目　录

第一部分

Linux 的安装和快速入门

在阅读这本书之前，读者可能已经了解了 Linux 的许多特色。这一部分作为一份快速入门教材，将帮助你在最短的时间内掌握有关 Linux 的基本概念、常用命令和一般使用方法。相信你只要仔细阅读，同时结合实际操作，学完这短短的几十页内容之后，就会惊喜地发现：自己已经当之无愧是一名 Linux“初段选手”了！

第一部分主要包括以下内容：获取并安装 Linux，登录和使用 Linux 系统，进行正文编辑，使用 Linux 常见工具，配置和使用图形用户环境，在 Linux 下进行简单编程等等。为了让读者能够真正地理解各种操作，这一部分会以通俗的文字介绍并解释一些重要概念和知识点，如文件系统、磁盘管理等。

我们知道，Linux 实际是一种运行在 PC 机上的 UNIX 系统的变体，也就是说它所具有的很多特性与 UNIX 是相同或类似的。但是，很多读者一直是 DOS / Windows 用户，从来没有接触过 UNIX，会不会在阅读过程中遇到很多困难呢？不用担心，这一部分在组织内容时考虑到了这一点。事实上，它正是为那些对 Linux 感兴趣，但还不曾使用过任何 UNIX 系统的读者编写的。作为 DOS / Windows 用户，你不会感到任何阅读困难；相反，由于书中经常将 Linux 环境与 DOS 进行比较，而你对 DOS 基本概念和命令已经有了一定了解，因此会觉得所读内容十分浅显而亲切——这正是我们所期待的效果。

如果读者对 Linux 的基本知识和一般使用已经很熟悉，那么不妨跳过第一部分，直接学习后面的内容。在本书的后续篇章中，将详尽讨论对计算机黑客更具吸引力的内容，介绍 Linux 内核知识以及 Linux 高级应用的示例。

第一章

Linux 的获取和安装

本章中，我们将教你如何选择、获取 Linux 套装软件并把它安装到自己的机器上，从而体会在微机上使用 UNIX 的感觉。除此之外，还要介绍一些重要的 Linux 资源，让你知道如何查找 Linux 的相关文档，如何寻求帮助。

在开始工作之前，首先告诉你一件事：Linux 和 DOS / Windows 是可以在同一台计算机上并存的，所以，请尽管放心大胆地去尝试。如果你觉得 Linux 实在让人头疼，那么可以随时回到心爱的 DOS / Windows 环境中来，什么都不会丢失。当然，配置和使用 Linux 也许不象 DOS 和 Windows 那么容易，但是，研究并解决问题，这本身不就是一种乐趣吗？

1.1 Linux 的获取

1.1.1 选择 Linux 版本

Linux 是遵循 GNU 公用许可证的自由软件，因此它的发行不是由一家公司或机构控制的。事实上，有众多机构和个人在 Linux 内核的基础上打包各种工具和应用程序，发行自己“品牌”的 Linux 套装软件，如著名的 Slackware、RedHat、Debian 等。这一情形带来的好处是，用户可以货比三家，根据自己的使用需求和经济预算选择喜欢的 Linux 版本。当然，软件的价格与质量、服务、易用性常常是对应的，这一点不能忘记。

首先，澄清一下“Linux 版本”的概念。Linux 套装软件（Linux distribution）是在 Linux 内核（Linux kernel）的基础上打包而成，因此 Linux 版本的概念包括两层含义，一是指套件中所使用的 Linux 内核的版本，二是指套装软件本身的版本。Linux 内核的发展与套件的发展是各自独立的。在本书中，除非特别说明，以后所提到的“Linux 版本”一般均指后者；而对于计算机黑客来说，Linux 内核往往是其关注的焦点。

所谓“内核”，是与应用程序、工具程序相对的一个概念。它是一个操作系统中最重要的部分，负责对所有运行的程序（操作系统中称为“进程”）进行管理，保证各进程能够公平地共享处理器以及各种计算机资源，包括内存、磁盘空间等。此外，由于内核代码包含了大量设备驱动程序，因此它还在用户的应用程序与计算机硬件之间提供了一个良好的、可移植的接口。

Linux 内核与该系统中其他软件组成部分一样，也是很多人共同努力的结果。由于大

量人员参与内核的开发，因此其更新速度很快。Linux 内核的“官方版本”由 Linus B.Torvald 维护。内核的版本号形式为：

```
major.minor.patchlevel
```

其中，major 是主版本号，minor 是次版本号，二者共同构成当前内核的版本号，patchlevel 是对当前内核版本的修正次数。例如，kernel 2.0.30 表示对内核 2.0 版本的第 30 次修正版。

根据约定，次版本号为偶数时，表示该内核是稳定的发布版本，对它所作的修正主要是消除各类错误，未增添新特性；次版本号为奇数时，则表示该内核是不稳定的开发版本，开发人员在其中添加了新特性。由于 Linux 内核开发工作的持续进行，因此内核的稳定版本与在其基础上进一步开发的不稳定版本（例如 kernel 2.0 与 2.1 版）总是同时并存。对于一般用户来说，在正式的实用场合，建议采用稳定的内核发布版本。

Linux 是自由软件，任何机构或个人都可以在遵守 GNU 公用许可证（General Public License，GPL）规定的条件下随意打包组合 Linux 软件和工具，以免费或收费方式发行。经过几年的迅猛发展，今天人们已经可以看到多种 Linux 的版本。

与 DOS 等操作系统不同，Linux 套件的结构比较松散，这使得该系统的组织结构具备很强的开放性，用户选择的灵活程度很大——这也正是吸引众多 Linux 爱好者的特色之一。然而，需要提醒读者注意的是，某些 Linux 套件看起来小巧玲珑，但它仅仅包含了一个系统运行所必需的最基本软件，许多实际上很重要的工具和应用（如 X Window）却由于文件尺寸的限制而被排除在外了。

本书建议：如果你打算用自己的 Linux 系统实际去完成一些工作，同时现有的资源条件（包括计算机硬件、资金等）也许可，那么就不要选择所谓“最小配置”的 Linux 套件，而应当尽可能采用各种工具软件和应用程序比较齐全的 Linux 套件版本，这样能够给未来的工作带来很多方便。

如何选择 Linux 版本？其实，只要你觉得合适就是好的。请继续阅读本书，当你明确了自己使用 Linux 的目的之后，衡量现有机器的硬件条件、自己的资金能力、上网条件，便可以作出恰当的决定。如果那时你作选择仍有困难，不妨向周围安装过 Linux 的朋友们寻求建议和帮助，同时别忘记充分利用本书介绍的各种 Linux 网络资源和文档（在附录中列出了部分参考书目）。其实，选择 Linux 套件最简单的办法就是随便挑一个流行版本——因为各种 Linux 版本所包含的主要软件是相同的；而使用流行的版本，则更容易找到同路人，与其他用户拥有“共同语言”，获得来自各方的帮助，这一点在使用 Linux 时十分关键。

1.1.2 获取 Linux

获取 Linux 软件的渠道主要有两个：

- 通过网络，如匿名 FTP 下载 Linux
- 购买 Linux 光盘

早年（大约 93、94 年），网络是发行 Linux 的主要方式：用户花费很长时间，通过 Internet 下载文件，然后将其逐一拷贝到多张软盘上——其中一张用作引导盘（boot disk），用于安装其他软盘上的内容。以这种方式安装 Linux 很费时间，麻烦而且容易出错。

现在，各种 Linux 软件越来越丰富，软件规模也越来越大，其中很多都以 CD-ROM 的形式发行，而高质量的 CD-ROM 驱动器也已经普及成为 PC 机的标准设备。由于通过 CD-ROM 安装 Linux 方便、快捷、可靠，有时还能够获得具有商业水准的技术支持，因此以前一种方法初装 Linux 的用户已经相对减少了。然而，由于通过网络可以最及时地获取各种软件的最新版本或补丁程序，因此在进行系统升级工作时，使用网络常常是必需的。

FTP 方式

如果你有条件使用 Internet，可考虑以匿名 FTP 方式获取 Linux。包含 Linux 相关软件的最著名的 FTP 站点为：

```
ftp://sunsite.unc.edu
```

你可以在目录 /pub/Linux/distributions 下找到各种不同的 Linux 套件版本，例如广受欢迎的 RedHat、Slackware、Debian 等。请读者注意，在下载 Linux 软件时，应当用二进制模式传输文件。

在国内，可从如下站点获取：

```
ftp://ftp.pku.edu.cn
ftp://ftp.hanwang.com.cn
```

要提醒读者的是，通常的 FTP 文件传输中，有两种不同的传输模式：二进制和文本模式。用户所下载的文件，如程序、压缩文件、归档文件等，多为二进制文件；另外一些文件，如 README 则为文本文件。以文本模式传输二进制文件，会导致下载的文件无法使用。

表 1-1 列出了提供 Linux 软件的部分其他 FTP 站点。请读者就近选择，以节省网络资源和传输费用。其中，tsx-11.mit.edu 及 fgb1.fgb.mw.tu-muenchen.de 是 Linux GCC 的官方站点，sunsite.unc.edu 及 ftp.informatik.tu-muenchen.de 等站点提供 ftpmail 服务，某些站点映射了其他站点的内容。

表 1-1 提供 Linux 的部分匿名 FTP 站点

域名	IP 地址	Linux 所在目录
tsx-11.mit.edu	18.172.1.2	/pub/linux
sunsite.unc.edu	152.2.22.81	/pub/linux
ftp.funet.fi	128.214.248.6	/pub/linux
net.tamu.edu	128.194.177.1	/pub/linux
ftp.mcc.ac.uk	130.88.203.12	/pub/linux
src.doc.ic.ac.uk	146.169.2.1	/packages/linux
fgb1.fgb.mw.tu-muenchen.de	129.187.200.1	/pub/linux
ftp.informatik.tu-muenchen.de	131.159.0.110	/pub/comp/os/linux
ftp.dfv.rwth-aachen.de	137.226.4.111	/pub/linux
ftp.informatik.rwth-aachen.de	137.226.225.3	/pub/linux
ftp.Germany.EU.net	192.76.144.75	/pub/os/Linux
ftp.ibp.fr	132.227.60.2	/pub/linux
ftp.uu.net	137.39.1.9	/systems/UNIX/linux
wuarchive.wustl.edu	128.252.135.4	/mirrors/linux
ftp.win.tue.nl	131.155.70.100	/pub/linux

ftp.stack.urc.tue.nl	131.155.2.71	/pub/linux
srawgw.sra.co.jp	133.137.4.3	/pub/os/linux
ftp.denet.dk	129.142.6.74	/pub/OS/linux
NCTUCCCA.edu.tw	140.111.1.10	/OS/Linux
nic.switch.ch	130.59.1.40	/mirror/linux
sunsite.cnlab-switch.ch	193.5.24.1	/mirror/linux
cnuce_arch.cnr.it	131.114.1.10	/pub/Linux
ftp.mcnash.edu.au	130.194.11.8	/pub/linux
ftp.dstc.edu.au	130.102.181.31	/pub/linux
ftp.sydutech.usyd.edu.au	129.78.192.2	/pub/linux

CD-ROM 方式

今天，CD-ROM 驱动器已经成为 PC 的标准设备，有越来越多的用户从 CD-ROM 安装 Linux，以 CD-ROM 为介质发行 Linux 也形成了相当的商业市场。

我们通常可以看到的 CD-ROM 版本的 Linux 有两类。一类仅仅提供 FTP 站点上 Linux 的映象文件，不提供技术支持，因此软件售价低廉；另外一类是由商业公司出版发行的 Linux 套件，其中包含 Linux 内核、应用程序、文档、安装界面、系统配置及管理工具等等。使用这种版本，用户可以获得技术支持，但其价格也比较昂贵。CD-ROM 套件的价格一般为几十美元，很多商家对定期升级软件的用户提供价格上的优惠。本书建议读者尽可能选用所包含内容丰富，版本较新的 Linux 套件，这样能够给安装和使用带来极大便利。

如果上网，你会注意到，很多以 CD-ROM 形式发行的 Linux 套件（例如 Slackware、Yggdrasil 和 RedHat）同时也可以在商家开办的站点通过 FTP 免费获取。但是，如果你的机器已经安装了 CD-ROM 驱动器，而且经济状况也不算太拮据，仍然建议你选用 CD-ROM 形式的 Linux 版本。因为相比 FTP 而言，使用 CD-ROM 版本不仅可以节省下载软件所花费的时间（也许还有金钱），而且可以享受一定的技术支持，获得内容详尽、使用方便的文档资料印刷本。

下面列出了几种常见的 Linux 套件版本及其发行商的网址和联系方法。各种 Linux 版本的具体特点本书不作逐一讨论，感兴趣的读者可以自行上网查询有关信息。

- **Craftworks Linux**
  发行商：Craftwork solutions, Inc.
  http://www.craftwork.com
  Email: info@craftwork.com

- **Debian Linux Distribution**
  发行商：Debian Linux Association
  http://www.debian.org
  ftp://ftp.debian.org/debian
  Email: info@debian.org

- **Linux Pro**
  发行商：WorkGroup Solutions, Inc.
  ftp://ftp.wgs.com/pub2/wgs
  Email: info@wgs.com

- **Red Hat Linux**

发行商： Red Hat Software
http://www.Red Hat.com
ftp://ftp.redat.com
Email： Red Hat@Red Hat.com

- **Trans-Ameritech Linuxware**
  发行商：Trans-Ameritech
  http://www.zoom.xom/tae
  Email: info@trans-am.com

- **Slackware**
  发行商：Walnut Creek CDROM
  http://www.cdrom.com/titles/os/slack96htm
  ftp://ftp.cdrom.com/pub/linux/slackware
  Email： info@cdrom.com order@cdrom.com support@cdrom.com

- **Yggdrasil Plug-and-Play Linux**
  发行商：Yggdrasil Computing, Inc.
  http://www.yggdrasil.ccm
  ftp://ftp.yggdrasil.com
  Email: info@ yggdrasil.com

获取 Linux 的其他途径

如果有条件，用户还可以通过 CompuServe 或 Prodigy 网络下载 Linux 软件。许多拨号 BBS 也提供 Linux。请读者访问下列新闻组和 FTP 站点，查看文档“Linux Distribution HOWTO”，获取有关 Linux 发布的具体信息。

```
news:comp.os.linux.announce
ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/Distribution-HOWTO
```

1.1.3 RedHat 简介

优秀的 Linux 套件有多种，本书只介绍一个——RedHat Linux。它是 Red Hat Software 公司的产品，曾被业界权威杂志 *InforWorld* 评为“最佳 Linux 套件”（Excellent Linux Implementation）。RedHat Linux 具有以下特点：

- 从 4.0 版开始，RedHat Linux 同时支持 Intel、Alpha、Sparc 三种硬件平台环境，这一点令 Red Hat 公司引以为豪。
- 安装时，在 RedHat 3.x 之前的版本中，用户必须在大量映象文件中搜索所需内容；而 4.x 版 RedHat Linux 对安装界面作了改进，使得整个安装过程变得一目了然。如果用户以 CD-ROM 方式安装 RedHat Linux，只需制作一张引导盘（boot disk）即可；如果安装过程中需要网卡（FTP 方式安装）或 PCMCIA 卡，才需要第二张扩充盘（supplemental disk）。
- RedHat 采用了先进的 RPM（RedHat Package Manager，“小红帽软件包管理器”）技术，使用户能够轻松方便地安装、升级、卸装各种应用程序和操作系统组件，其中包括 Linux 内核及 OS 基础本身。这样，RedHat 只要安装一次，以后根据需要升级、修正错误即可。现在，RPM 技术已被其他 Linux 套件广为采用。

- RedHat 中汇集了完整而精致的软件包和文档资料，例如，/usr/doc/ 目录下的各种说明文档 HOWTO、LPD、FAQ，各种服务器程序 **gopher**、**samba**、**httpd** 等等。
- RedHat 采用 PAM（Pluggable Authentication Modules）技术以加强系统安全和系统管理的扩充性。

当然，不同 Linux 版本各有其特色，每位用户完全可以根据自己的条件、需要和爱好自由选择。在本书中，限于篇幅，如果讨论的内容涉及到了 Linux 的具体版本，将以 RedHat 为例。

1.2 安装 Linux 的硬件需求

在安装 Linux 之前，首先应当了解它对计算机硬件的要求。

Linux 对计算机硬件的要求很低。Linux 操作系统本身是由广大 Linux 用户——其中包括大量 PC 用户——通过 Internet 共同开发的产物，因此它所支持的大部分硬件都是普通用户和开发人员已经拥有或者很容易得到的。对本书的多数读者——PC 用户来说，Linux 支持 80386/80486、Pentium 系统中绝大多数常见的硬件和外围设备。所以，如果在你手头有一台 IBM PC 兼容机，那么拿它来运行 Linux 一般不会有什么问题。需要注意的是，由于给未公开设备编写驱动程序涉及到了专利问题和法律问题，因此 Linux 一般不支持未公开设备。

下面列出了 Linux 所支持的部分硬件的列表。由于每一天都有新的硬件产品出现，每一天都有无数勤奋的 Linux 爱好者在为各种硬件设备编写或升级驱动程序，因此这份列表不是完整的。如果你有意了解有关 Linux 所支持硬件的更完整清单和最新信息，请查看文档“Linux Hardware HOWTO”以及其他网上信息。HOWTO 文档位于：

```
http://sunsite.unc.edu/LDP/HOWTO/
```

表 1-2 Linux 支持的部分硬件

硬件	Linux 支持的类型及附加说明
CPU	支持 Intel 80386、486、Pentium 和 Pentium II 处理器，AMD 处理器，Cyrix 处理器。如果你的 CPU 是 Intel 80386 或 486SX，虽然没有数学协处理器，同样可以使用 Linux，因为 Linux 内核可在无协处理器的情况下作 FPU 仿真计算。
主板	支持 ISA、VLB、EISA 和 PCI 总线结构的系统主板。
RAM	支持各种 EDO、SDRAM 或 FPM 内存，只要主板支持即可。内存的多少对 Linux 系统的运行状况影响很大。Linux 可用的最小内存为 2M，但是如果你打算在系统内运行多个大型程序，如 X Window、Emacs，或者有多名用户同时使用你的系统，本书建议至少安装 16M 内存，32M 或 64M 则更好。Linux 能够自动寻址所有内存，内存越大，系统运行起来速度就越快。如果内存严重不足，Linux 只有频繁使用在硬盘上开辟的交换分区作为虚拟 RAM，这会大大降低系统的整体性能。
硬盘控制器	支持标准 IDE、EIDE、MFM/RTL 控制器，部分 SCSI 接口的硬盘控制器。具体型号请查阅文档“Hardware compatibility HOWTO”和“SCSI HOWTO”。Linux 的最小系统可以在软盘上运行，但速度很慢。Linux 对硬盘大小的要求主要取决于系统欲实现的功能、所安装软件的多少、系统所支持的用户数目。一般情况下，有必要在硬盘上开辟专用的交换空间作为虚拟内存，以提高系统性能。Linux 可以同时支持多个硬盘驱动器。

软盘	支持 3.5 英寸软盘驱动器。5.25 英寸不推荐。
CD-ROM	支持几乎所有 SCSI 接口的 CD-ROM，部分 EIDE 接口的 CD-ROM。Linux 支持 CD-ROM 的 ISO-9660 文件系统。
显示器/显示卡	支持 VGA、EGA、CGA，Super VGA 显示卡及显示器，S3、ATI MACH 及 ET4000 等系列的加速卡。如果使用 X Window 图形环境系统，需要对视频硬件提出较高的要求。本书将在相应章节作详细讨论。
网络	支持各种通用的以太网卡及局域网适配器。
I/O 接口	支持各种标准串行口、并行口，部分多用户卡。
其他硬件	支持 SCSI 接口的磁带机、各种标准串行鼠标、各种并行打印机、各种串口、调制解调器、各种 SoundBlaster 兼容声卡，等等。

事实上，Linux 的最小安装对硬件的要求不高于 MS-DOS 和 Windows 3.x 的要求。至于更具体的硬件型号，本书就不一一列出了。

也许，你会发现自己现有的硬件条件“碰巧”满足 Linux 的需要，那么恭喜你。但同时提醒你一句：在内存和硬盘空间这两样东西上，请不要太过节省！ 安装一个完整的 Linux 套件通常需要 60MB 到 300MB 的硬盘空间，如果是一个实际的多用户系统，还需要加上为多名系统用户保留的空间、开辟用作虚拟内存交换分区的空间。此时，如果你只舍得为 Linux 投入 500MB 硬盘空间，不免就太紧张了。系统的运行性能对内存大小也十分敏感。由于 Linux 不存在 DOS 中的 640KB 访问限制，能够自动使用机器中的所有内存，因此在很多情况下仅仅增加内存数量，无须升级 CPU，就能够明显提升系统的整体性能。内存当然是越多越好。本书的建议是：如果你打算建立一个具有实际用途的 Linux 平台，16MB 内存是一个下限；如果你希望自己的 Linux 平台能够与一台商用 UNIX 工作站相媲美，内存至少应当有 32MB 或 64MB。

另外一点需要提醒读者注意：如果你打算新购置一台 PC 用于运行 Linux，在规划各种硬件时，请尽可能选择主流产品——原因很简单：拥有更多的同路人，避免曲高和寡。

1.3 Linux 的安装

获得了 Linux 软件，并确认各种计算机硬件设备满足要求后，就可以开始安装工作了。

我们可以直接在裸机上安装 Linux；然而，Linux 也可以在硬盘上与其他操作系统，如 MS-DOS、Microsoft Windows 或 OS/2 共存——亦即，为硬盘分区，然后将 Linux、MS-DOS 和 OS/2 分别安装到各自的分区（事实上，我们甚至可以在 Linux 内直接访问 DOS 文件并运行部分 DOS 程序）。

安装 Linux 所需花费的时间依具体机器的速度、Linux 的版本，以及你的运气而定。在此给出一个估计值：如果是在裸机上安装，或者只安装 Linux 系统，需要二十分钟到一小时时间；如果你打算在机器上同时保留两个或多个操作系统（如 Linux 和 DOS），则可能要花费更多的时间，安装过程中需要注意的问题也多些。当然，现在计算机速度大为提高，也许只要十几分钟你就解决所有问题了。

下面，重点讨论安装 Linux 的一般过程，主要目的在于理清概念。然后，简单地介绍如何安装 CD-ROM 版本的 RedHat Linux。在各种优秀的 CD-ROM 套装软件中包含着 README 和 FAQ 文件，给出了关于该套件的有用信息以及 Linux 常见问题的解

答。这些文件通常位于 CD-ROM 的主目录，它们可以在 Linux 中读取，也可以在 DOS / Windows 中读取。请读者尽早阅读这些文件，这对于配置和使用 Linux，进一步安装 Linux 软件很有帮助。

1.3.1 安装 Linux 的一般过程

本小节只是给出安装 Linux 的一般过程。如果你手头的 Linux 套件配有印刷本的安装使用手册，请一定仔细阅读，然后再动手开始安装。

我们已经获得了 Linux 软件。Linux 软件主要有两种形式：CD-ROM 版本的 Linux 套件，以及从 FTP 站点下载到硬盘的文件。本小节主要考虑安装 CD-ROM 版本 Linux 套件的一般过程（除了上述两种形式，获取 Linux 软件后再行安装之外，还可以用 FTP 方式直接从 FTP 站点安装 Linux）。

通常，你的机器上已经安装有 DOS 等操作系统。在开始安装 Linux 之前，要作好以下几项准备工作：

1. 准备两、三张优质的 MS-DOS 格式的 3.5 英寸高密软盘。它们将用来制作 DOS 引导盘和 Linux 的引导盘和根盘（root disk）。

2. 备份系统，制作 DOS 引导盘，如果要对硬盘进行重新分区（关于硬盘分区，后面再作讨论）——这通常是必须的，硬盘上包括 DOS 分区在内的原有数据会全部丢失。因此，在改变现有 MS-DOS 分区的大小之前首先应当备份整个系统，将所有重要文件，如程序源代码、数据文件等先拷贝到可靠的软盘或磁带上（待重新分区完毕后，再将这些文件拷贝回硬盘上新建的 DOS 分区）。

然后，取一张 3.5 英寸软盘制作 MS-DOS 引导盘，用如下命令

```
FORMAT /S A:
```

格式化软盘，并将一些重要文件和工具拷贝到 DOS 引导盘中。这样，如果在安装 Linux 的过程中出现致命错误，可用 DOS 引导盘恢复硬盘。在 DOS 引导盘内应当包含下列文件：

```
FDISK.EXE       FORMAT.COM      MSDOS.SYS       IO.SOYS
COMMAND.COM     SYS.COM         SCANDISK.EXE    SCANDISK.INI
```

最后，请在软盘上作好标记，或写上“DOS 引导盘”字样，以免混淆。

3. 整理并记录硬件信息。与 DOS 和 Windows 相比，Linux 对 PC 硬件的利用更为充分，对硬件配置准确性的要求也更为严格。请读者将计算机各种硬件设备，如主板、显示卡、显示器、调制解调器的手册整理一下，放在顺手的地方备用。然后，记录硬件的配置信息。如果你使用着 MS-DOS 5.0 以上的版本，可运行 Microsoft 诊断工具 **MSD.EXE** 打印一份报告；如果你使用的是 Windows 95，可以查看“控制面板”中“系统”的“设备管理器”信息。这些系统硬件的相关信息可在安装 Linux 及 X Window 等程序时供参考。

4. 对机器硬件可能出现的配置问题进行检查。这可以防止在安装 Linux 过程中发生不可恢复的死机。

安装 Linux 的过程并不复杂。具体步骤如下：

1. 制作 boot 盘和 root 盘

安装 Linux 之前，一般需要制作两张用于引导和建立最小系统的软盘，它们的名字通常叫做 boot 盘（引导盘）和 root 盘（根盘）。

boot 盘的作用相当于 DOS 的引导盘，用来启动计算机，展开系统内核；root 盘用来建立一个 Linux 最小系统，展开一些必要的基本程序，以便完成后续的安装工作。

无论通过 FTP 下载的 Linux 软件，还是 CD-ROM 形式的 Linux 套件，都包含大量磁盘映象文件。其中一个文件将被写到软盘，用来制作引导盘。有时，套件提供多个引导盘映象文件，此时你应当根据系统的硬件状况选择适用的映象文件。根据硬件选用引导盘映象时，你会发现 CD-ROM 的索引文件提供了几乎所有文件映象的详细说明；或者你可以从引导盘映象的文件名很容易地判断其内容。因此，以手工方式完成这一步操作并不困难。

接下来，取几张 MS-DOS 格式的 3.5 英寸的高密软盘，运行 MS-DOS 程序 **RAWRITE.EXE**，将选定的 boot 盘映象、root 盘映象分别拷贝到软盘上。命令如下，无须带参数：

```
C:\RAWRITE
```

根据提示回答要拷贝的文件名、要写入的软驱名称（如 A:），**RAWRITE** 程序会把 boot 盘的有关文件逐块拷贝到软盘中。同样使用 **RAWRITE** 命令，拷贝 root 盘映象。完成上述工作后，你就有了两张软盘：一张为 boot 盘，另一张为 root 盘。请为它们作好记号，以免混淆。注意，这两张盘以后在 MS-DOS 下是无法读的，因为从某种意义而言，它们为 Linux 格式。

在 UNIX 或 Linux 系统中，同样可以完成上述拷贝映象文件的操作，只是将 **RAWRITE** 命令换成 **dd**，这里就不作具体介绍了。

Linux CD-ROM 通常提供了安装指导，帮助用户创建 boot 盘和 root 盘。这些安装指导往往是一些 MS-DOS 安装程序（如 RedHat 的 RedHat.exe 程序）或 UNIX 脚本程序，或者兼而有之。如果你确实找到了这样的程序，那么可以运行它，然后按照提示一步步地进行就可以了。

提醒读者，为了安装 Linux，其实你并不一定非要首先运行 MS-DOS 或 Windows。然而，运行了 MS-DOS 或 Windows，就可以方便地访问光驱，使通过 CD-ROM 创建 boot 盘和 root 盘的工作变得容易一些。如果你的机器上还没有安装操作系统，可以在别的机器上创建 boot 盘和 root 盘，然后进行安装。更为方便的是，如果你的计算机 BIOS 支持从 CD-ROM 引导系统，则可利用某些 Linux 发行版本的 CD-ROM 安装盘直接引导计算机，并进入安装界面。例如，RedHat Linux 5.x 的 CD-ROM 发行版就可以直接从 CD-ROM 上引导。

有关制作 boot 盘和 root 盘的进一步信息，请参阅“Linux Bootdisk HOWTO”，该文件位于

```
http://sunsite.unc.edu/LDP/HOWTO/Bootdisk-HOWTO
```

2. 准备硬盘分区

如果你打算安装一个多重引导系统（即在一台机器上同时保留 Linux 和 DOS 或

Windows，并允许用户根据需要切换到不同的操作系统），则必须对硬盘重新分区，为 Linux 腾出空间，建立更多的硬盘分区（如果你有多个硬盘，当然也可以把各个操作系统安装到不同硬盘上）。

当我们在手头现有的计算机上安装 Linux 时，通常整个硬盘都已经分配给了 DOS、OS/2 等系统。要在硬盘上为 Linux 腾出空间，就必须改变各分区的大小，创建新分区，然后在新分区上重新建立文件系统。

切记：对硬盘重新分区，意味着删除硬盘上原有的一切文件。因此，对硬盘重新分区之前，千万不要忘记备份系统！

简而言之，分区（partition）就是从一个硬盘中划出供某个操作系统使用的空间。

在引入 Linux 系统时，如果你打算同时保留原有操作系统，那么可以有以下几个选择：

- 使用一个完全建立在软盘上的 Linux 最小系统。
- 以 UMSDOS 方式安装 Linux 系统，把 Linux 安装在 MS-DOS 分区的 \linux 目录下。此时系统以 DOS 格式存放 Linux 文件，通过 UMSDOS 的驱动程序将 MS-DOS 的 8.3 文件名转换成 Linux 的长文件名。用户可以通过在 DOS 下运行 **LOADLIN** 程序启动 Linux。相应地，删除 \linux 目录即可卸装 Linux。

这两种方式都不需要对硬盘大动干戈，重新建立分区。如果你只是想暂时尝试 Linux，它们可算是不错的选择。但在这两种方式下，系统的可用性、运行效率和安全性都比较差。如果你打算正式使用 Linux，本书不推荐上述方式。

- 如果系统挂有多个硬盘，可将其中之一直接分配给 Linux；如果现有的 MS-DOS 系统有多个主分区（如 C 和 D 盘），可转移文件，将其中之一（如 D 盘）腾空用于安装 Linux。这两种情况下，也可以省去分区工作。
- 如果你只有一个硬盘，硬盘上只有一个分区，而且已经全部用于其他操作系统，要安装 Linux 就必须对该硬盘重新分区。重新建立硬盘分区是比较麻烦的工作，必须备份原有系统的数据，并在新建的分区内重新安装原有的操作系统以及软件。

一个硬盘可以分为若干分区。例如在 DOS 中，如果系统把整个硬盘作为一个分区，则称其为 C 盘；如果 DOS 有多个分区，则依次称为 C 盘、D 盘和 E 盘。事实上，每个分区可以对应不同的操作系统。例如，我们可以在一个硬盘上建立三个分区，分别安装 Linux、MS-DOS 和 OS/2 系统。

在硬盘上建立的分区有三种类型：主分区、扩展分区和逻辑分区。由于受硬盘分区表的限制，在一个硬盘上最多只能建立四个主分区。为突破这一对分区数目的限制，引入了扩展分区。扩展分区本身并不存储数据，而是用来在其中建立多个逻辑分区。例如，我们可以在一个硬盘上建立三个主分区和一个扩展分区，然后在这个扩展分区内进一步建立若干个逻辑分区。这样，在不同的主分区和逻辑分区内可安装不同的操作系统。

有趣的是，我们往往要为 Linux 提供不止一个分区。这是因为在 Linux 中，交换空间（用作虚拟 RAM）和文件系统（至少有一个，在实际应用中常有多个文件系统）都要各自占据硬盘上的一个独立分区。关于文件系统的概念下面会有讨论，这里你只需要记住：Linux 往往需要占据不止一个分区。

如何规划属于 Linux 系统的硬盘空间呢？建议你考虑下列几项因素：

- Linux 系统本身所需要的硬盘空间。一般而言，安装完整的 Linux 套件大约会用掉 60MB 至 300MB 的硬盘空间，这还只是安装后软件本身所占用的空间，不包括系统运行时可能需要的额外空间。
- Linux 系统需要的交换分区。系统中通常必须规划出一部分硬盘空间专门用作“交换分区”，当运行程序时，如果物理内存耗尽，交换分区将用作虚拟 RAM。交换分区的大小取决于需要多少虚拟 RAM。一般来说，交换分区的大小为物理 RAM 的两倍。例如，用户有 8MB 物理内存，则需要 16MB 的硬盘交换分区。除非你的机器内存很多，否则建立交换空间总是有必要的。
- 在实际应用中，Linux 主机的可能用户数目。假如你的用户不少于 50 人，可以专门为所有用户建立一个独立的文件系统，分配给一个空间足够的硬盘分区。
- 系统所使用的主要应用软件。例如，你要是在 Linux 主机上运行网络新闻服务器，则会消耗大量的硬盘空间。对于这类应用，应当专门建立需要的文件系统，分配一个独立的硬盘分区。最好能够单独分配一个硬盘。

通过上面的讨论可以看到，通常情况下至少应当为 Linux 建立两个硬盘分区，一个用来建立文件系统，一个用作交换分区。作为初学者，可以按此办理。

对硬盘重新分区并不难。首先，用刚才制作的 DOS 引导盘将机器启动；然后，运行 **FDISK** 程序，选择要修改的硬盘，如 C 或 D 盘；接下来，根据菜单选项删除硬盘中要改变大小的分区，此时该分区内的所有数据均丢失；再根据菜单确定分配给 DOS 的硬盘分区大小，用于重建 DOS 分区。退出 **FDISK** 程序之后，我们可以用 **FORMAT** 命令格式化新建的 DOS 分区，并在该分区内重新安装 MS-DOS 操作系统、各种 DOS 软件，以及先前备份出来的文件；而硬盘中分出的其他区域暂时不用作任何操作，待将来建立 Linux 的文件系统和交换分区。

另外还有一个方便快捷的硬盘分区方法：使用 CD-ROM 附带的 **FIPS**。它是一个功能强大的 DOS 程序，能够在不删除硬盘原有数据的情况下对硬盘进行重新分区。具体使用方法请参阅该程序的说明文档。尽管 **FIPS** 的使用很安全，但拿它对硬盘重新分区之前，为你的重要文件作一个备份还是有必要的——至少，这可以使你在心理上获得安全感。

在硬盘上同时保留多个操作系统，就可以从不同的操作系统启动了。关于多重引导的知识，本书的第二部分有所讨论。

另外，请读者参考以下几篇 mini-HOWTO 文档：

```
Linux+DOS+Win95 mini-HOWTO
Linux+OS2+DOS mini-HOWTO
DOS-Win95-OS2-Linux mini-HOWTO
Linux+Win95 mini-HOWTO
Linux+WinNT-Loader mini-HOWTO
```

它们详细讨论了各种双重启动系统的配置。文档所在网址为：

```
http://sunsite.unc.edu/LDP/HOWTO/mini/
```

3. 从 Linux 引导盘引导计算机

插入刚才制作的 Linux 引导盘，从软盘上启动一个用于安装的小型 Linux 系统，此

时已能够访问 CD-ROM。

某些版本的 Linux 会提供一个安装菜单，用户可根据提示一步一步完成硬件检测，创建 Linux 分区，建立文件系统，安装系统软件，设定系统的启动方式（如硬盘启动、软盘启动）等工作。安装另外一些版本的 Linux 时，以软盘启动 Linux 后，会出现 login 提示符，用户可以用 root 身份登录，完成上述工作。

在继续之前，让我们初步了解有关文件系统和设备驱动的知识。在后续篇章中还会对它们作进一步讨论。

在 Linux 中，分区、文件系统、设备驱动是几个密切相关的概念。安装 Linux 时，某些版本的 Linux 套件要求用户自己用 **fdisk** 程序创建 Linux 分区，而另外一些版本则会提供友好的安装界面帮助用户完成这一工作。不论你遇到的是哪一种情况，都有必要了解 Linux 中关于分区、文件系统和设备的基本知识。

所谓文件系统，从本质上说就是经过格式化的硬盘、软盘或 CD-ROM 区域，用于保存文件。Linux 系统把所有数据组织成目录树上的文件；每个文件系统占据硬盘的一个分区，对应目录树的一个特定部分。例如，在任何 Linux 系统中都有一个主文件系统，名为“根文件系统”，它与顶层目录 / 相对应。

在安装 Linux 之前，必须建立好文件系统。可以只建立一个 Linux 文件系统，即根文件系统（相应地分配一个分区），然后把所有文件都存放在根文件系统中。也可以为 Linux 建立多个文件系统，每个文件系统占据硬盘上的一个独立分区，然后通过挂装（mount）操作把各文件系统结合成一个有机的整体。例如，若有两个文件系统，其中一个对应“/”，一个对应“/usr”，/usr 挂在 / 下，这时需要两个硬盘分区。

与单一文件系统相比，管理多文件系统比较复杂，但多文件系统可以获得更好的安全性。假设所有文件都存放在根文件系统中，那么如果由于某种原因造成根文件系统的损坏，所有文件会全部丢失；如果使用多文件系统，某个文件系统的损坏一般仅局限在其内部。另外，由于一个文件系统不能跨多个硬盘，因此建立多文件系统是利用多个硬盘的唯一方法。

下面谈一谈 Linux 中的分区和设备驱动。

仅何 DOS 用户都知道，在 DOS 下各个硬盘分区、软驱、光驱是用字母标识的，例如 C: 和 D: 代表硬盘，两个软盘分别用 A: 和 B: 表示，光驱用 E: 表示。

在 Linux 下，各种设备和分区的命名方法完全不同。软驱、硬盘以及硬盘分区都有各自的设备特殊文件，这些设备特殊文件均存储在文件系统的 /dev 目录，用来实现其他软件与各种系统设备（如软盘、硬盘、鼠标器）的交互。比如说，鼠标器的设备特殊文件名为 /dev/mouse。

要使用某种系统设备，只需要引用它的设备特殊文件的名称即可。例如，系统中安装了两个软驱，它们的设备特殊文件分别为：

```
/dev/fd0    第一软驱，对应于 MS-DOS 中的 A:
/dev/fd1    第二软驱，对应于 MS-DOS 中的 B:
```

如果系统中安装了两个 IDE 硬盘。第一硬盘有两个主分区和一个扩展分区，在扩展分区的内部含两个逻辑分区；第二个硬盘有三个主分区。则对应设备为：

```
/dev/hda    第一硬盘（整个硬盘）
/dev/hda1       第一硬盘的第一个主分区
```

```
/dev/hda2        第一硬盘的第二个主分区
/dev/hda3        第一硬盘的扩展分区
/dev/hda5        第一硬盘的第一个逻辑分区
/dev/hda6        第一硬盘的第二个逻辑分区
/dev/hdb     第二硬盘（整个硬盘）
/dev/hdb1        第二硬盘的第一个主分区
/dev/hdb2        第二硬盘的第二个主分区
/dev/hdb3        第二硬盘的第三个主分区
```

注意，/dev/hda4 被跳过，它对应第一硬盘的第四个主分区，在本例中不存在。另外，如果系统中安装的是 SCSI 硬盘，则驱动名称改为 /dev/sda、/dev/sdb 等，依此类推。

4. 建立 Linux 分区

用 **fdisk** 程序创建 Linux 分区。使用如下命令：

```
# fdisk /dev/hda
```

为系统的第一个 IDE 硬盘创建 Linux 分区。此时出现提示

```
Command (m for help):
```

表示 **fdisk** 正在等待命令。键入 **m** 可查看命令帮助列表，例如：命令 **l** 列出 **fdisk** 支持的分区类型、**p** 打印分区表、**d** 删除已有分区、**n** 添加新分区、**a** 切换分区为可启动或不可启动、**q** 不保存更改内容退出 **fdisk** 程序，等等。

请首先用 **p** 命令显示当前分区表，并记录下信息备用。例如：

```
Command (m for help): p
Disk /dev/hda: 64 heads, 63 sectors, 612 cylinders
Unit = cylinders of 4032 * 512 bytes
Device      Boot    Begin    Start    End      Blocks   Id  System
/dev/hda1   *       1        1        254      512032   6   DOS 16-bit>=32M
```

上述信息表示：在 /dev/hda1 上有一个 MS-DOS 分区，有 512032 块（500MB）。分区从柱面 1 开始，止于柱面 254，盘上共有 612 个柱面。ID 号 6 表示该分区的类型为“MS-DOS”。这样，有 612-254=358 个柱面可用于创建 Linux 分区。

分区工作并不复杂，可根据屏幕信息提示一步步地进行。在此提出几点请读者注意：

- 用 **n** 命令可创建新分区。默认状态下，**fdisk** 新添加的分区类型为“Linux native”（类型 ID 号为 83H）；如果你打算要建立交换分区，可用 **l** 命令查看 **fdisk** 支持的分区类型，然后用 **t** 命令把分区类型改为“Linux swap”（类型 ID 号为 82H）。
- 当确认一切改动无误后，用 **w** 命令将改动的分区表写入硬盘，然后退出 **fdisk** 程序。注意，运行 **fdisk** 所作的任何改动在使用 **w** 命令之前是不生效的，因此用户可以随意尝试各种配置，最后再作保存。如果对改动不满意，可以用 **q** 命令直接退出 **fdisk**。
- 不要用 **fdisk** 修改 Linux 之外的操作系统分区。
- 由于用户不能够从 1024 以上的柱面启动 Linux，因此必须把 Linux 的引导分区创建在 1024 柱面的范围之内。如果不能做到这一点，可以考虑以后从软盘

上启动 Linux。

5. 建立文件系统

创建好 Linux 分区之后，必须在（一个或多个）分区内建立文件系统，然后才能够存储文件。建立文件系统的工作在 MS-DOS 下称为“格式化”。Linux 支持多种文件系统，其中最著名的是 Ext2 文件系统。通常，Linux 套件可以为用户自动创建文件系统。也可以用手工方式创建文件系统。例如，命令

```
# mke2fs -c <partition> <size>
```

在 ***partition*** 分区创建一个大小为 ***size*** 的 Ext2 文件系统。

建立文件系统后，用户已经准备好在系统上安装软件。在各种版本 Linux 中，都有设置程序辅助用户完成后续操作，把 CD-ROM 上的软件安装到硬盘上。每个版本安装 Linux 软件的具体方法不同，细节请参阅其附带文档。

6. 制作引导盘，安装 LILO

安装到此，将要进行非常重要的 LILO 的设定。LILO 是一个优秀的系统工具，如果在硬盘中装了 LILO，就可以进入多操作系统选择菜单，选择用户需要的操作系统进行启动。例如，你可以选择从你的硬盘上启动 Linux、Windows 95 等多种操作系统。在设置程序中有相应的配置选项用来安装 LILO。

7. 重新引导 Linux，选择安装其他软件

这是最后一步工作。从硬盘上启动 Linux，以 root 登录，根据需要选择安装 CD-ROM 上的软件。

步骤 1 到 7，安装 Linux 的过程就是如此，不算太复杂吧！

接下来，请你参考上述过程，并对照自己手头 Linux 软件附带的文档，实际地安装一遍 Linux，概念和思路会更加清晰。建议你首先保存好自己的重要文件，然后在机器上大胆地多试验几次！

1.3.2 安装 RedHat Linux

RedHat 是一个比较成熟的 Linux 套件，提供了良好的安装界面。相信读者在了解 Linux 安装的一般步骤之后，已经能够根据 RedHat 安装程序给出的提示，顺利地完成安装工作。

在此，只简单地列出几个要点供参考。

1. 安装 RedHat 时，需要制作的两张启动用的软盘分别名为引导盘（boot disk）和扩充盘（supplemental disk），它们大致对应于前面讨论过的引导盘和根盘。
   在 RedHat CD-ROM 上有引导盘映象文件和扩充盘映象文件。制作引导盘，在 DOS 下用 **rawrite** 程序将引导盘文件 `boot.img` 写入一张 3.5 英寸高密软盘即可；类似地，制作扩充盘，则将文件 `supp.img` 写入软盘。
2. 用引导盘启动计算机。
   将引导盘插入软驱并启动计算机，会出现提示屏幕。按上下左右键、Tab 键和 Alt+Tab 键可移动光标。

3. 在系统安装过程中，会出现详细的提示菜单，要求用户选择显示模式、选择键盘、选择安装模式、选择光驱类型等等，并提示用户选择安装软件包、配置 X Window、设置 root 口令、安装 LILO。
4. 完成所有设置后，记住取出软盘，然后根据安装程序的提示重新启动机器。

如果一切正常，安装 Linux 的任务大功告成。也许，你对自己的当前配置不满意，可以修改或重新安装。本书建议读者多多尝试，记住：在实际应用 Linux 系统之前对它了解得越多，使用过程中遭遇的尴尬就越少。

1.4 Linux 资源，寻求帮助

对大多数 PC 用户来说，Linux 是一件很新鲜的事物，因此在安装和使用过程中遇到问题或困难不足为奇。

面对难题，我们只有两条路。一条路是知难而退，回到熟悉的环境，让问题“自然消失”，这是最简单的办法。本书希望你走的是另外一条道路，想办法去解决问题。

对于初学者而言，有助于解决问题的一个有效方法是，详细记录自己所作的操作以及计算机给出的重要信息。如果你在实际中遇到了问题，首先应当查阅所有可以利用的文档。本书附录给出了与 Linux 相关的文档和参考书籍的列表。注意，许多有关 UNIX 的书籍对于学习 Linux 同样很有帮助——理由非常明显。请相信，自己查书，自己分析，自己动手，冷静地独立解决问题对你迅速提高 Linux 水平大有帮助。

如果你对自己解决某个问题已经失去了信心，那也用不着自责。完全可以去寻求帮助，如咨询周围熟悉 Linux 安装和使用的朋友，或者上网咨询。

当你在网上张贴求助信息时，请注意以下几点：

- 首先，你一定要详细阅读所有与涉及问题有关的 HOWTO 文档和 FAQ 文档。各种 HOWTO 文档可以在站点 http://sunsite.unc.edu 的 /LDP/HOWTO 目录中找到。网友们对于明显欠缺常识或已经回答过多次的问题常常显得很不耐烦。
- 张贴求助信息时，一定要简明扼要地列出你的系统配置及其相关信息。如果别人不了解计算机系统的基本配置、所用软件、你的操作、结果、问题的症状，就无法准确回答。
- 提问时，请注意保持必要的礼貌；如果提问得不到回答，不要恼火，请从自己这方面找原因，也许同样问题在 FAQ 中已经清楚地给出了。

让我们感谢 GPL，它使得 Linux 成为一个金钱气息淡薄的世界，广大用户可以得到各种来自商家之外及时有效的帮助。我们有理由相信：在 Linux 世界中可以生活得很好，至少很有乐趣；会遇到问题，但能够及时解决——前提只要你愿意学习，知道如何在这个世界中生活。

到这里，对 Linux 获取和安装的讨论告一段落。也许，你已经在自己的 PC 机中安装好了 Linux，下一章讲述 Linux 的基本使用，请你继续跟上。

第二章

Linux 基 础

现在，你已经在自己的 PC 机中安装好了 Linux 和需要的软件。

Linux 与 DOS 不同，它是一个多用户、多任务的操作系统。为便于管理，系统要求使用者拥有一个合法的用户帐户。也许，目前在你的 Linux 系统中只有你自己一名用户（同时作为系统管理员，名为 root），即便如此，本书仍然建议你恪守系统管理员使用多用户系统时所必需遵循的一个良好规范，为自己建立一个普通的用户帐户。当你从事系统管理的工作时，以 root 身份登录；一般情况下，作为普通用户登录。这样做，虽然显得麻烦一些，但十分必要。因为 root 是一个特权登录帐号，能够超越 Linux 正常的安全检查。普通应用时，若以 root 身份登录，很可能由于误操作或其他原因对系统造成难以恢复的损坏。

如果你还没有建立自己的普通用户帐户，请以 root 身份登录，用 **useradd** 或 **adduser** 命令创建帐户。关于创建用户的方法，本章后面有介绍。

本章内容包括，如何作为普通用户登录到 Linux 系统，如何注销帐户，如何正确关机，如何使用最基本的 Linux 命令。另外，为了使读者对 Linux 系统有一个初步的认识，也会穿插一些有关系统管理、文件系统的基本知识。

本章不涉及过多过深的细节内容。了解 UNIX 的读者会发现，我们所讨论的不少知识实际上来自 UNIX。的确如此，因为 Linux 操作系统本身正是一种基于 PC 平台的 UNIX 兼容操作系统。

在开始下面的学习之前，本书假设：

- 读者已经掌握基本的 DOS 命令和概念
- 在读者的机器中已经安装好了 Linux，或许还包括 X Window 系统
- Shell 已经启动（Linux 中的 Shell 等价于 DOS 中的 COMMAND.COM）

2.1 登录、注销和关机

同 UNIX 系统一样，Linux 也是一个多任务、多用户的操作系统。换句话说，在 Linux 操作系统中可以有很多用户同时登录运行各自不同的应用程序，而在 MS-DOS 这样的单用户操作系统中，每一时刻只能够处理一名用户。

为识别不同的用户，系统要求每一名合法用户在使用 Linux 系统之前，首先必须登录。登录的过程包括输入用户登录名和口令，由于只有用户本人以及系统管理员才能够管

理这一口令，可以保证其他用户无法利用该用户的帐户登录并访问只有该用户才能访问的文件。

在下面的讨论中，假设你的用户名为 jerry，在系统中还有另外一个用户名为 tommy。

2.1.1 登录

普通用户登录 Linux 系统时，在屏幕上可以看到登录提示符：

```
login:
```

输入用户名（如你的用户名为 jerry），然后按回车键。要注意的一点是，Linux 象大多数 UNIX 系统一样区分字母的大、小写，小写字母和对应大写字母被看作是不同的字符。

如果该帐号设置了口令，系统将发出下列提示符请你输入口令：

```
password:
```

用户应当在提示符后正确地输入口令。为确保帐户的安全，用户输入的口令并不反显在屏幕上。

如果你正确键入了登录名和对应的口令，系统将启动 Shell 并给出提示符，请你输入命令；如果用户名或口令有误，会看到出错信息：

```
login incorrect
```

重新输入正确的用户名和口令，即可登录到系统中。作为普通用户帐号，一旦登录成功，系统就启动 shell 程序，并出现 shell 提示符。例如：

```
$
```

$ 为普通用户的提示符（超级用户 root 的提示符为 #），表示 shell 已经准备好接收用户的各种命令。此时，你可以输入各种 Linux 命令的名称及其所需的参数，让系统完成各种工作。系统执行命令的过程中，如有必要，你可以通过按组合键 Ctrl+C 来中断命令的执行。

使用各种 UNIX 系统（包括 Linux），都会与 shell 程序打交道，可供使用的 shell 程序有多种，例如 Bourne Shell 和 C Shell，它们的使用方法和功能都基本相同。Shell 类似于 MS-DOS 下的 COMMAND.COM 命令解释器，是用户与操作系统核心之间的接口，负责接收用户输入的命令并将其翻译成机器能够理解的指令。在大多数 Linux 版本中最常见的是 Bourne Shell 的一个免费兼容版本，名为 **bash**，后面将详细讨论。

2.1.2 注销和关机

作为 Linux 系统的用户，在学习其他基本操作之前，首先应当知道如何正确地注销帐户。要注销一个帐户，可以在 shell 提示符下使用命令 **logout** 或 **exit**，退出登录。这样，系统可以为用户作好关闭帐户前的准备工作。当然还有其他注销帐户的方法，但这两种方法最为安全。

```
$ logout
```

或者

```
$ exit
```

普通用户一般没有关机权限，只有系统管理员（root）才能关闭系统。那么，系统管理员如何关闭 Linux 系统？

从本质上说，Linux 是一种网络操作系统（虽然你的 PC 机目前也许尚未联网），这意味着在实际的 Linux 系统中，除了系统管理员本人之外，可能还有很多用户通过各种方式使用着 Linux 主机。因此在任何情况下，系统管理员都绝不能简单地用关闭计算机电源的方法来停止 Linux 系统的运行。另外，由于在正常工作时，系统为提高访问和处理数据的速度，将很多进行中的工作驻留在内存中，这种情况下，如果骤然关机，系统来不及将内存中的数据写到磁盘，会造成永久性的数据损坏和丢失。

在 Linux 系统中，安全的关机方法是使用 **shutdown** 命令，可向系统程序和用户发出关机消息和信号，使这些正在处于运行状态的程序和用户有足够的时间作好关机之前的准备工作。

shutdown 命令的一般格式为：

```
# shutdown switch time [message]
```

其中 ***switch*** 为命令开关，如 **-h** 表示执行命令后立即停机，**-r** 表示重新启动机器；***time*** 参数用来指定关闭系统之前等待的时间；***message*** 是发送给系统所有用户的关机警告信息。

例如：

```
# shutdown -h now
```

该命令将立即停止 Linux 运行。其中，参数 **now** 表示立即停止，该参数还可以用具体的时间值 hh:mm（时:分），或从当前时间起推算的分钟数来指定。例如：

```
# shutdown -h 12:00
```

该命令在中午 12:00 关闭系统。

```
# shutdown -r +10 "reboot for system test"
```

该命令在 10 分钟后重新启动系统，同时给用户发出警告信息“reboot for system test”。

用命令 **man shutdown** 可以查看 **shutdown** 命令的联机帮助，其中讲述了其他可用的选项。

还有另外的关机方法。例如，在文本模式屏幕下操作，可以按组合键 Ctrl+Alt+Del，待系统提示一切关机准备就绪后，即可关闭 PC 机；如果在 X Window 系统下工作，首先按组合键 Ctrl+Alt+Backspace，然后再按 Ctrl+Alt+Del。但是，如果你在一台拥有多名用户的实际的主机上运行 Linux 系统，一般不推荐采用这种关机方式。

再次提醒读者：不要直接关闭计算机的电源开关或按 Reset 按钮，因为 Linux 在内存中缓存着磁盘的输入输出操作，如果突然关机，系统内核来不及将缓冲区的数据写到磁盘上，就会丢失数据甚至破坏文件系统。

2.1.3 修改口令

使用一个实际的用户帐号，还应当学会如何修改帐号的口令。当你使用 Linux 时，

定期修改口令十分必要，特别当你的系统已经与网络相连，或者你的帐户有可能被别人恶意侵犯时。

修改口令所用的命令是 **passwd**。它会提示用户输入原先的口令和新口令。系统管理员具有更高的权限设置所有普通用户的口令，因此如果用户忘记了口令，可以向系统管理员求助。

2.2 文件系统概述

在实际的 Linux 系统中，会有多种经过格式化用来存储信息的随机存储设备，如硬盘、软盘、CD-ROM 等，而大硬盘则会被分割为多个容量适度的磁盘分区。此时，整个 Linux 系统存储在不同的设备或不同的磁盘分区中。系统存储的内容包括，普通文件（如文档、数据文件、可执行的二进制文件）、目录（其中保存该目录内其他文件和目录的名称、存储位置等信息，目录可以包含在上一级目录内，形成树形目录结构），以及特殊文件（如设备驱动程序）。

系统是如何组织这些设备或分区的呢？

在 DOS 下，每个设备或硬盘分区都有独立的根目录，如 A:\、B:\、C:\ 或 D:\ 等，不同设备或不同的硬盘分区中，文件和目录层次结构的根目录是各自独立的。

在 Linux 系统中情况有所不同：每个设备或硬盘分区构成一个文件系统，有其各自的顶层目录和目录层次结构；而在各文件系统之间，一个文件系统的顶层目录被生装（mount）到另一个文件系统的子目录上，如此操作，最终使所有的文件系统结合成一个无缝的统一整体，组织到一个大的树形目录结构中。该目录树的顶部是一个单独的根目录，名为 root，用 / 表示。根目录下是一些标准的子目录和文件。例如，在图 1-1 所示的 Linux 目录树中，文件系统 usr 和 home 等挂到 / 之下（并成为它的子目录），形成统一的树形目录结构。

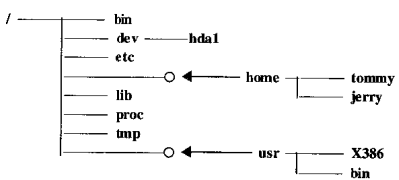


图 1-1 Linux 目录树结构

根目录 / 内包含 home、dev 和 bin 等子目录，而 home 目录中又包含了 tommy

和 jerry 目录，目录的最底层是各种普通文件和特殊文件。

与 DOS 中的概念类似：一个文件所在的目录可称为该文件的父目录；路径名称是一串以 / 分隔的目录或文件名（DOS 中用的是 \ 符号），用来确定文件或目录在树形目录结构中的位置；以 / 字符开头的路径名称是从根目录开始的绝对路径名；与绝对路径名对应，相对路径名是以其他目录名为开始来指定文件的路径名。利用路径名，可以无二义性地引用包含在不同目录中具有相同名称的两个或两个以上的文件或目录。

所谓当前目录，是指用户现在所处的目录。当用户运行一个程序时，运行中的程序（也称进程）将当前目录作为访问任何文件的相对路径名的起点。一般情况下，用户启动程序时所处的目录为默认的当前工作目录。要进入其他目录，可使用 **cd** 命令。例如：

```
$ cd /
```

上述命令执行目录切换，使根目录成为当前工作目录。

每个 Linux 目录中，还自动包含着“.”和“..”两个目录，它们分别是当前目录以及当前目录的父目录的别名。在指定相对路径名时，可以用“..”返回目录树的上一层。在根目录中，保留着“..”指向本身，这表明根目录就是它自身的父目录。

在 Linux 中，每个文件系统占据一个设备（如软盘、CD-ROM）或一个硬盘分区，对应于目录树的一个特定部分，每个文件系统的类型可以不同。Linux 能够识别的文件系统类型有多种，不同文件系统存储数据的具体格式不同；相同的一点是，各种信息都被组织成目录树下各分支中的一个个文件来保存——也许这也正是“文件系统”名称的由来。Linux 最常用的文件系统是 Ext2，本书第二部分将对它作详细讨论。

对于初学者而言，可以在自己的 Linux 机器中只建立一个文件系统，即根文件系统（相应地分配一个硬盘分区），然后把所有文件和目录都存放在根文件系统中。

当然，更好的做法是为 Linux 建立多个文件系统。在 Linux 的实际应用中，通常需要建立多个文件系统，每个文件系统分别占据硬盘上的一个独立分区。例如，建立两个文件系统，其中一个对应 / ，一个对应 /usr，这时需要两个硬盘分区。

使用多文件系统可以获得更加良好的安全性：假设把所有文件都存放在根文件系统中，那么如果由于外来的恶意侵入或意外的物理损伤造成根文件系统的破坏，其中包含的所有文件会全部丢失；而使用多个文件系统，某一文件系统的损坏通常仅局限在内部，其他文件系统中的文件和目录能够得以保全。使用多文件系统的另一个原因是，单一文件系统不能跨多个设备。因此，如果在你的系统有多个硬盘，在不同硬盘分别建立文件系统是利用零星空间（也许唯一）的好办法。

有关文件系统的详细讨论将在本书第二部分中给出。

2.3 用户帐号和口令

前面说过，为确保 Linux 系统的安全，你应当为自己建立一个有别于 root 的普通用户帐号（如 jerry），供你运行普通应用程序时登录。另外，系统中的其他用户，如 Tommy 也会要求你为他建立用户帐号。在多用户系统中，帐号是标识不同用户的唯一方法。

2.3.1 用户信息

在系统的文件目录树中有一个口令文件 /etc/passwd，它是允许每个用户阅读的普通文本文件，每一行包含了一名用户的详细信息。用 cat 命令可将该文件的内容列出：

```
$ cat /etc/passwd
```

每一条用户记录以字符“:”分隔成 7 个字段，格式如下：

```
username: encrypted password: UID: GID: full name: home directory: login Shel
l
```

具体含义见表 2-1。

2.3.2 口令

在 /etc/passwd 文件中保存的口令是经过加密处理的。由于口令的加密算法十分安全，因此 passwd 文件公开可读。然而，如果用户选择的口令字太简单（如字典中的现成单词），则很容易被人通过解密程序破译。

为解决这一问题，必须隐藏加密口令。在 Linux 中，可以用一些专用软件将加密的口令隐藏在单独的文件中（通常为 /etc/shadow），该文件不允许公开访问。有了额外的文件，还可以给用户记录增加更多的字段，如确定隔多长时间用户就必须更换口令等。

要提高帐户的保密性，用户也应当注意选择更好的口令。保证口令足够长（6 个字符以上），并混合使用大小写字母、数字和标点符号，这样可以给出更安全的口令。

事实上，并没有绝对安全的口令。只要时间足够，不管多好的口令都有可能被破译。为了达到最大限度的安全，最简单的方法是经常改变口令。在 Linux 系统中可以方便地使用 **passwd** 命令来改变口令，这在前面已经介绍过了。系统会提示用户输入新口令两次，以确保正确。

表 2-1 用户信息

字段名称	说明
Username	用户名。如 tommy、jerry，是用来标识系统中各用户的唯一标识符，用户登录时必须正确键入。用户名中可使用字母、数字、下划线“_”、句号“.”等。
Encrypted password	经过加密的口令。存储在系统中的口令采用了人们无法读懂的格式。如果这一字段为空，表明该用户无需口令。
UID	用户标识符。是系统分配给每个用户的一个独有的识别号。系统通常通过用户 ID，而不是用户名来操作和保存用户信息。
GID	用户组标识符。是用户的默认组 ID。有时候，一批用户需要在一个组内共同完成同一个项目，并允许他们共同访问一组特定的目录和文件。此时，这些用户可以由系统管理员被定义属于同一个或多个组。
Full name	用户全名，即用户的真实姓名，如 Jerry Lee。该字段通常用来保存用户的真实姓名和个人信息。
Home directory	用户的个人主目录。是用户在登录时最初所处目录的绝对路径。每一名用户都有自己的个人主目录，通常在目录 /home 下。例如，你的主目录为 /home/jerry。
Login Shell	用户登录成功后所执行命令的绝对路径，通常就是所启动的 shell 路径。如果此字段没有给出路径名，它的默认值是 /bin/bash。

2.3.3 建立和管理用户帐号

建立和管理用户帐号属于系统管理工作，这里只作简单介绍。

要建立一个用户帐号，系统管理员首先以 root 登录，然后运行 **useradd** 或 **adduser** 命令，以人机交互的方式完成建立新帐号的工作。程序会询问一些必需信息，如用户名、UID、GID、用户全名等，然后在 /etc/passwd 文件中自动为用户建立一个记录，并修改所有系统文件。查询联机帮助，可以知道 **useradd** 命令的具体用法。

类似地，删除用户帐号，可以运行 **userdel** 或 **deluser** 命令。

如果系统管理员想暂时停止某个用户登录系统的权利，可以不删除用户帐号，而只是在 /etc/passwd 文件中该用户记录行的口令字段前添加一个星号“*”，形式为：

```
username:*encrypted password: UID: GID: full name: home directory: login Shell
```

这样，该用户就无法登录了。

系统管理员有权设置和改变用户属性，如个人目录、口令等。

设置用户口令，同样可以用 **passwd** 命令，例如：

```
# passwd tommy
```

将改变 tommy 的口令。只有超级用户 root 才有权限改变其他用户的口令，普通用户只能用 **passwd** 命令改变自己的口令。

用户所在组、用户全名、登录 shell 等属性一般也是由系统管理员来设置的。在文件 /etc/group 中包含了用户组的有关信息，格式为：

```
group: password: GID: other member
```

例如：

```
users: *: 100: jerry, tommy
```

users 是一个普通用户组，GID 为 100，有两个组员 jerry 和 tommy。

一个用户可以同时属于多个组，只需将用户名分别添加到各组的记录行中就可以了。用命令 **addgroup** 或 **groupadd** 可以给系统增添新组；要删去一个组，只要在 /etc/group 文件中删除对应记录行即可。

2.4 主目录

当用户登录时，有一个专用目录与其登录名相联系，它是该用户的主目录（home）。主目录是用户登录到 Linux 系统时最初的工作目录，用户在系统内建立的所有文件和目录一般都放在他的主目录下。

用 **pwd** 命令可以确定当前工作目录在整个系统目录层次结构中所处的位置。因此，如果你在刚登录时使用 **pwd** 命令，可以给出主目录的位置：

```
$ pwd
/home/jerry
```

其中，/home/jerry 是你的主目录的绝对路径。

利用 **cd** 命令，可以将当前工作目录从主目录切换其他位置，**cd** 命令的参数为目标

目录的路径。例如，要将当前目录更改为根目录，可用下列命令：

```
$ cd /
```

使用 **ls** 命令，可以列出当前目录下的文件和目录清单。例如，在典型的 Linux 系统中，根目录下使用 **ls** 命令将得到以下输出：

```
$ ls
bin     dev     home        mnt     sbin        var
boot    dos     lib         proc    tmp         vmlinuz
cdrom   etc     lost+found  root    usr
```

Linux 中也可以用 **dir** 命令列出目录内容，该命令的语法和 **ls** 相同，是专为习惯于 DOS 命令的用户设计的。

综合运用 **pwd**、**cd** 和 **ls** 命令，用户可以考察 Linux 系统中的目录结构。

要返回用户的个人主目录，最简便的方式是使用不带参数的 **cd** 命令：

```
$ cd
```

当然，使用完整的 **cd** 命令同样有效。

第三章

文件和程序

Linux 系统中，所有的信息都组织成文件的形式，然后保存在树形目录层次结构中。从本质而言，用户的一切工作就是对文件的操作。使用系统的过程中，还会涉及到很多命令，这些命令实际上是一些程序，当用户键入程序名时执行。由此可以看到，掌握文件和程序的基本概念，理解众多文件如何组织，对于了解 Linux 系统的概貌，研究 Linux 系统的细节，进而有效地使用 Linux 系统很有帮助。

本章将讨论文件系统的层次结构、文件及目录的基本概念和访问权限，并介绍执行程序的一般方法。

3.1 文件系统的层次结构

文件系统是一个计算机系统内文件和目录的集合。在 Linux 中，为了方便地定位系统资源和程序，文件系统具有标准的层次结构，并构成统一的目录树。它从根目录 / 开始，根目录下面是一些重要的子目录，如 /bin、/etc、/dev、/usr 等等，这些目录包含了系统的配置文件和系统程序。此外，每个用户都有自己的个人主目录，用来存放属于该用户的个人文件。用户的个人目录一般放在系统的目录 /home 下，并以目录所有者的名字来命名。例如，jerry 的个人目录是 /home/jerry，属于他的所有文件，如程序、文档、数据文件等都存放在该目录中。

图 3-1 给出了一个简化后的 Linux 目录树结构。

用命令 **cd /** 切换到根目录，再执行命令 **ls -F** 列出根目录下的内容：

```
$ cd /
$ ls -F
```

在标准 Linux 目录树中，根目录的第一级子目录是：bin、boot、dev、etc、home、lib、lost+found、mnt、proc、root、sbin、tmp、usr、var 等。

下面挑选几个主要目录简单地提一下。在本书第二部分中还有详细讨论。

/bin

bin 是二进制（binaries）的缩写。许多基本的系统程序都驻留 /bin 目录中，用下列命令可以列出该目录下的文件：

```
$ ls-F /bin
```

其中包含许多我们所熟悉的名字，如 **cp**、**ls** 和 **mv** 等，实际上，它们正是这些常见用户命令所对应的程序。例如，键入 **cp** 命令，系统即运行程序 /bin/cp。

/boot

该目录用于存放供 LILO 使用的一些文件。

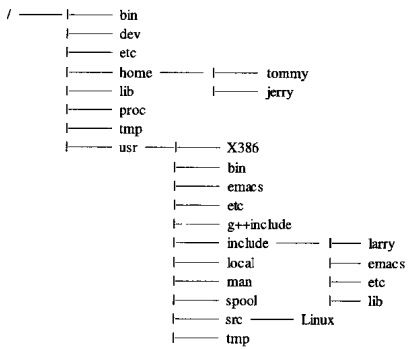


图 3-1 Linux 目录树结构

/dev

用 **ls -F** 命令可以列出 /dev 目录中的内容。该目录下是一些称为设备驱动程序的特殊文件，用于访问系统资源或设备，如软盘、硬盘、调制解调器、系统内存等。

有了这些文件，用户可以象访问普通文件一样方便地访问系统中的物理设备。例如，你可以象从一个文件中读数据一样，通过调用 /dev/mouse 文件从鼠标器读取输入信息。在 /dev 目录下，各种设备所对应的特殊文件以一定的规则命名。例如，/dev/fd0 文件指第一个软驱，/dev/fd1 指第二个软驱，/dev/hda 指整个第一个硬盘，而 /dev/hda1 指 /dev/hda 的第一个分区，等等。

/etc

包含系统管理所需要的大量配置文件和子目录，例如口令文件 /etc/passwd、系统初始化脚本文件 /etc/rc 等。该目录是系统中最重要的目录之一。

/home

包含所有用户的个人主目录。通常，用户的个人主目录以他的名字来命名，例如/home/jerry 是用户 Jerry 的个人目录。在新建的 Linux 系统中，/home 目录中可能没有任何用户。

/lib

该目录包含动态链接共享库的文件映象，这些文件为众多程序提供了共享代码。使用库文件，可以缩小可执行文件的尺寸，节省系统空间。

/lost+found

该目录通常为空，当文件系统发生故障时，用来存放找不到正确存储位置的文件。

/mnt

该目录通常为空，用户可以在该目录下临时挂装其他文件系统。

/proc

是 Linux 提供的一个“虚拟文件系统”，其中的文件都存放在内存中，而不是存放在磁盘中。它们指向正在系统中运行的各个进程，使得用户可以随时访问程序的运行信息。

/root

超级用户的主目录。

/sbin

用来存储系统管理员使用的基本可执行文件，如 **mount**、**fsck** 等等。

/tmp

程序执行时会产生一些临时信息，并把它们存放到一个暂时文件中，这些文件就放在/tmp 目录下。

/usr

这是一个非常重要的目录。其中包含一些子目录，用来存放系统的配置文件和最重要、最有用的大型软件包程序。前面讨论到的各目录对系统操作来说是最基本的，而在 /usr 中许多内容是可以任选的。如果没有 /usr 目录，系统中将只有 **cp**、**ls** 等程序，这样的系统会非常难用。

/var

该目录用来容纳大小经常发生变化或打算进一步扩充的目录或文件。例如，文件/var/adm 中包含着系统管理员感兴趣的内容，特别是系统日志，它记录了系统的所有错误或问题。

3.2 文件和目录

与 DOS 类似，Linux 中同样有文件和目录的概念。

在 Linux 中，各种信息都组织成文件，然后保存在磁盘中。文件有两类：普通文件和特殊文件。前者包括文本文件、数据文件、二进制可执行文件等，后者包括系统硬件的设备驱动程序等。而目录是一种用来存放一组文件的结构，其中保存着有关文件的名称、文件在磁盘上的存储位置等信息。目录可以嵌套并组织成树形，树的顶部是一个总目录 root，称为“根”，并用 / 表示。注意，所有 Linux 文件和目录名称（以及以后要讨论的 Linux 命令）中的字母都是大小写敏感的，这一点与 MS-DOS 不同。

用户输入的任何命令都是在当前工作目录下。用户登录成功后，当前工作目录是他的个人主目录，例如你的主目录是 /home/jerry。

如果要访问某个文件（或程序），可以给出文件的完整路径名，也可以只给出该文件与当前目录之间的关系。例如，在 jerry 下有目录 menu，menu 下有文件 lunch，查看 lunch 文件时，使用下列两个命令是等价的：

```
$ more /home/jerry/menu/lunch
```

以及

```
$ more menu/lunch
```

在命令行中，只要不用“/”作为文件名的第一个字符，系统就认为该文件的位置相对于当前目录（如上述第二行命令）；相反，如果用“/”作为文件名的起始字符，系统就把它当作完整的路径名（如上述第一行命令）。

3.2.1 固定链接

那么，文件在磁盘中是如何存储和组织的呢？

在磁盘上有一个数组，它的每一个元素称为一个索引结点（inode），保存着某一个文件的管理信息（如该文件的创建时间、文件的所有者、文件数据块在磁盘分区的存储位置等等）；同时，系统为存储在磁盘分区中的每一个文件分配一个号码，称为索引结点号（inode number），用来索引上述数组所保存的对应文件记录。一个文件的索引结点号与该文件的名称同时保存在目录中，形成一张联系文件名及文件索引结点号的表。在目录中，每一对文件名和索引结点号称为一个链接（link）。

类似地，目录也有自己的链接。每一个目录至少有两个链接：“.”和“..”，分别指向它本身和指向上一级目录。根目录 / 的“..”链接也指向其本身。

有了上述概念可以看到：同一个索引结点号可以与多个文件名建立链接，即允许一个文件拥有多个有效的路径名和文件名。参见图 3-2 所示。

这一功能十分有用。例如，我们可以为重要的文件或目录建立多个链接，提供“防删除”的功能，避免意外误删除造成严重后果。这样做的原理是，如果一个文件（或目录）的索引结点有一个以上的链接，删除操作只能破坏其中之一，而索引结点本身的其他链接仍然不受影响。当然，如果对只有一个链接的文件发出删除命令，索引结点、文件的数据块与目录的连接会被释放，文件也真正被删除。

例如，有一个文件 file_name1，用 **ls -i** 命令可以看到该文件的索引结点号：

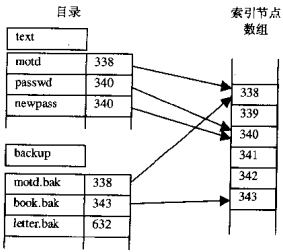


图 3-2　目录与索引结点数组之间的关系

```
$ ls -i file_name1
22191 file_name1
```

这里，文件 file_name1 在系统中的索引结点号为 22191。

ln 命令用来给一个已经存在的文件建立一个新的链接，现有路径名后跟以新建链接的路径名作为参数。例如，下列命令为文件 file_name1 创建一个名为 file_name2 的链接：

```
$ ln file_name1 file_name2
```

现在，访问 file_name1 或 file_name2 都是访问同一个文件。如果用户对 file_name1 进行修改，那么 file_name2 也同样改变。从这个角度看来，file_name1 和 file_name2 是同一个文件。用 **rm** 命令删除文件时，实际上只删除了一个文件链接。如命令

```
$ rm file_name1
```

则只有 file_name1 链接被删除了，而文件 file_name2 仍然存在。一个文件只有在没有任何文件链接时才被真正删除。通常情况下，每个文件只有一个链接，所以运行 **rm** 命令就可以删除。但是，如果一个文件有多个链接的话，要删除这个文件，就必须删除该文件的所有链接。用 **ls -l** 命令可以显示文件的链接数：

```
$ ls -l file_name1 file_name2
-rw-r-r--   2   jerry   user    12  Aug 5 16:51  file_name2
-rw-r-r--   2   jerry   user    12  Aug 5 16:51  file_name1
```

上述记录中，第二字段 2 表示文件的链接数。

用 **mv** 命令可以在一个文件系统的内部移动链接。实际上，它是在建立一个新链接的同时删除文件的原有链接；另外，由于 Linux 系统中没有专设 **rename** 命令，可以在目录内部用 **mv** 命令对文件重新命名。例如：

```
$ mv file_name1 file_name2
```

上述命令将 file_name1 文件改名为 file_name2。

如果将 **mv** 命令的第二个参数指定为目录，而将第一参数用文件名清单代替，可将所有这些文件移到指定目录下，并保持文件名不变。

上面所讨论的链接又称为固定链接，因为它直接创建了一个与索引结点号联系的链接。但是，由于每个文件系统（如一个硬盘分区）都有各自的索引结点数组，因此索引结点号只有在一个文件系统的内部才是唯一的；而在不同文件系统中，“同一个”结点号所指向的可能是完全不同的两个文件。这一情况意味着，固定链接只能用在一个文件系统的内部，而下一小节将讨论的符号链接则没有这个限制。

3.3 符号链接

如果要在不同文件系统的路径名之间建立链接，就不能使用固定链接共享索引结点的方式，而要采用符号链接（symbolic link）。

所谓符号链接，是一种在 DOS 下不存在的特殊的文本文件，其中包含着它提供链接的另一个文件的路径名，实际的数据信息则包含在另一个文件中。符号链接可以看作指向一个文件或目录的指针，用来替代它所指向的文件或目录，与 Windows 95 中的快捷方式相似。例如，用符号链接 /usr/X11 指向 /usr/X11R6，/dev/modem 指向 /dev/cua0 或/dev/cua1。执行命令时，如果涉及符号链接，则沿链接去访问包含实际数据的文件。

符号链接不仅可以用于单一的文件系统内部，也可用于多个文件系统之间，而索引结点链接只能用在一个文件系统的内部。因此，在两个文件之间建立链接，如果要求保证可移植性，应尽量使用符号链接。

3.4 文件权限和所有权

Linux 是一种多用户多任务的操作系统，允许多个用户同时登录和工作。用户在进入系统时，必须键入用户名和口令，而系统则通过帐户信息识别不同的用户。在 Linux 系统中，对每个文件（或目录）而言，都有四类不同的用户，他们是：

- root：系统超级用户，能够以 root 帐号登录
- owner：实际拥有文件（或目录）的用户，即文件属主
- group：用户所在用户组的成员
- other：以上三类之外的所有其他用户

其中，root 用户自动拥有读、写和执行所有文件的操作权限，而其他三类用户的操作权限可以分别授予或撤消。因此，每个文件为后三类用户建立了一组 9 位的权限标志，分别给文件属主、用户组和其他用户指定对该文件的读、写和执行权。

文件的属主可以控制该文件的属性，确定其他用户的操作权限。

用 **ls -l** 命令可以查阅文件操作权限标志的设置情况，请看下面的例子：

```
$ ls -l /bin/ls
-rwxr-xr-x  1   root    bin     27281   Aug 15 1995 /bin/ls*
```

该命令输出文件 /bin/ls 的相关信息。第一字段 -rwxr-xr-x 的第一个字符说明文件的类型。本例中的 - 表示，该行是关于一个普通文件的信息。这个位置也可以出现其他字符，如字母 d 表明该行是一个目录，字母 l 表明是一个链接。第一字段其余部分（9位，可分为 3 段，每段 3 位）是文件的权限标志，本例中：

```
rwx     表示文件属主的访问权限（允许读、写、运行）
r-x     表示文件属主所在组成员的访问权限（允许读、运行，不允许写）
r-x     表示所有其他用户的访问权限（允许读、运行，不允许写）
```

权限标志后面的数字说明文件的索引结点的目录链接数（1 个），后面几个字段依次为文件属主的登录名（root）、共享该文件访问的用户组组名（bin）、文件的字节数、文件最后修改的日期和时间、文件名本身。

看到这里，你就可以明白为什么只有身份为 root 的超级用户才能够删除文件 /bin/ls，因为其他所有用户都无此权限。要改变文件的访问权限，可使用下面的命令：

```
$ chmod <whoXperm> <file>
```

参数 ***whoXperm*** 中，***who*** 可以是“**u**”（文件属主）、“**g**”（用户组成员）或“**o**”（其他用户），***X*** 可以是“**+**”（添加权限）或“**-**”（撤消权限），***perm***可以是“**r**”（读）、“**w**”（写）或“**x**”（运行）。下面给出了几个例子：

```
$ chmod u+x file
```

允许文件属主运行 file 文件。该命令的简洁写法为：

```
$ chmod +x file
```

而命令：

```
$ chmod go-wx file
```

禁止小组成员和其他用户写、运行 file 文件。

```
$ chmod ugo+rwx file
```

允许所有人员读、写、运行 file 文件。

文件的访问权限还可以用三位八进制数字来指定。例如：-rwxr-xr-x 可用 755 表示（每个字符对应一位，--- 为 0，--x 为 1，-w- 为 2，-wx 为 3，等等）。上例中，rwx 为 7，r-x 为 5。一开始，你可能觉得使用这种表示方式有点困难，熟悉之后就很清晰了。

提醒大家一句，root 作为超级用户，可以修改任何用户文件的访问权限。

3.5　目录和目录权限

前面我们已经看到了 DOS 和 Linux 下文件的不同。在 DOS 下，根目录用“\”表示，而 Linux 下的根目录用“/”表示；类似地，DOS 下各级目录之间用“\”隔开，而 Linux 下各级目录用“/”隔开。下面给出了对比实例。

DOS 文件：

```
C:\PAPERS\GEOLOGY\MID_EOC.TEX
```

Linux 文件：

```
/home/jerry/papers/geology/mid_eocene.tex
```

目录同样具有不同的访问权限，前面对文件访问权限的讨论基本上也适用于目录。正确设置目录的权限位非常重要，而这一点往往容易被忽视。

对目录而言，r 权限表示，允许读目录下文件的名称，而不允许读文件本身的内容；x 权限表示，目录可以作为路径名来访问该目录下的文件，即用 **cd** 命令进入该目录并查看目录中可读文件的内容，但不允许读目录本身；w 权限表示可以删除该目录中包含的文件（当然还要要视该文件的访问权限而定）以及该目录本身。

注意，文件的访问权限依赖于它所在目录的访问权限。例如，某个文件即使被设置为 -rwxrwxrwx，但是如果用户没有访问该文件所在目录的权限，仍然无法访问它。换言之，要访问一个文件，就必须拥有该文件的路径名中所有目录的搜索权限，同时拥有该文件的读（或执行）权。

3.6 运行程序

前面我们已经用到了 **cd**、**ls** 等命令。它们实际上是系统提供的一些程序。

在 Linux 下运行一个程序与在 DOS 环境下类似，键入程序名就可以了。只要保存该程序的目录已经包括在环境变量 PATH 中，程序即可被启动。两种操作系统之间的差别在于，Linux 不象 DOS 那样可以执行当前目录下的程序，除非当前目录“.”已经被添加到 PATH 中。若要执行当前目录下的程序，可以这样键入：

```
./<prog>
```

下面是一个命令的典型格式：

```
$ command -s1 -s2 ... -sN  par1 par2 ... parN <input> output
```

其中，***-s1***、***-s2*** 至 ***-sN*** 为 ***command*** 的命令开关，***par1*** 至 ***parN*** 为命令参数。可以在同一行上键入多个命令，命令名称之间用分号隔开。如下所示：

```
$ command1; command2; ...; commandN
```

这就是运行程序的基本方法。Linux 环境下可以执行的程序很多，例如完成磁盘的格式化、文件压缩、程序的编译、文档的解压缩，等等。具体内容后面陆续介绍。

Linux 是一个多任务系统，可以同时执行多个进程（程序一旦运行，通常称为“进程”），允许用户将一个原先在后台执行的进程放置到前台并继续执行。此外，Linux 还允许用户启动多个会话（session），感觉上就象同时操作好几台计算机工作。这一内容也将放到后面的章节详细讨论。

下一小节简单介绍几个常用的系统操作命令。

3.6.1 常用的系统操作命令

本小节介绍的几个常用的系统命令主要用于目录和文件操作。限于篇幅，只列出了命令的基本功能和语法形式。详细的使用方法请用 **man** 命令查阅相应的联机手册。

cd

功能：改变当前目录。
语法：cd <dir>
说明：dir 是要切换的目的目录。
示例：cd /user

ls

功能：列出文件或目录的信息
语法：ls -s1 … -sN <file1> … <fileN>
说明：s1 到 sN 为命令开关，file1 到 fileN 是所给的文件或目录名称。
示例：ls -lF /home/jerry

cp

功能：拷贝文件到另一个文件或目录。
语法：cp <file1> … <fileN> <target>
说明：file1 到 fileN 是所要拷贝的文件名称，target 是目的文件或目录。
示例：cp /home/jerry/paper project

mv

功能：移动文件到另一个文件或目录。等效于拷贝文件后删除原有文件。
语法：mv <file1> … <fileN> <target>
说明：该命令也可以用来给文件重命名，与 DOS 命令 rename 类似。
示例：mv /home/jerry/paper project2

rm

功能：删除文件。
语法：rm <file1> … <fileN>
说明：注意，在 Linux 系统下没有 undelete 命令，文件删除后是不可恢复的。
示例：rm /home/jerry/paper

mkdir

功能：创建新目录。
语法：mkdir <dir1> … <dirN>
说明：可以一次创建多个新目录。
示例：mkdir /home/jerry/test

rmdir

功能：删除空目录。
语法：rmdir <dir>
说明：被删除的目录必须是空的，而且不允许删除当前工作目录。
示例：mkdir /home/jerry/test

man

功能：查阅指定命令或资源的联机手册。
语法：man <command>
说明：command 是要查阅有关帮助信息的命令名称。
示例：man ls

more

功能：显示指定文件的内容，每次一屏幕。
语法：more <file1> … <fileN>
说明：file1 到 fileN 是所要显示的文件。
示例：more /home/jerry/paper

cat

功能：通常用于连接显示几个文件，或一次显示整个文件。
语法：cat <file1> … <fileN>
说明：file1 到 fileN 是所要连接或显示的文件。
示例：more /home/jerry/paper

echo

功能：回显所给参数。
语法：echo <arg1> … <argN>
说明：arg1 到 argN 是所要回显的参数。
示例：echo "hello the world"

3.6.2 远程登录

在多用户系统中，一般的用户通过终端登录。普通终端包括一台用于显示输出的字符显示器，以及一个用于命令输入的键盘。一般而言，不同的终端通过串行口或多用户卡和系统相连，但是由于每个终端均需要一条物理的连接线路，因此终端的数量是有限的。

当 Linux 系统安装在 PC 机上时，绝大部分系统主要通过网络连接为用户提供远程登录服务，而很少提供通过串行口或多用户卡的终端连接。因为多个远程登录利用各自的虚拟网络线路连接到主机，从而能够共享一条网络线路，因此，从理论上来讲，远程登录的连接数量是不受限制的。

除上述终端和远程登录以外，用户也可以直接从安装 Linux 系统的计算机上登录。安装 Linux 本身的计算机的输入输出系统（显示器和键盘）又称为“控制台”，Linux 可同时提供 6 个虚拟的控制台，也称虚拟终端。用户按 Alt+F1 至 Alt+F6 键可在 6 个虚拟终端之间切换。

要登录到远程 Linux 主机上，首先需要目标主机提供远程登录服务，还需要知道远程主机的域名或 IP 地址。假设其域名地址为 remote.bigone.edu。只需键入以下命令：

```
$ telnet remote.bigone.edu
```

网络连接成功后，系统将向你提示用户名和口令，随后的工作和终端登录类似。如果知道远程计算机的 IP 地址，则可以直接利用 IP 地址登录。

远程登录实际建立在 TCP/IP 网络之上，所有支持 TCP/IP 网络协议的操作系统几乎均提供 `telnet` 程序，利用这一程序，你可以从 Windows 95 中登录到一个远程的 Linux 系统中。

第四章

磁 盘 管 理

实际的 Linux 系统拥有多种硬件，其中磁盘是主要的存储设备，包括系统本身在内的所有数据一般都保存在磁盘上。相应地，在整个系统设备管理工作中，磁盘管理占据着举足轻重的地位。

本章讲述涉及 Linux 设备管理的基本知识，重点是对磁盘的管理和使用。至于硬件设备本身的结构，超出了本书的范围，不作讨论。

4.1 Linux 中的设备

Linux 系统中可能有多种设备，如硬盘、软盘、CD-ROM、键盘、鼠标、打印机等，使用这些硬件时，需要用到硬件的相应设备特殊文件。

由于设备特殊文件具有文件的外在特征，因此应用程序能够象访问文件一样，通过 **open()**、**close()**、**read()** 和 **write()** 等系统函数方便地访问设备文件所对应的设备，从设备接受输入或将输出送到设备。应用程序对这些文件系统函数的调用，最终由内核通过调用相应的设备驱动程序接口而实现。设备驱动程序提供了操作系统内核与机器硬件之间的接口。

在 Linux 系统中，驱动程序主要完成以下功能：

- 对设备进行初始化
- 使设备开始运行或终止服务
- 在设备与系统内核之间传输数据
- 检测并处理设备错误

对于 Linux 操作系统来说，通常访问的设备有两类：一类是简单的字符型设备，如键盘、打印机等，这类设备传输的信息是一个个的字符数据；另一类是数据块型设备，如软盘、硬盘、CD-ROM 等，这类设备以数据块为单位传输信息。

与之相对应，有两类设备驱动程序，分别称为字符设备驱动程序和块设备驱动程序。二者的不同之处在于：与字符设备有关的系统调用直接由驱动程序的内部功能实现，而读、写块设备则主要通过数据缓存进行。通常只有在完成实际的输入输出操作时，才用到块设备驱动程序。

各类设备需要不同的设备驱动程序，例如软盘驱动器需要对应的软驱驱动程序。然而，同一个驱动程序通常可以支持一类设备，也就是说，用来控制软盘的驱动程序可以和所有

不同容量的软盘打交道。这样，如果系统中安装了多个软驱，实际上是用同一个驱动程序来控制的。

为了使应用程序能够象访问文件一样，用 **open()**、**close()** 等系统函数方便地访问设备，各种设备在系统目录层次的特定位置都有对应的文件名称。这些文件分别称为字符设备特殊文件和块设备特殊文件。标准情况下，所有的设备特殊文件都保存在目录树的 /dev 目录下。用 **ls** 命令可以查看设备信息：

```
$ ls -l /dev
brw-rw          1   root    floppy  2,0     Jul 18 1994 fd0
brw-rw----      1   root    floppy  2,1     Jul 18 1994 fd1
brw-rw----      1   root    disk    3,0     Jul 18 1994 hda
brw-rw----      1   root    disk    3,1     Jul 18 1994 hda1
brw-rw----      1   root    disk    3,2     Jul 18 1994 hda2
brw-rw----      1   root    disk    3,3     Jul 18 1994 hda3
brw-rw----      1   root    disk    3,4     Jul 18 1994 hda4
brw-rw----      1   root    daemon  6,0     Jul 18 1994 lp0
crw-rw-rw-      1   root    sys     1,3     Jul 18 1994 null
crw-rw-rw-      1   root    tty     5,0     Jul 18 1994 tty
crw--w--w-      1   pc      book    4,0     Jul 18 1994 tty0
crw--w--w-      1   pc      book    4,1     Aug 30 1994 tty1
```

每一行内容给出一个设备特殊文件的信息。其中，第一字段的第一个字符为 c 或 b，分别表示字符设备或块设备特殊文件；在日期字段之前有一对用逗号隔开的数字，分别为该设备的主设备号（major device number）和次设备号（minor device number）。

主设备号用来为一类字符设备或块设备指定设备驱动程序，调用其内部功能来处理这类设备的输入或输出请求；次设备号由驱动程序的子程序使用，主要用来确定当前的设备 I/O 请求与哪一个具体设备有关。

例如，在上面的清单中，软盘 0 和软盘 1（fd0 和 fd1）的主设备号相同，都是 2。当发生对 /dev/fd0 或 /dev/fd1 的 I/O 请求时，内核用主设备号 2 来确定要调用的设备驱动程序。在设备程序的内部，则以次设备号来区分具体是哪一台设备。例如，次设备号 0 对应 fd0，1 对应 fd1。

4.1.1 建立设备特殊文件

建立设备特殊文件的命令是 **mknod**。这一工作必须在 root 帐号下进行。命令格式为：

```
mknod file_name type major minor
```

其中，***file_name*** 是待建立的设备特殊文件的完整路径名，***type*** 参数可以取值 c 或 b，分别对应字符设备或块设备，***major*** 和 ***minor*** 是与这一设备特殊文件对应的主设备号和次设备号。例如：

```
# mknod /dev/mytty1 c 4 1
```

上述命令以主设备号 4 和次设备号 1 建立了一个字符设备特殊文件 /dev/mytty1。查看上面给出的清单可以发现，该组合的主设备号和副设备号已经存在（最后一行的 /dev/tty1），因此，这条命令只能建立一个备用的设备特殊文件，用来访问同一个硬件设备。

4.1.2 设备驱动程序原理

每一个设备驱动程序实质上是用来完成特定任务的一组函数集。驱动程序拥有一个称为 file_operation 的数据结构，其中包含指向驱动程序内部大多数函数的指针。引导系统时，内核调用每一个驱动程序的初始化函数，将驱动程序的主设备号以及程序内部的函数地址结构的指针传输给内核。这样，内核就能够通过设备驱动程序的主设备号索引访问驱动程序内部的子程序，完成打开、读、写等操作。

知道了操作系统内核如何调用特定设备驱动程序的内部函数，接下来看一看设备驱动程序与调用它的用户进程之间的关系。

当一个进程在 CPU 上运行时，Linux 将其作为当前进程。当前进程进行系统调用时，有可能会执行某个设备驱动程序内部的函数。也许，被调用的设备驱动程序函数会要求从硬件（如磁盘驱动器）进行 I/O 操作。

在这一情况下有两种解决方案：一是简单地要求 CPU 等待硬件完成输入/输出操作，然而由于硬件操作的速度一般比 CPU 处理速度慢几个数量级，因此这种做法将大大降低整个系统的速度；另外一种选择是，CPU 并不等待硬件完成输入/输出，而是由系统内核调度输入/输出操作继续进行，同时挂起当前进程。在后一种方案下，CPU 可以转而去处理其他进程，完成有用的工作，待硬件操作完毕，产生一个中断，再恢复先前挂起的进程。

需要注意的是，设备驱动程序是作为系统内核的一部分运行的，它的执行效率和运行逻辑会从根本上影响系统的整体性能。如果读者打算编写设备驱动程序，尤其要记住这一点。

4.1.3 常见设备种类

硬件的设备特殊文件放在系统的 /dev 目录中。下面看一看这些文件名称与具体的硬件是如何对应的。

- 以 fd 打头的文件是软盘设备。如 fd0 是第一软驱，fd1 是第二软驱。另外一些文件甚至指明了软盘的类型，如文件 fd1H1440 用于访问第一软驱中的 3.5 英寸高密软盘。
- /dev/console 文件用于访问系统的控制台，即直接与系统相连的显示器和键盘。
- /dev/ttys 和 /dev/cua 文件用来访问串行端口。例如，/dev/ttys 指 MS-DOS 下的串行端口 COM1，/dev/cua 用来访问调制解调器。
- 以 hd 打头的文件是 IDE 硬盘设备。/dev/hda 指整个第一硬盘，而 hda1 指第一硬盘 /dev/hda 的第一个分区。在系统中可能有多个 IDE 硬盘，它们的设备名称依次为 hda、hdb、hdc、……。一个硬盘可以有多个分区，设备名称分别为 hda1、hda2、……。
- 以 sd 打头的文件是 SCSI 设备。如果你有一个 SCSI 硬盘，那么就不会访问 /dev/hda 设备文件，而会访问 /dev/sda。此外，SCSI 磁带机用 st 设备驱动程序访问，SCSI CD-ROM 用 sr 设备驱动程序访问。
- 以 lp 打头的文件是并行端口设备。例如，/dev/lp0 指 MS-DOS 下的 LPT1。

- 以 tty 打头的文件是系统的虚拟终端。例如，/dev/tty1 指的是第一个虚拟终端，/dev/tty2 指第二个虚拟终端。通过按组合键 Alt+F1，……，ALT+F6 可以访问各虚拟终端。
- 以 pty 打头的文件是伪终端，用来提供一个终端进行远程登录。假如你的机器在一个计算机网络中，利用 **telnet** 远程登录时就要用到 /dev/pty 设备。

4.2 磁盘的格式化

格式化是磁盘管理中最基本的一项工作。也许你不曾想到，“**FORMAT A:**”这个简单的 DOS 命令实际完成了很多工作。事实上，当你执行 **FORMAT** 命令时，计算机会做以下三件事情：

- 对磁盘进行物理格式化
- 建立 A: 目录（即创建一个文件系统）
- 使 A: 目录可供用户访问，即“挂装”磁盘（mount a disk）

在 Linux 中，以上三步工作是各自分开完成的。也许这给系统的使用增添了技术复杂性，但它同时也意味着更大的灵活性。

创建文件系统成为独立的一步操作，这使得用户可以在 Linux 下选择不同的文件系统，例如，Ext2 和 MS-DOS。也许，你已经习惯于使用 DOS 格式的软盘，但其他格式可能更好——因为 DOS 格式至少无法支持长文件名。

下面，让我们以处理一张磁盘为例，说明格式化磁盘的一般过程。

4.2.1 物理格式化

首先，请你以 root 身份登录 Linux 系统并启动一次会话。用下列命令对一张标准的 1.44MB 高密软盘（A:）进行物理格式化：

```
# fdformat /dev/fd0H1440
```

4.2.2 创建文件系统

接下来，如果要在软盘上创建一个 Ext2 文件系统，可用下面的命令：

```
# mkfs -f ext2 -c /dev/fd0H1440
```

如果打算建立一个 MS-DOS 文件系统，在上述命令中以参数 **msdos** 代替 **ext2**。

4.2.3 挂装文件系统

最后，当磁盘上建立了文件系统之后，要使用这张磁盘，还必须用 **mount** 命令进行挂装。可用以下两个命令之一挂装磁盘：

```
# mount -t ext2 /dev/fd0 /mnt
```

或

```
# mount -t msdos /dev/fd0 /mnt/floppy
```

第一条命令挂装的是一张 ext2 格式的软盘，第二条命令挂装的是一张 DOS 格式的软盘。在上述两条命令中，/mnt/floppy 是系统中一个确定的目录，用作软盘文件系统的挂装点。在使用 **mount** 命令时，挂装点必须存在；如果不存在的话，可以用 **mkdir** 命令来创建。经过这一步操作，使得软盘文件系统看起来就好象在 /mnt 目录之下，软盘上的所有文件都将在驱动器的 /mnt 目录下显示出来。

4.2.4 卸除文件系统

经过以上三步操作，现在可以在这张磁盘上保存文件了。需要注意的是：在 DOS 下，软盘使用完毕，只需从软盘驱动器中直接取出即可，而在 Linux 中，取出磁盘之前首先必须用 **umount** 命令来卸除软盘的文件系统。例如：

```
# umount /dev/fd0
```

上述命令卸除第一软盘 /dev/fd0 的文件系统。此时，用户才可以从软驱中取出软盘。卸除文件系统的目的是，使得系统缓冲区的内容与磁盘上文件系统的实际内容一致。另外，将现有软盘的文件系统从挂装点卸除，就可以在同一点挂装别的文件系统。

4.2.5 其他讨论

经过上面的操作，你会发现在 Linux 中，过去对 A 盘进行的操作现在都用 /mnt 目录代替了。表 4-1 给出了 DOS 命令和对应 Linux 命令的对比。

表 4-1 DOS 命令和对应 Linux 命令的对比

DOS 命令	对应的 Linux 命令
C:/>DIR A:	$ ls /mnt
C:/>A:	$ cd /mnt

如果用户不习惯上面的命令形式，可以考虑使用 **mtools** 软件工具。这是一套命令集，其中的命令形式与 DOS 命令有良好的对应关系。**mtools** 工具的所有命令都以字母“**m**”开头，例如 **mformat**、**mdir**、**mdel** 等。它甚至支持长文件名，但无法设置文件权限。你只需象使用 DOS 命令一样使用即可。

不用说，对于软盘进行的操作同样适用于其他设备，例如，你可能想挂装一个硬盘或 CD-ROM。下面的命令用来挂装一个 CD-ROM：

```
# mount -t iso9660 /dev/cdrom /mnt
```

其中，iso9660 是 CD-ROM 所采用的文件系统。

再次提醒读者，在 Linux 系统中使用完毕可拆装的存储介质，如 CD-ROM、软盘，千万不要忘记卸除文件系统，然后再取盘或换盘，否则这些设备的系统信息与实际情况不同步，会造成很多麻烦。

此外，作为系统管理员，如果希望普通用户也可以挂装或卸除某些设备（如软盘和 CD-ROM），可以在系统的 /etc/fstab 文件中添加 user 选项，允许特定用户对特定设备使用 **mount** 和 **umount** 命令。通常，不应当允许普通用户装卸硬盘，因为这会对系统安全造成极大的威胁。

4.3 备份和恢复

在实际使用 Linux 系统的过程中，及时、经济、方便地对重要信息建立备份十分重要，道理也不言而喻；另外，系统难免会发生一些意外情况，造成重要文件或系统本身的损坏，因此学习一些恢复系统和文件的技术也是必要的。本小节讨论的正是这两方面的内容。

4.3.1 备份

软盘是一种常用的备份媒体。如果系统没有安装磁带驱动器的话，可以采用软盘备份——尽管它的速度很慢，而且不很可靠。另外，还可以用软盘来保存单个文件系统。

用软盘作备份，最简单的方法是使用 **tar** 命令：

```
# tar cvfzM /dev/fd0 /
```

上述命令通过软盘驱动器 /dev/fd0 来对系统作一个完整的备份。**tar** 的 **M** 选项表示，允许建立多卷备份，即当一个软盘写满时，**tar** 将提示换一张新盘。另外一个命令：

```
# tar xvfzM /dev/fd0
```

可用来恢复全部备份，该命令也可用于系统中的磁带驱动器，只需把上述命令中的设备改为 /dev/rmt0。

为整个系统作完整的备份需要耗费大量的时间和系统资源。因此，很多系统管理员采用增量备份的策略，例如，每个月将整个系统备份一次，每个星期只将上星期内改动的文件备份一次。在这种情况下，如果在某个月中系统被破坏了的话，只需要恢复上个月的全部备份。

采用增量备份策略时，可以用 **find** 命令找出那些在某个日期修改过的文件。另外，在 http://sunsite.unc.edu 站点可以找到许多有关管理增量备份的策略和程序。

除了上面介绍的归档性质的备份之外，还可以直接在软盘上创建文件系统，建立备份。这样做的好处是，只要进行挂装软盘的操作，就可以方便地访问其中的数据。

在软盘上建立文件系统很简单，就如同在硬盘上分区一样。例如：

```
# mkfs -f ext2 -c /dev/fd0H1440
```

上述命令在软盘 /dev/fd0 上建立一个 **ext2** 文件系统。文件系统的大小必须与软盘容量相适应，3.5 英寸高密软盘的容量为 1.44MB。

访问软盘时，首先必须用命令 **mount** 挂装文件系统，这一点前面已经介绍过了。一张软盘使用完毕，不要忘记用 **umount** 命令卸除文件系统，然后再从软驱中取出磁盘。在备份过程中，如果需要更换软盘，记住首先卸除第一个软盘，然后再挂装第二个软盘，依此操作。

4.3.2 恢复

恢复系统

有时，系统管理员会面临从一场灾难中恢复系统的问题。例如，管理员忘记了 root 口令，或者意外事故破坏了文件系统。不用害怕。因为通常情况下，大多数问题都可以在不重新安装整个系统的情况下得以解决。前提条件是，系统管理员拥有一张或几张用于紧急恢复的软盘。

所谓紧急恢复软盘，是一个可以独立启动的、不依赖于硬盘的完整的 Linux 系统，其中包含根文件系统、需要的应用例程，以及可启动的内核。例如，安装很多 Linux 套件时用到的引导盘和根盘就可以用作紧急恢复盘。

使用紧急恢复盘十分简单，只须用该盘启动系统，并以 root 登录即可。为了访问硬盘信息，需要以手工方式将硬盘的文件系统挂装到软盘的 /mnt 上。例如：

```
# mount -t ext2 /dev/hda2 /mnt
```

上述命令将硬盘 /dev/hda2 的 ext2 文件系统挂到 /mnt 上——注意：目前 / 在紧急恢复软盘上。经过这一步操作，硬盘上原有的根文件系统也就挂到了 /mnt 下，相应地，硬盘中的 /etc/passwd 文件则在 /mnt/etc/passwd 中。

如果系统管理员是忘记了口令，现在就可以修改 /mnt/etc/passwd 文件，清除 root 帐户的口令字段，然后以硬盘重新启动，用 **passwd** 命令重新设置口令。

其他很多恢复工作也可以在软盘建立的系统上完成。

恢复文件

在 Linux 中没有 **undelete** 命令，因此如果不小心删除了系统中的重要文件，是没有办法取消删除的。但是，我们可以从软盘把相关的文件重新拷贝到硬盘。例如，如果删除了系统中的 /bin/login（与登录有关的文件），只需用软盘启动，将硬盘的根文件系统挂装到 /mnt 上，并使用下列命令：

```
# cp -a /bin/login /mnt/bin/login
```

上述命令完成 login 文件的拷贝工作。其中，**-a** 选择项告诉 **cp** 命令，保留被拷贝文件的访问权限。

当然，如果删除的文件并不是基本的系统文件，而这些文件在紧急恢复软盘上没有，那就没有办法了，如果作了备份，当然可以恢复。

恢复库文件

如果不小心破坏了库文件或 /lib 中的符号链接，那么依赖于这些库文件的命令将无法运行。解决问题的最简单办法是，用紧急恢复盘启动，挂装硬盘的根文件系统，然后在 /mnt/lib 下修复库。

第五章

BASH

使用 Linux 系统时，必须与 shell 程序打交道。shell 有多种，各种 shell 的功能大致类似，它们相当于 MS-DOS 下的 COMMAND.COM 程序，是用户与系统之间的命令解释器，它负责接受用户输入，并将其翻译成操作系统能够理解的指令。同时，shell 还提供了许多定制工作环境的机制，并允许用户编写脚本，组合各种命令，然后象执行普通的 Linux 系统命令一样执行这些组合命令。最常见的两种 shell 名为 C Shell 和 Bourne Shell。

bash（Bourne Again SHell）是自由软件基金会发布的 Bourne Shell 的兼容程序。它包含了许多其他优秀 shell 的良好特性，功能十分全面。很多 Linux 版本都提供 **bash**。

本章以 **bash** 为重点，讲述与 shell 有关的一些概念。实际上，shell 的使用本身已经足以开设一门完整的课程，而本书在短短一章的篇幅内所提供的只能算是一个最初步的简介而已。要真正掌握 **bash** 或其他任何 shell，最重要的一点是多操作，实践出真知。此外提示你一点：作为 DOS 用户，学习 shell 的技巧之一是将它提供的特性和命令与 DOS 中的类似内容作比较，这样就能够较快实现知识的融会贯通。

本章还要粗略地介绍一下系统的初始化。

5.1 概述

在 Linux 系统的 `/etc/passwd` 文件中，如果一名用户对应的记录行中所指定的默认程序是 **bash**，那么当他登录到系统中时，看到的提示符 `$` 就是由 **bash** 程序给出的。

bash 的基本使用很简单，只要在提示符下键入命令名就可以了，一般也不要求掌握很多特别的知识。然而，如果你希望用好 **bash**，并赢得更高的效率，就必须了解通配符、输入/输出重定向、管道等概念，并知道进程和作业是如何管理的。

下面首先介绍 **bash**（也是所有 shell）中经常用到的一些概念和功能。这些概念，读者可能已经有所了解。

请注意，虽然在介绍这些概念或功能时，我们经常和 DOS 进行比较，但实际上这些概念或功能最初是出现在 UNIX 操作系统中的，DOS 只是这些功能的部分实现。从本章的学习中可以看到，Linux 作为一种 UNIX 操作系统，定义有完备而全面的 shell 功能。

5.1.1 通配符，路径名的扩展

DOS 环境下，我们经常用通配符，如“*”和“?”来替代文件名中的多个或一个字

符，此时 DOS 命令会通过一个称为“匹配”的过程，找到正确的文件，完成指定操作。Linux 提供了同样的功能。

在前几章中，Linux 命令中的所有路径名都是用全称表示的。其实，路径名也可用各种缩写方式来表达——即建立规则表达式（regular expression），然后由 shell 加以扩展，得到完整的路径名，再将其传送给命令。例如：

```
$ cp text/* backup
```

上述命令中的特殊字符“*”是通配符的一种，在这里表示任意文件名（以字符“.”开头的文件名称除外）。因此，该命令将 text 目录下的所有文件以原有的文件名复制到 backup 目录。

在 **cp** 命令实际执行之前，shell 已经完成了路径名的扩展，因此 **cp** 命令执行时并不知道“*”字符的存在。当 **cp** 命令读取传送来的命令行参数时，实际看到的可能是这样一条命令：

```
$ cp text/motd text/passwd backup
```

这是因为 shell 已搜索过 text 目录，找到该目录中的所有文件，然后将它们在命令行中一一列出并传送给 **cp** 命令。**cp** 命令能够一次复制多个文件到一个目录，因此上面的命令是有效的。

另一个常见的特殊字符是“?”，它也可以用在文件名中，并由 shell 加以扩展。字符“*”用来代表任意一个或多个字符，而“?”用来代表任何单个字符。请看下列命令的执行情况：

```
$ ls /dev/tty?
/dev/tty0   /dev/tty2   /dev/tty4   /dev/tty6   /dev/tty8
/dev/tty1   /dev/tty3   /dev/tty5   /dev/tty7   /dev/tty9
```

