写权限。当文件存在链接时不能删除。命令格式：

rm [-firRe] filename

filename 为要删除的文件或目录名称，可以用通配符。

其中选项：

e：删除后显示提示信息；

f：强制删除有写保护的文件；

i：删除每个文件前询问是否删除；

r/R：当 filename 是目录时，允许循环删除目录及其内容。

例 $rm -f /etc/host.bak

18. mkdir 命令

mkdir 命令用于创建目录。必须在所创建目录之上的父目录有“写”权限时才可创建子目录。命令格式：

mkdir [-mp] directoryname

directoryname 为要创建的目录名。

其中选项：

m：mode 选项可以在创建目录的同时设置存取权限，mode 是 8 进制表示的存取权限值，参见 chmod 命令；

p：所建目录为一个路径名时，若该路径中间的目录还没产生，则 UNIX 会自动建立中间的目录。

例 $mkdir –p /home/zhanglx/nettask 是在/home/zhanglx 下建立新目录。如果目录 zhanglx 不存在，该命令就自动创建 zhanglx 目录并在其下再创建 nettask 目录。

19. rmdir 命令

rmdir 命令用于删除空目录，如果目录不空则不能删除。命令格式：

rmdir [-p] directoryname

directoryname 为要删除的目录名。p 为删除指定目录的同时连同其空父目录一起删除。

例 命令$rmdir –p zhanglx/nettask 删除子目录 nettask，如果父目录 zhanglx 为空，则删除子目录 nettask 时将 zhanglx 一起删除。

20. man 命令

man 命令为用户提供使用命令、函数调用等的联机帮助。通常情况下 UNIX 把联机帮助信息放在目录/usr/share/man 或/usr/man 下，IBM 的 AIX 则在该 man 目录下又分为两个目录（分别是 cat1、info）和一个文件（whatis）。如果要提供给用户

默认使用路径，则应在系统配置文件中进行配置。命令格式：

man [command]

例　查看命令 who。

$man who

man19920 (8%)

Purpose

Identifies the users currently logged in.

Syntax

who [-a | -b -d -i -l -m -p -q -r -s -t -u -w -A -H -T] [File]

who am { i | I }

Description

The who command displays information about all users currently on the local system.
The following information is displayed: login name, tty, date and time of login.
Entering who am i or who am I displays your login name, tty, date and time you logged in.
If the user is logged in from a remote machine, then the host name of that machine
is displayed as well.

……（省去）

例　查看进程创建的系统调用。

$man fork

man19914 (3%)

Purpose

Creates a new process.

Libraries

fork: Standard C Library (libc.a)

vfork: Berkeley Compatibility Library (libbsd.a)SourceFile

TargetFile

Syntax

To Copy a File to a Directory

#include <unistd.h>

{ cp | copy } [-f] [-h] [-i] [-p] [-r | -R]

pid_t fork(void)e ... TargetDirectory

int vfork(void)tory to a Directory

Description } [-f] [-h] [-i] [-p] [--] { -r |

-R } SourceDirectory ... TargetDirectory

The fork subroutine creates a new process. The new process (child

……（省去）

除了上面介绍的基本系统命令外，其余命令会在后面的章节中逐渐介绍。

1.3 vi 编 辑 器

1.3.1 vi 简介

vi 是“ visual ”（可视）的意思，是 UNIX 提供的全屏幕编辑器。最初由加州大学 Berkeley 分校为 BSD UNIX 开发，后来作为标准包含在 UNIX 的所有版本中。

受到图形显示及键盘功能（如无功能键）的限制，当时的 vi 中没有提供任何图像及鼠标的使用，只使用键盘上的字母、数字、标点符号和 Esc 键完成对文件的编辑。在今天图形鼠标丰富的环境下，vi 看起来似乎有些过时，但它却仍是系统程序员及管理员最喜欢的编辑器之一，因为无论何时何地通过 Internet 登录到 UNIX 系统时只有用 vi 作编辑器最方便。

vi 窗口一屏只能显示 20 行内容，并可以上下移动窗口，可以从窗口上浏览文件全部内容。vi 编辑的文件大小有限制，最大行数为 25 000，每行最多 1 024 个字符。

1.3.2　模式

在 vi 中最常用的两个模式是命令模式和输入模式。

在命令模式中，每次键盘输入都是命令，如存盘、移动光标、页面滚动、删除文件、退出 vi 等。

输入模式是将输入内容保存到内存缓冲区，再对输入缓冲区中的数据进行编辑。编辑完成后，切换到命令模式，利用 w 命令（即 write）将已修改的文件写回存储设备（即磁盘）。所以，文件中的内容总是在输入模式下输入的。

进入输入模式的方式有以下几种。

· 在命令模式下键入 i（即 insert）。此时键盘上的任何字符均作为输入内容，且输入内容总是放在光标所在位置的左面（前面）。

· 在命令模式下键入 I（即 Insert）。此时键盘上的任何字符均作为输入内容，且输入内容总是放在光标所在行有效字符的最前面（行首）。

· 在命令模式下键入 a（即 append）。此时键盘上的任何字符均作为输入内容，且输入内容总是放在光标所在位置的右面（后面）。

· 在命令模式下键入 A（即 Append）。此时键盘上的任何字符均作为输入内容，且输入内容总是放在光标所在行有效内容的最后（行尾）。

· 在命令模式下键入 o（即 open）。此时键盘上的任何字符均作为输入内容，且输入内容总是放在光标所在行下面的一个新增加行。

· 在命令模式下键入 O（即 Open）。此时键盘上的任何字符均作为输入内容，且输入内容总是放在光标所在行上面的一个新增加行。

从输入模式转换到命令模式用 Esc 键。

1.3.3　vi 命令

1. 进入 vi

要进入 vi 编辑器编辑文件，执行命令

$vi filename↵（注意：新、老文件均可）

即进入命令模式。要输入文件内容则应该进入输入模式。

2. 退出 vi

退出 vi 有以下几种方式：

· 键入 Esc:q 则不存盘退出；

· 键入 Esc:wq 则存盘退出；

· 键入 Esc::q！则强制不存盘退出（放弃缓冲区内容）；
· 键入 Esc::wq！则强制存盘退出（放弃缓冲区内容）。

3. 在 vi 中定位光标

编辑文本时要把光标定位到插入文本、修改文本或删除文本的地方。现在有的终端可以使用方向键上、下、左、右，而有的终端却不能。

vi 为所有的终端定义了移动光标的键“ HJKL ”。在键盘上“ HJKL ”是依次排列的，使用非常方便，使用时移动光标的方向是“ 左、下、上、右 ”，如图 1.2 所示。

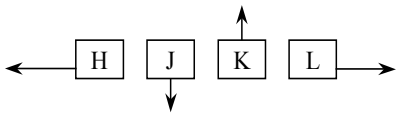

图 1.2 左、下、上、右移动光标

除了将光标上下左右移动之外，还有如下移动方式：
· 键入$，将光标移到所在行的最后一个字符；
· 键入^，将光标移到所在行的第一个非空格符；
· 键入 nG，将光标移到行号为 n 的行。用 Ctrl+G 显示光标所在行号。

4. 在 vi 中删除和改变文本

命令模式下通过以下几种方法删除和改变文本：
· 删除光标所在位置字符，键入 x；
· 删除光标所在行字符，键入 dd；
· 删除光标所在位置字符并进入输入模式，键入 s；
· 删除光标所在行字符并进入输入模式，键入 S；
· 修改光标所在位置字符，键入 R，然后进入替代状态，直到按 Esc 键为止；
· 修改光标所在位置字符，键入 r，然后再键入取代原字符的字符，即恢复到命令模式。

5. 恢复上一指令前的内容（撤销）

作了修改后又要撤销，恢复修改前的内容：
· 键入 u，恢复最后一个指令前的内容；
· 键入 U，恢复光标所在行的所有改变。

6. 在一行中寻找字符

- 键入 fx，将光标往右移到所指定字符 x 上；
- 键入 Fx，将光标往左移到所指定字符 x 上；
- 键入 tx，将光标往右移到所指定字符 x 的前面；
- 键入 Tx，将光标往左移到所指定字符 x 的前面；
- 键入；，以相同的方向重复上一找寻指令；
- 键入，，以相反的方向重复上一找寻指令。

7. 屏幕内容滚动

- 同时键入 Ctrl+D，向下滚动半页（12 行）；
- 同时键入 Ctrl+U，向回滚动半页（12 行）；
- 同时键入 Ctrl+F，向下滚动一页（24 行）；
- 同时键入 Ctrl+B，向回滚动一页（24 行）。

8. 编辑另一个文件

在 vi 命令方式下键入 e 则编辑另一个文件；键入 e!则放弃改动，重新加载文件内容到编辑缓冲区。

9. 系统异常关机时未存盘文件恢复

vi 会周期性地保存编辑缓冲区中的内容，如果系统异常关机（如异常掉电、死机等）时，用户未存盘退出，则用户可以再次登录到系统并用下列命令进入关机前的版本：

$vi –r filename↵

filenama 为文件名。

注意　有的系统可能不支持这项功能。

1.3.4　vi 内使用 Shell

在 vi 中可以运行 UNIX 的操作系统命令，语法是：

:!< unix_command ><Return>

在 unix_command 中可以使用%作为当前文件名的缩写；用#作为上次编辑文件名的缩写；用!作为前次命令的缩写。如：

:!cp #　myfile↵

把上次编辑过的文件复制到 myfile 文件。如果此时要重复这个命令，则只须用：

:!!

1.3.5 设置 vi 的工作环境

vi 是全屏幕编辑器，屏幕的特性会影响 vi。要想使 vi 工作正常，必须将正确的终端类型写入配置文件中。

文件/etc/terminfo 中放有终端的名称，B-Shell 下文件/etc/.profile（C-Shell 下是/etc/.login 或/etc/.rshrc 文件）中有终端信息。

如果终端类型与 vi 环境不吻合，则打开 vi 后会出现乱码，这时在 B-Shell 下需要将终端信息写入配置文件/etc/.profile（用户重新登录后仍然生效）：

```
TERM=vt100
export TERM
```

在 C-Shell 下则是在文件/etc/.login 或/etc/.rshrc 中写入（用户重新登录后仍然生效）：

```
setenv TERM vt100
```

如果以上命令不是写入文件中，而是在 Shell 下直接用命令写入，则用户重新登录后该设置不再生效。

1.4 本 章 小 结

本章在简单概述了 UNIX 操作系统的组成及特点的基础上，对 UNIX 操作系统中的一些基本使用和基本命令作了详细介绍，只有熟练掌握这些命令，才有助于对 UNIX 操作系统的使用，也才能完成一些简单的系统维护工作。

第 3 节介绍了 UNIX 下最常用到的全屏编辑器 vi。在一般的 UNIX 应用中，vi 扮演了一个非常重要的角色，几乎所有 UNIX 下的文本编辑都需要应用 vi，它是 UNIX 操作系统中需要熟练掌握的工具之一。

上 机 练 习

1. 从系统管理员处得到自己的用户名及口令，并用该用户名及口令登录系统，然后修改口令。
2. 利用 su 切换成为 root 用户，并使用 date 修改系统时间。
3. 利用 cd 改变工作目录，并使用 pwd 查看工作目录的变化。
4. 练习文件和目录的拷贝、移动、更名、删除等操作。

5. 使用 date 命令查看本地时间、格林威治时间。
6. 使用一个命令显示当前年份的完整日历。

7. 使用 who 命令查看登录到系统的用户信息。

8. 学习使用 man 查看各种命令的帮助。

9. 熟悉对于文本文件的各种操作，例如 cat、more、head、tail 等。

10. 熟练掌握 vi 的各种用法：

(1) 使用 vi 编辑文件 photo.txt，内容如下：

Start Photo Editor from another program

You can start Photo Editor from any program that supports OLE.

1　In the file you are working in, click where you want to insert an image.

2　On the Insert menu, click Object.

3　Click the Create New tab.

4　In the Object type box, click Microsoft Photo Editor.

If you want to scan a picture, click Microsoft Photo Editor Scan.

5　Click OK.

6　In the New dialog box, select the appropriate option.

For Help on an option, click the question mark and then click the option.

Note　You can also start Photo Editor as a separate program by using the same procedure that you use to start other programs.

(2) 删除下划线行。

(3) 将字“ Photo ”中的大写字母改成小写字母。

11. 下面左边的命令为 vi 编辑器模式，上机验证左边的命令是否有错，如果有错，怎样改正后可以和右边的解释配对？

(1) G	a、用字母 x 替换光标所在位置的字母
(2) /most	b、将光标移到文件中最后一行
(3) [Ctrl -g]	c、复制 4 行到缓冲区 x 中
(4) 2dw	d、将光标下移一行
(5) j	e、显示当前行的行号
(6) x4yy	f、将光标定位到第 66 行
(7) $	g、删除当前光标下的字符
(8) 0（零）	h、恢复缓冲区 1 的内容
(9) 66G	i、删除 2 个字符
(10) x	j、将光标定位在当前行的末尾
(11) rx	k、查找字 most
(12) lp	l、将光标定位在当前行的首行

习 题 一

1. UNIX 操作系统有哪些主要特性？
2. 为什么 UNIX 系统有较好的可移植性？
3. 请列出你所了解的 UNIX 系统？
4. 在 UNIX 系统的分层结构中包含哪些层次？
5. Shell 的目的何在？
6. 当不使用系统时，用户为什么要退出系统？
7. 你怎么理解 UNIX 系统是一个多用户系统，其应用如何？
8. 什么是 vi 编辑器，它有哪几种工作模式？
9. 把右面的解释和左面的命令连接起来。

(1) wc xxx yyy	a、复制 xxx 到 yyy
(2) cp xxx yyy	b、重命名 xxx 为 yyy
(3) ln xxx yyy	c、复制所有以 file 开始其后跟 2 个字符的文件名
(4) mv xxx yyy	d、删除当前目录的所有文件
(5) rm *	e、创建 xxx 的另外一个文件名为 yyy
(6) ls *[1-6]	f、显示 myfile 的内容
(7) cp file??source	g、复制 myfile 到 yyy
(8) pr –2 myfile	h、把所有文件名后面部分为 file 的文件内容加到文件 yy 中
(9) ls –i	i、以 2 列格式化 myfile
(10) pg myfile	j、列出所有以数字 1-6 结尾的文件名
(11) cat myfile	k、列出当前目录文件及其 I 节点
(12) cat myfile >yyy	l、创建名为 yyy 的文件包含 xxx 的字符数
(13) cat ?file >>yyy	m、一次一屏查看 myfile
(14) find .–name “file *” –print	n 、把 xyz 的第 2 列保存到 xxx 中
(15) find. –name xyz –size20 –print	o、一次一屏读 zzz
(16) cut –f2 xyz >xxx	p、寻找所有名为 xyz 大小为 20 个文件块的文件
(17) more zzz	q、显示所有文件名以 file 开头的文件

第 2 章　UNIX 文件系统

UNIX 系统成功的关键就在于其文件系统，最初的 UNIX 系统甚至连名字都没有，大家对它的称呼就是共享文件系统。所以文件系统在 UNIX 系统中起着非常重要的作用。

UNIX 文件系统除了具有一般操作系统文件系统的特点，如包含文件及属性说明、管理和操作文件、提供用户使用之外，还具有自己的特点：文件存放上的索引节点、文件内容访问上的保护、文件的动态增长、外围设备管理的文件化等。

本章主要内容

· 文件系统中的文件及目录：UNIX 系统中的文件及文件类型、目录及目录层次、文件及目录的所有者和权限。

· 文件系统结构：UNIX 文件系统的目录树、索引节点与目录、索引节点和磁盘块的分配。

· 文件系统类型：磁盘文件系统、虚拟文件系统、远程文件系统。

· 文件系统的管理命令：确定文件系统类型、维护文件系统、文件系统检测。

· 文件系统的启用：加载与卸载、加载本地文件系统、远程加载（共享网络文件系统）。

· 文件系统的备份和恢复：文件系统的备份、备份工具 dump、restore、cpio、tar、dd。

2.1　UNIX 文件及目录

从逻辑结构的观点看 UNIX 的文件系统，是由多级目录连接成一棵倒树型，文件可以放在该目录树的任何分枝上。

2.1.1　文件、目录及权限

文件及目录是文件系统中的基本元素，文件系统最基本的功能是提供给用户对文件及目录的访问及操作。

1. 文件类型

UNIX 文件系统将所有文件中的内容都看作是字符流，把所有文件都统称为流式文件。

在 UNIX 系统中，文件系统不决定文件类型，文件类型由文件本身的格式决定，文件类型与文件名没有必然联系，所以不能从文件名来判断文件类型，文件名的后缀也失去了特殊意义。

文件有以下几种类型。

（1）标准文件（用“ - ”表示）：用于存储任何类型的数据，是用户使用最普遍的文件类型。在这种类型的文件中，文件内容可以按文本、应用程序指定格式或系统可执行的二进制方式存储。

（2）目录文件（用“ d ”表示）：为了系统管理方便将多个相关的文件放入一个目录中成为目录文件。

（3）符号链接文件（用“ l ”表示）：在该类型的文件中包含一个链接指针，指针指向某个文件或目录，除指针外没有其他内容。实际上符号链接是多个文件名用一个物理存储来实现文件的共享。引入符号链接文件的优点是可以通过网络将相距较远的主机上的文件链接在一起共享；缺点是每次访问磁盘时要读盘多次，增加了访问磁盘的频率，开销大，另外索引节点还要占磁盘空间，对磁盘空间造成一定浪费。

（4）管道文件（用“ p ”表示）：向文件写入内容时是先进先出，不允许偏离次序。与其他文件不同的是用直接块而非间接块存储文件内容，文件内容是短暂的。

（5）特殊文件：是外围设备管理文件，包括字符设备文件（用“ c ”表示）和块设备文件（用“ b ”表示）。

（6）Socket 文件（用“ s ”表示）：是利用 TCP/IP 网络交互式进程通信的一种形式，Socket 文件就是操作系统中用网络的 TCP 接口通信的程序。

例 用命令 ls –l 查看文件类型。

```
$ls -l
total 1472
lrwxrwxrwx   1 bin      bin        8 Jun 16 2001       bin -> /usr/bin
drwxr-xr-x   2 root     system   512 Sep 11 2000       cdrom
-rw-r-----   1 root     system   463 Jun 15 2001       configassist.log
lrwxrwxrwx   1 bin      bin        8 Jun 16 2001       lib -> /usr/lib
-rw-r--r--   1 root     system   105429 Mar 06 20:52   smit.script
-rw-r--r--   1 root     system     0 Sep 17 2002       websm.log
```

```
$
```

命令输出的第一列中的第一个字符表示的是文件类型。第 1 行是一个符号链接文件（l），文件 bin 的符号链指向/usr/bin；第 2 行是一个目录 cdrom；第 3 行是一个标准文件 configassist.log。

例　用 file filename 命令可以查看标准文件的具体类型，如：

```
$file websm.log
websm.log:        empty
$file smit.script
smit.script:      C program text
$
```

标准文件 websm.log 是一个空文件；标准文件 smit.script 是一个 C 程序文本文件。

例　下面是用管道文件实现 who 命令的输出通过管道输入到 grep 命令。

寻找用户名为 gili 的用户现在正在使用系统的信息：

```
$who |grep gili
gili        pts/2        Mar 21 21:11      (211.162.148.21)
$
```

例　用 ln 命令可以创建一个符号链接文件。

命令格式：ln 旧文件名 新文件名

实际上就是在旧文件与新文件之间建立一个链接。

```
$ln /home/lybing/friends /home/zhoux/infor
```

表示在用户 zhoux 的私有目录下创建一个链接指向用户 lybing 的私有目录下的 friends 文件。

下面用命令 ls 查看该符号链接文件。

```
$ls –l
lrwxrwxrwx 1 zhoux other 22 Apr 2003 infor_html->../home.lybing/firends
```

2. 目录及目录层次

目录包含相关文件，在 UNIX 系统中文件及目录的组织层次是一棵倒树型，最顶层的节点是根节点（root，又称为根目录），是整个文件结构的根；树叶节点是文件（file），中间节点是目录（directory）。除根目录之外，任何目录都有其父目录。在同一目录下不允许有相同文件名的文件存在，不同目录下允许文件具有相同的名称。

图 2.1 是 AIX 的文件系统目录结构，该结构分为两部分：目录与文件系统。目

录包含系统文件；文件系统部分是逻辑卷，对它的访问通过 mount 到文件系统实现，系统引导时已经 mount。（详细内容请参见 AIX 系统管理）。

其中：

/bin 用于存放大部分可执行的命令和公用程序，少数会放在/usr/bin 下；

/dev 中包含设备管理文件，不可删除；

/etc 中存放系统管理数据文件和程序；

/lib 中存放子程序库；

/usr 中存放某些要成为系统的一部分以供别人使用的命令或文件；

/tmp 中存放临时工作文件；

/var 中存放经常变化的系统文件，如系统日志；

/home 中存放用户的根目录，用户第一次登录时进入默认目录。

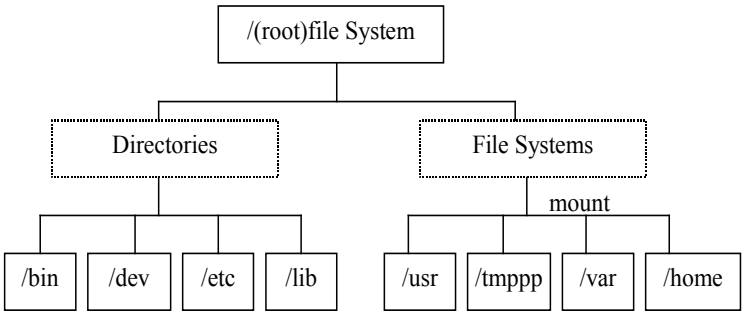

图 2.1 AIX 文件系统

3. 文件、目录的所有者和权限

UNIX 之所以被称为安全的操作系统，主要原因是除用户登录系统需要用户名和口令安全验证之外，还在于文件、目录有所有者和访问权限。

创建文件、目录的用户是文件、目录的所有者，文件、目录所有者所在的用户组为文件、目录的所有者组。UNIX 允许文件、目录的所有者用 chown 命令改变文件、目录的所有者，超级用户可以改变系统中所有文件、目录的所有者。

文件、目录的权限分为文件所有者（u）权限、所有者同组的用户组（g）权限、其他用户（o）权限。

文件、目录权限是可以对文件实施何种操作的权限，即文件、目录的访问权限。文件的访问权限包括：

- 读（r）：可以查看文件内容；
- 写（w）：可以修改文件内容；
- 执行（x）：可以像程序一样运行。

目录的访问权限包括：

· 读（r）：可以用 ls 命令显示该目录下文件的名称；

· 写（w）：可以在该目录下创建、修改、删除文件；

· 执行（x）：可以使用该目录下的文件，可以使用 cd 命令进入该目录，即用户必须对目录有执行权才能访问该目录。

除超级用户之外，只有所有者才能修改文件、目录的访问权限。

用命令 ls-l 可以查看文件、目录的所有者和权限。

例　用命令 ls –l 查看文件或目录的权限。

```
$ls -l
total 1472
lrwxrwxrwx    1 bin      bin          8 Jun 16 2001       bin -> /usr/bin
drwxr-xr-x    2 root     system     512 Sep 11 2000       cdrom
-rw-r-----    1 root     system     463 Jun 15 2001       configassist.log
lrwxrwxrwx    1 bin      bin          8 Jun 16 2001       lib -> /usr/lib
-rw-r--r--    1 root     system  105429 Mar 06 20:52      smit.script
-rw-r--r--    1 root     system       0 Sep 17 2002       websm.log
……
$
```

命令输出的第一列中的第 1 个字符为文件或目录的类型：d 表示目录；-表示普通文件；l 表示符号链接文件；第 2~4 个字符表示文件、目录所有者用户权限，第 5~7 个字符是文件、目录组用户权限，第 8~10 个字符是文件、目录其他用户权限。从第一个文件可见，符号链接文件 bin 的所有者有读、写、执行权，与所有者同组的用户有读、写、执行权，其他用户有读、写、执行权。

下面用命令修改文件、目录的所有权和访问权限。

(1) 用 chown 命令修改文件或目录的所有者

命令格式：chown new-owner-user name

其中 new-owner-user 为新所有者的用户名；name 为要修改的文件或目录名。

只有 root 用户才可以更改文件的所有者。

例　下面将文件 abctask 的所有者从 lilixx 变到 gigixx。

```
#ls –l
-rw-rw-r--    1 lilixx   group1     18   23 Feb 2002   abctask
#chown gigixx abctask
$ls –l
-rw-rw-r--    1 gigixx   group1     18   23 Feb 2002   abctask
```

也可以同时改变多个文件的所有者。

```
$ls –l
-rw-rw----      1 zhangx   group1      18   23 Feb 2002   matask1
-rw-rw----      1 zhangx   group1      18   23 Feb 2002   matask2
#chown wangyi mytask1 mytask2
$ls –l
-rw-rw----      1 wangyi   group1      18   23 Feb 2002   matask1
-rw-rw----      1 wangyi   group1      18   23 Feb 2002   matask2
```

(2) 用 chgrp 命令修改组

命令格式：chgrp new-groupid name

其中 new-groupid 为新组的组标识符；name 为要修改的文件名。

只有 root 用户和拥有该文件的用户才可以更改文件的组。如果你不是 root 用户，就只能将组更改为用户所属的其他组。

例　下面将文件 abctask 从原组 group1 变到新组 group2。

```
$ls –l
-rw-rw-r--      1 lilixx   group1      18   23 Feb 2002   abctask
#chgrp group2 abctask
$ls –l
-rw-rw-r--      1 gigixx   group2      18   23 Feb 2002   abctask
$
```

如果要修改目录中每个文件的所有者或组，在上面两个命令中加上选项-R 即可。

(3) 用 chmod 命令修改访问权

只有 root 用户或者文件的所有者才能更改文件的访问权。

要修改文件、目录的访问权，可以使用数字和字符两种命令格式。

· 数字命令格式：chmod number-mode name

其中 number-mode 为用数字模式表示的访问权：

no：读、写、执行三种权都没有；

x：有执行权；

w：有写权；

w、x：有写和执行权；

r：有读权；

r、x：有读和执行权；

r、w：有读和写权；

r、w、x：有读、写、执行权。

每次使用时所有者、组、其他用户的访问权分别用相应的数字带入。要记住这些数字有个规律：1（x）、2（w）、4（r）是基本数字；3 由 1+2 构成（x+w），5 由 1+4 构成（x+r），6 由 2+4 构成（w+r），7 由 1+2+4 构成（x+w+r）。只须记住基本数字即可。

name 为要修改的文件或目录名。

例　用数字方式修改访问权。

```
$ls –l
-rw-rw-r--        1 lilixx    group1       18    23 Feb 2002    abctask
```

要将访问权改为所有者有 rwx 权，对应数字为 7；组有 r 权，对应数字为 4；其他用户有 r 权，对应数字为 4。命令为：

```
$chmod 744 abctask
$ls –l
-rwxr--r--        1 lilixx    group1       18    23 Feb 2002    abctask
$
```

· 字符命令格式：chmod char-mode name

其中 char-mode 为用字符模式表示的访问权，由三部分构成：访问者、操作、访问权。

访问者表示为：

u：所有者；

g：所有者同组用户；

o：其他用户；

a：系统所有用户。

操作表示为：

+：加上所指定的权限；

-：取消所指定的权限；

=：赋予所指定的权限；

访问权表示为：

r：读；

w：写；

x：执行。

例　用字符方式修改访问权。


```
$ls –l
-r--r-----        1 lilixx    group1       18    23 Feb 2002    inputdata
```

```
$
```

要将访问权改为所有者有 rwx 权，命令为：

```
$chmod u+wx inputdata
$ls –l
-rwxr----        1 lilixx    group1       18    23 Feb 2002    inputdata
$
```

要取消组用户的读访问权，命令为：

```
$chmod g-r inputdata
$ls –l
-rwx------        1 lilixx    group1       18    23 Feb 2002    inputdata
$
```

要使组用户有读、写、执行权，命令为：

```
$chmod g=rwx inputdata
$ls –l
-rwxrwx---        1 lilixx    group1       18    23 Feb 2002    inputdata
$
```

2.1.2 文件和目录的上锁

为了在 UNIX 系统中配合数据库系统的使用，UNIX 系统 V 增加了对文件和记录的加锁机制。文件加锁是为了协调进程并发访问文件或记录时的同步机制。系统核心提供系统调用 fcntl 实现加锁。

fcntl 的使用在第 9 章 UNIX 系统调用中的文件系统调用部分介绍。

2.2 UNIX 文件系统结构

在 UNIX 文件系统中有大量的文件和目录。用户观察到的文件组织形式、使用到的文件内容实际上独立于文件在磁盘上的物理存储，是文件的逻辑结构。目录将文件的逻辑结构转换为文件在外存上的物理结构。文件系统结构主要介绍文件系统在磁盘上的实现。

2.2.1 UNIX 文件系统

在 UNIX 系统中文件系统在磁盘上包含 4 种块结构区：启动块、超级块、索引节点块和数据块。

（1）启动块：当启动操作系统时，启动程序将操作系统的内核从磁盘加载到

内存。包含启动程序的磁盘块被称为启动块，通常位于磁盘开始处。

（2）超级块：磁盘中的第 2 个块是超级块，用于存储文件系统信息，如磁盘的总容量、空闲块的数目等。2.2.3 节将对超级块作详细介绍。

（3）索引节点块：用于存储数据块中的各文件信息，这些文件信息是除文件名之外的文件的属性。2.2.2 节将对索引节点块作详细介绍。

（4）数据块：数据块主要包含所有用户文件（即用户数据）和与用户数据相关的一些特殊文件。

2.2.2　索引节点与目录

目录本身是一种文件，也称为文件目录。

通常操作系统的文件目录中有文件名和文件的其他属性。文件目录检索只依赖于文件名，为了加快操作系统检索文件目录的速度，UNIX 将文件名和文件的其他属性分开。在文件目录中放文件名，而将文件的其他属性单独形成数据结构放在一个索引节点中，这个索引节点就称为 inode（Index Node）。要找到文件的索引节点必须在文件目录中存放索引节点的指针，目录中除文件名之外还有索引节点的指针，图 2.2 是一个有 4 个文件的文件目录。

file0	0084
file1	1467
file2	4280
file3	1983

图 2.2　有 4 个文件的文件目录

文件名和索引节点之间的关系是一种链接关系。链接关系是一种逻辑关系，在 UNIX 系统中存在硬链接和符号链接（软链接）两种结构。

UNIX 操作系统采用索引节点的优点除了上面介绍的可以加快文件查找之外，另一个优点是可以将两个或多个文件名链接到一个物理文件，在 UNIX 系统中，一个物理文件只有一个索引节点但可以有多个文件名。这些文件名可以在相同的目录下，也可以在不同的目录下，这样也就实现了一种既方便又有效的文件共享方法。

操作系统中文件及目录的个数受到文件系统 inode 总数的制约。

当文件在磁盘上建立后，磁盘上就有一个惟一的磁盘索引节点 dinode。

磁盘索引节点与文件数据及磁盘块的关系如图 2.3 所示。

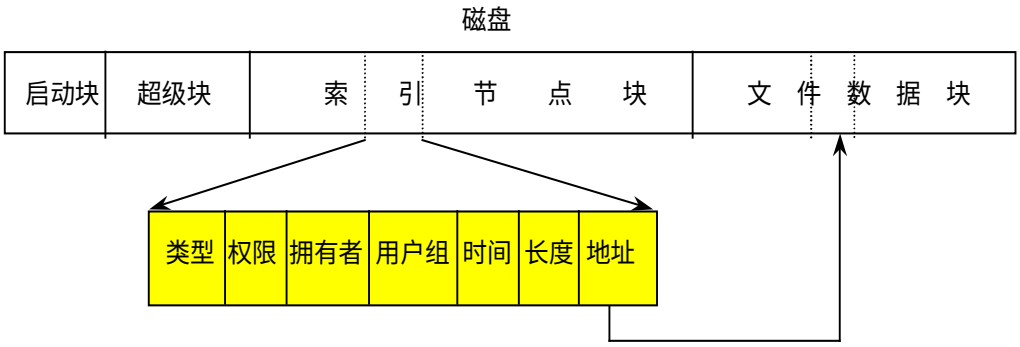

图 2.3 磁盘块、索引节点

磁盘索引节点包含如下信息：

· 文件类型；
· 文件访问权；
· 文件链接数目；
· 文件所有者标识符；
· 文件所有者组标识符；
· 文件长度；
· 文件创建时间；
· 文件存取时间（包括文件最后修改时间、最后访问时间）；
· 文件数据的磁盘地址指针表（文件的物理地址）。

同样，当文件在内存中打开时，内核在内存中也要建立一个相应的惟一的内存索引节点 iinode。内存中的索引节点内容绝大部分直接来自磁盘索引节点。除磁盘索引节点内容的字段之外，还补充了如下字段：

· 索引节点号：磁盘索引节点不需要该字段，因为磁盘索引节点以线性数组方式存储在磁盘上，内核只须用数组中的位置就可以标识磁盘索引节点号。而内存索引节点必须有这一个字段。
· 文件所属的文件系统的逻辑设备号。
· 内存索引节点的状态：是否上锁、是否有进程等待开锁、内存索引节点与磁盘索引节点是否相同、文件是否被修改过。

由于内存索引节点是互斥访问的资源，当有进程访问时索引节点处于上锁状态；当进程访问完后必须开锁释放；当进程访问时如果索引节点处于上锁状态就必须等待其他进程释放锁并唤醒等待。

· 文件的引用数：记录节点当前被多少个进程访问；内存索引节点的引用数必须为 0 时才能收回到空闲表上。

· 本节点在内存索引节点队列上的前向指针、后向指针。

内核依靠索引节点工作时也需要目录。进程和当前目录联系，将文件路径名转换为索引节点地址，从而得到文件的物理地址，对文件进行访问。系统中 0 进程的当前目录是根目录；其他进程的当前目录起源于其父进程的当前目录；要改变当前目录，要么通过命令 cd，要么通过系统调用 chdir。当前目录存储在进程的 u 区，系统根索引节点存储在操作系统的全局变量中。无论在什么情况下，内核都能找到索引节点并开始搜索文件的物理地址。

总之，有了索引节点，通过文件名得到文件的过程是：首先用文件名在目录中找到文件索引节点指针，再通过索引节点指针找到索引节点，然后在索引节点中得到文件的物理地址，从而得到文件。

2.2.3　索引节点和磁盘块的分配

在磁盘块中包含有引导块、超级块、索引节点块、文件数据块。

超级块不但在磁盘块中与索引节点块相邻，同时超级块中还有有关索引节点的关键信息。下面首先介绍超级块中包含的信息。

1. 超级块

超级块的组成：

- 文件系统规模；
- 文件系统中空闲块数目；
- 文件系统上可用空闲块表；
- 空闲块表中下一个空闲块的下标；
- 索引节点表的大小；
- 文件系统中空闲索引节点数目；
- 文件系统中空闲索引节点表；
- 空闲索引节点表中下一个空闲索引节点的下标；
- 空闲块表的锁字段和空闲索引节点表的锁字段；
- 用来指示超级块已被修改的标志。

2. 索引节点分配

在文件系统中存在一个索引节点线性表，该表的类型字段为 0 则表示该索引节点空闲，可以分配出去。当一个进程需要一个新的索引节点时，如果内核从空闲索引节点表中查找空闲索引节点则速度太慢、代价太大，为了加快查找速度，内核从超级块中得到空闲索引节点表，如果该表中有空闲索引节点，则可以分配索引节点，如果该表中没有空闲索引节点，则等待其他进程释放。对超级块中空闲索引节点表

的访问用锁机制实现进程互斥。即如果有其他进程在访问该空闲索引节点表，则本进程须等待其他进程访问完后解锁才能进去，进去的同时锁住该空闲索引节点表，保证本进程成功访问。

3. 磁盘块分配

在为进程分配索引节点之后，下一步进程就要将文件内容写到文件上，内核从文件系统中分配磁盘块。在超级块中有文件系统可用空闲（磁盘）块表。如果该空闲（磁盘）块表中有空闲（磁盘）块，则分配出去；如果文件系统没有空闲（磁盘）块，则显示出错信号。这里需要强调的是，磁盘块的分配过程与索引节点分配过程一样要访问超级块中的表，同样也用锁机制实现进程之间的互斥访问。

删除文件时，分配的索引节点和磁盘节点需要回收。回收是分配的相反过程，在回收时注意要写超级块中的信息字段。

2.3 UNIX 文件系统类型

文件系统是为了快速、方便、安全检索文件而采用的一种组织方法，用来定位和存储文件的目录结构。

在 UNIX 中用户进程对文件系统的访问通过系统调用实现。内核能将这种系统调用转换成对磁盘索引节点、内存索引节点及远程索引节点的操作，如图 2.4 所示，由于索引节点的不同，对文件的访问方式也不同，因而文件系统也不同。目前初步将文件系统分为三类：支持磁盘文件系统、虚拟文件系统和远程文件系统。

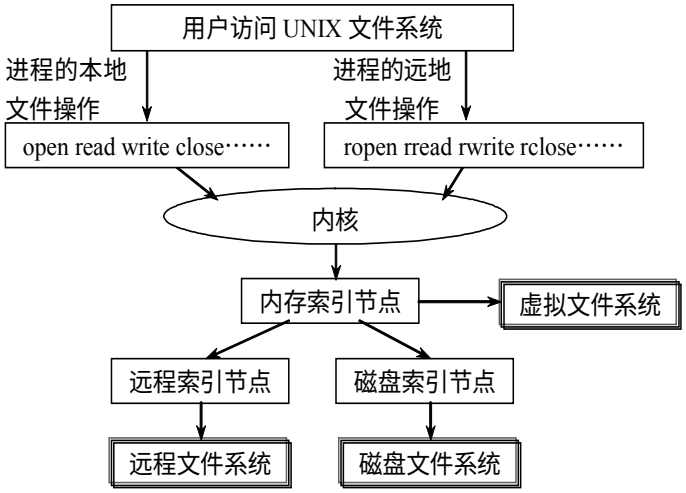

图 2.4 UNIX 文件系统

2.3.1　磁盘文件系统

磁盘文件系统是基于物理存储介质的文件系统，如硬盘、软盘和光盘等。磁盘文件系统分为 3 种：

· UFS：针对磁盘的文件系统。依据 UNIX 版本不同称为 Berkeley 快速文件系统、UNIX 文件系统；

· ISO9000：针对光盘 CD-ROM 的标准文件系统；

· MSDOS：运行 Windows 的文件系统，在 UNIX 系统中允许以软盘方式使用。

2.3.2　虚拟文件系统

虚拟文件系统是基于内存的文件系统，分为进程文件系统，临时文件系统和对换文件系统 3 种。

（1）proc（进程）文件系统：不是普通的文件系统，在物理盘上不存在，而是安装在路径/proc 下，在系统内存中为每个运行着的进程存取文件和目录。所以不是由系统管理员创建，而是由内核创建的。当进程出错或异常时作为运行跟踪使用。

需要注意的是进程文件系统/proc 不是目录，它不占用磁盘空间，也不能删除，更不能用删除/proc 目录中的进程命令来杀死进程。用命令 ls–l/proc 可以列出系统当前活跃的进程。

例　$ls –l /proc

```
total 2890
-rw-------    1 root     root            0 Aug 14 12:35 00000
-rw-------    1 root     root        34562 Aug 14 12:35 00001
-rw-------    1 root     root            0 Aug 14 12:35 00002
-rw-------    1 root     root       124665 Aug 14 12:35 00014
-rw-------    1 root     root        34554 Aug 14 12:35 000027
……
$
```

（2）tmpfs（临时）文件系统：使用本地内存进行磁盘读和写。访问 tmpfs 文件系统中的文件比访问 UFS 文件系统中的文件快。tmpfs 文件系统不是永久的，当文件系统卸载、系统关闭、重新启动时自行消除。系统路径/tmp 目录中默认的文件系统就是 tmpfs 文件系统类型。可以复制或移动文件到/tmp 目录，也可以从/tmp 目录复制或移动文件。

实际上 tmpfs 使用交换空间作为临时存储区，在下面的 swap 文件系统中会介绍怎样增加这样的对换区。

（3）swap（对换）文件系统：磁盘上用于系统虚拟存储的存储区，是使用 mkfile 和 swap 命令创建、供内核使用的文件系统。UNIX 版本不允许 swap 文件系统作为系统目录树的一部分安装，用户不会面对一个 swap 分区。

添加交换空间的方法有两种：一是在系统配置文件中配置，长期生效；一是通过命令 mkfile、swap，系统重新启动后不再生效。

2.3.3 文件系统管理文件

对文件系统的管理在 UNIX 系统中通过管理文件实现。不同的 UNIX 系统，管理文件的名称也各不相同。比如，在 AIX 系统中是/etc/filesystems，在 Solaris 系统中是/etc/vfstab，在 SunOS、IRIX、Linux、Tru64 UNIX 中是/etc/fstab，在 SCO UNIX 中是/etc/default/filesys，在 HP-UX 中是/etc/checklist。对该文件采用不同的配置方式可以实现不同类型文件系统的管理。下面以 Solaris 系统为例加以说明。

系统管理文件在 Solaris 系统中为/etc/vfstab（在其他系统中文件名称不同，详见表 2.2），也称为虚拟文件系统表，在文件系统管理中起重要作用，在该文件中一共有 7 个字段，分别表示的意义如下。

字段 1：device to mount（要加载的设备）。UNIX 中可以加载的设备有：

- 本地 UFS 文件系统专用的块设备（如硬盘片/dev/dsk/c0t0d0s1）；
- 远程文件系统资源名（例如 NFS 文件系统的 server1:/export/home/infor-a）；
- 用于交换的盘片名（例如/dev/dsk/c0t3d0s1）；
- /proc 目录和 proc 文件系统类型；
- CD-ROM 作为 HSFS 文件系统类型；
- /dev/diskette 作为 PCFS 或 UFS 文件系统类型。该字段也用于指定交换文件系统。

字段 2：Device to fsck（fsck 的设备）。原（字符）专用设备和 special 字段（例如/dev/rdsk/c0t0d0s0）标识的文件系统类型相符。它决定了由 fsck 使用的原接口。如果没有可用设备时使用短线（-），如只读文件系统、基于网络的文件系统。

字段 3：mount point（加载点）。系统默认的加载点目录（例如/dev/dsk/c0t0d0s6 的/usr）。

字段 4：FS type（文件系统类型）。由 special 字段标识的文件系统类型。

字段 5：fsck pass。由 fsck 使用的 pass 数决定是否检测文件系统。当字段包含短线（-）时不检测文件系统；当字段包含的数为 1 或大于 1 时检测文件系统。fsck pass 为 0 的非 fsck 文件系统需要检测。对于 UFS 文件系统，仅当该字段包含 0 时文

件系统不检测。当 fcsk 运行在多个 UFS 文件系统上，fsck pass 值为 1 或大于 1 并且使用了-o p 选项时，fsck 自动在不同的硬盘上以最大性能并行检测文件系统。当字段包含为 1 的值时顺序检测文件系统，否则 pass 值无效。

字段 6：mount at boot(引导时加载)。指定当系统引导时文件系统是否由 mountall 自动加载。注意该字段不能影响自动加载软件。

字段 7：mount option（加载选项）。在加载文件系统中使用逗点选项（不使用空格）分隔，使用短线（-）表示无选项，可用选项有 rw（读写）、ro（只读）或默认（输入短线“ - ”）。

在/etc/vfstab 文件中每个字段必须有一个入口，该字段如果没有值，则输入短线（-）。

例　通过系统配置文件/etc/vfstab 添加交换空间。

首先用 mkfile 命令创建一个交换文件系统：

```
#mkfile 2m /fs/swapfile
```

然后在/etc/vfstab 文件中加入行：

```
path-name - - swap – no –
```

加入后文件/etc/vfstab 中的内容为：

```
#cat /etc/vfstab
……
/fs/swapfile   -   -   sawp   -   no -
……
#
```

如果不用加载到 swap 中，删除相应的行即可。

在 SunOS 中使用 mkfile 命令可以创建一个交换文件系统，交换文件系统可以在本地磁盘或通过远程文件系统远程加载，再用 swap 命令添加交换空间。

例　用命令 swap、mkfile 添加、删除交换空间。

列出可用的交换文件系统：

```
$swap –l
swapfile              dev swaplo blocks     free
swapfs                 -          0 83525     72342
/dev/dsk/c0t3d0s1   32, 25     8 65512     35483
$
```

在本地创建交换文件系统（超级用户身份创建的交换文件用户进程不能访问），单位分别为 M（兆）、K（千）、b（字节）：


```
#mkfile 2m /fs/swapfile
```

```
#
```

将交换文件系统添加到交换空间：

```
#swap –a /fs/swapfile
```

是否添加成功可以再用命令 swap 验证：

```
#swap –l
swapfile              dev swaplo blocks     free
swapfs                 -        0 83525      72342
/dev/dsk/c0t3d0s1     32, 25    8 65512      35483
/fs/sawpfile           -        8   4080       4080
#
```

同样也可以将添加的交换空间删除，只须在 swap 命令中将-a 选项改为-d 选项。虽然没有了交换空间，但创建的交换文件系统仍然存在，却不能使用。

2.3.4 远程文件系统

远程文件系统是通过网络访问的文件系统。当一个系统上有远程文件系统时，其他系统可以通过网络访问它。

在 UNIX 中可用的网络文件系统有：

- NFS：网络或分布式文件系统，是系统默认的网络文件系统；
- RFS：远程文件系统。

通常情况下通过网络共享和加载到单个系统进行管理。

2.4 文件系统的管理命令

UNIX 提供了对文件系统的管理命令，格式如下：

```
command [-F type] [-v] [generic-options] [-o specific-options]
```

2.4.1 确定文件系统类型

用如下方式可以确定文件系统类型：

- 虚拟文件系统表/etc/vfstab 中的 FS type 字段；
- 本地文件系统用系统文件/etc/default.fs；
- 远程文件系统用/etc/dfs/fstypes 文件。

例 用命令 grep mount-point /etc/vfstab 可以找到文件类型描述行。


```
$grep /tmp /etc/vfstab
sawp                  /tmp    tmpfs    -       yes     -
```

```
$
```

例　在/etc/mnttab 中查找文件系统类型。

```
$grep /home/lxl /etc/mnttab
(pid119) /home/lxl nfs ro,ignore,map=/etc/home/auto_lxl, indirect, dev=18c0004 693004299
bestfriend: /home/bestfriend /tmp-mnt /home/bestfriend nfs rw, dev=18c0004 693234299
$
```

例　用命令 mount 确定文件系统加载点。

```
$mount
/on    /dev/dsk/c0t0d0s0    read/write    on Mon Apr 1    13:00:00 2003
/usr on /dev/dsk/c0t1d0s1 read/write    on Mon Apr 1    13:00:00 2003
/proc on /proc    read/write    on Mon Apr 1    13:00:00 2003
/usr/man on swsvr4 – 50: /export/svr4/man read/write/remote on Mon Apr1 13:00:00 2003
```

2.4.2　维护文件系统

1. df 命令

df 命令用于查看文件系统总空间和可用空间的信息，以及当前文件系统中已有的文件系统。命令格式：

```
df [ [ -P ] | [ -I | -M | -i | -t | -v ] ] [ -k ] [ -s] [ FileSystem…  | File  …]
```

在输出信息中：

Filesystem：是文件系统设备所在的位置；

x-blocks：是总块数；

used：是已使用的块数；

available：是剩余块数；

capacity：是文件系统空闲空间的百分比；

mounted on：是文件系统加载目录或加载点。

其中选项：

P：以 POSIX 格式显示文件系统信息；

I：显示块信息，已用空间信息，自由空间信息，已用空间百分比，文件系统的加载点；

M：在输出的第二列显示文件系统加载点信息；

i：表示用 inode 代替块显示；

t[fstype]：指定只有安装类型为 fstype 的文件系统被显示，Solaris 系统中该选项为 F；

v：显示所指定的文件系统的所有信息；

k：表示输出值以 1 K 为块单位；

s：从虚拟文件系统得到文件系统信息；

directory：要查看的目录名称。如果不用目录，则查看系统所有已经安装的文件系统。

例 在 AIX 系统中使用 df 命令。

```
$df
Filesystem    512-blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4          917504    270888   71%     2008     1% /
/dev/hd2         1400832    118368   92%    17567    10% /usr
/dev/hd9var        32768     23928   27%     1286    32% /var
/dev/hd3           32768     22864   31%       87     3% /tmp
/dev/hd1          466944    405080   14%      715     2% /home
$
```

从 df 命令可见，当前系统中有文件系统:/、/usr、/var、/tmp、/home。硬盘之所以分为多个文件系统，目的是为了保护数据，使维护简单。

```
$df -P
Filesystem    512-blocks      Used Available Capacity Mounted on
/dev/hd4          917504    646616    270888      71%    /
/dev/hd2         1400832   1282464    118368      92%    /usr
/dev/hd9var        32768      8848     23920      28%    /var
/dev/hd3           32768      9904     22864      31%    /tmp
/dev/hd1          466944     61864    405080      14%    /home
$
$df -I
Filesystem    512-blocks      Used      Free   %Used  Mounted on
/dev/hd4          917504    646616    270888     71%   /
/dev/hd2         1400832   1282464    118368     92%   /usr
/dev/hd9var        32768      8848     23920     28%   /var
/dev/hd3           32768      9904     22864     31%   /tmp
/dev/hd1          466944     61864    405080     14%   /home
$
```



```
$df -M
Filesystem    Mounted on    512-blocks    Free      %Used    Iused %    Iused
```

```
/dev/hd4        /          917504     270888    71%      2008       1%
/dev/hd2        /usr       1400832    118368    92%      17567      10%
/dev/hd9var     /var       32768      23920     28%      1286       32%
/dev/hd3        /tmp       32768      22864     31%      87         3%
/dev/hd1        /home      466944     405080    14%      715        2%
$
$df -i
Filesystem    512-blocks      Free    %Used     Iused     %Iused    Mounted on
/dev/hd4        917504      270888    71%       2008      1%        /
/dev/hd2        1400832     118368    92%       17567     10%       /usr
/dev/hd9var     32768       23920     28%       1286      32%       /var
/dev/hd3        32768       22864     31%       87        3%        /tmp
/dev/hd1        466944      405080    14%       715       2%        /home
$
$df -t
Filesystem    512-blocks        Used     Free     %Used     Mounted on
/dev/hd4        917504        646616   270888   71%         /
/dev/hd2        1400832       1282464  118368   92%         /usr
/dev/hd9var     32768         8848     23920    28%         /var
/dev/hd3        32768         9904     22864    31%         /tmp
/dev/hd1        466944        61864    405080   14%         /home
$
$df -v
Filesystem 512-blocks    Used     Free    %Used    Iused Ifree    %Iused    Mounted on
/dev/hd4     917504      646616   270888 71%      2008 227368    1%        /
/dev/hd2     1400832     1282464  118368 92%      17567 15856    10%       /usr
/dev/hd9var 32768          8848     23920 28%      1286      2810    32%       /var
/dev/hd3     32768          9904     22864 31%       87       4009      3%      /tmp
/dev/hd1     466944         61864    405080 14%      715      57653     2%      /home
$
$df -k
Filesystem    1024-blocks      Free      %Used     Iused     %Iused    Mounted on
/dev/hd4        458752         135444    71%       2008      1%        /
/dev/hd2        700416         59184     92%       17567     10%       /usr
```

```
/dev/hd9var     16384      11960     28%       1286      32%      /var
/dev/hd3        16384      11432     31%        87       3%       /tmp
/dev/hd1       233472     202540     14%        715      2%       /home
$
$df -s
Filesystem      512-blocks   Free*    %Used    Iused    %Iused Mounted on
/dev/hd4         917504      270888    71%      2008     1%      /
/dev/hd2        1400832      118368    92%     17567     10%     /usr
/dev/hd9var      32768       23920     28%      1286     32%    /var
/dev/hd3         32768       22864     31%       87      3%      /tmp
/dev/hd1        466944      405080     14%       715      2%     /home
$
```

在 Solaris 系统中 df 命令的输出与 AIX 系统中 df 命令的输出形式有差别，但包含的信息量几乎相同。

例　下面是在一个应用较为复杂的 Solaris 系统中使用 df 命令。

```
#df -k
Filesystem                  Kbytes     used    avail    capacity   Mounted on
/dev/dsk/c0t0d0s0          1489367  789621  640172        56%        /
/dev/dsk/c0t0d0s5          2056211 1106065  888469        56%        /usr
/proc                            0       0       0         0%        /proc
fd                               0       0       0         0%        /dev/fd
mnttab                           0       0       0         0%        /etc/mnttab
/dev/dsk/c0t0d0s4          2056211 10624426 970099          52%          /var
swap                       1078736      16  1079720        1%         /var/run
swap                       1290752  211032  1079720       17%         /tmp
/dev/dsk/c0t0d0s6           482455       9   434210        1%         /home
/dev/dsk/c0t0d0s3          4226867 1935711  2248899       47%         /opt
/dev/dsk/c0t0d0s7          4966562 2491578  2425319       51%         /cafs
/dev/vx/dsk/aimclog        4131866  577366  3513182       15%      /aimcwtmp
/dev/vx/dsk/rootdg/swap1   4986199 1049105  3887232       22%         /swap1
/dev/vx/dsk/aimcque        4131866    6811  4083737        1%        /aimcque
/dev/vx/dsk/mail2dg/ail2vol 200023746 1279275 178742097    1%          /m1
/dev/vx/dsk/mail1dg/ail1vol 200023746 1273957 178747415    1%          /m0
```

2. du 命令：显示文件或目录的块数

df 命令只能显示整个文件系统的信息，无法知道硬盘上个别目录或整个目录的情况，而 du 命令在这方面却有优势。命令格式：

du [-a | -s] [-k] [-l] [-r] [-x] [File …]

其中选项：

a：列出每个文件或目录磁盘使用情况。

s：只列出总的使用标志。

k：输出以 k 为单位。

l：对多个链接平均分配块数。

r：显示未访问文件和目录的名称，通常默认。

x：只显示在单个设备上的文件和目录的估计大小。

例　在 AIX 系统中路径/home/jijzhou 下用 du 命令。

```
$ls jijzhou
db2             db2.pdf         lliu        v7-dbacert.pdf    y1
netgate         hlliu           mbox        x
$
$du jijzhou
42429     jijzhou
$du -a    jijzhou
1         jijzhou/x
1         jijzhou/y1
7696      jijzhou/v7-dbacert.pdf
4         jijzhou/.sh_history
17360     jijzhou/db2.pdf
17360     jijzhou/db2
1         jijzhou/netgate
4         jijzhou/mbox
1         jijzhou/lliu
1         jijzhou/hlliu
42430     jijzhou
$
$du -s    jijzhou
42430     jijzhou
```

```
$du -k    jijzhou
21215     jijzhou
$du -l    jijzhou
42430     jijzhou
$du -r    jijzhou
42430     jijzhou
$du -x    jijzhou
42430     jijzhou
$
```

3. quota 命令：磁盘限额

UNIX 是多用户操作系统，为了保证用户的公共使用，通常情况下需要限制每个用户在系统中可用磁盘空间的数量。磁盘限额分为软件限额和硬件限额。

软件限额和硬件限额都是当用户的使用空间超过系统分配的使用空间时，通过如下信息警告用户超限：

/home:warnig,user disk quota exceeded.

此时用户必须将使用的磁盘空间减少到限制之内，如果不采取措施而继续使用，就会反复出现警告：

/home:warnig,user disk quota exceeded.

No space left on device.

（1）用 edquota 命令为用户定义可用磁盘空间

用 edquota 命令可以为用户定义可用磁盘空间，在进行磁盘空间定义之前需要考虑系统可用磁盘空间的总量及系统目前和将来的用户数，以便正确确定用户可用磁盘空间大小。命令格式：

用户定义 edquota [-u] [-p Proto-UserName] UserName …

用户组定义 edquota [-g [-p Proto-GroupName] GroupName …]

其中选项：

u：用户；

g：用户组；

p：为另一个用户或用户组定义磁盘限额时，可以复制原来已经定义好限额值的用户或用户组；

Proto-UserName：原来已经定义好限额值的用户；

Proto-GroupName：原来已经定义好限额值的用户组。

使用 edquota 命令实际上就是编辑一个文件 quotas（该文件还可以用 vi 编辑），

该文件位于系统根目录下，有些版本要求用 touch 命令手工创建，有些版本用 edquota 命令编辑时直接创建。

（2）用 quota 命令检查某个用户或用户组的限额

命令格式：

quota [-v] [UserName]　[GroupName]

其中选项：

v：表示列出指定用户或用户组的当前使用限制，如果命令中无此选项则表示只输出指定的用户或用户组是否超过其限制；

UserName：表示受限的用户帐号；

GroupName：表示受限的用户组帐号。

除了为用户编辑限额外，使用 quotaon 命令还可以为用户帐号打开文件系统进行限额。命令格式：

quotaon [file system]

其中 fiel system 是用户限额的文件系统，如 quotaon /home 命令为/home 文件系统激活作了限额。

除了运行 quotaon 命令外，限额文件系统中/etc/vfstab 文件内的条目必须指定 userquota 或 groupquota 选项。

使用命令 quotaoff 为文件系统关闭限额。命令格式：

quotaoff [file system]

2.4.3　文件系统检测

在系统掉电、非正常关闭以及系统内核出错等异常事件发生后，可能导致文件系统破坏或文件内容不一致。这种情况下，如果文件系统没有完全崩溃，在系统重新启动时会自动检测文件系统的一致性，大多数情况下文件系统检测能够修复一些问题。

用文件系统检测（fsck）程序检测文件系统。

检测文件系统之前首先需要查看是否需要检测，通过命令 fsck –m /dev/rdsk/sntndnsn 可以检测指定文件系统的超级块中的状态标志以决定是否要检测。

如果不带参数则检测/etc/vfstab 中 fsck pass 值大于 0 的所有 UFS 文件系统。

例　#fsck –m /dev/rdsk/c0t0d0s5

```
** /dev/rdsk/c0t0d0s5

ufs fsck:sanity check: /dev/rdsk/c0t0d0s5 needs checking

#
```

结果是需要检测。

```
#fsck –m /dev/rdsk/c0t0d0s6
** /dev/rdsk/c0t0d0s6
ufs fsck:sanity check: /dev/rdsk/c0t0d0s5 okey
#
```

结果是不需要检测。

如果确定需要检测，首先卸载文件系统，然后用命令 fsck 检测/etc/vfstab 中 fsck pass 值大于 0 的所有文件系统(也可以指定加载点目录或/dev/rdsk/cntndnsn 作为 fsck 的参数），并显示所有不一致的消息。

例 检测/dev/rdsk/c0t0d0s5 并纠正不正确的块计数。

```
#fsck /dev/rdsk/c0t0d0s5
checkfilesys: /dev/rdsk/c0t0d0s5
** Phase 1 –Check Block and Sizes
INCORRECT BLOCK COUNT 1=2529 (6 should be 2)
CORRECT?Y

** Phase 2 –Check Pathnames
** Phase 3 –Check Connectivity
** Phase 4 –Check Reference Counts
** Phase 5 –Cylinder Groups
Dynamic 4.3 FFFS
929 files, 8928 used,2851 free(75 frags, 347 blocks, 0.6% fragmentation)
/dev/rdsk/c0t0d0s5 FILE SYSTEM STATE SET TO OKAY
**** FILE SYSTEM WAS MODIFIED ****
$
```

2.5 文件系统的启用

每个 UNIX 系统在安装的硬盘上至少有两个文件系统。

· 根文件系统（也称为主文件系统）：包含了组成操作系统的程序和目录，在文件目录中用“ / ”表示。

· /stand 文件系统：包含引导程序和核心/stand/unix，此文件系统是只读的。

用命令 df 可以看到系统引导后的文件系统。

```
$df
```

```
Filesystem      512-blocks      Free %Used      Iused      %Iused      Mounted on
/dev/hd4          917504        270888   71%       2008        1%      :/
/dev/hd2         1400832        118368   92%      17567       10%      /usr
/dev/hd9var        32768         23928   27%       1286       32%      /var
/dev/hd3           32768         22864   31%         87        3%      /tmp
/dev/hd1          466944        405080   14%        715        2%      /home
$
```

当前系统中有文件系统:/、/usr、/var、/tmp、/home。这些文件系统都已经 Mount on。

2.5.1　加载与卸载

在 UNIX 系统中如果要使用一个文件系统，必须首先 Mount on。Mount on 也称为加载。由上面的 df 命令结果可见，用加载的方法使用文件系统，加载的所有文件系统都有一个加载点，即与 Mounted on 对应的列。

根文件系统总是加载的。将文件系统连接到系统可用的目录层次中的过程称为加载，对应于目录层次的点为加载点。反之就是卸载。

2.5.2　加载本地文件系统

要将已经建立好的本地文件加载到系统，首先必须确定加载点。加载点必须是操作系统已经被加载了的文件系统所连接的目录。

本地加载有两种方式：永久性自动加载（系统重新引导后仍然生效）；临时性手动加载（系统重新引导后不再生效）。

下面以 Solaris 系统中的永久性自动加载和临时性手动加载为例说明加载步骤。

（1）永久性自动加载步骤

首先创建要加载的文件系统，如用户 user1 下的目录 com-info。

然后确定加载点，如/usr/com。

在虚拟文件系统表/etc/vfstab 中创建入口，加入行：

```
/export/user1/com-info  -  /usr/com  ufs  -  yes  rw
```

则文件/etc/vfstab 中的内容为：

```
#cat /etc/vfstab
……
/export/user1/com-info  -  /usr/com  ufs  -  yes  rw
……
#
```

系统重新引导则自动生效。

卸载则在文件/etc/vfstab 中删除所加行，并重新引导系统。

（2）临时性手动加载、卸载

加载（将/export/user1/com-info 加载到/user/com 下）：

首先创建要加载的文件系统，如/export/user1 下的目录 com-info。

然后确定加载点，如/usr/com。

执行下列命令：

```
mount /export/user1/com-info /usr/com
```

命令执行完后立即生效。

```
$ls /usr/com
com-info
```

卸载：

```
umount /user/com
```

卸载之后再用 ls 命令：

```
$ls /user/com
(com-info 不存在)
$
```

2.5.3 远程加载（共享网络文件系统）

实现远程文件系统加载时既需要在远程主机上将要加载的文件设为共享，又需要在本地主机中挂接远程共享文件。

下面以 Solaris 系统为例介绍怎样实现远程加载。

（1）在远程主机上以超级用户身份启动 mounted 进程（如果 mounted 进程已经启动则不需再启动）：

```
#ps –ef |grep mounted
```

如果后台进程 mounted 已经启动，则直接进入共享；否则需用下面的命令启动进程 mounted：

```
#/etc/init.d/nfs.server start
```

再次检查进程 mounted 是否启动：

```
#ps –ef |grep mounted
```

如果进程 mounted 已经启动则完成，否则一直要进行启动进程 mounted。

（2）将要加载的目录（或文件）设为共享，如：/user/com-info，（注意：必须是远程主机上的超级用户）：


```
#share –F nfs –o ro /user/com-info
```

这里需要注意的是，如果想让远程机器启动后自动共享，可以在远程机器上通过文件配置方式实现：

在文件/etc/dfs/dfstab 中加入两行：

```
/etc/init.d/nfs.server
Share –F nfs –o ro /user/com-info
```

（3）在要共享的机器（本地机器）上以超级用户身份完成挂接：

```
#mounted remoted-system-name:/user/com-info /mnt
```

其中 remoted-system-name 是远程主机的 IP 地址或有效主机名；/user/com-info 是要共享的目录或文件；/mnt 是本地挂接点。

挂接成功后使用命令 ls，在目录/mnt 下可见/com-info：

```
#ls /mnt/
com-info
#
```

（4）如果不再需要共享时，只需要在远程主机上去掉所加上的配置：

如果使用文件配置则去掉文件中加入的两行。

如果使用 Share 命令，就用 unshare 命令去掉共享，并在本地主机上完成卸载：

```
#unmounted /mnt
#ls /mnt
#
```

2.6 文件系统的备份与恢复

计算机系统硬件可能发生故障，操作系统的系统管理员和用户可能犯错误；也有可能发生了天然灾害或遭到了人为攻击，最坏的结果是系统不能使用。作为一个操作系统的系统管理员，自然希望当意外发生以后被破坏的系统可以恢复正常工作，丢失的内容可以复得。因此系统管理员必须能够进行系统的备份与恢复。

2.6.1 备份

执行备份的目标可以是用户文件或整个文件系统。

备份要决定的关键问题是：

- 备份频率；
- 需要备份的内容；
- 用什么备份介质；
- 备份设备的选择。

对于备份频率，如果系统中的用户越多，则备份文件系统的频率应越高；如果系统的内容变化越大，备份文件系统的频率也应越高。从备份频率和备份内容考虑，通常的策略是每周全部备份一次或两次，每天进行部分备份。

选择备份介质时要考虑备份介质的速度和存储量，传统备份介质通常是磁带、光盘等。存储容量高、读写速度快的新介质的出现使得备份介质的选择面更广了。同时，备份介质的安全存放和使用也需考虑。

选择备份设备。由于 UNIX 系统中大磁盘空间（GB 或 TB）的使用，使得可选的备份设备有：

- 各种格式的磁带驱动器；
- 可拆卸的硬盘；
- CD-ROM；
- Magneto-optical cartridges（optical jukebox）；
- WORM 驱动器（一次写、多次读）。

系统备份通常会在单用户模式下进行，很少在系统活动状态中进行。主要是因为备份过程占用了大量系统资源（其中最主要的是 CPU 时间），因为备份时有大量的硬盘访问，导致系统性能降低。另外，当系统正在进行备份时，出现的用户访问系统会打开文件，而打开的文件不能备份。因此在系统备份时最好限制用户的访问并关闭应用。

备份软件支持备份自动进行：大部分备份软件允许用户设置备份调度表，备份进程可以自动进行周期性的初始化而不需手工干涉，这就避免了人为因素造成忘记备份或无时间备份。

系统备份中不包括/tmp、/var/tmp、/usr/var/tmp 目录中的文件，因为它们通常存放的是临时文件。另外，目录/proc 是一个内核向用户提供的系统内存进程信息，是临时性的，也不用备份。

备份有标准格式和定制格式两种。

在 UNIX 系统中支持系统备份的工具主要有：

- dump 和 restore；
- volcopy 和 labelit；
- tar、cpio、dd；
- 第三方厂商的商业备份工具。

2.6.2 备份工具 dump 和 restore

dump 和 restore 是 UNIX 系统提供的备份和恢复工具，各种版本的 UNIX 系统中都提供了该工具。dump 和 restore 备份和恢复时不提供日志文件。

1. dump 命令

dump 命令执行文件系统增量的存储操作，是一种高质量备份工具，可以备份到磁带、磁盘或一个磁盘文件。

dump 可以进行：

· 指定文件备份；

· 整个文件系统备份或增量备份；

· 一个确定日期后改变的文件备份；

· 上次备份后改变的文件备份。

表 2.1 是不同 UNIX 版本中的 dump 命令。不同版本中 dump 命令的名称和格式也不同。

表 2.1　不同 UNIX 版本中的 dump 命令

操作系统	dump 命令
AIX	backup 和 rdump
Solaris	ufsdump
HP-UX 和 SunOS、IRIX	(r)dump
SCO	xdump （只能处理 XENIX 文件系统）
SGI	dump 和 xfsdump
Tru64 Unix	dump 和 vdump
Linux	dump

常用的 dump 命令的格式为：

#dump levelunbdsf blkg-factor density size device-name file_system

其中参数：

level：指定 dump 进行的备份级别，可以是 0~9，其中 9 是默认级别。

· 0 级：全部文件系统备份。

· 1 级：增量备份，从上次 0 级备份以后发生的一切改变都要进行备份。而上一次更低一级的备份日期则从 dumpdates 文件（在 AIX、SunOS、Solaris、HP-UX 9.x、Linux、IRIX 系统中是/etc/dumpdates，在 HP-UX 10.0 中是/var/adm/dumpdates，在 SCO UNIX 中是/etc/ddate）中得到（由于 dump 没有日志文件，所以只能从 dumpdates 文件中得到）。

· 2~9 级：每个级别均备份自上一次向下低一级的备份以后的所有改变。如果没有低一级的备份，则采用更低一级的备份，例如，如果是一个 2 级备份，则要备份上一次 1 级备份以后的改变；如果没有 1 级备份，则备份上一次 0 级备份以后

的改变。

b：指定 dump 应该使用的块因子。块因子乘以物理块大小就是 dump 所写的完整块的大小。大多数 UNIX 系统物理块的大小是 1 024 字节，当指定块因子为 128 时，dump 实际写的块大小为 131 072 字节。

u：告诉 dump 更新 dumpdates 文件。由于 dumpdates 文件中列出了每个文件系统的原始设备以及设备上最近一次每个级别的备份，作增量备份时要用到这个文件，所以必须更新。如果没有该文件，系统管理员必须创建它。否则仍然可以使用 dump 命令，但不可能更新。dumpdates 文件的格式为：

```
/dev/rdsk/c0t1d0s0        0 Thu May 08 23:05:12 2003
/dev/rdsk/c0t2d0s0        0 Fri May 09 23:05:12 2003
/dev/rdsk/c0t3d0s0        0 Sat May 10 23:05:12 2003
/dev/rdsk/c0t4d0s0        0 Sun May 11 23:05:12 2003
/dev/rdsk/c0t5d0s0        0 Mon May 12 23:05:12 2003
```

从该文件可以得知备份的设备名、dump 级别（全部为 0 级）和时间。

n：告诉 dump 当它完成时要通知系统操作员组的成员（在文件/etc/group 中指定的操作员组中的用户）。如果备份时发生下列情况则需系统操作员组注意：

- 磁带到头或者 CD 没空间；
- 备份驱动器发生故障而导致写错误；
- 从源盘中读取数据错误。

d 和 s：告诉 dump 备份卷有多大。dump 使用这些数字来估计要使用何种磁带。d 为指定密度；s 为按英尺计算的磁带大小。如果想使用整个卷则值应该取大些。

f：指定备份设备。备份设备可以是磁带设备，或磁盘上的一个文件，或是一个 CD。如果想用磁带驱动器上的硬件压缩功能，需选择支持压缩功能的设备。

如果想把内容发送到远程设备上，应在设备名前加上远程主机名，即“远程主机名：设备”。同时还需在远程主机的/etc/.rhosts 文件中增加信任机。有的系统为了安全取消了 rsh 功能，这时用 ssh 功能代替。

如果不是远程机的信任机，则远程备份会受到拒绝，显示信息为：

```
Permission Denid.
```

超级用户可以用命令验证是否为远程主机的信任机：

```
#rsh remote_system uname –a
```

W，w：执行一次空运行来得到需要备份文件系统的信息。选项 w 显示所有文件系统信息，选项 W 仅显示需要备份的文件系统信息。

上面这些选项大多数都有默认值，选项及其参数不一定要在命令中给出。

例　把/opt 目录完全备份到/dev/rmt/1cbn 的本地磁带驱动器上。

#dump 0unbdsf 128 121000 11500 /dev/rmt/1cbn /opt

其中选项 b 指定的块因子为 128，选项 d 和 s 告诉 dump 备份卷为 121 000 和 11 500，选项 f 指名使用/dev/rmt/1cbn 设备来备份。

例　把/home 目录完全备份到/backup/home.dump 的光学驱动器或者 CD 驱动器上。

#dump 0unbdsf 128 121000 11500 /backup/home.dump /home

例　把/home 目录完全备份到 ccss（远程主机名）上的远程磁带驱动器/dev/rmt/1cbn 上。

#dump 0unbdsf 128 121000 11500 ccss:/dev/rmt/1cbn /home

需要注意的是应该尽量避免跨卷创建 dump 备份。因为如果在一个卷中作了 dump 而卷空间没有占满，在跨到第二个卷时，实际上 dump 会在第二个卷中将对应第一个卷的部分先添满，这样这部分空间就浪费掉了。

在 Solaris 系统中 dump 命令的名称变为 ufsdump，并且还增加了一些额外的选项：

l：如果磁带已满而 dump 还未完成，该选项会自动弹出磁带等待两分钟以便插入另一个磁带。通常该选项配合自动顺序加载器使用；

o：为了避免磁带被其他进程重写，在备份结束后自动弹出磁带；

a：把 dump 的内容写到存盘文件（archive_file）中，提供给 ufsrestore 命令检查是否一个文件已经位于给定的卷上而不必非得安装；

v：该选项对备份的和实际的文件系统进行比较，但该选项执行时要求文件系统卸载。

例　下面是一个 Solaris 系统中的所有文件系统全部备份的处理文件。

```
$cat dumpfile
ufsdump 0uf /dev/rmt/0n /dev/dsk/c0t0d0s0
ufsdump 0uf /dev/rmt/0n /dev/dsk/c0t0d0s5
ufsdump 0uf /dev/rmt/0n /dev/dsk/c0t0d0s4
ufsdump 0uf /dev/rmt/0n /dev/dsk/c0t0d0s6
ufsdump 0uf /dev/rmt/0n /dev/dsk/c0t0d0s3
ufsdump 0uf /dev/rmt/0n /dev/dsk/c0t0d0s7
ufsdump 0uf /dev/rmt/0n /dev/vx/dsk/aimclog
ufsdump 0uf /dev/rmt/0n /dev/vx/dsk/aimcwtmp
ufsdump 0uf /dev/rmt/0n /dev/vx/dsk/aimcque
```



```
ufsdump 0uf /dev/rmt/0n /dev/vx/dsk/mail2dg/mail2vol
ufsdump 0uf /dev/rmt/0n /dev/vx/dsk/mail1dg/mail1vol
```

#

运行备份过程：

#./dumpfile

DUMP:写入 32 千字节记录

DUMP:本级 0 日期转储：2003 年 07 月 07 日 星期一 16 时 55 分 55 秒

DUMP:上一级 0 日期转储：记元

DUMP:转储/dev/rdsk/c0t0d0s0(mailsvr2:/)到/dev/rmt/0n

DUMP:映射（传送Ⅰ）[规则文件]

DUMP:转储（传送Ⅱ）[目录]

DUMP:估计 1656884 块（808.81MB）

DUMP:转储（传送Ⅲ）[目录]

DUMP:映射（传送Ⅳ）[规则文件]

DUMP:60.79% 完成，在 0:06 完成

DUMP:在 1 卷上的 1656382 块[808.78MB]速率是 874KB/sec

DUMP:转储已完成

DUMP:2003 年 07 月 07 日 星期一 16 时 55 分 57 秒上的 0 级转储

DUMP:写入 0 日期转储：2003 年 07 月 07 日 星期一 17 时 12 分 42 秒

DUMP:上一级 0 日期转储：记元

DUMP:转储/dev/rdsk/c0t0d0s5(mailsvr2:/usr)到/dev/rmt/0n

DUMP:映射（传送Ⅰ）[规则文件]

DUMP:转储（传送Ⅱ）[目录]

DUMP:估计 2254376 块（1100.77MB）

DUMP:转储（传送Ⅲ）[目录]

DUMP:映射（传送Ⅳ）[规则文件]

DUMP:63.91% 完成，在 0:05 完成

DUMP:在 1 卷上的 2260350 块[1103。69MB]速率是 1098KB/sec

DUMP:转储已完成

DUMP:2003 年 07 月 07 日 星期一 17 时 12 分 42 秒上的 0 级转储

DUMP:写入 32 签字节记录

DUMP:本级 0 日期转储：2003 年 07 月 07 日 星期一 17 时 30 分 16 秒

DUMP:上一级 0 日期转储：记元

DUMP:转储/dev/rdsk/c0t0d0s4(mailsvr2:/var)到/dev/rmt/0n

DUMP:估计 2053468 块（1002.67MB）

……（省去）

2. dump 自动完成备份

可以使用 dump 自动定期完成系统备份。dump 自动备份是一个复杂的过程，通常将该过程写入一个 Shell 脚本文件中，该 Shell 脚本文件取名为 hostdump.sh。大多数 UNIX 版本支持脚本文件 hostdump.sh。

在使用 hostdump.sh 之前必须准备好一个卷、一个设备名和主机名，然后 hostdump.sh 执行自动检查所有文件系统的名字及类型并调用合适的命令执行，在卷上放置两个额外的 tar 文件：第一个列举了卷上所有的文件系统和被用来备份的命令，该 tar 文件位于磁带第一个分区的首部；第二个用于存放 dump 备份完成后再读取的每个备份表的内容信息。