上述命令列出了 tty0 到 tty9 共 10 个文件，它们的文件名都是在 tty 后面跟以一个字符，是 /dev 目录下与 tty? 模式匹配的文件。

除了用字符 ? 或 * 匹配任意单个或多个字符之外，还可以列出字符清单，要求匹配其中给出的任一字符。字符清单列在一对 [ ] 内。例如：

```
$ ls /dev/tty?[234]
/dev/ttys2  /dev/ttyp2  /dev/ttyq2  /dev/ttyr2  /dev/ttyS2
/dev/ttys3  /dev/ttyp3  /dev/ttyq3  /dev/ttyr3  /dev/ttyS3
/dev/ttys4  /dev/ttyp4  /dev/ttyq4  /dev/ttyr4  /dev/ttyS4
```

上面列出的文件名都是在 /dev/tty 后面跟任何一个单字符，然后再跟 2、3、4 中的任何一个数字。

匹配字符的清单还可以用一个范围给出。例如：

```
$ ls /dev/tty?[2-4]
```

数字 2 和 4 之间用字符“-”连接，给出了匹配字符的一个范围。这一命令和上面一条命令完全等效。

另外还有一种扩展路径名的方式，即指定字（而不是字符）的清单，对整个字（而不是对单个字符）进行匹配。在字清单中，字与字之间以“,”字符分隔，所有字列在一对 { } 内。例如，下面的命令在 /usr/tmp 下一次创建了四个子目录。

```
$ mkdir /usr/tmp/{bin, doc, lib, src}
$ ls /usr/tmp
bin doc lib src
```

记住，扩展路径名的工作是由 shell 完成的，shell 所调用的命令看到的是完整的命令参数，不必去处理命令行中出现的特殊字符。如果用户自己编写 shell 下执行的命令，也可以利用 shell 提供的这一服务，而不用考虑在所编写的命令内部去完成路径名的扩展。

5.1.2 引用特殊字符

有时，一条命令的参数中可能包含了部分或全部对 shell 来说有特殊含义的字符，如“*”和“?”等，这时必须通过“引用”技术，通知 shell 暂时忽略被引用字符的特殊含义，将其作为普通字符处理。shell 在引用特殊字符时用到了三种转义字符：反斜杠“\”、单引号“'”和双引号“"”。

将 \ 放在 shell 特殊字符之前，则 shell 忽略该字符原有的特殊含义。用这种方法时，必须在每一个特殊字符之前添加一个转义字符。例如：

```
$ cd
$ mv text/motd text/m\*\?
$ ls text
m*?  passwd
```

将字符串放在一对单引号（'）之间，则单引号之间所有字符的特殊含义都被忽略。例如，下面两条命令的功能是相同的：

```
$ cat 'text/m*?'
$ cat text/m\*\?
```

如果将字符串放在一对双引号（"）之间，本章所讨论到的路径名扩展字符的特殊含义也将被忽略。但是请注意，另外还有一些 **bash** 使用的特殊字符，即使放在双引号中仍然保留着它们的特殊含义。

5.1.3 命令补全

在 **bash** 中，还提供了自动补全命令行的功能。

在送入命令的过程中，你可以随时按 Tab 键，**bash** 会尝试补全已输入的不完整命令。例如，用 **passwd** 命令时，可以这样输入：

```
$ pass<Tab>
```

在提示符 $ 下，输入 **passwd** 命令的部分字符“**pass**”之后，按 Tab 键，**bash** 就会自动进行搜索，寻找以 Tab 键之前的字符串“**pass**”开头的命令名称。**passwd** 是符合条件的唯一命令，因此 **bash** 在输入的字符串“**pass**”后自动加上字符“**wd**”，补全命令名称。

如果用户输入的字符串太短，**bash** 可能无法唯一确定所需要的命令，机器会发出蜂鸣的警告声。此时，再按一次 Tab 键，**bash** 会给出可能的命令清单，用户可以根据提示，在命令中补充输入足够多的字符。这样，当用户再按 Tab 键时，**bash** 能够给出符合要求的唯一命令。

类似地，以文件名作为命令参数时，**bash** 同样也能自动补全文件的名称，用法与命令名的补全一样。例如：

```
$ tail -2 /etc/p<Tab>
```

如果 /etc 目录下以字母 p 开头的文件只有一个，**bash** 会显示这一文件的最后两行；如果 /etc 目录下以字母 p 开头的文件不止一个，**bash** 会发出警告声。此时，用户再按 Tab 键，**bash** 列出该目录中以 p 字母开头的所有文件，用户可以添加足够多的字符，使 **bash** 找到正确的文件，补全命令行，并执行命令。

5.1.4 输出重定向

Linux 提供了许多能有效完成特定任务的简单命令；通过组合这些命令，又可以方便地完成复杂的功能。

在默认情况下，Linux 命令的标准输入设备是键盘，从键盘接受命令的输入；标准输出设备是屏幕，将命令执行的结果送给屏幕显示。同样地，各种错误信息也送到屏幕显示。

然而，有时候需要从一个文件接受输入，或者将命令执行的结果送到一个文件。为达到这一目的，一种方法是编写专用命令，或在命令中添加用来指定文件名的参数；另外一种方法是采用 Linux 和 shell 提供的重定向功能，在必要时将程序的标准输入和输出进行重新定向。例如：

```
$ ls -l /usr/tmp > dir
```

上述命令的前半部分我们很熟悉，它生成/usr/tmp 目录的一个清单。通常情况下，这一清单显示在屏幕上。现在，在 **ls -l** 命令的后面跟了一个字符“>”和一个文件名，于是目录清单将送到指定的文件中，而不在屏幕上显示。

使用重定向功能时，如果指定文件不存在，将新建这一文件；如果指定文件存在，原有的内容将被覆盖。

记住，由于输出重定向是 shell 提供的功能，因此 **ls** 命令并不知道“>”符号的含义和功能。事实上，**ls** 命令甚至不知道“>”符号的存在。shell 负责将“>”符号和后面的文件名称移去，然后将命令行参数送给 **ls** 命令；**ls** 命令则象往常一样将输出的结果送到标准输出设备，而 shell 在 **ls** 命令执行之前，已经将它的标准输出从屏幕重定向到指定的文件，因此 **ls** 命令的输出结果被送到了指定文件。

其他重定向功能也类似。

使用上面的命令，输出结果将覆盖文件原有的内容。但是，有时候希望利用输出重定向将一条命令执行的结果追加到已有的文件之后，这时，可以使用非破坏性的追加重定向操作符“**>>**”，它是将两个“**>**”连在一起。例如：

```
$ ls /usr/tmp >> dir
```

要查看 dir 中的内容，可用下列命令：

```
$ cat dir
```

顺便提一句，如果在 **cat** 命令中缺省文件名，它将从标准输入设备（键盘）接受输入，直到文件结束符为止。在新一行的开始按组合键 Ctrl+d 可以结束从键盘的输入。

知道了这一点，可以用下列命令直接从键盘上将正文送入文件：

```
$ cat > text.file
any text entered here goes into text.file up to
<Ctrl+d>
$
```

上面的 <Ctrl+d> 表示按下组合键 Ctrl+d。

5.1.5 输入重定向

程序的输出可以写入文件。同样地，程序的标准输入也可以来自文件，而不是从键盘输入。输入重定向操作符是“<”。例如：

```
$ wc < /etc/passwd
21   42  775
```

与输出重定向类似，shell 负责处理“<”符号之后的部分，因此 wc 命令看不到文件名 /etc/passwd，而是认为输入仍然来自标准设备——键盘。

还有一种输入重定向的方式，称为 here 文档，它使用的重定向操作符为“<<”。

这种重定向功能告诉 shell，当前命令的标准输入来自命令行中一对分隔符之间的正文。例如：

```
$ wc << delim
> this text forms the content of the heredocument, which
> continues until the end of text delimiter
> delim
    4    17    98
```

注意，向 here 文档输入正文时，每一行前面的符号“>”是由 shell 作为提示符自动给出的，shell 用这个提示符告诉用户，当前命令尚未结束，请用户继续输入内容。

我们可以把任何字符放在重定向操作符“<<”的后面，用作正文开始之前的第一个分隔符，本例中用的是字符串 delim。here 文档的正文到键入第二个分隔符为止。第二个分隔符应出现在新一行的开头。此时，here 文档的正文（不包括开始和结束用到的分隔符）被重定向送给命令 wc，作为它的标准输入。

综合使用输入/输出重定向功能，可以组合命令，在系统中实现单一命令所没有提供的新功能。例如，用下列命令组可以统计目录中的文件数目：

```
$ ls /usr/bin > /tmp/dir
$ wc -w < /tmp/dir
   450
$ rm /tmp/dir
```

第一条命令 ls 列出 /usr/bin 目录中的内容，并将结果重定向输出给 /tmp/dir 文件；第二条命令对 ls 所输出文件的字数（也就是文件名的数目）进行统计，并显示总计 450 个。最后一条命令与输入/输出重定向没什么关系，只是将 /tmp 目录下建立的临时文件删除，免得浪费磁盘空间。

5.1.6 错误重定向

通常情况下，程序的标准输出和错误输出是作为两项不同工作分别处理的。

与程序的标准输出一样，错误输出也可以重定向。例如，要在屏幕上看到程序的正常输出结果，同时将程序所有的错误信息送到一个文件 err.file 中备查，可以用下列命

令来实现：

```
$ ls /usr/tmp 2> err.file
```

其中，操作符“2>”（或追加重定向符“2>>”）用来重定向错误输出设备。

如果要求将标准输出和错误输出送到同一个文件中，需要用到另外一个输出重定向操作符 “&>”。例如：

```
$ ls /usr/tmp &> output.file
```

上述命令的所有输出均送到文件 output.file。

5.1.7 管道

利用 shell 提供的管道（pipe）功能，可以把一条命令的标准输出作为另一个命令的标准输入。读者可以将这种机制看成一种数据管道，一条命令向管道的一端写入数据，而另一条命令从管道的另一端读取数据，这也是“管道”这一名称的由来。在效果上，这等同于通过一个临时文件将两个命令结合在一起。

管道使用的重定向操作符为“|”。可以用管道命令重写 5.1.5 小节最后一个命令，并得到同样的结果：

```
$ ls /usr/bin | wc -w
450
```

由于 Linux 是一个多任务操作系统，在执行上述管道命令时，将同时运行 **ls** 和 **wc** 两个程序，**ls** 的输出将作为 **wc** 的读入。

管道功能不仅可以结合两条命令。管道操作符可以连接任何数目的命令，将相邻的一对命令结合在一起，使前一条命令的输出作为后一条命令的输入，最初的输入经过多个命令的处理，最终得到需要的结果。

5.1.8 历史表

避免重复劳动是一件好事情，Linux 系统为用户从各种角度提供了这种机制，例如前面讲到的命令补全功能。

用户工作时，常常需要重复执行同一组命令。例如，开发和调试程序时，一般都必须不止一遍地作编辑、编译、运行、测试的工作。**bash** 会自动地用一个名为“历史表”（history list）的文件，把这些命令保存起来。在需要的时候，就可以直接使用这些命令，而避免了重复输入。通常，历史表内可以保存 500 行命令，足以容纳几天工作用到的所有命令。

历史表是由 **bash** 自动维护的。每次用户退出登录时，**bash** 将当前的历史表存入一个文件。这个文件在用户的起始目录，缺省名为 .bash_history（请注意，历史表的文件名以字符“.”打头，因此用 **ls** 命令显示这一文件时，必须使用 **-a** 开关才能够看到）。下一次登录，用户开始新一次会话，**bash** 会自动将新的命令追加到已经建立的历史表中。

用 **history** 命令，可以查看历史表内保存的命令行清单。

考虑到历史表很长（达数百行），在下面的示例中利用 **bash** 提供的管道功能将 **history** 命令的输出重定向送给 **tail** 命令，因此，在屏幕上只显示历史表的最后 5 行内容：

```
$ history | tail -5
511 cat > text.file
512 cd..
513 ls -al
514 cd book
515 history | tail -5
```

上面的命令给出了历史表最后 5 行的内容，其中每一行称为一个事件，各有其对应的事件号（如 511、515）。

利用事件号，很容易执行需要的命令。例如，要重复执行上述历史表中的第 515 号命令，只要在 shell 提示符后键入历史命令替换操作符“!”和指定的事件号“515”：

```
$ ! 515
history | tail -5
512 cd..
513 ls -al
514 cd book
515 history | tail -5
516 history | tail -5
```

注意，现在得到的历史表最后 5 行的内容已经改变了。

要重复最后一条命令，只要在 shell 提示符后连续键入两个特殊字符“!”：

```
$ !!
history | tail -5
513 ls -al
514 cd book
515 history | tail -5
516 history | tail -5
517 history | tail -5
```

上面介绍了如何利用事件号来指定要重复的命令。除此之外，还可以由 **bash** 搜索历史表的内容，找出需要的特定命令。例如，在下面的示例中，历史替换操作符“!”之后跟以要搜索的字符串“**ls**”，可以逆向搜索历史表，找出第一个以指定字符串“**ls**”开头的命令并执行：

```
$ !ls
```

对照前面给出的历史表，这里找到的是第 513 号事件 **ls -al** 命令。

另外，还可以对命令行中任何位置（而不仅限于命令行开头）出现指定字符串的事件进行搜索。此时，被搜索的字符串应当放在一对“?”之间——如果第二个“?”之后没有其他内容，则可以将其省略。例如：

```
$ !?234
```

上述命令找到了 5.1.1 小节中用到的命令：

```
ls dev/tty?[234]
```

有时，你可能希望对过去的命令稍加修改，然后再使用。这可以在 **bash** 中做到，只要在命令行后指定一个替换的选项。例如，选项 **:s/old/new** 将所用的历史命令中第一次出现的字符串“old”以“new”取代，下面给出一个具体的例子：

```
$ !?234?:s/23/3
```

这个例子中，实际执行的将是这样一条命令：

```
ls dev/tty?[34]
```

当然，我们只是为了说明历史表的使用方法才举出了这个例子。在实用中，对于简单的命令，或许直接重新输入来得更方便。

5.1.9 命令行编辑

除了选择历史表中的命令重复执行之外，**bash** 还允许用户对命令进行简单的编辑，然后执行修改后的命令。

要编辑命令行，首先必须找到需要的历史命令。这可以通过按键盘上的光标移动键实现。每按一次上移键，**bash** 在历史表中后退并显示上一命令行，按下移键，则向前移动一行。

找到需要的命令后，可以直接按回车重复执行这条命令（此时该命令被加到历史表的最后），也可以在执行之前对这一条命令进行编辑。编辑命令时，可以左右移动光标，在指定位置用 Backspace 键删除光标左侧的字符，然后键入取代的字符；完成修改工作后，按回车键执行这一新命令。

5.1.10 shell 函数

在 shell 中，用户还可以自己定制命令供日后使用。用户定制的命令称为 shell 函数，形式如下：

```
cmd_name( ) {list;}
```

其中，***cmd_name*** 是定制命令的名称，***list*** 则是由若干条简单命令或管道命令组成的一组命令列表，各命令之间以“**;**”分隔。shell 函数的功能实质上是通过函数体内 ***list*** 给出的一系列命令完成的。

例如，定义下列函数：

```
$ ll( ) {ls -l;}
```

就能用定制的命令 **ll** 完成 **ls -l** 同样的功能。

但是，这里有一个问题。定义了函数 **ll** 之后，也许你认为可以用 **ll -a** 来实现命令 **ls -al** 的功能，然而实际情况并非如此。

为了实现这一想法，必须对 **ll** 函数的定义作一些改动：

```
$ ll( ) {ls -l $*;}
```

我们注意到，定义中添加了 **$*** 两个字符。它们表示，如果用户在使用 **ll** 函数时给出了参数或命令开关，则将其插入 **ls -l** 命令，取代 **$*** 字符。经过重新定义，现在下面两条命令是等效的：

```
$ ll -a
$ ls -al
```

有关 shell 函数，讨论暂时告一段落。本书第七章将讲述 shell 脚本的编写和使用，它能够实现的功能更为强大。

5.2 进程

一般意义下，程序这个术语专指存储在磁盘文件内的静态的二进制可执行代码。Linux 系统中，程序文件通常保存在 /bin 或 /usr/bin 目录下。

进程是与程序相对应的一个动态的概念。当程序运行时，它的可执行机器代码与初始化数据从文件复制到内存，并与运行环境相结合，建立一个进程。

Linux 系统自动为每一个进程分配一个唯一的整数值来加以识别，称为进程识别号（PID）。所有的新进程都是从已有的进程中产生出来的，新进程是已有进程的子进程，已有进程为新进程的父进程。一个父进程可以生成多个子进程，而每个子进程又可以生成自己的子进程，这样，在 Linux 机器中运行的所有子进程、父进程、祖父进程构成一个树形层次结构。这个树形层次结构的根进程名为 **init**，它是系统启动时运行的第一个进程，所有其他进程都由它直接或间接产生。**init** 进程的 PID 为 1。

用 **ps** 命令可以查看当前所运行进程的信息。例如：

```
$ ps
FID   TTY   STAT   TIME   COMMAD
325   v01    S      0:00    -bash
359   v01    R      0:00    ps
```

上述信息表明，用户运行了两个进程。第一个进程是用户登录的 **bash** shell，它的 PID 为 325；另外一个进程是正在产生输出的 **ps** 命令，PID 为 359。

在上述输出结果中，TTY、STAT 和 TIME 字段分别表示进程控制的终端、它的系统状态和 CPU 使用时间。另外，在 COMMAND 字段的 **bash** 前面有一个字符“-”，这表明该进程是由系统运行的，而不是用户输入的键盘命令。**ps** 是 shell 响应用户命令而建立的进程。

在 **ps** 命令中加上适当的命令开关可以取得更多信息。例如，用 **-j** 开关可以给出 PPID 字段，显示父进程的 PID：

```
$ ps -j
PPID  PID  PGID  SID  TTY  TPGID  STAT  UID  TIME  COMMSND
   1  325   325  325  v01   393     S   500  0:00    -bash
 325  393   393  325  v01   393     R   500  0:00    ps -j
```

可以清楚地看到，shell 进程的 PPID 值为 1，表明它的父进程是 **init**。另外，shell 是 **ps -j** 进程的父进程。请注意，**ps** 进程的 PID 和上面例子中的不一样。这说明，重复运行同一个程序建立的却是两个不同的进程。

另外两个常用的命令开关是 **-x** 和 **-a**。例如：

```
$ ps -x
```

该命令列出没有终端控制的进程。

```
$ ps -a
```

该命令列出系统中所有用户运行的进程。

5.3 作业管理和虚拟终端

bash 可以同时运行多个命令，它在执行命令时，给每个命令分配一个作业号（job number），通过作业号对命令加以调度和控制。如果有多个命令通过管道行结合在一起，它们被作为一个作业来管理。

另外值得一提的是，Linux 支持多个虚拟终端，即允许一名用户通过同一个键盘和屏幕多次登录到 Linux 系统，同时拥有多个会话，真正实现多任务的同时操作。

5.3.1 后台作业

直到目前为止，我们所讨论的都是 **bash** 的前台命令。也就是说，用户在 shell 提示符下输入命令，然后按回车键，shell 启动一个新进程去执行命令，并在命令执行完毕后送回提示符，直到此时，用户才可以键入下一条命令。这种模式下，在命令运行的过程中，shell 本身处于休眠状态，不接受用户输入新命令，直到现有命令执行完毕。

事实上，作为一种多任务系统的 Linux 可以同时运行 shell 程序和其他命令程序。该模式下，在运行命令的同时，shell 可以立即给出提示符，允许用户输入新的命令。这是非常重要的一项功能，因为在实际操作中，如果执行某一个命令需要花很长时间，用户就可以在这期间去做其他工作，而不必苦苦等待。

用户可以根据需要在 shell 的上述两种执行模式中任选其一。如果你希望 shell 在未执行完某一条命令的情况下立即给出提示符，只要在该命令的后面加一个字符“&”，就可以将它放到后台去执行。例如：

```
$ ls -lR / > /tmp/ls.lR&
[1] 341
$
```

上述命令列出根目录下所有目录的清单，并将结果写入 /tmp/sl.IR 文件，由于这个命令的执行时间很长，因此在命令行后面加上 & 字符，将它放在后台执行。shell 不等待该命令执行完毕即给出提示符。

此时，命令执行的是一个后台作业。注意，shell 给出了这一个命令行的后台作业号 1 和命令的进程识别号 341。

执行后台作业时，shell 给出提示符之后，所有从键盘输入的内容都会被送给 shell。因此，一般情况下后台命令不允许从键盘接受输入，否则其执行将被挂起。

5.3.2 作业管理

shell 保存着当前执行的作业清单。通过作业管理，**bash** 可以将一个进程挂起，或在适当的时候恢复一个进程的执行。

要挂起当前的前台作业，只要按组合键 Ctrl+z。此时，当前作业停止运行，并返回 shell 提示符。例如：

```
$ cat >text.file
<Ctrl+z>
[1]+ Stopped          cat >text.file
```

上面，在执行命令时按 Ctrl+z 组合键，**bash** 将挂起当前运行的 **cat** 命令。

用 **jobs** 命令可以显示 shell 的作业清单信息，包括作业名称、作业号，以及作业当前所处的状态。例如：

```
$ jobs
[1]+ Stopped          cat >text.file
```

要恢复一个进程的执行，有两种方式。一是使用 **fg** 命令，将其放回前台，它将从挂起的位置继续向下执行；另外一种方式，用 **bg** 命令把作业放到后台去执行，此时该命令象普通的后台命令一样执行。

需要提醒读者注意的是，后台命令不允许从键盘接受输入，因此如果把上述 **cat** 命令放到后台去执行，由于它要求从键盘读入数据，因而会自动地被系统再一次挂起：

```
$ bg
[1]+ cat >text.file&
$ jobs
[1]+ Stopped (tty input)     cat >text.file
```

在缺省命令参数的情况下，**fg** 和 **bg** 命令操作最近停止的作业。从上面的 **job** 命令输出中可以看到，最近停止的默认作业号 [1] 后跟有一个“+”字符。

如果用户希望恢复执行作业表中的其他作业，可以在命令中用 % 字符显式地指明：

```
$ fg %1
cat > text.file
```

上述命令将 1 号作业 **cat** 放到前台运行。

有时候，挂起的作业不再需要恢复运行。此时，可以用 **kill** 命令清除（或称“杀死”）作业。例如：

```
$ cat >txt.file
<Ctrl+z>
[1]+ Stopped              cat > text.file
$ jobs
[1]+ Stopped              cat > text.file
$ kill %1
[1]+ Terminated       cat > text.file
```

上述命令序列中，首先按 Ctrl+z 组合键挂起 **cat** 命令，然后用 **jobs** 命令查看当前作业号，最后用 **kill** 命令将挂起的 **cat** 进程结束。

用 **logout** 命令退出系统时，必须保证所有进程已经结束。如果有作业处于被挂起的状态，系统会发出警告信息，同时退出命令失败。例如：

```
$ cat >txt.file
<Ctrl+z>
$ logout
There are stopped jobs.
```

此时，可以重新尝试退出系统，作业将自动结束，用户就能成功地退出。当然，用户也可以先处理被挂起的作业，然后再退出系统。

5.3.3 虚拟终端

Linux 是一个真正的多用户多任务操作系统，可以同时接受多个用户登录，而每个用户又可以各自运行多个进程。

一般情况下，用户是通过与 Linux 机器连接的键盘和屏幕登录到系统的。Linux 支持多个虚拟终端，即允许在一个物理键盘和屏幕上虚拟地建立多个终端。允许使用虚拟终端，也就意味着一名用户可以通过同一个键盘和屏幕多次登录到 Linux 系统，同时拥有多个会话。按键盘上的功能键，用户可以实现在各虚拟终端之间切换，即选择一个虚拟终端与实际的物理设备连接。

为证实这一点，请你先登录到系统中，然后按组合键 Alt+F2，屏幕会给出登录提示符：

```
login:
```

此时，你看到的是第二个虚拟终端，你可以按照前面章节介绍的方式又一次登录到系统中，并进行各种操作。要切换回第一个虚拟终端，只需按 Alt+F1。

通过按 Alt+F1 到 Alt+F6，一个 Linux 系统允许用户访问六个虚拟终端，你可以同时在几个虚拟控制台上工作。例如，开发软件时，用一个终端进行编辑，一个终端编译，另一个终端用来查询信息，每一个虚拟控制台的功能都很强大。当系统第一次启动时，使用默认的 Alt+F1 终端。

使用虚拟终端仍然存在某些限制。首先，虚拟终端只能在系统主机的键盘和屏幕上建立；另外，每一时刻你只能看到一个虚拟终端。但是，虚拟终端毕竟能够带来一种实实在在的多用户的感觉——方便地从一个虚拟终端切换到另一个终端，并开始新的工作。

在 5.2 节中已经讲过，使用命令 **ps** 可以列出正在运行的进程。

其中，第二字段 TTY 的内容是与这一进程相联系的虚拟终端号（V01 到 V06）。通常，当用户退出一次登录时，这一次会话使用的虚拟终端所控制的进程也随之全部结束。

然而，有时 TTY 字段的内容是一个问号“?”，这表示该进程不是在某一个虚拟终端上建立的，因此即使系统中没有用户登录，也能继续运行。

用 **tty** 命令，可以知道登录时使用的是哪一个虚拟终端。例如：

```
$ tty
/dev/tty1
```

这条命令给出了虚拟终端对应的特殊文件的名称，该文件在目录 /dev 下。/dev/tty1 是 1 号虚拟终端对应的特殊文件名。

用户还可以用 **kill** 命令发出信号，强制结束一个自己所控制的进程。命令格式为：

```
$ kill <PID>
```

其中，PID 是要结束的进程 PID 号。某些情况下，这一进程可能会忽视 **kill** 命令的结束信号。在 **kill** 命令中加上命令开关 **-9**，可以发出不允许进程忽视的信号。例如：

```
$ kill -9 <PID>
```

5.4 环境变量

用户常常希望根据自己的习惯和工作情况定义环境，shell 提供了这样的机制。除了使用前面介绍的输入/输出重定向、作业调度、命令行编辑等功能之外，用户还可以编写

简单的 shell 脚本，定制自己的命令。本书第八章将讨论有关 shell 脚本的编写，本小节只介绍 shell 中的环境变量。

对于 shell 来说，环境就是执行命令时用到的所有变量的集合。shell 象许多程序设计语言一样允许定义和使用变量，在其中保存一个字符串作为值。变量一旦定义好，就可以将它放到环境中，成为环境的一部分。

在 Linux 系统中，允许通过设置命令所需的变量来配置环境。由于各种 shell 设置和使用环境的方式存在差异，下面的讨论以 **bash** 为准。

给一个变量赋值很简单。例如：

```
msg1 = "hello the world"
```

这样，变量 msg1 就被赋值“hello the world”。记住，这些变量位于 shell 的内部，也就是说，只有 shell 才能够访问这些变量。

要提取变量的值，只要在变量名之前加一个字符“$”。例如，下面的例子用 **echo** 命令回显变量 **msg1** 的值：

```
# echo $msg1
hello the world
```

上述命令等价于：

```
# echo "hello the world"
hello the world
```

而下面的命令没有在变量名之前使用字符“$”，不能给出变量值：

```
# echo msg1
msg1
```

接下来，具体介绍一个实际的环境变量 PATH。

当你使用某一个命令时，shell 用环境变量 PATH 来寻找对应的可执行文件。例如，命令 **ls** 在目录层次结构中通常位于 /bin/ls 的位置。

而 PATH 变量可以为：

```
/bin:/usr/bin:/usr/local/bin:
```

它是 shell 所要查找的目录列表，每个目录之间用“:”分开。当你使用 **ls** 命令时，shell 首先在 /bin 目录中寻找，如果找不到，接着搜索 /usr/bin。

在很多系统中，可执行文件分布在许多不同的目录，例如 /bin、/usr/bin、/usr/local/bin 等。有了 PATH 参数，用户就不必记住这些命令的可执行文件具体的存储位置，给出命令的完整路径名（如 /usr/bin/cp），而只需要把 PATH 设置成希望 shell 自动查找的目录列表就可以了。

另外，利用环境变量还可以跟踪登录会话等重要信息。例如，HOME 变量用来保存用户的主目录名称：

```
# echo $HOME
/home/jerry
```

事实上，系统在 shell 内部已经定义了许多变量，shell 一旦开始运行，便设置这些变量的值。包括 PATH 和 HOME 在内的部分变量名称和用途列在表 5-1 中。

表 5-1 部分系统变量

变量名称	说明
PS1	保存用作 shell 提示符的字符串。通常，该变量的值为“$”。
PS2	保存用作 shell 第二提示符的字符串。当 shell 发现输入的命令不完整，需要进一步输入内容时使用这一提示符。在讨论 here 文档时，曾用到了第二提示符，那时变量 PS2 的值为“>”。
PATH	保存用“:”分隔的目录路径名列表，给出命令文件的位置。这样，当用户键入命令后，shell 将按 PATH 变量给出的列表顺序搜索这些目录，并执行第一个与所键入命令名一致的可执行文件。
PWD	保存当前工作目录的绝对路径名称。用 cd 命令可以设置 PWD 变量的值。
HOME	保存用户起始目录的路径名。
UID	保存当前用户的用户识别号 UID。注意，虽然 UID 是一个数，但它仍然是以字符串形式保存的。

5.5 系统初始化

DOS 环境下，有两个大家十分熟悉的文件：AUTOEXEX.BAT 和 CONFIG.SYS。启动时，它们用来对系统完成初始化，设置 PATH 和 FILES 等环境变量，有时还运行一个程序或批处理文件。

Linux 系统中，同样有一些初始化文件，它们在用户登录时自动执行设置工作环境。例如，如果你在登录后总是用 **mail** 命令来查看电子邮件，就可以把这个命令放在初始化文件中，这样，它就能自动执行了。

如果你只是希望设置 PATH 和其他环境变量，或者想改变登录时显示的消息，在登录之后自动运行一个程序，可以查看下列几个初始化文件：

```
/etc/issue                  设置登录前显示的消息
/etc/motd                   设置登录前显示的消息
/etc/profile                设置 PATH 和其他环境变量
/etc/bashrc                 设置别名和函数等
/home/jerry/.bashrc         设置你的别名和函数
/home/jerry/.bash_profile   设置环境，并启动你的程序
/home/jerry/.profile            同上
```

注意，某些配置文件和初始化文件（如/etc/inittab 和 /etc/rc）用来初始化主系统，诸如安装文件系统和初始化交换空间，对系统的正常运行起着十分关键的作用，请你在完全掌握其功能之后再对它们作修改。

有关这些系统初始化文件的详细使用，请查阅 **bash** 的联机手册。

第六章

Linux 的 GUI

经过几章的学习，你对 Linux 系统，特别是它的命令解释器 shell 已经有了初步的了解。Linux 是真正的多用户系统，提供了多任务管理、虚拟终端、命令行编辑，以及后面将要介绍的 shell 脚本等诸多功能。

然而，你是不是觉得 Linux 中还缺少了点什么？

对，是图形，Linux 有自己的图形用户界面吗？毕竟，从色彩斑斓的 Windows 世界骤然掉进灰白一片的字符堆中绝对称不上是一件令人兴奋的事情。回忆着往日使用 Windows 时亲切的图标和美丽的动画效果，即使有最令人信服的专家一遍遍地向你论证 Linux 如何强大，你还是会觉得，面对字符界面，做一个忠实的 Linux 用户实在太辛苦了。

其实，你根本用不着遗憾，因为在各种 UNIX 环境中同样可以使用功能强大的图形用户界面，它的名字是 X Window（通常简称为 X）。X 与 UNIX 的关系略微有些类似于 Windows 3.x 与 DOS 之间的关系，但两种情况又有所不同。

透过 X，你可以重见一片艳丽的世界，斑斓而全新的世界。

6.1 X

我们知道，Linux 是一种主要运行在 PC 机器上的 UNIX 操作系统。而 X Window 正是为 UNIX 家族成员服务的。

X Window 是一个功能强大的图形工作环境，最早的 X 系统代码由 MIT 开发，后来它成为在各种 UNIX 平台上图形界面的工业标准。目前，X 的最新版本是 X11R6，有关它的配置及使用的内容已经超出了本书范围，感兴趣的读者可以参阅有关专著。

本章中，我们实际所讨论的 X 名为 XFree86，是 X11R6 专供 Intel 平台运行的一个免费版，目前它的最新版本是 XFree86 3.2.2。XFree86 内包含运行一个图形界面所需要的二进制文件、支撑文件库和工具，可用于包括 Linux 在内的多种 UNIX 系统。下面，我们将把 XFree86 也简称为 X，不考虑它与 X11R6 之间的差别。。

本章将介绍如何在 Linux 系统上安装和配置 XFree86。对于一些细节内容，请你注意参阅定期公布的“XFree86 HOWTO”文档和 Usenet 新闻组 comp.window.x.i386UNIX 中与 XFree86 有关的讨论，以获取更多信息。

6.2 X 和 Windows

在 UNIX 系统中，X 提供了类似于 Windows 的图形环境。然而，X 与 Windows 的设计思想是不同的。

设计 X 时，考虑的主要目标是为 UNIX 工作站提供一个灵活而强大的图形工具；设计 Windows 时，则可能更加关注于使用中的便捷性以及漂亮统一的外观。由于这种思想渊源上的不同，X 与 Windows 之间有诸多区别也就不足为奇了。在此，我们只讨论其中最明显的一个。

使用过 Windows 的用户都知道，无论何时何地在哪一台计算机上使用 Windows，它的外观和使用感觉都是相同的；但是，不同机器上所安装的 X 系统的外观和使用感觉则可能完全不同。

造成这种差别的原因在于，Windows 是一个单一的程序，而 X 是一个客户/服务器系统。其中，X 服务器负责管理如何显示图形界面，如何响应键盘和鼠标的输入，如何建立和终止一个窗口，而它的外观则是由另外一个名为“窗口管理器”的客户组件所确定的。

对于不同的客户组件选择方案，对应着不同的 X 外观。最常见、最简单的窗口管理器为 fvwm。对它进行适当的配置，可以确定图形界面上的许多细节，例如窗口的背景色、窗口边框的颜色、窗口位置和边界的调整方式等等。除了朴素的 fvwm 之外，还有一些更加漂亮的窗口管理器，例如 fvwm2-95 和 The Next Level，它们给 X 提供了类似 Windows 95 的外观，并提供一些附加功能。

总之，对于 X 用户来说，如果使用了不同的窗口管理器，他们看到的实际界面就会完全不同。这种情况在 Windows 用户中是不存在的。

6.3 安装和配置 X

X Window 系统是一种客户/服务器应用程序，服务器程序用来控制键盘、鼠标和屏幕等 I/O 设备，而客户程序是使用服务器程序 I/O 功能的应用程序。一个 X Window 系统可以在一台机器上运行，也可以通过网络在不同的机器上分别运行服务器和客户程序。在服务器和客户之间，通过通信协议传输信息。安装 X Window，需要完成的主要工作就是根据硬件情况选择合适的 X 服务器程序，把它安装到机器上并进行适当的配置。

安装 X 之前，用户首先应当准备好计算机的显示器和显示卡的说明书，因为在配置过程中需要用到显示器的垂直和水平扫描频率、显示卡中的芯片组类型、显存的大小等参数。在安装和配置 X 时，你会发现需要确定的屏幕信息多而复杂，请详细查看，并根据硬件条件进行选择。

一般说来，安装和配置 X，一次成功是很幸运的事情，所以如果你碰上了什么问题，也不用太着急，多尝试几次，有必要的话可以寻求帮助，一定能够成功。

6.3.1 硬件要求

安装 X 之前，首先必须保证你的计算机在硬件上（主要是显示卡）满足要求。

X 并不支持所有的显示芯片组。因此，如果你在市场上购买新的显示卡或购买计算

机时已经带有显示卡，应当向销售商询问有关显示卡的型号和所用芯片组的详细信息，并要求销售商向你提供技术支持，这样，在安装 X 时才能够正确地填入信息，完成系统的配置。如果你无法确切地知道自己所使用的显示芯片，也可以用 XFree86 中的 **SuperProbe** 程序来确定显示卡的芯片组，后面将详细讨论。

表 6-1 列出了 XFree86 支持的 SVGA 芯片组。注意，由于软硬件的发展，X 支持的芯片组在不断增加，因此这里提供的信息并不是完整的。

表 6-1 XFree86 支持的 SVGA 芯片组

类型	具体型号
SVGA 标准芯片	➢ Tseng ET3000, ET4000AX, ET4000/W32 ➢ Western Digital/Paradise PVGA1 ➢ Western Digital WD90C00, WD90C10, WD90C11, WD90C24, WD90C30, WD90C31, WD90C33 ➢ Genoa GVGA ➢ Trident TVGA8800CS, TVGA8900B, TVGA8900C, TVGA8900CL, TVGA9000, TVGA9000i, TVGA9100B, TVGA9200CX, TVGA9320, TVGA9400CX, TVGA9420 ➢ ATI 28800-4, 28800-5, 28800-6, 68800-3, 68800-6, 68800AX, 68800LX, 88800 ➢ NCR 77C22, 77C22E, 77C22E+ ➢ Cirrus Logic GLGD5420, CLGD5422, CLGD5424, CLGD5426, CLGD5428, CLGD5429, CLGD5430, CLGD5434, CLGD6205, CLGD6215, CLGD6225, CLGD6235, CLGD6220 ➢ Compaq AVGA ➢ OAK OTI067, OTI077 ➢ Advance Logic AL2101 ➢ MX MX68000, M680010 ➢ Video 7/Headland Technologies HT216-32
SVGA 加速芯片	➢ 8514/A ➢ ATI Mach8, Mach32 ➢ Cirrus CLGD5426, CLGD5428, CLGD5429, CLGD5430, CLGD5434, CLGD6205, CLGD6215, CLGD6225, CLGD6235, ➢ S3 86C911, 86C924, 86C928, 86C805, 86C805i, 86C864, 86C964 ➢ Western Digital WD90C31, WD90C33 ➢ Weitek P9000 ➢ IIT AGX-014, AGX-015, AGX-016 ➢ Tseng ET 4000/W32, ET 4000/W32i, ET 4000/W32P

采用以上芯片组的显示卡可以在包括 VLB 和 PCI 的任何类型总线上运行。

随着时间的推移，上表所列的内容会越来越丰富，请你同时关注当前 XFree86 版本内附带的信息，其中通常包含着所有支持的显示芯片组列表。

如果你要在 Linux 下安装 XFree86，建议至少使用一台 486 机器，至少拥有 8MB 内存，并注意选用合适的显示卡。为了优化性能，请尽可能选用加速卡。有关 XFree86 系统下不同显示卡的基准等级会定期地在 Usenet 新闻组 `Comp.Windows.X i386UNIX` 和 `Comp.os.linux.misc` 中公布。

6.3.2 获取 X

XFree86 二进制文件的各种版本可以在许多 FTP 站点上找到。例如：

```
tsx-11.mit.edu:/pub/linux/packages 或
sunsite.unc.edu:/pub/Linux/X11
```

在这些站点中，你可以看到各种不同的 X 服务器软件，它们已经被打包为压缩文档，分别适用于不同的显示卡芯片组。你应当根据自己的硬件情况选择合适的 X 服务器。另外，一些重要的配置文件、include 文件、基本的字体文件也是必须的。

当然，如果你已经购买了 Linux 套装软件的光盘，通常其中已经包含 XFree86，此时便不需要通过网络下载 XFree86 了。

6.3.3 安装和配置 XFree86

安装

获取了需要的 XFree86 软件之后，接下来进行安装。

首先，以 root 身份登录到 Linux 系统，建立目录 /usr/X11R6，并用 **gzip** 命令将文件在 /usr/X11R6 中解开。解开文件后，应当根据机器所用显示卡的类型选择一个合适的 X 服务器程序。服务器程序应当安装在 /usr/X11R6/bin 目录下，并与同一目录下的 /usr/X11R6/bin/X 建立符号链接。

例如，你使用的是普通 SVGA 卡，没有选用加速卡，那么应当选择下面的服务器程序进行安装：

```
XF86_SVGA
```

要链接选定的 X 服务器和 /usr/X11R6/bin/X 文件，用以下命令：

```
# ln -sf /usr/X11R6/bin/XF86_SVGA /usr/X11R6/bin/X
```

多数情况下，这一步工作已经由 X Window 的安装脚本完成。

如果要选择其他的 X 服务器，只要将上述命令中的服务器文件改为相应的新名称即可。例如，你希望使用单色显示服务器，则应当选择 XF86_MONO 服务器，并用下面的命令将它与 /usr/X11R6/bin/X 重新建立符号链接：

```
# ln -sf /usr/X11R6/bin/XF86_MONO /usr/X11R6/bir/X
```

如果你无法确定自己使用何种服务器或者不知道显示卡的芯片组，可以运行 /usr/X11R6/bin 目录下的 **SuperProbe** 程序，该程序将试图确定显示芯片组的类型以及其他信息，记录后可留备后用。使用 **SuperProbe** 时会遇到的一个问题是，它有可能造成死机，这一点请用户注意。

配置

选择了适用的 X 服务器后，为使其能够正常运行，还必须对它进行配置。通常情况下，配置 XFree86 并不困难，但是，如果你所使用的硬件的驱动程序仍在开发中，或许会遇到小小的麻烦。

配置 X 服务器是十分重要的一项工作。如果配置不当，有可能使显示器在超出技术规范的状态下运行，导致显示器硬件的永久性损坏。因此，在配置工作开始之前，请准备好显示器和显示卡的说明书，随时备查。设定 X Window 时，会用到显示器的带宽、水平同步、纵向刷新频率等技术参数。

本小节介绍如何生成和编辑一个名为 /etc/XF86config 的配置文件，它包括若干信息段，各段分别定义不同硬件的配置情况。在实际进行配置工作时，你可以参考

XF86config 的联机手册，其中详细介绍了该文件的格式。另外，你还可以查看一个 X 配置的样本文件：

```
/usr/X11R6/lib/X11/XF86config.eg
```

你可以拷贝这一例子，并对它进行适当修改，然后用作自己的配置文件。需要注意的一点是，千万不要原搬照抄地使用这一文件，因为采用与硬件不配套的配置文件可能导致显示器在太高的频率下工作，造成损坏。

建立配置文件的另外一种方法是，使用 **xf86config** 和 **XF86Setup** 程序，它们能够在屏幕上以人机对话的形式提供各种可能的配置选项，帮助用户一步步完成工作。**xf86config** 和 **XF86Setup** 的不同之处在于，前者是字符界面的，而后者是图形界面的。

下面我们首先介绍配置文件 XF86config 本身的组成，通过这一学习，用户就会同时掌握 **xf86config** 和 **XF86Setup** 程序中各种信息和选项的含义，从而掌握 X 服务器的正确配置方法。在阅读过程中，你可以打开 XF86config.eg 文件，为它建立一个副本，然后跟随本书的讲解进行修改。

配置文件 XF86config 是一个文本文件，由七段信息组成。每一段的形式如下所示：

```
Section "<section-name>"
...
...
EndSection
```

其中，*section-name* 是段名，如 File、SeverFlags 等，在 Section "*<section-name>*" 和 EndSection 两行之间是具体的配置信息。

表 6-2 列出了组成 XF86config 文件的各信息段及其含义。请读者注意两点：Xf86config 文件的具体格式随版本的不同，可能会有微小的差别，表中给出了简化后的基本形式；另外，下表中各段内的参数只是我们为讨论方便而随意给出的值，它与你的机器并不匹配，所以千万不能照搬使用。

表 6-2 构成配置文件 XF86config 的信息段。

File 段：

```
Section "File"
    RgbPath "usr/X1R6/lib/X11/rgb"
    FontPath "usr/X1R6/lib/X11/fonts/misc/"
    FontPath "usr/X1R6/lib/X11//fonts/75dpi/"
EndSection
```

本段中的 RgbPath 行设置了 X11R6 RGB 颜色数据库的路径；FontPath 行设置了包含 X11 字体文件的目录路径。

由于 X Window 中有不同的字体文件，因此可能出现多个 FontPath 行。通常，这一段的信息不需要修改，只要保证每一种已经安装的字体都有一个对应的 FontPath 行就可以了

SeverFlags 段：

```
Section "SeverFlags"
EndSection
```

本段用来设置服务器的全局标志。

一般情况下，这段的内容是空的，或者仅仅是--些注释行。但即便如此，本段仍然必须包含在配置文件中。

Keyboard 段：

```
Section "Keyboard"
    Protocol "Standard"
    AutoRepeat 500 5
    ServerNumLock
EndSection
```

本段用来指定键盘的相关参数，如键盘协议类型等。
通常，其中的设置适用于多数情况，不需要修改。

Pointer 段：

```
Section "Pointer"
    Protocol "Mouse Systems"
    Device "/dev/mouse"
EndSection
```

本段用来设置鼠标（或跟踪球）等设备的相关参数，如鼠标协议类型、鼠标的设备特殊文件名称等等。

配置鼠标时，必须考虑选择适当的鼠标协议 Protocol，它不是指鼠标的型号或品牌。常见的鼠标协议包括：BusMouse、Logitech、Mcrosoft、MMSeries、Mouseman、MouseSystems、PS/2、MMHitTab 等等。

另外，Device 定义了访问鼠标的设备特殊文件，必须保证它列出的文件是存在的。

Monoitor 段

```
Section "Moniter"
Identifier "My monitor"
    Bandwidth 60
    HorizSync 30-38,47-50
    VertRefresh 50-90
    ModeLine "640x480" 25 640 664 760 800 480 491 493 525
    ModeLine "800x600" 36 800 824 896 1024 600 601 606 625
    ModeLine "1024x768" 65 1024 1088 1200 1328 768 783 789 818
EndSection
```

本段用来设置显示器的特性，如带宽 Bandwidth、水平同步 HorizSync、纵向刷新频率 VerRefresh、分辨率模式 ModeLine 等等。

Identifier 是用来定义 Monitor 段的标识符，供以后从文件中访问 Monitor 段时使用。

用户应当参考显示器的说明书，保证将这一段的参数设置正确。X 服务器运行时将检查这些参数，以保证在驱动显示器时不超出这里给出的技术规范。

Device 段：

```
Section "Device"
Identifier "My video card"
Chipset "et4000"
VideoRam 1024
Clocks 25.20 28.32 32.50 36.00 40.00 44.90 31.50 37.50
Clocks 50.30 56.70 64.90 72.00 80.00 89.80 63.00 75.10
EndSection
```

本段用来设置显示卡的特性。

初始化时，本段不包含任何内容。此时，只需要为 Identifier 赋值，它是 Device 段的标识符，供以后从文件中访问 Device 段时使用。本段中的其他参数，如显示卡所用的芯片组 Chipset、显存的大小 VideoRam、时钟 Clock 等，必须在使用 X 服务器检测硬件后，根据结果填写添加到本段。

Screen 段：

```
Section "Screen"
    Driver "SVGA"
    Device "My video card"
    Monitor "My monitor"
    Subsection "Display"
        Depth 8
        Virtual 1024 1024
        ViewPort 0 0
        Modes "800x600" "1024x768" "640x480"
        ViewPort 0 0
    Endsection
EndSection
```

本段说明了对某种服务器而言，所使用的显示器和显示卡的组合情况。

其中，Driver 定义了使用的 X 服务器程序，此处为 XF86_SVGA。Device 和 Monitor 是前面定义的相应段的标识符，供访问相应段时使用。

在子段 SubSection "Display" 中定义了在当前显示器和显示卡组合下，X 服务器的一些特性：Depth 代表颜色深度，即每一象素点的位数；Modes 是在 Monitor 段中用 ModeLine 命令定义的显示模式列表，前面的 ModeLine 为"1024x768"、"800x600"和"640x480"，因此这里设置相应的模式行 Modes "800x600" "1024x768" "640x480"，该行中的第一个模式是在 XFree86 启动时的默认值。XFree86 运行时，可以通过按组合键 Ctrl-Alt-Plus（Plus 指 + 键）和 Ctrl-Alt-Minus（Minus 指 - 键）在不同的显示方式之间切换。

Virtual 用来设置虚拟桌面的大小，即允许通过滑动鼠标指针使用比物理屏幕更大的显示区域；ViewPort 用来设置当 XFree86 启动时虚拟桌面左上角的坐标，如果缺省，则实际桌面在虚拟桌面的居中显示。

现在，你的 XF86config 配置文件除了有关显示卡的信息之外，已经基本就绪。下面用 X 服务器来检测其余信息，并将其填入配置文件的 Device 段。可以用下面的命令检测硬件：

```
$ x -probeonly
```

此时屏幕上将显示检测的结果。其中某些行的前面标有“**”，表明它是已经在 XF86config 文件中指定的值，另外一些行的前面标有“--”，表明它是由服务器程序检测硬件找到的值。你可以从中找出需要的信息来填写 X 配置文件的 Device 段，最终完成对 X 服务器的配置工作。

以上介绍了以纯手工方式配置 X 服务器的方法。如果你打算用 **xf86config** 或 **XF86Setup** 程序来完成配置，基本过程是相同的。你只须在 shell 下键入程序名作为命令，屏幕上就会给出提示，你根据所掌握的硬件信息，一步步回答问题就可以了。

6.4 运行 X

XF86config 文件配置完毕，就可以运行 X 服务器了。在 Linux 系统中，启动 X Window 的命令是：

```
$ startx
```

startx 是 **xinit** 的前端命令（在其他 UNIX 系统中，运行 X 的命令是

xinit），它可以启动 X 服务器并运行用户个人目录下 .xinitric 文件中的命令。 .xinitrc 是包含着可执行 X 客户程序的一个 shell 脚本。如果 .xinitrc 文件不存在，则使用系统的默认文件：

```
/usr/X11R6/lib/X11/xinit/xinitrc
```

标准的 .xinitrc 文件如下所示：

```
# ! /bin/sh
xterm      -fn  7x13 bold  -geometry 80x32 + 10 +50 &
xterm      -fn  9x15 bold  -geometry 80x34 + 30 -10 &
oclock     -geometry 70x70 -7+7 &
xsetroot   -solid   midnightblue &
exec twm
```

这一脚本将启动两个 **xterm** 客户，一个 **oclock**，并将根窗口的背景颜色设置为深蓝（midnigntblue），然后启动 **twm**，即窗口管理程序。

我们看到，**twm** 是用 shell 中的 **exec** 命令来执行的。因此，**twm** 进程将替代 **xinit** 进程；而 **twm** 进程一旦退出，X 服务器也将关闭。

（注意，在编写 .xinitrc 文件时，最后一条命令必须以 **exec** 开始，并且将该命令放在前台——即这一句的行尾没有 & 字符，否则，.xinitrc 文件中一旦启动客户，X 服务器就会被关闭。

要退出 **twm**，可以在桌面的背景中按鼠标键，此时弹出一个根菜单，按 Exit Twm 即可。还有另外一种退出 X 服务器的办法，即按组合键 Ctrl-Alt-Backspace，它将直接终止 X 服务器，并退出 X Window 系统。

在上面的例子中，.xinitrc 给出了一种最简单的配置。只须对它稍作改动，就能够配置出非常出色的程序来。可以这么说：初看起来 X 系统十分简单，但如果你根据自己的需要去定制 X，就会发现它的功能十分强大。

最后简要介绍一下 X 系统中最基本的窗口管理器 **fvwm** 和终端模拟程序 **xterm** 作为本章的结束。除了它们之外，现在已经有更出色的 fvwm95、AfterStep 等窗口管理器出现，中文终端 cxterm 也有很多人在使用。感兴趣的读者可以到有关站点下载软件，亲身尝试一下。

6.4.1 fvwm

利用 **fvwm** 窗口管理器，可以为用户提供一个虚拟桌面，并允许用户通过单独的页面管理窗口访问多个桌面。

当 **fvwm** 运行时，它搜索起始目录下的配置文件 .fvwmrc。这个文件可以设置窗口管理器运行环境中的很多细节内容，例如根据用户的爱好定制字体、窗口的背景色、未激活窗口和活动窗口的边框色、在窗口间切换时各窗口的堆叠形式、窗口大小、位置等，它还允许用户对鼠标和键盘操作进行定义，用来操作屏幕上的菜单，移动窗口、改变窗口的大小。

详细内容请参阅 **fvwm** 的联机手册。

6.4.2 xterm

xterm 是 X Window 中最常用的终端模拟程序。

xterm 提供了两个独立窗口，用于同时给出 VT102 和 4014 两种终端模拟。每一个窗口有一组对应的菜单。要激活一个窗口的菜单，只须将鼠标移到该窗口的边界内，然后同时按下键盘上的 Ctrl 以及一个鼠标键。

在 VT102 窗口内有三个菜单，分别对应鼠标的三个键。左键用于显示主菜单，中键用于显示 VT 选项菜单，右键显示 VT 字体菜单。

在 4014 窗口内有两种菜单。主菜单与 VT102 的相同，Tek 选项菜单用来控制显示字符的大小以及显示和激活 VT102 窗口。

详细内容请参阅 **xterm** 的联机手册。

第七章

工具和实用程序

Linux 系统提供了很多使用方便的工具和程序。本章将重点介绍三类，它们分别用来编辑正文、搜索和排序、压缩和解压缩文件。对于其他一些常用命令，也将作简要介绍。

7.1 正文编辑

对于不少读者来说，文字编辑是利用计算机完成的主要工作之一。在 Linux 系统中，有众多的文本编辑器可供选择使用，其中最著名的两个是 **vi** 和 **emacs**，二者都拥有极为广泛的用户群。

选用哪一种编辑器，一开始或许是出于偶然，后来则完全取决于用户的习惯。正如在 Windows 环境下，有的人对 Word 赞不绝口，而另外一些用户则始终对 WPS 青眼有加，理由很简单，“因为我用惯了”。

和 **emacs** 相比，**vi** 编辑器虽然很小，但功能很强，只是很多人觉得使用起来不太方便——而用惯之后却又爱不释手了；**emacs** 则不显得那么原始，而是漂亮得多，其中甚至提供了内置的 Lisp 语言，不过 **emacs** 的文件规模也相对较大。

本书讨论中将以 **vi** 编辑器为主，对 **emacs** 则一带而过。这是由于 **vi** 是 UNIX 系统提供的标准的屏幕编辑程序，如果读者打算通过学习 Linux 来掌握 UNIX 的使用，那么知道 **vi** 的使用方法是必需的。略过 **emacs** 的另一个原因是，它的基本使用比较简单，用不着花费太多笔墨。

7.1.1 vi

vi 是所有 UNIX 系统中最常用的文本编辑器。真正的 **vi** 编辑器是专用产品，不允许直接用于 Linux，但是 **vi** 有多种兼容程序，如 **elvis**、**vim**、**stevie** 和 **nvi**，允许自由地用于包括 Linux 在内的许多系统。这些编辑器的基本功能和使用方法与真正的 **vi** 几乎完全相同，因此在本书中统称为 **vi**。

vi 是一种屏幕编辑程序，屏幕显示的内容是被编辑文件的一个窗口。在编辑过程中，**vi** 只是对文件的副本进行修改，而不直接改动源文件，因此用户可以随时放弃修改的结果，返回原始文件；只有当编辑工作告一段落，用户明确给出命令保存修改结果，**vi** 才用修改后的文本取代原始文件。

和使用普通的命令一样，在 shell 提示符后直接键入程序名称即可打开 **vi** 编辑器：

```
$ vi <file>
```

其中 *<file>* 是待编辑文件的路径名。如果该文件不存在，上述命令新建一个文件。用户在 **vi** 编辑器中工作时，有三种不同的操作模式，它们是：

- 编辑模式
- 插入模式
- 命令模式

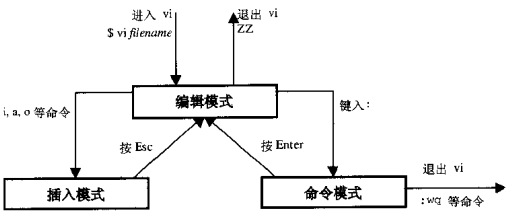


图 7-1 vi 的三种操作模式

用上面的 shell 命令进入的是编辑模式。用户在不同模式下完成不同的工作，图 7-1 给出了在三种模式之间切换的方法，下面分别进行讨论。

编辑模式

运行 **vi** 时，首先进入编辑模式。此时，用户可以利用光标控制键或字母组合键在被编辑的文件中移动光标，并对指定的正文内容进行剪切、粘贴、删除、插入等编辑操作。

在编辑模式下，各种功能是通过按字母组合键来实现的。例如，用户完成所有编辑工作后，要求退出编辑程序，回到 $ 提示符。此时，给出 ZZ 命令（连续按 **Z** 键两次），即可保存编辑结果并退出 **vi**。

下面将逐一讨论在编辑模式下常见功能的具体实现方式。请注意，编辑器区分字母的大、小写。

光标定位

要对正文内容进行修改，首先必须把光标移动到指定位置。移动光标的最简单的方式是按键盘的上、下、左、右箭头键。除了这种最原始的方法之外，用户还可以利用 **vi** 提供的众多字符组合键，在正文中移动光标，迅速到达指定的行或列，实现定位。例如：

```
k、j、h、l      功能分别等同于上、下、左、右箭头键
Ctrl+b          在文件中向上移动一页（相当于 PageUp 键）
Ctrl+f          在文件中向下移动一页（相当于 PageDown 键）
```

H	将光标移到屏幕的最上行（Highest）
*n*H	将光标移到屏幕的第 *n* 行
2H	将光标移到屏幕的第 2 行
M	将光标移到屏幕的中间（Middle）
L	将光标移到屏幕的最下行（Lowest）
*n*L	将光标移到屏幕的倒数第 *n* 行
3L	将光标移到屏幕的倒数第 3 行
w	在指定行内右移光标，到下一个字的开头
e	在指定行内右移光标，到一个字的末尾
b	在指定行内左移光标，到前一个字的开头
0	数字0，左移光标，到本行的开头
$	右移光标，到本行的末尾
^	移动光标，到本行的第一个非空字符

替换和删除

将光标定位于文件内指定位置后，可以用其他字符来替换光标所指向的字符，或从当前光标位置删除一个或多个字符。例如：

rc	用 c 替换光标所指向的当前字符
*n*rc	用 c 替换光标所指向的前 *n* 个字符
5rc	用 c 替换光标所指向的前 5 个字符
x	删除光标所指向的当前字符
*n*x	删除光标所指向的前 *n* 个字符
3x	删除光标所指向的前 3 个字符
dw	删除光标右侧的字
*n*dw	删除光标右侧的 *n* 个字
3dw	删除光标右侧的 3 个字
db	删除光标左侧的字
*n*db	删除光标左侧的 *n* 个字
5db	删除光标左侧的 5 个字
dd	删除光标所在行，并去除空隙
*n*dd	删除 *n* 行内容，并去除空隙
3dd	删除 3 行内容，并去除空隙

粘贴和复制

从正文中删除的内容（如字符、字或行）并没有真正丢失，而是被剪切并复制到了一个内存缓冲区中。用户可将其粘贴到正文中的指定位置。完成这一操作的命令是：

p	小写字母 p，将缓冲区的内容粘贴到光标的后面
P	大写字母 P，将缓冲区的内容粘贴到光标的前面

如果缓冲区的内容是字符或字，直接粘贴在光标的前面或后面；如果缓冲区的内容为整行正文，则粘贴在当前光标所在行的上一行或下一行。

注意上述两个命令中字母的大小写。**vi** 编辑器经常以一对大、小写字母（如 p 和 P）来提供一对相似的功能。通常，小写命令在光标的后面进行操作，大写命令在光标的前面进行操作。

有时需要复制一段正文到新位置，同时保留原有位置的内容。这种情况下，首先应当把指定内容复制（而不是剪切）到内存缓冲区。完成这一操作的命令是：

yy	复制当前行到内存缓冲区
*n*yy	复制 *n* 行内容到内存缓冲区
5yy	复制 5 行内容到内存缓冲区

搜索字符串

和许多先进的编辑器一样，**vi** 提供了强大的字符串搜索功能。要查找文件中指定字或短语出现的位置，可以用 **vi** 直接进行搜索，而不必以手工方式进行。搜索方法是：键入字符 / ，后面跟以要搜索的字符串，然后按回车键。编辑程序执行正向搜索（即朝文件末尾方向），并在找到指定字符串后，将光标停到该字符串的开头；键入 n 命令可以继续执行搜索，找出这一字符串下次出现的位置。用字符 ? 取代 / ，可以实现反向搜索（朝文件开头方向）。例如：

/*str1*	正向搜索字符串 *str1*
n	继续搜索，找出 *str1* 字符串下次出现的位置
?*str2*	反向搜索字符串 *str2*

无论搜索方向如何，当到达文件末尾或开头时，搜索工作会循环到文件的另一端并继续执行。

撤销和重复

俗话说，人难免犯错误，关键是如何改正错误。在编辑文档的过程中，为消除某个错误命令造成的后果，可以用撤消命令。另外，如果用户希望在新的光标位置重复前面执行过的命令，可用重复命令。

u	撤消前一条命令的结果
.	重复最后一条修改正文的命令

插入模式

读者已经看到，在 **vi** 的编辑模式下，输入的字符组合被视为命令。因此，如果要在正文中输入具体的文字内容，就必须暂时离开编辑方式，进入插入模式。在插入模式下，用户可以输入内容，将字符“插入”正文。完成输入正文的工作后，再切换回到编辑模式。

进入插入模式的方法有多种，下面立刻就要介绍；退出插入模式的方法是，按 ESC 键或组合键 Ctrl+[。

进入插入模式

在编辑模式下正确定位光标之后，可用以下命令切换到插入模式：

i	在光标左侧输入正文
a	在光标右侧输入正文
o	在光标所在行的下一行增添新行
O	在光标所在行的上一行增添新行
I	在光标所在行的开头输入正文
A	在光标所在行的末尾输入正文

正文替换

上面介绍了几种切换到插入模式的简单方法。另外还有一些命令，它们允许在进入插入模式之前首先删去一段正文，从而实现正文的替换。这些命令包括：

s	用输入的正文替换光标所指向的字符
*n*s	用输入的正文替换光标右侧 *n* 个字符
cw	用输入的正文替换光标右侧的字
*n*cw	用输入的正文替换光标右侧的 *n* 个字
cb	用输入的正文替换光标左侧的字
*n*cb	用输入的正文替换光标左侧的 *n* 个字
cd	用输入的正文替换光标的所在行
*n*cd	用输入的正文替换光标下面的 *n* 行
c$	用输入的正文替换从光标开始到本行末尾的所有字符
c0	用输入的正文替换从本行开头到光标的所有字符

命令模式

在 **vi** 的命令模式下，可以使用复杂的命令。在编辑模式下键入“:”，光标就跳到屏幕最后一行，并在那里显示冒号，此时已进入命令模式。命令模式又称“末行模式”，用户输入的内容均显示在屏幕的最后一行，按回车键，**vi** 执行命令。

退出命令

在编辑模式下可以用 ZZ 命令退出 **vi** 编辑程序，该命令保存对正文所作的修改，覆盖原始文件。如果只需要退出编辑程序，而不打算保存编辑的内容，可用下面的命令：

:q	在未作修改的情况下退出
:q!	放弃所有修改，退出编辑程序

行号与文件

编辑中的每一行正文都有自己的行号，用下列命令可以移动光标到指定行：

:*n*	将光标移到第 *n* 行

命令模式下，可以规定命令操作的行号范围。数值用来指定绝对行号；字符“.”表示光标所在行的行号；字符“$”表示正文最后一行的行号；简单的表达式，例如“.+5”表示当前行往下的第 5 行。例如：

:345	将光标移到第 345 行
:345w *file*	将第 345 行写入 *file* 文件
:3,5w *file*	将第 3 行至第 5 行写入 *file* 文件
:1,.w *file*	将第 1 行至当前行写入 *file* 文件
:.,$w *file*	将当前行至最后一行写入 *file* 文件
:.,.+5w *file*	从当前行开始将 6 行内容写入 *file* 文件
:1,$w *file*	将所有内容写入 *file* 文件，相当于 :w *file* 命令

在命令模式下，允许从文件中读取正文，或将正文写入文件。例如：

:w	将编辑的内容写入原始文件，用来保存编辑的中间结果
:wq	将编辑的内容写入原始文件并退出编辑程序（相当于 ZZ 命令）

```
:w file          将编辑的内容写入 file 文件，保持原有文件的内容不变
:a,bw file       将第 a 行至第 b 行的内容写入 file 文件
:r file          读取 file 文件的内容，插入当前光标所在行的后面
:e file          编辑新文件 file 代替原有内容
:f file          将当前文件重命名为 file
:f               打印当前文件名称和状态，如文件的行数、光标所在的行号等
```

字符串搜索

给出一个字符串，可以通过搜索该字符串到达指定行。如果希望进行正向搜索，将待搜索的字符串置于两个“/”之间；如果希望反向搜索，则将字符串放在两个“?”之间。例如：

```
:/str/           正向搜索，将光标移到下一个包含字符串 str 的行
:?str?           反向搜索，将光标移到上一个包含字符串 str 的行
:/str/w file     正向搜索，并将第一个包含字符串 str 的行写入 file 文件
:/str1/,/str2/w file     正向搜索，并将包含字符串 str1 的行至包含字符串 str2 的行写
                         入 file 文件
```

规则表达式

vi 中，可以在字符串内使用特殊字符，构成规则表达式，执行复杂的搜索。例如，下列命令包含特殊字符“ ^ ”：

```
:/^struct/       搜索一行正文，在正文的开头包含字符串 struct
```

而简单命令 :/struct/ 只能找出在行中任意位置包含字符串 struct 的第一行。类似地，

```
:/struct$/       搜索一行正文，在正文的末尾包含字符串 struct
```

常见的特殊字符包括：

```
^                置于待搜索的字符串之前，匹配行首的字
$                置于待搜索的字符串之后，匹配行末的字
\<               匹配一个字的字头
\>               匹配一个字的字尾
.                匹配任意单个正文字符
[str]            匹配字符串 str 中的任意单个字符
[^str]           匹配不在字符串 str 中的任意单个字符
[a-c]            匹配从 a 到 c 之间的任一字符
*                匹配前一个字符的 0 次或多次出现
\                忽略特殊字符的特殊含义，将其看作普通字符
```

正文替换

利用 :s 命令可以实现字符串的替换。具体的用法包括：

```
:s/str1/str2/        用字符串 str2 替换行中首次出现的字符串 str1
:s/str1/str2/g       用字符串 str2 替换行中所有出现的字符串 str1
:.,$ s/str1/str2/g   用字符串 str2 替换正文当前行到末尾所有出现的字符串 str1
:1,$ s/str1/str2/g   用字符串 str2 替换正文中所有出现的字符串 str1
```

```
:g/str1/s//str2/g    功能同上
```

从上述替换命令可以看到：g 放在命令末尾，表示对搜索字符串的每次出现进行替换；不加 g，表示只对搜索字符串的首次出现进行替换；g 放在命令开头，表示对正文中所有包含搜索字符串的行进行替换操作。

删除正文

在命令模式下，同样可以删除正文中的内容。例如：

```
:d                    删除光标所在行
:3d                   删除 3 行
:.,$d                 删除当前行至正文的末尾
:/str1/,/str2/d       删除从字符串 str1 到 str2 的所有行
```

vi 的功能选项

为控制不同的编辑功能，**vi** 提供了很多内部选项。利用 :set 命令可以设置选项。基本语法为：

```
:set option           设置选项 option
```

常见的功能选项包括：

```
autoindent            设置该选项，则正文自动缩进
ignorecase            设置该选项，则忽略规则表达式中大小写字母的区别
number                设置该选项，则显示正文行号
ruler                 设置该选项，则在屏幕底部显示光标所在行、列的位置
tabstop               设置按 Tab 键跳过的空格数。例如 :set tabstop=n，n 默认值为
8
```

shell 切换

在编辑正文时，利用 **vi** 命令模式下提供的 shell 切换命令，无须退出 **vi** 即可执行 Linux 命令，十分方便。语法格式为：

```
:! command            执行完 shell 命令 command 后回到 vi
```

7.1.2 GNU Emacs 简介

除了 **vi** 之外，Linux 环境下最常用的另外一个编辑器是 GNU Emacs，在此只作简单介绍。

GNU Emacs 是遵循 GNU 规范的编辑器。它的名称来自 Editor MACroS 的缩写，最早是由 GNU 的发起人 Stallman 在 MIT 的 PDP-10 机器上为文本编辑器 TECO 编写的一套宏命令。在最新的版本中，Emacs 集成了多国语言处理工具 MULE（MULtilingual Enhancement to GNU Emacs），为中国用户在 UNIX（以及 Linux）环境下编辑中文文件提供了强有力的工具。

Emacs 的使用范围很广，从 UNIX 平台到 DOS、OS/2 和 Windows NT 系统。由于它功能强大，界面出色，受到各类用户的广泛欢迎。用户可以利用 Emacs 调试程序、收发电子邮件、阅读网络新闻组、使用计算器，Emacs 还内置了 Lisp 语言，并提供拼写

检查、详细的在线帮助等功能，允许用户定制工作环境。

Emacs 是 Slackware 和 Red Hat 默认配置的程序。机器中安装了 Emacs 之后，在命令行上键入 **emacs** 即可将其打开。

如果读者有意快速了解 Emacs 的情况及基本使用，可以输入命令 Ctrl-h t，进入用户自学状态，参考 Emacs 本身附带的教材。在此限于篇幅，对 Emacs 的具体使用就不详细讨论了。

7.2 搜索和排序

所谓搜索，是在文件中找出包含特定字符串的行，或在清单中找出特定项；而排序是将正文中的行按指定顺序排列。

一般情况下，实现上述功能的命令是从标准输入设备（键盘）接收输入，将结果输出到标准输出设备（屏幕）。然而，我们可以利用它们构造出复杂的管道命令，用第一条命令读标准输入，将初步处理得到的结果通过管道作为下一条命令的输入，再进行第二步处理，然后将最终结果送到标准输出。

下面，对两类命令分别作具体介绍。

7.2.1 搜索

grep

首先介绍搜索程序 **grep**。在 **grep** 命令的输入中可以指定待搜索字符串的模式，搜索结果（即所有包含给定字符串模式的行）组成 **grep** 命令的输出。

例如，要在 /etc/passwd 文件中找出 carey 用户的信息，可以用用户名“carey”为字符串模式进行搜索：

```
$ grep carey /etc/passwd
carey: Ytla4ffkG2r02: 501: 500: : /usr1/carey: /bin/bash
```

搜索结果表明，系统中有一个登录名为 carey 的用户。

但是，上面这条简单的 **grep** 命令搜索的是整个文件，而不是 **passwd** 文件中每一行用户名所在的特定字段，因此，用简单的 **grep** 命令得到的结果有时并不是需要的。例如，在 **passwd** 文件的其他字段也碰巧找到了字母组合 carey，然而它却不是一个用户名。

为了增强命令的功能，**grep** 和 **vi** 一样也使用了规则表达式。

在 **grep** 命令的规则表达式中，使用的基本特殊字符集与 **vi** 所使用的相同；另外，**grep** 命令加上 **-E** 开关之后，还可以使用一组扩充的特殊字符集。表 7-1 和表 7-2 分别列出了 **grep** 使用的基本特殊字符和扩充特殊字符。

表 7-3 列出了 **grep** 的一些命令开关。

表 7-1　基本特殊字符

字符	功能
^	匹配行首的字
$	匹配行末的字
\<	匹配一个字的字头
\>	匹配一个字的字尾
.	匹配任意单个正文字符
[str]	匹配字符串 str 中的任意单个字符
[^str]	匹配不在字符串 str 中的任意单个字符
[a-c]	匹配从 a 到 c 之间的任一字符
*	匹配前一个字符的 0 次或多次出现
\	抑制后面的特殊字符的特殊含义，将其看作普通字符

表 7-2　扩充的特殊字符

字符	功能
+	重复匹配前一项 1 次以上
?	重复匹配前一项 0 次或 1 次
{j}	重复匹配前一项 j 次
{j, }	重复匹配前一项 j 次以上
{, k}	重复匹配前一项最多 k 次
{j, k}	重复匹配前一项 j 到 k 次
s | t	匹配 s 或 t 中的一项
(exp)	将表达式 exp 作为单项处理

表 7-3　grep 的命令开关

命令开关	功能
-E	用扩充规则表达式进行模式匹配
-i	不区分大小写
-n	在每一输出行前显示文件内的行号
-q	与其他命令一起使用时，抑制输出显示
-s	抑制文件的出错信息
-num	在每一匹配行前后各显示 num 行

注意：**vi** 和 **grep** 使用特殊字符的情形有所不同。在 **vi** 中，输入的任何字符都只能被编辑器看到，而 **grep** 所使用的任何参数首先必须经过 shell。由于规则表达式中用到的某些特殊字符对于 shell 来说同样具有特殊含义，因此必须通过引用要求 shell 抑制这些字符的特殊含义。关于引用的具体方法，前面第五章中有讨论。

回到前面讨论的例子。现在我们可以利用特殊 ^ 字符强制 **grep** 命令只在 passwd 文件每行的开头寻找特定的字符串模式。

```
$ grep '^carey' /etc/passwd
```

此时，除非在 passwd 文件每一行的第一字段出现 carey 字符串，否则 **grep** 不会给出结果。

注意，这里用单引号把包括特殊字符 ^ 在内的整个搜索模式括起来。shell 会将单引号去掉，将搜索模式送给 **grep** 命令。如果 **grep** 命令不产生输出，表示 passwd 文件中没有以 carey 开头的行。

最后讲述一个稍复杂的例子，搜索 **passwd** 文件中没有设置口令的记录行。

显然，如果某一行的第 2 字段（第 1，2 个冒号之间）是空的，就表示该帐号没有设置口令。请看下列命令：

```
$ grep -E '^[^:]+::' /etc/passwd
sync::2:2::/:/bin/sync
mtos::9876:9876:student login:/mtos/home:/bin/bash
```

输出结果表明，用户 sync 和 mtos 没有设置口令。

上述命令所使用的搜索模式为：

```
'^[^:]+::'
```

开始和结束的单引号用来引用中间出现的特殊字符，要求 shell 将引号内的字符串传给 **grep** 命令。**grep** 所看到的搜索模式为：

```
^[^:]+::
```

搜索时，记录行的开头用第一个 ^ 找到，空的口令字段用最后的一对冒号“::”来匹配，而 [^:]+ 可以对非冒号字符匹配一次或多次，这里正好用来匹配用户登录名。

另外，由于 + 是扩充的特殊字符，在 **grep** 命令中用到了 **-E** 开关。

find

find 命令用来彻底检查树形目录层次结构。它从指定的起点开始遍历目录的各分支，检查所有结点。下面举一个简单的例子。

首先用 **cd** 命令将当前目录切换为用户的起始目录：

```
$ cd
```

然后用 **find** 命令列出起始目录下的所有文件和子目录的清单：

```
$ find . -print
./backup
./backup/motd.bak
./backup/passwd.bak
./text
./text/passwd
./text/motd
```

上面的 **find** 命令用到两个参数：. 和 **-print**，表示搜索当前目录，显示其中发现的所有文件和子目录名称。

默认情况下，**find** 命令的搜索结果送到标准输出设备（屏幕）。利用 shell 提供的管道功能，也可以将输出送到其他命令作为输入。例如：

```
find . -print | grep passwd
./backup/passwd.bak
./text/passwd
```

在上面的管道命令行中，**find** 执行的结果输出给 **grep** 命令作为输入，然后由 **grep** 找出带有字符串 passwd 的文件或目录名。

由于在 **find** 命令中允许指定一组操作，对发现的文件和子目录进行操作，因此上述功能也可以用 **find** 命令单独完成，无须借助 **grep**。下面作详细介绍。

find 命令的一般格式如下：

```
find <pathname> -expressions
```

其中，*pathname* 是所要搜索的目录路径名列表，**find** 命令彻底搜索这些目录，产生文件名输出，*expressions* 是若干表达式，用来定义对文件进行的操作。如果不指定 *pathname*，默认值是当前目录；不指定 *-expression*，默认值是 *-print*。例如：

```
$ find
./backup
./backup/motd.bak
./backup/passwd.bak
./text
./text/passwd
./text/motd
```

我们注意到，这条命令的输出与前面的完全相同。

find 命令使用的表达式有三种：选项表达式、条件表达式和操作表达式。经过逻辑运算，每个表达式返回值“真”或“假”。如果 **find** 命令中有多个表达式组成了复合表达式，各表达式之间用逻辑操作符控制其执行顺序，复合表达式的值是最后实际求值的表达式的返回值。如果 **find** 命令用到两个表达式 *e1* 和 *e2*，则执行情况为：

e1 -a e2	仅当 *e1* 为真时，对 *e2* 求值
e1 e2	同上
e1 -o e2	仅当 *e1* 为假时，对 *e2* 求值
e1, e2	先 *e1* 后 *e2*，对两个表达式求值

如果没有括号 ( ) 改变求值顺序，复合表达式的求值按自左至右的顺序进行。因此，下面两个复合表达式的求值顺序完全相同：

```
e1 -o e2 -a e3
(e1 -o e2) -a e3
```

加上括号可以改变求值顺序：

```
e1 -o (e2 -a e3)
```

表达式中还可以使用逻辑非操作符：

```
! e1
```

如果 *e1* 为假，则 *!e1* 的值为真；反之，如果 *e1* 真，则 *!e1* 为假。

表 7-4 列出了 **find** 命令中用到的一些表达式、类型以及返回值。

在用 **find** 命令搜索文件和目录时，会受到文件和目录访问权限的约束。搜索过程中，如果碰到不允许访问的子目录，**find** 会向标准输出设备发出一条错误信息。

默认情况下，标准输出设备是屏幕，因此如果 **find** 命令使用了 -print 选项，可能会有大量出错信息打印在屏幕上。对于用户来说，这些信息通常毫无用处。

表 7-4 find 命令用到的表达式

表达式	类型	说明
-mount	选项表达式	防止 find 命令的搜索范围超出当前的文件系统。返回值通常为真。
-group grp	条件表达式	检查当前文件的 GID 是否与 grp 相同。如果二者一致，返回值为真，否则返回值为假。
-name pattern	条件表达式	检查文件名是否与 pattern 模式一致。pattern 可以用规则表达式给出。如果文件名与 pattern 一致，返回值为真，否则为假。
-type t	条件表达式	检查当前文件的类型是否为 t。目录的 t 值可以为 d，普通文件的 t 值为 f，链接的 t 值为 l，等等。如果当前文件的类型是 t 返回值为真，否则为假。
-user usr	条件表达式	检查当前文件属主 UID 是否是 usr。如果二者一致，返回值为真，否则为假。
-exec cmd	操作表达式	执行 cmd 命令。如果成功执行了 cmd 命令，返回值为真，否则为假。
-print	操作表达式	将当前文件名送到标准输出设备。返回值通常为真。

为了避免不必要的输出，可以把输出重定向到一个设备特殊文件 /dev/null。该文件类似于一个下水道，将没用的信息写入该文件，相当于扔掉这些内容。

最后讨论一个稍微复杂一些的例子，寻找一个目录子树下所有包含特定字符串“mycroft”的普通文件，给出它们的路径名清单。

要实现这一目标，一种办法是用 **find** 命令生成路径名，然后用 **grep** 命令检查这些文件，看文件名是否包含指定的字符串，然后显示所有包含指定字符串的文件路径名。用这种办法时，为防止 **grep** 命令输出有用信息的同时在屏幕显示错误信息，我们必须用重定向功能把错误信息重新定向到某个文件，如 /dev/null，这是比较麻烦的。

事实上，**grep** 命令提供了更简便的办法，用命令开关抑制错误输出：

```
$ find /etc -type f -exec grep -q -s mycroft {}\; -print
/etc/ HOSTNAME
/etc/hosts
/etc/lilo.conf
```

上述命令行中，**find** 命令产生 /etc 目录下所有路径名的清单，然后对第 1 表达式 -type f 求值。如果当前路径名属于普通文件，送回真值。

接着，对所有普通文件执行第 2 表达式 -exec grep -q -s mycroft {}\;，在每个普通文件中搜索 mycroft 字符串。在执行 **grep** 命令时，{} 已经用当前路径名代替，**-q** 和 **-s** 开关分别用来使 **grep** 命令不显示标准输出和出错信息。这一表达式用来产生真值或假值，决定是否执行第 3 表达式。

第 2 表达式中的反斜杠 \ 用来引用特殊字符（;），使 **shell** 不加改变地将它传送给 **find** 命令，如果第 2 表达式返回真值，则执行 -print 表达式，它的作用是显示当前路径名。

7.2.2　排序

sort

sort 是一种使用灵活的排序命令，它可以根据命令指定的顺序对输入的正文行进行排序，并将结果送到标准输出设备。**sort** 命令的输入可以来自命令行中指定的任何文件；如果未指定文件，则从标准输入设备接收输入。

sort 命令将每一个输入行看作若干字段的集合，以一个或多个字段作为排序键（sort key）对输入行进行排序操作。默认的字段分隔符为空格、Tab 等空白字符。用户可以根据需要选择其他不同的字段分隔符，由 **sort** 对文件进行不同类型的排序。

为便于讨论，本小节的例子中使用了一个虚构的口令文件 pw.test，它的内容如下所示（注意，为了使读者更容易看清楚文件的内容，本小节排版时在每条记录的各个冒号 : 后添加了空白，这样行与行之间能够对齐。在 pw.test 文件中这些空白实际是没有的，每一条记录就是一个连续不间断的字符串）。

```
root:   awmku76tr43d6:  0:    0:  :  /root/:          /bin/sh
pc:     bdhd74hs9jh3h:  500:  50:  :  /usr1/pc:       /bin/bash
carey:  esJ9ohd89I:     501:  50:  :  /usr1/carey:    /bin/bash
mot:    dhjd83kjdJS6D:  1500: 60:  :  /usr1/mot:      /bin/bash
grex:   cj8AjoWE8h8fs:  1500: 60:  :  /usr1/mot:      /bin/sh
```

由于 pw.test 文件的行中没有空白字符（见上面的解释），因而每一行被作为一个字段。用 **sort** 对该文件排序时，每一行看成一个字，并按字母顺序排列：

```
$ sort pw.test
carey:  esJ9ohd89I:     501:  50:  :  /usr1/carey:    /bin/bash
grex:   cj8AjoWE8h8fs:  1500: 60:  :  /usr1/mot:      /bin/sh
mot:    dhjd83kjdJS6D:  1500: 60:  :  /usr1/mot:      /bin/bash
pc:     bdhd74hs9jh3h:  500:  50:  :  /usr1/pc:       /bin/bash
root:   awmku76tr43d6:  0:    0:  :  /root/:         /bin/sh
```

为了执行复杂的搜索，在 **sort** 命令中可以使用许多命令开关。表 7-5 列出了几个最有用的开关：

表 7-5　sort 命令的常用开关

命令开关	功能
-b	忽视排序键前的空格字符
-f	不区分字母的大小写
-n	将排序键看作数字（默认情况下看作字符）
-r	从高到低进行排序（默认情况下是从低到高排序）
-o *file*	将排序结果输出到 file（默认情况下是标准输出设备）
-t *s*	用 s 作为字段分隔符（默认情况下是空格、Tab 等空白字符）

sort 命令在默认情况下将排序结果送到标准输出设备屏幕显示。如果必要，也可以将结果输出重定向到管道或文件。例如，用 -o 开关，可以将 file1 排序后的结果输出到 file2 文件：

```
$ sort file1 -o file2
```

除了上述命令开关，还允许指定复杂的排序键。例如：

```
-k s1,s2          用第 s1 字段到第 s2-1 字段作为排序键。
```

其中，*s1* 和 *s2* 为字段说明符，可以用 *f.c* 的形式给出，*f* 是字段号，*c* 是该字段中的字符位置，*f* 和 *c* 从 1 开始计。另外，一条 **sort** 命令允许指定多个 **-k** 开关。请看下面几个排序键的例子：

```
-k3               排序键从第 3 字段开始到行的结束
-k3,6             排序键是由第3，4，5字段
-k4,5 -k1,3       排序键是第 4 字段和第 1，2 字段
-k3.3,4           排序键是去除前 2 个字符后的第 3 字段
-k3.2,3,6         排序键是第 3 字段中的第 2、3、4、5 字符
```

知道了命令开关的功能，我们就可以对 pw.test 文件进行复杂的排序操作。首先用 -t：命令开关将字段分隔符改为“:”，这样 **sort** 就可以辨认每一行记录中的各个字段了。

下面，以第 3 字段 UID 作为排序键对文件进行排序：

```
$ sort -t: -k3,4 pw.test
root:    awmku76tr43df:  0:      0:  :   /root/:          /bin/sh
grex:    cj8AjoWE8h8fs:  1500:   60:  :   /usr1/mot:       /bin/sh
mot:     dhjd83kjdJS6D:  1500:   60:  :   /usr1/mot:       /bin/bash
pc:      bdhd74hs9jh3h:  500:    50:  :   /usr1/pc:        /bin/bash
carey:   esJ9ohd89I:     501:    50:  :   /usr1/carey:     /bin/bash
```

默认情况下，如果两行的排序键的值相同，将则将它们整行作为一个字进行排序。上例中，第 3 行和第 4 行排序键的值都是 1500，因此 **sort** 转而比较这两行的行首字母，并将 grex 所在行排在前面。

重复使用 -k 开关设置多个排序键的目的是，如果第一个排序键不能解决问题，依次使用第二个、第三个排序键。例如下面的例子指定，如果用第 3 字段无法确定行序，则用第 7 字段进行排序：

```
$ sort -t: -k3,4 -k7 pw.test
root:    awmku76tr43d6:  0:      0:  :   /root/:          /bin/sh
mot:     dhjd83kjdJS6D:  1500:   60:  :   /usr1/mot:       /bin/bash
grex:    cj8AjoWE8h8fs:  1500:   60:  :   /usr1/mot:       /bin/sh
pc:      bdhd74hs9jh3h:  500:    50:  :   /usr1/pc:        /bin/bash
carey:   esJ9ohd89I:     501:    50:  :   /usr1/carey:     /bin/bash
```

读者可能注意到，上面两个例子似乎并没有按 UID 值正确进行排序。因为这一字段现在的顺序是：0，1500，1500，500，501。

出现这一问题的原因很简单，因为 **sort** 并没有把 UID 作为数，而是作为字符串来处理的。因此，0 排在 1 前面，1 在 5 前面，得到现在的顺序。如果用户希望按照数值的大小排序，应当使用 -n 开关。此时，**sort** 按数字的大小顺序对字段进行排序：

```
$ sort -t: -n -k3,4 -k7 pw.test
root:    awmku76tr43d6:  0:      0:  :   /root/:          /bin/sh
pc:      bdhd74hs9jh3h:  500:    50:  :   /usr1/pc:        /bin/bash
carey:   esJ9ohd89I:     501:    50:  :   /usr1/carey:     /bin/bash
grex:    cj8AjoWE8h8fs:  1500:   60:  :   /usr1/mot:       /bin/sh
mot:     dhjd83kjdJS6D:  1500:   60:  :   /usr1/mot:       /bin/bash
```

最后举一个例子，说明如何以字段的一部分作为排序键。在下面的例子中，用口令字段的前两个字符作为排序键，并按从高到低逆序显示结果：

```
$ sort -t: -r -k2.1,2.3 pw.test
carey:   esJ9ohd891:    501:    50: :   /usr1/carey:    /bin/bash
mot:     dhjd83kjdJS6D: 1500:   60: :   /usr1/mot:     /bin/bash
grex:    cj8AjoWE8h8fs: 1500:   60: :   /usr1/mot:     /bin/sh
pc:      bdhd74hs9jn3h: 500:    50: :   /usr1/pc:      /bin/bash
root:    awmku/6tr43d6: 0:      0:  :   /root/:        /bin/sh
```

7.3　文件的归档、压缩和解压缩

在 UNIX 中，有几个应用程序被广泛使用，实现文件的压缩和归档。

7.3.1　归档

tar 是最常见的用来归档文件的命令，它不完成压缩。**tar** 命令的格式为：

```
tar <options> <file1> <file2> ... <fileN>
```

其中，*options* 是 **tar** 的命令参数选项。从 *file1* 到 *fileN* 是向归档文件中添加或从归档文件中展开的文件列表。下面举例说明：

```
# tar cvf backup.tar /etc
```

该命令将目录 /etc 下的所有文件打包成 **tar** 归档文件 backup.tar。其中，“cvf”是 **tar** 的命令参数：c 告诉 **tar** 创建一个归档文件，v 选项强制 **tar** 执行冗长模式，即在归档时打印每一个文件名，f 选项告诉 **tar** 下一个参数 backup.tar 是创建的归档文件的文件名。**tar** 的其余参数是添加到归档文件中的文件名和目录名。再如：

```
# tar xvf backup.tar
```

上述命令将 **tar** 文件 backup.tar 解开并放到当前目录下。

一般说来，这样做是很危险的。因为在从 **tar** 文件中解开归档文件时，当前目录中原有的同名文件将被覆盖。因此，在解开 **tar** 文件之前，知道自己在何处打开文件非常重要。例如，假设存档以下文件：/ect/hosts、/etc/group 和 /etc/passwd。

如果用下列命令：

```
# tar cvf backup.tar /ect/hosts /etc/group /etc/passwd
```

注意，目录名 /etc/ 被加到每一个文件名之前。

这种情况下，在解开归档文件时，如果希望将文件展开到正确位置，只需使用以下两条命令：

```
# cd /
# tar xvf backup.tar
```

文件能够自动地从归档文件中抽取出路径名，解开到正确的目录。

然而，如果用下面的命令归档文件：

```
# cd /etc
# tar cvf hosts group passwd
```

这样做，目录名没有存在归档文件中。因此在展开文件之前，首先必须用命令进入目录 /etc，然后再展开归档文件：

```
# cd /etc
# tar xvf backup.tar
```

由此可见，**tar** 文件应当在何处展开与它的创建方式有很大关系。使用下列命令：

```
# tar tvf backup.tar
```

在解开 **tar** 文件之前显示 **tar** 文件的索引，可以看到归档文件中与文件名联系的目录名，从而保证在正确位置上展开归档文件。

7.3.2 压缩和解压缩

tar 与 MS-DOS 的归档程序不同，在归档过程中并不压缩文件。因此，如果存储两个 1MB 的文件，最终获得的 **tar** 文件大小为 2MB。而 **gzip** 命令是用来压缩文件的（被压缩的文件并不一定必须是 **tar** 格式）。例如：

```
# gzip -9 backup.tar
```

上述命令用来压缩 backup.tar 文件，产生的文件名为 backup.tar.gz。命令开关 **-9** 告诉 **gzip** 采用最高的压缩比。**gzip** 每次只能压缩一个文件，压缩后的文件必须经解压缩才能使用。

gunzip 命令用来解开一个由 **gzip** 压缩的文件，用 **gzip -d** 也可以达到同样的效果。

gzip 是 UNIX 家族的一个新工具，多年来人们一直用 **compress** 命令来压缩文件，由于 **compress** 算法的软件专利之争以及 **gzip** 比 **compress** 具有更高的压缩效率，使得 **compress** 已经成为过去。**compress** 同样每次只能压缩一个文件。

用 **compress** 获得的压缩文件扩展名为 .Z。例如，backup.tar.Z 是 backup.tar 用 **compress** 压缩后的文件名，而 backup.tar.gz 是用 **gzip** 压缩后的文件名。**uncompress** 和 **gunzip** 命令都能用来展开以 **compress** 压缩的文件。详细内容请参阅联机手册。

在 Linux 下还可以找到其他压缩工具和解压工具，包括 **arj**、**zip** 和 **unzip** 等等。

7.3.3 归档、压缩和解压缩的联合使用

知道了如何归档、压缩文件，现在讨论上述命令的联合使用。要压缩存储一组文件，可用以下命令：

```
# tar cvf backup.tar /etc
# gzip -9 backup.tar
```

它们将目录 /etc 下的所有文件打包成 **tar** 归档文件 backup.tar，然后压缩为文件 backup.tar.gz。展开该文件时，用相反的命令：

```
# gunzip backup.tar.gz
# tar xvf backup.tar
```

当然，在展开 **tar** 文件之前要确保在正确的目录下。

利用管道和重定向命令，可以在一个命令行中完成以上所有工作。如下所示：

```
# tar cvf - /etc | gzip -9c > backup.tar.gz
```

这里，**tar** 文件被送往 **-**，表示 **tar** 的标准输出，通过管道作为 **gzip** 的输入，最后结果存在文件 backup.tar.gz 中。命令 **gzip** 的 -c 选项表示，将输出送到标准输出，重定向到文件 backup.tar.gz。

类似地，也可以用一条命令展开上述归档压缩文件：

```
# gunzip -c backup.tar.gz | tar xvf -
```

在 **tar** 命令的使用中还包括 **z** 选项，自动采用 **gzip** 压缩算法联机压缩/解压文件。例如：

```
# tar cvfz backup.tar.gz /etc
```

上述命令等价于：

```
# tar cvf backup.tar /etc
# gzip backup.tar
```

而命令：

```
# tar xvfz backup.tar.Z
```

可以用来替代下述命令：

```
# uncompress backup.tar.Z
# tar xvf backup.tar
```

详细信息请参阅 **tar** 和 **gzip** 的联机手册。

7.4　其他常用工具

在 Linux 系统之中可以使用的工具还有很多，这里无法一一详细讨论了。如果你对它们感兴趣，可以利用 archie 或 ftp 工具从 Linux 的相关 FTP 站点获取。

这一小节中，为了使读者对于在 Linux 中可以完成哪些任务有进一步的印象，给出一个简单的列表。

表 7-6 罗列了几种常用工具的名称及其基本功能。

表 7-6　常用工具及其功能

程序或命令	说明
at	使你能够在指定日期和时间运行程序。
awk	一种简单而功能强大的语言，可用来操作数据文件。例如，data.dat 是一个包含多个字段的数据文件，那么可以用命令 `$ awk '$2 ~ "abc" {print $1, "\t", $4}' data.dat` 打印数据文件 data.dat 中所有在第 2 字段内包含字符串“abc”的数据行的第 1 字段和第 4 字段。

cron	根据指定的日期和时间，定期完成某些任务。
delete-undelete	删除和恢复文件。
df	给出所有硬盘的有关信息。
doscmu	运行部分 DOS 程序及某些 Windows 3.x 程序，但需作一定调整。
file <filename>	给出文件名（ASCII 文件、可执行文件、归档文件，等等）。
tcx	压缩可执行二进制文件，保持其可执行。
joe	优秀的编辑器，键入 jstar 即启动。使用感觉类似于 WordStar 等编辑器。
less	优秀的文字浏览器，适当配置后可用来浏览经程序 gzip、tar 和 zip 处理过的文件。
mc	一个很好用的文件管理器。
pine	一个不错的 E-mail 程序。
script <script_file>	将屏幕上显示的内容拷贝到 script_file 脚本文件，直到输入命令 exit 为止。在调试程序时很有用。
sudo	允许普通用户做 root 才能做的一些事（例如格式化或挂装磁盘）。
uname -a	给出用户的系统信息

zcat 和 zless	用来阅读被 gzip 压缩的文本文件——无须先将它解压缩。例如：
	$ zless textfile.gz
	$ zcat textfile.gz | lpr

第八章

其 他

通过前几章的学习，读者已经熟悉了 Linux 的一般使用，达到了初步了解和掌握 Linux 的目的。然而，作为一名真正意义上的用户，如果不了解一些有关编程的知识，似乎总是一种缺憾。

在本书第一部分的最后一章中，将对 shell 脚本的编写、GNU C 的安装和使用进行初步的讨论。请读者注意，要真正全面掌握上述内容中的任何一种，都不是一朝一夕能够实现的。它们被专列到“其他”一章，也正是说明了其深度在某种意义上已经超出了 Linux 初级使用的范围。

如果你希望进一步掌握有关 shell 脚本和 GNU C 的知识，请参阅相关专著，或者查看联机帮助，上网检索资料。

8.1 shell 脚本编程入门

在 DOS 中经常会有一些例行的重复工作。这时，用户可以将经常重复的命令写成批处理文件，只要执行这个批处理文件就等于执行这些命令。Linux 系统中，提供类似功能的是 shell 脚本（shell script）。它比 DOS 批处理文件更强大，相对也较复杂。shell 脚本与一般的高级语言十分相似，可使用变量、子程序以及 `while`、`for`、`case`、`if-then-else` 等语法结构，功能相当强大。

8.1.1 概述

一般而言，shell 脚本与其他可执行文件（或命令）完全相同，差别在于，shell 脚本是以 ASCII 文本文件，而非二进位文件的形式储存。执行这一脚本时，必须由 shell 程序解释它的内容并转成一条条实际的命令来执行，这也正是 shell 脚本名称的由来。

写 shell 脚本（简称脚本）与在 DOS 下写批处理文件类似：编辑一个包含所需指令的 ASCII 文件，然后保存。写好脚本之后，必须用 **chmod** 命令将它的访问权限设置为可执行。比如：

```
$ chmod u+x filename
```

用上述命令，则只有文件属主本人可以执行 `filename` 脚本，其他人不能执行。又如：

```
$ chmod ug+x filename
```

此时，文件属主本人及同一用户组可以执行这一脚本，其他人不能执行。又如：

```
$ chmod +x filename
```

这种情况下，所有用户都可以执行 filename 脚本。

让我们举一个实际的例子。

```
$ cat > dirsize
ls /usr/bin| wc -w
<Ctrl+d>
$ chmod 700 dirsize
$ ls -l dirsize
-rwx------  1  pc  bock  20 Jun10 22:04  dirsize
$ dirsize
450
```

在上述命令行中，**cat** 把从键盘输入的管道命令 **ls /usr/bin | wc -w** 送入文本文件 **dirsize**。按组合键 **Ctrl+d** 终止键盘输入，然后，用 **chmod** 命令将文件 dirsize 的访问权限改为可执行，这样，脚本文件 dirsize 就建立好了。

我们可以用 **ls** 命令确认 dirsize 的权限是否已按要求设置好。现在，在 shell 提示符后键入文件名 dirsize，就可以执行前面定义好的管道命令，效果与直接在命令行键入管道命令完全一样。输出结果 450 是 /usr/bin 目录内的文件数。

脚本以行为单位来编写和执行。在上面这个最简单的例子中，脚本文件只有一行。每一行可以是命令、注解，或是流程控制指令等。如果某一行尚未完成，可以在行末加上“\”，这时下一行的内容就会接到这一行的后面，成为同一行，如下所示：

```
echo The message is too long so we have \
to split it into two lines
```

当脚本中出现字符“#”时，后面跟着的同一行文字即为注解，shell 不会对其进行解释。脚本与普通的命令行相同，可以在前台或后台执行；执行命令时需要设定一些环境变量。

8.1.2 shell 的指定

上面举了一个简单 shell 脚本的示例，我们假设它是在 **bash** 中执行的。

通常，在各种 shell 的脚本之间存在一些差异，因此不能将一种 shell 的脚本用另外一种 shell 来执行。那么，如何来为脚本文件指定一种 shell 呢？基本方式如下所述：

- 若脚本的第一个非空字符不是“#”，则使用 Bourne Shell；
- 若脚本的第一个非空字符是“#”，但不是以“#!”开头，则使用 C Shell；
- 若脚本以“#!”开头，则“#!”后面就是所使用的 shell 的完整路径名。

建议读者使用第三种方式明确指定所使用的 shell。

在各种 UNIX 中最常用到 Bourne Shell 以及 C Shell。Bourne Shell 的路径名通常为 /bin/sh，而 C Shell 的路径为 /bin/csh，在本书讨论中，将以与 Bourne Shell 兼容的 **bash** 为例。

除在脚本内部指定 shell 之外，也可以在命令行中强制指定。例如，要求 C Shell 执行 filename 脚本，用下列命令：

```
$ csh filename
```

此时，脚本的访问权限不一定要为可执行文件，脚本内部所指定的 shell 也会无效。

8.1.3 变量

作为一种编程语言，shell 脚本中允许使用字符串变量。然而，如果要进行数值运算，必须靠外部命令实现。字符串变量可分为四种：用户变量、系统变量、只读用户变量和特殊变量。

用户变量

由用户定义的变量，它只是当前 shell 的局部变量，不能被 shell 运行的其他命令或脚本使用。任何不包含空格的字符串都可用做变量名，给变量赋值可用下列方式：

```
var=string
```

其中 *var* 为变量名。如果要提取变量的值，必须在变量名前加上字符“$”，例如：

```
name=Tom
echo name
echo $name
```

其中，第一句给变量 name 赋值 Tom，第二句用 **echo** 命令回显变量 name 的名称，第三句用 **echo** 命令回显变量 name 的值。结果如下：

```
name
Tom
```

系统变量

也称环境变量。它与用户变量的差别在于，可以将其值传给 shell 运行的其他命令或脚本使用。

用 **export** 命令可以把一个用户变量 var 设为系统变量：

```
export var
```

也可以在给变量赋值的同时使用 **export** 命令。例如，下面的命令建立一个系统变量 name：

```
export name=Tom
```

只读用户变量

与用户变量相似，但它的值不能被改变。

使用 **readonly** 命令，可以把用户变量设为只读：

```
readonly var
```

注意，如果只键入 **readonly**，而不键入变量名 *var*，则列出所有只读变量。另外需要注意，系统变量不允许设为只读。

特殊变量

某些变量在一开始执行脚本时就设定且不再改变，它们被称为只读系统变量或特殊变

量。而上面已经说过，用户不能把一般的系统变量设为只读。

用户一旦进入 Linux 系统，系统变量和特殊变量就已经设定好了。表 8-1 和表 8-2 列出了几个常见的系统变量和特殊变量。

表 8-1 系统变量

系统变量	说明
HOME	保存用户起始目录的路径名。
PATH	保存目录路径名列表，给出命令文件的位置。
MAILCHECK	确定每隔多少秒检查是否有新的邮件。
PS1	保存用作 shell 提示符的字符串。
PS2	保存用作 shell 第二提示符的字符串。当 shell 发现输入的命令不完整，需要进一步输入内容时使用这一提示符。
MANPATH	保存 man 命令的搜寻路径

表 8-2 特殊变量

特殊变量	说明
$0	保存 shell 的执行名字。
$n	保存传送给当前 shell 的第 n 个命令参数的值，n 为 1 到 9。
$*	保存传送给当前 shell 的所有命令参数的列表
$#	保存传送给当前 shell 的命令参数的数目。
$$	保存当前 shell 的 PID。
$?	保存 Shell 最后所执行的命令的退出状态。

变量的赋值

给一个变量赋值，除了前面介绍的方法外，还可以通过键盘交互输入，或者用命令替换的方法将管道命令的输出赋值给变量。下面分别介绍。

1．键盘交互输入

shell 可以用 **read** 命令从键盘读入多行字符，并将它赋值给一个或多个变量。例如，下面是一个最简单的应用，将键盘输入的内容赋值给单个变量：

```
echo -n 'Enter some text'
read
echo The text was: $REPLY
```

我们可以把上述几行正文保存到一个名为 readtest1 的 shell 脚本中，然后运行这一脚本：

```
$ readtest1
Enter some text: THIS IS THE TEXT I ENTERED!
The text was: THIS IS THE TEXT I ENTERED!
```

我们注意到，脚本中的 **read** 命令后没有指定变量的名称，此时，从键盘读入的所有字符串赋值给系统变量 REPLY。如果在语句中指定了变量名，则输入的内容直接送给该变量。

上面讨论了 **read** 命令处理单个变量的情况：从键盘输入的（以空格分隔的）所有字均送给这一变量。

当 **read** 命令处理的是多个变量时，情况会稍复杂些：如果从键盘输入的字数比变量的个数多，最后一个变量会将剩下的所有字作为它的值；如果输入的字数比变量的个数少，则后面的变量会设成空字符串。

2．管道命令替换

在 shell 脚本中，允许将管道命令的输出赋值给一个变量。例如：

```
$ date
Wed Jun 14 22:50:52 BST 1995
$ datestore='date'
$ echo $datestore
Wed Jun 14 22:51:11 BST 1995
```

其中，第一条 **date** 命令显示当前时间；第二条命令将 **date** 放在单引号中，使它的标准输出直接赋值给变量 datestore；最后一条命令显示此时变量 datestore 的结果。

除了使用单引号，还可以用下面一种方式给变量赋值，效果是相同的：

```
$ datestore=$(date)
$ echo $datestore
Wed Jun 14 22:53:21 BST 1995
```

此外，利用 **basename** 命令，可以将完整的文件路径名作为参数给变量赋值，而变量得到的是去掉路径后的基本文件名。例如：

```
$ basefile='basename /usr/bin/man'
$ echo $basefile
man
```

8.1.4 登录脚本

前面已经介绍过，用户通过修改系统变量 PS1 的值可以改变 shell 提示符，在 PATH 中添加定制内容可以更方便地执行需要的程序。然而，如果每次登录后都必须完成这些操作，本身又是一件麻烦的事情。

下面介绍一种办法，建立个人的登录脚本 .bash_login，并将它存放在起始目录下，由 shell 自动读取并执行这一脚本。用户登录时，shell 会自动搜索起始目录，依次寻找下面 3 个特殊文件，并执行最先找到的一个。这三个文件是：

```
~/.bash_profile
~/.bash_login
~/.profile
```

这里，~ 表示用户的起始目录。同样地，当用户退出登录时，shell 也能执行个人定制的退出登录脚本 .bash_logout。通常，也应当将它存放在起始目录：

```
~/.bash_logout
```

除了执行个人脚本之外，**bash** 还允许系统管理员建立全局登录脚本，从系统整体考虑，设置用户的局部环境。全局登录脚本的名称为：

```
/etc/profile
```

任何用户登录时，**bash** 都执行这一脚本。全局登录脚本在个人定制脚本之前执行。这样，个人脚本能够进一步设置局部环境，使它适合用户的特殊要求。

8.1.5 捕捉信号

我们知道，按组合键 Ctrl+C 可以中断程序的运行。这是通过控制键盘的 Linux 内核向运行中的进程发送信号（signal）来实现的。

信号由特定的硬件和软件条件引起。信号共有约 30 种。利用加上开关的 **kill** 命令，可以给进程发送信号。格式为：

```
kill [-sig] pid
```

其中，*sig* 是表示具体信号的整数，如果 *sig* 值缺省，**kill** 命令发送默认的结束信号（terminate）。*pid* 是接受信号的进程识别号。在前面 5.3 节讨论作业管理和虚拟终端时，我们曾经用 **kill -9** 命令给进程发送了一个不允许忽视的终止信号。

常用信号及其对应的整数如表 8-3 所示。

表 8-3 常用的信号及其作用

信号	数值	说明
hangup	1	退出登录时，结束用户进程的信号。
interrupt	2	从键盘产生的中断信号（Ctrl+c）。
quit	3	从键盘产生的退出信息（Ctrl+\）。
kill	9	不允许忽视的强制结束进程的信号。
alarm	14	系统调用 **alarm()** 结束时产生的警告信号。
terminate	15	**kill** 命令的默认结束信号。

在 shell 脚本内部可以用 **trap** 命令捕捉信号。**trap** 命令有三种基本形式，分别对应三种不同的信号响应方式。

第一种形式：

```
trap "commands" signal-list
```

当脚本收到 *signal-list* 清单内列出的信号时，**trap** 命令执行双引号中的命令。

第二种形式，**trap** 不指定任何命令，接受信号的默认操作：

```
trap signal-list
```

对于大多数信号来说，默认操作是结束进程的运行。

第三种形式，**trap** 命令指定一个空命令串，允许忽视信号：

```
trap "" signal-list
```

下面是使用 **trap** 命令的一个典型例子，当程序非正常结束时删除临时文件，避免这些文件占用空间。

```
trap "rm -f /tmp/tmp$$;exit 0" 2 3
touch /tmp/tmp$$
#
# Rest of Shell script
#
```

```
trap 2 3
```

上述程序建立了一个临时文件 /tmp/tmp$$ 供脚本内部使用。如果收到了非正常结束脚本的信号（中断或退出），则首先删除这一临时文件，然后结束脚本的运行。第一句中，脚本准备捕捉信号 2 和 3 （从键盘发出的 Ctrl+C 和 Ctrl+\ 信号）。如果捕捉到信号 2 和 3 其中之一，将导致下列命令的执行：

```
rm -f /tmp/tmp$$;exit 0
```

其中，**rm** 命令删除临时文件 /tmp/tmp$$，而 **exit** 命令结束这一脚本的运行。注意，这里是用分号隔开的两条命令。

在正常情况下，这一脚本将用 **touch** 命令建立临时文件，接着执行脚本中的其余命令（此处用注释行“Rest of Shell script”代替）。最后一行用 **trap** 命令恢复信号 2 和 3 的默认操作。

8.1.6 控制程序流程

test 命令

shell 脚本相当于一种高级语言，允许对各种条件进行判断，并使用复杂的流程控制结构。在讲述流程控制之前，首先介绍 **test** 命令。**test** 的用途是测试一系列条件，然后返回相应的状态值。在流程控制中，必须用 **test** 命令来判断真假。它的格式为：

```
test expression
```

条件表达式 *expression* 是 **test** 命令的参数。当 *expression* 的值为真时，返回状态值零；当 *expression* 的值为假时，返回非零的状态值。

test 可以组合多个表达式，各表达式之间用逻辑运算符隔开。表 8-4 列出了常用的逻辑运算符。

表 8-4 常用的逻辑运算符

逻辑运算	说明
exp1 -a *exp2*	AND 运算。如果表达式 *exp1* 和 *exp2* 都为真，返回真值。
exp1 -o *exp2*	OR 运算。如果表达式 *exp1* 和 *exp2* 之一为真，返回真值。
! *exp*	NOT 运算。如果表达式 *exp* 为真，返回假值；否则返回真值。

test 命令可以用来检验几种不同类型的表达式，常见的有三种。

1. 检验文件的特征。此时，给出一个命令开关，后面跟以文件名。**test** 命令根据开关指定的特征对文件进行检验，根据检验结果返回一个状态值。表 8-5 列出了常用的命令开关。

表 8-5 用于检验文件特征的 test 命令开关

命令开关	说明
-e *file*	如果 *file* 文件存在，返回真值。
-f *file*	如果 *file* 文件是普通文件，返回真值。

-d file	如果 file 文件是目录，返回真值。
-r file	如果用户能读 file 文件，返回真值。
-w file	如果用户能写 file 文件，返回真值。
-x file	如果用户能执行 file 文件，返回真值。

2．比较字符串。在字符串表达式中，有一个或两个参数，它们可以是普通的字符串，也可以是 shell 变量的值。表 8-6 列出了几个常用的字符串表达式。

表 8-6 常用的字符串表达式

字符串表达式	说明
-z str	如果字符串 str 的长度为 0，返回真值，
-n str	如果字符串 str 的长度不为 0，返回真值。
str1=str2	如果字符串 str1 和 str2 相同，返回真值。
str1!=str2	如果字符串 str1 和 str2 不同，返回真值。

3．比较数值。利用数值表达式，可以将字符串或变量的内容作为数值来处理。常见的数值表达式见表 8-7。

表 8-7 常用的数值表达式

数值表达式	说明
num1 -eq num2	如果 num1 等于 num2，返回真值
num1 -ne num2	如果 num1 等于 num2，返回真值
num1 -lt num2	如果 num1 小于 num2，返回真值
num1 -gt num2	如果 num1 大于 num2，返回真值
num1 -le num2	如果 num1 小于或等于 num2，返回真值
num1 -ge num2	如果 num1 大于或等于 num2，返回真值

这里顺便提一下，**test** 命令能够将字符串作为数值比较，与此类似，**bash** 也可以进行简单的算术运算。算术表达式的形式如下所示：

```
$[expression]
```

请看具体运用：

```
$ num1=2
$ num1=$[num1*3+1]
4 echo $ num1
7
```

接下来，让我们介绍几种常见的流程控制结构作为 shell 脚本一节的结束。由于这些结构与常见计算机语言中没有太大差别，这里就不给出例子了。

请注意，在下列语法结构中，condition 都是 **test** 命令。

if-then 结构

语法：	说明：
if (condition)	执行条件命令 condition，如果它的返回值为真，则执行 the

then *then-commands* fi	*n-commands* 命令，否则直接跳出，执行 fi 后面的命令。

if-then-else 结构

语法：	说明：

if (*condition*)
then
  then-commands
else
  else-commands
fi
	执行条件命令 *condition*，如果它的返回值为真，则执行 *then-commands* 命令，否则执行 *else-commands*。
	执行完 *then-commands* 或 *else-commands* 后跳出，执行 fi 后面的命令。

if-then-elif 结构

语法：	说明：

if (*condition1*)
then
  commands1
elif (*condition2*)
then
  commands2
else
  commands3
fi
	首先执行条件命令 *condition1*，如果它的返回值为真，执行 *then-commands* 命令；如果 *condition1* 为假，则执行条件命令 *condition2*。
	此时，如果 *condition2* 为真，执行 *commands2*，否则执行 *commands3*。
	执行完 *commands2* 或 *commands3* 后跳出，执行 fi 后面的命令。

for-in 结构

语法：	说明：
for *var* in *arg-list* do *commands* done	从 *arg-list* 清单中依次取指定变量 *var* 的每一个值，然后用取得的值执行循环体内的 *commands* 命令。

while 结构

语法：	说明：
while (*condition*) do *commands* done	执行条件命令 *condition*，如果它的返回值为真，执行 do 和 done 之间的 *commands* 命令，接着转移到循环的顶部，重新检查条件；如果 *condition* 条件为假，则跳出循环，执行done后面的命令。

until 结构

语法：	说明：
until (*condition*) do *commands* done	它和 while 的不同只在于：while 在条件为真时，继续执行循环；而 until 在条件为假时，继续执行循环。

break 及 continue 结构

说明：

用于 for，while，until 等循环控制结构。
break 会跳至 done 后方执行，而 continue 会跳至 done，继续执行循环。

case 结构

语法：

```
case str in
pat1)
    commands1
    ;;
pat2)
    commands2
    ;;
pat3)
    commands3
    ;;
*)
    default-command
    ;;
esac
```

说明：

首先计算字符串 *str* 的值，然后将结果依次和表达式 *pat1*、*pat2*、*pat3* 等相比较，直到找到一个匹配的表达式位置。如果找到了匹配项，则执行相应的命令 *command1*、*command2* 或 *command3*，直到遇到一对分号为止；如果找不到匹配项，则执行默认的命令 *default-command*。

注意，*pat* 除了可以使用确定的字符串之外，也可以用通配符来指定。例如：

*	匹配任意一个字符串
?	匹配任意一个字符
[abc]	匹配 a，b 或 c 字符中之一
[a-n]	匹配从 a 到 n 的任一字符
|	多重选择

8.2 GNU C 的安装和使用

本小节将对如何在 Linux 系统上编写、编译、连接、运行和调试 C 程序作一概括的介绍。

8.2.1 安装 gcc

各种 Linux 套件通常都包含的 GNU C 编译器名为 **gcc**，它支持 C、C++、Objective C 等语言。**gcc** 的压缩文档可以从存放 GNU 软件的 FTP 站点获取，然后在指定目录下用 **tar** 命令解开。例如：

```
$ tar zxvf gcc-2.7.2.3.tar.gz
```

上述命令产生目录 gcc-2.7.2.3/。在这一目录内执行下列命令，编译和安装 gcc：

```
$ ./configure
$ make bootstrap
$ make compare
# make -k install
```

如果要将 **gcc** 安装到系统目录，上面的最后一步应用 root 帐号完成。标准情况下，**gcc** 安装在 /usr/local 的子目录下，例如 /usr/local/bin、/usr/local/lib、/usr/local/include 等。如果用户希望将 **gcc** 安装到其他目录，则应当在 **configure** 一步指定 --prefix。例如：

```
$ ./configure --prefix=usr
```

上述命令表明，将 **gcc** 安装到 /usr 的 /usr/bin、/usr/lib/、/usr/include 等子目录中。

安装完毕后，可用下列命令检验 **gcc** 已经正确安装：

```
$ gcc -v
```

现在用户就可以用 **gcc** 来编译 C 程序了。

如果用户希望用 c++ (g++) 来编译 C++ 程序，还必须编译安装 GUN 的 C++ 类库。首先，从 FTP 站点获取 libg++-2.7.2.tar.gz 文档，然后编译和安装 GNU 的 C++类库：

```
$ tar zxvf libg++-2.7.2.tar.gz
$ cd libg++-2.7.2
$ ./configure
$ make all
# make install
```

注意：如果前面编译 **gcc** 时用到了 --prefix 参数，那么现在编译 libg++ 时也要使用相同的 --prefix 参数；如果你希望将类库安装到系统目录，最后一步也应当用 root 帐号完成；如果你希望将 libg++ 和 libstdc++ 编译成共享库，应当在 **configure** 命令中使用如下参数，然后再编译安装：

```
$./configure -enable_shared
```

8.2.2 C 程序的编译和连接

用 **gcc** 编译 C 程序并生成可执行文件实际要经历如下四步：

- 预处理。调用 **cpp** 程序，对各种命令如 #define、#include、#if 进行分析。
- 编译。调用 **cc1** 程序，根据输入文件产生汇编语言。
- 汇编。调用 **as** 程序，用汇编语言作为输入产生 .o 扩展名的目标文件。
- 连接。调用连接程序 **ld**，将各目标文件放于可执行文件的适当位置。这一程序引用的函数也放在可执行文件中。

gcc 的基本用法是：

```
$ gcc -o prog main.c subr1.c subr2.c subr3.c
```

其中，-o 开关用来指定生成的可执行文件名为 prog。如果不指定 -o 参数，**gcc** 则使用默认的文件名 **a.out**。

如果用户希望单独编译每一个源文件，最后再进行连接，可以如下进行：

```
$ gcc -c main.c
$ gcc -c subr1.c
$ gcc -c subr2.c
$ gcc -c subr3.c
$ gcc -o prog main.o subr2.o subr3.o
```

其中，-c 开关表示编译生成目标文件，但不连接。最后一个 **gcc** 命令连接所有目标文件并构成可执行文件。此时，由于所有输入都已经是目标文件，不再需要编译和汇编，所以 **gcc** 只调用连接程序。

表 8-8 C 编程中涉及的文件类型

文件的扩展名	含义	gcc 所作的处理
.c	C 源文件	由 gcc 预处理和编译
.C .cc .cxx	C++ 源文件	由 gcc 预处理和编译
.m	Objective C 源文件	由 gcc 预处理，编译和汇编
.i	预处理后的 C 源文件	由 gcc 编译
.ii	预处理后的 C++ 源文件	由 gcc 编译
.s	汇编语言源文件	由 as 汇编
.S	汇编语言源文件	由 as 预处理和汇编
.o	编译后的目标文件	传送给 ld
.a	目标文件库	传送给 ld

在命令行中，gcc 是根据文件扩展名的不同来分别做相应处理的。表 8-8 列出了各种文件扩展名的含义以及所作的处理，表 8-9 列出了常用的 gcc 命令行开关。

表 8-9 gcc 编译选项开关

编译选项	说明
-c	对源程序进行预处理、编译和汇编，不进行连接。每个源程序产生一个目标文件。
-o file	定义输出的可执行文件名为 file。如果不使用 -o 选项，可执行文件的默认名称为 a.out。目标文件和汇编文件的输出对 source.suffix 分别是 source.o 和 source.s，预处理的 C 源程序的输出是标准输出 stdout。
-Dmacro, -Dmacro=defn	类似于源程序中的 # define。例如： $ gcc -c -DHAVE_GDBM -DHELP_FILE=\"help \"cdict.c 其中，第一个 -D 选项定义宏 HAVE_GDBM，在程序中可以用 #ifdef 去检查它是否被置；第二个 -D 选项将宏 HELP-DILE 定义为字符串 "help"，这对于控制程序打开哪个文件是很有用的。注意：由于使用了反斜杠 "\"，因此引号也成为该宏定义的一部分。
-Umacro	某些宏是被编译程序自动定义的。利用这些宏，可以指定在其中进行编译的计算机系统类型的符号，用户可以在编译某程序时加上 -V 选项以查看 gcc 默认定义了哪些宏。如果用户想取消其中某个宏定义，用 -Umacro 选项，这相当于把 #undef macro 放在要编译的源文件的开头。
-Idir	将 dir 目录加到搜寻头文件的目录列表中，并优先于 gcc 默认的搜索目录。在有多个 -I 选项的情况下，按命令行中 -I 选项的前后顺序搜索。dir 可使用相对路径，如 -I../inc 等。
-O	对程序编译进行优化，编译程序试图减少被编译程序的长度和执行时间，但其编译速度比不作优化慢，而且要求较多的内存。
-O2	允许比 -O 更好的优化，编译速度较慢，但结果程序的执行较快。
-g	产生一张用于调试和排错的扩展符号表。-g 选项使程序可以用 GNU 的调试程序 gdb 进行调试。优化和调试通常是不兼容的，同时使用 -g 和 -O（或 -O2）选项经常会使程序产生奇怪的运行结果，所以不要同时使用 -g 和 -O（或 -O2）选项。
-fpic、-fPIC	产生位置无关的目标代码，可用于构造共享函数库。

表 8-9 列出了 gcc 的编译选项。在 gcc 的命令行中，还可以使用连接选项（见表 8-10）。事实上，gcc 将所有不能够识别的选项传递给连接程序 ld。连接程序 ld 将几个目标文件和程序组合成一个可执行文件，它要解决对外部变量、外部过程、程序等的引

用。然而，我们永远用不着显式地调用 **ld**。利用 **gcc** 命令去连接各个文件很简单，即使在命令行里没有列出库程序，**gcc** 也能保证某些库程序以正确的次序出现。

表 8-10 gcc 连接选项开关

连接选项	说明
-Ldir	将 dir 目录加到搜寻 -L 选项指定的函数的目录列表中，并优先于 gcc 默认的搜索目录。在有多个 -L 选项的情况下，按命令行中 -L 选项的前后顺序搜索。dir 可使用相对路径。如 -L../lib 等。
-lname	在连接时使用函数库 libname.a，连接程序在 -Ldir 选项指定的目录下和 /lib、/usr/lib 目录下寻找这一库文件。在没有使用 -static 选项时，如果发现共享函数库 libname.so，则使用 libname.so 进行动态连接。
-static	禁止与共享函数库连接。
-shared	尽量与共享函数库连接。这是在 Linux 中连接程序的默认选项。

下面是一个使用 **gcc** 进行连接的例子：

```
$ gcc-o prog main.o subr.o -L../lib-lany -lm
```

8.2.3 创建函数库

有时用户需要建立自己的函数库。创建和更新函数库的命令是 **ar**。例如：

```
$ar rs lib-name list-of-files
```

上述命令创建一个新库。其中 r 选项表示把清单 *list-of-files* 中的目标文件添加到函数库 *lib-name*，如果 *lib-name* 库不存在则新建一个；s 选项表示为该库生成一个索引。又如：

```
$ ar rus lib-name list-of-files
```

上述命令用来更新一个库。命令将 *list-of-files* 所列文件的日期与库中原有版本进行比较，如果 *list-of-files* 中的文件比库中的版本更新，**ar** 就用新版本替代老版本；s 选项用来更新函数库的索引。

从函数库中删除一个或多个文件，可使用如下命令：

```
$ ar ds lib-name list-of-files
```

该命令删除由 *list-of-files* 列出的并在 *lib-name* 库中的所有文件。

从一个函数库中取一个或多个文件，可用如下命令：

```
$ ar x lib-name list-of-files
```

该命令并不修改函数库文件本身，而是从库中提取 *list-of-files* 列出的所有文件。通常，被提取文件的时间被标记成当前时间。但如果用 xo 选项取代 x 选项，则被提取文件的时间被标记成它们进入档案的时间。

8.2.4 利用 make 和 Makefile 自动编译

利用 **make** 程序和 Makefile 文件可以实现程序的自动编译。**make** 程序自动确

定一个软件包中需要重新编译的部分，并用特定命令去实现。准确使用 **make** 可以大大减少程序的编译时间，避免不必要的再编译。

使用 **make** 之前必须编写一个名为 makefile 的文件，描述软件包中各文件之间的关系，提供更新每个文件的命令，一般情况下，通过连接目标文件来更新可执行文件，通过编译源文件来更新目标文件。建立了合适的 makefile 文件之后，如果改动了某些源文件，只需使用简单的 shell 命令 **make**，就可以完成重新编译的工作。**make** 程序根据 makefile 的数据和每个文件的更改时间来确定哪些文件需要更新，并用 makefile 中定义的命令来更新文件。

如果在 **make** 程序中没有用 -f 选项指定一个 Makefile 文件，make 将在当前目录下按顺序寻找具有下列名称的文件：GNUmakefile、makefile、Makefile。推荐读者使用文件名 Makefile。因为它的第一个字母是大写，通常被列在一个目录的文件列表的前面。

使用 **make** 程序之前首先必须创建 Makefile 文件。

在 Makefile 中包含一些目标以及实现这些目标的一组组命令。所谓目标，也就是 **make** 程序所要完成的任务，目标通常是文件名，每个目标的完成依赖于其他一些目标或文件。例如，下面给出了一个简单的 Makefile 的例子：

```
# Here is a simple sample of Makefile
prog: prog.o subr.o
        gcc -o prog prog.o subr.o
prog.o: prog.cprog.h
        gcc -c -I. -o prog.o prog.c
subr.o: subr.c
        gcc -c -o subr.o subr.c
clean:
        rm -f prog *.o
```

上述 Makefile 文件定义了四个目标：prog、prog.o、subr.o 和 clean。目标从每行最左边开始书写，后面跟一个冒号，与目标相关的其他目标或文件列在冒号后面，并以空格隔开。然后另起一行书写 shell 命令，用来实现该目标。注意，每条 shell 命令行的第一个字符必须是 Tab 键，不能用一个或一串空格开头，否则 **make** 就会显示一条错误信息，指出 Makefile 中写错的那一行，然后退出，如：

```
Maketile:2:* * *missing separator. Stop.
```

一般情况下，键入 **make *target*** 即可调用 **make** 命令。其中，***target*** 是 Makefile 文件中定义的目标之一；如果缺省 ***target***，**make** 就生成 Makefile 文件中定义的第一个目标。对于上面 Makefile 的例子，由于 prog 是 Makefile 中定义的第一个目标，因此命令“**make**”与“**make prog**”是等价的。

make 为它执行的每一条命令行生成一个新的 shell。其结果是，被该 shell 执行的命令只在一个命令行内部有效，特别是 **cd** 命令，只能影响它所在的命令行。例如，在下列命令行中：

```
cd ../lib
gcc -c -o subr.o subr.c
```

第一行的 **cd** 命令对第二行是无效的。要想在编译 subr.c 之前进入 ../lib 目

录，可使用如下命令：

```
cd ../lib; gcc -c -o subr.o subr.c
```

在 Makefile 中，可使用续行号“\”将一个命令行延续成几行。但要注意，在续行号后面不能跟任何字符（包括空格和 Tab 键）。

make 在检查一个目标是否已经过时并需要更新时，采用的是按相关性递归的方法。**make** 在构建一个目标之前要生成该目标依赖的所有文件，并递归地前进，从而确保这些文件是最新的。**make** 采取如下步骤去生成一个目标：

1. 如果一个目标 task 不是作为一个文件而存在，它就是过时了。命令 **make task** 总是执行该任务。

2. **make** 检查所有与 task 目标相关的目标。对于不是 Makefile 中定义的任务，而只是文件的相关目标，则检查文件是否比 task 文件更新，文件中有一个更新则 task 就过时了。对于 Makefile 中定义任务的相关目标，则按同样的方法递归检查其是否过时，如果其中有任何一个过时了，则 task 也就过时了。

3. 从递归的底层向上，更新所有已过时的目标；只有当一个目标依赖的所有目标都已是最新的，才可以更新这个目标。

我们通过上面 Makefile 的例子来看一看目标更新的过程。现在假设我们修改了文件 subr.c。我们用如下命令更新目标 prog，即重新编译可执行文件 prog：

```
$make prog
```

由于目标 prog 依赖于目标 prog.o 和 subr.o，**make** 必须检查目标 prog.o 和 subr.o 是否过时。目标 prog.o 依赖于文件 prog.c 和 prog.h，**make** 检查目标文件 prog.o 和源文件 prog.c 和 prog.h 的日期，发现 prog.o 比它所依赖的源文件要新，即并不过时。再检查目标 subr.o，它依赖于文件 subr.c。由于我们编辑了 subr.c，它的日期比目标文件 subr.o 的日期要新，即 subr.o 过时了，从而依赖于 subr.o 的所有目标都过时了。**make** 用定义目标 subr.o 的 shell 命令来更新 subr.o：

```
gcc -c -o subr.o subr.c
```

由于目标 subr.o 过时并更新，导致目标 prog 已经过时，**make** 用定义目标 prog 的一组 shell 命令来更新它：

```
gcc -o prog prog.o subr.o
```

从而完成了“**make prog**”的任务。

如果我们是第一次编译上面这个软件，则因为 prog、prog.o、subr.o 等目标文件都不存在，按照约定，所有目标 prog、prog.o、subr.o 都是过时的，都必须更新，即必须从底向上执行定义这些目标的所有命令。

在上面 Makefile 的例子中，还定义了一个目标 clean。输入 **make clean** 命令，将执行 **rm-f prog *.o**。clean 目标是 Makefile 中常用的一种专用目标，即删除所有的目标模块。输入 **make clean** 命令时，make 就查看一个名为 clean 的文件，如果该文件不存在（我们约定，永远不在软件目录中使用具有这一名字的文件），**make** 就执行定义该目标的所有命令。

另一个经常用到的专用目标是 install。通常，它将编译完成的可执行文件和程序运行所需要的其他文件拷到指定的安装目录，并设置相应的保护。

为了简化命令的书写，在 Makefile 中可以定义一些宏（macro）和使用几个预定义的缩写。下面是几个很有用的缩写：

$@	代表该目标的全名
$*	代表已经删除了后缀的目标名
$<	代表该目标的第一个相关目标名

按照这缩写，前面 **Makefile** 的例子可改写成：

```
# Here is a simple sample of Makefile
prog:   prog.o subr.o
        gcc -o $@ prog.o subr.c
prog.o: prog.c prog.h
        gcc -c -o $@ @<
subr.o:subr.c
        gcc -c -o $@ $*.c
clean:
        rm -f prog *.o
```

这类缩写在编写默认的编译规则时很有用。

一个宏定义从一行的最左边开始书写，具有如下格式：

```
macro-name=macro-body
```

当 **make** 在处理这一 Makefile 时，就用 *macro-body* 替代 $(*macro-name*) 串。上面的 Makefile 的例子可用定义宏的方法使之更清晰：

```
# Here is a simple sample of Makefile
DEPENDS = prog.o subr.o
prog: $(DEPENDS)
        gcc -o$@ $(DEPENDS)
prog.o: prog.c prog.h
        gcc -c -I.-o $@ $*.c
subr.o: subr.c
        gcc -c -o $@ $*.c
clean:
        rm -f prog *.o
```

在定义一组编译选项时，宏定义也是很有用的。例如：

```
CFLAGS = -DDEBUG -g
```

上述宏定义中包含了两个编译选项。我们可以在 Makefile 中用下列命令编译用于调试的目标文件：

```
gcc -c $(CFLAGS) -o list.c
```

在 Makefile 文件中，如果没有给出从相关目标构造某一目标的命令，**make** 程序将适用隐含规则。

make 程序预定义了一些隐含规则，每个隐含规则适用于一个目标类型和它的相关类型的组合。对应同一目标类型的相关类型可能不止一个，例如，有多个规则可产生目标类型为 .o 的文件，如从 .c 文件用 C 编译器编译得到，或从 .f 文件用 Fortran 编译器得到。那么，如何确定究竟应该适用哪条隐含规则呢？

隐含规则是通过后缀（suffix）规定来实现的。带有哪些后缀的目标适用隐含规则是由 **make** 程序的内置目标 .SUFFIXES 相关目标列表定义的。默认的后缀列表是：.out、.a、.In、.o、.c、.cc、.C、.p、.f、.F、.r、.y、.l、.s、.S、.mod、.sym、.def、.h、.info、.dvi、.tex、.texinfo、.texi、.txinfo、.w、.ch、.web、.sh、.elc、.el。在所有隐含规则中，允许的目标或相关类型必须是上列类型中的一种。在这个列表中，位置靠前的类型具有更高的优先级，即如果有数条隐含规则的目标类型一样，**make** 程序按上面列表的顺序选用第一个其相关类型存在或可以构造的隐含规则。

例如，对下面的 Makefile：

```
# Here is a simple sample of Makefile
prog:  prog.o subr.o
       gcc-o prog prog.o subr.o
clean:
       rm-f prog * .o
```

prog 的相关目标 prog.o（和 subr.o）的构造规则没有定义，**make** 程序适用隐含规则。在默认的隐含规则中，可以生成目标类型为 .o 文件的相关类型有很多种，例如 .c、.cc、.C、.p、.f 等等。**make** 程序将按顺序找出第一个存在的或可以构造的相关类型，如果它代到了 prog.c 文件，它就适用从 .c 文件生成 .o 文件的隐含规则。如果 prog.c、prog.cc、prog.C 和 prog.p 文件都没有找到或不可构造，但找到了 prog.f，它就适用从 .f 文件生成 .o 文件的隐含规则，依此类推。

make 程序的一条隐含规则可用如下语句表示：

```
.c.o:
     $(CC)-c $(CPPFLAGS) $(CFLAGS) $<
```

这条隐含规则的目标类型是 .o，而相关类型是 .c。它确定了如何从 .c 文件生成 .o 文件。其中用到的 CC、CPPFLAGS、CFLAGS 在 **make** 程序里都有默认的定义。关于 **make** 程序定义的所有隐含规则的信息，请参考 GNU make 软件包附带的文档。

如果我们不想使用 **make** 程序定义的默认规则，可以定义自己的后缀规则，如：

```
.c.o:
      gcc -c -g -DDEBUG -DHELP_FILE=\"help" -o $* .o$<
```

我们也可以用 .SUFFICES：清除默认后缀列表，或者往后缀列表列中增加新的相关类型。例如：

```
.SUFFIXES:
.SUFFIXES: .c .o
.c.o:
      gcc -c -g -DDEBUG -DHELP_FILE=\"help" -o $* .o$<
```

SUFFIXES：首先清除后缀列表，所有的后缀规则（包括默认规则）都失效；而 .SUFFIXES：.c .o 将 .c、.o 添加到后缀列表里，以后所有的后缀规则可涉及 .c 和 .o 文件。

后缀规则适用于所有 UNIX 版本的 **make** 程序，GNU 的 **make** 程序提供了一种更方便的规则定义方式，称为模式（pattern）规则。模式规则的格式类似 Makefile 中一般的目标定义，但它使用“%”作为通配符，如

```
%.o: %.c
    gcc $(CFLAGS) -c $<
```

表示从 .c 文件生成 .o 文件的规则。

模式规则不依赖于后缀列表。它与后缀规则的另一个不同之处可用下面的例子来说明。

```
.c.o:
    $(CC) -c $(CFLAGS) $(CPPFLAGS) -o $@ $<
```

后缀规则说明了从 .c 文件产生 .o 文件的规则，但后缀规则不能有自己的相关目标。如下面的规则要求从相关文件 foo.h 编译产生文件 .c.o：

```
.c.o: foo.h
    $(CC) -c $(CFLAGS) $(CPPFLAGS) -o $@ $<
```

而模式规则却可以有自己的相关目标：

```
$.o: %.c foo.h
     $(CC)-c $(CFLAGS) $(CPPF;AGS) -o $@ $<
```

它定义了从 .c 文件生成 .o 文件的规则，而且使所有用这条规则生成的 .o 文件都与 foo.h 相关。

8.3 其他

我们以两张表格作为本章的结束。第一张表列出 Linux 环境下常见的文件扩展名，其中多数前面已经见过了，还有一些对你则可能是陌生的。另外一张表给出了一些 Linux 中文软件的名称。你在今后使用 Linux 的过程中，会对它们逐渐熟悉起来。

8.3.1 常见文件的扩展名

Linux 中有各种不同格式的程序、文档，它们的扩展名各不相同。下面给出是常见文件及其扩展名的列表。

表 8-11 常见的文件扩展名

扩展名	说明
1 ... 8:	联机手册页，用 **man** 读取。
arj:	**arj** 产生的压缩文件，用 **unarj** 解开。
dvi:	**TeX**（见后述）产生的输出文件。可用 **xdvi** 阅读；用 **dvips** 可将 dvi 文件转换成扩展名为 ps 的 PostScript 文件。
gif:	图形文件，用于 **seejpeg** 或 **xpaint**。
gz:	**gzip** 产生的压缩文件。
info:	**info** 文件（类似联机手册的一种文件），用于 **info**。
jpg 和 jpeg	图形文件，用于 **seejpeg**。
lsm	Linux Software Map 文件。一种用来说明软件包的 ASCII 无格式文件。
ps	PostScript 文件。用 **gs** 或 **ghostview** 来读取。
rpm	Red Hat Package Manager。可以安装在任何 Linux 系统上。
tgz、tar.gz	用 **tar** 归档，再用 **gzip** 压缩的文档。

tex	传给 **TeX** 处理的文本文件。**TeX** 是一个强大的排版程序。许多 Linux 套件中都附有 **TeX**。
texi	**texinfo** 文件，可生成 TeX 和 info 文件。用于 **texinfo**。
xbm、xpm、xwd	图形文件。用于 **xpaint**。
zip	用 **zip** 压缩归档的文档，用于 **zip** 和 **unzip**。
Z	用 **compress** 归档的文档。

8.3.2 一些有用的中文软件

直到目前为止，我们讨论的 Linux 都是英文界面的。为了能使 Linux 系统可以处理中文资料，必须安装中文系统。

有关中文环境的建立，将在本书第三部分详细讨论。本章以表格的形式介绍几种 Linux 中文软件，使读者对于 Linux 环境的中文使用条件有一个初步印象。

表 8-12 一些有用的中文软件

名称	说明
cjoe	Joe's Own Chinese Editor。joe 用于 UNIX 平台，是一个免费的专业 ASCII 文字编辑器，与 IBM PC 上的很多文字编辑器类似。cjoe 是中文化的 joe。
celvis	与 UNIX 平台的标准编辑器 vi/ex 十分类似，几乎支持所有 vi/ex 指令。用 celvis 可编辑同时含有中英文的文章。celvis 同时支持 GB2312-80 和 BIG5 码。
cvim	vim-4.2 的中文修补程序。具备与 vi 相似，而 celvis-1.3 没有的一些特性，如行号、回绕行，以及大文件的编辑。
he	DOS 上著名编辑器的 Linux 版。但它是一个共享程序，所处理的文件限制在一百行。
hztty	用于不同中文编码格式之间的转换。
ktty	一个与 hztty 很相似的工具，用来在 kterm 或 pxvt 上阅读中文。
Cemacs 和 CChelp for Emacs	Cemacs 是一种用 GNU Emacs 来显示并编辑中文文件的方法，必须在中文虚拟终端上运行 Emacs，CChelp 是一套提供中文辅助消息的系统。安装 CChelp 之后，用鼠标指向任意中文字并按下，就会显示这个词的相关信息，包括读音、英文解释等。它同时支持 GB 与 BIG5 码。
Mule	即 MULtilingual Enhancement to GNU Emacs。它对 GNU Emacs 作了补充，使之能够处理多种语言的编码系统。由于 Mule 将多字节编码系统在内部重新编码，因此可在一篇文章内同时使用中文（BIG5 与 GB 码）、日语、韩语、英语、泰语等。
hc	用于 BIG5 及 GB 码转换的程序。
ctin	一个用于中文消息的新闻讨论组阅读器。

第二部分

Linux 奥秘

Linux 以及许多在 GNU 通用公共许可证条款的保护下发行的软件，经常被人们误认为是“免费”软件，但是，GNU GPL 条款中所谓“free”的原意实际是“自由”，也就是说，用户享有自由发行软件，修改源代码并重新发行的权利，但要求用户同样遵循这 GPL 条款，并应赋予其他用户以相同的权利。

Linux 的迅速成长与用户享有修改并发行程序新版本的“自由”权利是分不开的。现在，中国用户也可以自由获取 Linux 以及许多应用程序或程序库的源代码，但如何才能最大程度地利用 Linux 呢？显然，冲击世界 Linux 的热浪，并不仅仅因为其低廉的成本和高的性能，而主要在于其真正开放的结构和所遵循的标准，以及 Linux 爱好者和开发者所奉行的信念。由于许多原因，使 Linux 在国内的推广比国外晚了好几年，所幸的是，去年以来有更多的软件爱好者开始了 Linux 的学习。但是，仅仅了解和掌握 Linux 的使用还远远不够，要真正利用 Linux 并发挥其最大潜力，需要深刻理解 Linux 系统及其内核结构。

这一部分通过对 Linux 内核的介绍，论述有关内存管理、进程、进程间通讯等方面的内容。在介绍内核组件的同时，介绍相应的系统工具。另外，也介绍了有关引导、登录以及安全性等方面的内容，各章的内容相对独立。通过这部分的学习，读者可以对 Linux 有比较深刻的理解，可为系统管理、应用开发和软件开发打下坚实的基础。

阅读这部分内容，首先要求读者掌握基本的 Linux 命令，并具有一定的上机操作经验，也需要读者具备一定的 PC 硬件和软件知识，尤其是 C 语言和数据结构。初学者可参阅本书第一部分。另外，Linux 内核在不断发展，这一部分描述的内容是基于 2.0.xx 版本的。本书写作的时候，流行的 Linux 发行版本所采用的 Linux 内核版本最高为 2.0.36，而最高的稳定版本是 2.2.3。

第九章

Linux 系统概述

在详细了解 Linux 奥秘之前，本章首先向读者简单介绍 Linux 的组成部分以及作为操作系统，Linux 为用户提供的主要服务。

9.1 操作系统的概念和组成部分

从程序员的角度来讲，操作系统提供了一个与计算机硬件等价的扩展或虚拟的计算平台。它抽象了许多硬件细节，程序可以以某种统一的方式进行数据处理，而程序员则可以避开许多硬件细节。从另一个角度讲，普通用户则把操作系统看成是一个资源管理者，在它的帮助下，用户可以以某种易于理解的方式组织自己的数据，完成自己的工作并和其他人共享资源。

实际上，操作系统一般由内核和一些系统程序组成，同时，还有一些应用程序帮助用户完成特定任务。内核是操作系统的灵魂，它负责管理磁盘上的文件、内存，负责启动并运行程序，负责从网络上接收和发送数据包等等。总而言之，操作系统实际是抽象的资源操作到具体硬件操作细节之间的接口。对 Linux 这样的多用户操作系统来说，它还需要避免用户对硬件的直接访问，并防止用户之间的互相干扰。

系统程序以及其他所有的程序在内核之上运行，程序和内核之间的接口由操作系统提供的一组“抽象指令”定义，这些抽象指令称为“系统调用”。所有运行在内核之上的程序可分为系统程序和用户程序两大类，但它们统统运行在“用户模式”之下。系统程序和用户程序之间的界限是模糊的。系统程序一般指运行系统所不可缺少的程序，例如 Linux 中的 shell；而用户程序则是给用户提供特定功能的程序，例如字处理程序或游戏程序。实际的操作系统中往往还包含一些工具程序（如编译器）以及一些联机文档。

9.2 Linux 内核的重要组成部分

Linux 内核由如下几部分组成：内存管理、进程管理、设备驱动程序、文件系统和网络管理等。下面分别简要介绍内核的主要组成部分。

9.2.1 内存管理

对任何一台计算机而言，其内存以及其他资源都是有限的。为了让有限的物理内存满

足应用程序对内存的大需求量，Linux 采用了称为“虚拟内存”的内存管理方式。Linux 将内存划分为容易处理的“内存页”，在系统运行过程中，应用程序对内存的需求大于物理内存时，Linux 可将暂时不用的内存页交换到硬盘上，这样，空闲的内存页可以满足应用程序的内存需求，而应用程序却不会注意到内存交换的发生。

有关内存管理的详细内容在第十章中讲述。

9.2.2 进程

进程实际是某特定应用程序的一个运行实体。在 Linux 系统中，能够同时运行多个进程，Linux 通过在短的时间间隔内轮流运行这些进程而实现“多任务”。这一短的时间间隔称为“时间片”，让进程轮流运行的方法称为“调度”，完成调度的程序称为调度程序。通过多任务机制，每个进程可认为只有自己独占计算机，从而简化程序的编写。每个进程有自己单独的地址空间，并且只能由这一进程访问，这样，操作系统避免了进程之间的互相干扰以及“坏”程序对系统可能造成的危害。

为了完成某特定任务，有时需要综合两个程序的功能，例如一个程序输出文本，而另一个程序对文本进行排序。为此，操作系统还提供进程间的通讯机制来帮助完成这样的任务。Linux 中常见的进程间通讯机制有信号、管道、共享内存、信号量和套接字等。

有关进程以及进程间通讯的详细内容在第十一章中讲述。

9.2.3 设备驱动程序

设备驱动程序是 Linux 内核的主要部分。和操作系统的其他部分类似，设备驱动程序运行在高特权级的处理器环境中，从而可以直接对硬件进行操作，但正因为如此，任何一个设备驱动程序的错误都可能导致操作系统的崩溃。设备驱动程序实际控制操作系统和硬件设备之间的交互。设备驱动程序提供一组操作系统可理解的抽象接口完成和操作系统之间的交互，而与硬件相关的具体操作细节由设备驱动程序完成。一般而言，设备驱动程序和设备的控制芯片有关，例如，如果计算机硬盘是 SCSI 硬盘，则需要使用 SCSI 驱动程序，而不是 IDE 驱动程序。

有关硬件以及设备驱动程序的详细内容在第十二章中讲述。

9.2.4 文件系统

和 DOS 等操作系统不同，Linux 操作系统中单独的文件系统并不是由驱动器号或驱动器名称（如 A: 或 C: 等）来标识的。相反，和 UNIX 操作系统一样，Linux 操作系统将独立的文件系统组合成了一个层次化的树形结构，并且由一个单独的实体代表这一文件系统。Linux 将新的文件系统通过一个称为“挂装”或“挂上”的操作将其挂装到某个目录上，从而让不同的文件系统结合成为一个整体。Linux 操作系统的一个重要特点是它支持许多不同类型的文件系统。Linux 中最普遍使用的文件系统是 Ext2，它也是 Linux 土生土长的文件系统。但 Linux 也能够支持 FAT、VFAT、FAT32、MINIX 等不同类型的文件系统，从而可以方便地和其他操作系统交换数据。由于 Linux 支持许多不同的文件系统，并且将它们组织成了一个统一的虚拟文件系统，因此，用户和进程不需要知道文件所在的文件系统类型，而只需要象使用 Ext2 文件系统中的文件一样使用它们。

实际上，Linux 利用虚拟文件系统，把文件系统操作和不同文件系统的具体实现细节分离了开来。

有关文件系统的详细内容在第十三章中讲述。

9.2.5 网络

Linux 和网络几乎就是同义语。Linux 实际就是 Internet 和 WWW 的产物。Linux 的开发者使用网络和 Web 进行信息交换，而 Linux 本身又用于各种组织的网络支持。众所周知，TCP/IP 协议是 Internet 的标准协议，同时也是事实上的工业标准。Linux 的网络实现支持 BSD 套接字，支持全部的 TCP/IP 协议。Linux 内核的网络部分由 BSD 套接字、网络协议层和网络设备驱动程序组成。

有关网络的基本概念在第十四章中讲述。

9.2.6 其他

除上述主要组成部分之外，内核还包含一些一般性的任务和机制，这些任务和机制可使 Linux 内核的各个部分有效地组合在一起，它们是上述主要部分高效工作的必要保证。

从结构上来讲，操作系统有微内核结构和单块结构之分，Windows NT 和 MINIX 是典型的微内核操作系统，而 Linux 则是单块结构的操作系统。微内核结构可方便地在内核中添加新的组件，而单块结构却不容易做到这一点。为此，Linux 支持可动态装载和卸载的模块。利用模块，可方便地在内核中添加新的组件或卸载不再需要的内核组件。

第十五章讲述上述内核机制和模块的概念。

图 9-1 说明了上述 Linux 内核的重要组成部分及其相互关系。

9.3 Linux 系统的主要服务

本节描述的主要服务也是 UNIX 系统所提供的主要服务，以后的各章节将详细描述这些服务。

9.3.1 init

init 作为每个 UNIX 系统的第一个进程而启动，同时也是引导过程中内核的最后一项工作。**init** 启动之后，它将继续引导过程，包括检查并加载文件系统，以及启动守护进程等。

和 **init** 相关的概念有“单用户模式”和“多用户模式”。在“单用户模式”中，只有 root 才能登录到系统中。这两个概念常引申为“运行级别”，上述两种模式分别对应 **init** 的两个运行级别，但 **init** 的运行级别有 6 个之多，其他运行级别可由应用程序使用。

系统关机时，**init** 负责“杀掉”所有的进程，卸挂所有文件系统并终止处理器的指令执行。

有关引导和关机以及 **init** 的详细内容将在第十六章中讲述。

9.3.2 终端登录

通过串行线路的终端登录和以及控制台登录（不运行 X Window 时）由 **getty** 程序提供。**init** 为每个终端启动一个单独的 **getty** 实例。**getty** 读取用户名并启动 **login** 程序，如果用户名和口令是正确的，**login** 则会启动 shell 程序。如果用户注销，或由于不正确的用户名和口令而导致 **login** 终止，**init** 就会启动一个新的 **getty**。用户的登录完全由系统程序完成，内核不参与登录过程。

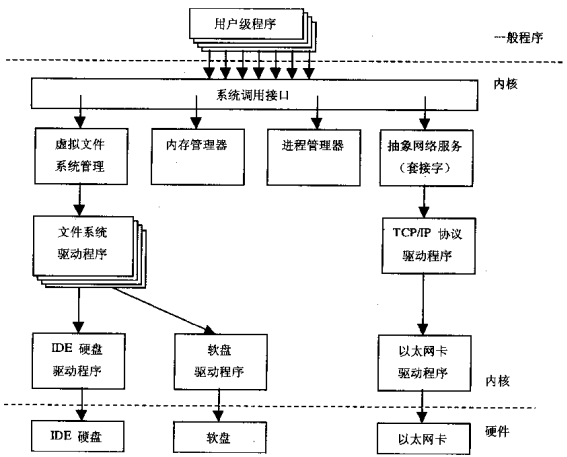


图 9-1 Linux 内核的重要组成部分

有关登录和注销的详细内容将在这一部分第十七章中讲述。

9.3.3 Syslog

内核和系统程序经常会产生一些错误、警告或其他类型的消息。这些消息通常对系统管理非常重要，因此有必要记录这些消息以备将来查看。这一工作由 **syslog** 程序完成。**syslog** 可按照产生消息的程序以及消息的重要性对消息进行整理，以方便查看。

9.3.4 周期命令执行：cron 和 at

系统管理员或用户经常要让一些周期性的任务自动完成，以便减少日常工作量，例如，每周定时清除 /tmp 目录中的文件等。

cron 是一个守护进程，它可以完成类似的工作。每个用户有一个 crontab 文件，用户可在这个文件中列出要执行的命令以及执行的时间，**cron** 则按照指定的时间和命令完成相应的工作。

at 和 **cron** 类似，但是由 **at** 执行的命令只能执行一次而不会重复执行。

9.3.5 图形用户界面

Linux 并没有直接将用户界面结合到操作系统内核中，相反，用户界面由用户级的程序实现。用户界面包括字符用户界面和图形用户界面。Linux 主要使用 X Window 系统（简称为 X）作为它的图形环境，但是，X 只是一个窗口系统，它仅仅管理窗口的建立、销毁以及窗口中的图形输出，而实际的窗口管理程序则需要单独实现。窗口管理程序的不同意味着不同的图形用户界面风格，这种多样化的界面风格虽然有很大的灵活性，但是它使得用户界面的学习变得有些困难。Linux 中，常见基于 X 的用户界面风格有 Xwmf、Xwmf2、Motif 以及 AfterStep 等。

9.3.6 网络

如前所述，Linux 支持完整的 TCP/IP 协议，许多基本的操作系统服务，例如文件系统、打印、备份等均可以通过网络实现，这使得网络的中心化管理成为可能。

9.3.7 网络登录

网络登录和通常的登录有一些不同。由于受到物理线路数量的限制，通过串行线路的登录数目是有限的，但是，每个网络登录可由一个单独的虚拟网络连接来代表，因此，网络登录的数量几乎不受限制。由于这一原因，不可能象终端登录一样为每个可能的网络登录运行单独的 **getty**。同时，网络登录的方式也是多样的，例如可使用 **telnet** 登录，也可使用 **rlogin** 登录。

为此，对于所有的网络登录方式，系统运行一个守护进程（**inetd**），这一守护进程监听所有的网络登录。当某个用户试图登录时，守护进程会启动一个新的自身实例处理登录，而原有的进程则继续监听。新进程接下来的工作和 **getty** 类似。

9.3.8 网络文件系统

通过网络服务可实现文件的共享，通常使用由 Sun Microsystem 开发的网络文件系统，即 NFS。在网络文件系统的帮助下，程序可象处理本地文件一样处理其他计算机上的文件，这样，信息的共享非常简单，因为它不需要对程序进行额外修改。

9.3.9 其他

电子邮件是计算机间常见的信息交换方式，系统提供了许多程序来收发电子邮件。

打印机是比较特殊的一种设备，因为同一时刻只能有一个用户使用打印机。系统通过

打印队列协调多个用户对打印机的使用。当打印机正在打印时，其他的打印作业在打印队列中排队，上一个打印作业完毕后，下一个打印作业自动开始。打印队列的管理软件可将程序的打印输出假脱机到磁盘上，这样，输出打印的程序不需要等待打印结束就可以继续运行。

9.4 目录树的标准布局

对大多数 Linux 发行版本而言，文件系统的目录树布局遵循 FSSTND 标准。遵循这一标准有利于编写或移植软件，同时也有利于进行系统管理和维护。虽然并不强制执行 FSSTND 标准，但遵循这一标准会给日常工作带来许多好处。本节简要描述基于 FSSTND 标准的 Linux 目录树布局，有关详细内容可参阅 FSSTND 文档。

完整的目录树可划分为小的部分，这些小部分又可以单独存放在自己的磁盘或分区上。这样，相对稳定的部分和经常变化的部分可单独存放在不同的分区中，从而方便备份或系统管理。目录树的主要部分有 root、/usr、/var、/home 等（图 9-2）。这样的布局可方便在 Linux 计算机之间共享文件系统的某些部分。

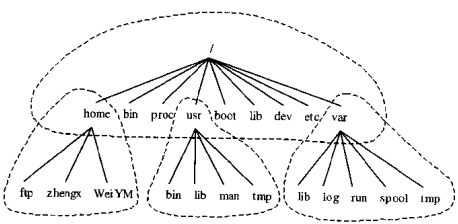


图 9-2 Linux 目录树的不同部分

目录树的不同部分在 Linux 系统中所扮演的角色由表 9-1 给出。

表 9-1 目录树的不同部分在 Linux 系统中所扮演的角色

名称	角色
root	root 文件系统中包含和特定计算机相关的内容，因此，root 文件系统一般保存在计算机的本地硬盘中，但也可以保存在 ramdisk 或网络驱动器中。root 文件系统中的内容包括：引导系统的必备文件，文件系统的挂装信息以及系统修复工具和备份工具等。
/usr	/usr 文件系统中包含通常操作中不需要进行修改的命令程序文件、程序库、手册和其他文档等。由于 /usr 中的文件并不和特定的计算机相关，也不会在通常的使用中修改，因此可通过网络共享这些文件。这样，当 root 安装了新的软件之后，所有共享这一文

	件系统的计算机均可以使用新的软件。
/var	/var 文件系统中包含经常变化的文件，例如打印机、邮件、新闻等的假脱机目录、日志文件、格式化后的手册页以及临时文件等。按传统习惯，/var 的文件通常包含在在 /usr 中，但这样就无法将 /usr 文件系统挂装为只读文件系统。
/home	/home 中包含用户的主目录，用户的数据保存在其主目录中，如果有必要，也可将 /home 划分为不同的文件系统，例如 /home/students 和 /home/teachers 等。将 /home 放置在单独的文件系统中便于进行用户数据的备份。
/proc	/proc 文件系统并不保存在磁盘上，相反，操作系统在内存中创建这一文件系统。其中包含一些和系统相关的信息，例如 CPU、DMA 通道以及中断的使用信息等。

上表中，目录树的不同部分又称为文件系统，但它们不必保存在单独的文件系统中。

9.4.1 root 文件系统

root 文件系统中包含一些关键文件，同时其内容也比较小。如果 root 文件系统被破坏，操作系统就无法正确引导。root 文件系统中包含的文件和目录见表 9-2。

表 9-2 root 文件系统中的文件和目录

/vmlinuz	文件。系统的标准引导映象，通常以压缩形式出现。
/bin	包含引导过程必需的命令，也可由普通用户使用。
/sbin	和 /bin 类似，尽管其中的命令可由普通用户使用，但由于这些命令属于系统级命令，因此无特殊需求不使用其中的命令。
/etc	包含与特定计算机相关的配置文件。
/root	root 用户的主目录。
/lib	root 文件系统中的程序要使用的共享库保存在该目录中。
/lib/modules	包含可装载的内核模块。
/dev	包含设备文件。
/tmp	包含临时文件，引导后运行的程序应当在 /var/tmp 中保存文件，因为其中的可用空间大一些。
/boot	包含引导装载程序要使用的文件。内核映象通常保存在这个目录中。因为多个内核映象会占用很多磁盘空间，因此可将该目录放置在单独的文件系统中。
/mnt	临时文件系统的挂装点。
/usr, /var, /home, /proc	其他文件系统的挂装点。

9.4.2 /usr 文件系统

/usr 文件系统中包含所有的程序文件以及联机文档，因此其内容通常很大。/usr 文件系统中包含的文件和目录见表 9-3。

表 9-3 /usr 文件系统中的目录

/usr/X11R6	包含 X 窗口系统的所有文件。
/usr/X386	和 /usr/X11R6 类似，但包含 X11 的 Release 5。
/usr/bin	绝大多数用户命令。其他命令包含在 /bin 和 /usr/local/bin 中。

/usr/sbin	root 文件系统中不需要的系统管理命令。
/usr/man, /usr/info, /usr/doc	分别包含手册页、GNU Info 文档以及其他杂项文档。
/usr/include	C 语言的头文件。
/usr/lib	程序和子系统所使用的不变的数据文件。
/usr/local	本地挂装的软件和其他文件的存放位置。

9.4.3 /var 文件系统

/var 通常包含系统运行过程中经常发生变化的文件。/var 文件系统中包含的目录见表 9-4。

表 9-4 /var 文件系统中的目录

/var/catman	格式化手册页的高速缓存。
/var/lib	包含系统运行时经常改变的文件。
/var/local	安装 /usr/local 中的程序的可变数据。
/var/lock	包含锁文件。
/var/log	包含程序产生的日志文件。
/var/run	该目录包含在下次引导之前有效和系统相关的信息。
/var/spool	排队任务的假脱机目录。
/var/tmp	包含大的临时文件，或者保存时间较长的临时文件。

9.4.4 /proc 文件系统

/proc 文件系统并不保存在磁盘上，相反，操作系统在内存中创建这一文件系统。/proc 文件系统中包含的文件和目录见表 9-5。

表 9-5 /proc 文件系统中的文件和目录

/proc/1	该目录中包含进程号为 1 的进程信息，每个进程在 /proc 目录下有一个以自己的进程号为名称的目录。
/proc/cpuinfo	有关 CPU 名称、型号、性能和类型的信息。
/proc/devices	当前内核中的设备驱动程序列表。
/proc/dma	当前使用的 DMA 通道。
/proc/filesystems	内核支持的文件系统。
/proc/interrupts	当前使用的中断信息。
/proc/ioports	当前使用的 I/O 端口。
/proc/kcore	系统物理内存的映象。

第十章

内 存 管 理

如果读者曾经在 DOS 下开发过应用程序的话，可能会对 DOS 的 640K 常规内存限制记忆犹新。为了在 DOS 下运行大的应用程序，人们开发了各种各样的方法，其中包括象 XMS 和 EMS 这样的内存访问规范，利用这些规范，程序可以使用 640K 之外的其他内存。但是，这并没有从根本上解决问题。计算机中安装的内存数量各不相同，如果要在安装有 4M 内存的计算机上运行 16M 的程序该怎么办呢？解决这一问题的办法之一就是采用“虚拟内存”，实际上，虚拟内存是最成功的解决办法，也是流行操作系统广泛采用的方法。

10.1 虚拟内存

虚拟内存的基本思想就是，在计算机中运行的程序，其代码、数据和堆栈的总量可以超过实际内存的大小，操作系统只将当前使用的程序块保留在内存中，其余的程序块则保留在磁盘上。必要时，操作系统负责在磁盘和内存之间交换程序块。

Linux 也采用虚拟内存管理机制，具体而言，虚拟内存可以提供如下好处：

- 大地址空间，对运行在系统中的进程而言，可用的内存总量可以超过系统的物理内存总量，甚至可以达到好几倍。运行在 i386 平台上的 Linux 进程，其地址空间可达 4GB。
- 进程保护。每个进程拥有自己的虚拟地址空间，这些虚拟地址对应的物理地址完全和其他进程的物理地址隔离，从而避免进程之间的互相影响。
- 内存映射。利用内存映射，可以将程序映象或数据文件映射到进程的虚拟地址空间中，对程序代码和数据的访问与访问内存单元一样。
- 公平的物理内存分配。虚拟内存机制可保证系统中运行的进程平等分享系统中的物理内存。
- 共享虚拟内存。利用虚拟内存可以方便隔离各进程的地址空间，但是，如果将不同进程的虚拟地址映射到同一物理地址，则可实现内存共享。这就是共享虚拟内存的本质，利用共享虚拟内存不仅可以节省物理内存的使用（如果两个进程的部分或全部代码相同，只需在物理内存中保留一份相同的代码即可），而且可以实现所谓“共享内存”的进程间通讯机制（两个进程通过同一物理内存区域进行数据交换）。

Linux 中的虚拟内存采用所谓的“分页”机制。分页机制将虚拟地址空间和物理地址空间划分为大小相同的块，这样的块称为“内存页”或简称为“页”。通过虚拟内存地址空间的页与物理地址空间中的页之间的映射，分页机制可实现虚拟内存地址到物理内存地址之间的转换。图 10-1 说明了两个进程的虚拟地址空间的部分页到物理地址空间的部分页之间的映射关系。

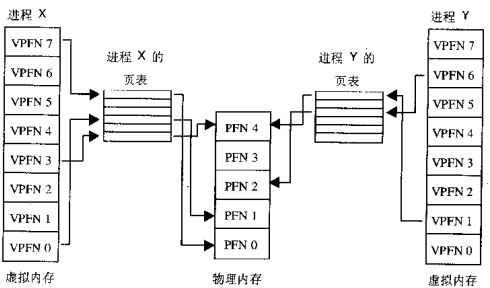
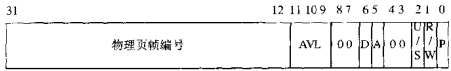


图 10-1 虚拟地址到物理地址的映射模型

i386 平台上的 Linux 页大小为 4K 字节，而在 Alpha AXP 系统中使用 8K 字节的页。不管是虚拟内存页还是物理内存页，它们均被给定一个唯一的“页帧编号（PFN）”。在上述映射模型中，虚拟内存地址由两部分组成，其中一部分就是页帧编号，而另一部分则是偏移量。CPU 负责将虚拟页帧编号翻译为相应的物理页帧编号。物理页帧编号实际是物理地址的高位，也称为页基地址，页基地址加上偏移量就是物理内存地址（这和 DOS 64K 段基地址及段偏移量类似）。为此，CPU 利用“页表”实现虚拟页帧编号到物理页帧编号的转换。

如图 10-1，操作系统可以为不同的进程准备进程的私有页表，每个页表项包含物理页帧编号、页表项的有效标志以及相应的物理页访问控制属性，访问控制属性指定了页是只读页、只写页、可读可写页还是可执行代码页，这有利于进行内存保护，例如，进程不能向代码页中写入数据，图 10-2 给出了 i386 系统中的页表项格式。参照图 10-1，CPU 利用虚拟页帧编号作为访问进程页表的索引来检索页表项，如果当前页表项是有效的，处理器就可以从该页表项中获得物理页帧编号，进而获得物理内存中的页基地址，加上虚拟内存中的偏移量就是要访问的物理地址；如果当前页表项无效，则说明进程访问了一个不存在的虚拟内存区，在这种情况下，CPU 将会向操作系统报告一个“页故障”，操作系统则

负责对页故障进行处理。



P: 页表项有效标志位
R/W: 页的读写属性位
U/S: 用户/系统属性位
A: 访问属性位
D: 写标志位

图 10-2 i386 系统中的页表项格式

一般而言，页故障的原因可能是因为进程访问了非法的虚拟地址，也可能是因为进程要访问的物理地址当前不在物理内存中，这时，操作系统负责将所需的内存页装入物理内存。

上面就是虚拟内存的抽象模型，但 Linux 中的虚拟内存机制要复杂一些。从性能的角度考虑，如果内核本身也需要进行分页，并为内核代码和数据页维护一个页表的话，则系统的性能会下降很多，为此，Linux 的内核运行在所谓的“物理地址模式”，CPU 不必在这种模式下进行地址转换，物理地址模式的实现和实际的 CPU 类型有关。

10.2 Linux 的内存页表

在 i386 系统中，虚拟地址空间的大小是 4G，因此，全部的虚拟内存空间划分为 1M 页。如果用一个页表描述这种映射关系，那么这一映射表就要有 1M 个表项，当每个表项占用 4 个字节时，全部表项占用的字节数就为 4M，为了避免占用如此巨大的内存资源来存储页表，i386 系列 CPU 采用两级页表。类似地，Alpha AXP 系统使用三级页表。Linux 为了避免硬件的不同细节影响内核的实施，假定有三级页表。如图 10-3 所示，一个虚拟地址可分为多个域，不同域的数据指出了对应级别页表中的偏移量。为了将一个虚拟地址转换为物理地址，处理器根据这三个级别域，每次将一个级别域中的值转换为对应页表的物理页偏移量，然后从中获得下一级页表的页帧编号。如此进行三次，就可以找出虚拟内存对应的实际物理内存。前面提到，不同 CPU 的页级数目不同，Linux 内核源代码则利用 C 语言的宏将具体的硬件差别隐藏了起来。例如，对 i386 的 Linux 内核代码来说，三级的页表转换宏实际只有两个在起作用。

10.3 内存页的分配和释放

系统运行过程中，经常需要进行物理内存页的分配或释放。例如，执行程序时，操作系统需要为相应的进程分配页，而进程终止时，则要释放这些物理页。再如，页表本身也

需要动态分配和释放。物理页的分配和释放机制及其相关数据结构是虚拟内存子系统的关键部分。

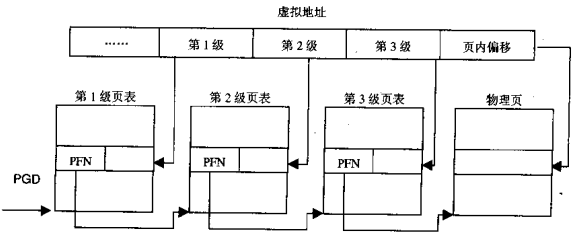


图 10-3 多级页表示意图

一般而言，有两种方法可用来管理内存的分配和释放。一种是采用位图，另外一种是采用链表。

利用位图可记录内存单元的使用情况。例如，如果某个系统有 1024 字节内存，而内存的分配单元是 4 字节，则可以利用 1024/（4*8） =32 个字节来记录使用情况。这 32 个字节的每个位分别代表相应分配单元的使用情况。如果位图中某个位为 1，则对应的分配单元是空闲的。利用这一办法，内存的分配就可以通过对位值的检测来简化。如果一次要分配 5 个单元的空间，内存管理程序就需要找出 5 个连续位值均为 1 的位图位置，但这种操作比较慢，因为连续的位有时要跨越字节边界。

利用链表则可以分别记录已分配的内存单元和空闲的内存单元。通常这些内存单元设计为双向链表结构，从而可加速空闲内存的搜索或链表的处理。这种方法相对位图方法要好一些，也更加有效。

Linux 的物理页分配采用链表和位图结合的方法。参照图 10-4，Linux 内核定义了一个称为 free_area 的数组，该数组的每一项描述某一种页块的信息。第一个元素描述单个页的信息，第二个元素则描述以 2 个页为一个块的页块信息，第三个元素描述以 4 个页为一个块的页块信息，依此类推，所描述的页块大小以 2 的倍数增加。free_area 数组的每项包含两个元素：list 和 map。list 是一个双向链表的头指针，该双向链表的每个节点包含空闲页块的起始物理页帧编号；而 map 则是记录这种页块组分配情况的位图，例如，位图的第 N 位为 1，则表明第 N 个页块是空闲的。从图中也可以看到，用来记录页块组分配情况的位图大小各不相同，显然页块越小，位图越大。

图 10-4 中，free_area 数组的元素 0 包含了一个空闲页（页帧编号为 0）；而元素 2 则包含了两个以 4 页为大小的空闲页块，第一个页块的起始页帧编号为 4，而另…

个页块的起始页帧编号为 56。

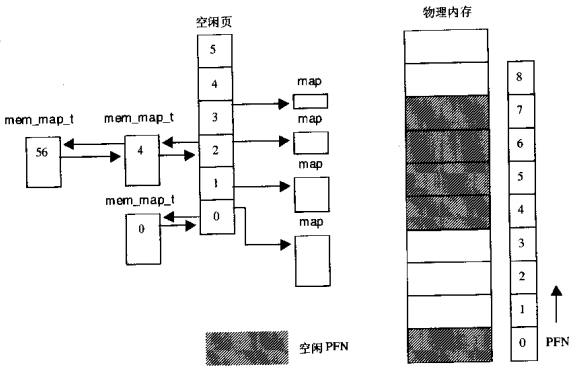


图 10-4　Linux 物理页块的分配示意

Linux 采用 Buddy 算法有效分配和释放物理页块。按照上述数据结构，Linux 可以分配的内存大小只能是 1 个页块，2 个页块或 4 个页块等等。在分配物理页块时，Linux 首先在 free_area 数组中搜索大于或等于要求尺寸的最小页块信息，然后在对应的 list 双向链表中寻找空闲页块，如果没有空闲页块，Linux 则继续搜索更大的页块信息，直到发现一个空闲的页块为止。如果搜索到的页块大于满足要求的最小页块，则只需将该页块剩余的部分划分为小的页块并添加到相应的 list 链表中。

页块的分配会导致内存的碎片化，而页块的释放则可以将页块重新组合成大的页块。如果和被释放的页块大小相等的相邻页块是空闲的，则可以将这两个页块组合成一个大的页块，这一过程一直继续，直到把所有可能的页块组合成尽可能大的页块为止。

知道了上述原理，读者可以自己想象系统启动时，初始的 free_area 数组中的信息。

10.4　内存映射和需求分页

当某个程序映象开始运行时，可执行映象必须装入进程的虚拟地址空间。如果该程序

用到了任何一个共享库，则共享库也必须装入进程的虚拟地址空间。实际上，Linux 并不将映象装入物理内存，相反，可执行文件只是被链接到进程的虚拟地址空间中。随着程序的运行，被引用的程序部分会由操作系统装入物理内存。这种将映象链接到进程地址空间的方法称为“内存映射”。

每个进程的虚拟内存由一个 mm_struct 结构代表，我们将在下一章中详细讲述该结构。该结构中实际包含了当前执行映象的有关信息，并且包含了一组指向 vm_area_struct 结构的指针。如图 10-5 所示，每个 vm_area_struct 描述了一个虚拟内存区域的起点和终点、进程对内存的访问权限以及一个对内存的操作例程集。操作例程集是 Linux 操作该内存区域时所使用的例程集合。例如，当进程试图访问的虚拟内存当前不在物理内存当中时（通过页故障），Linux 就可以利用操作集中的一个例程执行正确的操作，在这种情况下为 nopage 操作。

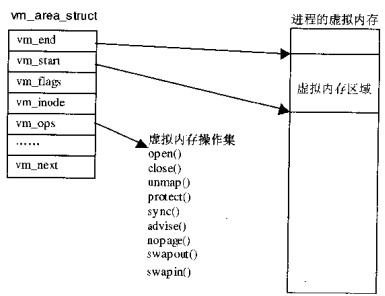


图 10-5 vm_area_struct 数据结构示意图

当可执行映象映射到进程的虚拟地址空间时，将产生一组 vm_area_struct 结构来描述虚拟内存区域的起始点和终止点，每个 vm_struct 结构代表可执行映象的一部分，可能是可执行代码，也可能是初始化的变量或未初始化的数据。随着 vm_area_struct 结构的生成，这些结构所描述的虚拟内存区域上的标准操作函数也由 Linux 初始化。

某个可执行映象映射到进程虚拟内存中并开始执行时，因为只有很少一部分装入了物理内存，因此很快就会访问尚未装入物理内存的虚拟内存区域。这时，处理器将向 Linux 报告一个页故障及其对应的故障原因。

这种页故障的出现原因有两种，一是程序出现错误，例如向随机物理内存中写入数据，这种情况下，虚拟内存是无效的，Linux 将向程序发送 SIGSEGV 信号并终止程序的运

行；另一种情况是，虚拟地址有效，但其所对应的页当前不在物理内存中，这时，操作系统必须从磁盘映象或交换文件中将内存装入物理内存。

那么，Linux 如何判断页故障发生时，虚拟内存地址是否是有效的呢？如前所述，Linux 利用 vm_area_struct 数据结构描述进程的虚拟内存空间，为了查找出现页故障虚拟内存相应的 vm_area_struct 结构的位置，Linux 内核同时维护一个由 vm_area_struct 结构形成的 AVL（Adelson-Velskii and Landis）树。利用 AVL 树，可快速寻找发生页故障的虚拟地址所在的内存页区域。如果搜索不到这一内存区域，则说明该虚拟地址是无效的，否则该虚拟地址是有效的。

也有可能因为进程在虚拟地址上进行的操作非法而产生页故障，例如在只读页中写入数据。这时操作系统会同样发送内存错误信号到该进程。有关页的访问控制信息（只读页、只写页、可读可写页、可执行代码页等）包含在页表项中。

对有效的虚拟地址，Linux 必须区分页所在的位置，即判断页是在交换文件中，还是在可执行映象中。为此，Linux 通过页表项中的信息区分页所在的位置。如果该页的页表项是无效的，但非空，则说明该页处于交换文件中，操作系统要从交换文件装入页（有关内存交换的内容在下一节中讲述）。否则，默认情况下，Linux 会分配一个新的物理页并建立一个有效的页表项；对于映象的内存映射来讲，则会分配新的物理页，更新页表项属性信息，并从映象中装入页。

这时，所需的页装入了物理内存，页表项也同时被更新，然后进程就可以继续执行了。这种只在必要时才将虚拟页装入物理内存的处理称为“需求分页”。

在处理页故障的过程中，因为要涉及到磁盘访问等耗时操作，因此操作系统会选择另外一个进程进入执行状态。

10.5 Linux 页缓存

经内存映射的文件每次只读取一页内容，读取后的页保存在页缓存中，利用页缓存，可提高文件的访问速度。如图 10-6 所示，页缓存由 page_hash_table 组成，它是一个 mem_map_t 数据结构的指针向量。页缓存的结构是 Linux 内核中典型的哈希表结构。众所周知，对计算机内存的线性数组的访问是最快速的访问方法，因为线性数组中的每一个元素的位置都可以利用索引值直接计算得到，而这种计算是简单的线性计算。但是，如果要处理大量数据，有时由于受到存储空间的限制，采用线性结构是不切合实际的。但如果采用链表等非线性结构，则元素的检索性能又会大打折扣。哈希表则是一种折衷的方法，它综合了线性结构和非线性结构的优点，可以在大量数据中进行快速的查找。哈希表的结构有多种，在 Linux 内核中，常见的哈希结构和图 10-6 的结构类似。要在这种哈希表中访问某个数据，首先要利用哈希函数以目标元素的某个特征值作为函数自变量生成哈希值作为索引，然后利用该索引访问哈希表的线性指针向量。哈希线性表中的指针代表一个链表，该链表所包含的所有节点均具有相同的哈希值，在该链表中查找可访问到指定的数据。哈希函数的选择非常重要，不恰当的哈希函数可能导致大量数据映射到同一哈希值，这种情况下，元素的查找将相当耗时。但是，如果选择恰当的哈希函数，则可以在性能和空间上得到均衡效果。

在 Linux 页缓存中，访问 page_hash_table 的索引由文件的 VFS（虚拟文件系

统）索引节点 inode 和内存页在文件中的偏移量生成。有关 VFS 索引节点的内容将在第十三章中讲述，在这里，应知道每个文件的 VFS 索引节点 inode 是唯一的。

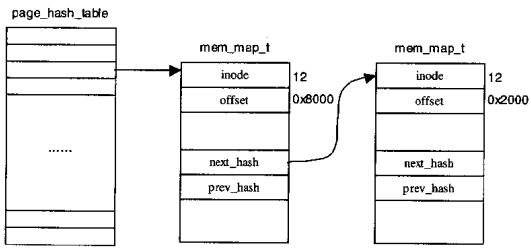


图 10-6 Linux 页缓存示意图

当系统要从内存映射文件中读取某页时，首先在页缓存中查找，如果发现该页保存在缓存中，则可以免除实际的文件读取，而只需从页缓存中读取，这时，指向 nun_map_t 数据结构的指针被返回到页故障的处理代码；如果该页不在缓存中，则必须从实际的文件系统映象中读取页，Linux 内核首先分配物理页然后从磁盘读取页内容。

如果可能，Linux 还会预先读取文件中下一页内容，这样，如果进程要连续访问页，则下一页的内容不必再次从文件中读取了，而只需从页缓存中读取。

随着映象的读取和执行，页缓存中的内容可能会增多，这时，Linux 可移走不再需要的页。当系统中可用的物理内存量变小时，Linux 也会通过缩小页缓存的大小而释放更多的物理内存页。

10.6 内存交换

当物理内存出现不足时，Linux 内存管理子系统需要释放部分物理内存页。这一任务由内核的交换守护进程 **kswaped** 完成，该内核守护进程实际是一个内核线程，它的任务就是保证系统中具有足够的空闲页，从而使内存管理子系统能够有效运行。

在系统启动时，这一守护进程由内核的 **init** 进程启动。当内核的交换定时器到期时，该进程开始运行。如果 **kswaped** 发现系统中的空闲页很少，该进程将按照下面的三种方法减少系统使用的物理页：

1. 减少缓冲区和页高速缓存的大小。页高速缓存中包含内存映射文件的页，可能包含一些系统不再需要的页，类似地，缓冲区高速缓存中也可能包含从物理设备中

读取的或写入物理设备的数据，这些缓冲区也可能不再需要，因此，这两个高速缓存可用来释放出空闲页。但是，同时处于这两个高速缓存中的页是不能丢弃的。Linux 利用“时钟”算法从系统中选择要丢弃的页，也即每次循环检查 mem_map 页向量中不同的页块，象时钟的分针循环转动一样。时钟算法的原理见图 10-7。每次内核的交换进程运行时，根据对物理内存的需求而选择不同页块大小的 mem_map 向量进行检查。如果发现某页块处于上述两个高速缓存中，则释放相应的缓冲区，并将页块重新收入 free_area 结构。

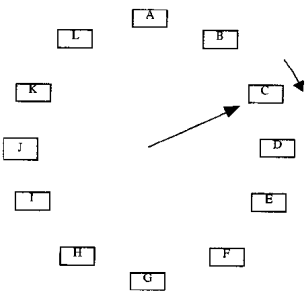


图 10-7 页交换的时钟算法

2. 将 System V 共享内存页交换出物理内存。System V 共享内存页实际是一种进程间通讯机制，系统通过将共享内存页交换到交换文件而释放物理内存。Linux 同样使用时钟算法选择要交换出物理内存的页。
3. 将页交换出物理内存或丢弃。**kswaped** 首先选择可交换的进程，或其中某些页可从内存中交换出或丢弃的进程。可执行映象的大部分内容可从磁盘映象中获取，因此，这些页可丢弃。选定要交换的进程之后，Linux 将把该进程的一小部分页交换出内存，而大部分不会被交换，另外，被锁定的页也不会被交换。Linux 利用页的寿命信息选择要交换的页，也即所谓“最近最少使用（LRU）”算法。

页的寿命信息保存在 mem_map_t 结构中。最初分配某个页时，页的寿命为 3，每次该页被访问，其寿命增加 3，直到 20 为至；而当内核的交换进程运行时，页寿命减 1。如果某个页的寿命为 0，则该页可作为交换候选页。如果是“脏”页（该信息保存在页表项中），则可将该页交换出物理内存。但是，进程的虚拟内存区域可具有自己的交换操作例程（定义在虚拟内存操作集中），这时，将利用该例程执行交换操作，否则，交换守护进程在交换文件中分配页，并将该页写入交换文件。

当某物理页交换到交换文件之后，该页对应的页表项被标志为无效，同时包换该页在交换文件中的位置信息；而被释放出的物理页则被收回到 free_area 数据结构中。

根据被释放的页数目，kswaped 会自动调节交换定时器的间隔，以便能够有足够的时间释放更多的页而保证足够的空闲页。

交换文件中的页是经过修改的页（通过在页表项中设置相应的位而标志该页为“脏”页），则当进程再次访问该页时，操作系统必须从交换文件中将该页交换到物理内存。

10.7 高速缓存

不管在硬件设计还是软件设计中，高速缓存是获得高性能的常用手段。Linux 使用了多种和内存管理相关的高速缓存：

- 缓冲区高速缓存：缓冲区高速缓存中包含了由块设备使用的数据缓冲区。这些缓冲区中包含了从设备中读取的数据块或写入设备的数据块。缓冲区高速缓存由设备标识号和块标号索引，因此可以快速找出数据块。如果数据能够在缓冲区高速缓存中找到，则系统就没有必要在物理块设备上进行实际的读操作。
- 页高速缓存：这一高速缓存用来加速对磁盘上的映象和数据的访问。它用来缓存某个文件的逻辑内容，并通过文件的 VFS 索引节点和偏移量访问。当页从磁盘上读到物理内存时，就缓存在页高速缓存中。
- 交换高速缓存：只有修改后（脏）的页才保存在交换文件中。修改后的页写入交换文件后，如果该页再次被交换但未被修改时，就没有必要写入交换文件，相反，只需丢弃该页。交换高速缓存实际包含了一个页表项链表，系统的每个物理页对应一个页表项。对交换出的页，该页表项包含保存该页的交换文件信息，以及该页在交换文件中的位置信息。如果某个交换页表项非零，则表明保存在交换文件中的对应物理页没有被修改。如果这一页在后续的操作中被修改，则处于交换缓存中的页表项被清零。当 Linux 需要从物理内存中交换出某个页时，它首先分析交换缓存中的信息，如果缓存中包含该物理页的一个非零页表项，则说明该页交换出内存后还没有被修改过，这时，系统只需丢弃该页。
- 硬件高速缓存：常见的硬件缓存是对页表项的缓存，这一工作实际由处理器完成，其操作和具体的处理器硬件有关，对这一缓存的描述已超出本书的讲述范围。

10.8 相关系统工具和系统调用

实际上，Linux 可以利用文件系统中通常的文件作为交换文件，也可以利用某个分区进行交换操作，因此，通常把交换文件或交换分区称为“交换空间”。在交换分区上的交换操作较快，而利用交换文件可方便改变交换空间大小。Linux 还可以使用多个交换分区或交换文件进行交换操作。本节主要讲述有关交换空间的系统工具。

10.8.1 建立交换空间

作为交换空间的交换文件实际就是通常的文件，但文件的扇区必须是连续的，也即，

文件中必须没有“洞”，另外，交换文件必须保存在本地硬盘上。

由于内核要利用交换空间进行快速的内存页交换，因此，它不进行任何文件扇区的检查，而认为扇区是连续的。由于这一原因，交换文件不能包含洞。可用下面的命令建立无洞的交换文件：

```
$ dd if=/dev/zero of=/extra-swap bs=1024 count=2048
2048+0 records in
2048+0 records out
```

上面的命令建立了一个名称为 extra-swap，大小为 2048K 字节的交换文件。对 i386 系统而言，由于其页尺寸为 4K，因此最好建立一个大小为 4K 倍数的交换文件；对 Alpha AXP 系统而言，最好建立大小为 8K 倍数的交换文件。

交换分区和其他分区也没有什么不同，可象建立其他分区一样建立交换分区。但该分区不包含任何文件系统。分区类型对内核来讲并不重要，但最好设置为 Linux Swap 类型（即类型 82）。

建立交换文件或交换分区之后，需要在文件或分区的开头写入签名，写入的签名实际是由内核使用的一些管理信息。写入签名的命令为 **mkswap**，如下所示：

```
$ mkswap /extra-swp 2048
Setting up swapspace, size = 2088960 bytes
$
```

这时，新建立的交换空间尚未开始使用。使用 **mkswap** 命令时必须小心，因为该命令不会检查文件或分区内容，因此极有可能覆盖有用的信息，或破坏分区上的有效文件系统信息。

Linux 内存管理子系统将每个交换空间的大小限制在 127M （实际为 (4096-10)*8*4096 = 133890048 Byte = 127.6875Mb）。可以在系统中同时使用 16 个交换空间，从而使交换空间总量达到 2GB。

10.8.2 使用交换空间

利用 **swapon** 命令可将经过初始化的交换空间投入使用。如下所示：

```
$ swapon /extra-swap
$
```

如果在 /etc/fstab 文件中列出交换空间，则可自动将交换空间投入使用：

```
/dev/hda5       none        swap       sw      0       0
/extra-swap     none        swap       sw      0       0
```

实际上，启动脚本会运行 **swapon -a** 命令，从而将所有出现在 /etc/fstab 文件中的交换空间投入使用。

利用 **free** 命令，可查看交换空间的使用。如下所示：

```
$ free
                total       used       free     shared    buffers
Mem:            15152      14896        256      12404       2528
-/+ buffers:    12368       2784
Swap:           32452       6684      25768
$
```

该命令输出的第一行（Mem：）显示了系统中物理内存的使用情况。total 列显示的是系统中的物理内存总量；used 列显示正在使用的内存数量；free 列显示空闲的内存量；shared 列显示由多个进程共享的内存量，该内存量越多越好；buffers 显示了当前的缓冲区高速缓存的大小。

输出的最后一行（Swap：）显示了有关交换空间的类似信息。如果该行的内容均为 0，表明当前没有活动的交换空间。

利用 **top** 命令或查看 /proc 文件系统中的 /proc/meminfo 文件可获得相同的信息。

利用 **swapoff** 命令可移去使用中的交换空间。但该命令应只用于临时交换空间，否则有可能造成系统崩溃。

swapoff -a 命令按照 /etc/fstab 文件中的内容移去所有的交换空间，但任何手工投入使用的交换空间保留不变。

10.8.3 分配交换空间

大多数人认为，交换空间的总量应该是系统物理内存量的两倍，实际上这一规则是不正确的，正确的交换空间大小应按如下规则确定：

1. 估计需要的内存总量。运行想同时运行的所有程序，并利用 **free** 或 **ps** 程序估计所需的内存总量，只需大概估计。
2. 增加一些安全性余量。
3. 减去已有的物理内存数量，然后将所得数据圆整为 MB，这就是应当的交换空间大小。
4. 如果得到的交换空间大小远远大于物理内存量，则说明需要增加物理内存数量，否则系统性能会因为过分的页交换而下降。

当计算的结果说明不需要任何交换空间时，也有必要使用交换空间。Linux 从性能的角度出发，会在磁盘空闲时将某些页交换到交换空间中，以便减少必要时的交换时间。另外，如果在不同的磁盘上建立多个交换空间，有可能提高页交换的速度，这是因为某些硬盘驱动器可同时在不同的磁盘上进行读写操作。

10.8.4 关于缓冲区高速缓存

Linux 采用了缓冲区高速缓存机制，而不同于其他操作系统的“写透”方式，因此有可能出现这种情况：写磁盘的命令已经返回，但实际的写操作还未执行。

基于上述原因，应当使用正常的关机命令关机，而不应直接关掉计算机的电源。用户也可以使用 **sync** 命令刷新缓冲区高速缓存。在 Linux 系统中，除了传统的 **update** 守护进程之外，还有一个额外的守护进程 **dbflush**，这一进程可频繁运行不完整的 **sync** 从而可避免有时由于 **sync** 命令的超负荷磁盘操作而造成的磁盘冻结。

dbflush 在 Linux 系统中由 **update** 启动。如果由于某种原因该进程僵死了，则内核会发送警告信息，这时需要手工启动该进程（/sbin/update）。

10.8.5 系统调用

如期所述，系统调用是应用程序和内核之间的功能接口。大部分的系统调用包含在 Linux 的 libc 库中，通过标准的 C 函数调用方法可以调用这些系统调用。但是，少数系统调用并没有相应的 libc 库接口，这时，可通过 **syscall** 系统调用使用这些系统调用：

```
syscall (SYS_num, arg1, ...);
```

其中，**SYS_num** 是系统调用的编号，**arg1** 是该系统调用的参数。在 i386 系统中，可最多传递 5 个系统调用参数，这和 CPU 可用的寄存器数量有关。

表 10-1 简要列出了和内存管理相关的系统调用。标志列中各字母的意义为：

m：手册页可查；
+：POSIX 兼容；
-：Linux 特有；
c：libc 包含该系统调用；
!：该系统调用和其他系统调用类似，应改用其他 POSIX 兼容系统调用。

在后面介绍系统调用时，使用相同的标志字母。

表 10-1 相关系统调用

系统调用	说明	标志
bdflush	将脏的缓冲区刷新到磁盘上	-c
brk	修改数据段（虚拟内存区域）的大小	mc
gettrlimit	获取资源限制	mc
gettrusage	获取资源使用情况	m
idle	将某个进程设置为可选择交换的进程	mc
mmap	将文件映射到进程的虚拟内存空间	mc
mprotect	读取、写入或执行保护内存	-
munmap	取消文件的内存映射	mc
swapoff	停止内存交换	m-c
swapon	启用内存交换	m-c
sync	刷新缓冲区，使磁盘和缓冲区同步	mc

第十一章

进程及进程间通讯机制

程序是保存在磁盘上的文件，其中包含了计算机的执行指令和数据，而进程则可以看成是运行中的程序。程序是静态的，而进程是动态的。和进程联系在一起的不仅有进程的指令和数据，而且还有当前的指令指针、所有的 CPU 寄存器以及用来保存临时数据的堆栈等，所有这些都随着程序指令的执行在变化。

进程在运行过程中，要使用许多计算机资源，例如 CPU、内存、文件等。Linux 是一个多任务操作系统，同时可能会有多个进程使用同一个资源，因此操作系统要跟踪所有的进程及其所使用的系统资源，以便能够管理进程和资源。

Linux 是一个多任务操作系统，它要保证 CPU 时刻保持在使用状态，如果某个正在运行的进程等待外部设备完成工作（例如等待打印机完成打印任务），这时，操作系统就可以选择其他进程运行，从而保持 CPU 的最大利用率。这就是多任务的基本思想，进程之间的切换由调度程序完成。

Linux 中的每个进程有自己的虚拟地址空间，操作系统的一个最重要的基本管理目的，就是避免进程之间的互相影响。但有时用户也希望能够利用两个或多个进程的功能完成同一任务，为此，Linux 提供许多机制，利用这些机制，进程之间可以进行通讯并共同完成某项任务，这种机制称为“进程间通讯（IPC）”。信号和管道是常见的两种 IPC 机制，但 Linux 也提供其他 IPC 机制。

本章主要描述 Linux 进程的管理、调度以及 Linux 系统支持的进程间通讯机制。

11.1 Linux 进程及线程

Linux 内核利用一个数据结构（task_struct）代表一个进程，代表进程的数据结构指针形成了一个 task 数组（Linux 中，任务和进程是两个相同的术语），这种指针数组有时也成为指针向量。这个数组的大小默认为 512，表明在 Linux 系统中能够同时运行的进程最多可有 512。当建立新进程的时候，Linux 为新的进程分配一个 task_struct 结构，然后将指针保存在 task 数组中。task_struct 结构中包含了许多字段，按照字段功能，可分成如下几类：

- 标识号。系统通过进程标识号唯一识别一个进程，但进程标识号并不是进程对应的 task_struct 结构指针在 task 数组中的索引号。另外，一个进程还有自己的用户和组标识号，系统通过这两个标识号判断进程对文件或设备的访

问权。

- 状态信息。一个 Linux 进程可有如下几种状态：运行、等待、停止和僵死。
- 调度信息。调度程序利用该信息完成进程之间的切换。
- 有关进程间通讯的信息。系统利用这一信息实现进程间的通讯。
- 进程链信息。在 Linux 系统中，除初始化进程之外，任何一个进程都具有父进程。每个进程都是从父进程中“克隆”出来的。进程链则包含进程的父进程指针、和该进程具有相同父进程的兄弟进程指针以及进程的子进程指针。另外，Linux 利用一个双向链表记录系统中所有的进程，这个双向链表的根就是 init 进程。利用这个链表中的信息，内核可以很容易地找到某个进程。
- 时间和定时器。系统在这些字段中保存进程的建立时间，以及在其生命周期中所花费的 CPU 时间，这两个时间均以 jiffies 为单位。这一时间由两部分组成，一是进程在用户模式下花费的时间，二是进程在系统模式下花的时间。Linux 也支持和进程相关的定时器，应用程序可通过系统调用建立定时器，当定时器到期时，操作系统会向该进程发送 SIGALRM 信号。
- 文件系统信息。进程可以打开文件系统中的文件，系统需要对这些文件进行跟踪。系统使用这类字段记录进程所打开的文件描述符信息。另外，还包含指向两个 VFS 索引节点的指针，这两个索引节点分别是进程的主目录以及进程的当前目录。索引节点中有一个引用计数器，当有新的进程指向某个索引节点时，该索引节点的引用计数器会增加计数。未被引用的索引节点的引用计数为 0，因此，当包含在某个目录中的文件正在运行时，就无法删除这一目录，因为这一目录的引用计数大于 0。
- 和进程相关的上下文信息。如前所述，进程可被看成是系统状态的集合，随着进程的运行，这一集合发生变化。进程上下文就是用来保存系统状态的 task_struct 字段。当调度程序将某个进程从运行状态切换到暂停状态时，会在上下文中保存当前的进程运行环境，包括 CPU 寄存器的值以及堆栈信息；当调度程序再次选择该进程运行时，则会从进程上下文信息中恢复进程的运行环境。

11.1.1 标识符信息

和所有的 UNIX 系统一样，Linux 使用用户标识符和组标识符判断用户对文件和目录的访问许可。Linux 系统中的所有文件或目录均具有所有者和许可属性，Linux 据此判断某个用户对文件的访问权限。对一个进程而言，系统在 task_struct 结构中记录如表 11-1 所示的四对标识符。

表 11-1 进程的标识符信息

uid 和 gid	运行进程所代表的用户之用户标识号和组标识号，通常就是执行该进程的用户。
有效 uid 和 gid	某些程序可以将 uid 和 gid 改变为自己私有的 uid 和 gid，系统在运行这样的程序时，会根据修改后的 uid 及 gid 判断程序的特权，例如，是否能够直接进行 I/O 输出等。通过 setuid 系统调用，可将程序的有效

	uid 和 gid 设置为其他用户。在该程序映象文件的 VFS 索引节点中，有效 uid 和 gid 由索引节点的属性描述。
文件系统 uid 和 gid	这两个标识符和上述标识符类似，但用于检查对文件系统的访问许可时。处于用户模式的 NFS 服务器作为特殊进程访问文件时使用这两个标识符。
保存 uid 和 gid	如果进程通过系统调用修改了进程的 uid 和 gid，这两个标识符则保存实际的 uid 和 gid。

11.1.2　进程状态信息

如前所述，Linux 中的进程有四中状态，如表 11-2 所示。

表 11-2　进程的状态信息

运行状态	该进程是当前正在运行的进程；或者，该进程是可以运行的进程，即正在等待调度程序将 CPU 分配给它。
等待状态	进程正在等待某个事件或某个资源。这种进程又分为可中断的进程和不可中断的进程两种。可中断的等待进程可被信号中断，而不可中断的等待进程是正在直接等待硬件状态条件的进程，在任何情况下都不能被中断。
停止状态	进程处于停止状态，通常由于接收到信号而停止，例如，进程在接收到调试信号时处于停止状态。
僵死状态	进程已终止，但在 task 数组中仍占据着一个 task_struct 结构。顾名思义，处于这种状态的进程实际是死进程。

11.1.3　文件信息

如图 11-1 所示，系统中的每个进程有两个数据结构用于描述进程与文件相关的信息。其中，fs_struct 描述了上面提到的两个 VFS 索引节点的指针，即 root 和 pwd。另外，这个结构还包含一个 umask 字段，它是进程创建文件时使用的默认模式，可通过系统调用修改这一默认模式。另一个结构为 files_struct，它描述了当前进程所使用的所有文件信息。从图中可以看出，每个进程能够同时拥有 256 个打开的文件，fs[0] 到 fs[255] 就是指向这些 file 结构的指针。文件的描述符实际就是 fs 指针数组的索引号。

在 file 结构中，f_mode 是文件的打开模式，只读、只写或读写；f_pos 是文件的当前位置；f_inode 指向 VFS 中该文件的索引节点；f_op 包含了对该文件的操作例程集。利用 f_op，可以针对不同的文件定义不同的操作函数，例如一个用来向文件中写数据的函数。Linux 利用这一抽象机制，实现了管道这一进程间通讯机制（将在后面详细描述）。这种抽象方法在 Linux 内核中非常常见，通过这种方法，可使特定的内核对象具有类似 C++ 对象的多态性。

Linux 进程启动时，有三个文件描述符被打开，它们是标准输入、标准输出和错误输出，分别对应 fs 数组的三个索引，即 0、1 和 2。如果启动时进行输入输出重定向，则这些文件描述符指向指定的文件而不是标准的终端输入/输出。每当进程打开一个文件时，就会利用 files_struct 的一个空闲 file 指针指向打开的文件描述结构 file。对文件的访问通过 file 结构中定义的文件操作例程和 VFS 索引节点信息来完成。

11.1.4 虚拟内存

在前一章中看到，进程的虚拟内存包含了进程所有的可执行代码和数据。运行某个程序时，系统要根据可执行映象中的信息，为进程代码和数据分配虚拟内存；进程在运行过程中，可能会通过系统调用动态申请虚拟内存或释放已分配的内存，新分配的虚拟内存必须和进程已有的虚拟地址链接起来才能使用；Linux 进程可以使用共享的程序库代码或数据，这样，共享库的代码和数据也需要链接到进程已有的虚拟地址中。在前一章中还看到，系统利用了需求分页机制来避免对物理内存的过分使用。因为进程可能会访问当前不在物理内存中的虚拟内存，这时，操作系统通过对处理器的页故障处理装入内存页。为此，系统需要修改进程的页表，以便标志虚拟页是否在物理内存中。同时，Linux 还需要知道进程地址空间中任何一个虚拟地址区域的来源和当前所在位置，以便能够装入物理内存。

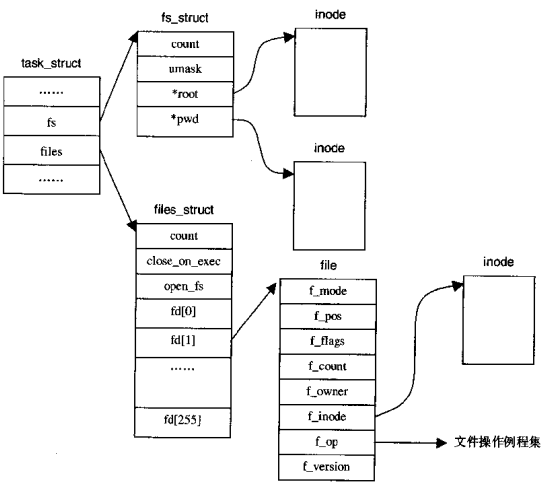


图 11-1 进程的文件信息

由于上面这些原因，Linux 采用了比较复杂的数据结构跟踪进程的虚拟地址。在进程

的 task_struct 结构中包含一个指向 mm_struct 结构的指针。进程的 mm_struct 则包含装入的可执行映象信息以及进程的页表指针。该结构还包含有指向 vm_area_struct 结构的几个指针，每个 vm_area_struct 代表进程的一个虚拟地址区域。

图 11-2 是某个进程的虚拟内存简化布局以及相应的进程数据结构。从图中可以看出，系统以虚拟内存地址的降序排列 vm_area_struct。每个虚拟内存区域可能来源不同，有的可能来自映象，有的可能来自共享库，而有的则可能是动态分配的内存区。因此，Linux 利用了虚拟内存处理例程（vm_ops）来抽象对不同来源虚拟内存的处理方法。

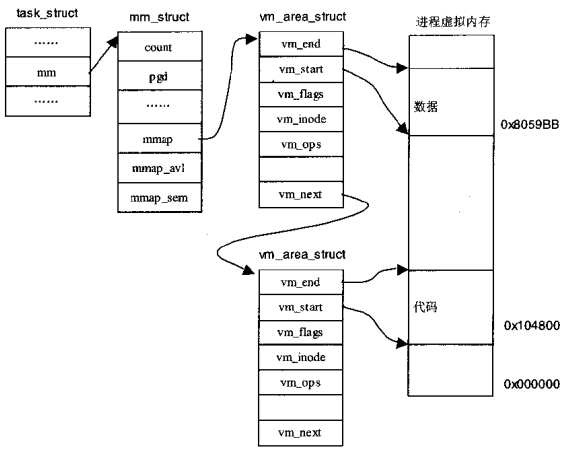


图 11-2　进程的虚拟内存示意

在进程的运行过程中，Linux 要经常为进程分配虚拟地址区域，或者因为从交换文件中装入内存而修改虚拟地址信息，因此，vm_area_struct 结构的访问时间就成了性能的关键因素。为此，除链表结构外，Linux 还利用 AVL（Adelson-Velskii and Landis）树组织 vm_area_struct。通过这种树结构，Linux 可以快速定位某个虚拟内存地址，但在该树中插入或删除节点需要花费较多的时间。

当进程利用系统调用动态分配内存时，Linux 首先分配一个 vm_area_struct 结构，并链接到进程的虚拟内存链表中，当后续的指令访问这一内存区域时，因为 Linux 尚未分配相应的物理内存，因此处理器在进行虚拟地址到物理地址的映射时会产生页故障(详细内容可参阅第十章)，当 Linux 处理这一页故障时，就可以为新的虚拟内存区分配实际的物理内存。

11.1.5 时间和定时器

Linux 保存一个指向当前正在运行的进程之 task_struct 结构的指针，即 current。每当产生一次实时时钟中断（又称时钟滴答），Linux 就会更新 current 所指向的进程的时间信息，如果内核当前代表该进程执行任务（例如进程调用系统调用时），那么系统就将时间记录为进程在系统模式下花费的时间，否则记录为进程在用户模式下花费的时间。

除了为进程记录其消耗的 CPU 时间外，Linux 还支持和进程相关的间隔定时器。当定时器到期时，会向定时器的所属进程发送信号。进程可使用三种不同类型的定时器来给自己发送相应的信号，如表 11-3 所示。

表 11-3 三种不同的进程定时器

Real	该定时器实时更新，到期时发送 SIGALRM 信号。
Virtual	该定时器只在进程运行时更新，到期时发送 SIGVTALRM 信号。
Profile	该定时器在进程运行时，以及内核代表进程运行时更新，到期时发送 SIGPROF 信号。

Linux 对 Virtual 和 Profile 定时器的处理是相同的，在每个时钟中断，定时器的计数值减 1，直到计数值为 0 时发送信号。

对 Real 定时器的处理比较特殊，将在第十五章中讲述。

11.1.6 关于线程

和进程概念紧密相关的概念是线程。线程可看成是进程中指令的不同执行路线。例如，常见的字处理程序中，主线程处理用户输入，而其他并行运行的线程在必要时可在后台保存用户的文档。与进程相关的基本要素有：代码、数据、堆栈、文件 I/O 和虚拟内存信息等，因此，系统对进程的处理要花费更多的开支，尤其在进行进程调度时。利用线程则可以通过共享这些基本要素而减轻系统开支，因此，线程也被称为“轻量级进程”。许多流行的多任务操作系统均支持线程。

线程有“用户线程”和“内核线程”之分。用户线程指不需要内核支持而在用户程序中实现的线程，这种线程甚至在象 DOS 这样的操作系统中也可实现，但线程的调度需要用户程序完成，这有些类似 Windows 3.x 的协作式多任务。另外一种则需要内核的参与，由内核完成线程的调度。这两种模型各有其好处和缺点。用户线程不需要额外的内核开支，但是当一个线程因 I/O 而处于等待状态时，整个进程就会被调度程序切换为等待状态，其他线程得不到运行的机会；而内核线程则没有各个限制，但却占用了更多的系统开支。

Linux 支持内核空间的多线程，读者也可以从 Internet 上下载一些用户级的线程库。Linux 的内核线程和其他操作系统的内核实现不同，前者更好一些。大多数操作系统单独定义线程，从而增加了内核和调度程序的复杂性；而 Linux 则将线程定义为“执行上下

文”，它实际只是进程的另外一个执行上下文而已。这样，Linux 内核只需区分进程，只需要一个进程/线程数组，而调度程序仍然是进程的调度程序。Linux 的 **clone** 系统调用可用来建立新的线程。

11.1.7 会话和进程组

由于 Linux 是一个多用户系统，同一时刻，系统中运行有属于不同用户的多个进程。那么，当处于某个终端上的用户按下了 Ctrl+C 键时（产生 SIGINT 信号），系统如何知道将该信号发送到哪个进程，从而不影响由其他终端上的用户运行的进程呢？

Linux 内核通过维护会话和进程组而管理多用户进程。如图 11-3 所示，每个进程是一个进程组的成员，而每个进程组又是某个会话的成员。一般而言，当用户在某个终端上登录时，一个新的会话就开始了。进程组由组中的领头进程标识，领头进程的进程标识符就是进程组的组标识符。类似地，每个会话也对应有一个领头进程。

同一会话中的进程通过该会话的领头进程和一个终端相连，该终端作为这个会话的控制终端。一个会话只能有一个控制终端，而一个控制终端只能控制一个会话。用户通过控制终端，可以向该控制终端所控制的会话中的进程发送键盘信号。

同一会话中只能有一个前台进程组，属于前台进程组的进程可从控制终端获得输入，而其他进程均是后台进程，可能分属于不同的后台进程组。

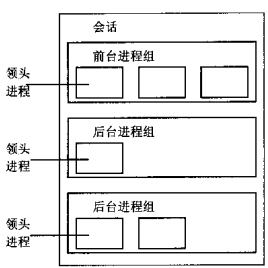


图 11-3 *会话和进程、进程组*

11.2 进程调度

由于 Linux 是一个单块式的操作系统，所以 Linux 中的进程有一些部分运行在用户模式，而另一些部分运行在内核模式，或称系统模式。运行模式的变化是通过系统调用完

成的。系统模式具有更加高级的 CPU 特权级，例如可以直接读取或写入任意的 I/O 端口，设置 CPU 关键寄存器等。Linux 中的进程无法停止当前正在运行的进程，它只能被动地等待调度程序选择它为运行进程，进程的切换操作需要高特权级的 CPU 指令，因此，只能在系统模式中进行，这样，当进行系统调用时，调度程序就有了机会进行进程切换。例如，当某个进程因为系统调用而不得不处于暂停状态时（例如等待用户键入字符），调度程序就可以选择其他的进程运行。进程在运行过程中经常要调用系统调用，因此，调度程序选择其他进程运行的机会还是较多的。但是，进程有时也可能在用户模式下花费大量的 CPU 时间却不调用系统调用，在这种情况下，如果调度程序只能等待该进程在系统调用时才进行进程切换的话，CPU 的分配就显得有些不太公平，更为极端的是，当某个进程进入死循环时，系统就无法响应用户了。为此，Linux 采用抢先式的调度方法，每个进程每次最多只能运行给定的时间段，在 Linux 中为 200ms。当一个进程运行超过 200ms 时，系统选择其他的进程运行，而原有进程则等待下次运行机会。这一时间在抢先式调度中称为“时间片”。

当需要选择下一个运行进程时，由调度程序选择最值得运行的进程。可运行进程实际是仅等待 CPU 资源的进程，如果某个进程在等待其他资源，则该进程是不可运行进程。Linux 使用了比较简单的基于优先级的进程调度算法选择新的进程。当调度程序选择了新的进程之后，它必须在当前进程的 task_struct 结构中保存和该进程相关的 CPU 寄存器和其他有关指令执行的上下文信息，然后从选定进程的 task_struct 结构中恢复 CPU 寄存器以及上下文信息，新的进程就可以继续在 CPU 中执行了。

对于新建的进程，其 task_struct 结构被置为初始的执行上下文，当调度进程选择这一新建进程时，首先从 task_struct 结构中恢复 CPU 寄存器，CPU 的指令计数寄存器（PC）恰好是该进程的初始执行指令地址，这样，新建的进程就可以从头开始运行了。

为了能够为所有的进程平等分配 CPU 资源，内核在 task_struct 结构中记录如表 11-4 所示的信息。

表 11-4 和进程调度相关的 task_struct 信息

字段	描述
policy（策略）	这是系统对该进程实施的调度策略。Linux 进程有两种类型的进程：一般进程和实时进程。实时进程比所有一般进程的优先级高，只有一个实时进程可以运行，调度程序就会选择该进程运行。对实时进程而言，有两种调度策略，一种称为“循环赛（round robin）”，另一种称为“先进先出（first in first out）”。
priority（优先级）	这是系统为进程给定的优先级，可通过系统调用或 **renice** 命令修改该进程的优先级。优先级实际是从进程开始运行算起的、允许进程运行的时间值（以 jiffies 为单位）。
rt_priority（实时优先级）	这是系统为实时进程给定的相对优先级。
counter（计数器）	这是进程运行的时间值（以 jiffies 为单位）。开始运行时设置为 priority，每次时钟中断该值减 1。

调度程序在如下几种情况下运行：当前进程处于等待状态而放入等待队列时；某个系统调用要返回到用户模式之前，这是因为系统调用结束时，当前进程的 counter 值可

能刚好为 0。下面是调度程序每次运行时要完成的任务：

1. 调度程序运行底半处理程序（bottom half handler）处理调度程序的任务队列。这些处理程序实际是一些内核线程，将在第十五章中讲述。
2. 在选择其他进程之前，必须处理当前进程。如果当前进程的调度策略为循环赛，则将当前进程放到运行队列的尾部；如果该进程是可中断的，并且自上次调度以来接收到信号，则任务状态设置为运行；如果当前进程的 counter 值为 0，则任务状态也变为运行；如果当前进程状态为运行，则继续保持此状态；如果进程既不处于运行状态，也不是可中断的，则从运行队列中移去该进程，这表明调度程序在选择最值得运行的进程时，该进程不被考虑。
3. 调度程序在运行队列中搜索最值得运行的程序。调度程序通过比较权重来选择进程。对实时进程而言，它的权重为 counter 加 1000；对一般进程而言，权重为 counter。因此，实时进程总是会被认为是最值得运行的进程。如果当前进程的优先级和其他可运行进程一致，则因为当前进程至少已花费了一个时间片，因此，总处于劣势。如果许多进程的优先级一样，则调度程序选择运行队列中最靠前的进程，这实际就是“循环赛”调度。
4. 如果最值得运行的进程不是当前进程，就需要交换进程（或切换进程）。进程交换的作用是保存当前进程的运行上下文，同时恢复新进程的运行上下文。交换的具体细节和 CPU 类型有关，但需要注意的是，交换进程时调度程序运行在当前进程的上下文中，另外，调度程序还需要设置某些关键的 CPU 寄存器并刷新硬件高速缓存。

Linux 内核已具备在对称多处理系统（SMP）上运行的能力。在多处理器系统中，每个处理器都在忙碌地运行着进程。当运行在某个处理器上的进程耗尽其时间片，或者该进程处于等待状态时，该处理器将单独运行调度程序来选择新的进程。需要注意的是，每个处理器有一个自己的空闲进程，而每个处理器也有自己的当前进程。为了跟踪每个处理器的空闲进程和当前进程，进程的 task_struct 中包含了正在运行该进程的处理器编号（processor 字段），以及上次运行该进程的处理器编号（last_processor 字段）。显然，当一个进程再次运行时，可由不同的处理器运行，但在不同处理器上的进程交换所需开支稍微大一些，为此，每个进程有一个 processor_musk 字段，如果该字段的第 N 位为 1，则该进程可以运行在第 N 个进程上，利用这一字段，就可以将某个进程限制在单个处理器上运行。

11.3 进程的创建

系统启动时，启动程序运行在内核模式，这时，只有一个进程在系统中运行，即初始进程。系统初始化结束时，初始进程启动一个内核线程（即 **init**），而自己则处于空循环状态。当系统中没有可运行的进程时，调度程序将运行这一空闲进程。空闲进程的 task_struct 是唯一一个非动态分配的任务结构，该结构在内核编译时分配，称为 init_task。

init 内核线程/进程的标识号为 1，它是系统的第一个真正进程。它负责初始的系统设置工作，例如打开控制台，挂装文件系统等。然后，**init** 进程执行系统的初始化程

序，这一程序可能是 /etc/init、/bin/init 或 /sbin/init。**init** 程序将 /etc/inittab 当作脚本文件建立系统中新的进程，这些新的进程又可以建立新进程。例如，**getty** 进程可建立 **login** 进程来接受用户的登录请求。有关系统启动的详细内容可参见第十六章。

新的进程通过克隆旧的程序（当前程序）而建立。**fork** 和 **clone** 系统调用可用来建立新的进程。这两个系统调用结束时，内核在系统的物理内存中为新的进程分配新的 task_struct 结构，同时为新进程要使用的堆栈分配物理页。Linux 还会为新的进程分配新的进程标识符。然后，新 task_struct 结构的地址保存在 task 数组中，而旧进程的 task_struct 结构内容被复制到新进程的 task_struct 结构中。

在克隆进程时，Linux 允许两个进程共享相同的资源。可共享的资源包括文件、信号处理程序和虚拟内存等。当某个资源被共享时，该资源的引用计数值会增加 1，从而只有两个进程均终止时，内核才会释放这些资源。图 11-4 说明了父进程和子进程共享打开的文件。

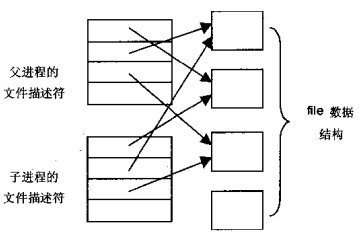


图 11-4 父进程和子进程共享打开的文件

系统对进程虚拟内存的克隆过程则更加巧妙些。新的 vm_area_struct 结构、新进程自己的 mm_struct 结构以及新进程的页表必须在一开始就准备好，但这时并不复制任何虚拟内存。如果旧进程的某些虚拟内存在物理内存中，而有些在交换文件中，那么虚拟内存的复制将会非常困难和费时。实际上，Linux 采用了称为“写时复制”的技术，也就是说，只有当两个进程中的任意一个向虚拟内存中写入数据时才复制相应的虚拟内存；而没有写入的任何内存页均可以在两个进程之间共享。代码页实际总是可以共享的。

为实现“写时复制”技术，Linux 将可写虚拟内存页的页表项标志为只读。当进程要向这种内存页写入数据时，处理器会发现内存访问控制上的问题（向只读页中写入），从而导致页故障。于是，操作系统可捕获这一被处理器认为是“非法的”写操作而完成内存页的复制。最后，Linux 还要修改两个进程的页表以及虚拟内存数据结构。

11.4 执行程序

和 UNIX 类似，Linux 中的程序和命令通常由命令解释器执行，这一命令解释器称为 shell。用户输入命令之后，shell 会在搜索路径（shell 变量 PATH 中包含搜索路径）指定的目录中搜索和输入命令匹配的映象名称。如果发现匹配的映象，shell 负责装载并执行该命令。shell 首先利用 **fork** 系统调用建立子进程，然后用找到的可执行映象文件覆盖子进程正在执行的 shell 二进制映象。

可执行文件可以是具有不同格式的二进制文件，也可以是一个文本的脚本文件。可执行映象文件中包含了可执行代码及数据，同时也包含操作系统用来将映象正确装入内存并执行的信息。Linux 使用的最常见的可执行文件格式是 ELF 和 a.out，但理论上讲，Linux 有足够的灵活性可以装入任何格式的可执行文件。

11.4.1 ELF

ELF 是“可执行可连接格式”的英文缩写，该格式由 UNIX 系统实验室制定。它是 Linux 中最经常使用的格式，和其他格式（例如 a.out 或 ECOFF 格式）比较起来，ELF 在装入内存时多一些系统开支，但是更为灵活。ELF 可执行文件包含了可执行代码和数据，通常也称为正文和数据。这种文件中包含一些表，根据这些表中的信息，内核可组织进程的虚拟内存。另外，文件中还包含有对内存布局的定义以及起始执行的指令位置。

我们分析如下简单程序在利用编译器编译并连接之后的 ELF 文件格式：

```
#include <stdio.h>

main ()
{
    printf ("Hello world!\n") ;
}
```

如图 11-5 所示，是上述源代码在编译连接后的 ELF 可执行文件的格式。从图 11-5 可以看出，ELF 可执行映象文件的开头是三个字符 'E'、'L' 和 'F'，作为这类文件的标识符。e_entry 定义了程序装入之后起始执行指令的虚拟地址。这个简单的 ELF 映象利用两个“物理头”结构分别定义代码和数据，e_phnum 是该文件中所包含的物理头信息个数，本例为 2。e_phyoff 是第一个物理头结构在文件中的偏移量，而 e_phentsize 则是物理头结构的大小，这两个偏移量均从文件头开始算起。根据上述两个信息，内核可正确读取两个物理头结构中的信息。

物理头结构的 p_flags 字段定义了对应代码或数据的访问属性。图中第一个 p_flags 字段的值为 FP_X 和 FP_R，表明该结构定义的是程序的代码；类似地，第二个物理头定义程序数据，并且是可读可写的。p_offset 定义对应的代码或数据在物理头之后的偏移量。p_vaddr 定义代码或数据的起始虚拟地址。p_filesz 和 p_memsz 分别定义代码或数据在文件中的大小以及在内存中的大小。对我们的简单例子，程序代码开始于两个物理头之后，而程序数据则开始于物理头之后的第 0x68533 字节处，显然，程序数据紧跟在程序代码之后。程序的代码大小为 0x68532，显得比较大，这是因为连接程序将 C 函数 **printf** 的代码连接到了 ELF 文件的原因。程序代码的文件大小和内存大小是一样的，而程序数据的文件大小和内存大小不一样，这是因为内存数据中，起始