如果要将一些参数写到 hostdump.sh 脚本文件中，使用命令：

```
#hostdump.sh level device logfile system_list
```

其中参数：

level：dump 的级别。

device：非回绕磁带驱动器（如/dev/rmt/0n）或磁盘上的一个文件、CD-ROM。

logfile：日志文件的绝对路径名，由于 hostdump.sh 脚本包括 stdio 和 stderr，所以该路径应该同卷相关联。

system_list：要备份的系统列表。脚本自动检查每个系统的 fstab 文件，并创建一个要备份的文件系统的列表。如果要备份整个系统，最好使用选项 system:/filesys 来指出要备份的文件系统；如果要限制包括的文件系统，则逐一列出文件系统：

```
system1　system2　system3:/filesys　system4:filesys　system5
```

不同版本的操作系统 fstab 文件的位置可能有区别，如表 2.2 所示。

表 2.2　不同 UNIX 版本中的 fstab 文件位置

操作系统	fstab 文件位置
AIX	/etc/filesystems
Solaris	/etc/vfstab
SunOS、IRIX、Linux、Tru64 Unix	/etc/fstab
SCO	/etc/default/filesys
HP-UX	/etc/checklist

如果不想备份一个完整的系统，则在系统名之后加上要备份的文件名，可以有多个，多个之间用空格隔开。

如果需要备份的系统有多个，要求按照出现顺序写到卷上，则 hostdump.sh 脚本在执行时自动检查每个系统的 fstab 文件，并创建一个要备份的文件系统列表。

例 把系统 ccss1、ccss2 进行 0 级备份到磁带驱动器/dev/rmt/1cbn 上，日志记录到/var/log/backup.log 文件。

#hostdump.sh 0 /dev/rmt/1cbn /var/log/backup.log ccss1 ccss2

例 把系统 ccss1 中根下的系统 home1、home2 进行 0 级备份到磁带驱动器/dev/rmt/1cbn 上，日志记录到/var/log/backup.log 文件。

#hostdump.sh 0 /dev/rmt/1cbn /var/log/backup.log ccss1:/home1 ccss1:/home2

3. 特殊情况的文件系统

如果需要排除一些文件系统时，可以将要排除的文件系统名写入 fstab.exclude 文件中，并将 fstab.exclude 文件放在所要排除的文件系统的系统上；如果需要包含没有在 fstab 文件中列出的文件系统，或者正常情况下被 hostdump.sh 所排除的文件系统，可以将文件系统名写入 fstab.include 文件中，文件 fstab.include 位于要备份的系统上。

例 假如在 Solaris 系统中，系统 prids 上需要排除/home 文件系统。通常情况下/home 文件系统会被备份，因为系统备份时/home 包含在/etc/vfstab 中。这时需要创建文件/etc/vfstab.exclude，在文件/etc/vfstab.exclude 中增加一行/home。

文件系统备份时通常不包含文件系统/tmp，假如现在备份需要包含/tmp 文件系统，则要创建文件/etc/vfstab.include，在文件/etc/vfstab.include 中增加一行/tmp。

现在文件系统通常会大于一个卷，此时用 fstab 方式备份需要用以下方法。

· 如果使用 hostdump.sh 自动备份，则在文件 fstab.exclude 中应该排除足够多的文件系统以使剩余的文件系统能够容纳在一个卷中。

· 运行 hostdump.sh 两次。第一次备份 fstab 文件中有但没有被 fstab.exclude 排除的文件系统；第二次只备份列于命令行中的文件系统。

例 现在要用 hostdump.sh 备份 Solaris 系统 pridns，完全备份该系统一个卷太小，现在用两个卷来备份。

备份步骤如下：

(1) 将整个文件系统分为两部分，由于用户主目录在/home 下，所以这样划分文件系统：

卷 1：操作系统文件/、/usr、/var、/opt、/etc 等。

卷 2：/home。

(2) 在第一个卷上备份除目录/home 外的整个系统，因此在 vfstab.exclude 中排除目录/home，即在文件 vfstab.exclude 中增加行：

/home

(3) 运行备份命令：

#hostdump.sh 0 dev1 /var/log/backup.log1 pridns

会备份除/home 之外的整个系统。

(4) 备份/home 到其他卷上：

#hostdump.sh 0 dev2 blocking-factor /var/log/backup.log2 pridns:/home

4. restore 命令

restore 命令用于检查 dump 程序创建的转储以恢复新文件或整个文件系统。命令格式：

restore [-ctrox]vbsfy blocking-factor file-number device-name

其中选项：

c：全部恢复，存储介质上的所有文件将被恢复。

t：用于显示卷的内容表，检查 dump 卷上包含哪些文件。在恢复时如果对于备份的内容不能确定或位置不能确定时，用该选项可以查看。该选项可以与 grep 命令结合使用。

r：指名卷的整个内容应该被恢复到当前工作目录下。在完全清楚要恢复的文件系统的情况下，把一个 dump 备份的整个内容读取到一个文件系统中以便恢复想要恢复的完整文件系统。

该恢复需要注意的是：

· 首先必须进入到要恢复的文件系统中；

· 如果是增量备份，现在要完全恢复，则必须依赖于增量备份所基于的更低级的 dump，例如，如果有三个 dump 备份：一个是从 1 日开始的 0 级备份，一个是从 5 日开始的 1 级备份，一个是从 10 日开始的 2 级备份，如果用选项 r 读取 0 级备份之后直接读取 2 级备份，而不读取 1 级备份，则 restore 会报告出错。

o：恢复文件时覆盖原来的文件。

x：用于提取恢复的文件名和路径名。

i：restore 与 tar 和 cpio 不同的地方是允许执行交互式恢复。

v：用该选项，restore 会显示大量的信息。

s：在开始读取磁带之前先跳过指定个数的磁带文件。

b：指定要读取的卷的块因子。查命令手册表得到乘数，将块大小除乘数即是块因子。通常情况下，大多数操作系统的块大小设置为 512。

f：指定要使用的备份驱动器（或者磁盘文件）的文件名。

y：恢复过程禁止询问。

内容表：restore 恢复时要读取 dump 备份时写的内容表以验证其内容。

通常在磁带上 dump 文件的格式如图 2.5 所示。在 dump 所创建的磁带卷上可能存在多个 dump 文件，每个文件用标记 EOF（End-Of-File）结束。如果要得到 dump 文件的内容表，可以通过选项-s 指明文件，也可以手工在磁带上定位文件，然后 restore 就可以读取该文件。

Dump 文件 1	EOF	Dump 文件 2	EOF	……	EOF	Dump 文件 n	EOF

图 2.5　磁带上 dump 文件格式

在 Solaris 系统中恢复命令变为 ufsrestore，其他选项与 restore 相同。

例　用命令读取 device 上的 dump 备份内容表，并重定向到文件/tmp/dump.list，然后从该文件中查找要恢复的文件 restorefile。

```
#restore tfy device >/tmp/dump.list
#grep restorefile /tmp/dump.list
2546                ./home/restorefile
```

例　恢复位于磁带/dev/rmt/1cbn 上用块因子 32 创建的 dump 备份的全部内容，使用命令：

```
#restore rvbfy 32 /dev/rmt/1cbn
```

该命令从最低级开始装载所有增量备份，直到最近的增量备份被装载以后才结束。如果有多个同级备份，则只需装载最近的一个。

例　在 Solaris 系统中恢复一个完全备份的文件系统。

· 首先加载要恢复的文件系统到临时加载点：mount /dev/dsk/cntndnsn。

· 为了安全，磁带要写保护。

· 更换到加载点目录，输入 cd /mnt 并回车。

· 将 0 级备份磁带的第一卷插入磁带驱动器。

· 输入 ufsrestore rvf /dev/rmt/n 并回车，多卷恢复在提示时取出第一个磁带后依照命令依次放入余下的磁带，0 级恢复完成。

· 输入 ls 并回车，显示该文件系统的文件列表，检测列表验证所有文件已经恢复。

· 删除 ufsrestore 创建的 restoresymtable，输入 rm restoresymtable。

· 从临时加载点/mnt 卸载文件系统：

```
cd /
umount /mnt
```

· 检测文件系统的不一致性，输入 fsck /dev/rdsk/cntndnsn。

· 对已恢复的文件系统执行 0 级备份，因为 restore 已经重新定位文件并更改

分配。

插入新磁盘，输入 ufsdump 0uf /dev/rmt/unit /dev/rdsk/cntndnsn。

在永久加载点加载文件系统：输入 mount /dev/dsk/cntndnsn，恢复的文件系统被加载并可使用。

例　恢复位于磁带/dev/rmt/1cbn 上用块因子 32 创建的 dump 备份文件/etc/passwd 和/etc/shadow 的内容，用命令：

#restore xbvsfy 32 /dev/rmt/1cbn ./etc/passwd ./etc/shadow

例　#restore tsbfy 2 32 /dev/rmt/1cbn

restore 按内容表读取文件，要读取的是磁带上的第二个文件，dump 创建该文件的块因子是 32，文件位于设备/dev/rmt/1cbn 上。

如果要用手工方式定位文件，则必须借助于磁带操作命令 mt，mt 命令对应的操作有：

· status：检查磁带设备的状态；
· rewind：回绕磁带至开始位置；
· offline：弹出磁带；
· fsf：前向移动文件，fsf n 表示磁带前向移动 n 个文件；
· fsr：前向移动记录，磁带 dump 不需要该命令。

例　#mt –t /dev/rmt/0cbn rewind 表示回绕磁带/dev/rmt/0cbn。

#mt –t /dev/rmt/0cbn offline 表示弹出磁带/dev/rmt/0cbn。

#mt –t /dev/rmt/0cbn fsf 1 表示磁带/dev/rmt/0cbn 前进到第二个文件处。

#mt –t /dev/rmt/0cbn status 表示得到磁带/dev/rmt/0cbn 的状态。

一旦把磁带移动到指定位置则可以利用命令读取磁带上的文件：

#restore tbfy 2 32 /dev/rmt/1cbn

此时不用选项-s。

内容表可以通过标准输出得到，也可以重定向到一个文件。这样，每个卷可以单独存在一个内容表，管理起来非常方便。

5. dump 和 restore 使用缺点

· 运行 dump 时，文件系统必须是不活跃的，即文件系统处于未安装或系统处于单用户模式下，否则输出结果可能不一致。如果 dump 是在活跃的文件系统下作磁带备份的，则 restore 进行增量恢复时可能产生混乱。
· dump 命令有时不能处理打开的文件和其他问题。
· dump 有很多版本，相互之间兼容性不好。

6. 学完 dump 和 restore，总结需要掌握的知识

· 如何使用 dump 来备份一个文件系统。
· 如何在多卷上进行备份。
· 如何得到一个 dump 卷的内容。
· 如何操作卷以及从一个由 dump 创建的备份中进行恢复。
· dump 和 restore 的局限性。

2.6.3 tar、cpio、dd

1. tar、cpio 与 dump 的主要区别

· dump 命令会在每个卷的开头位置写入一个内容表，当进行交互恢复 restore 时，该索引会被读取，在该索引表上可以运行 cd、ls 命令查看、选择要恢复的文件。这是 dump 优于 tar、cpio 的地方，tar、cpio 不具备此功能。
· dump 支持远程设备的备份，而 cpio 不支持。
· tar 和 cpio 的最大优点是通过文件系统来访问文件而与文件系统无关，然而，每种平台上都有不同版本的 tar 和 cpio，所以它们并不都兼容。

2. cpio

cpio 命令通过拷贝的方式实现文件或文件系统的备份与恢复，所以该命令也提供文件及目录的拷贝。cpio 有三个主要的选项与其使用相对应。

· cpio –o：创建一个备份。
· cpio –i：从备份中恢复。
· cpio –p：把文件从一个文件系统拷贝到另一个文件系统。

用 cpio 创建一个备份的 cpio 命令格式为：cpio –o [aBcv]

其中选项：

a：将 atime 重设成备份前的值。由于 cpio 经过文件系统实现备份，当 cpio 读取一个要备份的文件时会改变该文件的访问时间（atime）。这样一来，如果每天都作 cpio 备份，则备份程序会改变文件的访问时间，造成所有文件都会被使用的假像。所以 cpio 应该把 atime 重设回文件的原始值。

B：指定块因子大小。

c：指定 I/O 块大小。该选项与 B 选项是互斥的。

v：指定将备份的文件列表打印到标准出错（stderr）中。cpio 备份的实际数据将送往标准输出（stdout）。

例 在本地驱动器 device（可以是本地文件、设备）上创建/home 的完全备份。

```
#cd /home
#touch level.0.cpio.timestamp
#find . –print | cpio –oacvB>device
```

在本地驱动器 device（可以是本地文件、设备）上创建/home 的增量备份：

```
#cd /home
#touch level.1.cpio.timestamp
#find . –newer level.0.cpio.timestamp –print | cpio -oacvB>device
```

3. 用 cpio 备份到远程设备

由于 cpio 不支持远程设备备份，如果要做远程设备备份则要借助于管道将其输出到一个 rsh 命令。命令格式为：cpio –o [aBcv] | rsh remote_system dd of=device bs=5K

例　在远程磁带驱动器 device 上创建/home 的完全备份。

```
#cd /home
#find . –print | cpio –oacvB | (rsh remote-system dd of =device bs=5120)
```

cpio 备份的文件列表

cpio 命令从标准输入（stdio）中得到文件列表，默认情况下数据流被发送到标准输出（stdout）。可以使用下列两种方式得到具有相对于当前工作目录的备份文件列表：

· 用 ls 或 find 命令

```
ls | cpio -oacvB
```

· 创建一个包含文件，然后将包含文件发送到 cpio 的 stdin

```
cat /tmp/include | cpio –oacvB 或  cpio –oacvB</tmp/include
```

4. 用 cpio 恢复

在能够读取 cpio 备份的基础上可以选择：

· 全恢复、部分恢复；
· 恢复到当前目录还是文件系统；
· 用模式匹配进行恢复；
· 交互式恢复；
· 读取文件内容表。

用 cpio 进行恢复的命令格式为：

```
cpio –i [Bdkmtv] [patterns]
```


其中选项：

v 与 B 和 cpio 备份时相同；

t：cpio 结果不是恢复数据而是生成一个内容表；

k：作恢复时跳过卷中的坏区；

d：cpio 在需要时创建目录；

m：恢复文件备份时的原始时间。如果没有该选项，cpio 的默认动作就是将恢复后的文件修改时间设为恢复时间；

u：用户无条件地覆盖所有文件；

pattern：恢复匹配模式文件，如果恢复的是匹配模式之外的文件则用选项 f；

r：如果要恢复所有文件，当每个文件被恢复时交互式地询问是否重命名文件，如果输入一个空值则文件不会被恢复。

在 cpio 恢复时不把设备名作为参数，设备名通过 stdio 提供给 cpio。使用 dd 或者 cat 让 cpio 从管道中读取数据。

```
#cpio –option<device 或
#dd if=device bs=blocksize | cpio –option
```

例 使用 B 为 5 120 字节大小读取 cpio 卷，遇到回去跳过，用 ASCII 格式读区首部，详细输出，列出文件表：

```
$cpio –iBcktv < device
```

进行完整文件恢复，要求在需要的地方创建目录，恢复原始修改时间，无条件地覆盖文件，列出所恢复文件的名字：

```
cpio –iBcdkmuv<device
```

恢复匹配模式为 hom*的文件：

```
cpio –iBcdkmuv "hom*"
```

恢复匹配模式为 hom*之外的所有文件：

```
cpio –iBcdfkmuv "hom*"
```

交互式重命名恢复：

```
$cpio –iBcdkmrnv < device
```

如果用相对路径创建的备份卷，则恢复时只需进入到要恢复的目录下即可。

使用 cpio 也可以实现目录拷贝功能，但首先要进入到需要拷贝的目录：

```
$cd old-directory
$find . –print | cpio –padlmuv new-directory
```

将所有后缀为.c 的文件备份到软盘上：

```
#find / -name *.c –print | cpio –ocv>/dev/fd0
```


将/usr/tony 目录下的文件备份到软盘上：

```
#ls /usr/tony | cpio –ocv>/dev/fd0
```

将/usr/tony 目录下的文件归档到/usr/mikey/testfile 中：

#ls /usr/tony | cpio –ocv>/usr/mikey/testfile

显示软盘上由 cpio 命令备份的文件列表：

#cpio –icvt –I /dev/fd0

恢复软盘上由 cpio 命令备份的文件：

#cpio –icvmB –I /dev/fd0

5. tar

tar 命令与 cpio 命令相同，也是以拷贝方式完成文件的备份和恢复。存储文件可以是磁盘、磁带或文件。tar 是目前使用最多的备份及拷贝工具。

6. tar 备份

命令格式：tar cvfb [pattern]

其中选项：

c：表示要创建备份；

v：表示要详细的信息输出，如列出详细的文件名及文件大小；

b：块因子；

f：通常情况下 tar 的默认输出是磁带设备，用 f 选项表示要指定输出设备。输出设备可以是磁盘上的一个文件、光盘、磁带驱动器或者标准输出，也可以是远程系统及其上设备；

pattern：匹配模式。

7. tar 恢复

由于 tar 的备份容易读取，tar 恢复也比较简单。

命令格式：tar mopxvf device pattern

其中选项：

x：表示作恢复；

m、v、f、device、pattern：与备份相同；

o：表示要将恢复的文件的所有者改为正在作恢复操作的用户；

p：默认情况下恢复时不会恢复所有文件的属性，该选项表示要恢复文件的所有属性。

例　备份目录/home 到文件 file.tar。

#tar cvf file.tar /home

恢复文件 file.tar 到目录/home 下：

#tar xvf file.tar /home

将/usr/tony 目录下的所有文件和子目录备份到默认设备中：

#tar cv　/usr/tony

显示默认设备中备份的文件列表：

#tar tv

将默认设备中备份的文件恢复到当前目录下：

#tar xv

显示所有备份的设备列表和设备编号：

#tar

如果不知道需要恢复的文件名而要完成部分恢复，可以创建一个内容表并从中查找需要的文件：

#tar tf device>filetable　（创建备份的内容表）

#grep resfile filetable　（在内容表中查找要恢复的文件 resfile）

#tar xvf device home /resfile　（恢复 resfile 文件到目录 home）

8. dd

dd 是一个将文件或者原始数据以拷贝方式进行备份的命令。通常情况下 dd 命令在数据库备份上较有优势。与前面的 dump、cpio、tar 备份命令不同，dd 通过修改输入输出块的大小，压缩输入到管道的文件，完成管道输出数据流，实现在传输中把拷贝内容从一种格式转化为另一种格式。所以，使用固定块大小的卷，可以用 dd 读取并将结果通过管道输出到 restore 恢复。

dd 数据流操作正是 UNIX 文件数据流结构思想的体现，借助管道 dd 能够完成标准输入到标准输入，实现数据流从一个命令到另一个命令，从一个系统到另一个系统。而正是 UNIX 操作系统中提供的 rsh 和 ssh 使得从一个系统到另一个系统的 dd 操作得以实现。

9. 对远程的操作

把备份写到远程设备上实现的是从一个系统到另一个系统。

创建一个嵌有 rsh 和 dd 命令的子 Shell，将本地备份命令通过管道输出到该子 Shell：

#tar cvf .-| (rsh remote_system dd of=device obs=block_size)

在远程设备上运行备份：

通过将 dd 命令 rsh 到远程系统上，再在本地系统上读取数据流实现远程设备上的备份：

```
#rsh remote_host "dd if=device bs=blocksize"| tar xvBf –
```

在用 dd 命令时注意块大小必须明确指定。如果创建的卷要被 tar 读取，则使用 tar 能够认识的块大小（如 10 240）。

读取远程主机上的磁带：

```
#ssh remote_host "dd if=device bs=blocksize"| tar xvBf –
#ssh remote_host "dd if=device bs=blocksize"| restore rvf –
#ssh remote_host "dd if=device bs=blocksize"| cpio –itv
```

在远程主机的磁带上备份：

```
#dump 0bdsf 64 100000 - | ssh remote_host "dd if=device bs=64K"
#tar cvf - | ssh remote_host "dd if=device bs=10K"
#cpio –oacvB |ssh remote_host "dd if=device bs=5K"
```

10. 使用 dd 确定磁带块的大小

用 dd 读取一个数据块并写到磁盘上，从磁盘上该数据块的大小可知磁带块大小。下例命令 dd 使用 256 K 作为块大小读取数据，直到遇到第一个记录间隔为止。如果实际块大小小于 256 K，则会在第一个记录间隔停止；否则修改块大小继续试。

```
#dd if=device bs=256K of=/tmp/mokey count=1
```

2.7　本 章 小 结

文件系统是 UNIX 操作系统中最重要的组成部分之一，文件系统的管理也是 UNIX 操作系统的日常管理工作之一，因此要对文件系统有较深入的认识，掌握文件系统的构造、结构、类型，同时也要能熟练应用与其相关的管理命令。

UNIX 文件系统是一个安全的文件系统。系统中的文件具有访问权限。对文件拥有者、访问权限的管理命令在文件管理中非常重要。

UNIX 文件系统的最大特点是引入了索引节点。索引节点的引入可以加快查找文件的速度及实现文件的共享。

文件系统的备份和恢复是作为高级系统管理员必须具备并熟练掌握的知识。常用的备份和恢复工具中应用最多的是 tar 和 cpio，它们使用非常方便。

上 机 练 习

1. 熟练掌握 ls 的用法，熟悉文件和目录的各种权限，并尝试修改权限。
2. 练习为系统添加一个交换空间，熟悉/etc/vfstab 的内容。

3. 练习加载和卸载文件系统，并使用 du、df 查看磁盘使用情况。

4. 尝试为用户配置配额。

5. 熟悉各种备份和恢复方法。

6. 设计一个下列思想的实现方案：为了减少查找文件名时搜索目录的次数，将常用文件名用高速缓冲存储起来。

7. 在超级块空闲表的链接表上为何不能存储太多的空闲块号？链接表上一个磁盘块应存储多少个空闲块号比较合适？

8. 讨论用位图而不用块的链接表来记录空闲磁盘块的系统实现，并指出这一方案的优点和缺点。

习 题 二

1. 说出 UNIX 文件系统组成。

2. 说出 UNIX 能够识别的文件类型。

3. 说出 UNIX 系统的目录层次结构。

4. 什么是根目录？什么是用户根目录？

5. 什么是索引节点？文件名和索引节点的关系如何？它们之间是一对一还是一对多关系？

6. 什么是启动块、超级块、索引节点块和数据块？

7. 什么是文件系统的备份、恢复？什么是增量备份？

8. 分别将权限--x--xrwx，r-xrwx-wx，rwxr-x-wx 转化为 8 进制代码。

9. 在自己的用户根目录下创建一个目录并检查该目录的默认权限。

10. 当安装一个文件系统时，内核调用驱动程序的打开过程，但在系统调用结束时释放设备特殊文件的索引节点。当卸载一个文件系统时，内核存取设备特殊文件的索引节点，调用驱动程序关闭过程，然后释放索引节点。试对索引节点操作及驱动程序打开和关闭的顺序与打开和关闭一个块设备的顺序进行比较和说明。

第 3 章　UNIX 系统管理

UNIX 是多用户操作系统，普通用户使用操作系统，而系统管理员则完成操作系统的管理和维护工作，即系统管理。

系统管理包括：系统运行级别管理、用户及组管理、文件系统管理、进程管理、设备管理、网络管理、任务自动调度管理、性能管理。由于文件系统管理、进程管理、设备管理、网络管理是系统管理中较为复杂的工作，其内容独立成章，本章主要介绍这 4 大管理之外的系统管理。

本章主要内容

- 系统引导、运行与系统关闭：系统运行级选择、系统关闭命令、系统的重新引导。
- 用户及组管理命令：用户管理文件、添加和删除用户及用户组。
- 系统管理员与用户通信：系统管理员管理消息发送、系统用户之间的通信。
- Solaris 系统管理工具 Admintool：管理工具 Admintool 的功能及使用。
- AIX 系统管理工具 SMIT：管理工具 SMIT 的功能及使用。
- 任务自动调度：任务自动调度工具 cron 与 at 的使用。
- 性能管理：系统性能、收集系统性能统计信息、性能管理命令。

3.1　系统引导、运行与系统关闭

3.1.1　系统引导

系统引导是将 UNIX 系统主机开机引导，从磁盘中调度 UNIX 系统核心程序并装入内存执行的过程。

通常有三种系统引导方式。

- 本地硬盘引导：以本地安装的硬盘上的引导区为标准启动机器，加载引导区并将控制权交给 UNIX 系统内核，开始系统运行。UNIX 操作系统所在的硬盘是根盘。

- 无盘网络引导：无盘工作站上没有安装根盘，通过远程网络上的文件服务

器提供引导所需要的文件和程序，然后将这些文件存放在无盘工作站的 RAM 中运行。

· 单用户/维护引导：系统引导后进入一种单用户方式，此时其他用户无法登录到系统，系统只接受超级用户的登录和命令请求，这种方式是系统管理员对系统进行维护和管理，也称维护模式。在该模式下可以执行安装或更新软件，也可以运行检查系统的诊断程序。

系统引导是一个比较复杂的工作过程，主要由以下步骤构成。

(1) 打开电源、进入 PROM 启动过程：打开计算机时，计算机上用来确认功能部件的内存 PROM 具有厂家给定的代码。

(2) PROM 自检后装入系统引导块：PROM 中的代码用于系统自身硬件验证，确认安装在系统中的硬件设备与代码内容相匹配后，寻找可引导的介质，初始化引导程序并将控制权交给引导程序。

(3) 引导程序引导操作系统：从硬盘加载核心程序到内存并将发现的硬件清单交给系统内核。

(4) 完成操作系统初始化：引导程序将控制权交给内核，内核确认设备是否能正常工作并创建进程 sched（进程标识符 pid 为 0 的进程）。由 sched 进程创建子进程 init（pid 为 1），init 进程执行初始化任务，读系统/etc/inittab 文件，启动系统的管理进程系列（驻守进程 daemon）：inetd、rlogind、lpsched、nfsd、mountd 等。

(5) 内核完成所有进程的启动：内核完成检查后开始按照系统配置文件和运行级启动系统服务。为登录用户授予访问权限。

3.1.2　系统运行级

1. 运行级

系统运行级是系统运行时所处的一种状态，不同的运行级在用户登录及使用上有一定的限制。通常 UNIX 有 10 个运行级，如表 3.1 所示。

表 3.1　系统运行级定义

运行级	描述
0	关电状态。
1 或 S	单用户状态（系统管理员状态）。
2	多用户状态（不输出资源），用于隔离标准操作，非服务系统。
3	多用户状态（输出资源），用于远程文件共享，为网络服务系统的标准操作系统。
4	目前为使用的运行级，管理员可以定义多用户系统状态，可以在本地设置和定义。

（续表）

运行级	描述
5	软件重启动状态。用于可维护的活动类型及运行诊断程序，也可以从另一个替换磁盘引导。
6	关闭重新启动。用于拆卸系统立即重新引导到标准状态。
A b c	当 init 命令请求改变运行状态 a、b 或 c 时，不杀死当前运行级上运行的进程。只在启动新进程时为其赋予新状态。
Q 或 q	指示 init daemon 进程重读并执行 inittab 文件。

通常情况下，系统运行在多用户输出资源运行级，运行级为 3。有时为了文件安全会选择不输出资源的运行级 2，这时系统不提供任何网络服务。在系统备份或系统出现问题时的修复会选择运行级 1。有的 UNIX 版本将运行级 4 定义为厂家维护模式。

2. inittab 文件

系统的运行级定义及管理文件为/etc/inittab 文件。该文件中有系统的默认运行级及各运行级别定义。

Solaris 系统中/etc/inittab 文件实例为：

```
is:3:initdefault:                #default run level 3
p3:s1234:powerfail:/usr/sbin/shutdown –y –i5 –g0 >/dev/console 2>&1
                                 #powerfailure
shutdown to                      #maintenance run
→level 5
s0:0:wait:/sbin/rc0 >/dev/console 2>&1 </dev/console      #execute during run
→level 0
s1:1:wait:/usr/sbin/shutdown –y –iS –g0 >/dev/console 2>&1 </dev/console
        #when requested run
→level 1
                              #reboot into single
→user mode
s2:23:wait:/sbin/rc2 >/dev/console 2>&1 </dev/console    #execute during run
→level 2 & 3
s3:3:wait:/sbin/rc2 >/dev/console 2>&1 </dev/console     #execute during run
→level 3
```

```
s5:5:wait:/sbin/rc5 >/dev/console 2>&1 </dev/console    #execute during run
→level 5
s6:6:wait:/sbin/rc6 >/dev/console 2>&1 </dev/console    #execute during run
→level 6
sc:234:respawn:/usr/lib/saf/sac –t 300                   #execute during run
→accounting ****
c0:234:respawn:/usr/lib/saf/ttymon –g –h –p " 'uname –n'console login: "-T sun
–d /dev/console –l console –m ldterm,ttcompat            #start the console
→port
```

该文件的前两行标识了在系统 init 进程进入运行级前运行的脚本。之后 4 行用于控制运行级定义。

init 文件行格式为：identifier:Run Level:Action:Command

identifier 为标识符；Run Level 为运行级。

Action 为 init 命令应向进程实施的动作。具体的动作有：

- respawn 表示启动进程并在其死后重启该进程；
- wait 表示启动进程，在继续到下一项取得运行状态之前，等待进程结束；
- once 表示若进程没有执行则启动它，但不等它结束就继续下一项；
- boot 表示只在引导时执行该项但不等待其完成；
- bootwait 表示只在引导时执行该项并等待其完成；
- powerfail 表示当系统收到 powerfailure 信号时执行；
- powerwait 表示当系统收到 powerfailure 信号时执行并直到执行结束；
- off 表示若与该项相关的进程正在执行则杀死该进程，也用于注释不用的终止行；
- ondemand 只用于 a、b、c 运行级，与 respawn 类似；
- initdefault 用于指出默认运行级，必须处于文件的第一行，且只列出一个运行级；
- sysinit 用于激活在试图访问系统控制台时需要完成的进程。

Command 表示要执行的 Shell 命令。

通常使用 vi 编辑器打开 inittab 文件，也有用于更改该文件的程序，或用命令修改该文件记录

比如在 AIX 系统中用如下命令：

- chitab 命令修改 inittab 文件的记录；
- lsitab 命令列出 inittab 文件的记录；
- mkitab 命令在 inittab 文件中增加记录；

· rmitab 命令在 inittab 文件中删除记录。

3. 选择运行级

/sbin/init 命令与/etc/telinit 命令

使用/sbin/init 命令或/etc/telinit 命令可以不用修改/etc/inittab 文件中的默认运行级定义而只需重新引导系统，就能改变正在运行系统的运行级。

当/sbin/init 命令或/etc/telinit 命令赋予系统一个新的运行级时，则读取/etc/inittab 文件并为该特定运行级执行所有的运行控制脚本。但如果/etc/inittab 文件不存在，系统就只能引导成单用户模式。

/sbin/init 命令格式为：/sbin/init n

/etc/telinit 命令格式为：/etc/telinit n

n 为运行级。

4. 查看系统的运行级

用命令 who –r 可以查找系统的运行级，同时系统还会显示上次改变运行级的日期和时间。

```
$who –r
run level 3 Sep 8 13:35 3 0S
$
```

运行控制脚本

当 init daemon 激活并读取/etc/inittab 文件时，执行运行控制脚本，这些控制脚本在目录/etc 或/sbin 中，称为 RC Shell 脚本。运行控制脚本在引导进程期间执行，调用方式即 rc 之后的数字就是运行级。运行控制脚本的主要功能是执行用于为被选运行级启动服务的脚本。这些脚本通常放在/etc/rc#.d 目录结构中，其中#代表运行级。

在系统引导时 rc 脚本所作的工作有：

· 设置计算机主机的名字和 IP 地址；
· 配置网络接口；
· 检查文件系统；
· 安装文件系统；
· 从/tmp 目录中删除文件；
· 设置区域时间；
· 启动 daemon；
· 启动网络服务；

· 启动应用。

5. 运行控制目录

运行控制脚本保存在/etc 目录中，每个运行的脚本保存在特殊的目录/etc/rc#/rc#.d 下。这些脚本用于开始或停止属于该运行级的服务。如 init daemon 运行/etc/rc3 运行控制脚本时，rc3 只执行/etc/rc3.d 目录中的文件。每个目录包含所有与该运行级相关的启动和关闭脚本。rc 脚本的设计目的是传递参数 start 和 stop。当调用一个带 start 参数的脚本时，该脚本以提问方式启动服务；当调用一个带 stop 参数的脚本时，该脚本杀死服务进程。另外还可以编写函数，在引导或关闭过程序列中指定具体开始或停止服务的位置。

目前有 7 个运行控制目录可供系统选择：

(1) /etc/rc0.d 用于停止系统的脚本；

(2) /etc/rc1.d 用于单用户或维护方式的脚本；

(3) /etc/rc2.d 进入多用户模式的脚本；

(3) /etc/rc3.d 用于启动网络文件共享服务的多用户模式的脚本；

(4) /etc/rc4.d 用于厂商提供的系统维护模式的脚本；

(6) /etc/rc5.d 由用户定义，系统未用；

(7) /etc/rc6.d 用于重引导系统的脚本。

6. 启动和杀死脚本

下例是一个/etc/rc2.d 目录的清单。

K12acct

K14lp

K25xntp

K30nfs.server

K35nfs.client

K55snmpd

K76volmgt

S03MOUNTFSYS

S05RMTMPFILES

S19perf

S21syssetup

S22acct

S25sysid.net

S49inet

S66uucp

S66sysid.sys

S70nfs.server

S73cron

S80sendmail

S90syslog

S90volmgt

S93SUNWmg.sync

如果该目录中有 README 文件，则在该文件中包含了选择 S 或 K 脚本顺序号的原则，否则按目录中的顺序 K 脚本先执行，S 脚本后执行，如果脚本号码相同，则按大写在前小写在后的顺序执行。

7. /etc/init.d 目录

该目录包含操作系统为进程提供的初始化和终止脚本。在系统中可以使用这些脚本为进程传递 start 或 stop 参数。

下面是 Solaris 系统中/etc/init.d 目录的一个实例。

ANNOUNCE	autoinstall	inetsvc	rpc	syslog.orig
MOUNTFSYS	buildmnttab	lp	sendmail	ufs_quota
Cron	standardmounts	README	devvlinks	syssetup
volmgt	drvconfig	nfs.client	nfs.server	sysid.net
sysid.sys	acct	inetinit	rootuser	syslog
perf	audit			

在/etc/init.d 目录中每个脚本都有一条 case 语句接收 start 或者 stop 变量。例如当需要执行文件/etc/init.d/nfs.server start 时，case 语句指向/etc/rc2.d/S70nfs.server。

在 AIX 系统中没有该目录，AIX 将所有的运行控制脚本放入/etc 目录中并以 rc 开头，AIX 系统使用命令 startsrc 和 stopsrc 打开和关闭子系统进程。

HP 系统中也没有/etc/init.d 目录。所有控制运行的脚本置于/etc/rc.config.d 目录下。使用方式与/etc/init.d 相同。另外还可以手工调用要激活的脚本并按需要传送 start 和 stop 参数。

8. 系统引导操作

打开 UNIX 主机电源且硬件设备自检通过后出现 ok 计算机固化程序提示符 ok PROM，这时应该开始引导系统进入不同运行级：

进入多用户状态，在提示符 ok PROM 后输入 boot 并回车。

进入单用户状态，在提示符 ok PROM 后输入 boot -s 并回车。

当需要对系统文件或内核作临时更改时，使用交互式引导，在提示符 ok PROM 后输入 boot -a 并回车，进入交互式引导模式。

有时需要忽略引导进程，有时不能出现 ok PROM，这需要使用特定功能健 Stop-A。

有时系统进入监控程序，出现监控提示符>，则输入 n 得到 ok 提示符。

系统引导信息存放在文件/var/adm/messages 中，要查看引导消息可以用命令：

/usr/sbin/dmesg 或 more /var/adm/messages

建议不要直接打开文件/var/adm/messages。

3.1.3 系统关闭

1. 系统关机

UNIX 不允许直接关掉电源，必须执行关机命令，在系统完成了关闭操作之后才可以断电。只有超级用户才有权执行关机命令。

执行关闭系统操作有三种情形：关掉电源、关机、重新引导。

通常在以下情况下需要关闭系统或重新引导：

- 关掉电源；
- 添加系统硬件；
- 安装新软件后；
- 修改软件的配置后；
- 系统处于无法返回的挂起状态；
- 系统性能严重下降；
- 文件系统可能遭到破坏；
- 关闭系统的命令有 shutdown、halt、reboot、init、telinit。

2. shutdown 命令

使用 shutdown 命令关闭操作系统。该命令向所有的系统用户发出关闭系统的通知，默认情况下等待 60 s 后关闭系统。

shutdown 命令格式为：

shutdown [-f file | mesg] [-g time] [-i init-level] [-y]

其中：

file 文件中包含有 shutdown 第一步中系统管理员发给所有终端用户的消息；

time 是 shutdown 等待关机的时间；

init-level 是 shutdown 将系统转入的运行级别。默认情况下转入 0 级；

y 表示 shutdown 对所有的交互问题均以 yes 回答。

在不同系统中 shutdown 命令路径不同。

Solaris 系统：/usr/sbin/shutdown

SunOS 系统：/etc/shutdown

HP/UX 系统：/etc/shutdown

AIX 系统：/usr/sbin/shutdown

例　在 Solaris 系统中 shutdown 在 200 s 内关机，命令格式为：

#/usr/sbin/shutdown –g200 –i0

3. halt 命令

halt 命令用于立即关机，不会给系统用户发出关机通知。halt 命令不严格执行 rc 关闭脚本，不是最佳的系统关闭方法。

命令格式：#halt

4. reboot 命令

使用 reboot 关闭系统并重新引导系统。该命令不发送关闭系统的通知给系统用户，不严格执行 rc 脚本，也不是最佳的关机命令。

命令格式：#reboot

如果系统添加了新的软、硬件就需要关机重新引导，用命令：reboot –r

该命令表示运行一个重新配置的脚本，装载模块目录中列出的所有设备驱动程序并创建相应的硬件节点。

5. /sbin/init 命令与/etc/telinit 命令

前面已经介绍过 init 和 telinit 命令，它们都用来修改运行级，当运行级为 0 时则关闭电源。这两个命令可以互换，但 telinit 命令使用 rc 脚本删除进程，能可靠地关闭系统。

3.2　用户及组管理命令

用户和组管理是系统管理的主要职责。系统管理员通过对用户的管理控制对 UNIX 系统的访问。

1. 用户管理信息

在 UNIX 系统中用户的管理信息有：

- 用户帐号（用户名）及用户标识符（uid）；
- 用户组帐号及组标识符（gid）；
- 用户注释；
- 用户默认 Shell；
- 用户主目录及主目录权限；
- 用户密码及有效期。

2. 用户管理文件

在 UNIX 系统中常用的用户管理文件有：

- /etc/passwd 文件；
- /etc/shadow 文件；
- /etc/security/shadow 文件；
- /etc/group 文件。

这些文件只有系统管理员才有权限修改。

3.2.1 用户管理文件

1. etc/passwd 文件

/etc/passwd 文件是系统用户认证访问权限的第一个文件。该文件中的行格式如下：

login_name:password:uid:gid:user info:home_directory:default_shell

其中：

login_name 为用户帐户，可以用 1~8 个字符表示，区分大小写，避免使用数字开头。

password 为加密后的用户登录系统的密码。

uid 为系统指定给每个用户的惟一数值，即用户标识符（操作系统进程管理时用此数值作为拥有该进程的用户标识），32 位系统中标识符在 0~60 000 之间。

gid 为用户组标识。

user info 为用户注释信息（如用户身份、电话号码、特性等）。

home_directory 为用户的主目录（家目录）。

default_shell 为用户登录系统时默认执行的 Shell 程序，通常为/bin/sh，表示执行 Bourne Shell。如果是 Korn Shell 则用/usr/bin/ksh。如果是 C Shell 则用/usr/bin/csh。

如果是 TC Shell 则用/usr/bin/tcsh（较少使用）。如果是 Bash Shell 则用/usr/bin/bash（Linux 系统）。

例　/etc/passwd 文件实例。

```
$cat /etc/passwd
root:!:0:0::/:/usr/bin/sh
daemon:!:1:1::/etc:
bin:!:2:2::/bin:
sys:!:3:3::/usr/sys:
adm:!:4:4::/var/adm:
uucp:!:5:5::/usr/lib/uucp:
blademan:!:246:1::/home/blademan:/usr/bin/sh
lmy:!:249:1::/home/lmy:/usr/bin/sh
xwen:!:251:1::/home/xwen:/usr/bin/sh
imnadm:*:254:201::/home/imnadm:/usr/bin/sh
ldap:*:255:1::/home/ldap:/usr/bin/sh
duj:!:415:1::/home/duj:/usr/bin/sh
……
$
```

2. /etc/shadow 文件

在 Solaris 系统中用/etc/shadow 文件管理用户密码，该文件中的行与/etc/passwd 文件中的行完全对应。该文件主要用于系统管理员修改或取消用户密码。

/etc/shadow 文件中行的格式为：

login_name:encrypted_password:last_change:min:max:warn:inaction:expire:reserved

其中：

login_name 为登录用户帐号。

encrypted_password 为加密后的用户密码。

last_change 为口令上次被修改的日期距离 1970.1.1 的天数。

min 为要求口令被更改的最小间隔天数。

max 为口令有效的最大天数。

warn 为口令到期前多少天发出警告。该警告在用户登录时发出。

inaction 为至登录前某个帐号不活跃的天数。

expire 为用户禁止登录并且其帐号不可用的绝对日期。

reserved 为保留区域，空白。

例 /etc/shadow 文件实例。

```
$cat /etc/shadow
root:ASHFNadn:10390:0:0::::
daemon:NP:6645::::::
bin:NP:6645::::::
sys: NP:6645::::::
adm:NP:6645::::::
uucp: NP:6645::::::
blademan:NP:6645::::::
lmy: NP:6645::::::
xwenNP:6645:::::::
imnadm:*ASK*:9807:76:23:12:40::
ldap:*kfjkj:10255:2:45:2:::
duj:MKHS*:12415:34:56::::
……
$
```

3. /etc/security/passwd 文件

AIX 系统与 Solaris 系统不同，没有/etc/shadow 文件管理用户密码，而用/etc/security/passwd 文件管理用户密码。

/etc/security/passwd 文件实例：

```
wenxie:
password = usjflgk2gk5mrn
lastupdate = 901274567
flags = ADMIN,NOCHECK
jiexich:
password = Ndkfj6k63vus
lastupdate = 901274567
flags =
```

其中：

password 表示加密后的用户密码。

lastupdate 表示口令最近改变时间距离 1970.1.1 的秒数。

flags 表示用户使用 login、passwd 和 su 命令的约束。约束可能是一个用逗号分

开的清单，如 ADMIN,ADMCHG,NOCHECK 或 blank（默认）。

4. /etc/group 文件

文件/etc/group 管理用户组。文件中的行格式为：

groupname:password:gid:user-list

其中：

groupname 为用户组名。

password 为组密码。

gid 为组的标识符。

user-list 为同组用户清单。

/etc/group 文件实例：

```
root::0:root0
other::1
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
uucp::5:root,uucp
mail::6:root
tty::7:root,tty,adm
lp::8:root,lp,adm
nuucp::9:root,nuucp
staff::10:
daemon::11:root,daemon
sysadmin::12:
operator::1000:teacher,professor
admin::2000:stu01,stu02,stu03
nobody::60001:
noaccess::60002:
$
```

5. /etc/security/group 文件

在 AIX 系统中除了用/etc/group 文件作为组基本管理之外，还用/etc/security/group 文件配合管理组。/etc/security/group 文件实例如下：


```
sys:
```

```
admin = true
adm:
admin = true
adms = ppadmn,ftp02
dmdusers
admin = false
```

其中：

admin 的可能值为：true 表示将该组设为系统管理组，只有超级用户可以更改它；false 表示定义标准用户组，该组可被超级用户和该组所列出的其他用户更改。默认值为 false。

adms 表示该组为可执行系统管理任务的用户。

3.2.2 用户管理命令

UNIX 系统提供命令管理用户。

1. 添加用户命令

使用 useradd 命令可在系统用户管理文件中增加一个新的用户条目。该命令也可用于增加一个用户到组中。命令格式：

useradd [-c comment] [-d dir] [-e expire] [-f inactive] [-g group] [-G group [, group…]] [-m [-k skeldir]] [-u uid [-o]] [-s shell] login

其中：

-c comment 表示用户的注释。

-d dir 表示用户的主目录（用户根目录）。通常情况下，如果系统默认用户主目录为 /export/home，新增用户的帐号为 zijich，则新增用户的主目录为 /export/home/zijich。

-e expire 指定登录的终止日期，该日期之后用户禁止登录。

-f inactive 表示在该用户帐号宣布无效之前允许该帐号登录的最大天数，通常是一个正整数。如果为 0 表示该值无效。

-g group 表示一个已存在的组标识符，表示新增用户要加入的主要组。

-G group 表示一个已存在的组标识符，表示新增用户要加入的附加组。

-m：如果新增用户的主目录不存在则创建一个用户主目录，如果已经存在则新增加的用户对其有读、写和执行权限。

-k skeldir 是一个包含有被复制到新增用户起始目录中的框架信息（如.profile 文件）的目录，必须是已经存在的目录。

-u userid 表示新增用户的标识符（uid），是 0~65 535 之间的一个整数。UNIX 系统为了更方便地管理用户，除用户名之外还给每个用户增加了用户标识符，每个用户都有一个惟一的标识符。有时将用户帐号称为系统用户的外标识符，而将 uid 称为系统用户的内标识符。

-s shell 表示新增用户登录时默认的 Shell。

login 表示新增用户的帐号（用户名）。

例　#useradd –c "Zhang Li Ming 87653421"-d /export/home/zhanglm –e 089823 –f 300 –g admin –m –k /usr/local/bin/skel –u 1001 –s /bin/csh zhanglm

如果新增用户在系统中已经存在，则会出现报错信息。

在 AIX 系统中该命令名为 mkuser。

例　#mkuser home = /home/zhanglm shell = /usr/bin/csh group = staff, printq su = false zhanglm

2. 删除用户命令

当一个用户不再访问系统时，应该删除该用户帐号（如果用户较长一段时间不用，可采用将用户帐号修改为不能登录方式较好）。删除用户的同时也将用户的所有相关信息消去。命令格式：

userdel [-r] login

其中-r 表示删除用户主目录及所有的子目录。

在 AIX 系统中命令格式为：

rmuser [-p] login

其中-p 表示删除/etc/security/passwd 文件中的密码信息。

例　在 Solaris 系统中删除用户 zhanglm。

#userdel –r zhanglm

在 AIX 系统中删除用户 zhanglm：rmuser –p zhanglm

用该命令进行 Shell 编程时需要用该命令的返回信息以便确认命令执行情况。返回信息有如下几种情况：

0：命令成功。

2：命令语法无效，显示 uesrdel 命令常用的信息。

6：删除的用户登录名不存在。

8：删除的用户登录名正在使用。

10：删除命令不能更新文件/etc/group，但已经从/etc/passwd 文件中删除了用户登录名。

12：不能删除或改变起始目录。

3. 修改用户命令

利用命令 usermod 可以修改用户登录所需的所有系统文件信息。该命令格式为：

usermod [-u uid] [-g group] [-G group [,group…]] [-d dir[-m]] [-s shell] [-c comment] [-l new_logname] [-f inactive] [-e expire] login

在 Shell 编程中该命令返回值为：

2：无效的命令语法。

3：选项中存在无效的参数。

4：-u 选项中指定的 uid 正在使用。

5：口令文件中存在错误。

6：欲改变的登录名或组登录的 Shell 不存在。

8：欲改变的登录名正在使用。

9：新的登录名已被使用。

10：不能更新/etc/group 文件，将执行其他的更新请求。

11：没有足够的空间用于移动起始目录，但仍执行其他更新。

12：不能完成将旧起始目录移向新的起始目录。

在 AIX 系统中命令名为 chuser，工作过程类似。

4. 组命令

(1) 创建组

命令格式：groupadd [-g gid] group

在 AIX 系统中命令格式为：mkgroup [-a] [-A] group

系统对创建组命令的返回值为：

0：命令成功。

2：命令语法无效。

3：选项中存在无效的参数。

4：gid 不惟一。

9：组不惟一。

10：无法更新/etc/group 文件。

(2) 修改组

通过命令 groupmod 修改给定组在文件/etc/group 中的信息。

命令格式：groupmod [-g gid] [-n newname] group

该命令的返回值为：

0：命令成功。

2：命令语法无效。

3：选项中存在无效参数。

4：gid 号不惟一。

6：欲改变的组不存在。

9：为组指定的新名字已经存在。

10：不能更新/etc/group 文件。

在 AIX 系统中使用命令 chgroup 修改组。

(3) 删除组

用命令 groupdel 删除组以及/etc/group 系统文件中的相关信息。

命令格式：groupdel group

在 AIX 系统中命令格式为：rmgroup name

groupdel 命令的返回值为：

0：命令成功。

2：命令语法无效。

6：欲删除的组不存在。

10：不能更新/etc/group 文件。

(4) 查看用户属于哪个组

如果用户要查看自己属于哪个组，可以用命令：groups

```
$groups
staff
$
```

3.3　系统管理员与用户通信

当系统管理员完成管理任务时，如系统安装新软件、更改系统环境、系统重新引导等，可能影响系统用户，这时系统管理员就需要将一些通知及时发送给用户。所以，系统管理员与用户之间的通信也是系统管理的一部分。

3.3.1　系统管理员通知本机用户

系统文件/etc/motd 中的内容可以在用户登录系统时显示。系统管理员对该文件有写权限。文件中的内容应该及时更新，短小精练，用户登录系统时才会注意。

3.3.2　发送消息到系统的单个用户

利用 write 命令将消息发送给正在使用系统的用户。在用户登录窗口中会显示

消息。如果用户打开的窗口有多个时，就在控制窗口中显示消息。除了系统管理员可以发送消息给指定用户之外，普通用户也可以利用该命令发送消息到指定用户。实际上，如果发送消息和接收消息的双方同时打开 write 信道，该应用实现的就是双向对话。

1. 直接将消息发送给单个用户

首先由发送方输入：write username（username 是接收消息的用户帐号）并回车；接着输入要发送的消息，发送完后按 Ctrl+D 结束。

接收方的屏幕上则显示相应消息。

例 用户 wangxh 发送消息给用户 zhanglm。

```
$write zhanglm
This evening I'll go to Shanghai by plane.
Please send mail to me.
By!
Ctrl+D
$
```

在用户 zhanglm 的窗口中显示消息：

```
Message from wangxh on ttyp20 at 14:00…
This evening I'll go to Shanghai by plane.
Please send mail to me.
By!
EOF
```

2. 通过文件将消息发送给单个用户

如果要发送的消息较长，则可将消息写到一个文件中再发送。

```
write username<filename
```

例 用户 wangxh 发送消息给用户 zhanglm。

```
$cat>sendfile
This evening I'll go to Shanghai by plane.
Please send mail to me.
By!
$write zhanglm<sendfile
$
```


在用户 zhanglm 的窗口中显示消息：

Message from wangxh on ttyp20 at 14:00…

This evening I'll go to Shanghai by plane.

Please send mail to me.

By!

EOF

3.3.3　发送消息到系统或网络中的所有用户

1. 使用 wall 命令同时发送消息给系统的所有用户

步骤是：

首先输入 wall 并回车；接着输入要发送的消息；写完后按 Ctrl+D 结束。

例　$wall

This evening wangxh will go to Shanghai by plane.

Please send mail to him

By!

Ctrl+D

$

在用户的窗口中显示消息：

Broadcast message from wangxh on ttyp20 at 14:00…

This evening wangxh will go to Shanghai by plane.

Please send mail to him.

By!

EOF

2. 用 rwall 命令同时发送消息给其他系统的所有用户

步骤是：

首先输入 rwall hostname 并回车；接着输入要发送的消息；写完后按 Ctrl+D 结束。

例　wangxh 在主机 host01 中给主机 host11 中的用户发送消息。

$rwall host11

This evening wangxh will go to Shanghai by plane.

Please send mail to him

By!

Ctrl+D

```
$
```

在主机 host11 的用户窗口中显示消息：

```
Broadcast message from host01:wangxh at 14:00…
This evening wangxh will go to Shanghai by plane.
Please send mail to him.
By!
EOF
```

3.4 Solaris 系统管理工具 Admintool

3.4.1 Admintool 简介

Solaris 系统中提供了 Admintool 工具用于管理系统用户、用户组、打印机等。Admintool 是图形窗口，只有 Solaris 的图形终端和主控台才能使用。另外，只有系统管理组用户才有权运行该工具。

3.4.2 Admintool 使用

1. 启动 Admintool

在命令窗口中输入：#Admintool&并回车（表示后台启动 Admintool），显示 Admintool：Users 窗口。

2. 用户管理

· 添加用户帐户到系统

(1) 在 Browse 菜单中选择 Users 项。
(2) 在 Edit 菜单中选择 Add 项，出现 Admintool：Add User 窗口。
(3) 在用户窗口中输入相应信息：用户帐号、UID、主目录等。
(4) 单击 OK 按钮完成添加用户。

· 修改用户帐户信息

(1) 在 Browse 菜单中选择 Users 项。
(2) 在 Edit 菜单中选择 Modify 项，出现 Admintool：Modify User 窗口。
(3) 在窗口中修改相应的信息，完成后单击 OK 按钮。

· 删除用户帐户信息

(1) 在 Browse 菜单中选择 Users 项，从窗口中选中要删除的用户帐号。
(2) 在 Edit 菜单中选择 Delete 项，显示警告窗口。单击 Delete 按钮删除用户帐

户，如果要删除主目录，选中 Delete Home Directory，则用户主目录也被删除。

(3) 完成后单击 OK 按钮。

· 禁用用户帐户信息，最早的禁用方法是锁定用户密码

(1) 在 Browse 菜单中选择 Users 项。

(2) 在 Edit 菜单中选择 Modify 项，出现 Admintool：Modify User 窗口。

(3) 在窗口 Password 中选择 Account is Locked。

(4) 完成后单击 OK 按钮。

另外还可以通过设定密码有效期、登录帐户终止期、访问间隔期来控制帐户的访问。

3. 用户组管理

· 添加用户组到系统

(1) 在 Browse 菜单中选择 Groups 项。

(2) 在 Edit 菜单中选择 Add 项，出现 Admintool：Add Group 窗口。

(3) 在窗口中输入相应信息：用户组名、GID、成员等，成员之间用冒号隔开。

(4) 完成信息输入后单击 OK 按钮。

· 修改用户组信息

(1) 在 Browse 菜单中选择 Groups 项。

(2) 在 Edit 菜单中选择 Modify 项，出现 Admintool：Modify Group 窗口。

(3) 在窗口中修改相应的信息，完成后单击 OK 按钮。

· 删除用户组信息

(1) 在 Browse 菜单中选择 Users 项，从窗口中选中要删除的用户组。

(2) 在 Edit 菜单中选择 Delete 项，显示警告窗口。单击 Delete 按钮删除用户组。

(3) 完成后单击 OK 按钮。

4. 设置本地打印机

用 Admintool 可以设置对打印机的访问或配置本地打印机。

· 配置本地打印机的步骤

(1) 输入 Admintool&并回车。

(2) 在 Browse 菜单中选择 Printers 项，出现 Admintool：Printers 窗口。

(3) 在 Edit 菜单中选择 Add and Local Printer 项，出现 Admintool：Add Local Printer 窗口。

(4) 在窗口中输入打印机名和描述，选择打印机端口、类型、文件内容和故障通知。

(5) 如果要将打印机设置为默认打印机，选中 Default Printer 选项，如果每次要打印标题，选中 Always Print Banner。

(6) 如果需要限制用户访问，更改 User Access List 中的内容。

(7) 单击 OK 按钮完成打印机配置，信息添加到打印机窗口中。

· 设置网络打印机的步骤

(1) 输入 Admintool&并回车。

(2) 在 Browse 菜单中选择 Printers 项，出现 Admintool：Printers 窗口。

(3) 在 Edit 菜单中选择 Add and Access to Printer 项，出现 Admintool：Add Access to Printer 窗口。

(4) 在窗口中输入打印机名、打印服务器名和描述。

(5) 如果要将打印机设置为默认打印机，选中 Default Printer 选项。

(6) 单击 OK 按钮完成打印机配置。

3.5 AIX 系统管理工具 SMIT

3.5.1 SMIT 简介

在 AIX 系统中用管理工具 SMIT（System Management Interface Tool）提供多层次的系统管理。主要的管理功能有：

· 软件的安装和维护；
· 设备管理；
· 系统存储管理；
· 安全及用户管理；
· 通信应用和服务；
· 打印管理；
· 问题监测；
· 性能和资源分配；
· 系统环境；
· 进程管理；
· 应用管理。

3.5.2 SMIT 使用

在 SMIT 的使用上采用多级对话窗口方式。最上级窗口也称为主窗口，运行命令：smit 则出现主窗口，在主窗口中选中相应的管理功能则进入相应的功能管理子

窗口。这样可以逐级进入窗口。使用非常灵活、方便。如果不从主窗口进入，也可以通过命令直接进入子窗口。SMIT 的详细使用请参见“ AIX 系统管理 ”。

如果要进入用户管理功能，在主菜单上选中“ 安全及用户管理 ”进入“ 安全及用户管理 ”窗口，另外也可以输入命令：smit security 进入“ 安全及用户管理 ”窗口。该窗口中有 4 个选项：

- 用户：用于增加、修改、删除用户帐户；
- 组：用于增加、修改、删除用户组；
- 密码：用于设置和修改用户密码；
- 登录控制：用于限制计费或终端用户访问。

3.6　任务自动调度

系统管理员负责系统管理任务，为了维护系统的正常运行，需要做大量周期性日常维护工作，如日志系统维护、文件系统备份、重要文件及目录的清理等。UNIX 系统提供的任务自动调度功能为系统管理员完成这些周期性的工作带来极大方便，常用的完成自动调度功能的命令有 cron、batch、at。

3.6.1　周期性间隔时间调度命令 cron

UNIX 是多任务操作系统。利用系统提供的 cron 程序，系统可以按一定时间间隔自动完成多任务的调度。

cron 会选择系统比较空闲时去执行任务。cron 进程每分钟唤醒一次（cron 的最短时间间隔为 1 分钟），检查所有存储的 crontab 文件并根据系统忙闲情况决定是否在当前时间执行。

1. crontab 文件

crontab 文件分为系统 crontab 文件和用户 crontab 文件，这两类文件都由 crontab 工具创建和维护。

系统 crontab 文件包含系统周期性调度和与系统管理有关的命令。通常情况下，系统中该文件路径名为/etc/crontab 或/usr/lib/crontab。

在系统管理员限制下用户也可以使用 crontab 命令提交自己的 crontab 文件，用户 crontab 文件以用户帐号命名，位于目录/usr/spool/cron/crontabs 或/var/spool/cron 下。

系统管理员可以限制用户的 crontab 功能。不同系统限制和允许 cron 的文件目录不同，常见的文件路径有/usr/lib/cron/、/etc/、/usr/spool/cron/。如果 cron.allow 和 cron.deny 都存在，则在 cron.allow 中列出的用户可以使用 cron；如果 cron.allow 文

件不存在，则检查 cron.deny，不在该文件中的用户可以使用 cron，如果该文件内容为空，表示所有用户都可以使用 cron；如果 cron.allow 和 cron.deny 文件都不存在则只有超级用户才有权使用 cron。

在系统中配置的与 crontab 相关的文件和目录还有：

· /usr/lib/cron/queuedefs、/usr/cron.d/queuedefs：队列描述文件；

· /etc/default/cron：包含 cron 默认设置的文件；

· /usr/lib/cron/log、/var/cron/log、/var/log/cron：包含 cron 行为的日志文件；

· /usr/lib/cron/olog、/var/cron/olog：当日志文件大小超过系统 ulimit 时，则移向 olog。

crontab 文件内容形式如下：

```
#mm hh dd mm ww command    (说明行)
0   2   *   *   *   /etc/cron.d/logbak
0   3   *   *   2   /usr/lib/newsyslog
0   4   *   3   5   /etc/cron.d/logbak
0   5   4   6   *   /usr/lib/newsyslog
```

在该文件中的每一行都用 6 个字段来说明一个命令及其执行时间，前 5 个字段用来指定命令执行时间，依次为分（0~59）、时（0~23）、天（1~31）、月（1~12）、星期（0~6）；最后一个字段为执行的命令。字段之间用空格分隔，同一字段内以逗号“ , ”和短横线“ - ”分组，换行用“ % ”符号。如：0 2 * * 1,3 /etc/cron.d/logbak 表示在每星期一和星期三的二时自动执行/etc/cron.d/logbak 程序。

要使修改的 crontab 文件内容生效，必须重新启动进程 cron。cron 在检查命令执行时要参考 crontab 文件的时间戳，以重新加载最新的 crontab 文件。

2. cron 作业配置

在文件/etc/default/cron 中可以配置 cron 作业路径、cron 行为日志及日志文件的大小、cron 作业一次运行拥有的最大进程标识符 pid（默认的最大进程 pid 为 100）。

· 配置 cron 作业路径

通过环境变量 PATH 的设置可以配置 cron 作业路径：

超级用户作业 SUPATH=in /etc/default/cron

普通用户作业 PATH=in /etc/default/cron

· 配置 cron 日志

cron 日志为 cron 行为的记载。通过配置/etc/default/cron 文件中的变量 CRONLOG 来决定是否需要日志（用 CRONLOG=YES 表示要日志，用 CRONLOG=NO 表示不要日志）；用变量 MAXLOGSIZE 配置日志文件的最大块数

（默认的块大小为 512 字节，日志文件大小为最大块数与块大小相乘），如果不配置日志文件的最大块数，则默认的最大块数为 2 048。

· 配置最大进程 pid

在 cron 文件中用变量 MAXCRON 设置最大进程 pid。

例　配置需求情况为：

超级用户 cron 作业在路径/etc/default/cron。

普通用户 cron 作业在路径/etc/default/cron。

cron 需要日志，日志文件的最大块数为 4 096。

一次运行 cron 作业的最大进程 pid 为 1 000。

以下是/etc/default/cron 文件内容配置：

```
CRONLOG=YES
MAXLOGSIZE=4096
SUPATH=in /etc/default/cron
PATH=in /etc/default/cron
MAXCRON=1000
```

除 cron 开始和结束要产生记录外，cron 每次执行命令还要产生一条记录，因此 cron 日志文件经常非常大。有些系统（如 Solaris）中用日志监测器来查看日志是否超越限制，如果超限则将日志文件移到一个指定文件。对于没有日志监测器的系统，首先应该将日志文件的长度设置为 0，过一段时间如果没有错误产生，则运行 echo –n>/var/cron/log（也可自己设置文件名）将其重定向到文件，但是不能用 rm 直接删除日志文件，否则系统会出错。

3. 增加、修改 cron 任务

增加和修改 cron 任务的方法有：

· vi 手工编辑文件 crontab

使用 vi 编辑器手工编辑文件 crontab 时，首先备份系统和用户的 crontab 文件：用 cp 命令进行两次备份，其中一份以备出错时使用，而在另一个备份文件上作增加或修改，最后用修改正确的文件去覆盖正在工作的 crontab 文件。

· 用 crontab 工具编辑文件 crontab

使用 crontab 工具只能编辑用户 crontab 文件（系统 crontab 文件必须用 vi 手工方式编辑）。方法有两种：

方法 1：

首先用 vi 创建一个包含 cron 项的文本文件 filename，然后用命令#crontab filename 验证内容的有效性。

方法 2：

首先创建 crontab 文件的一个拷贝。然后用 crontab –-e 对文件 crontab 的一个拷贝进行处理。

在 crontab 文件修改好后，为了可靠起见，可以用命令 crontab –l 得到 crontab 文件的清单，并用重定向将其保存起来以便以后使用。

4. 删除 cron 任务

有三种方式可以删除 contab 作业：

· 如果要删除 corntab 中的所有任务，用命令 corntab –r 或 contab –d 删除 corntab 文件；

· 如果要删除 corntab 中的部分任务，用 corntab –e 命令编辑 corntab 文件，删除与要删除的任务相对应的行，然后使用 corntab filename 命令执行更新过的文件。有的系统不支持 corntab –e 命令，可采用下面一种删除方式；

· 如果要删除 corntab 中的部分任务而系统拥有包含 cron 命令的最原始文件，用 vi 编辑器编辑该原始文件，删除与要删除的任务相对应的行，并用 corntab filename 命令重新执行更新过的文件。

5. 诊断 cron 故障

cron 是一个自动调度器，用于完成 crontab 文件内容的自动调度，常常会发生的故障有：

· crontab 内容的修改没有生效：可能是手工更改 crontab 文件后没有重新引导该文件；

· cron 作业被跳过而没有执行：注意时间区域是否有效；

· 某些 cron 作业没有执行：手工运行该命令，检查是否能够执行。

用命令 echo"This is a debug test">/dev/console 查看调度时间主控制台是否会产生显示信息；或用命令 echo"This is a debug test"查看是否会产生一个 E-mail。

· cron 作业没有执行，Shell 脚本不工作：检查 Shell 脚本的运行权限及 Shell 环境参数设置情况。

· 不能重新启动 cron 作业：最后的处理只能是系统重新引导。

要彻底搞清楚 cron 作业没有执行的原因，建议最好查看 cron 日志文件。

3.6.2 在指定时间执行命令 at

有些系统命令，特别是常时间占用 CPU 的命令并不希望立即执行，而是想安排在 CPU 稍微空闲的某一指定时间执行，以平衡 CPU 使用。在 UNIX 系统中 at 工具

可以满足这一要求。使用 at 工具的作业称为 at 作业。

在 AIX 和 Solaris 系统中使用 atd daemon 进程处理 at 作业；在另外一些系统中，at 作业不用单独的进程处理而依赖于 cron 进程运行，在 crontab 文件中存在 atrun 项，可以在 crontab 文件中设置执行 atrun 的时间间隔。

at 工具使系统从标准输入或脚本文件中得到命令，而 at 的输出是标准输出、重定向文件或 E-mail 邮件通知。

1. at 命令

命令格式：at [-q queue] [-bdlmr] [-f filename] time [date | +increment]

其中：

time 为执行命令的时间。时间可以是用两位数字表示的小时，用 4 位数字表示的小时和分钟（之间用分号隔开）。

date 为执行命令的日期。可以是某周的某一天（如 Sunday、Monday、Tuesday、……），也可以是某月的某一天（1~31），也可以是 today 或 tomorrow。

increment 为增量的时间、日期，格式为+n units，n 为整数。units 为：minutes、hours、days、weeks、months、years。利用 increment 可以改变时间和日期的使用。如 next month 表示+1 unit（+1 month）。

例 at 指定的时间。

```
$at 0309am Dec 24,2002
$at 0309am December 24
$at 3:09am Sunday
$at now tomorrow
```

2. at 作业

指定一个 at 作业：

如果文件 filename 中包含要执行的作业，就将其安排为 at 作业。

```
$at –m 8am tomorrow<filename
$at –m –f filename
```

at 作业提交后系统会为该作业分配一个作业号（job-id），在某些系统中该作业号由文件/var/spool/at/.SEQ 控制。提交后的 at 作业在作业队列中等待执行。

用命令 atq 或 at –l 按调度执行的时间序列列出系统中的作业，输出信息中最前面的数字表示作业号。

例 $atq

```
67 2003.05.13 02:12 a
```

68 2003.05.15 02:12 a

69 2003.05.17 02:12 a

输出结果中表示作业号的分别为 67、68、69，最后的 a 表示为 at 作业。

在 UNIX 系统中不同类型的作业会排在不同的作业队列中。系统中的作业队列有：

a：at 队列；

b：batch 队列；

c：cron 队列；

=：当前队列。

例 $atq

67 2003.05.13 02:12 =

68 2003.05.15 02:12 a

69 2003.05.17 02:12 a

70 2003.05.20 02:12 b

如果要列出某一个作业队列，则在命令 atq 或 at 中增加选项-q 和-l。

$at –lq a

68 2003.05.15 02:12 a

69 2003.05.17 02:12 a

当 at 作业完成后，可以通过重定向方式将 at 作业的执行结果存到文件中，或通过 E-mail 方式将作业的执行结果发给用户。

如果想取消某个 at 作业，可以用命令：

at –d [job-id…]或

atrm [job-id…]

将作业号为 job-id 的 at 作业从 at 作业队列中删除。如果不送入作业号，则表示取消所有 at 作业。普通用户只能取消自己的作业，超级用户可以取消所有的 at 作业。

3. 在系统负载较轻时执行作业

通常情况下 at 作业的执行不考虑系统当时的负载情况。但是从系统性能平衡考虑，最好能在系统负载较轻时安排这些作业执行。

用命令 at –b 或 batch 提交作业就能在系统负载低于一个平均值时执行该作业。

以 batch 提交的作业是批处理作业，其权限低于 cron 和 at 作业。

4. 控制对 at 的访问

在 UNIX 系统中文件 at.allow 和 at.deny 用于限制用户对 at 的访问。

不同的系统，at.allow 和 at.deny 文件所在的目录不同，通常的目录是/usr/lib/cron 或/usr/etc/。若 at.allow 文件存在则只有该文件中列出的用户可以使用 at；若 at.allow 文件不存在，则 at.deny 文件中列出的用户不能使用 at，若 at.deny 文件中的内容为空则所有用户都能使用 at；若 at.allow 和 at.deny 文件都不存在，则只有超级用户才能使用 at。

3.6.3　作业控制

作业是用户提交给系统处理的一个任务，包括用户程序 、数据 、对程序运行进行控制和处理有关的信息。

按作业调度方式，作业可以分为批处理作业和终端型作业。批处理作业是将多个作业放在一起交给系统，在经过作业调度后才能进入内存就绪；终端型作业是用户终端直接将作业送入内存的作业，不需要经过作业调度。在 UNIX 系统中大量用户通过终端直接提交自己的作业进入内存。只有较少的作业才是批处理作业。

按作业运行方式的不同，作业又可以分为前台作业和后台作业。前台作业是可以直接与终端交互的作业；后台作业是无法与终端直接交互的作业。任何时候，一个终端只能有一个前台作业而可以有多个后台作业。在 UNIX 系统中前台作业和后台作业可以互换。

作业控制包括终止一个作业、暂停一个作业、重新启动暂停作业、将后台作业切换到前台等。

1. 用 jobs 指令显示用户当前的所有作业

命令格式：jobs

例　$jobs

```
[1] +Runing find /u/micael/-print>files>
[2] –Suspended grep netw | nawk –f files>data &
[3] –Stoped (tty input) cat>/tmp/files
$
```

该指令运行结果中最前面括号中的数字表示作业代码号，“+”表示现行作业，“-”表示前一个作业。Runing 表示正运行的作业；Stoped 表示已停止的作业；Suspended 表示挂起的作业。

2. 用 kill 指令终止后台作业

暂停前台作业用 Ctrl+Z；终止后台作业用 kill 指令。

命令格式：kill %pid（pname）

pid 为作业进程号；pname 为作业进程名。

例 $kill %3

终止作业号为 3 的后台作业。

$kill %netw

终止作业名为 netw 的后台作业。

3. 用 stop 指令暂停后台作业

命令格式：stop %pid（pname）

pid 为作业进程号；pname 为作业进程名。

例 $stop %find

暂停后台作业 find。

4. 用 fg 指令将暂停的作业重新启动并带到前台执行

命令格式：fg %pid（pname）

pid 为作业进程号；pname 为作业进程名。

例 $fg find

启动作业 find 并在前台运行。

5. 用 bg 指令将暂停的作业重新启动并带到后台执行

命令格式：bg %pid（pname）

pid 为作业进程号；pname 为作业进程名。

例 $bg find

启动作业 find 并在后台运行。

3.7 性 能 管 理

性能管理是系统管理员的一项非常重要而繁琐的工作，关系到整个系统的硬件、软件和应用等方面的技术。为了使系统稳定、可靠地运行，系统管理员必须不断监控并调整系统的性能。

系统性能管理主要是在通过运行命令得到系统性能统计信息的基础上，进行系统性能分析，提出性能策略，以性能策略为指导调整系统性能，使系统性能稳定可靠。

3.7.1 系统性能

要了解系统性能，首先应该知道系统本身的配置情况，从系统配置理解系统各个部件之间的相互关系及影响；其次，还要针对系统配置对系统进行测试和验证。

1. 系统配置

在系统配置上系统管理员应该知道系统中的 CPU 参数指标及 CPU 的个数、系统总线、系统磁盘和磁盘子系统的情况。

在知道系统配置的基础上，用系统应用软件及提供的软件工具着手收集性能统计数据，根据统计数据分析出影响系统性能的因素：

· 内存或磁盘空间是否构成“瓶颈”，如果构成“瓶颈”，则应增加多少内存和磁盘空间？

· 用户的使用时间是否使得系统负载分配不合理，怎样调整用户使用时间？

· 系统设备是否配置合理，如何改进？

在明确了影响系统性能的因素之后，可能还需对这些因素进行测试和验证，用模拟系统不断模拟和修改变化条件，得到系统较为满意的性能。

2. 性能监控与跟踪

系统监控与跟踪和性能调整关系紧密，系统监控与跟踪可以了解系统配置情况，为成功调整性能奠定基础。

监控是一个连续过程，一旦发现出现了性能问题，就用监控收集的数据予以确认。

端口监控程序主要用于监控系统的硬件端口，发现系统用户和系统设备使用情况及对系统提供的服务访问的情况。

端口监控程序和服务访问的实现通过监测器进行。当某个请求到达时，自动设置请求服务设备和操作系统之间建立通信所需要的参数，并将控制交给能提供所需服务的进程。

端口监控程序和服务访问提供两种类型的端口监控程序：listen 和 ttymon。

listen 端口监控程序检测和控制网络服务访问、远程打印机和文件系统请求。

ttymon 端口监控程序监测和控制用户登录。

在 Solaris 系统中提供了服务访问工具（SAF），主要用于负责注册和监控终端、打印机、调制解调器端口状态，控制资源，允许用户本地或远程登录、通过网络访问打印机及文件。

系统文件/usr/adm/wtmp 可以跟踪用户的登录和使用信息。用 ac 命令可以生成该文件。命令格式：

ac [-w wtmp] [-p] [-d] [username]…

其中选项：

w：指定除文件/usr/adm/wtmp 之外的另一个文件用于跟踪用户的登录和使用信息；

P：表示按个人的总量显示输出结果，不用该选项则表示按系统全体用户的总量显示输出结果；

d：表示产生一份用户信息的报告，每天 24 小时运行。

username：表示系统只为 username（用户名）产生报告。

系统文件/usr/adm/wtmp 也被称为记帐文件。根据该文件的数据可以将系统性能与用户访问联系起来。

除 ac 命令之外 last 命令用于显示记帐文件中的某个用户使用或终端登录情况。

系统文件/usr/adm/acct 用于收集进程记帐信息。要使该文件能够收集进程信息必须先用命令 accton 启动进程。经过一段时间该文件逐渐变大，当空闲磁盘空间下降到系统总磁盘空间的 2%以下时，记帐进程会停止；当空闲磁盘空间上升到系统总磁盘空间的 4%以上时，记帐进程会重新启动。

用 sa 命令进行信息汇总。命令格式：

/etc/sa [-abcdlmr] [filename]

其中选项：

filename：表示如果不用/usr/adm/wtmp 文件还可以另外指定一个文件 filename；

a：表示在报告中列出全部命令；

b：表示根据系统时间和用户时间之和与调用次数的比值对数据排序；

c：表示在输出中增加各项时间占总时间的百分比；

d：表示根据磁盘 I/O 次数进行排序；

l：表示分别报告系统时间和用户时间；

m：表示显示各用户的进程数和 CPU 时间；

r：表示将排序次序变成反序。

3.7.2 性能调整

通过系统资源的使用情况来调整性能。

1. 内存与调页

在正常工作负载时监控系统确定可接受的调页频率。调页频率可接受值依赖于用户所希望的响应时间。响应时间太长，调页率过高，则应考虑增加内存。

2. 磁盘空间与用户访问

磁盘空间的大小会影响用户使用情况。当磁盘空间剩余低于 25%时，应该增加磁盘空间或重新分布文件在磁盘上的情况，达到均衡使用各磁盘。

用户访问磁盘的频率过高会影响响应时间，应考虑将部分用户文件移到另一个访问频率低的磁盘上,如果有多个磁盘控制器还应考虑磁盘连到磁盘控制器的情况。

3.7.3　收集性能统计信息

下面的命令可用于收集操作系统信息或当前用户运行进程信息。

1. ps 命令

ps：显示正在运行的进程信息。

例　#ps –aux

```
USER PID   %CPU   %MEM   VSZ   RSS   TT   STAT   STATED   TIME      CMD
root   0     3.3     0.0    282   16    P3    I     10Sep   00:02:00 (swapper)
root   1     1.5     0.0    242   126    ??   I     10Sep   01:02:00   init
Jak   125     0.0     0.0    145   1120   ??    S     10Sep   00:02:00   irc
togle 223     0.0     0.0    245   320   P4    R     10Sep   00:02:00   ps-uax
#
```