的 2200 字节是预先初始化的数据，初始化值来自 ELF 映象，而其后的 2048 字节则由执行代码初始化。

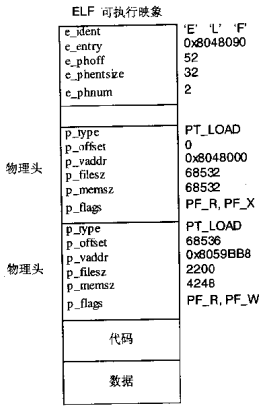


图 11-5 一个简单的 ELF 可执行文件的布局

如上一章中所描述的，Linux 利用需求分页技术装入程序映象。当 shell 进程利用 **fork** 系统调用建立了子进程之后，子进程会调用 **exec** 系统调用（实际有多种 **exec** 调用），**exec** 系统调用将利用 ELF 二进制格式装载器装载 ELF 映象，当装载器检验映象是有效的 ELF 文件之后，就会将当前进程（实际就是父进程或旧进程）的可执行映象从虚拟内存中清除，同时清除任何信号处理程序并关闭所有打开的文件（把相应 `file` 结构中的 `f_count` 引用计数减 1，如果这一计数为 0，内核负责释放这一文件对象），然后重置进程页表。完成上述过程之后，只需根据 ELF 文件中的信息将映象代码和数据的起始和终止地址分配并设置相应的虚拟地址区域，修改进程页表。这时，当前进程就可以开始执行对应的 ELF 映象中的指令了。

和静态连接库不同，动态连接库可在运行时连接到进程虚拟地址中。对于使用同一动态连接库的多个进程，只需在内存中保留一份共享库信息即可，这样就节省了内存空间。当共享库需要在运行时连接到进程虚拟地址时，Linux 的动态连接器利用 ELF 共享库中的符号表完成连接工作，符号表中定义了 ELF 映象引用的全部动态库例程。Linux 的动

态连接器一般包含在 /lib 目录中，通常为 ld.so.1、libc.so.1 和 ld-linux.so.1。

11.4.2 脚本文件

脚本文件实际是一些可执行的命令，这些命令一般由指定的解释器解释并执行。Linux 中常见的解释器有 **wish**、**perl** 以及命令 shell，如 **bash** 等。

一般来说，脚本文件的第一行用来指定脚本的解释程序，例如：

```
#!/usr/bin/wish
```

这行内容指定由 **wish** 作为该脚本的命令解释器。脚本的二进制装载器利用这一信息搜索解释器，如果能够找到指定的解释器，该装载器就和上述执行 ELF 程序的装载过程一样装载并执行解释器。脚本文件名成为传递给解释器的第一个命令参数，而最初的第一个参数则成为现在的第二个参数，依此类推。为解释器传递了正确的命令参数后，就可由脚本解释器执行脚本。

11.5 信号

信号是 UNIX 系统中最古老的进程间通讯机制之一，它主要用来向进程发送异步的事件信号。键盘中断可能产生信号，而浮点运算溢出或者内存访问错误等也可产生信号。shell 通常利用信号向子进程发送作业控制命令。

在 Linux 中，信号种类的数目和具体的平台有关，因为内核用一个字代表所有的信号，因此字的位数就是信号种类的最多数目。对 32 位的 i386 平台而言，一个字为 32 位，因此信号有 32 种，而对 64 位的 Alpha AXP 平台而言，每个字为 64 位，因此信号最多可有 64 种。Linux 内核定义的最常见的信号、C 语言宏名及其用途如表 11-5 所示：

表 11-5 常见信号及其用途

值	C 语言宏名	用途
1	SIGHUP	从终端上发出的结束信号
2	SIGINT	来自键盘的中断信号（Ctrl-C）
3	SIGQUIT	来自键盘的退出信号（Ctrl-\）
8	SIGFPE	浮点异常信号（例如浮点运算溢出）
9	SIGKILL	该信号结束接收信号的进程
14	SIGALRM	进程的定时器到期时，发送该信号
15	SIGTERM	kill 命令发出的信号
17	SIGCHLD	标识子进程停止或结束的信号
19	SIGSTOP	来自键盘（Ctrl-Z）或调试程序的停止执行信号

进程可以选择对某种信号所采取的特定操作，这些操作包括：

- 忽略信号。进程可忽略产生的信号，但 SIGKILL 和 SIGSTOP 信号不能被忽略。

- 阻塞信号。进程可选择阻塞某些信号。
- 由进程处理该信号。进程本身可在系统中注册处理信号的处理程序地址，当发出该信号时，由注册的处理程序处理信号。
- 由内核进行默认处理。信号由内核的默认处理程序处理。大多数情况下，信号由内核处理。

需要注意的是，Linux 内核中不存在任何机制用来区分不同信号的优先级。也就是说，当同时有多个信号发出时，进程可能会以任意顺序接收到信号并进行处理。另外，如果进程在处理某个信号之前，又有相同的信号发出，则进程只能接收到一个信号。产生上述现象的原因与内核对信号的实现有关，将在下面解释。

系统在 task_struct 结构中利用两个字分别记录当前挂起的信号（signal）以及当前阻塞的信号（blocked）。挂起的信号指尚未进行处理的信号。阻塞的信号指进程当前不处理的信号，如果产生了某个当前被阻塞的信号，则该信号会一直保持挂起，直到该信号不再被阻塞为止。除了 SIGKILL 和 SIGSTOP 信号外，所有的信号均可以被阻塞，信号的阻塞可通过系统调用实现。每个进程的 task_struct 结构中还包含了一个指向 sigaction 结构数组的指针，该结构数组中的信息实际指定了进程处理所有信号的方式。如果某个 sigaction 结构中包含有处理信号的例程地址，则由该处理例程处理该信号；反之，则根据结构中的一个标志或者由内核进行默认处理，或者只是忽略该信号。通过系统调用，进程可以修改 sigaction 结构数组的信息，从而指定进程处理信号的方式。

进程不能向系统中所有的进程发送信号，一般而言，除系统和超级用户外，普通进程只能向具有相同 uid 和 gid 的进程，或者处于同一进程组的进程发送信号。产生信号时，内核将进程 task_struct 的 signal 字中的相应位设置为 1，从而表明产生了该信号。系统不对置位之前该位已经为 1 的情况进行处理，因而进程无法接收到前一次信号。如果进程当前没有阻塞该信号，并且进程正处于可中断的等待状态，则内核将该进程的状态改变为运行，并放置在运行队列中。这样，调度程序在进行调度时，就有可能选择该进程运行，从而可以让进程处理该信号。

发送给某个进程的信号并不会立即得到处理，相反，只有该进程再次运行时，才有机会处理该信号。每次进程从系统调用中退出时，内核会检查它的 signal 和 block 字段，如果有任何一个未被阻塞的信号发出，内核就根据 sigaction 结构数组中的信息进行处理。处理过程如下：

1. 检查对应的 sigaction 结构，如果该信号不是 SIGKILL 或 SIGSTOP 信号，且被忽略，则不处理该信号。
2. 如果该信号利用默认的处理程序处理，则由内核处理该信号，否则转向第 3 步。
3. 该信号由进程自己的处理程序处理，内核将修改当前进程的调用堆栈帧，并将进程的程序计数寄存器修改为信号处理程序的入口地址。此后，指令将跳转到信号处理程序，当从信号处理程序中返回时，实际就返回了进程的用户模式部分。

Linux 是 POSIX 兼容的，因此，进程在处理某个信号时，还可以修改进程的 blocked 掩码。但是，当信号处理程序返回时，blocked 值必须恢复为原有的掩码值，这一任务由内核完成。Linux 在进程的调用堆栈帧中添加了对清理程序的调用，该清理程序可以恢复原有的 blocked 掩码值。当内核在处理信号时，可能同时有多个信号需要由用户处

埋程序处理，这时，Linux 内核可以将所有的信号处理程序地址推入堆栈帧，而当所有的信号处理完毕后，调用清理程序恢复原先的 blocked 值。

11.6 管道

管道是 Linux 中最常用的 IPC 机制。利用管道时，一个进程的输出可成为另外一个进程的输入。当输入输出的数据量特别大时，这种 IPC 机制非常有用。可以想象，如果没有管道机制，而必须利用文件传递大量数据时，会造成许多空间和时间上的浪费。

在 Linux 中，通过将两个 file 结构指向同一个临时的 VFS 索引节点，而两个 VFS 索引节点又指向同一个物理页而实现管道。如图 11-6 所示。

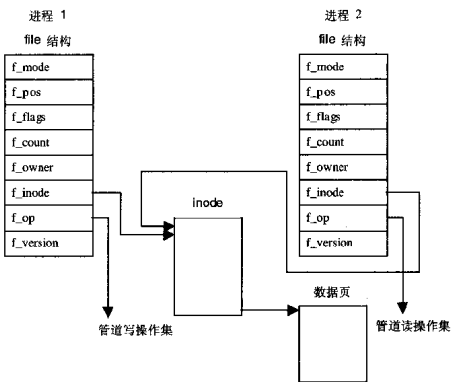


图 11-6　管道示意图

图 11-6 中，每个 file 数据结构定义不同的文件操作例程地址，其中一个用来向管道中写入数据，而另外一个用来从管道中读出数据。这样，用户程序的系统调用仍然是通常的文件操作，而内核却利用这种抽象机制实现了管道这一特殊操作。管道写函数通过将字节复制到 VFS 索引节点指向的物理内存而写入数据，而管道读函数则通过复制物理内存中的字节而读出数据。当然，内核必须利用一定的机制同步对管道的访问，为此，内核使用了锁、等待队列和信号。

当写进程向管道中写入时，它利用标准的库函数，系统根据库函数传递的文件描述符，可找到该文件的 file 结构。file 结构中指定了用来进行写操作的函数（即写入函数）地址，于是，内核调用该函数完成写操作。写入函数在向内存中写入数据之前，必须首先检查 VFS 索引节点中的信息，同时满足如下条件时，才能进行实际的内存复制工作：

- 内存中有足够的空间可容纳所有要写入的数据；
- 内存没有被读程序锁定。

如果同时满足上述条件，写入函数首先锁定内存，然后从写进程的地址空间中复制数据到内存。否则，写入进程就休眠在 VFS 索引节点的等待队列中，接下来，内核将调用调度程序，而调度程序会选择其他进程运行。写入进程实际处于可中断的等待状态，当内存中有足够的空间可以容纳写入数据，或内存被解锁时，读取进程会唤醒写入进程，这时，写入进程将接收到信号。当数据写入内存之后，内存被解锁，而所有休眠在索引节点的读取进程会被唤醒。

管道的读取过程和写入过程类似。但是，进程可以在没有数据或内存被锁定时立即返回错误信息，而不是阻塞该进程，这依赖于文件或管道的打开模式。反之，进程可以休眠在索引节点的等待队列中等待写入进程写入数据。当所有的进程完成了管道操作之后，管道的索引节点被丢弃，而共享数据页也被释放。

Linux 还支持另外一种管道形式，称为命名管道，或 FIFO，这是因为这种管道的操作方式基于“先进先出”原理。上面讲述的管道类型也被称为“匿名管道”。命名管道中，首先写入管道的数据是首先被读出的数据。匿名管道是临时对象，而 FIFO 则是文件系统的真正实体，用 **mkfifo** 命令可建立管道。如果进程有足够的权限就可以使用 FIFO。FIFO 和匿名管道的数据结构以及操作极其类似，二者的主要区别在于，FIFO 在使用之前就已经存在，用户可打开或关闭 FIFO；而匿名管道在只在操作时存在，因而是临时对象。

11.7 System V 的 IPC 机制

为了和其他系统保持兼容，Linux 也提供三种首先出现在 UNIX System V 中的 IPC 机制。这三种机制分别是：消息队列、信号量以及共享内存。System V IPC 机制主要有如下特点：

- 如果进程要访问 System V IPC 对象，则需要在系统调用中传递唯一的引用标识符。
- 对 System V IPC 对象的访问，必须经过类似文件访问的许可检验。对这些对象访问权限的设置由对象的创建者利用系统调用设置。
- 对象的引用标识符由 IPC 机制作为访问对象表的索引，但需要一些操作来生成索引。

在 Linux 中，所有表示 System V IPC 对象的数据结构中都包含一个 ipc_perm 结构，该结构中包含了作为对象所有者和创建者的进程之用户标识符和组标识符，以及对象的访问模式和对象的访问键。访问键用来定位 System V IPC 对象的引用标识符。系统支持两种访问键：公有和私有。如果键是公有的，则系统中所有的进程通过权限检查后，均可以找到 System V IPC 对象的引用标识符。但是，只能通过引用标识符引用 System V IPC

对象。

Linux 对这些 IPC 机制的实施大同小异，我们在这里只主要介绍其中两种：消息队列和信号量。

11.7.1　消息队列

一个或多个进程可向消息队列写入消息，而一个或多个进程可从消息队列中读取消息，这种进程间通讯机制通常使用在客户/服务器模型中，客户向服务器发送请求消息，服务器读取消息并执行相应请求。在许多微内核结构的操作系统中，内核和各组件之间的基本通讯方式就是消息队列。例如，在 MINIX 操作系统中，内核、I/O 任务、服务器进程和用户进程之间就是通过消息队列实现通讯的。

Linux 为系统中所有的消息队列维护一个 msgque 链表，该链表中的每个指针指向一个 msgid_ds 结构，该结构完整描述一个消息队列。当建立一个消息队列时，系统从内存中分配一个 msgid_ds 结构并将指针添加到 msgque 链表。

图 11-7 是 msgid_ds 结构的示意图。从图中可以看出，每个 msgid_ds 结构都包含一个 ipc_perm 结构以及指向该队列所包含的消息指针，显然，队列中的消息构成了一个链表。另外，Linux 还在 msgid_ds 结构中包含一些有关修改时间之类的信息，同时包含两个等待队列，分别用于队列的写入进程和队列的读取进程。

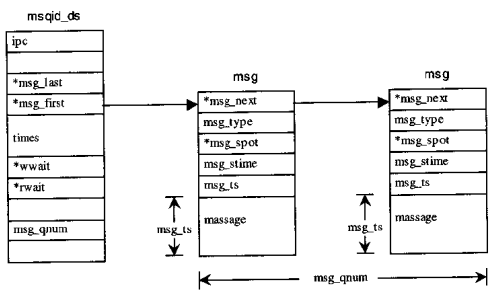


图 11-7　System V IPC 机制——消息队列

消息队列的写入操作和读取操作是类似的，以消息的写入为例，步骤如下：

1. 当某个进程要写入消息时，该进程的有效 uid 和 gid 首先要和 ipc_perm 中的访问模式进行比较。如果进程不能写入，系统调用返回错误，写操作结束。

2. 如果该进程可以向消息队列写入，则消息可以复制到消息队列的末尾。在进行复制之前，必须判断消息队列当前是否已满。消息的具体内容和应用程序有关，由参与通讯的进程约定。
3. 如果消息队列中当前没有空间容纳消息，则写入进程被添加到该消息队列的写等待队列，否则，内核分配一个 msg 结构，将消息从进程的地址空间中复制到 msg 结构，然后将 msg 添加到队列末尾，这时，系统调用成功返回，写操作结束。
4. 调用调度程序，调度程序选择其他进程运行，写操作结束。

如果有某个进程从消息队列中读取了消息，则系统会唤醒写等待队列中的进程。

读取操作和写入操作类似，但进程在没有消息或没有指定类型的消息时进入等待状态。

11.7.2 信号量

信号量的概念由 E. W. Dijkstra 于 1965 年首次提出。信号量实际是一个整数，进程在信号量上的操作分两种，一种称为 DOWN，而另外一种称为 UP。DOWN 操作的结果是让信号量的值减 1，UP 操作的结果是让信号量的值加 1。在进行实际的操作之前，进程首先检查信号量的当前值，如果当前值大于 0，则可以执行 DOWN 操作，否则进程休眠，等待其他进程在该信号量上的 UP 操作，因为其他进程的 UP 操作将让信号量的值增加，从而它的 DOWN 操作可以成功完成。某信号灯在经过某个进程的成功操作之后，其他休眠在该信号量上的进程就有可能成功完成自己的操作，这时，系统负责检查休眠进程是否可以完成自己的操作。

为了理解信号量，我们想象某机票定购系统。最初旅客在定票时，一般有足够的票数可以满足定票量。当剩余的机票数为 1，而某个旅客现在需要定两张票时，就无法满足该顾客的需求，这时售票小姐让这个旅客留下他的电话号码，如果其他人退票，就可以优先让这个旅客定票。如果最终有人退票，则售票小姐打电话通知上述要定两张票的旅客，这时，该旅客就能够定到自己的票。

我们可以将旅客看成是进程，而定票可看成是信号量上的 DOWN 操作，退票可看成是信号量上的 UP 操作，而信号量的初始值为机票总数，售票小姐则相当于操作系统的信号量管理器，由她（操作系统）决定旅客（进程）能不能完成操作，并且在新的条件成熟时，负责通知（唤醒）登记的（休眠的）旅客（进程）。

在操作系统中，信号量的最简单形式是一个整数，多个进程可检查并设置信号量的值。这种检查并设置操作是不可被中断的，也称为“原子”操作。检查并设置操作的结果是信号量的当前值和设置值相加的结果，该设置值可以是正值，也可以是负值。根据检查和设置操作的结果，进行操作的进程可能会进入休眠状态，而当其他进程完成自己的检查并设置操作后，由系统检查前一个休眠进程是否可以在新信号量值的条件下完成相应的检查和设置操作。这样，通过信号量，就可以协调多个进程的操作。

信号量可用来实现所谓的“关键段”。关键段指同一时刻只能有一个进程执行其中代码的代码段。也可用信号量解决经典的“生产者——消费者”问题，“生产者——消费者”问题和上述的定票问题类似。这一问题可以描述如下：

两个进程共享一个公共的、匡定大小的缓冲区。其中的一个进程，即生产者，向缓冲区放入信息，另外一个进程，即消费者，从缓冲区中取走信息（该问题也可以一般化为 m

个生产者和 n 个消费者）。当生产者向缓冲区放入信息时，如果缓冲区是满的，则生产者进入休眠，而当消费者从缓冲区中拿走信息后，可唤醒生产者；当消费者从缓冲区中取信息时，如果缓冲区为空，则消费者进入休眠，而当生产者向缓冲区写入信息后，可唤醒消费者。

Linux 利用 semid_ds 结构来表示 System V IPC 信号量，见图 11-8。和消息队列类似，系统中所有的信号量组成了一个 semary 链表，该链表的每个节点指向一个 semid_ds 结构。从图 11-8 可以看出，semid_ds 结构的 sem_base 指向一个信号量数组，允许操作这些信号量数组的进程可以利用系统调用执行操作。系统调用可指定多个操作，每个操作由三个参数指定：信号量索引、操作值和操作标志。信号量索引用来定位信号量数组中的信号量；操作值是要和信号量的当前值相加的数值。首先，Linux 按如下的规则判断是否所有的操作都可以成功：操作值和信号量的当前值相加大于 0，或操作值和当前值均为 0，则操作成功。如果系统调用中指定的所有操作中有一个操作不能成功时，则 Linux 会挂起这一进程。但是，如果操作标志指定这种情况下不能挂起进程的话，系统调用返回并指明信号量上的操作没有成功，而进程可以继续执行。如果进程被挂起，Linux 必须保存信号量的操作状态并将当前进程放入等待队列。为此，Linux 在堆栈中建立一个 sem_queue 结构并填充该结构。新的 sem_queue 结构添加到信号量对象的等待队列中（利用 sem_pending 和 sem_pending_last 指针）。当前进程放入 sem_queue 结构的等待队列中（sleeper）后调用调度程序选择其他的进程运行。

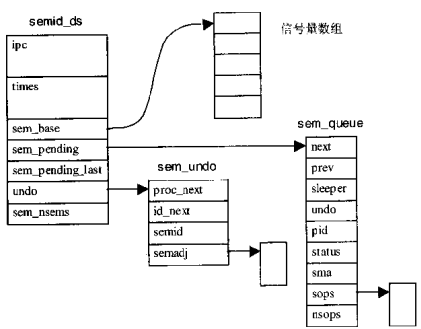


图 11-8　System V IPC 机制——信号量

如果所有的信号量操作都成功了，当前进程可继续运行。在此之前，Linux 负责将操

作实际应用于信号量队列的相应元素。这时，Linux 检查任何等待的或挂起的进程，看它们的信号量操作是否可以成功。如果这些进程的信号量操作可以成功，Linux 就会将它们从挂起队列中移去，并将它们的操作实际应用于信号量队列。同时，Linux 会唤醒休眠进程，以便可在下次调度程序运行时可以运行这些进程。当新的信号量操作应用于信号量队列之后，Linux 会接着检查挂起队列，直到没有操作可成功，或没有挂起进程为止。

和信号量操作相关的概念还有“死锁”。当某个进程修改了信号量而进入关键段之后，却因为崩溃而没有退出关键段，这时，其他被挂起在信号量上的进程永远得不到运行机会，这就是所谓的死锁。Linux 通过维护一个信号量数组的调整链表来避免这一问题。

11.7.3 共享内存

在第十章中看到，进程的虚拟地址可以映射到任意一处物理地址，这样，如果两个进程的虚拟地址映射到同一物理地址，这两个进程就可以利用这一虚拟地址进行通讯。但是，一旦内存被共享之后，对共享内存的访问同步需要由其他 IPC 机制，例如信号量来实现。Linux 中的共享内存通过访问键来访问，并进行访问权限的检查。共享内存对象的创建者负责控制访问权限以及访问键的公有或私有特性。如果具有足够的权限，也可以将共享内存锁定到物理内存中。

图 11-9 是 Linux 中共享内存对象的结构。和消息队列及信号量类似，Linux 中也有一个链表维护着所有的共享内存对象。

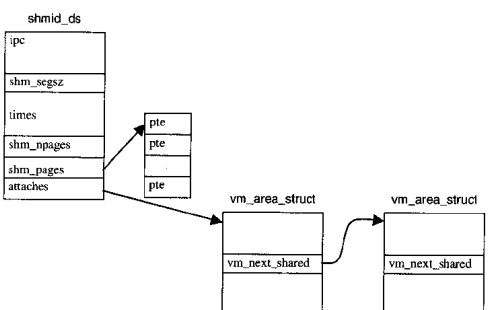


图 11-9 System V IPC 机制——共享内存

参照图 11-9， 共享内存对象的结构元素说明如下：

- shm_segsz：共享内存的大小；
- times：使用共享内存的进程数目；
- attaches：描述被共享的物理内存映射到各进程的虚拟内存区域。
- shm_npages：共享虚拟内存页的数目；
- shm_pages：指向共享虚拟内存页的页表项表。

在利用共享内存时，参与通讯的进程通过系统调用将自己要共享的虚拟地址区域附加到 attaches 指向的链表中。

某个进程第一次访问共享虚拟内存时将产生页故障。这时，Linux 找出描述该内存的 vm_area_struct 结构，该结构中包含用来处理这种共享虚拟内存的处理函数地址。共享内存页故障处理代码在 shmid_ds 的页表项链表中查找，以便查看是否存在该共享虚拟内存的页表项。如果没有，系统将分配一个物理页并建立页表项。该页表项加入 shmid_ds 结构的同时也添加到进程的页表中。此后，当另一个进程访问该共享内存时，共享内存页故障处理代码将使用同一物理页，而只是将页表项添加到这一进程的页表中。这样，前后两个进程就可以通过同一物理页进行通讯。

当某个进程不再共享其虚拟内存时，利用系统调用将自己的虚拟地址区域从该链表中移去，并更新进程页表。当最后一个进程释放了自己的虚拟地址空间之后，系统释放所分配的物理页。

当共享的虚拟内存没有被锁定到物理内存时，共享内存也可能会被交换到交换空间中。

11.8 套接字

套接字和上述的 IPC 机制有所不同，它能够实现不同计算机之间的进程间通讯，关于套接字的讨论在第十四章中进行。

11.9 相关系统工具及系统调用

11.9.1 系统工具

Linux 中主要有三个命令可以查看当前系统中运行的进程。**ps** 命令可报告进程状态；**pstree** 可打印进程之间的父子关系；**top** 则可用来监视系统中 CPU 利用率最高的进程，也可以交互式地操作进程。

kill 命令则用来向指定进程发送信号，如果没有指定要发送的信号，则发送 SIGTERM 信号，该信号的默认处理是终止进程的运行。如果查看系统所支持的所有信号编号，可用 **kill -l** 命令获取信号清单。

mkfifo 可用来建立命名管道（FIFO）。

11.9.2 系统调用

表 11-7 简要列出了和进程及进程间通讯相关的系统调用。标志列中各字母的意义可

参见 表 10-1 的说明。

表 11-7 相关系统调用

系统调用	说明	标志
alarm	在指定时间之后发送 SIGALRM 信号	m+c
clone	创建子进程	m-
execl, execlp, execle, ...	执行映象	m+!c
execve	执行映象	m+c
exit	终止进程	m+c
fork	创建了进程	m+c
fsync	将文件高速缓存写入磁盘	mc
ftime	获取自 1970.1.1 以来的时区.秒数	m!c
getegid	获取有效组标识符	m+c
geteuid	获取有效用户标识符	m+c
getgid	获取实际组标识符	m+c
getitimter	获取间隔定时器的值	mc
getpgid	获取某进程之父进程的组标识符	+c
getpgrp	获取当前进程之父进程的组标识符	m+c
getpid	获取当前进程的进程标识符	m+c
getppid	获取父进程的进程标识符	m+c
getpriority	获取进程/组/用户的优先级	mc
gettimeofday	获取自 :970.1.1 以来的时区.秒数	mc
getuid	获取实际用户标识符	m+c
ipc	进程间通讯	-c
kill	向进程发送信号	m+c
killpg	向进程组发送信号	m!c
modify_ldt	读取或写入局部描述符表	-
msgctl	消息队列控制	m!c
msgget	获取消息队列标识符	m!c
msgrcv	接收消息	m!c
msgsnd	发送消息	m!c
nice	修改进程优先级	mc
pause	进程进入休眠，等待信号	m+c
pipe	创建管道	m+c
semctl	信号量控制	m!c
semget	获取某信号量数组的标识符	m!c
semop	在信号量数组成员上的操作	m!c
setgid	设置实际组标识符	m+c
setitimer	设置间隔定时器	mc
setpgid	设置进程组标识符	m+c
setpgrp	以调用进程作为领头进程创建新的进程组	m+c
setpriority	设置进程/组/用户优先级	mc
setsid	建立一个新会话	m+c
setregid	设置实际和有效组标识符	mc
setreuid	设置实际和有效用户标识符	mc
settimeofday	设置自 1970.1.1 以来的时区.秒数	mc
setuid	设置实际用户标识符	m+c

shmat	附加共享内存	m!c
shmctl	共享内存控制	m!c
shmdt	移去共享内存	m!c
shmget	获取/建立共享内存	m!c
sigaction	设置/获取信号处理器	m+c
sigblock	阻塞信号	m!c
siggetmask	获取当前进程的信号阻塞掩码	!c
signal	设置信号处理器	mc
sigpause	在处理下次信号之前，使用新的信号阻塞掩码	mc
sigpending	获取挂起且阻塞的信号	m+c
sigprocmask	设置/获取当前进程的信号阻塞掩码	-c
sigsetmask	设置当前进程的信号阻塞掩码	c!
sigsuspend	替换 sigpause	m+c
sigvec	见 sigaction	m
ssetmask	见 sigsetmask	m
system	执行 shell 命令	m!c
time	获取自 1970.1.1 以来的秒数	m+c
times	获取进程的 CPU 时间	m+c
vfork	见 fork	m!c
wait	等待进程终止	m+c
wait3, wait4	等待指定进程终止 （BSD）	mc
waitpid	等待指定进程终止	m+c
vm86	进入虚拟 8086 模式	m-c

第十二章

硬件和设备驱动程序

Linux 最初在 i386 平台上开发，i386 平台也是其主要支持的计算机硬件平台，但是，Linux 也同时支持其他类型的 CPU，例如 Alpha AXP、PowerPC 以及 MIPS R4600 等 CPU。

计算机中，CPU 是核心组件，但是完整的计算机还包括其他一些组件，这些组件共同协调完成用户任务。总线为不同类型的组件提供通讯途径。到目前为止，出现了许多总线类型，例如 ISA、PCI 等。本章主要介绍 Linux 对 PCI 总线的支持。

对计算机的输入输出设备进行管理和控制是操作系统的主要功能之一。操作系统必须向设备提供操作指令，还需要处理来自设备的中断请求。一般来说，操作系统还要为程序提供设备的访问接口。Linux 通过设备驱动程序为应用程序提供了统一抽象的接口，从而隐藏了大量不同设备之间的区别和细节。本章介绍 Linux 对硬件中断的处理以及对设备驱动程序的支持。

在 Linux 的安装和系统维护中，会遇到许多与特定硬件或设备相关的问题。这些问题的正确解决建立在对相关概念的正确理解之上。本章讲述 Linux 系统中常见的设备以及一些相关概念。

12.1 处理器和总线

Linux 主要为 i386 系列处理器开发。Linux 使用了这种处理器的“保护模式”，在这种模式下，处理器可以支持许多高级特性，从而使多任务和虚拟内存的实现成为可能。众所周知，Intel 的 80x86 系列处理器只有 386 以上的处理器才支持保护模式，因此，Linux 不能运行在以 8088 为处理器的 PC 或 PC/XT 机上。因为 AMD 和 Cyrix 生产的一些芯片和 80386 是兼容的，因此，Linux 也可以运行在 AMD K6 等作为处理器的 PC 机上。

Linux 内核主要由 C 语言编写，因此，将 Linux 移植到其他平台上是比较容易的，而且，Linux 内核源代码的组织也使移植过程更加容易。到目前为止，Linux 所支持的其他类型的 CPU 有：Alpha AXP、PowerPC 以及 MIPS R4600 等。

总线是将计算机中不同类型的硬件组织在一起，并为它们提供通讯保证的计算机关键硬件。总线定义了硬件之间进行通讯的“协议”，遵循同一种协议的硬件可在同一条总线上协调工作。从物理上看，总线由计算机主板上传送信号的线路以及附属的控制芯片组成。

各种设备大多以控制卡的形式插在总线槽上。总线协议包括一些物理上的约定，例如电气特性以及控制卡尺寸等，另外，总线协议也定义了总线上的信号时序以及总线的最大数据传输速率（以 MHz 为单位）等特性。不同的协议形成了不同的总线。在流行 PC 机中，常见的总线类型如表 12-1 所示。

表 12-1 PC 中的常见总线类型

名称	说明
ISA 总线	ISA 是“工业标准结构”的英文缩写。ISA 总线是当前应用最为广泛的总线类型。最初用于 IBM 的 PC AT 机中，一次可传输 16 位数据，最大的数据传输率为 5MB/sec。
VESA 局部总线	VESA 是“视频电子标准协会”的英文缩写。VESA 局部总线，也即 VLB，可提供处理器和视频卡之间的高速数据传输，典型的 VLB 传输速率为 30MB/sec。
EISA 总线	EISA 是“扩展工业标准结构”的英文缩写。EISA 总线的传输速率为 30MB/sec。但该总线很少使用。
PCI 总线	PCI 是“外设组件互连接”的英文缩写。PCI 是最新的高性能总线。它的时钟频率为 33 MHz，一次可传输 64 位数据。如果 PCI 总线用来传输 32 位数据，可获得 33 * 4 = 132 MB/sec 的传输速率。

Linux 支持上述四种总线类型，但尚不支持 MCA 总线（微通道结构总线）。

PCI 总线现在广泛使用在基于 Pentium 的计算机中，一方面是因为该总线的数据吞吐量大，另一方面是因为该总线和具体的处理器无关，也就是说，同一个 PCI 设备，既可以使用在 i386 计算机中，也可以使用在 Alpha AXP 计算机中。PCI 总线的设计也使各种 PCI 外设卡可直接插入 PCI 总线槽中，而不需要考虑各种额外的特殊逻辑，例如，ISA 设备中的跳线设置。但是，正因为 PCI 总线的灵活性，所以需要操作系统额外的软件支持。

12.2 Linux 对 PCI 总线的支持

12.2.1 PCI 总线的结构

如图 12-1 所示，是一个简单的 PCI 系统的逻辑示意图。每条 PCI 总线上的设备数目是有一定限制的，因此该系统使用 PCI-PCI 桥将不同的 PCI 总线粘合在一起。不同的 PCI 总线有唯一的编号，CPU 处于 PCI 0 号总线上。为了和老的 ISA 设备兼容，利用 PCI-ISA 桥可在 PCI 总线上连接 ISA 总线。利用 PCI-PCI 桥和 PCI-ISA 桥，可以突破单个 PCI 总线的限制而连接许多设备。一般而言，服务器和桌面计算机的 PCI 总线拓扑结构是不同的。

我们知道，ISA 设备有两种地址空间，一种是 I/O 端口空间，另一种是存储器空间。和 ISA 设备相比较，PCI 设备有三种地址空间，分别是：I/O 端口空间、存储器空间和配置空间。不同的 CPU 类型对地址空间类型的支持不同，i386 处理器的 I/O 端口和存储器空间是分开的，利用不同的处理器指令操作这些地址；Alpha AXP 处理器却只有存储器空间，也就是说，端口的输入输出和内存访问的指令是一样的。因此，Alpha AXP 处

理器利用一种独特的内存映射方式将部分虚拟地址空间映射为 PCI 的地址空间。

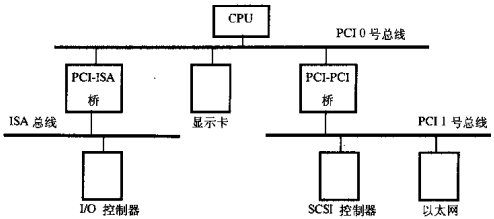


图 12-1 简单的 PCI 系统拓扑图

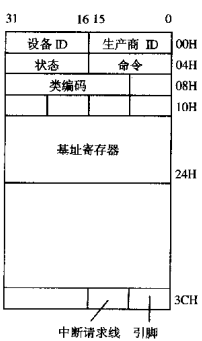


图 12-2 PCI 设备的设备头结构

PCI 的前两种地址空间和 ISA 的两种地址空间类似，设备驱动程序利用这些地址空

间在系统和设备之间通讯。除此之外，PCI 设备，包括 PCI-PCI 桥和 PCI-ISA 桥都包含称为“配置头”的数据结构，这些数据结构包含在 PCI 设备的配置空间中。配置头在配置空间中的位置和相应的 PCI 插槽相关，系统可以利用这些配置头判断相应的 PCI 插槽中是否存在 PCI 设备。

图 12-2 是设备头的结构示意图。一般而言，设备头的长度为 256 字节，利用与特定硬件相关的代码可读取这些设备头。根据其中的信息，操作系统可为每个已插入 PCI 插槽的 PCI 设备分配 PCI I/O 端口的数量、地址，以及 PCI 存储器的长度和起始位置（由基地址寄存器定义），也可以配置 PCI 设备的中断以及中断请求线。但是只有专门为特定硬件系统设计的配置代码才可以读取和设置这些信息，在 i386 系统中，这种代码通常由系统 BIOS 提供。详细信息可参见有关的 PCI 总线规范。

12.2.2 Linux 中 PCI 设备的初始化

Linux 中的 PCI 设备初始代码可划分为如下三个部分：PCI BIOS、PCI 修正和 PCI 设备驱动程序。PCI BIOS 由一组标准的 PCI 设备访问功能函数组成，这些代码对不同的平台来说是一样的。PCI 修正部分为非 Intel 的系统所特有。对基于 Intel 的系统来说，系统引导时，可由系统 BIOS 完整配置 PCI 系统，而对 Alpha AXP 系统而言，需要利用 PCI 修正代码来完成 PCI 系统的配置。PCI 设备驱动程序实际并不是一个真正的设备驱动程序，它只在引导时由操作系统调用进行 PCI 设备的初始化。下面主要描述 PCI 设备驱动程序。

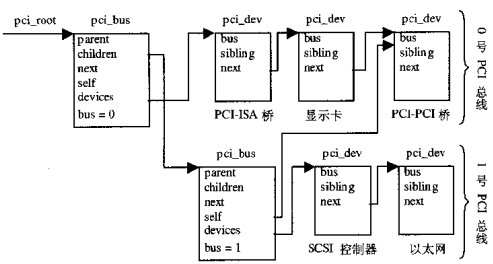


图 12-3 Linux 内核中的 PCI 数据结构

PCI 设备驱动程序利用 PCI BIOS 的功能函数在系统中扫描所有的 PCI 设备（包括两种桥设备），并建立如图 12-3 所示的数据结构，该数据结构实际是图 12-1 所示系统 PCI 拓扑结构的一个写照。

图 12-3 中的数据结构对应于图 12-1 所示的 PCI 系统。PCI 初始化代码首先扫描 PCI 0 号总线，为每个可能的 PCI 插槽读取设备头中的制造商标识和设备标识。如果发现一个已填充有效标识信息的设备头，则说明对应的 PCI 插槽中有合法的 PCI 设备，因此，系统为该设备建立一个 pci_dev 结构。同一总线上所有的 pci_dev 数据结构形成了一个 pci_devices 链表。

如果在搜索 0 号总线时发现 PCI-PCI 桥设备，则系统会建立一个 pci_bus 结构，并和代表 PCI 0 号总线的 pci_bus 结构以及 pci_dev 结构一起形成一个由 pci_root 所指的树形结构。接下来，初始化代码将继续在次总线上扫描其他 PCI 设备，直到扫描完所有的 PCI 设备为止。如果在次总线上发现有 PCI-PCI 桥设备，则初始化代码会继续扫描从属总线。

12.3 计算机和设备间的数据交换方式

12.3.1 查询和中断

如前所述，不管是 ISA 设备还是 PCI 设备，设备驱动程序通过设备的 I/O 端口空间以及存储器空间完成数据的交换。例如，网卡一般将自己的内部寄存器映射为设备的 I/O 端口，而显示卡则利用大量的存储器空间作为视频信息的存储空间。利用这些地址空间，驱动程序可以向外设发送指定的操作命令，例如要求硬盘控制器将磁头移动到指定的柱面。通常来讲，外设的操作耗时较长，因此，当 CPU 实际执行了命令指令之后，驱动程序可采用两种方式等待外设完成操作。一种是查询方式，驱动程序在提交命令之后，就开始查询设备的状态寄存器，当状态寄存器表明操作完成时，驱动程序可继续后续处理。另一种方式是利用中断，驱动程序提交命令之后，立即进入休眠状态（即放弃 CPU 的使用权，而其他进程就有机会运行了），设备结束操作之后，会产生中断信号，操作系统则根据设备的中断信号负责唤醒驱动程序。显然，查询方式白白浪费了大量的 CPU 时间，而中断方式才是多任务操作系统中最有效利用 CPU 的方式。

12.3.2 直接内存访问

利用中断，系统和设备之间可以通过设备驱动程序传送数据，但是，当传送数据量很大时，因为中断处理上的延迟，利用中断的方式就不太有效了。例如，SCSI 硬盘可在 1 秒之内传输 50 MB 的数据，而通常的中断延迟（中断产生到相应的设备驱动程序被调用之间的时间）大约为 2 毫秒，显然，利用中断对整体数据传输速度的影响是非常大的。

利用“直接内存访问（DMA）”可解决这一问题。DMA 可允许设备和系统内存之间在不经处理器参与之下传输大量数据。在 PC 中，有 8 个 DMA 通道，设备驱动程序可使用其中的 7 个。每个 DMA 通道有一个 16 位的地址寄存器和一个 16 位的计数寄存器，可分别用来定义一次 DMA 操作的系统内存起始地址和内存大小。设备驱动程序在利用 DMA 之前，需要选择 DMA 通道并定义上述寄存器以及数据的传输方向，即读取或写入，然后将设备设定为利用该 DMA 通道传输数据。设备完成操作之后，立即就可以利用该 DMA 通道在设备和系统内存之间传输数据。传输完毕之后产生中断以便通知

设备驱动程序进行后续处理。在利用 DMA 进行数据传输的同时，CPU 仍然可以继续执行指令。

在利用 DMA 进行数据传输时，有几个需要注意的问题：

1. DMA 地址寄存器代表系统物理地址的低 16 位，加上页寄存器中的高 8 位地址，DMA 所能访问的地址限制在低 16 MB（早期的 DMA 控制器限制更严重）。
2. 因为在 DMA 操作时，CPU 仍然在执行指令，因此要防止写入的物理内存区域被虚拟内存机制交换到交换空间中，这可通过锁定相应的物理页避免。
3. DMA 通道是有限资源。有些设备固定使用同一个通道，例如，软盘驱动器使用通道 2；有些设备可通过硬件跳线选择 DMA 通道；另外一些设备则更加灵活，它可以使用任何一个空闲的 DMA 通道，为此，Linux 内核为每个 DMA 通道维护一个 dma_chan 数据结构，根据该结构中的信息可判断指定的 DMA 通道是否已被分配。

12.4 中断及中断处理

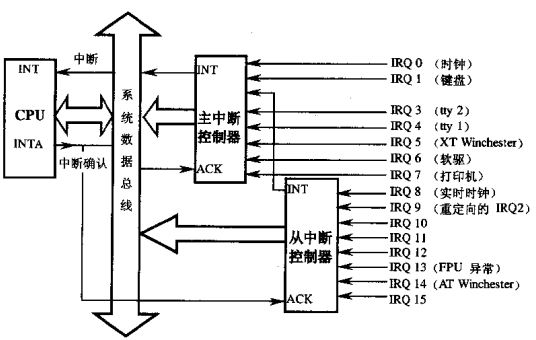


图 12-4 32 位 i386 PC 的中断处理硬件

如上所述，查询方式实际是同步驱动设备的方式，而中断方式实际是异步驱动设备的方式，利用中断，系统可更加有效地利用 CPU。本节讲述 Linux 是如何处理中断的。

12.4.1　中断处理硬件

CPU 在一些外部硬件的帮助下处理中断。中断处理硬件和具体的系统相关，但一般而言，这些硬件系统和 i386 处理器的中断系统在功能上是一致的。图 12-4 所示是 PC-AT 微机中的中断处理硬件。

外部设备产生的中断实际是电平的变化信号，信号的变化出现在中断控制器的 IRQ（中断请求）管脚上，这一信号首先由中断控制器处理。中断控制器可以响应多个中断输入，它的输出连接到 CPU 的 INT 管脚，CPU 在该管脚上的电平变化可通知处理器产生了中断。如果 CPU 这时可以处理中断，CPU 会通过 INTA（中断确认）管脚上的信号通知中断控制器已接受中断，这时，中断控制器可将一个 8 位数据放置在数据总线上，这一 8 位数据也称为中断向量号，CPU 依据中断向量号和中断描述符表（IDT）中的信息自动调用相应的中断服务程序。图 12-4 中，两个中断控制器级联了起来，从属中断控制器的输出连接到了主中断控制器的第 3 个中断信号输入，这样，该系统可处理的外部中断数量最多可达 15 个，图的右边是 i386 PC 中各中断输入管脚的一般分配。

中断控制器中的控制寄存器实际映射到了 CPU 的 I/O 地址空间中，通过对寄存器的设置，可设定中断控制器屏蔽某些中断，也可以指定中断控制器的特殊响应方式，因此，中断控制器也称为可编程中断控制器。在 Linux 中，两个中断控制器初始设置为固定优先级的中断响应方式。有关可编程控制器的详细信息可参阅有关的资料。

12.4.2　Linux 的中断处理软件

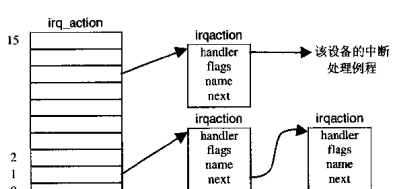


图 12-5　Linux 的中断处理数据结构

在 i386 系统中，Linux 启动时要设置系统的中断描述符表，即 IDT。IDT 中包含各个中断（以及异常，诸如浮点运算溢出）的服务程序地址，中断服务程序地址由 Linux 提供。每个设备驱动程序可以在图 12-5 所示的结构（irq_action）中注册自己的中断及中断处理程序地址。Linux 的中断服务程序根据 irq_action 中的注册信息调用相应的设备驱动程序的中断处理程序。和硬件相关的中断处理代码隐藏在中断服务程序中，这样，设备驱动程序的中断处理程序可在不同平台之间方便移植。一般而言，CPU 在处理中断

时，首先要在堆栈中保存与 CPU 指令执行相关的寄存器（例如指令计数寄存器），然后调用中断服务程序，中断服务程序结束时再恢复这些寄存器。

irq_action 实际是一个数组，其中包含指向 irqaction 的指针，每个数组元素分别定义一个 IRQ。Linux 内核提供相应的操作函数，设备驱动程序可调用这些操作函数设置相应的中断处理函数。一般在系统启动时，由各个设备驱动程序通过如下途径获取相关的设备 IRQ 并设置对应的 irq_action 数组元素所指向的 irqaction 结构：

1. 对某些设备，例如软盘驱动器，其 IRQ 是固定的。软盘驱动器的 IRQ 为 6。
2. 设备驱动程序可通过探测程序自动探测某些 ISA 设备的 IRQ。
3. 不能自动探测的 ISA 设备，可通过启动参数指定设备的 IRQ。
4. 对 PCI 设备，设备的配置头信息中已经存在有相应的设备 IRQ，设备驱动程序只需读取该配置信息。

对 ISA 设备 IRQ 的探测过程如下。设备驱动程序要求设备完成某项可导致中断的操作，然后，系统打开所有未被分配的中断。如果设备产生中断，系统将接收到该中断，然后通过读取中断控制器的状态寄存器可得知该中断对应的 IRQ，它就是 ISA 设备的 IRQ。当然，这种探测方式在某些情况下是无法正常工作的，例如，当 ISA 设备的 I/O 地址空间也未知时，就无法利用这种方式。这种情况经常发生在配置 ISA 网络卡的时候，这时就需要利用启动参数指定设备的 IRQ 和 I/O 端口地址。

对 PCI 设备，由于可方便获得设备配置头信息中的 IRQ，因此设置工作相对容易些。但因为 PCI 设备只能在四个引脚（分别为 A、B、C 和 D）上产生中断，所以当系统中有四个以上的 PCI 设备时，就会发生 IRQ 重叠的情况。这时，irq_action 数组中的某个元素会同时指向多个 irqaction 结构（见图 12-5），而当产生该 IRQ 中断时，Linux 会依次调用每个 irqaction 指定的中断处理程序。

在设备驱动程序的中断处理过程中，设备驱动程序读取设备状态寄存器的值，从而可了解操作结果。在某些特定情况下，设备驱动程序还要进行其他处理，但中断处理过程中只读取设备状态，以便能够快速结束中断处理。如果还要进行其他处理，会通过下面要讲到的内核机制在适当的时候进行。

12.5 设备驱动程序

12.5.1 设备驱动程序的概念

在多任务操作系统中，有很多理由需要内核建立应用程序和设备之间的抽象接口，而不是由应用程序直接操作硬件。为此，操作系统一般提供设备驱动程序来专门完成对特定硬件的控制。设备驱动程序实际是处理或操作硬件控制器的软件，从本质上讲，它们是内核中具有高特权级的、驻留内存的、可共享的底层硬件处理例程。

在 Linux 系统中，一个基本的特点是它抽象了设备处理。所有对硬件设备的操作和通常的文件一样，利用标准的系统调用可在设备上进行打开、关闭、读取或写入操作。系统中的每个设备由“设备特殊文件”来代表。例如，/dev/hda 代表系统中的第一个 IDE 硬盘，而 /dev/sda1 则代表系统中 SCSI 硬盘上的第一个分区。每个由相同的设备驱动程序控制的设备具有相同的主设备号，而次设备号则用来区分同类设备中不同的设备。

设备特殊文件的 VFS 索引节点中包含设备号信息。如果通过系统调用访问设备，则内核可通过该 VFS 索引节点中的设备号信息调用适当的设备驱动程序。

Linux 中的设备分为字符设备、块设备和网络设备三种。字符设备是不通过缓冲而直接进行读取和写入操作的设备。块设备只能以块（通常为 512 字节或 1024 字节）的倍数大小读取或写入。对块设备的访问一般通过缓冲区高速缓存访问，并且可随机访问。网络设备一般通过 BSD 套接字接口访问，详细内容将在第十四章中讲述。

尽管 Linux 中很多类型的设备驱动程序，但它们有一些共同的特点，如表 12-2 所示。

表 12-2　Linux 设备驱动程序的共同特点

作为内核的一部分	设备驱动程序作为内核的一部分而存在，因此“恶意”的驱动程序会破坏系统。
提供内核接口	设备驱动程序要为内核或内核的子系统提供标准接口。
利用内核机制和服务	设备驱动程序要使用一些诸如内存分配、中断传输及等待队列等内核机制和服务。
可装载	大部分设备驱动程序可作为内核模块在必要时装入，而不再需要时卸载。
可配置	Linux 设备驱动程序可内建到内核中，可在编译时指定内建的设备驱动程序。
动态性	如果装载的设备驱动程序启动时没有找到对应的设备，则该设备驱动程序只是占用了一些系统内存，对系统并没有害处。

12.5.2　设备驱动程序的内存分配

设备驱动程序作为内核的一部分，不能使用虚拟内存，因此也不能依赖于任何一个进程运行。和内核的其他部分一样，设备驱动程序也利用各种数据结构跟踪所控制的设备。这些数据结构可作为驱动程序代码的一部分而静态分配，但这种方法会导致内核变大，甚至浪费内存。为此，大部分驱动程序利用内核提供的服务函数动态分配和释放非分页的系统内存。内核以 2 的幂为大小分配内存，例如 128 字节或 512 字节。驱动程序要求的字节数被圆整到下一个页块边界上，这样，释放的物理内存可以重新组合成大的页块。

当系统内存较少时，内核会通过交换操作将某些物理内存页交换到交换空间中，为此，内核会将请求分配的驱动程序暂时挂起，直到有足够的内存时才分配内存并返回成功值。但是，有时驱动程序不允许内存分配上的延迟，因此，可请求内核在这种情况下返回失败值。另外，当驱动程序需要分配用来进行 DMA 操作的内存时，须指定该内存应当是可进行 DMA 操作的内存（处于低 16MB 地址空间中）以便能正确分配内存。

12.5.3　设备驱动程序和内核的接口

每种类型的驱动程序，不管它们是字符设备、块设备还是网络设备，均为内核提供相同的调用接口。于是，内核可以以相同的方式处理非常不同的设备，例如，内核可通过相同的函数调用让 SCSI 和 IDE 硬盘完成相同的工作。表 12-2 还说明了 Linux 设备驱动程序具备的其他一些特点。为了实现这些特点，Linux 为每种不同类型的设备驱动程序维护相应的数据结构，以便定义统一的接口并实现驱动程序的可装载性、动态性等。

对字符设备而言，Linux 内核维护一个 chrdevs 结构体数组，该数组的元素实际

是 device_struct 结构（图 12-6）。主设备号用作访问 chrdevs 数组的索引，例如，tty 设备的主设备号为 4。每个 device_struct 结构包含两部分信息。name 是设备驱动程序注册的设备名称；另一部分包含设备的标准操作例程集，其中的操作例程由设备驱动程序定义，并分别完成指定的设备操作。

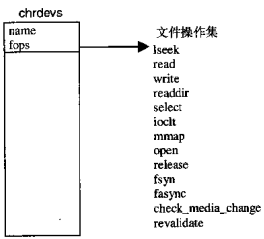


图 12-6 Linux 内核维护的字符设备数据结构

当代表某个字符设备的设备特殊文件被打开时，内核负责进行一些设置工作，以便能够调用正确的设备操作例程。和其他普通的文件或目录一样，设备文件也由 VFS 索引节点代表。VFS 索引节点中包含设备的主设备号和次设备号，每个 VFS 节点和一组文件操作函数的指针关联。系统在建立一个代表字符设备文件的 VFS 索引节点时，该 VFS 节点的文件操作函数指针被设置为默认的字符设备操作函数指针。这时实际只有一个文件操作函数被定义，即文件的打开操作。应用程序打开某个字符特殊文件时，该打开操作将使用 VFS 节点中的主设备号在 chrdevs 数组中检索相应的设备操作函数地址，同时在进程的文件数据结构中建立一个 file 结构（参见第十一章），该结构的文件操作函数指针指向设备驱动程序定义的设备操作函数指针。此后，应用程序在该设备特殊文件上进行的读取、写入等操作就映射到了特定的字符设备操作。

和字符设备类似，Linux 内核利用 blkdevs 数组记录所有的块设备，每个数组元素也是一个 device_struct 结构体，主设备也当作数组索引使用。在 device_struct 结构体中，块设备驱动程序为通常的文件操作提供接口。除此之外，块设备在内核中的数据结构比字符设备要复杂一些，主要是因为内核为块设备提供缓冲区高速缓存，实际的读写操作由缓冲区缓存协调调用，因此，还需要驱动程序为缓冲区缓存提供接口。为此，内核利用 blk_dev 数组（图 12-7）定义这些接口。访问该数组的索引仍然是主设备号，每个块设备驱动程序在 blk_dev 数组的相应位置填充 dlk_dev_struct 结构。blk_dev_struct 结构包含一个处理请求的函数地址，以及一个指向 request 结构体的指针。实际上，多个 request 结构形成了一个链表，而

每个 request 结构代表由缓冲区高速缓存请求设备读取或写入的请求信息。

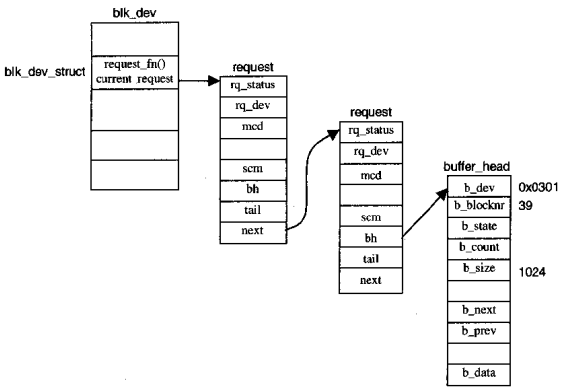


图 12-7　Linux 块设备驱动程序的数据结构

每当缓冲区缓存要从注册设备中读取数据或向设备写入数据时，缓冲区缓存就会在 blk_dev_struct 中添加一个 request 结构。从图 12-7 可看出，每个 request 结构中包含一个指向 buffer_head 结构的指针，每个 buffer_head 结构定义要读取或写入的数据块。进行读写操作时，buffer_head 结构由缓冲区缓存锁定，这时，等待该缓冲区操作结束的进程被阻塞。当请求被添加到空的请求队列时，驱动程序的请求函数（request_fn）被调用，由该函数完成请求队列的处理。

当驱动程序完成请求处理之后，它从 request 结构中移去 buffer_head 结构并标志该结构包含新数据，然后解锁 buffer_head 结构。对 buffer_head 结构的解锁将唤醒所有正在等待该操作完成的进程。

request 结构并不是动态分配的，每次需要新的 request 时，该结构从一个大的静态 all_request 链表中取得，对应的请求被处理之后，该结构标志为空闲的 request 结构。

12.5.4　网络设备

网络设备由 Linux 的网络子系统使用，其重要功能是数据的发送和接收。Linux 系

统中的网络设备和一般的硬件设备有如下不同之处：

- 某些网络设备是硬件设备，而有些网络设备则是软件设备，并不存在实际的硬件设备与之对应，如回环设备。
- 一般块设备或字符设备特殊文件可以通过 **mknod** 命令建立，而网络设备只有在系统引导时发现和初始化之后才存在。

Linux 内核利用 device 数据结构来描述网络设备。在系统引导期间，网络设备驱动程序在 Linux 的网络子系统进行初始化时注册设备信息。device 数据结构中包含了设备信息，以及一些用来传输数据的函数地址。在 Linux 网络子系统中传输和接收的数据包，均由 sk_buff 数据结构代表。在第十四章中，我们将详细介绍网络子系统中的数据结构及其数据传输，本小节主要讲述 device 数据结构和网络设备的初始化。

1. 网络设备数据结构

device 数据结构中包含如表 12-3 所示的设备信息。

表 12-3 device 数据结构包含的主要信息

名称	系统在引导时发现并初始化网络设备之后，网络设备文件的名称自动出现在 /dev 目录中。网络设备的名称遵循一定的标准，每个名称代表设备的类型，而同一类型的设备从 0 开始编号，例如，所有的以太网设备名称为 /dev/eth0、/dev/eth1 和 /dev/eth2 等。常见网络设备有：	
	/dev/ethN	以太网设备
	/dev/slN	SLIP 设备
	/dev/pppN	PPP 设备
	/dev/lo	回环设备
总线信息	其中包括设备驱动程序用来控制设备的信息。如下所示：	
	中断请求	该设备使用的中断请求号
	基地址	该设备使用的控制和状态寄存器的基地址
	DMA 通道	该设备使用的 DMA 通道
接口标志	接口标志描述了网络设备的特征和能力。如下所示：	
	IFF_UP	接口已经过初始化，正在运行
	IFF_BROADCAST	device 中的广播地址有效
	IFF_DEBUG	设备调试能力打开
	IFF_LOOPBACK	该设备是回环设备
	IFF_POINTTOPOINT	该设备是点到点链路设备（PPP 和 SLIP 设备）
	IFF_NOTRAILERS	无网络追踪器
	IFF_RUNNING	资源已分配
	IFF_NOARP	不支持 ARP（地址解析协议）
	IFF_PROMISC	设备处于混杂接收模式，设备将忽略数据包的目标地址信息而接收所有的数据包
	IFF_ALLMULTI	接收所有的 IP 多点传送帧
	IFF_MULTICAST	可以接收 IP 多点传送帧
协议信息	这类信息描述网络协议层如何使用设备：	
	mtu	网络可以传输的最大的数据包尺寸，不包括必须添加的链路层头数据。该值可以由某些协议层（如 IP）用来选择合适的数据包大小。
	地址族	设备支持的地址族。常见的协议族是 AF_INET，即 Internet 地址族。
	类型	描述设备所连接的网络介质的硬件接口类型。Linux 网络设备支持的介质类型包括：以太网、令牌环、X.25、SLIP、PPP 和 Apple Localtalk

		等。
	地址	和网络设备相关的地址信息，包括 IP 地址。
数据包队列	包含了等待由该网络设备发送的 sk_buff 数据包队列。	
支持函数	每个设备包含了一组标准的例程，协议层可以将这些例程当作设备链路层的部分而调用。其中包括设置、帧传输例程，以及添加标准数据帧头和收集统计信息的例程。ifconfig 命令使用这些统计信息。	

2. 网络设备的初始化

和其他的 Linux 设备驱动程序一样，网络设备也可以建立到 Linux 内核中。每个可能的网络设备由 device 数据结构代表，所有的 device 数据结构包含在由 dev_base 指针指向的设备链表中。网络协议层需要网络设备完成特定的工作时，调用保存在 device 数据结构中的网络设备服务例程。但在最开始，每个 device 设备中只包含一个初始化或检测例程的地址。

网络设备驱动程序有两个问题需要解决。首先，并不是所有建立到 Linux 内核中的网络设备驱动程序均有相应的可控制设备。其次，不管底层设备是哪种硬件型号，以太网设备始终以 /dev/eth0、/dev/eth1 等方式命名。第一个问题容易解决，当系统引导时，每个网络设备的初始化例程被调用，该初始化例程可返回一个值表明是否存在该设备所描述的实际硬件，若不存在，只需将该设备在 dev_base 链表中的节点删除。反之，网络设备驱动程序填充 device 结构中的其余信息以及支持函数。

第二个问题，即网络设备特殊文件名称的动态分配问题，解决起来稍微复杂一些。在设备链表中，一共有 8 个标准的入口，从 eth0，eth1 一直到 eth7。它们的初始化例程均是一样的，该例程依次调用内核中的以太网设备驱动程序，当某个驱动程序发现对应的以太网设备时，则从 eth0 开始依次填充空闲的 device 数据结构。这时，网络设备驱动程序还要初始化物理硬件，搜集有关的总线信息。有时，一个驱动程序会找到多个可控制的网络设备，这种情况下，它将填充多个 /dev/ethN device 数据结构。当所有 8 个标准的 /dev/ethN 全部分配之后，就不再继续检测其他的以太网设备。

下面的小节主要包括一些常见硬件及其相关概念，同时介绍相关系统工具，这些内容对正确安装和维护 Linux 系统有相当大的帮助。

12.6　硬盘

如图 12-8 所示是硬件的物理结构示意图。从图中可以看出，硬盘实际是由多个“磁盘片”组成的，每个磁盘片均有两个“面”。在磁盘出厂低级格式化之前，每个面上均匀分布磁粉。经过低级格式化之后，每个面上的磁粉被组织成一圈圈的“磁道”，而磁道又被划分成为“扇区”，扇区是真正保存数据的地方，各磁道上的扇区数是相同的。另外，硬盘还有一个术语称为“柱面”，它指的是由所有面上的同一磁道假象形成的几何柱面，该几何柱面的轴就是盘片的旋转轴。每个面对应有一个磁头，用来感应磁道上磁粉的磁极方向，经过其他电路处理可将磁粉的磁极转化为计算机能够识别的二进制数据。磁头能够

沿盘片的径向移动，在盘片旋转运动的辅助下，磁头可以读取盘片上所有扇区中的数据。“磁盘控制器”完成对来自 CPU 的命令的解释，并控制磁头的移动和读写。

如果我们知道了硬盘的磁头个数、柱面个数、每磁道上的扇区个数（这些参数称为硬盘的“几何参数”）以及每扇区上可存储的字节数，就可以计算出这个硬盘的容量。例如，某计算机的磁头个数为 128，每磁道上的扇区个数为 63，一共有 788 个柱面，每扇区可容纳 512 字节，因此，该硬盘的容量为 128 × 63 × 788 × 512 = 3,253,469,184（字节）= 3102.75（兆字节）。

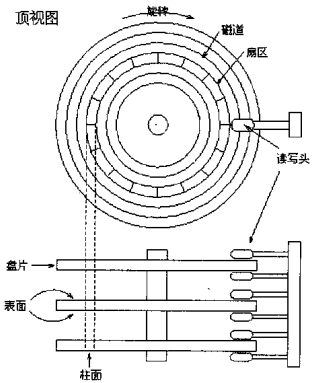


图 12-8 硬盘的物理结构示意图

上面所举的例子中，该硬盘一共有 128 个磁头，也就是说，它一共有 64 个磁盘片。但根据常识，薄薄的硬盘中不可能存在这样多的磁盘片。实际上，现在常见的大容量硬盘的这些参数并不对应于实际的物理参数。这样定义的原因是由于 PC 机早期 BIOS 设计上的一个缺陷。早期的 BIOS 在 CMOS RAM 中存放这些参数，由于 CMOS RAM 的容量有限，初始设计为不能存放大于 1024 的磁道个数（那时的硬盘容量一般为 40 MB）。但是，随技术的发展，硬盘的容量越来越大，这主要是磁道个数增多的原因，而不是磁头个数增加的原因。为了和原有的 BIOS 兼容，硬盘制造商仍然利用小于 1024 的磁道数定义硬盘参数，而通过增加磁头数等方法保证所定义的逻辑参数能够和物理参数匹配。同时，随着磁道数的增多，又引入了另一个问题。处于盘片中心处的磁道周长小，而处于盘

片外沿的磁道周长大，这样，外沿磁道上可容纳的扇区个数比处于中心处的磁道可容纳的扇区个数要多。硬盘制造厂商为了尽量增加硬盘容量，实际所采用的也是变化的扇区数。但是，无论硬盘的实际构造如何，硬盘制造商却始终给出符合上述结构的逻辑参数，从而保证硬盘访问软件在一定程度上的一致性，而逻辑参数到物理参数的转换则由硬盘控制器完成。

上面所讲的问题主要出现在 IDE 接口的硬盘中，这种硬盘也是当今 PC 机中使用最为广泛的硬盘。SCSI 硬盘是另外一种可提供大容量、高速度数据访问的硬盘。这种硬盘没有 IDE 硬盘这样的问题，它利用顺序号定位扇区，而顺序号到物理磁头、柱面、扇区的转换由硬盘控制器完成，由于这一原因，操作系统对 IDE 和 SCSI 硬盘的驱动程序各不相同。

磁头的跨磁道读写一般因为需要径向移动而需要花费大量的时间，这一般用“平均寻道时间”来衡量。如果一个硬盘的平均寻道时间很长，则说明硬盘的速度很低。在文件系统的组织中，操作系统一般会尽量将同一个文件的数据放置在连续的扇区中，这样，磁头在读取过程中可以少一些寻道时间，从而提高硬盘的整体速度。然而，随着文件的删除和扇区的重新分配，一些文件就无法保证存储在连续的扇区中，这时，硬盘的读写速度将受到影响。这种情况一般称为“磁盘的碎片化”。许多操作系统都包含一些可通过对文件存储扇区的整理而降低碎片化程度的系统工具。

在 Linux 中，每个硬盘由单独的设备文件表示。通常，可有两个或四个硬盘，这一般取决于 PC 的主板和 BIOS。这些硬盘分别由 /dev/hda、/dev/hdb、/dev/hdc 和 /dev/hdd 表示，而 SCSI 硬盘的个数则不受上述限制，每个 SCSI 总线上可以有 7 个 SCSI 设备，因而可最多连接 7 个硬盘，分别由 /dev/sda、dev/sdb、dev/sdc、dev/sdd、dev/sde、dev/sdf 和 dev/sdg 表示。

12.7　软盘

软盘的结构和硬盘类似，只是软盘的磁头包含在驱动器内部，而盘片却是可以从驱动器中拿出的，因而软盘也称“可移动介质”。因为软盘只有一个盘片，因此，软盘盘面一般固定为 2 个，而磁道、扇区数量的不同也将软盘分为不同的类型。现在最常见的是 3½ 英寸大小的 1.44 MB 软盘。这种软盘一般称为双面高密度盘，Linux 中表示为 /dev/fd0H1440，H 表示高密度，1440 表示容量为 1440 K 字节，约为 1.44 M 字节，fd0 则表示 BIOS 定义的第一个软盘驱动器（DOS 中的 A: 驱动器）。其他种类的软盘也按上面的方法表示，例如 /dev/fd1D720，表示第二个软盘驱动器，双面双密度容量为 720 K 字节的软盘。

所有的软盘种类总共约有 9 种，但 Linux 可通过特殊的设备文件自动检测插入的软盘种类，即 /dev/fd0 和 /dev/fd1。因此，一般利用这两个设备文件访问软盘驱动器。

12.8 格式化和分区

12.8.1 格式化

如上所述，磁盘在能够使用之前，必须首先经过低级格式化，低级格式化的目的是将磁盘上的磁粉组织成规定的磁道和扇区，经过低级格式化的磁盘才可以使用。“低级格式化”一词，实际来自 MS-DOS，因为 DOS 的 **FORMAT** 命令一般完成两个步骤，一是低级格式化，二是建立文件系统。为了区分这两个过程，建立文件系统的过程又称为“高级格式化”。在 Linux 中，这两个步骤是分开进行的，分别称为“格式化”和“建立文件系统”，因此，下面的描述中我们严格区分“格式化”和“建立文件系统”这两个术语。

硬盘在出厂时，已经经过了格式化，一般不需要再次进行格式化，经常性的格式化会减低硬盘的使用寿命。

因为硬盘构造上的差别，硬盘的格式化通常需要特殊的程序来完成，在 DOS 中，这种功能一般由 BIOS 或特殊的 DOS 程序提供，而在 Linux 中，这两种方法都不太容易利用。和硬盘比较起来，软盘经常被格式化，同时，软盘的格式化也比较容易实现，这是因为软盘驱动器的构造较硬盘驱动器透明。可利用下面的命令格式化软盘：

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 KB.
Formatting ... done
Verifying ... done
$
```

在上述命令中，通过命令参数指定了要格式化的软盘参数：双面双密度、1440 K 字节，并且插入第一个驱动器中。如果要使用 /dev/fd0 这一可自动检测的设备文件，则需要利用 **setfdprm** 命令指定软盘参数：

```
$ setfdprm /dev/fd0 1440/1440
```

利用 **setfdprm**，还可以为特殊软盘指定格式信息。

利用 **fdformat** 格式化软盘时，这一程序还可检验软盘上的坏扇区。通过在扇区上写入、读出并比较来检验扇区。如果某个扇区经过多次检验仍然错误的话，则认为该扇区为坏扇区，**fdformat** 会在这种情况下终止检验并退出。

磁盘上的坏扇区信息一般由文件系统维护，以免将来使用坏扇区保存数据。对于硬盘，少量的坏扇区还可以在（低级）格式化过程中由磁盘驱动器维护，写入坏扇区的数据，硬盘驱动器会自动映射到其他正常的扇区中。但这样会降低硬盘的性能。如果硬盘上的坏扇区数量较多，且集中出现在某处，则可以通过分区避免使用这些坏扇区。由于操作系统一般使用“块”来分配存储空间（一个块一般是两个连续的扇区），因此，文件系统中实际维护的是坏块的信息。

不管是软盘或硬盘，都可以利用 **badblocks** 命令检查坏块。例如：

```
$ badblocks /dev/fd0H1440 1440
718
719
$
```

在上面的例子中，**badblocks** 报告 718 和 719 两个块是坏块，但不对这两个块作

任何标记。如果上述软盘中包含文件系统，则应当利用 **fsck** 将坏块信息添加到由文件系统维护的坏块表中，如果该软盘不包含文件系统，则利用 **mkfs** 命令建立文件系统时，会自动检查并建立坏块表。

12.8.2 分区

硬盘可以划分为不同的“分区”，每个分区当作单独的硬盘一样操作。对硬盘进行分区的原因有：

- 硬盘很大，分区可提高硬盘的访问效率。例如，对 FAT 分区来说，低于 400 MB 的分区性能较好。
- 需要安装多个操作系统。例如，如果要在某台计算机的同一张硬盘上同时安装 Windows 95 和 Linux 两种操作系统，就必须对该硬盘进行分区，并将两个操作系统安装在不同的分区中。
- 为方便管理而分区。在不同的分区上放置不同的文件系统，可方便系统管理和维护。

在安装操作系统时，分区可能是最重要的步骤之一。如果要建立多重引导（多个操作系统）系统，必须了解每个操作系统对分区的要求，并认真制定分区计划。为此，必须掌握有关分区的概念和术语。（注：这些概念是 i386 平台所特有的，对于其他平台的计算机，例如 Alpha 工作站，分区、引导的方法不同。）

1. 主引导记录、引导扇区和分区表

硬盘的分区信息包含在硬盘的第一个扇区中，也即第一个面的第一个磁道的第一个扇区中。通常将硬盘的第一个扇区称为“主引导记录 （MBR）”。主引导记录中实际包含了一小段程序，系统 BIOS 读取主引导记录，然后执行。该程序首先读取“分区表”，判断活动分区（即标记为可引导的分区），然后读取活动分区的第一个扇区。活动分区的第一个扇区又称为“引导扇区”，其概念和作用与主引导记录类似，但由于主引导记录的特殊地位而拥有一个特殊名称。引导扇区负责装入并启动操作系统。

分区的方法实际和硬件无关，也和 BIOS 无关，但是如果要让来自不同软件制作者的操作系统安装在同一个计算机中，没有一个分区的标准是不行的。当然，有些操作系统不遵循这样的标准，因此也很难和其他操作系统共存。Linux 现在可以和 Microsoft 的操作系统共存，它实际遵循 IBM PC 机的硬盘分区方法。这种方法中，分区表指定了硬盘中的分区个数、各分区类型、起始位置等信息。

2. 主分区、扩展分区和逻辑分区

Linux 在 i386 平台上所遵循的分区方法将分区划分为两种类型：主分区和扩展分区。每个硬盘可有四个主分区，但其中只有一个可作为引导分区，即活动分区。如果使用扩展分区（每个盘上仅能有一个扩展分区），则主分区最多只能有三个。每个扩展分区可以进一步划分为四个逻辑分区，逻辑分区不能作为引导分区。

下面列出的是某台计算机的分区情况：

```
$ fdisk -l /dev/hda
Disk /dev/hda: 128 heads, 63 sectors, 788 cylinders
Units = cylinders of 8064 * 512 bytes
```

```
    Devie   Boot    Begin    Start    End      Blocks      Id  System
/dev/hda1     *        1        1     261    1052320+     b  Win95 FAT32
/dev/hda2              262      262     522    1052322      5  Extended
/dev/hda3              523      523     779    1036224     83  Linux native
/dev/hda4              780      780     788      36288     82  Linux swap
/dev/hda5              262      262     522    1052320-     6  DOS 16-bit >=32M
$
```

图 12-9 表示了该硬盘的分区情况。

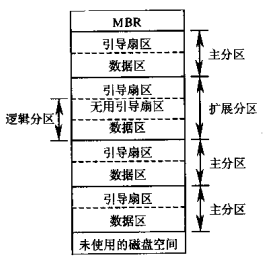


图 12-9 某硬盘的分区情况

上述硬盘中包含三个主分区，分别为 /dev/hda1、/dev/hda3 和 /dev/hda4；一个扩展分区，为 /dev/hda2。扩展分区又划分为一个逻辑分区，/dev/hda5。主分区和扩展分区中各有一个引导扇区。

分区类型

除了在主引导记录中有一个分区表之外，在扩展分区中也有一个分区表。前者定义主分区和扩展分区的分区信息；而后者定义逻辑分区的分区信息。分区表中为每个分区包含有一个字节的信息，用来说明该分区的类型。指定分区类型的目的只是为了让不同的操作系统能够区别这些分区，以避免同时使用不同的分区。例如，图 12-9 所描述的硬盘分区就为不同的操作系统所用。但是，操作系统可能并不判断分区类型。例如，Linux 在任何类型的分区中均可以建立任意类型的文件系统。相反，Microsoft 的操作系统则根据分区类型判断是否可访问，因此，非 FAT、VFAT、FAT32 或 NTFS 等分区是无法识别的。

不同分区类型在分区表中的类型字节值不同，表 12-4 给出了部分分区类型的类型值。

表 12-4 常见的分区类型

类型	说明	类型	说明	类型	说明
0	空	40	Venix 80286	a5	BSD/386
1	DOS 12-bit FAT	51	Novell	b7	BSDI fs
2	XENIX root	52	Microport	b8	BSDI swap
3	XENIX usr	63	GNU HURD	c7	Syrinx
4	DOS 16-bit <32M	64	Novell	db	CP/M
5	Extended	75	PC/IX	e1	DOS access
6	DOS 16-bit ≥32M	80	Old MINIX	e3	DOS R/O
7	OS/2 HPFS	81	Linux/MINIX	f2	DOS secondary
8	AIX	82	Linux swap	ff	BBT
9	AIX bootable	83	Linux native		
a	OS/2 Boot Manage	93	Amoeba		
b	Win95 FAT32	94	Amoeba BBT		

利用 **fdisk** 进行分区时，可以获得同样的分区类型表。

硬盘的分区

和 DOS 一样，Linux 中用来分区的程序也称为 **fdisk**。和 DOS 的 **fdisk** 不一样的是，Linux 的 **fdisk** 不关心分区类型，从而能够对各种类型的分区进行操作。**cfdisk** 和 **fdisk** 类似，但 **cfdisk** 可在全屏幕方式下进行操作。只要掌握了上述概念，利用这两个程序进行分区就不会太困难。

进行分区时，需要注意如下问题：

- 使用 IDE 硬盘时，因为 BIOS 的限制，引导分区（包含可引导内核文件的分区）只能完全包含在前 1024 柱面内。
- 应当确保分区包含偶数个扇区，这是因为文件系统数据块的大小一般为 2 个扇区。
- 在修改分区之前，应当备份重要数据。许多 MS-DOS 的实用工具可简化分区过程，并能保证数据的安全，这些工具包括 **fips** 和 **qpmagic** 等。

分区的对应设备文件

每个主分区、扩展分区以及逻辑分区和相应的设备文件对应。这些文件的名称一般是在表示整个硬盘的设备文件之后添加分区编号。主分区或扩展分区的编号按顺序从 1 到 4；逻辑分区的编号从 5 开始，依次编号，最大为 8。例如，/dev/hda2 指第一个 IDE 硬盘的第二个主分区或扩展分区。

12.8.3 无文件系统的磁盘

并不是所有的磁盘或分区以文件系统的方式使用，例如，Linux 以原始方式使用交换分区；许多软盘中的数据也是以原始方式组织的，例如，Linux 的引导软盘中就没有文件系统。避免使用文件系统有如下好处：

- 可以充分利用磁盘空间，因为文件系统总是有一些额外的开销。
- 能够提高磁盘在不同系统之间的兼容性。例如，利用 **tar** 备份的数据，几乎可在任意一个操作系统中读取。

➢ 方便数据的备份。

无文件系统磁盘的使用主要利用 **dd** 命令。如下所示，可将磁盘上的数据保存到文件系统中的 *floppy-image* 文件中：

```
$ dd if=/dev/fd0H1440 of=floppy-image
2880+0 records in
2880+0 records out
$
```

利用如下的命令可将文件系统中的 *floppy-image* 文件以原始方式保存到磁盘上：

```
$ dd if=floppy-image of=/dev/fd0H1440
2880+0 records in
2880+0 records out
$
```

MS-DOS 中用来读取原始软盘的工具主要有 **hd-copy** 和 **rawwrite** 等。

12.9 其他存储设备

12.9.1 CD-ROM

由于其大容量和非易失性，CD-ROM 现在经常作为软件的发行介质，但和硬盘比较起来，CD-ROM 的速度较低。

通常而言，CD-ROM 上的数据按国际标准 ISO9660 组织。这是一个非常小的文件系统，类似 MS-DOS 的 FAT 文件系统。但正因为它小，所以几乎能被任何一种操作系统映射为自己的文件系统。

PC 机上常见的 CD-ROM 驱动器接口为 EIDE 以及 SCSI，有些老式的还通过声卡连接。一般而言，通过 IDE 连接时的设备文件命名方式和硬盘一样。通常，在 /dev 目录中，有一个符号链接 /dev/cdrom 指向实际的 CD-ROM 设备：

12.9.2 磁带

和常见的录音磁带类似，磁带只能顺序存取，因此速度很低，但它也支持随机访问，也就是说，可以从一处跳到另一处。因为磁带的高容量和低成本，所以经常用来进行数据的归档和备份。

12.10 显示卡和监视器

如果仅在文本方式下使用 Linux，则不需要对显示卡和监视器进行特殊设置。一般而言，常见的显示卡和监视器均支持标准的 80 列、25 行的 16 色文本显示模式。但是，如果要在 Linux 上配置运行 X Window（简称 X），则必须了解自己的显示卡和监视器。Linux 的多数商业发行版本均提供一个比较友好的界面，可用来设置显示卡和监视器。例如，在 Red Hat 5.x 中，**Xconfigurator** 可帮助你完成大多数的设置工作。这一程序

可以检测出大多数常见的显示卡型号及其参数。但是，当你的显示卡无法自动检测时，有时需要利用更高级的 **xf86config** 或 **XFree86Setup** 等程序进行设置。在利用 **xf86config** 程序进行设置之前，必须了解有关显示卡和监视器的参数及其意义。

一般而言，显示卡是控制监视器的电路板，插入 PC 主板的 ISA、PCI 或 AGP 插槽中，有些主板则直接包含有图形芯片组。

12.10.1　光栅扫描监视器

所有图形卡的操作原理都是一样的。显示卡中的显示 RAM（VRAM）中存储着要显示的图象，显示卡负责根据 VRAM 中的数据产生信号并在监视器上显示图形。

监视器是用于显示图形或文本的物理设备，对常见的 PC 机而言，显示屏幕一般是荧光显象管，电子束来回扫描并让荧光发光而产生图象。笔记本电脑一般配备的是液晶显示屏。

如图 12-10 所示，监视器中的图象由大量的水平扫描线形成，监视器中的电子束在荧光屏上来回扫描而生成扫描线。

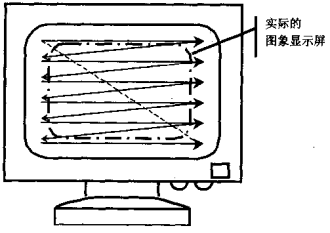


图 12-10　光栅扫描监视器的原理

荧光屏上的单个发光点称为一个“象素”。通过控制扫描电子束的强度就可以显示一条图象线。荧光在显示屏上的显示时间很短，但是由于图象的重复绘制，加上人眼的视觉暂留，可在人脑中形成稳定的图象。一般的监视器可在一秒内重绘 50～90 次。

参照图 12-10，电子束每扫描完一条光栅线之后，要折回到下一条光栅线的开头扫描，电子束的这种运动称为“水平折回”。类似地，电子束扫描完一整屏之后，要重新从第一条光栅线开始扫描，这种运动称为“垂直折回”。这两种运动分别对应监视器的两个参数：“水平扫描频率”和“垂直扫描频率”。

12.10.2 彩色监视器

彩色监视器通过三原色，红、绿、蓝的组合显示颜色。彩色监视器荧光屏上的象素是呈三角形分布的红绿蓝荧光点。红绿蓝电子束分别以不同的强度扫描红绿蓝荧光粉，就可以显示出许多不同的颜色。

12.10.3 调色板和分辨率

我们知道，监视器能够显示的颜色可以上万种，而标准的 VGA 16 色模式只能同时显示 16 种颜色。能够同时显示的颜色数实际是由图形卡的显示模式决定的，同时受到监视器的限制。为了处理上的方便，图形卡预先定义一种调色板，例如包含 256 种颜色的调色板，程序可通过指定调色板中的颜色索引号，即 0 到 255，来指定某象素的实际颜色。通常，调色板中的颜色是可以通过编程改变的。

标准的 VGA 16 色模式定义了 16 种颜色，因此每个象素的颜色可由 4 位二进制数表示；而 256 色模式下，需要用 8 位二进制数表示一个象素。

当前许多流行的显示卡都可以利用 2 个字节（16 位）或 3 个字节（24 位）来表示象素的颜色。这时，用来表示象素的字节直接包含红绿蓝三种颜色。因此，对于 24 位的显示模式来说，它可以显示 $256 \times 256 \times 256 = 16777216$ 种颜色，通常称为“真彩色”，而这种模式也需要更多的显示内存来存储整个图象。

分辨率指水平扫描线和垂直扫描线的数量。不同的显示卡模式对应不同的分辨率，但最高分辨率一般受监视器的限制。常见的分辨率为 640 × 480，即水平可以有 640 个象素，垂直可有 480 个象素。另外，800×600、1024×768 等也是常见的分辨率。

12.10.4 显示内存

显示卡在随机访问储存器中，也即显示内存中保存象素。显示内存的大小直接影响可显示的分辨率和颜色数。对于 256 色、1024×768 的显示模式来讲，就需要 $1024 \times 768 = 786432$ 字节的显示内存。

12.10.5 点时钟

在配置 X Window 的显示卡时，经常要遇到点时钟（dot clock）这一术语。点时钟指显示卡控制整屏扫描线的速率。点时钟的值以每秒能够绘制的点的数量表示。

对于 640×480 的显示模式来说，可见点的数量有 $640 \times 480 = 307200$ 个。为了获得稳定的图象，至少要以每秒 72 次的速度绘制所有这些点，因此，显示卡要在一秒内显示 $307200 \times 72 = 22118400$ 个点，因此，点时钟的值大约为 22 MHz。

但是，实际上点时钟的值要稍微大一些，这是因为扫描线的范围要稍微比 640 × 480 大一些。因此，对 640×480，刷新频率为 72 Hz 的模式，点时钟大约为 25.2 MHz。

老的显示卡支持一组固定的点时钟，而新的显示卡则可以通过编程指定点时钟。如果显示卡的点时钟是可编程的，则 X Server 可以以任意可接受的值操作显示卡，但需要指定用来控制点时钟的芯片，即时钟芯片。

12.10.6 XFree86

我们知道，各种显示卡的内部构造不同，为了提高不同显示卡之间的兼容性，视频电子学标准协会建立了 VESA 标准。VESA 提供了一组附加的 BIOS 功能，利用这组功能可以访问大多数 Super VGA 的扩充模式和功能。在 MS-DOS 中，利用 VESA BIOS 访问 VESA 兼容的 Super VGA 是非常方便的。但是，由于 Linux 建立在 80386 的保护模式之上，因此不能直接利用 VESA BIOS。实际上，Linux 中的 XFree86 通过对显示卡的直接控制而进行操作。因此，某些显示卡可在 XFree86 中支持，而有些，尤其是多数 AGP 显示卡还不能由 XFree86 支持，幸运的是，大多数 AGP 显示卡和某些 Super VGA 卡兼容，因此，也可以利用兼容显示卡对应的 X Server。

监视器对 XFree86 也同样重要，显示卡控制监视器，显示卡给出的点时钟如果和监视器不兼容，也同样无法正常显示图象。显示分辨率不仅和显示内存有关，也和监视器支持的刷新频率有关。

对 XFree86 来说，有关显示卡，以及鼠标、键盘的信息保存在 /etc 目录中的 XF86Config 文件中。不同的 Linux 发行版本保存该文件的位置可能不同，Red Hat 5.x 在 /etc/X11/ 目录中保存该文件。

该配置文件本身由若干信息组成，这些信息以段（Section）的方式组织，分别说明不同的硬件配置情况，见表 12-5。

表 12-5 XF86Config 文件中的主要配置信息

Files	指定 RGB 数据库路径，以及字体文件路径。
Server flags	指定 X Server 的选项。
Keyboard	指定键盘参数。
Pointer	指定鼠标参数。
Moniter	指定监视器参数。
Device	指定显示卡参数。
Screen	给出显示卡、监视器以及 X Server 的综合说明。

一般而言，段的格式如下所示：

```
Section "SectionName"
    Parameter1
    Parameter2
EndSection
```

对 Monitor 和 Device 段而言，可以定义多个不同段，每个段利用不同的字符串标识，如下所示：

```
# 定义两个 Monitor 段
Section "Monitor"
    Identifier     "My monitor"
    VendorName     "Unknown"
    ModelName      "Unknown"
    BandWidth      75.0
    HorizSync      30-64
    VertRefresh    55-90
    Modeline       "640x400"    25.175 640 664 760 800    400 409 411 450
...
```

```
EndSection

Section "Monitor"
    Identifier     "ViewSonic 15GS"
...
EndSection


# 定义两个 Device 段
Section "Device"
    Identifier     "Number Nine GXE64 with S3 Trio64"
...
EndSection

Section "Device"
    Identifier     "Generic VGA"
    VendorName     "Unknown"
    Chipset        "generic"
#    VidoeRam      256
#    Clocks        25.2 28.3
EndSection
```

然后在定义 Screen 段时，可利用标识字符串来引用不同的监视器和显示卡参数，这种方法在一定程度上提高了配置的灵活性。XF86Config 中一般定义有多个 Screen 段，不同的 Screen 段针对不同的 X Server 而设置。X Window 启动时，自动利用最佳（即最能发挥显示卡和监视器性能的设置） Screen 段中的信息。典型的 Screen 段的定义如下：

```
Section "Screen"
    Driver      "accel"
    Device      "Number Nine GXE64 with S3-Trio64"
    Monitor     "My monitor"
    Subsection "Display"
        Depth   8
        Modes   "640x480" "800x600" "1024x768"
        ViewPort  0 0
        Virtual   1024 768
    EndSubSection
    Subsection "Display"
        Depth   16
        Modes   "640x480" "800x600"
        ViewPort  0 0
        Virtual   800 600
    EndSubSection
    Subsection "Display"
        Depth   32
        Modes   "640x400"
        ViewPort  0 0
        Virtual   600 400
    EndSubSection
```

其中，Screen 段中的 Driver 参数指定了所使用的 X Server。XFree86 所使用的 X Server 分为 4 大类，见表 12-6。

表 12-6 X Server 的分类

vga2	VGA 单色模式。
vga16	标准 VGA 16 色模式。
svga	指 Super VGA 的 640x480 256 色模式。
accel	指各种加速 X Server，需指定明确的显示卡芯片组。

不同的 Display 子段一般用于不同的颜色模式，上述例子中指定了三种颜色模式。X Window 启动时自动使用第一个 Display 子段的信息，但可通过指定 -bpp（bytes per pixels，每象素的字节数，和 Display 子段的 depth 参数一样）明确指定使用的 Display 子段，例如：

```
startx -- -bpp 32
```

Display 子段中的 Modes 参数指定了一组显示卡和监视器能够支持的显示模式。"800x600" 等模式名称在 Monitor 段中定义。当某种模式定义有多个显示分辨率时，可在 X Window 运行时利用 Ctrl+Alt+Plus 和 Ctrl+Alt+Minus 在不同的分辨率之间切换，这里的 Plus 和 Minus 是数字键盘上的 "+" 和 "-" 键。如果指定了一个显示卡或监视器无法支持的显示模式时，可利用 Ctrl+Alt+Backspace 按键强制关闭 X Window。

当显示内存的数量比指定模式所需的内存要多时，X Server 可利用多余的内存建立一个比物理屏幕大的虚拟屏幕。这时，可利用 Virtul 指定虚拟桌面的大小，而利用 ViewPort 可指定物理屏幕的左上角在虚拟屏幕中的位置。

参见上面的 Device 段，VendorName、BoardName 和 Chipset 分别指定显示卡的类型和芯片组。对于标准 VGA 模式来说，这些参数是无关紧要的，也不需要指定显示内存（VideoRam）的大小和点时钟（Clocks）。对于许多没有可编程点时钟的显示卡来说，Clocks 参数指定了该显示卡支持的所有点时钟。

参见上面的 Monitor 段，其中指定了监视器的许多技术参数，包括带宽（BandWidth，MHz）、水平同步频率（HorizSync，kHz）和垂直刷新速率（VertRefresh，Hz）。这些参数一般由厂家公布在监视器手册中，并且不能随意设定，否则可能会损坏监视器。

Monitor 段中的 Modeline 是监视器最重要的参数，它可以用两种等价格式给出，一种是单行格式，一种是多行格式，其中包含的数据是一致的。利用单行格式时，其语法如下：

```
Modeline "name" CLK HRES HSS HSE HTOT   VRES VSS VSE VTOT flags
```

表 12-7 解释了上述这些参数的含义。

表 12-7 Modeline 的参数含义

"name"	指定该模式的名称，在 Screen 段中用该名称引用此模式。
CLK	该模式的点时钟。对具有固定点时钟的显示卡来说，该点时钟必须和 Device 段中给出的某个点时钟值相等。
HRES HSS HSE HTOT	水平时序参数。HRES 是光栅扫描线上以象素为单位的水平分辨率。从 图 12-10 可见，实际分辨率要比可见分辨率高，HTOT 即指定实际的象素点

	数。HSS 是水平同步信号的起始点，而 HSE 是水平同步信号的终止点。
VRES VSS VSE VTOT	垂直时序参数。VRES 是光栅扫描线上以象素为单位的垂直分辨率。从 图 12-10 可见，实际分辨率要比可见分辨率高，VTOT 即指定实际的垂直象素点数。VSS 是垂直同步信号的起始点，而 VSE 是垂直同步信号的终止点。
flags	指定模式标志。例如，对隔行扫描方式来说，值为 Interlace。利用该标志也可以指定同步信号的极性，可取 +HSync、-HSync、+VSync 和 -VSync 等值。这些标志可从监视器手册中查到。

如果我们没有任何其他可用资料，却需要配置 modeline，则可从监视器手册中查到特定模式下的垂直刷新速率（Hz）和水平同步频率（kHz）。有时监视器只提供这两个参数的有效区间。一般而言，点时钟和这两个频率之间有如下等式关系：

```
CLK = RR * HTOT * VTOT
CLK = HSF * HTOT
```

其中 RR 指监视器的屏幕刷新频率（处于监视器的垂直刷新频率区间），而 HSF 是监视器的水平扫描频率。例如，当屏幕刷新频率为 72 Hz 时，对于某给定的点时钟，从第一个等式可计算得到 HTOT * VTOT，从第二个等式可计算得到 HTOT，最终可得到 VTOT 以及 HTOT。

这时，所需参数只剩下 HSS、HSE、VSS 和 VSE（HRES 和 VRES 由显示模式的分辨率确定）。但这几个参数只能通过试验才能得到，这些参数应满足如下条件：

```
HTOT > HSE > HSS > HRES
VTOT > VSE > VSS > VRES
```

12.11 键盘和鼠标

键盘和鼠标通常是操作系统的标准输入设备，Linux 也不例外。任何能够在 DOS 下正常工作的键盘，也同样能够在 Linux 中正常工作。但是，Linux 的键盘有一些 DOS 所没有的特殊属性。鼠标是另外一种输入设备，如果不使用鼠标，就无法在 X Window 系统中进行高效操作。鼠标的接口类型有许多种，常见的如串行鼠标、PS/2 接口鼠标等。本节重点讲述 Linux 中和键盘、鼠标相关的术语和概念，以及相应的设置工具。

12.11.1 键盘布局

“键盘布局”指键盘上各个键的布局。一般来说，一个键盘上的键可划分为几部分：打字键、功能键、光标键和数字键等。不同的键盘布局一般具有不同的键个数，主要区别在于功能键的个数、是否有光标键以及是否有数字键等。

Linux 内核一般以某个唯一的数值来表示不同的键，该数值称为“键码”。和 DOS BIOS 中的扫描码概念一样。虽然有些键的名字是一样的，例如，打字键“1”和数字键“1”，它们都表示数字“1”，但它们在 Linux 内核中由两个不同的数字表示。再比如，打字键“#”和打字键“3”一般由同一个键代表，Linux 用同样的数值表示它们，用户击键时这些键所代表的是数字还是符号，这取决于键盘上的修饰键。当用户按打字键“3”时，同时按住了“Shift”键，则说明该键代表的是符号“#”，而不是数字“3”。如果按打字

键“a”时，按住了“Ctrl”键，则该键代表特殊的 ASCII 字符。因此，Shift 键和 Ctrl 键以及 Alt 键等一般称为修饰键。键盘驱动程序根据用户按下的键的键码，以及当前修饰键的按下状态等，将用户的输入翻译为相应的 ASCII 值。

和 Shift 等修饰键不同，Caps Lock 等键虽然也是修饰键，但它们的功能略微有些不同。这些键称为“切换键”或“锁定键”，这种键有：Caps Lock、Num Lock 和 Scroll Lock 键。用户按一次切换键后，键盘上的某些键所代表的含义就会发生变化，再按一次切换键，这些键的含义复原。常用的切换键为 Caps Lock，用来切换字母键的大小写。键盘驱动程序实际为每种切换键保留有一个标志位或标志值，按照这些标志将用户的输入翻译为正确的 ASCII 值或键值。用户按下不同的切换键后，键盘驱动程序还要同时修改键盘右上角 LED 灯的点亮状态。

上述过程是键盘驱动程序在 Linux 虚拟控制台上的处理。因为 Linux 将键盘看成一种字符设备，如果直接读取该设备，则可以获得键盘上所有键的按键状态，从而改变默认的键盘处理逻辑。X Window 对键盘的处理实际就是这样的，它将键盘的键码直接翻译为自己的键符号。

12.11.2 键盘的重复延迟和重复率

这两个概念和 Windows 95 的相应概念一样。用户按下某个键后，经过一定时间键盘会自动重复该键，这一时间就是键盘的重复延迟时间。重复率则指键盘的重复速率。如果重复速率太高，用户按某个键时就可能获得比意想的输入多的键输入。默认情况下，PC 键盘的重复延迟设置为 250 ms，而重复延迟则设置为 10.9 次每秒（cps）。利用 **kbdrate** 命令可修改键盘的这两个物理参数，例如：

```
$ kbdrate -r 12.0  d 500
```

将重复率设置为 12.0 cps，而重复延迟设置为 500 ms。不带任何参数时，**kbdrate** 命令将这两个参数设置为默认参数。

在 X Window 下，也可以利用上述命令修改这两个参数。除此之外，X Window 还提供了 **xset** 命令，可利用该命令取消键盘的重复功能，例如：

```
$ xset r off
```

会取消重复功能，而利用

```
$ xset r on
```

会打开重复功能。但是，**xset** 命令不能设置键盘的重复率。

12.11.3 Linux 中的键盘映射

对于英语来说，键盘上的字母键直接和英语字母表中的字母对应，但是对于非英语的语种来说，情况就不太一样了。例如，德语中的“ß”字母就没有直接的键和它对应，为此，Linux 提供“键盘映射”或“键盘翻译”，利用键盘映射可将某些键转换为特殊键。

前面提到，X Window 直接处理了键盘的输入输出端口，因此，在 Linux 虚拟控制台下和 X Window 下使用不同的键盘映射方法。在 Linux 虚拟控制台上，可利用 **loadkeys** 命令将特殊按键映射为特殊字符；而在 X Window 中，必须使用 **xmodmap** 命

令完成键盘映射。这些命令均按照字符映射表文件（文本文件）中的规定完成相应的转换。在 X Window 启动时，它会参考 Linux 文本模式下的字符映射表，因此可获得某些一致的键映射。

字符映射表文件保存在 /usr/lib/kbd/keytables 目录下，defkeymap.map 是默认的字符映射表文件。利用命令：

```
$ loadkeys fr.map
```

可装入 fr.map 所规定的字符映射表。这时，按下“.”会显示“:”。命令

```
$ loadkeys -d
```

可装入默认字符映射表。对于非默认的键盘映射，可在启动时在 shell 脚本中装入特殊的映射表。

对 X Window 而言，它对键盘的处理过程分如下两个步骤：

1．X Server 首先将键码转换为键符号名（keysym）。文件 /usr/include/X11/keysymdef.h 中包含所有的符号名。X Server 能够区分修饰键带来的不同，因为 keysymdef.h 中区分了两种不同的键，例如对“a”和“A”，分别用“KS_a”和“KS_A”定义。

2．X Server 将键符号翻译为 ASCII 字符串。对于大多数的键来说，该字符串只包含一个字符，而对于功能键等特殊按键来说，则包含多个字符。例如，F5 键对应的默认 ASCII 字符串为 “5~”。

利用 **xmodmap** 工具可修改键盘和键符号名之间的对应关系。例如，X Window 中“A”的键码为 30，而“Q”的键码为 16。如果建立文件 maptest：

```
keycode 38 = A
keycode 24 = Q
```

在 **xterm** 中运行

```
$ xmodmap maptest
```

之后，将发现“A”键和“Q”键交换了过来。

上述的 maptest 文件实际就是一个简单的 X Window 映射文件。但需要注意的是，文件中的 keycode 和 Linux 内核对键值的定义是不一样的，一般而言，X Window 中的键码要比内核的键值大 8。利用 **showkey** 命令可以查看内核对键值的定义。例如，运行 **showkey** 并前后按下“A”和“Q”后，程序的输出为：

```
$ showkey
kb mode was RAW
...
keycode 30 press
keycode 30 press
keycode 16 press
keycode 16 press
```

该程序给出的是“A”和“Q”的内核键值。

在 XF86Config 文件中，Keyboard 段用来指定键盘参数，一般而言，这些参数不需要特殊设置：

```
Section "Keyboard"
```

```
    Protocol    "Standard"
    AutoRepeat  500 5
EndSection
```

12.11.4 鼠标接口

当前 PC 中最常见的鼠标接口是串行接口、PS/2 辅助设备端口以及 Microsoft 总线鼠标等。另外，还有一些不太常见的鼠标接口，它们有些是上面这些接口的变种，有些则依赖于上面的接口。表 12-8 列出了这些鼠标接口。

表 12-8 常见鼠标接口

接口类型	说明	设备名称
串行接口	这种鼠标象调制解调器一样连接到 PC 的串行端口。Microsoft、Logitech 鼠标也具备串行接口。	/dev/ttyS0 /dev/ttyS1
PS/2 辅助设备端口	这种鼠标通过键盘控制器上的辅助设备端口连接。	/dev/psaux
Microsoft 总线鼠标	这种鼠标连接插在 PC 主板的接口卡上。	/dev/inportbm
Logitech 总线鼠标	这种鼠标和 Microsoft 总线鼠标类似，但所遵循的协议不同。	/dev/logibm
ATI-XL 总线鼠标	这是 Microsoft 总线鼠标的变种，它直接连接到 ATI-XL 显示卡上。	/dev/atibm
其他	一些内置在笔记本电脑中的接口，一般而言，这种接口依赖于 PS/2 接口进行连接。	

所有的鼠标均需要占用中断请求线 IRQ，某些还需要一些 I/O 端口地址。串行端口 COM1 和 COM2 默认分别使用 IRQ4 和 IRQ3；总线鼠标默认使用 IRQ5，有时可能会造成和声卡、SCSI 控制器之间的中断冲突。PS/2 辅助端口总使用 IRQ12，该设置是无法改变的。

12.11.5 鼠标设备名称

和键盘以及调制解调器一样，鼠标设备也是字符设备。对于不同的鼠标接口，Linux 使用不同的特殊设备文件名称，见表 12-8。

为方便起见，利用 /dev/mouse 作为通用的鼠标设备，这一设备文件通过符号连接指向实际的鼠标设备。例如，在某笔记本电脑中，利用如下命令可看到符号连接情况：

```
$ ls -l /dev/mouse
lrwxrwxrwx      1    root     root     10 Sep 4 21:08 mouse -> /dev/psaux
```

利用这种约定，运行在使用不同鼠标接口的计算机中的程序均可以利用 /dev/mouse 设备文件访问鼠标。

12.11.6 鼠标协议

鼠标协议指鼠标用来包装鼠标移动和按钮状态信息的约定。鼠标协议由鼠标数据的接受者使用，用来解释鼠标的移动和按钮状态。鼠标协议对 XFree86 来说非常重要，因为 X Server 必须能够解释和使用鼠标数据，以便可以在屏幕上绘制鼠标的移动，并响应鼠标

的按钮事件。表 12-9 给出了常见的鼠标协议。

表 12-9 常见鼠标协议

协议名称	对应的鼠标类型
BusMouse	Microsoft 和 Logitech 总线鼠标。
Logitech	老的 Logitech 串行鼠标。新的 Logitech 鼠标使用 Microsoft 或 Mouseman 协议。
Microsoft	Microsoft 或其他串行鼠标。
MMSeries	MMSerics 串行鼠标。
Mouseman	Logitech Mouseman 鼠标。
MouseSystems	MouseSystems 鼠标。
PS/2	所有连接到 PS/2 辅助端口的鼠标。
MMHitTab	MouseMan HitTablet。
Xqueue	只有在 keyboard 协议也使用 Xqueue 时，才使用该协议。

12.11.7 鼠标和 XFree86

在 XFree86 中，鼠标数据的接受者是 X Server。X Server 至少需要了解如下信息：

- 鼠标的设备文件名称。可以指定为 /dev/mouse，也可以直接指定为相应的设备文件。
- 鼠标为发送移动和按钮状态信息而使用的协议。

这些鼠标参数，在 /etc/X11/XF86Config 文件的“Pointer”段中指定。除此之外，还需要设置一些其他信息。例如，对 Logitech 串行鼠标，还需要指定 BaudRate 和 SampleRate 参数，分别代表波特率和采样频率。如果鼠标为两键鼠标，还可以指定模拟三键鼠标，例如：

```
Emulate3Buttons
```

对于 MouseSystems 鼠标，需要在 Pointer 段中指定如下参数：

```
ClearDTR
ClearRTS
```

某些三键鼠标，当单击中间键时，会发送一个左右键的同时单击信息，可以利用如下参数关闭这一功能：

```
ChordMiddle
```

典型的鼠标 Pointer 段如下所示：

```
Section "Pointer"
    Protocol    "Microsoft"
    Device      "/dev/mouse"
    Emulate3Button
EndSection
```

12.12 打印机

11.12.1 打印机及其设备文件

典型的 PC 具有一个并行端口和两个串行端口，大多数打印机连接在计算机的并行端口上。在 Linux 中，并行端口的设备名称和设备号如表 12-10 所示。

表 12-10 Linux 中的并行端口设备名称和设备号

I/O 端口	设备名称	主设备号	次设备号
0x3bc	/dev/lp0	6	0
0x378	/dev/lp1	6	1
0x278	/dev/lp2	6	2

在大多数 PC 机中，LPT1 端口使用的 I/O 地址为 0x378，因此，一般的并行设备名称为 /dev/lp1。

12.12.2 假脱机和打印作业

假脱机指能够在后台进行打印的能力。Linux 中的打印环境由许多程序组成，它们也支持假脱机。假脱机目录是包含打印文件的目录。

打印任务指利用单个打印命令要求打印的内容。Linux 中的打印程序在假脱机目录中排列各个打印任务，然后由后台进程周期性地将打印作业从假脱机目录中按顺序发送到打印机。

12.12.3 打印作业控制

在 Linux 中，可以利用命令 **lpr** 将打印作业放置在打印作业队列中等待打印。例如，可用如下的命令打印文件 test.txt：

```
$ lpr test.txt
```

lpr 命令将 test.txt 文件复制到假脱机目录（一般处于 /var/spool 目录）中，然后由 **lpd** 后台进程将文件发送到打印机。如果要打印的文件非常大，则可以利用 **lpr** 命令的 -s 参数，建立一个指向欲打印文件的符号连接，这样，就不需要进行复制。

利用 **lpr** 的 -p 参数，可以指定打印机类型。例如：

```
$ lpr -Phplj test.txt
```

指定打印机名称为 hplj，该名称实际定义在 /etc/printcap 文件中。也可以利用 PRINTER 这一 shell 环境变量指定默认的打印机。

利用 **lpq** 命令可检验打印队列。例如：

```
$ lpq
lp is ready and printing
Rank    Owner   Job Files                 Total Size
active  root    1   test.txt              14186 bytes
1st     WeiYM   2   support.c             42644 bytes
```

Rank 列中的 active 表明该打印作业是正在打印的打印作业。

利用 **lprm** 命令可取消打印作业。利用该命令时，需要指定打印作业的编号。例如：

```
$ lprm 1
dfA001Aa00235 dequeued
cfA001Aa00235 dequeued
```

利用 **lprm -** 命令可取消所有的打印作业。

利用 **lpc status** 命令可查看打印机状态。例如：

```
$ lpc status
lp:
    queuing is enabled
    printing is enabled
    no entries
    no daemon present
```

如果打印机支持 PostScript，则可以直接输出 PostScript 文件。

12.12.4 Linux 的打印原理

在 MS-DOS 中，我们经常利用 **copy** 命令直接向打印机端口输出要打印的文件。在 Linux 中，如果登录为 root，并且保证打印机工作正常，也可以利用类似的命令直接输出到打印机：

```
$ cat test.txt > /dev/lp1
```

但是，上述命令在多用户和多任务系统中是非常有害的。在多任务系统中，通常利用假脱机方式打印。在 Linux 中，每个打印机对应有一个假脱机目录，每个要打印到该打印机的打印作业保存在对应的假脱机目录中，每个文件对应一个打印作业。一个后台进程，即打印守护进程 **lpd**，周期性地检验假脱机目录中的新文件，如果发现有新文件，则负责发送到打印机。打印作业实际在假脱机目录中按队列排队，因此，多个打印作业形成了一个打印队列。

如上所述，我们可以使用 **lpd**、**lpr**、**lpq**、**lprm** 以及 **lpc** 等命令操作或查看打印队列。除此之外，在 Linux 的打印处理中，/etc/printcapp 文件也扮演着重要的角色。

如前所述，利用 **lpc** 命令可以查看打印机状态，实际上，该命令是一个交互式命令，可以利用一些预先定义的命令控制打印机。例如：

```
$ lpc
lpc> help
Commands may be abbreviated. Commands are:

abort   enable  disable help    restart status  topq    ?
clean   exit    down          quit    start       stop    up
lpc> stop hplj
...
lpc> q
$
```

表 12-11 列出了常用的 **lpc** 命令。

表 12-11 常用的 lpc 命令

命令	说明	使用许可
restart *printer-name*	重新启动打印机守护进程（lpd）。可使用 all 指代所有的打印机。	任何人
status *printer-name*	显示指定打印机的状态。如果不指定打印机名称，则显示默认打印机（lp）的状态。	任何人
help *command-name*	显示帮助信息。	任何人
Exit	退出 lpc（仅能用于交互式方式）	任何人
Quit	退出 lpc（仅能用于交互式方式）	任何人
abort *printer-name*	和 stop 命令类似，但不等待当前作业完成。当打印重新开始时，当前作业继续。	root
clean *printer-name*	从打印机队列中清除所有的作业，包括活动作业。	root
disable *printer-name*	关闭指定打印机的打印作业假脱机。之后，用户无法利用 lpr 命令打印。	root
down *printer-name message*	关闭假脱机并停止打印机守护进程，效果等同于 disable 和 stop 命令的组合使用。之后，用户运行 lpq 命令时，将显示 *message*。	root
enable *printer-name*	打开指定打印机的打印作业假脱机。	root
start *printer-name*	打开打印机守护进程。之后，守护进程就可以打印指定打印机的假脱机目录中的任何作业。	root
stop *printer-name*	等待当前打印作业完成后，关闭打印机守护进程。之后，守护进程停止打印假脱机目录中的作业。打印停止后，用户仍然可以使用 lpr 提交打印作业，但只有利用 start 命令打开打印机守护进程之后才能打印这些作业。	root
topq *printer-name job-id*	将指定的打印作业挪到打印机队列的前面。如果使用用户名指定 *job-id*，则所有该用户的打印作业被提前。	root
up *printer-name*	down 命令的反操作，该命令打开假脱机并启动打印机守护进程，等同于 enable 和 start 命令的组合使用。	root

/etc/printcap 文件是 Linux 打印环境的核心。该文本文件描述了不同打印机的打印能力。下面是 /etc/printcap 文件定义一个 HP LaserJet 打印机的例子：

```
# HP LaserJet printer
lp|hplj|Laserjet-ADDL|HP Laserjet 4M in Advanced Development Lab:\
   :lp=/dev/lp1:\
   :sd=/usr/spool/lp1:\
   :mx#0:\
   :if=/usr/spool/lp1/hplj-if.pl:\
   :lf=/usr/spool/lp1/hplj-lcg:\
   :sh:
```

上面的例子定义了如下内容：

- 将打印作业发送到设备 /dev/lp1（lp 域）。
- 在 /usr/spool/lp1 目录保存排队的打印作业（sd 域）。
- 不限制打印作业的大小（mx 域）。
- 通过 /usr/spool/lp1/hplj-if.pl 筛选器传递打印文件（if 域），筛选

器等同于 Windows 95 中的打印机驱动程序。

- 在文件 /usr/spool/lp1/hplj-log 中保存错误日志（lf 域）。
- 将打印作业发送到设备 /dev/lp1（lp 域）。
- 打印作业间的隔离页（sh 域）。

其他用来定义打印机的域可参见有关文档。

12.13 其他外设

Linux 能够支持的外设多种多样，除了上面的这些外设外，还有声卡、调制解调器、网络设备、PCMCIA 卡等。和这些设备相关的安装和配置问题可参阅有关文档。

第十三章

文 件 系 统

Linux 操作系统的一个重要特点是它支持许多不同类型的文件系统。Linux 中最普遍使用的文件系统是 Ext2，但 Linux 也能够支持 FAT、VFAT、FAT32、MINIX 等不同类型的文件系统。

磁盘在经过分区之后，单个的物理磁盘就被划分为多个逻辑分区，每个分区上可存在一个文件系统。我们知道，块设备实际是可以包含文件系统的设备，不管块设备的具体构造如何，Linux 文件系统均将它们当作线性块的集合而访问。每个块设备驱动程序的任务，就是将逻辑块的请求转化为对应块设备可以理解的命令。类似地，对文件系统来说，不管基础的块设备是什么，它都能够进行一致的操作。利用 Linux 的文件系统，用户根本不用关心由不同硬件控制器控制的，处于不同物理介质上的不同文件系统之间的区别。如果将网络服务器的磁盘或分区挂装到本地文件系统上，用户可以和操作本地文件系统一样操作它们。

文件系统中的文件实际是数据的集合。文件系统不仅包含了这些文件中的数据，而且还包含文件系统的结构、符号链接以及文件的安全保护等信息。

MINIX 是 Linux 最初使用的文件系统，但是由于 MINIX 文件系统实际是教学用操作系统 MINIX 的文件系统，因此其功能尚不完备，并且性能欠佳。因此，1992 年 4 月份，专为 Linux 的设计的第一个文件系统 EXT（Extended File System）应运而生，但是，EXT 文件系统仍然有些粗糙，而且性能也同样是个问题。1993 年，在许多人的努力下，EXT 2 （Second Extended File System）文件系统作为 EXT 文件系统的兄弟诞生了。EXT 2 完全具备了高级文件系统的特点，性能也非常好。

在 EXT 文件系统的开发过程中，引入了一个非常重要的概念，即虚拟文件系统（VFS）。VFS 作为实际文件系统（EXT）和操作系统之间的接口，将实际的文件系统和操作系统隔离开来。在 VFS 的帮助下，Linux 可以支持除 EXT 之外的许多文件系统类型。各文件系统为 VFS 提供一致的接口，从而将不同文件系统的细节隐藏了起来。对操作系统的其他部分，以及运行在操作系统之上的程序而言，所有的文件系统都是一样的。

对 VFS 来说，它一方面要保证快速访问实际文件系统中的数据，一方面还要保证文件和数据能够正确保存。这两个方面实际是互相矛盾的，Linux 通过高速缓存协调这两个需求。在高速缓存中，Linux 不仅缓存数据，而且还管理着操作系统和块设备之间的接口。

本章主要讲述 Linux 的 Ext2 文件系统、虚拟文件系统以及缓冲区高速缓存。

13.1 Ext2 文件系统

Ext2 作为 Linux 可扩展的、功能强大的文件系统而设计，其发明者为 Remy Card。在 Linux 中，Ext2 是迄今为止最为成功的文件系统，而且作为所有 Linux 发行版本的基本文件系统。

和其他文件系统一样，Ext2 文件系统中的文件保存在数据块中。对同一个 Ext2 文件系统而言，其上所有的数据块大小都是一样的。和某些文件系统不同的是，不同的 Ext2 文件系统的数据块大小可以不一样，数据块大小实际在建立 Ext2 文件系统时指定，并且作为文件系统的基本参数保存在文件系统中。单个文件所占用的空间按数据块为单位分配，因此，如果某个文件的大小为 1025 字节，而数据块的大小为 1024 字节，则该文件将占用两个数据块。平均而言，每个文件会浪费半个数据块的空间。但是，鱼和熊掌不可兼得，我们为了提高文件系统的访问速度，为了文件系统便于管理，而不得不浪费一些磁盘空间。

块设备上的大多数数据块用来保存文件的实际数据，而部分数据块则用来定义文件系统的结构。Ext2 文件系统利用索引节点（inode）来描述文件系统的拓扑结构。在单个文件系统中，每个文件对应一个索引节点，而每个索引节点有一个唯一的整数标识符。文件系统中所有文件的索引节点保存在索引节点表中。Ext2 文件系统中的目录实际是一种特殊文件，它们也有对应的索引节点，索引节点指向的数据块中包含该目录中所有的目录项（文件、目录、符号链接等），每个目录项对应自己的索引节点。

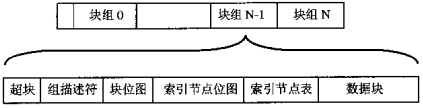


图 13-1 Ext2 文件系统的物理布局

如图 13-1 所示，EXT 2 文件系统分布在块结构的设备上。文件系统不必了解数据块的物理存储位置，它保存的是逻辑块的编号。实际上，由块设备驱动程序完成逻辑块编号到物理存储位置的转换。Ext2 文件系统将逻辑分区划分为块组（Block Group）。每个块组重复保存着对文件系统的完整性非常关键的信息，而同时也用来保存实际的文件和目录数据。文件系统关键信息的重复存储有助于文件系统在发生故障时还原。

13.1.1 Ext2 索引节点

在 Ext2 文件系统中，索引节点是基本的数据块，每个文件和目录有且只有一个索引节点与之对应。每个块组中的索引节点保存在索引节点表中，同时也保存有跟踪节点分配情况的位图。图 13-2 是 Ext2 索引节点的结构图，其中的各个数据域在表 13-1 中描述。

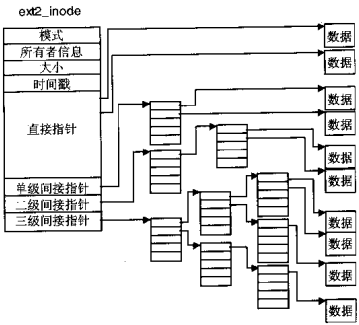


图 13-2　Ext2 索引节点

表 13-1　Ext2 索引节点数据域的描述

模式	该数据域包含索引节点所描述的对象类型，以及用户的许可信息。在 Ext2 中，每个节点可描述一个文件、目录、符号链接、块设备、字符设备或一个 FIFO。
所有者信息	文件或目录所有者的用户和组标识符。
大小	文件以字节为单位的大小。
时间戳	该索引节点的创建时间以及索引节点的最后改动时间。
数据块	包含该索引节点所描述的数据的数据块指针。对于前 12 个数据块指针来说，它们指向的数据块是包含实际文件数据的数据块，而后面的三个指针则包含间接数据块指针。

这里需要对上表中的数据块信息作进一步解释。前 12 个数据块指针直接指向包含文件数据的数据块，而其后的三个数据块则是间接指针。这样的安排是为了适应文件的大小变化。假如文件系统可保存的最大文件为 4MB，而每个数据块的大小为 1024 字节。如果索引节点全部利用直接数据块指针的话，则需要 4096 个数据块指针，而实际上，文件系统中大部分的文件不可能有如此之大。如果要保存 1 MB 的文件，实际只需 1024 个数据块指针，因此大量的数据块指针被浪费了。

为了解决上述问题，多数利用索引节点的文件系统利用多级数据块指针。参见图 13-2，假设数据块大小为 1024 字节。利用前 12 个直接指针，可保存最大为 12 KB 的文件，对这些文件的访问将非常迅速。如果文件大小超过了 12 KB，则利用单级间接指针，这一指针指向的数据块保存有一组数据块指针，这些指针依次指向包含实际数据的数据块。如果每个数据块指针占用 4 个字节，则每个单级间接指针数据块可包含 1024 / 4 = 256

个数据块指针，因此，利用直接指针和单级间接指针可保存 $12 \times 1024 + 256 \times 1024 = 268$ KB 的文件。当文件超过 268 K 字节时，再利用二级间接指针。类似地，可计算利用直接指针、单级间接指针和二级间接指针时可保存的最大文件为：

```
12 × 1024 + 256 × 1024 + 256 × 256 × 1024 = 65804 KB
```

约为 65 MB 大小。依次类推，得出利用三级间接指针时，可保存的最大文件为 16842764 KB ≈ 16 GB。

13.1.2 Ext2 文件系统的超块

文件系统的超块（Superblock）包含了文件系统的基本大小和形式。其中包含的数据由文件系统管理程序用来进行文件系统的维护。每个块组中包含有相同的超块信息，但通常只需读取块组 0 的超块。超块中包含的信息在表 13-2 中列出。

表 13-2 Ext2 超块中的信息

幻数	幻数用来标识超块的类型。文件系统的挂装软件可据此判断是否为有效的 Ext2 文件系统超块。当前的 Ext2 文件系统幻数为 *0xEF53*。
修订级别	包含两个数据，分别为主修订级别和次修订级别。文件系统的挂装软件可据此判断文件系统是否支持某种特定的功能。实际上，该信息作为兼容性标志而存在。
挂装计数和最大挂装计数	利用这两个信息，系统可以判断文件系统是否需要彻底检验。文件系统每挂装一次，挂装计数加 1。如果挂装计数达到最大挂装计数，则系统会提示应进行检查。
组块编号	保存该超块的组块编号。
数据块大小	文件系统的数据块大小，以字节为单位。
每组的数据块个数	每组的数据块个数，该值在建立文件系统时设置并保持不变。
空闲数据块	文件系统中空闲数据块的个数。
空闲索引节点	文件系统中空闲索引节点的个数。
首索引节点	文件系统第一个索引节点的编号。一般而言，Ext2 文件系统的首索引节点就是"/"目录的目录项。

13.1.3 Ext2 块组描述符

每个块组中包含有一个数据结构描述该块组。和超块类似，所有块组的块组描述符重复出现在所有的块组中。每个块组描述符包含的信息如表 13-3 所示。

表 13-3 块组描述符

数据块位图	包含块组数据块分配位图的数据块编号。
索引节点位图	包含块组索引节点分配位图的数据块编号。
索引节点表	包含块组索引节点表的起始数据块编号。
空闲块计数，空闲索引节点计数，已用目录计数	

分配位图在分配或释放数据块、索引节点时使用。

块组描述符保存在每个块组超块的后面，每个块组中包含所有块组的描述符。块组 0 中包含的超块信息和块组描述符是实际使用的数据，由于文件系统被破坏而需要还原时使用其他块组中重复的超块和块组描述符。

13.1.4 Ext2 目录

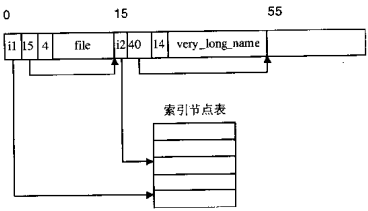


图 13-3 目录项的布局

如图 13-3 所示是内存中的目录项布局。在 Ext2 文件系统中，目录实际是特殊文件。目录文件中包含了目录项的列表，而每个目录项包含的信息如表 13-4 所示。

表 13-4 目录项信息

索引节点	目录项的索引节点。
名称长度	目录项名称的长度，以字节为单位。
名称	目录项名称。

每个目录的前两个目录项始终是标准的“.”和“..”，分别代表“本目录”和“父目录”。

当用户需要打开某个文件时，首先要指定文件的路径和名称，根据路径和名称，文件系统可搜索到文件的对应索引节点，从而可读取文件中的数据。例如，假定要搜索文件 /home/WeiYM/mbox，在 Ext2 中，该文件的定位过程如下（参照图 13-4）：

1. 文件系统按照超块中根目录的索引节点，以及索引节点中的数据块编号信息，可找到根目录的目录项列表。
2. 在目录项列表中搜索名称为 home 的目录项，得到对应索引节点编号为 6。
3. 按照索引节点 6 中的数据块编号，可找到 /home 目录的目录项。
4. 在 /home 目录项列表中找出目录项 WeiYM，得到对应索引节点编号为 26。
5. 按照索引节点 26 中的数据块编号，可找到 /home/WeiYM 目录的目录项。
6. 从 /home/WeiYM 目录的目录项列表中可以获得文件 mbox 的索引节点。

获得目标文件的索引节点之后，随后就可以利用该索引节点访问文件中的数据。

13.1.5 Ext2 文件系统中数据块的分配和释放

我们知道，在频繁的文件创建和删除过程中，极容易使文件系统碎片化。过分碎片化的文件系统中，逻辑上连续的数据块在物理上处于不连续的位置，从而导致文件的存取速度减慢，最终影响系统性能。Ext2 文件系统通过为文件分配物理上相近的数据块，或者至少让数据块保持在同一块组而避免文件系统的过分碎片化。只有在不得已时，Ext2 文件系统才为单个文件在不同的块组中分配数据块。

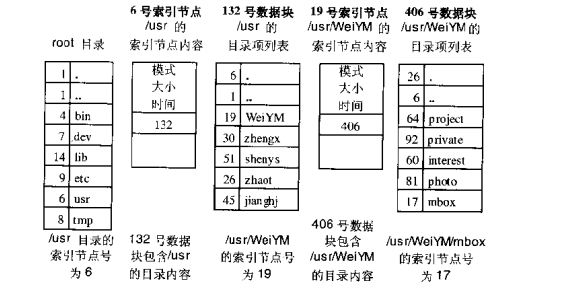


图 13-4 文件的定位过程

当某个进程试图在文件中写入数据时，Linux 文件系统首先检查要写入的数据是否已超过文件的末尾，如果已超过，则文件系统要分配新的数据块。在文件系统为该文件分配数据块并将数据写入数据块之前，进程处于等待状态，完成实际的数据写入之后，进程才能继续运行。因为数据块的分配或释放最终要修改超块中的信息，因此，进程在分配数据块之前，首先要锁定超块，这样，其他要写数据的进程也必须等待当前写数据的进程结束写过程。对超块的访问依照“先到先服务”的排队原则。

锁定超块之后，进程首先检查是否有足够的空闲数据块，如果没有足够的空闲块，则数据的写入就要失败，进程最终会放弃对文件系统超块的锁定控制。

如果文件系统中有足够的空闲空间，则系统尝试分配新的数据块。如果 Ext2 文件系统具备数据块的预分配功能，则只需从预分配块中获取一个数据块。预分配块实际是在已分配数据块位图中预先保留的数据块。代表要分配数据块的文件的 VFS 索引节点中包含两个专用于 Ext2 文件系统的数据域：prealloc_block 和 prealloc_count。这两个数据域分别代表第一个预分配数据块的编号和预分配数据块的数目。通过指定文件的预分配信息，可加速文件的数据块分配，并可保证在一定的文件大小范围内，保持文件数据块的连续性。如果没有预分配的数据块，或者预分配功能被关闭，则 Ext2 文件系统必须

分配新的数据块。Ext2 文件系统首先查看和文件最后一个数据块相邻的数据块是否空闲。逻辑上讲，这个数据块是最理想的，因为连续的数据块可保证最高的存取效率。如果相邻数据块已被占用，则搜索范围加宽到和理想块相邻的 64 个块范围之内。如果能够在相邻连续的 64 个块中找到空闲数据块，虽然该数据块并不是最理想的，但至少比较接近，且处于同一块组中。

如果和理想块相邻连续的 64 个块中没有空闲块，则文件系统在其他块组中寻找空闲块。块分配代码首先以 8 个块为单位（1 簇）分配空闲块，如果没有空闲簇，则寻找更小的簇。上述过程持续进行，直到分配到数据块为止。如果数据块的预分配被打开，则还需要更新 `prealloc_block` 和 `prealloc_count` 数据域。

不管在哪里分配到了数据块，块分配代码需要更新新块所在块组的块位图，并在缓冲区缓存中分配一个数据缓冲区。数据缓冲区由文件系统的支持设备标识符和块编号唯一标识。缓冲区中的数据被清零，而缓冲区被标志为“脏”，表明其中的数据尚未写入物理磁盘。最后，超块本身也被标志为“脏”，表明已被改动，最后解锁超块。

如果其他进程正在等待该超块解锁，则等待队列中的第一个进程可开始运行，并获得对超块的互斥访问。进程数据写入新的数据块中，如果尚有更多的数据写入，则整个分配过程再次开始。

数据块的释放过程相对直接，但仍然需要获得对超块的互斥访问。

13.2　虚拟文件系统

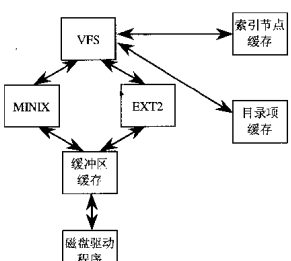


图 13-5　虚拟文件系统和实际文件系统之间的关系示意图

图 13-5 是虚拟文件系统和实际文件之间的关系示意图。虚拟文件系统必须管理所有可在任意时刻挂装的不同文件系统。为此，Linux 虚拟文件系统维护一些描述整个虚拟文

件系统，以及实际已挂装文件系统的数据结构。虚拟文件系统对文件的描述方法和 Ext2 文件系统类似，也使用了超块和索引节点。为此，有必要描述虚拟文件系统的超块和索引节点，以便能够正确区分 VFS 和 Ext2。

每个文件系统在初始化时，首先在 VFS 中进行注册。如果文件系统内建于内核中，则初始化过程发生在系统引导时；如果文件系统作为内核可装载的模块，则实际挂装某个文件系统时进行初始化。当某种基于块设备的文件系统（包括 root 文件系统）被挂装，VFS 必须读取其超块。不同类型的文件系统所对应的超块读取例程必须能够理解实际文件系统的拓扑结构，并且能够将实际的超块结构映射为 VFS 超块结构。每个 VFS 超块包含了文件系统信息，并且还包含一些完成特定功能的函数指针。例如，某个超块代表一个已挂装的 Ext2 文件系统，则超块中包含有专门读取 Ext2 文件系统索引节点的函数地址。每个 VFS 超块中包含指向实际文件系统第一个 VFS 索引节点的指针。对于 root 文件系统而言，第一个索引节点就是代表“/”目录的节点。对 Ext2 文件系统来说，这种映射关系非常高效，而对其他文件系统则略微有些低效。

进程在访问目录和文件的过程中，会调用系统例程对 VFS 节点进行遍历。因为每个文件和目录均由一个索引节点表示，因此，有相当多的索引节点会被重复访问。由于这一原因，为了提高索引节点的访问速度，VFS 将这些节点保存在索引节点高速缓存中。如果某个节点当前不在高速缓存中，则调用专用于某种文件系统的索引节点读取例程，以便读取适当的索引节点。读取的索引节点会保存在高速缓存中，而较少使用的 VFS 索引节点会从高速缓存中剔除。

VFS 同时维护一个目录高速缓存，以便能够快速找到频繁使用的目录索引节点。目录高速缓存并不保存目录本身的索引节点，这些索引节点应当保存在索引节点高速缓存中；目录高速缓存实际保存的是完整目录名到对应索引节点编号的映射关系。

所有的 Linux 文件系统使用共同的缓冲区高速缓存，所有的文件系统以及 Linux 内核使用同一个缓冲区高速缓存。缓冲区高速缓存不依赖于任何文件系统，它和 Linux 内核用来分配、读取和写入数据缓冲区的机制集成在一起。Linux 文件系统独立于底层介质以及设备驱动程序而存在，这种机制具有非常明显的优点。所有的块设备在 Linux 内核中注册，并且提供一致的、基于块的、异步的接口。当文件系统从物理介质中读取数据时，实际是向控制该设备的驱动程序发送读取物理数据块的请求。缓冲区高速缓存负责将块设备驱动程序的接口集成在一起。文件系统在读取数据块时，这些块被保存在全局的缓冲区高速缓存中。缓冲区高速缓存中的缓冲区由唯一的设备标识符以及块编号标识，当进程频繁访问相同的数据时，这些数据就可以直接从高速缓存中读出，而不必进行实际的磁盘读取操作。

13.2.1 VFS 超块

每个已挂装的文件系统由一个 VFS 超块代表。VFS 超块中包含的信息由表 13-5 列出。

表 13-5 VFS 超块中的信息

设备	包含文件系统的块设备标识符。对于 /dev/hda1，其设备标识符为 *0x301*。
索引节点指针	这里包含两个索引节点指针。mounted 索引节点指针指向文件系统的第一

	个节点；covered 指针指向代表文件系统挂装目录的节点。root 文件系统的 VFS 超块没有 covered 指针。
数据块大小	文件系统中数据块的大小，以字节为单位。
超块操作集	指向一组超块操作例程集的指针。这些例程由 VFS 用来读取和写入索引节点以及超块。
文件系统类型	指向文件系统的 file_system_type 数据结构的指针。
文件系统的特殊信息	该文件系统的特殊信息。

13.2.2 VFS 索引节点

和 Ext2 文件系统一样，VFS 中的文件、目录等均由对应的索引节点代表。每个 VFS 索引节点中的内容由文件系统专属的例程提供。VFS 索引节点只存在于内核内存中，实际保存于 VFS 的索引节点高速缓存中。VFS 索引节点中的信息由表 12-6 给出。

表 12-6 VFS 索引节点中的数据域

设备	包含该文件或其他任何 VFS 索引节点所代表的对象的设备标识符。
索引节点编号	索引节点编号，在文件系统中唯一。设备和索引节点的组合在 VFS 中唯一。
模式	VFS 索引节点代表的对象类型，以及相应的访问权限。
用户标识符	所有者标识符。
时间	建立、修改和写入的时间。
块大小	数据块的大小，以字节为单位。
索引节点操作集	指向索引节点操作例程集的指针。这些例程专用于该文件系统，用来执行针对该节点的操作。
计数	系统组件使用该 VFS 节点的次数。计数为零，表明该节点可丢弃或重新使用。
锁定	该数据域用来锁定 VFS 节点。
脏	表明 VFS 节点是否被修改过，若如此，该节点应当写入底层文件系统。
文件系统的特殊信息	和文件系统相关的特殊信息。

13.2.3 文件系统的注册

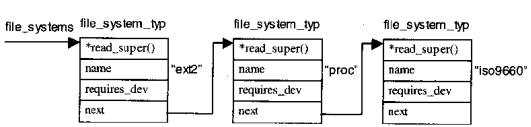


图 13-6 注册的文件系统

如前所述，对特定文件系统的支持可内建于 Linux 内核之中，也可以以可装载模块

的形式动态装载。如果内建于内核，则系统启动时进行文件系统的注册和初始化，否则在模块装载时注册，而在模块卸载时注销。每个文件系统的初始化例程在 VFS 中注册，并由一个 file_system_type 数据结构代表，其中包含了文件系统的名称以及一个指向对应 VFS 超块读取例程的地址。图 13-6 所示是内核中的 file_system_type 链表，链表头由 file_systems 指定。每个 file_system_type 结构成员由表 13-7 给出。

表 13-7 file_system_type 结构成员

超块读取例程 read_super	当属于该文件系统类型的实际文件系统被挂装时，VFS 调用该例程读取超块。
文件系统名称 name	文件系统的名称。例如 Ext2、vfat、iso9660 等。
要求设备 requires_dev	该文件系统是否需要设备支持？并不是所有的文件系统都需要特定设备，例如，/proc 文件系统并不需要块设备。

查看 /proc 文件系统的 filesystems 文件内容，可了解当前注册的文件系统类型：

```
$ cat /proc/filesystems
        Ext2
        msdos
nodev   proc
```

13.2.4 文件系统的挂装和卸装

用户（一般是 root）在挂装文件系统时，要指定三种信息：文件系统的名称、包含文件系统的物理块设备和文件系统在已有文件系统中的挂装点。例如：

```
$ mount -t iso9660 /dev/hdc /mnt/cdrom
```

虚拟文件系统对上述命令的执行过程如下：

1. 寻找对应的文件系统信息。VFS 通过 file_systems 在 file_system_type 组成的链表中根据指定的文件系统名称搜索文件系统类型信息。
2. 如果在上述链表中找到匹配的文件系统，则说明内核具有对该文件系统的内建支持。否则，说明该文件系统可能由可装载模块支持，VFS 会请求内核装入相应的文件系统模块，此时，该文件系统在 VFS 中注册并初始化。
3. 不管是哪种情况，如果 VFS 无法找到指定的文件系统，则返回错误。
4. VFS 检验给定的物理块设备是否已经挂装。如果指定的块设备已被挂装，则返回错误。
5. VFS 查找作为新文件系统挂装点的目录的 VFS 索引节点。该 VFS 索引节点可能在索引节点高速缓存中，也有可能需要从挂装点所在的块设备中读取。
6. 如果该挂装点已经挂装有其他文件系统，则返回错误。因为同一目录只能同时挂装一个文件系统。
7. VFS 挂装代码为新的文件系统分配超块，并将挂装信息传递给该文件系统的超

块读取例程。系统中所有的 VFS 超块保存在由 super_blocks 指向的 super_block 数据结构指针数组中。

8. 文件系统的超块读取例程将对应文件系统的信息映射到 VFS 超块中。如果在此过程中发生错误，例如所读取的超块幻数和指定的文件系统不一致，则返回错误。
9. 如果成功挂装，则所有已挂装的文件系统形成如图 13-7 所示的数据结构。

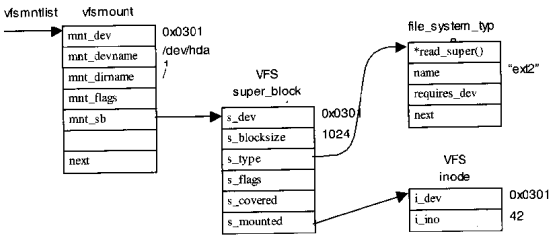


图 13-7 已挂装的文件系统数据结构

从图 13-7 中可看到，每个已挂装的文件系统由 vfsmount 结构描述，所有的 vfsmount 结构形成了一个链表，该链表头由 vfsmntlist 指针代表。系统中还有两个指针，vfsmnttail 和 mru_vfsmnt，分别指向链表尾和最近使用过的 vfsmount 结构。每个 vfsmount 结构包含保存该文件系统的块设备号，文件系统挂装点的目录名称，以及指向为该文件系统分配的 VFS 超块的指针。而 VFS 超块中则包含描述文件系统的 file_system_type 结构指针和该文件系统根节点指针。

如果文件系统中的文件当前正在使用，该文件系统是不能卸装的。如果文件系统中的文件或目录正在使用，则 VFS 索引节点高速缓存中可能包含相应的 VFS 索引节点。检查代码在索引节点高速缓存中，根据文件系统所在设备的标识符，查找是否有来自该文件系统的 VFS 索引节点，如果有且标志为“脏”，则说明该文件系统正在被使用，因此，文件系统不能被卸装。否则，查看对应的 VFS 超块。如果该文件系统的 VFS 超块标志为“脏”，则必须将超块信息写入实际的文件系统。上述过程结束之后，对应的 VFS 超块被释放，vfsmount 数据结构从 vfsmntlist 链表中断开并释放。

13.2.5 VFS 中文件的定位

为了在虚拟文件系统中定位给定文件的索引节点，VFS 必须每次分解文件路径名中的一个目录名，并依次搜索每个中间目录名的 VFS 索引节点。因为 VFS 在 VFS 超块中记录了每个文件系统根的 VFS 索引节点，因此搜索过程从根索引节点开始，具体过程

和 Ext2 文件系统的文件定位过程类似。在搜索目录索引节点时，VFS 在目录高速缓存中搜索，如果能够在目录高速缓存中找到对应的索引节点编号，则在索引节点高速缓存中搜索索引节点，否则从底层文件系统中获取。

13.2.6 VFS 索引节点高速缓存

VFS 索引节点实际是一个哈希表（或散列表），哈希表的每个入口包含一个 VFS 索引节点链表的头指针，该链表中的每个 VFS 索引节点具有相同的哈希值。节点的哈希值根据文件系统所在块设备的标识符，以及索引节点的编号计算得到。当虚拟文件系统要访问某个节点时，它首先在 VFS 索引节点中搜索。每个要搜索的 VFS 索引节点可计算得到一个对应的哈希值，然后，VFS 将该哈希值作为访问哈希表的索引，如果对应的哈希表入口指向的索引节点链表中包含有要搜索的索引节点（相同的设备标识符和相同的节点编号），则说明该索引节点包含在高速缓存中，这时，找到的索引节点访问计数加 1，否则，必须寻找一个空闲的 VFS 索引节点，并从底层文件系统中读取该索引节点。VFS 可选择多种方法获取空闲索引节点。如果系统能够分配更多的 VFS 节点，则 VFS 从内核中分配内存页并将其划分为新的空闲节点，然后放在节点列表中。系统所有的 VFS 节点保存在由 first_ionde 指向的链表中。如果系统已经具备了所有可利用的索引节点，则必须寻找一个可重复利用的索引节点。当前使用计数为 0 的索引节点可作为重复利用的节点，对文件系统的根节点来说（以及其他重要的节点），其使用计数始终大于 0，因此不会被重复使用。如果找到可重复利用的索引节点，则必须在使用之前清除该节点。如果该节点标志为“脏”，则需要写回到实际的文件系统中；如果该节点被锁定，则必须等待解锁。

无论利用哪种方法获得了空闲的 VFS 节点，VFS 会调用文件系统专有的例程从底层文件系统中读取信息并填充该索引节点。在该节点被填充时，这一新的 VFS 节点的使用计数为 1，同时被锁定，以免其他进程访问该节点。填充之后，节点被解锁。

为了获得实际使用的 VFS 节点，文件系统可能要访问许多其他的节点。读取目录内容时，只有最后一个目录的索引节点是实际要使用的，但同时必须读取中间目录的索引节点。

13.2.7 VFS 目录高速缓存

为了加速对频繁目录的访问过程，VFS 维护一个目录项高速缓存。实际文件系统读取目录之后，该目录项的细节添加到目录高速缓存中。下次搜索相同的目录项时，可从高速缓存中快速得到。但是，只有名称相对较短的目录项（15 个字符长度）才被高速缓存。

目录高速缓存页由一个哈希表组成。哈希值由保存文件系统的设备号以及目录名称计算得到，该值作为访问哈希表的索引。

为了保证缓存的有效性并及时更新，VFS 保持一组“最近使用 （LRU）”目录缓存项表。当某个目录项第一次放入高速缓存时，被添加到第一级 LRU 表的后面。当高速缓存全部占用时，该操作会取代第一级 LRU 表前面的已有项。当该目录再次被访问时，该目录项自动提升到第二级目录项缓存列表的后面，同样会取代第二级缓存列表前面的二级缓存目录项。这样，处于各个缓存表前面的项实际是最近没有被访问的目录项，否则，它们应当处于各目录项列表的后面。

13.3　缓冲区高速缓存

如前所述，所有的 Linux 文件系统使用共同的缓冲区高速缓存。所有块设备的读取和写入请求通过标准的内核例程调用，以 buffer_head 数据结构的形式提交给设备驱动程序（参见本书 12.5.3 小节）。buffer_head 中包含了唯一标识块所在设备的设备标识符，以及要读取或写入的块编号。为了加速对物理块设备的访问过程，Linux 使用了缓冲区高速缓存。在高速缓存中，任意给定的时间内，存在着来自不同物理设备的数据块，而且处于不同的状态。任何用来从块设备中读取或写入块设备的块缓冲区，都要经过缓冲区高速缓存。随着时间的流逝，缓冲区占用的缓存可能让给其他更加需要缓存的缓冲区，也有可能因为频繁访问而保持在缓存中。

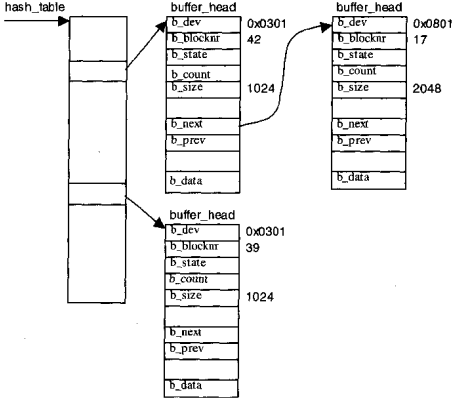


图 13-8　缓冲区高速缓存的哈希表结构

缓冲区高速缓存分为两个功能部分。第一部分是空闲块缓冲区表，每个系统支持的缓冲区大小对应一个表，系统的空闲块缓冲区在第一次创建和最后被丢弃时在这些表中排队。当前 Linux 所支持的缓冲区大小为 512、1024、2048、4096 和 8192 字节。第二部分是高速缓存本身，和 VFS 索引节点高速缓存类似，缓冲区高速缓存也是一个具有相似结构的哈希表。哈希索引由块所在的设备标识符以及块编号计算得到。图 13-8 是缓冲区

高速缓存的哈希表结构。块缓冲区可能在某个空闲缓冲区表中，也有可能在高速缓存中。如果块缓冲区在高速缓存中，则这些缓冲区也在 LRU 表中排队。每个缓冲区类型具有一个 LRU 列表，当前 Linux 支持的缓冲区类型见表 13-8。

表 13-8 Linux 的缓冲区类型

干净的	未使用的、新的缓冲区。
锁定的	当前锁定的缓冲区，正等待写入。
脏的	脏的缓冲区，其中包含新的有效数据，但尚未写入块设备。
共享的	共享的缓冲区。
非共享的	曾经被共享，但当前不再共享的缓冲区。

当文件系统需要从底层物理设备上读取数据块时，它首先试图从缓冲区高速缓存中读取。如果无法从缓冲区高速缓存中得到数据块，它会从适当大小的空闲缓冲区表中获得一个干净的缓冲区，并将该缓冲区放入缓冲区高速缓存。如果所需的缓冲区当前在高速缓存中，该缓冲区可能是经过更新的，也有可能不是。如果未经更新，或该缓冲区是新的块缓冲区，则文件系统必须请求设备驱动程序读取适当的数据块。

和其他的高速缓存一样，Linux 也需要维护缓冲区高速缓存，以便能够高效运行，并为不同块设备公平分配缓存项。为此，Linux 利用 **bdfush** 内核守护进程完成一些常务工作，而其他一些工作则随着缓存的使用而自动完成。

13.3.1 bdflush 内核守护进程

bdflush 内核守护进程是一个简单的内核线程，它可以动态地对系统中具有许多“脏”缓冲区的情形作出响应。它作为系统线程在系统启动时运行，它在系统中注册的进程名称为 **kflushd**。通常情况下，该进程处于休眠状态，在如下情况下，该进程被唤醒：

- 分配和丢弃缓冲区时，如果系统检测到脏缓冲区的比例达到指定值。默认情况下该值为 60%。
- 系统急需缓冲区时。

利用 **update** 命令，可查看并控制许多和该守护进程相关的值：

```
# update -d
bdflush version 1.4
0:    40 Max fraction of LRU list to examine for dirty blocks
1:   500 Max number of dirty blocks to write each time bdflush activated
2:    64 Num of clean buffers to be loaded onto free list by refill_freelist
3:    64 Dirty block threshold for activating bdflush in refill_freelist
4:    15 Percentage of cache to scan for free clusters
5:  3000 Time for data buffers to age before flushing
6:   500 Time for non-data (dir, bitmap, etc) buffers to age before flushing
7:  1884 Time buffer cache load average constant
8:     2 LAV ratio (used to determine threshold for buffer fratricide).
#
```

当缓冲区因为写入数据而变脏时，这些缓冲区被链接到 BUF_DIRTY LRU 表中。**bdflush** 会试图将合理数量的脏缓冲区写入磁盘，这一数量默认为 500，该参数也可以

通过 **update** 命令查看和修改。

13.3.2 update 进程

update 不仅是一个命令，而且也是一个守护进程。该守护进程一般在系统初始化时以超级用户身份运行，它周期性地把所有老的脏缓冲区刷新到磁盘上。它通过调用一个类似 **bdflush** 的系统服务完成刷新任务。当缓冲区变脏时，系统以系统时间作标记，表明该缓冲区应当刷新到磁盘。每次 **update** 运行时，它在系统的脏缓冲区中查找已到达刷新时间的缓冲区，每个到达刷新时间的缓冲区都会被写入磁盘。

13.4 /proc 文件系统

和其他文件系统不同的是，/proc 文件系统并不是一个真正存在于块设备上的文件系统。这正表明了 Linux 虚拟文件系统的强大功能。在系统的初始化过程中，/proc 文件系统在 VFS 中注册。当 VFS 需要打开其中的目录或文件时，它利用内核信息建立这些文件和目录。

/proc 文件系统为用户提供了一个查看内核内部工作情况的窗口。

13.5 特殊设备文件

和所有的 UNIX 版本一样，Linux 利用特殊文件代表系统的硬件设备。设备文件实际并不占用任何文件系统的空间，它只是作为访问设备驱动程序的入口。Ext2 文件系统和 VFS 文件系统将设备文件作为特殊索引节点实现。设备分为块设备和字符设备两种。设备文件的索引节点中包含对应设备的主设备号和次设备号，分别标识设备类型以及该设备在同类设备中的编号。

13.6 相关系统工具和系统调用

13.6.1 Linux 支持的文件系统

由于 Linux 采用了虚拟文件系统，从理论上讲，Linux 可以支持所有的文件系统。表 13-9 给出了 Linux 所支持的重要文件系统。

表 13-9 Linux 支持的重要文件系统

MINIX	MINIX 操作系统使用的文件系统，Linux 最初使用该文件系统开发。是最老的、也是最可靠的文件系统。但该文件系统的能力受到一定限制，例如，省略了某些时间戳，文件名程度不能超过 30 个字符等。
xia	MINIX 文件系统的更新版，突破了部分限制，但没有新的功能特色。
Ext2	功能强大的 Linux 土生土长的文件系统。
ext	Ext2 文件系统的老版本，该文件系统可转换到 Ext2 文件系统。
msdos	和 MS-DOS 兼容的 FAT 文件系统。

umsdos	对 Linux 中 msdos 文件系统驱动程序的扩展，以便能够支持长文件名、所有者、许可、链接和设备文件。它使得通常的 msdos 文件系统可以当作 Linux 固有的文件系统一样使用。
vfat	Microsoft 对原 fat 文件系统的扩展，可以支持长文件名。
iso9660	该文件系统是标准的 CD-ROM 文件系统。对该标准更加流行的 Rock Ridge 扩展允许长文件名的自动支持。
nfs	允许在多台计算机之间共享文件系统的网络文件系统。
hpfs	高性能文件系统，是 OS/2 的文件系统。
ntfs	Windows NT 的文件系统。
sysv	System V/386、Coherent 和 Xenix 的文件系统。

在多个操作系统共存的系统中，有时难免使用除 Ext2 之外的文件系统。如果没有这方面的需求，则应当使用 Ext2 文件系统，因为 Ext2 和 VFS 之间最接近，从而可获得最高的性能。

13.6.2 建立文件系统

利用 **mkfs** 命令建立，或初始化文件系统。实际上，每个文件系统类型对应有自己单独的初始化命令。**mkfs** 只是最为一个前台的程序而存在，它实际根据要建立的文件系统类型调用相应的命令，文件系统类型由 **mkfs** 命令的 -t 参数指定。其他常用的 **mkfs** 命令参数如下：

-c 检查坏块并建立相应的坏块清单

-l *filename* 从指定的文件 *filename* 中读取初始坏块。

如果要在软盘上建立一个 Ext2 文件系统，可利用如下的命令：

```
$ fdformat -n /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ mkfs -t Ext2 -l bad-blocks /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for Ext2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

上述命令首先利用 **fdformat** 格式化软盘，然后利用 **badblocks** 命令检查坏块，并将坏块清单重定向输出到文件 bad_blocks 中。最后，利用 **mkfs** 命令建立文件系统，bad_blocks 文件作为初始的坏块清单。

如果利用 **mkfs** 命令的 -c 参数，可省略 **badblocks** 命令。但文件系统建立之后，仍有必要利用 **badblocks** 命令进行检查。

13.6.3 文件系统的挂装和卸装

通常来讲，文件系统的挂装和卸装只能由超级用户完成。挂装命令 **mount** 可如下使用：

```
# mount -t vfat -r /dev/hda5 /mnt/dosd
#
```

上述命令指定了要挂装的文件系统类型，文件系统所在设备以及挂装点，并将文件系统挂装为只读。如果要在系统启动时自动挂装文件系统，则可以在 /etc/fstab 文件中指定。有关内容可参见 fstab 手册页。

利用 **umount** 命令卸装文件系统，该命令只需一个参数，即希望卸挂的设备或挂装点：

```
# umount /mnt/dosd
#
```

或

```
# umount /dev/hda5
#
```

13.6.4 检查文件系统的完整性

利用 **fsck** 命令可检查文件系统中存在的错误，如果需要，可强制该命令修改错误。一般而言，文件系统经过了大量严格周密的测试，因此在正常使用情况下不会导致错误，除非因为掉电、硬件失效或操作者的失误而导致文件系统被破坏。

对于系统启动时自动挂装的文件系统，系统会自动进行检测，但是在如下情况下，不进行彻底的文件系统检测：存在 /etc/fastboot 文件；对 Ext2 文件系统，尚未达到最大挂装计数。其他文件系统则需要手工检测，例如软盘。

fsck 只能检查尚未挂装的文件系统，这主要是因为该命令需要以原始方式读写磁盘。

13.6.5 检查磁盘错误

应当周期性地检查磁盘上的坏块。利用 **badblocks** 命令可完成坏块的检查。**badblocks** 会报告所有的坏块清单，将该清单传递给 **fsck** 后，可避免文件系统在坏块上保存数据。**badblocks** 和 **fsck** 命令的组合使用方法如下例所示：

```
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ fsck -t Ext2 -l bad-blocks /dev/fd0H1440
Parallelizing fsck version 0.5a  (5-Apr-94)
e2fsck 0.5a, 5-Apr-94 for Ext2 FS 0.5, 94/03/10
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Check reference counts.
Pass 5: Checking group summary information.

/dev/fd0H1440: ***** FILE SYSTEM WAS MODIFIED *****
/dev/fd0H1440: 11/360 files, 63/1440 blocks
```

s

13.6.6 碎片化问题

我们提到过，Ext2 文件系统可通过特定的数据块分配方法来避免磁盘的过分碎片化。但是，经常性的文件删除和创建操作不可避免会产生碎片，对 Ext2 文件系统，可利用专用工具进行整理。在 MS-DOS 或 Windows 95 中，有大量工具可用于 FAT 或 VFAT 文件系统的整理。在整理文件系统之前，应当首先备份磁盘上的数据，以免在整理过程中发生不可恢复的错误。

13.6.7 其他文件系统工具

df 命令可列出所有已挂装文件系统中的空闲空间。

sync 命令用于将缓冲区高速缓存中的脏数据块刷新到磁盘上。

对于 Ext2 文件系统，还有一些特殊的工具。**tune2fs** 可调整 Ext2 文件系统的参数，其中包括最大挂装计数、两次检查之间的最大时间间隔、为 root 用户保留的数据块个数等；**dumpe2fs** 可显示大部分来自 Ext2 文件系统超块的信息；**debugfs** 是文件系统调试器，可用来直接访问文件系统中的数据；**dump** 和 **restore** 分别用于 Ext2 文件系统的备份和还原。

13.6.8 系统调用

表 13-10 简要列出了和文件系统相关的系统调用。标志列中各字母的意义可参见表 10-1 的说明。

表 13-10 相关系统调用

系统调用	说明	标志
bdflush	将脏缓冲区刷新到磁盘	-c
chdir	改变工作目录	m+c
chmod	修改文件属性	m+c
chown	修改文件所有权	m+c
chroot	设置新的根目录	mc
close	关闭文件	m+c
creat	创建文件	m+c
dup	建立某文件描述符的复本	m+c
dup2	复制某文件描述符	m+c
fchdir	改变工作目录	
fchmod	见 chmod	mc
fchown	修改文件的所有权	mc
fclose	关闭文件	m+!c
fcntl	文件/描述符控制	m+c
flock	修改文件锁定	m!c
fpathconf	获取文件信息	m+!c
fread	读取二进制文件流数据	m+!c

fstat	获取文件状态	m+c
fstatfs	获取文件系统状态	mc
fsync	将文件高速缓存写入磁盘	mc
ftime	获取自 1970.1.1 以来的时区＋秒数	m!c
ftruncate	修改文件大小	mc
fwrite	向文件流中写入二进制数据	m+!c
getdtablesize	获取文件描述符表的大小	m!c
ioctl	操作某个字符设备	mc
ioperm	设置某个 I/O 端口的许可	m-c
iopl	设置所有 I/O 端口的许可	m-c
link	建立已有文件的硬链接	m+c
llseek	大文件的 lseek	-
lseek	修改某文件描述符的当前位置指针	m+c
lstat	获取文件状态	mc
mkdir	建立目录	m+c
mknod	建立设备	mc
modify_ldt	读取或写入局部描述符表	-
mount	挂装一个文件系统	mc
open	打开一个文件	m+c
pathconf	获取文件的相关信息	m+!c
read	从文件中读取数据	m+c
readv	从文件中读取数据块	m!c
readdir	读取目录	m+c
readlink	获取符号链接的内容	mc
rename	移动/重命名文件	m+c
rmdir	删除某个空目录	m+c
select	进程休眠，直到某个文件描述符上有操作为止。	mc
setfsgid	设置文件系统的组标识符	
setfsuid	设置文件系统的用户标识符	
setup	初始化设备并安装 root	-
stat	获取文件状态	m+c
statfs	获取文件系统状态	mc
symlink	建立文件的符号链接	m+c
sync	内存和磁盘缓冲区之间的同步	mc
sysfs	获取已配置的文件系统信息	
truncate	修改文件大小	mc
ulimit	获取/设置文件限制	c!
umask	设置文件的建立掩码	m+c
umount	卸挂文件系统	mc
unlink	空闲时删除文件	m+c
utime	修改索引节点的时间项	m+c
write	向文件写入数据	m+c
writev	向文件写入数据块	m!c

第十四章

网 络

Linux 和网络几乎就是同义语，Linux 就是 Internet 和 WWW 的产物。Linux 的开发者使用网络和 Web 进行信息交换，而 Linux 本身又用于各种组织的网络支持。众所周知，TCP/IP 协议是 Internet 的标准协议，同时也是事实上的工业标准。Linux 的网络实现支持 BSD 套接字，支持完整的 TCP/IP 协议。本章描述 Linux 是如何支持 TCP/IP 网络协议的。

14.1 TCP/IP 协议

本节简要描述 TCP/IP 协议，有关该协议的详细内容，读者可参阅其他文献。

TCP/IP 协议是 Internet 网的基本协议，它实际由许多协议组成，并以协议组的形式存在，其中的主要协议有：传输控制协议（TCP）、用户数据报协议（UDP）、网际协议（IP）、网际信报控制协议（ICMP）和地址解析协议（ARP）等。

直接连接到 Internet 网上的主机必须具有唯一的地址，这一地址称为“IP 地址”。IP 地址由 4 个数字组成，中间用句点隔开，每个数字的取值范围一般在 0～255 之间，例如：192.1.1.1。在 Internet 上，主机的 IP 地址分为五类，分别是 A、B、C、D 和 E 五类。主机的唯一 IP 地址一般从 A、B 或 C 类地址中派生；D 类地址用来将计算机组织成一个功能组；而 E 类地址是试验性的，当前不可用。另外，一些特殊的 IP 地址被保留用于特殊目的，例如，127.0.0.1 就是用来特指本地主机的回环地址。

和主机 IP 地址相关的概念还有子网掩码，TCP/IP 软件利用子网掩码判断数据传输的目标主机是否和源主机处于同一子网中。例如，某主机的 IP 地址为 192.1.1.1，而子网掩码为 255.255.255.0，如果目标主机的 IP 地址为 192.1.1.4，则说明目标主机和源主机处于同一子网中，而如果目标主机的 IP 地址为 192.1.2.1，则说明不在同一子网中。上述判断通过利用子网掩码计算两台主机所在的子网地址而实现。主机 IP 地址和子网掩码的二进制与运算的结果称为“子网地址”，例如，主机 IP 地址 192.1.1.1 和子网掩码 255.255.255.0 的二进制与运算的结果为 192.1.1.0，即该主机所在子网的地址为 192.1.1.0，对地址为 192.1.1.4 的主机来说，可计算该主机所在子网地址为 192.1.1.0，于是说明目标主机和源主机处于同一子网中，而 192.1.2.1 却不在同一子网中。

子网掩码也同时定义了子网中主机的最大数目。如果子网掩码为 255.255.255.0，在上面的例子中，IP 地址从 192.1.1.1 到 192.1.1.254 的主机均可出现在同一子网中（IP 地

址 192.1.1.0 和 192.1.1.255 分别作为子网地址和子网中的广播地址），因此，该子网中的主机数目最多为 254 台。

因为数字式的 IP 地址非常难于记忆，因而通常利用主机域名标识主机，例如，bbs.tsinghua.edu.cn 是清华大学 BBS 服务器的域名，其 IP 地址为 202.112.58.200。但在利用域名标识主机的同时，也需要能够将域名转换为 IP 地址的机制，这种机制称为“域名解析”。在一般的 TCP/IP 主机中，这一名称可通过静态的 hosts 文件指定，也可通过 DNS （分布式名称服务器）服务器获得，在 Linux 中，/etc/hosts 文件指定静态的主机名称，而 etc/resolv.conf 文件指定 DNS 服务器的 IP 地址。

每当连接到其他计算机进行数据传输时，该计算机的 IP 地址就用于数据的传递。数据被划分为小的 IP 数据包发送，该数据包的前面是目标和源主机的 IP 地址，然后是数据本身，最后是数据的校验和以及其他有用的信息。利用校验和信息，目标主机可以判断数据的传输是否正确。由于数据包的物理传输介质不同，因此，针对不同的物理传输介质，IP 数据包还要进一步划分为小的数据包或报文进行传输。这种情况下，目标主机还要将小的报文重新装配成大的数据包，以便进行后续处理。

一般而言，逻辑上处于同一子网的两台主机处于同一局域网中，如果目标主机和源主机处于同一子网，就可通过某种机制获得目标主机的网卡物理地址，从而利用局域网技术实现数据传输。从主机的 IP 地址获得物理地址的机制称为“地址解析”，在 TCP/IP 协议中，地址解析可由专门的协议（地址解析协议）完成。如果目标主机和源主机不在同一子网中，这时的数据传输就要通过其他计算机完成，如果目标主机和源主机跨越大的地理距离，则可能要通过许多计算机的参与才能实现数据的传输。跨越不同子网的数据传输通过网关或路由器实现。当 TCP/IP 软件发现数据传输的目标主机处于其他子网时，它首先将数据发送到网关，然后由网关选择适当的路径传输，直到数据到达目标主机为止。Linux 维护一个路由表，每个目标 IP 地址均对应一个路由表项。利用路由表项，Linux 可将每个跨子网的 IP 数据包发送到一个适当的主机（路由器）。系统中的路由表实际是动态的，并随着应用程序的网络使用情况和网络拓扑结构的变化而变化。

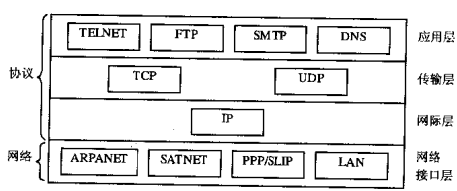


图 14-1 TCP/IP 协议层次结构

如前所述，TCP/IP 实际是以协议组的形式存在的，图 14-1 是 TCP/IP 协议层次结

构。从图中可看出，TCP/IP 映射为四层的结构化模型。这一模型也称为网际协议组（Internet Protocol Suit），可划分为网络接口、网际、传输和应用四层。

网络接口层（Network Interface Layer）负责和网络的直接通讯。它必须理解正在使用的网络结构，诸如令牌环和以太网等，并且还要提供允许网际层与之通讯的接口。网际层负责和网络接口层之间的直接通讯。

网际层（Internet Layer）主要完成利用网际协议（IP）的路由和数据包传递。传输层上的所有协议均要使用 IP 发送数据。网际协议定义如下规则：如何寻址和定向数据包；如何处理数据包的分段和重新组装；如何提供安全性信息；以及如何识别正在使用的服务类型等。

但是，由于 IP 不是基于连接的协议，因此它不能保证在线路中传输的数据不会丢失、破坏、重复或颠倒次序。这由网络模型中的高层，即传输层或应用层负责。网际层中还有一些其他的协议：网际信报控制协议（ICMP），网际组管理协议（IGMP）以及地址解析协议（ARP）等。

传输层（Transport Layer）负责提供应用程序之间的通讯。这种通讯可以是基于连接的，也可以是非基于连接的。这两种连接类型的主要差别在于是否跟踪数据以及是否确保数据发送到目标等。传输控制协议（Transmission Control Protocol, TCP）是基于连接的协议，能提供可靠的数据传输；而用户数据报协议（User Datagram Protocol, UDP）是非基于连接的协议，不能确保数据的正确传输。

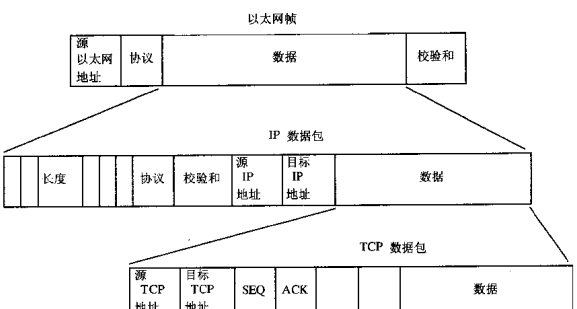


图 14-2 TCP 数据包的传输

Internet 协议组的应用层（Application Layer）作为应用程序和网络组件之间的接口而

存在，其中存在大量的协议，包括简单网络管理协议（Simple Network Management Protocol，SNMP）、文件传输协议（File Transfer Protocol，FTP）、简单邮件传输协议（Simple Mail Transfer Protocol，SMTP）等。

图 14-2 给出了 TCP 数据包的在网际协议组中的传输情况。TCP 利用 IP 数据包传输它自己的数据包，这时，IP 数据包中的数据是 TCP 数据包本身。UDP 也利用 IP 数据包进行数据的传输，在这种情况下，接收方的 IP 层必须能够知道接收到的 IP 数据包要发送给传输层中的哪个协议。为此，每个 IP 数据包头中包含一个字节，专门用作协议标识符。接收方的 IP 层利用这一标识符决定将数据包发送给传输层的哪一个协议处理。和上面的情况类似，同一台主机上利用同一协议进行通讯的应用程序可能有许多，因此，也需要一种机制来标识应由哪一个应用程序处理同一种数据包。为此，应用程序利用 TCP/IP 协议进行通讯时，不仅要指定目标 IP 地址，还要指定应用程序的“端口”地址。端口可唯一标识应用程序，标准的网络应用程序使用标准的端口地址，例如 Web 服务器的标准端口为 80。在网络接口中，IP 地址和端口地址合称为“套接字”。

IP 协议层可利用许多不同的物理介质传输 IP 数据包，图 14-2 中，IP 数据包进一步包装在以太网数据帧中传输。除以太网外，IP 数据包还可以在令牌环网等其他物理介质上传输。以太网数据帧头中包含了数据帧的目标以太网地址，以太网地址实际就是以太网卡的硬件地址或物理地址，一般由 6 位整数组成，如 00-A0-0C-13-CC-78。以太网卡的物理地址是唯一的，一些特殊的物理地址保留用于广播等目的。因为以太网数据帧和 IP 数据包一样，可以传输不同的协议数据，因此，数据帧中也包含一个标识协议的整数。

在以太网中，数据的传输是通过物理地址或硬件地址实现的，而 IP 地址实际只是一种概念性的逻辑地址，因此，在类似以太网这样的网络中，必须采用地址解析协议（ARP）将 IP 地址翻译为实际的硬件地址。ARP 负责为 IP 所请求的任意一个本地 IP 地址找出其本地物理地址。ARP 使用本地广播来寻找主机的物理地址，并在内存的高速缓冲区中维护最近所映射的物理地址。如果目标 IP 地址是本地地址，ARP 可发送一个本地广播请求获取目标 IP 主机的物理地址，并将物理地址返回给 IP。如果 IP 发现目标 IP 地址处于远程子网中，则数据包必须发送到路由器，这时，ARP 可替 IP 找到路由器的物理地址。

14.2 Linux 的 TCP/IP 网络层

图 14-3 描述了 Linux 的网络软件结构，和 TCP/IP 协议的结构类似，Linux 以分层的软件结构实施 TCP/IP 协议组。

BSD 套接字由一般性的套接字管理软件 INET 套接字层支持。INET 套接字管理着基于 IP 的 TCP 或 UDP 协议端。如前所述，TCP 是基于连接的协议，而 UDP 是非基于连接的协议。在传输 UDP 数据包时，Linux 不需要关心数据包是否安全到达目的端。但对 TCP 数据包来说，Linux 需要对数据包进行编号，数据包的源端和目的端需要协调工作，以便保证数据包不会丢失，或以错误的顺序发送。IP 层包含的代码实施了 Internet 协议，IP 层需处理数据包的报文头信息，并且必须将传入的数据包发送到 TCP 或 UDP 两者中正确的一层处理。在 IP 层之下是 Linux 的网络设备层，其中包括以太网设备或 PPP 设备等。和 Linux 系统中的其他设备不同，网络设备并不总代表实际的物理

设备，例如，回环设备就是一个纯软件设备，而且，网络设备并不能通过 Linux 的 **mknod** 命令建立，而是只有在底层软件发现并初始化这些设备之后才会出现在 /dev 目录下。ARP 协议提供地址解析功能，因此它处于 IP 层和网络设备层之间。

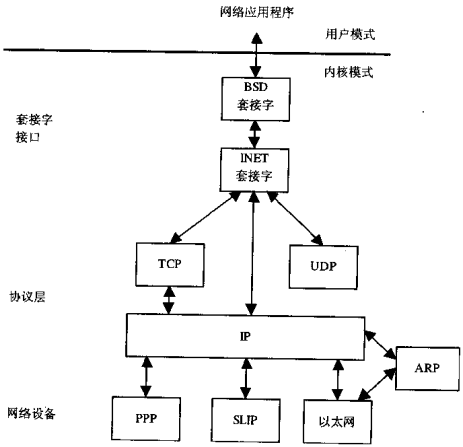


图 14-3 Linux 的网络层

14.3 BSD 套接字接口

套接字既可看成是支持多种网络操作形式的接口，也可看成是一种进程间通讯接口。在一条通讯连接中，每个参与通讯的进程有一个套接字描述相应的连接端。我们可以将套接字看成是某种特殊类型的管道，但和管道不同的是，套接字并不限制其中可以包含的数据数量。Linux 支持多种套接字种类，不同的套接字种类称为“地址族”，这是因为每种套接字种类拥有自己的通讯寻址方法。Linux 所支持的套接字地址族见表 14-1。

表 14-1 Linux 支持的套接字地址族

套接字地址族	描述
UNIX	UNIX 域套接字。
INET	通过 TCP/IP 协议支持的 Internet 地址组。
AX25	Amater radio X25
IPX	Novell IPX
APPLETALK	Appletalk DDP
X25	X25

和虚拟文件系统类似，Linux 将上述套接字地址族抽象为统一的 BSD 套接字接口，应用程序关心的只是 BSD 套接字接口，而 BSD 套接字由各地址族专有的软件支持。一般而言，BSD 套接字可支持多种套接字类型，不同的套接字类型提供的服务不同，Linux 所支持的 BSD 套接字类型见表 14-2，表 14-1 中的套接字地址族并不一定全部支持这些套接字类型。

表 14-2 Linux 所支持的 BSD 套接字类型

BSD 套接字类型	描述
流（stream）	这种套接字提供了可靠的双向顺序数据流，并可保证数据不会在传输过程中丢失、破坏或重复出现。流套接字通过 INET 地址族的 TCP 协议实现。
数据报（datagram）	这种套接字也提供双向的数据传输，但是并不对数据的传输提供担保，也就是说，数据可能会以错误的顺序传递，甚至丢失或破坏。这种类型的套接字通过 INET 地址族的 UDP 协议实现。
原始（raw）	利用这种类型的套接字，进程可以直接访问底层协议（因此称为原始）。例如，可在某个以太网设备上打开原始套接字，然后获取原始的 IP 数据传输信息。
可靠发送的消息	和数据报套接字类似，但保证数据被正确传输到目的端。
顺序数据包	和流套接字类似，但数据包大小是固定的。
数据包（packet）	这并不是标准的 BSD 套接字类型，它是 Linux 专有的 BSD 套接字扩展，可允许进程直接在设备级访问数据包。

进程在利用套接字进行通讯时，采用客户－服务器模型。服务器提供某种服务，而客户使用这种服务。WWW 服务器就是很好的例子，WWW 服务器提供 web 页，而客户程序，即浏览器则读取 web 页并显示 web 页的内容。服务器首先创建一个套接字，并将某个名称绑定到该套接字上，套接字的名称依赖于套接字的底层地址族，但通常是服务器的本地地址。套接字的名称或地址通过 sockaddr 数据结构指定。对于 INET 套接字来说，服务器的地址由两部分组成，一个是服务器的 IP 地址，另一个是服务器的端口地址。已注册的标准端口可查看 /etc/services 文件。将地址绑定到套接字之后，服务器就可以监听请求链接该绑定地址的传入连接。连接请求由客户生成，它首先建立一个套接字，并指定服务器的目标地址以请求建立连接。传入的连接请求通过不同的协议层最终到达服务器的监听套接字。服务器接收到传入的请求后，如果能够接受该请求，服务器必须创建一个新的套接字来接受该请求并建立通讯连接（用于监听的套接字不能用来建立通讯连接），这时，服务器和客户就可以利用建立好的通讯连接传输数据。

BSD 套接字上的详细操作和具体的底层地址族有关，底层地址族的不同实际意味着寻址方式、采用的协议等的不同，因此 TCP/IP 连接的建立过程和 AX25 连接的建立过程有很大的区别。前面提到，Linux 利用 BSD 套接字层抽象了不同的套接字接口。在内核的初始化阶段，内建于内核的不同地址族分别以 BSD 套接字接口在内核中注册。然后，随着应用程序创建并使用 BSD 套接字，内核负责在 BSD 套接字和底层的地址族之间建立联系。这种联系通过交叉链接数据结构以及地址族专有的支持例程表建立。例如，每个地址族具有专有的套接字创建例程，应用程序在建立新的套接字时，BSD 套接字接口可利用该例程建立新的 BSD 套接字。

在内核中，地址族和协议信息保存在 protocols 向量中。每个地址族由其名称（例如“INET”）以及相应的初始化例程地址代表。在引导阶段初始化套接字接口时，内核调用每个地址族的初始化例程，这时，每个地址族注册自己的协议操作集。协议操作集实际是一个例程集合，其中每个例程执行一个特定的操作。注册的协议操作集保存在 pops 向量中，向量中包含指向 proto_ops 数据结构的指针。proto_ops 数据结构由地址族类型和一组套接字操作例程指针组成，这些例程是特定的地址族所特有的。pops 向量的索引是地址族的标识符，例如，INET 地址族的标识符为 2。

14.4 INET 套接字层

INET 套接字层是用于支持 Internet 地址族的套接字层。它和 BSD 套接字之间的接口通过 Internet 地址族套接字操作集实现，如前所述，这些操作集实际是一组协议的操作例程。网络的初始化过程中，这一操作集在 BSD 套接字层中注册，并且和其他注册的地址族操作集一起保存在 pops 向量中。BSD 套接字层通过调用 proto_ops 结构中的相应函数执行任务，例如，当应用程序给定 INET 地址族来创建 BSD 套接字时，将利用 INET 套接字创建函数来执行这一任务。在每次的套接字操作函数调用中，BSD 套接字层向 INET 套接字层传递 socket 数据结构来代表一个 BSD 套接字，但在 INET 套接字层中，它利用自己的 sock 数据结构来代表该套接字，因此，这两个结构之间存在着链接关系，这一关系可从图 14-4 中看出。

在 BSD 的 socket 数据结构中存在一个 data 指针，该指针将 BSD socket 数据结构和 sock 数据结构链接了起来。通过这种链接关系，随后的 INET 套接字调用就可以方便地检索到 sock 数据结构。实际上，sock 数据结构可适用于不同的地址族，在建立套接字时，sock 数据结构的协议操作集指针指向所请求的协议操作集。如果请求 TCP 协议，则 sock 数据结构的协议操作集指针将指向 TCP 的协议操作集。

14.4.1 建立 BSD 套接字

在利用 Linux 系统调用建立新的套接字时，需要传递套接字的地址族标识符、套接字类型以及协议。内核首先利用套接字的地址族标识符搜索 pops 向量。如果指定的地址族以内核模块的形式实现，**kerneld** 守护进程负责装入相应的模块。如果能够建立该 BSD 套接字，内核将为该套接字分配新的 socket 数据结构。socket 数据结构实际是 VFS 索引节点数据结构的一部分，分配新的 socket 数据结构实际就是分配新的 VFS 索引节点。因为可以和操作普通的文件一样操作套接字，而所有的文件均由一个 VFS

索引节点代表，因此，为了在套接字上支持文件操作，也必须以 VFS 索引节点来代表 BSD 套接字。

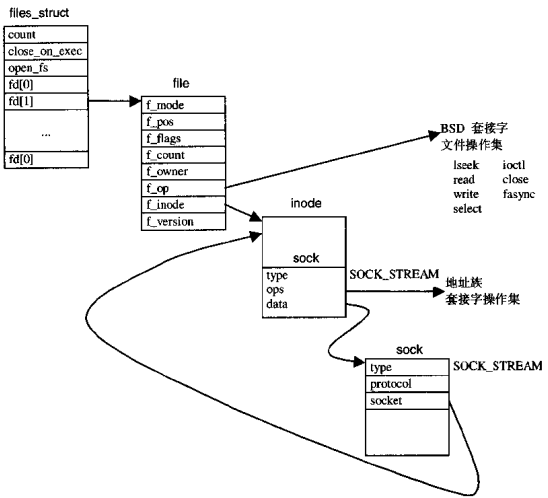


图 14-4 Linux BSD 套接字数据结构

参照图 14-4，新创建的 BSD socket 数据结构包含有指向地址族专有的套接字例程的指针，这一指针实际就是 proto_ops 数据结构的地址，proto_ops 数据结构可从 pops 向量中检索到。BSD 套接字的套接字类型设置为所请求的 SOCK_STREAM 或 SOCK_DGRAM 等。然后，内核利用 proto_ops 数据结构中的信息调用地址族专有的创建例程。

之后，内核从当前进程的 fd 向量中分配空闲的文件描述符，该描述符指向的 file 数据结构被初始化。初始化过程包括将文件操作集指针指向由 BSD 套接字接口支持的 BSD 文件操作集。所有随后的套接字（文件）操作都将定向到该套接字接口，而套接字接口则会进一步调用地址族的操作例程，从而将操作传递到底层地址族。

14.4.2 在 INET BSD 套接字上绑定地址

为了监听传入的 Internet 连接请求，每个服务器都需要建立一个 INET BSD 套接字，并且将自己的地址绑定到该套接字。绑定操作主要在 INET 套接字层中进行，还需要底层 TCP 层和 IP 层的某些支持。将地址绑定到某个套接字上之后，该套接字就不能用来进行任何其他的通讯，因此，该 socket 数据结构的状态必须为 TCP_CLOSE。传递到绑定操作的 sockaddr 数据结构中包含要绑定的 IP 地址，以及一个可选的端口地址。通常而言，要绑定的地址应该是赋予某个网络设备的 IP 地址，而该网络设备应该支持 INET 地址族，并且该设备是可用的。利用 ifconfig 命令可查看当前活动的网络接口。被绑定的 IP 地址保存在 sock 数据结构的 recv_addr 和 saddr 字段中，这两个字段分别用于哈希查找（下面讲述）和发送用的 IP 地址。端口地址是可选的，如果没有指定，底层的支持网络会选择一个空闲的端口。因为小于 1024 的端口号作为标准端口号已经分配给了标准应用程序，因此，底层网络只会分配大于 1024 的端口号，而且，没有超级用户的权限，进程不能使用小于 1024 的端口号。

当底层网络设备接受到数据包时，它必须将数据包传递到正确的 INET 和 BSD 套接字以便进行处理，因此，UDP 和 TCP 各自维护一个或多个哈希表，用来查找传入 IP 消息的地址，并将它们定向到正确的 socket/sock 对。因为 TCP 是面向连接的协议，因此在处理 TCP 数据包时涉及到的内容比起处理 UDP 数据包来多。

UDP 维护的哈希表（upd_hash 表）包含了已分配的 UDP 端口。这一哈希表由 sock 数据结构的指针组成，并可基于端口号经哈希函数索引。当 UDP 哈希表远远小于可容忍的端口数量时（upd_hash 表中只能容纳 128 即 UDP_HTABLE_SIZE 个表项），哈希表中的某些表项指向由 sock 数据结构链接起来的链表（利用 sock 的 next 指针）。

TCP 维护的哈希表不止一个，因此更加复杂一些。但是，TCP 并不在绑定过程中将绑定的 sock 数据结构添加到哈希表中，在这一过程中，它仅仅判断所请求的端口号当前是否正在使用。在监听操作中，该 sock 结构才被添加到 TCP 的哈希表中。

14.4.3 在 INET BSD 套接字上建立连接

创建一个套接字之后，该套接字不仅可以用于监听入站的连接请求，也可以用于建立出站的连接请求。对类似 UDP 的非连接协议，该套接字操作涉及的内容不多，但对面向连接的协议，如 TCP 来说，该操作涉及到一个重要的过程：建立两个应用程序之间的虚拟电路。

出站连接只能建立在处于正确状态的 INET BSD 套接字上，因此，不能建立于已建立连接的套接字，也不能建立于用于监听入站连接的套接字。也就是说，该 BSD socket 数据结构的状态必须为 SS_UNCONNECTED。UDP 协议并不在应用程序之间建立虚拟电路，它所发送的信息均是数据报，有可能不能正确到达目标应用程序，但是，它也支持 BSD 套接字的 connet 操作。在 UDP INET BSD 套接字上的连接操作只是简单地设置远程应用程序的地址，即 IP 地址和 IP 端口号。另外，它还建立一个路由表项的缓存，这样，在该 BSD 套接字上发送的 UDP 数据包不需要再次检查路由数据库（除非该路由非法），INET sock 数据结构中的 ip_route_cache 指针指向这一缓存路由信息。如果没有给

定寻址信息的话，缓存的路由和 IP 寻址信息将自动用于消息的发送。此后，UDP 将 sock 的状态设置为 TCP_ESTABLISHED。

对于 TCP BSD 套接字上的连接操作，TCP 必须建立一个 TCP 消息，该消息包含连接信息并发送到给定的目标 IP。TCP 消息中包含了有关连接的消息、一个唯一的消息起始顺序号，以及传输和接收窗口的大小等信息。在 TCP 中，所有的消息均是编号的，初始的顺序号用作第一个消息编号，Linux 选择合理的随机数以避免受到恶意的协议攻击。由 TCP 连接的某一端发送的每条消息，如果由另一端成功接收到，则接收端发送确认消息，那些未经确认的消息则需要重新发送。传输和接收窗口的尺寸，是指在未获得确认的情况下可发送出站的消息数量。最大的消息尺寸根据发出连接请求一端的网络设备决定，如果接收端的网络设备所支持的最大消息尺寸较小，则该连接将使用较小的最大消息尺寸。这时，建立出站连接请求的应用程序必须等待目标应用程序的接受或拒绝响应。如果 TCP sock 正在等待传入消息，则该 sock 结构添加到 tcp_listening_hash 表中，这样，传入的 TCP 消息就可以定向到该 sock 数据结构。当出站连接请求发送出去之后，TCP 将启动一个定时器，如果在该定时器到期之前目标应用程序没有响应，则该连接请求最终会因超时而失败。

14.4.4 监听 INET BSD 套接字

当某个套接字被绑定了地址之后，该套接字就可以用来监听专属于该绑定地址的传入连接。网络应用程序也可以在未绑定地址之前监听套接字，这时，INET 套接字层将利用空闲的端口编号并自动绑定到该套接字。套接字的监听函数将 socket 的状态改变为 TCP_LISTEN。

对于 UDP 套接字来说，套接字状态的改变已经足够了，但对 TCP 来说，它现在需要将套接字的 sock 数据结构添加到两个哈希表中，这两个哈希表分别为 tcp_bound_hash 和 tcp_listening_hash，这两个哈希表的哈希函数均以端口号为参数。

当接收到某个传入的 TCP 连接请求时，TCP 建立一个新的 sock 数据结构来描述该连接。当该连接最终被接受时，新的 sock 数据结构将变成该 TCP 连接的内核底半部分，这时，它要克隆包含连接请求的传入 sk_buff 中的信息，并在监听 sock 数据结构的 receive_queue 队列中将克隆的信息排队。克隆的 sk_buff 中包含有指向新 sock 数据结构的指针。

14.4.5 接受连接请求

UDP 不支持连接的概念，接受 INET 套接字的连接请求只应用于 TCP 协议。接受操作在监听套接字上进行，最终从原始的监听 socket 中克隆一个新的 socket 数据结构。其过程如下所述。接受操作首先传递到支持协议层，即 INET 中，以便接受任何传入的连接请求。如果底层的协议，例如 UDP 不支持连接，则 INET 协议层上的接受操作将失败。相反，接受操作进一步传递到实际的协议，例如 TCP 上。接受操作可以是阻塞的，也可以是非阻塞的。接受操作为非阻塞的情况下，如果没有可接受的传入连接，则接受操作将失败，而新建立的 socket 数据结构被抛弃。接受操作为阻塞的情况下，执行阻塞操作的网络应用程序将添加到等待队列中，并保持挂起直到接收到一个 TCP 连

接请求为至。当连接请求到达之后，包含连接请求的 sk_buff 被丢弃，而由 TCP 建立的新 sock 数据结构返回到 INET 套接字层，在这里，sock 数据结构和先前建立的新 socket 数据结构建立链接。而新 socket 的文件描述符（fd）被返回到网络应用程序，此后，应用程序就可以利用该文件描述符在新建立的 INET BSD 套接字上进行套接字操作。

14.5 IP 层

14.5.1 套接字缓冲区

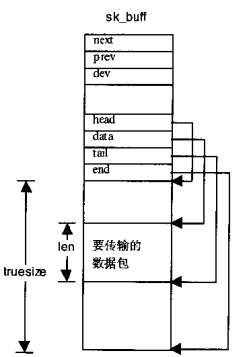


图 14-5 Linux 的套接字缓冲区（sk_buff）

网络协议的分层实施也带来了一些问题，其中比较明显的一个问题是，每个协议都需要在发送数据时添加协议头和协议尾，而在接收数据时去掉协议头和协议尾，因此，当数据在不同的协议之间传递时，由于每层都要寻找自己特定的协议头和协议尾，从而导致数据缓冲区的传递非常困难。一种解决办法是在不同的协议层之间复制数据缓冲区，但这种方法的效率较低，为此，Linux 利用套接字缓冲区（sk_buff）在协议层和网络设备之间传送数据。sk_buff 包含了一些指针和长度信息，从而可让协议层以标准的函数或方法对应用程序的数据进行处理。

如图 14-5 所示，每个 sk_buff 均包含一个数据块、四个数据指针以及两个长度字段。利用四个数据指针，各协议层可操纵和管理套接字缓冲区的数据，这四个指针的用途在表 14-2 中列出。

表 14-2 sk_buff 的四个数据指针

指针名称	用途
head	指向内存中数据区的起始地址。sk_buff 和相关数据块在分配之后，该指针的值是固定的。
data	指向协议数据的当前起始地址。该指针的值随当前拥有 sk_buff 的协议层的变化而变化。
tail	指向协议数据的当前结尾地址。和 data 指针一样，该指针的值也随当前拥有 sk_buff 的协议层的变化而变化。
end	指向内存中数据区的结尾。和 head 指针一样，sk_buff 被分配之后，该指针的值也固定不变。

sk_buff 的两个长度字段，len 和 truesize，分别描述当前协议数据包的长度和数据缓冲区的实际长度。sk_buff 的处理代码提供了一些标准函数用来为应用程序数据添加或去除协议头及协议尾，利用这些函数可安全地操纵 sk_buff 结构中的 data 及 tail 字段，这些函数如表 14-3 所示。

表 14-3 sk_buff 指针字段的标准操作函数

函数名称	功能
push	该函数将 data 指针移向数据区的前端并增加 len 字段的值。在发送数据的过程中，利用该函数可在数据的前端添加数据或协议头。
pull	该函数和 push 函数的功能相反，它将 data 指针移向数据区的末尾，并减小 len 字段的值。该函数可用于从接收到的数据头上移去数据或协议头。
put	该函数将 tail 指针移向数据区的末尾并增加 len 字段的值。在发送数据的过程中，利用该函数可在数据的末端添加数据或协议尾。
trim	该函数和 put 函数的功能相反，它将 tail 指针移向数据区的前端，并减小 len 字段的值。该函数可用于从接收到的数据尾上移去数据或协议尾。

在 sk_buff 数据结构中，还包含一些用来实现 sk_buff 循环双向链表结构的指针。内核中也包含一些在上述链表头上或链表尾上添加或删除 sk_buff 结构的一般例程。

14.5.2 接收 IP 数据包

在第十二章中讲到，各设备的 device 数据结构保存在 dev_base 表中，网络设备和其他设备一样，也在系统的引导初始化过程中在 dev_base 表中注册自身的 device 数据结构。每个网络设备的 device 数据结构描述了设备自身，并提供一组回调函数，各协议层可以在需要网络驱动程序完成任务时调用这些函数。一般而言，调用这些函数时需要指定要传输的数据以及网络设备的地址。当网络设备从网络上接收到数据包时，它必须将接收到的数据转换为 sk_buff 数据结构。在网络设备接收到这些数据时，

经转换的 sk_buff 数据结构添加到 backlog 队列中排队。当 backlog 队列变得很大时，接收到的 sk_buff 数据将会被丢弃。当新的 sk_buff 添加到 backlog 队列时，网络的底半部分被标志为运行就绪状态，从而可让调度程序选择底半处理程序进行处理。

调度程序最终会运行网络的底半处理程序，这时，网络底半处理程序将处理任何等待传输的数据包，但在这之前，底半处理程序会首先处理 sk_buff 结构的 backlog 队列。底半处理程序必须确定将接收到的数据包传递到哪个协议层。

在 Linux 进行网络层的初始化时，每个协议要在 ptype_all 链表或 ptype_base 哈希表中添加 packet_type 数据结构以进行注册。packet_type 数据结构包含协议类型、指向网络设备的指针、指向协议的接收数据处理例程的指针，以及在链表或哈希表中指向下一 packet_type 结构的指针等。内核利用 ptype_all 用来检查从任意网络设备上接收到的数据包，但通常不使用 ptype_all 链表。ptype_base 是一个哈希表，其哈希函数以协议标识符为参数，内核利用该哈希表判断应当接收传入的网络数据包的协议。通过检查上述两个表，网络底半处理程序可以找出与传入 sk_buff 的协议类型匹配的 packet_type 项。匹配的 packet_type 协议可能不止一个（如监视所有的网络流量时），这种情况下，网络底半处理程序会复制新的 sk_buff，最终，sk_buff 会传递到一个或多个目标协议的处理例程。

14.5.3 发送 IP 数据包

数据包可以由应用程序生成，也可以网络协议生成（如建立连接时）。不管数据是如何生成的，网络处理代码必须建立 sk_buff 结构以包含该数据，并且在协议层之间传递数据时，需要添加不同的协议头和协议尾。

sk_buff 需要传递到某个网络设备上传输，首先，IP 协议需要决定要使用的网络设备，网络设备的选择依赖于数据包的最佳路由。对于只利用调制解调器和 PPP 协议连接的计算机来说，路由的选择比较容易。数据包可能发送到本地的回环设备，或 PPP 调制解调器连接端的网关。但对连接到以太网的计算机来说，路由的选择是比较复杂的，这是因为同一以太网中可能连接着许多台计算机。

对每个要传输的 IP 数据包，IP 利用路由表解析目标 IP 地址的路由。对每个可从路由表中找到路由的目标 IP 地址，路由表返回一个 rtable 数据结构描述可使用的路由。这包括要使用的源 IP 地址、网络设备的 device 数据结构的地址以及预先建立的硬件头信息。该硬件头信息和网络设备相关，包含了源和目标的物理地址以及其他的介质信息。对以太网来说，硬件头信息可见图 14-2，而源和目标地址就是以太网卡的物理地址。硬件头和路由信息一起缓存，这是因为该硬件头信息需要添加到所有利用该路由传输的 IP 数据包，而硬件头信息的建立需要花费时间。硬件头中的物理地址信息有时需要利用 ARP 协议获得，这种情况下，传出的数据包要延迟到 ARP 将目标物理地址解析之后才能发送。将解析到的硬件头信息和路由一起缓存，可避免利用该路由发送的 IP 数据包再次利用 ARP 解析物理地址。

14.5.4 数据包的分段和重组

每个网络设备有其最大的数据包尺寸，它不能传输和接收大于最大尺寸的数据包。当

数据包的尺寸大于最大允许尺寸时，IP 可以将数据分段成较小的单元以适合于网络设备能够允许的最大数据包尺寸。为此，IP 协议头中包含有分段字段，由一个标志和分段偏移量组成。

当 IP 数据包能够传输时，IP 从 IP 路由表中找到发送该 IP 数据包的网络设备，网络设备对应的 device 数据结构中包含有一个 mtu 字段，该字段描述最大的传输单元(字节为单位)。如果设备的 mtu 小于等待发送的 IP 数据包的大小，就需要将该 IP 数据包划分为小的片段(mtu 指定的大小)。每个片段由一个 sk_buff 代表，其中的 IP 头标记为数据包片断，以及该片断在 IP 数据包中的偏移。最后的数据包被标志为最后的 IP 片断。如果分段过程中 IP 不能分配 sk_buff，则传输失败。

IP 片断的接收较片断的发送更加复杂一些，因为 IP 片断可能以任意的顺序接收到，而在重组之前，必须接收到所有的片断。每次接收到 IP 数据包时，IP 要检查是否是一个分段数据包。当第一次接收到分段的消息时，IP 建立一个新的 ipq 数据结构，并将它链接到由等待重组的 IP 片断形成的 ipqueue 链表中。随着其他 IP 片断的接收，IP 找到正确的 ipq 数据结构，同时建立新的 ipfrag 数据结构描述该片断。每个 ipq 数据结构中包含有其源和目标 IP 地址、高层协议的标识符以及该 IP 帧的标识符，从而唯一描述了一个分段的 IP 接收帧。当所有的片断接收到之后，它们被组合成单一的 sk_buff 并传递到上一级协议层处理。如果定时器在所有的片断到达之前到期，ipq 数据结构和 ipfrag 被丢弃，并假定消息已经在传输中丢失，这时，高层协议需要请求源主机重新发送丢失的消息。

14.6 地址解析协议

地址解析协议的任务，就是将 IP 地址翻译为物理的硬件地址，例如以太网地址。在 IP 将数据以 sk_buff 的形式传递到网络设备传输时，IP 需要进行上述翻译。IP 首先检查设备是否需要硬件头，如果需要，则继续检查数据包的硬件头是否需要重新建立。如前所述，Linux 利用缓存以避免频繁重建硬件头信息。如果硬件头需要重新建立，则调用设备特有的硬件头重建例程。所有的以太网设备使用相同的一般性硬件头重建例程，而这些例程实际使用 ARP 服务将目标 IP 地址转换为物理地址。

ARP 协议本身是非常简单的，它包含两种消息类型：ARP 请求以及 ARP 回应。ARP 请求包含了需要转换的 IP 地址，而 ARP 回应则包含要转换的 IP 地址对应的硬件地址。ARP 请求广播到网络上，因此，对于以太网来讲，所有连接到以太网的计算机均会看到该 ARP 请求。最终，拥有该 IP 地址的计算机将响应该 ARP 请求，从而生成包含物理地址的 ARP 回应消息。

Linux 中的 ARP 协议层建立有一个 arp_table 数据结构表，其中的每个表项描述一个 IP 地址到物理地址的翻译。这些表项在需要翻译 IP 地址时建立，而在表项失效时删除。每个 arp_table 数据结构中包含的字段如表 14-4 所示。

表 14-4 arp_table 数据结构中的字段

字段	功能
最近使用时间	该 ARP 表项最近使用过的时间。

最近更新时间	该 ARP 表项最近更新过的时间。
标志	表述该表项状态的字段，例如表项是否完整等。
IP 地址	该表项所描述的 IP 地址。
硬件地址	经翻译的硬件地址。
硬件头	指向缓存硬件头的指针。
定时器	这是 timer_list 的一个表项，用来跟踪 ARP 请求的响应超时。
重试次数	该 ARP 请求重试过的次数。
sk_buff 队列	等待解析该 IP 地址的 sk_buff 项链表。

ARP 表由一个指针表（即 arp_tables 向量）组成，而这些指针指向 arp_table 项形成的链。这些表项是经过缓存的，从而可提高表项的访问速度。利用其 IP 地址的最后两个字节可生成访问上述指针表的索引，最终可从表项链中查找到正确的 arp_table 项。如前所述，Linux 还对预先建立的硬件头进行缓存，实际是将 arp_table 缓存在 hh_cache 数据结构中。

当请求进行某个 IP 地址的翻译，而当前没有对应的 arp_table 项时，ARP 必须发送 ARP 请求消息。它首先在表中建立一个新的 arp_talbe 表项，然后将包含网络数据包的 sk_buff 在 arp_table 表项的 sk_buff 队列中排队。对于这些排队等待翻译 IP 地址的 sk_buff 来说，传输该 sk_buff 的协议层将接收到通知，说明需要等待以应付 ARP 地址的翻译。UDP 协议不关心丢失的数据包，而 TCP 却需要在建立好的 TCP 链路上传输。如果拥有该 IP 地址的计算机返回了它的硬件地址，arp_table 就被标志为完整表项，而排队的 sk_buff 将从该队列中移走以继续传输。硬件地址被写入每个 sk_buff 硬件头。

ARP 协议层也必须响应指定其 IP 地址的 ARP 请求。ARP 协议层通过生成一个 packet_type 数据结构而注册自己的协议类型，这样，网络设备接收到的 ARP 数据包均会被传递到 ARP 协议层处理。在响应 ARP 请求时，它利用保存在接收设备的 device 数据结构中的硬件地址生成 ARP 回应消息。

网络拓扑结构可能随时间而改变，而 IP 地址也有可能赋予不同的网络设备。例如，在 DHCP 系统中，网络设备的 IP 地址是动态分配的。为了保证 ARP 表中包含最新的表项，ARP 运行一个周期的定时器检查所有的 arp_table 是否过期。但这时不能移去包含缓存硬件头的表项，删除这些由其他数据结构依赖的表项是非常危险的。有些 arp_table 表项是永久性的，因此它们是不会被释放的。系统不允许 ARP 表增长得太大，因为每个 arp_table 均要花费一定的内核内存。当需要分配新的表项，而 ARP 表已经到达其最大尺寸时，系统搜索最旧的表项，通过删除这些旧表项而减小 ARP 表的尺寸。

14.7 IP 路由

IP 的路由功能决定将定向于特定 IP 地址的 IP 数据包发送到哪里。在传输 IP 数据包时，可能会有多种选择。目标地址能否到达？如果能够到达，应利用哪个网络设备发送？如果有多个网络设备可用于发送数据包，那么哪一个网络设备用于传输更好一些？IP 路由数据库中维护的信息可回答上面的这些问题。系统中包含有两个数据库，最重要的是

转发信息数据库，其中包含了一个详尽的、由已知的 IP 地址和对应的最佳路由组成的清单。一个小的、更加快速的数据库，即路由缓存，可用于快速查找目标 IP 的路由。和所有的缓存一样，其中只包含一些最频繁访问的路由，这些路由信息是从转发信息数据库中派生出来的。

路由信息通过 BSD 套接字接口的 IOCTL 请求执行添加和删除操作。这些操作最终传递到协议处理。INET 协议层只允许具备超级用户权限的进程添加和删除 IP 路由。这些路由信息可能是固定的，也有可能是随时间变化的动态路由。绝大多数系统使用固定路由，除非该系统作为路由器使用。路由器运行路由协议，路由协议可经常性地检查所有已知 IP 地址的路由可用性。非路由器系统称为末端系统。路由协议以守护进程（**gated**）的方式实现，也通过 BSD 套接字接口的 IOCTL 添加和删除路由。

14.7.1 路由缓存

在查找一个 IP 路由时，首先检查路由缓存。如果路由缓存中没有匹配的路由，系统就在转发信息数据库中查找路由。如果在转发信息数据库中也找不到路由，IP 数据包就不能成功发送，而应用程序会接收到发送失败的通知。如果路由不在路由缓存，而在转发信息数据库中，则生成新的缓存项并添加到缓存路由中。路由缓存是一个哈希表（ip_rt_hash_table），其中包含的指针指向由 rtable 数据结构形成的链。路由表的索引通过基于 IP 地址的两个最低字节的哈希函数计算出来。这两个字节是不同目标 IP 地址中最可能不同的部分，从而可提供最大的哈希值分布。每个 rtable 项包含路由信息，即目标 IP 地址、到达该目标 IP 地址而使用的网络 device 结构地址，以及可用来发送的最大消息尺寸等信息。其中也包含一个引用计数、使用计数以及最后使用过的时间戳（以 jiffies 度量）。每次使用该路由时，引用计数增加以表明使用该路由的网络连接数量；当应用程序停止使用该路由时，该计数减少。每次查找该路由并且在哈希项指向的 rtable 项链中定位时，使用计数增加。路由缓存中所有项的最近使用时间戳会周期性地检查，以查看 rtable 是否太旧，如果某个路由最近尚未使用，则从路由缓存中丢弃该 rtable 表项。如果路由一直保留在路由缓存中，则系统对它们进行排序以便将使用得最多的表项排列在哈希链的前面，这样，当查找该路由时，能够快速找到。

14.7.2 转发信息数据库

如图 14-6 所示，是系统中的转发信息数据库结构。该数据库是一个相当复杂的数据结构，虽然该结构的组织相当有效，但仍然不能算是一个快速的数据库。如果对每一个要传输 IP 数据包，均要从该数据库中查找路由，则系统性能将非常差。利用路由缓存则可以避免重复地在转发信息数据库中查找路由。

每个 IP 子网由一个 fib_zone 数据结构代表。包含在 fib_zones 哈希表中的指针指向所有的 fib_zone，该哈希表的哈希索引由 IP 子网地址生成。到达同一子网的所有路由由一对 fib_node 和 fib_info 数据结构描述，这两个数据结构排列在 fib_zone 数据结构中的 fz_list 上。当该子网中的路由数量变大时，则生成一个哈希表以便快速找到 fib_node 数据结构。

对于相同的 IP 子网，可能存在多个路由，而这些路由可通过多个网关中的其中一个网关。IP 路由层不能允许到某个子网的多个路由使用相同的网关，也就是说，如果到某

个子网的路由有多个，则必须保证每个路由使用不同的网关。和每个路由相关的是它的 metric，它用来度量路由的优劣。从本质上讲，路由的 metric 指到达指定子网前，该路由要经过的 IP 子网数量，路由的 metric 值越高，该路由越坏。

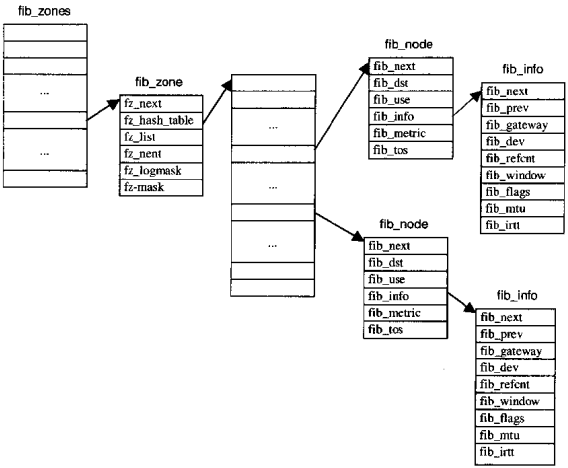


图 14-6 转发信息数据库结构

14.8 相关系统工具和系统调用

由于篇幅的限制，本书不具体讲解和网络相关的系统工具，读者可参阅相关著作。本书第三部分中有一些专题专门讨论 Linux 的网络应用。

表 14-5 简要列出了和网络（套接字）相关的系统调用。标志列中各字母的意义可参见表 10-1 的说明。

表 14-5 相关系统调用

系统调用	说明	标志
accept	接收套接字上连接请求	m!c
bind	在套接字绑定地址信息	m!c
connet	连接两个套接字	m!c
getpeername	获取已连接端套接字的地址	m!c
getsockname	获取套接字的地址	m!c
getsockopt	获取套接字上的设置选项	m!c
listen	监听套接字连接	m!c
recv	从已连接套接字上接收消息	m!c
recvfrom	从套接字上接收消息	m!c
send	向已连接的套接字发送消息	m!c
sendto	向套接字发送消息	m!c
setdomainname	设置系统的域名	mc
sethostid	设置唯一的主机标识符	mc
sethostname	设置系统的主机名称	mc
setsockopt	修改套接字选项	mc
shutdown	关闭套接字	m!c
socket	建立套接字通讯的端点	m!c
socketcall	套接字调用多路复用转换器	-
socketpair	建立两个连接套接字	m!c

第十五章

其 他 内 核 机 制

15.1 底半处理

我们知道，发生中断时，处理器要停止当前正在执行的指令，而操作系统负责将中断发送到对应的设备驱动程序去处理。在中断的处理过程中，系统不能进行其他任何工作，因此，在这段时间内，设备驱动程序要以最快的速度完成中断处理，而其他大部分工作在中断处理过程之外进行。Linux 内核利用底半处理过程帮助实现中断的快速处理。

图 15-1 是与底半处理过程相关的内核数据结构。bh_base 代表的指针数组中可包含 32 个不同的底半处理过程。bh_mask 和 bh_active 的数据位分别代表对应的底半处理过程是否安装和激活。如果 bh_mask 的第 N 位为 1，则说明 bh_base 数组的第 N 个元素包含某个底半处理过程的地址；如果 bh_active 的第 N 位为 1，则说明必须由调度程序在适当的时候调用第 N 个底半处理过程。

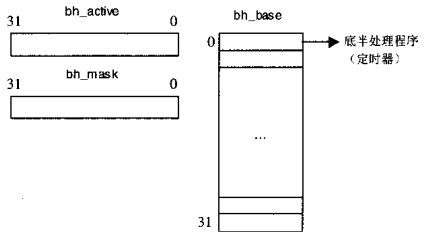


图 15-1 底半处理数据结构

bh_base 数组的索引是静态定义的，定时器底半处理过程的地址保存在第 0 个元

素中，控制台底半处理过程的地址保存在第 1 个元素中等等。典型来说，每个底半处理过程和相应的任务队列关联。当 bh_mask 和 bh_active 表明第 N 个底半处理过程已被安装且处于活动状态，则调度程序会调用第 N 个底半处理过程，该底半处理过程最终会处理与之相关的任务队列中的各个任务。因为调度程序从第 0 个元素开始依次检查每个底半处理过程，因此，第 0 个底半处理过程具有最高的优先级，第 31 个底半处理过程的优先级最低。

内核中的某些底半处理过程是和特定设备相关的，而其他一些则更一般一些。表 15-1 列出了内核中通用的底半处理过程。

表 15-1 Linux 中通用的底半处理过程

TIMER（定时器）	在每次系统的周期性定时器中断中，该底半处理过程被标记为活动状态，并用来驱动内核的定时器队列机制。
CONSOLE（控制台）	该处理过程用来处理控制台消息。
TQUEUE（TTY 消息队列）	该处理过程用来处理 tty 消息。
NET（网络）	该处理过程用于一般网络处理。
IMMEDIATE（立即）	这是一个一般性处理过程，许多设备驱动程序利用该过程对自己要在随后处理的任务进行排队。

当某个设备驱动程序，或内核的其他部分需要将任务排队进行处理时，它将任务添加到适当的系统队列中（例如，添加到系统的定时器队列中），然后通知内核，表明需要进行底半处理。为了通知内核，只需将 bh_active 的相应数据位置为 1。例如，如果驱动程序在 immediate 队列中将某任务排队，并希望运行 IMMEDIATE 底半处理过程来处理排队任务，则只需将 bh_active 的第 8 位置为 1。在每个系统调用结束并返回调用进程之前，调度程序要检验 bh_active 中的每个位，如果有任何一位为 1，则相应的底半处理过程被调用。每个底半处理过程被调用时，bh_active 中的相应为被清除。bh_active 中的置位只是暂时的，在两次调用调度程序之间 bh_active 的值才有意义，如果 bh_active 中没有置位，则不需要调用任何底半处理过程。

15.2 任务队列

任务队列是内核将任务延迟到以后处理的一种方法。任务队列和底半处理过程经常结合起来使用，例如，定时器任务队列在定时器底半处理过程中进行处理。任务队列的数据结构很简单，实际就是普通的单向链表结构，见图 15-2，每个 tq_struct 数据结构作为任务队列的节点，包含了一个例程地址和指向一些数据的指针。当任务队列中的节点被处理时，将调用例程并传递数据指针。

内核中的任何部分（例如驱动程序）都可以建立并使用任何队列，由内核建立并维护的三个一般性任务队列在表 15-2 中列出。

表 15-2 Linux 内核中三个一般性任务队列

TIMER（定时器）	该队列用来排队需要在系统时钟滴答之后尽可能快地完成的任务。每次时钟滴答时，如果该队列中包含有元素，则定时器队列底半处理例程标

	记为活动状态。在随后运行的调度程序中，定时器队列底半处理例程被调用，从而定时器队列中排队的任务也被处理。
IMMEDIATE（立即）	该队列在调度程序处理活动的底半处理程序时处理。因为 IMMEDIATE 底半处理过程的优先级较低，因此比起定时器底半处理过程，对这些任务的处理要稍微拖后一些。
SCHEDULER（调度程序）	该任务队列由调度程序直接处理。该队列用来支持系统中的其他任务队列，这种情况下，要运行的任务实际是处理某个任务队列的例程。

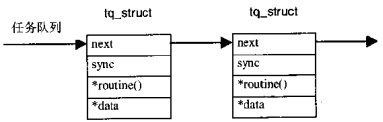


图 15-2　任务队列数据结构

在处理任务队列时，队列中第一个元素从队列中移出，并用空指针代替。移出操作必须是一个原子操作，也就是说，是不能被中断的操作。队列中每个处理例程依次调用。队列中的元素通常是静态分配的数据，因为没有内建用来丢弃已分配内存的机制，因此，任务队列的处理过程简单移向后续的链表元素。对已分配内核内存的清除工作由任务本身完成。

15.3　时间和定时器

对一个操作系统来说，它必须具有调度未来任务的能力。如果必须在相对精确的时间内调度某个任务，操作系统必须具有一定的机制来实现该功能。为了给操作系统提供这样的机制，PC 机中一般存在一个可编程的间隔定时器，定时器可以以指定的时间周期性地中断处理器。这种周期性的定时器中断在操作系统中一般称为时钟滴答，它的作用就象音乐中的节拍一样，协调着系统中所有的活动。

除此之外，操作系统还必须具备一定的接口记录系统的时间，并为程序提供时间服务。一般来讲，操作系统和计算机硬件一起维护着系统中的时间。在 PC 机中，Linux 利用 BIOS CMOS 中记录的时间（称为“硬件时钟”）作为系统启动时的时间基准，而在系统运行时，利用时钟滴答测量系统的时间（称为“软件时钟”）。例如，启动时从 CMOS 中读取的时间为 1998.2.1 0:00:00，假设系统的时钟滴答为每秒 100 次，则经过 1000 次时钟滴答之后，时间为 1998.2.1 0:00:10。Linux 利用 jiffies（瞬时）作为系统时间的测量基准，所有的时间都从 1970.1.1 0:00:00 开始计算，系统启动时，将 CMOS 中记录的时间转化为从 1970.1.1 0:00:00 算起的 jiffies 值。在 Linux 内核中，时间以格林尼

治时间记录，将格林尼治时间转换为本地时间的任务则由应用程序负责，实际上，Linux 内核中没有任何时区的概念。

利用上面的方法记录时间是有问题的，类似现在的 2000 年问题。假定一个 jiffies 等于 1/100 秒，并利用 32 位无符号长整型整数保存 jiffies 值，则可以计算得到能够记录的最大秒数为 42949672.96 秒，约合 1.38 年，因此这种方法无法在实际当中使用。实际上，Linux 的 jiffies 值由两部分组成，分别用 32 位无符号整数记录自 1970.1.1 00:00:00 开始的秒数以及秒数千分值。这样，Linux 可正确处理的时间值最大到 1970 年后的 138 年，即 2108 年，而时间的计量也可精确到千分之一秒。在到达 2108 年之前，人们早就会想出更好的办法来计时。

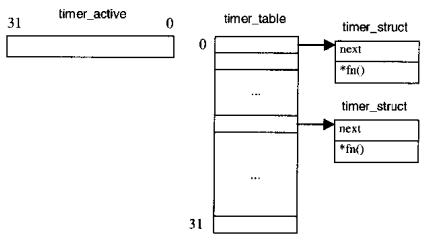


图 15-3 Linux 中的两种系统定时器，(a) 老定时器结构；(b) 新定时器结构

Linux 具有两种类型的系统定时器，这两种定时器均具有对应的例程队列，必须在到达给定的系统时间时调用，但这两种定时器的具体实现方法有一些不同。图 15-3 说明了这两种定时器机制。第一种定时器机制，也是老的定时器机制，利用一个可保存 32 个指

针的数组定义定时器。每个指针可指向一个 timer_struct 结构，而 timer_active 是活动定时器的掩码（这和底半处理过程的数据结构类似）。数组中的元素通常是静态定义的，在系统初始化过程中填充这些元素。第二种定时器机制，是比较新的定时器机制，它用链表结构以定时器到期时间的升序组织定时器。

这两种定时器均利用 jiffies 值作为定时器的到期时间。如果某个定时器要在 5 秒之后到期，则必须将 5 秒转换为对应的 jiffies 值，加上当前的系统时间后（以 jieffies 为单位），得到的便是该定时器到期的系统时间。每次系统时钟的滴答中，定时器底半处理过程被标记为活动状态，当调度程序随后运行时，定时器队列将得到处理。定时器底半处理过程要处理上述两种类型的定时器。对老的系统定时器，检验 timer_active 中的相应位，以便确定活动的定时器。如果活动定时器已到期（到期时间大于或等于当前系统的 jiffies），则调用对应的定时器例程，并清除 timer_active 中的相应位。对新的系统定时器，检验链表中的 timer_list 数据结构，每个到期的定时器从链表中移出，而对应的定时器例程被调用。新的定时器机制可以将参数传递到定时器例程中。

15.4　等待队列

在进程的执行过程中，有时难免要等待某些系统资源。例如，如果某个进程要读取一个描述目录的 VFS 索引节点，而该节点当前不再缓冲区高速缓存中，这时，该进程就必须等待系统从包含文件系统的物理介质中获取索引节点，然后才能继续运行。

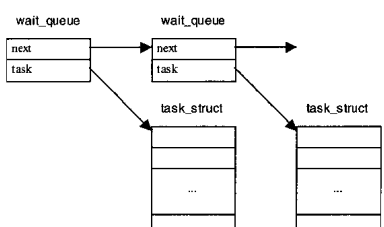


图 15-4　Linux 中的等待队列

Linux 利用一个简单的数据结构来处理这种情况。如图 15-4 所示，是 Linux 中的等待队列，该队列中的元素包含一个指向进程 task_struct 结构的指针，以及一个指向等待队列中下一个元素的指针。

对于添加到某个等待队列的进程来说，它可能是可中断的，也可能是不可中断的。当可中断的进程在等待队列中等待时，它可以被诸如定时器到期或信号的发送等事件中断。如果等待进程是可中断的，则进程状态为 INTERRUPTIBLE；如果等待进程是不可中断的，则进程状态为 UNINTERRUPTIBLE。

15.5 Buzz 锁

Buzz 锁，也即“自旋锁”，是用来保护数据和代码段的最原始方法。利用 Buzz 锁，可每次只允许一个进程进入关键代码段。Linux 利用 Buzz 锁限制对某些数据结构域的访问，并利用一个整型域作为锁。每个要进入关键代码段的进程首先试图将该整数的值从 0 修改为 1。如果当前值为 0，则进程可以立即进入关键代码段，而整数值变为 1；如果当前值为 1，则说明其他进程已进入该关键代码段，进程循环检查整数值，直到值变为 0，这时进程可修改值为 1，并进入关键代码段。进程在退出关键代码段时，将 Buzz 锁的值修改为 0，以便其他进程可以进入该关键段。

对用来保存 Buzz 锁的内存的访问必须是原子操作，也就是不能被中断的操作。大部分 CPU 提供特殊的指令支持 Buzz 锁的原子操作，当然，也可以利用非缓存的内存实现 Buzz 锁。

15.6 信号量

信号量也用来保护关键代码或数据结构。我们都知道，关键代码段的访问，是由内核代表进程完成的，如果让某个进程修改当前由其他进程使用的关键数据结构，其后果是不堪设想的。可以利用 Buzz 锁实现对关键代码段和数据的互斥访问，但是，如前所述，Buzz 锁是一种非常原始的方法，并且由于对 Buzz 锁值的循环重复测试，无法为系统提供更好的性能。取而代之，Linux 利用信号量实现对关键代码和数据的互斥访问，同一时刻只能有一个进程反问某个关键资源，所有其他要访问该资源的进程必须等待直到该资源空闲为止。等待进程处于暂停状态，而系统中的其他进程则可运行如常。

Linux 信号量数据结构中包含如表 15-3 所示的信息。

表 15-3 Linux 信号量数据结构中包含的信息

count（计数）	该域用来跟踪希望访问该资源的进程个数。正值表示资源是可用的，而负值或零表示有进程正在等待该资源。该计数的初始值为 1，表明同一时刻有且只能有一个进程可访问该资源。进程要访问该资源时，对该计数减 1，结束对该资源的访问时，对该计数加 1。
waking（等待唤醒计数）	等待该资源的进程个数，也是当该资源空闲时等待唤醒的进程个数。
等待队列	某个进程等待该资源时被添加到该等待队列中。
lock（锁）	用来实现对 waking 域的互斥访问的 Buzz 锁。

假定该信号量的初始计数为 1，第一个要求访问资源的进程可对计数减 1，并可成功访问资源。现在，该进程是“拥有”由信号量所包含的资源或关键代码段的进程。当该进

程结束对资源的访问时，对计数加 1。最优的情况是没有其他进程和该进程一起竞争资源所有权。Linux 针对这种最常见的情况对信号量进行了优化，从而可以让信号量高效工作。

当某个进程当前拥有资源时，如果其他进程要访问该资源，它首先将信号量计数减 1。因为现在计数值是负值（-1），因此该进程不能进入关键段，相反，它必须等待资源当前的拥有者释放所有权。Linux 将等待进程置入休眠状态，直到所有者退出关键段时唤醒。等待进程将自己添加到信号量的等待队列中，然后循环检测信号量 waking 域的值，当 waking 非零时调用调度程序。

关键段的所有者增加信号量的计数，如果计数大于或等于 0，表明其他进程正在处于休眠状态而等待该资源。在最优情况下，信号量的计数将返回到初值 1，因此没有必要进行额外的工作。在其他情况下，资源的拥有者要增加 waking 计数，并唤醒处于信号量等待队列中的休眠进程。当休眠进程被唤醒之后，waking 计数的当前值为 1，因此可以进入关键段，这时，它减小 waking 计数，将 waking 计数的值还原为 0。对信号量 waking 域的互斥访问利用信号量的 lock 域作为 Buzz 锁而实现。

15.7　模块

从结构上来讲，操作系统有微内核结构和单块结构之分，Windows NT 和 MINIX 是典型的微内核操作系统，而 Linux 则是单块结构的操作系统。微内核结构可方便地在系统中添加新的组件，而单块结构却不容易做到这一点。为此，Linux 支持可动态装载和卸载的模块。利用模块，可方便地在内核中添加新的组件或卸载不再需要的内核组件。大多数 Linux 内核模块是设备驱动程序以及伪设备驱动程序（网络驱动程序、文件系统等）。