在输出的列中“ %CPU ”表示进程使用 CPU 的时间，“ %MEM ”表示进程使用内存的情况，“ VSZ ”表示使用虚拟内存的情况。“ RSS ”表示各进程实际使用的物理存储量，其中包括了多个进程共享的部分内存。“ STAT ”显示了各进程的状态，有下列进程状态：

R：进程运行；

S：进程睡眠，不可中断；

s：进程睡眠；

I：进程空闲；

T：进程停止；

H：进程挂起；

W：进程被换出到磁盘上；

>：进程超过了内存所需的软限制；

+：带有一个控制终端的进程组领导者；

N：进程在优先级降低的情况下运行；

<：进程在优先级升高的情况下运行。

如果用 ps 命令显示出有很多进程都被换出到磁盘上，则应把大部分精力放在物理内存和虚拟内存的调整上；如果 ps 命令显示结果是某个进程占用 CPU 时间百分比很高，则应该用 nice 或 renice 命令暂时降低该进程的优先级，推迟该进程结束的时间以使其他进程结束时间提前；如果 ps 命令显示出不该存在于系统的进程，则用命令 kill 杀死它们。

2. top 命令

top 命令可以得到某一时刻的系统性能。该命令列出系统中消耗 CPU 资源最多的 15 个进程。命令格式：

top [-sdilnqu] [-dcount] [-stime] [-ofield] [-Uusername] [number]

其中选项：

s：表示显示结果中应包含系统进程（如 sched、init）；

b：表示在批处理方式下运行。输出结果写入一个文件中。除^c 和^\外，所有的键盘输入无效；

n：表示在非交互模式下运行，这时和 b 选项相同；

i：表示在交互模式下运行，支持键盘的输入和光标定位；

l：表示不显示空闲进程；

q：表示在优先级为-20 的情况下运行时，如果进程运行太慢就用 reniac 提高命令的执行效率；

u：表示不必将 uid 转换为用户名；

d count：表示更新屏幕显示结果 count 次之后退出 top 命令；

s time：设置连续两次更新屏幕显示的时间间隔；

u username：表示只显示属于用户 username 的进程。

例 用不带选项的 top 命令得到系统性能的使用情况。

```
#top
```

3. uptime 命令

uptime 显示从系统引导到现在已过的时间长度，以及过去的 1、5 和 15 分钟内所执行的平均工作数信息。

例 #uptime

```
12:23PM    up 72 days,   21:06,   1 user,   load average: 0.00, 0.03, 0.02
#
```

4. w 命令

w 显示与“谁在系统中”有关的信息，包括 CPU 使用情况，注册时间、用户进程空闲时间等信息。

例　#w

```
12:24PM    up 72 days,   21:07,   1 user,   load average: 0.00, 0.02, 0.02
User       tty          login@      idle      JCPU      PCPU what
liuxm      pts/0        12:21PM        0         0         0 -ksh
#
```

5. vvmstat 命令

vmstat 显示虚拟内存信息，包括调页、磁盘、内存和 CPU 的统计数据。在 vmstat 命令中包含行数和时间间隔两个参数。命令格式：

vmstat [-ims] [-c count] [-M core] [-N system] [-w wait] [disks]　或

vmstat [-fsi] [Drives] [Interval] [Count]

其中选项：

i：表示显示自从系统启动以来每个设备的中断次数；

M core：表示显示与展开 core 文件（默认值为/dev/komem）后得到的名字列表值对应的列表；

N system：表示从 system 而不是从默认值/bsd 中展开名字列表；

m：显示内核内存使用情况统计信息；

s：显示与调页相关的活动的统计信息。

例　显示进程信息、内存信息、换页信息、故障信息及 CPU 使用信息。

```
#vmstat
kthr       memory                page                faults          cpu
----- ----------- ------------------------ ------------ -----------
 r   b     avm     fre   re   pi   po     fr     sr          cy    in    sy    cs us sy id wa
 0   0 12372 29776     0    0    0    0      0      0 108     34    26    0    1 99    0
#
```

显示进程信息、内存信息、换页信息、故障信息及 CPU 使用信息，时间间隔 3s。

```
# vmstat 3
kthr       memory                page                faults          cpu
----- ----------- ------------------------ ------------ -----------
 r   b     avm     fre   re   pi   po   fr     sr   cy    in      sy    cs    us    sy id wa
```

```
0   0 12372 29776    0    0    0    0    0    0 108    34   26    0    1 99    0
0   0 12372 29775    0    0    0    0    0    0 105   217   26    0    1 99    0
0   0 12372 29775    0    0    0    0    0    0 107    83   28    1    0 99    0
0   0 12372 29775    0    0    0    0    0    0 107    83   27    0    1 99    0
0   0 12372 29775    0    0    0    0    0    0 106    87   30    0    0 99    0
0   0 12372 29775    0    0    0    0    0    0 106    83   29    0    0 99    0
0   0 12372 29775    0    0    0    0    0    0 108    83   29    0    1 99    0
0   0 12372 29775    0    0    0    0    0    0 112    83   26    0    0 99    0
0   0 12372 29775    0    0    0    0    0    0 117    83   27    0    1 99    0
0   0 12372 29775    0    0    0    0    0    0 114    83   25    0    1 99    0
0   0 12372 29775    0    0    0    0    0    0 107    83   29    0    0 99    0
#
```

输出信息有进程状态、内存使用情况、调页情况、故障和 CPU 使用情况。

其中：

kthe 中的 r 为运行队列进程数、b 为等待资源的阻塞队列进程数、w 为就绪换出队列进程数。

memory 中的 avm 为当前处于活跃进程的页面数（页面大小为 1 024 字节）、fre 为空闲页表中可用的页面数。如果 fre 很大但有大量的调页活动，则表明实际物理内存容量不足，应增加物理内存容量或者将系统负载分布到两段时间中完成。

Page 是关于缺页故障和调页活动的信息，是间隔的平均值，以秒为单位，其中 re 是从非活跃列表中重新收回的页面数，pi 是调入的页数，po 是调出的页数，fr 是空闲页数，sr 是考虑要换出的页数，sr 是通过页替换算法扫描的页面，cy 是按页替换算法的时钟周期。

fault 是采样间隔平均每秒的捕获和中断率，其中 in 是设备中断，sy 是系统调用，cs 是内核线程上下文切换。

cpu 是 CPU 使用时间故障百分比，其中 us 是用户时间、sy 是系统时间、id 是 CPU 空闲时间，wa 是 CPU 空闲期间系统是否有未完成的磁盘或者 NFS I/O 请求。

下面显示与调页相关的活动的统计信息。

```
#vmstat -s

      4536791 total address trans. faults

        18317 page ins

       305705 page outs

            0 paging space page ins

            0 paging space page outs
```

```
          0 total reclaims
    1759399 zero filled pages faults
       6051 executable filled pages faults
          0 pages examined by clock
          0 revolutions of the clock hand
          0 pages freed by the clock
      14531 backtracks
          0 lock misses
          0 free frame waits
          0 extend XPT waits
      10764 pending I/O waits
     278536 start I/Os
     278536 iodones
  168559972 cpu context switches
  682892310 device interrupts
          0 software interrupts
          0 traps
  218881153 syscalls
#
```

输出的是系统启动以来的统计信息。

6. iostat 命令

iostat 命令显示与磁盘和终端 I/O 有关的统计信息。由于磁盘是外围设备，磁盘 I/O 的速度较慢，从磁盘读数据或向磁盘写数据需要耗费相当长的时间。然而由于存在高速缓存（cache）和缓冲区，读写磁盘数据可以暂存缓存，不用每次都访问磁盘。命令格式：

iostat [drive…] [interval] [count]或

iostat [-c count] [-M core] [-N system] [-w wait] [drives]

例　#iostat

```
tty          rz1             rz2              cpu
tin  tout    bps  tps    bps  tps          us  ni  sy  id
30      2      2    0     39    2          10   0   4  78
0      60      0    0     43    2           2   0   6  84
32     42      0    0      0    0           8   0   7  88
```

输出中 tin、tout 分别表示每秒钟输入/输出的总字符数。

bps 为磁盘每秒钟传输的比特数，tps 为磁盘每秒钟的实际传输次数。

us 为用户模式下运行所花费的 CPU 时间，ni 为优先级，sy 为系统时间，id 为 CPU 空闲时间占总 CPU 时间的百分比。

用该命令可以判断应用程序的调度行为是否正确，如果 tcp 值很高，则应用程序访问磁盘的次数太频繁，应该作相应调整；另外用该命令还可以判断何时系统的磁盘负载过重，是否应该增加磁盘容量或用其他磁盘分载。

ipcs：显示与内部进程通信（IPC）有关的信息。

netstat：显示与网络活动相关的数据结构统计信息。

snoop：捕获并显示为诊断网络问题而发出的网络报文。

tpcdump：收集网络中的报文信息，允许用户通过 booler 表达式选择报文。

ping：诊断网络的连接和阻塞问题。

nfsstat：显示与网络文件系统（NFS）有关的信息。

traceroute：显示路由到目的地的主机报文，用于诊断网络拥塞。

例　下面是 AIX 在环境中使用 iostat。

```
#iostat
tty:    tin       tout     avg-cpu:   % user     % sys      % idle     % iowait
0.0                2.1                0.1        0.6        99.3          0.0
Disks:          % tm_act       Kbps        tps       Kb_read     Kb_wrtn
hdisk1             0.1          0.2         0.0         74482     1220069
hdisk0             0.0          0.0         0.0             0           0
cd0                0.0          0.0         0.0             0           0
#
```

3.8 本 章 小 结

本章和下一章中的内容对于一个要成为系统管理员的人来说尤为重要，都是一些日常维护中需要掌握的知识。

对系统的引导过程、运行级概念、运行控制脚本的维护应该是每个 UNIX 管理者都应该掌握的，是基础中的基础，有助于更深入地理解整个系统是如何初始化的。

用户和用户组的管理，是 UNIX 操作系统的几大特点之一，也是日常维护的重点之一；熟练掌握系统自带的管理配置工具，对于减化管理操作和提高正确性都有很大的帮助，也是一个不可忽视的内容。

上 机 练 习

1. 练习引导系统和关闭系统。
2. 练习用命令 useradd 添加一个用户 student，然后用 userdel 命令删除该用户。
3. 熟悉 Solaris 管理工具 Admintool。
4. 熟悉 AIX 管理工具 SMIT。
5. 练习用 cron 实现一个任务自动调度的全部过程。
6. 练习用 at 实现一个任务在给定时间自动调度的全部过程。
7. 熟悉本章节介绍的性能管理命令，尝试分析各种命令的执行结果，对系统进行合理调整。

习　题　三

1. 叙述 UNIX 系统引导过程包含那些处理步骤？
2. 在 UNIX 中运行级分为几级？为何要设置这些运行级？各级有何特点？
3. 运行控制脚本在什么目录下？
4. 在 UNIX 中系统管理员如何与系统对话？
5. 在 UNIX 中普通用户如何拒绝与其他用户对话？
6. 关闭系统命令 shutdown、halt、reboot、init、telinit 各有何特点？
7. 增加用户和组，系统要修改哪些文件？

第 4 章　UNIX 设备管理

设备管理是操作系统的功能之一，设备管理包括终端、磁带、软盘、CD-ROM、硬盘和打印机等设备的管理和使用。UNIX 操作系统在设备管理上实现了将设备管理与文件系统相联系，通过文件系统来管理设备，把每个连入系统的设备都与一个特殊文件相关联，每个设备都被看成特殊文件（字符文件与块文件），用文件管理方法实现设备管理。

本章主要内容

- 设备管理概述：首先分析设备类型，然后介绍设备与文件系统的关系。
- 终端管理：终端管理主要包括终端的设置（终端的连接、终端的配置、终端的开启）、终端的管理与管理命令（设置终端类型、设置主控台、stty 命令）。
- 磁带管理：磁带管理主要包括磁带命名、磁带操作命令（tape、tar、cpio、pax）。
- 软盘管理：软盘管理主要包括软盘使用、软盘操作命令（软盘的加载与卸载）。
- CD-ROM 管理：在 CD-ROM 管理部分主要介绍卷管理、CD-ROM 与卷管理，并附带介绍软盘与卷管理。
- 硬盘管理：硬盘管理部分主要介绍硬盘命名、硬盘片、测试硬盘、硬盘复制。
- 打印管理：打印管理部分主要针对 LP 打印服务、打印管理与维护、打印机的使用三个方面展开。

4.1　设备管理概述

UNIX 系统将设备分为两类：块设备和字符设备。

块设备是按固定大小数据块进行信息存储和访问的设备，不同系统数据块的大小不同，通常从 512~32 768 字节。特征是传输速率高。常用的块设备有硬盘。

字符设备是按字符进行信息存储和访问的设备，这类设备按字符流的方式进行数据发送和接收，常用的字符设备有终端、打印机、鼠标和网络接口。

在 UNIX 系统中为了管理方便，常常用物理设备名与逻辑设备名称呼设备。物理设备名针对的是直接与计算机相连的输入/输出设备的真实名称；逻辑设备名是从用户 I/O 请求到物理设备发生动作之间的管理方式中的设备代码称呼方式，是一个抽象的逻辑概念，可以不指具体某台物理设备。

4.1.1　设备与文件系统

在 UNIX 系统中对设备管理的实现通过文件子系统与设备接口进行，将每个设备映射成文件系统目录树中的一个节点。由于每类设备都对应一个设备驱动程序，通过文件子系统将设备驱动程序与操作系统的核心关联起来。图 4.1 为硬件设备、设备驱动程序、核心与文件子系统的关系。

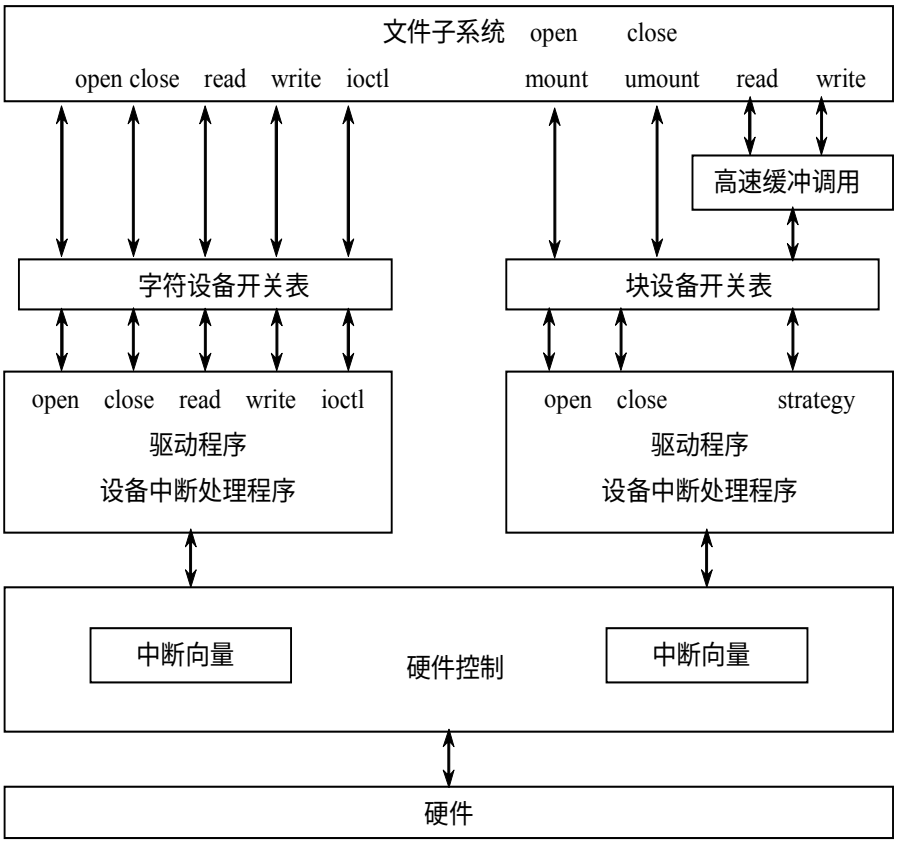

图 4.1　设备驱动程序、核心与文件子系统关系

无论是用户使用硬件设备还是系统管理员管理硬件设备，最终都要通过文件系统中的系统调用来实现。系统调用通过设备开关与核心中的设备管理程序相对应，设备管理程序对应于设备驱动程序，设备驱动程序借助中断向量对应于硬件设备。这种用户通过文件子系统使用设备、系统管理员通过文件子系统管理设备的方式就是 UNIX 系统用文件系统管理设备的思想。

计算机中对磁盘类块设备的访问频率非常高，为了使块设备能与计算机的 CPU 速度匹配，在系统内存中开辟了高速缓存，因此在图 4.1 中文件子系统和块设备开关表之间存在高速缓冲调用。

4.1.2 文件系统中设备管理目录

在 UNIX 文件系统的根目录下存在一个目录/dev，这是一个用于设备管理的特殊目录，也称为设备管理目录。在该目录下存放有供所有用户使用的设备管理程序，用户用逻辑设备名调用里面的程序。

例 $cd /dev

```
$ ls -l
total 16
crw-rw----   1 root     system    10,  0 Jun 16 2001   IPL_rootvg
cr--r----T   1 root     system     8,  0 Jun 16 2001   audit
crw-r-----   1 root     system     3,  0 Jun 16 2001   bus0
br--r--r--   1 root     system    12,  0 Jun 16 2001   cd0
crw-rw-rw-   1 root     system    14,  0 Jun 16 2001   clone
crw--w--w-   1 root     system     4,  0 Jun 16 2001   console
crw-rw--w-   1 uucp     uucp       2,  2 Jun 24 17:06  dtremote
brw-rw-rw-   1 root     system    17,  0 Apr 08 18:00  fd0
brw-rw-rw-   2 root     system    17,  1 Jun 16 2001   fd0h
brw-rw-rw-   2 root     system    17,  2 Jun 16 2001   fd0l
brw-rw----   1 root     system    10,  8 Jun 12 2002   hd1
brw-rw----   1 root     system    10,  5 Jun 12 2002   hd2
brw-rw----   1 root     system    10,  7 Jan 15 2002   hd3
$
```

输出信息第一列第一个字符“ b ”表示块设备，“ c ”表示字符设备；最后一列是设备的逻辑名，如 fd1、hd1、console；第 4 列（即 system 后的列）表示设备类型号，如 hd1、hd2、hd3 的设备类型号都为 10，fd0、fd01、fd0h 的设备类型号都为 17。

用户或应用程序发送给逻辑设备的数据经过设备管理与驱动程序传递给某个指定的物理设备；相反，用户或应用程序读出的数据也一定要经过设备管理与驱动程序从物理设备中获得。

系统管理员可以使用命令 mknode 创建设备文件，块设备文件可以看成是字符设备文件的特例，如可以将磁盘文件定义为字符文件，但必须在逻辑文件名前加一

个字母“ r ”，如 rfd0。

4.2　终 端 管 理

由于 UNIX 系统是多用户系统，通常情况下系统只有一个主控台，绝大多数用户使用系统要通过终端登录。

4.2.1　终端设置

1. 终端连接

直接连接（本地终端）：主机上的用户多路卡提供多个串口，终端通过标准 RS-232 电缆线，用交叉方式（2-3、3-2、7-7）直接连接到主机。

间接连接（远程终端）：远程终端通过 Modem 或网络连接。如果用 Modem 连接，就用非交叉方式的 RS-232 电缆线，主机与 Modem 相连，Modem 与终端相连。如果用网络连接，通常用 TCP 应用中的 telnet 登录到系统。

2. 终端配置

终端可以是一般终端，也可以是 PC 机。

用 Windows 操作系统中提供的终端仿真程序可以将 PC 机当终端使用。

一般终端应该配置，在终端配置菜单上选择通信波特率（通常是 9 600）、数据位（通常是 8 位）、停止位（通常是 1 位）、奇偶校验位（通常是无）等。

3. 开启终端

本地终端首先在主机上用命令：#enable /dev/ttyname，其中 ttyname 是终端设备号，如 ttyna 表示连接到主机串口 n 的第一个设备，ttynb 表示连接到主机串口 n 的第二个设备，ttync 表示连接到主机串口 n 的第三个设备，依此类推；在主机上配置好以后，终端打开后屏幕上就会显示：login：提示符，否则用命令检查终端：

```
#disable /dev/ttyname
#date>/dev/ttyname
```

如果连接正确则终端屏幕上会出现日期信息。否则显示连接线有问题等。

4.2.2　终端管理

目前 UNIX 系统可以支持多种字符终端。在 SYSV 中，主机支持的终端类型的二进制文件放在目录/usr/lib/terminfo/下，如 vt100 终端和 xterm 终端分别对应/usr/lib/terminfo/v/vt100 和/usr/lib/terminfo/x/xterm 文件。在 BSD 版本中，主机支持

的终端类型放在文本文件/etc/termcap 中。

对 BSD 版本的终端，在系统实现上该文本文件最终还是要转换成二进制文件。方法是通过终端编译器（Terminfo Compiler）编译生成目标码文件，并将目标码文件写到/etc/terminfo/v/vt100。普通用户不能直接写终端管理文件，可以用命令：

setenv TERMINFO directory-name

指定自己的环境变量 TERMINFO 并将编译结果写到文件/etc/terminfo/v/vt100。

终端管理文件除了上面的可登录终端类型管理文件之外，还有系统的启动文件/etc/initab（其中包含生成终端进程的配置），/etc/gettydefs 终端行特性数据库文件，/etc/termcap 终端特性数据库文件。

4.2.3 终端管理命令

除了可以通过终端管理文件设置终端外，用户还可以用命令或用户环境文件设置终端。

1. 设置终端类型

如果用户使用的是 C Shell，则用命令：

$setenv TERM termtype

其中 termtype 表示终端类型，如 vt100。

如果用户使用的是 Bourne Shell 或 Korn Shell，则用命令：

$TERM=termtype；export TERM

其中 termtype 表示文件/etc/termcap 中定义的一种终端类型名，如 vt100。这种命令修改方式只在本次登录期间生效，下次登录时还需重新输入该命令。如果要自动永久生效，则采用下面两种方式。

- UNIX 中用户环境变量放在隐含文件 $HOME/.login（C Shell）或 $HOME/.profile（B Shell 或 K Shell）中，TERM 是环境变量，也放在该文件中，在该文件中加入与命令 Shell 相对应的命令：

$setenv TERM termtype 或$TERM=termtype；export TERM

则长期生效。

- 文件/etc/ttytype 定义了各终端设备所对应的默认终端类型，可以用命令查看所对应的终端类型：

#tty

/dev/tty01

本设备对应的终端号为 vt01，再看文件/etc/ttytype

#cat /etc/ttytype

```
ansi      tty01
ansi      tty02
ansi      tty03
```

从 tty1a 终端所在行可见该终端类型为 ansi。如果要修改终端类型，直接编辑该文件。比如将 ansi tty01 改为 vt100 tty01；修改完后再在对应的文件.login（如果是 C Shell）中加入一行：

```
tset –s –Q>/tmp/tset $$；source /tmp/tset $$；/bin/rm /tmp/tset $$
```

其中 tset –s –Q 执行的输出送入文件/tmp/tset $$，内容是：

```
set noglob;
setenv TERM vt00;
unset noglob;
```

之后 source /tmp/tset $$表示执行/tmp/tset $$中的内容，执行完后用/bin/rm /tmp/tset $$命令删除临时文件/tmp/tset $$。

如果是 B Shell 则在文件.profile 中加入一行：

```
eval `tset –s1`
```

修改完成后要退出系统再次登录，系统会根据终端设备情况自动设置终端类型。

2. 设置主控台

在 UNIX 系统中主控台直接和主机相连，是系统运行的主窗口，常用于系统维护和管理。如果需要也可以将某一台用户终端设置成主控台。

将用户终端设置成主控台的步骤为：

(1) 在文件/etc/default/boot 中变量 SYSTTY 控制当前主控台对应的终端设备，SYSTTY=0 表示系统用第一块显卡作为主控台；SYSTTY=1 表示系统用串口 1（COM1）所连接的终端作为主控台。所以应该设置成：

```
SYSTTY = 1
```

(2) 在文件/etc/default/login 中，变量 OVERRIDE 定义当系统带保护的口令字数据库遭到破坏后，仍然允许 root 用户登录的终端设备。把 OVERRIDE 变量定义为：

```
OVERRIDE=tty01
```

修改完成后再次引导系统，则与串口 1 所连接的终端 tty01 成为主控台。

要恢复原来的主控台，只需恢复以上两个文件/etc/default/boot、/etc/default/login 即可。

如果只是临时需要改变主控台，则不必修改文件，只需在系统引导时键入：

```
boot:hd(40)unix systty=sio
```


将主控台改回主机自己的键盘和显示器，用：

boot:hd (40) unix systty = cn

3. stty 命令

stty 命令可以显示和暂时修改串行线当前的所有操作特性。命令格式：

stty [-a]或 stty mode

其中命令格式 stty [-a]可以显示与特定终端相连的串行线的操作状态，选项 a 表示 all，显示所有状态。没有选项则表示终端当前的串行线操作特性。

命令格式 stty mode 用于显示和修改指定终端的串行线特性。

```
#stty </dev/tty1a
speed 9600 band:
-parity hupel clocal
#
```

模式 mode 有：

speed：终端波特率；

cs7，cs8：字符长度；

parenb：是否进行奇偶校验。Parenb=yes 表示是，Parenb=no 表示否；

intr：中断进程键；

quit：退出键；

erase：删除键；

kill：删除整行键；

eof：文件结束符；

stop：暂停键；

start：恢复由 stop 挂起的输出；

switch：把 Shell 的当前层变为控制层；

sane：将所有控制状态重置为合理值。

例 修改/dev/tty01 终端的波特率、中断键为 Ctrl+C。

```
$stty 9600</dev/tty01
$stty intr "^C"
```

中断设置乱了后可以用命令：$stty sane 恢复。

4.3 磁 带 管 理

磁带是一种大容量外部存储设备。磁带自身的结构特点决定了在磁带上只能顺序存取文件。不同种类的磁带，其容量不相同，通常从上百 MB 到 GB。对磁带的

操作命令有 tar、cpio、pax。

4.3.1　磁带命名

磁带放在磁带驱动器中使用，常用的磁带驱动器种类较多，主要有如下几种。

- SCSI 磁带驱动器：驱动器为 0.5 英寸前端加载的轴-轴型或 4 mm 和 8 mm 螺旋扫描型，磁带为 0.25 英寸盒式。这种磁带驱动器一般与主机适配卡号和类型有关，另外还与控制器有关。
- EIDE 磁带驱动器：这种类型的驱动器与控制器关系密切，有主 IDE 和从 IDE 控制器，主设备和从设备。
- 盒式磁带机：这种磁带机与控制器类型、中断号及 I/O 基址有关，具有普通全尺寸盒式磁带机。

在一个系统中可能同时存在 SCSI 和非 SCSI 磁带驱动器。一个 SCSI 控制器最多可以带 8 个 SCSI 磁带设备，一个非 SCSI 控制器最多可以带 4 个非 SCSI 磁带设备。

用户一般通过逻辑设备名使用磁带驱动器，并不关心磁带驱动器类型及设备物理名。文件系统中的子目录/dev/rmt 支持不同的磁带设备文件。

4.3.2　磁带操作命令

由于磁带的容量非常大，在 UNIX 系统中常用磁带作系统备份。除了用于系统备份的命令外，磁带操作还可以使用以下命令。

1. tape 命令

tape 命令主要用于维护磁带机。命令格式：

tape [-8cfis] commamd [磁带设备名]

其中选项：

8：表示 QIC-80 mini 盒式磁带机；

c：表示 QIC-02 盒式磁带机；

f：表示 QIC-40mini 盒式磁带机；

i：表示 Irwin mini 盒式磁带机；

s：表示 SCSI 磁带机。

command 有：

- reten：拉紧磁带；
- rewind：将磁带倒到开始处；
- format：格式化；

- erase：擦除并重新拉进磁带；
- reset：重置磁带控制器和驱动器；
- status：报告磁带状态；
- eod：将磁带进到数据结尾处；
- rfm：将磁带进到下一个文件标志处；
- amount：报告当前或最后一次传输的数据量；
- wfm：写一个文件标志；
- unload：卸载磁带。

如果没有给出设备名，该命令就从文件/etc/default/tape 中获得默认设备名进行操作。

例 查询磁带状态：

#tape –s status

将磁带倒到开始处：

#tape –s rewind

卸载磁带：

#tape –s unload

2. tar 命令

tar是一个绝大多数UNIX系统都支持的命令,可以将文件或目录树复制到磁带。tar 命令的缺点是不知道文件系统的边界、全路径名长度不能超过 255 个字符、不能复制空目录和特殊文件（设备文件）、不能创建多个磁带卷。

(1) 用 tar 命令将文件复制到磁带

首先进入到要复制的文件目录，然后输入：

tar crvf devname filename1 filename2……

devname 是设备的逻辑名，如 Solaris 系统中磁带设备的逻辑名为/dev/rmt/n。

filename1，filename2，……，filenamen 是要复制的文件名（也可以是文件名的正则表达式），可以将 n 个文件一起复制。

其中选项：

c：表示用覆盖方式写到磁带上，覆盖磁带上已存在的文件内容；

r：表示用添加方式写到磁带上，不覆盖磁带上原有的内容；

f：表示之后紧跟的是设备逻辑名；

v：表示显示文件复制信息。

例 在 Solaris 系统中用 tar 命令将/home/mikey 目录下的文件 mikey.name 复制到磁带。

```
$cd /home/mikey
$tar rvf /dev/rmt/0 mikey.name
a mikey.name 46 blocks
$
```

如果用添加方式写到磁带上则更安全：

```
$cd /home/mikey
$tar tvf /dev/rmt/0 mikey.name
a mikey.name 46 blocks
$
```

系统文件/etc/default/tar 中有设备的逻辑名与对应行标号，在 tar 命令中可以使用对应的行标号代替设备逻辑名。

```
$cat /etc/default/tar
archive1=/dev/rmt/0    18        360        y
archive2=/dev/rmt/1    18        360        y
archive3=/dev/rmt/2    10        720        y
archive4=/dev/rmt/3    18        0          y
archive=/dev/rmt/8     18        720        y
$
```

其中行格式为：行标号=设备名　块因子　容量　设备类型

块因子表示复制文件时的块大小，以 512 字节为单位。容量表示设备存储空间大小，以 KB 为单位，容量为 0 表示该设备已没有可用空间。设备类型为 y 表示磁带设备，为 n 表示非磁带设备。

例　$tar rvl mikey.name 相当于：

```
$tar rvf /dev/rmt/0 mikey.name
```

$tar cvl mikey.name 相当于：

```
$tar cvf /dev/rmt/0 mikey.name
$
```

由于 tar 命令不知道文件边界，不能创建多个磁带卷，所以，如果要复制多于一个磁带容量的内容时，通常在容量字段中填入磁带容量大小，这样当复制的内容超过磁带容量时，系统会显示换上新磁带以继续的提示。

(2) 用 tar 命令列出磁带上的内容

输入命令：tar tvf devname

除选项 t 表示列表指定文件外，其他选项同 tar 复制命令相同。

例　在 Solaris 系统中应用 tar 命令列出磁带中的内容。

```
$tar tvf /dev/rmt/0
rw-rw-rw-    6693/10 May 24 10:32 1999    mikey.name
$
```

(3) 用 tar 命令从磁带上获取文件

要获取磁带中的所有文件，输入命令：tar xvf devname

要获取磁带中的指定文件，输入命令：tar xvf devname filename1 filename2 ……

例　在 Solaris 系统中应用 tar 命令获取磁带中的所有文件。

首先进入到要获取的文件所在目录，然后输入命令：

```
$tar xvf /dev/rmt/0
x mikey.name, 6693bytes, 46 tape blocks
$
```

用 tar 命令获取磁带中的指定文件 mikey.name：

首先进入到要获取的文件所在的目录，然后输入命令：

```
$tar xvf /dev/rmt/0 mikey.name
x mikey.name, 6693bytes, 46 tape blocks
$
```

3. cpio 命令

cpio 命令利用标准输入/输出实现复制文件，利用重定向将标准输入/输出方式转向文件的路径名列表。

与 tar 命令不同，cpio 命令具有如下特点：

- 可以复制特殊文件（设备文件）；
- 恢复文件时可以跳过磁带上的坏区；
- 可以在不同系统之间移植、创建多个磁带卷。

(1) 用 cpio 命令将所有文件复制到磁带

首先进入到要复制的文件目录，然后输入：

```
ls |cpio –oc>devname
```

该命令覆盖磁带上的所有文件。

其中 devname 是设备的逻辑名；选项 o 表示复制文件；选项 c 表示以 ASCII 形式写头信息，便于移植。

例　在 Solaris 系统中应用 cpio 命令将/home/mikey 目录下的所有文件复制到磁带。

```
$cd /home/mikey
$ls |cpio –oc>/dev/rmt/0
```

```
46 blocks
$
```

(2) 用 cpio 命令列出磁带上的内容

输入命令：cpio –civt<devname

选项 i 读取磁带内容；v 显示输出（格式与 ls –l 命令相似）；t 列表显示磁带上的内容。

例　在 Solaris 系统中应用 cpio 命令列出磁带中的内容。

```
$cpio –civt</dev/rmt/0
10666    winsor    6693    May 24 10:32 1999    mikey.name
$
```

(3) 用 cpio 命令从磁带上获取文件

要获取磁带中的所有文件，输入命令：cpio –icv<devname

要获取磁带中的指定文件，输入命令：cpio –icv　`filename´　<devname

例　在 Solaris 系统中应用 cpio 命令获取磁带中的所有文件。

首先进入到要获取文件即将移至的目录，然后输入命令：

```
$cpio –icv</dev/rmt/0
mikey.name
$
```

用 cpio 命令获取磁带中的指定文件 mikey.name：

首先进入到要获取文件即将移至的目录，然后输入命令：

```
$cpio -icv mikey.name </dev/rmt/0
x mikey.name
$
```

4. pax 命令

在满足标准 POSIX 的 UNIX 系统中，pax 为可移植存盘互换，兼容性比 tar 和 cpio 命令更好。cpio 命令可以复制文件、特殊文件（设备文件）或需要多磁带卷的文件系统。pax 命令的缺点是不知道文件的边界，文件全路径名不能超过 255 个字符。

用 pax 复制到磁带的文件可以用 tar 命令从磁带中列出和获取。

(1) 用 pax 命令将目录中的所有文件复制到磁带

要将目录中的所有文件复制到磁带，首先进入到要复制的文件目录，然后输入：

```
#pax –w<devname
```


其中-w 选项表示将文件写到磁带。

例 在 Solaris 系统中应用 pax 命令将/home/mikey 目录下的所有文件复制到磁带。

```
$cd /home/mikey
$pax –w</dev/rmt/0
$
```

(2) 用 pax 命令列出磁带上的内容

要列出磁带上的内容，输入命令：pax –l<devname

选项-l 列出磁带内容。

例 在 Solaris 系统中应用 pax 命令列出磁带中的内容。

```
$pax –l</dev/rmt/0
./mikey.name
$
```

(3) 用 pax 命令从磁带上获取所有文件

要获取磁带中的所有文件，输入命令：pax –r<devname

例 在 Solaris 系统中用 pax 命令获取磁带中的所有文件。

首先进入到要获取文件即将移至的目录，然后输入命令：

```
$pax -r</dev/rmt/0
$
$ls –l
-rw-rw-rw-    6693/10 May 24 10:32 1999    mikey.name
$
```

4.4 软 盘 管 理

使用最多的可移动存储介质是软盘，有 5.25 英寸和 3.5 英寸两种类型。现在 5.25 英寸的软盘几乎不再使用。3.5 英寸的高密软盘容量为 1.44 M 字节。

4.4.1 软盘使用

在软盘应用命令中通常使用软盘的逻辑名称。软盘分为两类：块设备类型和字符设备类型。在 Solaris 系统中块软盘驱动器设备名为/dev/diskette，字符软盘设备名为/dev/rdiskette。软盘初次使用之前通常要格式化，在 UNIX 系统中格式化为 UFS 文件格式。

ufs 格式化命令为：fdformat

为了能和 DOS 系统格式兼容，在 UNIX 系统中也可以将软盘格式化为 PCFS 文

件格式。

pcfs 格式化命令为：fdformat -d

从驱动器中取出软盘可以用命令 eject，表示默认的设备是/dev/diskette。

4.4.2　软盘操作命令

同磁带驱动器一样，命令 tar、cpio、pax 也可用于软盘。具体使用见磁带管理一节。

在软盘格式化之后，可以使用命令 mount 将软盘加载到操作系统的根文件系统上。

命令格式：mount devname mountpoint

其中 devname 为软盘驱动器逻辑名；mountpoint 为根文件系统中的一个空目录，该目录是软盘要加载上去的地方。

将软盘加载到根文件系统上之后，对软盘可以使用文件系统的命令，如 cp、ls、rm。使用完软盘后，在将软盘从驱动器中拿出之前必须卸载，卸载命令为 eject。

例　将软盘安装到目录/mnt 上。

```
$mount /dev/rdiskette /mnt
```

将文件/home/liuxm/filetest 复制到软盘：

```
$cp /home/liuxm/filetest /mnt/filetest
```

卸载软盘：

```
$eject
```

如果要长期使用软盘，可以配置文件/etc/vfstab（各系统该文件名称不同，详见表 2.2），配置后只要系统开机引导软盘就已经加载。配置文件/etc/vfstab 的方法是在文件/etc/vfstab 中加入一行：

```
#cat vfstab
#device        device       mount     FS      fsck     mount     mount
#to mount      to fsck      point     type    pass     at boot   options
#
/dev/rdiskette     -            /mnt      ufs      no       yes       -
```

如果选择了 mount at boot 这项，则表示系统引导时自动生效。

4.5　CD-ROM 管理与卷管理

CD-ROM 是一个只读的外部存储设备，对 CD-ROM 的管理方式与磁带和软盘不同。通常会把 CD-ROM 当作文件系统管理，即以加载、卸载方式使用。另外，卷

管理方式主要就是针对 CD-ROM 的使用提出的。

对于许多刚开始接触 UNIX 系统的用户来说，用加载方式使用存储设备比较棘手。为了避免反复加载、卸载的麻烦，在有的 UNIX 系统中引入了卷管理。

在 Solaris 系统中可以用卷管理自动加载 CD-ROM 和软盘。卷管理方式不仅限于系统管理员，一般用户也可以使用，同时远程用户也可以使用加载到远程系统上的 CD-ROM 和软盘。

卷管理依赖系统后台运行的卷管理程序/usr/sbin/vold，卷管理进程 vold 用来确定设备的配置文件/etc/vold.conf，并将卷管理消息写入文件/var/adm/vold.log 中。如果卷管理出现问题，可以查看日志文件/var/adm/vold.log。

1. CD-ROM 与卷管理

如果系统中有两个以上的 CD-ROM 驱动器，则卷管理自动在/vol/dev 目录中为每个额外的驱动器创建多个子目录，其中只有一个以文件系统访问方式进行，其余则以设备管理方式进行。下面分别以本地和远程两种方式介绍 CD-ROM 的卷加载使用。

- 本地使用 CD-ROM

将 CD-ROM 插入驱动器，卷管理自动将 CD-ROM 文件系统加载到/cdrom 加载点。这样对 CD-ROM 的访问就可以用文件系统命令，如 cd、ls、cp 等。使用完 CD-ROM 后，用 eject cdrom 命令弹出 CD-ROM。

- 远程使用 CD-ROM

一台 UNIX 主机可以通过网络使用另一台 UNIX 主机的 CD-ROM，这种方式称为远程使用 CD-ROM。远程使用 CD-ROM 的步骤有：

(1) 在远程具有 CD-ROM 的系统上（CD-ROM 提供者，远程系统）运行后台进程 mounted，并将自己的 CD-ROM 设为共享。

用命令：$ps –ef | grep mounted 可以查看该进程是否启动，已在后台运行。

如果进程 mounted 正在后台运行，则以超级用户身份输入命令：share –F nfs –o ro /cdrom/cdrom0。

如果进程 mounted 不在后台运行，则用下列两种方法启动进程 mounted：

a. 作为超级用户输入命令：/etc/init.d/nfs.server start0

确认 mounted 进程启动后再输入命令：share –F nfs –o ro /cdrom/cdrom0

b. 在文件/etc/dfs/dfstab 中加入两行：

/etc/init.d/nfs.server

Share –F nfs –o ro /cdrom/cdrom0

a 方式在系统重新引导后不生效；b 方式不受系统引导影响，长期生效。

(2) 在本地系统（要访问 CD-ROM 的系统）中输入命令：

mount remote-system-name(or IP-addr):/cdrom/cdrom0 mount-point

其中 remote-system-name(or IP-addr)为远程主机名或 IP 地址；mount-point 为本地加载点（通常是/mnt）。

上述步骤完成后在本地系统上就可以使用远程系统的 CD-ROM，在本地加载目录下，如/mnt，可以用文件系统操作命令对远程 CD-ROM 进行操作。

如果不再使用远程的 CD-ROM，就要取消前面加上的操作。远程主机上用 unshare 命令，如果是修改的文件，则删除加入的两行。本地系统卸载用 unmount mount-point。

2. 软盘与卷管理

用卷管理方式使用软盘与 CD-ROM 相似，只是软盘的加载不是自动加载，而是用命令 volcheck 加载，软盘的加载点是/floppy。加载后可以用文件系统操作命令对软盘进行操作。

如果不再使用软盘，则用命令 eject 卸载。

例　软盘没有加载时。

```
$cd /floppy
$ls
mcoper
$
```

软盘加载后：

```
$volcheck
$cd /floppy
$ls
mcoper      unnamed_floppy
$cd unnamed_floppy
$ls
lost+found
$cp /home/liuxm/nettask/test1
$ls
test1    lost+found
$
```

卸载软盘：

```
$eject
```

$

4.6 硬 盘 管 理

硬盘是系统最重要的存取设备，如果 UNIX 系统的主硬盘发生故障，则可能引起系统崩溃。本节介绍的硬盘管理主要针对的是 Solaris 系统环境。

4.6.1 硬盘命名

在 UNIX 系统中硬盘同样也使用逻辑设备名，硬盘同软盘一样，有块设备类型和字符设备类型。在 Solaris 系统中块类型的设备管理文件目录名为/dev/dsk，字符类型的设备管理文件目录名为/dev/rdsk。

通过总线与系统连接的硬盘命名为 cMtNdXsY。其中 M 为逻辑控制器号，N 为物理总线目标号，X 为驱动器号，Y 为盘片或分区号（通常为 0~7）。

例　总线连接控制器块类型硬盘设备名。

第一个控制器的第一个 SCSI 目标地址的第一个硬盘的第一个盘片的块设备接口为：/dev/rdsk/c0t0d0s0。

第一个控制器的第一个 SCSI 目标地址的第一个硬盘的第二个盘片的块设备接口为：/dev/rdsk/c0t0d0s1。

第一个控制器的第一个 SCSI 目标地址的第一个硬盘的第三个盘片的块设备接口为：/dev/rdsk/c0t0d0s2。

第一个控制器的第一个 SCSI 目标地址的第二个硬盘的第一个盘片的块设备接口为：/dev/rdsk/c0t0d1s0。

第一个控制器的第二个 SCSI 目标地址的第二个硬盘的第一个盘片的块设备接口为：/dev/rdsk/c0t1d1s0。

直接与系统连接的硬盘命名为 cMdXsY。其中 M 为逻辑总线目标号，X 为驱动器号，Y 为盘片或分区号（通常为 0~7）。

例　直接连接控制器块类型硬盘设备名。

第一个控制器的第一个硬盘的第一个盘片的块设备接口为：/dev/rdsk/c0d0s0。

第一个控制器的第一个硬盘的第二个盘片的块设备接口为：/dev/rdsk/c0d0s1。

第一个控制器的第一个硬盘的第三个盘片的块设备接口为：/dev/rdsk/c0d0s2。

第一个控制器的第二个硬盘的第一个盘片的块设备接口为：/dev/rdsk/c0d1s0。

第一个控制器的第二个硬盘的第二个盘片的块设备接口为：/dev/rdsk/c0d1s1。

4.6.2　硬盘片

硬盘盘片是硬盘的基本单位，操作系统将盘片作为单独的硬盘驱动器。文件存储在文件系统中，每个盘片只存储一个文件系统，文件系统不能跨越多个盘片。在设置硬盘文件系统时，根据系统配置和安装到系统的软件的需要，既要选择盘片的大小又要选择使用的盘片。

CPU 不同，盘片的设置方式也不同。比如 CPU 是 SPARC 的系统，整个硬盘用于 Solaris 系统，其中：

0 号盘片用于/文件系统，包含构造操作系统的文件系统和目录；

1 号盘片用于 swap 文件系统，提供虚拟内存或者交换空间；

2 号盘片上没有文件系统；

3 号盘片用于/export，包含可选的客户系统需求的操作系统版本；

4 号盘片用于/export/swap 文件系统，提供客户系统虚拟内存空间或者交换空间；

5 号盘片用于/opt 文件系统，包含用户的应用软件；如果安装时没有给该文件系统分配盘片，则/opt 目录放在盘片 0 中；

6 号盘片用于文件系统/usr，包含用户执行的操作系统命令、文件、系统程序（如 init 和 syslogd）、库例程序；

7 号盘片用于文件系统/home 或/export/home，包含用户帐号创建的文件及用户家目录。

不同应用的盘片设置方式可以不同，但 0 号盘片、1 号盘片分别用于/和/swap 不变。

4.6.3　测试硬盘

在指定盘片分配空间后，超级用户可以用命令 prtvtoc 显示硬盘分区信息。

命令格式：#prtvtoc /dev/rdsk/cntndnsn

例　# prtvtoc /dev/rdsk/c0t0d0s2

```
* /dev/rdsk/c0t0d0s2 partition map
*
* Dimensions:
*          512 bytes/sector
*           35 sectors/track
*            6 tracks/cylinder
*          210 sectors/cylinder
*         1019 cylinders
```

```
*          970 accessible cylinders
*
* Flags:
*       1: unmountable
*      10: read-only
*


*               First
* Partition Tag Flags Sector
     0          0     00          0
     1          0     00      24150
     2          0     00          0
     6          0     00      74550
Sertor Last
Count Sector Mount Directory
24150    24149
50400    74549
204540 204539    /
12990 204539
#
```

4.6.4 硬盘复制

预防硬盘损坏的方法之一就是硬盘复制。用 dd 命令可以实现硬盘复制。

复制硬盘必须在源盘和目的盘几何特性完全相同的情况下，用超级用户身份才能进行。步骤为：

(1) 为了在重新启动时发现复制盘，有主盘的系统需要/reconfigure 文件，输入 touch /reconfigure 并回车；

(2) 关闭系统：init 0；

(3) 将复制盘连接到系统并开机，在 ok 提示符下用 boot 引导系统；

(4) 输入 dd if=/dev/dsk/device-name of=/dev/dsk/device-name bs=blocksize 并回车，其中输入文件 if 是主盘设备，输出文件 of 是复制盘设备；

(5) 检测新的文件系统：输入 fsck /dev/rdsk/device-name 并回车；

(6) 加载复制盘的 root 文件系统 输入 mount /dev/rdsk/device-name/mnt 并回车；

(7) 指向正确的设备名：编辑复制盘上的/etc/vfstab 文件；

(8) 卸载复制盘的 root 文件系统：输入 umount /mnt 并回车；

(9) 输入 init 0 关闭系统；

(10) 以单用户模式引导系统：输入 boot diskn-s 并回车；

(11) 输入 sys-unconfig 并回车，恢复复制盘配置，在该盘配置恢复后关闭系统；

(12) 输入 boot diskn 并回车，引导复制盘；

(13) 提供相应的系统信息，如主机名、系统时区、网络的 IP 地址、掩码等；

(14) 引导完成，用超级用户登录验证系统。

例　硬盘复制。

首先用系统光盘进入超级用户：

```
#touch /reconfigure
#init 0
ok PROM:boot
#dd if=/dev/dsk/c0t0d0s0 of=/dev/dsk/c0t1d0s2 bs=100k
#fsck /dev/dsk/c0t1d0s2
#mount /dev/dsk/c0t1d0s2 /mnt
#cd /mnt/etc
#vi vfstab
```

（为新盘改入口）

```
#cd /
#umount /mnt
#init 0
ok PROM:boot disk1 –s
#sys-unconfig
#init 0
ok PROM:boot disk2
```

在系统引导过程中输入系统参数。引导完成后以超级用户身份进入系统验证新硬盘是否正常。

4.7　打印机管理

打印机将输出数据打印在打印纸上保存，打印机是计算机中非常重要的外围设备。同其他操作系统一样，UNIX 系统管理也包括对打印机的管理。UNIX 系统支持的打印机有针打印机、喷墨打印机、激光打印机。

4.7.1 安装打印机

按打印机与系统的连接方式分并行打印机和串行打印机。

1. 将打印机与主机端口相连

首先将主机的串口与并口打开，用命令：

#mkdev serial 或#mkdev parallel

分别将主机的串口与并口配置到系统。

然后：

- 如果是串行打印机，则通过 RS-232 电缆把主机的串口与打印机相连，并输入命令：

#disable /dev/ttyxx

禁止连着打印机的主机串口有其他登录活动。其中 ttyxx 表示连打印机的主机串口号。

- 如果是并行打印机，使用标准的接口电缆把主机的并行端口与打印机相连。

最后：测试打印机连接是否正确：

- 如果是串行打印机连接到主机的串口 COM1，用命令：

#date>/dev/tty1a

- 如果是并行打印机连接到主机的并口 LPT1，用命令：

#date>/dev/lp0

如果打印机连接正确，就会在接受命令后打印出当前日期。

2. 添加打印机到打印队列

用 lpadmin 命令将打印机添加到打印队列。

- 用命令：lpadmin –p printer-name –v /dev/null 定义打印机名和打印机使用的端口设备。/dev/null 为默认设备名。
- 用命令：lpadmin –p printer-name –i /usr/lib/lp/model/netstandard 定义打印机使用的接口脚本。
- 用命令：lpadmin –p printer-name –o dest=access-name:port –o protocol=protocol –o timeout=value 定义打印机目的地、协议和超时值。
- 用命令：lpadmin –p printer-name –I content-type –T printer-type 指定文件内容类型和打印机类型。

3. 打印机安装好后就可以使用 lp 命令发送打印作业

大多数 UNIX 系统可以利用系统的管理工具，如 Solaris 的 Admintool 图形管理工具和 AIX 的 SMIT 对打印机进行设置和管理。

4.7.2　LP 打印服务管理

在 UNIX 系统中采用假脱机 Spooling 技术来管理打印作业。用户打印文件的请求被送到打印作业队列中，打印按先后顺序进行。

LP 打印服务依靠后台进程 lpsched 获得打印机的设置和配置信息，更新 LP 系统文件。除 enbable 和 disable 在/usr/bin 和/usr/lib/lp/local 中外，LP 的所有命令都从/usr/bin 目录移到/usr/sbin 目录中与 accept 和 reject 命令符号连接。不管用户请求来自于应用程序还是命令，都由 lpsched 后台程序规划所有的本地打印请求。一个打印客户和服务器只能有一个 lpsched 进程运行，该进程由控制脚本/etc/rc2.d/s80lp 在系统引导时产生，用命令/usr/sbin/lpshut 停止进程运行，也可以使用/usr/sbin/lpsched 命令重新启动。

现在 UNIX 的打印管理程序与网络客户/服务器模式完全匹配，打印管理软件中既有打印服务程序又有客户应用程序，在服务器上既要安装打印提供服务又要安装客户服务。

Solaris 版本中增加的网络接口脚本/usr/lib/lp/model/netstandard 专为网络打印机设计，收集执行网络打印所需的打印缓冲程序和打印数据库信息，并将这些信息送到打印输出模块；从 netstandard 接口脚本调用新的打印输出模块 netpr 打印作业，建立与打印机的网络连接。为了满足网络打印服务的需求，在每个打印客户和打印服务器中必须至少有一个 lpNet 后台进程，该进程在系统引导时启动。如果 lpNet 进程受 lpsched 和 lpshut 进程影响停止了，必须重新启动。

4.7.3　打印管理与维护

对打印机需要做的管理和维护工作主要有：

- 设置系统的默认打印机；
- 开启和关闭打印机；
- 启动和停止打印服务；
- 发送、拒绝、转移和注销打印请求；
- 查看打印机状态。

除了前面介绍的各系统专用的图形管理工具之外，命令 lpadmin 也可用于管理和维护打印机，特别是用于网络打印服务器。设置网络打印服务器通常包括设置打印机名称、类型、文件内容类型、服务器名称、网络打印访问者（有时通过端口名指定）、打印机的 IP 地址、网络协议和打印超时值（通常默认为 10 s）、打印服务器故障通知策略等。

例　设置一个打印服务器，打印机名为：littledog，网络打印机访问名为：

Epson3100；网络访问协议为 TCP，超时值为 10，接口为/usr/lib/lp/model/netstandard，打印机类型为 PS，内容类型为 postscript，设备为/dev/null。

将打印机连接到网络，超级用户登录服务器后完成以下操作：

```
#lpadmin –p littledog –v /dev/null
#lpadmin –p littledog –i /usr/lib/lp/model/netstandard
#lpadmin –p littledog –o dest:Epson3100 –o protocol=tcp –o timeout=5
#lpadmin –p littledog –I postscript –T PS
#cd /etc/lp/fd
#for filter in *.fd; do
>name=`basename $ filter.fd`
>lpfilter –f $ name – F $ filter
>done
#accept castle
```

4.7.4　打印机使用

1. lp 命令

在 UNIX 系统中通过命令 lp 使用打印机。命令格式：

lp [cmw] [-d dest] [-n number] [-t title] filename

其中：

c 表示复制一份打印文件到 spool 中，再打印复制文件；

m 表示打印完毕后用 E-mail 方式通知用户；

w 表示文件打印后在用户终端上显示打印信息。若用户这时未登录系统，还可以发出通知；

d dest 指定由特定打印机打印，dest 为打印机名或类别，如果为类别，则为此类打印机中的第一台，若省略 d 选项，则以系统变量 LPDEST=printname 设定的名称 printname 为准；

t title 将标题打印在报表的标题页上；

n number 为打印份数；

filename 为要打印的文件名称。

例　打印文件 file1、file2、file3 各 2 份。

$lp –n 2 file1 file2 file3

用激光打印机打印文件 file，共 5 份，打印完后用 E-mail 通知用户。

$lp –m –n 5 d laser file

2. pr 命令

用 pr 打印会将文件分割成页打印。命令格式：

pr [+page] [-dntm] [-h title] [file…]

其中：

+page 为整数表示的页码，表示从该页开始打印。默认值为 1；

d 表示可以产生双行空白；

t 表示可以省略最前面和最后面的额外空白行；

n 表示在每行前面加上行号；

h 用来描述表头，取代每页最前面的文件名称；

m 和并所有输入文件并打印，每个输入文件的数据占一个列，以数据在文件中的次序排列。

例　打印文件 stdio.h，并附上标题“ STDIO File ”。

$pr –h “STDIO File” stdio.h

4.8　设备管理中的“ 流 ”机制

为了提高 I/O 子系统的模块性和灵活性，Ritchie 提出了“ 流 ”机制。“ 流 ”是一组线性的、全双工的连接进程与设备驱动程序的队列。每个队列中的一个成员用于输入、另一个成员用于输出。当一个设备驱动程序从设备上接收到数据后，“ 流 ”将输入队列的数据送给一个正在读的进程；同样，当一个写进程向“ 流 ”的输出队列写数据时，“ 流 ”从输出队列中将数据送给设备驱动程序。

由于不同的设备驱动程序常常具有一些相同的功能，若干个用户命令可能在不同的介质上完成公共的逻辑功能。“ 流 ”的每个队列与某内核模块的一个实例相联系，如各种网络通信协议模块等，从而克服了设备驱动程序功能性重复的缺点，实现了同一个设备驱动程序的不同逻辑功能分开。

“ 流 ”的每个队列所包含的数据结构有：

- 执行系统调用 open 的一个打开过程；
- 执行系统调用 close 的一个关闭过程；
- 将消息放入流队列中的“ 放入 ”过程；
- 流队列被调度执行时的“ 服务 ”过程；
- 指向“ 流 ”队列中下一个的指针；
- 指向等待服务的消息表的指针；
- 指向维护流队列状态的私用数据结构的指针；

- 用于维护流队列状态的标志位及调度和控制流量高、低水准线的数据。

一个具有流驱动程序的设备是一个字符设备，在字符设备开关表中有一个特殊字段，该字段指向流初始化结构。当内核执行系统调用 open 时，检查字符设备开关表对应的这一个字段，如果在字符设备开关表中没有发现入口，则该设备驱动程序不是一个流驱动程序，内核执行通常的字符设备过程。如果发现该设备在字符设备开关表中有一个入口，则打开一个流驱动程序，并由内核分配两个队列对：一个队列作为流的头标，另一个给驱动程序使用。这样，当其他进程打开此设备时，内核可以通过索引节点指针找到先前分配的流，并调用该流上所有模块的打开过程。模块间通信靠在一个流上向相邻模块传递消息来实现。当某进程向一个流写数据时，内核将数据从用户空间拷贝到该流头标的消息块中，流头标模块调用下一个队列模块的放入过程，该过程处理该消息后立即传递给下一个队列。

在流实现中存在的一些现象：

- 流是线性地连接，不容易在内核中产生多路复用；
- 在流实现时，由于模块不必在使用流的进程的上下文中运行，也并非所有的进程均匀分担流模块的执行开销，所以记录进程日志比较困难；
- 如果没有数据，系统会自动使终端进入原始状态，而引入流后则违反了这一现象。

由此可见，只有通过不断地改进流才能更好地满足系统需要。

4.9 本 章 小 结

每个 UNIX 主机一般都会有一个或者多个设备，为了保证系统正常运行，需要对这些设备进行管理配置。

UNIX 系统采用了通过文件子系统的方式管理设备，每个设备都是以逻辑设备名的方式出现，这是 UNIX 操作系统独有的特色。用户在使用设备时，回避了繁琐、复杂的设备物理名称和物理特征，只需将逻辑设备名用于相应的 Shell 命令即可。

磁盘、磁带和 CD-ROM，这些设备可以通过加载的方式挂到文件系统目录结构上，用文件系统命令就能操作这些设备。除了加载方式之外，有的操作系统还引入了卷管理，卷管理带来更大的方便：直接用文件系统命令操作设备。

UNIX 系统在设备管理上较为先进的思想是“流”。本章最后介绍了“流”的概念及其实现。

上 机 练 习

1. 熟悉各种设备的逻辑设备名。
1. 练习终端管理命令。
2. 练习磁带管理命令。
3. 练习软盘管理命令。
4. 练习 CD-ROM 管理命令。
5. 练习硬盘管理命令。
6. 练习打印机管理命令。

习　题　四

1. UNIX 系统设备管理完成哪些功能？与其他操作系统比较，UNIX 系统设备管理有何不同？
2. 请说明 UNIX 系统对设备分类管理的意义及实现过程。
3. 在 UNIX 系统中 I/O 的有缓冲区管理主要解决什么问题？
4. 请描述 UNIX 系统中对磁盘写操作提供的基本支持功能。

第 5 章　Shell 及其编程

在 UNIX 中 Shell 是操作系统外壳。它不但是用户与操作系统交互的命令解释器，还是一个具有环境个性化的程序设计语言。

本章主要内容

- Shell 概述：主要介绍 Shell 的主要功能及 B-Shell、C-Shell、K-Shell 的特征。
- Shell 脚本：介绍怎样形成可执行的 Shell 脚本。
- Shell 脚本变量：Shell 脚本变量包括环境变量、特殊变量、用户自定义变量。
- Shell 控制结构：主要介绍控制 Shell 脚本结构的语句：if then else、case、for、while、until、break 和 continue。
- Shell 函数：Shell 函数定义及脚本中调用 Shell 函数。
- Shell 工具：包括通知 trap、信息文件（临时文件、日志文件）的创建及 logger。
- Shell 脚本编程应用实例：介绍一些典型的 Shell 脚本实例。

5.1　Shell　概　述

Shell 作为 UNIX 操作系统的外壳，具有的主要功能有：

- 命令解释功能：将用户可读的命令转换成计算机可理解的命令，并控制命令执行。
- 输入/输出重定向：UNIX 系统将键盘作为标准输入、显示器作为标准输出。在默认情况下，Shell 将命令的输入流定向为键盘、输出流（包括错误输出流）定向为显示器。当这些定向不能满足用户需求时，用户可以在命令中用符号“ > ”或“ < ”重新定向，如定向到文件。
- 管道线处理：利用管道将一个命令的输出送入另一个命令，实现多个命令组合完成复杂命令的功能。
- 系统环境设置：用 Shell 命令设置环境变量，维护用户的工作环境。
- 程序设计语言：Shell 命令本身可以作为程序设计语言，将多个 Shell 命令组合起来，编写能实现系统或用户所需功能的程序。

在 UNIX 中系统主要提供三种 Shell：Bourne Shell、C Shell 和 Korn Shell。

5.1.1　Bourne Shell

Bourne Shell 由 Steve Bourne 在贝尔实验室开发，是大多数 UNIX 操作系统默认的 Shell，系统提示符为“ $ ”，简称为 B-Shell。

如果在用户管理文件/etc/passwd 中将用户的 Shell 设置成 sh，用户的默认 Shell 就是 B-Shell，即用户登录进入的是 B-Shell。如果用户在 C-Shell、K-Shell 环境下输入命令 sh 也可以切换到 B-Shell。从 Shell 退出则键入“ exit ”。

例　下面的/etc/passwd 中用户的默认 Shell 被设置成 B-Shell。

```
$cat /etc/passwd
root:!:0:0::/:/usr/bin/sh
daemon:!:1:1::/etc:
bin:!:2:2::/bin:
sys:!:3:3::/usr/sys:
adm:!:4:4::/var/adm:
uucp:!:5:5::/usr/lib/uucp:
blademan:!:246:1::/home/blademan:/usr/bin/sh
lmy:!:249:1::/home/lmy:/usr/bin/sh
xwen:!:251:1::/home/xwen:/usr/bin/sh
imnadm:*:254:201::/home/imnadm:/usr/bin/sh
ldap:*:255:1::/home/ldap:/usr/bin/sh
duj:!:415:1::/home/duj:/usr/bin/sh
……
$
```

例　Shell 切换：B-Shell 切换为 C-Shell，C-Shell 切换为 B-Shell，exit 退出 Shell。

```
$
$csh
%sh
$exit
%exit
%$
$
```


与 B-Shell 相对应的用户环境设置文件（用户主目录中的隐含文件）为.profile。该隐含文件中包含有用户的环境变量设置语句。语句格式为：

```
$name＝value;　export $name
```

其中 name 为环境变量名；value 为设置的环境变量值。

该语句也可以直接在 Shell 提示符下用命令输入作为临时环境变量设置，退出 Shell 后不再生效。

例 $cat .profile

以下是默认的/etc/skel/local.profile 文件：

```
# @ (#) local.profile 1.4          03/09/15    SMI
stty - istrip
PATH=/usr/bin: /usr/ucb: /etc: .
Export PATH
#
# if possible, start the windows system.t
#
if ('tty'=="/dev/console") then
   if ("$TERM"=="sun" -o    "$TERM" == "AT386" ) ; then
      if [$ {OPENWINHOME: - " "} = " " ] then
         OPENWINHOME =/usr/openwin
      export OPENWINHOME
   fi
   echo " "echo "Starting OpenWindows in 5 second (type Control -C to intrrupt) "sleep 5
   echo " "$ OPENWINHOME/bin/openwin
   clear                  # get rid of annoying cursor rectangle
   exit                 #logout after leaving windows system
fi
$
```

例 直接用命令设置终端类型为 vt100。

显示终端变量 TERM 的值：

```
$echo $TERM
TERM
$
```

设置终端变量 TERM 的值：

```
$TERM=vt100; export $TERM        #
$
```

显示终端变量 TERM 的值：

```
$echo $TERM
vt100
```

```
$
```

5.1.2　C Shell

C Shell 是 Bill Joy 在 Berkeley 开发的，因其在程序语言结构上和 C 相似，故称为 C Shell（或 C-Shell），系统提示符为“ % ”。如果在用户管理文件/etc/passwd 中将用户的 Shell 设置成 csh，用户的默认 Shell（登录进入）就是 C-Shell。在其他 Shell 环境下通过输入命令 csh 可以切换到 C-Shell。当用户在/etc/passwd 文件中将 Shell 设置为 C-Shell 后，在用户主目录下将存在两个隐含文件.cshrc 和.login，它们是用户环境配置文件。文件.login 中的命令在用户登录时自动执行，文件.cshrc 中的命令在登录后执行。

C-Shell 使用 set 命令设置变量的值（set 命令后无参数则显示当前 C Shell 定义的所有变量），格式为：

```
set name = value
```

其中 name 为变量名；value 为变量值。

例　%set TERM=vt100

与其他 Shell 比较，C-Shell 更适合交互式环境。C-Shell 最突出的特征是有历史（history）命令。历史命令用于存储最近使用的命令。用户可以定义存储历史命令的数量、显示历史命令、编辑并重新使用编辑后的命令。history 可以用 Shell 命令临时设置，也可以在.cshrc 文件中加入语句设置。

例　设置 C Shell 的 history。

```
%set history=4
%cd
%pwd
/home/zhouxx
%cd zhouxx
%history
21 set history=5
22 cd
23 pwd
24 cd zhouxx
%
```

输入! !则重复前一条命令：


```
% !!
history
```

```
21 set history=5
22 cd
23 pwd
24 cd zhouxx
%
```

输入!$ 则重复前一条命令的最后一词。

例 %cd /home/zhouxx/showtest

```
%cp !$ passtest
cp /home/zhouxx/showtest passtest
%
```

输入!n 重复执行编了号的命令。

例 %!23

```
/home/zhouxx
%
```

输入!n s/oldstring/newstring 编辑 history 中不正确的命令：

```
%history
21 set history=5
22 cd
23 pwd
24 cd zhouxx
25 ping www.sine.com
26 tar cvf /dev/rmt/0 seht
27 telnet ftp.scu.edu.cn
%!25 s/e/a
```

注意 有的 UNIX 系统未设置该项功能。在 Linux 上也无此功能。

另外，C Shell 还有一个特别的用途，就是可以将新的 Shell 命令归到默认 Shell 路径下。方法是只需在使用新命令之前先用命令：rehash。

例 现有新编 Shell 命令 netsend。

```
%netsend
netsend:Commans not found
%rehash
%netsend
%
```

5.1.3　Korn Shell

Korn Shell（简称 K-Shell）由贝尔实验室的 David Korn 开发。Korn Shell 和 Bourne Shell 有许多兼容的地方，如系统提示符也是%，环境变量的设置格式完全相同，不同的只是 Korn Shell 有些内置功能可以从 Shell 直接定义。与 C Shell 相比，Korn Shell 除了有历史命令之外，还有更好的命令编辑形式。

Korn Shell 的用户环境初始化文件是.profile 和.ksh-env（又可以命名为 kshrc）。文件.profile 和.ksh-env 在功能上分别相当于 C Shell 中的.login 和.cshrc。环境变量在文件.profile 中定义。

例　显示环境变量用命令 set –o，启用选项用命令 set –o optionname，禁止选项用命令 set +o optionname，这些变量在文件.ksh-env 中设置。

```
$set -o
Current option settings are:
allexport          off
bgnice             on
emacs              off
errexit            off
gmacs              off
ignoreeof          off
interactive        on
keyword            off
markdirs           off
monitor            on
noexec             off
noclobber          off
noglob             off
nolog              off
notify             off
nounset            off
privileged         off
restricted         off
trackall           off
verbose            off
vi                 off
```

```
viraw                off
xtrace               off
$
```

与 C-Shell 一样，K-Shell 也有历史文件，默认存放在用户家目录下的.sh_histoyr 文件中，用法相同。

5.2 Shell 脚 本

Shell 具有程序设计功能，如果在使用系统时需要用重复、复杂的命令完成某项工作，就可以利用 Shell 编程实现。这样用户在定义自己命令的同时也实现了系统 Shell 命令的扩充。同一般的结构化程序不同，Shell 程序是按行解释，不需要编译成目标程序，也不需要连接成可执行的目标码，只要 Shell 脚本是可执行程序就可以直接运行。

由 Shell 命令构成并由 Shell 执行的命令文件称为 Shell 脚本（Shell Script）。从广义上看，任何从键盘上键入的 Shell 命令都是 Shell 脚本。

Shell 脚本由编辑器 vi 创建。为了使脚本便于理解，以#为注释符，#之后直到行末的字符串为注释。脚本中忽略 Tab 键和空格，内容过长超过一行时用反斜线\表示换行继续。脚本文件属性必须是可执行的才可以运行，否则要通过命令 chmod 将脚本文件属性设置为可执行。

执行 Shell 脚本有三种方式。

方式 1：直接键入 Shell 脚本文件名，这种运行方式表示启动新的 Shell 执行该脚本。

方式 2：sh Shell 脚本文件名，这种运行方式表示启动新的 Shell 执行该脚本。

方式 3：. Shell 脚本文件名，这种运行方式表示在原 Shell 下执行该脚本。

注意 有时运行脚本文件时 UNIX 系统不能辨识当前路径，用户只有输入文件的全路径名或用./文件名的方式告诉 Shell 到当前目录下寻找文件名。

例 shell 脚本 jobbegin 及其运行。

```
$cat jobbegin
# My job begin every day
date
cd /home/liuxim/project
echo 'my working directory is: '
pwd
# script end
```

```
$
```

将该脚本文件属性设置为可执行后，就可以运行了。

用方式 1 运行：直接键入脚本文件名运行。

首先确定当前的工作目录：

```
$pwd
/home/liuxim
$
```

直接输入脚本名运行脚本文件：

```
$jobbegin
Fri Mar 28 17:09:31 TAIST 2003
my working directory is:
/home/liuxim/project
$
```

运行完后再确认当前工作目录：

```
$pwd
/home/liuxim
$
```

从脚本执行可见，脚本结束前进入了目录/home/liuxun/project。但脚本运行完后用 pwd 命令却显示当前工作目录是/home/liuxun。之所以出现这样的情况是因为脚本执行启动了一个新 Shell 进程，脚本执行完后立即结束该新 Shell 进程，又回到原主 Shell，所以最后的 pwd 命令得到的仍然是原工作目录。

用方式 2 运行：键入 sh 脚本文件名。

```
$pwd
/home/liuxim
$
$jobbegin
Fri Mar 28 17:09:31 TAIST 2003
my working directory is:
/home/liuxim/project
$
$pwd
/home/liuxim
```



```
$
```

结果和方式 1 相同。

用方式 3 运行：键入脚本文件名。

```
$pwd
/home/liuxim
$
$. jobbegin
Fri Mar 28 17:09:31 TAIST 2003
my working directory is:
/home/liuxim/project
$
$pwd
/home/liuxim/project
$
```

脚本执行结果与最后用 pwd 命令结果所显示的目录均是/home/liuxun/project。之所以这样是因为脚本执行没有启动新 Shell 进程，仍然在原 Shell 下。

具体应用中用户可以根据自己的需要选择一种运行方式。

5.3 Shell 脚本变量

同其他程序设计语言一样，在 Shell 脚本中也可以使用变量。通常将 Shell 变量分为环境变量和临时变量。临时变量的值随着 Shell 程序的结束会自然消失。临时变量又可分为系统特殊变量和用户自定义变量。下面分别介绍环境变量、特殊变量和用户自定义变量。

5.3.1 环境变量

在系统中变量值保持不变，不随 Shell 程序的结束而消失的变量称为环境变量。环境变量在系统配置文件中设置，用大写字母表示。环境变量决定了用户的工作环境。常用的环境变量有 PATH、TERM、HOME、PS1、PS2、LOGNAME。

- PATH 变量：用来设定指令搜索路径。当用户键入命令时，shell 会在 PATH 变量所设定的路径下从左至右寻找。PATH 内含有一串 Shell 所要查找的路径名称，用冒号隔开。

例 $echo $PATH

```
/usr/bin: /bin: /usr/ucb: /etc:
$
```


变量的值可以通过命令修改。

- TERM 变量：用来设置用户使用的终端类型(系统主控台 console 不用设定)。如果用户通过网络登录到 UNIX 系统，则应将 TERM 值设置成与系统相应终端类型一致。

例　在 Windows 中用 telnet 登录到 UNIX 时需将终端配置成 vt100。

```
$TERM=vt100; export TERM
$echo $TERM
vt100
$
```

- HOME 变量：用来设定用户的默认目录（即用户根目录或家目录）。用户登录系统时默认进入该目录。

例　$echo $HOME

```
/home/liuxim
$
```

- PS1 变量：设置系统提示符，默认为#（超级用户）、$（B-shell 和 K-Shell）、%（C-Shell）。

例　$echo $PS1

```
$
$PS1=/home/liuxim; export PS1
/home/liuxim
```

- PS2 变量：设置系统的次提示符，默认为>。有的 UNIX 系统不支持该变量。

例　$bc

```
>112 + 34
>146
>(345-11)*2
>668
>Ctrl+d
$
```

用命令 env 可以显示目前所有的环境变量。

例　$env

```
_=/usr/bin/env
LANG=en_US
LOGIN=liuxim
```



```
IMQCONFIGCL=/etc/IMNSearch/dbcshelp
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
```

```
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin
LC__FASTMSG=true
IMQCONFIGSRV=/etc/IMNSearch
CGI_DIRECTORY=/var/docsearch/cgi-bin
LOGNAME=liuxim
MAIL=/usr/spool/mail/liuxim
LOCPATH=/usr/lib/nls/loc
DOCUMENT_SERVER_MACHINE_NAME=localhost
USER=liuxim
AUTHSTATE=compat
SHELL=/usr/bin/ksh
ODMDIR=/etc/objrepos
DOCUMENT_SERVER_PORT=49213
HOME=/home/liuxim
TERM=vt100
MAILMSG=[YOU HAVE NEW MAIL]
PWD=/home/liuxim
DOCUMENT_DIRECTORY=/usr/docsearch/html
TZ=TAIST-8TAIDT
A__z=! LOGNAME
$
```

5.3.2 特殊变量

Shell 定义了如下几种特殊变量。

- 位置参数$0，$1，$2，$3，$4，$5，$6，$7，$8，$9：执行 Shell 命令时用于存放命令及命令参数。

其中：$0 存放命令名，$1、$2、$3、$4、$5、$6、$7、$8、$9 从左至右分别存放命令参数。

命令 shift 用于移动位置参数，将$2 中的内容移至$1，$3 中的内容移至$2，$4 中的内容移至$3，依次类推，但位置参数$0 不参加移位。

shift 命令格式为：shift [n]

n 表示向左移动参数的个数，默认值为 1。

有的 UNIX 操作系统位置参数缓存区比较大，除存储以上 10 个位置参数外，还可以保留更多的位置参数，通过移位可以将它们移入前面的位置参数中。

例　在 AIX 系统中试验位置参数及 shift 命令，脚本为 movelab1。

```
$cat movelab1
## this program show shift
echo $0
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift
echo $1,$2,$3,$4,$5,$6,$7,$8,$9
$movelab1 11 22 33 44 55 66 77 88 99
movelab1
11,22,33,44,55,66,77,88,99
22,33,44,55,66,77,88,99,
33,44,55,66,77,88,99,,
44,55,66,77,88,99,,,
55,66,77,88,99,,,,
66,77,88,99,,,,,
77,88,99,,,,,,
88,99,,,,,,,
99,,,,,,,,
$
$cat movelab2
## this program show shift
echo $0
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift 2
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift 2
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift 2
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift 2
```



```
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift 2
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift 2
```

```
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift 2
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;shift 2
echo $1,$2,$3,$4,$5,$6,$7,$8,$9;
$movelab2 a b c d e f g h i j k l m n o p q
movelab2
a,b,c,d,e,f,g,h,i
c,d,e,f,g,h,i,j,k
e,f,g,h,i,j,k,l,m
g,h,i,j,k,l,m,n,o
i,j,k,l,m,n,o,p,q
k,l,m,n,o,p,q,,
m,n,o,p,q,,,,
o,p,q,,,,,,
q,,,,,,,,
$
```