利用内核模块的动态装载性具有如下优点：

- 将内核映象的尺寸保持在最小，并具有最大的灵活性；
- 便于检验新的内核代码，而不需重新编译内核并重新引导。

但是，内核模块的引入也带来了如下问题：

- 对系统性能和内存利用有负面影响；
- 装入的内核模块和其他内核部分一样，具有相同的访问权限，因此，差的内核模块会导致系统崩溃；
- 为了内核模块访问所有内核资源，内核必须维护符号表，并在装入和卸载模块时修改这些符号表；
- 有些模块要求利用其他模块的功能，因此，内核要维护模块之间的依赖性。
- 内核必须能够在卸载模块时通知模块，并且要释放分配给模块的内存和中断等资源；
- 内核版本和模块版本的不兼容，也可能导致系统崩溃，因此，严格的版本检查是必需的。

15.7.1　装载模块

有两种方法可用来装载模块：

- 利用 **insmod** 命令手工将模块插入内核；
- 由内核在必要时装载模块，称为“需求装载”。

利用需求装载时，内核通过向守护进程 **kerneld** 发送请求而装载适当的模块。这一守护进程实际是一个普通的用户进程，通常在系统引导时启动，并具有超级用户权限。**kerneld** 进程在启动时为内核打开一个进程间通讯通道，内核可以利用该通道向 **kerneld** 进程发送任务的执行请求。**kerneld** 进程的首要任务是装载和卸载模块，另外，该进程也负责其他一些任务，例如打开和关闭 PPP 链接等。**kerneld** 实际运行相应的工具完成任务，对装载模块而言，它利用的是 **insmod** 命令。因此，该进程实际是代表内核完成某些任务的代理。

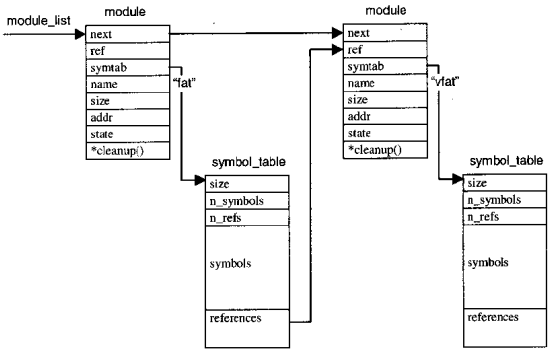


图 15-5 装入 VFAT 和 FAT 之后的内核模块表

执行 **insmod** 命令时，必须指定要装载模块的位置；对需求装载的内核模块，通常保存在 /lib/modules/kernel-*version*。和系统的其他程序一样，内核模块实际是经连接的目标文件，但模块是可重定位的，也就是说，为了让装入的模块和已有的内核组件之间可以互相访问，模块不能连接为从特定地址执行的映象文件。模块可以是 a.out 或 elf 格式的目标文件。**insmod** 利用一个特权系统调用，可找到内核的导出符号表，符号成对出现，一个是符号名称，另外一个是符号的值，例如符号的地址。内核维护一个由 module_list 指针指向的 module 链表，其中第一个 module 数据结构保存有内核的导出符号表（见图 15-5）。并不是所有的内核符号均在符号表中导出，而只有一些特殊的符号才被添加到符号表中。例如，“request_irq”是一个导出符号，它是一个内核例程，可由驱动程序申请控制某个特定的系统中断。利用 **ksyms** 命令或查看 /proc/ksyms 文件内容，可非常方便地看到所有的内核导出符号及其符号值。利用

ksyms 命令，不仅可以看到内核的所有符号，也可以看到只由以装载模块导出的符号。**insmod** 命令将模块读到它本身的虚拟内存中，然后利用内核导出的符号表，修正尚未解析的对内核例程的引用。这种修正实际是对模块在内存中的映象进行修正，**insmod** 将符号的地址写入模块中适当的位置而实现修正。

insmod 命令修正模块对内核符号的引用之后，再次利用特权系统调用请求内核分配足够的物理内存空间保存新的模块。内核将分配新的 module 数据结构以及足够的内核内存，并将新模块添加在内核模块表的末尾。新的内核模块标记为 UNINITIALIZED（未初始化）。图 15-5 是装入 VFAT 和 FAT 模块之后的内核模块表。图中并没有表示出第一个模块，它实际是一个伪模块，仅仅用来保存内核的导出符号表。利用 **lsmod** 命令可列出所有已装载的内核模块以及它们的内在依赖性。内核为新模块分配的内核内存映射到 **insmod** 进程的地址空间中，这样，**insmod** 就可以将模块复制到新分配的内存中。**insmod** 还对模块进行重新定位，经重定位之后，新的模块就可以从新分配的内核地址开始运行了。显然，模块不能期望自己能够在不同的 Linux 系统，或前后两次装入时被装载到相同地址，重定位操作可通过对模块映象中适当的地址进行修正而解决这一问题。

新的模块也要向内核导出符号，由 **insmod** 建立相应的符号表。每个内核模块必须包含模块的初始化和清除例程，这些例程作为每个模块均具备的例程而不被导出，但 **insmod** 必须知道它们的地址，并将地址传递给内核。**insmod** 同样利用特权系统调用将模块的初始化和清除例程地址传递给内核。

新的模块添加到内核之后，它必须更新内核符号集并修改使用新模块的模块。由其他模块依赖的模块必须在自身符号表的末尾维护一个引用表，并指向其他模块的 module 结构。例如，图 15-5 表明 VFAT 文件系统模块依赖于 FAT 文件系统模块，因此，FAT 模块包含一个对 VFAT 模块的引用，该引用在装入 VFAT 模块时添加。

内核成功调用模块的初始化例程之后继续模块的安装，最后，模块状态被设置为 RUNNING（运行）。模块的清除例程保存在 module 数据结构中，在卸载模块时由内核调用。

15.7.2　卸载模块

和模块的装载类似，可利用 **rmmod** 命令手工卸载模块，当对需求装载的模块则由 **kerneld** 在不再需要时自动卸载。每次 kerneld 的空闲定时器到期时，它会利用系统调用将当前不再使用的需求装载模块从内核中移走。启动 **kerneld** 时指定该定时器的时间，通常的时间为 180 秒。

如果内核的其他部分依赖于装入的模块时，该模块不能卸载。例如，如果挂装了 FAT 文件系统，则不能卸载已装入的 FAT 文件系统模块。**lsmod** 命令的输出会显示已安装模块的使用计数，例如：

```
Module:      #pages:        Used by:
msdos           5              1
vfat            4              1  (autoclean)
fat             6              2  (autoclean)
```

使用计数就是依赖于该模块的内核实体个数。模块的使用计数保存在模块映象的第一个长整型中。但是，这一长整型中还包含有 AUTOCLEAN 和 VISITED 标志。这两个标

志均由需求装载的模块使用。具有 AUTOCLEAN 标志的模块是系统认为可以自动卸载的模块。具有 VISITED 标志的模块表明正由其他内核组件使用，当任何其他内核组件使用该模块是设置该标志。当 **kerneld** 请求系统移走不使用的需求装载模块时，系统首先寻找可以移走的模块，但系统只查看标记为 AUTOCLEAN，并且状态处于 RUNNING 的模块。如果上述模块的 VISITED 标志被清除，则系统将卸载该模块，否则系统会清除 VISITED 标志并查看下一个模块。

假定某个模块是可卸载的，则系统调用其清除例程释放分配该模块的内核资源。相应的 module 数据结构被标志为 DELETED 并从内核模块链表中断开。所有由该模块依赖的模块，系统会修改它们的引用表以便取消依赖性。最后，系统释放模块的内核内存。

15.8 相关系统工具和系统调用

15.8.1 显示和设置时间

如前所述，Linux 内核中保持着格林尼治时间，要获得本地时间，系统必须维护时区信息。系统时区一般由符号链接 /etc/localtime 确定。但是，不同的用户可具有自己私有的时区设置，这一般通过 **TZ** 环境变量来设置。

date 命令可显示当前日期和时间，如：

```
$ date
Wed Feb 17 09:56:45 CST 1999
$
```

上述命令执行结果显示北京时间。利用 **-u** 选项，可显示格林尼治时间，如：

```
$ date -u
Wed Feb 17 01:56:54 UTC 1999
$
```

也可以利用 **date** 命令设置内核的软件时钟：

```
# date 07142157
Sun Jul 14 21:57:00 CST 1999
# date
Sun Jul 14 21:57:02 CST 1999
#
```

date 命令可显示和设置软件时钟，而 **clock** 命令则用来同步硬件和软件时钟。系统引导时，可利用该命令读取硬件时钟并设置软件时钟。如果需要同时设置硬件和软件时钟，则可首先利用 **date** 命令设置软件时钟，然后利用 **clock** 命令的 **-w** 选项设置硬件时钟。

通常来说，PC 机的硬件时钟，即 BIOS 所维护的时钟记录本地时间，如果硬件时钟记录格林尼治时间，则必须利用 **clock** 命令的 **-u** 选项，否则 **clock** 会认为硬件时间为本地时间。

由于 Linux 系统软件时钟的实现利用了定时器中断，如果系统中运行的进程过多，则对定时器中断的响应时间会增加，从而使软件时钟变得不可靠起来。但硬件时钟通常是比较精确的，因此，如果经常引导 Linux 系统，则软件时钟相对精确。如果要调整硬件

时钟，最简单的方法是修改 BIOS 时钟，也可以利用上述的方法，先用 **date** 设置软件时钟，再利用 **clock** 命令的 **-w** 选项设置硬件时钟。

在网络环境中，有一些计算机可提供非常精确的时间，这些计算机通常称为“时间服务器”。利用 **rdate** 和 **netdate** 命令可同步本地计算机和时间服务器之间的时钟。

15.8.2　管理内核模块

利用 **insmod** 命令可手工装入内核模块；利用 **lsmod** 可查看当前装入的内核模块以及需求装载模块的使用计数及标志信息；利用 **rmmod** 则可以卸载指定的模块。

15.8.3　系统调用

表 15-4 简要列出了和时间、定时器以及模块相关的系统调用。标志列中各字母的意义可参见 表 10-1 的说明。

表 **15-4**　相关系统调用

系统调用	说明	标志
adjtimex	设置或获取内核时间变量	c
create_module	为可装载内核模块分配空间	-
delete_module	卸载内核模块	-
get_kernel_syms	获取内核符号表或其大小	-
getitimer	获取间隔定时器的值	mc
gettimeofday	获取自 1970.1.1 以来的时区和秒数	mc
init_module	插入可装载内核模块	-
setitimer	设置间隔定时器	mc
settimeofdat	设置自 1970.1.1 以来的时区和秒数	mc
stime	设置自 1970.1.1 以来的秒数	mc
time	获取自 1970.1.1 以来的秒数	m+c

第十六章

引 导 和 关 机

这一部分前几章的内容主要集中在 Linux 内核上，同时也简要阐述了与各内核组件相关的系统工具和系统调用，这部分后面几章的内容主要讲述尚未在前几章中论及的、与系统管理有关的内容。

本章解释 Linux 系统的引导和关机过程。不同系统平台的引导过程略微有些不同，本章以 Intel 平台为主讲述 Linux 系统的引导过程。由于 Linux 使用缓冲区高速缓存技术，而且是一个多用户系统，因此，关机时必须遵循正确的过程，否则会造成数据丢失和文件系统的破坏。

和系统的引导、关机过程相关的概念还有单用户模式、运行级别等。对这些概念的正确理解是配置和维护系统的必要条件。

16.1 Linux 的引导过程

一般而言，操作系统的引导过程分两个步骤。首先，计算机硬件经过开机自检（POST）之后，从软盘或硬盘的固定位置装载一小段代码，这段代码一般称为“引导装载器”。然后，由引导装载器负责装入并运行操作系统。引导装载器非常小，一般只有几百个字节，而操作系统庞大而复杂。利用上述两阶段的引导过程，可将计算机中的固化软件保持得足够小，同时也便于实现对不同操作系统的引导。

不同计算机平台引导过程的区别主要在于第一阶段的引导过程。对 PC 机上的 Linux 系统而言，计算机（即 BIOS）负责从软盘或硬盘的第一个扇区（即引导扇区）中读取引导装载器，然后，由引导装载器从磁盘或其他位置装入操作系统。

对典型的 PC 机 BIOS 而言，可配置为从软盘或从硬盘引导。从软盘引导时，BIOS 读取并运行引导扇区中的代码。引导扇区中的代码读取软盘前几百个块（依赖于实际的内核大小），然后将这些代码放置在预先定义好的内存位置。利用软盘引导 Linux 时，没有文件系统，内核处于连续的扇区中，这样安排可简化引导过程。但是，如果利用 LILO（LInux LOader）也可从包含文件系统的软盘上引导 Linux。

从硬盘引导时，由于硬盘是可分区的，因此引导过程比软盘复杂一些。BIOS 首先读取并运行硬盘主引导记录中的代码，这些代码首先检验主引导记录中的分区表，寻找到活动分区（即标志为可引导分区的分区），然后读取并运行活动分区之引导扇区中的代码。活动分区引导扇区的作用和软盘引导扇区的作用一样：从分区中读取内核映象并启动内

核。和软盘引导不同的是，内核映象保存在硬盘分区文件系统中，而不象软盘那样保存在后续的连续扇区中，因此，硬盘引导扇区中的代码还需要定位内核映象在文件系统中的位置，然后装载内核并启动内核。通常，最常见的方法是利用 **LILO** 完成这一阶段的引导。**LILO** 可配置为装载启动不同的内核映象，甚至可以启动不同的操作系统，也可以通过 **LILO** 指定内核命令行参数。有关 **LILO** 的内容，可参见有关的 HOWTO 文档。

从软盘引导和从硬盘引导各有优点，但常见的引导方式是从硬盘引导。但在多重引导系统中（多个操作系统），从硬盘引导可能带来一些麻烦，因此，通常的做法是，首先从软盘引导，等系统稳定之后，再安装 **LILO** 从硬盘上引导。

Linux 内核装入之后，Linux 内核进行硬件和设备驱动程序的初始化，然后运行 **init**。**init** 是 Linux 内核启动的第一个用户级进程，其进程标识号始终为 1。该进程在系统引导和关机过程中扮演重要角色。在本章后面的小节中，将详细介绍 **init** 进程。Linux 内核进行的初始化工作可大体描述如下：

1. Linux 内核一般是压缩保存的，因此，它首先要进行自身的解压缩。内核映象前面的一些代码完成解压缩。
2. 如果系统中安装有可支持特殊文本模式的、且 Linux 可识别的 SVGA 卡，Linux 会提示用户选择适当的文本显示模式。但是，如果在内核的编译过程中预先设置了文本模式，则不会提示选择显示模式。该显示模式也可通过 **LILO** 或 **rdev** 设置。
3. 内核接下来检测其他的硬件设备，例如硬盘、软盘和网卡等，并对相应的设备驱动程序进行配置。这时，内核会输出一些硬件信息，类似下面的输出：

```
LILO boot:
Loading linux.
Memory: sized by int13 088h
Console: 16 point font, 400 scans
Console: colour VGA+ 80x25, 1 virtual console (max 63)
pcibios_init : BIOS32 Service Directory structure at 0x000f7510
pcibios_init : BIOS32 Service Directory entry at 0xfd7d2
pcibios_init : PCI BIOS revision 2.10 entry at 0xfd9e6
Probing PCI hardware.
Calibrating delay loop.. ok - 231.83 BogoMIPS
Memory: 30760k/32704k available (748k kernel code, 384k reserved, 812k data
)
Swansea University Computer Society NET3.035 for Linux 2.0
NET3: UNIX domain sockets 0.13 for Linux NET3.035.
Swansea University Computer Society TCP/IP for NET3.034
IP Protocols: IGMP, ICMP, UDP, TCP
Linux IP multicast router 0.07.
VFS: Diskquotas version dquot_5.6.0 initialized
Checking 386/387 coupling... Ok, fpu using exception 16 error reporting.
Checking 'hlt' instruction... Ok.
Linux version 2.0.36 (root@porky.Red Hat.com) (gcc version 2.7.2.3) #1 Tue
 Oct 13 22:17:11 EDT 1998
Starting kswapd v 1.4.2.2
Serial driver version 4.13 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16550A
Real Time Clock Driver v1.09
Ramdisk driver initialized : 16 ramdisks of 4096K size
ide: i82371 PIIX (Triton) on PCI bus 0 function 57
```

```
    ide0: BM-DMA at 0xfcd0-0xfcd7
    ide1: BM-DMA at 0xfcd8-0xfcdf
hda: HITACHI_DK237A-32, 3102MB w/512kB Cache, CHS=788/128/63, UDMA
hdc: CD-224E, ATAPI CDROM drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
ide1 at 0x170-0x177,0x376 on irq 15
Floppy drive(s): fd0 is 1.44M
FDC 0 is a post-1991 82077
md driver 0.36.3 MAX_MD_DEV=4, MAX_REAL=8
scsi : 0 hosts.
scsi : detected total.
Partition check:
 hda: hda1 hda2 < hda5 > hda3 hda4
```

4. 接下来，内核挂装 root 文件系统。root 文件系统的位置可在编译内核时指定，也可通过 **LILO** 或 **rdev** 指定。文件系统的类型可自动检测。如果由于某些原因挂装失败，则内核启动失败，最终会终止系统。root 文件系统一般挂装为只读文件系统。
5. 此后，内核启动 **init** 进程（位于 /sbin/init）。**init** 进程的启动工作在后面的小节中详细讲述。一般而言，它要启动一些重要的后台守护进程。
6. 然后，**init** 切换到多用户模式，并为每个虚拟控制台和串行线路启动一个 **getty** 进程。**getty** 进程管理用户从虚拟控制台和串行终端上的登录。根据不同的配置，**init** 也可以启动其他进程。
7. 至此，系统的引导过程结束。

16.2 关机

对 Linux 系统来说，必须始终以正确的方式关机。因为 Linux 利用了缓冲区高速缓存，写入磁盘的数据并不立刻写入物理磁盘中，因此，直接断电会导致数据的丢失或文件系统的破坏。

对多任务系统来说，可能有许多后台进程处于运行状态，只有利用正确的关机操作才能保证所有的后台进程能够保存自己的数据。

在 Linux 系统中，可利用 **shutdown** 命令关机。根据不同的应用环境，**shutdown** 可有不同的使用方法，但通常有两种使用方法：

- 如果使用某 Linux 系统的用户只有一个，则通常的过程如下：退出所有正在运行进程；注销所有的虚拟终端；在某虚拟终端上以 root 身份登录；保证处于 root 文件系统，然后运行命令 **shutdown -h now**，这时，系统立即进入关机过程。
- 如果系统中现有许多用户，则应当使用 **shutdown -h +*time* *message*** 命令。该命令可通知所有的用户将在 ***time*** 指定的时间内关机，提醒用户退出所有程序，并保存数据。***time*** 是以分钟计的时间，***message*** 是显示在每个终端以及 **xterm** 上的警告信息。该警告信息可在关机前自动重复。

当实际的关机过程开始后，所有的文件系统（除 root 文件系统外）均被卸挂，所有的用户进程被强行杀掉，守护进程被关闭。之后，**init** 打印信息表明可以关闭电源了，这时才是真正关闭电源的时候。

也有无法正常关机的情况发生，如果内核处于应急状态而终止 CPU，这时就不得不关闭电源，然后再重新启动计算机。但是，如果由于其他原因导致无法正常关机，而内核和守护进程仍然处于运行状态，这时，应当等待足够的时间（十几分钟）让 **update** 守护进程刷新缓冲区缓存之后，再关闭电源。

16.3 重新引导

Linux 系统的重新引导有两种办法，一种是先关机，然后再打开电源，重新引导；另外一种是利用 **shutdown** 命令的 **-r** 选项。

在大多数 Linux 系统中，DOS 中常见的 CTRL+ALT+DEL 热启按键可运行 **shutdown -r now** 命令，从而可以重新引导系统。但是，该按键的动作是可以配置的，可以将系统配置为按 CTRL+ALT+DEL 键时不运行 **shutdown -r now** 命令，也可以限定按上述组合键时运行重新引导命令的用户。

16.4 紧急引导软盘

在错误安装 LILO 等情况下，Linux 系统可能无法从硬盘引导，这时就必须利用其他途径引导。通常在 PC 机中，可从软盘引导系统并解决问题。为此，必须具备紧急引导软盘。

大多数 Linux 的商业发行版本可以在安装过程中建立紧急引导软盘。但是，这些软盘中常常只包含系统内核，而用来解决问题的程序可能不在该软盘中。因此，有时需要建立定制引导软盘，同时也需要经常更新自己的紧急引导软盘。

16.5 init

init 是内核启动的第一个用户级进程，它在 Linux 系统中扮演着重要的角色。尽管 **init** 对系统至关重要，但绝大多数情况下不需要用户过多关心它的配置及运行。通常而言，只有在需要建立串行终端，设置拨入和拨出调制解调器，或者改变默认运行级别时才需要配置 **init**。

内核结束自身的引导过程之后（装入内存，初始化硬件和设备驱动程序之后），启动用户级进程 **init**。通常来说，内核首先寻找 /sbin/init。如果内核无法找到 **init**，则试图寻找 /bin/sh，在无法找到 **sh** 的情况下，系统的引导过程失败。

init 启动之后，首先完成如下任务：

1. 检查文件系统；
2. 清除 /tmp；
3. 启动各种服务；
4. 为每个终端和虚拟控制台启动一个 **getty**。**getty** 负责用户的登录，第十七章将详细介绍登录和注销过程。

在上述任务完成之后，即说明系统的引导过程结束。

每当某个终端或虚拟控制台上的用户注销之后，**init** 进程为该终端或虚拟控制台重

新启动一个 **getty**，以便能够让其他用户登录。另外，**init** 进程还负责管理系统中的“孤儿”进程。如果某个进程创建子进程之后，在子进程终止之前终止，则子进程成为孤儿进程。**init** 进程负责“收养”该进程，即孤儿进程会立即成为 **init** 进程的子进程。这种处理有重要的技术原因，一方面是为了易于处理进程表和进程树。

init 进程的变种较多，大多数 Linux 的发行版本采用 **sysvinit**（由 Miquel van Smoorenburg）编写，由于基于 System V 的设计而得名。UNIX 的 BSD 版本有不同的 **init**。主要区别在于是否具有运行级别：System V 有运行级别，而 BSD 没有运行级别。但这种区别并不是本质的区别。本章所描述的 **init** 基于 **sysvinit**。

16.6　启动 getty：/etc/inittab 文件

在 **init** 启动时，它读取 /etc/initab 配置文件。在系统的正常运行过程中，如果为 **init** 进程发送 SIGNHUP 信号（以 root 用户身份运行 **kill -HUP 1**），则 **init** 进程会重新读取 /etc/inittab 文件，从而在不经重新引导系统的情况下，即可以让 **init** 的配置文件生效。

/etc/inittab 文件中的每一行由四个字段组成，各字段之间由分号分隔：

```
id:runlevels:action:process
```

上述各字段的含义在表 16-1 中描述。另外，/etc/inittab 文件中可包含空行，以“#”开头的行作为注释行。

表 16-1　/etc/inittab 文件中各字段的含义

id	该字段作为 inittab 文件中各行的标识符。对定义 getty 的各行来说，该标识符指定 **getty** 运行的终端（即设备文件名称中 /dev/tty 之后的字符）。对其他行来说，除了具有长度限制之外，没有特殊要求，但是，和数据库表的主键类似，该字段应具有唯一值。
runlevels	该字段指定运行级别，各运行级别由单个的数字表示，可表示多个运行级别，但不能包含任何分隔符。运行级别将在下节中描述。
action	该字段指定该行的动作，例如，respawn 指下一字段指定的命令退出后，需要重新运行该命令，而 once 则表示下一字段的命令只运行一次。
process	该字段指定要运行的命令。

如果要在第一个虚拟控制台上启动 getty，并在所有通常的多用户运行级别（2 ~ 5）上运行，则应如下定义：

```
1:2345:respawn:/sbin/getty 9600 tty1
```

第一个字段取值为 1，表明该行用来定义 /dev/tty1；第二个字段表明该行应用于多个运行级别 2、3、4 和 5；第三个字段表明当 **getty** 退出之后，需要再次运行（从而可让用户登录、注销之后再次登录）；最后一个字段说明在第一个虚拟终端上运行 **getty** 的命令。

如果要为某个系统添加终端或拨入调制解调器线路，则需要在 /etc/inittab 中添加相应行，每一行可定义一个终端或一条拨入线路。具体信息可参见手册页：init(8)、

inittab（5）和 getty（8）。

16.7 运行级别

运行级别指 **init** 以及整个系统的一个运行状态，它定义了系统所提供的服务。运行级别由数字表示，各数字所代表的运行级别及其含义由表 16-2 给出。

表 16-2 运行级别

0	终止系统。
1	单用户模式（为特殊管理任务所用）。
2~5	通常的操作（用户定义）。
6	重新引导。

由用户定义的运行级别（2~5）可由系统管理员自行决定。有些管理员利用运行级别定义可运行的子系统，例如，是否运行 X，是否可操作网络等。但通常不需要修改默认的运行级别。不同的 Linux 发行版本对运行级别 2~5 的定义可能不同。例如，Red Hat Linux 5.x 定义运行级别 5 为进入 X Window，从而以 **xdm** 取代字符的虚拟终端并以图形方式登录。

如下所示，是 /etc/inittab 中定义运行级别的某行：

```
12:2:wait:/etc/init.d/rc 2
```

上述行的第二个字段定义该行应用于运行级别 2。第三字段中的 wait 表明对后面定义的命令，**init** 只运行一次，并且进入该运行级别之后，**init** 要等待该命令结束。

第四个字段中的命令实际指定了建立某个运行级别的硬性工作，它要启动尚未启动的服务，终止不应当在新的运行级别中运行的服务等。

当 **init** 启动时，它要在 /etc/inittab 中寻找指定默认运行级别的行：

```
id:3:initdefault:
```

根据默认的运行级别，**init** 将在正常启动之后运行包含该默认运行级别的所有行。

利用 **telinit** 或 **init** 命令，可切换到其他的运行级别，例如：

```
telinit 1
```

将进入运行级别 1，也即单用户模式。

通过为内核指定 **single** 或 **emergency** 等命令行参数，可请求 **init** 在启动时进入非默认的运行级别，从而选择单用户模式。单用户模式将在 16.10 小节中详细描述。

16.8 /etc/inittab 文件的特殊设置

/etc/inittab 具有一些特殊的设置，可让 **init** 响应某些特殊情况。这些特殊设置由第三个字段的特殊关键词指定。表 16-3 给出了一些特殊关键词。

表 16-3 /etc/initab 中的特殊设置关键词

powerwait	允许在电源失效的情况下，由 **init** 关闭系统。这一关键词假定使用了不间断电源（UPS），并且监视 UPS 的软件通知 **init** 电源已关闭。
ctrlaltdel	当用户在控制台键盘上按 CTRL+ALT+DEL 键时，允许 **init** 重新引导系统。但系统管理员可将系统配置为忽略该按键组合。
sysinit	系统引导之后要运行的命令。该命令通常清除 /tmp 目录。

有关 /etc/inittab 文件特殊设置关键词的详细信息，可参见手册页 inittab (5)。

16.9 单用户模式

单用户模式，即运行级别 1，是最重要的运行级别，在这一级别，只有系统管理员可以使用计算机，并且只有非常少的系统服务处于运行状态。对于一些系统管理任务来说，单用户模式是不可缺少的，例如，如果要在 /usr 分区上运行 **fsck**，则需要卸挂该文件系统。除非将所有的系统服务杀掉，否则无法卸挂该文件系统。

利用 **telinit** 可请求处于运行状态的系统进入单用户模式。在引导时，通过为内核提供 **single** 或 **emergency** 命令行参数，也可进入单用户模式。

引导时，如果自动的 **fsck** 失败，**init** 运行的引导脚本将自动进入单用户模式。在这种情况下，通过进入单用户模式，可避免使用坏的文件系统。

出于安全性考虑，正确配置的系统会在进入单用户模式之前询问 root 用户的密码。

第十七章

登录和注销

本章描述在登录和注销过程中，后台进程、日志文件和配置文件之间的交互情况。

17.1 终端登录

图 17-1 描述了通过终端登录时，**init**、**getty**、**login** 以及 shell 之间的交互情况。参照图 17-1，用户通过终端登录的过程可描述如下：

1. **init** 确保为每个终端连接（或虚拟终端）运行一个 **getty** 程序。
2. **getty** 监听对应的终端并等待用户准备登录。
3. **getty** 输出一条欢迎信息（保存在 /etc/issue 中），并提示用户输入用户名，最后运行 **login** 程序。
4. **login** 以用户作为参数，提示用户输入密码。
5. 如果用户名和密码相匹配，则 **login** 程序为该用户启动 shell。否则，**login** 程序退出，进程终止。
6. **init** 程序注意到 **login** 进程已终止，则会再次为该终端启动 **getty**。

在上述过程中，唯一的新进程是 **init** 利用 **fork** 系统调用建立的进程，而 **getty** 和 **login** 仅仅利用 **exec** 系统调用替换了正在运行的进程。

串行线路需要一个单独的程序来感知用户的登录请求，这是因为感知过程稍微复杂一些。**getty** 还需要适应连接速度以及其他设置，这对拨入连接来说尤其重要，因为这些参数可能会在不同的调用过程中改变。

getty 和 **init** 程序一样，也有许多版本，各种版本各有其优缺点。如果要利用拨入连接，则应当了解不同 **getty** 之间的差别。

17.2 网络登录

当连接到网络中的计算机相互之间进行通讯时，每台计算机上参与通讯的程序之间通过称为“虚拟连接”的链路进行通讯。这种虚拟连接可看成是一条假想的电缆，在一条实际的电缆中，可以同时存在许多不同的虚拟连接。不同的程序可在互不影响的情况下通过专属于自己的虚拟连接进行通讯。在同一物理电缆上，甚至可以有多台计算机共享电缆而互不影响。

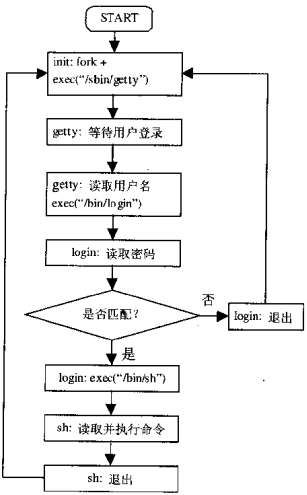


图 17-1 通过终端登录时，init、getty、login 以及 shell 之间的交互情况

显然，虚拟连接只有当两个计算机希望建立通讯连接时才存在。这种网络连接的特点，使得不可能象通常的登录一样，为每一条可能的虚拟连接事先准备一个 **getty** 和 **login** 程序。因此网络登录和通常的登录有重要的区别。

实际上，由一个单独的 **inetd** 进程（对应于 **getty**）处理所有的网络登录。当该进程注意到有引入的网络登录时，它会启动一个新的进程处理单个连接，而原先的进程则继续监听新的登录请求。

但是，每个网络登录的协议可能不同，例如用户可通过 **telnet** 登录，也可通过 **rlogin** 登录。除了网络登录之外，不同的网络服务（如 FTP、HTTP 等）具有自己不同的协议和虚拟连接，因此，为每个不同的虚拟连接类型建立单独的监听进程是无法满足需求的，相反，只有一个可区别不同连接类型的进程，即 **inetd**，负责监听连接并启动正确的程序来提供服务。有关 **inetd** 的详细信息可参阅其他资料。

17.3 login 程序

login 程序的主要任务是验证用户，并通过设置串行线路的许可和启动 shell 为用户设置初始环境。

login 程序的部分任务是输出文件 /etc/motd 的内容，并检查电子邮件。如果在用户的主目录中建立文件 .hushlogin，则可以避免执行上述任务。

如果存在文件 /etc/nologin，则禁止登录。该文件通常由 **shutdown** 或相关程序建立。如果 **login** 发现该文件存在，则拒绝接受登录，在退出之前，**login** 会输出该文件中的内容。

login 可通过 **syslog** 记录所有的失败登录请求。通过 root 用户的设置，**login** 还可以记录所有的登录情况。这两种登录日志可有效跟踪系统入侵者。

当前登录的用户列举在文件 /var/run/utmp 中。该文件的内容在系统重新引导或关机之前有效，当系统引导时，该文件被清除。该文件列举所有的用户、登录终端（或网络连接），以及其他有用的信息。系统命令 **who**、**w** 或其他类似命令均通过检查该文件的内容输出登录用户。

所有成功的登录在 /var/log/wtmp 中记录。该文件可在无任何限制的情况下增长，因此，应当通过 **cron** 等命令周期性地清除该文件的内容。**last** 命令实际通过检查该文件的内容而输出最近的登录情况。

utmp 和 **wtmp** 文件均以二进制方式保存，因此，必须通过特定程序查看其中的内容。

17.4 xdm

在多数 Linux 发行版本中，均将某个 init 运行级别赋予特殊的功能：引导至 X Window。例如，Red Hat Linux 5.x 利用运行级别 5：

```
x11:5:respawn:/usr/bin/X11/xdm -nodaemon
```

从上面看出，进入或切换到运行级别 5 时，系统利用 **xdm** 启动 X Window。**xdm** 实际是 X Display Manager 的缩写，系统引导之后，将一直运行在 X Window 之中，除非切换到其他运行级别。

17.5 访问控制

按照惯例，用户帐号数据库包含在 /etc/passwd 文件中。某些系统使用“影象密码”，密码保存在 /etc/shadow 文件中。利用 NFS 或其他存储用户帐号数据库的方法，可以让多台计算机共享同一个用户帐号数据库。

用户数据库不仅包含用户名及其密码，同时也包含一些附加的信息，例如用户的真实名称、主目录以及登录 shell 等。因为这些附加信息是公共信息，因此，任何人都可以读取这些信息。由于这一原因，密码通常以加密形式保存。但是，任何一个可以读取该文件的用户，都可以不经实际登录而利用各种加密方法猜测密码。影象密码试图解决这一问题，

它将密码保存在单独的文件中，而只有 root 才能读取该密码文件。

不管是否利用影象密码，必须确保系统中所有的密码都是“好的”密码，也就是说，密码应该是不易猜中的。**crack** 程序可用来猜测密码，任何可被 **crack** 程序找出的密码都不是好密码。系统管理员可以利用 **crack** 程序避免用户采用坏的密码。利用 **passwd** 程序，也可以强制性地让用户采用好的密码。

用户组数据库包含在 /etc/group 文件中，对于利用影象密码的系统，可能会有一个 /etc/shadow.group 文件。

root 通常不能从大多数终端和网络登录，而只能从 /etc/securetty 文件中列出的终端登录。但是，其他用户在任意一个终端登录之后，利用 **su** 命令可成为 root。

17.6 shell 启动

当某个交互式的登录 shell 启动之后，它要自动执行一个或多个预先定义的文件。不同 shell 所运行的文件不同。但通常来说，大多数 shell 要运行一些全局的文件，例如，Bourne shell 及其变种会运行 /etc/profile，除此之外，它还要运行用户主目录中的 .profile 文件。

利用全局的脚本文件，系统管理员可以为所有用户定义公共的用户环境，尤其是可通过设置 **PATH** 包含一些局部的命令目录。另外，利用类似 .profile 的文件，可让用户定制自己的运行环境。

第十八章

安 全 性

在网络世界中，系统安全性日益受到重视。本章描述在 Linux 系统中如何管理用户帐号，如何保护重要数据不受侵害。

18.1 用户帐号及其配置

18.1.1 用户帐号

当许多人共用某台计算机，或在网络中，许多用户可以共享同一资源时，通常需要利用某种机制区别不同的用户。因此，在许多操作系统中，每个用户具有自己唯一的用户名，而用户名则用来登录系统。但是，系统中为某个用户所专有的信息却不只有用户名。帐号就是指专属于某个用户的全部文件、资源以及其他数据。

在 Linux 中，内核以纯粹的整数，即用户标识符或 **uid**，作为区分不同用户的“身份号”。内核之外，用户数据库定义用户标识符和文字用户名之间的映射关系。要建立用户，需要在用户数据库中添加用户信息，并为该用户建立主目录，有时还需要为用户建立初始的运行环境。

用户的建立可以以 root 身份手工进行，也可以利用 Linux 系统提供的各种工具完成。不同的 Linux 系统提供的工具名称（**adduser** 或 **useradd** 等）或功能不太一样，但它们都完成相同的工作。本章 18.1.4 小节将讲述用户帐号的手工建立过程。

18.1.2 用户组

利用用户组可在很大程度上简化管理工作。用户可通过 **groups** 程序查看自己所属的用户组，例如：

```
$ groups
users
```

用户组信息保存在 /etc/group 文件中，该文件的格式和 /etc/passwd 类似，如下所示：

```
group:passwd:GroupID:list of users
```

其中，第一个字段是用户组名称；第二个字段是组密码，一般的 Linux 系统忽略该

字段：第三个字段为组标识符，一般而言，root 组的标识符为 0，其他 0~100 的组标识符属于系统组，而用户定义的组应当取大于 100 的组标识符；第四个字段是属于该组的用户清单，各用户名之间用空格分隔。

系统管理员可以通过编辑该文件而添加用户组，也可以利用 **groupadd** 等系统程序添加用户组；利用 **groupmod** 命令，可修改用户组的组名称或编号；利用 **groupdel** 命令，可删除用户组。

18.1.3 /etc/passwd 以及其他信息文件

在上一章中提到，基本的用户数据库是文本文件 /etc/passwd（密码文件），其中列出了所有的用户及其相关信息。该文件中每一行定义一个用户，其中包含 7 个字段，字段之间由分号隔开。这 7 个字段的用途如下：

1. 用户名；
2. 密码，加密形式；
3. 数字的用户标识符；
4. 数字的组标识符；
5. 帐号的全名或其他说明信息；
6. 主目录；
7. 登录 shell，即登录时要运行的程序路径名称。

系统中的任何用户均可以读取该文件的内容，因此，所有人均可以读取任意一个用户的密码字段，即第二个字段。密码是加密保存的，但是，所有密码均是可以破译的，尤其是简单的密码，更可以不花大量时间就可以破译。所以，在密码文件中保存密码是不太明智的做法。

许多 Linux 系统利用影象密码以避免在密码文件中保存加密的密码，它们将密码保存在单独的 /etc/shadow 文件中，只有 root 才能读取该文件，而 /etc/passwd 文件只在第二个字段中包含特殊的标记。

大多数系统并不关心数字的用户标识符以及组标识符，但是如果要使用网络文件系统（NFS），则必须在所有的系统中，让同一用户帐号具备相同的用户标识符和组标识符，这是因为 NFS 也以数字的用户标识符标识用户。

当新用户的主目录建立之后，系统会以 /etc/skel 目录中的文件初始化主目录。/etc/skel 目录实际作为用户主目录结构的模板或“骨架”，系统管理员可以利用 /etc/skel 目录中的文件为新用户建立相当好的初始环境。

系统管理员也可以利用 /etc/profile 等文件为所有用户建立全局配置。

18.1.4 手工建立用户帐号

要手工建立用户帐号，可按如下步骤：

1. 利用 **vipw** 程序编辑 /etc/passwd 文件，为新的帐号添加新行。这里使用 **vipw** 进行编辑的原因是它可以在编辑文件时锁定被编辑的文件，而其他编辑器可让多个用户同时更新相同的文件。添加的新行应当保持正确的语法，应将密码字段设置为“*”，以避免用户登录。
2. 如果要添加新的用户组，则需要利用 **vigr** 程序编辑 /etc/group 文件。

3. 利用 **mkdir** 为用户建立主目录。
4. 从 /etc/skel 目录中复制文件到新用户的主目录。
5. 利用 **chown** 和 **chmod** 命令修改主目录的所有权和许可，可利用 **-R** 选项。
6. 利用 **passwd** 命令设置用户密码。

在最后一步设置用户密码之后，该帐号就可以被用户使用了。

18.1.5 修改用户属性

利用表 18-1 所示的命令，可修改用户帐号的不同属性。

表 18-1 修改用户属性的常用命令

chfn	修改帐号的全名字段。
chsh	修改帐号的登录 shell。
passwd	修改帐号密码。
chgrp	修改用户所属用户组，只有 root 可用该命令。

超级用户可利用表 18-1 中的命令修改任何帐号的属性，而通常的用户只能修改自己帐号的属性。

18.1.6 删除用户或暂时禁止用户

要删除用户，必须首先删除属于该用户的所有文件，以及邮箱、打印任务、**cron** 和 **at** 任务等等。然后，在 /etc/passwd 和 /etc/group 文件中删除相关行。另外，某些用户文件可包含在用户主目录之外，利用 **find** 命令可搜索这些文件：

```
find / -user username
```

某些 Linux 系统提供特殊工具用来删除用户帐号，通常名称为 **deluser** 或 **userdel**。

有时需要暂时禁止某帐号的使用，例如，使用该帐号的用户临时长期外出。禁止某用户帐号的最好办法是修改其登录 shell。可将登录 shell 修改为特殊程序，而该程序只输出一些说明性的信息，这样，用户可清楚了解发生了什么事，如果采用修改用户名或密码的方法，则会让用户不知所措。

18.2 文件的访问许可

UNIX 系统中，文件和目录的访问许可有三种，即读、写和执行，分别用 r、w 和 x 表示。文件和目录均有其所有者，所有者可能是文件或目录的初始创建者，也可能是经命令 **chown** 指定的用户；类似地，也有文件的属主组的概念，文件初始建立时，文件的属主组就是所有者所在的用户组，但也可以利用 chgrp 命令改变文件的属主组。文件的访问许可分别赋予文件所有者、属主组中的用户以及除上述用户之外的其他用户，分别简称用户、组和其他。通常，文件的访问许可由 8 进制的数字表示，表 18-2 给出了 8 进制数字代表的含义。

表 18-2 8 进制表示的文件访问许可

用户（u）			组（g）			其他（o）		
	r	w	x	r	w	x	r	w	x
rwx =	1	1	1 = 7	1	1	1 = 7	1	1	1 = 7
rw =	1	1	0 = 6	1	1	0 = 6	1	1	0 = 6
rx =	1	0	1 = 5	1	0	1 = 5	1	0	1 = 5
r =	1	0	0 = 4	1	0	0 = 4	1	0	0 = 4
wx =	0	1	1 = 3	0	1	1 = 3	0	1	1 = 3
w =	0	1	0 = 2	0	1	0 = 2	0	1	0 = 2
x =	0	0	1 = 1	0	0	1 = 1	0	0	1 = 1
无 =	0	0	0 = 0	0	0	0 = 0	0	0	0 = 0

从表中可以看出，文件对用户（所有者、属主组中的用户或其他用户）的许可由分别表示读、写和执行的 8 进制位表示，这些位是否有值，即表明了是否拥有对应的访问许可。例如，用户许可为 8 进制 7，表明所有者具有读、写和执行的权限。

如果要将某特定文件权限设置为用户可读取和执行（rx）、组用户可读取（r）而其他用户不可访问，则该文件的访问许可可表示为 540。三个数字分别表示用户、组和其他的访问许可。

利用 **chmod** 可修改文件的访问许可，例如：

```
# chmod 0700 test
```

则文件 test 的访问许可可描述为：所有者可读、写和执行，其他用户拒绝访问。

除此之外，文件访问属性中还有两个位，分别称为 SETUID 和 SETGID 位。任意用户在执行已设定 SETUID 和 SETGID 位的程序时，用户的有效 uid 和有效 gid 将改变为程序映象文件的所有者。我们在第十一章介绍进程时，提到过进程的有效 uid 和 gid 的概念。超级用户可通过系统调用 **setuid** 将某个进程的有效 uid 以及实际 uid 设置为其他用户，而普通用户只能通过运行已设定有 SETUID 和 SETGID 的程序而临时改变自己的访问许可。利用这种特色，可以让普通用户执行通常情况下只能由超级用户才能实现的功能程序。为了设定程序的 SETUID 和 SETGID 位，需要在上述 8 进制的 3 位数字之前添加相应的数字：4 表示设定 SETUID 位，而 2 表示设定 SETGID 位。例如：

```
# chmod 4755 test
```

设定文件 test 的 SETUID 位。当利用数字表示文件的访问许可时，如果用不足 4 位的 8 进制数字表示，则默认在前面添加 0。因此，

```
# chmod 55 test
```

的效果和

```
# chmod 0055 test
```

的效果一致。

也可以利用符号方式定义文件的访问许可，u 代表文件的所有者，g 代表用户组，o 代表其他；r 代表读取，w 代表写入，x 代表执行。例如：

```
# chmod u=rw test
```

将把所有者对文件 test 的访问许可设置为读取和写入。有关 **chmod** 命令的详细用法可参见有关手册页。

对文件来说，用户拥有执行许可，则假定该文件是可执行文件或脚本文件，系统以可执行文件或脚本文件处理该文件；对目录来说，如果拥有执行许可，则用户可利用 **cd** 目录改变到该目录，但如果没有读取许可，则不能用读取目录中的内容列表（即不能利用 **ls** 命令列目录内容）。

利用 **umask** 命令可设置用户的默认文件许可，用户创建的文件，将具备 **umask** 所设定的默认许可。需要注意的是，**umask** 使用的许可位定义了文件不具备的许可，另外，还应当在 8 进制的 **umask** 许可位中移去用户的执行位信息，**umask** 只在目录上设置读写许可，并不设置执行许可，因此，如果要将默认许可设置为 600，应利用如下的命令：

```
$ umask 077
```

再如，**umask 011** 将默认许可设置为 666，而 **umask 022** 将默认许可设置为 644。

利用 **chown** 可将文件的所有权赋予其他用户，例如：

```
# chown test user1
```

将文件 test 的所有者改变为 user1。

18.3 访问设备

因为通过系统中的设备特殊文件可直接访问硬件，因此，对这些文件的许可设置不当，就有可能造成严重的安全问题。尤其要注意的是系统中属于磁盘驱动器和内存的特殊设备文件，这些文件包括：

```
/dev/hda    /dev/hda1    /dev/hda2    ...
/dev/hdb    /dev/hdb1    /dev/hdb2    ...
...
/dev/sda    /dev/sda1    /dev/sda2    ...
...
/dev/mem    /dev/kmem
```

所有这些文件必须为 root 所拥有，而它们的访问许可应设置到最小的范围内。对磁盘设备文件的直接读写，可绕过操作系统的文件系统而直接访问磁盘上的字节。有意的或无意的修改可能造成系统的瘫痪，或重要数据的丢失。内存文件则直接映射到计算机的内存区，因此对内存设备文件的读取，也可能导致同样的问题。

鉴于上述原因，这些特殊设备文件必须由 root 用户拥有，其他用户不能具有读写许可。同时，应当避免以 root 登录完成一些非系统管理方面的任务，以免无意中直接访问这些文件而导致不良后果。

18.4 root 帐号

由于 root 帐号所拥有的可控制一切的特权，因此应当慎重使用 root 帐号。使用 root 帐号的规则是：

1. 除非绝对必要，不以 root 身份登录；
2. 在系统中尽量少设属于 root 组的帐号，而应当以特殊用户代替 root 完成特定的系统管理任务。

18.5 备份数据

虽然，硬件和软件的可靠性正在提高，但仍然有许多原因可能导致用户丢失数据，极端的例子有，恶意的人为原因或者由于不可抗拒的自然灾难等。因此，不管你的 Linux 系统作为个人用途还是商业用途，对关键文件的备份是不能忽视的。但是，象其他的数据一样，备份数据仍然有可能随时间的流失而受到物理的或其他原因的损害，因此，采用正确的备份策略尤为重要。

18.5.1 选择备份介质

在进行备份之前，选择适当的备份介质是首要任务。在选择备份介质时，要考虑到成本、可靠性、速度、可获得性以及可用性等。

备份的典型介质是磁盘和磁带。比较起来，磁盘具有非常便宜、相对可靠、速度不快、易获得等特点，但不适合于大量数据的备份；而磁带则具有比较便宜、相对可靠、比较快速、相对容易获得等特点，同时适合于进行大量数据的备份。

18.5.2 选择备份工具

传统的 UNIX 备份工具有 **tar**、**cpio**、**dump**（**dd**）等。有许多第三方的工具也可用来进行备份。备份方式及介质的选择通常影响备份工具的选择，表 18-3 给出了上述三个备份工具之间的比较（**tar** 和 **cpio** 工具非常相似）。

表 18-3 tar（cpio）和 dump（dd）之间的比较

备份工具	特点	优点	缺点
tar	通常用来进行文件的归档，可用于磁盘和磁带等任何介质。	可从归档文件中检索单个的文件。	效率较低，不支持直接的备份级。
dump	直接读取文件系统（原始方式），通常用于磁带备份。	直接的文件系统访问可不影响文件属性中的时间戳，也更加高效。 直接支持备份级。	备份程序专用于特定的文件系统类型。Linux 的 **dump** 命令只能识别 Ext2 文件系统。

18.5.3 简单备份

简单备份方案指第一次备份全部数据，而以后的备份只包含自上次备份后发生变化的数据。第一次的备份称为“完全备份”，其后的备份则称为“增量备份”。

如果有六盘磁带，则可利用这六盘磁带实现有效的简单备份方案：

每周末（周五）利用完全备份在 1 号磁带上备份完整的数据，2 号磁带到 5 号磁带则用于下周一到下周四的增量备份。到周五时，利用 6 号磁带进行完全备份，注意，这

时不应当用 1 号磁带进行完全备份。其后的增量备份可先用 5 号磁带，然后用 4 号，最后再用 2 号磁带，这样可以最大程度地保护数据。

完全备份可利用 **tar** 来实现：

```
# tar -create -file /dev/ftape /usr/src
tar: Removing leading / from absolute path names in the archive
#
```

Linux 使用的是 GNU 的 **tar** 版本，这一版本可使用长的选项名称，并且可以处理备份个能放在单张磁盘或磁带中的情况，还可以处理非常长的路径名。如果不能放在单张磁盘或磁带中，则必须使用多卷选项：-multi-volume (-M)。

备份完成之后，可利用比较选项来检验备份的正确性 (-compare (-d))。

利用 **tar** 的 -newer (-N) 选项可建立增量备份：

```
# tar -create -newer '3 Mar 1999' -file /dev/ftape /usr/src -verbose
tar: Removing leading / from absolute path names in the archive
usr/src/
usr/src/linux-2.0.36-includes/
usr/src/linux-2.0.36-includes/include/
usr/src/linux-2.0.36-includes/include/linux/
usr/src/linux-2.0.36-includes/include/linux/modules/
usr/src/linux-2.0.36-includes/include/asm-generic/
usr/src/linux-2.0.36-includes/include/asm-i386/
usr/src/linux-2.0.36-includes/include/asm-mips/
usr/src/linux-2.0.36-includes/include/asm-alpha/
usr/src/linux-2.0.36-includes/include/asm-m68k/
usr/src/linux-2.0.36-includes/include/asm-sparc/
#
```

应当注意，**tar** 不能自动察觉文件索引节点信息的变化，如果利用 **find** 命令，可以将当前文件系统的状态和新近备份的文件清单相比较，从而找出文件系统的变化。具体的脚本和程序可以在 Linux 的 ftp 站点中找到。

利用 **tar** 的 -extract (-x) 选项可提取备份中的文件：

```
# tar -extract -same-permissions -verbose -file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-2.0.36-includes/
usr/src/linux-2.0.36-includes/include/
usr/src/linux-2.0.36-includes/include/linux/
usr/src/linux-2.0.36-includes/include/linux/hdreg.h
usr/src/linux-2.0.36-includes/include/linux/kernel.h
...
#
```

通过指定文件名称，也可以提取特定的文件：

```
# tar xpvf /dev/fd0H1440 usr/src/linux-2.0.36-includes/include/linux/hdreg.h
usr/src/linux-2.0.36-includes/include/linux/hdreg.h
#
```

如果要查看某备份卷中的文件，可利用 -list (-t) 选项。

tar 不能正确处理删除的文件。如果在完全备份和增量备份之间删除了某个文件，则恢复备份之后，该文件会再次出现。

18.5.4 多级备份

上述的简单备份方案更加适合于个人用户或小型站点，而对于大型的商业站点来说，多级备份则更为适合。

上面的简单备份例子中，增量备份以日为周期进行，如果我们以不同的周期进行多个系列的简单备份，就形成了多级备份。

如图 18-1 所示，0 级备份使用两个磁带，1 号磁带和 2 号磁带，并以月为周期，每月的第一个周一进行完全备份，轮流使用两个磁带；1 级备份使用四个磁带，以周为周期形成每月的 1 级简单备份系列；而 7 号磁带到 10 号磁带以日为周期形成每周的 2 级简单备份系列。

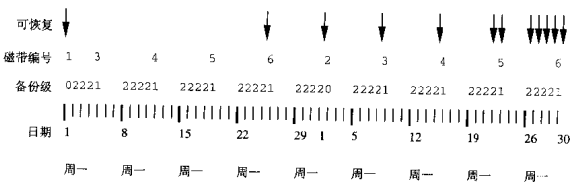


图 18-1 多级备份示意图

利用如图 18-1 所示的多级备份方案，我们利用 10 盘磁带就可以将备份的可恢复时间延续到 2 个月，而在上小节中的简单备份方案中，使用 6 盘磁带却只能达到 2 周的可恢复备份。因此，多级备份是有效延长备份的可恢复时间的廉价方案。

利用表 18-4 所示的备份方案，还可以有效降低进行完整恢复时需要使用的增量备份磁带的数目。该方案来自 dump 的手册页提供的建议，这个方案可将备份和恢复的时间同时保持在较低的水平，在一次恢复中使用的磁带数量，依赖于两次完全备份之间的时间长短。

表 18-4 有效的多备份级方案

磁带	备份级	备份日期（假定从月初开始）	恢复用磁带
1	0	1	1
2	3	2	1, 2
3	2	4	1, 3
4	5	5	1, 2, 4
5	4	7	1, 2, 5

6	7	8	1, 2, 5, 6
7	6	10	1, 2, 5, 7
8	9	11	1, 2, 5, 7, 8
9	8	13	1, 2, 5, 7, 9
10	9	14	1, 2, 5, 7, 9, 10
11	9	15	1, 2, 5, 7, 9, 10, 11
...	9	16	1, 2, 5, 7, 9, 10, 11, ...

dump 具有内建的多级备份支持，而 **tar** 和 **cpio** 需要在第三方软件的帮助下才能支持多级备份。有关 **dump** 的使用可参见相关的联机文档或手册页。

18.5.5 压缩备份

为了节省备份数据占用的空间，人们通常会想到进行压缩备份，也就是说，在备份之前，首先进行压缩。GNU 的 **tar** 工具可利用 `-gzip (-z)` 选项对整个备份进行压缩（利用 **tar** 和 **gzip** 压缩程序之间的管道通讯）。但是，压缩备份有时会造成不可挽回的数据丢失，因为压缩文件中的每一位信息都有举足轻重的作用，如果某一位数据发生错误，则会影响到其后所有的数据。为了避免这样的问题，可单独对每个文件进行压缩，而不是对整个备份数据进行压缩，这样，就可以避免发生大面积的数据丢失。**afio** 命令（**cpio** 命令的变种）可对文件进行单独压缩后备份。

最后，再说明一下哪些数据需要备份，而哪些数据不需要进行备份。明显需要进行备份的数据：

- 用户数据 （/home）；
- 系统配置文件 （/etc）；
- 软件配置文件。

而如下内容通常不必进行备份：

- 非常容易安装的软件；
- /proc 文件系统。

第三部分

Linux 实战举例

读完第二部分相信你对 Linux 有了比较深刻的理解。在这一部分里将介绍四个专题：内核的编译、网络应用、中文环境和基于 Xlib 的应用程序开发基础。这四个专题独立成章，你可以按顺序阅读，也可以直接阅读感兴趣的章节。

Linux 的内核版本不断推陈出新，速度之快足以使微软跌掉下巴。而每个新版本都有令人惊喜的进展，或是加进了新的功能，或是修正原有的错误。你正在使用 Linux 内核是否支持你刚买来的显示卡？不支持？！不用灰心，Download 一个新版内核试试吧。本书的第十九章为你介绍如何编译一个新的内核。

如今，上网是一种时尚，网上丰富的资源让人欲罢不能。Linux 与网络的关系简直就是密不可分。也许你还不知道，许多网络服务器使用的就是 Linux 操作系统。你有兴趣知道它们是怎样工作的吗？本书第二十章将为你讲解如何设置 Web 服务器和 Proxy 服务器。你甚至可以亲手试一试，将你的电脑设为网络服务器。如果你是拨号上网的用户，你也可以在这一章中找到感兴趣的内容。读完第二十章你将会对网络有一个全新的认识。

Linux 是 free 的资源，而我国诸多电脑爱好者对其涉足甚少。究其原因恐怕是 Linux 对中文的支持不象 Windows 95/98 那么友好的缘故。第二十一章将为你全面系统地介绍 Linux 的中文环境。读完该章后你将可以在 Linux 上毫无困难地使用中文。

如果你从未在 Linux/UNIX 上编过程序，你不妨看一看第二十二章。该章给出了一个简单的 Xlib 应用程序，讲解了基于 Xlib 的应用程序开发原理。你可以将例子程序输入计算机，调试通过后，你将看到一个 X 界面。尽管它不能完成任何任务，相信你在看到它出现在你的计算机屏幕上时还是会有一番惊喜——你已经成为有能力开发应用程序的高手了！

第十九章

内核编译

如第十五章所述，Linux 内核（Kernel）由内存管理、进程管理、设备驱动程序、文件系统和网络管理等几部分组成。它是程序与硬件的仲裁者。它为所有执行中的程序提供内存管理，并确保它们都能够均衡（或不均衡，如果你愿意）地分享处理机的运算资源。此外，它还提供了一个良好的界面让刚刚提到的程序能够透过它与硬件沟通。当然，这只是一些基本功能，内核所处理的工作比这还要更复杂一些。一般地说，新版本的内核能够与更多的硬件沟通（也就是说，拥有更多的设备驱动程序），有更好的进程管理，比旧版本效率更高、更稳定，并且修正了旧版中的一些错误。因而在下述情况下你需要编译新内核：

- 想删除内核中实际系统根本用不上的设备驱动程序：内核是常驻内存的，删除无用的设备驱动程序将减少内核本身所占用的内存数量，从而在内存资源贫乏时为其他程序提供更多的内存空间；
- 现有内核不支持或没有将实际系统上的某些硬件驱动程序编译进去；
- 某些应用程序需要较新的内核版本支持才能运行；
- 使系统更富有效率；
- 修正旧版中的错误；
- 使你的 Linux 系统永远是最先进的。

19.1 准备工作

19.1.1 了解你现有的内核版本号

内核版本号遵循以下约定：

```
major.minor.patchlevel
```

其中，major 是主版本号，minor 是次版本号，major.minor 就是当前内核的版本号。而 patchlevel 是对当前内核的修正次数。如 2.0.34 表示对内核 2.0 版的第 34 次修正版。偶数的内核版本（如 1.2，2.0 等）是“稳定”的发布版，对它的修正通常是去除程序中的小毛病（即所谓的“臭虫”），而没有新的特性加入。奇数内核版本（如 1.3，2.1 等）是“开发”版，对它的修正包括开发者想加进的新代码以及对原代码“臭虫”的修正。

我们可以用 **uname -a** 命令来查看系统当前的内核版本：

```
$ uname -a
Linux lark 2.0.34 #2 Sun Oct 12 10:26:31 CST 1997 i586 unknown
```

它表示系统的内核版本是 2.0.34，第二次（# 2）编译于 1997 年 10 月 12 日 10:26:31，星期日，节点是 lark，CPU 类型是 Pentium。

19.1.2 了解新内核的基本情况

1. 新内核支持哪些硬件

在 Hardware-HOWTO 文件中列出了很多 Linux 支持的硬件。新内核自然也支持这些硬件，但每个版本的更新总要加入一些新的内容，故而 Hardware-HOWTO 文件中没有列出的硬件，新内核也可能支持。大家可以查看 Linux 源程序中的 config.in 文件，或者在配置内核的过程中查看。在配置内核时将列出标准的内核源程序支持的所有硬件，但这并不是 Linux 所支持的全部硬件；许多普通的设备驱动程序（PCMCIA 驱动程序以及某些磁带机的驱动程序等）是独立维护、发行的可加载模块。可加载模块是内核的一部分（通常是设备驱动程序），但是并没有被编译到内核中。它们被分别编译，绝大多数情况下可以随时将它们插入运行中的内核或从中卸载。这是一种非常好的方式，我们在自己的应用中也应该学习这种思想。许多常用的设备驱动程序，例如 PCMCIA 驱动程序以及 QIC-80/40 磁带机的驱动程序等就是可加载模块。

2. 新内核对其他软件的版本要求

在新内核源代码的 Documentation 目录中有一名为 Changes 的文件。该文件列出了正确编译内核和支持内核正确运行所需要的各种软件的版本说明。例如，2.2.3 版本的内核对其他软件版本的最低要求如下所示（第三列给出了显示当前系统各种软件版本的命令）。当然，要升级内核的用户不一定需要 Changes 文件中所列的全部软件，这取决于用户将要编译的内容。

```
Last updated: January 18, 1999
Current Author:
Chris Ricker (kaboom@gatech.edu or chris.ricker@m.cc.utah.edu).
Current Minimal Requirements
*****************************
Upgrade to at *least* these software revisions before thinking
you've encountered a bug!  If you're unsure what version you're
currently running, the suggested command should tell you.

- Kernel modules            2.1.121         ; insmod -V
- Gnu C                     2.7.2.3         ; gcc -version
- Binutils                  2.8.1.0.23      ; ld -v
- Linux libc5 C Library     5.4.46          ; ls -l /lib/libc.so.*
- Linux libc6 C Library     2.0.7pre6       ; ls -l /lib/libc.so.*
- Dynamic Linker (ld.so)    1.9.9           ; ldd --version or ldd -v
- Linux C++ Library         2.7.2.8         ; ls -l /usr/lib/libg++.so.*
- Procps                    1.2.9           ; ps -version
- Procinfo                  15              ; procinfo -v
- Psmisc                    17              ; pstree -V
- Net-tools                 1.49            ; hostname -V
- Loadlin                   1.6a
- Sh-utils                  1.16            ; basename -v
```

```
- Autofs          3.1.1          ; automount -version
- NFS             2.2beta40      ; showmount -version
- Bash            1.14.7         ; bash -version
- Ncpfs           2.2.0          ; ncpmount -v
- Pcmcia-cs       3.0.7          ; cardmgr -V
- PPP             2.3.5          ; pppd -v
- Util-linux      2.9q           ; chsh -v
```

3. 需要多大的硬盘空间

这与你的系统的特殊配置有关。压缩过的 Linux 2.2.3 版源代码约占 13 MB，解压缩后约有 51 MB，此外还需要更多的硬盘空间来实际编译这些东西。当然这与你将要在内核中配置多少东西有关。例如，在某部机器上，配有网络支持，3Com 3C503 的驱动程序，并且配置了三种文件系统，这就需要 30 MB。加上压缩过的 Linux 源代码，这个配置大概需要用掉 43 MB。若不需要网络设备（但仍需要网络支持），加上声卡驱动，结果会用掉更多的空间。另外，新版本的内核几乎总是比旧版本的占更多的空间。所以，在编译新内核之前要确保有足够的硬盘空间。

4. 需要多长的时间

这取决于 CPU 速度、RAM 的大小以及内核配置的多少等。对大多数心急的人而言，答案是“很久”。在一台 16 MB RAM 的 486DX4-100 电脑上，一个支持五种文件系统、支持网络、以及有声卡驱动的 1.2 版内核可以在二十分钟以内完成。类似的配置，在一台 386DX/40（8 MB RAM）上大约需要用 1.5 个小时。而在目前较为常用的 Pentium 或 Pentium II 个人电脑上则要快得多。

19.1.3 获取源文件

你可以从匿名 ftp 站点，例如 `ftp.funet.fi` 的 `/pub/Linux/PEOPLE/Linus` 目录下获得，或是其他镜像站点。其文件名一般标记为 `linux-x.x.xx.tar.gz`（或是 `linux-x.x.xx.tar`），其中的 `x.x.xx` 是版本编号。而文件名标记为 `patch-x.x.xx.tar.gz`（或是 `patch- x.x.xx.tar`）的文件是“补丁”文件。

强烈建议你去访问镜像站点，而不要直接到 `ftp.funet.fi` 去！下面是一些镜像站点或存有 Linux 内核的其他站点及其目录：

```
美国:     sunsite.unc.edu:/pub/Linux/kernel
美国:     tsx-11.mit.edu:/pub/linux/sources/system
英国:     sunsite.doc.ic.ac.uk:/pub/UNIX/Linux/sunsite.unc-MIRROR/kernel
奥地利:   ftp.univie.ac.at:/systems/linux/sunsite/kernel
德国:     ftp.Germany.EU.net:/pub/os/Linux/Local.EUnet/Kernel/Linus
德国:     sunsite.informatik.rwth-aachen.de:/pub/Linux/PEOPLE/Linus
法国:     ftp.ibp.fr:/pub/linux/sources/system/patches
澳大利亚: sunsite.anu.edu.au:/pub/linux/kernel
日本:     sunsite.sut.ac.jp:/pub/archives/lunux/kernel
```

一般来说，最好选择 `sunsite.unc.edu` 的镜像站点。`sunsite.unc.edu/pub/Linux/MIRRORS` 文件中列出了已知的 `sunsite.unc.edu` 镜像站点。如果你没有办法上 ftp，在新闻组 `comp.os.linux.announce` 上会定期刊登一个存放 Linux 的 BBS

系统的列表，试着到那儿去找一下吧。也可以去看一下 http://www.linux.org/。

19.1.4 解开源程序包

以 root 登录，然后 **cd** 到 /usr/src。如果你在安装 Linux 时已经安装了内核源代码，在这个目录下应该已经存在一个名为 linux/ 的子目录，其中存放的是系统当前版本的内核源程序。若没有安装内核的源代码，则在这个目录中至少应该有 Linux 内核的头文件，存放在 linux/include 子目录下，在编译大多数 Linux 下的系统软件时需要这些文件。应该保留 linux/include 子目录下的所有文件，以便在内核编译不成功或是运行不稳定时可以恢复以前的内核配置。但即将编译的新内核源程序也要存放在 /usr/src/linux 下，故而在解开新内核源程序包之前，应把 linux/ 目录改成其他名字，最好是根据当前使用的内核版本来修改其名称。用“**uname -r**”命令可以显示当前的内核版本号。所以，如果“**uname -r**”显示“2.0.24”，就可以用“**mv**”命令把 linux 目录改名为 linux-2.0.34。如果用户保存有当前内核的源程序，为了节省硬盘空间可以直接把除了 linux/include 之外的其他子目录都删掉。总之要确定在解开新内核源代码之前，/usr/src 目录下没有“linux”这个子目录。

下面就让我们来解开新内核的源程序包。在 /usr/src 目录下，用

```
# tar  zxvf linux-x.x.xx.tar.gz
```

来解开内核源程序包。如果你拿到的是 .tar 文件而不是 .tar.gz ，那么就用

```
# tar  xvf linux-x.y.z.tar
```

屏幕上将闪过所有源程序的名字及其所在目录。之后，在 /usr/src 将会出现一个新的“linux”子目录，其中包含 x.x.xx 版本的 Linux 内核的全部源程序（patch-x.x.xx.tar.gz 或是 patch- x.x.xx.tar 文件的用法请参考有关文档）。**cd** 到 linux/ 目录下然后查看 README 文件，里面应该有一段标题为“INSTALLING the kernel”或类似的文字，请仔细阅读该文档。

这时要检查一下 19.1.2 中提到的对其他软件版本要求的问题。

然后，检查一下 /usr/include/asm，/usr/include/linux，/usr/include/scsi 是不是正确地符号连接到 /usr/src/linux/include 下的对应目录。如果不是，可以这样做：

```
$ cd  /usr/include
# rm  -fr asm linux scsi
# ln  -s  /usr/src/linux/include/asm-i386  asm
# ln  -s  /usr/src/linux/include/linux  linux
# ln  -s  /usr/src/linux/include/scsi  scsi
```

上面 asm-i386 目录对于其他体系结构，如 sparc 或 alpha，则应换成 asm-sparc 或 asm-alpha。

最后，为确保源程序目录树中没有残留的 .o 文件和其他从属文件，使用如下命令做彻底清理：

```
$ cd  /usr/src/linux
# make  mrproper
```

下面我们就可以进行内核编译了。

19.2 内核编译

19.2.1 内核配置

以root 身份在 /usr/src/linux 下执行 **make config** 命令来配置内核。所谓内核配置就是选择内核支持哪些特性和编译哪些设备驱动。**make config** 命令需要 **bash** 才能工作，所以要确定 Shell 是 /bin/bash。用 **make config** 命令等效的命令有 X Window 形式的 **make xconfig**，和文字选单模式的 **make menuconfig**。Linux-2.2.3 内核的 X Window 配置界面如图 19-1 所示。

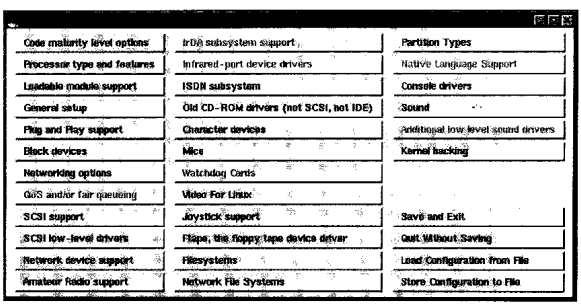


图 19-1 Linux-2.2.3 内核的 X Window 配置界面

由图 19-1 可以看出，Linux-2.2.3 内核代码成熟的配置见表 19-1。

表 19-1 Linux 2.2.3 内核的编译配置

Code maturity level options	代码成熟度选项
Processor type and features	处理器类型和特色
Loadable module support	可加载模块支持
General setup	一般设置
Plug and Play support	即插即用设备支持
Block devices	块设备
Networking options	网络选项
SCSI support	SCSI 支持

SCSI low-level drivers	SCSI 低级驱动
Networking device support	网络设备支持
Amateur Radio support	业余收音机支持
ISDN subsystem	ISDN 子系统
Old CD-ROM drivers (not SCSI, not IDE)	老式光驱驱动（非 SCSI、IDE）
Character devices	字符设备
Mice	鼠标
Video For Linux	Linux 的视频
Joystick support	操纵杆支持
Ftape, the floppy tape device driver	Ftape 设备驱动
Filesystems	文件系统
Network File Systems	网络文件系统
Partion Types	分区形式
Console drivers	控制台驱动
Sound	声音
Kernel hacking	内核监视

Networking options 窗口如图 19-2 所示。

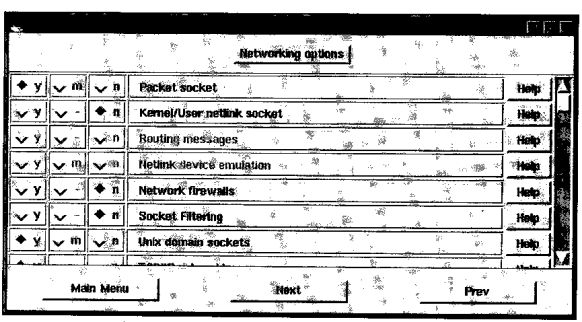


图 19-2 网络选项窗口

选取各个菜单项，回答“y”、“n”或“m”及相应问题即可完成内核配置工作。“y”表示对相应特性的支持或相应设备驱动程序将被编译进内核，“n”则相反，而“m”则表示把对相应特性的支持或相应设备驱动编译成可加载模块。其间可击点“Help”寻求帮助。

下面介绍一下几个重要选项的配置：

1. Code maturity level options（代码成熟程度选项）

- Prompt for development and/or incomplete code/drivers [N/y/?] 内核中有些代码或驱动还处在测试阶段，运行不稳定。如果有兴趣参与测试或确实需要其中的某些特性，选择“y”；如果希望得到一个稳定的内核请选择“n”。

2. Processor type and features（处理器类型和特色）

- Processor family (386, 486/Cx486, 586/K5/5x86/6x86, Pentium/K6/TSC, PPro/6x86MX) [PPro/6x86MX] Linux-2.2.3 内核允许选择处理器类型，可供选择的处理器有：386, 486/Cx486，586/K5/5x86/6x86，Pentium/K6/TSC，PPro/6x86MX，缺省为 PPro/6x86MX。
- Math emulation (CONFIG_MATH_EMULATION) [N/y/?] 如果没有协处理器（一台 386 或 486SX），这里要回答“y”。如果有协处理器却还回答“y”，那也不必太担心——Linux 还是会去使用实际存在的协处理器而忽略掉内核的模拟程序。唯一的影响是编译出来的内核变大（用掉更多的内存）。目前常用的处理器都有协处理器，因而本选项选“n”。
- MTRR (Memory Type Range Register) support (CONFIG_MTRR) [N/y/?] 在 Intel Pentium、Pro/Pentium II 系统中 MTRR 可用于控制处理器存取内存。当有 PCI 或 AGP 总线的 VGA 卡时该选项特别有用，可使图象写入速度提高 2.5 倍以上。选择该选项将产生 /proc/mtrr 文件用于管理 MTRR，供 X 服务器使用。详细情况请参考该选项的联机帮助文件。
- Symmetric multi-processing support (CONFIG_SMP) [Y/n/?] 该选项用于对称的多处理器系统，即计算机上有多个 CPU。如果你的计算机只有一个 CPU 或者你不太清楚，请选则“n”，因为该模式既可以在单 CPU 的机器也可以在多 CPU 机器上运行，当然对于多 CPU 机器系统只使用其中的一个 CPU。如果选择“y”，在大多数的单 CPU 机器上也能运行，但速度要慢一些。对于多 CPU 的计算机应该选“y”，在下面的“Enhanced Real Time Clock Support”选项中也要选“y”，此时“Advanced Power Management”将被关闭。

3. Loadable module support（可加载模块支持）

- Enable loadable module support (CONFIG_MODULES) [Y/n/?] 目前越来越多的驱动使用可加载模块，故而此处应选“y”。如果选择“n”，下面所有的特性或驱动将不能被编译成可加载模块。
- Set version information on all symbols for modules (CONFIG_MODVERSIONS) [N/y/?] 通常当使用新内核时模块需要重新编译。而在此处选择“y”，在编译新内核时则可以安全地使用同一模块。这一点对于非内核源程序产生的模块而言非常有用。但要注意，该选项需要 **modprobe** 程序。模块支持所需要的所有软件均在“modutils”包中（该软件包的安装位置和版本信息请参考 Documentation/Changes 文件）。如果该软件包中没

有 **genksyms** 程序，而你又选择了“y”，内核编译将失败。如果没有非内核源程序产生的模块，最好还是选择“n”。

- Kernel module loader (CONFIG_KMOD) [N/y/?] 一般地说，将某些驱动和/或文件系统设为可加载模块时，还要加载相应的模块（使用程序 **insmod** 或 **modprobe**）。此处若选择“y”，内核将自动加载这些模块（参考 Documentation/kmod.txt）。

4. General setup（一般设置）

- Networking support (CONFIG_NET) [Y/n/?] 若不打算上网请选择“n”。否则选“y”，并在“Networking options（网络选项）”和“Networking device support（网络设备支持）”选项中选择网络协议定义网络设备。
- PCI support (CONFIG_PCI) [Y/n/?] 目前大多数个人电脑使用 PCI，故此处一般选“y”。
- PCI access mode (BIOS, Direct, Any) [Any] 在 PCI 系统中，BIOS 可被用于探测 PCI 设备并决定其配置。但是一些老式 PCI 主板上的 BIOS 中有“臭虫”，若使用上述功能可能导致错误。还有一些内嵌的基于 PCI 的系统根本就没有 BIOS。Linux 可以不使用 BIOS 直接探测 PCI 设备。这个选项允许你选择 Linux 该如何探测 PCI 设备。若选择“BIOS”，Linux 将使用 BIOS；若选择“Direct”，则不使用“BIOS”；选择“Any”时，内核将首先不使用 BIOS 直接探测 PCI 设备，若失败再使用 BIOS。最好选择缺省值“Any”。
- PCI quirks (CONFIG_PCI_QUIRKS) [Y/n/?] 如果 BIOS 有问题，可能导致 PCI 总线设立错误或不是最优。选择“y”可以纠正该错误。如果确信 BIOS 没有问题，请选择“n”以减少内核的大小。
- Backward-compatible /proc/pci (CONFIG_PCI_OLD_PROC) [Y/n/?] 以前的内核支持 /proc/pci 文件，而新版内核使用新的 /proc 接口（/proc/bus/pci）。为保持兼容性该选项一般选“y”。
- MCA support (CONFIG_MCA) [N/y/?] MCA 是一种类似于 PCI 或 ISA 的总线，详细内容请参考 Documentation/mca.txt 文件。
- SGI Visual Workstation support (CONFIG_VISWS) [N/y/?] 该选项只在特定的机器上可以运行（参考 Documentation/sgi-visws.txt），所以一般选缺省值“n”。
- System V IPC (CONFIG_SYSVIPC) [Y/n/?] 进程间通讯是一类库函数和系统调用，它使得进程同步并可交换信息。若选择“n”，许多程序将无法运行。
- BSD Process Accounting (CONFIG_BSD_PROCESS_ACCT) [N/y/?] 如果选择“y”，通过特殊的系统调用，用户级别的程序可以指示内核将进程的信息写入文件。这些信息包括创建时间、所有者、命令名、内存消耗等。若不需要此功能请选“n”。
- Sysctl support (CONFIG_SYSCTL) [Y/n/?] sysctl 接口提供了一种不重新编译内核或不重新启动系统即可修改特定的内核参数和变量的途径。

这当然是一个不错的选择，应回答“y”。详细内容请参考Documentation/sysctl/下的文件。注意，开启该选项将使内核最少增大8 KB。

- Kernel support for a.out binaries (CONFIG_BINFMT_AOUT) [Y/m/n/?] Kernel support for ELF binaries (CONFIG_BINFMT_ELF) [Y/m/n/?] a.out（Assembler.OUTput）是早期的UNIX运行库和执行代码使用的一种格式，后来逐渐被ELF（Executable and Linkable Format）所代替，但有些程序仍使用a.out格式，所以应该选“y”。现在很多执行代码只用ELF格式发布，该选项当然要选择“y”。
- Kernel support for MISC binaries (CONFIG_BINFMT_MISC) [Y/m/n/?] 在Linux上，有些代码需要专用的解释器才能执行，如JAVA、Pythonhe和Emacs-Lips等。如果内核加入此项，在运行它们时只需要键入其名字即可，Linux的内核会负责寻找相应的解释器来执行它们。该选项一般选“y”。
- Parallel port support (CONFIG_PARPORT) [N/y/m/?] 如果要使用并口设备（如打印机等）请选“y”。
- Advanced Power Management BIOS support (CONFIG_APM) [N/y/?] 该选项主要针对电池供电的笔记本型等计算机，并要配有支持APM的BIOS。对于大多数台式机而言选“n”即可。

5. Plug and Play support（即插即用设备支持）

- Plug and Play support (CONFIG_PNP) [N/y/?] 允许内核自动配置外设，选“y”。

6. Block devices（块设备）

- Normal PC floppy disk support (CONFIG_BLK_DEV_FD) [Y/m/n/?]
  Enhanced IDE/MFM/RLL disk/cdrom/tape/floppy support (CONFIG_BLK_DEV_IDE) [Y/m/n/?]
  Include IDE/ATAPI CDROM support (CONFIG_BLK_DEV_IDECD) [Y/m/n/?]
  上述三个选项是对常用的软驱、IDE硬盘和CD-ROM的支持，一般均应选“y”。

7. Networking options（网络选项）

- Packet socket (CONFIG_PACKET) [Y/m/n/?] 一些应用程序使用Packet协议直接同网络设备通信而不通过内核中的其他中介协议，如tcpdump。若想让其工作，选“y”，也可以选“m”（参考Documentation/modules.txt）。如果不能确定，选“y”。
- Kernel/User netlink socket (CONFIG_NETLINK) [N/y/?] 该驱动允许内核的特定部分或模块同用户过程进行双向通信。如果你不太清楚该选项的含义，选“y”。

- Network firewalls (CONFIG_FIREWALL) [N/y/?] 若要支持防火墙，选“y”。
- UNIX domain sockets (CONFIG_UNIX) [Y/m/n/?] 许多常用的程序（如，X Window、syslog 等）使用套接字，即使是未连接任何网络。此处应该选“y”，除非你使用的是内嵌式系统或类似的系统。
- TCP/IP networking (CONFIG_INET) [Y/n/?] TCP/IP 是 Internet 核大多数局域以太网使用的协议。某些程序如 X Window 系统需要 TCP/IP 支持。即使系统没有与其他计算机相连，也可以用回环形式在计算机内部实现网络功能，如 **ping**、**telenet** 等。所以要选“y”。
- The IPX protocol (CONFIG_IPX) [N/y/m/?] 若希望支持 IPX 协议选“y”。
- Appletalk DDP (CONFIG_ATALK) [N/y/m/?] 若希望支持 Appletalk DDP 选“y”。

8. SCSI support （SCSI 支持）和 SCSI low-level drivers（SCSI 低级驱动）

若有 SCSI 设备请选择相应选项。

9. Networking device support（网络设备支持）

选择相应的网卡。若使用拨号上网请根据 ISP 提供的服务选择 PPP 或 SLIP，通常是 PPP。

10. ISDN subsystem（ISDN 子系统）

尚未见到我国支持 ISDN（Integrated Services Digital Networks）的报道，故该选项选“N”。

11. Character devices（字符设备）

- Virtual terminal (CONFIG_VT) [Y/n/?] 所谓的虚拟终端是指在一个物理终端上可以运行多个虚拟终端，也称为虚拟控制台。这一点非常有用，比如一个终端可以用于收集系统信息，另一个终端可以用于用户文本编辑，而第三个终端又可以启动 X，这三个作业之间是并行的，可通过热键切换。该选项一般要选“y”。
- Support for console on virtual terminal (CONFIG_VT_CONSOLE) [Y/n/?] 系统控制台（system console）用于接收内核的全部信息，并可在单用户模式下登录。如果此选项选“y”，可将一个虚拟终端用作系统控制台。一般要选“y”。
- Standard/generic (dumb) serial support (CONFIG_SERIAL) [Y/m/n/?] 对于大多数系统，串行口上有鼠标、调制解调器等设备，应选“y”。
- Support for console on serial port (CONFIG_SERIAL_CONSOLE) [N/y/?] 若选“y”，可将一串行口用作系统控制台。一般选“n”。
- Extended dumb serial driver options (CONFIG_SERIAL_EXTENDED)

[N/y/?] 如果希望使用“dumb”的非标准特性（如 HUB6 支持等），选“y”。一般选“n”。

- Non-standard serial port support (CONFIG_SERIAL_NONSTANDARD) [N/y/?] 若有非标准串口（如 Cyclades、Digiboards 等），选“y”。一般选“n”。
- UNIX98 PTY support (CONFIG_UNIX98_PTYS) [Y/n/?] 所谓伪终端（pseudo terminal，PTY）是由主端和从端两部分组成的一种软件设备。从端设备与物理终端相同，而主端设备则由过程使用从从端读入数据或向从端写入数据，从而模拟一个终端。通常主端是 telnet 服务器和 xterms。GNU C 函数库 glibc 2.1 及以后的版本支持 UNIX98 PTY。如果此处选择“y”，则要保证安装 glibc 2.1 或更新的版本，同时“/dev/pts filesystem for UNIX98 PTYs”选项也要选“y”。此处选“n”更安全一些。
- Maximum number of UNIX98 PTYs in use (0-2048) (CONFIG_UNIX98_PTY_COUNT) [256] 对台式机而言缺省 256 完全可以满足要求。
- Mouse Support (not serial mice) (CONFIG_MOUSE) [Y/n/?] 若计算机上连有 PS/2 等非串口鼠标，选“y”。若鼠标是插在串行口上的那种，选“n”。

12. Mice（鼠标）

选择相应的鼠标驱动和特色支持：ATIXL 鼠标、Logitech 鼠标、Microsoft 鼠标、PS/2 鼠标、C&T 82C710 鼠标、PC110 数字板、QIC-02 磁带机、Watchdog Timer、“/dev/nvram support”和“Enhanced Real Time Clock Support”。

13. Video For Linux（Linux 的视频）

- Video For Linux (CONFIG_VIDEO_DEV) [N/y/m/?] 支持音频/视频的捕捉合成设备和 FM 卡等。

14. Joystick support（游戏操纵杆支持）

- Joystick support (CONFIG_JOYSTICK) [N/y/m/?] 如果你有游戏操纵杆，此项回答“y”，并选择相应的驱动。详细情况请参考 Documentation/joystick.txt。

15. Ftape, the floppy tape device driver（Ftape 设备驱动）

- Ftape (QIC-80/Travan) support (CONFIG_FTAPE) [N/y/m/?] 如果在软盘控制器上连有磁带机，该选项选“y”。一些磁带机，如 Seagate 的“Tape Store 3200”、Iomega 的“Ditto 3200”、Exabyte 的“Eagle TR-3”等，本身带有高速控制器，此处选择“y”也能支持。如果控制器比较特殊，如 CMS FC-10、FC-20，Mountain Mach-II，或其他基于 Intel 82078 的 FDC，则需要在下面的“Floppy tape controllers”子选项中选择合适的选项，并修改 IRQ、DMA 通道和 IO 基址。一般个人电脑不配磁带机，应选“n”。

16. Filesystems（文件系统）

你可以在此选择内核支持何种文件系统。下面列出的是几个常用的选项。

- Quota support (CONFIG_QUOTA) [N/y/?] 系统管理员如果想给用户设置磁盘定额，选择“y”。
- Kernel automounter support (CONFIG_AUTOFS_FS) [Y/m/n/?] **automounter** 是一个能够按照需要自动安装远程文件系统的工具。如果你不是处于一个大型分布式网络中，那么不需要使用 **automounter**，选“n”。若选“y”或“m”请参考联机帮助。
- DOS FAT fs support (CONFIG_FAT_FS) [N/y/m/?] 如果你准备在 Linux 中使用基于 FAT 的文件系统，如 MS-DOS，VFAT（Windows95）和 UMSDOS 文件系统，该选项应选“y”。它本身不是一种文件系统，但为内核提供 FAT 支持。
- ISO 9660 CDROM filesystem support (CONFIG_ISO9660_FS) [Y/m/n/?] ISO 9660 CDROM 文件系统是 CDROM 使用的标准文件系统。有光驱的系统均应选“y”。
- Minix fs support (CONFIG_MINIX_FS) [N/y/m/?] Minix 文件系统已被 ext2 所取代，但它仍被用在 root/boot 盘和 RAM 虚拟盘中。所以应选“y”或“m”以便系统可以读这些通用的软盘格式。
- /proc filesystem support (CONFIG_PROC_FS) [Y/n/?] /proc 是提供系统运行状态信息的虚拟文件系统，没有它系统将不能正常运转，必须选“y”。
- Second extended fs support (CONFIG_EXT2_FS) [Y/m/n/?] ext2 是 Linux 的标准文件系统，此处应选“y”，除非准备把 Linux 作为在一个 DOS 分区中的 UMSDOS 文件系统运行。
- System V and Coherent filesystem support (CONFIG_SYSV_FS) [N/y/m/?] 是否支持 Xenix FS，SystemV/386 FS 和 Coherent FS？一般选“n”。若选“y”，请参考联机帮助文件。

17. Network File Systems（网络文件系统）

- Coda filesystem support (advanced network fs) (CONFIG_CODA_FS) [N/y/m/?] Coda 是一种类似于 NFS 的先进网络文件系统，在某些特性上比 NFS 更为优越。若希望支持该文件系统请选“y”或“m”，并参考 Documentation/filesystems/coda.txt。
- NFS filesystem support (CONFIG_NFS_FS) [Y/m/n/?] 如果想使用 NFS 协议挂上和使用远程系统的磁盘分区，选“y”或“n”。
- SMB filesystem support (to mount WfW shares etc.) (CONFIG_SMB_FS) [N/y/m/?] 如果需要系统对 SMB 文件系统的支持，需选“y”。SMB（Server Message Buffer）是 Windows for Workgroup（WfW），Windows 95、Windows NT 和 Lan Manager 等用来实现在局域网上文件和打印机共享协议。选择该项后可以用特殊的程序挂卸局域网上的 SMB 文件系统。象

使用 NFS 文件系统一样方便地使用 Windows 95/NT 等操作系统的共享文件。若想使用 Windows 95 上的共享文件系统，还必须选择“SMB Win95 bug work-around”以修正 Windows 95 的 SMB 服务器的某些缺陷。

- NCP filesystem support (to mount NetWare volumes) (CONFIG_NCP_FS) [N/y/m/?] 若需要 NCP 文件系统支持，选“y”。

18. Partion Types（分区形式）

- BSD disklabel (BSD partition tables) support (CONFIG_BSD_DISKLABEL) [N/y/?]
- Macintosh partition map support (CONFIG_MAC_PARTITION) [N/y/?]
- SMD disklabel (Sun partition tables) support (CONFIG_SMD_DISKLABEL) [N/y/?]
- Solaris (x86) partition table support (CONFIG_SOLARIS_X86_PARTITION) [N/y/?]

这四个选项是对四种特殊分区形式的支持，一般不会用到，故选“n”。若要选“y”，请参考联机帮助文件。

19. Console drivers（控制台驱动）

- VGA text console (CONFIG_VGA_CONSOLE) [Y/n/?] 该选项允许你通过一个通用 VGA 的标准显示方式在文本方式下使用 Linux。一般应选“y”。
- Video mode selection support (CONFIG_VIDEO_SELECT) [N/y/?] 该选项允许内核启动时进行文本模式选择（利用 BIOS 的一些功能）。一般选“n”。

20. Sound（声音）

- Sound card support (CONFIG_SOUND) [N/y/m/?] 如果计算机上装有声卡，此处选“y”。注意，你需要知道声卡的各项参数：I/O 端口、中断和 DMA 通道等。然后选择相应的驱动。如果声卡是 PnP 型的，并希望在启动时使用 ISA PnP 工具（参考 http://www.roestock.demon.co.uk/isapnptools/），那么需将声卡驱动编译为可加载模块，选“m”，并在 PnP 设置之后调用该模块（sound.o）。

21. Kernel hacking（内核监视）

- Magic SysRq key (CONFIG_MAGIC_SYSRQ) [N/y/?] 若选“y”，你可以对系统进行一些控制，即使是系统工作不正常时，详情请参阅联机帮助。建议你选“n”，除非你真的有能力控制一切。

19.2.2 编译内核和用新内核引导

内核配置完成后，在 /usr/src/linux 目录下执行 **make dep** 命令以正确建立所有从属文件。从属文件中包含着每个 .c 文件中引用的头文件（.h）的全路径名。完成后，在旧版本的内核中你还应该做 **make clean**。这会清除内核编译的所有目标文件以及其他东西。在重建一个内核之前不要忘记这个步骤。

在完成 **make dep** 及 **make clean** 工作之后，就可以执行 **make zImage** 或 **make zdisk** 命令（这部份需要的时间较长）。**make zImage** 将会编译内核，并且在 /usr/src/linux/arch/i386/boot 目录下产生一个名为 zImage 的文件，这就是新的内核映象文件（如果 **make zImage** 产生的内核太大，编译过程的最后将提示用 **make bzImage** 命令编译产生一个 bzImage 文件）。**make zdisk** 的功能类似于 **make zImage**，但是它会把内核放到 /dev/fd0 上。“zdisk”对于测试新内核很方便，如果它工作不正常，只要把磁盘拿掉用旧的内核启动即可。当然它还有其他的一些用途。内核是经过压缩的，所以在名字前面有个“z”。压缩过的内核执行的时候会自行解压缩。

如果用户在配置内核时选择了可加载模块，这时必须做 **make modules** 和 **make modules_install**。可加载模块将被安装到类似 /lib/modules/2.2.3/ 的目录下。更详细的信息请参考 Documentation/modules.txt 文档。

现在，需要备份老的内核，以便在新内核不能正常运行时使用老的内核。例如把老内核 /vmlinuz 备份为 /vmlinuz.old。备份后，把新的内核 /usr/src/linux/arch/i386/boot/zImage 拷贝到系统放置老内核的地方覆盖老内核，例如拷贝为 /vmlinuz，然后编辑 /etc/lilo.conf 文件，在后面加进老内核引导的一段：

```
# For old kernel
image = /vmlinuz.old      # 老内核的文件名
   label = old            # 引导老内核所用的标记
   root  = /dev/hda1      # 根文件系统
   read-only
```

现在需要重新安装 LILO（通常是运行命令 /sbin/lilo）。注意，每次安装新的内核之后都必须重新运行 **lilo**，即使没有改变 /etc/lilo.conf 文件。

至此内核编译完成，可以关机用新内核引导了。如果新内核无法引导或工作不正常，可重新用老的内核引导（如，在 LILO boot: 后输入 old），然后检查新内核配置，尽量减少不必要的模块和驱动后重新编译。

19.2.3 附加的套件

Linux 内核中有许多源代码本身并没有说明的特性；这些特性一般是经由外来的软体来利用，在这里列出三个有用的套件：

1. kbd

Linux 的控制台有着令你吃惊的更多的特色，如切换字型，重新对映键盘，切换显示模式的能力等等。kbd 这套软体提供了能够让使用者做这些动作的支持程序，此外还有许多字型和几乎足以适用任何键盘的键盘对映表。可在放置内核源代码的站点上找到该软

件。

2. util-linux

Rik Faith（faith@cs.unc.edu）收集了一大堆 Linux 的工具，叫做 util-linux。现在则由 Nicolai Langfeldt（util-linux@math.uio.no）维护。可从 sunsite.unc.edu 的 /pub/Linux/system/misc 下获得。它包括了象 **setterm**，**rdev** 以及 **ctrlaltdel** 与内核有关的工具。就象 Rik 所说的，不要想都不想就把它装上去！你不需要安装此套件中的每一个东西，而且如果你真地这样做的话可能会引起严重的问题。

3. gpm

gpm 是 general purpose mouse 的简写。这个程序可以让你使用不同种类的鼠标在虚拟控制台之间剪贴，以及做一些其他的事情。

19.3 常见问题及解决方法

1. make clean

如果新内核有一些很奇怪的表现，有可能是因为在编译内核前忘了做清除 **make clean** 工作。症状从你的内核不正常地崩溃到奇怪的输入输出问题，一直到极慢的执行速度等等不一而足。当然最好也要确定做了 **make dep** 工作。

2. 巨大或缓慢的内核

如果新内核占用了大量的内存，或者它真的是很大很大，或者编译得过于缓慢，那么可能是因为你配置了太多不必要的东西（设备驱动程序，文件系统等等）。尽量不要配置你不会用到的东西，因为它会无谓地占用内存。内核过于臃肿的最明显的症状就是内存与硬盘之间频繁地进行交换，你的硬盘将转个不停。

用户可以用计算机全部内存的数量减掉 /proc/meminfo 里面的“total mem”或“**free**”指令所得的内存数量来得知内核使用了多少内存。也可以执行 **dmesg**（或者也可以查看内核的记录文件）。看起来就象这一行：

```
Memory: 15124k/16384k available (552k kernel code, 384k reserved, 324k data)
```

一个配置了很少垃圾的 386 显示如下：

```
Memory: 7000k/8192k available (496k kernel code, 384k reserved, 312k data
)
```

3. 内核无法编译

如果它没有被编译，那么可能是某个修补文件失败，或者是源代码有问题。也有可能是因为 **gcc** 版本不正确或坏掉了。确定 README 里所描述的符号链结都已正确建立。一般说来，如果内核不能编译，这表示在某些地方有严重的错误，可能必须重新安装某些工具。

在少数情况下，**gcc** 可能会由于硬件问题而当掉，错误信息类似于“xxx exited with signal 15”。如果发生了此类问题可以先试着重新安装 **gcc**。然后将外部 cache 关掉，减少一些 RAM。若内核编译成功，那么就是计算机的硬件有问题。进一步的信息请参考 http://www.bitwizard.nl/sig11/。

4. 新内核不能启动

这可能是因为没有执行 **lilo**，或是没有正确的配置。比如，应该是“boot = /dev/hda1”，而不是“boot = /dev/hda2”等。

5. 系统显示 “warning: bdflush not running”

这是一个相当严重的问题。从 1.0 版以后的内核开始（大概是在 1994 年四月二十日左右），有个名为“**update**”会周期性地更新文件系统缓冲区的程序被升级或取代掉了。下载“**bdflush**”的源代码，然后编译（你可能会希望在旧版的内核下执行编译及安装）。它会以“**update**”为名自行安装，并且在重新开机以后新内核应该会运作良好。

6. 系统说 undefined symbols 而且无法编译

你可能有 ELF 编译器（gcc 2.6.3 或以后的）而且是 1.2.x（或更早的）内核源代码。一般修正的方法是将这几行加到 arch/i386/Makefile 的顶端：

```
AS=/usr/i486-linuxaout/bin/as
LD=/usr/i486-linuxaout/bin/ld -m i386linux
CC=gcc -b i486-linuxaout -D__KERNEL__ -I$ (TOPDIR)/include
```

这会以 a.out 程序库来编译 1.2.x 内核。

7. 系统显示关于 obsolete routing requests 的奇怪信息

取得新版的 **route** 程序及其他与 **route** 有关的程序。/usr/include/linux/route.h（这是 /usr/src/linux 下的一个文档）已经做了修改。

8. 防火墙功能无法在 1.2.0 上工作

至少升级到 1.2.1 版。

9. “Not a compressed kernel image file”（非压缩内核映象文件）

不要用在 /usr/src/linux 产生的 vmlinux 做为你的启动内核映象，[..]/arch/i386/boot/zImage 才是正确的。

10. 关于可加载模块

可加载模块能够节省内存，而且很容易设定。模块工具可以从你取得内核的地方找到，如 modules-x.y.z.tar.gz。选择与当前内版本相等或稍低的最接近的 x.y.z，用 **tar zxvf modules-x.y.z.tar.gz** 解开。**cd** 到它产生的目录（modules-x.y.z），看一下 README，然后按照安装指示执行（通常很简单，就象 **make install** 之类的）。然后在 /sbin 应该出现 **insmod**、**rmmod**、**ksyms**、**lsmod**、**genksyms**、**modprobe** 以

及 **depmod** 等程序。

insmod 命令可将一个模块插入运行中的内核。模块通常以 .o 为结尾，例如要插入 drv_hello.o 模块，就用 **insmod drv_hello.o** 命令。要了解当前内核正在使用的模块，用 **lsmod** 命令：

```
blah# lsmod
Module:         #pages:  Used by:
drv_hello           1
```

“drv_hello”是模块的名称，它用了一页 (4K) 的内存，而且目前没有其他的内核模块使用它。要卸载此模块，用 **rmmod drv_hello**。注意 **rmmod** 需要的是模块名称，而不是文件名。其他工具用法请查阅有关在线文档。

第二十章

网 络 应 用

Linux 与众不同的之处不仅是因为它的价格，关键在于其傲人的能力。

- Linux 是个真正的 32 位多任务系统，其稳固性与能力足以应用在大学甚至大型公司。
- 既可以在目前看来极低级的 386 平台上运行，也可以在研究中心才会用到的巨型超级并行计算（ultra-parallel）计算机上。
- 实现了跨平台使用，Intel/Sparc/Alpha 等平台上都有可用的版本，而且实验性的版本可支持到 Power PC 以及内嵌（embedded）在其他的操作系统中（SGI，Ultra Sparc，AP1000+，Strong ARM，MIPS R3000/R4000）。
- 最后，说到网络能力，Linux 当然是个不错的选择。不仅是因为网络与操作系统被紧密地集成在一起，同时还有相当可观的应用程序等着大家来自由地使用。

本书的第十四章给大家介绍了网络的基本概念，本章首先概要地介绍 Linux 对网络的支持，然后具体介绍两个方面的应用实例：PPP 和 Apache 的应用。

20.1 Linux 对网络的支持

20.1.1 网络通讯协议

Linux 支持许多不同的网络通讯协议：

1. TCP/IP 通讯协议

Linux 从一开始就提供 TCP/IP 的网络能力。虽说是东拼西凑而成，但是却稳定、快速、可靠，同时也是 Linux 成功的关键因素之一。

2. TCP/IP 通讯协议第 6 版本（IPv6）

有时也被写成 IPng（下一代网际网络通讯协议）是 IPv4 通讯协议的升级版本，用来解决定址上的许多问题。这些问题包括：可用的 IP 地址不足，缺乏处理即时信息流的机制，缺乏网络的安全控制等等。实行扩充定址法之后，就能增加 IP 定址的空间（IPv6 的地址长度是 IPv4 的四倍），同时路由选择的效率将极大的提高。IPv4 采用的是分级定址法，根据网络的规模大小分为 A、B、C 三类，没有弹性，造成路由表过大；而 IPv6 采

用的是分类定址法，仅区分使用类型的范围，其余根据实际需要以 CIDR 方式分配，地址空间的分配更有效率，并能减缓路由表的增大。Linux 已经有支持 IPv6 的 beta 版本，至于正式的版本可能要等到 2.2.3 版本以后的新版 Linux 内核的发行。

3. IPX/SPX 通讯协议

IPX/SPX（网际网络封包交换/循序封包交换）是由 Novell 公司以“Xerox 网络系统”（XNS）通讯协议为蓝本，开发出来的专用通讯协议。IPX/SPX 通讯协议在 1980 年代初期很有名，几乎成为 Novell 公司 NetWare 产品的代名词。NetWare 成为第一代局域网络操作系统（NOS）的非官方标准。Novell 公司同时也为他们的网络操作系统配备了商业应用程序套件和用户端网络连接工具。

Linux 为 IPX/SPX 通讯协议提供了非常全面的支持，使得它能够被设定成：IPX 路由器（router）；IPX 桥接器（bridge）；NCP 客户端和/或 CP 服务器端（文件共享）；Novell 打印客户端，Novell 打印服务器端；开启 PPP/IPX 通讯协议，让 Linux 成为一个 PPP 的服务器端/用户端；通过 IP 管道（tunnel），使得两个执行 IPX 通讯协议的网络能够透过唯一的 IP 路径连通等等。

4. AppleTalk 通讯协议

Appletalk 是“苹果电脑”网络互连通讯协议群的代名词。它采用 peer-to-peer 对等式的网络模型，并提供基本的网络功能，例如文件及打印机的共享。每部机器可以同时成为用户端与服务器端，但是每部“苹果电脑”都需要装上相应软硬体才行。

Linux 支持全部的 Appletalk 网络功能，Netatalk 就是一个在内核层次支持 AppleTalk 通讯协议的实例。它能支持 AppleTalk 的路由选择，透过 AFP（AppleShare）提供 UNIX 和 AFS 文件系统的服务，提供 UNIX 打印机服务，以及透过“打印机存取协议”（PAP）存取 AppleTalk 打印机。

5. 广域网络（WAN）通讯协议 X.25，Frame-relay，... 等等

许多第三方合作厂商提供了可在 Linux 上使用的支持 T-1，T-3，X.25 以及 Frame Relay 等的产品。一般而言，这类产品需要特别的硬件。厂商除了提供硬件之外，也会提供通讯协议驱动程序的支持。

6. ISDN 通讯协议

Linux 内核内嵌有 ISDN 支持。内核模块 Isdn4linux 可以控制 ISDN PC 卡，并且可以将之模拟成使用 Hayes 命令集（“AT”命令）的调制解调器。

7. PPP，SLIP，PLIP 等通讯协议

Linux 内核内嵌有 PPP(端点-对-端点-通讯协议)，SLIP(串行线路使用 IP)，以及 PLIP（并行线路使用 IP）等通讯协议的支持。PPP 通讯协议是一般个人用户连接 ISP（Internet 服务提供商）最常用的方法。PLIP 通讯协议则是二部机器时便宜的连线方法，使用并行口和特制的缆线，连线速度可达 10kBps 到 20kBps。

8. 业余无线电通讯协议

Linux 内核内嵌有业余无线电通讯协议的支持。尤其令人感兴趣的是它支持 AX.25。AX.25 通讯协议提供连接导向和非连接导向二种操作模式，可以以自己的方式作点对点的连线，也可以传送其他通讯协议，如 TCP/IP 和 NetRom。

9. ATM 通讯协议

目前 Linux 对 ATM 通讯协议的支持还不成熟。有一个测试版的应用程序支持：纯 ATM 连线（PVCs 和 SVCs）；ATM 网络上运行 IP 通讯协议（IP over ATM）；ATM 网络模拟局域网络（LAN emulation)等等功能。

20.1.2 网络硬件的支持

Linux 支持各式各样的网络硬件，甚至一些过时的硬件也包含在内。请参考 Hardware How-To 以及 Ethernet How-To。

20.1.3 文件与打印的共享

Linux 可以同时作为文件与打印的服务器，于是成为一个重要的解决方案。

1. 在 Apple 的环境里

Linux 的 netatalk 使得 Macintosh 机器的用户端将 Linux 系统视为网络上的另一台 Macintosh，然后连上 Linux 服务器，分享其文件系统，共享其打印机。

2. 在 Windows 的环境里

应用程序套件 Samba 可以让大多数的 UNIX（特别是 Linux）系统集成到 Microsoft 的网络里，成为客户端或服务器端。当作服务器端时，Windows 95，Windows for Workgroups，DOS，以及 Windows NT 等用户端能够使用 Linux 的文件系统与打印等服务。当作用户端时，Linux 工作站可以将共享的 Windows 文件系统安装（mount）成本地的文件系统。

3. 在 Novell 的环境里

Linux 可以被构造成 NCP 的用户端或服务器端，因此 Novell 与 UNIX 两类的用户端可以透过 Novell 网络使用文件以及打印等服务。

4. 在 UNIX 的网络环境里

建议使用 NFS 的方法来共享文件。NFS 是“网络文件共享”（Network File Sharing）的缩写，是一种通讯协议，最初由 Sun Microsystems 公司开发，是多部机器之间共享文件的方法。用户端会将服务器端“分享出来（exported）”的文件系统“安装（mounts）”在自己的文件系统下。所安装的文件系统在用户端机器上看起来，就好象是机器本身的文件系统一样。

20.1.4 Linux 对 Internet/Intranet 所提供的服务

Linux 是 Internet/Intranet（网际网络/内部网络）上常用的服务器平台。Intranet 主要

是应用在组织内部，目的是发布和集合公司内部的信息。Linux 可为 Internet 与 Intranet 提供的服务有：电子邮件、网络新闻、Web 服务器、Web 浏览器、FTP、域名服务器（DNS）、DHCP 与 bootp 通讯协议、网络信息（NIS）、认证等。本章的后半部分将详细介绍其中的 Web 服务器和代理服务器功能。

20.1.5 远端执行应用程序服务

UNIX 最惊人的特性之一就是它支持以远端和分布的方式执行应用程序，也许迄今为止许多新手还是不知道。

1. Telnet 方式

Telnet 是一个程序，它让人们使用远端的电脑，就好象是真的在那台电脑面前一样。Telnet 是 UNIX 上最强大的工具之一，它使得远端管理机器成为可能。从用户的角度来讲，它是个有趣的程序。它可以让用户在 Internet 的任何地方，以远端的方式取用其文件及程序。

2. 远端命令方式

在 UNIX 中，特别是在 Linux 上，远端命令方式的出现，让我们能够透过 shell 与远端的电脑沟通。例如：rlogin，让我们能够以与 telnet 类似的方法登录远端机器；rcp 让我们能够与远端机器之间进行远端的文件传输等等。值得一提的是透过 remote shell 下命令程序 rsh，我们可以不必实际登录到远端机器，就能在该机器上执行命令。

3. X 视窗方式

任何 X 视窗系统，均由两部分组成——X 服务器和一个或多个 X 用户端。服务器直接控制屏幕显示，并且监控所有的输出入设备例如键盘，鼠标屏幕等。用户端，则正好相反，无法直接取用屏幕——它要通过服务器，才能操作所有的输入输出动作。用户端是“真正”执行应用程序或是其他工作的地方。每当用户端与服务器连线时，服务器就会开启一或多个视窗，为该用户端管理输入输出动作。

4. 虚拟网络计算（VNC）方式

虚拟网络计算（Virtual Network Computing，简称 VNC）。它基本上是一个远端显示系统，让我们不仅在执行程序的机器上能看到计算作业的桌面环境，而且在 Internet 的任何地方，即使使用各种不同的机器也都能看到。你可以在 Windows NT 或 95 的机器上执行 MS-Word 程序，而将输出结果显示在 Linux 机器上；也可以在 Linux 机器上执行应用程序，而将输出结果显示在其他 Linux 或 Windows 机器上。令人吃惊的是你只需使用一个移植到 Linux 的 SVGAlib 图形程序库，就可以让只有 4Mb 内存的 386 机器变成全功能的 X 终端机。

20.1.6 Linux 对网络互连的支持

Linux 的网络功能包罗万项。一部 Linux 机器，可以被构造成路由器（router），桥接器（bridge）等等。下面介绍一些特别有用的网络功能：

1. 路由器（Router）

Linux 的内核内嵌有路由选择（routing）功能。一部 Linux 机器，可以被构造成一台 IP 或 IPX 路由器，而其花费仅是商业路由器的零头而已。最近发布的内核，包含了一些设定路由器的特殊功能选项：