- 与位置参数有关的特殊变量：

$#：能提供给脚本的位置参数个数；
$$：当前 Shell 脚本的进程号；
$!：上一个后台命令的进程号；
$?：执行最后一条命令的退出状态；
$*：位置参数字符串，所有位置参数字符串组成一个大的字符串；
$@：位置参数字符串，每个位置参数字符串都是一个单独的字符串。

例 $cat exer

```
echo $1,$2,$3,$4,$5,$6,$7,$8,$9
echo '$#='$#
echo '$$='$$
echo '$!='$!
echo '$?='$?
echo '$*='$*
echo '$@='$@
$exer 1 2 3 4 5 6 7 8 9 a bc
1,2,3,4,5,6,7,8,9
$#=11
$$=24056
```

```
$!=14361
$?=0
$*=1 2 3 4 5 6 7 8 9 a bc
$@=1 2 3 4 5 6 7 8 9 a bc
$
```

5.3.3　用户自定义变量

用户自定义变量是用户根据自己的需要在 Shell 中定义的临时变量。用户定义变量除数字之外还可以是字母、下划线和其他文字，长度不限。用户定义变量要区分大小写。

- 变量定义：用户自定义变量格式为 variable-name=value。

在定义等式中等号两端不允许有空格，执行时将右端的计算结果指定给左端。如果用户变量值中有空格，必须用引号将变量值括起来。用户变量的使用与环境变量一样，要在变量名前加符号“ $ ”，表示要使用该变量的值。用户定义变量非常灵活，需要改变时可以随时重新定义。

- 清除变量：如果所设置的变量不需要时可以清除。

格式为：unset variable-name

例　先定义 exer 的值为/home/liuxim/exer，然后清除 exer。

```
$exer=/home/liuxim/exer
$echo $exer
/home/liuxim/exer
$unset exer
$echo $exer
$
```

- 字符与变量组合：可以将字符与变量、变量与变量组合在一起使用。组合格式分别为 char$variable，$variable1$variable2。

例　$d=day

```
$echo "Sun$d, Mon$d, Tues$d, Wednes$d, Thurs$d, Fri$d, Satur$d"
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
$h=/home
$me=/liuxim/exer
$echo $h$me
/home/liuxim/exer
$
```

- 变量测试：使用变量时如果不能确定该变量是否已经定义，可以用变量测试语句：

{variable-name: -new-value}

表示如果没有定义则用新值（new-value）定义；如果已经定义则维持原定义。

例 首先设置 state 的值为 early。

```
$state=order
$echo "Everything was in a state of ${state:-disorder}"
Everything was in a state of order
$unset state
$echo "Everything was in a state of ${state:-disorder}"
Everything was in a state of disorder
$
```

- 命令替换：命令替换是将一个 Shell 命令的输出作为另一个命令的参数，还可以赋予变量。为了区别一般字符串，用倒单引号括起命令。

例 $echo "Now my work directory is 'pwd' "

```
Now my work directory is /home/liuxim
$time='date'
$echo $time
Sun Mar 30 09:39:28 TAIST 2003
```

- Shell 中的引号：双引号、单引号、倒单引号在 Shell 中有不同的意义。双引号内的内容如果包含$和单引号则保留 Shell 中的定义，包含倒单引号则作命令替换；单引号内的内容完全为普通字符串，不作命令替换；倒单引号内的内容只能是命令。

例 $echo "The present directory is 'pwd' "

```
The present directory is 'pwd'
$echo "The present directory is 'pwd' "
The present directory is /home/liuxim/project
$echo 'pwd'
/home/liuxim/project
$echo " 'date' "
Sun Mar 30 09:34:00 TAIST 2003
$
```

5.3.4　显示变量

用命令 set 可以显示所有变量，包括环境变量和临时变量。

例　$set

```
AUTHSTATE=compat
CGI_DIRECTORY=/var/docsearch/cgi-bin
DOCUMENT_DIRECTORY=/usr/docsearch/html
DOCUMENT_SERVER_MACHINE_NAME=localhost
DOCUMENT_SERVER_PORT=49213
ERRNO=10
FCEDIT=/usr/bin/ed
HOME=/home/liuxim
IFS=''
IMQCONFIGCL=/etc/IMNSearch/dbcshelp
IMQCONFIGSRV=/etc/IMNSearch
LANG=en_US
LC__FASTMSG=true
LINENO=1
LOCPATH=/usr/lib/nls/loc
LOGIN=liuxim
LOGNAME=liuxim
MAIL=/usr/spool/mail/liuxim
MAILCHECK=600
MAILMSG='[YOU HAVE NEW MAIL]'
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
ODMDIR=/etc/objrepos
OPTIND=1
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin
PPID=4612
PS1='$ '
PS2='> '
PS3='#? '
PS4='+ '
PWD=/home/liuxim
```

```
RANDOM=2030
SECONDS=656
SHELL=/usr/bin/k
TERM=vt100
TERM_DEFAULT=lft
TMOUT=0
TZ=TAIST-8TAIDT
USER=liuxim
_=echo
var1=11
var2=22
var3=33
var4=''
$
```

5.3.5 shell 输入/输出命令

在 Shell 中用 read 命令从标准输入读入数据并将读入的数据赋值给变量。命令格式：

```
read variable-name1 [variable-name2,…]
```

命令的标准输出是早已熟悉的 echo 命令。

例

```
$read var1 var2 var3 var4
11 222 3333 4444 55555
$echo $var1 $var2 $var3
11 222 3333
$echo $var3
3333
$echo $var4
4444 55555
$read var1 var2 var3 var4
11 22 33
$echo $var1 $var2 $var3
11 22 33
$echo $var4
```

```
$
```

可见，输入的值太多会将多余的值全部赋予最后一个变量；输入的值太少，后面的变量被赋予空值。

echo 是换行标准输出语句。echo 输出多个空格时必须用单引号括起，否则再多的空格也被认为只有一个。

echo 可以使用通配符：

“ * ”表示任意字符；

“ ？ ”表示一个字符；

“ A% ”表示 A 之后任意多个字符；

“ %A ”表示 A 之前任意多个字符；

“ A-Z ”表示 A 到 Z 之间的任意字符；

“ \c ”表示不换行，连续输出；

“ \\ ”表示要显示反斜线“ \ ”；

“ \n ”表示换行输出。

例　$echo "Do you see that notice? It says '　　……' "

```
Do you see that notice? It says　　……
$echo "Do you see that notice? It says　　……"
Do you see that notice? It　says ……
$echo *
db2 db2red.pdf hlliu jobbegin lliu mbox project v7-dbacert.pdf x y1
$echo db*
db2 db2red.pdf
$echo [a-z]*
db2 db2red.pdf hlliu jobbegin lliu mbox project v7-dbacert.pdf x y1
```

5.3.6　shell 中的运算

Shell 提供基本的算术运算有＋（加）、－（减）、＼＊（乘）、／（除）、%（余数）。使用这些运算符时要用 expr 语句处理，只提供整数运算。需要注意的是在运算符的前后需留有空白。

例　$expr 24 / 6

```
4
$expr 124 \* 2
248
$
```

5.4 shell 控制结构

在 Shell 中可以使用语句控制程序流程。

5.4.1 if then else 语句

if then else 语句提供条件测试。根据测试的条件执行相应的一系列语句。

1. 基本的 if 语句格式

```
if test-condition
   then commands
fi
```

语义表示为：如果测试条件（test-condition）为真，则执行命令（commands）。

例 $cat iftest1

```
# This script exer iftest1
if [ 'abc' = 'abc' ]
then echo ‘ok’
fi
# The script end
$iftest1
ok
$
```

2. 一般 if 语句格式

```
if test-condition
   then commands1
   else commands2
fi
```

语义表示为：如果测试条件（test-condition）为真，则执行命令（commands1）；否则（测试条件 test-condition 为假）执行命令（commands2）。

例 $cat iftest2

```
# This script exer iftest2
#This script is more please
echo Please type in the users name
```

```
read user
if grep $user /etc/passwd
then echo $user is the the system user
else echo $user is not the system user
fi
# The script end
$iftest2
Please type in the users name
liuxim          ←输入用户名
liuxim is the system user
$
$iftest2
Please type in the users name
guanzl          ←输入用户名
guanzl is not the system user
$
```

3. 扩充 if 语句格式

```
if test1-condition
   then commands1
   else if test2-condition
      then commands2
      fi
      else commands3
fi
```

其中 else if 可以缩写成 elif，若使用 elif 则可省略 fi。

```
if test1-condition
   then commands1
   elif test2-condition
      then commands2
   else commands3
fi
```


语义表示为：如果测试条件（test1-condition）为真，则执行命令（commands1）；否则再判测试条件（test2-condition），如果为真，则执行命令（commands2），如

果测试条件（test2-condition）为假，则执行命令（commands3）。

例 $cat iftest3

```
# This script exer iftest3
#This script is search directory
echo The Program Begin, Please Input Directory
read direc
if [ "$direc" = "/home/liuxim/project" ]
   then echo 'the directory=/home/liuxim/project'
      elif [ "$direc" = "/home/liuxim" ]
      then echo 'the directory=/home/liuxim'
   else echo $direc
fi
# The script end
$
$iftest3
The Program Begin, Please Input Directory
/home/liuxim/project          ←输入目录名
the directory=/home/liuxim/project
$
$iftest3
The Program Begin, Please Input Directory
/home/liuxim          ←输入目录名
the directory=/home/liuxim
$
$iftest3
The Program Begin, Please Input Directory
/home          ←输入目录名
/home
$
$ ./iftest3
The Program Begin, Please Input Directory
jkdgf
jkdgf
$
```

4. 测试条件

在 if 语句中测试条件非常重要，下面介绍几种测试条件。

- 数字类测试条件

data1 –eq data2：当 data1 等于 data2 时为 true；

data1 –ne data2：当 data1 不等于 data2 时为 true；

data1 –lt data2：当 data1 小于 data2 时为 true；

data1 –gt data2：当 data1 大于 data2 时为 true；

data1 –le data2：当 data1 小于等于 data2 时为 true；

data1 –ge data2：当 data1 大于等于 data2 时为 true。

- 文件类测试条件

-d file：当 file 存在且是一个目录时为 true；

！-d file：当 file 不是一个目录时为 true；

-e file：当 file 存在时为 true；

-f file：当 file 存在且是一个标准文件时为 true；

！-f file：当 file 不存在时为 true；

-k file：当 file 存在且粘滞位被设置时为 true；

-L file：当 file 存在且为符号链时为 true；

-r file：当 file 存在且为可读时为 true；

-s file：当 file 存在且文件大小大于 0 时为 true；

-w file：当 file 存在且为可写时为 true；

-x file：当 file 存在且为可执行时为 true；

-t fd：当 fd 在一个终端上被打开时为 true；

file1 –nt file2：file1 比 file2 新（根据修改日期）时为 true；

file1 –ot file2：file1 比 file2 时间早（根据创建日期）时为 true。

- 字符类测试条件

string1 = string2：当两个字符串相等时为 true；

string1 != string2：当两个字符串不相等时为 true；

-z string：当字符串长度为 0 时为 true；

-n string：当字符串长度非 0 时为 true。

- 其他测试条件

! expression：当表达式 expression 为 false 时为 true；

expression1 -a expression2：当表达式 expression1 和 expression1 都为 true 时为 true；

expression1 -o expression2：当表达式 expression1 或 expression1 为 true 时为 true。

5.4.2 case 语句

case 是一个多选择语句，根据变量与哪种模式匹配而执行相应的语句序列，采用相应的动作。

case 语句的基本格式为：

```
case value in
Mode1)    command11
          command12
          ……
          command1n ;;
Mode2)    command21
          command22
          ……
          command2n ;;
          ……
Moden)    commandn1
          commandn2
          ……
          commandnn ;;
esac
```

其中 value 可以是变量或常数，值的后面为单词 in，每个模式后面为右括号，是正则表达式。当匹配发现取值符合某一模式时就从该模式开始执行所有命令直到该模式结束。

例 显示键盘输入字母。

```
$cat casetest1
## This is the script test case
## test the key input
echo Please enter the leter you manted
read letter
case $letter in
    a) echo letter is a;;
    b) echo letter is b;;
    c) echo letter is c;;
    d) echo letter is d;;
```

```
        e) echo letter is e;;
        f) echo letter is f;;
        g) echo letter is g;;
        *) echo    letter is between h and z
esac
## the script end;
$
$casetest1
Please enter the leter you manted
a          ←输入字母
letter is a
$
$casetest1
Please enter the leter you manted
b          ←输入字母
letter is b
$casetest1
Please enter the leter you manted
x          ←输入字母
letter is between h-and z
$
```

5.4.3　for 语句

for 为循环语句。

1. 基本格式

```
for variable in list
   do
      commands
   done
```

其中 variable 是变量。如果变量值在列表中就执行一次循环，完成 do 到 done 之间的所有命令。

下面分别就列表的几种典型在例题中加以说明。

例　列表为变量


```
$cat fortest1
## the script for test
```

```
for letter in x y z
do
   echo $letter
done
## the script end
$
$fortest1
x
y
z
$
```

例 列表为字符串。

```
$cat fortest2
## the script for test
for word in "Welcome to China"
do
   echo $word
done
## the script end
$
$fortest2
Welcome to China
$
```

例 列表为 ls 命令结果。

```
$cat fortest3
## the script for test
for filedir in 'ls'
do
   echo $filedir
done
## the script end
$
$fortest3
fortest1
fortest2
dortest3
$
```

例　列表为位置参数。

```
$cat fortest4
## the script for test
for posipara in $*
do
   echo "Please Enter $posipara in this line"
done
## the script end
$
$fortest4 111 222 333 444
Please Enter 111 in this line
Please Enter 222 in this line
Please Enter 333 in this line
Please Enter 444 in this line
$
```

2. 嵌套语句格式

```
for variable1 in list1
do
   for variable2 in list2
   do
      commands
   done
done
```

例　检测各服务器是否运行。

```
$cat fortest5
## the script for test
servers="mail1.scxx.edu mail2.scxx.edu"
users="stu01 stu02"
for server in $servers
do
   for user in $users
   do
      if grep $user /etc/passwd
         then echo "server:$server    user:$user      ok!!!"
         else echo "server:$server    user:$user      no???"
```

```
        fi
    done
done
## the script end
$
$fortest5
server:mail1.scxx.edu    user:stu01    ok!!!
server:mail1.scxx.edu    user:stu02    no???
server:mail2.scxx.edu    user:stu01    no???
server:mail2.scxx.edu    user:stu02    ok!!!
$
```

5.4.4 while 语句

while 是一个不断执行一系列命令的循环语句。

基本语句格式为：

```
while test-condition
   do
      commands
   done
```

当 while 测试 test-condition 为真时，Shell 进入 while 的主体反复执行，直到测试条件为假时才停止。其中循环体在 do 和 done 之间。

例 一般的循环应用。

```
$cat whiletest1
## the script is while test
apple=0
while [ $apple lt 4 ]
do
   apple='expr $apple + 1'
   echo "apple=$apple"
done
#the script end
$
$whiletest
apple=1  see more please visit: https://homeofbook.com
apple=2
```

```
apple=3
apple=4
$
```

例　常用 while 循环作周而复始的工作，要停止循环键入 Ctrl+D。下面是一个不断接受键盘输入的 Shell script。

```
$cat whiletest2
## This script test while
echo "type Ctrl+d to terminate the script"
echo "Please enter the wonderful place"
counter=0
while read place
do
   echo "$counter.    $place is the wonderful place"
   counter='expr $counter + 1'
done
## this script end
$
$whiletest2
type Ctrl+d to terminate the script
Please enter the wonderful place
Paris
0.    Paris is the wonderful place
New York
1.    New York is the wonderful place
ShangHai
2.    ShangHai is the wonderful place
Ctrl+d
$
```

5.4.5　until 语句

until 与 while 一样，都是循环语句，但处理方式正好相反，即条件为真时停止。在某些时候，until 语句有一定优势。

基本语句格式：


```
until    test-condition
do
```

```
    commands
done
```

例 监控一个重要目录是否被删除。

```
$cat untiltest1
## this script until test
echo "Please enter the controlleddir name:"
read controlleddir
until [ ! -d $controlleddir ]
do
   sleep 2
done
echo "The $controlleddir is deleted!"
## the script end
$
$untiltest1
Please enter the controlleddir name
/home/leader
The /home/leader is deleted!
$
```

5.4.6 break 和 continue 语句

使用 break 和 continue 语句控制循环。

break 命令在作一些处理后跳出循环或跳出 case 语句。如果是位于一个多层嵌套循环中，还可以指定跳出的循环层数。

命令格式：break n

n 为要跳出的循环层数，默认值为 1。

continue 命令也是在循环中跳转，但与 break 不同的是，它不是跳出循环，而是跳出循环步，回到循环头部继续执行循环。

命令格式：continue n

n 为要跳出的循环层数，默认值为 1。

例 前面的例题，如果键入的地名是 Beijin 则用 break 跳出 while，结束脚本。

```
$cat breaktest1
```



```
## this is break test script
echo "type Ctrl+d to terminate the script"
```

```
echo "Please enter the wonderful place"
counter=0
while read place
do
   echo "$counter.    $place is the wonderful place"
   if [ "$place" = "Beijin" ]
      then break
   fi
   counter='expr $counter + 1'
done
## this script end
$
$breaktest1
type Ctrl+d to terminate the script
Please enter the wonderful place
Paris
0.    Paris is the wonderful place
New York
1.    New York is the wonderful place
ShangHai
2.    ShangHai is the wonderful place
Beijin
3.    Beijin is the wonderful place
$

$cat continuetest1
## this is continue test script
echo "type Ctrl+d to terminate the script"
echo "Please enter the wonderful place"
counter=0
while read place
do
   echo "$counter    $place is the wonderful place"
   if [ "$place" = "Beijin" ]
```

```
        then continue
    fi
    counter='expr $counter + 1'
done
## this script end
$
$continuetest1
type Ctrl+d to terminate the script
Please enter the wonderful place
Paris
0.  Paris is the wonderful place
New York
1.  New York is the wonderful place
ShangHai
2.  ShangHai is the wonderful place
Beijin
3.  Beijin is the wonderful place
Berlin
3.  Berlin is the wonderful place
Venice
4.  Venice is the wonderful place
Ctrl+d
$
```

5.5 Shell 函 数

除了在 Shell 中可以定义变量之外，在 Shell 中也可以定义函数。Shell 函数是将 Shell 的一些语句或命令集组合在一起形成能完成特定功能的程序块。当 Shell 脚本需要时可以直接调用。

5.5.1 函数定义

Shell 函数的基本格式为：


```
func name( )
{
```

```
    commands
}
```

5.5.2　脚本中函数调用

通常将函数看成是脚本中的一段代码，在使用函数之前必须先定义该函数，使用时利用函数名直接调用。

例　下面首先创建函数，然后调用函数。

```
$cat funcexer
## In this script defining function abc ( ) and using abc( )
abc ( )
{
   echo "This is the function abc"
}

echo "The abc functon begin"
abc
echo "The abc functon end"
## funcexer end
$
$funcexer
The abc functon begin
This is the function abc
The abc functon end
$
```

执行函数时保留当前 Shell 和内存信息，Shell 脚本和函数之间的参数传递利用变量直接传递。

函数有两种方式可以返回到脚本中调用函数的控制部分：

- 当函数正常执行到末尾时直接返回；
- 用 return 语句返回时带值 0 或 1，0 表示函数调用无错，1 表示有错。

例　$cat funcreturn

```
inputxyz ( )
{
   echo "x:add"
   echo "y:delete"
```

```
    echo "z:update"
    echo "e:exit"
}

inputxyz
read xyze
case $xyze in
    x) echo "add a line";;
    y) echo "delete a line";;
    z) echo "update a line";;
    e) return 0;;
    *) echo "input error"
    return 4;;
esac
return 0
$
$funcreturn
x: add
y:delete
z:update
e:exit
s
input error
$
```

5.5.3 Shell 中使用函数

经常使用的 Shell 函数还可以放入函数文件中，然后将函数文件加载到 Shell 中。这样在 Shell 下就可以像使用命令一样直接键入该函数运行。

函数加载步骤：

(1) 文件的头部包含语句#!/bin/sh；

(2) 定位文件格式：./pathname/filename；

(3) 键入命令：$. /func-name（表示点、空格、斜线、函数名）。

用命令 set 检查函数是否已经加载：

```
$set
```

```
AUTHSTATE=compat
CGI_DIRECTORY=/var/docsearch/cgi-bin
DOCUMENT_DIRECTORY=/usr/docsearch/html
FCEDIT=/usr/bin/ed
HOME=/home/liuxim
……
USER=liuxim
finduser=( )
{
read user
if grep $user /etc/passwd
   then echo $user is the the system user
   else echo $user is not the system user
fi
}
LANG=en_US
LC__FASTMSG=true
……
```

函数成功加载后在 Shell 提示符下只需键入函数名即可执行。

例　$finduser

```
liuxim is the the system user
$
```

不用 Shell 函数时可以用命令 unset 删除它。

例　$unset finduser

```
$
```

删除后再用 set 检查就没有该函数了。

5.6　Shell　工　具

UNIX 操作系统提供了一些命令用于 Shell 脚本的调试，将这些命令称为 Shell 工具。

常用的有：trap、logger、eval 等。

5.6.1 通知 trap

通知 trap 是操作系统监控命令，用于在 Shell 脚本中捕捉信号。用户在执行 Shell 程序时，为了终止正在执行的 Shell 程序并回到系统提示符下，按下 Del 或 Break 键，但这并不是正常终止，系统可能需要在中断后执行一些后处理工作，因此可能会产生许多临时文件。从根本上讲这些临时文件正是 trap 捕捉到 Del 或 Break 操作后所产生的操作。

trap 命令格式：trap operateing signals

其中 signals 表示捕捉到的信号；operating 是专门用来处理捕捉到信号后所采取操作的函数，使用时用双引号（“ ”）将 operateing 括起来。

在 UNIX 系统中可以发出的 signals 信号及其描述如表 5.1 所示。

表 5.1 UNIX 系统中可以发出的 signals 信号及其描述

signal	描述
0	退出 shell，按 Ctrl+D 时产生。
1	挂起（hangup），按 Ctrl+S 时产生，表示通信挂起。
2	中断（interrupt），按 Del 或 Ctrl+Break 或 Ctrl+C 时产生。
3	退出（Quit）。
4	非法命令（illegal instrution）。
5	跟踪 trap（trace instruction）。
6	IOT 命令。
7	EMIT 命令。
8	异常的浮点格式（floating point exception）。
9	终止进程（kill），无法被忽略的信息。
10	系统总线出错（Bus error）。
12	系统调用时提供的参数不当。
13	输出到管道时无进程接收。
14	超时报警（alarm timeout）。
15	软件结束信号。

例 trap “commands” 3 4 表示如果捕捉到信号 3、4，则执行相应的 commands 命令。

trap “”2,3 表示如果捕捉到信号 2、3 则忽略信号 2、3，用户不终止该脚本。

trap 2,3 表示复位信号 2、3，用户可以终止该脚本。

例 $traptest1


```
#!/bin/sh
```

```
deletedir ()
{
   echo "delete directory function begin"
   rm /home/liuxim/exer
   echo "delete directory function end"
   exit
}
counter=0
trap "deletedir" 2
while :
   do
      counter='expr $counter + 1'
      echo $counter
   done
## the script end
$
$cat traptest1
1
2
3
……
129
<Ctrl><C>
delete directory function begin
delete directory function end
$
```

例　清除临时文件。

```
$cat traptest2
#!/bin/sh
## this is trap script
rmtmpf   ( )
{
```



```
   echo "now receiving interrupt"
   echo "Do you want to deleting the tmp file?(y/n)"
```

```
    read yn
    case $yn in
      y) rm /tmp/*.$$ 2>/dev/null
         echo “temp file deleted”
         exit;;
      n) ;;
    esac
}

trap "rmtmpf" 1 2 3
file1=/tmp/file1.$$
file2=/tmp/file2.$$
echo "begin"
while :
do
    df >>$file1
      ps xa >>$$file2
    done
$
$traptest2
begin
now receiving interrupt
<Ctrl><C>
Do you want to deleting the tmp file?(y/n)y
temp file deleted
$
```

如果 trap 后不带参数，Shell 会显示到现在为止曾修改过的任何 trap。

实际上，操作系统之外的 TCP/IP 网络管理中的简单网络管理协议（SNMP）中也用 trap 来检测和管理网络。从 trap 命令可见，UNIX 和网络管理是一致的。

5.6.2 创建信息的文件

在编写脚本程序时随时需要保存一些临时文件和日志文件供调试程序用，相应的就需要创建这些文件。

- 临时文件创建。

- 日志文件创建：使用 date 命令可以创建日志文件。

修改系统日期的命令：date –option +%day%month%year

用符号“ + ”表示设置当前的日期和时间。

修改系统日期：

```
$ date +%d%m%Y
12032003
$date +%d-%m-%Y
12-03-2003
$date +%A" "%e" "%B" "%Y
Wednesday 12 March 2003
```

例　创建日志文件。

```
$cat logcrt
#!/bin/sh
# This script create log file
yom='date +%d%m%Y'          #修改系统日期类型
logfile=/var/adm/log/log_bak.$yom>$logfile
## end
$
```

这样就产生了日志文件 log_bak.12032003。

例　创建临时文件。

```
$cat tmpfilecrt
#!/bin/sh
# This script create temp file
tempfile=/tmp/tempfile.$$
df –tk>tempfile
rm /tmp/*.$$
#end
$
```

运行脚本创建临时文件 tempfile.396，其中 396 是进程号。

5.6.3　logger

UNIX 系统的一个特征是存在日志文件。日志文件通常位于目录/var/adm 或目录/var/log 下，取名为 messages。messages 文件在文件/etc/syslog.conf 中配置，它含有用来发送各种不同类型消息的工具及优先级。当系统管理员需要监控脚本报告及

某一特定时间段的用户登录或访问情况时，命令 logger 用于向文件/etc/syslog.conf 发送消息。

logger 命令格式：logger –p –i message

其中 p 为优先级；-i 为在消息中记录发送消息的进程号。

```
$cat /etc/syslog.conf
# @(#)34            1.9    src/bos/etc/syslog/syslog.conf, cmdnet, bos411, 9428A410j 6
/13/93 14:52:39
#
# COMPONENT_NAME: (CMDNET) Network commands.
#
# FUNCTIONS:
#
# ORIGINS: 27
#
# (C) COPYRIGHT International Business Machines Corp. 1988, 1989
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# /etc/syslog.conf - control output of syslogd
#
#
# Each line must consist of two parts:-
#
# 1) A selector to determine the message priorities to which the
#       line applies
# 2) An action.
#
# The two fields must be separated by one or more tabs or spaces.
#

# format:
#
```

```
# <msg_src_list>                    <destination>
#
# where <msg_src_list> is a semicolon separated list of <facility>.<priority>
# where:
#
# <facility> is:
#           * - all (except mark)
#           mark - time marks
#           kern,user,mail,daemon, auth,... (see syslogd(AIX Commands Reference))
#
# <priority> is one of (from high to low):
#           emerg/panic,alert,crit,err(or),warn(ing),notice,info,debug
#           (meaning all messages of this priority or higher)
#
# <destination> is:
#           /filename - log to this file
#           username[,username2...] - write to user(s)
#           @hostname - send to syslogd on this machine
#           * - send to all logged in users
#
# example:
# "mail messages, at debug or higher, go to Log file. File must exist."
# "all facilities, at debug and higher, go to console"
# "all facilities, at crit or higher, go to all users"
#   mail.debug              /usr/spool/mqueue/syslog
#   *.debug                 /dev/console
#   *.crit                          *
syslog.conf: END
```

5.6.4 eval

eval 是一个在 Shell 下执行的命令，在命令前加上 eval，eval 将会首先扫描命令行两层，然后再执行该命令。

例 $cat evaltest1

```
#!/bin/sh
# eval test script
eval echo evaltest
eval echo $#
# end
$
$evaltest1 a b c d e
evaltest
5
$
```

例 $date_1="date"

```
$echo $date_1
date
$eval $date_1
Tue Apr  1 17:11:40 TAIST 2003
$
```

例 文件 infor 中有两列，现在要给第一列中的变量名赋予相应的第二列的值，用 eval 实现。

```
$cat infor
name              value
lixinhua           9500
qiujielun          9800
wanghong          10000
yangxiaolin        9900
zhuxiaomei        10020
$cat evaltest2
#!/bin/sh
## This script is eval test
```

```
while read name value
do
   eval echo "${name}=${value}"
done
$
$evaltest2
name=value
lixinhua=9500
qiujielun=9800
wanghong=10000
yangxiaolin=9900
zhuxiaomei=10020
$
```

5.7　Shell Script 编程应用实例

例 1　下例是按/etc/hosts 文件中的 IP 地址 ping 所有主机的脚本。

```
$cat pinghost
more /etc/hosts |grep -v "^#" |while read every
do
   hostaddr='awk '{print $1}' '
   for host in $hostaddr
      do
         ping -cl $host
      done
done
# end
$
```

例 2　下例是一个定期查看邮箱的程序，当邮箱的内容变化时，程序行显示“ you have mail”。

```
$cat checkmail
#checkmail: watch mailbox for growth
          see more please visit: https://homeofbook.com

PATH=/bin:/usr/bin
```

```
MAIL=/var/mail/'whoami'          #system dependent

t=60
x='ls -l $MAIL'
while :
do
   y='ls -l $MAIL'
   echo $x $y
   x="$y"
   sleep $t
done | awk '{ if ($5<$14) print "you have mail"}'
$
```

例 3 系统中有些日志文件增长十分迅速，每天手工检查这些日志文件的长度并倒换这些日志文件（通常给文件名加个时间戳）非常繁琐。可以编写一个脚本自动完成这项工作。

如果某个日志文件超过了特定的长度，它的内容就被倒换到另一个文件中，并清除原有文件内容。

该脚本文件的长度限制由变量 BLOCK_LIMIT 设定。这个数字代表了块数目。在本例中是 8 块（每块大小是 4 K 字节）。可以按照自己的需求把这个数字设得更高。所有要检查的日志文件名都保存在变量 LOGS 中。

```
$cat logroll
#!/bin/sh
# logroll
# roll over the log files if sizes have reached the MARK
# could also be used for mail boxes.
# limit size of log 4096k
BLOCK_LIMIT=8

MYDATE='date+%d%m'
# list of logs to check …… you will be difficulty!
LOGS="/var/spool/audlog /var/spool/networks/netlog /etc/dns/named_log"
for LOG_FILE IN $LOGS
```



```
do
   if [-f $LOG_FILES]; then
```

```
    #get block size
    F_SIZE='du –a $LOG_FILE |cut –f1'
    else
       echo "basename $0'cannot find $LOG_FILE">&2
       #could exit here,but I want to make sure we hit all logs
       continue
    fi

if ["$F_SIZE" -gt "$BLOCK_LIMIT"];
then
    #copy the log across and append a addmm date on it
    cp $LOG_FILE $LOG_FILE$MYDATE
    #create/zero the new log
    >$LOG_FILE
    chgrp admin $LOG_FILE$MYDATE
fi
done
```

该程序中使用了一个 for 循环来依次检查每一个日志文件，使用 du 命令来取日志文件长度。如果相应的文件长度大于 BLOCK_LIMIT 变量所规定的值，那么该文件将被拷贝到一个文件名含有时间戳的文件中，并改变这个文件的所属的组，原先的文件长度将被截断为 0。

5.8　本 章 小 结

本章主要介绍了如何编写一个 Shell 脚本程序，涉及到了变量、语法结构、函数的用法以及一些相关工具的使用。熟练掌握 Shell 脚本程序的设计，对日常系统维护有很大帮助，很多重复进行的、繁琐的事情，都可以交给 Shell 脚本程序，实现高效率的系统管理。

上 机 练 习

1. 用 Shell 命令确定自己登录的是何种 Shell？是否与/etc/passwd 文件中定义的 Shell 相符？
2. 用命令从当前的 Shell 切换到另一种 Shell。
3. 编写一个简单 Shell 脚本完成：将 who 命令的输出结果定向到文件 file 中，并用 more 命

令查看文件 file 的内容。

4. 设计一个 Shell 脚本：将当前工作目录下的文件按由大到小的顺序列出所有文件名。

5. 设计一个 Shell 脚本：将所有的输入整数相加。

6. 设计一个 Shell 脚本：将现在的系统时间用 banner 命令显示出来。

7. 通过自己设计一些简单的 shell 程序，熟悉变量、函数以及各种控制结构的用法。

习 题 五

1. UNIX 的 Shell 主要有哪几种类型？

2. UNIX 用户可以选择哪些 Shell 环境？

3. Shell 程序的特点是什么？

4. 请说明 Shell 编写程序时如何将编写完成的 Shell 程序提交给系统运行。

第 6 章　UNIX 实用程序

UNIX 实用程序是 UNIX 系统提供的应用软件工具，用于帮助用户处理文件应用等问题。

本章主要内容

本章要介绍的 UNIX 实用程序有：

- grep：用于在文件中搜索指定的内容。
- sort：用于文件排序的实用程序。
- sed：用于文件行阅读，也可以按匹配格式阅读文件内容。
- comm、diff、cmp：用于比较两个或多个文件内容。
- awk：用于在文件或字符串中按一定规律阅读。

6.1　grep

6.1.1　grep 介绍

grep 的全称是“Global Regular Expression Print”（全局正则表达式打印），即将满足全局正则表达式的内容打印在显示器上。也就是用来在文本文件中搜索匹配指定正则表达式的所有行，并将其写到标准输出显示器上。grep 是一个程序家族，除 grep 之外，还有两个功能相似的附加指令 egrep 和 fgrep。egrep 相当于 grep – e，fgrep 相当于 grep – f。

6.1.2　grep 命令

grep 命令在执行时首先建立一个只保存一行文本的模式空间，然后将文件中匹配后的下一行复制到模式空间中并对该模式空间应用正则表达式，如果匹配存在，则将该行从模式空间中复制到显示器上。通过反复执行该过程，直到输入文件的最后一行为止。

由于 grep 命令借助于行的模式空间匹配，不能修改输入文件，不能对输入文件作增加、删除。

命令格式：grep [选项] [查找模式] [文件名 1,文件名 2,……]

其中选项包括：

b：在输出的每一行前显示包含匹配字符串的行在文件块中的编号；

c：只显示匹配行的数量；

e expression：指定检索使用的模式，用于防止以“ - ”开头的模式被解释为命令选项；

f expfile：从 expfile 文件中获取要搜索的模式，一个模式占一行；

h：在查找多个文件时，指示 grep 不要将文件名加入到输出之前；

i：比较时不区分大小写；

l：显示第一个匹配串所在的文件名并用换行符将其隔开，当在某文件中多次出现匹配串时，不重复显示此文件名；

n：在输出前加上匹配串所在行的行号（文件首行行号为 1）；

v：只显示不包含匹配串的行；

s：抑制所有输出，即哑模式；

x：只显示整行完全匹配的行。

grep 命令在指定的输入文件中查找与模式匹配的行。如果没有指定文件，则从标准输入中读取。

正常情况下，将每个匹配的行显示到标准输出。如果要查找的文件是多个，则在每一行输出之前加上文件名。

对 grep 命令的使用还需注意以下几个方面：

- 在命令后键入搜索模式，再键入要搜索的文件。其中，文件名列表中也可以使用特殊字符，如“ * ”，也可以生成全部文件的文件名列表。如果想在搜索模式中包含有空格的字符串，可以用单引号把要搜索的模式括起来，用来表明搜索模式是由包含空格的字符串组成的。否则 Shell 会把空格认为是命令行参数的定界符，而 grep 命令将把搜索模式中的单词解释为文件名列表中的一部分。在下面的例子中，grep 命令在文件 example 中搜索模式“ text file ”。

例 $grep 'text file' example

- 用户可以在命令行上用 Shell 特殊字符生成需要搜索的文件名列表。在下面的例子中，特殊字符“ * ”用来生成一个文件名列表，该列表包含当前目录下的所有文件。该命令将搜索出当前目录下所有文件中与模式匹配的行。

例 $ grep data *

- 特殊字符在搜索一组指定的文件时非常有用。例如，如果想搜索所有 C 程序源文件中特定的模式，可以用“ *.c ”来指定文件名列表。假设用户的 C 程序中包含一些不必要的转向语句（goto 语句），想要找到这些语句，可以用如下的命令来

搜索并显示所有包含 goto 语句的代码行：

```
$grep goto *.c
```

- 用户可以在命令行上键入搜索模式，也可以使用-f 选项从指定文件中读取要搜索的模式。每个搜索模式在文件中占一行。如果经常要搜索一组常见字符串，这个功能非常有用。在下面的例子中，用户要在文件 exam 中搜索字符串“ editor ”和“ create ”，就把要搜索的模式放在文件 mypats 中，然后，grep 命令从文件 mypats 中读取要搜索的模式。

```
$cat mypats
editor
create
$grep -f mypats exam
```

例　在文件/etc/passwd 中寻找所有包含 anonymous 的行。

```
$grep 'anonymous' /etc/passwd
```

例　在文件 file 中寻找所有包含 Doc（不区分大小写）的行。

```
$grep –i 'doc' file
```

例　在当前目录中搜索文件名与字符“ data ”相匹配的所有文件，并在屏幕上列出文件名，文件名之间换行分开。

```
$grep –l 'data'*
```

6.1.3　grep、fgrep 和 egrep 命令

grep、fgrep 和 egrep 命令都属于 grep 家族。它们的特点是：

- grep 命令一次只能搜索一个指定的模式；
- egrep 命令检索扩展的正则表达式（包括表达式组和可选项）；
- fgrep 命令检索固定字符串，不识别正则表达式，是快速搜索命令。

可见，grep 命令的搜索功能比 fgrep 强大，因为 grep 命令的搜索模式可以是正则表达式，而 fgrep 却不能，但 fgrep 更快。

如果搜索规则只需要序列表达式，则快速 grep 是最好的实用程序。

例　搜索文件 file 中包含“？”的所有行及行号。

```
$fgrep –n '?' file
```

例　搜索文件 file 中所有以字符开始的行。

```
$egrep –i '^[A-Z]' file
```

6.1.4 grep 与正则表达式

正则表达式（Regular Expression）是一组由一个或多个字符组成的字符串，可以吻合某种特殊的模式。

在基本正则表达式中，下列特殊字符有特定的含义：

？：匹配任意一个字符；

*：匹配一个或多个字符；

^：匹配一行的开始处的字符；

$：匹配一行的结尾处的字符；

[]：匹配在括号中描述的一组字符，只要满足此组字符中的任何一个字符即可；

！：匹配紧跟在后但不在[]内的字符。

另外需要注意的是：

- 正则表达式后面跟一个“ + ”号，表示匹配此正则表达式的一次或多次出现；
- 正则表达式后面跟一个“ ？ ”号，表示匹配此正则表达式的零次或多次出现；
- 两个正则表达式之间用“ | ”号隔开，表示匹配两个正则表达式中的一个；
- 正则表达式可以用“ ( ) ”括起来。

例 删除文件 filetest 中的所有空行。

```
$grep –v '^$' filetest>filetest1
```

这是一个用选项“ -v ”实现取逆的例子。用只包含行开始和结尾运算符“ ^$ ”的模式选择所有空行，在此基础上取逆，并将内容放入一个新文件 filetest1 中。

例 将文件 filetest1 中第一个非空字符为“ A ”的行显示在屏幕上。

```
$grep '^*A' filetest1
```

例 将文件 filetest1 中以分号结尾的行及行号显示在屏幕上。

```
$grep -n ';$' filetest1
```

6.2 sort

6.2.1 sort 介绍

sort 用于对文件的内容进行排序，也可以用于检查文件是否已排序，还可以将多个文件结合并排序形成一个大文件达到合并的效果。

通常情况下，sort 会对文件内容按行排序，在比较字符时，根据字符的 ASCII 值大小排序。

6.2.2　sort 使用

sort 命令格式为：sort [options] [files]

files 为要排序的文件名。

其中选项：

+POS1 [-POS2]：指定一个或几个字段作为排序关键词，字段位置从 POSl 开始，到 POS2 为止（包括 POSl，但不包括 POS2）。如果不指定 POS2，则关键词为从 POSl 到行尾。比如，sort +0-1 file1 表示按文件 file1 中的第 1 个字段（从第 1 个字段开始到第 2 个字段结束）排序；sort +0-2 file1 表示按文件 file1 中的第 1、2 字段（从第 1 个字段开始到第 3 个字段结束）排序。

-b：忽略排序字段或关键词中开头的空格。

-c：检查指定文件是否已经排序。

-d：在关键词中只考虑[a-zA-Z0-9]字符。

-f：将关键词中的小写字母转换成大写字母。

-g：按照通常的数字值顺序作比较，暗含-b。

-k：POS1[,POS2]从关键词 POS1 开始，到 POS2 结束（包含 POS1 和 POS2）。字段数和字符偏移量都从 1 开始计数。

-m：合并已经排好序的文件，不排序。

-M：作为月份比较：“ JAN ”<“ FEB ”。

-n：按照字符串的数值顺序比较，暗含-b。

-o FILE：将结果写入 FILE 而不是标准输出。

-r：颠倒比较的结果。

例　sort 排序的文件很大时，可以用管道和 more 命令，如果要保留排序结果，可将其重定向到一个新文件。即：$sort file1 | more 或 $sort file1>filenew。

例　$cat file

```
Will,Peter
Anny,Smith
Moly,Robert
Bill,Karen
Gobi,Seka
$sort –c file
sort:disorder: Anny,Smith
$sort file | sort –c
$
```

用选项 c 可以检查文件是否已经排序，命令执行结果首先将没有排序的行数据显示出来。检查到没有排序时，再用命令 sort 排序并将结果通过管道送给 sort 检查是否已经排序，最后结果是完成排序，所以没有输出。

例 $sort –m file1 file2 file3>file

用选项-m 将三个已经排好序的文件 file1、file2、file3 连接成一个大文件 file。如果不用选项-m，同样也可以将它们合并成一个大文件：

```
$sort file1 file2 file3>file
```

例 用数据项排序时排序号码从 0 开始，下面以第 3 个数据项作为排序的关键字段对文件 file 排序，直到最后。

```
$sort +2 file
```

例 某城市 1999~2002 年上半年某商品销售量文件 sales。

```
1999   800    600   200   100   60    40
2000   1500   900   400   250   100   80
2001   2400   1500  600   400   200   110
2002   2800   1400  800   300   120   90
```

现在检查该文件是否按第 1 字段排序，如果未排序，显示未排序文件的第 1 行：

```
$sort –c +0-1 sales
```

现在检查该文件是否按第 5 字段排序，如果未排序，显示未排序文件的第 1 行：

```
$sort –c +4-5 sales
disorder：2002   2800   1400   800   300   120   90
```

6.3 sed

6.3.1 sed 介绍

sed 是流编辑器（Stream Editor）的头字母缩写。实际上 sed 并非一个真正的编辑器，它不能改变最初文件的内容。只是按行浏览输入文件，对输入文件的每行执行命令列表，所有被改变的输出都写入标准输出中，如果要保存，则需重定向到一个文件。

sed 实现时首先为输入文件的每行提供一个行编号，用来定位文本中的行。sed 对每行执行的操作为：

- 与 grep 非常相似，sed 将一个输入行复制到模式空间中，模式空间是一个保存用于处理一个或多个文本行的特定缓冲区；（注意：grep 的模式空间只装入一行。）

- 将脚本中所有指令，从上到下依次应用到匹配指令中指定地址的所有模式空间行；
- 将模式空间中的内容复制到输出文件中。

sed 只有一个命令时，从键盘直接输入。大多数情况下，sed 有多个命令，这些命令被放在一个文件中，该文件被称为 sed 脚本。

6.3.2　sed 使用

sed 命令格式为：sed [*options*] [*command*] [*files*]

在命令行使用 sed 时，实际命令要加单引号。

sed 也允许加双引号。不论是使用 Shell 命令行方式或脚本文件方式，如果没有指定输入文件，sed 就从标准输入中接受输入。

sed 选项如下：

n：禁止自动输出，sed 不将编辑行写到标准输出。sed 命令默认时无选项 n，表示标准输出显示所有行。

p：命令可以用来打印编辑行。

c：下一个命令是编辑命令。使用多项编辑时加入此选项。如果只用到一条 sed 命令，此选项无用，但指定它也没有关系。

f：指出有一个脚本文件紧跟在命令行选项后，即调用 sed 脚本文件。

e：默认选项。表示脚本是一个命令，直接在命令行上，不是一个脚本文件。

格式 1：$sed -e 'address command' inputfile

格式 2：$sed -f scriptfile.sed inputfile

格式 1 以命令方式执行 sed 应用程序，在格式 2 中 sed 应用程序执行 sed 脚本，脚本文件名为 scriptfile.sed，这里文件名后缀为.sed 并非一定要求，此处只是为了突出 sed 脚本而取为.sed 后缀。

例　将 sed 脚本文件 mywork.sed 作用于文件 file1、file2、……。

```
$sed -f mywork.sed file1 file2 …
```

这里 mywork.sed 为 sed 脚本文件。

由于 sed 不会修改源文件，如果想要保存改动内容，就需要将所有输出重定向到另外一个文件。

例　$sed 'commands' source_file>output_file

这里直接执行 sed 命令。

6.3.3 文本查询

sed 处理源文件时，默认从第一行开始，有两种方式定位文本：

- 使用行号，可以是一个简单数字，或是一个行号范围；
- 使用正则表达式。

使用 sed 定位文本的方式有：

x：x 是一个行号，如 1。

x,y：表示行号范围从 x 到 y，如 2,5 表示从第 2 行到第 5 行。

/pattern/：查询包含模式的行，如/student/或/[a-z]/。

/pattern/pattern/：查询包含两个模式的行，如/student01/student04/。

/pattern/,x：在给定行号上查询包含模式的行，如/student/,3。

x,/pattern/：通过行号和模式查询匹配行，如 3,/student/。

x,y!：查询不包含指定行号 x 和 y 的行，如 1,2!。

6.3.4 sed 基本编辑命令

sed 编辑命令有：

p：打印匹配行。

=：显示文件行号。

a\：在定位行号后附加新文本信息。

i\：在定位行号后插入新文本信息。

d：删除定位行。

c\：用新文本替换定位文本。

s：使用替换模式替换相应模式。

r：从另一个文件中读文本。

w：写文本到一个文件。

q：第一个模式匹配完成后退出或立即退出。

l：显示与 8 进制 ASCII 码等价的控制字符。

{}：在定位行执行的命令组。

n：从另一个文件中读文本下一行，并附加在下一行后。

g：将模式 2 粘贴到/pattern n/。

y：传送字符。

n：延续到下一个输入行；允许跨行的模式匹配语句。

6.3.5　sed 使用例子

1. 使用 p 显示行

p 只显示 sed 脚本中的行。

print 命令格式为：[address[,address]]p

例　文件名为 article.txt，下面首先看一下该文件内容。

```
$cat article.txt
Sodd's Second Law:
Sooner or later, the worst possible set of
circumstances is bound to occur.
$
```

sed 使用打印模式打印出文件中的第 3 行：

```
$sed -n '3p' article.txt
circumstances is bound to occur.
$
```

打印文件中的第 2、3 行：

```
$sed -n '2,3p' article.txt
Sooner or later, the worst possible set of
circumstances is bound to occur.
$
```

2. 打印模式

打印模式打印匹配的一行。

例　上例中的文件 article.txt，要打印匹配单词 later 的行。

```
$sed –n '/later/p' article.txt
Sooner or later, the worst possible set of
$
```

3. 显示整个文件

要打印整个文件，只需将行范围设为第一行到最后一行，最后一行用$表示。

例　显示文件 article.txt 中的所有行。

```
$sed –n '1,$p' article.txt
```



```
S odd's Second Law:
Sooner or later, the worst possible set of
```

```
circumstances is bound to occur.
$
```

4. 匹配任意字母

查询以 ble 结尾的任意单词。

例 显示文件 article.txt 中与字母 ble 相匹配的行。

```
$sed –n '/.*ble/p' article.txt
Sooner or later, the worst possible set of
$
```

5. 打印行号

要打印行号，使用格式/pattern/=。

例 显示文件 article.txt 中的所有行,将与字母 ble 相匹配的行号显示出来。

```
$sed '/ble/=' article.txt
Sodd's Second Law:
2
Sooner or later, the worst possible set of
circumstances is bound to occur.
$
```

也可以是：

```
$sed -n '/ble/=' article.txt
2
$
```

如果要打印行号及匹配行，必须使用两个 sed 命令，并使用 e 选项。第一个命令打印模式匹配行，第二个使用=选项打印行号。

```
$sed -n -e '/ble/p' -e '/ble/=' article.txt
Sooner or later, the worst possible set of
2
$
```

6. 匹配元字符

匹配元字符$前，必须使用反斜线\屏蔽其特殊含义，例如匹配价格为 4 美元的报价 ‘ /\$4.00/p’。

7. 追加文本

追加操作格式如下：

```
[address] a\
text1\
text2\
…
textn
```

address 指定一个模式或行号，定位新文本的附加位置，sed 对 a\后的文本进行追加操作。注意每一行后面都有一个斜划线（\），这个斜划线代表换行，最后一行不加斜划线；sed 会将这些文本追加到匹配模式行的后面。

```
$cat append.sed
#!/bin/sed  – f
/bound/ a\
That ’ s all.\
OK
$chmod u+x append.sed
$./append.sed article.txt
Sodd's Second Law:
Sooner or later, the worst possible set of
circumstances is bound to occur.
That's all.
OK
$
```

8. 插入文本

插入操作格式如下：

```
[address] i\
text1\
text2\
…
textn
```

插入文本类似追加文本，命令为 i\；sed 会将这些文本插入到匹配模式行的位置。


```
$cat insert.sed
```

```
#!/bin/sed -f
/bound/ i\
Hah…….
$chmod u+x insert.sed
$./insert.sed article.txt
Sodd's Second Law:
Sooner or later, the worst possible set of
Hah…….
circumstances is bound to occur.
$
```

9. 修改文本

修改操作格式如下：

```
[address] c\
text1\
text2\
…
textn
```

sed 会将匹配模式行替换成这些文本。

```
$cat change.sed
#!/bin/sed -f
/worst/ c\
more and more
$chmod u+x change.sed
$./change.sed article.txt
Sodd's Second Law:
more and more
circumstances is bound to occur.
$
```

10. 删除文本

删除操作格式如下：

```
[address[,address]]d\
```


删除第一和第二行：

```
$sed ‘ 1,2d ’ article.txt
circumstances is bound to occur.
$
```

11. 替换文本

替换命令用于替换指定模式，格式为：

[address[, address]] s/pattern-to-find/replacement-pattern/[gpwn]

s 选项通知 sed 这是一个替换操作，并查询 pattern-to-find，成功后用 replacement-pattern 替换它。

替换选项如下：

g 默认情况下只替换第一次出现的模式,使用 g 选项替换全局所有出现的模式；

p：默认情况下 sed 将所有被替换行写入标准输出，加 p 选项将使-n 选项无效，-n 选项不打印输出结果；

w：文件名使用此选项将输出定向到一个文件。

例　替换 the 为 THE。

```
$ sed 's/the/THE/' article.txt
Sodd's Second Law:
Sooner or later, THE worst possible set of
circumstances is bound to occur.
$
```

例　替换全局的 e 为 E，并将替换的结果写入文件 new.txt。

```
$sed 's/e/E/gw new.txt' article.txt
Sodd's SEcond Law:
SoonEr or latEr, thE worst possiblE sEt of
circumstancEs is bound to occur.
$cat new.txt
Sodd's SEcond Law:
SoonEr or latEr, thE worst possiblE sEt of
circumstancEs is bound to occur.
$
```

6.3.6　sed 与 grep

grep 实用程序也是用于查找和打印文件中匹配一个正则表达式的行。如果不能使用 grep 实用程序，可以用 sed 实用程序实现同样的功能。sed 实用程序的功能比

grep 更强大，grep 中的一切都可以用 sed 实现。但 grep 执行速度比 sed 快。

6.4　comm、diff、cmp 指令

在 UNIX 中命令 comm、diff、cmp 分别用于比较两个或多个文件内容。

6.4.1　comm 命令

comm 命令用于比较两个文件中是否有相同行。命令格式：

```
comm [options] file1 file2
```

file1 和 file2 为要比较的两个文件的文件名。

其中选项：

-1：屏蔽左边文件（file1）中不同于右边文件（file2）的行（或内容）；

-2：屏蔽右边文件（file2）中不同于左边文件（file1）的行（或内容）；

-3：屏蔽两个文件中不同的行（或内容）；

-l：认为输入文件根据当前的 locale 进行过排序（应该给 sort 提供-l 选项）；

-help：显示帮助信息，然后结束；

-version：显示版本信息，然后结束；

comm 逐行比较已排好序的文件 file1 和 file2，在输出中的第 1 列为 file1 的独有行，第 2 列为 file2 的独有行，第 3 列为两个文件的共有行。

通常 comm 命令对排过序的文件更有用。

例　表 6.1 是文件 file1 与 file2 的内容。

表 6.1　文件 file1 与 file2 的内容

file1	file2
the same of 1	the same of 1
the differ of file1	the differ of file2
the same of 2	the same of 2
the differ of file11	the differ of file22
the differ of file111	the differ of file222
the same of 3	the same of 3
the same of 4	the same of 4
the end of file1	the end of file2

```
$comm. file1 file2

                the same of 1
the differ of file1
```

```
                    the differ of file2
                                the same of 2
        the differ of file11
                    the differ of file22
        the differ of file111
                    the differ of file222
                                the same of 3
                                the same of 4
        the end of file1
                   the end of file2
        $
```

例
```
$sort –u file1>temp1
$sort –u file2>temp2
$comm. temp1 temp2>file
```

6.4.2 diff 命令

diff 命令显示两个文件之间行对行的差别，并且标出差别，以便修改第 1 个文件使它与第 2 个文件相匹配。除两个文件之外，diff 也可用于一个文件与一个目录或两个目录。当 diff 用于一个文件与一个目录时，表示查找该目录下是否存在同名文件。当 diff 用于两个目录时，表示比较两个目录下同名文件的差别。命令格式：

```
diff [options] [directory] file1 file2
```

file1 和 file2 是要比较的两个文件的文件名。

其中选项：

-a：所有的文件都视为文本文件逐行比较；

-b：忽略空格引起的变化；

-B：忽略插入删除空行引起的变化；

-brief：仅报告文件是否相异，注重的是差别的细节；

-c：使用上下文输出格式；

-C：行数（一个整数）；

-e：输出为一个有效的 ed 脚本；

-I：忽略大小写；

-q：仅报告文件是否相异，不报告详细的差异；

-r：比较目录时，递归比较所有找到的子目录。

若两个文件的数据相同，则 diff 不显示任何信息。

例　文件 file1 与 file2 的内容如表 6.2 所示。

表 6.2　文件 file1 与 file2 的内容

file1	file2
the same of 1	the same of 1
the same of 2	the same of 2
file1	file2
the same of 3	the same of 3
the same of 4	the same of 4
file11	file22
file111	file222
the same of 5	the same of 5
	file2
	difference1
	file2
	difference2

```
$diff file1 file2
3c3
<file1
---
>file2
6,7c6,7
<file11
<file111
---
>file22
>file22
8a9,10
>file2 difference1
>file2 difference2
```

diff 结果是这两个文件前面两行相同，未指出差异。第一个差别（3c3）指出第 3 行必须修改。符号“ < ”后的内容为 file1 中的内容。符号“ > ”后的内容为 file2 中的内容。即 file1 中的第 3 行内容为 file1，file2 中的第 3 行内容为 file2。8a9,10 指出 file2 中的内容比 file1 多出两行。

从该例可以看出 diff 比较后的动作有：

- 更改（c）：说明为了让两个文件保持相同，file 2（后面一个文件）应采取的动作，更改包括删除和替换；
- 附加（a）：说明为了让两个文件保持相同，file 1（前面一个文件）应添加的行；
- 删除（d）：说明为了让两个文件保持相同，file 1（前面一个文件）应删除的行。

例 将上例中的两个文件位置对换。

```
$diff file2 file1
3c3
<file2
---
>file1
6,7c6,7
<file22
<file222
---
>file11
>file11
9,10d8
<file2 difference1
<file2 difference2
```

例 如果文件 file1 在目录 dir1 下，文件 file2 在目录 dir2 下，则有：

```
$diff dir1 dir2
diff dir1/file1 dir2/file2
3c3
<file1
---
>file2
6,7c6,7
<file11
<file111
```



```
---
>file22
```

```
>file22
8a9,10
>file2 difference1
>file2 difference2
```

6.4.3　cmp 命令

cmp 命令用于按字节检查两个文件，显示两个文件不同数据的位置，若两个文件内容相同，则不显示任何信息。命令格式：

```
cmp [options] file1 file2 [skip1 [skip2]]
```

file1 和 file2 为要比较的两个文件的文件名。

其中选项：

-l：显示每个不同处和不同的字节内容。不同处以 10 进制表示第几个字节，不同字节内容以 8 进制表示；

-s：不显示不同的数据，只显示执行 cmp 指令后的 exit 值（exit value）。

- 若要比较的两个文件内容相同，则 exit value 为 0；
- 若要比较的两个文件内容不相同，则 exit value 为 1；
- 若要比较的文件不存在或没有存取权限，则 exit value 为 2。

在不带选项的情况下，cmp 将停止在第 1 个不同的字节上，报告这个字节所处的位置。

例　$cat file1

```
axyzg
7890
$cat file2
axyzg
7890
$cat file3
axyzg
as9u
$cmp file1 file2
$cmp file1 file3
file1 file3 differ:char 8, line 2
$cmp -l file1 file2
```



```
6     67 141
7     70 163
```

9 60 165

在输出中，字节个数应该算上换行符，所以两个文件中不同的第 2 行第 1 个字符是第 6 个字符，然后是第 7 个字符，第 9 个字符。67 是数字 7 的 8 进制值，141 是字符 a 的 8 进制值；70 是数字 8 的 8 进制值，163 是字符 s 的 8 进制值；60 是数字 0 的 8 进制值，165 是字符 u 的 8 进制值。

注意　数字 0 的 8 进制值为 60，1 的 8 进制值为 61，……7 的 8 进制值为 67，8 的 8 进制值为 70，9 的 8 进制值为 71。

字符 a 的 8 进制值为 141，b 的 8 进制值为 142，c 的 8 进制值为 143，……

6.5 awk

6.5.1 awk 介绍

awk 是 Shell 提供的工具，可以参与 Shell 程序设计，是一种自解释的程序设计语言。其名称来源于贝尔实验室的开发者 Alfred.Aho、Peter.Weinberger、Brain.Kernigham 的姓。awk 能从一个或多个文本文件或字符串中逐个记录或逐行扫描；将每个记录与匹配模式相比较；当发现匹配格式时，抽取数据或格式化报文，或执行相应的文本操作；awk 并不改变输入文本文件中的内容。与 sed 相比，sed 只能按编辑的匹配格式读取文件，而 awk 却是程序设计语言；与 grep 相比，grep 只能查找，而 awk 却能进行操作。

awk 借鉴了 C 语言的结构，包含有：

- 条件与循环执行语句；
- 数值变量与字符串变量；
- 正则表达式；
- pringf 语句。

6.5.2 使用 awk

awk 的通用格式为：

awk [-F field-separator] [parameter …] ['prog'] [-f awk-script-file] [input_file…]

其中参数：

-F field-separator：-F 选项之后的 field-separator 为字段分隔符，默认的字段分隔符是空格、制表符；

parameter：为不同的变量赋值；

prog：为匹配的字符串，默认记录为行，如果 prog 内容为空（即' '），则为所有行；

-f awk-script-file：表示用-f 选项之后的 awk 脚本文件存放 awk 命令及操作结果；

input_file：为要处理的文件序列，如果未指定输入文件，则 awk 接受标准输入（即键盘输入），并将结果显示在标准输出（即屏幕）上。

1. 基本使用方式

在 Shell 中直接用 awk 命令。

命令格式：awk [-F field-separator] [‘ prog ’] [input_file…]

使用 awk 时一个记录是一行，以换行符结束。记录由字段构成，空格、制表符是默认的字段分隔符。

如果指定了-F 选项,awk 按指定的分隔符分隔字段方式每次读一条记录或一行。

如果未指定-F 选项，awk 按分隔符为空格或制表符分隔字段方式每次读一条记录或一行。

如果一个记录中有 NF 个字段，awk 分别用$1，$2，…，$NF 访问这些字段，特殊值$0 表示整行。

例　$awk '$0 ~ /abc/' myfile

显示 myfile 文件中含 abc 的所有行。

例　在系统文件/etc/passwd 中存放了系统的用户信息，每个字段用冒号作分隔符，即“ -F: ”。

```
$cat /etc/passwd
root:hl32xkzs96fj:0:1::/bin/sh
…
lilx:dkfjiixx48kd:8:34:uu-fh:/export/home/lilx/:/bin/sh
shld::9:34:bj:/export/home/shld/:/bin/sh
zhjg:llgujt59fjjf02ff:10:34:bj:/export/home/zhjg/:/bin/sh
$
```

下面的操作命令查看系统用户账号：

```
$awk –F: '{print $1}' /etc/passwd
root
…
lilx
shld
```

```
zhjg
shld
$
```

例 一个公司由4个部门：销售（sales）、生产（producer）、开发（develop）、管理（manage）组成。管理数据文件中的每个记录包括3个字段：部门代码（ID）、部门名称（NAME）、部门人员数（NUMBER）。管理数据文件内容afba.dat如下：

```
1        sales          10
2        producers      50
3        develops       08
4        manages        09
```

如果要查看开发部门的人员数，用awk命令：

```
$awk '$2=="develops" {print}' afba.dat
3        develops       08
$
```

如果要查看人员数大于10的部门，用awk命令：

```
$awk '$2>10 {print}' afba.dat
2        producers      50
$
```

2．awk模式

awk语句使用包含模式。

模式分为：关系表达式、逻辑表达式、正则表达式和BEGIN-END。

关系表达式和逻辑表达式的操作符如表6.3所示。表达式使用时须用小括号括起。

表6.3 awk允许的测试

操作符	含义
!	逻辑非
&&	逻辑与
‖	逻辑或
x==y	x等于y
x!=y	x不等于y
x>y	x大于y
x<y	x小于y
x>=y	x大于等于y

x<=y	x 小于等于 y
x~re	x 匹配正则表达式 re
x!~re	x 不匹配正则表达式 re

例　在/etc/passwd 中的第二个字段是加密后的用户口令，用模式匹配$2==" "测试该字段是否为空，查看是否有用户没有口令。

```
$awk –F: '$2 == " " ' /etc/passwd
shld::9:34:bj:/export/home/shld/:/bin/sh
```

从上面运行的结果可见，用户 shld 没有口令。

例　下面查询/etc/passwd 中 uid（passwd 中的第 3 字段）大于等于 8 的用户信息。

```
$awk –F: '{if ($3>=8) print $0}' /etc/passwd
lilx:dkfjiixx48kd:8:34:uu-fh:/export/home/lilx/:/bin/sh
shld::9:34:bj:/export/home/shld/:/bin/sh
zhjg:llgujt59fjjf02ff:10:34:bj:/export/home/zhjg/:/bin/sh
…
$
```

正则表达式用//括起来，规则与 gred 的正则表达式基本相同，格式如下：

```
$awk '/正则表达式/' 文件名…
```

正则表达式常用的有：

```
~/string/        匹配含有字符串的行
~/[0-9].*/       匹配含有数字的行
~/[A-Z].*/   匹配含有大写字母的行
~/[a-z].*/       匹配含有小写字母的行
~/^.$/       匹配只有一个字符的行
!~/[0-9].*/ 匹配不含数字的行
!~/[A-Z].*/ 匹配不含大写字母的行
!~/[a-z].*/ 匹配不含小写字母的行
```

例　查找/etc/passwd 文件中用户账号为“ zhangjg ”的行。

```
$awk –F: '$1 /zhjg/'  /etc/passwd
zhjg:llgujt59fjjf02ff:10:34:bj:/export/home/zhjg/:/bin/sh
$
```


注意　测试正则表达式时可以匹配[]内的任意字符或单词。

例　使用[]符号查询大小写信息。查询有 Intel 和 intel 的所有记录用表达式

'/[Ii]ntel/'

```
$cat personalpc.txt
student01  01  intel486  32M   20G
student02  02  Intel586  64M   40G
student03  03  Intel586  128M  200G
student04  04  intel486  32M   20G
student05  05  intel486  16M   4G
$
$awk '/[Ii]ntel/' personal.txt
student01  01  intel486  32M   20G
student02  02  Intel586  64M   40G
student03  03  Intel586  128M  200G
student04  04  intel486  32M   20G
student05  05  intel486  16M   4G
```

BEGIN-END 用在模式的参数中。使用 BEGIN 显示变量和初始化变量，在 BEGIN 后的括号中列出的操作在 awk 开始扫描之前执行；用 END 显示最终结果，在 END 后的括号中列出的操作在 awk 扫描完全部的输入之后执行。

例　下面是 books 文件：

```
Author    Title   Prices
Dell      Mices   29.85
Dics      Cats    22.56
Jack      Tigers   18.98
Smith     Lions   20.36
```

下面用 awk 工具打印出价格清单，并在价格清单上打印出日期。

```
$awk
>'BEGIN {print "Price List" ;
>print " ' " 'date' " ' "}
>NR>1{sum +=$3;print$2, "$" $3}
>END {print "Average price=$", sum/(NR-1)}' books
Price List
Sun Aug 11 06:45:35 GMT 2002
Mices  $29.85
Cats   $22.56
Tigers $18.98
```

```
Lions  $20.36
Average price=$22.94
```

3．awk 的预定义变量

在 awk 中变量无须定义，直接使用。其数据类型是数值、数组或字符串，依据变量第一次在 awk 语句中出现的情况和上下文确定。变量类型不确定时默认为字符串。awk 将数值型变量初始化为 0，字符串型变量初始化为空" "。

在 awk 中预定义的变量如表 6.4 所示。

表 6.4　awk 中的预定义变量

变量名	含义
ARGC	命令行参数的个数
ARGV	命令行参数的数组
ENVIRON	支持队列中系统环境变量的使用
FILENAME	string=当前输入的文件名
FNR	在当前文件中当前记录数（对输入文件起始值为 1）
FS	输入字段分隔符（默认为空格或制表符）
NF	当前记录的字段数
NR	当前记录数（为全部输入文件）
OFMT	数值的输出格式（默认=%.6g)）
OFS	输出字段的分隔符（默认为空格或制表符）
ORS	输出记录的分隔符（默认为换行）
RS	控制记录的分隔符（默认为换行）

例　练习使用 NF、NR、FS、FILENAME。

```
$awk '{print NF,NR,FS $0} END {print FILENAME}' personalpc.txt
5 1   student01  01  intel486  32M   20G
5 2   student02  02  Intel586  64M   40G
5 3   student03  03  Intel586  128M  200G
5 4   student04  04  intel486  32M   20G
5 5   student05  05  intel486  16M   4G
personalpc.txt
$
```

4．awk 操作

awk 除了具有简单的文本处理功能之外，真正的实力还在于当模式匹配成功后能采取的相应操作。操作中除默认打印外，还能对输入内容作字符串运算、数值运算（如计数、累计求和、求平均数等）、函数运算。

- 字符串运算：包括关系运算、逻辑运算和赋值操作符。

关系运算：< <= > >= == != >> ~ !~

逻辑运算：! && ||

赋值操作符：= += -= *= /= %= ^ ++ --

- 数值运算：+ - * / %
- 函数运算：awk 有许多内部函数，如表 6.5 所示。

表 6.5 awk 中的函数

函数	含义
cos(表达式)	求表达式的余弦
sin（表达式）	求表达式的正弦
exp(表达式)	对表达式取自然指数
log（表达式）	求表达式的自然对数
int（表达式）	求表达式的整数部分，向 0 截取
length（表达式）	求字符串的长度
index(字符串 1，字符串 2)	字符串 2 在字符串 1 中的位置；不在则返回 0
split（字符串，a，c）	以字符 c 为分隔符将 s 分隔成 a[1]…a[n]并返回 n
sprintf（格式，…）	格式化
sub（字符串，m）	用$0 中最左边最长的子串代替 m
substr（字符串，m）	求起始于位置 m 的字符串的子串
substr（字符串，m，n）	求起始于位置 m 的字符串的 n 个字符的子串
gsub(字符串，m)	在整个$0 中用 m 替代字符串
gsub(字符串，m，n)	在整个 n 中用 m 替代字符串
match(m，n)	测试 m 是否包含匹配 n 的字符串

例 将前面 personalpc.txt 文件中所有的 cpu 改为 586。

```
$awk 'gsub(/tel486/,tel586) {print $0}' personalpc.txt
student01  01  in  32M  20G
student04  04  in  32M  20G
student05  05  in  16M  40
$
```

例　$awk 'BEGIN {print split("1111#2222#3333#4444", abcdarray, "#")}'

```
4
```

split 返回的 4 是数组 abcdarray 的下标数。数组取值如下：

```
abcdarray[1]=1111
abcdarray[2]=2222
abcdarray[3]=3333
abcdarray[4]=4444
```

例　$awk 'BEGIN {print match("ABCDXYZ", /x/)}'

```
0
$awk 'BEGIN {print match("ABCDXYZ", /X/)}'
5
$
```

执行结果为 0 表明没有找到匹配；执行结果为 5 表明找到的匹配字符在所找字符串中的首字符的位置为 5。

例　length 求字符串的长度。

```
$awk 'length ($0)>20 {print NR}' personalpc.txt
1
2
3
4
5
$
```

6.5.3　awk 脚本

1. awk 脚本及应用

在数据处理上可以将数据的预处理（初始化）、数据处理、处理结果、后处理写入一个脚本中，该脚本称为 awk 脚本。用编辑器 vi 可以创建 awk 脚本。

如果 awk 脚本名为 file.awk，则该 awk 脚本应用形式为：

```
$awk file.awk inputfile
```

其中 inputfile 为要处理的数据文件。

在一个 awk 脚本中，包括模式（patterns）与动作（actions）两部分。模式是正则表达式，动作表示当模式匹配为真时应该执行的语句或指令，与 C 语言完全相似。

在一个 awk 脚本中，模式后面跟一个动作，用大括号{}把每个动作括起来。

如果模式在当前输入中匹配为真，则执行相应动作。

如果 awk 脚本中没有模式部分，则动作在每一行都执行。

如果 awk 脚本中没有动作部分，则默认动作是将每一行打印到显示器。

例　一个商店的商品种类有相机（camera）、扫描仪（scanner）、传真机（electrograph）、打印机（printer）、电话机（telephone）。销售数据文件每条记录包含 3 个字段：月（numMon）、商品名（Name）、销售额（Sales）。数据文件 comm.dat 的内容为：

```
1        cameras          80260
1        scanners         12570
1        electrographs      13250
1        printers          7920
1        telephones         3450
2        cameras          68150
2        scanners         10550
2        electrographs     10230
2        printers          12360
2        telephones         6710
```

下面编写两个 awk 脚本计算扫描仪的总销售额和所有商品的总销售额。

```
$cat scanner.awk
#This is the scanner.awk script
{
  if ($2 == "scanners")
  {
    scannersales += $3
  }
  print "This is scanner sales:$" , scannersales
}
END
$
$cat commsales.awk
#This is the commsales.awk script
$3 == 0 {next}
{
```

```
  totalsales += $3
}
END
{
  print "This is total sales:$" , totalsales
}
$
$awk  – f scanner.awk comm.dat
This is scanner sales:$23120
$awk  – f commsales.awk comm.dat
This is total sales:$225450
$
```

2．awk 脚本中的控制语句

上例 awk 脚本中出现的 next 语句是一个控制语句，控制脚本的执行。

next 语句的作用是：结束当前记录的处理并开始下一条记录的处理。如果没有更多的记录需要处理，当指定了结束动作时，则将控制转换到结束动作，立即跳到脚本结尾。

除了 next 语句之外，awk 脚本中用于控制执行的语句还有 getline 和 exit。

getline 语句也是读一条记录，与 next 不同的是，它不会返回到主体的开始处，而会将 getline 语句执行结果应用到 getline 语句之后的动作和语句，继续执行脚本。

awk 脚本必须有一个输入，要么为标准输入，要么为传递参数的文件。传递参数的文件可以自动被脚本处理或使用 getline 语句读取，当读到文件结尾时，awk 将控制传递给结束部分。如果没有结束部分，则终止执行。

getline 语句相当于一个函数调用，具有如下特点：

- 输入可以直接指向$0 或单独的变量。如果不指明，默认指向$0。
- getline 返回一个可以被估计的值：

1：读记录成功。

0：文件结尾：由于 getline 读文件时并不自动跳到文件结尾的结束处理，当语句的其他部分没有处理时，应该测试返回值 0 和结束处理分支。

-1：读错误。

- 可以从另一个文件输入，格式为：

  getline Variable<file〉

其中 Variable 是可选的，如果没有提供该变量，默认为$0。

例 原数据文件 lines.dat 中每条记录包含两个字段。

```
Number    ID
1         stu01
2         stu02
3         stu03
4         stu04
5         stu05
6         stu06
```

现在要求相邻两行相互交换，即：第 1 行与第 2 行交换，第 3 行与第 4 行交换，第 5 行与第 6 行交换。

用 getling 语句实现。

```
$cat interch.awk
#This is the interchange awk script
{
  if ((getline eventline) == 1
  {
    print eventline
    print $0,$1
  }
  else
    print $0,$1
}
$
```

脚本中首先将偶数行读入变量 eventline，然后返回值测试文件结尾。当 getline 成功返回 read(1)，先打印 eventline，再打印$0。

如果数据文件有奇数行，则最后一行不能和相邻行交换，只需将最后一行打印出来就行；如果数据文件有偶数行，则最后一行也能和上一个相邻行进行交换，顺利结束。

```
$awk –f interch.awk lines.dat
Number        ID
2             stu02
1             stu01
4             stu04
```

```
3          stu03
6          stu06
5          stu05
```

exit 语句用于终止脚本，常常用于发生错误时的脚本终止。如果指定了结束模式，则执行结束语句后退出；如果没有指定结束模式，则立即退出脚本。另外，当结束模式中使用 exit 时，脚本会立即终止，不执行语句的其他部分。

3. awk 脚本中的循环语句

C 语言中的循环语句 while、for、do-while 在 awk 脚本中仍然可以使用。

6.6 本 章 小 结

本章主要介绍了 UNIX 下强大的文本过滤和处理工具，以往需要花费几个小时编写 C 语言程序才能达到的效果，现在用 UNIX 提供的实用程序只需要几分钟就可以完成，甚至有时用一行代码就可以解决问题，而且可读性和可维护性都要好得多。

在 UNIX 实用程序中 grep 最简单、应用最多，如果 grep 实用程序不可用时，可以用 sed 实用程序实现同样的功能。awk 可以解决许多复杂的问题，特别是在对数据文件的处理上最为突出。

实际应用中，我们常常希望能发现同一功能的两个文件在内容上的差别，这时要用文件比较命令 comm、diff、cmp。

上 机 练 习

1. 编写一个 grep 命令，从文件 file 中选择只有三个字符的行。

2. 编写一个 grep 命令，将文件 file 中含有字符 UNIX 的行放入到一新文件中。

3. 创建文件 sch001。文件中各行用制表符 Tab 隔开：

```
Name     ID     Years    Salary
Juan      1352    10       19000
Anne     2135    08       17000
George    3578    11       20000
Bell      4136     12       25000
```

(1) 用命令查看 Juan 的 ID。

(2) 用命令查看 Juan 的 Salary。

(1) 编写一个 Shell 脚本计算出全部人员的 Salary 总和。

4．创建文件 sch002。文件中各行用制表符 Tab 隔开：

名　姓	年龄	电话号码
John Adams	35	8535-1433
Anne Blue	63	8515-4288
John Smith	51	8536-1466
Ben White	76	8535-1453
George Bush	53	8545-6433
Ted White	87	8435-2433
Janet Blue	79	8478-1678

(1) 使用 sed 命令重新组织该文件并让姓在前名在后，姓、名、年龄之间用一个空格隔开。

(2) 按年龄从大到小重新排列。

(3) 将电话号码按字典顺序重新排列。

5．下面是学生的考试成绩：

Num	math	chinese	chem	physical
3241	89	78	85	88
3242	87	88	89	80
3243	98	87	91	90
3244	78	89	80	82
3245	70	76	69	67
3246	75	67	90	76

(1) 编写 awk 脚本计算出每个人的平均成绩及所有人的平均成绩。

(2) 编写 awk 脚本将每人的所有科目成绩按等级转换为以下五个等级：

- 优：91~100
- 良：81~90
- 中：71-80
- 及格：61~70
- 不及格：60 分以下

6．创建下列文件并用制表符分隔字段。

First Name	Last Name	Hr Wage	Work Hr
Dan	Red	22.00	31
George	Blue	20.00	31
Mary	Green	19.00	24
Bob	White	19.00	23
Steve	Gold	18.00	22

```
John        Brown       18.00        12
Bruce       Silver       17.00        11
Susan       Gold        17.00        0
John        Purple       15.00        8
Geoge       Bush        12.00        5
```

(1) 编写一个 awk 脚本打印出工作时间为 0 的员工的全名。

(2) 编写一个 awk 脚本打印出小时工资小于 17 的员工的全名。

(3) 编写一个 awk 脚本打印出工作时间在 20~25 的所有员工名。

(4) 编写一个 awk 脚本打印出 George Blue 的总薪水。

(5) 编写一个 awk 脚本打印出全部员工的总薪水。

习　题　六

1. 说明 grep 家族命令：grep、egrep、fgrep 之间的区别。

2. 在 UNIX 系统中有了 grep 为什么还要用 sed?

3. 描述 awk 语句 next 与 getline 的差异？

4. 分别编写 awk 命令模仿 Shell 命令：

```
$cp file1 file2
$cat filetest
$head -30 filetest
$tail +30 filetest
```

5. awk 脚本文件是如何创建的？如何应用？在 awk 脚本中如何使用注释？

第 7 章　UNIX 软件开发工具

UNIX 系统支持标准的软件开发工具，如 FORTRAN、C 及 JAVA 等。在这些语言中，C 语言与 UNIX 系统关系特别紧密，UNIX 核心程序的绝大多数都由 C 语言编写，C 是 UNIX 的系统语言。所以学会在 UNIX 系统中使用 C 语言是非常必要的。本章内容主要是针对 C 语言的编译程序、连接程序、程序加载和程序调试等开发环境，C 语言本身的学习不包括在此。

本章主要内容

- C 程序处理过程
- C 语言编辑器 CC
- 程序维护 make
- 程序调试
- 源代码控制系统 SCCS
- 其他的 C 程序设计工具。

7.1　C 程序处理过程

在一般的程序开发过程中，一个源文件要成为可执行文件需经历多步处理：前置处理器、编译器、优化器、汇编程序器、连接和加载，最后才成为可执行文件。如图 7.1 所示。

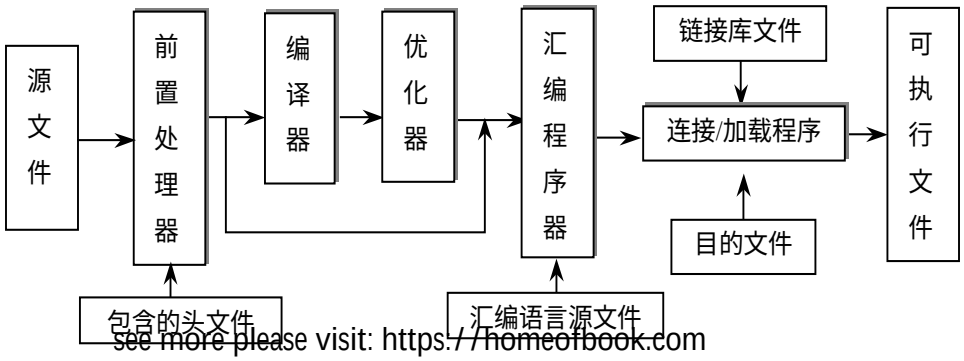

图 7.1　程序处理步骤

UNIX 的 C 开发环境主要包括前置处理器、编译器、优化器、汇编程序器、连接和加载部分，它们以运行 C 程序开发包 CC 作为标志，其作用是将 C 源程序处理成可执行程序。

在 CC 的处理过程中，文件、汇编文件等不同类型的文件可以一起使用。用文件扩展名可以区别不同的文件类型，表 7.1 列出了文件类型与扩展名的对应关系。

表 7.1 文件类型与文件扩展名

文件类型	扩展名
C 源文件	.c
包含文件	.h
前置处理之后的 C 文件	.i
汇编文件	.s
目标文件	.o
链接库文件	.a
C 语言程序打印文件（C 的源列表）	.S
汇编程序打印文件（汇编列表）	.L
连接成打印文件	.map
FORTRAN 的源文件	.f

除 CC 开发包之外，UNIX 还提供了程序调式软件包、程序性能监控包等。

7.2 CC 命 令

7.2.1 CC 命令格式

CC 又称为 C 编辑器。用于开发 C 语言程序。

命令格式：cc [-option] source-file object-file [-l library……]

其中：source-file 是 CC 的 C 源文件名；object-file 是 cc 命令的输出文件名。

整个 cc 命令包含的处理过程分为三个阶段：

第一阶段是前置处理，由前置处理器展开文件的包含、宏、条件编译和文本替换；

第二阶段是编译处理 parse，将源代码翻译成目标机器的汇编语言，将汇编语言内容翻译成自由文本语法树形式，并建立符号表；

第三阶段转换成机器语言，此时的结果为由机器码和未解析的引用表组成的目标模块。最后阶段实现优化功能，将目标模块与其他库文件模块连接在一起成为可执行模块。

7.2.2　前置处理

前置处理在 C 编译过程中主要处理以#include 句子为内容的包含文件、宏替换和条件编译。通常情况下前置处理器会在系统的默认目录中寻找包含文件。

如果程序中出现有#include<stdio.h>语句,则表示程序要将文件 stdio.h 包含进来，编译 cc 时会在系统的默认目录中寻找该文件。

如果程序中出现#define message string 语句，则是宏替换，表示程序中出现 message 的部分要用 string 替换；比如#define AB 256，表示程序中出现 AB 的部分要用 256 替换。

如果作 cc 编译时加入条件#if、#ifdef、#ifndef、#else、#endif，则表示编译时要考虑条件。格式为：

```
#if 条件表达式（不为 0）
  语句 1
#else
  语句 2
#endif
```

或者：

```
#ifdef 字符串（已经定义）
  语句 1
#else
  语句 2
#endif
```

在作前置处理时可以有选项：

-I pathname：描述包含文件要搜索的路径，如果在该路径下找不到包含文件，则在默认路径下搜索；

-D identifier：定义标识符 identifier 的值，默认值为 1。此处定义的优先级高于程序中的定义；

-P：将前置处理后的程序名取为与源程序具有相同的前缀名，而用 i 作扩展；

-E：生成前置处理文件的同时将前置处理结果由标准输出设备（屏幕）输出；

-C：前置处理的输出文件中保留注释语句；

-U identifier：取消标识符 identifier 的定义，与-D identifier 相反。

7.2.3 编译程序

编译程序将源代码内容翻译成自由文本语法树形式，并建立符号表。该过程的选项有：

-dc：指出编译是动态还是静态，此选项会传递给 ld 程序；

- -dy 表示动态编译，这是默认方式；
- -dn 表示静态编译。

-g：该选项使得 cc 之后生成的可执行程序能用 debug（sdb）调试；

-c：只产生 object 文件，该 object 文件不用连接器 linker 即可执行；

-o name：对可执行文件命名，使 cc 产生的可执行文件名不是默认的 a.out；

-O：编译时采取最优化方式；

-S：保留 cc 所产生的汇编语言寄存文件，并且该文件具有与源文件相同的前缀名而后缀是.s；

-V：打印编译器、优化器、汇编语言和连接编辑器的版本；

-L dir：额外加上 ld 要寻找链接库的目录；

-l name：最后搜索链接库，传给 ld。

例 下面的命令编译程序 exer.c 并生成一个可执行程序 exer。

```
$cc –o exer exer.c
```

如果不用选项-o 则生成的可执行程序名为 a.out。

例 程序包含两个模块 mainexer.c 和 subexer.c，现在用 cc 命令编译 mainexer.c 和 subexer.c,并将编译后的目标模块直接连接成一个可执行程序 exer。

```
$cc –o exer mainexer.c subexer.c
```

例 现在用 cc 命令编译并生成一个目标文件 exer.o。

```
$cc –c exer.c
```

例 连接目标模块 mainexer.o 和 subexer.o 生成可执行的程序 exer。

```
$cc –o exer mainexer.o subexer.o
```

7.2.4 UNIX 连接器（Link Editor）

除了可以用 cc 命令连接之外，还可以直接用 ld 命令将编译后的程序目标模块和外在目标模块组合在一起形成可执行文件，这一过程称为连接。ld 连接过程可以分析模块间的调用关系及搜索程序调用的链接库模块。ld 的连接分为静态和动态两种，静态连接把有关链接库（通常是含.a 后缀的）中的模块直接连接到可执行文件中，执行时对运行环境的依赖性较小；动态连接只记录有关链接库中的模块调用关

系，在程序执行时动态地取链接库中有关模块，这样连接可执行文件比较小，但对运行环境依赖性大。ld 连接结果默认为文件 a.out，用选项 o 可以指定连接结果文件名。

例　下面用 ld 将 mainexer.o、subexer.o、/usr/sys/lib/libabds.o、/lib/acdit.lib 连接成目标模块 a.out。

```
$ld mainexer.o subexer.o /usr/sys/lib/libabds.o /lib/acdit.lib
```

7.2 5　UNIX 文件库

在 UNIX 程序开发中，可以利用 ar 命令将单个或多个目标文件结合成一个库文件，方便自己或他人调用。

ar 命令格式：ar key filelib [files]

其中：

filelib 为结合后的文件库名。

files 为要结合的文件名。

key 描述 ar 所要执行的动作，key 的值和意义如表 7.2 所示，这些值可以组合。

表 7.2　Key 值及其意义

Key 值	意义
d（delete）	从 filelib 中删除文件。
r（replace）	将 files 加入或取代文件 filelib，加入时加在 filelib 最后。
q（quick）	将 files 加在文件 filelib 最后，加入过程中不检查 files 在 filelib 中是否存在。
t（type-name）	显示打印 filelib 内的所有文件名称。
p（print）	打印全部文件的内容。
m（move）	将 files 移到 filelib 的最后。
x（access）	从 filelib 获取文件，加上 files 选项后则包含 files 的内容。获取的文件存到当前目录下，获取后不影响文件库内容。
v（verbose）	新增文件库时会一个一个显示文件信息。

例　下面将 exwec1、exer2、exer3、exer4 结合到文件库 exer.a 的最后并一个一个显示文件信息。

```
$ar rv exer.a exer1 exer2 exer3 exer4
ar:creating exer.a
r - exer1
r - exer2
```

```
r – exer3
r – exer4
$
```

下面显示库 exer.a 中的所有文件名称：

```
$ar t exer.a
exer1
exer2
exer3
exer4
$
```

从库 exer.a 中删除 exer1：

```
$ar d exer.a exer1
$ar t exer.a
exer2
exer3
exer4
$
```

从库 exer.a 中获取文件 exer1：

```
$ar x exer.a exer1
$ls –l exer1
-rw-r—r--   1  liuxm others    124 Oct 10:34:34 exer1
$
```

7.3 程序维护 make

在程序设计模块化的环境下，一个程序通常由若干个子程序构成，相应地，一个模块通常由若干个子模块构成。在程序开发过程中，为了保证每次编译连接后的结果软件始终是由全部最新模块构成，不可能每次都把所有的模块重新编译一遍，也不可能一个一个分别编译。程序维护 make 通过管理每个模块文件的修改时间来识别程序模块的修改情况，在每次编译时自动判断应对哪些模块重新编译，以保证编译连接后的结果软件始终是由最新模块构成。

程序维护 make 的实现借助于文件目录中存在一个文件，该文件由程序员创建，默认文件名为 makefile 或 Makefile，用户也可以指定别的文件名。在该文件中有各模块之间的相互依赖关系及编译连接执行的命令。

7.3.1　makefile 文件

在 makefile 文件中有依赖行和规则行。

依赖行强调程序模块由哪些子模块构成，这些子模块的状态依赖于各自对应的子程序状态。如图 7.2 所示，最终运行目标程序 exer 由模块 exer1.o、exer2.o、exer3.o、exer4.o 连接构成，而模块 exer1.o 依赖于 exer1.c 和 head1.h、exer2.o 依赖于 exer2.c、exer3.o 依赖于 exer3.c 和 head3.h、exer4.o 依赖于 exer4.c。

规则行强调的是实现编译或连接。

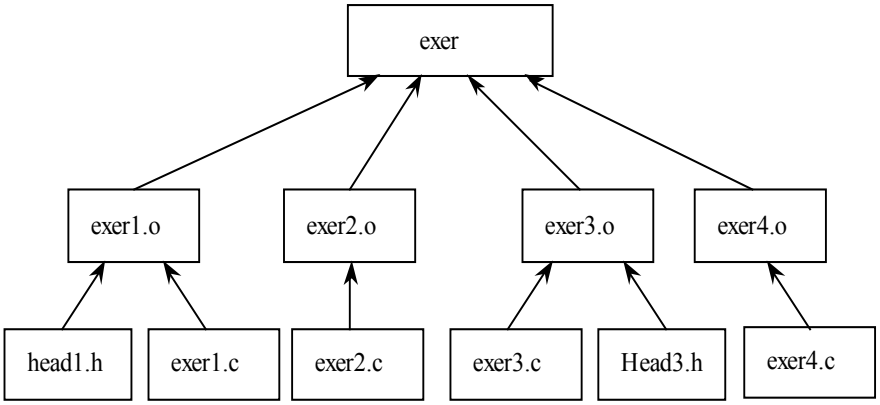

图 7.2　模块间的依赖关系

makefile 文件可以用 vi 编辑器创建。文件中的依赖行和规则行格式为：

```
target:components
[tab]rule
```

注意规则行前面的空闲部分为 Tab 键而非空格键。冒号前的文件名为目标，目标依赖于冒号后的部件。规则行说明当目标比部件旧时用规则更新。

当依赖和规则要放到同一行中时必须用分号（；）隔开。

对上面的例子，文件 makefile 的基本内容为：

```
exer:exer1.o exer2.o exer3.o exer4.o
   cc -o exer exer1.o exer2.o exer3.o exer4.o
exer1.o:exer1.c head1.h
   cc -c exer1.c
exer2.o:exer2.c
   cc -c exer2.c
exer3.o:exer3.c head3.h
   cc -c exer3.c
```

```
exer4.o:exer4.c
    cc -c exer4.c
```

在 makefile 文件中起始字符是“ # ”的行表示注释行。

7.3 2 运行 makefile 文件

若取名为 makefile，则运行时用

```
$make
```

即可；

若取名为其他，如 file，则运行时用

```
$make -f file
```

makefile 文件运行时通常从第一行开始，所以一般情况下将最后要生成的应用程序放到第一依赖行。如果只要生成最新的中间行，如上例只要更新 exer1.o 则可以用

```
$make exer1.o
```

7.3 3 makefile 中的宏应用

在 make 命令后或 makefile 的依赖行中可以应用宏。常用宏来代表一些在多个地方使用并且还可以变化的文件名或选项，以便节约重复修改所花的时间及避免遗漏，其作用类似于 C 语言中的 define。

makefile 中的宏有用户定义和系统内部定义两类。用户定义宏必须在 makefile 中或命令行中明确定义，形式为 name=value；系统定义宏与用户定义宏不同，不能由用户定义。系统 Shell 中的环境变量的值可以作为系统定义宏的值。

宏使用时需在宏名前加$符号，如果宏字符多于一个，用括号“ () ”将宏括起来，宏名一般要大写，在宏名中不能使用 Tab 键和冒号。

例

```
OBJECT=exer1.o exer2.o exer3.o exer4.o
exer:$(OBJECT)
cc -o exer $(OBJECT)
```

当在 make 命令后定义的宏名与文件 makefile 中定义的宏名相同时，在 make 命令后定义的宏名会覆盖文件 makefile 中定义的宏名。如：

```
$make "OBJECT= exer3.c head3.h"
```

则在 makefile 中 OBJECT 就不再是“ exer1.o exer2.o exer3.o exer4.o ”了。

有几个常用的系统定义宏，由于这些宏名的前面均有标记“ $ ”，在使用时还需在其前面再加上“ $ ”，即有两个“ $ ”符号。

常用的系统定义宏有：

$*：表示当前目标名去除后缀，如当前目标是 exer.o，则$*表示 exer；

$@：表示当前目标的全路径名。用于用户定义的目标名依赖行；

$<：表示比给定的目标文件时间标记更新的依赖文件名；

$?：表示比目标文件时间标记更新的依赖文件名列表，只在 makefile 文件中使用显示规则时才用。

例　TARGET=hello

```
OBJS=hello.o

$(TARGET):$(OBJS)
@echo ---------------
@echo Making $@
@echo ---------------
cc -o $@ $(OBJS)
hello.o:hello.c hello.h
@echo ---------------
@echo Making $@
@echo ---------------
cc -c $*.c
clean:
@rm -fr $(OBJS)
@rm -fr $(TARGET)
@echo $(TARGET) cleaned......
```

7.4　调试程序（dbx）

程序调试器是程序运行时监视并控制程序的工具。UNIX 支持源程序级和目标程序级的程序调试器，主要的调试器有：dbx、sdb。

dbx 由加州大学 Berkeley 分校开发，sdb 由贝尔实验室开发。dbx、sdb 调试器都支持用户单步执行程序并监控指定量的变化。用程序调试器首先要求在程序编译时带选项-g，该选项的意义表示在可执行模块中加入调试器控制。

调试器控制的主要功能有：

- 启动或终止程序的运行（run，stop）；
- 使程序单步运行；

- 设置或清除程序断点；
- 显示或设置变量值；
- 跟踪语句执行（trace）；
- 跟踪变量值的变化。

例 $cc -g -o exer exer.c

在 dbx 下运行 exer：

```
$dbx exer
dbx>list 1,10              (列出源程序 1-10 行)
dbx>stop at 15             (在源程序中第 15 行设置断点)
dbx>clear 10               (取消在源程序中第 10 行设置断点)
dbx>stop at f1             (在源程序中的函数 f1 处设置断点)
dbx>run                    (开始 exer 程序运行)
dbx>stop                   (单步执行)
dbx>set var1=87                (设置变量 var1 的值为 87)
dbx>print var1             (停在某一断点时打印变量 var1 的值)
dbx>print array[x1+x2]     (计算 x1+x2 并打印数组元素的值)
dbx>continue               (从断点处继续执行)
dbx>help                   (帮助信息，列出 dbx 的所有命令)
dbx>help run               (帮助信息，列出 dbx 的 run 命令)
dbx>quit                   (退出 dbx)
```

core 文件是在程序执行出错（即出现“coredump”）时生成的一个内存映像文件。在 core 文件中有出错时的内存地址，所以 core 可以帮助程序设计人员查询出错语句，找到程序错误原因。在用户环境配置文件.profile 或.cshrc 中设置。如果设置成

```
limit coredumpsize = 0
```

则程序出现“coredump”时不生成 core 文件。如果设置成

```
unlimit coredumpsize
```

则程序出现“coredump”时会生成 core 文件。

7.5 源代码控制系统 SCCS

源代码控制系统（Source Code Control System）是一种在程序开发和程序维护过程中帮助程序设计人员对源代码进行管理的实用程序。SCCS 的主要功能是自动跟踪置于 SCCS 管理之下的不同版本的程序差异并对程序实施一定的安全保护。如果

一个程序置于 SCCS 的管理之下，就必须通过 SCCS 命令来恢复、修改或更新，SCCS 记录文件信息的变化并指出引起变化的修改者、修改时间、修改原因。为了节约存储空间，SCCS 只保留一个基本的程序及每一次作的修改记录，通过 SCCS 命令可以恢复程序的任一版本。

7.5.1　admin 命令

利用 admin 命令可以把程序放在 SCCS 管理系统中。admin 命令的格式为

```
admin  -n  -ifile s.sccsname
```

其中：file 是要放入 SCCS 中的程序文件；s.sccsname 是该文件的 SCCS 格式名称，被称为编码文件，用户不能直接访问。所有 SCCS 格式文件都以 s.开头。

例　admin -n -iexer.c s.exer.c 将 exer.c 放入 SCCS 中，取名为 s.exer.c。

如果有多个程序要放入 SCCS 中则用循环语句完成。

例　将所有 C 源程序放入 SCCS 中。

```
for f in *.c
do
   admin  -n  -i$f  s.$f
done
```

7.5.2　get 命令

前面介绍 s.文件不能直接访问，要读写 s.文件必须用 get 命令。get 命令的格式为

```
get  -pe filename
```

其中：filename 是要访问的文件名称；选项 p 是检查程序文件的最后版本；选项 e 是获取文件以便修改，所获取的文件名是去掉了 s.的 SCCS 文件名。

例　下面显示存在于 s.exer.c 内最新的源程序版本。

```
$get  -p s.exer.c
$
```

下面由 s.exer.c 获取最新的版本，并将获取数据存于 exer.c：

```
$get  -e s.exer.c
1.1
new delta 1.2
19lines
$
```

下面只获取文件但不读写该文件：

```
$get s.exer.c
1.1
19lines
$
```

下面只获取所有文件但不读写这些文件：

```
$get s.*
1.1
19lines
$
```

7.5.3 delta 命令

修改完 SCCS 获取的文件后，在 SCCS 下用 delta 命令将新版的文件存回去。delta 程序寻找去掉 s.头的文件名，记录与源版本的差异处。每一次修改清单称为一个 delta，每一个 delta 都有一个由 2~4 个数字组成的标识符串，标志该 delta 的版本号，该标识符串就是该 delta 的 SID。SID 格式为

release.level.branch.sequence

其中，版本号为 release，是标识一组功能相似的 delta。如果程序作了重大的修改，则应更改版本号；等级号为 level，如果程序作了较大修改，则应更改等级号；分级号为 branch，如果程序作了一般修改，则应更改分级号；顺序号为 sequence，如果程序作了一些修改，则应更改顺序号。

例 $delta s.exer.c

```
comments ?fix timing bug
NO id keywords (cm7)
1.5
8 insert
2 delete
6 update
$
```

例 Solaris 操作系统的版本号为 2.6.x.x。

7.5.4 prs 命令

用 prs 命令获取 SCCS 文件信息。

命令格式：prs 文件名称

prs 将以反向（最新版本先印）方式显示文件的各个旧版本。

```
$prs s.exer.c
s.main.c:
D 1.7 2000/04//23 18:58:30 steue 2 100001/00000/00002
MRs:
COMMENTS:
fix timing bug

D 1.7 2000/04//23 18:58:30 steue 1 100001/00000/00002
MRs:
COMMENTS:
Date and time created 2000/04//23 18:58:30 by steue
$
```

图 7.3 是 SCCS 的命令使用。

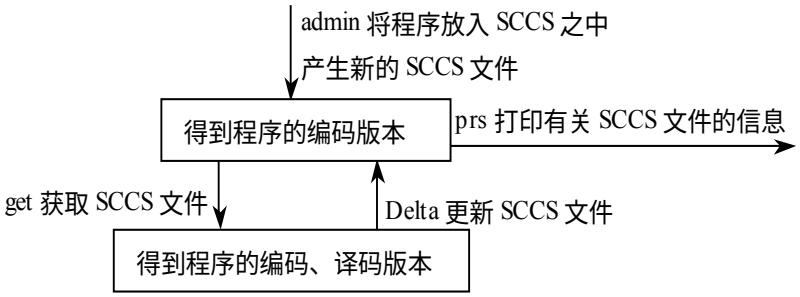

图 7.3　是 SCCS 的工作命令原理

7.6　其他的 C 程序设计工具

C 程序检查工具 lint 程序可以发现 C 源程序中的错误和不协调的地方。与 cc 相比，cc 的任务是尽快完成编译而不着重于程序检错；lint 的任务是尽可能地挑剔程序而不真正编译程序，对 C 源程序几乎是个挖掘者，完成严格的检查。

lint 主要从以下几个方面对源程序进行检查：

- 程序中多余的未访问的函数及变量；
- 无法执行的语句及子程序；
- 访问未赋初值的局部变量；

- 函数参数类型、个数、返回值的使用不一致；
- 类型、类型说明与引用、类型转换不一致；
- 指出废代码；
- 不可移植代码；
- 某些奇怪的结构形成的错误变种；
- 链接库函数使用不一致或不正确；
- 模块之间的不一致性。

lint 命令的格式为：lint filenames

例 用 lint 检查 exer1.c、exer2.c、exer3.c、exer4.c 中的函数之间的不一致。

```
$lint exer1.c exer2.c exer3.c exer4.c
```

lint 在执行过程中经过首次扫描找出函数内的问题；其次再经过第二次扫描找出各函数之间的问题。

7.7 本 章 小 结

C 语言永远是 UNIX 的精髓所在，本章介绍了 C 语言的编译和连接工具、调试工具和一些辅助工具。熟练掌握这些工具，是在 UNIX 环境下进行 C 语言程序开发的必要条件。

上 机 练 习

1. 用 cc、ld 完成 C 语言程序的编译和连接，并在代码中尝试使用条件编译。
2. 利用调试工具，对自己的程序进行调试，熟悉掌握一些常用调试命令的用法。
3. 为自己的程序编写一个 make 文件。
4. 熟悉各种辅助工具的使用。

习 题 七

1. 试说明怎样用 C 编辑器 cc 完成编译功能？
2. 试说明 make 作为程序管理器是如何使用的？
3. 试说明怎样用 dbx 调试程序？
4. 试说明什么时候用源代码控制系统 SCCS？

第 8 章　UNIX 进程管理及进程通信

UNIX 操作系统是多用户多任务分时操作系统，在 UNIX 系统中进程是资源分配和调度的基本单位，多任务的实现依赖于进程的并发执行。进程管理是操作系统的基本管理任务之一。

本章主要内容

- UNIX 进程及描述：主要介绍 UNIX 系统中的进程、进程状态转换、进程与区的概念、进程与进程表、进程与 U 区、进程映像。
- 进程控制：详细分析进程创建与终止过程的实现方式及进程调度思想。
- 进程间通信：主要介绍 UNIX 系统提供的进程间通信工具——信号、管道、消息、共享存储区、信号量，并通过进程通信实例进行说明。
- 进程管理命令：进程管理命令主要有 ps、kill、nice、sleep、wait。

8.1　UNIX 进程及描述

8.1.1　UNIX 系统中的进程

进程是程序运行。UNIX 系统中的进程呈现出多级结构。所有进程都会生成其子进程，子进程再生成子子进程，依次类推。最上层的进程是系统的根进程，由系统生成，没有父进程。除根进程之外，所有进程都有父进程。

为了管理方便,UNIX 操作系统核心给每个进程分配一个惟一的进程标识符 pid。

每个进程都属于一个用户，进程的用户标识符影响进程的优先级和系统调用的使用，子进程会继承父进程的用户标识符。但在子进程高于父进程的使用权的情况下允许子进程有不同于父进程的用户标识符。

如果进程的拥有者和进程所对应的文件拥有者具有相同的用户标识符，则称该用户标识符为有效用户标识符。通过有效用户标识符可以获取文件。子进程刚创建时具有的用户标识符和有效用户标识符与其父进程的用户标识符相同。若子进程执行的文件通过 setuserid 和 setgroup id 命令修改了进程的拥有者或拥有者组时，该进程的有效用户标识符就会变为执行文件的用户标识符或组用户标识符，临时作

为与该文件用户同组的身份执行。

在操作系统中可以直接从终端读写的进程为前台进程，而正在运行却又无法直接与终端通信的进程为后台进程，任何时候一个终端只有一个前台进程而可以拥有多个后台进程。如果要使一个程序在后台运行只须运行时在程序名后加&，如：

```
$ ./httpd &↵
```

表示启动进程 httpd 在后台运行。

UNIX 系统引导过程是：系统产生进程 scheduled（也称为 sched），其 pid 为 0，由进程 sched 通过系统调用 fork 创建进程/etc/init，其 pid 为 1，在系统被加载到机器内存后，核心先执行/etc/init 进程，设置系统初值。在创建/etc/bcheckrc 和/etc/brc 进程后，再通过系统调用 fork 产生几个驻守在机器上的子进程（这些子进程称为驻守进程——daemon）和 getty 进程集（负责监视每个终端上的用户注册情况），准备接收用户登录系统。

驻守进程是长期在后台运行的进程，通常包括 sched、init、logger、update、cron、lpsched 等。如果是网络服务器，则还包含网络服务进程，如 httpd、sendmail、ftp 等。

Shell 本身是一个进程，无论何时，当用户注册登录后，系统为用户创建一个 Shell 进程（即用户的 login Shell），键入一个命令或执行一个程序时，Shell 会产生一个相应的子进程。在子进程中还可以创建子进程，待用户退出（exit）后又把控制权还给 Shell 进程。

8.1.2 进程状态及其转换

在 UNIX 系统中进程从产生到结束的生命周期中要经历许多状态变化，主要有如下几个状态。

(1) 创建：进程刚存在但还没有就绪，即进程的初始状态。进程 0 没有该状态。

(2) 内存就绪：进程已经具备执行条件，等待核心进程调度程序调度执行。

(3) 换出就绪：为了节约内存，将处于内存就绪状态的进程换出内存置于磁盘上，换出就绪进程等待核心程序将其换入内存，成为内存就绪状态后才能被调度执行。

(4) 核心态执行：执行的是操作系统核心程序，如果用户程序中出现系统调用则系统调用部分切换到核心态下执行。

(5) 用户态执行：执行的是用户程序。

(6) 内存睡眠：进程在内存中处于一种相对静止状态，将内存睡眠状态的进程唤醒后进入内存就绪状态。

(7) 换出睡眠：换出的进程处于睡眠状态，将其唤醒后进入换出就绪状态。

(8) 被强占：当用户进程完成系统调用从核心态返回用户态时，核心抢先调度另一个进程执行，而将从核心态返回的该用户进程置于被强占状态。处于被强占状态的进程与处于内存就绪进程很相似，等待再次被调度执行。

(9) 僵死：也称为进程结束状态。处于该状态的进程已不存在，但会在其父进程中留下该进程的结束码及计时统计信息等记录。当进程通过系统调用 exit 会进入僵死状态，同样进程接收到软中断信号而需异常结束时也会进入僵死状态。僵死状态是进程的最后一个状态。

进程的状态及其转换如图 8.1 所示。

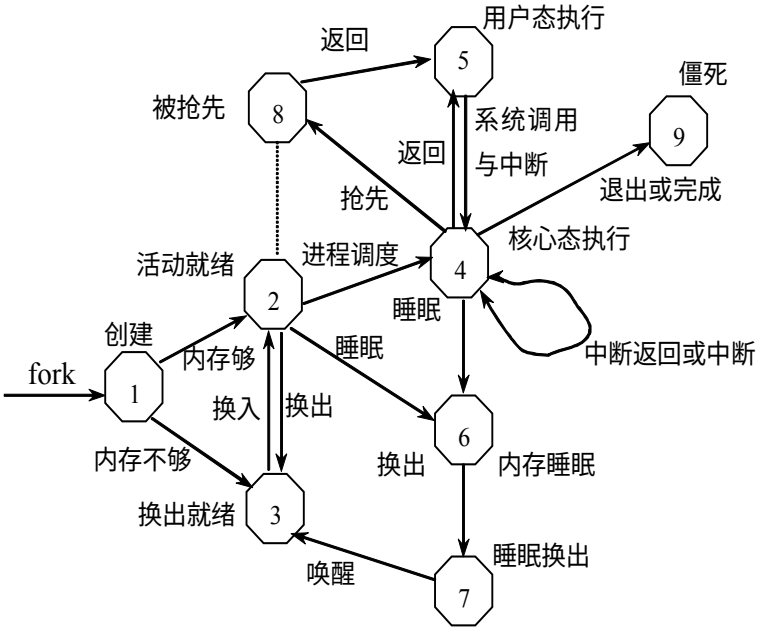

图 8.1　进程状态转换图

8.1.3　进程与区

进程的自包含性使得进程运行及运行环境相对独立。UNIX 系统中的进程由正文段、数据段和堆栈段组成。

正文段包含进程运行所需的程序代码，数据段包含进程运行所需的数据，堆栈段包含进程运行所需的系统栈信息。UNIX 系统把进程的逻辑地址空间化分为一个个独立的可共享和保护的逻辑区，并且每个区中进程的地址空间是连续的。执行相同程序的多个进程共享正文区，而数据区和堆栈区则为每个进程独有。为了管理这些区，在 UNIX 操作系统的内核中设置了区表，一个区对应于一个区表表项，表项描述了在系统中活动区的信息。区表表项的主要内容有：

(1) 指向进程数据区文件的索引节点指针；

(2) 区的类型；

(3) 区的大小；

(4) 区在物理内存中的地址；

(5) 区的状态：锁住、请求中、正在装入内存中、已被装入内存（有效）；

(6) 区被进程引用数。

操作系统核心可以对区进行的操作有如下几种。

(1) 锁区、解锁区：由于区是一个互斥访问的临界资源。进程要访问它时，如果没有其他进程访问就可以进去访问，访问时必须上锁，访问完后要解锁；如果有其他进程访问就必须等待。

(2) 分配区：当内核分配一个新区给进程时，将区从空闲存储空间附接到进程的存储空间，并将该区区表表项的第一个可用项从自由表中取出来，放到活动区表中，写上相应的区表表项内容。注意，当该区和其他进程共享时，在区表表项中引用该区的进程数上应该加 1，否则该区容易被其他进程删去。

(3) 释放一个区：进程完成后，应该释放非共享的区，使该区和进程的存储空间断接。

(4) 回收区：将已经释放的区放还到空闲存储空间，同时还将该区的区表表项放回到自由表项。

(5) 复制区的内容：当进程申请到区后，如果该区不是共享区，内核就将相应的内容复制到该区；如果该区是共享的，内核只须将该区的引用计数加 1 即可。

8.1.4 进程与进程表

在 UNIX 系统的内核中存在一个进程表，该表中的每个表项称为进程表项，它描述了系统中活跃进程的信息。进程表项中包含的字段有：

(1) 进程所属用户及用户组标识符；

(2) 进程标识符；

(3) 标识进程状态的字段；

(4) 进程的大小信息字段和进程状态转换时内核切换进程上下文所需数据信息字段；

(5) 描述进程睡眠的所有参数；

(6) 内核将进程在用户态执行与核心态执行转换次序的调度参数；

(7) 记录一个进程收到但未处理的软中断信号；

(8) 进程的所有计时信号。

进程表常驻内存，内核可以对进程表中的项内容进行读、写。

8.1.5　进程与 u 区

除进程表项之外，u 区也用于描述进程控制信息，是进程表的补充。与进程表不同的是，u 区中的信息是被内核操作的进程的私有信息（这正是 u 代表用户的意思）。由于内核只对正在运行的进程的 u 区进行存取，所以结构变量 u 中的信息总是表示正在执行进程的 u 区信息，内核通过 u 区信息得到进程表项的指针，相应地也就能识别当前进程。u 区中主要的信息有：

(1) 指向当前正在执行的进程表项的指针，从而得到与该 u 区相对应的进程表结构；

(2) 与用户标识符相关的决定进程各种特权的项，如进程对文件的存取权限；

(3) 与文件结构有关的项，如记录进程已打开的文件描述符，当前根及本进程的文件系统环境；

(4) 当前系统的调用参数、返回值及错误码；

(5) 记录进程、子进程在用户态、核心态运行时间的定时器；

(6) 输入/输出参数，如进程要读写的数据量、用户地址空间中的数据地址、读写文件的字节数、读写方式、缓冲区长度及地址等；

(7) 进程对中断、软中断信号进行处理的参数；

(8) 进程上下文切换时保护现场的各项数据，如通用寄存器中的用户区地址、寄存区的当前值、进程栈指针、CPU 状态字；

(9) 进程正文段、数据段、堆栈段长度。

进程只有在核心态时才能访问它的 u 区。内核进程调度时根据进程的物理地址得到与进程对应的 u 区，通过 u 区中的逻辑地址找到进程。内核得到进程 u 区的同时会修改 u 区的地址空间使之成为可存取。

8.1.6　进程映像

进程映像也称为进程上下文，用于描述进程及其运行环境。进程映像由用户地址空间内容、硬件寄存器内容和与该进程有关的内核数据结构组成。

用户地址空间内容主要有如下几种。

(1) 进程的正文区：是进程中可由多个进程共享的区域，包括运行程序的机器码及常量等。

(2) 进程的数据区：是处于用户态的进程可读、写、执行的区域，含有属于进程私有的程序和数据。

(3) 进程的堆栈区：由核心栈和用户栈组成。核心栈是进程在核心态运行时执

行子程序嵌套调用的下堆栈和发生中断事件后用于保留现场的中断保留区，可被处于核心态的进程存取。用户栈主要由进程在用户态运行时函数调用的参数、局部变量等数据组成。

(4) 区表和页表：是进程的正文区、数据区、堆栈区共同占有进程的逻辑地址空间，区表和页表是地址变换机制中的重要成分。地址变换机制将进程的逻辑地址转换为内存中的物理地址。

硬件寄存器内容包括：

(1) 程序计数器：该计数器中的内容是 CPU 将要执行的下条指令的虚地址；

(2) 通用寄存器：该寄存器存放进程执行期间所产生的中间结果及参数；

(3) 处理机状态寄存器：该寄存器中的内容包括当前执行的进程访问方式，是核心态还是用户态、中断优先级、中断之前的访问方式；

(4) 栈指针：该指针指向栈中下一项的当前地址。

与该进程有关的内核数据结构有：

(1) 进程的进程表表项（定义进程的状态）；

(2) 进程的 u 区（包含进程的控制信息）；

(3) 本进程区表表项和区表（决定了进程的正文区、数据区、栈区等）；

(4) 核心栈（内核过程的栈结构）；

(5) 进程的系统级上下文动态部分。

进程映像组成如图 8.2 所示。

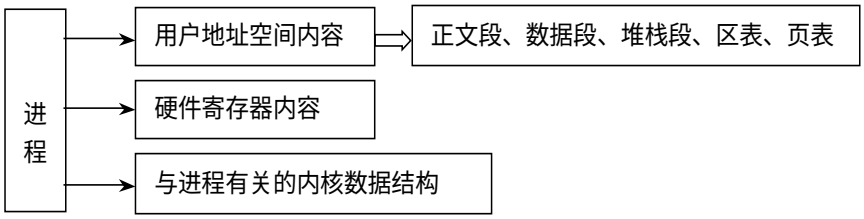

图 8.2 进程映像图

内核使用的是 u 区的逻辑地址，并且一次只能存取一个 u 区，所以在一定程度上 u 区决定了正在系统中运行的进程的上下文。进程上下文切换就是正在核心态运行的进程由于各种原因而转换到其他状态，核心调度程序再次将在内存中就绪的进程调度到处理器中成为核心运行状态所发生的进程上下文改变。进程发生上下文切换的几种情形可以从进程状态转换图中分析得出：

(1) 进程在核心态运行时使自己睡眠，转换为在内存中睡眠；

(2) 当进程在用户态运行时发生了系统调用或中断而进入核心态运行，核心态运行完后返回到用户态运行时；

(3) 当核心态运行的进程执行了系统调用 exit 而进入僵死状态时。

内核为了保证进程上下文切换时数据结构的完整性和一致性，会禁止任意的上下文切换。即使能够进行上下文切换，也要在切换前做好该做的工作、相应的加锁及解锁工作必须到位、各种队列链接必须完成等。内核调度程序从进程的内存就绪队列中选取哪一个进程到核心态运行是进程的调度策略，在下一节的进程调度部分会详细介绍。

8.2 进 程 控 制

8.2.1 进程的创建与终止

进程是操作系统中活跃着的“生命”，从创建到终止一代又一代，生生不息。

UNIX 系统中创建进程的方法是用系统调用 fork()。调用系统调用 fork()的进程是父进程，所创建的新进程为子进程。除进程 0 之外，系统中所有的进程都是系统调用 fork()创建的。

系统调用 fork()的实现过程是：首先由内核为新进程在进程表中分配一个表项并给新进程分配一个惟一的进程标识符 pid；然后通过拷贝父进程的进程映像来创建一个新进程，新进程将获得其父进程地址空间的一份拷贝；最后增加与该进程相关联的文件表和索引节点表的引用数，并将子进程的进程标志返回给父进程，收到父进程返回的信号则标志新进程创建完毕。

fork()系统调用生成子进程的详细步骤如图 8.3 所示。包含如下步骤：

(1) 如果系统内存够，则能创建进程，否则调用退出；

(2) 为要创建的进程分配一个空闲的进程表项和惟一的进程标识符 pid；

(3) 如果进程所属用户的进程数没超过系统限制，则能创建进程；否则调用退出；

(4) 设置进程状态为创建，拷贝父进程表项数据到进程表项中；

(5) 将当前目录的索引节点和改变的根目录的引用数加 1，文件表中打开文件的引用数加 1；

(6) 拷贝父进程上下文到进程的上下文；

(7) 如果正在执行的是其父进程则设置进程状态为就绪，否则初始化 u 区的计时域。

子进程创建后拷贝其父进程的上下文，而且可以使用父进程的当前目录和根目录以及已打开的文件，只是它们的引用计数要加 1。

当父进程创建子进程时希望子进程结束后能把控制返回给父进程，所以子进程

不能覆盖父进程给予的控制区。

当进程任务完成后，为了收回进程所占用的系统资源和减少父进程的负担，进程可以用系统调用 exit()来实现进程的自我终止。进程终止可以在正常情况下，也可以在非正常情况下。

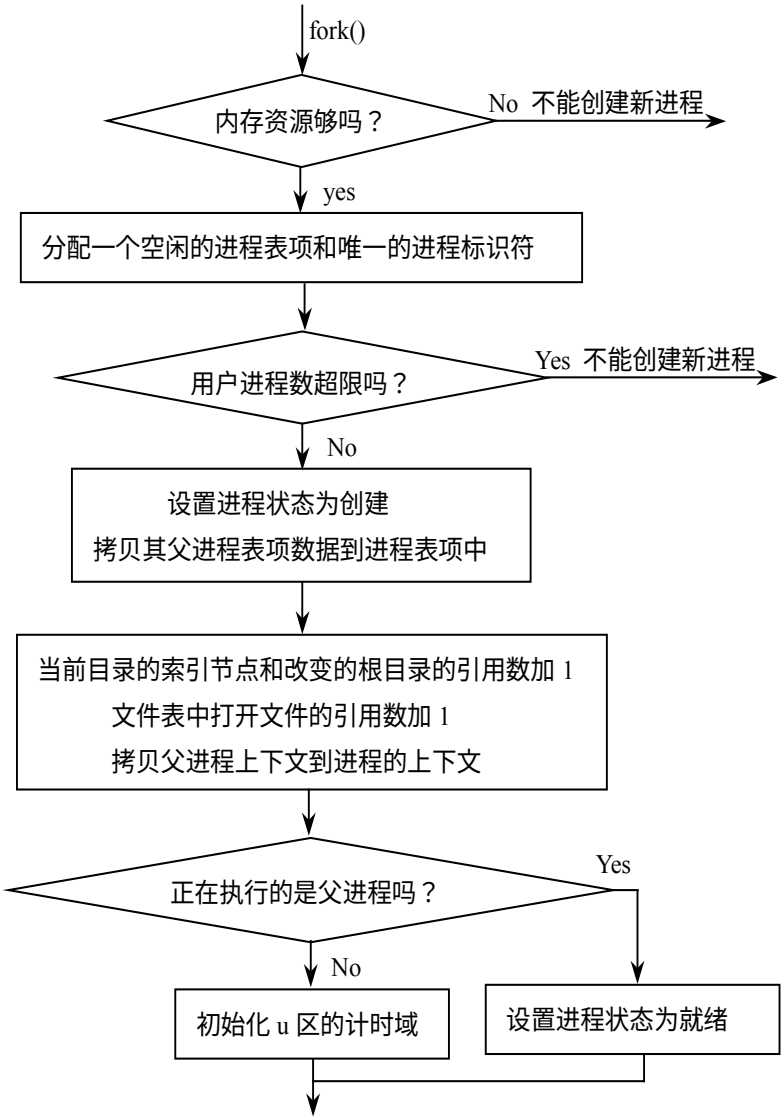

图 8.3 fork()系统调用生成子进程的步骤

系统调用 exit()的实现详细过程如图 8.4 所示。包含如下步骤：

(1) 关闭进程的软中断信号；

(2) 如果是与控制终端相连的进程组组长，则向本组进程发出挂起信号；

(3) 关闭所有打开的文件；

(4) 释放进程的上下文、当前目录和改变的根目录；

(5) 记录进程运行时的统计信息；

(6) 进程状态转换为进入僵死状态；

(7) 将自己尚未结束的子进程交给 init 进程，使这些子进程成为 init 的合法子进程；

(8) 保留进程表项，通知父进程并等待父进程。

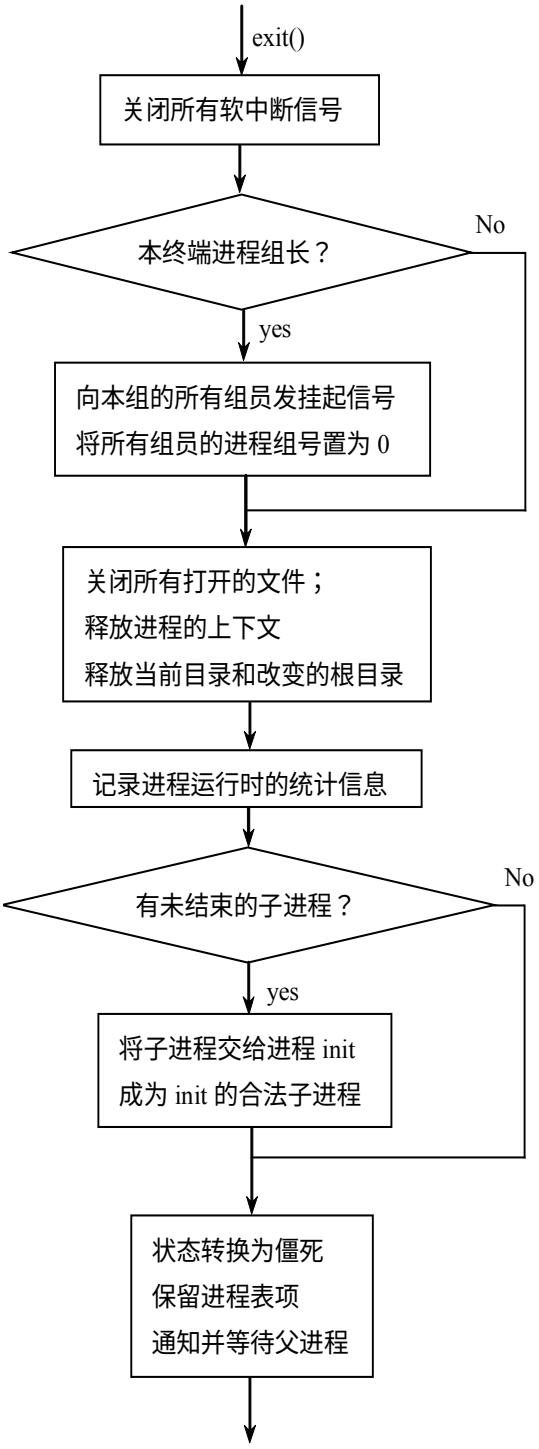

图 8.4　系统调用 exit

在子进程和父进程的终止关系上系统调用 wait()起着非常特殊的作用。父进程调用 wait()的目的是自己要结束。但是面对自己的子进程，父进程应该怎么做呢？父进程是这样处理自己的子进程的：

- 如果父进程调用wait()将自己挂起,而此时自己又无子进程,则调用wait()会返回一个出错码。
- 如果父进程调用 wait()将自己挂起，而此时其子进程已进入僵死状态，则调用 wait()的结果是收回子进程的进程表项，并将子进程的执行时间加到父进程的执行时间上。
- 如果父进程调用 wait()将自己挂起，而此时其子进程不能结束，则父进程等待子进程终止，接收子进程的返回状态和返回参数。

另外系统调用 exec()也与进程创建有一定的关系，在系统调用一章中会和系统调用 fork、wait、exit 在程序设计中的应用一起介绍。

8.2.2 进程调度

进程调度的任务是决定具备执行条件的进程，即处于内存就绪状态的进程，哪个可以获得 CPU 的执行权。

由于 UNIX 是分时操作系统，终端用户作业是直接提交到系统内存的，除批处理作业之外,在系统中不存在用户作业调度。所以 UNIX 系统中发生最多的是进程调度。

在进程调度中，系统内核给每个进程分配一个 CPU 处理的时间片，当时间片到后内核抢占该进程并调度另一个进程。在系统中每个进程都被分配一个优先级。UNIX 进程调度采用的策略是多级反馈循环调度法（Round Robin with Multiple Feedback）。该方法的思想是将处于内存就绪状态的进程按进程优先级排成就绪队列，即就绪队列 1，将就绪队列 1 中的进程按优先级调入 CPU 中运行。给进程分配一个时间片 S1，如果该进程在运行时间片 S1 到时还没有完成，则放弃在 CPU 中运行进入就绪队列 2；同样，对就绪队列 2 中的进程也按优先级调度，如果运行时间片 S2 到时没有完成，则进入就绪队列 3；依此类推，直到进程完成为止。

需要补充的是：

- 只有就绪队列 1 中的进程全部调度完后才能调度就绪队列 2 中的进程，同样，只有就绪队列 2 中的进程全部调度完后才能调度就绪队列 3 中的进程，依此类推，直至最后的就绪队列；
- 就绪队列 1 的优先权等级最高，时间片最短，就绪队列 2 的优先权等级次之，时间片较长，随着就绪队列往下，优先权逐个下降，时间片逐个增长。如图 8.5 所示。

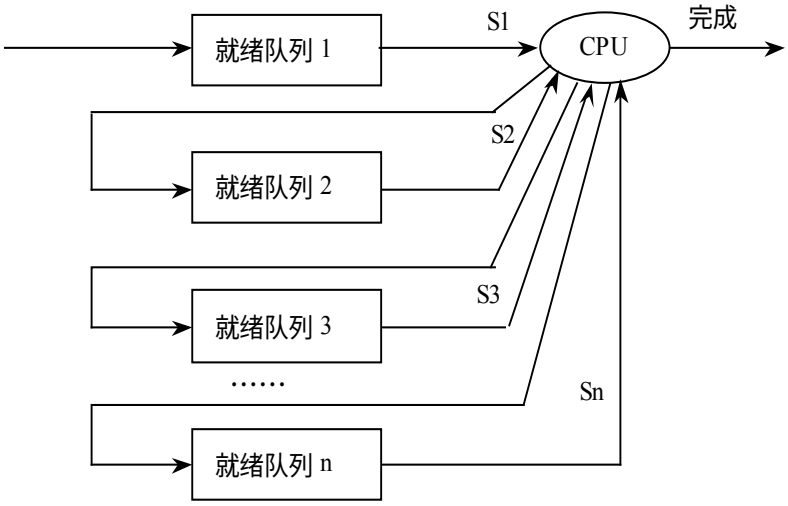

图 8.5　UNIX 系统进程调度算法

一般情况下，一个较长的用户进程的执行要经过多次反馈循环调度才能完成。而在UNIX系统设计上选取时间片时会尽量将绝大多数Shell命令执行放在一个时间片内完成。

在进程调度中要关注的问题有如下几种。

(1) 优先级：在 UNIX 系统中每个进程都有一个优先级，用于进程调度。进程优先级分为核心优先级和用户优先级两类。

在核心优先级中，当一个软中断信号到达时，若进程正处于在可中断优先级上睡眠，则进程立即被唤醒，这时进程的优先级为可中断优先级；否则，若进程处于不可中断优先级上时则不能将其唤醒，让其继续睡眠。系统中处于对换、等待磁盘 I/O、等待缓冲区、等待索引节点的进程是不可中断的；系统中等待终端输入、终端输出、子进程退出的进程是可中断的。

通常用户优先级分为 n+1 级，第 0 级为最高优先级，第 n 级为最低优先级。用户优先级是它最近使用 CPU 时间的函数。进程优先级的范围如图 8.6 所示。每个优先级都有一个逻辑上相关联的进程队列，即一个优先级分成若干个优先数，优先数越小，在同一队列上的优先等级越高。

(2) 策略与优先级的计算：进程优先级是动态变化的，在 UNIX 系统 V 中采用如下方法计算进程的优先数：

```
priority=recentusing_cpu/2+user+nzero+nice
```

其中，recentusing_cpu 为每个进程最近一次使用 CPU 的时间，user 和 nzero 是基本用户优先数的阀值，nice 是系统允许用户通过系统调用或 Shell 命令设置的进程优先数偏置值（即 nice 值），nice 可以是 0~40 之间的一个数，普通用户使用 nice 的结果是使进程降低优先级。不同系统计算进程优先数的公式不尽相同，但基本的

成分都包含进程在“最近的过去”使用 CPU 的时间除以一个常数与优先数基值和用户偏置值三部分之和。

由于新创建的进程从未使用过 CPU，具有较高优先级。随着进程的被调度执行，其 recentusing_cpu 值不断增加，因此在时钟中断处理时可能被其他 recentusing_cpu 值较小的进程所抢占，而随着其他进程的执行和本进程的 recentusing_cpu 衰减（除以常数），当其优先数小于当前进程和其他进程时，又会重新抢占 CPU。

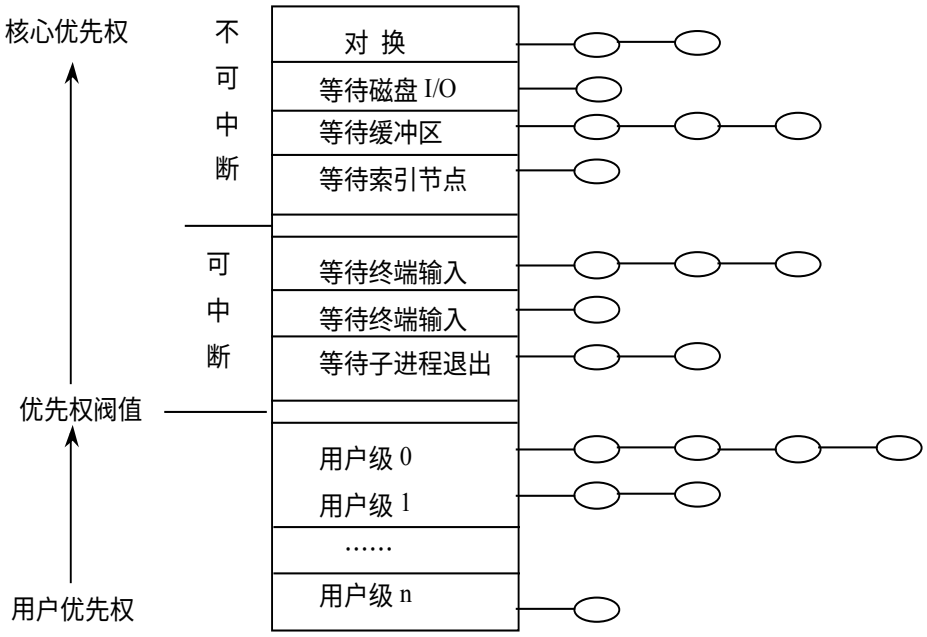

图 8.6 系统优先级

(3) 调度时机：在 UNIX 中有两种情形需要进程调度：一是在 CPU 中的进程为了等待 I/O 事件，或为了与其他进程保持同步以及进程异常结束等而放弃 CPU 时；二是在 CPU 中的进程由核心态转入用户态时高优先级的进程强占 CPU 时。

(4) 调度标志的设置：在 UNIX 中进程调度的标志有：runrun、runin 和 runout。runrun 是发现某进程的优先级高于当前进程的优先级时需要设置的标识，以要求进程调度程序进行调度；runin 是进程已换入内存的标志；runout 是进程已换出内存的标志。

(5) 调度的实现：进程调度的实质是完成进程之间的上下文切换的过程。该过程首先检查是否有进程上下文切换及系统是否允许进行上下文切换，若允许，则保留当前进程的进程上下文（特别是寄存器的内容及栈指针）；然后进程 0 根据调度算法从内存就绪队列或被抢占队列中选取一个优先级最高的进程进入 CPU，如果内存就绪队列或被抢占队列中无进程，则调度进程循环等待某进程变为内存就绪状态；

最后，被选中进入 CPU 的进程成为当前进程，当其进程映像恢复与该进程有关的寄存器内容和栈指针时，新进程就开始正常执行。

8.3　进程间通信

在 UNIX 系统中进程与进程之间需要相互交换数据，即进程通信。进程通信可以发生在同一主机的进程之间，也可以发生在主机与主机的进程之间。UNIX 进程通信方式有信号、管道、消息、共享存储区和信号量。

8.3.1　信号

1. 什么是信号

信号是对中断机制的一种模拟方式，故被称为软中断信号（Signal）。指当某进程发生了异常情况时给同一用户的其他进程按照事先约定的信息类型发出通知。所以每个进程在运行时都要通过信号机制检查是否有其他进程发送的信号到达。如果有信号到达则中断正在执行的程序并转向与信号相对应的处理程序，执行对异常事件的处理。在处理完成后再返回原中断点继续执行。

例　当系统运行某程序时，如果用户发现问题就用 Ctrl+C 停止程序运行，实际上就是 Shell 进程接收 Ctrl+C 并向进程发送了信号，进程接收到该信号后按事先约定方式作了相应处理。

在 UNIX 系统 V 中有 19 种软中断信号，UNIX BSD 系统中有 32 种信号，它们的前 15 种相同。表 8.1 中列出了信号值、信号名称及其意义。

表 8.1　信号

值	名称	意义
01	SIGHUP	进程挂起
02	SIGINT	当用户键入“Del”时中断
03	SIGQUIT	当用户键入“Quit”时退出
04	SIGILL	当用户键入了非法命令时
05	SIGTRAP	启动进程跟踪代码的执行
06	SIGIOT	IOT 命令
07	SIGEMT	EMT 命令
08	SIGFPE	浮点数例外
09	SIGKILL	杀死或终止进程
10	SIGBUS	总线错误
11	SIGSEGV	进程试图访问虚地址空间以外的位置，段非法

表 8.1 信号 （续表）

值	名称	意义
12	SIGSYS	系统调用中出现非法参数
13	SIGPIPE	向某个非读的进程管道中写入数据
14	SIGALARM	当某进程希望在某时间后接收信号时发出此信号作为闹钟
15	SIGTERM	软件终止信号
16	SIGUSR1	用户自定义信号 1
17	SIGUSR2	用户自定义信号 2
18	SIGCLD	某个子进程死亡
19	SIGPWR	电源掉电
20	SIGWINS	窗口大小被改变
21	SIGSOCK	Urgent Socket Condition
22	SIGPCON	Pollable event
23	SIGSTOP	停止
24	SIGTSTP	由用户停止（即交互式停止）
25	SIGCONT	继续执行（continued）
26	SIGTTSTI	停止终端输入
27	SIGTTSO	停止终端输出
28	SIGVTE	Virtual Timer expired
29	SIGPTE	Profiling time expired
30	SIGTIME	超过 CPU 的时间
31	SIGSIZE	超过文件大小的极限
32	SIGSOCKIO	Socket I/O possible

其中发出 SIGQUIT、SIGILL、SIGTRAP、SIGIOT、SIGEMT、SIGFPE、SIGBUS、SIGSEGV、SIGSYS 信号的进程会造成内存转储，系统处理信号时把进程的内存映像写入进程当前目录下的文件 core 中。这就是为什么系统发生问题时可以见到有一个 core 文件产生的原因。

在 UNIX 操作系统中信号类型值和信号类型名通常在 signal.h 文件中定义。

2．信号的发送

信号发送是发送进程用系统调用 kill 并通过核心向一个或一组进程发送信号。普通用户只能向自己的进程或同组进程发送信号，超级用户可以向系统中的所有进程发送信号。信号发送后如果目标进程正在一个可被中断的优先级上睡眠，则核心会立即将其唤醒。

在应用程序中要使用信号的系统调用，必须在程序中包含头文件 signal.h，系

统调用 kill 的语法格式为：

```
int kill(pid,sig);
int pid,sig;
```

其中，pid 是指定信号发往的进程或进程组的标识符，通常：

- pid 为正值，则直接指定接收信号的进程标识符；
- pid 为 0，则核心将信号发送给与发送进程同组的所有进程；
- pid 为-1,则核心将信号发送给所有用户标识符真正等于发送进程的有效用户标识符的进程；
- 如果调用者不允许将信号传送到由 pid 指定的进程,则 kill 函数调用失败，函数返回值为-1，并且设定错误信号 errno。

参数 sig 指定欲发送信号的类型，可以使用信号的号码，也可以使用符号名。

例 将软件终止信号(信号码为 15)发送给进程 pro1,查得 pro1 的 pid 为 129。

```
kill(129,15) 或
kill(129,SIGTERM)；
```

信号的符号名表可以使用 Shell 命令 kill －1 得到。

例 在系统 Solaris 中使用命令 kill －l。

```
$kill －l
HUP INT QUIT ILL TRAP ABRT EMT FPE
KILL BUS SEGV SYS PIPE ALRM TERM USR1
USR2 CLD PWR WINCH URG POLL STOP TSTP
CONT TTIN TTOU VTALRM PROF XCPU XFSZ WAITING
LWP FREEZE THAW RTMIN RTMIN+1 RTMIN+2 RTMIN+3 RTMAX-3
$
```

另外，在 Shell 中也可以用 kill 作为命令发送信号，命令格式为：

```
kill [-sig] pid
```

例 用命令 kill 将信号 SIGUSR2 送到进程 129。

```
$kill －17 129
```

3．信号的接收和处理

当多种类型的信号到达时，接收信号进程的信号域中有多个域被置位，但对同类信号，接收进程只能记住其中一个。

接收信号用信号系统调用 signal(sig,func)，该系统调用同时用于设置对信号的处理方式。其中：

sig 是进程接收信号的类型。

func 分为三种情形：

- func=1，sig 类信号被屏蔽，进程不接收该类信号。
- func=0，进程在接收到信号后终止自己，进程在自我终止前，核心在当前目录中建立 core 文件并将进程的内存映像写入其中，给程序员调试程序带来方便。
- func 为非 0、非 1 整数，func 的值是信号处理程序的指针。

如果系统调用 signal 失败，则返回值为-1。

接收信号进程信号检测发生在：

一个进程要进入或退出一个低优先级睡眠状态时，或一个进程即将从核心态返回用户态时。

如果进程正处于核心态，如正在执行系统库函数处理，即使收到信号也不理睬；只有当它返回到用户态时才处理信号。处理方式按信号设置方式进行。

通常情况下，进程检测到一个信号时的默认操作相当于执行了临时系统调用 exit 终止该进程，此时父进程要了解有关信号则通过操作返回码的低 8 位信息。详见 exit 系统调用。

4. 信号应用举例

在 UNIX 系统中信号应用非常广泛，如系统对异常事件的异常处理。下面通过一些例子说明信号应用。

例　该程序用 kill 调用在两个进程之间发送信号。程序建立了两个进程，通过向对方发送信号 SIGUSR1 实现这两个进程的同步。

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<signal.h>
#include<sys/types.h>
int ntimes=0;
main()
{
  int pid,ppid;
  int p_catch( ),c_catch;
  /*  设置父进程的 SIGUSR1    */
  signal(SIGUSR1,p_catch);
  /*  系统调用 fork()生成子进程      */
  switch (pid=fork())
```

```
    {
      case -1:
        printf("fail fork\n");
        exit(1);
      case 0:
        /*  设置子进程的信号 SIGUSR1   */
        signal(SIGUSR1,c_catch)
        /*   得到父进程标示符   */
        ppid=getppid();
        for(;;)
        {
          sleep(1);
          /*  在子进程中发送信号 SIGUSR1 给父进程   */
          kill(ppid,SIGUSR1);
          pause();
        }
        break;
        default;
        for(;;)
        {
          pause;
          sleep(1);
          /*  在父进程中发送信号 SIGUSR1 给子进程   */
          kill(pid,SIGUSR1);
        }
    }
}
void p_catch()
{
  printf("parent process caught the signal %d\n" , ++ntimes);
}
void c_catch()
{
  printf("child process caught the signal %d\n" , ++ntimes);
```

```
}
```

这两个进程都处于死循环中，在接收到对方发送来的信号之前，都在暂停等待中（pause），直到对方发来的一个信号到达。

信号被捕捉处理输出提示信息，并用 kill 发送一个信号给对方，按下中断键来终止程序。

程序运行结果：

```
parent process caught the signal 1
child process caught the signal 1
parent process caught the signal 2
child process caught the signal 2
parent process caught the signal 3
child process caught the signal 3
......
<Ctrl（中断程序）>
```

例　下面是接收键盘输入 delete 信号而中断程序执行的处理。

已知在 signal.h 中定义 SIG_IGN 为不理睬接收信号。

已知在 signal.h 中定义 SIG_DEF 为恢复对接收信号的默认处理。

```
/***** This program is signal int, the program's name is sigexer1
********************/
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<signal.h>
void signalint(signum)
int signum;
{
  signal(SIGINT,SIG_IGN);
  printf("\nhi! The program have received signal and signum=%d;\n" ,signum);
}
main()
{
  int i;
  for (i=1; i<6; i++)
  {
```

```
        signal(SIGINT,signalint);
        printf("i=%d\n",i);
        sleep(1);
      }
      printf("The Program End.\n");
      exit(0);
    }
```

该程序正常运行时：

```
$sigexer1
i=1
i=2
i=3
i=4
i=5
The Program End.
```

但是如果在程序运行到屏幕上打印出 i=4 时从键盘上送入信号 SIGINT（Ctrl+C），则程序接收到该信号，作相应的信号处理，处理完后返回到原主程序。

```
$sigexer1
i=1
i=2
i=3
i=4
delete        ←用户第一次键入
hi! The program have received signal and signum=2;        ←转作信号处理
i=5
$
```

例　下面程序生成 6 个子进程，子进程生成后挂起等待。父进程发送信号 SIGINT 给子进程，子进程收到该信号后自我终止。

```
/***** This program is signal int, the program's name is sigexer2
*******************/
#include<stdio.h>
#include<unistd.h>
#include<signal.h>
main()
```

```
{
  int i;
  for (i=1; i<6; i++)
  {
    if (fork()==0 )          /* 产生子进程 */
    {
      printf("i=%d pid=%d\n",i,getpid()); /* 得到子进程的 pid*/
     pause();           /*挂起进程的系统调用*/
    }
  }
  printf("sleeping...\n");
  sleep(3);
  printf("exiting...\n");
  if (kill(0,SIGINT)==-1 )
  {
    printf("Could not send signal.\n");
  }
  exit(0);
}
$sigexer2
i=1 pid=14587
i=2 pid=14588
i=3 pid=14589
i=4 pid=14590
sleeping...
i=5 pid=14591
exiting...
```

8.3.2 管道

1. 管道

管道（pipes）是 UNIX 操作系统的特征。管道允许进程之间按先进先出的方式传送数据。管道的实现是采用一个连接写进程和读进程的文件系统中的共享文件作为数据存储，该管道文件又称为 pipe 文件。写进程将数据写入管道文件，读进程从

管道文件中读出数据。管道是一种操作系统的临界资源，所以对管道的应用要考虑进程同步。

在 UNIX 系统中存在两种类型的管道：

- 无名管道（Unnamed Pipes）：无名管道是一个用系统调用 pipe()建立的无名临时文件（当然也无路径名），只能用 pipe()返回的文件描述符来标识该文件，因此只有调用 pipe()的进程及其子孙进程才能识别此文件描述符，才能利用该管道读写数据。当进程不需要使用无名管道时由核心收回其索引节点。
- 有名管道（Named Pipes）：有名管道是利用系统调用 mknod 建立的可以长期存在的有名文件。由于具有路径名，其他进程自然知道其存在，也可以利用该路径来访问该文件。对有名管道的访问与对文件的访问完全相同。用系统调用 open()打开、系统调用 read()读、系统调用 write()写。如果一个进程不使用有名管道时用系统调用 unlink()解除连接。

2. 无名管道

建立管道的系统调用格式为：

```
int pipe(filedes);
int fileds[2];
```

其中 filedes 是文件描述符，由 filedes[0]和 filedes[1]组成。filedes[1]是写入管道的描述符，filedes[0]是从管道读的描述符。

核心创建管道需要完成以下工作。

- 分配磁盘和内存索引节点：核心查看系统中有空闲索引节点便分配一个空闲磁盘索引节点、内存索引节点；否则创建管道的系统调用失败返回。
- 为读进程分配文件表项：如果有空闲文件表则核心为读进程分配一个表项并初始化；否则系统调用失败，释放前面建立的磁盘索引节点后返回。
- 为写进程分配文件表项：如果有空闲文件表则核心为写进程分配一个表项并初始化；否则系统调用失败，释放前面建立的磁盘索引节点及读进程的文件表项后返回。
- 分配用户文件描述符：在读进程、写进程的文件描述符表中，分别分配一个表项,并把两个表项在其对应文件描述符表中的偏移量 filedes(0)和 filedes(1)分别返回给读进程和写进程。图 8.7 说明了用管道通信时的数据结构。

可见，程序必须通过文件描述符 filedes[0]和 filedes[1]来访问管道，并且只有创建管道的进程或其子进程才能使用。为了提高进程对管道的访问效率，只使用文件索引节点中的直接地址项 i_addr(0)~i_addr(9)。如果每个盘块为 4 KB 大小，则管道文件的最大长度被限制在 10 KB。操作系统核心将索引节点中的直接地址项

作为一个循环队列管理，设置一个写指针和一个读指针，对管道的访问按先进先出顺序进行。

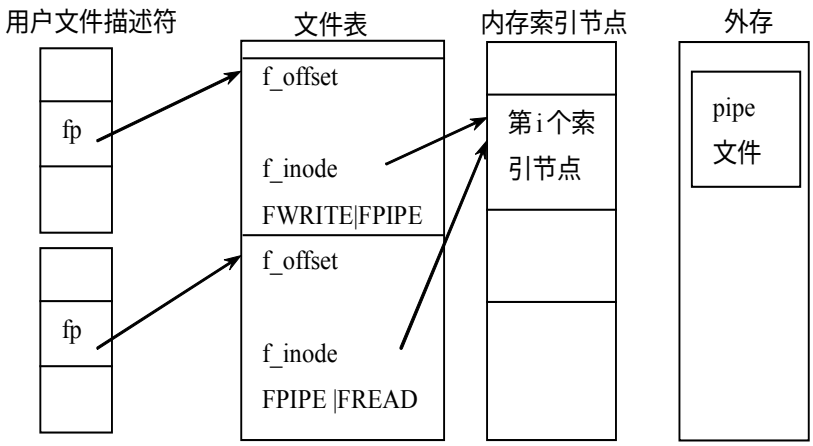

图 8.7 与 pipe 系统相关的数据结构

管道的实质就是文件，写进程和读进程对文件索引节点中的直接地址项访问需要互斥。所以进程要访问管道前先检查是否上锁，没有上锁才能进入并立即加锁访问，访问完后解锁；如果访问前检查到已经上锁则进程必须睡眠，等待操作进程结束解锁后将其唤醒。

(1) 进程写管道

进程向管道写数据时须注意管道大小限制，管道中已有的数据与要写入数据之和不能大于管道的容量。如果管道中无空间存放要写入的数据时，写进程要睡眠等待读进程将数据读出后将其唤醒后才能再写。

写管道的系统调用格式为：

```
write(filedes[1],buf,size);
int filedes[1],size;
char buf[];
```

其中 filedes[1]为写入管道的描述符；buf 是要写入管道的数据；size 是写入的数据位组长度。

(2) 进程读管道

进程从管道读数据时如果管道中已经有数据，进程则按先进先出的原则通过读指针读取数据，读结束后核心修改索引节点读指针并唤醒写进程；如果进程要读的数据不在管道中，进程则睡眠等待写进程写入数据并将其唤醒。

读管道的系统调用格式为：


```
read(filedes[0],buf,size);
```

```
int filedes[1],size;
char buf[];
```

其中 filedes[0]为从管道读的描述符；从管道读出的数据放入 buf 中；size 是读出的数据位组长度。

例　下面的程序是同一进程中首先创建一个无名管道，然后向该管道写入字符，再将写入的字符读出。

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
char string []="welcome to unnamed pipe";
int main()
{
  int filedes[2];
  char inbuf [256];
  if (pipe(filedes)<0)        /* 创建可擦写的无名管道 */
  {
    perror ("could not create unnamed pipe");
    exit(1);
  }
  write(filedes[1],string,23);   /*写入 filedes[1]的数据是从 filedes[0]读出的*/
  memset(inbuf, 0, sizeof(inbuf));
  read(filedes[0],inbuf,23);
  printf("%s\n",inbuf);
  exit(0);
}
```

程序执行的结果是打印出：

```
welcome to unnamed pipe
```

例　父子进程通过管道进行通信。

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
char string []="welcome to named pipe";
```

```
main()
{
  int filedes[2],i,pid;
  char inbuf[256];
  if (pipe(filedes)<0)        /* 创建可擦写的无名管道 */
  {
    perror("could not create unnamed pipe");
    exit(1);
  }
  if ((pid = fork())<0)        /* 创建子进程*/
  {
    perror("could not create subprocess");
    exit(2);
  }
  if (pid>0)        /* 父进程*/
  {
    write(filedes[1],string,23);
    printf("the parent process sended\n");
    wait(0);
  }
  if (pid==0)        /* 子进程*/
  {
    memset(inbuf, 0, sizeof(inbuf));
    read(filedes[0],inbuf,23);
    printf("the subparent process received\n");
    printf("%s\n",inbuf);
  }
  exit(0);
}
```

程序执行的结果是