- 多目的传播（Multicasting）：可让 Linux 机器成为一个将 IP 包传播到多个目的地址的路由器。使用 MBONE 时，就需要这种路由器。MBONE 是 Internet 上一种高频宽的网络，它能够载送声音和影象的广播信号。
- 策略性 IP 路由选择（IP policy routing）：一般路由器（router）处理所收到的包时，仅以包的最终目的地址为路由选择的依据，但是路由选择也可以将源地址与包所抵达的网络一起纳入考虑。

2. 桥接器（Bridge）

Linux 的内核内嵌有以太网络桥接器（ethernet bridge）支持，其作用就是让不同以太网区段（Ethernet segments）上面的各个节点，使用起来就象是在同一个以太网络上一样。多部桥接器（Bridge）放在一起，再加上 IEEE802.1 标准的 spanning tree 演算法的使用，可以建构一个更大的以太网络。

3. IP 伪装（Masquerading）功能

IP 伪装在 Linux 上是一个处于发展中的网络功能。如果一部 Linux 主机连接到 Internet，而且其 IP 伪装功能处于开启状态，则与它相连的其他电脑（不论是在相同的 LAN 上，或是通过调制解调器相连）就算是他们没有使用正式分配的 IP 地址，都同样可以连往 Internet。这降低了上网的费用，因为可以多人使用同一条调制解调器连线来上 Internet，同时也增加了安全性（从某些方面来看，其功能象是一个防火墙（firewall），因为外界网络无法连接非正式分配的 IP 地址）。

4. IP 计帐（Accounting）功能

这也是 Linux 内核的选用功能，用于 IP 网络流量的追踪、包的记录，以及产生一些统计的结果。

5. IP 别名(aliasing）功能

Linux 内核所提供的这个功能，使得我们可以在同一个低阶网络设备的驱动程序下，设定多重的网络地址（例如，在同一块以太网络卡上，设定两个 IP 地址）。可以依照服务器程序所监看网络地址的不同，来区分不同的服务功能（例如，“多重主机（multihosting）”、“虚拟网域（virtual domains）”以及“虚拟主机服务（virtual hosting services）”等）。

6. 网络流量控制（Traffic Shaping）功能

网络流量控制功能，是一种虚拟的网络服务，它可以限制输出到另一个网络设备的信息流速率。这个功能在某些场合（ISP）特别有用，它能够限制每个使用者可以使用的频宽。例如，某些阿帕奇（Apache）模块（仅限于网页服务），可以用来限制客户端 IP 连

线的个数或所使用的频宽。

7. 防火墙（Firewall）功能

防火墙是将私有网络从公用网络环境中保护并独立出来的设备。它能够依据每个封包所含的源地址、目的地址、端口以及封包形态等信息来控制封包的流通与否。

Linux 内核内嵌有防火墙支持，同时还存在不同类型的防火墙工具套件（toolkits）。TIS 和 SOCKS 就是这种防火墙工具套件。这二种防火墙工具套件非常完整，若能与其他工具合并使用，则可阻断/重导各类的网络流量与协议。而且经由设定文件或 GUI 程序，可以实现不同的网络流量控制策略。

8. 端口转递（Port Forwarding）功能

有互动能力站点越来越多，他们使用 cgi-bins 或 Java applets 程序，来存取数据库或其他服务。而这类存取方式，可能会造成安全上的问题，所以数据库所在的机器，不应该直接连上 Internet。端口转递功能针对这类存取问题，提供了较为理想的解决方案。透过防火墙，进入到特定端口编号的 IP 封包，可以被改写，然后转递到内部实际提供服务的服务器上。内部服务器所回复的封包也会被改写，使其看起来是来自防火墙。

9. 负载均衡（Load Balancing）功能

通常对数据库/网页的存取，在多个用户端同时向一个服务器提出服务请求时，会有负载均衡的需求。负载均衡的功能就是指存在多部相同的服务器时，将服务请求转送到负载较轻的服务器上去。可以应用网络地址转换（Network Address Translation，简称 NAT）技术的子功能 IP 伪装来达到这个目的。网络管理者可以用一个逻辑服务器集合共享同一个 IP 地址的做法，取代过去仅使用单一服务器提供网页服务或其他应用的方式。利用负载均衡的算法，将连线要求转向至特定的服务器上。这个虚拟的服务器会改写进来与出去的封包，所以用户端对服务器的存取是透明的，他们会以为只有一台服务器。

10. EQL（串行连线的负载均衡驱动程序）

EQL 已被集成到 Linux 的内核中。如果你有二条串行连线接到其他电脑（通常需要二部调制解调器和二条电话线路），而且你在线路上面使用 SLIP 或 PPP（可以在电话线路上，传递 Internet 流量的通讯协议），此时使用 EQL 驱动程序就可以将二条串行连线看成一条二倍速的连线。当然，另外一端也必须支持这个功能才可以。

11. 代理服务器（Proxy Server）

在 Linux 上存在有数种代理服务器。一个普遍的解决方案就是阿帕奇的 proxy 模块（本功能将在以后的章节中详细叙述）。另一个更完整与稳定的 HTTP proxy 工具程序就是 SQUID。

12. 按需拨接（Dial on demand）功能

按需拨接（dial on demand）功能，使得电话拨接动作完全透明，使用者只会看到有一条固定的网络线路连接到远端的站点。

13. 建立通讯协议隧道

Linux 内核允许我们建立通讯协议隧道（也就是说将通讯协议封装起来）。它可以在 IP 网络连线上建立 IPX 隧道，这使得两个 IPX 网络可以透过唯一的 IP 网络连线互接。它也可以建立 IP-IP 隧道。

20.1.7 Linux 对网络管理的支持

1. 网络管理的应用

网络管理与远端管理这方面的应用工具非常多。目前较为吸引人的两个远端管理计划是 linuxconf 和 webmin。此外还有许多其他管理工具，如网络流量分析工具，网络安全工具，监视工具，设置工具等等。你若有兴趣可以到 `http://www.sunsite.unc.edu/pub/Linux/system/network/` 上去看一看。

2. 简易网络管理协议（SNMP）

简易网络管理协议（Simple Network Management Protocol，简称 SNMP）是提供 Internet 网络管理服务的通讯协议。通过它我们可以监视和设置路由器、桥接器、网卡、交换器（switch）等网络设备。

20.2 PPP

PPP（点对点协议）是在串行连接上运行 IP （网际网络协议）以及其他网络协议的一种机制，串行连接可以是直接的串行连接（使用 null-modem 缆线）或是使用调制解调器和电话线的连接（当然也包括如 ISDN 的数字线路）。

使用 PPP，可以把你的 Linux PC 连接到 PPP 服务器上并存取该服务器所连接的网络资源（几乎）就如直接连接在该网络上一样。你也可以把你的 Linux PC 设为一台 PPP 服务器，这样一来其他电脑就可以拨入你的电脑并且存取你的资源。因为 PPP 是一种点对点 (peer-to-peer) 的系统，因此你也可以通过两台 Linux PC 上的 PPP 把网络连接在一起（或是把局域网络连接到网际网络上）。与标准的以太网络连线相比，最主要的差异当然是速度——标准的以太网络连线理论上可以达到 10 Mbps（每秒百万位），而调制解调器最快是 56 kbps（每秒千位）。同时，依据 PPP 连线的型态，某些应用以及服务在使用上可能会有些限制。

PPP 是一种完完全全的点对点协议，拨接的机器以及接受拨接的机器之间（在技术上）并没有差异。当你拨入一个节点要建立 PPP 连线时，你是客户端。你所连线的那台机器是服务器端。你若设定一台 Linux 机器使其接收并处理拨入的 PPP 连线，那么你正在设立一台 PPP 服务器。

任何 Linux PC 都可以是 PPP 服务器端或客户端——如果你有一个以上的串行口（以及调制解调器，如果有必要）的话还可以同时扮演这两种角色。如前所述，就 PPP 而言，一只连线建立那么客户端与服务器端之间并没有什么差异。为了清楚起见，本文把启动呼叫（即“拨入”）的那台机器称作客户端，而把回应电话，核对拨入请求（利用使用者代号，密码以及其他可能的机制）的那台机器称作服务器端。

在下面的章节里我们将分别讨论如何将 Linux PC 设定为客户端和服务器端。

在这方面 Linux 有许多不同的套件，它们有自己的特点和行为模式。特别要指出的是，Linux（以及 UNIX）电脑有两种不同的初始化方式，界面设定等。这两种分别是 BSD 系统初始化与 System V 系统初始化。最常用的套件有：Slackware，使用 BSD 形式的系统初始化；Red Hat（及其 Caldera），使用 SysV 系统初始化（略作修改）；Debian，使用 SysV 系统初始化。

BSD 形式的系统初始化通常将其启动文件放在 /etc/ 下面，这些文件是：

```
/etc/rc
/etc/rc.local
/etc/rc.serial
（也可能有其他文件）
```

最近，一些 BSD 系统起始模式的启动文件存放在 /etc/rc.d 目录下，而不是将所有的东西都放到 /etc 下。

System V 起始模式将启动文件存放在 /etc/... 或 /etc/rc.d/... 及下列的一系列子目录中：

```
drwxr-xr-x   2 root     root         1024 Jul  6 15:12 init.d
-rwxr-xr-x   1 root     root         1776 Feb  9 05:01 rc
-rwxr-xr-x   1 root     root          820 Jan  2  1996 rc.local
-rwxr-xr-x   1 root     root         2567 Jul  5 20:30 rc.sysinit
drwxr-xr-x   2 root     root         1024 Jul  6 15:12 rc0.d
drwxr-xr-x   2 root     root         1024 Jul  6 15:12 rc1.d
drwxr-xr-x   2 root     root         1024 Jul  6 15:12 rc2.d
drwxr-xr-x   2 root     root         1024 Jul 18 18:07 rc3.d
drwxr-xr-x   2 root     root         1024 May 27  1995 rc4.d
drwxr-xr-x   2 root     root         1024 Jul  6 15:12 rc5.d
drwxr-xr-x   2 root     root         1024 Jul  6 15:12 rc6.d
```

20.2.1 将 PPP 设定为客户端

因为 PPP 需要设置网络设备，变更内核以及诸如此类的动作，所以需要以 root 身份登录以完成这些工作。

1. 下载/安装 PPP 软件

如果你的 Linux 套件中并未包含 PPP 软件，你可以从 ftp://sunsite.unc.edu/pub/Linux/system/network/serial/ppp/ 下载 ppp-2.3.4.tar.gz，这是 Linux PPP daemon 软件。由于目前常用的 Red Hat 等 Linux 套件中均包含有 PPP 软件，所以 ppp-2.3.4 的编译和安装本书不作叙述，请读者参考随机及其他相关文档。

2. 在编译内核时加入 PPP 支持

Linux 的 PPP 运作包含两部分：上面提到的 PPP daemon；PPP 的内核支持。详细内容请参阅上一章的有关叙述。

如果在启动时内核报告如下信息，那么说明内核已将 PPP 支持编译在内：

```
PPP Dynamic channel allocation code copyright 1995 Caldera, Inc.
PPP line discipline registered.
```

3. 从 ISP 处取得必要的信息

要正确地设定 PPP 服务器，因而必须取得 PPP 服务器如何运行的信息：

- 拨接服务的电话号码。如果你使用的是分机，还需要知道外线的拨出号码——0 或 9？
- 服务器使用动态还是静态的 IP 地址？如果服务器使用静态的 IP 地址，那么需要知道在 PPP 连线中你这端要使用哪个 IP 地址。如果 ISP 提供一合法 IP 地址的次网络，那么需要知道你能使用的 IP 地址及网络屏蔽。大部分的网际网络服务提供者都使用动态的 IP 地址，这对于你可以使用的服务会有些限制。
- ISP 的域名服务器 IP 地址是什么？虽然只需要一个，但最少应该得有两个。
- 该服务器是否需要使用 PAP/CHAP？如果是这样需要知道你所使用的用户名（user name）和密码（password）。
- 服务器启动 PPP 的方式？是自动启动还是在登录后发出什么指令来启动服务器端的 PPP？如果你必须执行指令来起动 PPP 的话，是什么指令？
- 如果服务器是微软的 Windows NT 系统，它是否使用微软的 PAP/CHAP 系统？

4. 设定串行口及调制解调器

为了连上 PPP 服务器并取得最佳的资料传输速率，你的串行口和调制解调器必须设定正确。

（1）关于串行口

注意：DOS 下的 COM1 在 Linux 下是/dev/ttyS0（和/dev/cua0）。以前，Linux 用 cuaX 表示拨出的串行口名而 ttySx 表示拨入的名称。自内核 2.0.X 之后应该可以用 ttySx 同时表示拨入及拨出的名称，据称 cuaX 的设备名称将在未来版本的内核中消失。

如果你有四个串行口，标准 PC 上让 COM1 与 COM3 共用 IRQ4 且让 COM2 与 COM4 共用 IRQ3。如果其他设备和串行口共用一个 IRQ 的话可能会有问题，必须确定调制解调器串行口拥有自己唯一的一个 IRQ。许多现在的串口卡（与品质较佳的主机板上的串行口）允许将串行口上的 IRQ 移开。如果你在用 Linux 2.0.X 内核，可以用“**cat /proc/interrupts**”检查使用中的 IRQ，将得到类似的输出结果：

```
           CPU0
  0:    6766283   timer
  1:      91545   keyboard
  2:          0   cascade
  4:     156944   + serial
  7:     101764   WD8013
 10:     134365   + BusLogic BT-958
 13:          1   math error
 15:    3671702   +serial
```

这里显示了一个串行口在 IRQ4（一个鼠标），一个串行口在 IRQ15。

如果对计算机的硬件不熟悉，尽量不要去改变这些设置。如果必须改变，一定要确实知道你在做些什么！因为不单需要打开电脑的外壳，在板卡上拔下并调整 jumper，你还需要知道其他硬件和软件对 IRQ 的需求！如果真地将串行口移到了非标准的 IRQ，那么

你需要告诉 Linux 每一个口所用的 IRQ 地址。这可以用“**setserial**”指令完成，而且最好将它放到 rc.local，或在 SysV 系统中由 rc.local 所呼叫的 rc.serial 里面成为开机过程的一部份，比如加入指令：

```
/bin/setserial -b /dev/ttyS2 IRQ 11
/bin/setserial -b /dev/ttyS3 IRQ 15
```

此外如果使用由 kernald 进程所负责的动态载入串行模块还要注意，必须在每次载入模块时重新设定 IRQ 。这是因为如果串行模块被卸载之后，Linux 就会忘了这些特殊的设定。

还要注意串行口与速度相容性的问题。如果你使用的是高速（外接式）调制解调器（14400 baud 或更高），那么串行口速度必须能够处理这种调制解调器，特别是当调制解调器在压缩信息时。这时串行口需要 UART (Universal Asynchronous Receiver Transmitter)，比如常见的是 16550(A)。如果你在使用一部旧的机器（或旧的串行卡），很可能串行口上只有旧的 8250 UART，这在使用高速调制解调器时可能会发生问题。使用指令“**setserial -a /dev/*ttySx***”可以显示使用的 UART 类型。如果没有 16550A 型的 UART，最好去买一块新的串口卡。

（2）配置调制解调器

为完成这件工作请仔细阅读你的调制解调器使用手册！大部分的调制解调器都有 PPP 所需求的出厂预设选项。最基本的配置是：硬件流量控制 (RTS/CTS) (&K3——许多采用 Hayes 指令集的调制解调器)。还要注意其他的设置（使用标准 Hayes 指令）：

表 20-1 一些标准 Hayes 指令选项

选项	含 义
E1	开启指令的本地回应（chat 运作所需）
Q0	回报执行结果代码（chat 运作所需）
S0=0	关闭自动回应（除非你想让调制解调器接听电话）
&C1	只在连线之后侦测载波
&S0	DataSetReady（DSR）永远设为开启

如果你的调制解调器支持 split speed operation，应该把调制解调器的串行接口锁定在最高速度（通常是 115200 baud，但对 14400 调制解调器来说可能是 38400 baud）。还有一点，AT&F 指令可以使调制解调器恢复出厂时的设置。

另外还要注意串行流量控制的问题。有可能出现信息的输入量超出了电脑所能处理的范围的情况（电脑可能忙着做其他的事——Linux 是用户多任务操作系统）。为了确保信息不丢失（在缓冲区中的资料不会超载），需要一些控制信息流量的方法。有两种方法可以用于串行线路：使用硬件信号(Clear To Send/Request to Send - CTS/RTS)；使用软件信号(control S and control Q 即 XON/XOFF)。虽然后者用在终端机（文字）连接上可能很好，但是在 PPP 上的资料使用整个 8 bits 编码空间——而且信息中有可能存在会被转成 control S 以及 control Q 的位元组。所以，如果调制解调器设成使用软件流量控制的话，传输很容易被扰乱！对于使用 PPP 的高速连接（使用 8 bits 资料编码）来说必须使用硬

件流量控制。

（3）测试调制解调器的拨出功能

完成串行口与调制解调器的设定之后应试着拨到 ISP 上看看能否连上以确定设定是否有误。使用终端机通讯软件（如 minicom），拨到你想使用的 PPP 服务器。（注意：在这个阶段我们并不尝试建立 PPP 连接——只是要证实我们拥有正确的电话号码，以及为了登录并且起动 PPP，服务器究竟传送给我们什么信息）。在这个过程中，可以截取（记录到一个文件里）整个登录过程或者是小心地记下远端服务器传来的提示输入使用者名称及密码的信息（以及任何建立 PPP 连接需要下达的指令）。

如果你的服务器使用 PAP，你可能看不到登录的提示符号，而是（以文字表示的）连接通讯协议（看起来象是垃圾）出现在你的屏幕上。有些服务器可以用文字模式的使用者名称/密码方式或使用 PAP 登录，所以如果你的 ISP 使用 PAP 但你没有立刻在屏幕上看到垃圾，也并不表示你做错了。有些系统要求你先输入一些起始的文字，然后才启动标准的 PAP 程序。有些 PPP 服务器是被动的——它们就坐在那里等待而不送出任何信息，直到客户端拨入并送出合法的连接控制协议。如果你要连接的 PPP 服务器是被动模式的，你将看不到任何垃圾！有些服务器在你按下 ENTER 前不会启动 PPP——因此如果你正确登录但未看到垃圾的话可以按 ENTER 试试！

Linux 机器在每次拨接的时候要能够辨认的两个主要的提示：一个是要求输入使用者名称的提示；另一个是要求输入密码的提示。有些服务器每次登录时的提示不一样，所以最好多试几次。

如果服务器自动起动 PPP 的话，一旦登录完成，屏幕上将出现“垃圾”——这是 PPP 服务器端传送给你的机器以起动并且配置 PPP 连线的信息：

```
~y}#.!}!!!} }8}!}$}%U}'}&} } } } }%}& ...}'}'}(}^: .~~y}
```

这个时候，你就可以挂断你的调制解调器（通常是，快速地键入 +++ 一旦你的调制解调器回应 OK 然后接着键入 ATHO 指令）。

5. 设置域名服务器(DNS)

除了执行 PPP 及自动登录 PPP 服务器的文件之外，还有一些配置文件必须设置，以便让你的电脑能将诸如 www.interweft.com.au 的名称解析为 IP 地址，以便真正与那台电脑连接。这些文件是：/etc/resolv.conf 和 /etc/host.conf 。

PPP 服务器系统管理人员一般应该提供两个 DNS 的 IP 地址（只需要一个，但是在出问题时两个会更好一些）。按如下方式编辑（如果没有的话就建一个新的）/etc/resolv.conf 文件：

```
domain your.isp.domain.name
nameserver 10.25.0.1
nameserver 10.25.1.2
```

其拥有者和权限应为：

```
-rw-r--r--   1 root     root          73 Feb 19 01:46 /etc/resolv.conf
```

如果已经存在 /etc/resolv.conf 文件，只要简单的把 PPP 连线的 DNS 服务器 IP 地址加到已有的文件中即可。

同时还要检查 /etc/host.conf 文件是否设置正确：

```
order hosts,bind
multi on
```

它告诉你的名称解析器在向域名服务器询问之前先使用主机名称文件中的信息。

6. PPP 与 root 权限

因为在你的 Linux 电脑与另一个 PPP 服务器建立连接时需要操作网络界面（PPP 界面是一个网络界面）与内核的路由表，pppd 需要使用 root 的权限。

如果 root 以外的使用者要能建立 PPP 连线，那么 pppd 程序应该设为以 root 的身份执行（setuid）：

```
-rwsr-xr-x   1 root     root        95225 Jul 11 00:27 /usr/sbin/pppd
```

如果/usr/sbin/ppd 不是这样设定的，那么以 root 的身份执行“**chmod u+s /usr/sbin/pppd**”。这样就可以让 pppd 以 root 的权限执行，即使是由一般使用者启动。这使得一般使用者执行的 pppd 具有必要的权限建立网络界面及内核路由表。

但是以 root 的身份执行的程序在安全上存在潜在的漏洞，所以你对于设定为“setuid”的程序必须非常地小心。许多程序（包括 pppd）经过精心处理，使得用 root 的身份执行的危险降到了最低，所以这样做应该是安全的（天知道）。

如果你希望系统里的任何使用者都能启始 PPP 连接，那么应该把 ppp-on/off 指令文件设定为所有的人都可以读取 / 执行。如果不希望任何人都能起始 PPP 连线，那么就需要建立一个 PPP 组（编辑 /etc/group 文件）并且将 pppd 设定为以 root 的权限执行，拥有者是 root 而群组是 PPP，而其他的权限都关闭：

```
-rwsr-x---   1 root     PPP         95225 Jul 11 00:27 /usr/sbin/pppd
```

使 ppp-on/off 指令文件由使用者 root 以及群组 PPP 所拥有，使得 ppp-on/off 指令文件能由群组 PPP 读取 / 执行：

```
-rwxr-x---   1 root     PPP           587 Mar 14  1995 /usr/sbin/ppp-on
-rwxr-x---   1 root     PPP           631 Mar 14  1995 /usr/sbin/ppp-off
```

关闭其他的存取权限，把能够起动 PPP 的使用者加入 /etc/group 文件的 PPP 群组里。

即使如此，一般使用者仍然无法以软件的方式中止连接。执行 ppp-off 指令文件需要 root 权限。然而，任何使用者都可以关掉调制解调器（或将电话线由内接式调制解调器拔下）。

另外一种(更好的)办法是允许使用者使用 sudo 来起动 ppp 连接。这样可以提供更佳的安全性并且可以让你设定由可信任的使用者使用指令文件来启动/结束连接。使用 sudo 可以让任何可信任的使用者安全地启动/结束 PPP 连接。

7. 设定 PPP 选项

（1）选项文件

PPP 使用几个文件来建立并设定 PPP 连接。必须以 root 身份登录来建立相应目录，并编辑这些设定 PPP 连线所需的文件，即使你想让所有的使用者都能使用 PPP。

这些文件在 PPP 2.1.2 与 2.2 中的名称与位置都不同。在 PPP 2.1.2 中这些文件是：

```
/usr/sbin/pppd                  # PPP 执行文件
/usr/sbin/ppp-on                # 拨号/连线指令文件
/usr/sbin/ppp-off               # 断线指令文件
/etc/ppp/options                # 所有连线所使用的选项
/etc/ppp/options.ttyXX          # 给某一特定通讯口使用的选项
```

在 PPP 2.2 中这些文件是：

```
/usr/sbin/pppd                    # PPP 执行文件
/etc/ppp/scripts/ppp-on           # 拨号/连线指令文件
/etc/ppp/scripts/ppp-on-dialer    # 拨号的 chat 指令文件部份
/etc/ppp/scripts/ppp-off          # 断线指令文件
/etc/ppp/options                  # 所有连线所使用的选项
/etc/ppp/options.ttyXX            # 给某一特定通讯口使用的选项
```

在 /etc 目录里应该有个目录（如果不存在就以这样的权限创建）：

```
drwxrwxr-x   2 root     root         1024 Oct   9 11:01 ppp
```

如果这个目录已经存在，它应该包含有一个名为 options.tpl 的选项文件样板。它包含所有 PPP 选项的解释，所以请你把它印出来（配合 pppd 的线上使用手册来阅读）。虽然你可以使用这个文件作为创建 /etc/ppp/options 文件的基础，但是参照 options.tpl 自己建立一个可能会更好 ——短得多而且比较容易阅读/维护。

如果有多个串行线路/调制解调器（典型的例子是 PPP 服务器），可以建立一个通用的 /etc/ppp/options 文件，其中包含共同的选项。再为每一个需要个别设定以建立 PPP 连接的串行线路分别设立选项文件。这些文件名为 options.ttyx1，options.ttyx2 依此类推（其中 x 是与所使用的串行口相应的标识字符）。

对于单一 PPP 连线，可以直接使用 /etc/ppp/options 文件，也可以把所有的选项放进 pppd 指令作为参数。下面是一个简单的例子（无 PAP/CHAP 时）：

```
# /etc/ppp/opt_ons (NO PAP/CHAP)
#
# 避免 pppd 进入背景执行
-detach
#
# 使用调制解调器控制线
modem
# 使用 uucp 形态的锁定文件以避免他人取用串行设备
lock
# 使用硬件流量控制
crtscts
# 在路由表中将此连接建立为预设递送设备
defaultroute
# 不使用任何“逸出”控制序列
asyncmap 0
```

```
# 最大传送封包大小为 552 bytes
mtu 552
# 最大接收封包大小为 552 bytes
mru 552
#
#-------END OF SAMPLE /etc/ppp/options (no PAP/CHAP)
```

如果你使用 PPP 来连线到好几个不同的站点去的话，那么你就可以在 /etc/ppp/options.site 里面为每个站点建立选项文件，然后在你连线时指定选项文件作为 PPP 指令的参数（在 pppd 的指令中使用 file option-file 参数）。

（2）PAP/CHAP 认证

如果 PPP 服务器使用 PAP 认证（Password Authentication Protocol）或（Challenge /Handshake Authentication Protocol）认证，就需要多做一些工作。在上面的选项文件中，加上下面几行：

```
#
# 告诉 pppd 使用你的 ISP 名称做为认证过程中的“主机名称”
name <your ISP user name>     # 你需要改变这一行
#
# 如果你执行 PPP *服务器* 并且需要使用 PAP 或 CHAP 认证，适当除去下面的
# 注解。不要在你做为客户端连上服务器时使用此选项（即使它使用 PAP 或 CHAP）
# 因为这是告诉服务器必须为它自己在你的机器上提供认证（这几乎是不可能的，
# 所以连接会失败）。
#+chap
#+pap
#
# 如果你使用在 /etc/ppp/pap-secrets 文件中经编码的密码，除去下面一行的注解。
# 注意：这和 Windows NT 上的远端存取服务里的微软编码密码是不同的。
#+papcrypt
```

同时必须建立密码文件 /etc/ppp/pap-secrets 和 /etc/ppp/chap-secrets。这些文件的拥有者必须为 root，群组为 root，同时为了安全起见文件权限应为 740。假定 ISP 给你的使用者名称是 fred 而密码是 flintstone，那么应该在 /etc/ppp/options[.*ttySx*] 里设定 name fred 这个选项，并且在 /etc/ppp/pap-secrets 文件中设置：

```
# Secrets for authentication using PAP
# client  server   secret          acceptable local IP addresses
fred       *       flintstone
```

上面四栏以空格分界，而且如果使用动态 IP 最后一项可以是空白。目前的 pppd 版本要求你要有互相验证的方法——这就是说你必须能够让从你的机器到远端服务器以及从远端服务器到你的机器这两种验证都能进行。所以，如果你的机器是 fred 而远端是 barney 的话，在各自的 /etc/ppp/options.ttySx 里，你应设置 name fred remotename barney，而远端机器应设为 name barney remotename fred。在 fred 机器上的 /etc/ppp/chap-secrets 文件为：

```
# Secrets for authentication using CHAP
# client     server  secret          acceptable local IP addresses
fred         barney  flintstone
barney       fred    wilma
```

barney 机器上的 /etc/ppp/chap-secrets 文件为：

```
# Secrets for authentication using CHAP
# client     server  secret          acceptable local IP addresses
barney       fred    flintstone
fred         barney  wilma
```

如果你同时使用几家 ISP 的帐号。如果用户名都不相同这并不是问题。然而，如果在两个系统（或者甚至是全部）上具有相同的用户名，那么从 /etc/ppp/pap-secrets 中进行正确地选择时会有问题。不必担心，PPP 提供了克服此困难的机制。 PPP 允许你使用 remotename 选项为远端服务器设定一个“假想名称”。假设在连接两部 PPP 服务器都使用名称 fred，以此方式建立 /etc/ppp/pap-secrets 文件：

```
fred    pppserver1      barney
fred    pppserver2      wilma
```

这样，与 pppserver1 连线时，在 pppd 选项中使用 name fred remotename pppserver1，而与 pppserver2 连线时使用 name fred remotename pppserver2。

8. 以手动方式建立 PPP 连接

现在已经建立了 /etc/ppp/options 和 /etc/resolv.conf 等文件（以及 /etc/ppp/pap|chap-secrets 文件，如果有需要的话），下面可以通过手动建立 PPP 连接来测试这些设定（一旦手动连接成功，我们将会使该过程自动化）。注意这需要通讯软件能在不复位调制解调器的情况下结束。可以使用 Minicom，用 ALT+Q 结束 （旧版 Minicom 使用 CTRL_A_Q）。

确定以 root 身份登录。启动通讯软件（如 minicom），拨接到 PPP 服务器并且象平常一样的登录。然后，在不要复位调制解调器的情况下结束通讯软件并且在 Linux 的提示符号下（以 root 的身份）键入“**pppd -d -detach /dev/ttySx 38400 &**”。-d 选项开启侦错功能——ppp 连线起动时的“对话”将会记录到系统记录中去，便于调试。此时用户可以通过发出“**ifconfig**”指令观察 PPP 设备的情况：

```
ppp0      Link encap:Point-Point Protocol
          inet addr:10.144.153.104  P-t-P:10.144.153.51 Mask:255.255.255.0
          UP POINTOPOINT RUNNING  MTU:552  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0
          TX packets:0 errors:0 dropped:0 overruns:0
```

其中 inet addr:10.144.153.104 是该连接中用户这端的 IP 地址。P-t-P:10.144.153.51 是服务器端的 IP 地址。

如果没有列出 ppp 设备或得到的类似于下面的信息，用户就需要检查各项设置了：

```
ppp0      Link encap:Point-Point Protocol
          inet addr:0.0.0.0  P-t-P:0.0.0.0  Mask:0.0.0.0
          POINTOPOINT  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0
          TX packets:0 errors:0 dropped:0 overruns:0
```

用户现在还可以使用“**route -n**”命令来查看路由表：

```
Kernel routing table
Destination  Gateway       Genmask          Flags  MSS    Window Use Iface
10.144.153.3 *             255.255.255.255 UH     1500     0        1 ppp0
```

```
127.0.0.0       *               255.0.0.0       U     3584   0        11 lo
10.0.0.0        *               255.0.0.0       U     1500   0        35 eth0
default         10.144.153.3    *               UG    1500   0         5 ppp0
```

请注意，其中有两项指到我们的 ppp 设备。第一项是主机递送（以 H 标志表示）并且允许我们看到这部我们正在连接的主机——但再来就没有了。第二项是预设路由（由 pppd 的 defaultroute 选项所建立的）。这份递送路径告诉我们的 Linux PC 将任何不在内部以太网络的封包送到我们指定的网络——也就是到 PPP 服务器本身。而 PPP 服务器负责为我们将封包递送到网际网络并将回应的封包发送回给我们。如果在路由表中没有这两个项目，那么一定出了问题。特别是如果你的系统记录（syslog）显示一信息告知 pppd 无法取代已存在的预设递送路径，那么你一定是在什么地方预设了递送路径。必须检查一下系统的起始文件以找出预设递送是在哪里建立的(由 route add default... 指令指定的)。将它改成：route add net...。

现在试着 ping 其他机器（不是 PPP 服务器自己），例如：

```
% ping sunsite.unc.edu
PING sunsite.unc.edu (152.2.254.81): 56 data bytes
64 bytes from 152.2.254.81: icmp_seq=0 ttl=254 time=190.1 ms
64 bytes from 152.2.254.81: icmp_seq=1 ttl=254 time=180.6 ms
64 bytes from 152.2.254.81: icmp_seq=2 ttl=254 time=169.8 ms
64 bytes from 152.2.254.81: icmp_seq=3 ttl=254 time=170.6 ms
64 bytes from 152.2.254.81: icmp_seq=4 ttl=254 time=170.6 ms
(按下 CTRL + C 终止)
--- sunsite.unc.edu ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 169.8/176.3/190.1 ms
```

如果没有收到任何回应，试着 ping 一下 ISP 提供的 DNS 的 IP 地址。如果能 ping 通，那么问题出在 /etc/resolv.conf 文件。

如果一切正常，键入 ppp-off 指令结束连线，一段短暂的停顿之后，调制解调器应该会自己挂断电话。如果这个指令无法动作的话，那么关掉调制解调器并消除 pppd 所建立的锁定文件：**rm -f /var/lock/LCK..ttySx**

9. 连接自动化

一旦手动登录成功，最好建立一些指令文件来自动完成连接工作。如果 ppp 套件安装正确，应该有两（三）个例子文件：PPP-2.1.2 中为 ppp-on 和 ppp-off，在 /usr/sbin 下；在 PPP-2.2 中为 ppp-off、ppp-on 和 ppp-on-dialer，在 /etc/ppp/scripts 目录下。如果你正在使用 PPP 2.1.2，强烈建议删除这些例子文件。它们有潜在的问题。

（1）ppp-on 指令文件

ppp-on 指令文件如下：

```
#!/bin/sh
#
# Script to initiate a PPP connection. This is the first part of the
# pair of scripts. This is not a secure pair of scripts as the codes
# are visible with the 'ps' command.  However, it is simple.
#
# These are the parameters. Change as needed.
TELEPHONE=555-1212      # The telephone number for the connection
ACCOUNT=george          # The account name for logon (as in 'George Burns')
```

```
PASSWORD=gracie          # The password for this account (and 'Gracie Allen'
)
LOCAL_IP=0.0.0.0         # Local IP address if known. Dynamic = 0.0.0.0
REMOTE_IP=0.0.0.0        # Remote IP address if desired. Normally 0.0.0.0
NETMASK=255.255.255.0 # The proper netmask if needed
#
# Export them so that they will be available to 'ppp-on-dialer'
export TELEPHONE ACCOUNT PASSWORD
#
# This is the location of the script which dials the phone and logs
# in.  Please use the absolute file name as the $PATH variable is not
# used on the connect option.  (To do so on a 'root' account would be
# a security hole so don't ask.)
#
DIALER_SCRIPT=/etc/ppp/ppp-on-dialer
#
# Initiate the connection
#
#
exec /usr/sbin/pppd debug /dev/ttySx 38400 \
        $LOCAL_IP:$REMOTE_IP \
        connect $DIALER_SCRIPT
```

你可以编辑这个指令文件，加入你的用户名、密码和 ISP 的电话号码。而且，如果你在 /etc/ppp/options 文件里设定了 IP 地址，删除“$LOCAL_IP:$REMOTE_IP \”
这一行。另外确定变量 DIALER_SCRIPT 指向将要使用的拨号指令文件（ppp-on-dialer）的全部路径名称。

（2）ppp-on-dialer 指令文件

ppp-on-dialer 指令文件如下：

```
#!/bin/sh
#
# This is part 2 of the ppp-on script.  It will perform the connection
# protocol for the desired connection.
#
/usr/sbin/chat -v                                       \
        TIMEOUT         3                               \
        ABORT           '\nBUSY\r'                      \
        ABORT           '\nNO ANSWER\r'                 \
        ABORT           '\nRINGING\r\n\r\nRINGING\r'    \
        ''              \rAT                            \
        'OK-+++\c-OK'   ATH0                            \
        TIMEOUT         30                              \
        OK              ATDT$TELEPHONE                  \
        CONNECT         ''                              \
        ogin:--ogin:    $ACCOUNT                        \
        assword:        $PASSWORD
```

ppp-on-dialer 指令文件用于实际建立起用户的 PPP 连接。chat 指令文件是一系列“期待字串”和“送出字串”的配对。我们在送出某些字符串之前，一般都要先期待某些字符串出现。如果我们在没有先接收到任何字符串的情况下要送出某些字符串的话，必须使用空的期待字串（通过 ‘’ 指明）。在没有送出任何字符串的情况下要期待某些字符串作法也类似。而且，如果字符串包含好几个字（例如 NO CARRIER），必须把字符

串用引号括住，这样 chat 才会把字符串当作是一个字符串。另外 chat 指令文件一般全都放在同一行里。反斜线是用来让一行可以跨过数行的实体行（便于人们阅读）而并不是指令文件本身组成的一部份。下面我们来逐行解释一下 ppp-on-dialer 指令文件：

exec /usr/sbin/chat –v：起动 chat ，-v 告诉 chat 将其所有的输出 / 入拷贝到系统记录里（通常是/var/log/messages）。chat 指令文件的运行稳定可靠后去掉-v。

TIMEOUT 3：设定接收输入超时时限为三秒。

ABORT '\nBUSY\r'：如果接收到 BUSY 字串，就中止执行。

ABORT '\nNO ANSWER\r'：如果接收到 NO ANSWER 字串，中止执行。

ABORT '\nRINGING\r\n\r\nRINGING\r'：如果接收到（重复的）RINGRING 字串，中止执行。这是因为有人正在给你打电话！

"\rAT：不期待调制解调器传送任何信息并且送出 AT 字串。

OK-+++\c-OK ATH0：它所代表的是...期待 OK ，如果没有收到（因为调制解调器并不在指令模式下）那么送出 +++ （使调制解调器返回指令模式的标准 Hayes 相容调制解调器字串）并且期待 OK；接著送出 ATH0 （调制解调器挂断字串）。这是为了应付调制解调器无法断线的情况。

TIMEOUT 30：设定指令文件其余部份执行的超时时限为 30 秒。

OK ATDT$TELEPHONE：期待 OK （调制解调器对 ATH0 指令的回应）并且拨接到我们想要呼叫的号码。

CONNECT ''：期待 CONNECT 字串（连接成功）并且不送出任何回复信息。

ogin:--ogin: $ACCOUNT：期待登录提示(...ogin:)，但是如果在超时前没有接收到该提示，则送出一个返回字元(return)，然后再次等待登录提示。当接收到提示时，送出用户名（存放在$ACCOUNT 环境变量里）。

assword: $PASSWORD：期待密码提示并且送出密码（存放在$PASSWORD 环境变量里）。

这份 ppp-on-dialer 指令文件适用于登录后会自动启动 pppd 的服务器。然而某些服务器需要明白地在服务器上启动 PPP。我们可以这样做：在这指令文件的尾端（在 password 这行后面）加上期待/送出字符串对——寻找 Shell 提示符（特别小心对 Bourne Shell 有特殊意义的字符，如$或/和左右方括号等）。一旦 chat 找到了 Shell 提示符，chat 必须用相应指令启动 ISP 的 PPP 服务器。比如 PPP 服务器使用标准的 Linux Bash 提示符[hartr@kepler hartr]$，我们需要键入 ppp 命令以启动服务器上的 PPP，可以在 ppp-on-dialer 指令文件尾端加入“hartr--hartr ppp”。注意，别忘了在前一行结束前加上 \ 以便让 chat 认为这整个指令文件是在一行里！

如果你的 ISP 使用 PAP/CHAP 验证，那么 chat 指令文件会简单得多。chat 指令文件所要做的只是拨接电话，等待接上线，然后就让 pppd 去处理登录事宜：

```
#!/bin/sh
#
# This is part 2 of the ppp-on script. It will perform the connection
# protocol for the desired connection.
#
exec /usr/sbin/chat  -v                                 \
        TIMEOUT      3                                  \
        ABORT        '\nBUSY\r'                         \
        ABORT        '\nNO ANSWER\r'                    \
        ABORT        '\nRINGING\r\n\r\nRINGING\r'       \
```

```
        ''                      \rAT                            \
        'OK-+++\c-OK'   ATH0                                    \
        TIMEOUT         30                                      \
        OK              ATDT$TELEPHONE                          \
        CONNECT         ''                                      \
```

（3）ppp-off 指令文件

ppp-off 指令文件如下：

```
#!/bin/sh
##################################################################################
#
# Determine the device to be terminated.
#
if [ "$1" = "" ]; then
        DEVICE=ppp0
else
        DEVICE=$1
fi

################################################################################
#
# If the ppp0 pid file is present then the program is running. Stop it.
if [ -r /var/run/$DEVICE.pid ]; then
        kill -INT `cat /var/run/$DEVICE.pid`
#
# If the kill did not work then there is no process running for this
# pid. It may also mean that the lock file will be left. You may wish
# to delete the lock file at the same time.
        if [ ! "$?" = "0" ]; then
                rm -f /var/run/$DEVICE.pid
                echo 'ERROR: Removed stale pid file'
                exit 1
        fi
#
# Success. Let pppd clean up its own junk.
        echo "PPP link to $DEVICE terminated."
        exit 0
fi
#
# The ppp process is not running for ppp0
echo 'ERROR: PPP link is not active on $DEVICE'
exit 1
```

类似于 **pppd** 使用 ***-d*** 选项打开侦错信息记录，***debug*** 选项也是这一功能。因为我们正在使用新的指令文件建立新的连线，可以先不管此选项。（警告：记录 pppd 的信息可能会大大增加系统记录文件，如果你的硬盘空间不够大，在尝试了数次失败的连线之后将出现问题）。如果一切工作正常，可以去除此选项。

如果你不使用 /etc/ppp/options 或 /etc/ppp/options.ttySx 文件中的 ppp 选项，可以在 **pppd** 上用 ***file*** 选项指定使用的文件名，例如：

```
exec /usr/sbin/pppd debug file options.myserver /dev/ttyS0 38400 \
```

10. 测试连线指令文件

开启两个 Xterm（如果你在 X 下）或开启两个虚拟主控台并且以 root 登录。在其

中一个窗口或虚拟控制台中，发出指令“**tail -f /var/log/messages**”（或是其他系统记录文件）以查看调试信息。在另一个窗口或虚拟控制台中发出指令“**ppp-on &**”（或是任何你所编辑的 /usr/sbin/ppp-on 文件名）。如果你没有在这个指令的后面指定“**&**”（使得指令文件进入背景执行），那么在 ppp 结束（连线结束时）之前你都不能回到终端提示下。现在切换到追踪系统记录文件的窗口，将会出现如下信息：

```
Oct 21 16:09:58 hwin chat[19868]: abort on (NO CARRIER)
Oct 21 16:09:59 hwin chat[19868]: abort on (BUSY)
Oct 21 16:09:59 hwin chat[19868]: send (AT^M)
Oct 21 16:09:59 hwin chat[19868]: expect (OK)
Oct 21 16:10:00 hwin chat[19868]: AT^M^M
Oct 21 16:10:00 hwin chat[19868]: OK -- got it
Oct 21 16:10:00 hwin chat[19868]: send (ATDT722298^M)
Oct 21 16:10:00 hwin chat[19868]: expect (CONNECT)
Oct 21 16:10:30 hwin chat[19868]: ^M
Oct 21 16:10:22 hwin chat[19868]: ATDT722298^M^M
Oct 21 16:10:22 hwin chat[19868]: CONNECT -- got it
Oct 21 16:10:22 hwin chat[19868]: send (^M)
Oct 21 16:10:22 hwin chat[19868]: expect (ogin:)
Oct 21 16:10:23 hwin chat[19868]: kepler login: -- got it
Oct 21 16:10:23 hwin chat[19868]: send (hartr^M)
Oct 21 16:10:23 hwin chat[19868]: expect (ssword:)
Oct 21 16:10:23 hwin chat[19868]:  hartr^M
Oct 21 16:10:23 hwin chat[19868]: Password: -- got it
Oct 21 16:10:23 hwin chat[19868]: send (??????^M)
Oct 21 16:10:23 hwin chat[19868]: expect (hartr)
Oct 21 16:10:24 hwin chat[19868]: [hartr -- got it
Oct 21 16:10:24 hwin chat[19868]: send (ppp^M)
Oct 21 16:10:27 hwin pppd[19872]: pppd 2.1.2 started by root, uid 0
Oct 21 16:10:27 hwin pppd[19873]: Using interface ppp0
Oct 21 16:10:27 hwin pppd[19873]: Connect: ppp0 <--> /dev/cua1
Oct 21 16:10:27 hwin pppd[19873]: fsm_sdata(LCP): Sent code 1, id 1.
Oct 21 16:10:27 hwin pppd[19873]: LCP: sending Configure-Request, id 1
Oct 21 16:10:27 hwin pppd[19873]: fsm_rconfreq(LCP): Rcvd id 1.
Oct 21 16:10:27 hwin pppd[19873]: lcp_reqci: rcvd MRU
Oct 21 16:10:27 hwin pppd[19873]: (1500)
Oct 21 16:10:27 hwin pppd[19873]:  (ACK)
Oct 21 16:10:27 hwin pppd[19873]: lcp_reqci: rcvd ASYNCMAP
Oct 21 16:10:27 hwin pppd[19873]: (0)
Oct 21 16:10:27 hwin pppd[19873]:  (ACK)
Oct 21 16:10:27 hwin pppd[19873]: lcp_reqci: rcvd MAGICNUMBER
Oct 21 16:10:27 hwin pppd[19873]: (a098b898)
Oct 21 16:10:27 hwin pppd[19873]:  (ACK)
Oct 21 16:10:27 hwin pppd[19873]: lcp_reqci: rcvd PCOMPRESSION
Oct 21 16:10:27 hwin pppd[19873]:  (ACK)
Oct 21 16:10:27 hwin pppd[19873]: lcp_reqci: rcvd ACCOMPRESSION
Oct 21 16:10:27 hwin pppd[19873]:  (ACK)
Oct 21 16:10:27 hwin pppd[19873]: lcp_reqci: returning CONFACK.
Oct 21 16:10:27 hwin pppd[19873]: fsm_sdata(LCP): Sent code 2, id 1.
Oct 21 16:10:27 hwin pppd[19873]: fsm_rconfack(LCP): Rcvd id 1.
Oct 21 16:10:27 hwin pppd[19873]: fsm_sdata(IPCP): Sent code 1, id 1.
Oct 21 16:10:27 hwin pppd[19873]: IPCP: sending Configure-Request, id 1
Oct 21 16:10:27 hwin pppd[19873]: fsm_rconfreq(IPCP): Rcvd id 1.
Oct 21 16:10:27 hwin pppd[19873]: ipcp: received ADDR
Oct 21 16:10:27 hwin pppd[19873]: (10.244.153.51)
Oct 21 16:10:27 hwin pppd[19873]:  (ACK)
```

```
Oct 21 16:10:27 hwin pppd[19873]: ipcp: received COMPRESSTYPE
Oct 21 16:10:27 hwin pppd[19873]: (45)
Oct 21 16:10:27 hwin pppd[19873]:  (ACK)
Oct 21 16:10:27 hwin pppd[19873]: ipcp: returning Configure-ACK
Oct 21 16:10:28 hwin pppd[19873]: fsm_sdata(IPCP): Sent code 2, id 1.
Oct 21 16:10:30 hwin pppd[19873]: fsm_sdata(IPCP): Sent code 1, id 1.
Oct 21 16:10:30 hwin pppd[19873]: IPCP: sending Configure-Request, id 1
Oct 21 16:10:30 hwin pppd[19873]: fsm_rconfreq(IPCP): Rcvd id 255.
Oct 21 16:10:31 hwin pppd[19873]: ipcp: received ADDR
Oct 21 16:10:31 hwin pppd[19873]: (10.144.153.51)
Oct 21 16:10:31 hwin pppd[19873]:  (ACK)
Oct 21 16:10:31 hwin pppd[19873]: ipcp: received COMPRESSTYPE
Oct 21 16:10:31 hwin pppd[19873]: (45)
Oct 21 16:10:31 hwin pppd[19873]:  (ACK)
Oct 21 16:10:31 hwin pppd[19873]: ipcp: returning Configure-ACK
Oct 21 16:10:31 hwin pppd[19873]: fsm_sdata(IPCP): Sent code 2, id 255.
Oct 21 16:10:31 hwin pppd[19873]: fsm_rconfack(IPCP): Rcvd id 1.
Oct 21 16:10:31 hwin pppd[19873]: ipcp: up
Oct 21 16:10:31 hwin pppd[19873]: local  IP address 10.144.153.104
Oct 21 16:10:31 hwin pppd[19873]: remote IP address 10.144.153.51
```

OK，现在完成了对 PPP 的配置，下面用户就可以使用 PPP 网络了。当用户不想使用 PPP 的时候，可以使用标准的 ppp-off 指令终止 PPP 连接（——你必须是 root 或者是 PPP 群组的组员！）

20.2.2 使用 PPP 连接两个网络

从原理上说，连接一台 Linux PC 到 PPP 服务器和使用 PPP 连接两个局域网络没有什么差别，因为 PPP 是一种点对点的协议。但是建议你了解一下路由是如何建立的问题。你可以阅读 NET-2 howto 和 Linux Network Administrator Guide (NAG)，或是《TCP/IP Network Administration》(O'Reilly and Assoc - ISBN 0-937175-82-X)。如果你想在连接的某一边使用一个网络号码的次网络分割，你可以参考 Linux Subnetworking mini-HOWTO 。

为了连接两个局域网络，你使用的必须是不同的 IP 网络编号（或是同样网络编号的次网络），而且必须使用静态的 IP 地址或使用 IP 伪装。如果你想要使用 IP 伪装，参阅 IP masquerade mini-howto 的有关介绍。

1. 设定 IP 地址

同其他局域网络的网络管理人员协商 PPP 界面使用的 IP 地址。 如果你使用静态的 IP 地址，这可能也会要求你拨接特定的电话号码。

对于这种连线，最好还是使用特定的调制解调器及串行口，并编辑与之对应的 /etc/ppp/options[.ttySx] 文件（可能需要改变 /etc/ppp/options 文件，并且也要为其他的连线建立适当的 options.ttySx 文件）。使用前面所讲的设定静态 IP 地址拨接的方法，在上述选项文件里设定 PPP 连接中你这一端的 IP 地址。

2. 设定递送路径

你必须设定你的局域网络上的封包经过 PPP 连结所建立的界面递送出去。分两阶段进行。

首先，必须建立从执行 PPP 连接的机器到远方网络的递送路径。如果该连接通往网

际网络，那么可以通过 pppd 的选项 'defaultroute'，自行建立的预设递送路径来处理，你不必做任何事。然而如果只是连接两个局域网络，那么必须加入一个指定的网络递送路径——在 /etc/ppp/ip-up 指令文件中使用 'route' 指令（参考后面的内容）。

然后，告诉局域网络上的其他电脑这台 Linux 电脑是通往远方网络的“闸道”。当然，这些工作在该连接的另一端也要做！这就需要指定的网络递送路径，而不是预设递送路径。

3. 网络安全

如果使用 PPP 连接将你的局域网络连接到网际网络上去，或者只是连接到其他局域网络，你都必须考虑安全性的问题。强烈建议你设立防火墙！

你还应该在以此方式连到外面的局域网络或网际网络之前，先通知你的局域网络管理者。否则你可能惹上严重的麻烦！

4. 建立连接之后 - /etc/ppp/ip-up 指令文件

一旦 PPP 连接建立后，pppd 会找寻 /etc/ppp/ip-up 指令文件。如果这个指令文件存在并且可执行，那么 PPP 服务程序就会自动执行该指令文件。这允许你自动执行任何必需的特殊递送路径指令及任何你想在每次 PPP 连接启动时执行的动作。

/etc/ppp/ip-up 指令文件就是一般普通的 Shell 指令文件，可以做任何指令文件能做的事。例如，你可能想要 sendmail 赶快处理在邮件伫列中等待外送的信息。类似地，你也可以在 ip-up 里插入一些指令取回电子邮件（使用 POP）。不过使用 /etc/ppp/ip-up 也有一些限制：

- /etc/ppp/ip-up 使用局限的环境变量以增加安全性。也就是说你必须给出执行文件的全部路径等。
- 技术上来说，/etc/ppp/ip-up 是一个程序而非指令文件。它可以被直接执行，因而第一行必须是标准的 file magic (#!/bin/bash) 并且能被 root 读取及执行。

如果你连接的是两个局域网络，你需要设立一个到其他局域网络的指定递送路径。这可以很容易地通过 /etc/ppp/ip-up 指令文件办到。不过，在你的机器有多个 PPP 连接时要麻烦一些。因为 /etc/ppp/ip-up 这个指令文件是每一个 ppp 连线启动时都要执行的，所以必须小心地为每一个起动的连接设置正确的递送指令！

下面谈一下如何处理电子邮件的问题：

当两个局域网络的连接建立之后，你可能想要确定放在伫列中的电子邮件被清出——送到它的目的地。可以调用 sendmail 来完成。在 pppd 传递给指令文件的特定参数上使用 bash 的 `case' 叙述来完成这个工作。例如，下面所示是用来处理同某大学的广域网络连接和通往某个人家里以太网络的（也是由相同的 PPP 服务器处理）/etc/ppp/ip-up 指令文件。

```
#!/bin/bash
#
# Script which handles the routing issues as necessary for pppd
# Only the link to Newman requires this handling.
#
# When the ppp link comes up, this script is called with the following
# parameters
#       $1      the interface name used by pppd (e.g. ppp3)
#       $2      the tty device name
```

```
#       $3      the tty device speed
#       $4      the local IP address for the interface
#       $5      the remote IP address
#       $6      the parameter specified by the 'ipparam' option to pppd
#
case "$5" in
# Handle the routing to the Newman Campus server
        202.12.126.1)
                /sbin/route add -net 202.12.126.0 gw 202.12.126.1
# and flush the mail queue to get their email there asap!
                /usr/sbin/sendmail -q &
                ;;
        139.130.177.2)
# Our Internet link
# When the link comes up, start the time server and synchronise to the world
# provided it is not already running
                if [ ! -f /var/lock/subsys/xntpd ]; then
                        /etc/rc.d/init.d/xntpd.init start &
                fi
# Start the news server (if not already running)
                if [ ! -f /var/lock/subsys/news ]; then
                        /etc/rc.d/init.d/news start &
                fi
                ;;
        203.18.8.104)
# Get the email down to my home machine as soon as the link comes up
# No routing is required as my home Ethernet is handled by IP
# masquerade and proxyarp routing.
                /usr/sbin/sendmail -q &
                ;;
        *)
esac
exit 0
```

如果你执行连往广域网络的连接，你可以跟远端局域网络的网络管理者协调，请他们执行完全相同的动作。例如，在上面的例子中 Newman 校园网那一端的 /etc/ppp/ip-up 指令文件可如下设置：

```
#!/bin/bash
#
# Script which handles the routing issues as necessary for pppd
# Only the link to Hedland requires this handling.
#
# When the ppp link comes up, this script is called with the following
# parameters
#       $1      the interface name used by pppd (e.g. ppp3)
#       $2      the tty device name
#       $3      the tty device speed
#       $4      the local IP address for the interface
#       $5      the remote IP address
#       $6      the parameter specified by the 'ipparam' option to pppd
#
case "$5" in
        203.18.8.4)
                /usr/sbin/sendmail -q
                ;;
        *)
esac
```

```
exit 0
```

如果你只能使用动态 IP 地址方式的 PPP 连线连往 ISP，那么必须通过 ISP 机器上的帐号取得你的电子邮件。通常使用 POP（Post Office Protocol）协议来完成。可以使用“popclient”程序，当然可以在 ip-up 指令文件中加入相应的处理以便自动完成该任务。你还可以用类似的方法使用 slurp 或其他软件获取网络新闻等服务。

5. 使用 /etc/ppp/ip-down

你可以建立 /etc/ppp/ip-down 指令文件在连接结束之后执行，它可以用来还原任何在 /etc/ppp/ip-up 指令文件中做的特殊动作。

20.2.3 建立 PPP 服务器

有许多方法可以用于建立 PPP 服务器。这里仅介绍其中的一种（使用 Cyclades 多口串行卡以及一组自动转接的电话线路）：

1. 编译内核

你必须在内核中包含 IP forwarding 的功能。其他的功能 IP firewalls，accounting 等视情况而定。如果你使用多口串行卡，那么必须在内核中包含必要的驱动程序！

2. 服务器系统的概观

（1）使用相同的使用者名称 / 密码配对提供拨接 PPP 帐号以及 Shell 帐号。这样做的好处（对我们而言）是使用者只需要一个帐号就可以使用所有种类的连线。

（2）为了安全起见，在 PPP 服务器节点与网际网络之间设置有防火墙。当然这会限制某些使用者的存取。

（3）使用者建立连往 PPP 服务器连接的步骤（当然是在他们拥有有效的帐号后）是：

- 拨入 PPP 服务器自动转接拨号器（一个接有若干调制解调器的电话号码——第一台空闲的调制解调器会接听拨入的电话）。
- 使用一对有效的使用者名称和密码登录。
- 在 Shell 提示符号下，发出 ppp 指令以启动服务器上的 PPP 程序。
- 启动他们机器上的 PPP（可以是执行 Windows，DOS，Linux，MAC OS 或任何操作系统的机器）。

（4）PPP 服务器为每个拨接口设立不同的 /etc/ppp/options.tty*XX* 文件，使用动态分配的方法为远端设定 IP 地址。服务器对远端的客户端使用代理地址解析协议（proxyarp）来递送封包（用相应的 pppd 选项加以设定）。这避免了使用 routed 或是 gated。

（5）pppd 自动进行检测使用者是否从他们那端挂断，并控制调制解调器挂断，同时停掉 PPP 连接。

3. 所需的全部软件

你需要下列的软件：

- Linux，适当地编译以包含必要的选项。
- 适合于你内核的 pppd 版本。
- 一套能够自动处理调制解调器通讯的 getty 程序。这里我们使用的是 getty_ps2.0.7h，也可以考虑使用 mgetty。mgetty 可以侦测出使用 pap/chap 的呼叫（ pap 是 Windows 95 使用的标准）并自动地起动 pppd。
- 你的拨接使用者能够访问的域名服务器（DNS）。如果有可能的话应该执行你自己的域名服务器。

4. 设定标准的拨接(Shell access)

在设立你的 PPP 服务器之前，你的 Linux 机器必须能够处理标准的拨接访问。这里我们不做叙述，请你参阅 getty 的有关文件和 Serial HOWTO 里有关这项设定的内容。

5. 设立 PPP 选项文件

你需要为所有拨接口设立 /etc/ppp/options 文件，在该文件里加入通用的选项，比如：

```
asyncmap 0
netmask 255.255.254.0
proxyarp
lock
crtscts
modem
```

注意：没有使用任何(明显的)递送设定，特别是没有 defaultroute 选项。理由是因为 PPP 服务器所要做的就是将封包从 PPP 客户端递送到你的局域网络或网际网络并且将封包由你的局域网络递送到你的客户端。

proxyarp 选项在 PPP 服务器的地址解析协议(ARP)表中设立一个代理地址解析协议项目，其含义是“将所有要给 PPP 客户端的封包都送给我”。这是建立 PPP 客户端的递送路径最简单的方式，但你不能用这种方式在两个局域网络之间递送封包。你必须加入适当的网络递送选项而不能使用代理地址解析协议。

你可以通过为每一个拨接口分配 IP 地址的方法完成 IP 地址的动态分配。

下面是为每个拨接口建立一个 /etc/ppp/options.ttyXX 文件。在该文件里只要简单地放入本地（服务器）的 IP 地址及该口所要使用的 IP 地址。例如：kepler:slip01。注意，在这个文件里你可以使用合法的主机名称。

6. 设定 pppd

因为起动 ppp 连接隐含着配置内核设备（网络界面）及控制内核路由表的动作，所以需要特别的权限——事实上需要完整的 root 权限。

幸运的是， pppd 已被设计成可以安全地设定为以 root 的身份执行。所以你必须执行“**chmod u+s /usr/sbin/pppd**”。检查 pppd 文件是否是：

```
-rwsr-xr-x   1 root     root        74224 Apr 28 07:17 /usr/sbin/pppd
```

如果你没有这样做，使用者将不能设立他们的 PPP 连接。

7. 为 pppd 设定一个全域的别名(alias)

为了简化 PPP 拨接用户的连线程序，你可以建立一个全域的别名（放在 /etc/bashrc），这样在他们登录之后只要发出一个简单的指令就能起动服务器端的 ppp。使用如下语句：

```
alias ppp="exec /usr/sbin/pppd -detach"
```

其含义是：

- exec ：指以此指令所执行的程序替换正在执行的程序（在这个例子中是 Shell）。
- pppd -detach ：起动 pppd 并且不要把产生的程序放入背景执行。这确保 pppd 结束时不会留下任何程序。

当一个使用者登录时，他们在“w”的输出如下所示：

```
6:24pm  up 3 days,  7:00,  4 users,  load average: 0.05, 0.03, 0.00
User     tty      login@  idle   JCPU   PCPU  what
harry    ttyC0    3:05am  9:14                -
```

OK，这就是基本的 PPP 服务系统！感觉如何？

20.2.4 在 null modem (直接连线)上使用 PPP

没有调制解调器事情就变得简单多了。

首先，选择其中一部机器做为服务器，在串行口上设立 getty 以便让你可以从客户端使用 minicom 去取用此串行口以测试连接性。成功之后，可以除去这个 getty，除非你想用使用者名称/密码来确认连线。实际上你拥有这两部机器的实体控制权，你完全没有必要这样做。

现在，在服务器端除去 getty 并确认你已在两部机器上正确地使用 'setserial' 来设定串行口。

所有你要做的就是在两个系统上启动 pppd。假设你在两台机器上都使用 /dev/ttyS3 建立连线，那么在两部机器上执行指令：

```
pppd -detach crtscts lock <local IP>:<remote IP> /dev/ttyS3 38400 &
```

这样连接就建立起来了，但要注意你还没有指定递送路径。你可以在每部机器上用 **ping** 指令来测试连接。如果这样可以的话，终止其中一个 pppd 程序以结束连接。

20.3 阿帕奇（Apache）的应用

阿帕奇（Apache）是资源开放的 HTTP 服务器，其目标是与当前的 HTTP 标准保持同步，提供安全高效可扩充的 HTTP 服务功能。从 1996 年 4 月之后，阿帕奇成为 Internet 上最流行的 Web 服务器。Netcraft 1999 年 1 月所作的 WWW 服务器站点调查表明，Internet 上超过 53% 的 Web 站点使用阿帕奇（如果其衍生物也算在内的话将达到 58%），超过了其他任何一种 Web 服务器。

20.3.1 编译启动阿帕奇

阿帕奇的最新信息可以在阿帕奇网站 http://www.apache.org/ 上找到。该处会列出目前发行的版本，任何更新的公开测试版，同时还有镜像（mirror）网站与匿名文件传输（ftp）站点的细节。

1. 编译阿帕奇

新版的阿帕奇支持所谓的“可选模块”。然而，为了使这些模块富有效率，服务器必须知道哪些模块要编译进去；这就需要产生一段程序码（modules.c）来列出它们。如果你对阿帕奇提供的标准模块集满意，而且打算继续让它保持原样，那么你可以直接编辑阿帕奇提供的 Makefile，并且使用同以前一样的方法编译。如果想要使用可选模块，那么无论如何必须执行配置指令文件。做法如下：

（1）编辑“Configuration”文件。该文件包含每一种机器的 Makefile 设置，同时在此之后还有个额外的节段列出了要编译进去的模块，以及包含这些模块的文件名称：

a) 选择适合你机器的一种编译器以及编译选项。

b) 去掉想包含进去的模块上面的注解或者加上你自己撰写的自制（custom）模块的一些新行。注意，如果想要有 DBM 验证必须明确地配置进去（只要消掉对应的行上面的注解即可）。

（2）执行“**Configure**”这个指令 Shell：

```
% Configure
Using `Configuration' as config file
%
```

这将产生新版的 Makefile 以及 modules.c 文件。如果想要维护多个配置，那么可以这样，例如：

```
% Configure -file Configuration.ai
Using alternate config file Configuration.ai
%
```

（3）键入“**make**”进行编译。

编译之后，在 src/ 目录下将产生名为“httpd”的可执行文件。阿帕奇的可执行文件发行套件会提供这个文件。

放进阿帕奇发行套件里的模块是经过测试，并且有许多阿帕奇开发成员在使用的模块。在 <URL:http://www.apache.org/dist/contrib/modules/> 上可以找到这些成员或第三方成员（third parties）所提供的应付特殊需要或功能的其他模块。在该网页上有如何将这些模块连结到阿帕奇内核程序码里去的说明。

2. 配置阿帕奇

服务器将会读取三个配置指令文件。任何指令都可以出现在这些文件的任何一个中。这些文件的名称是相对于服务器的根目录（server root）的，而根目录是通过 **ServerRoot** 指令，或是由 -d 命令标志指定。依照惯例这些文件是：conf/httpd.conf，conf/srm.conf 和 conf/access.conf。conf/httpd.conf 包含控制服务程序运行的指令，文件名称可以配合 -f 命令标志加以改变。conf/srm.conf 包含控制服务器

提供给客户端的文件规格的指令，文件名称可以配合 **ResourceConfig** 指令加以改变。conf/access.conf 包含控制文件存取的指令，文件名称可以配合 **AccessConfig** 指令加以改变。注意，并不一定必须完全遵从这些惯例。服务器也会读取一个包含 MIME 文件型态的文件；这个文件名称通过 **TypesConfig** 指令设定，通常预设为 conf/mini.types。

在 ../conf 目录下有三个配置文件的发行版本：srm.conf-dist，access.conf-dist 以及 httpd.conf-dist。可以它们复制成所需的 srm.conf, access.conf 及 httpd.conf 文件。

首先编辑 httpd.conf，设定服务器一般的属性：端口号，执行者的身分等等。

接下来编辑 srm.conf 文件，设定文件树的根目录，指定诸如服务端解析的 HTML 或内部的 imagemap 解析等等功能。

最后编辑 access.conf 文件，做存取（access）的基本设定。

3. 启动阿帕奇

调用 **httpd** 程序，使用 **-f** 参数设定 httpd.conf 所在的完整路径，执行服务器程序。比如一个最普遍的例子：

```
/usr/local/etc/apache/src/httpd -f /usr/local/etc/apache/conf/httpd.conf
```

现在服务器应该已经开始执行。

httpd 程序可以通过网际网络服务程序 inetd 在每一次有连线要进入 HTTP 服务的时候启动；或者是另一种方式，它也可以作为服务程序（daemon）持续地执行，处理请求。无论选择哪一种方式，都必须设定 **ServerType** 指令告诉服务器它要如何执行。

下列选项可以用在 httpd 的命令列上：

-d *serverroot* 把 ServerRoot 这个参数的起始值设定为 *serverroot*。可以通过配置文件里面的 **ServerRoot** 命令来加以改变。预设值为 /usr/local/etc/httpd。

-f *config* 启动时执行在 *config* 文件里面的指令。如果 config 没有以 / 作为开始的话，那么它会被当作相对于 ServerRoot 的路径。预设值是 conf/httpd.conf。

-X 以单一程序（single-process）模式执行，只用于内部除错；服务程序不会脱离终端作业或是产生子程序。不要使用这个模式来提供正常的网页服务。

-v 打印 httpd 的版本，然后结束。

-? 打印 httpd 的选项列表，然后结束。

4. 记录文件

服务程序启动时，把父程序 httpd 的进程号存放在 log/httpd.pid 文件中。这个文件名称可以使用 **PidFile** 指令加以改变。管理者用程序号来重新启动或终止服务程序：一个 HUP 信号会使服务程序重新读取其配置文件，而一个 TERM 信号会使其终止执行。如果程序不正常死掉（或被杀掉），那么必须杀掉 httpd 子程序。

服务器会将错误信息记录到一个记录文件中，该文件预设为 log/error_log，文件名称可以通过 **ErrorLog** 指令设定；可以为不同的虚拟主机设定不同的错误记录。

服务器一般会将每个请求记录到一个传输文件，该文件预设为 logs/access_log。

文件名称可以通过 **TransferLog** 指令设定；可以为不同的虚拟主机设定不同的传输记录。

20.3.2 WWW 服务器的配置

WWW 服务器的行为完全取决于三个配置文件的设置：conf/httpd.conf，conf/srm.conf 和 conf/access.conf。这三个配置文件中的配置语句成为指令。下面分别进行介绍。

1. httpd.conf

Red Hat 5.1 自带的文件 httpd.conf 是一个可供参考的模板：

```
# This is the main server configuration file. See URL http://www.apache.org/
# for instructions.

# Do NOT simply read the instructions in here without understanding
# what they do, if you are unsure consult the online docs. You have been
# warned.

# Originally by Rob McCool

# ServerType is either inetd, or standalone.

ServerType standalone

# If you are running from inetd, go to "ServerAdmin".

# Port: The port the standalone listens to. For ports < 1023, you will
# need httpd to be run as root initially.

Port 80

# HostnameLookups: Log the names of clients or just their IP numbers
#   e.g.  www.apache.org (on) or 204.62.129.132 (off)
# You should probably turn this off unless you are going to actually
# use the information in your logs, or with a CGI.  Leaving this on
# can slow down access to your site.
HostnameLookups off

# If you wish httpd to run as a different user or group, you must run
# httpd as root initially and it will switch.

# User/Group: The name (or #number) of the user/group to run httpd as.
#  On SCO (ODT 3) use User nouser and Group nogroup
#  On HPUX you may not be able to use shared memory as nobody, and the
#  suggested workaround is to create a user www and use that user.
User nobody
Group nobody

# The following directive disables keepalives and HTTP header flushes for

# Netscape 2.x and browsers which spoof it. There are known problems with
# these

BrowserMatch Mozilla/2 nokeepalive
BrowserMatch Java1.0 force-response-1.0
```

```
BrowserMatch JDK/1.0 force-response-1.0

# ServerAdmin: Your address, where problems with the server should be

# e-mailed.

ServerAdmin root@localhost

# ServerRoot: The directory the server's config, error, and log files
# are kept in
# NOTE!  If you intend to place this on a NFS (or otherwise network)
# mounted filesystem then please read the LockFile documentation,
# you will save yourself a lot of trouble.

ServerRoot /etc/httpd

# BindAddress: You can support virtual hosts with this option. This option
# is used to tell the server which IP address to listen to. It can either
# contain "*", an IP address, or a fully qualified Internet domain name.
# See also the VirtualHost directive.

#BindAddress *

# ErrorLog: The location of the error log file. If this does not start
# with /, ServerRoot is prepended to it.

ErrorLog /var/log/httpd/error_log

# TransferLog: The location of the transfer log file. If this does not
# start with /, ServerRoot is prepended to it.

TransferLog /var/log/httpd/access_log

# PidFile: The file the server should log its pid to
PidFile /var/run/httpd.pid

# ScoreBoardFile: File used to store internal server process information.
# Not all architectures require this.  But if yours does (you'll know becaus
e
# this file is created when you run Apache) then you *must* ensure that
# no two invocations of Apache share the same scoreboard file.
ScoreBoardFile /var/run/apache_status

# ServerName allows you to set a host name which is sent back to clients for
# your server if it's different than the one the program would get (i.e. use
# "www" instead of the host's real name).
#
# Note: You cannot just invent host names and hope they work. The name you
# define here must be a valid DNS name for your host. If you don't understan
d
# this, ask your network administrator.

#ServerName new.host.name
# CacheNegotiatedDocs: By default, Apache sends Pragma: no-cache with each
# document that was negotiated on the basis of content. This asks proxy
# servers not to cache the document. Uncommenting the following line disable
s
# this behavior, and proxies will be allowed to cache the documents.
```

```
#CacheNegotiatedDocs

# Timeout: The number of seconds before receives and sends time out

Timeout 300


# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.

KeepAlive On

# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited amount.
# We reccomend you leave this number high, for maximum performance.

MaxKeepAliveRequests 100

# KeepAliveTimeout: Number of seconds to wait for the next request

KeepAliveTimeout 15

# Server-pool size regulation.  Rather than making you guess how many
# server processes you need, Apache dynamically adapts to the load it
# sees --- that is, it tries to maintain enough server processes to
# handle the current load, plus a few spare servers to handle transient
# load spikes (e.g., multiple simultaneous requests from a single
# Netscape browser).

# It does this by periodically checking how many servers are waiting
# for a request.  If there are fewer than MinSpareServers, it creates
# a new spare.  If there are more than MaxSpareServers, some of the
# spares die off.  These values are probably OK for most sites ---

MinSpareServers 8
MaxSpareServers 20

# Number of servers to start --- should be a reasonable ballpark figure.

StartServers 10

# Limit on total number of servers running, i.e., limit on the number
# of clients who can simultaneously connect --- if this limit is ever
# reached, clients will be LOCKED OUT, so it should NOT BE SET TOO LOW.
# It is intended mainly as a brake to keep a runaway server from taking
# UNIX with it as it spirals down...

MaxClients 150

# MaxRequestsPerChild: the number of requests each child process is
#  allowed to process before the child dies.
#  The child will exit so as to avoid problems after prolonged use when
#  Apache (and maybe the libraries it uses) leak.  On most systems, this
#  isn't really needed, but a few (such as Solaris) do have notable leaks
#  in the libraries.

MaxRequestsPerChild 100

# Proxy Server directives. Uncomment the following line to
```

```
# enable the proxy server:

#ProxyRequests On

# To enable the cache as well, edit and uncomment the following lines:

#CacheRoot /usr/local/etc/httpd/proxy
#CacheSize 5

#CacheGcInterval 4
#CacheMaxExpire 24
#CacheLastModifiedFactor 0.1
#CacheDefaultExpire 1
#NoCache a_domain.com another_domain.edu joes.garage_sale.com

# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, in addition to the default. See also the VirtualHost command

#Listen 3000
#Listen 12.34.56.78:80

# VirtualHost: Allows the daemon to respond to requests for more than one
# server address, if your server machine is configured to accept IP packet
s
# for multiple addresses. This can be accomplished with the ifconfig
# alias flag, or through kernel patches like VIF.

# Any httpd.conf or srm.conf directive may go into a VirtualHost command.
# See also the BindAddress entry.

#<VirtualHost host.some_domain.com>
#ServerAdmin webmaster@host.some_domain.com
#DocumentRoot /www/docs/host.some_domain.com
#ServerName host.some_domain.com
#ErrorLog logs/host.some_domain.com-error_log
#TransferLog logs/host.some_domain.com-access_log
#</VirtualHost>
```

下面按照出现的顺序对其中用到的指令进行解释：

ServerType 指令：

用法：ServerType 型态　　缺省：ServerType standalone

用于：server config　　状态：内核

ServerType 指令用于设定系统执行服务器的方式。可以是下列型态之一：inetd，服务器将由系统程序 inetd 执行，启动服务器使用的命令加在 /etc/inetd.conf 文件里；standalone，服务器将会作为服务程序（daemon）执行，启动服务器使用的命令加在系统启动指令文件里（/etc/rc.local or /etc/rc.d/...）。inetd 在这两个选项里较少使用。因为接收到每个 http 连线就会从头开始执行一份新的服务器拷贝；连线结束后，该程序结束。每次连线要付出的代价很高，但是出于安全方面的考虑，某些管理者喜欢这个选项。standalone 是 ServerType 使用最常用的设定，因为它更富有效率。服务器只需启动一次，即可服务所有的连线。如果使用阿帕奇来服务于一个拥有众多访问的站点，standalone 大概是唯一的选择。如果对安全性有很高的要求，可以用 inetd 模式来执行阿帕奇。两种方式都不能绝对确保安全性，大部分人喜欢用 standalone 方式，而 inetd 方

式相对而言更有能力抵御黑客的攻击。

Port 指令

用法：Port 数字　　缺省：Port 80

用于：server config　状态：内核

Port 指令用于设定服务器监听的网络端口号。数字是 0 到 65535 之间的某个值；某些端口号（特别是低于 1024 的）保留给特殊的功能。参阅 /etc/services 里定义的一些端口的列表；标准 http 协定使用 80 端口。Port 80 是 UNIX 的一个特别端口。所有低于 1024 的端口号都是保留给系统使用的，一般使用者（non-root）不能使用它们。但使用者可以使用较高的端口号。要使用 80 端口你必须以 root 帐号启动服务器。在连接到该端口后，接受请求之前，阿帕奇将会切换为通过 **User** 指令所设定权限较低的使用者身分。如果你不能使用 80 端口，选择任何其他没有使用到的端口。非 root 使用者必须选择高于 1023 的端口号，比如 8000 。

安全问题：如果你是以 root 启动服务器，确定不要把 User 设为 root 。如果你以 root 身分处理连线的话，你的站点可能会暴露在攻击下。

User 指令

用法：User 使用者的识别码　　缺省：User #-1

用于：server config　　状态：内核

User 这个指令设定服务器用来回答请求的使用者识别码。为能够使用这个指令，必须以 root 身分起始执行独立的服务器。UNIX 使用者识别码可以是下列情况之一：使用者的名称，通过名称来识别使用者；# 后跟使用者编号，通过使用这个编号识别他们。

使用者应该不能有存取外界所不能看到的文件的权限，而且与此类似，使用者应该不能执行对 httpd 要求而言没有意义的程序码。建议你特别为执行这个服务器设立新的使用者以及群组。某些管理者使用 nobody，但是这并非永远可行或合适的。注意：如果你以非 root 使用者的身分启动这个服务器，它将无法切换到权限较低的使用者，会继续以原使用者身份来执行。如果你真的是以 root 启动这个服务器，那么这个父程序一般仍然以 root 身分在执行。

安全问题：不要把 User（或 Group）设为 root 除非你确实知道你在做什么以及会有怎么样的危险。

Group 指令

用法：Group UNIX-群组　　缺省：Group #-1

用于：server config　　状态：内核

Group 指令设定服务器回答要求时所处的群组。为了能够使用这个指令，必须以 root 身分起始执行独立的服务器。UNIX 群组可以是下列情况之一：群组的名称，通过名称识别群组；# 后跟群组号码，通过号码识别某个群组。建议你特别设立一个群组来执行这个服务器。某些管理者使用 nogroup 这个使用者，但是这并非永远可行或合适的。注意：如果你以非 root 使用者的身分启动这个服务器，它将无法切换到指定的群组，并且取而代之的是它将会继续以原来的使用者所属群组来执行。

ServerAdmin 指令

用法：ServerAdmin 电子邮件地址　用于：server config, virtual host

状态：内核

ServerAdmin 设定电子邮件地址，服务器回传任何错误信息给客户端时会包含这个地址。设定此地址是很有用的，例如 ServerAdmin www-admin@foo.bar.com。因为使用者不一定知道他们是跟在哪台服务器打交道！

ServerRoot 指令

用法：ServerRoot 目录名称　　缺省：ServerRoot /usr/local/etc/httpd
用于：server config　　状态：内核

ServerRoot 指令设定服务器所在的目录。它一般会包含有子目录 conf/ 以及 logs/。其他配置文件的相对路径就是相对于这个路径。

BindAddress 指令

用法：BindAddress 服务器地址　　缺省：BindAddress *
用于：server config　　状态：内核

一台 UNIX 的 http 服务器可以监听到该服务机器每个 IP 地址的连线或者只注意该服务机器的一个地址。服务器地址可以是：

- *
- 一个 IP 地址
- 一个完整的网际网络域名

如果此值是 * 的话，那么该服务器将会监听每一个 IP 地址的连线，否则它将只监听指定的 IP 地址。这个选项可以用来作为另一种支持虚拟主机的方式取代 <VirtualHost> 节段。

ErrorLog 指令

用法：ErrorLog 文件名称　　缺省：ErrorLog logs/error_log
用于：server config, virtual host　　状态：内核

ErrorLog 指令用于设定错误记录文件名称，服务器将会把其遇到的每个错误记录到这个文件中去。如果文件名称不是以 / 开始的话那么它就会被假设为相对于 ServerRoot 的路径。如果是 ErrorLog /dev/null，则会有效地关掉错误记录。

TransferLog 指令

用法：TransferLog 文件—管线　　缺省：TransferLog logs/transfer_log
用于：server config, virtual host　　状态：基础
模块：mod_log_common

TransferLog 指令设定服务器记录连线请求的文件名称。文件—管线是下列情况之一：

- 一个文件名称　　一个相对于 ServerRoot 的文件名称
- '|' 跟随著一个指令　从标准输入接收参考记录信息的程序。注意如果虚拟主机从主要服务器继承 RefererLog 设定的话不会起动新的程序。

安全问题：如果在此使用程序，它将会以起动 httpd 的使用者身分执行。如果服务器由 root 起动那么此程序就是由 root 执行：所以要确定该程序的安全性。

PidFile 指令

用法：PidFile 文件名称　　缺省：PidFile logs/httpd.pid
用于：server config　　状态：内核

PidFile 指令设定服务器记录服务程序的程序号码所使用的文件。如果文件名称不是以 / 开始那么它会被假定是相对于 ServerRoot 目录。PidFile 只用在独立（standalone）模式。给服务器发送信号，可以关闭再重新打开错误记录与传输记录，以及重新读取配置文件。而这一功能是通过向列在 PidFile 里的程序号码传送 SIGHUP（kill -1）信号来现的。

ServerName 指令

用法：ServerName 完整的领域名称　用于：server config, virtual host
状态：内核

ServerName 指令设定服务器的主机名称，只有在建立重导 URL 的时候使用。如果没有指定，那么服务器会试图从其 IP 地址来决定。但这个方法可能不可靠，或者无法回传适当的主机名称。例如：ServerName www.wibble.com。如果正式（canonical）名称是 monster.wibble.com 它还能够使用上述的名称。

TimeOut 指令

用法：TimeOut 数字　缺省：TimeOut 1200
用于：server config　状态：内核

TimeOut 指令设定服务器接收一个请求以及完成一个请求最长的等待时间，以秒为单位。如果花费了比 TimeOut 更多的秒数来让客户端传送请求或接收回应，服务器将会中断该连线。因此 TimeOut 限制了一次可以传输的最大资料量。对于大文件，以及慢速网络传输时间可能会很久。

MinSpareServers 指令

用法：MinSpareServers 数字　缺省：MinSpareServers 5
用于：server config　状态：内核

MinSpareServers 这个指令设定最大闲置（idle）子服务程序数量。闲置子服务程序是目前没有处理要求的程序。如果有少于 MinSpareServers 的暂停程序，那么父程序会以最高每秒一个的速率建立新的子程序。只有在非常忙碌的站点上才有调整这个选项的需要。

MaxSpareServers 指令

用法：MaxSpareServers 数字　缺省：MaxSpareServers 10
用于：server config　状态：内核

MaxSpareServers 这个指令设定最大闲置(idle)子服务程序数量。如果有多于 MaxSpareServers 的暂停程序，那么父程序会终止超过此数量的子程序。只有在非常忙碌的站点上才有调整这个选项的需要。通常总是把这个参数设为非常大的数目。

StartServers 指令

用法：StartServers 数字　缺省：StartServers 5
用于：server config　状态：内核

StartServers 指令设定启动时建立的子服务程序数量。因为程序的数量是依据负载动态控制的，通常不必调整这个参数。

MaxClient 指令

用法：MaxClients 数字　缺省：MaxClients 150

用于：server config 状态：内核

MaxClients 指令设定所能支持的同时存取要求的最大数目，不会建立多于该值的子程序。

MaxRequestPerChild 指令

用法：MaxRequestsPerChild 数字 缺省：MaxRequestsPerChild 0
用于：server config 状态：内核

MaxRequestsPerChild 这个指令设定一个独立的子服务程序可以处理的请求数量。在处理 MaxRequestsPerChild 个请求之后，子程序将会被终止。如果 MaxRequestsPerChild 为 0 的话，那么该程序永远不会被终止。MaxRequestsPerChild 设为非 0 值做为限制有两个好处：通过内存使用量限制程序使用的内存数量；通过赋予程序有限的存留时间，可以在服务器负载降低时协助减少程序的数目。

<VirtualHost> 指令

用法：<VirtualHost 地址>...</VirtualHost> 用于：服务器配置
位于：内核部份

<VirtualHost> 以及 </VirtualHost> 用来把一组指令包装起来，这些指令将只会应用到某个特定的虚拟主机上。任何可用于虚拟主机的指令都可以使用。当服务器接收到某个特定主机上的文件请求时，它会使用包装在 <VirtualHost> 节段里的配置指令。地址可以是虚拟主机的 IP 地址或虚拟主机的 IP 地址的完整领域名称。

2. srm.conf

Red Hat 5.1 自带的文件 srm.conf：

```
# With this document, you define the name space that users see of your http
# server.  This file also defines server settings which affect how requests
# are serviced, and how results should be formatted.

# See the tutorials at http://www.apache.org/ for
# more information.

# Originally by Rob McCool; Adapted for Apache

# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.

DocumentRoot /home/httpd/html

# UserDir: The name of the directory which is appended onto a user's home
# directory if a ~user request is recieved.

UserDir public_html

# DirectoryIndex: Name of the file or files to use as a pre-written HTML
# directory index.  Separate multiple entries with spaces.

DirectoryIndex index.html index.htm index.shtml index.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 filenam
e
# extensions

AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*

AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .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 .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.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 /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^

# DefaultIcon is which icon to show for files which do not have an icon
# explicitly 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 filenames which directory indexing should ignore
# Format: IndexIgnore name1 name2...

IndexIgnore */.??* *~ *# */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 the type of from filename extensions.

DefaultType text/plain

# AddEncoding allows you to have certain browsers (Mosaic/X 2.1+) uncompress
# information on the fly. Note: Not all browsers support this.

AddEncoding x-compress Z
AddEncoding x-gzip gz

# AddLanguage allows you to specify the language of a document. You can
# then use content negotiation to give a browser a file in a language
# it can understand. Note that the suffix does not have to be the same
# as the language keyword --- those with documents in Polish (whose
# net-standard language code is pl) may wish to use "AddLanguage pl .po"
# to avoid the ambiguity with the common suffix for perl scripts.

AddLanguage en .en
AddLanguage fr .fr
AddLanguage de .de
AddLanguage da .da
AddLanguage el .el
AddLanguage it .it

# LanguagePriority allows you to give precedence to some languages
# in case of a tie during content negotiation.
# Just list the languages in decreasing order of preference.

LanguagePriority en fr de

# Redirect allows you to tell 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

# Aliases: Add here as many aliases as you need (with no limit). The format
is
# Alias fakename realname

# Note that if you include a trailing / on fakename then the server will
# require it to be present in the URL. So "/icons" isn't aliased in this
# example.

#Alias /icons/ /usr/local/etc/httpd/icons/
Alias /icons/ /home/httpd/icons/

# ScriptAlias: This controls which directories contain server scripts.
# Format: ScriptAlias fakename realname

ScriptAlias /cgi-bin/ /home/httpd/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 (see below)
# Format: AddHandler action-name ext1

# To use CGI scripts:
#AddHandler cgi-script .cgi

# To use server-parsed HTML files
AddType text/html .shtml
AddHandler server-parsed .shtml

# Uncomment the following line to enable Apache's send-asis HTTP file
# feature
#AddHandler send-as-is asis

# If you wish to use server-parsed imagemap files, use
AddHandler imap-file map

# To enable type maps, you might want to use
#AddHandler type-map var
# To enable the perl module (if you have it installed), uncomment the
# following section
#
#Alias /perl/ /home/httpsd/perl/
#<Location /perl>
#SetHandler perl-script
#PerlHandler Apache::Registry
#Options +ExecCGI
#</Location>

# To enable the php module (if you have it installed), use
#AddType application/x-httpd-php .phtml

# Action lets you define media types that will execute a script whenever
# a matching file is called. This eliminates the need for repeated URL
# pathnames for oft-used CGI file processors.
# Format: Action media/type /cgi-script/location
# Format: Action handler-name /cgi-script/location

# MetaDir: specifies the name of the directory in which Apache can find
# meta information files. These files contain additional HTTP headers
# to include when sending the document

#MetaDir .web

# MetaSuffix: specifies the file name suffix for the file containing the
# meta information.

#MetaSuffix .meta

# Customizable error response (Apache style)
#  these come in three flavors
```

```
#
#   1) plain text
#ErrorDocument 500 "The server made a boo boo.
#  n.b.  the (") marks it as text, it does not get output
#
#   2) local redirects
#ErrorDocument 404 /missing.html
#  to redirect to local url /missing.html
#ErrorDocument 404 /cgi-bin/missing_handler.pl
#  n.b. can redirect to a script or a document using server-side-includes.
#
#   3) external redirects
#ErrorDocument 402 http://some.other_server.com/subscription_info.html
#
```

DocumentRoot 指令

```
用法：DocumentRoot 目录/文件名
缺省：DocumentRoot /usr/local/etc/httpd/htdocs
用于：server config, virtual host      状态：内核
```

该指令设定 httpd 从哪个目录提供文件服务。除非符合 **Alias** 这样的指令，服务器把所要求的 URL 附加到文件根 (document root) 来组合存取文件的路径。例如：DocumentRoot /usr/web 。这样，对于 http://www.my.host.com/index.html 文件的存取便会参照到 /usr/web/index.html 。

UserDir 指令

```
用法：UserDir directory                缺省：UserDir public_html
用于：server config, virtual host      状态：基础      模块：mod_userdir
```

UserDir 这个指令设定使用者 home 目录里的一个真实目录，当接收到一个对使用者文件所发出的请求时会使用这个目录。 Directory 可以是 Disable ，用来关掉这项特色；或者是一个目录的名称。如果没有关掉，那么以 http://myserver/~UNIX-username 作为开始的一个 URL 请求将会被转换成以 home-dir/directory 作为开始的文件名称，其中 home-dir 是 UNIX-username 这个使用者的 home 目录。例如：UserDir public_html，那么一个对 http://myserver/~foo56/adir/file.html 的请求将传回 http://myserver/home/foo56/public_html/adir/file.html 文件。

DirectoryIndex 指令

```
用法：DirectoryIndex local-url local-url
用于：server config, virtual host, directory, .htaccess
需求：Indexes      状态：基础          模块：mod_dir
```

DirectoryIndex 这个指令设定当客户端通过指定没有以文件做结尾的目录名称请求该目录的索引时所要找寻的来源列表。Local-url 是在服务器器上相对于请求的目录的文件；它通常是目录里某个文件的名称。可以有好几个 URL ；服务器会回传它找到的第一个。如果这些来源都不存在，那么服务器将会自行产生一份该目录的列表。例如：DirectoryIndex index.html，然后，对于 http://myserver/docs/ 的请求若该来源设定的文件存在的话会回传 http://myserver/docs/index.html，如果不存在就回传该目录的列表。注意该文件并不需要是相对于该目录的，比如：DirectoryIndex index.html index.txt /cgi-bin/index.pl 。这会使得

/cgi-bin/index.pl 这个 CGI 指令文件在 index.html 以及 index.txt 在该目录下都不存在的情况下被起动。

FancyIndexing 指令

```
用法：FancyIndexing 布尔值
用于：server config, virtual host, directory, .htaccess
需求：Indexes    状态：基础    模块：mod_dir
```

FancyIndexing 指令设定目录的象征索引选项。布尔值可以是 on 或 off。参考 **IndexOptions** 指令。

AddIcon 指令

```
用法：AddIcon 图标 名称 名称 ...
用于：server config, virtual host, directory, .htaccess
需求：Indexes    状态：基础    模块：mod_dir
```

设定显示在文件名称之后的图标以便于标识。图标可以是某个图标的相对 URL 或是给非图形化浏览器使用的文字标签。对于目录可以使用 ^^DIRECTORY^^ 这个名称，^^BLANKICON^^ 可以用在空白行（使列表的格式正确），名称也可以是扩充文件名，替代字元表示式，部份或完整的文件名称。例如：

```
AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/dir.xbm ^^DIRECTORY^^
AddIcon /icons/backup.xbm *~
```

注意：尽可能使用 **AddIconByType**，而不是 AddIcon。

AddIconByEncoding 指令

```
用法：AddIconByEncoding icon mime-encoding mime-encoding ...
用于：server config, virtual host, directory, .htaccess
需求：Indexes    状态：基础    模块：mod_dir
```

设定显示在有 mime-encoding 的文件名称之后的图标以便于标识。图标可以是某个图标的相对 URL 或是给非图形化浏览器使用的文字标签。Mime-encoding 是个符合要求的编码内容的替代字元表示式。例如：

```
AddIconByEncoding /icons/compress.xxbm x-compress
```

AddIconByType 指令

```
用法：AddIconByType icon mime-type mime-type ...
用于：server config, virtual host, directory, .htaccess
需求：Indexes    状态：基础    模块：mod_dir
```

设定显示在 mime-type 的文件名称之后的图标以便于标识。图标可以是某个图标的相对 URL 或是给非图形化浏览器使用的文字标签。Mime-encoding 是个符合要求之 mime 型态的替代字元表示式。例如：

```
AddIconByType (TMG, /icons/image.xbm) image/*
```

DefaultIcon 指令

```
用法：DefaultIcon url
用于：server config, virtual host, directory, .htaccess
```

需求：Indexes 状态：基础 模块：mod_dir

DefaultIcon 指令设定当不知道指定的图标是什么时显示的图标。url 是该图标的相对 URL 。例如：DefaultIcon /icon/unknow.xbm 。

AddDescription 指令

用法：AddDescription 字串 文件 文件 ...
用于：server config, virtual host, directory, .htaccess
需求：Indexes 状态：基础 模块：mod_dir

设定某文件所要显示的描述，供象征索引（FancyIndexing） 使用。文件是指所要描述的文件的扩展名，部份文件名称，替代字元表示式或完整文件名称。字串以双引号（"）括起。例如：

```
AddDescription "The planet Mars" /web/pics/mars.gif
```

ReadmeName 指令

用法：ReadmeName 文件名称 ...
用于：server config, virtual host, directory, .htaccess
需求：Indexes 状态：基础 模块：mod_dir

ReadmeName 指令设定要附加到索引列表后面的文件的名称。文件名称是指要读入的文件，而且是相对于索引的目录。服务器首先会把他当作是 HTML 文件试图读入文件名称.html ，否则就当它是普通文本文件读入。例如：ReadmeName README。当产生/web 目录的索引时，服务器首先将会找寻 /web/README.html 这个 HTML 文件，若找到则将其读入，否则会读入 /web/README 这个普通文本文件（如果存在的话）。

HeaderName 指令

用法：HeaderName 文件名称
用于：server config, virtual host, directory, .htaccess
需求：Indexes 状态：基础 模块：mod_dir

HeaderName 指令设定插入索引列表顶部的文件名称。文件名称是要读入的文件之名称，而且相对于索引的目录。服务器首先会把他当作是 HTML 文件试图读入文件名称.html，否则就当它是普通文本文件读入。例如：HeaderName HEADER，当产生 /web 目录的索引时，服务器首先将会找寻 /web/HEADER.html 这个 HTML 文件，若找到就将其读入，否则会读入 /web/HEADER 这个普通文本文件（如果存在的话）。

IndexIgnore 指令

用法：IndexIgnore 文件 文件 ...
用于：server config, virtual host, directory, .htaccess
需求：Indexes 状态：基础 模块：mod_dir

IndexIgnore 指令把列出目录时要隐藏的文件加到列表中。文件是指所要忽略的文件的扩展名，部份文件名称，替代字元表示式或是完整文件名称。多个 IndexIgnore 指令会把文件继续加入列表中而不会取代前面的设定的忽略文件列表。缺省情况下，此列表包含“.”。例如：

```
IndexIgnore README .htaccess *~
```

AccessFileName 指令

用法：AccessFileName 文件名称 缺省：AccessFileName .htaccess
用于：server config, virtual host 状态：内核

把文件回传给客户端的时候，如果目录的存取控制文件没有有启动，服务器会在到达这份文件的路径中的每个目录里以此名称搜寻存取控制文件。例如：AccessFileName .acl，在回传 /usr/local/web/index.html 这份文件前，此服务器将会读取/.acl, /usr/.acl, /usr/local/.acl 以及 /usr/local/web/.acl 以取得指令，除非以如下的方式关闭它们：

```
<Directory />
AllowOverride None
</Directory>
```

DefaultType 指令

用法：DefaultType mime-型态 缺省：DefaultType text/html
用于：server config, virtual host, directory, .htacess
需求：FileInfo 状态：内核

有些时候服务器会被要求提供某份文件，该文件不能通过其 MIME 型态对映来决定型态。服务器必须被告知客户端文件所包含的型态（content-type），所以不知道型态的时候它便使用 DefaultType。例如：DefaultType image/gif 适于包含许多没有 .gif 扩展名的 gif 图形目录。

AddEncoding 指令

用法：AddEncoding mime-enc 扩展名 扩展名
用于：server config, virtual host, directory, .htacess
需求：FileInfo 状态：基础 模块：mod_mime

AddEncoding 指令以指定的编码型态把可能作为文件名称结尾的扩展名加入文件扩展名列表。Mime-enc 用在以该扩展名结尾的文件的 mime 编码。例如：

```
AddEncoding x-gzip gz
AddEncoding x-compress Z
```

这将会使以 .gz 结尾的文件被标记为使用 z-gzip 编码，以 .Z 结尾的文件被标记为使用 x-compress 编码。

AddLanguage 指令

用法：AddLanguage mime-lang 扩展名 扩展名
用于：server config, virtual host, directory, .htaccess
需求：FileInfo 状态：基础 模块：mod_mime

AddLanguage 指令以指定的语言把可能作为文件名称结尾的扩展名加入文件扩展名列表。Mime-lang 是以此扩展名作为名称结尾的文件的 mime 语言，这是在所有作为编码的扩展名移除之后再决定的。例如：

```
AddEncoding x-compress Z
AddLanguage en .en
AddLanguage fr .fr
```

那么 xxxx.ez.Z 将会被当作压缩过的英文文件。虽然内容的语言已经报告给客户端，浏览器不太可能使用此信息。AddLanguage 这个指令对内容协商（content negotiation）更有用，这样服务器可以参考客户端的语言回传文件合适的语言版本。

LanguagePriority 指令

用法：LanguagePriority mime-lang mime-lang
用于：server config, virtual host, directory, .htaccess
需求：FileInfo 状态：基础 模块：mod_mime

LanguagePriority 指令用于存在多语言版本，而客户端没有表示要参照何种语言的情况下各语言版本的优先权。mime-lang 列表按递减的顺序。例如：LanguagePriority en fr de，在请求 foo.html 时，若 foo.html.fr 以及 foo.html.de 都存在，但浏览器没有表示要参照的语言，则回传 foo.html.fr 。

Redirect 指令

用法：Redirect url-路径 url 用于：server config, virtual host
状态：基础 模块：mod_alias

Redirect 指令将旧的 URL 对应到新的 URL 上去。新的 URLs 会被回传给客户端以便配合新的地址再次进行访问。url-路径，任何以这个路径作为开始的文件请求都将回传一个错误并重导到以 url 作为开始的新的(%-encoded) url 上去。例如：Redirect /service http://foo2.bar.com/service ，如果客户端请求 http://myserver/service/ foo.txt 则会被告知应该去访问 http://foo2.bar.com/service/foo.txt。注意：无论配置文件里的次序如何，Redirect 指令优先于 Alias 以及 ScriptAlias 指令。

Alias 指令

用法：Alias url-路径 目录-文件名称 用于：server config, virtual host
状态：基础 模块：mod_alias

Alias 指令使文件可以存放在 DocumentRoot 之外的本地文件系统里。以“url-路径”开始的 URLs 将会被对映到以“目录-文件名称”开始的本地文件。例如：Alais /image /ftp/pub/image，对于 http://myserver/images/foo.gif 的这个请求会使得服务器回传文件 /ftp/pub/image/foo.gif 。

ScriptAlias 指令

用法：ScriptAlias url-路径 目录-文件名称 用于：server config, virtual host
状态：基础 模块：mod_alias

ScriptAlias 指令除了把目标目录标记为包含 CGI 指令文件以外其功能与 Alias 指令相同。以“url-路径”作为开始的 URLs 将会被对映到以“目录-文件名称”作为开始的指令文件。例如：ScriptAlias /cgi-bin/ /web/cgi-bin/ ，对于 http://myserver/ images/foo 的这个请求会使服务器去执行指令文件 /web/cgi-bin/foo 。

AddType 指令

用法：AddType mime-type 扩展名 扩展名
用于：server config, virtual host, directory, .htaccess

需求：FileInfo 状态：基础 模块：mod_mime

AddType 指令用以设定指定的内容型态可能作为文件名称结尾的扩展名。mime-type 是以该扩展名结尾的文件的 mime 型态。这是在所有作为编码以及语言的扩展名排除之后再决定的。例如：AddType image/gif GIF

新的 mime 型态建议使用 AddType 指令加入而不要修改 TypesConfig 文件。注意，与 NCSA httpd 不同，这个指令不能用来设定特殊文件的型态。

3. access.conf

Red Hat 5.1 自带的 access.conf 文件：

```
# access.conf: Global access configuration
# Online docs at http://www.apache.org/

# This file defines server settings which affect which types of services
# are allowed, and in what circumstances.

# Each directory to which Apache has access, can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).

# Originally by Rob McCool

# Be a little more paranoid
<Directory /home>
Options Indexes IncludesNOEXEC
AllowOverride None
</Directory>

# This should be changed to whatever you set DocumentRoot to.

<Directory /home/httpd/html>

# This may also be "None", "All", or any combination of "Indexes",
# "Includes", "FollowSymLinks", "ExecCGI", or "MultiViews".

# Note that "MultiViews" must be named *explicitly* --- "Options All"
# doesn't give it to you (or at least, not yet).

Options Indexes Includes ExecCGI FollowSymLinks

# This controls which options the .htaccess files in directories can
# override. Can also be "All", or any combination of "Options", "FileInfo",
# "AuthConfig", and "Limit"

AllowOverride None

# Controls who can get stuff from this server.

order allow,deny
allow from all

</Directory>

# /usr/local/etc/httpd/cgi-bin should be changed to whatever your
# ScriptAliased CGI directory exists, if you have that configured.
```

```
<Directory /home/httpd/cgi-bin>
AllowOverride None
Options ExecCGI
</Directory>

# Allow server status reports, with the URL of http://servername/server-statu
s
# Change the '.your_domain.com' to match your domain to enable.

#<Location /server-status>
#SetHandler server-status

#order deny,allow
#deny from all
#allow from .your_domain.com
#</Location>

# There have been reports of people trying to abuse an old bug from pre-1.1
# days.  This bug involved a CGI script distributed as a part of Apache.
# By uncommenting these lines you can redirect these attacks to a logging
# script on phf.apache.org.  Or, you can record them yourself, using the scri
pt
# support/phf_abuse_log.cgi.

<Location /cgi-bin/phf*>
deny from all
ErrorDocument 403 http://phf.apache.org/phf_abuse_log.cgi
</Location>

# You may place any other directories or locations you wish to have
# access information for after this one.
```

Options 指令

用法：Option 选项 选项 ...
用于：server config, virtual host, directory, .htaccess
需求：Options　　状态：内核

Options 指令控制某个特定目录所能使用的服务器选项。选项可以设为 NONE，在此情况下没有额外的选项会起动，或者可以是下列的一个或几个：

- All　　除了 MultiViews 以外的所有选项。
- ExecCGI CGI　　指令文件的执行权限。
- FollowSymLinks　　服务器将会跟照目录里的符号链结。
- IncludesNOEXEC　　服务器端包含（Server-side include）的权限。
- Indexes　　如果有个 URL 对映到所要求的目录，而且目录里面并没有目录索引（DirectoryIndex，例如 index.html）存在，那么服务器将会传回这个目录格式化后的列表。
- MultiViews　　允许 MultiViews 内容协商。
- SymLinksIfOwnerMatch　　只有在目标文件或目录与符号链结的拥有者相同时服务器才会去跟照符号链结。

如果多个选项可以应用到某个目录上，那么只有最底下的设定会真正应用；选项不会合并。例如：

```
<Directory /web/docs>
Options Indexes FollowSymLinks
</Directory>
<Directory /web/docs/spec>
Option MultiViews
</Directory>
```

这样只有 Mutilviews 会设定到 /web/docs/spec 目录上。

AllowOverride 指令

用法：AllowOverride override override ... 缺省：AllowOverride All
用于：directory 状态：内核

服务器找到 AccessFileName 指定的文件时需要知道该文件所宣告的哪些指令可以改变以前的存取信息。Override 可以设为 None，在此情况下服务器将不会读取该文件，而设为 All 的话服务器将会允许所有的指令，或是下列其中之一：

- AuthConfig 允许使用验证指令（AuthDBMGroupFile，AuthDBMUserFile，AuthGroupFile，AuthName，AuthType，AuthUserFile 以及 require）
- FileInfo 允许使用控制文件型态的指令（AddEncoding，AddLanguage,AddType, DefaultType 以及 LanguagePriority）
- Indexes 使用允许控制目录索引的指令（AddDescription，AddIcon，AddIconByEncoding，AddIconByType，DefaultIcon，DirectoryIndex，FancyIndexing，HeaderName，IndexIgnore，IndexOptions 以及 ReadmeName）
- Limit 允许使用控制存取主机的指令（allow, deny 以及 order）
- Options 允许使用控制特定目录特色的指令（Options 以及 XbitHack）

<Directory> 指令

用法：<Directory 目录> ... </Directory> 用于：server config, virtual host
状态：内核

<Directory> 以及 </Directory> 用来把一组指令包装起来，这些指令将只应用到所指明的目录及其子目录上。任何可以用于 directory 的指令都可以使用。“Directory”是到某目录的完整路径，或是替代字元串。在替代字元串中“?”代表任何单一字元，而“*”则代表任何顺序的一些字元。例如：

```
<Directory /usr/local/httpd/htdocs>
Option Indexes FollowSymLinks
</Directory>
```

如果有多个 directory 节区符合包含该文件的目录（或其父目录），那么 .htaccess 文件里的指令是以最短先符合的顺序加以应用的，例如：

```
<Directory />
AllowOverride None
</Directory>

<Directory /home/*>
AllowOverride FileInfo
</Directory>
```

存取 /home/web/dir/doc.html 这份文件的步骤是：

a) 应用 AllowOverride None 指令（关掉 .htaccess 文件）。

b) 应用 AllowOverride FileInfo 指令（目录 /home/web）。

c) 应用任何在 /home/web/.htaccess 里面的 FileInfo 指令。

节段 directory 主要用在 access.conf 文件里，但它们可以出现在任何配置文件里。<Directory> 指令不能嵌套使用，而且不能出现在 <Limit> 节段里。

20.3.3 代理服务器的设置

代理服务器（Proxy Server），一般是指防火墙上的应用程序，它把防火墙内的 Intranet 和防火墙外广阔的 Internet 连接起来。除了可以监控网络和记录传输信息外，它还可以提供企业级的文件缓存、复制和地址过滤服务。任何应用程序如需要与外界通讯必须首先与代理服务器交谈，以代理服务器为中介出入防火墙。

目前常用的一种代理服务器是带文件缓存（cache）的 HTTP 代理服务器。当一个客户程序被设置成使用局域网上的 HTTP 代理服务器时，它的 HTTP （或 FTP）请求将被发往代理服务器，而不是文件实际上的源地址。当代理服务器接到这一请求后，首先在自己的缓存区里寻找所要求的文件，如果缓存区里没有，则从文件的源地址处下载（并存入缓存区），再把结果发回客户程序。以后同样的请求将从缓存区得到应答，文件从代理服务器经局域网传到客户程序。每个文件只从广域网下载一次，避免了同一文件的重复传输，节省了网络带宽和客户程序下载文件的时间。缓存区会及时清除过时文件，并避免缓存哪些实际上是由程序产生的应答（如 CGI 程序的输出，对每次请求其结果都可能不同）。

一些常用的 HTTP 服务器，如 CERN HTTPD、Apache，可以被设置成带缓存的代理服务器。另有一些是被专门设计成代理服务器或与现存的 HTTP 服务器共同使用，可以支持更多的协议，并附加其他功能，如自动翻译等，常用的有 DeleGate 和 Squid 等。

使用 HTTP 代理服务器需要：安装、设置、运行 HTTP 代理服务器；设置客户程序。

1. 代理服务器的设置

下面以 Apache HTTP 服务器为例着重讲一下，安装、设置、运行 HTTP 代理服务器。Apache HTTP 服务器的详细介绍和安装见前面两小节，这里只介绍与代理服务器有关的内容。

Apache HTTP 服务器只在 1.1 或更高的版本中含有代理模块，它实现了对 FTP，（SSL）的 CONNECT，HTTP/0.9 和 HTTP/1.0 等协议请求的代理/缓存功能。在缺省的情况下，代理模块没有被编译进服务器的可执行文件 httpd，如想使用 Apache 的代理功能，必须在编译 httpd 前改变有关设置，加进代理模块。下面就简要介绍如何编译有代理功能的 Apache HTTP 服务器。

首先，解开 Apache HTTP 服务器源程序包安装在 /usr/local/etc 下，应产生 apache_x.x.x/目录。做符号连接：**ln -s apache_x.x.x httpd**。代理模块的源程序在 apache_x.x.x/src 的子目录 moudles/proxy 下。在 apache_x.x.x/src 目录下，拷贝文件 Configuration.tmpl 到 Configuration，再编辑 Configuration。把其中一行

```
# Module proxy_module   modules/proxy/libproxy.a
```

前面的注释符“#”去掉。对 Configuration 的其他改动依照 INSTALL 文件的说明进行。执行“**./Configure**”和“**make**”，如一切顺利，将产生含代理功能的 HTTP 服务器的