```
the parent process sended
the subparent process received
welcome to named pipe
```

3．有名管道

利用无名管道无法实现不同用户进程之间的数据通信，而 UNIX 的有名管道是利用系统调用 mknod 建立的可以长期存在的文件，对该管道文件的访问只受文件权限限制而与进程关系无关。有名管道通过系统调用 read 和 write 对该管道文件进行访问，但访问之前必须用系统调用 open()打开文件，访问完后用系统调用 close()关闭文件。

系统调用 mknod 用来在 UNIX 文件系统中建立特殊文件，包括有名管道、设备文件和目录。该系统调用的格式为

mknod(pathname,type and permissions,dev);

其中：pathname 是要建立的特殊文件名（有名管道名、目录名）；type and permission 给出特殊文件类型和访问权限；dev 为块特殊文件和字符特殊文件规定的主设备号和次设备号（此处是有名管道，dev 参数为 0）。此处用该系统调用创建一个有名管道，内核为管道文件分配一个索引节点并在索引节点中设置该文件类型是有名管道。

(1) 有名管道的打开

有名管道文件有目录项并且可以通过路径名存取。进程打开有名管道与打开正规文件方式一样，为了表示因读写而打开有名管道的进程数目，内核用了文件索引节点的读计数和写计数。

对有名管道的打开和访问必须采用进程同步。如果没有希望接收的数据，则为读而打开管道的进程打开管道无意义，所以通常情况下首先判断有无希望接收的数据，若无则会睡眠等待而不用打开有名管道，只有在有接收数据时才打开管道；为写而打开管道的进程在打开管道前也要先看管道中的数据是否已被读进程读完，若是读完才能进入管道写数据，否则睡眠等待而不用进入管道。

打开管道的系统调用格式为

int open(path,flags);

该调用返回文件描述符（整数）。其中 path 为文件名；flags 为打开类型，按 POSIX 标准包括：O_RDONLY（只读）、O_WRONLY（只写）、O_RDWR（读写）、O_APPEND、O_CREAT、O_EXCL、O_NOCTTY、O_NONBLOCK、O_TRUNC。

(2) 有名管道的读写

对有名管道读写权限的要求与无名管道不同，不需要进程之间存在相关关系，只要求进程对有名管道具有存取权即可。同时读管道进程的数目不要求与写管道进程数目相同。

在系统调用 write 期间内核会将数据放在管道设备上，为了加快访问效率按需要只将索引节点的直接地址块作为磁盘块分配给管道（这一点与普通文件不同），

核心则将索引节点的直接地址块作为一个循环队列管理，先进先出。借助于索引节点指针标记进程读写管道，是为了让所有进程都能找到该标记，这与正规文件采用的文件表不同，前面介绍过无名管道就是用文件表，因为父子进程可以共享文件表。

下面分析进程读写管道数据。

当一个进程正在写管道，要写的字节数与管道中已有的字节数之和小于或等于管道容量时，由于管道是按先进先出原则进行，并且管道中的数据是以缓冲队列方式管理，所以内核在进程每次写之后自动增加管道大小，管道数据量随着一次一次写而增加，内核不会让进程写操作覆盖管道中的数据。这与正规文件在文件尾写入数据情形相同。当写进程将全部数据写入管道时，内核会记下管道的索引节点写指针，以使后面的写进程接着写，同时唤醒所有等待读数据的进程。

当一个进程正在写管道，要写的字节数与管道中已有的字节数之和大于管道容量时，内核会在管道写满时对索引节点作相应标记，写进程进入睡眠等待，直到有读进程将数据读出后再唤醒该写进程继续写。当写进程不止一个时可能存在一个进程写入管道中的数据不连续，其中会插入另一个进程写入的数据。

当一个进程读管道，核心检查到管道中有数据时，读进程就从索引节点读指针标记处开始读，每读完一块后核心会减少相应数量的管道大小，读完后核心会将当前偏移量保存在索引节点中并唤醒等待写进程。

当一个进程要读的数据量大于管道内的数据时，读进程会在读完管道中的所有数据后进入睡眠，等待写进程写入后再次读。

有名管道读写系统调用与无名管道形式相同。

(3) 管道的关闭

关闭管道时内核首先处理读写进程：

如果无写进程、有读进程，内核会唤醒读进程并让其读系统调用空数据返回。

如果有写进程、无读进程，内核会唤醒写进程并发送一个发生错误的信号（向某个非读的进程管道中写入数据）给写进程。

除了处理读写进程外，内核还必须释放所有数据块并修改索引节点指明管道为空。

例　下面的程序是同一进程中首先创建一个有名管道,然后向该管道写入字符，再将写入的字符读出。

```
#include <sys/stat.h>
#include <fcntl.h>
char filename[] = "./myfifo";

char string[]="welcome to named pipe";
main(argc,argv)
```

```
        int argc;
        char *argv [];
        {
          int filedes[2];
          char buf[256];
          /* 创建可擦写的有名管道 myfifo                              */
          if (mknod(filename, O_CREAT | S_IRUSR | S_IWUSR, 0) == -1)
          {
            perror("mknod error");
            exit(1);
          }
          filedes[0]=open(filename, O_RDONLY);     /*以只读方式打开管道 myfifo */
          filedes[1]=open(filename, O_WRONLY);     /*以读写方式打开管道 myfifo */
          write(filedes[1],string,21);   /*写入 filedes[1]的数据是从 filedes[0]读出
的*/
          printf("wrote message: |%s|\n", string);
          sleep(3);
          memset(buf, 0 ,sizeof(buf));
          read(filedes[0],buf,21);
          printf("read message: |%s|\n", buf);
          unlink(filename);
          exit(0);
        }
```

执行结果：

```
wrote message: |welcome to named pipe|
read message: |welcome to named pipe|
```

例　FIFO 用特殊文件表示，也称为命名管道，但与 pipe 管道不同。由于 pipe 管道用临时文件，当没有进程使管道打开时 pipe 管道就会消失，而 FIFO 即使所有进程都将其关闭时，FIFO 仍继续存在。具有适当权限的任一进程都可以存取 FIFO。FIFO 可以在应用程序中用系统调用创建。该系统调用格式为：

```
int mkfifo(path,mode);
const char * path;
mode_t mode;
```


其中：path 为所创建管道的特殊文件路径名；mode 为存取管道的模式。

例 用命名管道建立一个客户/服务器关系。客户进程和服务器进程将运行在同一个平台上。服务器进程在后台运行，客户进程在前台运行。客户进程接收用户的 Shell 命令，这个命令通过公共命名管道传送给服务器。一旦接收到这个命令，服务器就执行命令，然后服务器进程将命令执行的内容通过一个私有命名管道返回给客户进程，并由客户进程将内容显示在屏幕上。

这两个进程将进行的步骤为：

- 服务器生成公共命名管道（对所有客户可见）；
- 客户生成自己的私有管道；
- 客户进程提示输入信息，接收命名输入；
- 客户将私有管道名和用户输入命令写到公共管道；
- 服务器读取公共管道，得到私有管道名和命令；
- 服务器使用 popen-pclose 序列来执行用户输入的命令，其输入经过私有管道送回客户进程，并在后面加上服务器当前的时间；
- 客户显示命令的输出、服务器时间和本地时间。

```
/*客户-服务器管道的头文件  */
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <limits.h>
#include <time.h>

#define PUBLIC "pipetest/PUBLIC"
#define B_SIZE(PIPE_BUF/2)
//  传给公共 FIFO 消息的格式由 struct 语句说明
//  第一个成员 fifo_name 包含了私有管道的名字
//  第二个成员 cmd_line 包含了将要被服务器执行的命令
struct message
{
        see more please visit: https://homeofbook.com
  char fifo_name[B_SIZE];
  char cmd_line[B_SIZE];
```

```
}
// 服务器端进程
#include <mypub.h>
main(void)
{
  int n,done,dummyfifo,publicfifo,privatefifo;
  struct message msg;
  FILE *fin;
  static char buffer[PIPE_BUF];
  time_t ticks;
  char buff[30];
  // 创建公共命名管道
  mknod(PUBLIC,S_IFIFO|0666,0);
  // 打开公共命名管道准备读写，设置非阻塞方式
  if((publicfifo=open(PUBLIC,O_RDONLY))
  = =-1||(dummyfifo=open(PUBLIC,O_WRONLY|O_NDELAY)) = = -1)
  {
    perror(PUBLIC);
    exit(1);
  }
  //消息可由公共管道读出
  while(read(publicfifo,(char*)&msg,sizeof(msg))>0)
  {
    n=0
    done=0;
    // 以写方式打开私有管道，O_NDELAY 防止 OPEN 产生死锁现象
    do
    {
      // 打开失败
      if ((privatefifo=open(msg.fifo_name,O_WRONLY|O_NDELAY))= = -1)
        sleep(3);
      else
      {
        // 打开成功
```

```
                fin=popen(msg.cmd_line, "r");   // 执行消息结构中传来的命令
                while((n=read(fileno(fin),buffer,PIPE_BUF))>0)
                {
                  write(privatefifo,buffer,n);  // 写入私有管道
                  memset(buffer,0x0,PIPE_BUF);  // 清空确保没有多余字符
                }
                pclose(fin);
                done=1;      // 纪录成功
                ticks=time(NULL);
                // 传递服务器时间
               write(privatefifo, "\nthe server time is:" ,20);
               sprintf(buffer, "%.24s\r\n" ,ctime(&ticks));
               write(privatefifo,buff,strlen(buff));
                close(privatefifo);
              }
            }
            while ((++n<5)&&(!done));
            if (!done)
              write(fileno(stderr),     "\nNOTE:SERVER**NEVER**accessed     private
FIFO\n" ,48);
            }
          }
          // 客户端进程
          #include<mypub.h>
          void main(void)
          {
            int n,privatefifo,publicfifo;
            static char buffer[PIPE_BUF];
            struct message msg;          // 消息结构变量
            time_t ticks;
            char buff_local[30];
            // 生成私有命名管道的名字
            sprintf(msg.fifo_name, "pipetest/fifo%d" ,getpid());
            printf( "the private pipe name is %s\n" ,msg.fifo_name);
```

```
// 创建对所有进程都有读写权限的私有管道
if (mknod(msg.fifo_name,S_IFIFO|0666,0)<0)
{
  perror(msg,fifo_name);
  exit(1);
}
// 打开公共管道准备写入
if((publicfifo=open(PUBLIC,O_WRONLY)) = = -1)
{
  // 如果公共管道未被服务器端创建，OPEN 函数失败
  perror(PUBLIC);
  exit(2);
}
while(1)
{
  // 用户输入命令提示
  write(fileno(stdout), "\ncommand>" ,9);
  // 将存储命令结构成员置为空，确保无多余字符
  memset(msg.cmd_line,0x0,B_SIZE);
  // 得到用户输入并将其保存在 cmd_line 中
  n=read(fileno(stdin),msg.cmd_line,B_SIZE);
  ticks=time(NULL);
  if (!strncmp( "quit" ,msg.cmd_line,n-1))    // 检查用户是否想退出
    break;
  write(publicfifo,(char*)&msg,sizeof(msg));  // 写入公共管道
  // 打开私有管道，并读取所返回的命令输出
  if ((privatefifo=open(msg.fifo_name,O_RDONLY)) = = -1)
  {
    perror(msg.fifo_name);
    exit(3);
  }
  while((n=read(privatefifo,buffer,PIPE_BUF))>0
  {
    write(fileno(stderr),buffer,n);
```

```
        }
        // 打印本地时间
        printf( "the client time is" );
        sprintf(buff_local, "%.24s\r\n" ,ctime(&ticks));
        printf( "%s" ,buff_local);
        close(privatefifo);
      }
      closed(privatefifo);
      unlink(msg.fifo_name);   // 删除私有管道
    }
```

8.3.3 消息（message）

消息是 UNIX 最早的进程通信方式。消息是格式化可变长的信息单元。消息允许一个进程将格式化的数据流发送到其他进程。接收进程可以将接收到的多个消息排成消息队列。

1. 消息机制的数据结构

由于进程要发送的消息中既有通信的数据，又有通信的数据管理信息及通信进程之间的信息，因此消息的长度不固定。通常将消息分为消息首部和消息数据区两部分。在 UNIX 消息机制中为了管理方便还使用了消息队列头表。消息首部和消息队列头表是消息机制中两种重要的数据结构。

消息首部：记录有消息类型、消息大小、指向消息数据区的指针、消息队列的链接指针等信息。

消息队列头表：该表中的每一个表项对应消息队列的消息头，该消息头中记录了该消息队列中第一个消息的指针、最后一个消息的指针、消息队列中消息的数目、队列中消息的总字节数、队列允许的消息数据的最大字节总数、最近一次执行发送操作的时间和进程标识符、最近一次执行接收操作的时间和进程标识符等。消息的数据结构如图 8.8 所示。

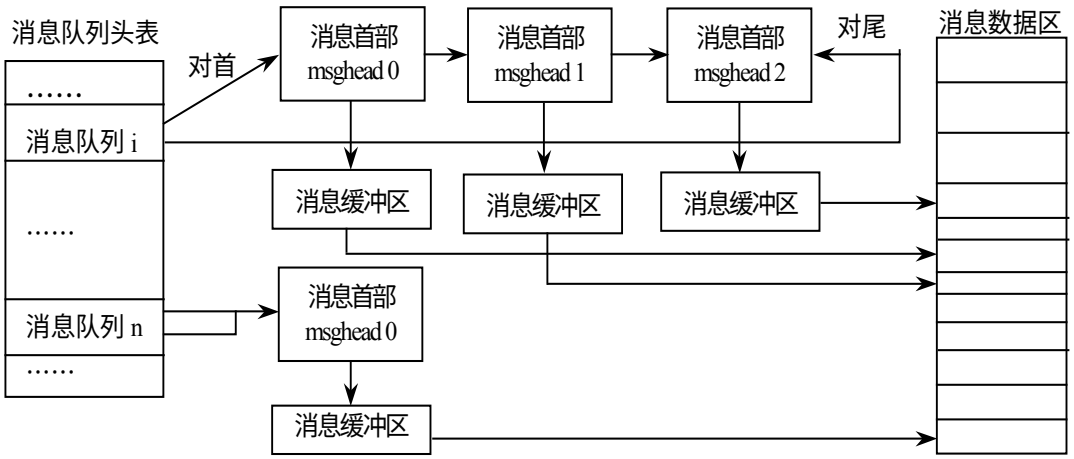

图 8.8　消息中的数据结构

2．消息队列的建立和消息描述符的获取

在 UNIX 中每个消息队列除了用户指定的关键词（名字）之外，为方便用户访问还设置了一个消息队列描述符。

进程利用 msgget 系统调用创建消息队列并返回一个指定消息队列的消息描述符。其他进程则用该消息队列描述符访问消息队列。

msgget 系统调用的格式为：

```
int msgget(key,msgflg);
key_t key;
int msgflg;
```

该系统调用的返回值是消息描述符，它是搜索消息队列头表表项的索引。

其中，key 是用户指定的消息队列关键词；msgflg 是用户设置的标志和访问方式。

完成该系统调用的步骤为：核心搜索消息队列头表以确定是否存在指定名字的消息队列；若不存在则分配一个新的消息队列头并初始化它，随后返回给用户一个消息队列描述符，若存在则只检查消息队列的权限便返回。

3．消息的发送和接收

用消息通信的进程可以用 msgsnd 系统调用发送消息，用 msgrcv 系统调用接收消息。

msgsnd 系统调用的格式为：

```
int msgsnd(msgid,msgp,msgsz,msgflg);
int msgid,msgsz,msgflg;
```

```
struct msgbuf * msgp;
```

其中，msgid 是由 msgget 返回的消息队列描述符；msgp 是指向用户消息缓冲区的指针。在消息缓冲区 msgbuf 中有消息类型和消息正文：

```
{
  long mtype; /*消息类型*／
  char mtext[ ];   /*消息的正文*/
}
```

msgsz 是 mtext 长度的字节数；msgflg 规定了无内存空间存储消息时，进程是等待还是立即返回。

msgsnd 系统调用的工作步骤如下。

(1) 核心检查消息队列描述符和权限及消息长度等是否合法，合法则继续进行发送过程，否则不能发送并返回。

(2) 核心为消息队列分配数据区并将用户消息缓冲区中的消息正文拷贝到消息数据区。

(3) 分配消息首部并链接到消息队列的末尾，并在消息首部中填写消息类型、消息大小、指向消息数据区的指针，修改消息队列头中的数据，唤醒等待消息的进程。

msgrcv 系统调用的格式为：

```
int msgrcv(msgid,msgp,msgsz,msgtpy,msgflg);
int msgid,msgsz,msgflg;
long msgtpy;
struct msgbuf * msgp;
```

其中，msgid、msgp、msgsz、msgflg 与 msgsnd 中的对应参数相同，msgtpy 是选读的消息类型。

msgrcv 系统调用的工作步骤如下。

(1) 核心检查消息队列描述符和权限是否合法，合法则继续进行接收过程，否则不能接收并返回。

(2) 分别就 msgtpy 三种不同情况进行处理：

- msgtpy=0，核心寻找消息队列中的第一个消息并返回给调用者；
- msgtpy 为正整数，核心返回给定消息类型的第一个消息；
- msgtpy 为负整数，核心在所有类型值小于或等于 msgtpy 绝对值的消息中选出类型值最低的一个消息返回。

(3) 当所返回消息小于或等于用户的请求时，核心将消息正文拷贝到用户区并将此消息从消息队列中删除，唤醒睡眠等待的发送进程；当所返回消息大于用户的

请求时，则出错返回。

(4) 如果用户不规定消息的大小，则核心不论消息大小统一拷贝到用户区。

4．消息队列的操纵

用户建立了消息队列后，用 msgctl 系统调用读取消息队列的状态信息，同时还可以对消息队列的一些信息进行修改。

msgctl 系统调用的格式为：

```
int msgctl(msgid,cmd,buf);
int msgid,cmd;
struct msgid_ds *buf;
```

其中，cmd 是规定的命令，有三种类型：

- IPC_STAT：用于查询消息队列中的消息数目、队列中的最大字节数、最后一个发送消息的进程标识符、发送时间等信息的命令；
- IPC_SET：用于设置和改变消息队列的用户标识符、消息队列的权限等属性的命令；
- IPC_RMID：消除消息队列的标识符命令。

例　在同一进程中用消息队列通信。

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MYMSGKEY 98
struct msgform    /*消息缓冲区内容*/
{
  long mtype;     /*消息类型*/
  char mtext [50];  /*消息正文*/
};
int main ()
{
  struct msgform msg, msg2;
  int msgid,pid,*pint;

  msgid=msgget(MYMSGKEY,IPC_CREAT|0600);    /*创建消息队列，得到消息描述符
msgid*/
        see more please visit: https://homeofbook.com
  pid=getpid();
```

```
    pint=(int*)msg.mtext;
    *pint=pid;   /*把进程标识符拷贝到消息正文*/
    msg.mtype=666;
    msgsnd(msgid,&msg,sizeof(int),0);       /*发送消息*/
    printf("the message sended %d\n",*pint);
    msgrcv(msgid,&msg2,sizeof(msg2.mtext),666,0);         /*接收消息*/
    printf("the message:received %d\n",*(int*)msg2.mtext);
    msgctl(msgid, IPC_RMID, 0);
    return 0;
}
```

程序执行结果为：

```
the message sended 15582
the message:received 15582
```

8.3.4 共享存储区

共享存储区是进程之间通信的又一种方式。

1. 共享存储区

共享存储区是多个进程共享它们的虚拟地址空间，通过对存储在该地址空间中的数据直接进行读写实现进程之间互通数据的目的。共享存储区是 UNIX 进程通信机制中通信速度最快的通信方式。

系统内存空间中的共享存储区连接着多个进程的虚拟地址空间，一个进程的虚拟地址空间中的部分又可以连接多个共享存储区。如图 8.9 所示。

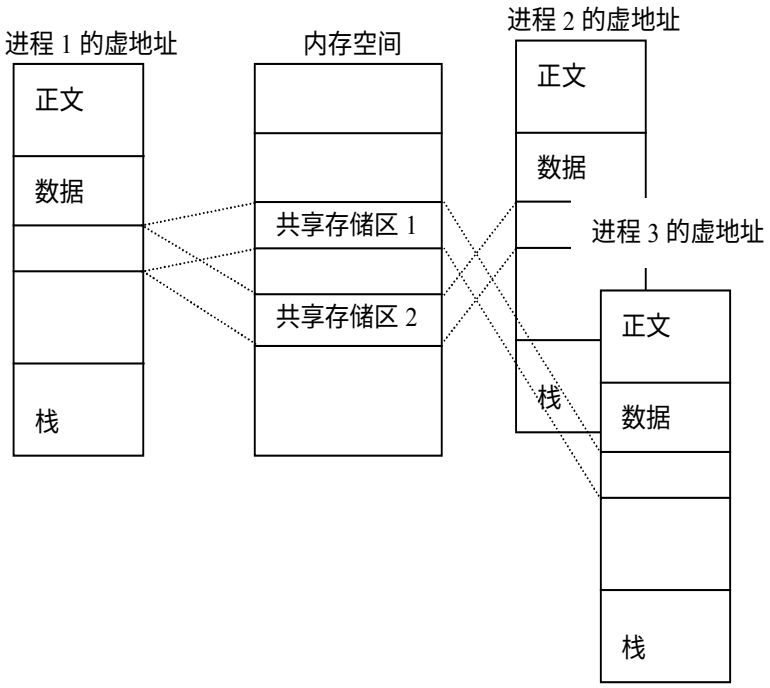

图 8.9　多个进程共享存储区通信

共享存储区的系统调用方式与消息通信的系统调用方式类似。

系统调用 shmat 从逻辑上将一个共享存储区附接到一个进程的虚地址空间上。

系统调用 shmdt 从一个进程的虚地址空间断开一个共享存储区。

系统调用 shmctl 改变共享存储区的属性及查询状态等相关参数。

进程用对普通内存的读写指令来读写共享存储区；在附接上共享存储区以后，该区就成为进程虚地址空间的一部分。进程以存取虚地址空间的方法进行存取，所以对共享存储区的读写不须系统调用。进程在读写共享存储区时需要考虑进程的互斥及同步问题，会采用上锁、解锁及睡眠、唤醒等方法来解决。

2．共享存储区的建立

进程用共享存储区通信之前首先应利用系统调用 shmget 建立一个共享存储区。

系统调用 shmget 的格式为：

```
int shmget(key,size,shmflg);
key_t key;
int size,shmflg;
```

其中：key 是共享存储区的关键词（名字）；size 是共享存储区的字节大小；shmflg 是用户设置的标志，通常用 IPC_CREAT 表示如果系统中无指名的共享存储区，则核心立即建立一个共享存储区，如果系统中已经建立有共享存储区，则忽略 IPC_CREAT。

在建立共享存储区时内核首先根据共享存储区的关键词查找共享存储区表，如果表的表项中存在该区，并且权限可以接受，则返回该表项的描述符；如果表的表项中不存在该区，用户又在标志 IPC_CREAT 中设置了创建一个新存储区，内核在证实共享存储区参数 size 符合系统共享存储区最大最小限制后,决定分配一个共享存储区。

内核首先从自由链表中取出第一个可用表项，放到活动表中，上锁并标识共享类型。设置该区的索引节点指针，分配引用计数，核心在共享存储区和系统区表中为新建立的共享存储区分配一个空表项，在共享存储区表中填上存储区的关键词、大小、页表起始地址、指向系统区表的指针等，另外内核还要在共享存储区表项中设置当它最后一个附接的进程退出时该区不应释放的标志，最后返回共享存储区描述符 shmid。

3．共享存储区的附接与断开

进程在获得了共享存储区的描述符后，使用系统调用 shmat 将一个共享存储区附接到它的进程虚地址空间上后，才能成为进程虚地址空间的一部分。

系统调用 shmat 的格式为：

```
char *shmat(shmid,shmaddr,shmflg);
int shmin,shmflg;
char * shmaddr;
```

系统调用的返回值是共享存储区所附接到的进程虚地址 viraddr。其中：shmid 是共享存储区的描述符；shmflg 是共享存储区的读写标志，shmflg 值为 SHM_RDONLY 表示只读，shmflg 值为 0 表示可擦写；shmaddr 是用户给定的共享存储区附接到的进程虚地址空间。

当进程不需要共享存储区时，用系统调用 shmdt 将共享存储区与进程虚地址空间断开。

系统调用 shmdt 的格式为：

```
int shmdt(shmaddr);
char shmaddr;
```

其中 shmaddr 是要断开连接的虚地址，即前面附接系统调用返回值 viraddr。

4．共享存储区的操纵

系统调用 shmctl 对共享存储区的状态信息进行读取和修改、断开进程与共享存储区的连接。当所有进程都断开了与共享存储区的连接时，便可以取消该共享存储区。

系统调用 shmctl 的格式为：

```
int shmctl(shmid,cmd,buf);
int shmid,cmd;
struct shmid_ds, *buf;
```

其中：buf 是用户缓冲区地址；cmd 是操作命令，该命令分为：

- 用于查询存储区的长度、当前连接的进程数、共享区创建者标识符等情况的命令；
- 用于设置或改变共享存储区权限、当前连接进程计数等属性的命令；
- 用于对共享存储区加锁、解锁的命令；
- 用于删除共享存储区标识符的命令。

例　共享存储区。

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHMKEY 666
#define K 1024
int shmid;
int main()
{
  int *pint, i;
  char *addr1,*addr2;
  shmid=shmget(SHMKEY,10*K,0777|IPC_CREAT);
  addr1=shmat(shmid,0,0);
  addr2=shmat(shmid,0,0);
  printf("addr1 0x%x addr2 0x%x\n\n",addr1,addr2);
  pint=(int*)addr1;
  for (i=0;i<256;i++)
    *pint++=i;
  pint=(int*)addr2;
  for (i=0;i<256;i++)
    printf("pos %d: %d\n",i,*pint++);
  shmdt(addr1);
  shmdt(addr2);
  shmctl(shmid,IPC_RMID,0);
```

```
    return 0;
}
```

运行结果：

```
addr1 0x40015000 addr2 0x40018000


pos 0: 0
pos 1: 1
...
pos 254: 254
pos 255: 255
```

8.3.5 信号量

信号量是一组原子操作，分别用 PV 表示，通常信号量值表示互斥共享使用的资源的值。P 操作表示减少一个信号量的值，V 操作表示增加一个信号量的值。通常一个信号量由以下几部分组成：

- 信号量的值；
- 最后操作信号量的进程的 pid；
- 等待信号量值增加的进程数；
- 等待信号量值为零的进程数。

信号量的系统调用允许若干进程在一组信号量上同步操作，一组信号量也称为信号量集。

1. 信号量集的创建

信号量系统调用 semget 用于创建信号量集，其格式为

```
int semget(key,nsems,semflag);
```

返回值为信号量的描述符。其中：key 为信号量集的名字；nsems 为信号量集中信号量的数目；semflag 与系统调用 msgget 中的 msgflg 含义一样，是用户设置的标志和访问方式。

核心在创建信号量集时首先检查用户是否具有建立信号量集的权限、建立的信号量数目是否合法以及当前用户是否有足够的资源。检查通过后便根据参数 nsems 分配信号量表项及信号量集表项并初始化，返回信号量集的描述符。如果用户系统调用 semget 的目的是为了获得信号量集的描述符，则仅当 nsems 既不为 0 又不大于该信号量集中的信号量数目时，核心才返回信号量集的描述符。

2．对信号量集的操纵

在信号量集创建成功后，用系统调用 semop 对信号量集进行操纵。其格式为：

```
int semop(semid,sops,nsops);
int semid;
struct sembuf * * sops;
unsigned nsops;
```

其中，semid 是 semget 返回的信号量集描述符；sops 是指向信号量集操作结构数组的指针；nsops 是该数组中 sembuf 结构的数目，结构 sembuf 描述如下：

```
struct sembuf {
  short sem_num;
  /*sem_num 为信号量编号，即在信号量集中的序号，表示要操作的信号量数组的元素
*/
  short sem_op;
  /*sem_op 为要进行的操作*/
  short sem_flg;
  /*为标志，如 IPC_NOWAIT 标志表示立即返回*/
}
```

其中 sem_op 为要进行的操作，核心根据 sem_op 来改变信号量的值：

- 如果 sem_op 为正值，则将其值加到信号量的值上，即作 V 操作；
- 如果 sem_op 为负值，则作 P 操作；
- 如果信号量的值大于操作值的绝对值，则核心将一个负整数加到信号量的值上，否则，核心将已经操作了的信号量恢复到系统调用开始时的值；
- 如果 sem_flg 和 IPC_NOWAIT 为真，便立即返回，否则进程睡眠等待。

8.3.6　进程通信应用实例——多程序间共享内存

```
/* 服务器端 server.c */
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#define SEGSIZE 1024
int shmid,semid;
```

```
char *shm;
union semun
{
  int val;
  struct semid_ds* buf;
  unsigned short* array;
};
int sem(key_t key) //生成信号量
{
  union semun sem ;
  int semid;
  sem.val=0;
  semid=semget(key,1,IPC_CREAT|0666);
  if (semid==-1)
  {
    printf("create semaphore error\n");
    exit(-1);
  }
  //初始化信号量
  semctl(semid,0,SETVAL,sem);
  return semid;
}
void d_sem(int semid) //删除信号量
{
  union semun sem;
  sem.val=0;
  semctl(semid,0,IPC_RMID,0);
}
int p(int semid)
{
  struct sembuf sops={0,+1,IPC_NOWAIT};
  return(semop(semid,&sops,1));
}
int v(int semid)
```

```
{
  struct sembuf sops={0,-1,IPC_NOWAIT};
  return(semop(semid,&sops,1));
}

void exit_func(int signum)
{
  shmdt(shm);
  shmctl(shmid,IPC_RMID,0);
  d_sem(semid);
  exit(0);
}
int main()
{
  key_t key;
  int i;
  char msg[50];
  struct semid_ds buf;
  signal(SIGINT,exit_func);
  signal(SIGQUIT,exit_func);
  key=ftok("/" ,0);
  shmid=shmget(key,SEGSIZE,IPC_CREAT|0600);
  if(shmid==-1)
  {
    printf("create shared momery error\n");
    return -1;
  }
  shm=(char*)shmat(shmid,0,0);
  if((int)shm==-1)
  {
    printf("attach shared momery error\n");
    return -1;
  }
  semid=sem(key);
```

```
  i=0;
  for(;;)
  {
    p(semid);
    sleep(1);
    msg[4]='0'+i;
    msg[5]='\0';
    sprintf(shm, "data%d", i++);
    v(semid);
  }
  return 0;
}
/* 客户端 client.c */
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#define SEGSIZE 1024
union semun
{
  int val;
  struct semid_ds* buf;
  unsigned short* array;
};
char *shm;
int sem(key_t key)
{
  union semun sem ;
  int semid;
  sem.val=0;
  semid=semget(key,0,0);
  if (semid==-1)
  {
```

```
    printf("get semaphore error\n");
    exit(-1);
  }
  return semid;
}
//等待信号量变成 0
void waitv(int semid)
{
  struct sembuf sops={0,0,0};
  semop(semid,&sops,1);
}
void exit_func(int signum)
{
  shmdt(shm);
  exit(0);
}
int main()
{
  key_t key;
  int shmid,semid;
  signal(SIGINT,exit_func);
  signal(SIGQUIT,exit_func);
  key=ftok("/", 0);
  shmid=shmget(key,SEGSIZE,0);
  if(shmid==-1)
  {
    printf("get shared momery error\n");
    return -1;
  }
  shm=(char*)shmat(shmid,0,0);
  if((int)shm==-1)
  {
    printf("attach shared momery error\n");
    return -1;
```

```
    }
    semid=sem(key);
    for(;;)
    {
      sleep(2);
      waitv(semid);
      printf("the msg get is %s\n" ,shm);
    }
    return 0;
  }
```

在终端 1 上运行服务端，在终端 2/3 上分别运行一个客户端，客户端可以得到以下类似的结果：

```
the msg get is data2
the msg get is data4
...
```

8.4 进程管理命令

8.4.1 ps 命令

ps 命令用于显示处于活动状态的进程信息。

命令格式：ps [option]

选项 option 主要有：

-e：列出正在执行的所有 Shell 进程，e 表示“everyone”；

-f：列出进程的全部清单；

-l：列出进程的长清单；

-t：列出某终端下产生的所有进程；

u username：列出用户 username 的所有进程。

例 #ps

```
PID   TTY   TIME   CMD
89     01    2:30   csh
78     01    2:28    ps
1924          0:00   htt
2438   1ft0  0:00    get
```

```
        2662    -    0:36    sen
        2866    -    0:15    dtl
        #
        #ps -el
        F    S UID PID  PPID  C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
        200003 A  0     1     0     0  60  20  80a   672                  -  6:54
ini
        340001 A 254  1914    1     0  60  20  6599   272 70073114        -  0:00
htt
        240001 A   0  2438    1     0  60  20  2589  408 7007e644       lft0  0:00
get
        240001 A   0  2662  4670    0  60  20  6dfb   948                  -  0:36
sen
        40001  A   0  2866    1    0  60   0  3d4f   536                -  0:15 dtl
        #
        #ps  – fu zhangh
        UID  PID  PPID  C  STIME   TTY  TIME CMD
        254  1914    1    0   23:45   -  0:00  htt
        254  1867    1    0   22:45   -  0:10  sne
        254  1428    1    0   22:45   -  0:10  rec
```

8.4.2　kill 命令

kill 命令用于中断后台进程或向正在运行的进程发送信号，在前面“ 进程通信 ”一节中已经介绍过该命令，此处主要补充说明的是当 kill 命令格式为：

```
kill  – 9 pid
```

时，则表示终止正在运行的进程号为 pid 的进程。

在终止进程前一定要知道所终止进程的 pid，用命令 ps – e 可以得到所要终止进程的 pid。

例　系统管理员要终止进程 httpd，首先用 ps 命令查得进程 httpd 的 pid 为 5198，然后用命令 kill-9 5198 终止该进程。最后再用命令 ps – ef | grep httpd 验证是否已经终止该进程。

```
        #ps  – ef | grep httpd
        UID  PID  PPID    C  STIME   TTY  TIME CMD
        root 5198    1    0   08:45   -   09:10 httpd
```

```
# kill -9 5198
#ps –ef | grep httpd

#
```

8.4.3 nice 命令

安排以更低或者更高优先级运行命令,即在输入命令的同时设定不同的优先级。

命令格式：nice [-increment] command [arguments]

该命令将降低命令 command 的 CPU 调度优先级，若使用 increment 参数，则优先级降低值为 increment,默认值为 20。increment 为负数表示要提高命令 command 的优先级。

例 降低优先级 10 级运行程序 client。

```
$nice –30 client
$
```

例 提升优先级 10 级运行程序 server，并带有参数-n。

```
$nice - –10 server -n
$
```

8.4.4 sleep 命令

sleep 命令使进程睡眠，命令格式为：sleep n

其中 n 为睡眠的秒数。

例 睡眠 5 秒

```
$sleep 5
$
```

8.4.5　wait 命令

wait 命令为等待后台进程结束，命令格式为：wait [pid]

如果指定了 pid，则等待该进程结束。

如果没有指定 pid，则等待该 Shell 中所有进程结束。

例　等待进程号为 12888 的进程结束。

```
$wait 12888
$
```

8.5　本 章 小 结

进程是操作系统中的一个重要概念，了解其相关知识并学会管理，无论对于系统维护还是程序设计都是有好处的。特别是进程的状态转换、进程表和 u 区等相关内容以及进程是如何启动和结束的、进程如何进行复制。

在这一章中还介绍了进程之间的通信，进程通信在 UNIX 系统开发中非常重要。利用系统提供的通信方式实现进程之间的通信比用其他语言方式实现更加优越。进程通信部分应该反复、细致地体会并上机练习编程。

上 机 练 习

1．练习 fork 的用法，理解父进程和子进程的概念，以及调用 fork 后它们之间的相互关系怎样。

2．熟悉 UNIX 系统中各种信号的意义。

3．熟悉本章介绍的几种进程间通信方式，例如信号、管道、消息、共享存储区和信号量等在程序设计中的基本用法。

4．分别编写一段利用信号、管道、消息、共享存储区和信号量进行进程间通信的程序。

5．上机练习本章第 4 节介绍的各种进程管理命令的使用。

6．当进程接收到一个软中断信号后，内核在进程的当前目录中创建一个“core”文件的同时会拷贝进程的 u 区、正文区、数据区和堆栈区。编写一段程序实现一旦检测到产生了“core”文件则将其删除。

习　题　八

1．什么是用户进程？什么是核心进程？简述用户进程与核心进程的异同。

2．什么是 UNIX 系统中区的概念？

3．试述进程区表用来做什么？

4．UNIX 操作系统为什么要采用多级反馈调度算法进行进程调度？

5．试编写程序来比较消息系统调用和在有名管道上的 read 和 write 的性能与速度。

6．用 UNIX 进程概念及结构思想说明设计一个 ps 命令的实现过程。

7．叙述进程作上下文切换的过程。

8．当进程正在将用户地址空间的数据拷贝到系统内核时，突然发生地址出错，这时会出现什么情况？

第 9 章　UNIX 系统调用

本章主要介绍 UNIX 的系统调用。系统调用是用户程序和操作系统核心之间的接口，用于程序员编程。UNIX 系统提供的系统调用有：输入/输出、文件系统、进程控制等。

本章主要内容

- UNIX 系统调用概述 :主要介绍 UNIX 系统调用及系统调用与一般函数调用的区别。
- 文件系统调用主要包括：

creat：创建文件
open：打开文件
close：关闭文件
read：读文件
write：写文件
lseek：读写时在文件中定位
stat、fstat、lstat：查询文件状态信息
link、unlink：文件的链接和拆链
select：监视文件描述符
fcntl：文件或记录的加锁操作
ioctl：设备特殊文件
dup：拷贝文件描述符
access：检查进程对文件的读写权
mount、umount：加载和卸载
chmod、chown：改变文件的访问模式和文件拥有者
chroot：改变调用进程私用的根目录

- 进程系统调用主要包括：

fork：创建新进程
exec：执行程序
exit：终止当前进程

wait：等待子进程终止，实现父进程与子进程同步

getpid：得到进程标识符

nice：改变进程优先级

brk：允许动态改变进程数据区大小

- 系统调用实例。

9.1　UNIX 系统调用概述

系统调用是 UNIX 操作系统核心提供给用户程序使用的操作系统服务，系统调用主要提供用户程序对文件进行读写、进程的创建、删除和控制以及数据的输入/输出等。由于 UNIX 系统核心的大多数由 C 语言编写，所以系统调用主要出现在 C 程序的函数中。

在 C 语言中存在一系列链接库，在链接库中含有一系列函数调用，如链接库 stdio.h 中含有标准输入、输出操作的 fopen、fclose，fread、fwrite、fseek、ftell 等，用户应用程序调用这些链接库函数。这些函数位于操作系统高层，在系统调用之上，需要通过系统调用才能和操作系统核心取得联系。

虽然系统调用也是一种函数调用，但与一般的函数调用相比，在表现方式上和实现效果上有明显不同：

- 表现方式上的不同：一般链接库调用时直接由调用过程转向被调用过程；系统调用不允许由调用过程直接转向被调用过程，要借助于信号（软中断），先进入系统核心，再转向相应的系统调用。
- 实现效果上的不同：程序函数库调用时只是子程序调用，一直在用户态上进行，不会进入操作系统核心，不能直接使用系统的底层服务；系统调用是从用户态切换到核心态下进行，直接进入操作系统核心使用底层服务，故优先级高，响应速度快，效率高。

图 9.1 显示了用户应用程序、库函数、系统调用与系统硬件的关系。

用户应用程序能够直接使用系统调用是 UNIX 操作系统的特点之一，UNIX 之所以能被大家首选为程序开发平台，与这一特点有很大关系。目前 UNIX 系统大约可以提供 70 多个系统调用，主要分类为文件系统调用、进程系统调用、数据通信系统调用和系统维护系统调用。这些系统调用实现了 UNIX 系统内的标准化，系统调用方式与操作系统的硬件平台及厂商无关，用户的应用程序可以在不同的 UNIX 系统之间移植。

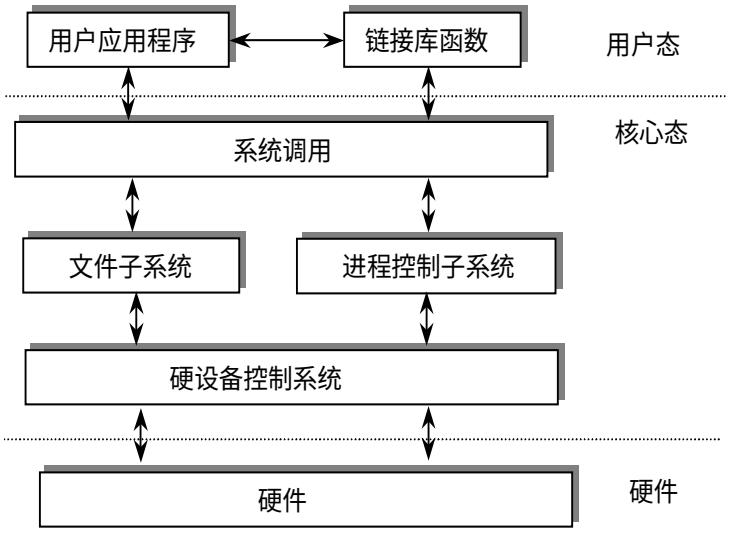

图 9.1 UNIX 系统与系统调用

9.2 文件系统调用

文件系统调用是 UNIX 系统调用的重要部分。用户应用程序通过文件系统调用来使用文件资源。

在用户程序中有关文件的操作主要有：创建文件、打开文件、读写文件、关闭文件。与文件操作相对应，文件系统调用有 creat、open、read、write、close。除这些调用之外，文件系统调用还增加了管理文件的系统调用 link、unlink、stat、fstat、lstat 等。

9.2.1 系统调用 creat 创建文件

系统调用 creat 创建一个新文件或重写一个旧文件，并将文件描述符 fd 返回给用户程序。用户利用文件描述符对文件进行读写。

系统调用格式为：

```
int creat(path,smode);
char *path;
int smode;
```

该系统调用的返回值为文件描述符。其中：path 为创建的文件名（包含全路径）；smode 为文件实际的权限。在文件修改模式命令 chmod 中，用户对文件读、写、执行的访问权 mode 用 3 位 8 进制数 mnk 表示（详见文件系统一章中的 chmod 部分），例如 mode=751 说明文件的所有者有读、写、执行权，同组用户有读、执行权，其他用户只有执行权限。如果用户先用 umask xyz 命令对文件的权限作了限制，则实际

建立的权限 smode 就是 xyz 与 mnk 作“ 与 ”运算的结果，例如 xyz=177，smode=177 & 751=175。默认情况下 xyz 为 777，则 smode 就是 mode。

系统核心按照 smode 指定的权限创建文件，如果创建成功，则返回文件描述符，否则返回-1。

返回值为-1 表示系统调用产生了错误，为了查出错误的种类，系统调用会在一个外部整数 errno 中留下一个错误号给应用程序判定。

如果文件已经存在，则重写一个旧文件意味着要将文件中原有内容清掉。

C 语言中的链接库函数 fopen 可以创建文件，但是不能指定文件的访问权限。

例　在用户目录/home/liuxx/下创建文件 file1，该文件的所有者有读、写、执行权，同组用户有读、执行权，其他用户只有执行的权限。

```
#include <stdio.h>
#include <fcntl.h>
#define smode 00751
int main()
{
  int fd;
  if ((fd = creat("/home/ liuxx /file1", smode)) == -1)
    perror("Can't create file /home/ liuxx /file1");
  else
    printf("create /home/liuxx/file1 ok!");
  exit(0);
}
```

9.2.2　打开文件 open

文件创建好以后，必须用系统调用 open 将文件打开之后才能对文件进行读/写操作。

系统调用格式为：

```
int open(path,rwmode);
char *path;
int rwmode;
```

该系统调用返回值为打开文件的描述符。如果打开文件成功，则返回文件描述符，否则返回-1。其中 path 为含有全路径的要打开的文件名；rwmode 为头文件 fcntl.h 中定义的对打开文件的访问模式：

rwmode = 0 表示可读，O_RDONLY；

rwmode = 1 表示可写，O_WRONLY；

rwmode = 2 表示可读、可写，O_RDWR。

例 在用户目录/home/liuxx/下已有文件 file1，将该文件打开并且可读、可写。

```
#include <stdio.h>
#include <fcntl.h>
#define rwmode 2
int main()
{
  int fd;
  if ((fd = open("/home/liuxx/file1", rwmode)) == -1)
    perror("Can't open file /home/liuxx/file1");
  else
    printf("open /home/liuxx/file1 ok!");
  exit(0);
}
```

9.2.3 关闭文件 close

当用户在文件使用完之后不再需要时，用系统调用 close 断开用户程序与文件之间的通路，关闭该文件。

系统调用格式为：

```
int close(fd);
int fd;
```

其中 fd 为文件描述符,根据 fd 从用户文件描述符表中得到指向文件表项的指针 fp，由 fp 得到文件表项中的 f.count，并对 f.count 作减 1 操作。操作结果如果不为 0，表示还有进程在使用该文件，此时不能收回文件表项；操作结果为 0，表示无进程使用该文件，将此文件表项置空。

9.2.4 读文件 read

当文件打开后，可以用系统调用 read 对文件进行读操作。系统调用 read 的格式为：

```
int read(fd,buffer,count);
int fd;
char *buffer;
```

```
        unsigned count;
```

其中：fd 是 open 返回的文件描述符；buffer 是一个缓冲区，用于存放所读的数据；count 是要读的字节数；如果系统调用成功，则返回实际读取的字节数。在读时，如果读指针已经到达文件末尾但还没有读够指定的字节数 count，也立即停止，所以 number 的值可以小于等于 count。如果试图读一个已被加锁的文件，则 read 进程必须等待直到锁被打开。

在 read 系统调用中如果未指定需要读的字节数，则通常是 1 个字节。

例　首先打开文件/etc/passwd，然后将文件中前 1 024 字节的内容读到缓冲区 buffer 中。

```
#include <stdio.h>
#include <fcntl.h>
#define rwmode 0
int main()
{
  int fd;
  char buffer[1024];
  int n;
  if ((fd = open("/etc/passwd", rwmode)) == -1)
    perror("Can't open file /etc/passwd");
  else
    printf("open /etc/passwd ok!\n");
  n = read(fd,buffer,sizeof(buffer) - 1);
  buffer[n] = '\0';
  printf("%s\n",buffer);
  close(fd);
  exit(0);
}
```

例　对同一个文件用两个文件描述符读。

```
#include <stdio.h>
#include <fcntl.h>
#define rwmode 1
main()
{
  int fd1,fd2;
```

```
  char buffer[2048];
  if ((fd1 = open( "/etc/passwd" , rwmode)) = = -1)
    error( "First can ' t open file /etc/passwd" );
  else
    printf( "Firat open /etc/passwd ok!" );
  if ((fd2 = open( "/etc/passwd" , rwmode)) = = -1)
    error( "Second can ' t open file /etc/passwd" );
  else
    printf( "Second open /etc/passwd ok!" );
  read(fd1,buffer,1024);
  printf( "%s\n" ,buffer);
  close(fd1);
  read(fd2,buffer,2048);
  printf( "%s\n" ,buffer);
  close(fd2);
}
```

9.2.5 写文件 write

写系统调用是在文件被 open 系统调用打开后，从缓冲区 buffer 将指定字节的内容写到由 open 返回的文件描述符所指定的文件中。write 系统调用格式为：

```
int write(fd,buffer,count);
int fd;
char *buffer;
unsigned count;
```

其中：fd 是 open 返回的文件描述符，该文件存放由 write 写入的内容；buffer 是一个缓冲区，用于存放要写的内容；count 是要写的字节数；如果系统调用成功，则返回实际写的字节数。在写时，如果写指针已经到达文件末尾但还没有写够所指定的字节数 count，立即停止，所以 number 的值可以小于等于 count。如果试图写一个已被加锁的文件，则 write 进程必须等待直到锁被打开。

在 write 系统调用中如果未指定需要写的字节数，则通常是 1 个字节。

例 拷贝文件，该拷贝不保护源文件的权限，也不保护与目标文件相同的文件内容。


```
#include <stdio.h>
#include <fcntl.h>
```

```
#include <errno.h>
#define resounce_mode 0
#define destination_mode 0774
#define FILESIZE 1024
int main(int argc,char *argv[])
{
  int resource_fd,destination_fd;
  int readbytes,writtenbytes;
  char buffer[FILESIZE], *p;
  if (argc != 3)
  {
    printf("Usage: copy from resource file to destination file\n  %s src_file
          dest_file\n",argv[0]);
    exit(1);
  }
  if ((resource_fd = open(argv[1], resounce_mode)) == -1)
  {
    perror("Can't open source file");
    exit(1);
  }
  if ((destination_fd = creat(argv[2], destination_mode )) == -1)
  {
    perror("Can't create destination file");
    exit(1);
  }
  while (readbytes = read(resource_fd,buffer,FILESIZE))
  {
    p = buffer;
    if ((readbytes == -1) && (errno != EINTR))
      break;
    else if (readbytes>0)
    {
      while (writtenbytes = write(destination_fd, p, readbytes))
      {
```

```
            if ((writtenbytes == -1) && (errno != EINTR))
              break;
            else if (writtenbytes == readbytes)
              break;
            else if (writtenbytes > 0)
            {
              p += writtenbytes;
              readbytes -=  writtenbytes;
            }
          }
          if (writtenbytes == -1)
            break;
        }
      }
      close(resource_fd);
      close(destination_fd);
      exit(0);
    }
```

通常在文件param.h中用参数NOFILE限制一个用户应用程序能同时打开的文件数，一般NOFILE=20（Linux上一般是256）。

9.2.6 文件系统调用lseek

文件系统调用read和write是对文件的顺序访问，每次操作都紧跟在上次操作之后。如果要通过I/O位置指定实现对文件的随机顺序存取，则需要系统调用lseek的配合使用。系统调用lseek的格式为：

```
int lseek(fd,offset,origin);
int fd,origin;
long offset,pos;
```

其中：fd为文件描述符；origin是文件读写位置移动的初始值，特殊点为：

SEEK_SET = 0，表示文件开始；

SEEK_CUR = 1，表示文件当前位置；

SEEK_END = 2，表示文件尾。

offset是文件读写位置移动的偏移量，是一个长整型值，如偏移量为0，则为0L，如偏移量为1 034，则为1 034L；返回值是文件读写位置移动到的新的绝对值，

即随后的读写开始处，如果系统调用出错，则返回-1。

系统调用 lseek 依靠调整文件表中的字节偏移量值来实现，其处理文件就像处理一个大的数组一样，非常方便，但是访问速度却比 read、write 慢。

例　在文件/home/netstaff/sev1 的末尾添加内容。

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#define rwmode 1
int main()
{
  int fd,n;
  long pos,offset;
  char buffer[512];
  if ((fd = open("/home/xiexie/file1", rwmode)) == -1)
    perror("Can't open file /home/netstaff/sev1");
  else
    printf("open /home/netstaff/sev1 ok!\n");
  if (lseek(fd,0L,SEEK_END) == -1)
  {
    printf("lseek can't moving");
    close(fd);
    return -1;
  }
  else
  {
    strcpy(buffer,"append something\n");
    write(fd,buffer,strlen(buffer));
    printf("write completed\n");
  }
  close(fd);
  exit(0);
}
```

9.2.7 文件系统调用 stat、fstat 和 lstat

系统调用 stat、fstat 和 lstat 用于进程查询文件的状态，返回文件类型、文件大小和文件所有者、文件的存取权限、文件的修改时间、链接计数、索引节点号等。系统调用是将文件所对应索引节点中的文件信息字段写到进程的数据结构中。stat 和 fstat 系统调用用于得到硬链接文件的信息，lstat 系统调用用于得到软链接文件的信息。格式分别为：

```
int stat(path,statbuf);
char *path;
struct stat *statbuf;

int fstat(fd,statbuf);
int fd;
struct stat *statbuf;

int lstat(path,statbuf);
char *path;
struct stat *statbuf;
```

其中：path 是文件名，fd 是系统调用 open 返回的文件描述符；statbuf 是数据结构地址，该数据结构用于存放文件信息。struct stat 结构在文件 stat.h 中定义，文件的基本内容如下：

```
struct stat
{
  mode_t   st_mode;          /* 文件模式，包括文件类型、访问许可机制等 */
  ino_t     st_ino;          /* 文件 inode 号 */
  dev_t     st_dev;          /* 文件驻留的逻辑设备 ID*/
  dev_t     st_rdev;         /*设备的 ID，用于字符文件或块文件*/
  nlink_t    st_nlink;  /*文件的链接数*/
  uid_t      st_uid;          /*文件拥有者的用户 ID*/
  gid_t      st_gid;          /*文件拥有者组的 ID*/
  off_t      st_size;         /*文件逻辑长度*/
  time_t     st_atime;  /*文件最后一次访问时间*/
  time_t     st_mtime;  /*文件最后一次修改时间*/
  time_t     st_ctime;  /*文件状态信息最后修改时间*/
```

```
    long        st_blksize;     /*I/O 预取块大小*/
    long        st_blocks; /*块预留数*/
}
#define S_IFMT     0170000   /*文件类型*/
#define S_IFDIR         0040000   /*目录*/
#define S_IFCHR    0020000   /*字符设备文件*/
#define S_IFBLK    0060000   /*块设备文件*/
#define S_IFREG    0100000   /*普通文件*/
#define S_IFIFO         0010000   /*FIFO 文件*/
#define S_IFLNK    0120000   /* 符号链接*/
#define S_IFSOCK   0140000   /* 套接字 */
#define S_ISUID         04000     /*执行时 setid*/
#define S_ISGID         02000     /*执行时 setgid*/
#define S_ISVTX    01000     /*粘着位*/
#define S_IREAD    00400     /*文件属主读许可*/
#define S_IWRITE   00200     /*文件属主写许可*/
#define S_IEXEC    00100     /*文件属主执行/搜索许可*/
```

在系统 Shell 中可以用命令 ls 显示这些信息。

例　查询文件信息。

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>
#define rwmode 0
int main(int argc,char *argv[])
{
  int fd;
  long pos,offset;
  char buffer[1024];
  struct stat statbuf;
  if (argc!= 2)
  {
    fprintf(stderr, "Usage: %s filename\n",argv[0]);
    exit(1);
```

```
    }
    if (stat(argv[1],&statbuf) == -1)    /*用文件名查文件状态*/
      printf("stat can't get %s states\n",argv[1]);
    else
      printf("stat: can get %s states\n",argv[1]);
    if ((fd = open(argv[1], rwmode)) < 0)
      printf("%s: can't open\n",argv[1]);
    else
    {
      if (fstat(fd,&statbuf) == -1)    /*用 fd 查文件状态*/
        printf("fstat can't get %s states\n",argv[1]);
      else
        printf("fstat: can get %s states\n",argv[1]);
      close(fd);
    }
    exit(0);
  }
```

9.2.8 文件系统调用 link 和 unlink

系统调用 link 是为文件建立一个新的路径名或为文件建立一个新的目录项,是文件名与索引节点之间的链接。在 UNIX 中分为硬链接和软链接，这两种链接的差异很小。硬链接借助于目录项将文件名和索引节点直接链接起来；软链接借助于符号链（又称为符号链接）将链接文件作为指向另一个文件名的指针使用。

每一个索引节点含有该索引节点的硬链接数，即含有该索引节点的目录项的总数。用 link 建立的硬链接使索引节点中的链接计数项加 1，仅分配目录项，并不占用多余的磁盘空间。

系统调用 link 的格式为：

```
int link(path1, path2);
char * path1, * path2;
```

其中 path1 是指向已存在文件路径名的指针，即源文件名；path2 是指向新文件路径名的指针，即新文件名。

文件系统对文件的每个系统调用 link 都有一个路径名,进程可以用任意一个路径名存取文件。

例　将三个路径名指向同一文件。

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>
#define rwmode 0
int main()
{
  int fd[3];
  int i, n;
  char buffer[512];
  link("/home/stu01/exer1.c","/home/stu02/exer1.c");
  link("/home/stu03/exer1.c","/home/stu02/exer1.c");
  if ((fd[0] = open("/home/stu01/exer1.c", rwmode)) == -1)
    perror("Can't open file /home/stu01/exer1.c");
  else
    printf("open /home/stu01/exer1.c ok!\n");
  if ((fd[1] = open("/home/stu02/exer1.c", rwmode)) == -1)
    perror("Can't open file /home/stu02/exer1.c");
  else
    printf("open /home/stu02/exer1.c ok!\n");
  if ((fd[2] = open("/home/stu03/exer1.c", rwmode)) == -1)
    perror("Can't open file /home/stu03/exer1.c");
  else
    printf("open /home/stu03/exer1.c ok!\n");
  for (i=0;i<3;i++)
  {
    n = read(fd[i],buffer,sizeof(buffer) - 1);
    buffer[n]= '\0';
    printf("file%d: %s\n",i+1,buffer);
    close(fd[i]);
  }
  exit(0);
}
```


系统调用 unlink 与系统调用 link 正好相反，是拆掉文件的一个路径名或目录

表项。如果一个文件的路径名或目录表项被拆掉了，就不能再使用该路径名对文件进行操作。系统调用 unlink 的格式为：

```
int unlink(path);
char *path;
```

其中 path 是指向要拆掉的文件路径名指针。

对文件 filename 使用系统调用 unlink(“filename1”)，就是把文件名 filename1 拆掉，这之后与该文件名有关的系统调用与操作都不能再使用，如不能使用 open(〝filename1〞,smode)。特殊情况是，如果在文件打开期间用了系统调用 unlink 并执行成功,那么打开文件的进程就不能成功使用与文件路径名有关的操作或系统调用，如 stat、stat 等。但是由于打开时文件引用计数已经加 1，尽管内核执行了系统调用 unlink，却并不清除该文件的内容，仍然可以使用与该文件描述符有关的正常操作或系统调用，如 write、read、lseek、fstat 等，只有在文件用了系统调用 close 被关闭以后，才完成 unlink 清除文件内容。

下面的例子说明了上述情况，open 之后用 unlink，后面的 stat 就不能成功执行，而 fstat、read 能成功执行。

例 unlink 使用。

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>
#define rwmode 0
int main(int argc,char *argv[])
{
  struct stat statbuf;
  char buffer[1024];
  int fd,readbytes;
  if (argc != 2)
  {
    fprintf(stderr, "Usage: %s filename\n",argv[0]);
    exit(1);
  }
  if ((fd = open(argv[1], rwmode)) < 0)      /*打开文件*/
  {
    printf("%s: can't open\n",argv[1]);
```

```
            exit(1);
          }
          else
            printf("%s can open\n",argv[1]);
          if (unlink(argv[1]) == -1)    /*拆掉文件路径名或索引表项*/
          {
            printf("%s: can't unlink\n",argv[1]);
            exit(1);
          }
          else
            printf("%s unlink succeeded!\n",argv[1]);
          if ((readbytes=read(fd,buffer,sizeof(buffer)-1))>0)    /*读该文件 1024 字节
内容*/
          {
            buffer[readbytes]='\0';
            printf("read succeeded!\n");
            printf("%s",buffer);
          }
          if (fstat(fd,&statbuf) == -1)     /*用 fd 查文件状态*/
            printf("fstat can't get %s states\n",argv[1]);
          else
            printf("fstat: can get %s states\n",argv[1]);
          if (stat(argv[1],&statbuf) == -1)     /*用文件名查文件状态*/
            printf("stat can't get %s states\n",argv[1]);
          else
            printf("stat get %s states\n",argv[1]);
          exit(0);
        }
```

9.2.9　系统调用 select

当进程执行 I/O 操作时，进程会阻塞等待 I/O 事件的到来。但是当进程要从多个输入源输入数据时，进程无法知道哪一个输入设备的文件描述符（由于 UNIX 对设备管理通过设备文件实现）会产生下一个输入，这时就要求识别输入的多文件描述符，在 UNIX 系统中主要有以下几种识别方式：

(1) 用轮询方法、使用非阻塞的 I/O；

(2) 用 SIGPOLL 信号，使用异步 I/O；

(3) 使用 POSIX.1b 异步 I/O；

(4) 使用 select 来阻塞直至输入有效；

(5) 使用 poll 来阻塞直至输入有效；

(6) 创建一个独立的进程来监控每个文件描述符。

UNIX 的 BSD 提供了系统调用 select 用于监控文件描述符。与轮询不同，select 调用在监控时，读、写等进程不必阻塞等待，这样不会浪费 CPU 的时间。该系统调用的格式为：

```
int select(fd_n,readfds,writefds, exceptfds,timeout);
int fd_n;
fd_set * readfds, * writefds, * exceptfds;
struct timeval * timeout;
```

其中：fd_n 是文件描述符集中要被检测的数值，必须大于要检测的最大文件描述符加 1；readfds 是将被读监控的文件描述符集；writefds 是将被写监控的文件描述符集；exceptfds 用于检测异常文件描述符；timeout 是 select 被强制返回的时间周期期限，timeout 为 NULL 时 select 可以无限地阻塞而不返回。使用该系统调用应包含 time.h（在 sys 目录中）。如果 select 被某个信号软中断，则该系统调用返回值为-1，并设置 errno 为 EINTR。

用整数数组存放文件描述符集，对该数组的位操作有：

FD_SET：设置文件描述符集的位；

FD_CLR：清除文件描述符集的位；

FD_ZERO：设置文件描述符集的所有位；

FD_ISSET：使用 select 之后用来检测屏蔽码 fdset 中对应于文件描述符的位是否被设置。

例 使用 select 进行读监控的程序片断。

```
#include <string.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>
void monitor_fds(int fd[],int num_fds,int max_fd)
{
  fd_set readfdset;
```

```
int i,num_ready,num_now,readbytes;
char buffer[255];
num_now = num_fds;
while (num_now>0)
{
  FD_ZERO(&readfdset);      /*设置文件描述符屏蔽*/
  for (i=0; i<num_fds; i++)
    if (fd[i]>=0)
      FD_SET(fd[i],&readfdset);
    num_ready = select(max_fd,&readfdset,NULL,NULL,NULL);
    for (i = 0; i<num_fds && num_ready>0; i++)
    {
      if ((fd[i]>= 0) && FD_ISSET(fd[i],&readfdset))
      {
        readbytes = read(fd[i],buffer,sizeof(buffer));
        num_ready--;
        if (readbytes>0)
          ; //call io_processing(buf,readbytes);       /*调用 IO 处理*/
        else
        {
          fd[i] = -1;
          num_now--;
        }
      }
    }
  }
}
```

9.2.10　fcntl 系统调用

系统调用 fcntl 实现对打开文件（用系统调用 open）或记录的加锁操作。该系统调用格式为：

```
int fcntl(fd,cmd,arg);
int fd,cmd;
long arg;
```

其中：fd是文件描述符；cmd是锁操作的类型；arg规定各种参数，通常在文件fcntl.h中定义，含义如下。

F_DUPFD：返回大于等于arg的最低序号的文件描述符。

F_SETFD：以arg的低位设置“执行后关闭”标志（若为1，文件在系统调用exec后是关闭的）。

F_GETFD：返回“执行后关闭”标志的值。

F_GETFL：设置文件状态标志（O_NDELAY是不因等待I/O而睡眠；O_APPEND是附加欲写的数据到文件尾）。

F_GELEL：取文件状态标志。

F_GELLK：对参数arg中规定的锁，读取它阻碍应用的第一个锁，然后改写arg参数。如果这样的锁不存在，就把arg中的1_type修改为F_UNLCK。

F_SETLK：对由arg规定的文件上锁或解锁，如果不能上锁则返回-1。

F_SETLKW：对由arg规定的文件上锁或解锁，如果不能上锁，则睡眠。

在一个文件中几个读锁可以重叠使用，但读锁不能和写锁重叠使用。

```
struct flock
{
  short 1_type;  /*锁操作类型:F_RDLCK 加读锁;F_WRLCK 加写锁;F_UNLCK 解锁*/
  short 1_whence;  /*锁偏移量从文件头开始(0);锁偏移量从当前文件指针开始(1);
                     锁偏移量从文件尾开始（2）;*/
  long 1_start;   /*按照 1_whence 进行解释的字节偏移量*/
  long 1_len;    /*欲上锁的字节数,若为 0,从 1_start 到文件尾上锁*/
  short 1_pid;    /*对文件上锁的进程 ID*/
}
```

锁操作包括：

测试属于其他进程的锁并立即返回，指明是否发现其他锁；

设置一个锁并进入睡眠直到成功；

设置一个锁，如果不成功，立即返回；

内核关闭文件（用系统调用close）时，自动释放由某个进程设置的锁。

9.2.11 ioctl系统调用

ioctl是设备特殊文件的系统调用，用于获取设备的状态信息和设置设备控制选项。该系统调用的格式为：


```
int ioctl(fds,request,&info_t);
int fds,request;
```

```
struct info_t;
```

其中：fds 是设备文件描述符；request 是系统调用发出的请求；info_t 是存放设备信息的数据结构。

对于音频设备，数据结构 info_t 为数据结构 audio_info，在文件 audio.h 中定义，详细内容如下：

```
typedef struct audio_info
{
  audio_prinfo_t  play;              /*输出状态信息*/
  audio_prinfo_t  record;            /*输入状态信息*/
  uint_t          monitor_gain;      /*输入到输出 mix*/
  uchar_t         output_muted;     /*如果输出 muted 非零*/
  uchar_t_xxx[3];                   /*保留将来使用*/
  uint_t_yyy[3];                    /*保留将来使用*/
}  audio_info_t;
```

其中 audio_prinfo_t 定义为：

```
struct audio_pinfo
{
  /*下面的值描述音频数据编码*/
  uint_t  sample_rate;  /*每秒采样*/
  uint_t  channels;    /*隔行扫描信道数*/
  uint_t  precision;    /*每次采样比特数*/
  uint_t  encoding;    /*数据编码方法*/
  /*下面的值控制音频设备配置值*/
  uint_t  gain;        /*音量级别*/
  uint_t  port;        /*选择 I/O 端口*/
  uint_t  avail_ports;  /*可提供的 I/O 端口*/
  uint_t  _xxx[2];     /*保留将来使用*/
  uint_t  buffer_size;  /*I/O 缓冲大小*/
  /*下面的值描述现在设备状态*/
  uint_t    samples;    /*已转换的采样数*/
  uint_t    eof;         /*用于输出的文件终止计数*/
  uchar_t   pause;       /*暂停则非零，继续则零*/
  uchar_t   error;       /*溢出为非零，否则为零*/
  uchar_t   waiting;     /*进程要访问则非零*/
```

```
    uchar_t    balance;      /*立体声信道平衡*/
    ushort_t   minordev;
    /*下面的值是只读设备状态标志*/
    uchar_t    open;        /*如果打开访问许可则非零*/
    uchar_t    active;      /*如果 I/O 激活*/
  } audio_pinfo_t
```

audio_pinfo_t 是 audio_info_t 数据结构的成员，buffer_size 是数据结构 audio_pinfo_t 的成员,buffer_size 记录了在数据传送给读请求之前设备驱动程序所积聚的音频数据块的大小。如果程序发送和接收的块与相应的 buffer_size 设置匹配，音频的发声效果更理想。在音频程序中，使用 ioctl 可以确定块的大小。

例　在 Sun Solaris 系统中，使用 AUDIO_GETINFO 请求系统调用 ioctl 提供音频设备信息。ioctl 系统调用为：

```
ioctl(audio_fd,AUDIO_GETINFO,&audio_info_t);
```

9.2.12　其他的文件系统调用

- dup 系统调用

系统调用 dup 用于将一个文件描述符拷贝到用户文件描述符表中的第一个空槽中，并给用户返回一个新的文件描述符。该系统调用的格式为：

```
int dup(fd);
int fd,newfd;
```

其中：fd 为要拷贝的文件描述符；系统调用返回新的文件描述符。通过文件描述符的复制，文件所对应的引用计数加 1，文件表项中增加了指向该文件的描述符表项。用系统调用 close（fd）只关闭文件描述符，不能关闭 newfd，必须用系统调用 close（newfd）才能关闭 newfd。

- access 系统调用

系统调用 access 用于检查调用进程对文件是否具有读、写、执行权，此处基于的是真正用户标识符，而不是有效用户标识符。该系统调用的格式为：

```
int access(filename,mode);
char * filename;
int mode;
```

其中：filename 为调用进程要检查的文件路径名；mode 为读、写、执行组合模式。

- mount 系统调用

操作系统管理设备时将设备看作一个文件。系统调用 mount 将一个设备中的文件系统挂到一个已存在的文件系统目录树中。系统调用 mount 允许用户以文件系统

方式存取设备中的数据。该系统调用的格式为：

```
int mount(special path,directory path,mmode);
char * special path, * directory path;
int mmode;
```

其中：special path 是含有要安装文件系统的设备特殊文件名；directory path 是已存在的文件系统目录中的目录，要安装的文件系统将要挂到的地方，也称为挂接点；mmode 是安装后的文件系统能被访问的方式，与系统调用 open 中的 rwmode 相似。如果用系统调用

```
mount("/dev/rdsk/rm0" , "/mnt" ,0)
```

则表示将设备特殊文件所对应的设备中的文件系统挂到已存在的文件系统上去，挂接点是已存在文件系统的/mnt 目录，挂上以后对该目录可进行的操作是只读。

- umount 系统调用（umount 与 unmount 在多数 UNIX 上通用，但似乎 umount 是趋势，这里改用 umount）

与系统调用 mount 相反，umount 是从已使用的文件系统上拆掉一个已经挂接的设备文件系统。该系统调用的格式为：

```
int umount(dir);
char *dir;
```

其中 dir 是挂接点。

和 mount("/dev/rdsk/rm0", "/mnt", 0)对应的拆掉挂接是 umount("/mnt")。

- chmod 系统调用

系统调用 chmod 用于修改文件或目录的访问模式（权限），与命令 chmod 完全对应。该系统调用的格式为：

```
int chmod(filename,mode);
char * filename;
int mode;
```

其中：filename 为文件或目录名；mode 为访问模式，详见命令 chmod。

- chown 系统调用

该系统调用用于改变一个文件或目录的所有者或用户组，只能由系统的超级用户或文件、目录的所有者使用。该系统调用的格式为：

```
int chown (name,owner,group);
char * name;
int owner, grup;
```


其中：name 为文件或目录名；owner 和 group 为新的（改变后）的所有者或用户组的标识符（ID）。

改变了所属关系后原来的所有者便失去了原有的权限。

- chroot 系统调用

系统调用 chroot 用于改变调用进程私用的根目录。该系统调用的格式为：

```
int chroot(filename);
char * filename;
```

filename 是新设置的根。

9.2.13 系统调用综合示例

在 UNIX 系统中，标准输入设备的文件描述符为 0；标准输出设备的文件描述符为 1；标准错误输出设备的文件描述符为 2。

例 把键盘输入输出到屏幕，按 Ctrl+D 结束。

```
#include <stdio.h>
int main()
{
  int count;
  char buffer[512];
  while ((count = read(0,buffer,sizeof(buffer))) > 0)
    write(1,buffer,count);
  exit(0);
}
```

在 UNIX 的工作站上大多配置有设备，像音频设备（audio、麦克风和扬声器），先进先出（FIFO）设备。这些设备的管理文件是特殊文件，在 UNIX 系统中对这些特殊文件的读写操作与普通文件一样。

例 在 Sun 工作站上使用音频设备，只有主控台用户才能使用这些设备。管理音频设备的目录为/dev/audio。本例通过系统调用实现从麦克风读数据并将读入的数据写到扬声器文件的一个录音过程。

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stropts.h>
#include <errno.h>
```



```
#define BUFSIZE 1024
int main()
```

```
{
  int audio_fd,readbytes,writtenbytes,storedbytes;
  char buf[BUFSIZE], *p;
  /*打开麦克风、扬声器管理特殊文件*/
  while (((audio_fd = open("/dev/audio",O_RDWR)) == -1) && (errno == EINTR))
  {
    perror("Can't open microphone");
    exit(1);
  }
  if (audio_fd<= 0)
  {
    perror("Open microphone error");
    exit(1);
  }
  /*完成从麦克风特殊文件读，并将读的数据写到文件/dev/audio 过程*/
  for(;;)
  {
    /*从麦克风特殊文件读数据到 buf*/
    while (((readbytes = read(audio_fd,buf,BUFSIZE)) == -1) && (errno ==
EINTR))
    {
      perror("Can't read microphone");
      exit(1);
    }
    /*从 buf 写数据到扬声器设备特殊文件*/
    storedbytes = readbytes;
    p = buf;
    while (storedbytes > 0)
    {
      if ((writtenbytes = write(audio_fd,p,storedbytes)) >= 0)
      {
        storedbytes -= writtenbytes;
        p += writtenbytes;
      }
```

```
        else if (errno != EINTR)
        {
          break;
        }
      }
    }
    /*关闭设备特殊文件*/
    close(audio_fd);
    exit(0);
  }
```

音频设备是块传送设备，如果所写入的缓冲区与音频设备驱动程序的传送块大小不相同，可能会使语音听起来断断续续，这时应该用系统调用 ioctl 检测缓冲区大小与设备信息的匹配情况。

对音频设备，定义在 audio.h 中的 audio_info 数据结构 audio_info_t 记录了配置信息。

下面的程序是从音频设备读取传送块到一个缓冲区，传送块大小为 BLKSIZE 个字节。音频设备是以非阻塞 I/O 方式打开。

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define BLKSIZE 1024  /*此处定义传送块大小为 1024*/
int main()
{
  char buffer[BLKSIZE], *bp;
  int read_bytes;
  int audio_fd;
  unsigned needed_bytes;
  if ((audio_fd = open("/dev/audio",O_NONBLOCK | O_RDWR)) == -1)
  {
    perror("Can't open audio device");
    exit(1);
  }
```

```
    bp=buffer;
    needed_bytes = BLKSIZE;
    while (needed_bytes != 0)
    {
      printf("#\n");
      read_bytes = read(audio_fd,bp,needed_bytes);
      if ((read_bytes = -1) && (errno != EAGAIN))
        break;
      if (read_bytes >0)
      {
        bp += read_bytes;
        needed_bytes -= read_bytes;
      }
    }
    close(audio_fd);
    exit(0);
  }
```

9.3　进程系统调用

在 UNIX 系统中，用户应用程序通过进程系统调用控制进程。本节从程序设计角度介绍进程系统调用。

9.3.1　fork 系统调用

在 UNIX 系统中进程 0 是系统引导时创建的，除此之外，系统中所有的进程均由系统调用 fork 创建。fork 系统调用用于创建一个新的子进程，该系统调用的格式为：

```
pid = fork();
```

其中 pid 是创建的子进程的标识符。

要创建子进程，要求系统有足够的存储资源，并且创建子进程的用户所拥有的正在运行的进程数要在系统许可的条件下。只要进程表大小许可，超级用户可以按需要创建任意多个进程。一般用户不能使用进程表的最后一行，而超级用户可以使用进程表的最后一行，这就是为什么在紧急情况下超级用户可以用命令 kill 终止其他进程。如果不能满足条件则该系统调用失败，返回值为-1。

如果系统调用成功，则内核完成以下操作。

(1) 为子进程在进程表中分配一个空的 proc 结构，子进程继承父进程真正用户标识符和有效用户标识符以及优先级等。

(2) 为子进程分配一个惟一的进程标识符 pid，该进程标识符比系统最近一次分配的进程标识符大 1；如果进程标识符已达到最大值，则从 1 开始查找原来已经分配，而进程已经结束的标识符。

(3) 复制一个父进程上下文的逻辑副本。对于父进程可被共享的部分，如正文区，内核只增加该共享区的引用计数而不真正复制；对于父进程非共享部分则将其复制到子进程的内存区。

(4) 增加与该进程相关联的文件表和索引节点表的引用数。

(5) 父进程调用 fork 的返回值为子进程的标识符，如果创建子进程发生错误，则返回值为-1；子进程调用 fork 的返回值为 0。

父进程创建一个子进程，子进程和父进程在系统中并发运行。如图 9.2 所示。

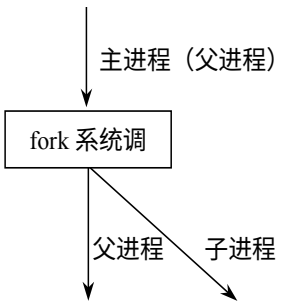

图 9.2 fork 系统调用图

例 fork 系统调用创建进程，创建后父进程得到子进程的标识符，子进程得到值 0。

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
  int pid;
  printf("Now this is process 1.\n");
  printf("System calling fork () will start.\n");
  pid = fork( );
  if (pid == 0)                /*子进程调用返回值为 0*/
```

```
        printf("This is child process.\n");
      else if (pid>0)              /*父进程调用返回子进程的进程标识符*/
        printf("This is process 1 (parent process).\n");
      else
        printf("fork failed.\n");
      printf("The program end.\n");
      exit(0);
    }
```

pid 为 0 时的分支是子进程运行部分。

pid 大于 0 的部分是原进程（即父进程）运行部分。

通常情况是：

```
    if (pid = = 0)
       call subprocess_program;        /*调用子进程处理程序*/
    else if (pid>0)
      complete parent_process_program;     /*完成父进程处理程序*/
```

这样，父进程和子进程并发执行。由于父进程、子进程有相同的优先级，所以父、子进程运行的顺序是随机的。

例　利用 fork 创建 daemon（守护进程）。

```
#include <stdio.h>
#include <fcntl.h>
int main()
{
  int pid;
  pid=fork();     /*创建子进程*/
  if (pid>0)     /* 正常结束父进程 */
    exit(0);
  else if (pid<0)
  {
    perror("fork error");
    exit(-1);
  }
  for (;;)
  {
    printf("hello......\n");
    sleep(3);
```

```
    }
}
```

9.3.2　exec 系统调用

系统调用 exec 可以使一个进程调用或引用一个新程序。事实上，引用的程序副本覆盖了进程的存储空间。在 exec 调用之前存在的用户级上下文的内容被引用的程序覆盖而不存在。所以原进程（主进程）使用该系统调用后不能继续，继续进行的是新的程序内容。如图 9.3 所示。

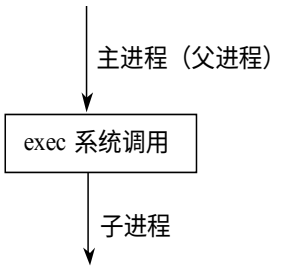

图 9.3　exec 系统调用图

该系统调用的格式为：

```
int execve(filename,argv,envp);
char * filename;
char * argv[];
char * envp[];
```

其中：filename 是要引用的可执行程序文件名；argv 是一个指向所执行程序的字符串参数数组指针；envp 是指向所执行程序环境的字符串数组指针。

系统调用 exec 要完成的操作如下。

(1) 调用 namei 过程获得可执行文件的索引节点并检查是否可执行及用户是否具有执行权限。如果可以执行，则将新 exec 参数拷贝到一个临时缓冲区，释放原参数占用的空间。

(2) 断开与原进程链接的各个区并回收其占用的各个空间。

(3) 根据可执行文件头中的信息为其分配新区并链接到进程上，如果内存空间允许则装入内存。

(4) 将 exec 参数从临时缓冲区拷贝到新的用户区并设置用户态寄存器上下文等。

虽然 exec 系统调用后原进程结构被覆盖，但系统环境变量及相应值列表会被继承下来，表 9．1 列出了这些被继承的属性及其系统调用。

表 9.1 exec 系统调用后被继承的属性极其系统调用

被继承的属性	相关系统调用
进程标识符（PID）	getpid()
父进程标识符（PPID）	getppid()
进程组标识符（PGID）	getpgid()
会晤期成员关系（SID）	getsid()
实际用户标识符（UID）	getuid()
实际用户组标识符（GID）	getgid()
增补的用户组标识符（GROUPS）	getgroups()
报警信号剩余时间	alarm()
当前工作目录	getcwd()
根目录	umask()
进程信号掩饰	sigprocmask()
未决的信号	sigpending()
已用时间	times()

基于系统调用 execve 的调用方式有 execl、execle、execlp、execv、execve、execvp。通常把 execve 作为最基本的系统调用，而其他 5 种调用是库函数调用，这些库函数调用最终都基于 execve 系统调用。exec 系统调用在使用时后面加上了字符 l、v、e、p，它们代表的意义如下。

l：表示用长格式，要求参数指针数组的每一个都作为独立的参数传递，最后以空指针 0 结束。execl(“/usr/bin/ps”,“ps”,“-ef”,“log”,0)；指定“ps –ef log”Shell 命令。

v：表示用 argv 指针数组传递运行参数。filename 是文件路径，argv[0]是可执行程序名，之后是运行参数。

e：表示用 enp 指针数组传递环境参数，如果要为进程指定新的环境，用 execve 或 execle。

p：表示执行时在环境变量 PATH 指定的目录中搜索路径名指定的文件，否则只在当前目录中搜索文件。如果 PATH 已经指定路径/bin,则为 execlp(“ps”,“ps”,“-ef”,“log”,0)。

例 在子进程中用 execl 系统调用，如果调用成功，则去执行 ps 命令而不会继续执行该子进程，即不会打印出“This is subprocess”。

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
```

```
int main()
{
  int pid;
  printf("Now this is process 1.\n");
  printf("System calling fork () will start.\n");
  if ((pid = fork()) == -1)
  {
    perror("fork failed");
    exit(1);
  }
  else if (pid == 0)    /*子进程系统调用返回值 pid 为 0*/
  {
    printf("This is child process.\n");
    if (execl("/bin/ps","ps","-ef",0) < 0)
    {
      perror("excel of ps failed");
    }
    printf("This is subprocess.\n");
    exit(0);
  }
  else if (pid > 0)    /*父进程系统调用返回值 pid 为 fork 的正实值*/
    printf("This is process 1 (parent process).\n");
  exit(0);
}
```

例　在子进程中用 execevp 系统调用来实现命令行参数传递的命令。

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>
int main(int argc, char *argv[])
{
  int pid, status;
  printf("Now this is main process.\n");
  printf("System calling fork() will start.\n");
```

```
    if ((pid = fork( )) == -1)
    {
      perror("fork call failed.\n");
      exit(1);
    }
    else if (pid == 0)    /*子进程系统调用返回值 pid 为 0*/
    {
      printf("This is child process.\n");
      if (execvp(argv[1], &argv[1]) < 0)
      {
        perror("execvp call failed.\n");
        exit(1);
      }
    }
    else while (pid != wait(&status))    /* 父进程等待子进程结束 */
    if ((pid == -1) && (errno != EINTR))
      break;
    exit(0);
  }
```

9.3.3 exit 系统调用

为了及时回收进程所占用的资源和减少父进程的影响，一般用户进程在完成任务后应尽快结束。在 UNIX 系统中提供系统调用 exit 用于进程的自我结束。每个结束后的进程进入僵死状态（SZOMB 或者 ZOMBIE），除保留进程表项 proc 结构外，要释放占有的资源（如关闭已经打开的文件），撤除进程上下文（释放内存）。该系统调用的格式为：

```
void exit(status);
int status;
```

其中 status 是返回给结束进程父进程的参数，为整数。

核心为 exit 系统调用完成的操作有如下几种。

(1) 不再接收任何软中断并关闭软中断处理函数。如果结束进程是与某一终端相连系的进程组组长，则必须发软中断信号通知本组进程“挂起”，并将这些进程交给进程 pid 为 1 的进程。

(2) 关闭所有已打开的文件，释放所有进程区及内存，释放当前目录并修改根

目录的索引节点。

(3) 将进程运行过程中产生的各种统计信息写到全局日志文件中。

(4) 将进程状态置为“僵死”，向父进程发送“子进程死”（SIGCLD）的软中断信号，核心切换进程上下文并调度另一个进程执行。

9.3.4　wait 系统调用

系统调用 wait 同步父进程与子进程的终止。调用进程挂起自己等待其子进程因暂停或终止发来软中断信号为止。如果在 wait 调用之前子进程已经暂停或终止，则调用进程做完处理工作后便返回而不用等待。该系统调用的格式为：

```
pid_t wait(status);
int * status;
```

其中 status 是子进程的退出状态码。核心对系统调用 wait 作如下操作。

(1) 查找调用进程是否有子进程，没有则返回出错码。

(2) 如果找到一个子进程处于“僵死”状态，将子进程的执行时间加到调用进程的执行时间上，释放子进程的进程表项。

(3) 如果有多个子进程处于“僵死”状态，则调用进程等待最先结束的那个子进程返回。

(4) 如果没有查到“僵死”状态的子进程，调用进程在可被中断的优先级上睡眠，等待其子进程发来软中断信号将其唤醒。

有的系统还提供系统调用 waitpid 通过 pid 指名要等待的子进程。

例　父进程等待子进程结束。子进程中的 execl 系统调用应该失败，会打印出 excel of ps failed.。然后父进程调用 wait，打印出 status 码。

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
  int pid;
  int status;
  printf("Now this is process 1.\n");
  printf("System calling fork () will start.\n");
  if ((pid = fork()) == -1)
  {
    perror("fork failed");
```

```
    exit(1);
  }
  else if (pid == 0)    /*子进程系统调用返回值 pid 为 0*/
  {
    printf("This is child process.\n");
    if (execl("/usr/bin/abcd","abcd","-ef","log",0)<0)
    {
      perror("excel of ps failed");
      exit(5);
    }
  }
  wait(&status);
  printf("status:%xH\n",status);
  printf("This is process 1 (parent process).\n");
  exit(0);
}
```

9.3.5 getpid、getppid、getuid、geteuid、getgid、getegid 系统调用

系统调用 getpid 用于获取调用进程的 pid。该系统调用的格式为：

```
pid_t getpid();
```

例 得到进程的 pid。

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
  /*得到调用进程的真正用户 ID*/
  printf("Owner user ID: %ld\n", (long)getuid());
  /*得到进程的 ID*/
  printf("Process ID: %ld\n", (long)getpid());
  /*得到父进程的 ID*/
  printf("Parent process ID: %ld\n", (long)getppid());
  exit(0);
```

```
}
```

系统调用 getuid 返回的是调用进程的真正用户 ID，系统调用 geteuid 返回的是调用进程的有效用户 ID；系统调用 getgid 返回的是调用进程的真正用户组 ID，系统调用 getegid 返回的是调用进程的有效用户组 ID。

9.3.6　brk 系统调用

系统调用 brk 用于改变进程数据区的大小。该系统调用的格式为：

```
int brk(endds);
void *endds;
```

其中 endds 为改变后进程数据区的最高虚地址值，也称为 break 值。

另一种调用方式为：

```
void* sbrk(increment);
ptrdiff_t increment;
```

其中：increment 为以字节为单位对当前 break 值的改变量；返回值是调用前的 break 值。实际上 sbrk 是一个调用 brk 的 C 语言库函数。

如果要新增加进程的数据空间，则新分配的数据空间虚拟接在老地址空间之后，进程地址空间得到连续扩展。

核心为该系统调用完成的操作如下。

(1) 检查新的进程大小是否小于系统的最大值，只有小于时系统调用才能继续，否则返回。

(2) 检查进程新的数据区与原来分配的虚地址空间是否重叠，有则返回。

(3) 为进程的数据区上锁。

(4) 调用算法 growreg 分配额外的内存给数据区并增加进程的大小字段。

(5) 清新数据区的地址空间。

(6) 对新数据区解锁进程。

例

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
int callno = 0;
char *cp, buf[255];
void catcher(int signo)
{
  callno++;
```

```
    printf("catch signal %d %dth call at address %u\n", signo,callno,cp);
    sbrk(256);
}
int main()
{
    void *oldendds;
    signal(SIGSEGV,catcher);
    oldendds = sbrk(0);
    printf("The firat sbrk call completed. Original brk value %u\n", oldendds);
    cp = buf;
    for (;;)
      *cp++ = 1;
}
```

程序在 AT&T3B20 计算机上运行时的输出：

```
original brk value 140924
caught sig 11 1th call at addr 141312
caught sig 11 2th call at addr 141312
caught sig 11 3th call at addr 143360
caught sig 11 4th call at addr 143360
......
caught sig 11 11th call at addr 143360
caught sig 11 12th call at addr 145408
......
caught sig 11 19th call at addr 145408
```

在系统调用 signal 安排捕获“段违例”（Segmentation Violation）软中断信号后，该进程调用 sbrk 并打印出它初始的 break 值。然后进入循环，每次循环使字符指针加 1 并写其内容。直到它试图向一个超出其数据区的地址写内容，从而引起一个段违例软中断信号。捕获到该信号后，函数 catcher 调用 sbrk 为数据区再分配 256 个字节。进程从循环中的断点继续执行，写入新的数据空间。当循环中再次超出数据区时，整个过程又重复一遍。

9.3.7 nice 系统调用

系统调用 nice 用于控制进程调度的优先权。该系统调用的格式为：

```
int nice(value);
```

```
int value;
```

value 值是加入到进程优先权计算公式中的一个值，进程优先权数计算公式为：

$$\text{进程优先权数} = \frac{\text{最近使用CPU时间}}{\text{常数}} + \text{基本用户优先数} + \text{nice value}$$

进程的优先权数越大，进程的优先级越低。需要注意的是只有超级用户才能提高进程优先权的 nice 值，并且只有超级用户才能提供低于指定阀值的 nice。系统调用 nice 只作用于正在运行的进程，一个进程不能重新设置其 nice 值，进程运行效果不理想时不能再修改 nice 值，只能用 kill 终止进程。

9.3.8　stime、time、times、alarm 系统调用

stime、time、times、alarm 都是与时间有关的系统调用。其中 stime、time 用于全局的系统时间，times、alarm 用于单个进程的时间。

在 UNIX 系统中，利用定时器处理操作系统中的进程调度、网络协议包发送与接收超时以及系统统计数据定时更新等事务。系统时间是以秒为单位保存的，起始时间定义为格林威治时间 1970 年 1 月 1 日零点整。

stime、time、times、alarm 系统调用定义包含在文件 time.h 中。

stime 是超级用户将一个表示当前时间的值赋予一个内核全局变量。调用格式为：

```
int stime(pvalue);
long pvalue;
```

pvalue 是一个长整数指针，表示时间。当 long 类型为 32 位时，从 1970 年 1 月 1 日零点整开始要溢出则在 2038 年后；当为 unsigned long 时，要溢出则在 2106 年后。

由 stime 设置内核时间后，系统调用 time 用于查询该时间变量。系统调用 time 的格式为：

```
time_t time(tloc);
time_t *tolc;
```

调用成功后，tloc 为指向从 1970 年 1 月 1 日零点整开始至今的秒数的指针；如果系统调用不成功，则返回-1 并设定相应的错误号。

系统调用 times 用于调用进程在执行时的累计时间加上其所有“僵死”子进程曾执行所花费的累计时间，也就是调用进程及其“僵死”子进程执行所花费的所有累计时间。该系统调用的格式为：


```
clock_t times(buffer);
struct tms * buffer;
```

```
struct tms
{
  time_t  tms_utime;   /*进程的用户时间*/
  time_t  tms_stime;   /*进程的内核时间*/
  time_t  tms_cutime;   /*子进程的用户时间*/
  time_t  tms_cstime;   /*子进程的内核时间*/
}
```

times 调用成功则返回从过去的任意时刻开始所经过的时间。如果系统调用不成功，则返回-1 并设定相应的错误号。

例 计算处理器上正处于运行状态的进程产生的时间碎片。

```
#include <stdio.h>
#include <sys/times.h>
int main()
{
  struct tms p_start;
  struct tms p_end;
  long pt1;
  long pt2;
  long pt3;
  if ((pt1 = times(&p_start)) == -1)
    perror("Can't get start time");
  else
  {
    if ((pt2 = times(&p_end)) == -1)
      perror("Can't get end time");
    else
    {

pt3=p_end.tms_utime+p_end.tms_cstime-p_start.tms_utime-p_start.tms_cstime;
      printf("Process running's fraction time = %f\n",
(double)(pt3)/(pt2-pt1));
    }
  }
  exit(0);
```

```
}
```

系统调用 alarm 设置闹钟软中断信号。系统调用的格式为：

```
unsigned int alarm(seconds);
unsigned seconds;
```

seconds 为设置的闹钟时间，表示在 seconds 秒后安排一个 SIGALRM 闹钟软中断信号。该系统调用返回值是从该调用开始到软中断信号出现所剩余的时间量。

例　程序每隔半分钟检查一次文件的存取时间，如果文件被修改过就打印出一条消息。

```
#include <stdio.h>
#include <signal.h>
#include <sys/stat.h>
#include <sys/types.h>
void wakeup(int signo)
{}
int main(int argc, char *argv[])
{
  struct stat statbuf;
  time_t fltime;    /*时间*/
  if (argc != 2)
  {
    printf("Usage: %s filename\n", argv[0]);
    exit(1);
  }
  fltime = (time_t)0;      /*time_t 见文件系统调用中的 stat*/
  for(;;)
  {
    if (stat(argv[1],&statbuf) == -1)    /*查询文件状态*/
    {
      perror("stat error");
      exit(1);
    }
    if (fltime != statbuf.st_atime)      /*不为前面 stat 得到的文件最后一次修
改时间*/
    {
```

```
        if (fltime != 0)
          printf ("file %s acceeded\n",argv[1]);
        fltime = statbuf.st_atime;
      }
      signal(SIGALRM,wakeup);   /*置闹钟*/
      alarm(1);
      pause();          /*睡眠等待软中断*/
    }
  }
```

9.4 系统调用实例

例 1 利用管道，实现命令“ps -ea | grep init”的效果。

```
#include <stdio.h>
#include <unistd.h>
int main()
{
  char buf[255];
  int fd[2], n;
  pipe(fd);  //创建管道
  if (fork() > 0)
  {
    close(fd[1]);
    dup2(fd[0], 0);    //将标准输入复位向到读管道
    close(fd[0]);
    execlp("/bin/grep", "grep", "init", NULL);
  }
  else
  {
    close(fd[0]);
    dup2(fd[1], 1);    //将标准输出复位向到写管道
    close(fd[1]);
    execlp("/bin/ps", "ps", "-ea", NULL);
  }
```

```
    exit(0);
}
```

例 2　简单定时器的实现。

```
#include <signal.h>
#include <unistd.h>
#include <sys/time.h>
void prompt_info(int signo)
{
  //定时器触发后需要执行的代码
  printf("时间已经过去了两秒钟\n");
}
int main()
{
  struct sigaction act;
  struct itimerval value;
  //设置信号处理函数
  act.sa_handler = prompt_info;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGPROF, &act, NULL);
  //定时器每 2 秒触发一次
  value.it_value.tv_sec = 2;
  value.it_value.tv_usec = 0;
  value.it_interval = value.it_value;
  setitimer(ITIMER_PROF, &value, NULL);
  while(1);
  exit(0);
}
```

例 3　UNIX 系统中的邮件守护进程通知用户未处理邮件。系统通常会把邮件存储在目录/var/mail 下，每个用户在该目录下有一个目录存放各自的邮件。该目录中与用户登录名相同的文件包含所有该用户未读的邮件。如果用户有未读邮件则该邮件文件 open 会成功，蜂鸣通知用户，然后程序 close 邮件文件。


```
#include <stdio.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>
#define MAILFILE "/var/mail/xiexie"
#define SLEEPTIME 1
int main(void)
{
  int mailfd;
  for (;;)
  {
    if ((mailfd = open(MAILFILE,O_RDONLY)) != -1)
    {
      fprintf(stderr, "\007");      /*蜂鸣通知*/
      close(mailfd);
    }
    sleep(SLEEPTIME);
  }
}
```

例 4　从磁盘读入文件内容，显示并发送到消息队列，从消息队列读入内容并写入文件。

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
//宏定义
#define MIN(x,y)(x)<(y)?(x):(y)            // 返回最小值
#define RANDOM 12                    // 任意的非零数，用于 ftok 函数
#define DATA_MSG 111                 // 定义数据消息类型为 111
#define END_MSG 999                  // 定义数据消息类型为 999
#define CONTENT_LENGTH 16            // 定义每一条消息的内容长度为 16
#define MSG_DIR "/home/stu01"              // 定义消息队列目录
```

```
#define INPUT_FILE "/home/stu01/in.txt"   // 读入的磁盘文件名
#define INPUT_FILE "/home/stu01/out.txt"  // 写入的磁盘文件名
// 定义消息结构
typedef struct
{
  long mtype;
  char mtext[CONTENT_LENGTH];          //每条消息包含的内容长度可任意自定
}Message;
//创建连接到消息队列的函数
int connectMSG()
{
  int msgid,oflag;
  key_t key;
  char pathname[]=MSG_DIR;             //消息队列目录
  oflag=S_IRWXU|S_IRWXG|S_IRWXO|IPC_CREAT;
  key=ftok(pathname,RANDOM);
  msgid=megget(key,oflag);
  if (msgid>=0)
  {
    return(msgid);                     //创建成功返回消息队列号
  }
  else
  {
    return(-1);                              //创建失败返回-1
  }
}
//用于发送一条消息的函数
int send(int msgid.Message *buf)
{
  int flag,length;
  length=CONNECT_LENGTH*sizeof(char);
  flag=IPC_NOWAIT;                   //非阻塞调用

  if (msgsnd(msgid,buf.length,flag)!=0)          //消息发送，失败给出提示信息
  {
```

```
    printf("Message send error!\n");
    return(-1);
  }
  else
  {
    return(0);
  }
}
//用于接收一条消息的函数
int receive(int msgid,long msg_type,char *buffer)
{
  int n,flag,length;
  Message buf;
  length=CONTENT_LENGTH *sizeof(char);
  flag=IPC_NOWAIT;                    //非阻塞调用
  n=msgrcv(msgid,&buf,length,msg_type,flag);   //消息接收
  if (n>0)                  //成功将消息内容复制到缓冲区，并返回收到的 char 数
  {
    strcpy(buffer,buf,mtext);
    return(n);
  }
  else
  {
    return(-1);                                  //失败返回-1
  }
}
main(int argc, char * argv)
{
  int i,m,n
  //消息队列用参数
  int msgid;
  int oflag,length,flag;
  long msg_type;
  //消息发送缓冲区
```

```
Message buf;
//消息数据接收缓冲区
char buffer[CONTENT_LENGTH];
key_t key;
//文件操作用参数
FILE *fp;
char ch;
int number;
//程序结束标识
int end_flag;
//创建和连接到消息队列
msgid=connectMSG();
if (msgid==(-1))
  exit(0);                          //出错，结束程序
//打开磁盘文件
if ((fp=fopen(INPUT_FILE,"r"))==NULL)
{
  printf("open input file error!\n");
  exit(0);
}
//定义消息类型为数据类型
buf.mtype=DATA_MSG;
number=0;
ch=fgetc(fp);
while(ch!=EOF)
{
  putchar(ch);                      //屏幕输出
  buf.mtext[number]=ch;             //写入消息队列
  if(number==(CONTEXT_LENGTH-1))  //消息内容缓冲区满，可以发送一条消息了
  {
    send(msgid,&buf);
    memset(buf.mtext,0,CONTEXT_LENGTH);       //消息内容清零
    NUMBER=0;
  }
```

```
  else
  {
    number++;
  }
  ch=fgetc(fp);                          //继续文件输入
}
if (number!=0)
{
  send(msgid,&buf);                  //发送最后一条数据消息
}
buf.mtype=END_MSG;              //定义消息类型为结束字符
memset(buf.mtext,0,CONTEXT_LENGTH);     //清零
send(msgid,&buf);
printf("\Nread file and send message finished!\n");
fflush(NULL);                          //强制缓冲区输出
system(("ipcs- q");                        //系统调用，显示当前消息队列状态
fclose(fp);                            //关闭读入的文件
 //发送消息结束，下面开始接收消息并写入文件
//连接到消息队列
msgid=connectMSG();
if (msgid=(-1))exit(0);                    //出错结束程序
//打开磁盘文件
if((fp=fopen(OUTPUT_FILE,"w+"))==NULL)
{
  printf("open output file error!\n");
  exit(0);
}
//循环，直到收到结束消息
end_flag=1;
while(end_flag)
{
  msg_type=DATA_MSG;                   //数据消息类型
  n=receive(msgid,msg_type,buffer);
  if(n!=(-1))                          //成功收到一条数据消息
```

```
        {
          //取 n 和 strlen(buffer)的最小值，保证最后一条消息不会多收数据
          n=MIN(n,strlen(buffer));
          for (i=1;i<n;i++)
          {
            putchar(buffer[i]);                    //输出到屏幕
           fputc(buffer[I].fp);                    //输出到文件
          }
        }
        else                                       //接收数据消息出错，检查是否为正常结束
        {
          msg_type=END_MSG;              //结束消息类型
          n=receive(msgid,msg_type,buffer);
          if (n!=(-1))                             //收到结束消息
          {
            end_flag=0;                            //结束程序标志置位
            printf("n\Receive message and write file finished!");
            fflush(NULL);
            system("ipcs -q");                     //系统调用，显示当前的消息队列状态
            sprintf(buffer,"ipcrm msg %d",msgid);
            system(buffer);                        //系统调用，删除消息队列
          }
        }
      }
      fclosed(fd);                                 //关闭输出文件
    }
```

9.5　本 章 小 结

系统调用是一组用于实现各种系统功能的子程序。用户可以通过系统调用函数在自己的应用程序中使用它们。从某种角度来看，系统调用和普通的函数调用非常相似，区别仅仅在于，系统调用由操作系统核心提供，运行于核心态；而普通的函

数调用由函数库或用户自己提供。

事实上 UNIX 操作系统提供的系统调用远远超过在此介绍的，本章只是介绍一些常用的系统调用，起到抛砖引玉的作用，更多的知识需要读者自己去发掘。第一章学习的 man 命令就是一个获取帮助的好途径。

上 机 练 习

1．上机验证本章的演示程序，最好能根据自己的想法加以发挥。

2．利用程序试验其他途径获知的系统调用函数，通过 man 命令获取该函数相关的知识。

习　题　九

1．设计一个系统调用，该系统调用将一个已存在的文件截为任意大小（提供一个参数），并说明实现方法。

2．把系统响应时间定义为它完成一个系统调用所占用的平均时间。把系统吞吐量定义为在一个给定时间段内系统所能执行的进程数目。请描述磁盘高速缓冲为何有助于缩短响应时间？

第 10 章　UNIX 窗口系统

为了用户使用方便、接口更加友好，UNIX 操作系统同其他操作系统一样，配置了图形用户接口 GUI。在 UNIX 的图形接口中，应用程序以窗口方式操作，所以图形接口又称为窗口系统。

本章主要内容

- X 窗口系统：主要介绍了 UNIX 的 X 窗口系统及层次：Xlib、Toolkits 及 Openlook 和 Motif。
- 通用桌面系统：主要介绍通用桌面系统 CDE 的功能及使用。
- 用 X-Window 开发程序简介。
- 用 Motif 开发程序实例。

10.1　X 窗口系统

图形用户接口 GUI 的引入改变了计算机的使用，用户只需选择和操作图形窗口就可以方便地完成所需工作，使得计算机的操作脱离了单纯命令输入方式。

UNIX 操作系统同其他操作系统一样，除了提供命令之外，也提供了窗口系统。在 UNIX 系统中最早和最有影响的窗口系统是 X 窗口系统。

10.1.1　X 窗口系统概述

X 窗口系统（X-Window）是由麻省理工学院（MIT）、IBM、AT&T、HP 和 Sun 等多家机构和公司（又称为 X 集团）联合推出的用于计算机和 UNIX 用户之间进行交流的窗口系统，简称为 X，1987 年推出了有名的 X 窗口系统第 11 版本，称为 X11。

X 集团为了建立不依赖于特定硬件的图形和文字显示窗口系统，提出了窗口标准，即 X 协议。X 协议规定了不同厂商之间提供的 X 窗口接口标准，操作系统开发者可以根据各自的需要和喜好在遵循 X 协议的基础上开发自己的“窗口系统”。

X 窗口系统的结构基于网络方式，建立在网络环境中，是由服务器（x-server）软件、客户机（x-client）软件和连接在服务器与客户机之间的通信链路所构成的图形接口。客户端可以建立一个或多个与服务器应答方式进行的通信。服务器

（x-server）软件主要接收客户机（x-client）发出的请求并控制显示硬件、完成在屏幕上打开窗口以及控制窗口尺寸及位置、将事件标志回送给客户机。客户机主要提供对 X 的支持和图形用户接口程序。

一台 UNIX 主机可以通过网络连接多台 X 终端。X 终端与 UNIX 系统配套并遵循 X 协议。在 X 终端上配置有键盘、显示屏幕、鼠标和网络。该终端的只读存储器中固化有 X 服务软件。除 X 终端外，PC 的 Windows 系统中也可以运行 X 服务软件，即 PC 机同样也可以称为 X 终端。现在，许多 UNIX 系统，如 AIX、Solaris，都有与 Windows 系统配套的 X 服务器（x-server）软件。

10.1.2　X 窗口系统层次

X 窗口系统包含三个层次，如图 10.1 所示。

图 10.1　X 窗口系统层次

基于 X 的应用软件通过调用 X 的一系列 C 语言函数实现各种功能，这些函数被称为 Xlib。Xlib 是一种底层库，用来编写图形和交互接口程序非常灵活，提供了建立窗口、画图、处理用户操作事件等基本功能。但是由于建立在底层，Xlib 使用较复杂、繁琐。为此推出了建立在 Xlib 上层的库函数软件工具包 Toolkits。

Toolkits 软件工具包将常用的图形接口，如窗口、菜单、按键等，称为 Toolkits 的组件（Widget），这些组件按面向对象编程方式组织到一起供应用程序使用。Toolkits 软件工具包中的内置组件（Intrinsics）允许在其上建立新的组件。MIT 和 Digital 推出的 Xt Toolkit 就是在 Toolkits 中最常见的一种，简称 Xt。Xt 不是 X11 标准中的一部分，但通常还是与 X11 一起提供。所以，除实际上的一些基本绘图功能外，各种图形交互接口基本都能借助各种 Toolkit 完成。

在 X 窗口的 Xt Toolkit 上，Sun 公司和 AT&T 公司合作开发了 OpenLook Tool 软件包，该软件包开发窗口系统简单快捷。在相当长一段时间内用 OpenLook Tool 开发非常流行，其中 Sun 公司用 OpenLook Tool 开发了有名的 OpenWindows 系统。

继 Sun 的 OpenLook Tool 之后，开放软件基金会（Open Software Foundation，

即 OSF）也在 Xt Toolkit 上建立了 Motif Toolkit。Motif Toolkit 是一套内容丰富、风格统一、视觉效果好的交互图形组件库。由于 Toolkit 在使用上比 OpenLook Tool 更加简单，Motif 一出现立即引起业界的注意并成为与 OpenLook Tool 抗衡的力量。Motif Toolkit 不仅是一套交互图形组件库，还是一个窗口管理器（Motif Window Manager，即 MWM）。窗口管理器 MWM 提供图形人机交互接口的风格标准，称为 Motif 风格。现在 Motif 已经用在许多工作站上，成为工作站上图形交互接口的一种实际上的工业标准。

10.2 通用桌面环境 CDE

虽然基于 X-Window 系统用 Openlook 或 Motif 可以建立各厂家的窗口系统，但是所建的窗口系统太多会给用户的使用带来麻烦。为此，IBM、HP、SunSoft、Digital 和 Fujitsu 等公司联合研究通用开放式软件环境，并最终产生了通用桌面环境（Common Desktop Environment，即 CDE）。

10.2.1 CDE 简述

CDE 是 UNIX 的标准桌面系统，不仅为所有支持平台提供了基本的应用程序和通用接口，还为系统管理员和应用程序员提供了通用的服务。支持 CDE 的系统有：Sun Solaris、HP/UX、IBM AIX 和 Digital UNIX。在这些系统中 CDE 是默认安装。另外，Linux 和 Free BSD 也支持 CDE。

10.2.2 CDE 桌面

- 进入桌面

使用 CDE 首先必须在登录窗口中用用户名和口令登录，进入桌面。

- 桌面组件

进入 CDE 桌面后，桌面提供的组件有：窗口、前面板、式样管理程序、文件管理程序、应用管理程序、工作空间对象。

窗口：是带有控制的方框，包含软件应用程序，可以移动、缩放或放到其他空间。

前面板：位于显示器下部的窗口。提供日常使用的控制、指示器、子面板和选择工作空间开关。

式样管理程序：简单制订桌面的颜色、工作空间幕布、字体大小以及键盘、鼠标、窗口特性。

文件管理程序：以图形方式显示系统中的文件、活页夹、应用程序。用文件管

理程序图表可以直接打开文件进行编辑，避免使用 vi 编辑器。

应用管理程序：通过动作图标提供对访问频率高的应用程序的访问。可以启动应用程序。

工作空间：屏幕上的一个区域，可以将工作窗口放入其中并安排，在使用完后收起来。

- 离开桌面

当离开桌面时即从桌面注销，CDE 将保存当前会话过程，重新注册到 CDE 桌面时会显示原先退出的工作。注销桌面可以从前面板中选择“ EXIT ”退出，也可以在工作空间中选择注销。在工作空间中注销的方法是首先将鼠标移到工作空间幕布上，再单击鼠标右键，出现工作空间菜单，选择注销。

- 锁屏

如果不想注销而要离开时，为了防止系统被非法使用，可以选择锁屏。这时只需按前面板中的锁屏控制即可。

10.3　用 X-Window 开发程序简介

这部分简单介绍用 X-Window 开发程序的基本知识。开发程序主要步骤如下。

- 在程序开始中加入隐含头文件

```
include<X11/Xlib.h>     /* 常在/usr/lib/X11 下 */
include<X11/Xuil.h>
include<X11/Xos.h>
```

- 在程序中实现到服务器的连接

```
Display XopenDisplay(char *name);
```

该语句为显示服务器的名字。

其中 Display 是一个结构，name 如果为 Null，则表示默认号为 0 的 server，0 号屏幕。

Screen=DefaultScreen(display);系统任屏幕返回为 0，否则为：

Screen=DefaultDepth(display，screen);

- 获取根窗口信息

```
XgetGeometry();
```

该语句不仅可以得到根窗口，还可以得到屏幕中任何窗口的几何信息。

- 建立窗口

```
Window XcreateWindow(para1,para2,…,para13);
```


para1 为 display 在那一个 display 上建立窗口。

para2 为 display 窗口的父窗口。

其余为窗口的 width、height 等。

- 位图和图标

```
#define icon_width 1b;
#define icon_height 1b;
Static unsigned char icon_bits[]={0x3f,0x3f,…,0x00};
```

- 颜色设计

colormap 生成颜色查找表。

- 窗口管理系统通信

```
void XSetWMProperties(……);
```

主要功能是向窗口管理系统发送消息，使窗口管理器正常管理窗口，窗口管理器可以在窗口外加一个边框。

- 选择事件类型

X-Window 下只有选择了事件才会开始接收事件，与 Window 下开发程序不同。

```
XselectInput(Display,Window,EventMask);
EventMask=ExposureMask;
```

- 建立服务器资源，如字体等

```
GC:XCreate GC();
```

建立字体：XloadQueryFont();

- 窗口映像可见

```
XmapWindow();
```

使刚建立的窗口显示在屏幕上。

- 设置事件循环

```
while(1)
{
  XnextEvent(display,&report);
  /*表示有消息时向下执行*/
  Switch (report,type)
  {
    case Expose;
    case Button Press;
  }
};
```


该循环是一个死循环，一直不停地执行语句检测。

以上只是 X-Window 开发程序的基础，介绍的目的是要大家了解用 X-Window 开发程序是一个很底层的工作，非常复杂、繁琐。现在这种方式用得很少。

10.4　用 Motif 开发程序实例

用 Motif 风格在用户窗口编程中帮助用户实现接口设计，提供一致又易用的用户接口管理，使得用户在程序设计中容易控制应用程序的执行。

在 Motif 风格指南中对输入的键盘和鼠标的导向模式、选择与激活应用程序的各部件模式、用户接口部件的选择和布局，以及交互窗口管理程序设计模式都进行了定义。因此，用 Motif 编程对用户来说非常标准。

下面给出用 Motif 设计程序的一个例子。

- 主程序名为 welcomemotif.c
- 用户接口定义为 uimotif.uil

```
procedure                          /*过程段*/
  welcome_button_activate();
object                                    /*对象段：主窗口*/
{
  S_MAIN_WINDOW：XmMAINWindow{
  arguments{};
  controls{                         /*主窗口由对象和菜单组成*/
    XmMenuBar          s_menu_bar;
    XmBulletinBoard    welcome_main;
  };
};
object                                    /*对象：公告版*/
{
  welcome_main：XmMulletinBoard{
    controls{
      XmLabel        welcome_label;
      XmPushButton   welcome_button;
    };
};
object                                    /*对象：公告版中之按钮*/
{
```

```
        welcome_button：XmPushButton{
          arguments
          XmNx = 35;
          XmNy = 70;
          XmNlabelString=compound_string( 'Hi!' ,separate=true)& 'Friends' ;
        };
        callacks{
          XmNactivateCallback = procedure welcome_button_activate();
        };
      };
      /*      公告版中之标签      */
      object
      welcome_label：XmLabel{
      argements{
      XmNlabelString     =     compound_string('Press     button     once     to
goodluck!\n' ,separate=true)
      &compound_string('Press button two to goodbyte!\n' ,separate = true)&
            'Press button three to exit!\n' ;
      /*         菜单条        */
      object {
        S_menu_bar：XmMenuBar{
          arguments{XmNorientation = XmHORIZONTAL;
          XmNspacing = 10;
        };
        controls{        /*  菜单条包括 4 个下拉式菜单   */
        XmCascadeButton    file_menu_entry;
        XmCascadeButton    edit_menu_entry;
        XmCascadeButton    print_menu_entry;
        XmCascadeButton    help_menu_entry;
        };
      };
```



```
      object file_menu_entry：XmCascadeButton{
        argements{};
```

```
    controls{};
    callbacks{};
  };
  object edit_menu_entry：XmCascadeButton{
    argements{};
    controls{};
    callbacks{};
  };
  object print_menu_entry：XmCascadeButton{
    argements{};
    controls{};
    callbacks{};
  };
  object help_menu_entry：XmCascadeButton{
    argements{};
    controls{};
    callbacks{};
  };
  end module;
```

- mainmotif.c

```
#ifdef REV_INFO
#ifdef lint
static char SCCSID[] ="OSF/Motif:@(#) welcomemotif.c"
#endif   /* lint */
#endif   /* REV_INFO */

#include<stdio.h>
#include<Mrm/MrmAool.h>
static MrmHierarchy   s_MrmHierarchy;
static MrmCode    class;
static void welcome_button_activate();
static MrmCount       regnum = 1;
static char *vec[]={"welcomemotif.uid" };
static MrmRegisterArg regvec[] ={
```

```
  {"welcome_button_activity" ,(caddr_t)welcome_button_activate}
};

/*       主程序        */
int main(argc,argv)
int argc;
char * *argv;
{
  Widget toplevel,welcomemain = NULL;
  Arg arglist[1];
  MrmInitialize();
  /*     Xt 工具箱初始化        */
  toplevel = XtInitialize(
  "Hi" , "welcomeclass" ,NULL,0, &argc,argv);

  XtSetArg(arglist[0],XtNallowShellResize,TRUE);
  XtSetValues(toplevel,arglist,1);
  /*   定义 Mrm.hierarchy(仅一个文件)   */
  if (MrmOpenHierarchy(1,vec,NULL,&s_MrmHierarchy)!=MrmSUCCESS){
    printf("can't open hierarchy\n" );
  }
  if (MrmRegisterNames(regvec,regnum)!=MrmSUCCESS)
    printf("can't register names \n" );
  if (MrmFetchWidget(s_MrmHierarchy,"welcome_main" ,toplevel,
    &welcomemain,&class)!MrmSUCCESS)
    printf("can't get interface\n" );
  XtManagechild(welcomemain);
  XtRealizeWidget(toplevel);
  XtMainLoop();
  Return(0);
}
/*              回调函数              */

static void welcome_button_activate(widget,tag,callback_data)
  Widget  widget;
```

```
    char *tag;
    XmAnyCallbackStruct *callback_data;
    {
      Arg arglist[2];
      static int call_count=0;

      call_count+=1;
      switch (call_count)
      {
      case 1;
        XtSetArg(arglist[0],XmNlabelString,
        XmStringLtoRCreate("Good Luck!" , " "));
        XtSetArg(arglist[1],XmNx,11);
        XtSetValue(widget,arglist,2);
        break;
      case 2;
        XtSetArg(arglist[0],XmNlabelString,
        XmStringLtoRCreate("Goodbyte!" , " "));
        XtSetArg(arglist[1],XmNx,11);
        XtSetValue(widget,arglist,2);
        break;
      case 3;
        exit(1);
        break;
      }
  }
```

程序执行的结果是：

创建一个窗口，在该窗口中显示内容：

“Press button once to good luck!

Press button two to goodbyte!

Press button three to exit.”。

在窗口下部的按钮上显示“Hi!Friends!”。

如果用户单击按钮一次，则显示 Good Luck！

如果用户单击按钮两次，则显示 Goodbyte！

如果用户单击按钮三次，则显示 exit！

10.5 本 章 小 结

由于字符接口的 UNIX 命令很难记忆又不友好。所以现在大多数的 UNIX 系统都提供了图形化的接口——X 窗口系统，很多操作在图形接口的帮助下变得异常快捷。

上 机 练 习

1. 在一个 X 窗口系统中尽情发挥，为所欲为。

习 题 十

1. 什么是 X-Window？UNIX 系统使用 X-Window 解决了什么问题？
2. X-Window 作为图形用户界面具备哪些应用特性？
3. 试描述 X-Window 的工作方式，即包括的主要功能模块有哪些？

第 11 章　UNIX 网络及其管理

UNIX 操作系统以其出色的网络功能在网络应用领域起着举足轻重的作用。最早 TCP/IP 协议的实施就是针对加州大学 Berkeley 分校的 BSD UNIX 系统，为网络提供服务的主机大部分都由 UNIX 系统承担。所以网络配置和管理是 UNIX 操作系统学习的一个重要内容。

本章主要内容

- 网络基础：简述网络和 UNIX 中的网络协议。
- TCP/IP 网络：主要包括 TCP/IP 提供的网络服务、配置 TCP/IP、TCP/IP 接口管理。
- 路由管理：主要包括路由、路由选择表、内核路由选择表、静态路由和动态路由、操作内核路由选择表。
- 执行路由选择协议：主要包括 IP 转发、RIP 简介、OSPF 简介、RDISC 简介、BGP 简介、UNIX 路由选择协议实现、驻守进程 gated 和 routed 的配置和管理。
- UNIX 系统中的 PPP：主要介绍 PPP 协议及其用途。
- 网络服务：域名服务、Web 服务、邮件服务、FTP 服务。
- NIS：主要介绍 NIS 概念、使用 NIS。

11.1　UNIX　网　络

计算机网络将物理位置分散的计算机连接在一起，实现共享资源和传送信息的目的。在网络体系结构上，较早的有 ISO 定义的 OSI（开放系统互连）模型，该模型由 7 层构成，分别是：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。按照 7 层模式构造网络非常复杂和繁琐。为了简化工作，人们将网络分为通信子网和资源子网。通信子网主要由网络通信设备连接而成；资源子网负责提供网络资源。所以计算机网络在结构上不一定要包含全部的 7 层。而是根据具体应用要求选择一定的层次构成，即只要有通信子网和资源子网就行。

在计算机网络上，存在许多遵循不同协议标准的网络，如以太网、令牌网、令牌环网、x.25 网、ISDN、ATM、SNA 等。为了实现不同网络协议的网络互连互通，在

美国国防部的资助下，推出了 TCP/IP（传输控制协议／网际协议），并对该协议进一步标准化，当时在美国各大学流行 UNIX 操作系统正好满足该协议的实现环境。因此 TCP/IP 协议就融入到了 BSD 版的 UNIX，并且出乎意料地获得了极大成功。

从大体上看 TCP/IP 协议主要包含 IP 层和 TCP 层，没有 OSI 的最下面两层，最下面两层借助于局域网协议。从驻留 IP 的网络层开始，之上是包含 TCP 的传输层。TCP/IP 的实现基于服务器／客户模式。启动通信的设备为客户，应答的设备为服务器，服务器响应客户的请求。

UNIX 系统除了支持 TCP/IP 之外，几乎还支持其他所有的 CCITT 和 IEEE 的协议。但用的最多的还是 TCP/IP。

11.2 TCP/IP

用于提供 TCP/CP 服务及管理是 UNIX 系统的主要应用。学习 UNIX 系统，必须掌握在 UNIX 系统中 TCP/IP 的实现和配置。

11.2.1 TCP/IP

Internet 用 TCP/IP 协议将大学、科研机构、政府、组织、商业等部门数以万计的计算机连接在一起共享资源和交换信息。TCP/IP 提供的信息服务主要有：

- Web 服务（信息浏览）；
- DNS 服务（域名服务）；
- Telnet 服务（远程登录）；
- FTP 服务（远程文件传输）；
- SMTP 服务（简单邮件传输协议）；
- SNMP（简单网络管理协议）；
- NFS（网络文件服务器，由 Sun 公司开发）；
- RPC（远程过程调用）；
- ICMP（Internet 控制信息协议）。

11.2.2 配置 TCP/IP

在 UNIX 中配置 TCP/IP 之前必须了解网络配置信息。在 TCP/IP 上的每台机器都有一个 IP 地址和自己的名字（主机名）。主机名用于帮助用户更容易地使用机器。在 UNIX 系统中主机名包含在 hosts 文件中。系统管理员要配置网络必须从网络管理员处得到主机的名字、有效 IP 地址、子网掩码和广播地址等信息。

UNIX 系统中与配置 TCP/IP 网络信息相关的文件主要有：/etc/hosts、

/etc/hosts.equiv 和/etc/services、/etc/networks、/etc/netwasks。Solaris 系统与其他系统有微小差别，这些网络配置文件放在目录/etc/inet 下。

1. hosts 文件

hosts 文件中包含有主机名到 IP 地址的映像表。只有超级用户才有权修改该文件。该文件格式为：

```
#vi hosts
#
# Internet host table
#
127.0.0.1       localhost
11.23.2.45      mailsvr2adm   loghost
11.10.1.45      mailcls2
212.89.114.122  mailsvr2
212.88.114.1    router
11.23.2.98      catng
11.23.2.99      storsv
11.23.2.44      mailsvr1adm
212.88.214.123  mailsvr1
#212.88.214.126  mailsvr2v
```

这是一个与UNIX脚本类似的文件，#后的内容是注释，先是IP地址，再是主机名。

一个IP地址可以对应多个主机名，第一个是正式主机名，后面的是别名。第一行localhost是本地主机项，是配置在机器中的回送网络接口，用于允许主机上运行的客户/服务器应用相互通信，不能修改或删除。

从文件中可以看到IP地址11.23.2.45对应的主机名是mailsvr2adm和loghost，虽然loghost是别名，但是必须在该文件中定义，这是系统进程syslogd要求的。系统进程syslogd主要负责为系统作日志，将消息发给名为loghosts的主机，主机在日志文件/var/adm/messages中记录消息内容。

hosts文件的用途有：

- 网络应用该文件的主机名得到 IP 地址；
- 改变主机的 IP 地址；
- 修改完该文件中的 IP 地址后永久生效。后面介绍的用命令 ifconfig 也可以修改 IP 地址，但当主机重新引导后不再生效。

2. netmask 文件

netmask 文件用于配置主机网络接口的子网掩码。该文件格式为：

```
#The /etc/netmasks file for netmask
#
#class C network split into 30 host network
202.113.122.0  255.255.255.224
#class B network split into 1022 host network
173.110.0.0    255.255.240.0
```

该文件只有两列，第一列表示网络号，第二列定义连接到指定网络上的网络接口所设置的网络屏蔽。

3. networks 文件

在网络中每个 IP 子网可以分配一个名字，文件/etc/neworks 用于定义子网名字。该文件格式为：

```
#
#  network numbers
#
loopback      127
novellnet    130.57


#
#   internet networks
#
arpanet      10  arpa
milnet       26
```

文件中子网名在前面，随后是网络号，最后是网络别名。

4. resolve.conf 文件

在名字解析时用域名服务器（DNS）作为文件/etc/hosts 中主机的域，需要查找文件 resolve.conf，文件 resolve.conf 的格式为：

```
domain siwor.com
nameserver1   10.75.12.110
nameserver2   10.75.12.200
nameserver2   10.75.12.103
```

文件第一行表示主机所在域，即默认值，为 siwor.com。域中任何用户主机名解析的结果均会加上 siwor.com，如某一个用户主机名为 pridna，则解析后为 pridna.siwor.com。通常会配置两个服务器做域名解析，一个为主，另一个为辅。按顺序依次查询。

5. services 文件

/etc/services 文件用于包含 TCP 服务名、TCP 端口号和用于连接服务器的 TCP 协议清单。

该文件格式为：

```
#
#        tcp services. Sockets and so on
echo      7/udp
echo      7/tcp
discard    9/udp    sink null
discard    9/udp    sink null
systat     11/tcp
daytime    13/udp
daytime    13/tcp
netstat     15/tcp
ftp data    20/tcp
ftp        22/tcp
telnet      23/tcp
smtp       25/tcp
time        37/tcp
time        397/udp
name       42/udp
whois       43/udp
domain      53/udp
hostname    101/tcp
sunrpc      111/udp
sunrpc      111/tcp
```

若机器要求服务，则在/etc/services 中建立一项。

6. /etc/inetd.conf 文件

/etc/inetd.conf 文件是 inetd 的配置文件。UNIX 系统提供的许多服务可能只是偶尔使用，但提供网络服务的守护进程必须一直监听端口，这样导致许多守护进程长期监听但收不到信号却占用大量系统资源。为了避免这种情况，用一个名为 inetd 的“超级守护进程”取代这些进程。inetd 同时监听许多公开端口，当要连接到 inetd 正在监听的某个端口时，inetd 启动相应守护进程管理该端口上的服务。显然用一个进程监听取代多个进程监听节约了系统资源。在系统中每个运行的新网络服务都需要输入到 inetd.conf 文件中。

文件 inetd.conf 的格式为：

```
#echo stream tcp nowait root internal
echo dgram udp wait root internal
discard stream tcp nowait root internal
#discard dgram udp wait root internal
daytime stream tcp nowait root internal
#daytime dgram udp wait root internal
#chargen stream tcp nowait root internal
```

在 inetd.conf 文件中有 6 个域,分别是服务名、socket 类型(有 stream、dgram、raw 和 rdm）、传输层协议、是否允许多个站连接（wait 允许一次一个，nowait 允许多个）、服务进程的用户名和启动进程所带的参数。

在修改 inetd.conf 文件后 inetd 进程需要重新启动才能激活。

11.2.3 TCP/IP 接口管理

在 UNIX 系统中常连接多种不同的网络，存在不同的网络接口，每个接口在系统中都有一个接口号（接口名）。对这些接口必须进行监控、配置。

1. 查看系统网络接口

用命令：ifconfig －a 查看网络接口参数。

```
#ifconfig －a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
inet 127.0.0.1 netmask ff000000
ge0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
inet 192.168.2.1 netmask 255.255.255.0 broadcast 192.168.2.255
#
```

命令 ifconfig –a 输出行中：

- 最左面的为接口号（接口名），如 lo0；
- LOOPBACK 表示该接口是机器的回送接口；
- BROADCAST 表示接口正在发送和接收广播；
- RUNNING 表示网卡驱动程序和流模块正在运行；
- MULTICAST 表示接口支持多目广播预约；
- UP 表示接口在线；
- inet、netmask、broadcast 后分别是接口的 IP 地址、掩码和广播地址；
- mtu 是数据链路接口的最大传输单元。

2. 改变网络接口参数

命令 ifconfig 可以立刻改变网络接口参数，但系统重新引导后修改不再生效。要长期生效必须用修改文件 hosts 的方式。

ifconfig 命令格式：ifconfig interface-name para

例　设置接口 ge0 的 IP 地址为 212.98.312.23，屏蔽为 255.255.255.168，在线。

```
#ifconfig ge0 inet 212.98.312.23 netmask 255.255.255.168 up
```

使接口 ge0 离线：

```
#ifconfig ge0 down
```

3. 查看网络连接状态

命令 netstat -a 用于查看网络连接状态，输出信息非常丰富。

```
$netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4          0         2   ibm.telnet                211.115.48.225.1487
ESTABLISHED
tcp4       0      0  *.www                  *.*                    LISTEN
tcp4              0            0    localhost.49213                    *.*
LISTEN
tcp4       0      0  *.smux                 *.*                     LISTEN
tcp4       0      0  *.dtspc                 *.*                     LISTEN
tcp4       0      0  *.32770                 *.*                     LISTEN
tcp4       0      0  *.time                  *.*                     LISTEN
```

```
tcp4       0      0  *.daytime              *.*                    LISTEN
tcp4       0      0  *.chargen              *.*                    LISTEN
tcp4       0      0  *.discard              *.*                    LISTEN
tcp4       0      0  *.echo                 *.*                    LISTEN
tcp        0      0  *.exec                 *.*                    LISTEN
tcp4       0      0  *.klogin               *.*                    LISTEN
tcp        0      0  *.login                *.*                    LISTEN
tcp4       0      0  *.kshell               *.*                    LISTEN
tcp        0      0  *.shell                *.*                    LISTEN
tcp        0      0  *.telnet               *.*                    LISTEN
tcp        0      0  *.ftp                  *.*                    LISTEN
tcp4       0      0  *.sunrpc               *.*                    LISTEN
tcp4       0      0  *.6000                 *.*                    LISTEN
tcp4       0      0  *.32768                *.*                    LISTEN
udp4       0      0  *.snmp                 *.*
udp4       0      0  *.32788                *.*
udp4       0      0  *.time                 *.*
udp4       0      0  *.daytime              *.*
udp4       0      0  *.chargen              *.*
udp4       0      0  *.discard              *.*
udp4       0      0  *.echo                 *.*
udp4       0      0  *.32787                *.*
udp4       0      0  *.32786                *.*
udp4       0      0  *.32785                *.*
udp4       0      0  *.32784                *.*
udp4       0      0  *.32783                *.*
udp4       0      0  *.ntalk                *.*
udp4       0      0  *.32781                *.*
udp4       0      0  *.sunrpc               *.*
udp4       0      0  *.syslog               *.*
udp4       0      0  *.xdmcp                *.*
```



```
Active UNIX domain sockets
SADR/PCB Type  Recv-Q Send-Q Inode  Conn  Refs  Nextref    Addr
```

```
7005d400  dgram  0    0 13398640 0      0          0    /dev/SRC
70057fc0
7005e600 stream   0    0 1317c720  0     0      0    /tmp/.X11-unix  /X0
7007f100
70056c00 dgram    0    0  0 70057f80         0          0
70057f00
7005d200 dgram    0    0 130dae20          0 70057b80     0 /dev/log
70057f80
7005d000 dgram    0    0 1368bf40          0          0      0 /dev/.SRC-unix
/SRCFBcBMa
70057f40
70056a00 dgram    0    0 1321eb20         0          0          0 /dev/.SRC-unix
/SRCKWcBMb
70057ec0
7004c200 dgram    0    0 1357e8a0          0          0          0 /dev/.SRC-unix
/SRCVjcBMc
70057d40
7004f800 dgram        0       0          0 70057f80         0 70057f00
70057d80
7004a600 dgram  0       0 13039680       0          0          0 /dev/.SRC-unix
/SRC_lcBMd
70057d00
7004a800 dgram  0       0          0 70057f80         0 70057d80
70057dc0
7004a200 dgram    0       0 131b3720      0          0          0 /tmp/.PMDV1
70057c80
70044e00 dgram    0       0          0 70057f80         0 70057dc0
70057b80
#
```

可见，命令 netstat －a 的输出分为 UDP、TCP 和激活 UNIX 域的套接字三个部分。

- 在 UDP 输出中

Local Address 是所连主机的 IP 地址或名称、正在使用的端口号或服务名，“ * ”表示有一个驻守进程在监听端口，但没有建立连接。

State 表示连接的状态，有 Idle（空闲）、Unbound（非捆绑）。

- 在 TCP 输出中

Local Address 是所连主机的 IP 地址或名称、正在使用的端口号或服务名，“*”表示有一个驻守进程在监听端口，但没有建立连接。

Foreign Address 是远程主机的 IP 地址或名称、端口号或服务名，“*”表示 IP 地址或名称不详。

State 表示连接的状态，有：BOUND（捆绑）、UNBOUND（非捆绑）、IDLE（闲置状态）、CLOSED（套接字不在使用）、LISTEN（监听进入连接）、SYN_SENT（激活尝试建立连接）、SYN_RECEIVED（正在进行连接的初始化同步）、ESTABLISHED（建立连接）、CLOSED_WAIT（远程连接关闭并等待关闭套接字）、FIN_WAIT_1（套接字关闭并且正在关闭连接）、LAST_ACK（正在等待上一个应答）、FIN_WAIT_2（套接字关闭并且正在等待远程主机关闭）。

4. 网络接口统计信息

命令 netstat -i 显示主机每个接口的统计信息。

```
$ netstat -i
Name  Mtu   Network      Address              Ipkts Ierrs     Opkts Oerrs
Coll
lo0   16896 link#1                            374     0      2682     0     0
lo0   16896 127          localhost             374     0      2682     0
0
lo0   16896 ::1                               374     0      2682     0     0
en0   1500  link#2       8.0.5a.fc.c2.57   20013029     0   3444802     0
0
en0   1500  202.115.48.      ibmtc          20013029     0   3444802     0
0
```

命令 netstat -i 的输出中：

- Name 为接口名；
- Mtu 是数据链路接口的最大传输单元；
- Network 是接口连接的网络名或网络号；
- Ipkts 是接口已处理包的数量；
- Ierre 是接口曾收到的输入错误包的数量；
- Opkts 是接口输出包的数量；
- Oerre 是接口输出错误包的数量；

- Coll 是接口以太网冲突的数量；
- Queue 表示网线问题而不能发出的包。

命令 netstat -s 可以得到网络通信协议的统计信息。

```
%netstat -s
Ip:
    631 total packets received
    0 forwarded
    0 incoming packets discarded
    629 incoming packets delivered
    482 requests sent out
    1 dropped because of missing route
Icmp:
    2 ICMP messages received
    0 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 2
    2 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 2
Tcp:
    0 active connections openings
    0 passive connection openings
    0 failed connection attempts
    0 connection resets received
    2 connections established
    614 segments received
    479 segments send out
    1 segments retransmited
    0 bad segments received.
    0 resets sent
Udp:
    0 packets received
    2 packets to unknown port received.
```

```
    0 packet receive errors
  2 packets sent
  ......
  #
```

命令 netstat –s 的输出中分别有 IP 包、Icmp 包、Tcp 包、Udp 包的发送、接收信息以及丢弃信息等。

11.3 路 由 管 理

出于对网络应用需求不同等原因需要将网络分段。如果网络中节点之间存在大量通信，则应该将这些节点独立出来，既满足内部节点之间的通信在内部进行，又能满足内部和外部节点之间通信的高效性，这就需要路由管理。同样，将网络中传送目的不同、协议不同的数据分为不同的网络接口，也需要路由管理。

路由管理是网络实现的关键，UNIX 操作系统中的路由管理包括在操作系统中配置静态路由、默认路由以及管理路由表。

11.3.1 路由

Internet 将物理位置分散的各种计算机网络互连在一起相互通信。这些互连的网络之间通过路由器相互连接。

路由器是实现这种路由选择功能的专用计算机。路由器这个术语来源于 OSI 术语，其作用是转发网络间的数据包，而路由就是选择数据包发送路径的过程。

为了将路由的工作原理形象化，可以想象一个大的网络中有很多小网络，它们通过路由器相互互联，在两个远程各有一个主机。当一个主机想要发送一个数据包给另外一端的主机时，会将这个包发送到离它最近的一个路由器。当这个路由器接收到这个数据包时，会选择下一个到达目标地址的路由器；数据包到达下一个路由器后，由这个路由器选择再下一个路由器；这样，数据包从起始路由器出发，经过层层路由器的转发，最终到达能够直接发送到目标主机的路由器上。

IP 路由被看作是一种地址路由。意思是数据包在 Internet 中的传输只能依靠数据包中的目标 IP 地址。路由器就是根据这个 IP 地址来转发数据包的。

通过使用网络屏蔽，一个 IP 地址可以被分割为网络部分和主机部分。路由器在判断数据包的目标地址是否存在于与它直接相连的网络中（不需要再转发）时，会先将目标地址的网络部分提取出来，与自己的 IP 地址相比较。如果两者一致，意味着这个数据包可以直接发送到目标主机。

11.3.2 路由选择表

在实现路由选择协议时需要使用路由选择表。网络变化会引起路由选择表作相应改动。

在路由选择表中包括一些地址对。

- 目标网络的网络地址。
- 将数据包发送到哪个网络的路由器的 IP 地址：在路由表中列出的路由器的

IP 地址必须是与本机直接相连的。使用目标网络地址代替目标主机地址会使路由寻找更加有效，同时使路由表更小。

- 主机路由：虽然路由寻找通常是用于寻找网络而不是用于寻找单独的主机，但在 IP 路由中却允许建立单独主机的路由。单个主机路由让管理员在控制网络的使用和定制特殊路由时能更加灵活。建立一个特殊的针对单个独立主机的路由在网络查错时非常有用。

在网络中无论在路由器上还是在主机上都有路由选择表。

在 UNIX 主机中，维护路由选择表是一项非常重要的工作。系统管理员可以手工方式输入路由，即静态路由；也可以执行路由选择协议自动更新主机路由和路由器的路由选择表。手工方式花费的时间长，但比较安全；自动方式花费的时间少，但存在安全问题。

11.3.3 内核路由选择表

UNIX 主机的主机路由表存在于操作系统的内核，又被称为内核路由表或内核转发表，存储执行 IP 数据报转发所需要的目的地和网关清单。在内核路由表中目的地可以由一个主机、一个网络号、一个通配符或者“默认”表示。如果本地主机到目的地址之间不能直接传输，则由网关指出本地主机应将报文转发到哪个机器，若网关是本地主机的 IP 地址，则该主机将执行最后的报文传输。

用命令 netstat -r 可以查看内核路由表。

```
%netstat -r
Routing tables
Destination          Gateway              Flags   Refcnt        Use
Interface
127.0.0.1            127.0.0.1            UH      9             92752       lo0
130.87.218.0         130.87.218.18        U       3             328326189   nf0
default              130.87.216.1         UG      5             395597      le0
130.87.216.0         130.87.217.103       U       0             0           le0
```

在命令 netstat –r 的输出中包含如下一些项。

Flags 项中包含的字母有：

U：表示路由已经启动并可用；

H：指出路由是主机类型的路由；

G：表示路由使用中间网关执行 IP 数据报传输；

D：表示该路由作为 ICMP 重定向的结果加入到表显示内核路由中，ICMP 重定向是一个数据包，该数据包由路由器发出，用于通知本地主机存在到达最终目的地的

较短路径。

Refcnt 表示使用相同数据链路地址的路由数量。

Use 表示使用该路由已转发包的数量。

Interface 显示执行最后发送的接口，如果数据包转发使用中间网关则该域为空。

内核路由表包括三种类型的路由：主机路由（Host Routes）、网络路由（Network Routes）、默认路由（Default Routes）。

- 主机路由（Host Routes）

主机路由是内核路由选择表中指向单个 IP 地址或主机的路由记录。主机路由在 Flags 标志中用“H”标识。如：

Destination	Gateway	Flags	Ref	Use	Interface
211.199.121.69	12.13.56.1	UH	0	4282	le0

表示内核路由表中一个主机路由：本地主机通过网关 12.13.56.1 可以到达目的主机 211.199.121.69。

在 UNIX 系统中如果知道一个主机路由，则首先从路由选择表中选择它；如果没有主机路由则选择网络路由或默认路由。

- 网络路由（Network Routes）

在内核路由选择表中最常用的路由为网络路由。网络路由代表主机可达的并且可作为目标的整个网络。网络路由在 Flags 标志中用“G”标识。如：

Destination	Gateway	Flags	Ref	Use	Interface
198.112.15.78	12.13.56.69	UG	0	8981	

表示内核路由表中一个网络路由：则本地主机将网络目标为 198.112.15.78 的数据报转发到 IP 地址为 12.13.56.69 的路由器上。

- 默认路由（Default Routes）

内核路由表中的默认路由表示最后实施的路由。当主机搜索匹配目标 IP 地址的主机路由或网络路由时，如果在路由选择表中没有找到，主机就使用默认路由转发数据报。如：

Destination	Gateway	Flags	Ref	Use	Interface
default	12.13.56.99	UH	0	3584	

表示内核路由表中一个默认路由：则本地主机将数据报转发到 IP 地址为 12.13.56.69 的路由器上。

在网络中的每个 UNIX 主机只能定义一个默认路由。

11.3.4 静态路由和动态路由

静态路由是路由选择表中的固定项，由系统管理员以手工方式加入到路由选择表中。对于简单网络，特别是配置不经常改变或者安全性要求较高的网络，选择静态路由最好。

动态路由是采用路由选择协议自动加入到路由选择表中，通过网络中的路由器之间动态地交换路由信息而实现的。免去了系统管理员用手工方式配置路由表的繁琐工作。

动态路由选择表中的记录可表现为 ICMP 重定向的结果。ICMP 重定向是在主机向一个路由器上转发一个数据报而该路由器知道一条更好的路径，则主机将数据报转发到这条更好的路径上时发生。ICMP 在主机使用默认路由但却有多个路由器可用于该路由时，发生 ICMP 重定向。ICMP 重定向通常会导致在内核路由表中增加主机路由，即在 Flasgs 项中增加标识“ D ”。

11.3.5 操作内核路由选择表

系统管理员可以用命令修改内核路由选择表。

1. 用 route 命令增加、删除、刷新、查找内核路由表中的路由

系统管理员可以用/usr/sbin/route 命令对内核路由表进行操作。

route 命令可以增加或者删除内核路由表中的主机路由、网络路由、默认路由。但 route 命令只能临时修改路由，当系统重新引导后 route 命令的修改就不生效，应为它不在启动和配置文件中改变路由。

- 用 route 命令增加、删除内核路由表中的路由

route 命令增加路由的语法为：route [-fn] (add|delete) [host|net] destination gateway count

其中选项和参数的解释如下。

-f：让 route 命令“ flush（激活）”所有的路由。

-n：以数字（IP 地址）的形式显示所有主机和网络的 IP 地址。默认网络地址被显示为“ default ”。

add：在路由表中添加主机、网络或默认路由。

delete：从路由表中删除指定的主机、网络或默认路由。

net：指出目标为一个网络。当目标地址为一个网络地址时使用这个参数，例如 192.45.3。

host：指定目标为一个主机。

destination：指出数据包会被发送到的主机或网络的 IP 地址。destination 可以为一个主机名（或者在/etc/hosts 中列出的别名），一个网络名（或者在/etc/networks 中列出的别名）,一个 Internet 地址，或者关键词“ default ”。如果指定了“ default ”关键词，默认的网关条目会变为 gateway。如果数据包的目标地址与路由表中的任何一个地址都不匹配，就会被送到这个 gateway 指定的地址。

gateway：指出到达目标需要通过的网关节点。这个地址必须是 IP 地址或者一个主机名。如果远程路由器支持代理 arp（地址解析协议）路由，也可以使用主机自己的 IP 地址。

count：是一个整数，指明网关是本地主机还是远程主机。如果 count 大于 0，网关就是一个远程主机。如果等于 0，网关就是本地主机。默认为 0。

例 在内核路由表中增加一条主机 (201.42.15.199) 路由，网关为 11.11.2.5：

```
$route add host 201.42.15.199 11.11.2.5
```

在内核路由表中增加一条网络（202.48.15.0）路由，网关为 11.11.2.9：

```
$route add net 202.48.15.0 11.11.2.9
```

在内核路由表中增加一条默认路由，默认路由器为 11.121.2.1：

```
$route add default 11.121.2.1
```

在内核路由表中删除前面增加的三条路由：

```
$route delete 201.42.15.199 11.11.2.5
$route delete 202.48.15.0 11.11.2.9
$route delete default 11.11.2.1
```

- 用命令 route flush 刷新内核路由选择表中所有的静态、动态路由

```
#route flush
default   201.42.15.199    done
e30       loopback       done
#
```

- 如果路由发生问题，可以用命名 route get 和 route monitor 查找问题所在

命令 route get 为一个目标查找并显示路由。

命令 route monitor 持续报告任何有关路由选择信息的变化、路由选择查找错误或者其他值得怀疑的网络部分。

2. 检查路由表

可以使用/usr/sbin/netstat -rn 命令检查路由表。

```
# netstat -rn
```

```
Routing tables
Destination Gateway Flags Refs Use Interface Pmtu PmtuTime
193.55.8 192.45.3.8 UG 0 0 lan0 1500
default 192.45.3.1 UG 0 0 lan0 1500
#
```

11.4 执行路由选择协议

在网络路由配置中，如果网络结构复杂或希望路径可以重新自动配置时，应使用路由选择协议。在 TCP/IP 中可以采用多种不同的路由选择协议，如路由信息协议（Routing Information Protocol，RIP）、开放式最短路径优先（Open Shortest Path First，OSPF）、路由发现协议（Router Discovery Protocol，RDISC）、边界网关协议（Border Gateway Protocol，BGP）等，本节主要介绍这些协议。

11.4.1 IP 转发

IP 转发是在不同子网之间以 IP 路由方式转发数据包的过程。一般用路由器实现，但是拥有多网络接口的主机也可以实现。

11.4.2 RIP 简介

路由信息协议（RIP）作为加州大学的部分成果，是 BSD 版本的 UNIX 系统的网络组成。目前，大多数 UNIX 系统中都有 RIP。RIP 采用广播技术、以用户数据报的方式将路由表中的网络距离和网络地址播放到网络中的其他网关上，这种方式增加了网络上的流量，降低了网络传播速度。

以 RIP 为基础的互联网络中，任意两个网络之间不能多于 15 个路由器，所以 RIP 协议方式支持的网络规模有限。

11.4.3 OSPF 简介

开放式最短路径优先（OSPF）是一种开放式的链路状态路由选择协议，开发该协议的目的是希望成为 Internet 中的主协议。该协议的准确描述应该是最优化路径。

OSPF 使用 IP 数据报头标中的信宿地址和服务类型信息来建立路由，从网络拓扑信息的路由表、OSPF 网关使用牺牲度量来决定最短路径。度量包括路由速度、通信量、可靠性、安全性及有关的其他方面。OSPF 算法可同时用于外部路由和内部路由，可将一个大型网络分成几个小的区域，每个区域都有自己的网关和路由算法。

OSPF 允许在不同的网段上使用不同的子网掩码，不但网络的可伸缩性高而且子网上的主机数目较大，可以支持各种长度的子网，与 RIP 协议相比能支持更大的互联网。

11.4.4　RDISC 简介

路由发现协议（RDISC）与 RIP、OSPF 不一样，它实质上不是路由协议，其功能只是用于发现网络上存在的路由器和主机。常见的有基于 ICMP 协议的路由发现协议（ICMP Router Discovery Protocol，IRDP）。

11.4.5　BGP 简介

边界网关协议（BGP）是自治系统间的路由协议，是一种外部网关协议，用于处理各 ISP 之间的路由传递。其特点是有丰富的路由策略，这是 RIP、OSPF 等协议无法做到的，因为它们需要全局的信息计算路由表。BGP 通过 ISP 边界的路由器加上一定的策略，选择过滤路由，把 RIP、OSPF、BGP 等的路由发送到对方。全局范围的、广泛的 Internet 是 BGP 处理多个 ISP 间路由的实例。

11.4.6　UNIX 路由选择协议实现

当一个 UNIX 主机中有两个或两个以上的网络接口时，该 UNIX 系统通过配置可以作为 IP 路由选择器。被配置成路由选择器的 UNIX 系统中应该驻守有路由选择协议进程。

用 UNIX 主机实现 IP 路由选择转发（IP Forwarding）功能的步骤如下。

1. 打开 IP 转发功能

不同的 UNIX 系统，打开 IP 转发功能的方式不同。

- Solaris 系统：安装了两个或多个网络接口之后自动打开 IP 转发。另外用手工方式输入命令：ndd –set /dev/ip ip_forwarding 1 也可以打开 IP 转发。
- AIX 系统：使用工具 SMIT，将主机转换为路由器。
- HP/UX 系统：使用工具 SAM，在命令行输入 sam。图形接口下将主机转换为路由器。
- Linux 系统：echo 1 > /proc/sys/net/ipv4/ip_forward。
- Digital UNIX 和 SCO System V 系统：自动打开。
- SCO Unix Ware 系统：使用/etc/inet/menu 命令激活一个菜单即允许主机作为网关配置。

2. 实现 RDISC 协议和 RIP 协议或 OSPF 协议的配合

在 UNIX 系统上要实现 RDISC，需要驻守进程 in.rdisc（有的 UNIX 系统，如 SCO UNIX，又称为 irdd）。in.rdisc 驻守进程的配置需要选项。

除 RDISC 协议的实现要求系统中有驻守进程 RDISC（in.rdsc）外，还需要 RIP 协议实现的系统驻守进程 gated 或 routed。但是 routed 与 gated 相互排斥，不能同时使用（只有 gated RIP I 才能与 routed 一起使用）。routed 用于简单网络，gated 才能实现与 OSPF 协议的配合使用。

在有的系统中用进程 irdd 实现 RDISC，如 SCO UNIX。控制进程 irdd 的文件为 /etc/irdd.conf，此文件中的语法为：keyword=value，value 为 8、10、16 进制值。现已被承认的关键字有如下几个。

- interface：该关键字表示用一个具体的网络接口表示配置信息。接口用接口名表示，如 lo0。每个接口配置都承认关键字 advertise、advmax、advmin、broadcast、discover、lifttime 和 preference。在配置每个接口时，若使用这些关键字，默认值由关键字代替并重新写入。若在具体的接口配置时未指定关键字，则使用默认值初始化。
- broadcast：指出报文以广播地址而非多目广播地址发送。
- defadvertise：指出默认行为是广告所有的接口，若以每个接口为基础，该关键字可以忽略。若该关键字未指定，默认行为是不广告接口。只有系统配置为路由器时才使用该关键字。若系统配置为主机，则忽略该关键字。
- defadvmin：指出默认时广告时间的最短间隔（以秒为单位），仅在系统配置为路由器时使用。若关键字未指定，默认值为最大值的 75%，最小的合法值为 3。
- defadvmax：指出默认时广告时间的最大间隔（以秒为单位），仅在系统配置为路由器时使用。若关键字未指定，默认值为 600，最小的合法值为 4。
- deflifetime：指出广告的默认寿命值。若关键字未指定，默认值为最大广告间隔时间的 3 倍。仅在系统配置为路由器时使用。
- defprefence：指出广告的默认优先值。若关键字未指定，默认优先值为 0，默认值为 0x80000000 则意味着主机应该忽略该系统。只有系统配置为路由器时使用。

例 一个将系统配置为路由器的/etc/irrd.conf 文件，该文件中 irdd 执行的功能有：

除 e1B0 接口外，所有接口有多目广播。

广告的优先权为 2 400，广告的最大时间间隔为 30 分钟，广告的有效时间为 40 分钟。

```
#sample route discovery configuration
defadvertise                  #advertise all interfaces
dafadvmax=1800              #thirty minutes
deflifetime=2700              #forty-five minutes
interface e1B0 broadcast       #no multicast on this lan
```

11.4.7　驻守进程 gated 和 routed 的配置和管理

1. 驻守进程 gated

驻守进程 gated 是一个公开的路由选择进程，可以支持多种不同的路由选择协议。控制进程 gated 的文件默认为配置文件/etc/gated.conf。

文件/etc/gated.conf 的内容：

```
$more /etc/gated.conf
interfaces {
#interface hme0 passive;
interface qfe0 passive;
interface qfe1 passive;
#interface qfe2 passive;
};
autonomoussystem 150;
snmp off;

rip  off  {
  broadcast;
  defaultmetric 5;
  interface le version 2 multicast;
};

static {
  default  gateway x.x.x.1  preference  140 retain;
#10.0.0.0 gateway x.x.x.1 preference  140  retain;
};
```

该文件中的语句由 token 组成，token 之间用空格或 Tab 键分开，语句结束用分号“；”。用“#”和“/*”开头的行都是注释行。

Gated 发布的版本目前主要有：RIP I、RIP II、OSPF，下面分别说明。

- 用 gated 执行 RIP I

用/etc/gated.conf 文件配置实现 RIP I，应该在文件/etc/gated.confgated 中加入行：

```
#
# enable rip i
#
/*        打开所有接口的 rip：包括进出          */
/*        在 gated daemon 启动时在路由表中加入静态缺生路由          */
rip yes {
  interface all ripin ripout;
};
static{
  default gateway 110.110.11.10 retain;
};
```

- 用 gated 执行 RIP II

用/etc/gated.conf 文件配置实现 RIP II，应该在文件/etc/gated.confgated 中加入行：

```
#
# enable rip i
#
/*        打开所有接口的 rip version 2：包括进出          */
/*        在 gated daemon 启动时在路由表中加入静态缺生路由          */
rip yes {
  interface all version 2 ripin ripout;
};
static{
  default gateway 110.110.11.11 retain;
};
```

- 使用 gated 执行 OSPF

用/etc/gated.conf 文件配置实现 OSPF，应该在文件/etc/gated.confgated 中加入行：


```
#
# using ospf, no rip
```

```
#
rip no;
ospf yes;
backbone {
  #any ospf packet are considered valid
  authtype none;
  interface all {priority 5};
};
```

2. 驻守进程 routed

- routed 配置文件

文件/etc/gateways 控制进程 routed。启动 routed 进程时，读取/etc/gateways 文件并安装定义在路由选择表中的路由。

例　文件/etc/gateways 实例。

```
$more /etc/gateways
net 150.101.17.0 gateway 150.101.16.1 metric 1 passive
net 0.0.0.0 gateway 150.101.16.2 metric 1 passive
```

文件/etc/gateways 中每行的格式为：

[net|host] [net-name|host-name] [/net-mask] gateway route-address [metric cost] [active|passive]

[net|host]：用于指定网络中每条记录时用 net 开头；用于指定主机的每条记录时用 host 开头。

net-name：网络 IP 地址或文件/etc/networks 中网络的名字，路由包的最后目的地。

host-name：主机名或文件/etc/hosts 中主机的 IP 地址或名字，路由包的最终目标。

route-address：用于转发报文的路由器的 IP 地址。

cost：路由的开销，范围为 1~16，其中 16 为不可达。默认为 0。

active | passive：路由可为激活或不激活。激活路由在一段时间内若未收到新信息则过期，不激活路由在 RAM 中永久存在。默认为不激活。

例　UNIX 系统作为路由器，IP 地址为 154.69.77.1。现在要配置文件/etc/gateways 启动该 UNIX 路由器中的 routed 进程。下面分为到网络和到主机两种情况配置。

发到网络（213.120.225.0）的包先被发到该 UNIX 路由器。该项在路由表中标

为“激活”，路由的开销设置为 4：

```
net 213.120.225.0 gateway 154.69.77.1 metric 4 active
```

- routed 执行 RIP Ⅰ

当系统作路由器使用时，routed 进程只有和 gated 进程执行 RIP Ⅰ协议时才能相互配合，即这时的 routed 执行 RIP Ⅰ。这时 routed 进程运行必须带选项-s。命令 routed –s 启动 RIP 协议并向所有配置在路由器上的网络接口广播。在启动进程 routed 后系统作为路由器运行，应避免用手工方式改变路由，否则路由表会无效。惟一的修复方法是停止 routed 进程再重新启动。

除执行 RIP Ⅰ之外，由于 routed 和 gated 容易发生冲突，若想运行 routed，必须确认文件 gated.conf 不存在。如果文件 gated.conf 存在，则必须删除或重命名，否则系统下次启动时会执行 gated 进程而非 routed 进程。

11.5 UNIX 中的点到点协议（PPP）

点到点协议（Point-to-Point Protocol）PPP 主要用于分散的网络环境中用调制解调器（Modem）和电话线上网的情况。PPP 的目的在于提供在网络节点间建立一种与其他网络层次的协议能进行通信的链路方法。

1. PPP 协议的主要功能

- 为 IP 数据包传输提供异步串行通信接口。
- 在需要时建立连接。
- 配置选项协商连接。
- 连接终端并自动挂断。

在 PPP 中 UNIX 系统通常作为服务器将一组用调制解调器上网的计算机连接到 Internet，如图 11.1 所示。UNIX 服务器为上网计算机客户端提供上网的电话号码、用户登录名及口令、域名服务器 DNS 地址等参数。几乎所有版本的 UNIX 都提供 PPP 及安装接口。

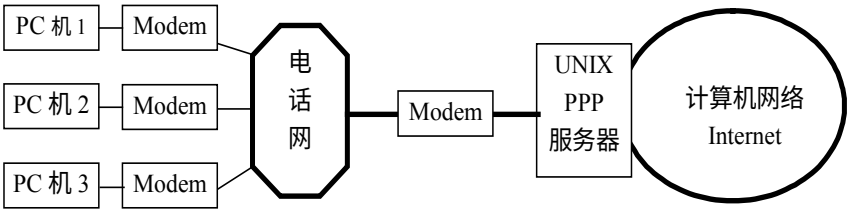

图 11.1 用 PPP 服务器上网

2. UNIX 系统中 PPP 协议的主要特征

- 遵循 RFC 标准连接终端并自动挂断，完成 Internet 点到点通信。
- 提供 CRC 错误检查。
- 支持全双工通信。
- 系统管理员可以控制通信传输率、加密方式及其他特征。

3. 配置允许客户端连接的 UNIX 系统 PPP 服务器

配置服务器的 PPP 服务需要完成以下工作：

- 配置 Modem 到服务器并设置正确的拨入电话号码；
- 为 PPP 连接增加一个用户登录，用户登录后能享用的是 PPP 服务；
- 表示客户计算机的 IP 地址和 DNS 服务器；
- 配置 PPP 接口选项。

只有客户端也将 Modem 配置到客户端系统后，客户端和服务器双方才能通过 Modem 通信。

11.6　网 络 服 务

Internet 实现了网络与网络之间的互连，进入 Internet 的计算机可以应用 Internet 提供的各种服务。目前 UNIX 系统承担了绝大多数的 Internet 网络服务。UNIX 系统提供的网络服务主要有域名服务、Web 服务、邮件服务、FTP 服务、代理服务。

11.6.1　域名服务

在 Internet 中用 IP 地址进行网络访问是可行的。但是 IP 地址的表示方式是 32 位二进制数，划分为 4 个字节，每个字节之间用一个小圆点分开，这种表示方式没有一定特征，不便于记忆和使用。为此，人们想到了用体现个性特征的字符地址代替 IP 地址在网络上使用。这样的字符地址就是域名，用域名同样能访问网络。由于在网络内部的数据交换仍然需要 IP 地址，当用户用域名作为地址访问 Internet 时，则要求网络中有能将域名转换成 IP 地址的服务器，将用户输入的域名地址转换为网络中的 IP 地址。同样，该服务器也需要将 IP 地址转换成用户的域名。域名服务就是完成域名与 IP 地址之间的转换。提供域名服务的主机叫做域名服务器。UNIX 系统能够作为 Internet 上的域名服务器。

1．层次化的域名空间

Internet 的域名空间是一种基于树型结构的分层命名，如图 11.2 所示。

图中最上面的是根域名，之下一层表示的是顶级域名，其中的 net、com、edu、gov、mil、org 分别代表在 Internet 顶级注册的网络机构、工商企业、教育、政府、军事、组织单位，而 cn、uk、ca、jp、hk、tw 分别是中国、英国、加拿大、日本、香港、台湾等一些国家和地区的域名。

域名的描述排列从小到大，如 mail.scu.edu.cn，按主机名（mail）、域名（scu）、扩展名（edu）、顶级域名（cn）。每个域都有自己的域名服务器。域名服务器除了知道自己域中所有的主机名之外，还要了解下一个相关域的服务器名。通过域名服务可以访问 Internet 上连接的任何一台计算机。

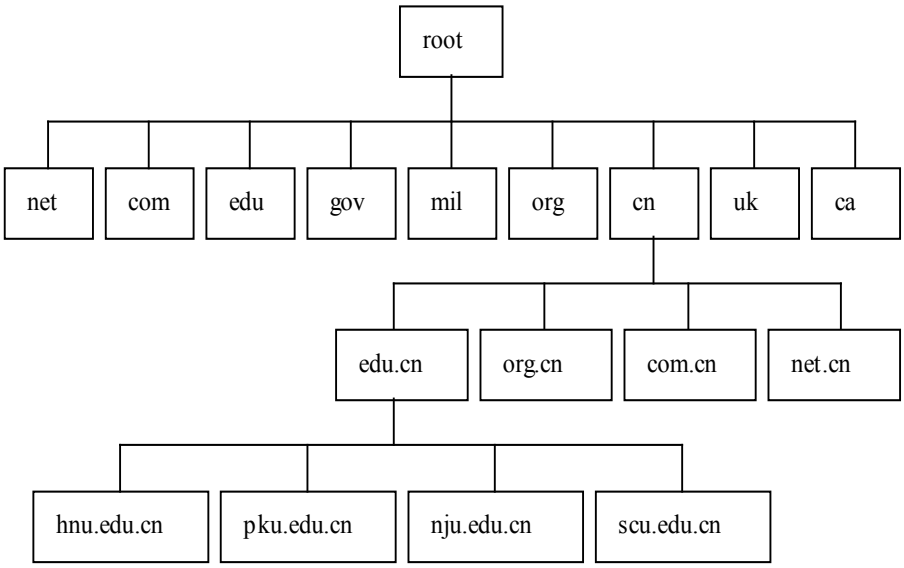

图 11.2 Internet 域名结构

- 层次域名解析：域名服务器中保存有域的树型结构信息。Internet 中有成千上万台域名服务器。每台域名服务器负责解析本级域名及本级域名之内的域名。如 edu.cn 域名服务器负责解析 edu.cn 和 edu.cn 之内的域名，如 pku.edu.cn、nju.edu.cn、scu.edu.cn 等；而 scu.edu.cn 域名服务器负责解析 scu.edu.cn 和 scu.edu.cn 之内的域名，如 mail.scu.edu.cn、math.scu.edu.cn、cs.scu.edu.cn 等。在 Internet 上这些域名服务器正是以这样的层次式解析方式相互联系在一起构成整个 Internet 的域名解析。

2．公共和私有的域名空间

前面介绍的 Internet 域名空间是公共域名空间。如果要将本地网络和一个公众

网络相连接，则要求有官方的 IP 地址和域名。如果已经向所在域的上级管理中心申请注册了一个域，那么该域就可以独立地管理域中计算机的域名。正如前面介绍的多级域关系。

但是如果要将所在的网络和公众网相互分离，如通过代理服务器分离，这样在网络内部就可以任意选择域名和 IP 地址，这样的域名空间是私有域名空间，在 Internet 上不被承认，只能应用在用户网络内部。如果要在 Internet 上使用网内域名,则需将网内私有域名空间的顶级域名在 Internet 的上一级域名管理中心注册并且遵循命名的惯例才能使用。

3．域名服务器

IP 网络中依靠 Internet 地址进行通信。当用户和应用程序通过主机名查找主机时，需要在数据包发送之前，由 IP 协议层将这个主机名（主机域名）解析成为一个 IP 地址。主机名解析成 IP 地址的转换在网络连接建立之前通过一个运行在域名解析服务器中的 gethostbyname()库函数来实现。这个库函数也被称为解析器，因为它的作用是将主机名解析为一个 IP 地址。相反，同样也要将 IP 地址解析成主机名（域名）。将 IP 地址解析成为主机名是在域名解析服务器中通过调用一个名为 gethostbyaddr()的库函数来实现的。

域名解析服务器中有域名解析数据库，该数据库是通过文件/etc/hosts 体现的。在该文件中存有 IP 地址和域名的对应关系。

在 Internet 中域名服务器非常重要，作为域名服务器需要完成的主要功能有：

- 响应客户机的请求，将主机名解析为 IP 地址；
- 划分名称空间；
- 将域名解析请求转发到其他域名服务器；
- 提供域名服务程序，即系统守护进程 named。

例　域名解析。

在 scu.edu.cn 域名服务器中的域名解析：

```
cs.scu.edu.cn <--->212.115.128.1
math.scu.edu.cn <--->212.115.134.1
art.scu.edu.cn <--->212.115.123.1
```

为了网络域名解析稳定可靠，同一网络中应该有主、辅域名服务器互为备份。但是主域名服务器对它所包含的数据来说是最有权威的，是由上一级的域赋予它的权利。它可以依次创建子域并分配权限。与辅服务器相比较而言，主服务器上的信息是最新的信息。辅域名服务器是一个备份服务器。它按一个可以设置的时间间隔从主服务器上更新数据。辅域名服务器可以选择是否在本地磁盘上保留原始记录。

这可以在 named.boot 中进行配置。

除了主、辅域名服务器之外，有的网络中还设置有缓存服务器。缓存服务器不在本地磁盘上存储数据，所有数据都放在系统缓存中。在多用户系统中使用缓存服务器可以加速域名查找速度。缓存服务器提供一个非正式的备份方式，与在本地存储数据的辅域名服务器相比较，这种方式最大的优点是：在一个新的子域加入到一个主域名服务器中时，它不需要更新本地 DNS 数据库中的数据。

4. 配置域名服务器

配置主域名服务器的过程为：

- 向上级管理中心注册域名；
- 在/etc/hosts 文件中定义全格式的主机名（域名）；
- 修改/etc/rc.config.d/namesvrs 文件并启动 named 守护进程；
- 配置 DNS 客户端的功能。

例 将 sanfran 节点配置为一个主域名服务器，这个服务器所在的域为 scu.edu.cn。

(1) 向上级 Internet 管理中心（中国教育网管理中心，管理域名 edu.cn 及其下域名）注册域名 scu.edu.cn。同时还需提供主域名服务器和辅域名服务器的名称和 IP 地址。

(2) 提供完全格式的/etc/hosts 文件。

UNIX 操作系统能够将/etc/hosts 中的数据直接转换成主域名服务器上的 DNS 数据库。为了实现这个功能，hosts 文件中的所有条目需要完全格式的主机名。而旧主机名可以作为别名存在，hosts 文件中域名服务器不能解析的子域条目必须删除（注意，localhost 条目必须保留）。下面的例子是 scu.edu.cn 主机上已经修改好的 hosts 文件内容：

```
#vi /etc/hosts
127.0.0.1 localhosts
202.115.32.39 scu.edu.cn
212.115.128.1 cs.scu.edu.cn     cs
212.115.134.1 math.scu.edu.cn   math
212.115.123.1 art.scu.edu.cn     art
```

(3) 创建一个目录保存 DNS 数据库文件。

hosts_to_named 程序会创建几个 DNS 数据文件。在默认情况下，这些文件会存储在/etc/named.data 目录下。另外还可以使用 mkdir 命令手工创建这个目录.


```
mkdir /etc/named.data
```

```
cd /etc/named.data
```

(4) 为 hosts_to_named 程序创建一个 param 文件。

hosts_to_named 是一个强大的创建 DNS 数据库文件的工具。hosts_to_named 自动搜索一个 param 文件来决定对哪些域的主机提供域名解析服务。

例　sanfran 域名服务器的 param 文件如下所示：

```
vi /etc/named.data/param
-d ca.hp.com          <---这里填上你的域名
-n 128.1.1            <---这里填上你的子域的地址
-z 128.1.1.1          <---这里填上你的辅域名服务器的 IP 地址
-b /etc/named.boot    <---指定 DNS 启动文件存放位置
```

文件/etc/named.data/param 中的选项说明如下。

-d：在后面加上一个域名，指定这个域名服务器所服务的域，由于一些域名服务器同时为多个域提供域名解析，这时就需要多个-d 选项。

-n：在后面加上这个域中的每个子域的名称，由于很多主机存在于子域中，这时就需要多个-n 选项。

-z：创建辅域名服务器需要下载的配置文件。因为辅域名服务器需要从主域名服务器上下载一个配置文件，其中包含主域名服务器的 IP 地址和这个域的其他信息。

-b：决定 DNS 启动文件存储的位置，标准的位置是/etc/named.boot。

其他选项可以参见 hosts_to_named 名称的联机帮助。

(5) 创建 hosts_to_named 的 DNS 数据文件和启动文件。

hosts_to_named 工具自动使用/etc/hosts 文件来生成 DNS 数据文件,使用的是在 param 文件中定义的选项。如：

```
# hosts_to_named -f param
Translating /etc/hosts to lower case ...
collecting network data ...
       128.1
creating list of multi-homed hosts ...
creating "A" data (name to address mapping) for net 128.1 ...
creating "PTR" DATA (address to name mapping) for net 128.1 ...
creating "MX" (mail exchanger) data ...
Building default db.root file ...
Building default boot.sec.save for secondary servers ...
Building default boot.sec for secondary servers ...
```

```
Building default boot.casheonly for cashing only servers ...
done
```

(6) 下载一个 db.cashe 文件，其中包含 root 服务器的 IP 地址列表。

hosts_to_named 工具能够创建几乎所有的 DNS 数据库文件，只有 db.cashe 文件必须手工创建，其中包含有 root 域名服务器地址。可以从 Internet 上 ftp 下来一个包含当前 root 服务器地址的文件。由于 root 服务器列表随时都可能改变，可能会需要定时下载这个文件。

(7) 修改/etc/rc.config.d/namesvrs。

手工启动 named 进程，不需要重启机器。

为了让域名服务器的守护进程“ named ”在系统启动时一起启动，需要将/etc/rc.config.d/namesvrs 中的 NAMED 变量设置为“ 1 ”。例如：

```
vi /etc/rc.config.d/namesvrs
...
NAMED=1
NAMED_ARGS=""
...
# /sbin/init.d/named start
```

named 可以带参数启动，但是通常对 NAMED_ARGS 变量的值都不进行设置。

5. 配置辅域名服务器的步骤

- 为 DNS 数据文件创建一个目录。
- 从主域名服务器上 ftp 一个配置文件。
- 从主域名服务器上 ftp 一个 db.127.0.0 和 db.cache 的拷贝。
- 从主域名服务器上 ftp 附加的 DNS 数据文件（可选）。
- 修改配置文件/etc/rc.config.d/namesvrs，并启动 named 守护进程。
- 设置辅域名服务器上客户端的功能。

例　创建一个辅域名服务器的步骤如下。

(1) 为 DNS 数据文件创建一个目录。

在辅域名服务器上创建一个单独的目录存放数据库文件和配置文件。

大部分辅域名服务器在本地保存本域的 DNS 数据文件。这个数据库文件一般存储在/etc/named.data 目录下。

```
# mkdir /etc/named.data
# chmod 755 /etc/named.data
```


(2) 从主服务器上 ftp 一个启动配置文件。

named进程在启动时会参考/etc/named.boot中的设置来决定DNS数据库文件存放的位置。可以从主服务器上下载一个 named.boot 文件。

主服务器上可能会有两个版本的 named.boot 文件：/etc/named.data/boot.sec.save 和/etc/named.data/boot.sec。如果希望辅域名服务器在系统重启后仍旧保留所有的 DNS 数据库文件的一个完整磁盘拷贝，可以选择下载/etc/named.data/boot.sec.save 文件。这样做的目的在于：如果在辅域名服务器启动时主服务器无效，辅域名服务器仍然可以使用存储在本地的 DNS 数据文件来启动 DNS 服务。

如果选择下载/etc/named/boot.sec 文件，辅域名服务器就不会主动维护本地磁盘的数据库文件，在这种情况下，如果辅域名服务器启动时主服务器处于无效状态，named 进程就不能够启动。

需要注意的是只能有一个 boot 文件。named 守护进程启动时会自动读取这个文件中的配置信息。

```
ftp 128.1.1.1 <--- FTP 连到你的主域名服务器
get /etc/named.data/boot.sec.save /etc/named.boot
quit
```

(3) 从主服务器 ftp 一个 db.127.0.0 和 db.cache 文件。

每个域名服务器都有两个 DNS 数据文件。db.127.0.0 文件用来解析 loopback 地址。而域名服务器依靠 db.cache 文件来查找根域名服务器。从主域名服务器上可以下载这两个文件。

```
ftp 128.1.1.1                    FTP 连接到主服务器
get /etc/named.data/db.127.0.0     get db.127.0.0
get /etc/named.data/db.cache       get db.cache
quit
```

(4) 从主域名服务器上 ftp 另外的数据库文件（任选项）。

如果想要辅域名服务器在本地磁盘中保存所有的 DNS 数据文件的拷贝，需要从主服务器上下载所有的数据文件。第一次下载这个文件后，以后辅域名服务器会自动按照一定的时间间隔从主服务器上更新数据。

```
ftp 128.1.1.1                  <--- ftp to the promary server
mget /etc/named.data/db.*        <--- GET the DNS database files
quit                           <-- 退出 ftp
```

(5) 修改/etc/rc.config.d/namesvrs。

手工启动 named 进程，不需要重新启动系统。

```
# vi /etc/rc.config.d/namesvrs
```

```
NAMED=1
NAMED_ARGS=""
# /sbin/init.d/named start
```

named 可以带参数运行，但是通常都不对 NAMED_ARGS 变量进行设置。

与设置主域名服务器不同的是，设置辅域名服务器时不需要使用 hosts_to_named 命令，hosts_to_named 命令不是用来创建辅域名服务器和 cache_only 服务器的。使用 rcp，ftp 从主服务器上传送需要的 BIND 文件，如果有必要，也可以手工修改 BIND 文件。如果是私有域（不与公众网相连），就必须手工创建 db.cache 文件，因为这时的根服务器必须自己来指定。

与设置主域名服务器的另外一个不同之处是：可以选择将 DNS 数据文件存储在本地，也可以选择不存储在本地。但无论哪种服务器上都必须要有 db.127.0.0 这个文件。在本地存储大量的 DNS 数据文件要占用磁盘空间，但在一直无法存取主服务器时这种方式会起作用，因为可以在不能立即和主服务器通信的情况下重新启动辅域名服务器，但是在超时之前主服务器必须重新投入使用。

6．设置 DNS 客户端

在一个 DNS 域中所有的主机，包括主域名服务器和辅域名服务器，都应该被配置为 DNS 客户端。将一个主机配置为一个 DNS 客户端可以确保主机的解析器使用指定的 DNS 域名服务器来解析域名和 IP 地址，而不是用本地的 hosts 文件。配置步骤如下。

(1) 创建/etc/resolv.conf 文件。

域名解析器的配置文件是/etc/resolv.conf。resolv.conf 文件有两个重要的组成部分：

a 创建一个“查找”列表。

/etc/resolv.conf 文件中的 search 关键词定义了一系列的域名，解析器在进行主机名解析时会搜寻这个列表。这个列表中至少应该列出自己主机所在的域。为了以后方便，也可以增加其他域名。

在这个搜索列表中添加其他域可以让用户不必输入完全格式的主机名，例如，下面的 resolv.conf 文件在搜索列表内包含有 ca.hp.com 这个域，用户想要远程登录(telnet)到 sanfran 时就可以简单地输入“telnet sanfran”。而由于 ga.hp.com 不包含在这个搜索列表中，所以想要远程登录（telnet）到“atlanta.ga.hp.com”时就不能简单地键入“telnet atlanta”，而需要键入完全的域名。


```
vi /etc/resolv.conf
search ca.hp.com    hp.com      <—— 在这里列出通常存取的域。
```

b 在/etc/resolv.conf 文件中增加“ nameserver ”条目。

客户端的解析器在进行主机名和 IP 地址解析时必须要知道使用的是哪个 DNS 服务器。在/etc/resolv.conf 文件中最多可以配置三个名称服务器；如果第一个域名服务器没有响应，解析器会自动尝试用第二个服务器；如果第二个域名服务器仍然没有响应，解析器会自动尝试用第三个服务器。

由于解析器会自动用 resolv.conf 文件中 DNS 服务器的排列顺序来查找域名服务器，可以使用如下的方法来达到负载均衡的目的：在一些主机上，将主服务器列在前面，在另外一些主机上将辅域名服务器列在前面。

```
vi /etc/resolv.conf
search ca.hp.com      hp.com    <---在这里列出通常存取的域
nameserver 128.1.1.1            <--- 在这里列出主域名服务器的 IP 地址
nameserver 128.1.1.2            <--- 在这里列出辅域名服务器的 IP 地址
search          ca.hp.com    hp.com
nameserver      128.1.1.1
nameserver      128.1.1.2
```

(2) 修改/etc/nsswitch.conf 文件。

多数 UNIX 系统都可以通过使用本地 hosts 文件、NIS、DNS 三种方式来解析主机名。/etc/nsswitch.conf 文件的作用就是用来决定使用哪个服务进行名称解析，如果没有/etc/nsswitch.conf 文件，系统会默认使用 DNS。如果有一个/etc/nsswitch.conf 文件，找到“ hosts ”这一行，确认 DNS 是第一个选项。

```
cat /etc/nsswitch.conf
...
hosts          dns nis files
...
```

一旦配置了/etc/resolv.conf 和/etc/nsswitch.conf 文件,解析器会立即开始使用 DNS 进行名称解析。

(3) 修改/etc/hosts 文件。

由于现在大多数主机名都使用 DNS 进行解析，可以选择删除/etc/hosts 文件中的大多数条目。但是，也可以保留一些关键条目以备在域名服务器失效时使用，至少需要保留 localhosts 条目和自己的主机名。

在主域名服务器上，应该保存本域中所有主机的信息；在使用 hosts_to_named 之前，确认/etc/hosts 文件中的主机名正确并且格式完整。同样也可以选择以别名方式保留非全格式的主机名。如在 la.ca.hp.com 上，修改后的 hosts 文件如下：


```
# vi /etc/hosts
```

```
127.0.0.1    localhost
128.1.1.3    la.ca.hp.com    la
```

(4) 修改~/.rhosts，/etc/hosts.equiv 和其他文件。

有些工具需要将接收到的数据包中的 IP 地址反向解析为域名。如果以下的文件存在，这些文件中应该包含完全格式的域名：

```
~/.netrc
/etc/hosts.equiv
/var/adm/inetd.sec
```

例　主机 la 上的.rhosts 文件包含如下信息：

```
#vi ~/.rhosts
oakland.ca.hp.com
sanfran.ca.hp.com
la.ca.hp.com
```

(5) 使用 nslookup 测试 DNS。

```
#nslookup
>server    128.1.1.1     #选择一个域名
>oakland.ca.hp.com       #将个主机名解析为一个 IP
>128.1.1.2               #解析一个 IP 地址为主机名
>exit
Name Server:    sanfran.ca.hp.com
Address:        128.1.1.1
Trying DNS
Name:           oakland.ca.hp.com
Address:         128.1.1.2
```

系统默认的配置是：如果 DNS 失效就从 DNS 切换为 NIS，如果 NIS 也失效，就从 NIS 切换为/etc/hosts。

如果配置了多种名称服务方式，并且允许在一种服务无效时自动切换使用另一种服务，这时可能出现的一种情况就是：对同样的请求，如果请求发生的时间不同，响应这种请求的服务不同，就会得到不同的结果。所以一定要确保所有的名称解析服务解析出来的结果是一致的。

注意　更改系统默认的名称服务查找配置会使查错变得复杂。

如果 swtrace 选项没有设置，nslookup 可能返回错误的数据源。


```
#nslookup la
Name Server: sanfran.ca.hp.com
```

```
Address: 128.1.1.1
Name :      la
Address: 128.1.1.3
#
```

但是如果使用 swtrace 选项可能会出现另外一种结果：

```
#nslookup
Name Server: sandran.ca.hp.com
Address: 128.1.1.1
>set swtrace
>la
Name Server: sanfran.ca.hp.com
Address: 128.1.1.1
lookup source is DNS
*** No address (A) record available for la
switching to next source in the policy
lookup source is NIS
Default NIS Server: oakland
Address: 128.1.1.2
Name: la
Address: 128.1.1.3
>
```

7. DNS 启动

- 启动文件 named.boot

named.boot 必须放在/etc 目录下,启动文件告诉域名服务器数据库文件的存放位置和指定的域使用哪种类型的域名服务器。

数据文件的位置是可配置的，建议创建一个单独的目录，目录名字可以随意选择。

注意　域名 cache 数据存放在内存中，而不是存放在 db.cache 文件中，这个文件仅仅被用来指明根服务器的 IP 地址。

辅域名服务器上的 named.boot 文件和主服务器上的非常接近，只是在关键词 primary 的地方用 secondary 代替，同时 IP 地址为辅域名服务器的 IP 地址。

named 守护进程在启动时会重新读取服务器的数据文件。所以无论对数据文件做了何种修改，都应该重新启动该项进程服务。

重启动服务的命令：kill -9 named_pid 首先停止该项进程，然后再用启动命令：/etc/named & 启动。

- 加载 DNS 数据文件

当系统的运行模式为 2 或者更高时，/sbin/rc2.d 中的一个启动脚本就会执行。这个脚本链接到运行脚本 /sbin/init.d/named 。这个脚本从 /etc/rc.config.d/namesvrs 中获取适当的变量。域名服务器用/usr/sbin/named 命令启动。这是一个后台运行的守护进程。当 named 被激活时，它会从启动文件 /etc/named.boot 中找到数据库文件的位置。

NAMED_ARGS 变量很少使用。启动域名服务的语法和选项如下：

```
named [ -d debuglevel] [ -p port_number] [ -b bootfile]
```

其中选项：

-d：显示调试信息。-d 后面的数字决定信息显示的级别。调试输出信息被重定向到/var/tmp/named.run 文件中。

-p：使用另外一个端口号。

-b：使用另外一个启动文件而不是/etc/named.boot。

在任何时候，都可以使用运行脚本来手工停止或者重启动 named。

```
#/sbin/init.d/named stop
#/sinb/init.d/named start
```

注意　named 进程只允许在 BIND 服务器上运行，而不能在 BIND 客户端运行。

11.6.2　Web 服务

WWW 技术的迅速发展使得 Web 已不仅仅是一种信息传播的手段。Web 已经成为具有为用户提供应用、数据库、多媒体及软件管理的一个重要平台。

Web 服务主要包括：Web 服务器、网络和 Web 客户。

通过 Web 服务器可以在 Internet 上发布信息。Web 服务器的作用是为客户请求提供 URL 应答。Web 网页的内容可以是静态的，也可以动态生成。通常用 HTML 语言编写静态网页，用 CGI 为 Web 服务器添加动态内容。

1. HTTP 协议

HTTP 协议称为超文本传输协议，RFC2616 定义其 1.1 版本，是使用分布合作式超媒体信息系统的应用层协议。

Internet 上的所有 HTTP 传输都通过 TCP 实现，默认情况下服务器通过端口（服务）80 满足 HTTP 请求。HTTP 请求由客户程序发出，如可以通过浏览器 IE 发出，服务器向客户回送应答。

Web 浏览器用 GET 命令发出请求，Web 服务器用 POST 命令应答，返回用户请求的 URL。如果服务器能够成功地识别出 URL 所制定的资源，则向 Web 浏览器送回“text/html”应答。

2. URL

URL 的全称为统一资源定位器，通过所有的协议定位网络资源。

URL 的语法为：[protocol://]host[:port][/path]

例如：http://202.113.2.53:8080/proxy/list.txt

http://www.scu.edu.cn

其中 protocol 为应用层协议名；host 表示所有能获得其 IP 地址主机的主机名，Web 客户机根据客户机系统的具体配置，通过 DNS 服务器或 hosts 文件获得 IP 地址信息；port 指定了服务器接受外部请求的 TCP 端口，如果协议为 HTTP，未指定 port 值，则默认为 80。

被访问资源的惟一标识符是相对文件根的路径，文件根是服务器上含有相应 Web 内容的位置。如 Web 服务器以/usr/local/web 为文件的根，而 Web 客户机中则是目录/。所有文件都通过相对文件根的位置进行访问。Web 服务器不允许任何客户机访问文件根之上的目录树。

通常情况下 URL 和 URI 可以交换使用。

3. Web 服务器配置

以 Apache Web Server 为例，在运行 Web 服务之前，必须先配置好服务器配置文件 httpd.conf、access.conf 和 srm.conf。

- 服务器属性配置文件 httpd.conf

httpd.conf 文件含有服务器的通用属性，如服务器类型、http 端口、服务名、注册数据位置、虚拟主机配置等信息，Web 服务器启动时首先分析 httpd.conf 文件。文件 httpd.conf 位于/ust/local/httpd 目录中，而运行 httpd.conf 的错误记录和传输记录则存放在/usr/local/httpd/logs 目录下的文件 error_log 和 transfer_logs 中。

httpd.conf 文件的样本格式为：

```
#cat httpd.conf
#
# ServerType is either inetd, or standalone
ServerType standalone
```

```
# Port
Port 80

# ServerName
ServerName www.scu.edu.cn

# Web usee,Web group
User web
Group web

# ServerAdmin：E-mail address
ServerAdmin webmaster@scu.edu.cn

# ServerRoot:The directory of the server’s config,error, and log files are
kept in
ServerRoot /usr/local/httpd

# Errorlog:The location of the log files
ErrorLog logs /error_log

# TransferLog
TransferLog logs /access_log

# PidFile:The file the server should log its pid to
PidFile logs /httpd.pid

# ScoreBoardFile:File used to store internet server
ScoreBoardFile logs /status

HostnameLookups off
Timeout 300
KeepAlive On
```



```
MaxKeepAliveRequests 150
KeepAliveTimeout 20
```

```
StartServers 5
Maxclients 200
```

从文件可见：

服务器的类型为 standalone，服务端口号为 80，服务器名为 www.scu.edu.cn。

Web 服务器可以作为用户 Web 和组 Web 运行，为了安全，不能以超级用户身份运行。

所有与 Web 服务器相关的问题都应以 E-mail 方式报告给服务器管理员,地址为 webmaster@scu.edu.cn。

KeepAlive On 表示服务器打开执行连续的 HTTP 请求，建立单一的 TCP 或永久的连接，这样的连续请求减少了打开和关闭每个 HTTP 请求套接字的开销，提高了 Web 服务器的性能。

KeepAliveTimeout 20 指定服务器在每个客户机发出连续请求之间等待 20 秒来维持永久 TCP 连接。超过此限的任何一条请求将建立一条新的 TCP 连接。

MaxKeepAliveRequests 150 表示客户机每次发出请求建立 HTTP 连接的个数为 150。

Maxclients 200 表示服务器为了防止过载，限定在同一时间连接到服务器上的客户机数为 200。

- 资源配置文件 srm.conf

文件 srm.conf 可以设置和修改 URL 的文件根位置、服务和结果格式等项目。srm.conf 文件的样本格式为：

```
# DocumentRoot: By default, all requests are taken from this directory, but
symbolic links
# and aliase may be used to point other location.
DocumentRoot /web

# DirectoryIndex:Name of the file or files to use as a pre-written HTML
# directory index. Separate multiple entries with spaces.
DirectoryIndex index.html default.html greetings.cgi

# FancyIndexing is whether you want fancy directory indexing or standard
FancyIndexing on
```



```
# AddIcon tells the server which icon to show for different files or filename
extensions.
```

```
        AddIconByType (TXT, /icon/text.gif) text /*
        AddIconByType (IMG, /icon/image2.gif) image /*
        AddIconByType (SND, /icon/sound2.gif) sudio /*
        AddIconByType (VID, /icon/movie.gif) video /*

        AddIcon /icons/binary.gif .bin .exe
        AddIcon /icons/binhex.gif .hqx
        AddIcon /icons/tar.gif .tar
        AddIcon /icons/world2.gif .wr1 .wri1.gz .vrml .vrm .iv
        AddIcon /icons/compressed.gif .Z .z .tgz .rz .zip
        AddIcon /icons/a.gif .ps .ai .eps
        AddIcon /icons/layout.gif .html .shtml .htm .pdf
        AddIcon /icons/text.gif .txt
        AddIcon /icons/c.gif .c
        AddIcon /icons/p.gif .p1 .py
        AddIcon /icons/f.gif .for
        AddIcon /icons/dvi.gif .dvi
        AddIcon /icons/uuencoded.gif .uu
        AddIcon /icons/srcipt.gif .conf .sh .shar .csh .ksh .tcl
        AddIcon /icons/tex.gif .tex
        AddIcon /icons/bomb.gif core

        AddIcon /icons/back.gif ..
        AddIcon /icons/hand.right.gif README
        AddIcon ("DIR", /icons/folder.gif)  ^ ^DIRECTORY ^ ^
        AddIcon /icons/blank.dif ^ ^BLANKICON^ ^

        # DefaultIcon is which icon t show for files which do not have an icon explocotly
set.
        DefaultIcon /icons/unknown.gif

        # AddDescription allows you to place a short description after a file in server
generated
          indexes.
```

```
        # Format:AddDescription "description" filename

        # ReadmeName is the name of the README file the server will look for by default.
        # Format:ReadmeName name
        #
        # The server will first look for name.html, include it if found, and it will
then
        # look for name and include it as plaintext if found.
        #
        # HeaderName is the name of a file which should be prepended to directory
indexes.

        ReadmeName README
        HeaderName HEADER

        # IndexIgnore is a set of filename which directory indexing should ignore
        # Format: IndexIgnore name1 name2 …
        IndexIgnore .gif .jpg .??* *-*#HEADER* README* RCS

        # AccessFileName: The name of the file to look for in each directory for access
        # control information.
        AccessFileName .htaccess

        # DefaultType is the default MIME type for documents which the server cannot
        # find type of from filename extensions.
        DefaultType text/plain

        # Redirect allows you to tall clients about documents which used to exist in
your
        # server ' s namespace,but do not anymore. This allows you to tell the clients
where
        # to look for the relocated document.
        # Format: Redirect fakename url
        Redirect /subscriptions http://subscription s.mcp.com
```

```
        # Aliases:Add here as many aliases as you need(with no limit). The format is
Alias
        # fakename realname. Notes that if you include a training / on fakename then
the server
        # will require it to be present in the URL. So "/icons" isn ' t aliased inthis
example.
        Alias /icons /opt/web/icons
        Alias /applets /opt/web/applets

        # Scriptalias:This controls which directories contain server scripts.
        # Format:ScriptAlias fakename realname
        Scriptalias /cgi.bon /opt/web/cgi.bin

        # If you want to use server side includes, or CGI outside ScriptAliased
directories,
        # uncomment the following lines.

        # AddType allows you to tweak mime.types without actually editing it, or to
make certain
        # files to be certain types. Format:AddType type /subtype ext1

        # AddHandler allows you to map certain file extensions to "handlers" , actions
unrelated
        # to filetype. These can be either built into the server or added with the Action
        # command. Format:AddHandler action name ext1

        # To use CGI scripts:
        AddHandler cgi.script cgi

        # Customizable error response
        # these come in three flavors
        #  1) local redirects
        ErrorDocument 404 /cgi.bin/missing_handler.p1
```

```
# n.b. can redirec to a script or a document using server side-includes.


#  2)external redirects
ErrorDocument 402 http://subscription.map.com/subscription_info.html
```

从文件可见：

DocumentRoot 指定了 Web 查找被请求的 URL 的具体位置。

DirectoryIndex 指定了一些可能用到的索引文件。该文件首先运行 index.html 文件，如果运行失败，则运行 default.html 文件，如果再次失败则运行 greetings.cgi 文件。

FancyIndexing 指定可以打开或关闭状态。如果为打开状态，则允许显示完整的目录信息。

AddIcon 指定了在显示具有指定扩展名、目录或空格行的文件时所用的图像。

Alias 用于服务器寻址 Web 用户以别名存放内容的文件系统，即使 DocumentRoot 指向 /web，URL http://www.mcp.com/images/txt.gif 也会使 Web 服务器从 /opt/web/images 目录读文件。

Redirect 用于将浏览器/客户机重定向到 Internet 上的另一个 Web 站点，即 http://www.mcp.com/subscriptions 的相应内容被重定向到：http://subscriptons.mcp.com。

Actions 用于为给定的扩展名制定新的动作。在文件之前根据文件扩展名写出响应文件的 CGI 脚本。

除上面介绍的之外，在配置文件的最后还提供了用来处理由服务器返回的错误信息。

- access.conf 文件

access.conf 文件指定了实现某些类型服务允许访问服务器的环境，为每个目录设定了访问控制。access.conf 文件的样本格式为：

```
# Set up access for our DocumentRoot
<Directory /web>


#This may also be "none" ,"All", or any combination of "Indexs",
#"include" ,"FollowSymLinks" ,"ExecCGI" ,or "MultiViews".


#Note that "Multiviews" must be named "explicitly"
```



```
Options Indexes FollowSymLinks
```

```
        #This controls which options the .htaccess files in directories can override.
Can also be
        #"All" ,or any combination of "Options" ,"FileInfo" ,"AtthConfig" ,and "Limit"
        AllowOverride None

        # Controls who can get stuff from this server.
        Order allow, deny
        Allow from all

        </Directory>

        #CGI directory exists, if you have that configured.

        <Directory /opt/web/cgi.bin>
        AllowOverride None
        Options None
        </Directory>

        #Allow server status reports, with the URL of http://www.mcp.com/server-status
        <Location /server-status>
        SetHandler server-status
        Order deny,allow
        Deny from all
        Allow from .mcp.com
        </Location>
        <Location /cgi-bin/phf*>
        deny from all
        ErrorDocument 403
        </Location>
```

文件中 allow from 和 deny from 用于指定 IP 地址、主机名或主机名组的访问或拒绝访问。order 用于指定这些命令的处理顺序，当允许大多数用户访问时会允许使用 order allow deny，否则使用 order deny allow。order allow deny 说明只有当客户匹配 allow from 所定义的标准时，才能允许对服务器资源的请求。如果客户名出现在 allow from 表中，则允许客户的访问。如果客户名字出现在 deny from

表中或不在 allow from 表中，则拒绝访问。

4. 只用一个配置文件的 Web 服务

前面介绍用三个文件 httpd.conf、srm.conf 和 access.conf 配置 Web 服务，也可以用一个配置文件 httpd.conf 配置 Web 服务，但要在文件 httpd.conf 中加入：

```
AccessConfig /dev/null
ResoureConfig /dev/null
```

5. 启动和终止 Web 服务

启动 Web 服务用命令：

```
#httpd -d /usr/local/httpd
```

终止 Web 服务用命令：

```
#cat cat/usr/local/httpd/logs/httpd.pid
522
#kill -TREM 522
```

该 kill 命令向父 httpd 进程发出 TERM 信号立即杀死全部 httpd 子进程及自身。

11.6.3　邮件服务

电子邮件是 Internet 中的主要应用。

电子邮件从发送端计算机发送后要借助于邮件服务器才能到达接收端计算机。发件人所在的邮件服务器通过 sendmail 程序接收和保存用户邮件，并将邮件发送到收件人所在的邮件服务器上，收件人再从自己的邮件服务器上下载邮件到自己的计算机。邮件服务器是整个邮件系统的关键，目前绝大多数邮件服务器由 UNIX 操作系统承担。

1. 简单邮件传输协议 SMTP

不同计算机系统的 E-mail 应用程序之间存在一定的差异，有的是操作系统提供的图形工具，有的是操作系统提供的命令工具，有的是操作系统之上的应用软件，但是这些 E-mail 应用程序都满足相同的标准，定义了 E-mail 相同的格式和传输协议，简单邮件传输协议（Simple Mail Transport Protocol:SMTP）则是这一类协议。

在邮件服务器之间的邮件通信标准是简单邮件传输协议 SMTP。最初的 SMTP 协议非常简单，主要定义普通文本格式的英文邮件，后来通过多用途的网际邮件扩充协议（MIME）和 SMTP 扩展，使得在邮件中可以使用多种数据类型文件，如汉字、图像和声音等。

SMTP 提供端到端传输，用于把邮件从一台机器转发到另一台机器，传输时寻址用“ 用户名@域名 ”类型，在源地址和目的地址之间建立通信信道，并通过该信道发送数据，完成邮件传输。在传输过程中，SMTP 将邮件头和邮件体进行封装发送。

SMTP 邮件传输协议在邮件的邮件头中包含如下信息。

- Return-Path：返回邮件发送者的路径，由最后一个传输节点将该消息加入邮件头中。
- Received：含有每个转发该邮件的服务器地址及其他信息，如每个传输节点收到邮件的日期和时间，这些信息可用于跟踪传输路径。
- Date：SMTP 服务器收到邮件的日期和时间。
- From：邮件发送者的地址。
- Subject：用户输入的邮件主题。
- Sender：邮件发送者。
- To：邮件接收者，可以是多个地址。
- Cc：接收转发来的邮件拷贝的用户地址。
- Error-to：当传输过程中发生错误时接收错误信息的地址。
- Message-ID：惟一识别邮件及其版本的标识符，由发送邮件的客户机产生。

2. POP 协议

POP 协议又称为邮局协议，是客户方的启动协议。用于接收端计算机从邮件服务器上接收新邮件。POP 的作用是每次检查邮箱中是否有新邮件，如果有，则用下载方式接收新邮件。POP 的作用正好与 SMTP 相反。

作为 POP 服务器必须运行 POP 驻守进程 popd 来处理客户连接接收邮件。popd 进程在收到客户的邮件接收请求后，检查用户邮件帐号和口令，如果正确则下载邮件并根据客户机的设置处理用户在服务器邮箱中的邮件，如删除或保留副本。

作为 POP 客户端应设置邮件服务器的位置、用户帐号名和口令。

3. sendmail 程序

sendmail 程序使用 SMTP 协议在 Internet 上传输邮件。UNIX 邮件服务器上运行程序 sendmail，用于接收和保存用户的邮件，并将发送队列中的邮件发送到收件人所在的服务器上。

sendmail 进程在处理邮件时首先分析收件人的 E-mail 地址以确定并分离收件人的用户名和域名；然后根据收件人的域名查找目的域邮件服务器。sendmail 的配置借助于工具实现，典型的工具有 M4 宏处理器。

sendmail 软件的发布含有自动创建 sendmail.cf 文件的全部 M4 宏文件，M4 宏

处理器根据用户配置的参数创建一个 sendmail.cf 文件。通常将 sendmail 的发布放在/opt/src 目录下。下面是/opt/src 下用于创建文件 sendmail.cf 的 mail-hub.mc 文件。

```
#cat /opt/src/sendmail.8.8.6/cf/cf/mail-hub.mc
#VERSIONID is a macro that stuffs the version information into the sendmail.cf file
VERSIONID( ' @(#)Learnix Sendmail May 3, 1998 ' )

#You must specify an OSTYPE to properly configure things such as the
#pathname of the help and status files, the flags needed for the local mailer,
#and other important things
OSTYPE (AIX) dn1

#This example is specific to the pubbook.cn domain.
DOMAIN(pubbook.cn)dn1

#These describe the mailer that will be supported by the pubbook.cn domain
MAILER(local)dn1
MAILER(smtp)dn1
#
```

下面用 M4 命令创建 sendmail.cf 文件：

```
/usr/ccs/bin/m4 ../m4/cf.m4 mail-hub.me>sendmail.cf
```

得到了 sendmail.cf 文件后还要将 sendmail.cf 放在目录/etc/mail 下，并重新启动进程 sendmail：

```
#/etc/mail/sendmail [-option]
```

选项 option 可以为：

v：表示冗长模式。

bd：表示在 Daemon 模式下运行。

bi：表示初始化别名数据库。

bm：表示邮件发送者。

bp：表示打印邮件队列。

bs：表示将 SMTP 发送到标准输出。

bt：表示地址检测模式。

q lh：表示每隔一小时重新发送。

d n：表示调试模式，可用于检测与恢复故障，参数 n 的值及含义为：

0：通用调试

1：显示发件人信息

3：打印平均负载

4：足够的磁盘空间

5：显示事件

6：显示失败的邮件

7：将文件名排队

8：DNS 域名解释

9：跟踪 RFC1413 询问

10：显示收件人传输情况

11：显示所选的传输代理

12：显示相关的主机映像

13：显示传输

14：显示邮件头域信息

15：显示网络获得的请求活动

16：向外连接

17：列出 MX 主机

18：显示 SMTP 应答

19：显示 ESMTP 参数

20：显示解析传输代理

21：跟踪重写规则

22：跟踪地址符号化

27：跟踪别名

41：显示配对失败原因

例 #/etc/mail/sendmail -d 0

```
#/etc/mail/sendmail -d 1
#/etc/mail/sendmail -d 2
```

sendmail 配置文件为 sendmail.cf，该文件格式为：

```
#cat sendmail.cf
O AliasFile=/etc/mail/aliases
O QueueDirectory=/var/spool/mqueue
O DeliveryMode=background
O TempFileMode=0600
```

```
O HelpFile=/etc/mail/sendmail.hf
O SmtpGreetingMessages=$j Sendmail $v$z; $b
O StatusFile=/etc/mail/sendmail.st
#
```

选项 option 用于确定 sendmail 参数值，用大写字母“O”设置。

如果 UNIX 系统没有 sendmail 程序，可以从站点：www.sendmail.org 免费下载。

11.6.4　FTP 服务

文件传输协议（FTP）是 UNIX 中最常用的网络服务之一。设置匿名服务器相当简单。而设置虚拟 FTP 主机和单独的 FTP 帐号则需要一些技巧。

1. 匿名 FTP

匿名 FTP 是最常见的服务，一台机器就是一个单独的 FTP 服务器。大多数 UNIX 版本会自动完成这一设置。通常 FTP 的根目录为/home/ftp。如果机器上未设置匿名 FTP 服务，则需要设置，步骤如下。

(1) 在/etc/passwd 文件中指定匿名 FTP 用户及 FTP 用户主目录。

FTP 服务的守护进程 ftpd 会识别出匿名用户 FTP，系统将 FTP 看成一个普通用户，将访问目录设置为/etc/passwd 文件中指定的匿名 FTP 用户的主目录，通常情况下为/home/ftp 目录。FTP 目录的属主应为 root 而且只有 root 才能写入。同样/home/ftp/bin 的属主也应为 root 而且只有 root 才能写入。它其中应该包含 ls 程序。/home/ftp/bin/ls 的属主应为 root，其访问权限应该为--x--x--x 模式，如果不是这样，可以用 chmod 111 /home/ftp/bin/ls 修改。/home/ftp/lib 应该包含 libc.so.5。这些内容可以在/lib 目录下找到。

如果打算用列表将用户和组 ID 翻译成名字，则需要创建/home/ftp/etc 目录。它应当具有 755 访问权限，并且还应该包含名字和 ID 相关联的 passwd 和 group 文件。FTP 的口令字段并未使用，应当置为空。惟一需要存在的字段是 username、UID 和 GID。

上载和下载目录/home/ftp/pub 应该具有 755 访问模式，并且属主应为 FTP。这样才能允许某些用户向目录中上载内容或从目录中读取内容。当然也可以按自己的需要修改所有权和访问权限。

FTP 用户在/etc/passwd 中的条目应为：

```
ftp:*:14:50:FTP user:/home/ftp:
```


FTP 是由 inetd 控制的服务，因此在/etc/services 中也要有一个对应项。一般都设置好了，如果没有，可以手工加入一项：

ftp 21/tcp

(2) 系统中其他用户的 FTP 访问。

当其他用户帐号（不是匿名 anonymous 或 FTP）连接系统时，必须满足三个条件才能授予访问权限：

- 用户名和口令必须有效；
- 用户名必须不在/etc/ftpusers 中；
- 用户必须有一个有效的 Shell，即 Shell 必须出现在/etc/shells 列表中。

2. 设置仅可以进行 FTP 连接的帐号

设置用户帐号仅能进行 FTP 连接是可能的。这些帐号类似于匿名帐号并且能够与之共存。由于根目录已被重置，需要设置 bin、lib 和 etc 目录，设置的方法同在匿名帐号中设置一样。

要设置特定的帐号仅可以进行 FTP 访问需要编辑/etc/ftpaccess 文件。

例 设置两个用户，ftpbob 和 jane 作为只能使用 FTP 功能的用户。编辑文件/etc/ftpaccess 成为：

```
#cat /etc/ftpaccess
#
#/etc/ftpaccess
#
class all real,guest,anonymous *
class ftponly ftpbob,jane *
loginfails 5
readme README * login
readme README * cwd= *
message /welcome.msg login
message .message cwd= *
compress yes all
tar yes all
overwrite yes real
chmod no guest,anonymous
delete no guest,anonymous
rename no guest,anonymous
overwrite no guest,anonymous
guestgroup ftponly
```

```
#
```

class 行设置用户组和主机匹配模式以匹配远程主机。“ * ” 匹配所有主机。

相应的/etc/passwd 项为：

```
jane:9pthxXoQVw:518:518:jane's FTP-only Acount:/a/ftp/jane:/bin/false
ftpbob:l8Leijpehfp:518:518:Bob's FTP-only
Acount:/a/ftp/ftpbob/./incoming:/bin/fales
```

这两个用户的 Shell 都是有效的 Shell，并且还会拒绝 Shell 访问。主目录将变成 FTP 会话的根目录。除此之外，ftpbob 帐号会修改目录到/a/ftp/ftpbob/incoming。

注意　ftpaccess 文件还设置了其他一些参数：

real 是任何有效帐号的关键词。

anonymous 代表任何匿名用户。

guest 指 guest 级别的访问帐号。

compress 和 tar 项通知 ftpd 何时允许动态的压缩和解压缩，存盘和恢复（tar/untar）。

README 和 messages 项设置文件搜索模式以便在连接和进入目录（README）时自动显示信息。

在本例中，低级访问帐号（anonymous 和 guest）不能删除文件、修改文件模式或重命名文件。

Loginfails 完成期望的工作，在指定次数的登录尝试失败后放弃连接。

11.7　NIS

11.7.1　NIS 概念

NIS 为网络信息服务，是一种通过网络提供多目录服务的 RFC 应用层服务。

在网络中用户经常在不同服务器上使用 telnet、FTP、E-mail 等网络服务，而每台服务器都必须用相应的用户名及口令才能登录，这样就存在要维护大量用户名及口令的问题。NIS 提供对网络资源的集中存储，使用 NIS 可以使整个网络共享一个单一的中心用户文件，同一用户对所有计算机的访问只需一个单一输入项，简化了用户使用的繁琐。

UNIX 支持网络文件系统（NFS），NFS 服务将 NFS 服务器装载到客户文件系统上工作。网络上的多台计算机组成一个 NFS 子网络，每个 NFS 子网络中有主服务器。通过 NIS，可以将多个 NFS 子网络映像到一个 NIS，使用一个 NIS 主服务器。在 NIS

主服务器中每个 NFS 子网络占一项并保持不同的访问权限，与/etc/passwd 中的用户管理相似。NIS 各项主要的访问权限有：项目所有者、项目组所有者、其他用户，但是这种访问权限与传统的文件系统读、写和执行权限不同。NIS 主服务提供查询、传输主文件的拷贝、访问用户文件和系统管理。

NIS 系统也是一个利用 TCP 或 UDP 方式进行通信的客户机/服务器模式的分布式访问系统，网络上每台使用 NIS 的客户计算机都访问 NIS 主服务器。为了预防在网络上出现传输及意外事件，主服务器发生错误后客户机仍然可以访问，同样也配置了 NIS 从服务器。NIS 不限制系统用户，任何文件都可以安装使用 NIS。

11.7.2 使用 NIS

在许多 UNIX 系统中都支持 NIS，如 Solaris2.x 支持 NIS+（增强型的 NIS）。NIS+的管理命令主要有：

- nistladm：显示、添加、修改和删除 NIS+表中的信息；
- nisgrep：在 NIS+表中搜索信息；
- nismatch：在 NIS+表中搜索信息；
- niscat：显示 NIS+表中的所有内容。

NIS 通过 NIS 表管理用户、工作站和网络资源。NIS+表及表中的信息为：

- host 表：管理 NIS 域中主机的网络地址和主机名；
- bootparams 表：管理 NIS 域中每个无盘客户的根目录、交换目录和复制目录的位置；
- password 表：管理 NIS 域中每个 NIS+主服务器的密码信息及指向隐藏文件的指针；
- cred 表：管理主服务器访问域中的信息和客户的权限凭证；
- group 表：域中组的密码、组 ID 及组成员；
- netgroup 表：整个网络中的所有组；
- aliases 表：域中工作站的别名信息；
- timezone 表：域中工作站的时区；
- networks 表：域中网络及其规范名；
- netmasks 表：域中网络及其屏蔽；
- ethers 表：域中每个工作站的以太网地址；
- services 表：域中使用的 IP 服务器和它们的端口号；
- protocols 表：域中使用的 IP 协议列表；

- RPC 表：域中可用的 RPC 服务的 RPC 程序个数；
- auto_home 表：域中所有用户的主目录位置；

- auto_master 表：自动加载程序映像信息。

例　如果要查找名为 micky 的工作站的 IP 地址，可以利用 nisgrep 命令查找表 host，得到该工作站的 IP 地址。

```
#nisgrep micky host
micky micky 10.0.0.1 micky's comment
```

11.8　本 章 小 结

没有 UNIX 就没有网络，或者说没有网络就没有 UNIX，可见它们的关系非常紧密。

现在的 Internet 中，TCP/IP 协议无所不在。而早期的网络技术就是根据 UNIX 系统的需求及功能发展起来的。由于 UNIX 系统具有运行稳定、性能可靠、系统安全及良好的网络环境等特点，因此成为构建网络环境主机的首选平台。

本章着重介绍了 TCP/IP 协议在 UNIX 系统中的应用及实现。涉及到了 UNIX 系统的网络概念、网络地址配置、网络通信状态监控、路由协议及其原理，以及基于网络的 TCP 服务：域名服务、Web 服务、邮件服务、FTP 服务等。

本章中最核心的内容是网络服务器的配置及网络管理环境。

上 机 练 习

1. 熟悉各种网络相关的配置文件，查看一台主机上对应的配置文件，并理解其内容。
2. 练习 ifconfig、netstat 命令的用法。
3. 观察一台主机的路由表，并理解其含义。
4. 练习怎样将一台 UNIX 主机配置成 Web 服务器。
5. 请说明配置 FTP 服务器需要哪些步骤。

习 题 十 一

1. 请在自己的 Linux 系统下建立网络连接。

2. 请说明建造 UNIX 系统的网络平台时以下文件：/etc/hosts、/etc/networks、/etc/hostname 中存放什么内容？有什么作用？

3. 请说明主 DNS 服务器和辅 DNS 服务器在配置上有什么差别？

4. 在 UNIX 系统的网络通信中可以使用哪几类协议？各有什么作用？

5. 请说明系统怎样管理邮件服务器的邮件用户。

第 12 章　UNIX 安 全

在网络环境中，作为网络服务器的 UNIX 系统的安全性非常重要。UNIX 系统的安全性应该首先体现在系统设计上要符合安全性策略,其次是操作系统本身的安全，另外还要从网络环境和信息安全方面保证 UNIX 系统的安全。

本章的主要内容

- 安全性策略：介绍系统的安全策略。
- 操作系统安全：操作系统安全主要包括用户登录安全、文件安全、系统日志、进程统计日志、syslog 服务、遭到网络攻击需要采取的措施、Solaris 的基础安全模块 BMS、系统服务。
- 防火墙：防火墙部分主要包括包过滤技术、应用网关、代理服务器。

12.1　安全性策略

UNIX 主机在网络中承担着重要的网络服务功能，因此 UNIX 主机系统的安全非常重要。目前系统的安全策略主要从以下几个方面考虑 UNIX 系统的安全。

(1) 整个网络系统的安全：取决于 UNIX 服务器、网络上的工作站、网络通信设备等设施的安全。

(2) 用户安全：采用授权用户访问，每个访问的用户使用注册帐号，通过用户注册帐号管理用户文件的存取控制和用户进程运行。

(3) 系统管理员安全：要求系统管理员具有较高的管理素质，包括系统管理技术及个人修养素质。

(4) 操作系统的安全：系统配置不当，系统文件默认配置可能会带来一些安全隐患。

(5) 软件安全：系统中运行的守护进程、应用程序进程以及网络协议等软件都可能存在安全漏洞。

(6) 保持系统的日志记录：系统管理员通过日志记录可以查看历史记录，有助于识别系统事件发生的原因和情况。

(7) 合法的软件使用权限：限制下载不合法的软件，保证软件运行不对系统造

成病毒和危害。

(8) 系统设计的缺陷：网络协议 TCP/IP 本身没有考虑安全，因此要在提供网络服务的情况下采取防火墙技术保证 UNIX 服务器的安全。

12.2 操作系统安全

从 UNIX 操作系统考虑，安全包括：

- 系统保密：防止系统内信息的非法泄漏；
- 系统完整：防止系统软件程序和数据被非法删除和破坏；
- 系统有效：系统资源权限有效，授权用户才可以随时存取资源。

操作系统用户使用 login 登录系统，getty 进程监控用户的登录终端，系统通过用户帐号识别用户并赋予用户相应的访问权限，保证系统资源有效。在用户使用期间通过用户审核确定用户是否对系统造成破坏，保证系统的完整。

12.2.1 用户登录安全

在 UNIX 下 login 程序成为将非法用户排除在计算机之外的第一道防线。login 程序根据文件/etc/passwd 确定是否允许用户进入系统并通知操作系统用户具有的权限。/etc/passwd 是系统所有用户信息的 ASCII 文本文件，其中包含有加密的用户口令及用户默认的 Shell、用户家目录。系统中的所有用户都可以读/etc/passwd 文件,这样极不安全,因为 Internet 上有许多黑客程序能破解该文件中的加密口令。

Solaris 系统把用户密码放在另一个文件/etc/shadow 中，由于/etc/shadow 文件不是所有用户都可读的，提高了系统的安全性。

通常情况下系统应确保每个用户有一个用于登录的帐号，从安全考虑不应使用通用帐号。如果多个用户要使用相同的帐号登录系统则建立组帐号。另外每个用户帐号应该使用一个好的口令，避免使用生日、家人名字、门牌号、电话号码及常用术语等。

在网络上窃听用户登录信息非常容易。在广播局域网上用户只要将网卡配置成“混杂”模式（Miscellaneous）就可以监听网络上的所有数据包；如果在服务器上安装窃听软件“sniffer”就可以得到远程用户的帐号和口令。所以应尽量避免在网络上使用允许明文传输用户名和口令的协议。

12.2.2 文件安全

UNIX 系统中系统配置文件的安全非常重要，应该将这些文件维护好。这些文件是：

- 帐号及口令文件/etc/passwd；
- 主机列表文件/etc/hosts；
- 网络列表文件/etc/networks；
- 协议列表文件/etc/protocols；
- 主机信任列表文件/etc/hosts.equiv。

12.2.3 系统日志

在 UNIX 系统中提供了多种系统安全日志。

- lastlog（对应文件为/var/adm/wtmp）：记录每个用户最近的成功登录和不成功登录的信息。
- loginlog（对应文件为/var/adm/acct）：记录所有不成功登录信息。
- utmp(x)（对应文件为/var/adm/utmp(x)）：记录当前登录的所有用户信息。用于跟踪系统已登录的用户名、用户终端、连接时间、系统关机信息。用 who 命令可以查看文件中的内容。
- wtmp(x)（对应文件为/var/adm/wtmp(x)）：记录每个用户登录退出的时间。用于跟踪系统所有注册用户及用户组、显示用户名、用户终端、连接时间、系统关机信息。用 last 命令和 ac 命令可以查看文件中的信息。
- sulog（对应文件为/var/adm/sulog）：记录切换用户命令（su）的使用情况。
- pacct（对应文件为/var/adm/pacct）：记录用户执行的命令和资源的使用情况。
- lpd（对应文件为/var/adm/lpd-errs）：记录打印机问题。
- cron（对应文件为/var/log/cron）：记录 cron 程序的使用。
- nis.trans（对应文件为/var/nis/trans.log）：记录 NIS 名称空间的更改。

例 登录日志。

用 who 命令查询 utmp 文件并显示当前登录的所有用户：

```
#who
root   tty1      Jul 21 19:39
```

w 命令提供的信息比 who 更加丰富：

```
#w
18:41:27 up 12 days, 23:02,  2 users,  load average: 0.97, 0.72, 0.66
USER    TTY     FROM          LOGIN@   IDLE   JCPU   PCPU   WHAT
root      tty1      -                  21Jul03  2days  48:18  48:18   top
```


用 users 命令提供最简单的登录用户查询功能：

```
#users
root
```

用 last 命令查询 wtmp 文件：

```
#last
ftp      ftpd6473     10.2.3.1     Sun Aug  1 12:39 - 12:41  (00:01)
ftp      ftpd10272    10.2.3.13    Sat Aug  2 16:02 - 16:20  (00:17)
ftp      ftpd9359     10.2.1.254   Sat Aug  2 15:20 - 15:27  (00:06)
......
```

用 ac 命令显示每天的总连结时间：

```
#ac -d
Aug 1 total 270.45
Aug 2 total 104.29
Today total 179.02
```

用 ac 命令显示每个用户的总连接时间：

```
#ac -p
root 193.23
ftpadmin 3.35
webadmin 133.40
```

12.2.4 进程统计日志

UNIX 可以记录用户所运行的每条命令。对于跟踪入侵来说有很大的帮助。与登录日志不同，进程统计子系统在默认情况下不启动，要由超级用户 root 通过手工方式启动，启动命令为：

```
#accton filename
```

其中 filename 为指定的日志文件名，必须在执行该命令之前存在。

用命令 lactcomm 或 sa 查看进程统计日志。如果要关闭该进程统计日志，用命令：

```
#accton
```

例 打开进程日志：

```
#accton acctlog
```

关闭进程日志：

```
#accton
```


查看进程统计日志：

```
#lastcomm acctlog
```

```
sa              root     stdin      0.00 secs Tue Feb 22 10:45
ls              root     stdin      0.00 secs Tue Feb 22 10:45
ls              root     stdin      0.01 secs Tue Feb 22 10:45
accton    S     root     stdin      0.00 secs Tue Feb 22 10:45
#
#sa
   6       0.01re       0.00cp         0avio       287k
   4       0.00re       0.00cp         0avio       288k    ***other
   2       0.00re       0.00cp         0avio       285k    ls
```

12.2.5　syslog 服务

UNIX 系统提供了通过 syslogd 守护进程实现的 syslog 通用服务，任何程序都可以用 syslog 记录事件。除了记录本地事件外，还可以通过网络记录另一个主机上的事件。进程 syslogd 也可以将系统事件输出到文件或设备，syslog 提供日志系统的简单接口，在程序中嵌入相应的日志记录代码，syslog 记录事件信息实现针对进程的入侵检测功能。

系统引导时按/etc/syslog.conf 文件中的配置执行 syslogd 守护进程，也可以用 syslog 命令指定配置文件，命令格式为：

```
/etc/syslog [-mN] [-f filename] [-d]
```

其中选项：

m：设置时间间隔标记，默认值为 20 分钟；

f filename：用于指定除/etc/syslog.conf 文件外的配置文件；

d：调试。

```
#vi /etc/syslog.conf
#
# Copyright (c) 1991-1993, by Sun Microsystems, Inc.
#
# syslog configuration file.
#
# This file is processed by m4 so be careful to quote (' ') names
# that match m4 reserved words.  Also, within ifdef's, arguments
# containing commas must be quoted.
#
*.err;kern.notice;auth.notice                   /dev/console
```

```
        *.err;kern.debug;mail.crit,daemon.notice        /var/adm/messages
        #
        # Log all TCP Wrapper connections
        #
        local3.info                                     /var/adm/tcpdlog
        *.alert;kern.err;daemon.err                      operator
        *.alert                                          root

        *.emerg                                          *

        # if a non-loghost machine chooses to have authentication messages
        # sent to the loghost machine, un-comment out the following line:
        #auth.notice                            ifdef('LOGHOST' , /var/log/authlog,
@loghost)

        mail.debug                        ifdef('LOGHOST' , /var/log/syslog, @loghost)

        #
        # non-loghost machines will use the following lines to cause "user"
        # log messages to be logged locally.
        #
        ifdef('LOGHOST' , ,
        user.err                                         /dev/console
        user.err                                         /var/adm/messages
        user.alert                                        'root, operator'
        user.emerg                                       *
        )
```

文件/etc/syslog.conf 中每项占用一行，分为三个域，分别是 selector、tab、action。一行中可以有多个 selector 域，相互之间用分号隔开。

每个 selector 域又分为 facility（表示发出消息的设备）和 severity（重要性级别）。表示为：facility.severity。可用的重要性级别有：

emerg：紧急情况；

alert：需引起立即注意的情况；

crit：对紧急情况的警告；

err：除 emerg、alart、crit 之外的其他错误；

warning：警告信息；

notice：需要引起注意的地方，重要性上比 warning 和 err 低，主要包括不必要的错误情况；

info：信息；

debug：在调试模式下运行程序所产生的信息；

none：隐藏该项的信息。

/etc/syslog.conf 文件中的 action 域用于通知守护进程 syslogd 将消息发往何处：

- 如果消息发往文件则文件名用“ / ”字符开头，发来的消息附在文件的尾部；
- 如果消息发往另一台主机，则主机名以“ @+主机名 ”表示，发来的消息给接收主机的守护进程 syslogd；
- 如果消息发给多个用户，用户名之间用逗号隔开，发来的消息给用户登录的终端；
- 如果 selector 域中含有字符“ * ”，则表示发来的消息给所有登录到系统中的用户。

在/etc/syslog.conf 文件中用以下字符表示所用的设备：

auth：认证系统，如 login；

cron：cron 和 at；

daemon：系统守护进程；

kern：内核消息；

user：由用户应用程序创建；

mail：邮件系统消息；

lpr：行式打印机伪脱机系统；

news：USENET；

uucp：UUCP；

local0-7：为系统本地保留；

mark：时间戳消息；

*：以上除 mark 之外的所有值。

例　/etc/syslog.conf 文件配置。

守护进程 syslogd 将 cron 产生的所有消息发送给系统监视器：

```
cron.*  /dev/console
```


守护进程 syslogd 将邮件系统产生的所有消息发送给邮件系统的管理员 mailadmin：

mail.* mailadmin

将重要的消息发给另一台备份主机 server2（遇到网络黑客攻击时黑客所作的攻击操作会被另一台主机记下）：

kern.*;auth.*@server2.sccd.com.cn

12.2.6 遭到网络攻击及需要采取的措施

- 查看系统和网络日志文件

日志文件中记录了各种行为类型，如用户登录、用户 ID 改变、用户对文件的访问、授权及认证信息。入侵者通常会在系统日志文件中留下访问痕迹，查看日志文件能够发现入侵企图和入侵行为，采取相应的应急措施。

- 系统目录和文件的异常改变

文件系统中包含有所需软件、数据文件以及重要信息文件，对这些文件和目录的访问权限（创建、修改、删除）经常遭到网络黑客的修改和破坏。被改变以后必须恢复。

- 程序执行中的异常行为

作为网络服务器的 UNIX 系统中运行的程序有网络服务、用户启动程序以及特定的应用程序，这些程序对应着进程的执行。每个进程具有不同的访问权限、访问的系统资源和数据文件等。如果某个进程遭到黑客攻击则可能引起整个系统中其他进程的运行失效，导致系统失败。

12.2.7 Solaris 的基础安全模块 BSM

Sun 公司的 Solaris 是应用较广泛的 UNIX 系统，其内部提供了基础安全模块 BSM（Basic Security Mode）作为安全子系统运行，使得 Solarios 系统安全符合 TCSEC 标准中的 C2 级安全。

基础安全模块 BSM 提供了审计记录管理工具，用于对 BSM 进行管理和配置。BSM 包括的文件有：

/etc/security/bsmconv：用于使 BSM 功能生效；

/etc/security/bsmunconv：用于禁止 BSM 功能；

/usr/sbin/auditd：auditd 守护进程用于控制审计记录文件的产生及所处的位置；

/usr/sbin/audit：用于控制 auditd 守护进程的运行；

/usr/sbin/auditconfig：用于读取和设置审计参数；

/etc/security/audit_starup：用于审计子系统初始化脚本；

/etc/security/audit_class：用于定义审计类型；
/etc/security/audit_event：用于定义审计事件及类型；
/usr/sbin/praudit：用于打印审计记录文件的内容。

12.2.8 系统服务

在操作系统管理上系统管理员应该知道系统中运行的程序和服务。如果不知道某个程序的作用及程序的使用者，则应该删除该程序。一旦发生了错误，系统管理员应该恢复运行的程序和服务，保证这些程序和服务有完善的备份。

TCP/IP 协议包中包含了并非每个服务器都需要的内容。为了系统安全，在系统中没有必要保留不必要的服务，应将它们从系统中删除，就如同将门关好一样。删除服务时应该配合文件/etc/inetd.conf 中所对应的服务行，删除不必要的服务行，并重新引导系统。

在 UNIX 系统中可以删除的服务有：

bootp：为客户提供网络信息；

finger：提供用户信息；

netstat：发布当前网络信息，如网络的连接情况等；

rexd：远程执行应用程序；

rusersd：显示已登录的用户；

systat：发布当前系统信息；

talk：提供用户间文本通信；

tftp：提供一般的文件传输；

uucp：提供 UNIX 到 UNIX 的拷贝；

walld：允许向全部用户发消息。

12.3 防 火 墙

在网络环境中不但要保证 UNIX 系统能提供正常的网络服务，还要采取一定的安全措施使得 UNIX 服务器有安全环境。防火墙技术就是用于将要连入 Internet 的专用网络（私有网络或 Intranet）与 Internet 隔离，实现专用网络和 Internet 之间的访问在一定的限制条件和监控措施情况下进行，从而对专用网络形成保护。这样只有专用网络的 Web 服务器和连到 Internet 的路由器才存在于防火墙之外，而内部邮件服务器等服务器和专用网络内部的主机则位于防火墙内。

防火墙的主要功能有：

- 提供外部网络的侵入保护，进出防火墙都必须通过集中登录、监视和审查过程；
- 阻塞非授权的外部 IP 访问；
- 选择专用网络为外部网络提供的具体服务或拒绝访问，如邮件服务器只提

供邮件服务；

- 对外部 Internet 隐藏专用网络；
- 在日志中详细记录通过防火墙的所有登录用户和 IP 地址。

防火墙安全技术主要包括：包过滤（Packet Filter）、应用网关（Application Gateway）和代理服务器（Proxy Server）。

12.3.1 包过滤（Packet Filter）

包过滤用在网络层中对数据包实行有选择的通过。

包过滤过程首先根据用户过滤需求选择过滤规则并设置系统内过滤逻辑，然后依据系统内过滤逻辑检查数据流中的每个数据包，根据数据包中的源地址、目的地址、所用的 TCP 端口与链路状态确定是否允许该数据包通过。比如可以根据曾经带来问题的用户名单制订拒绝指定用户发送数据包，也可以过滤 telnet 应用。

包过滤的实现方式有以下三种：

- 在路由器上除完成路由选择的数据转发之外还同时进行包过滤；
- 在 UNIX 工作站上使用包过滤软件进行包过滤；
- 在路由设备上实现，如屏蔽路由器。

包过滤虽然可以防止外部攻击，增强系统的安全性。但却存在以下问题：

- 如果数据包中的源地址和目的地址是伪造的，则包过滤无法防止非法传输；
- 如果只依赖包过滤实现安全，则过滤规则非常复杂，并且随着网络和用户的变化必须不断改变过滤规则，实现难度大；
- 如果要过滤的数据包与其他数据包之间存在一定的关系，则包过滤无法检查存在的关系。

对于小型网络来说，最常使用的安全措施是在路由器上实现包过滤，不需另外添加设备。对于中型网络来说，包过滤是网络安全的第一道防线，在考虑网络速度及网络性能的情况下，针对具体要求不同选择是否需要单独的设备。对于大型网络来说，会将包过滤和网关、代理服务器一起作为网络安全措施加以考虑。

12.3.2 应用网关（Application Gateway）

应用网关是在网络应用层上的协议过滤。针对特别的网络应用服务协议对数据包进行分析并形成报告，并由专用的 UNIX 工作站完成。应用网关需要维护通过的每条连接的状态信息，并在用户访问前将用户或应用程序的信息作为认证依据。应用网关的实现比包过滤复杂。

12.3.3 代理服务器（Proxy Server）

代理服务器是设置在 Internet 防火墙网关的专用应用级软件，代理服务器准许网络管理员允许或拒绝特定的应用程序或一个应用的特定功能，如拒绝 telnet 服务。

代理服务器提供的主要功能通常有：

- 用户 IP 过滤；
- 用户帐号过滤；
- 用户 TCP 端口应用限制；
- 用户访问时间限制；
- 用户访问跟踪；
- 用户日志信息；
- 用户计费。

通常代理服务器的实现是在 UNIX 服务器上安装代理软件。将代理服务器、Web 服务器、路由器一起均置于网络进出口部位。

12.4 本 章 小 结

当今 Internet 上有数之不尽的 UNIX 主机，多数肩负着重要的网络服务，甚至还保存有极其重要的数据。为了保证其不受破坏，我们需要根据自身的特点，制定相应的安全策略，合理配置系统、加强操作的保密性以及注意对异常事件的监控，才能提高安全性，预防灾难的发生，做到防患于未然。

在系统安全性上，尽管不同的系统需要从不同方面加以考虑，但是系统日志却给系统管理员提供了详细的第一手资料，值得重视。今天，大多数网络在系统安全上都采用了代理服务器方案。

上 机 练 习

1．观查一台主机的配置文件，查看一些相关信息，分析系统是否存在缺陷或者已有异常事件发生。

2．对上述情况提出自己的改进意见。

习 题 十 二

1. 请说明系统有哪些日志文件？哪些容易被修改？
2. 分析一台 UNIX 邮件服务器可能遭到的网络攻击，应该采取什么措施？
3. 为了系统安全考虑，在系统中与服务无关的哪些进程可以删除？
4. 请说明为什么要用防火墙？包过滤技术主要包括哪些？

参 考 文 献

1 [美] Bootle K S 著．精通 UNIX．李永峰等译．北京：电子工业出版社，1997

2 陈慧蓉编著．UNIX 系统基础．北京：清华大学出版社，1998

3 苏倍庆．最新 UNIX 系统学习手册．北京：学苑出版社，1993

4 [美] Kuo P 著．最新 UNIX 开发使用手册．前导工作室译．北京：机械工业出版社，1999

5 [美] Bach M J 著．UNIX 操作系统设计．陈葆玉译．北京：机械工业出版社，2000

6 [美] Forouzan B A，Gilberg R F 著．UNIX 和 Shell 程序设计权威教程．彭松虎译．北京：清华大学出版社，2003

7 [美] Robbins K A，Robbins S 著．实用 UNIX 编程．刘宗田译．北京：机械工业出版社，1999

8 [美] Kernighan B W，Pike R 著．UNIX 编程环境．陈向群译．北京：机械工业出版社，1999

9 [美] Preston W C 著．UNIX 备份与恢复．李如豹译．北京：机械工业出版社，2003

10 孟庆昌等编著．UNIX 使用与系统管理．北京：清华大学出版社，2000

11 张红光，李福才编著．UNIX 操作系统教程．北京：机械工业出版社，2003

12 Mourne S R．The UNIX System．Addition-Wesley Publishing Company，1988

13 Bach M J．The Design of the UNIX Operating System．Printice-Hall Inc，1988