可执行文件 httpd。

其次，对 Apache HTTP 服务器配置文件 httpd.conf 做适当改动，增加与代理/缓存有关的指令，使 HTTP 服务器工作在带缓存的代理服务器状态。

httpd.conf 中与代理/缓存有关的 Apache 指令有：

ProxyRequests	ProxyRemote	ProxyPass	ProxyBlock
CacheRoot	CacheSize	CacheMaxExpire	CacheDefaultExpire
CacheDirLength	CacheGcInterval	CacheDirLevels	NoCache
CacheLastModifiedFactor			

下面分别加以介绍。

ProxyRequests 指令

```
用法：ProxyRequests on/off   缺省：ProxyRequests off
```

允许或禁止使用 Apache 的代理功能。注意设置 ProxyRequests 为“off”并没有使 ProxyPass 指令失效。

ProxyRemote 指令

```
用法：ProxyRemote <match> <remote-server>
```

定义此代理服务器的远程代理。<match> 可以是远程服务器支持的 URL 形式，表示对于该 URL，使用远程代理服务器；可以是 URL 的一部分，只要完整的 URL 中含有这一部分，就使用远程代理服务器；也可以是“*”，表示对所有的请求都使用远程代理服务器。<remote-server> 是远程服务器 URL 的一部分，定义为：

```
<remote-sever> = <protocol>://<hostname>[:port]
```

<protocol> 是用来与远程服务器通讯的协议；Apache 代理模块只支持“http”。例如：

```
ProxyRemote http://students.com/ http://mirror-students.com:8080
ProxyRemote * http://clever.com
ProxyRemote ftp http://ftpproxy.mydomain.com:8080
```

在最后一个例子中，代理服务器将把所有的 FTP 请求封装成一个 HTTP 代理请求，并把它传递给另一个可以处理它的代理服务器。

ProxyPass 指令

```
用法：ProxyPass <path> <url>
```

这条指令允许远程服务器被映射到本地服务器中；本地服务器这时不是通常意义的代理，而是看起来更象远程服务器的镜像。<patch> 是本地的一个虚拟路径；<url> 是远程服务器 URL 的一部分。假设本地服务器的地址为 http://web.org，那么

```
ProxyPass /mirror/fff http://fff.com
```

将把一个本地请求 http://web.org/mirror/fff/aaa 隐含地转换成一个到 http://fff.com/aaa 的代理请求。

ProxyBlock 指令

```
用法：ProxyBlock <word/host/domain list>
```

给出了一个由空格分开的词语、节点和域名的列表。含有列表中词语、节点或域名的 HTTP、HTTPS 和 FTP 文件请求被代理服务器阻断。代理模块在启动时将试图决定列表中可能的节点名的 IP 地址，并把它们缓存起来以备将来做符合检验。例如：

```
ProxyBlock joes_garage.com some_host.co.uk rocky.wotsamattau.edu
```

而 ProxyBlock * 将阻断到所有节点的连接。

CacheRoot 指令

用法：CacheRoot <directory>

设置存放缓冲文件的目录。注意这个目录必须是 httpd 服务器可写的。

CacheSize 指令

用法：CacheSize <size>　　缺省：CacheSize 5

设置缓存区的大小，以 Kb（1024 bytes）为单位。尽管实际占用的空间可能增长而超过这个上限，但垃圾收集机制将删除文件直到占用的空间等于或小于这个上限。

CacheMaxExpire 指令

用法：CacheMaxExpire <time>　　缺省：CacheMaxExpire 24

在没有检查原始服务器之前被缓存的 HTTP 文件将最多保存 <time> 小时。因此文件最多可能过期 <time> 小时。即使随文件提供了一个失效期，这个限制也将被执行。

CacheDefaultExpire 指令

用法：CacheDefaultExpire <time>　　缺省：CacheDefaultExpire 1

如果文件是通过一个不支持失效期的协议获取的，则使用 <time> 小时作为失效期。**CacheMaxExpire** 不能覆盖这个设置。

CacheLastModifiedFactor 指令

用法：CacheLastModifiedFactor <factor>　　缺省：CacheLastModifiedFactor 0.1

如果原始 HTTP 服务器没有提供文件失效期，依照如下公式估计失效期：

失效期 = 离最新一次修改的时间 * <factor>

例如，如果文件最新一次修改是 10 小时前，且 <factor> 是 0.1，则失效期将被设为 10 * 0.1 = 1 小时。如果失效期比 **CacheMaxExpire** 设置的时间长，则后者优先。

CacheGcInterval 指令

用法：CacheGcInterval <time>

每隔 <time> 小时检查缓存区，如果占用的空间已经超过 **CacheSize** 设置的上限就删除文件。

CacheDirLevels 指令

用法：CacheDirLevels <levels>　　缺省：CacheDirLevels 3

设置缓存区中子目录的层数。被缓存的数据将在 **CacheRoot** 下存储所设置的子目录层数。

CacheDirLength 指令

用法：CacheDirLength <length>　缺省：CacheDirLength 1

设置代理缓存子目录名的字母数。

NoCache 指令

用法：NoCache <word/host/domain list>

给出一个由空格分开的词语、节点和域名列表。符合列表中词语、节点和域名的 HTTP 和不用口令的 FTP 文件将不被代理服务器缓存。代理模块在启动时将试图决定列表中可能的节点名的 IP 地址，并把它们缓存起来以备将来做符合检验。例如：

```
NoCache joes_garage.com some_host.co.uk bullwinkle.wotsamattau.edu
```

而 NoCache * 将彻底禁止缓存。

下面就一些一般性设置问题做些说明：

（1）控制使用代理。可以通过通常的 <Directroy> 控制块来控制对代理服务器的使用。例如：

```
<Directory>
<Limit GET PUT POST DELETE CONNECT OPTIONS>
order deny, allow
demy from [不允许使用此代理服务器的节点名或IP地址]
allow from [允许使用此代理服务器的节点名或IP地址]
</Limit>
</Directory>
```

<Files> 块也可以用来控制对代理服务器的使用。

（2）为什么文件类型 xxx 不能用 FTP 下载？很可能是因为 xxx 类型的文件在代理服务器的 mime.types 配置文件中没有被定义为 application/octet-stream，可以加上这么一句：

```
application/octet-stream bin dms lha lzh exe class tgz taz
```

（3）为什么 Apache 使用代理模块时启动较慢？如果使用了 **ProxyBlock** 或 **NoCache** 指令，服务器在启动时将查找节点名的 IP 地址，并把它们缓存起来以备将来符合检验。查找节点名的 IP 地址将需要几秒或更长的时间。

设置好 `httpd.conf` 文件后，在启动代理服务器之前，一般应产生 `CacheRoot` 指定目录。注意这个目录的 owner 和 group 应该与 httpd.conf 中指定的服务器运行时的 owner 和 group 相同。

现在就可以按通常启动 httpd 的方法来启动代理服务器了。

2. 客户程序的设置

设置客户程序很简单，以 Netscape Communicator 为例，经过如下步骤即可完成设置。运行 Netscape，选择菜单选项 Edit -> Preferences…，Netscape 将弹出 Preferences 窗口；在左侧 Category 中，选择 Advanced -> Proxies；在 Proxies 的三个选项中，缺省是“Direct connection to the internet”，即不使用代理服务器；若要使用代理服务器，一般应选“Manual proxy configuration”，并按右边的“View…”，弹出“View Manual Proxy Configuration”窗口，添入相应的代理服务器节点名和端口号后 OK 即可。

第二十一章

中 文 环 境

Linux 象众多的软件产品一样，不是我们中国人的产品。但它已经作为最流行的免费 UNIX 操作系统进入中国。其推广和应用在很大程度上取决于其中文化。目前我国大陆和台湾都有自己的中文计划。希望更多的朋友为 Linux 的中国化做出贡献。是挑战，还是机遇？

大家知道中文在电脑上是由两个字节编码组成的。最常见的编码方式有大陆地区使用的 GB 编码和台湾地区的 BIG5 编码。而问题就出在这儿！许多程序没有考虑到输入的资料可能是 non-ASCII 码的问题。它们往往假设所要处理的资料都是 ASCII 码，更糟糕的是，当它遇到 non-ASCII 码时，常常假设它不存在，而将它的第八个位元截去！这就是所谓的 8-bit clean 问题。网络上的通讯程序也常常只能传输七位元的资料。早期的 sendmail 程序就是一个例子。sendmail 只能接送含七位元的信件，导致传送中文信件时，必须采用各种编码格式（如 uuencode, base64, QP 等），这就给收信者带来很大的麻烦。除了无法处理 non-ASCII 码资料的问题之外，应用程序无法辨识中文编码也是一大问题。也就是说，很多程序（即使能正确处理八位元的资料）都将一个中文字视为两个独立的字节组。在许多情况下这顶多是引起乱码，而在某些场合下某些中文字所含的特殊内码对某些应用程序具有特别的意义，这将导致严重的错误，或是死机。

21.1 中文字符集及编码

目前在中国大陆和香港台湾地区使用着不同的中文字符集标准。

21.1.1 GB 码

中国大陆最常用的基本中文字符集是 GB 2312-80。GB 指“国标”，即国家标准。这个中文（简体）字符集标准包括 7445 个字符，其中 6763 个是汉字。汉字分为两级，第一级汉字按音序排列，第二级汉字先按部首后按剩余笔画排列。其字符排列如下：

1 区：94个符号
2 区：72个数字
3 区：94个满宽度 GB 1988-89字符（对应于可打印的ASCII字符）
4 区：83个日文平假名
5 区：86个日文片假名
6 区：48个大写和小写希腊字母

7 区：66个大写和小写斯拉夫（俄语）字母
8 区：26个拼音和37个注音字符
9 区：76个制表符（09-04到09-79）
16区-55区：3755个汉字（一级汉字；最后一个是 55-89）
56区-87区：3008个汉字（二级汉字；最后一个是 87-94）

另外一个字符集 GB/T 12345-90 在字符的数量和安排上几乎等同于 GB 2312-80，但所有的简体字都用相应的繁体字替代了。字符集 GB 1988-89 对应于 ASCII 字符或 ISO 646，但现金符号（0x24）用中国的人民币符号取代了 ASCII 字符集中的美元符号。

中文字符集的编码一般是双字节码，对于每个字符集标准都可能有几种编码体系。对于 GB 2312-80 字符集，常用的编码有 EUC 和 HZ。EUC 是 Extended UNIX Code 的缩写，它是在 ISO 2022-1993 中定义的一种编码体系，用来处理大的字符集。它在各种 UNIX 系统上被广泛地用来对各种语言的字符集进行编码。EUC 包括四个代码集（code set），编号为 0 到 3。

表 20-1 EUC 在中国大陆字符集的实现

EUC 代码集	编码范围
代码集 0（ASCII 或 GB 1988-89）	0x21~0x7E
代码集 1（GB 2312-80）	0xA1A1~0xFEEE
代码集 2	未使用
代码集 3	未使用

经常把 GB 2312-80 字符集的 EUC 编码称为国标码或 GB 码。GB 码的第一个字节是 0xA1 ~ 0xFE，其中 0xA1 ~ 0xF7 对应 GB 2312-80 字符集的 87 个区，0xF8 ~ 0xFE 没有定义；第二个字节是 0xA1 ~ 0xFE，对应每个区最多 94 个字符。注意：GB 码的两个字节都是 8 位码（ >0x7F）。这一点对中文处理很有利。

21.1.2 HZ 码

在这里还要向大家介绍一下 HZ 编码体系。这种编码体系是解决有些电子邮件接收和处理 8 位码时乱码问题的方法之一。在 HZ 码中，GB 码的两个字节的最高位都被设置为 0，中文字符编码的第一个字节被限制在 0x21 ~ 0x77，第二个字节的范围是 0x21 ~ 0x7E；在中文字符串之间使用由两个可打印字符组成的转换序列隔开。对于单字节字符集（ASCII），转换序列为“~}”，其十六进制代码为 0x7E7D；对于双字节字符集（GB2312-80），转换序列为“~{”，其十六进制代码为 0x7E7B。在 HZ 码中，波浪号（0x7E）被解释为逃逸字符。在单字节字符模式下要使用波浪号时，必须连用两个，即“~~”表示“~”。

21.1.3 Big5 码

目前在台湾等地最为常用的中文字符集是大五（Big5）字符集。大五字符集由 94 个区组成，每个区有 157 个字符，分为两组，分别有 63 个和 94 个字符：

1 区：157个符号
2 区：157个符号
3 区：94个符号

4 区-38区：5401个汉字（一级汉字；最后一个是38-63）
41区-89区：7652个汉字（二级汉字；最后一个是89-116）

大五字符集的编码体系一般称为大五码或 Big5 码。其编码规则是：对于双字节字符，第一个字节的编码范围为 0xA1 ~ 0xFE，第二个字节的编码范围为 0x40 ~ 0x7E 和 0xA1~0xFE；对于单字节字符（ASCII）的编码范围为 0x21 ~ 0x7E。这样，大五码的第一个字节都是 8 位码，第二个字节中的第二组（0xA1~0xFE）是 8 位码，而第一组（0x40 ~ 0x7E）则为 7 位码，软件在处理时就要比 GB 码麻烦一些。

21.2　中文化方法

21.2.1　修改源代码

有源代码的软件中文化比较容易，程序员一般要做两件事：

（a）中文字符和 ASCII 字符的混合显示；

（b）中文输入。

原软件需要改动的地方一般有：

- 混合显示，使用新的函数取代原来的 XDrawString() 和 XDrawImageString()
- 窗口尺寸计算，使用新的函数取代原来的 XTextWidth() 和 XTextExtents()
- 编码探测，判断是否为中文编码或是中文编码的哪个字节
- 窗口事件的重处理，把功能键和输入键码分开，输入键码作汉字转换
- 汉字接收处理
- 处理字库，GC 等

X 窗口管理器一般提供源代码，所以大都采用这种汉化方法。

21.2.2　“包装”原理

通过“包装”（WRAP）方案，很多原本不支持中文显示和输入的 X Window 系统上的应用程序，即使没有源代码，只要经中文化“包装”后，不经任何修改，不必重新编译，就可显示和输入中文。其原理是利用 UNIX 系统运行动态连接程序时的 PRELOAD 机制，对 X Window 系统函数动态库中的某些函数进行了“包装”和替换，使之能够实现支持中文显示，并可与中文输入服务器连接实现中文输入。但这种“包装”有个原则，即应不影响对 ASCII 码的处理。

UNIX 系统上广泛使用着共享函数库。应用动态连接的程序并没有把要调用的函数做进程序中，只是在运行时才从共享函数库中调出所用函数。可用“**ldd**”命令查看程序与哪些共享库连接，如：

```
% ldd  /usr/X11R6/bin/xterm
    libXaw.so.6 = >/usr/X11R6/lib/libXaw.so.6 (0x4000a000)
    libXmu.so.6 = >/usr/X11R6/lib/libXmu.so.6 (0x40042000)
    libXt.so.6 = >/usr/X11R6/lib/libXt.so.6 (0x40054000)
    libSM.so.6 = >/usr/X11R6/lib/libSM.so.6 (0x40092000)
    libICE.so.6 = >/usr/X11R6/lib/libICE.so.6 (0x4009b000)
    libXext.so.6 = >/usr/X11R6/lib/libXext.so.6 (0x400b000)
    libX11.so.6 = >/usr/X11R6/lib/libX11.so.6 (0x400bb000)
```

```
        libc.so.5 => /lib/libc.so.6 (0x4014c000)
```

在 Linux 操作系统上，如果在当前 Shell 下定义环境变量 LD_PRELOAD ——共享目标文件的列表，则程序在执行时将先从 LD_PRELOAD 指定的共享目标文件中寻找程序用到的所有共享函数，如果没有找到，程序才去与之动态连接的共享库中寻找。注意，如果某个程序是 set-user-ID 或 set-group-ID 的，程序在执行时将只搜索那些在程序连接生成时 LD_RUN_PATH 变量或-R 选项指定的路径上的共享目标，其他共享目标一律被忽略。也就是说 LD_PRELOAD 变量将不起作用。

利用动态连接的 PRELOAD 机制，可以对 X Window 系统的某些函数进行“包装”，以便原来只是用来处理 ASCII 码的函数可以根据输入情况有区别地处理 ASCII 码和中文的编码，并把“包装”后的函数做成共享库，利用设置 LD_PRELOAD 变量使一些 X 应用程序调用“包装”后的 X 函数，而不是直接调用 X 函数库中的函数，这样就实现了不更改应用程序的源代码即能支持中文。

例如设置 LD_PRELOAD 变量如下：

```
% setenv LD_PRELOAD  /usr/local/lib/libwrap.so
```

在 libwrap.so 中有经过重新编写的 X 函数 XNextEvent（），它对与中文处理有关的事件进行了特别处理。xterm 程序运行时，将首先去 libwrap.so 文件中寻找 XNexEvent（）函数，而不是在没有 LD_PRELOAD 变量时直接去 libX11.so.6 中去寻找。

下面举例说明如何对 X 函数 XDrawString（）进行中文“包装”。XDrawString（）函数的定义是：

```
XDrawString (display, d, gc, x, y, string, length)
    Display * display;
    Drawable d;
    GC gc;
    int x, y;
    _Xconst char * string;
    int length;
```

其功能是：在可绘制的 d 上（可以是窗口，也可以是象素图），从位置（x，y）开始，将长度为 length 的字符串 string 用 gc 中指定的字库画出来，简单地说，就是在 X 窗口的某个位置画出 length 个字符。如果 string 中含有中文字符，XDrawString 只会把它们当做两个 ASCII 扩展字符画出来。

另有一个函数 XDrawString16（），其定义是

```
XDrawString16 (display, d, gc, x, y, string, length)
    Display * display;
    Drawable d;
    GC gc;
    int x, y;
    _Xconst XChar2b * string;
    int length;
```

其功能是：在可绘制的 d 上，从位置（x，y）开始，将长度为 length 的 16 位编码的字符串 string 用 gc 中指定的字库画出来。如果 string 是一个中文字符串，同时 gc 中指定了一种中文 X 字库，那么就可以用这个函数将中文字符串用指定的中文字库画在 X 窗口里。

“包装”后的 XDrawString 函数应该能够把一个既有 ASCII 字符又有中文字符的混合字符串分别用合适的 ASCII 字库和中文字库在 X 窗口的正确位置画出来。下面就是“包装”后的 XDrawString 函数结构：

```
XDrawString (display, d, gc, x, y, string, length)
{
    handle = dlopen("/usr/X11R6/libX11.so.6",RTLD_LAZY);
    realfunc = (int * )dlsym(handle, "XDrawString");
    将string分成ASCII字符和中文字符分开的若干段(str);
    for (每段str) {
      if (str是ASCII字符串) {
           设置gc中字库为ASCII字库;
           计算str长度len
           realfunc(display, d, gc, x, y, str, strlen(str));
            # 用libX11.so.6中的 XDrawString
         }
      else{ # str是中文字符串
           设置gc中字库为中文字库;
           计算str 长度len;
           XDrawString16(display, d, gc, x, y, str, strlen(str)/2);
        }
       x = x + len;
    }
}
```

这样，通过对 XDrawString 函数的包装，应用程序中调用 XDrawString 画字符串的语句都能正确地画出中文字符串了。类似地，还可以包装其他与字符串和文本输出相关的函数，使它们都能正确处理中文字符，从而使原来不支持中文显示的软件在中文化“包装”后可以显示中文。

而令 X Window 系统的软件支持中文输入是通过“包装”XNextEvent（）、XLookupString（）等函数实现的——在这些函数里加进与中文输入相关事件的处理功能，使客户程序能使用中文输入服务器。例如加进发送键盘事件到中文输入服务器，接收中文输入服务器返回的事件等语句。

使用中文化“包装”方法的软件实例有：台湾人 Weijr（weijr@magic.math.ntu.edu.tw）首先编写的 XA（Xcin Anywhere）；中国科学院高能物理研究所的于明俭编写的 Chinput 套件等。

XA（Xcin Anywhere）对 XLookupString（），XNextEvent（），XDrawString（），XDrawImageString（）等几个 X 函数进行了包装，使用的中文输入服务器是 xcin。XA 基本上可以在 Netscape，rxvt，xterm，color_xterm，cxterm，xedit，xjed，xpostit，xclipboard，xxgdb，xman 等软件上实现中文显示和输入。XA 可从 ftp://magic.math.ntu.edu.tw/pub/XA/ 目录下载。而由陈向阳先生维护的 GB 码版 XA 增加了对 XOpenDisplay，XTextExtents 等函数的包装，重新计算中西文字符串的长度，改进了在非等宽 ASCII 字库间中文的显示，使中西文在 Netscape 等软件中的显示连贯；修改了确定中文字库高度和宽度的规则，使中西文的对比更谐调；考虑了 X11R6 的 X 服务器和 SUN Solaris 的 OpenWin X 服务器对可缩放字库的处理不同，使“包装”在两个操作系统上都可有较好效果。GB 版的 XA 使用由方汉改编的支持 GB 码的 xcin 作为中文输入服务器。GB 码版的 XA 可从下列地址获取：ftp://ftp.ihep.ac.cn

/pub/chinese/system/。

XA 的使用很简单，用户可以在 X Window 启动文件里设置 LD_PRELOAD 变量指向 XA 的共享目标文件 wrap.so，这样所有在 X 窗口管理器里运行的程序都被中文化“包装”了。用户也可以用软件包提供的 Shell 脚本文件 xa 启动某一个应用程序，如：xa netscape，这样 LD_PRELOAD 只对 netscape 进程有效。

中文化套件 Chinput 中对 X 函数的“包装”进一步深化，增加了对多种事件的处理。它可支持 GB，Big5，日文 EUC，韩文 EUC 等多种编码，并可对编码进行实时切换；对半汉字显示和光标位置进行了修止，使移动光标或鼠标刷行时保持汉字输出完整；支持屏幕抓词自动翻译，即与套件中的 CDict 软件配合可以用鼠标刷词进行英汉对译。

Chinput 中文“包装”的使用方法与 XA 类似。可以设置 LD_PRELOAD 变量，也可以用 Shell 脚本 run 运行某个程序。Chinput 的详细情况请参阅中文输入一节。

21.2.3 常用的 X Window 中文化解决方案

目前常用的 X Window 中文化解决方案是：

中文化 X 窗口管理器 + X Window 的中文化“包装”软件+中文输入服务器

这种方案不需要对大量的 X 应用程序进行逐个的中文化，甚至不必知道应用程序的源代码。笔者认为这是一种较为快捷的折衷方案，类似于英文 Windows3.1+中文之星+汉化的应用程序。关于中文输入服务器请参阅中文输入一节。

可以使用如下的软件组合：

中文化的 fvwm95 + XA（Xcin Any where） + Xcin

中文化的 fvwm95 + xa（XA 的 GB 码版） + xcingb(Xcin 的 GB 码版)

中文化的 fvwm95 + Chinput 套件

21.3 X Window 的中文字库

21.3.1 常用字库

X Window 系统下的中文软件使用的中文字库大致有以下几类：

- BDF：Bitmap Distribution Format, X Window 系统字库的位图格式。
- HBF：Hanzi Bitmap Font，汉字位图字库，每个汉字的位图占同样大的空间。
- PCF：Portable Compiled Format, X Window 系统显示字库的标准格式，可跨平台使用。
- TTF：True Type Font，多在 Windows 下使用的轮廓线字库。

这四种字库文件的扩展名通常是 .bdf、.hbf、.pcf、.ttf。但要注意 .hbf 文件只是字库描述文件，汉字位图则存放在另外的文件里。

如果想使用中文字库作为 X Window 系统显示字库，必须把 HBF、BDF、TTF 等格式的字库转换成 PCF 格式。但有些中文软件在中文显示和打印时可直接处理使用 HBF、BDF、TTF 等格式的字库。

四种字库文件相互转换的程序有：

- bdftopcf：X Window 系统自带
- hbftobdf 和 bdftohbf：ftp：//ftp.ifcss.org/pub/software/fonts/utils /hbf.tar.gz
- getbdf：从 X 服务器中取出某显示字库，形成 BDF 文件
- ttf2bdf：在 freetype 软件包里（ftp：//ftp.physiol.med.tu-muenchen.de/pub/freetype/）或 ftp：//sunsite.une.edu/pub/Linux/X11/fonts/

由于目前很多软件（如 Netscape）使用两个字节的最高位都为 0 的编码去索引 PCF 字库，为使从 HBF 文件转换的 BDF 及 PCF 字库能为更多的软件作为显示字库使用，在对使用标准 EUC 编码的 GB 码 HBF 文件（即中文字符的两个字节的最高位都为 1）进行转换时可把各个 HBF 文件中的编码范围作修改。如 cc48s.hbf 中的下列几行

```
DEFAULT_CHAR 0xA1A1
......
HBF_BYTE_2_RANGE 0xA1-0xFE
......
HBF_CODE_RANGE 0xA1A1-0xA9FE cc48.sym 0
HBF_CODE_RANGE 0xB0A1-0xD7FE cc48s.1 0
HBD_CODE_RANGE 0xD8A1-0xF7FE cc48s.2 0
```

可修改成

```
DEFAULT_CHAR 0x2121
......
HBF_BYTE_2_RANGE 0x21-0x7E
......
HBF_CODE_RANGE 0x2121-0x297E cc48.sym 0
HBF_CODE_RANGE 0x3021-0x577E cc48s.1 0
HBF_CODE_RANGE 0x5821-0x777E CC48s.2 0
```

并把 CHARSET_ENCODING "1" 改为 CHARSET_ENCODING "0"。

另外，HBF 文件多用字库的简称或别名作为字库名（HBF 文件中的“FONT”项）。但很多软件需要用到中文字库完整地符合 XLFD 规则的字库名，如 Netscape 根据字库名的“CHARSET_REGISTRY”是否含“gb2312”来搜索 X 服务器上的中文字库。另外用字库的简称或别名命名的字库不能用作可缩放（scalable）字库。所以有必要把 HBF 文件中的“FONT”项作修改。如对 cc48s.hbf，可将下行

```
FONT cc48s
```

改为

```
FONT -CC-Song-medium-r-normal-jiantizi-48-480-75-75-c-480-GB2312.198C-0
```

另外，还可以把不同点阵和不同字体的中文 HBF 字库的“FOUNDRY”项改成统一的名称，如“cclib”或其他，以纳入一套显示字库的体系。这样很多软件用同样的索引就可以找到所有点阵所有字体的中文 X 字库。

做完中文 PCF 字库后，可给所有的中文 X 字库加上别名，以便使用。特别是有些程序（如 CXterm）只能识别短的字库名。

21.3.2 中文字库的安装

依照上一小节的说明整理出一套中文 X 字库后，可把 PCF 文件用 gzip 压缩成 .pcf.gz（或.pcf.Z）文件，以节省磁盘空间。Linux 上的 X11R6 版本的 X 服务器可以使用压缩字库。Linux 上 X Window 系统的字库存放在 /usr/X11R6/lib/X11/fonts 目录的各子目录下，如 100dpi/，75dpi/，Typel/，misc 等。可以把中文 X 字库的 PCF 文件（*.pcf.gz）拷贝到 misc/ 目录下，然后执行如下命令：

```
cd /usr/X11R6/lib/x11/fonts/misc
/usr/X11R6/bin/mkfontdir
```

“**mkfontdir**”将搜索 misc/ 目录下的所有字库并更新字库目录文件 fonts.dir。fonts.dir 文件中现在已增加了所有中文字库的列表，类似下面的几行：

```
ccs16.pcf.gz -cclib-song-medium-r-normal--16-160-75-75-c-160-gb2312.1980-0
ccs16f.pcf.gz -cclibf-song-medium-r-normal--16-160-75-75-c-160-gb12345.1990-0
ccs24.pcf.gz -cclib-song-medium-r-normal--24-240-75-75-c-240-gb2312.1980-0
cch24.pcf.gz -hei-medium-r-ncrmal--24-240-75-75-c-240-gb2312.1980-0
cck24.pcf.gz -cclib-kai-medium-r-normal--24-240-75-75-c-240-gb2312.1980-0
ccfs24.pcf.gz -cclib-fangsong-medium-r-normal--24-240-75-75-c-240-gb2312.1980-0
ccs24f.pcf.gz -cclibf-song-medium-r-ncrmal--24-240-75-75-c-240-gb12345.1990-0
cc48s.pcf.gz -cclib-song-medium-r-normal--48-480-75-75-c-480-gb2312.1980-0
cc48h.pcf.gz -hei-medium-r-normal--48-480-75-75-c-480-gb2312.1980-0
cc48k.pcf.gz -kai-medium-r-normal--48-480-75-75-c-480-gb2312.1980-0
cc48fs.pcf.gz -cclib-fangsong-medium-r-normal--48-480-75-75-c-480-gb2312.1980-0
```

可以根据这几行增加中文字库的别名，即把字符串“.pcf.gz”去掉，得到：

```
ccs16 -cclib-song-medium-r-normal--16-160-75-75-c-160-gb2312.1980-0
ccs16f -cclibf-song-medium-r-normal--16-160-75-75-c-160-gb12345.1990-0
ccs24 -cclib-song-medium-r-normal--24-240-75-75-c-240-gb2312.1980-0
cch24 -hei-medium-r-normal--24-240-75-75-c-240-gb2312.1980-0
cck24 -cclib-kai-medium-r-normal--24-240-75-75-c-240-gb2312.1980-0
ccfs24 -cclib-fangsong-medium-r-normal--24-240-75-75-c-240-gb2312.1980-0
ccs24f -cclibf-song-medium-r-normal--24-240-75-75-c-240-gb12345.1990-0
cc48s -cclib-song-medium-r-normal--48-480-75-75-c-480-gb2312.1980-0
cc48h -hei-medium-r-normal--48-480-75-75-c-480-gb2312.1980-0
cc48k -kai-medium-r-normal--48-480-75-75-c-480-gb2312.1980-0
cc48fs -cclib-fangsong-medium-r-normal--48-480-75-75-c-480-gb2312.1980-0
```

把上列字库别名加进 misc/ 目录下的 fonts/alias 文件中。中文字库的安装即告完成。可以用“**xset**”命令让 X 服务器立即重读所有字库目录，更新 X 服务器中存放的字库信息：

```
% xset fp rehash
```

现在就可以使用新安装的中文 X 字库了。下次启动 X 服务器的时候，无须再使用 **xset** 命令，因为 /usr/X11R6/lib/X11/fonts/misc 在 X 服务服务器缺省的字库搜索路径上，X 服务器在启动时已搜索所有在缺省的字库搜索路径上的目录并建立了所有字库的信息。

如果希望把中文 X 字库放到一个单独的地方，可以在 /usr/X11R6/lib/X11/fonts 目录下创建一个目录如 Chinese/。把中文 X 字库的 PCF 文件（*.pcf.gz）拷贝到 Chinese/ 目录下，用“**mkfontdir**”命令创建 fonts.dir 文件，把中文字库的别名存放在 fonts.alias 文件中拷贝到 Chinese/ 下。然后用

命令：

```
% xset fp + /usr/X11R6/lib/X11/fonts/Chinese/
```

把 Chinese/ 目录加到当前字库搜索路径上并更新 X 服务器中存放的字库信息，就可以用新安装的中文 X 字库了。但是 Chinese/ 目录尚不在 X 服务器的缺省字库搜索路径上。为把 Chinese/ 加到 X 服务器的缺省字库搜索路径上，使得每次启动 X 服务器不必手动执行“**xset**”命令，可以修改 /etc/XP86Config 文件，搜寻定义 FontPath 的语句，把 Chinese/ 目录加进 FontPath 变量：

```
FontPath "/usr/X11R6/lib/X11/fonts/Chinese/"
```

如果一个无特权的用户想建立自己的 X 字库目录，如$ HOME/xfonts，可依上面的说明拷贝字库文件到该目录中，创建 fonts.dir 和 fonts.alias 文件，执行：

```
% xset fp + $ HOME/xfonts
```

并把这条命令加进 $ HOME/.xsession 文件，则以后这个用户再使用 X 服务器的时候，自己的字库目录 $ HOME/xfonts 会被自动加进当前字库搜索路径上。

21.3.3 可缩放字库

一个符合 XLFD 规则的 X Window 系统字库名共有 14 个域（field），如

```
-cclib-song-medium-r-normal-jiantizi-24-240-75-75-c-240-gb2312.1980-0
```

其中第 7, 8, 9, 10, 12 五个域的意义分别是：

```
PIXEL_SIZE POINT_SIZE RESOLUTION_X RESOLUTION_Y AVERAGE_WIDTH
```

这五个数值中只有三个是相互独立的，另外两个可以根据这三个计算得到。一个 X 显示字库的名字（而不是别名）如果符合 XLFD 规则，则在被装入 X 服务器的时候就成为可缩放字库。可以指定上述五个域中的任意三个，另外两个用“0”或“*”来代替，由 X 服务器来决定装入 X 服务器的显示字体的宽度和高度。如在程序中用下面的名字请求 X 服务器装入中文字库：

```
-cclib-song-medium-r-normal-jiantizi-*-260-75-75-c-*-gb2312.1980-0
```

将得到近似 26x26 点阵的简体宋体字。在 Liunx 上，由于 X11R6 版本的 X 服务器对显示字体高度和高度的计算只取决于 PLXEL_SIZE 和 AVERAGE_WIDTH，所以在请求装入可缩放字库的时候可以只给出这两个值，其他三个值用“0”或“*”来代替。如：

```
-cclib-song-medium-r-normal-jiantizi-26-*-*-*-c-260-gb2312.1980-0
```

将装入 26x26 的简体宋体字库。

由原字型通过缩放得到的字型显示效果通常比原字型粗糙。

21.3.4 中文 X 字库的共享

建立 X 字库服务器可以把本机上的 X 字库提供给网络上其他计算机使用。对于中文 X 字库，建立 X 字库服务器有时是必要的，因为中文 X 字库文件一般都比较大，使用 X 字库服务器可避免联网的每台机器上都保存一套字库，占用很多存储空间。而且通

常由一个人维护更新中文 X 字库会更方便有效。

首先，创建字库服务器配置文件，如 fs7100.conf，其主要内容如下：

```
port = 7100          # X字库服务器占据端口号
client-limit = 10    # 此字库服务器最多允许客户连接数
clone-self = on      # 当一个字库服务器达到最多连接数时，启动新服务器
catalogue = /usr/X11R6/lib/X11/fonts/Typel,
            /usr/X11R6/lib/X11/fonts/Speedo,
            /usr/X11R6/lib/X11/fonts/misc,
            /usr/X11R6/lib/X11/fonts/75dpi,
            /usr/X11R6/lib/X11/fonts/100dpi,
            /usr/X11R6/lib/X11/fonts/Chinese
                          #字库目录
default-point-size = 120    # 缺省点数 x 10
default-resolutions = 100, 100, 75, 75  # 缺省分辨率100×100和75×75
```

然后启动 X 字库服务器：

```
% xfs  -cf  fs7100.conf &
```

在客户机上用“**xset fp + tcp/*server_name:port***”命令把 X 字库服务器加到当前字库搜索路径上，如：

```
% xset fp + tcp/lark:7100
```

21.4 中文输入

Linux 并不是为中国人设计的，因而也没有全面地考虑中文输入问题。就目前的状况来看，中文输入的解决大多采用附加套件的方法，而没有将其纳入内核的支持。而这些软件套件又大多采用客户机/服务器的模式工作。服务器为客户机提供输入服务。常用的中文输入服务器有 Xcin 和 Chinput 等。

21.4.1 Xcin＋crxvt

Xcin 是 X Window Chinese Input 的缩写，是在 X Window 系统模式下运行的中文输入系统，其客户机是 crxvt。Xcin 利用 X Window 系统的客户机/服务器模式工作，系统上只要启动一个 Xcin，便可为许多 crxvt 模拟终端提供输入服务，因而占用系统资源较少。Xcin 目前由台湾的居士先生（thhsieh@twclx.phys.ntu.edu.tw）负责维护，用户可从下面的地址获得 Xcin 的源程序包：ftp://linux.cis.netu.edu.tw/packages /chinese/xcin/。

Xcin 的安装遵循一般的 GNU 软件包安装的规则。以 xcin-2.3.02 为例，步骤如下：

先取得 xcin-2.3.02.tar.gz 文件，并在某个目录下解开。

```
tar xzvf xcin-2.3.02.tar.gz
....
cd xcin-2.3.02
./configure    (请依照显示的信息修改安装选项)
make
make install
```

中国科学院高能研究所的方汉先生将 Xcin+crxvt 改编成可支持 GB 码的版本：

```
ftp://ftp.ihep.ac.cn/pub/chinese/system/xcingb-2.2tar.gz
```

经过 X Window 系统的中文化包装（WRAP）后，Xcin 可为 X Window 系统的很多应用软件提供中文输入服务。

21.4.2 Chinput 套件

Chinput 套件是目前为止功能较为强大的中文集成套件。它包括输入工具、屏幕抓词、自动汉化、中文 EZWGL。Chinput 由中国科学院高能物理研究所计算机中心的于明俭（yumj@sun.ihep.ac.cn）编制，可以从下列地址取得 Chinput：ftp://ftp.ihep.ac.cn/pub/chinese/packages/Chinput-1.3.tra.gz。该软件采用客户间通讯（Inter-Client Communication）方式实现键盘输入的汉字转换。用户键入的字符输送到本服务器，服务器负责根据对应的输入方法进行汉字转换。

同时该软件具有良好的用户界面，采用三维效果的无边框窗口，窗口不接受 FocusIn（除非有意设置，即使已是聚集窗口，用户仍然不受边界影响），并且永远位于窗口堆栈（stacking order）的最上方（与 Windows 系统上的汉字外挂平台外观效果类似）。软件支持两种显示模式：单行和双行。单行模式所占空间较少，适合嵌入其他窗口中。软件支持 16 点阵和 24 点阵字型号，字体可以由用户任意指定。输入窗口大小可在一定范围内变化。窗口的尺寸变化可以通过客户程序按 F3 或 SHIFT + F3 键完成。另外，还可以通过资源文件($ HOME/Chinput.ad)或 API Configuration 来改变窗口的属性。软件支持字体浏览与输入，用户按下钮中的三角形按钮，便弹出与窗口为一体的字体（X 字库）浏览窗口并且可以把字符输出到聚焦窗口。其主要作用是帮助用户快速输入特殊汉字符号。客户程序可以把服务器嵌入到任何位置，从而作为自己的子窗口。嵌入后窗口不能再移动，而只能靠客户程序设备其位置。

Chinput 的输入方法基于 Cxterm，使用方法与 Cxterm 相同（参见中文终端一节）。

该软件中包含一个按钮辅助工具条，它启动后探测并启动服务器，显示目前的编码和输入法。在第三和第四个按钮上按鼠标键便可弹出菜单，设置服务器的语言编码和输入方法。工具条的显示对服务器的单行显示模式非常有用，因为单行显示时，服务器上并不显示编码和输入方法。工具条与服务器是相对独立的，不使用工具条时服务器照常工作。在服务器的使用过程中，如果客户程序改变了它的状态（输入方法或编码），服务器会通知工具条相应的信息以改变工具条的显示。工具条和服务器的关闭都不影响对方工作，工具条中还有有关服务器和工具条本身的使用说明，和一个时钟。除此之外，用户还可以在资源文件 Chinput.ad 中设置工具条的缺省位置（位于屏幕的哪一角），时钟的显示模式（模拟式或数字式）和工具菜单中所使用的外部命令（软件）等等。

工具菜单中的中文转换（Chinese Convert）软件需要以下外部程序：

- hc-30：GB 与 BIG5 转换
- HZ-2.0：HZ 与 GB 转换
- gb2jis.tra.gz：GB 转换成 JIS
- jis2gb.tar.gz：JIS 转换成 GB
- gb2ps.2.02.tar.gz：GB 转换成 PS

工具菜单中的日文转换（Japanese Convert）软件需要以下外部程序：

- jconv.c：Shift-JIS，EUC，NewJIS，OldJIS，NECJIS 之间的转换

软件设计直接使用 Xlib，处理速度较快，以 X11，Xt，Xm 等为基础的软件都可以很方便地使用该服务器。

该软件还提供了如下 API：

- HZclientInit（）：初始化客户软件
- HZqueryServer（）：获取服务器的有关信息，如协议，版本，编码形式，目前状态等
- HZconfigServer（）：改变服务器的设置或状态。如编码，面板颜色，输入方法，窗口锁定窗口嵌入等
- HZsendKey（）：输送键盘输入到服务器
- HZprocInput（）：处理服务器回送或自身回送

使用该 API 对现有软件的在输入方面的汉化相当容易，只须把原来的键盘处理改为对客户信息（Client Message）的处理即可。

该软件使用的中文 X 字库存放在 ftp://ftp.ihep.ac.cn/pub/chinese/fonts/pcf/ 目录。

21.4.3 其他问题

1. 新增输入法

目前常见的输入法表格有两种格式：tit 及 cin。这两种都是纯文字格式（换句话说可以直接用文本编辑器来查看）。但各种中文系统为了加快搜索速度，多半提供工具程序将纯文本转换为特殊的二进制文件。如果需要安装某种输入法，必须取得其 tit 或 cin 表格，或是转换后的格式。

下面以在 Xcin 上新增呒虾米输入法为例进行说明：

呒虾米输入法的输入法表格可在 ftp://ftp.cis.nctu.edu.tw/UNIX/Chinese/Boshiamy/ 上找到。其他一些输入法 tit 文件也可从 ftp://ftp.ifcss.org/pub/software/x-win/cxterm/dict/ 上找到。

利用 xcin 的工具程序 cin2tab 将 cin 表格转换为 tab 文件：

```
% cin2tab boshiamy.cin
```

将产生 boshiamy.tab 及 boshiamy.tab.rev 两个文件。将它们放到 xcin 的目录中。启动 xcin：

```
% xcin -in9 boshiamy.tab
```

然后用 CTRL + ALT + 9 即可使用呒虾米输入法。

2. 只能显示不能输入时的一些设置

在使用汉化的文本编辑软软件时，可能会有只能显示中文，却无法接受中文输入的现象。这时你可以试着自己修改两个地方，使 Linux 系统可以接受中文输入。首先必须要

在你使用的 Shell 起始文件中增加 locale 的设定（关于 locale 详情请看 locale mini-HOWTO）。

另外还要在自己的主目录中（$ Home ）的 .inputrc 文件（若无此文件，请自行建立）增加与输入有关的设定。下面是 Shell 起始文件和 .inputrc 文件的相关设定，可参考使用：

Bash Shell：请在 /etc/profile 增加下面的内容：

```
stty cs8 -istrip
stty pass8
export LANG=C
export LC_CTYPE=iso-8859-1
```

Tcsh Shell：请在 /etc/csh.login 或 /etc/csh.cshrc 增加设定：

```
stty cs8 -istrip
stty pass8
setenv LANG C
setenv LC_CTYPE iso-8859-1
```

$HOME/.inputrc 文件增加设定：

```
set convert-meta off
set output-meta on
```

21.5 中文编辑

可用于中文编辑的软件有 LaTeX + CJK、Emacs、jvim、ChinesePower、ChiTex、六书等。本节向你介绍 LaTeX + CJK 和 Emacs。

21.5.1 LaTeX + CJK

TeX/LaTeX 是一套非常好的文字编辑软体，因其强大的排版能力和优秀的输出品质早已为广大的 Linux 用户所喜爱和采用。CJK 是一个 LaTeX 的宏套件（macro package），可使用户在 TeX 文件中使用 CJK（Chinese/Japanese/Korean）的文字编码。

安装 CJK 之前必须先安装好一种 TeX/LaTeX 套件，其中 teTeX 是较好的一种。目前许多的 Linux 发行套件都已包含了 teTeX/LaTeX。如果没有的话，可以自行安装，请参考 teTeX HOWTO 。

CJK 可以从 ftp://ftp.ifcss.org/pub/software/tex/ 处下载。

CJK 支持 GB 码和 Big5 码，可以根据需要决定安装对某一种或两种编码的支持。CJK 可使用 HBF 和 TTF 字库。可以从 ftp://ftp.ifcss.org/pub/software/fonts/gb/hbf 目录下取来 48x48 点阵 GB 码四种字体（宋体、黑体、楷体和仿宋体）HBF 字库，需要的文件有：

```
cc48s.1 cc48s.2 cc48s.hbf
cc48h.1 cc48h.2 cc48h.hbf
cc48k.1 cc48k.2 cc48k.hbf
cc48fs.1 cc48fs.2 cc48fs.hbf
cc48.sym
```

CJK 的安装较为复杂，所以必须谨慎行事。下面仅就 CJK GB 码支持的安装过程进行说明，Big5 码支持的安装过程请参考 CJK 软件包的说明文件。

(1) 首先必须知道 teTeX/LaTeX 套件的安装目录。teTeX 套件缺省的安装目录是 /usr/local/teTeX。可以用命令“**which latex**”来确定一下。假设 teTeX 安装在 /usr/local/teTeX 目录下，teTeX 定义了一个内部变量 $TEXMF 指向 /usr/local/teTeX/texmf 目录。这是一个常用的变量，为了方便可以在安装过程中用命令：“**setenv TEXMF /usr/local/teTeX/texmf**”定义。注意 LaTeX 不必定义这个变量。

(2) 将 CJK-x.x.x.src.tar.gz 在某个临时目录下解开，产生 CJK/ 目录。CJK/x_x.x/doc 目录下有关于 CJK 安装和使用的重要文档，务必仔细阅读，特别是 INSTALL 文件和 teTex/ 子目录下的文件。

(3) 进入 CJK/x_x.x 目录，将 texinput 子目录移至 $TEXMF/tex/latex 下，并改名为 CJK。

(4) 创建 $TEXMF/fonts/hbf/chinese 目录，并将中文 HBF 字库文件拷贝到这个目录下。

(5) 进入 CJK/x_x.x/utils/hbf2gf/ 目录，用 gcc 编译 hbf2gf 程序。将可执行文件拷贝到 /usr/local/teTeX/bin 目录下。

(6) 根据 cfg/ 目录下的 gsfs14.cfg 文件修改和创建 cc48s.cfg，cc48h.cfg，cc48k.cfg 和 cc48fs.cfg 四个文件。下面是 cc48s.cfg 文件的一个例子：

```
hbf_header          $TEXMF/fonts/hbf/chinese/cc48s.hbf
mag_x               1
threshold           128
comment             GB SongTi
design_size         12
target_size         12
y_offset            -13
nmb_files           -1
output_name         cc48s
checksum            123456789
dpi_x               300
pk_files            no
tfm_files           yes
long_extension      no
coding              codingscheme GuoBiao encoded TeX text
pk_directory        $TEXMF/fonts/pk/modeless/chinese/cc48s/
tfm_directory       $TEXMF/fonts/tfm/chinese/cc48s/
```

其中 hbf_header 项要与中文 GB 码 HBF 文件存放的地点一致。**latex** 和 **divps** 命令（见后）执行时会将分别产生的 TFM 和 PK 型字库存放在 tfm_directroy 和 pk_directroy 目录里。这两个目录是自动产生的，不需要手动建立。

(7) 创建 $TEXMF/hbf2gf/ 目录，将 cc48s.cfg，cc48h.cfg，cc48k.cfg 和 cc48fs.cfg 四个文件拷贝到该目录下。

(8) 编辑 $TEXMF/fontname/special.map 文件，在文件末尾增加四行：

```
cc48s           chinese         cc48s
cc48fs          chinese         cc48fs
cc48k           chinese         cc48k
cc48h           chinese         cc48h
```

并将

```
cc  public  concrete
```

移至文件的最后，以免成为搜索中文字库名 cc48* 的障碍。

(9) 依照 CJK/x_x.x/doc/teTeX/ 目录中的 *.diff 文件，对 /usr/local/teTeX/bin 目录下的相应文件进行修改。建议不要使用 patch 指令，因为可能有版本不符的问题。

(10) 在 $TEXMF/tex/latex/CJK/GB 目录下根据 c10fs.fd 的例子编制 c10song.fd，c10fs.ld，c10hei.fd 和 c10kai.fd 文件。可将 c10fs.fd 中的“gsfs14”字符串更改为“cc48fs”以生成新的 c10fs.fd文件，再对新的 c10fs.fd进行修改（字符串“c10fs.fd”，“fs”，“cc48fs”）得到 c10song.fd，c10hei.fd 和 c10kai.fd。

(11) 把 /usr/local/teTeX/bin 加到环境变量 PATH 的最前端，然后运行

```
# texconfig  rehash
# texconfig  hyphen
```

更新 TeX 输入文件数据库 $TEXMF/lx-R 文件。

至此 GB 码的 LaTeX 安装完毕。现在可以用 /CJK/x_x.x/examples/GB.tex 来作测试：

```
% latex  GB.tex
```

latex 会根据 GB.tex 使用了哪些汉字来产生一些 TFM 文件，这些 TFM 文件一旦产生就被放入 cc48s.cfg 文件中指定的 tfm_directory 目录中，以便下一次用到同一汉字时使用。

latex 完成后，用 **dvips** 命令产生 PostScript 文件：

```
% dvips  GB
```

同样 **dvips** 会根据 GB.tex 使用了哪些汉字来产生一些 PK 文件，这些 PK 文件一旦产生就被放入 cc48s.cfg 文件中指定的 pk_directory 目录中，以便下一次用到同一汉字时使用。

如果 **dvips** 顺利完成，将产生 GB.ps，可以用 **gs** 命令读 GB.ps，也可以用 **xdvi** 命令来读 GB.dvi。

下面是一个 CJK 文件的例子：

```
    \documentclass[12pt]{article}
    \usepackage{CJK}
    \begin{document}
    \begin{CJK*}{GB}{kai}
这是一个例子（楷体），This is a example
    \CJKfamily{fs}
这是一个例子（仿宋），This is a example
    \end{CJK*}
    \end{document}
```

中文 CJK TeX 文件与一般 LaTeX 主要不同之处在于：

(1) 所谓 LaTeX 的 preamble （\documentclass 至 \begin{document}）区域中，必须有 **\usepackage{CJK}** 这个命令。

(2) 所有的汉字必须放在 \begin{CJK*}{GB}{...} 和 \end{CJK*}之间。

(3) 换字型用 **\CJKfamily** 命令。

注意，LaTeX 对西文的处理不因使用 CJK 扩展而改变。

21.5.2 Emacs

Emacs 是 GNU 的创始人 Richard Stallman 开发的一种“可扩展的，可定制的，实时显示的编辑器和计算环境”。它功能强大，是一个完整的计算支持环境。MULE 是 GNU Emacs 的多语种增强（MultiLingual Enhancement）。MULE 的文本缓冲区可以同时包含多种语言的字符：中文、日文、阿拉伯语等。MULE 还为每种语言提供了输入法。Emacs 从 20.1 版本开始，将 MULE 作为自己的一部分包含了进来。

Emacs 的编译安装过程遵循 GNU 的软件安装规则。在安装时可以不改变任何东西。Emacs 的缺省安装目录是 /usr/local/share/emacs，可执行文件是 /usr/local/bin/emacs。

为使 Emacs 在启动时就自动设置成中文环境，可以在自己的主目录下创建一个名为 .emacs 的文件。文件 .emacs 的内容为 Lisp 语句：

```
(setup-chinese-gb-environment)
(global-set-key [f6] 'toggle-input-method)
```

第一句设置中文 GB 环境，第二句将 F6 键定义为激活(或取消)缺省中文输入法的热键。

在主目录下的 .Xdefaults 文件后附上一句：

```
emacs.font:-*-fixed-medium-r-normal-*-16-*-*-*-*-*-*-*
```

它定义了与中文字库配合使用的 ASCII 字库。注意只能使用上面的长字库名，否则中文字库无法自动加载。

各种输入法定义在 Emacs 的安装目录下的 ./20.x/leim/leim-list.el 文件中。缺省的中文输入法定义在 ./20.x/lisp/language/china-util.el 文件中。可以在自己的主目录下的 .emacs 文件后附加一条语句定义别的输入法为缺省输入法，如

```
(setq default-input-method 'chinese-tonepy')
```

将缺省的输入法定义为带调拼音。

在使用 Emacs 的过程中，可以依次键入“CTRL + x”、“RET”、“CTRL + \”，来改变当前的输入法（“RET”表示 Return 键）。

21.6 中文打印

21.6.1 cnprint，ps2cps，gb2ps

这三种工具软件的工作原理是将文件转换为可以打印中文的 PostScript 格式。因此

要保证打印机能够打印 PostScript 文件。如果打印机不直接支持 PostScript，可以安装 ghostscript，请参考 Printing HOWTO。

1. cnprint

cnprint 将中文文件转换为 PostScript 文件以供打印。使用上和标准的打印指令一样。它支持 GB，HZ，BIG5，CNS，JIS，EUC，Shift-JIS，KSC，UTF8 和 UTF7 等多种编码，支持竖向排版，支持页面分栏，可设定字体大小，可实现 GB、HZ、Big5 间代码转换编码。在 ftp://ftp.ifcss.org/pub/software/UNIX/print 上可找到源程序文件 cnprint280.tar.gz。cnprint280.tar.gz 只包含了五个文件：

```
cnprint.i          cnprint.cmd        cnprint280.README
cnprint.c          cnprint.help
```

用下列方法编译：

```
gcc cnprint.c -o cnprint
mv cnprint /usr/local/bin
mv cnprint.1 /usr/local/man/man1
```

cnprint 使用 HBF 字库，因而要保证安装了完全的 HBF 字库。例如，若想使用简体仿宋字库 ccfs24.hbf，需将 ccfs24.hbf、 cclib.n24 和 ccsym.24 三个文件放在某一目录下，比如 /usr/local/lib/chinese/HBF/。然后在环境变量中指定 HBF 字型的完整目录：

```
export HBFPATH="/usr/local/lib/chinese/HBF/"
```

另外文件 cnprint.cmd 中包含了 cnprint 的一些预置值，可以加以修改，使之指向 HBF 字库。然后将它放到 $HBFPATH 中：

```
cp cnprint.cmd $HBFPATH
```

现在即可使用“**cnprint -w *FILENAME***”将中文文件转换为 PostScript 文件。详细的用法请参考 man cnprint。

2. ps2cps

ps2cps 的功能是将原本无法以中文输出的 PostScript 文件转换成可以以中文输出。例如 Netscape 在打印时是先将文件转成 PostScript，但其输出的 PostScript 却不包含中文字型，这使得原来是中文的部分变成乱码。这个程序可以读入 PostScript 文件，将其乱码的部分改成中文，则其输出结果即可在任何可以打印 PostScript 文件的打印机上打印。

该软件位于 ftp://linux.cis.nctu.edu.tw/packages/chinese/misc/ps2cps-0.1.tgz。解开后视需要修改 Makefile：

```
BINPATH    : 可执行文件 (ps2cps) 的安装路径
PS2CPSPATH: PS2CPS 的源程序路径
PS2CPSRC   : PS2CPS 源程序名称
```

再用 **make all install** 安装即可。

注意该软件同 cnprint 一样需要安装好 HBF 中文字库。并要修改 ps2cpsrc 文件：

```
HBF_PATH:      定义 HBF 字库所在目录
HBF_NAME:      定义 HBF 字库名称 ( .hbf , 不含路径 )
CH_WORD_SHIFT: 定义中文字体位移。
```

其中最后一项用来调整中文字体的位置。由于有些中文字体与英文字体可能不在同一水平线上，故可以设此变量做上下调整。其值为 -1.0 至 +1.0 之间。

ps2cps 的用法是：

```
ps2cps INPUT.ps > OUTPUT.ps
```

3. gb2ps

gb2ps 是另一种可以打印 GB 与 HZ 编码的工具程序。其源程序 gb2ps.2.02.tar.gz 可以从 ftp://ftp.ifcss.org/pub/software/UNIX/print 下找到。

字库：csong24.ccf、ckai24.ccf、cfang24.ccf 和 chei24.ccf 可从 ftp://ftp.ifcss.org /pub/software/fonts/gb/misc/ 下载。将这些字库安装在某个目录下，例如 /usr/local/lib/chinese/CFONT 。

在编译 gb2ps 之前先更改 **Makefile** 的设定

```
CFONT=/usr/local/lib/chinese/CFONT/
COVERPAGE=/usr/local/lib/chinese/lib/cover.ps
```

然后执行：

```
make
cp gb2ps /usr/local/bin
```

21.6.2 中文(GB) PostScript 字库

目前 Linux 上的打印系统通常是用一些软件（如前面介绍的 cnprint 等）把非 PostScript 文件(或不支持中文打印的 PostScript 文件)转换成支持中文打印的 PostScript 文件（对 PostScript 文件则不作改变）。但由于没有中文 PostScript 字库体系，现有的可形成 PostScript 文件的中文软件（如 chpower 和中文 LaTeX（CJK））大都是把用到的汉字字型以位图或轮廓形式全部存储在 PostScript 文件中，而不是象西文那样，使用外部的 PostScript 字库体系。这就导致 PostScript 文件体积太大，且打印效果欠佳。在台湾，已经有一些从 Big5 码的 TrueType 字库转换成 Big5 码的 PostScript 字库的软件，如 nut2cjk，但还没有建立用中文 PostScript 字库编写中文 PostScript 文件的规则体系。中国科学院高能物理所的陈向阳先生分析了目前各种中文软件对中文字库的处理方法，并参考了日文文字处理软件，总结出一套中文(GB) PostScript 字库，初步建立了构造和使用中文(GB) PostScript 字库的基本原则。这套中文(GB) PostScript 字库软件包存放在 ftp://ftp.ihep.ac.cn/pub/chinese /parkages/ 下面。

中文(GB) PostScript 字库目前包含四种字体，即宋体、黑体、楷体、仿宋体。对每种字体，有 94 个中文 Type1 PostScript 字库文件，以 PFB(Printer Font Binary)的格式存储，它们分别对应于 GB2312-80 编码的 94 个区（0xA1-0xFE）：

```
GB-Songal -- GB-Songfe          :   gbsongal.pfb -- gbsongfe.pfb
GB-Heial  -- GB-Heife           :   gbheial.pfb  -- gbheife.pfb
```

```
GB-Kaia1  -- GB-Kaife            :   gbKaia1.pfb  -- gbKaife.pfb
GB-FangSongal -- GB-FangSongfe   :   gbfsal.pfb   -- gbtsfe.pfb
```

其中左侧是字库名，右侧是对应的字库文件。

在上述 Type1 中文 PostScript 字库的基础上，定义了四种 Type0（复合型）中文（GB）基本字库：

```
GB-Song      (字库文件GB-Song.ps   )
GB-Hei       (字库文件GB-Hei.ps    )
GB-Kai       (字库文件GB-Kai.ps    )
GB-FangSong  (字库文件GB-FangSong.ps)
```

它们的 FmapType 是 2，即每次从要显示的字符串中读出两个字节，从第一个字节（即汉字的高 8 位）确定子字库，用第二个字节作为索引从子字库中取出汉字的轮廓字型。

在中文(GB)基本字库和西文 Type1 型 PostScript 字库的基础上，定义了四种 Type0 中文(GB)正体字库：

```
GB-Song-Regular GB-Song + Times-Roman          (字库文件GB-Song-Regular-ps)
GB-Hei-Regular  GB-Hei  + Times-Bold           (字库文件GB-Hei-Regular.ps)
GB-Kai-Regular  GB-Kai  + Helvetica            (字库文件GB-Hai-Regular.ps)
GB-FangSong-Regular GB-FangSong  +Helvetica (字库文件GB-FangSong-Regular.ps
)
```

它们的 FmapType 是 4，即从要显示的字符串中读出一个字节，根据其最高位是“0”还是“1”确定是西方字符还是中文字符，从而决定查找西方 PostScript 字库还是再取一个字节查找中文(GB)基本字库。

同上，定义了四种 Type0 中文 (GB)斜体字库。

```
GB-Song-Italic  GB-Song + Times-Italic       (字库文件GB-Song-Italic.ps)
GB-Hei-Italic   GB-Hei  + Times-BoldItalic   (字库文件GB-Hei-Italic.ps)
GB-Kai-Italic   GB-Kai  + Helvetica-Oblique (字库文件GB-Kai-Italic.ps)
GB-FangSong-Italic  GB-FangSong  +  Helvetica-Oblique
                                             (字库文件GB-FangSong-Italic.ps
)
```

四种 Type0 中文（GB）固定宽度字库（为使中文字符的宽度是西方字符的二倍，将西文字符的宽度缩减六分之一）：

```
GB-Song-Fixed    GB-Song      +  Courier       (字库文件GB-Song-Fixed.ps)
GB-Hei-Fixed     GB-Hei       +  Courier-Bold  (字库文件GB-Hei-Fixed.ps)
GB-Kai-Fixed     GB-Kai       +  Courier       (字库文件GB-Kai-Fixed.ps)
GB-FangSong-Fixed  GB-FangSong  +  Courier   (字库文件GB-FangSong-Fixed.ps
)
```

四种 Type0 中文（GB）固定宽度斜体字库（为使中文字符的宽度是西方字符的二倍，将西文字符的宽度缩减六分之一）：

```
GB-Song-FixedItalic GB-Song      +  Courier-Oblique
                              (字库文件GB-Song-Fixed.ps)
GB-Hei-FixedItalic  GB-Hei       +  Courier-BoldOblique
                              (字库文件GB-Hei-Fixed.ps)
GB-Kai-FixedItalic  GB-Kai       +  Courier-Oblique
                              (字库文件GB-Kai-Fixed.ps)
```

```
GB-FangSong-FixedItalic  GB-FangSong+ Courier-Oblique
                          (字库文件GB-FangSong-Fixed.ps)
```

为与西方 Times 和 Courier 系列字库对应，通过别名和 Type0 字库文件定义了以下八种字库（最后两个字库中 GB-Kai 的高度拉长了十分之一）：

```
GB-Times-Roman           (GB-Song-Regular的别名)
GB-Times-Italic          (GB-Song-Italic的别名)
GB-Times-Bold            (GB-Hei-Regular的别名)
GB-Times-BoldItalic      (GB Hei-Italic的别名)
GB-Courier               (GB-FangSong-Fixed的别名)
GB-Courier-Oblique       (GB-FangSong-FixedItalic的别名)
GB-Courier-Bold          GB-Kai + Courier-Bold
                         (字库文件GB-Courier-Bold.ps)
GB-Courier-BoldOblique   GB-Kai + Courier-BoldOblique
                         (字库文件GB-Courier-BoldOblique.ps)
```

为支持西文 ASCII 字符和中文（GB）字符的混合使用，应当使用上述定义的中文（GB）正体，斜体，固定宽度，固定宽度斜体 PostScript 字库，而不要直接使用中文（GB）基本字库 GB-Song，GB-Hei，GB-Kai 和 GB-FangSong，因为中文（GB）基本字库没有引用西方 PostScript 字库。

为使 Ghostscript 能显示中文 PostScript 文件，必须把中文(GB)PostScript 字库加进 Ghostscript 的字库目录，并在 FontMap 文件中增加相应的定义。但无须修改 Ghostscript（或其他显示 PostScript 文件的软件）的任何源代码。

安装中文（GB）PostScript 字库并设置 Ghostscript 的方法如下：

（1）首先要确定 Ghostscript 相关文件（初始化文件，字库等）安装在哪个目录下。

（2）将软件包 psfonts-gb-x.x.tar.gz 打开在某个临时目录下。将 ./fonts 目录下的所有字库文件（GB-*.ps，gb*.pfb）拷贝到 Ghostscript 的字库目录下。

（3）将 Fontmap.gb 文件附加到 Ghostscript 原有的 Fontmap 文件后，如

```
cat Fontmap.gb >> /usr/local/share/ghostscript/5.10/Fontmap
```

如果用户没有 root 权限，则可使用个人目录（如 $ HOME/gsfonts）存放这些文件。把 ./fonts 目录下所有字库文件(GB-*.ps)和所有 gb*.pfb 文件拷贝到$ HOME/gsfonts 目录中，把 Fontmap.gb 也拷贝到$ HOME/gsfonts 目录改成 Fontmap 。设置环境变量 GS_LIB 指向 $ HOME/gsfonts ：

sh 或 bash：

```
GS_LIB = $ HOME/gsfonts
exprot GS_LIB
```

csh 或 tcsh:

```
setenv GS_LIB $ HOME/gsfonts
```

（4）把 ./bin 目录下几个文件 ns2cps，txt2cps，jy2cps 拷贝到在用户的命令搜索路径变量 PATH 上的任何目录，如拷贝到 /usr/local/bin 下。

中文（GB）PostScript 文件的编写规则基本等同于西方 PostScript 文件。但由于西方 PostScript 字库基本上都是 Type1 字库，而中文 PostScrip 字库无论如何也不能只使用一个 Type1 字库，同西文字库类比直接引用的应该是 Type0 字库，所以编写中文 PostScript

文件的规则与西文 PostScript 文件略有不同。如在很多软件产生的 PostScript 文件中，经常会有类似下面修改西文字库的 Encoding 项的语句：

```
/F0
   /Times-Roman findfont
   dup length dict begin
      {1 index /FID ne {def} {pop pop} ifelse} forall
      /Encoding isolatin1encoding def
   currentdict end
definefont pop
```

其中“/Encoding isolatin1encoding def”一句把 Times-Roman 字库的 Encoding 项更改为“isolatin1encoding”并重新定义了字库。“isolatin1encoding”是西文 Type1 字库的一种标准 Encoding。但如果对 GB-Times-Roman 进行这样的操作将会导致 invalidfont 的错误。因为 GB-Times-Roman 是 Type0 字库，其 Encoding 项与 Type1 字库的 Encoding 项有本质上不同的意义。所以在中文 PostScript 文件中不允许出现以上这种类型的语句。除了上面这种类型的语句外，中文（GB）PostScript 文件同西方 PostScript 文件的编写规则基本相同。

中文（GB）PostScript 字库无条件地允许 ASCII 字符和汉字的混合使用。如果以前使用 Times-Roman 字库，现在简单地用 GB-Times-Roman 代替即可。

Shell 脚本文件 ns2cps 可以对 Netscape 存储的 PostScript 文件进行修改产生中文 PostScript 文件。如果用户的打印机队列使用 Ghostscript 作为输入过滤器，现在就可以打印经过 ns2cps 转换后的中文 PostScript 文件。如果用户想直接从 Netscape 中打印，可以打开 Netscape 的“File-->Print...”菜单，设置“Print Command”项为“ns2cps -|lpr”就可以从 Netscape 中直接打印所有页面。

Shell 脚本文件 txt2cps 可将普通中文文本文件转换成中文（GB）PostScript 文件，其用法是

```
txt2cps [-fn chinese_font] text_file
```

txt2cps 缺省是使用固定宽度的宋体字库 GB-Song-Fixed，但用户可以用“_fn”选项来指定使用其他字库，如

```
txt2cps -fn GB-Kai-Fixed report.tx t> report.cps
```

Shell 脚本文件 jy2cps 可以利用 Acrobat Reaber 将中文 PDF 文件转换成中文（GB）PostScript 文件。其用法是：

```
jp2cps pdf_file [ps_file]
```

缺省的输出文件名是将输入文件的扩展名由 .pdf 改为 .cps。

21.7 中文终端

CXterm 是 X Window 的中文模拟终端，可能是最古老也是最常用的中文显示/输入环境。它提供各种中文内码模式，包含 BIG5，HZ，GB 等等。因而本节着重介绍 CXterm。

21.7.1 安装 CXterm

CXterm 的最新版本是 cxterm5.0.p3.tar.gz（5.0 版）。这个压缩文件里已经包含了 CXterm 与中文字型。该文件可以从 ftp://ftp.ifcss.org/pub/software/x-win/cxterm/ 下载，或从 ftp://ftp.Red Hat.com/pub/contrib/hurricane /i386/ 下载 RPM 包装格式的 cxterm-color-5.0p3-1.i386.rpm，cxterm-color-big5-5.0p3-1.i386.rpm，cxterm-color-gb-5.0p3-1.i386.rpm。

解开压缩文件：

```
tar -xvzf cxterm5.0.p3.tar.gz
```

这将会产生目录 cxterm-5.0，然后进入该目录执行 config.sh：

```
% cd cxterm-5.0
% ./config.sh
```

如果想让系统上所有的使用者都能使用 CXterm，必须使用 root 的权限来执行 "**./config.sh**"。接下来按照提示完成安装：

```
--- BASIC MENU ---
0. Read COPYRIGHT Notice
1. Compile, Install, and Configure 'CXTERM 5.0' in One Step
--- OPTION MENU ---
2. Compile cxterm (not to install)
3. Install cxterm (after successful compilation in 2)
4. Install additional Chinese font(s) for your X window
5. Configure your account for using cxterm (after installation in 3)
x. Exit

Please choose (0/1/2/3/4/5/x) :
```

如果你想让一切自动完成，请选 1。然后，输入安装的目录，比如 /usr/local/chinese/bin。如果 1 安装失败，请按 2—3—5 的顺序安装，以查出问题出在何处。在这个压缩文件里还附有两个中文字库，选 1 和 3 都会自动安装这些字库。也可选 4 安装其他字库。装完之后，还需将 cxterm 与 CXterm 放在搜索目录之中。

```
export PATH=$PATH:/usr/local/chinese/bin
```

在 ftp://ftp.ifcss.org/pub/software/x-win/cxterm 下还有 CXterm 的 color patch。使用此修补文件可使 CXterm 显示各种颜色。假设 CXterm 的原始文件放在 /tmp/cxterm-5.0：

```
cp cxterm-5.0.p3-color.patch.gz /tmp
gzip -d cxterm-5.0.p3-color.patch.gz
patch < cxterm-5.0.p3-color.patch
cd cxterm-5.0
./config.sh
```

21.7.2 CXterm 的使用

安装程序产生了一个 Shell 脚本 CXterm。CXterm 文件中设置了正确运行可执行文件 cxterm 所必需的 X 资源并调用 cxterm 运行中文模拟终端。所以应执行脚本文件

CXterm，而不是 cxterm，否则 cxterm 将无法正确设定 X 资源，继而无法输入或显示中文。

CXterm 的缺省资源存放在文件 CXterm.ad 中。可以改变 CXterm.ad 设置，也可以在命令行重新定义。标准的 X Toolkit 命令行参数和 xterm 的命令行参数同样适用于 CXterm。CXterm 常用的命令行参数有：

- -fn <英文字库> 指定显示用英文字库。
- -fh <中文字库> 指定显示用中文字库，其字符宽度应为英文字符的两倍，高度相同。
- -fhb <中文字库> 指定显示用黑体中文字库，其宽度和高度应与“-fh”指定的标准体中文字库一致。
- -hm <输入法> 指定 cxterm 启动时的初始输入法。缺省是“ASCII”。
- -hz <编码> 指定 cxterm 的编码体系，GB 或 BIG5。缺省是 GB。
- -GB 表示 cxterm 使用 GB 码，等同于“-hz GB”。
- -BIG5 表示 cxterm 使用 BIG5 码，等同于“-hz BIG5”。
- -hls <行间距> 指定 CXterm 窗口文本区相邻两行的间距（以象素为单位）。

如，要使用 GB 编码，请使用命令：

```
CXterm -gb
```

在 CXterm 窗口的底部有一个输入区，用以显示中文信息。CXterm 为 GB 码提供了无调拼音、带调拼音、缩写拼音、五笔字型、区位码和内码等等输入方法，可用 F1 到 F11 或 SHIFT + F1 到 SHIFT + F11 激活切换。F1 可在当前输入法和 ASCII 码状态之间切换。

第二十二章

基于 XLIB 的应用程序开发

Linux 是由众多电脑爱好者协同开发的系统。在了解了 Linux 的基本知识后，你也许希望加入到开发者的行列中。本章将通过构造一个简单的 X 窗口程序，讲解 Linux 上的 X 应用程序的开发原理。你可将这个例子作为模板来构造更加复杂的应用程序。本章要求读者具有 C 语言的基本知识。

由于诸如 Motif 等高级的开发工具包的出现，现在开发一个比较完善的 X 应用程序已不是那么困难。由于篇幅的限制，本书并不打算介绍 Motif 等工具包的应用，而是想介绍一些更为基础的东西——使用 Xlib 编程。众所周知，Motif 等高级图形界面工具包是建立在 Xlib 的基础之上的，用 Xlib 编写 X 窗口程序是 Motif 编程的基础，而且在 Motif 应用程序中有时仍经常需要使用 Xlib 的很多函数和功能。

22.1 基础知识

22.1.1 头文件

本章使用 C 语言，在程序的开头需要包含一些头文件。基于 X 窗口编程，除了要包含 C 语言的头文件之外，还要包含 X 特有的一些头文件。<X11/Xlib.h>，<X11/Xutil.h> 和 <X11/Xos.h> 是几乎所有 X 程序都需要包含的头文件。<X11/Xlib.h> 和 <X11/Xutil.h> 包含了一些主要的 Xlib 函数、数据结构、宏以及常量的定义，其中 <X11/Xlib.h> 中又依次包含了许多其他的头文件，如 <X11/X.h> 等。第三个头文件 <X11/Xos.h> 通过包含编译过程中依赖于操作系统的一些文件，使得 X 程序具有可移植性。除了这三个头文件之外，根据程序中所调用的 Xlib 函数或其他函数，还需要增加相应的头文件，如 <X11/Xatom.h> 等。

22.1.2 变量

在 X 程序中，除了 C 或 C++ 中标准变量说明外，还要使用一些与 X 有关的变量类型，主要有 Window、Display、Pixmap、XSizeHints、XEvent、GC、XFontStruct 等等。下面简略介绍一下：

- Window：一个整数标识符（ID），用来标识一个 X 窗口，由 XCreateWindow() 或 XCreateSimpleWindow() 等函数产生并返回。

- Display：一个结构，包含了 X 服务器与屏幕的信息。只有在调用 XOpenDisplay() 连接服务器后，才给该结构赋值。
- Pixmap：象素图，同 Window 一样是一个整数标识符，指向内存中的一定的图形数据，Pixmap 可以以图象方式显示在屏幕上。
- XSizeHints：也是一个结构，用于向窗口管理程序提供有关顶层应用窗口大小等信息。
- XEvent：一个储存有关事件信息的联合，根据事件的类型，可解释为许多单个结构中的一种，比如 XButtonEvent 等。
- GC：Graphic Context 的缩写，是图形上下文标识符，其中包含绘图的若干信息。请参考以后的内容。
- XFontStuct：包括字体信息的一个结构。

22.1.3 服务器资源

服务器资源是 X 服务器管理的一个信息集。资源类型包括颜色、字体、图形上下文、象素等，程序在使用这些资源前需将它们加载或创建。由于服务器内存是有限的，如果在程序中频繁加载新的资源，势必使应用程序的效率大大降低。因此，对于一些程序中常用的资源，一般应在主程序产生和加载，在程序其他部分中直接使用，以达到资源共享。对于一些特殊的资源，可以在需要时随时加载，以满足程序中的特殊需求，但在使用完以后，应立即释放。在本章的例子程序中，采用了子程序 load_font() 和 getGC() 来加载字体和产生用于把字符和图形画在窗口内的图形上下文。

22.1.4 图形上下文

在 X 上绘图是通过一些图元来实现的，X 支持的图元主要有画点、画线、画文字、图形填充等等。但是这些图元并不能完全决定大多数图形的细节，绘图中的细节选项是由称为图形上下文（Graphics Contents，简称 GC）的资源来描述的。对于任何绘图操作，都需要指定所使用的 GC。GC 就象绘图用的笔，它控制了所绘图形的前景背景颜色、线条宽度、线型等一系列要素。简而言之，对于整个绘图操作，图元负责确定画什么，而 GC 则决定如何画。

22.1.5 事件

1. 事件的概念

在 X 中，键盘击键、鼠标在窗口中移动或鼠标按键动作、用窗口管理程序改变应用程序窗口的大小等等，都是事件。事件是 X 的核心部分之一。X 应用程序是一种事件驱动的程序，在事件驱动的程序中，程序在完成了初始化之后就把程序的控制转交给用户，用户对程序进行操作，程序接受到需要处理的事件后进行相应的处理，然后再将控制交给用户，等待新的用户事件。这是一个不断循环的过程，即所谓的事件循环，直到程序接受到某种事先约定的用于退出程序的事件，或者程序运行到某种退出条件时，才退出事件循环并退出程序。注意：我们的讨论范围限定在 Xlib 对事件的处理，它是所有基于 X 窗口系统的应用程序的事件处理机制的基础，但目前一些高级的工具包提供了更完善也更方

便的事件处理方法，我们的目的是让大家理解有关事件处理的基本原理。

通常，X 应用程序的事件处理包括三个的步骤：首先是为每个窗口选择事件，然后将窗口映象出来，最后进入事件循环，等接受事件后进行相应的处理。

2. 事件的结构

在 Xlib 中，事件的信息保存在一定的数据结构中，共有 30 余种不同的事件结构。最简单的事件结构为 XAnyEvent 事件，其结构定义为：

```
typedef struct {
    int type;
    unsigned long serial;/* # of last request processed by server */
    Bool send_event;     /* true if this came from a SendEvent request */
    Display *display;    /* Display the event was read from */
    Window window;   /* window on which event was requested in event mask
*/
} XAnyEvent;
```

XAnyEvent 中包含了各种事件的最基本信息，因此实际上所有的 X 事件结构都包含了 XAnyEvent 结构中的各个成员，除此之外，多数事件结构中还包含更多的信息。各种事件结构中的第一个成员都是 type 项，即事件的种类；serial 项是服务器处理的上一个请求的序号，在调试程序时可能用到；send_event 是一个表明事件来源的标志，Ture 表示事件来自其他客户程序（一个应用程序可以用 XSendEvent() 向其他程序发送事件），False 则表示事件来自服务器；display 项表明事件来自的服务器；window 表明了选择与接收事件的窗口。

XEvent 是一个联合，它包括所有事件的结构。联合中第一个成分即为事件类型 type，程序可由 XEvent 中的 type 项来确定事件类型。XEvent 联合的定义为：

```
typedef union _XEvent {
    int type;         /* must not be changed; first element */
    XAnyEvent xany;
    XKeyEvent xkey;
    XButtonEvent xbutton;
    XMotionEvent xmotion;
    XCrossingEvent xcrossing;
    XFocusChangeEvent xfocus;
    XExposeEvent xexpose;
    XGraphicsExposeEvent xgraphicsexpose;
    XNoExposeEvent xnoexpose;
    XVisibilityEvent xvisibility;
    XCreateWindowEvent xcreatewindow;
    XDestroyWindowEvent xdestroywindow;
    XUnmapEvent xunmap;
    XMapEvent xmap;
    XMapRequestEvent xmaprequest;
    XReparentEvent xreparent;
    XConfigureEvent xconfigure;
    XGravityEvent xgravity;
    XResizeRequestEvent xresizerequest;
    XConfigureRequestEvent xconfigurerequest;
    XCirculateEvent xcirculate;
    XCirculateRequestEvent xcirculaterequest;
    XPropertyEvent xproperty;
    XSelectionClearEvent xselectionclear;
```

```
    XSelectionRequestEvent xselectionrequest;
    XSelectionEvent xselection;
    XColormapEvent xcolormap;
    XClientMessageEvent xclient;
    XMappingEvent xmapping;
    XErrorEvent xerror;
    XKeymapEvent xkeymap;
    long pad[24];
} XEvent;
```

22.2 创建一个简单的 X 窗口程序

一个典型的 X 应用程序一般应包括下列内容：

- 同 X 服务器建立联系
- 获取屏幕信息
- 使用窗口创建函数产生窗口
- 向窗口管理程序发送指示
- 选择需要的事件类型
- 加载需要的字体
- 选择合适的颜色
- 产生图形上下文（GC）来控制绘图操作
- 窗口显示
- 进入事件循环，等待事件
- 对事件进行响应
- 出错处理

下面各小节按照程序的一般流程顺序，对其中的主要内容进行介绍。你在阅读时，可参考源程序中的相应部分。

22.2.1 同 X 服务器建立联系

一个 X 程序首先要作的事情就是与 X 服务器建立联系，对应用程序而言，这相当于打开显示器。可采用 Xlib 的函数 XOpenDisplay() 完成这一工作，其用法是：

```
Display XOpenDisplay(name)
    char * name;
```

其中，name 为要连接的服务器的显示器名，形式为“host:server.screen”，比如，“tttt:0.0”表示主机 tttt 的第一个服务器的第一个显示屏幕。若不定义 name，它应该被设置为 NULL，此时 XOpenDisplay 将自动采用环境变量 DISPLAY 来定义。你可采用“**echo $ DISPLAY**”命令来了解当前 DISPLAY 的内容，也可采用“**setenv DISPLAY = *name***”命令来改变 DISPLAY。通常在应用程序中不指定 name，这样可以在运行时通过设置 DISPLAY 变量或用命令行参数来改变程序显示屏幕。

22.2.2 获取屏幕信息

为了获取良好的视觉效果，许多应用程序，在创建窗口前需要知道屏幕的大小、支持

的颜色等信息。可通过下列两种途径来获得这些信息。

第一种方式是利用宏命令，如 DefaultScreen()、DefaultDepth()、DisplayWidth() 和 DisplayHeight() 等，这些宏命令多数在 Xlib.h 中定义。需要注意的是该方式仅能获取根窗口信息。比如，在服务器与 Xlib 成功地建立连接以后，可采用下列语句来获取屏幕信息：

```
Display display;
int  screen;

display = XOpenDisplay();
screen = DefaultScreen(display);
```

注意其中 DefaultScreen() 不是一个函数，而是一个宏命令，它返回 X 根窗口的屏幕信息，因为一个 X 服务器一般拥有多个屏幕。

又比如：

```
int depth;
depth = DefaultDepth(display, screen);
```

其中宏命令 DefaultScreen() 返回颜色的位数（深度），如 depth 为 1，则是单色系统，如 depth 为 8 或 24，则为 8 位或 24 位的彩色显示系统。

宏命令 DisplayWidth() 和 DisplayHeight() 可返回屏幕的大小，以象素为单位的大小，即屏幕上垂直方向和水平方向的点数：

```
int  width, height;
width = DisplayWidth(display, screen);
height = DisplayHeight(display, screen);
```

常用的宏命令还有：

```
char * ServerVendor(display)      返回厂商名字字符串
int VendorRelease(display)        返回厂商服务器版本号
int ProtocalVersion(display)      返回X版本号
int ProtocalRevision(display)     返回R版本号
```

第二种方式是用函数 XGetGeometry() 或 XGetWindowAttributes() 来获取任意窗口的几何尺寸和属性。两函数的定义为：

```
int XGetGeometry(display, root, x, y, width, height, border_width, depth)
    Display * display;
    Window * root;
    int *x, * y;
    unsigned int * width, * height;
    unsigned int * border_width;
    unsigned int * depth;

int XGetWindowAttributes(display, window, window_attributes)
    Display * display;
    Windows window;
    XWindowAttributes * window_attrbutes;
```

其中，Display 是程序运行的显示，window 是要获得其特性的窗口，window_attributes 是包含窗口各种信息的结构，结构类型为 XWindowAttributes，其定义为：

```
typedef struct {
```

```
    int x, y;              /* location of window */
    int width, height;     /* width and height of window */
    int border_width;      /* border width of window */
    int depth;             /* depth of window */
    Visual *visual;        /* the associated visual structure */
    Window root;           /* root of screen containing window */
#if defined(__cplusplus) || defined(c_plusplus)
    int c_class;           /* C++ InputOutput, InputOnly*/
#else
    int class;             /* InputOutput, InputOnly*/
#endif
    int bit_gravity;       /* one of bit gravity values */
    int win_gravity;       /* one of the window gravity values */
    int backing_store;     /* NotUseful, WhenMapped, Always */
    unsigned long backing_planes;  /* planes to be preserved if possible */
    unsigned long backing_pixel;
                           /* value to be used when restoring planes */
    Bool save_under;       /* boolean, should bits under be saved? */
    Colormap colormap;     /* color map to be associated with window */
    Bool map_installed;    /* boolean, is color map currently installed*/
    int map_state;         /* IsUnmapped, IsUnviewable, IsViewable */
    long all_event_masks;     /* set of events all people have interest in*/
    long your_event_mask;     /* my event mask */
    long do_not_propagate_mask;/* set of events that should not propagate *
/
    Bool override_redirect;  /* boolean value for override-redirect */
    Screen *screen;          /* back pointer to correct screen */
} XWindowAttributes;
```

22.2.3 产生窗口

在 X 应用程序中创建窗口需要调用 Xlib 的窗口创建函数，基本的函数是 XCreateWindow()，其定义为：

```
Window XCreateWindow(display, parent, x, y, width, height, border_width,
                     depth, class, visual, valuemask, attributes)
    Display * display;
    Window  parent;
    int x, y, depth;
    unsigned int width, height, border_width, class;
    Visual  * visual;
    unsigned long valuemask;
    XSetWindowAttributes  * attributes;
```

其中，display 是与程序相连的显示。parent 是父窗口，对于程序的最上级窗口来说，其父窗口应是系统的根窗口。class 代表创建窗口的类，可以是 InputOutput、InputOnly 或 CopyFromParent，分别代表输入输出型、只输入型或从父窗口继承型。结构 XSetWindowAttributes 中定义了窗口需要设置的各种参数，其定义为：

```
typedef struct {
    Pixmap background_pixmap;     /* background or None or ParentRelative */
    unsigned long background_pixel;  /* background pixel */
    Pixmap border_pixmap;            /* border of the window */
    unsigned long border_pixel;      /* border pixel value */
    int bit_gravity;                 /* one of bit gravity values */
    int win_gravity;                 /* one of the window gravity values */
    int backing_store;               /* NotUseful, WhenMapped, Always */
```

```
    unsigned long backing_planes;     /* planes to be preseved if possible */
    unsigned long backing_pixel;      /* value to use in restoring planes */
    Bool save_under;                  /* should bits under be saved? (popups) */
    long event_mask;                  /* set of events that should be saved */
    long do_not_propagate_mask;       /* set of events that should not propagate */
    Bool override_redirect;           /* boolean value for override-redirect */
    Colormap colormap;                /* color map to be associated with window */
    Cursor cursor;                    /* cursor to be displayed (or None) */
} XSetWindowAttributes;
```

利用上述函数创建窗口，需要定义大量参数。首先必须建立窗口的若干属性，将这些属性放在 XSetWindowAttributes 结构中，同时还要建立一个屏蔽，告诉 X 服务器一些信息。

XCreateSimpleWindow() 是一个简单的 X 窗口创建函数，可以大大简化窗口创建工作，其定义为：

```
Window  XCreateSimpleWindow(display, parent, x, y, width, height,
    border_width, border, background)
    Display * display;
    Window  parent;
    int     x, y ;
    unsigned int  width, height, border_width;
    unsigned long border, background ;
```

其中 x，y 是窗口在父窗口中的位置坐标（从窗口的左上角算起），width、height、border_width 分别是窗口的宽、高和边框宽度，border 和 background 则为窗口的边框和背景的颜色。

窗口创建后，程序中还可以利用 XSetWindowAttributes() 函数修改窗口属性，其用法可以参考 X 程序员手册或系统的联机帮助。

22.2.4 图标、字体和颜色

1. 图标

按键是图形用户界面中最常用的成份之一，很多程序在按键上使用一定图案的图标，从而使界面更美观，操作更直观。在 X 窗口中，按键图标可以用 X 系统附带的应用程序 bitmap 画出，然后在 X 应用程序中将这个位图文件包含进来，在需要时调用函数 XCreateBitmapFromData() 将其转换为 Pixmap 类型的数据，以便在 X 窗口中画出。该函数的用法为：

```
#include "bitmap_file"
Pixmap  icon;
icon = XCreateBitmapFromData(display, window, icon_bits,
                             icon_width, icon_height);
```

其中，icon_bits、icon_width 和 icon_height 分别是位图文件 bitmap_file 中定义的位图表和位图尺寸常量，下面就是一个位图文件的例子：

```
# define icon_width 16
# define icon_height 16
static unsigned char icon_bits[ ]={
   0x3f, 0x3f, 0x3f, 0x3f, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xfc, 0xfc, 0xfc
```

```
    0x00, 0x00, 0x00, 0x00, 0x3f, 0x3f, 0x3f, 0x3f, 0x00, 0x00, 0x00, 0x00
,
    0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x00, 0x00, 0x00};
```

2. 字体

应用程序要在窗口中显示字符或字符串，需要先选定字体加载到 X 服务器上，并将加载的字体设定为 GC 中的字体，然后才能利用 X 绘制字符串的函数在窗口中画出字符串。加载字体可以使用 XLoadQueryFont() 或 XLoadFont() 函数。

多数程序采用 XLoadQueryFont() 加载字体，它返回一个类型为 XFontStruct 的结构，如下所示：

```
typedef struct {
    XExtData *ext_data;  /* hook for extension to hang data */
    Font      fid;       /* Font id for this font */
    unsigned direction;  /* hint about direction the font is painted */
    unsigned min_char_or_byte2;  /* first character */
    unsigned max_char_or_byte2;  /* last character */
    unsigned min_byte1;          /* first row that exists */
    unsigned max_byte1;          /* last row that exists */
    Bool all_chars_exist;    /* flag if all characters have non-zero size*
/
    unsigned default_char;   /* char to print for undefined character */
    int      n_properties;   /* how many properties there are */
    XFontProp  *properties;  /* pointer to array of additional properties*
/
    XCharStruct  min_bounds; /* minimum bounds over all existing char*/
    XCharStruct  max_bounds; /* maximum bounds over all existing char*/
    XCharStruct  *per_char;  /* first_char to last_char information */
    int      ascent;         /* log. extent above baseline for spacing */
    int      descent;        /* log. descent below baseline for spacing */
} XFontStruct;
```

用 XLoadQueryFont() 加载时，字体标识符存储在 XFontStruct 结构中的 fid 域中，即 XFontStruct.fid。而用 XLoadFont() 函数则可简单确定字体标识符。

当字体加载完后，可用 XSetFont() 函数将字体的标识符设置到图形上下文中。如：

```
Display * display;
GC  gc;
XFontStruct   * fontstr;

fontstr = XLoadQueryFont(display, "9x15");
XSetFont(display, gc, fontstr->fid);
```

当程序不再使用某一字体时，应及时用 XFreeFont() 或 XUnloadFont() 函数将其释放，前者用于释放 XFontStruct 类型数据，而后者用于释放 Font 类型数据。

3. 颜色

颜色是任何与图形有关的程序中常用的资源，在 X 应用程序中，颜色是由对应的象素值（Pixel）代表和实现的。由于象素值本身实际上是一个无符号长整型数（unsigned long），从其取值并不能直接看出所对应的颜色，因此应用程序中一般不直接通过选择象素值来选择颜色，而是使用专门的颜色（Color）数据结构。这些颜色可以用颜色名或颜色的红绿蓝三色成份来定义，然后通过调用有关 X 函数来进行颜色分配，返回对应的象

素值，也可以直接采用一些宏命令得到系统中缺省的颜色，如用 BlackPixel() 和 WhitePixel() 可以得到系统中的黑色和白色所对应的象素值。

在 X 窗口系统中，使用色表（Colormap）的概念来定义应用程序可使用的颜色。在多数情况下，应用程序可采用系统标准的隐含色表。可以用下面的方法取得系统隐含色表：

```
depth = DefaultDepth(display, screen);
colormap = DefaultColormap(display, screen);
```

颜色的使用涉及到的内容较多，由于篇幅所限，本书不做过多的叙述，请大家参考有关文献。

22.2.5 与窗口管理器建立联系

窗口创建以后，应用程序应向窗口管理系统发出提示，告诉窗口管理系统一些基本的信息，以便管理器能够正常管理应用程序产生的窗口，正确处理该窗口与其他窗口之间的关系。一个最基本的信息集包括窗口名、窗口的最小化图标名、命令行参数、窗口位置、大小等。向窗口管理系统发出提示的函数为：

```
void XSetWMProperties(display, window, window_name, icon_name,
                      argc, argv, normal_hints, wm_hints, class_hints)
    Display  * display;
    Window  window;
    XTextProprty  window_name,  icon_name;
    int      argc;
    char  ** argv;
    XSizeHints  * normal_hints;
    XWMHints  * wm_hints;
    XClassHint  * class_hints;
```

其中 XSizeHints 结构的定义为：

```
typedef struct {
    long flags;       /* marks which fields in this structure are defined *
/
    int x, y;              /* obsolete for new window mgrs, but clients */
    int width, height;   /* should set so old wm's don't mess up */
    int min_width, min_height;
    int max_width, max_height;
    int width_inc, height_inc;
    struct {
         int x;            /* numerator */
         int y;            /* denominator */
    } min_aspect, max_aspect;
    int base_width, base_height;      /* added by ICCCM version 1 */
    int win_gravity;                  /* added by ICCCM version 1 */
} XSizeHints;
```

XWMHints 结构的定义为：

```
typedef struct {
    long flags;  /* marks which fields in this structure are defined */
    Bool input;  /* does this application rely on the window manager
                    to get keyboard input? */
    int initial_state;        /* see below */
    Pixmap icon_pixmap;       /* pixmap to be used as icon */
    Window icon_window;       /* window to be used as icon */
```

```
    int icon_x, icon_y;        /* initial position of icon */
    Pixmap icon_mask;          /* icon mask bitmap */
    XID window_group;          /* id of related window group */
    /* this structure may be extended in the future */
} XWMHints;
```

XClassHint 结构的定义为：

```
typedef struct {
    char *res_name;
    char *res_class;
} XClassHint;
```

22.2.6 选择事件类型

并不是每一个应用程序窗口都能处理任何事件，要想窗口支持某一事件的处理必须首先选择该事件。选择事件最简单的方式是先设置事件屏蔽（EventMask），用来指定程序接受哪些事件类型，再调用 XSelectInput ()。用法如下：

```
XSelectInput(display, window, event_mask)
    Display * display;
    Window window;
    long event_mask;

event_mask = ExposureMask | KeyPressMask | ButtonPressMask;
```

其中事件屏蔽 event_mask 用运算符“|”进行或运算，来选择多种事件。比如上例中表示选择暴露事件、键盘键按下事件和鼠标按钮按下事件。

在 X 应用程序中，下面三类事件一般是要选择的：一是窗口被屏幕上的其他窗口全部或部分遮挡之后重新显示，这种事件称为暴露（Exposure）事件，窗口在暴露事件之后需要重画其中的内容；二是窗口尺寸变化以后，重新计算并显示，这种事件称作尺寸改变（Resize）事件，例子程序中没有选择该事件，你可以看一下有什么后果；三是接受用户的鼠标或键盘操作事件。在 X 窗口系统中，缺省设置并不接受这些事件，即使发生也不予理睬，因而应用程序应根据实际需要选择有关的事件，以完成相应的处理。

需要注意的是，在选择事件以后和进入事件等待循环之前，接受事件的窗口必须出现在屏幕上（参见窗口显示一节），否则子窗口将错过第一次产生绘图等操作的暴露事件。

下面介绍几种常用的事件：

（1）进入/离开事件

在 X 中，鼠标进入或离开窗口时，将产生 EnterNotify 或 LeaveNotify 鼠标事件，其事件选择屏蔽分别为 EnterWindowMask 和 LeaveWindowMask。键盘上有键输入时，将进入输入焦点所在的窗口。相应的，当窗口获得输入焦点和失去输入焦点时，将产生 FocusIn 和 FocusOut 事件，其事件选择屏蔽为 FocusChangeMask。

（2）暴露事件

暴露事件是由 X 产生的一个与窗口有关的事件类型，它通知应用程序哪个窗口或窗口的哪个区域已经由不可见变为可见。由于 X 窗口系统本身不保存窗口中的内容，所以当窗口重新显示时，程序需要重画窗口的内容或者将保留在内存变量中的窗口内容重新复

制到窗口上。暴露事件的选择屏蔽是 ExposureMask。

XExposeEvent 是 X 暴露事件的数据结构，XEvent 联合中的 xexpose 项，其结构定义为：

```
typedef struct {
    int type;
    unsigned long serial;/* # of last request processed by server */
    Bool send_event;     /* true if this came from a SendEvent request */
    Display *display;    /* Display the event was read from */
    Window window;
    int x, y;
    int width, height;
    int count;           /* if non-zero, at least this many more */
} XExposeEvent;
```

其中，(x,y)是暴露部分的左上角坐标（相对于窗口左上角），width 和 height 则为暴露部分的宽度和高度。

（3）鼠标事件

大多数 X 软件假定鼠标有三个按键，但实际上 X 系统本身定义了五个鼠标按键（在最常用的三键鼠标上，第四和第五个按键是由同时按下两个按键来模拟的）。

鼠标事件可分为两类：

一类是用户按下或松下鼠标某一按键，事件分别是 ButtonPress 和 ButtonRelease，称为鼠标按键事件。事件选择屏蔽为 ButtonPressMask 和 ButtonReleaseMask。数据结构为 XButtonEvent，它同时还定义了 XButtonPressedEvent 和 XbuttonReleasedEvent 两个别名（如下所示），在 Xevent 联合中为 xbutton 项。

```
typedef struct {
    int type;            /* of event */
    unsigned long serial;/* # of last request processed by server */
    Bool send_event;     /* true if this came from a SendEvent request */
    Display *display;    /* Display the event was read from */
    Window window;       /* "event" window it is reported relative to */
    Window root;         /* root window that the event occured on */
    Window subwindow;    /* child window */
    Time time;           /* milliseconds */
    int x, y;            /* pointer x, y coordinates in event window */
    int x_root, y_root;  /* coordinates relative to root */
    unsigned int state;  /* key or button mask */
    unsigned int button; /* detail */
    Bool same_screen;    /* same screen flag */
} XButtonEvent;
typedef XButtonEvent XPuttonPressedEvent;
typedef XButtonEvent XPuttonReleasedEvent;
```

另一类是用户移动鼠标的事件，如 MotionNotify 等，称为鼠标移动事件。事件选择屏蔽有 ButtonMotionMask、Button1MotionMask、Button2MotionMask、Button3MotionMask 等等。鼠标移动事件的数据结构为 XMotionEvent，别名为 XPointerMovedEvent（如下所示），在 Xevent 联合中为 xmotion 项。

```
typedef struct {
    int type;            /* of event */
    unsigned long serial;/* # of last request processed by server */
```

```
    Bool send_event;      /* true if this came from a SendEvent request */
    Display *display;     /* Display the event was read from */
    Window window;        /* "event" window reported relative to */
    Window root;          /* root window that the event occured on */
    Window subwindow;     /* child window */
    Time time;            /* milliseconds */
    int x, y;             /* pointer x, y coordinates in event window */
    int x_root, y_root;   /* coordinates relative to root */
    unsigned int state;   /* key or button mask */
    char is_hint;         /* detail */
    Bool same_screen;     /* same screen flag */
} XMotionEvent;
typedef XMotionEvent XPointerMovedEvent;
```

由 XButtonEvent 和 XMotionEvent 数据结构的定义可以看出，鼠标事件中常用的信息有：事件发生在哪一个窗口（windows）、事件的 XY 位置（x，y）以及鼠标状态（state 等）等。可用

```
window = event.xbutton.window;
```

等语句获取信息。其中 state 不仅可以反映出鼠标按键的状态，而且还报告键盘上的 Shift、Ctrl 等控制键的状态，从而可以实现复杂的操作。要想对这些键进行检查，需要使用一些屏蔽对状态域中某些位进行检查，如 Button1Mask、Button2Mask、ShiftMask、ControlMask 等。用法如下：

```
if(event.xmotion.state & Button1Mask) {
    ...    / * process the event of pointer motion while button1 pressed */
   }
```

（4）键盘事件

几乎所有的工作站都支持 KeyPress 和 KeyRelease 事件，即键盘键的按下和放开事件。但个别计算机不支持 KeyRelease 事件，因而应尽量避免使用。键盘事件定义了 XKeyEvent 结构（如下所示）来储存键盘的信息，在 XEvent 联合中为 xkey 项。

```
typedef struct {
    int type;             /* of event */
    unsigned long serial;/* # of last request processed by server */
    Bool send_event;      /* true if this came from a SendEvent request */
    Display *display;     /* Display the event was read from */
    Window window;        /* 'event' window it is reported relative to */
    Window root;          /* root window that the event occured on */
    Window subwindow;     /* child window */
    Time time;            /* milliseconds */
    int x, y;             /* pointer x, y coordinates in event window */
    int x_root, y_root;   /* coordinates relative to root */
    unsigned int state;   /* key or button mask */
    unsigned int keycode;/* detail */
     Bool same_screen;    /* same screen flag */
} XKeyEvent;
typedef XKeyEvent XKeyPressedEvent;
typedef XKeyEvent XKeyReleasedEvent;
```

X 系统使用了 KeySym（键盘表）的概念来获取不同的键盘键。KeySym 允许程序有较强的可移植性，能适应不同类型的键盘。KeySym 的一个基本任务就是把某一指定的

机器键码转换成通用的 ASCII 码。X 系统中提供了 XLookupString() 函数来完成这一工作。用法如下：

```
XKeyEvent event;
XComposeStatus status;
KeySym  Keysym;
int  bufferlen = 64;
char keybuffer[65];
int  num;

num = XLookupString(&event, keybuffer, bufferlen, &keysym, &status);
if (num == 1)  printf ("Key % c on keyboard is pressed.\n",keybuffer [0]);
```

所有的 KeySym 都定义在头文件 <keysymdef.h> 中。

22.2.7 创建和设置 GC

程序使用 GC 之前，必须先创建之。基本的创建方法是采用函数 XCreateGC()，用法如下：

```
GC  XCreateGC(display, drawable, valuemask, values)
      Display           * display;
      Drawable          drawable;
      unsigned long     valuemask;
      XGCValues         * values;
```

其中，display 就是程序采用的显示；drawable 可以是一个窗口（Window），也可以是一个象素图（Pixmap）；values 是 XGCValues 类型的一个结构指针；valuemask 是一个屏蔽，通过判断与 XGCValues 结构中各成员对应的二进制位为“0”还是为“1”确定真正被读取的成员。

XGCValues 数据结构的定义为：

```
typedef struct {
    int function;              /* logical operation */
    unsigned long plane_mask;/* plane mask */
    unsigned long foreground;/* foreground pixel */
    unsigned long background;/* background pixel */
    int line_width;            /* line width */
    int line_style;            /* LineSolid, LineOnOffDash, LineDoubleDash *
/
    int cap_style;             /* CapNotLast, CapButt,
                                  CapRound, CapProjecting */
    int join_style;            /* JoinMiter, JoinRound, JoinBevel */
    int fill_style;            /* FillSolid, FillTiled,
                                  FillStippled, FillOpaeueStippled */
    int fill_rule;             /* EvenOddRule, WindingRule */
    int arc_mode;              /* ArcChord, ArcPieSlice */
    Pixmap tile;               /* tile pixmap for tiling operations */
    Pixmap stipple;            /* stipple 1 plane pixmap for stipping */
    int ts_x_origin;           /* offset for tile or stipple operations */
    int ts_y_origin;
    Font font;                 /* default text font for text operations */
    int subwindow_mode;        /* ClipByChildren, IncludeInferiors */
    Bool graphics_exposures; /* boolean, should exposures be generated */
    int clip_x_origin;         /* origin for clipping */
    int clip_y_origin;
```

```
    Pixmap clip_mask;          /* bitmap clipping; other calls for rects */
    int dash_offset;           /* patterned/dashed line information */
    char dashes;
} XGCValues;
```

其中具体每一成员都对应一个屏蔽位，将该位设为 1 时表示对应的结构成员将真正被读取，0 则表示它被忽略。对应每一位屏蔽位，X 均定义了屏蔽符号，在使用时可以通过按位或（“|”）进行组合。表 22-1 列出了 XGCValues 结构的各个成员对应的屏蔽符号、设置位及成员的缺省值。

表 22-1 XGCValues 结构的屏蔽符号和缺省值

结构成员	屏蔽符号	设置位	缺省值
function	GCFunction	0	GXcopy
plane_mask	GCPlaneMask	1	各位全为 1
foreground	GCForeground	2	0
background	GCBackground	3	1
line_width	GCLineWidth	4	0
line_style	GCLineStyle	5	LineSolid
cap_style	GCCapStyle	6	CapButt
join_style	GCJoinStyle	7	JoinMiter
fill_style	GCFillStyle	8	FillSolid
fill_rule	GCFillRule	9	EvenOddRule
arc_mode	GCArcMode	22	ArcPieSlice
tile	GCTile	10	用前景象素填充的象素图
stipple	GCStipple	11	用象素 1 填充的象素图
ts_x_origin	GCTileStipXOrigin	12	0
ts_y_origin	CCTileStipYOrigin	13	0
font	GCFont	14	（与运行环境有关）
subwindw_mode	GCSuswindowMode	15	ClipByChildren
graphics_exposures	GCGraphicsExposures	16	True
clip_x_origin	GCClipXOrigin	17	0
clip_y_origin	GCClipYOrigin	18	0
clip_mask	GCClipMask	19	None
dash_offset	GCDashOffset	20	0
dashes	GCDashList	21	4（即[4, 4]）

下面的程序片段说明了如何创建、设置 GC：

```
Display * display;
Screen screen;
GC gc1, gc2;
XGCValues gcvalues;
...

/* assign values to members of XGCValues */
gcvalues.foreground = Blackpixel(display, screen);
gcvalues.background = WhitePixel(display, screen);

/* create two GCs */
```

```
gc1 = XCreateGC(display, RootWindow(display, screen),
                (GCForeground | GCBackground), &gcvalues);
gc2 = XCreateGC(display, RootWindow(display, screen),
                (GCForeground | GCBackground), &gcvalues);


/* change the values of gc1 */
XSetForeground(display, gc1, gc2, WhitePixel(display, screen));
XSetBackground(display, gc1, gc2, BlackPixel(display, screen));
```

上述程序片段中，同时创建了两个不同特征的 GC，这在实际应用程序中有时很有用。因为在一些服务器上不同 GC 之间的切换要比改变 GC 成员的设置更快。创建 GC 是要占用一定的系统资源的，因此 GC 不再使用时应及时释放它所占用的资源——使用 XFreeGC ( ) 函数，其用法为：

```
XFreeGC(display, gc)
    Display * display;
    GC gc;
```

22.2.8　窗口显示

X 窗口在创建后并不是马上就显示出来，而是要将它们映象到显示上，可以使用 XMapWindow（display, window）函数（用法如下）。注意，要使窗口可见，程序必须向窗口管理器发送窗口管理信息以使窗口管理器能正常地处理窗口和绘图，同时暴露事件也必须设置以便在窗口中把图形画出来。

```
XMapWindow(display, window)
Display display;
Window window;
```

22.2.9　事件循环和处理

1. 事件循环

前已述及，大多数 X 程序在完成初始化后，就进入一个无限的事件等待循环，接收到一个事件后调用相应的处理程序进行处理，然后仍返回事件循环，等待新的事件。在例子程序中使用 *while* (1) {} 语句实现这一循环（参见事件处理的例子）。

2. 事件处理

可由 XNextEvent() 函数来完成接收并挑选事件类型这一工作，XNextEvent() 检查程序的事件队列是否有事件，若有，返回事件队列中的第一个事件；若没有，则等待，直至队列中有事件出现为止。用法如下：

```
Display * display;
XEvent event;

while(1) {
XNextEvent(display, &event);
switch(event.type) {
    case Expose:
      ... /* do necessary procedures to handle this exposure event. */
```

```
    case ButtonPress:
      … /* check which button is pressed and do corresponding work. */
    case KeyPress:
      … /* check which key is typed and to corresponding work. */
    … /* process other events. */
    }
}
```

22.2.10 绘图

前已述及，整个绘图操作，图元负责确定画什么，而 GC 则决定如何画。

1. GC对绘图的控制

GC 对绘图的控制体现在 XGCValues 数据结构成员的取值上。下面对一些常用的成员进行介绍：

（1）与画线有关的成员

包括六个 XGCValues 结构成员：line_width、line_style、cap_style、join_style、dashes 和 dash_offset。

line_width：控制线宽，以象素点为单位，可取非零值或零（零代表用最快算法来画）；

line_style：控制线型，可取 LineSolid、LineOnOffDash 或 LineDoubleDash 三者之一，分别对应于实线、断续线和双断续线。

cap_style：控制端点形状，可取 CapButt、CapNotLast、CapProjecting 或 CapRound；

join_style：控制两条相连线段的连接方式，可取 JionBevel、JoinMiter 或 JoinRound；

dashes：定义断续线的图案，可取 LineOnOffDash 或 LineDoubleDash。

dash_offset：定义断续线起点的类型（仅当 line_style 为 LineOnOffDash 或 LineDoubleDash 时才起作用）。

其中 line_width、line_style、cap_style 和 join_style 可用 XSetLineAttributes（）函数进行修改，而 dashes 和 dash_offset 则可用 XSetDashes( ) 来重新设置。

（2）与填充方式有关的成员

在 X 应用程序中，对图形的填充分为多边形填充和弧线填充两种，对应于成员 fill_rule 和 arc_mode。

fill_rule：可取 EvenOddRule 或 WindingRule。EvenOddRule 方式是根据区域被填充次数是奇/偶数来决定是/否进行填充，这是 X 系统隐含的 GC 方式；WindingRule 方式则不管区域是否已被填充或已被填充几次都进行填充。一般情况下，这两种填充方式的效果是相同的，只有当填充形状十分复杂时才会表现出区别。可用函数 XSetFillRule() 来改变。

arc_mode：可取 ArcPieSlice（扇形方式填充）或 ArcChord（弦线方式填充）。ArcPieSlice 方式对弧线和弧线两端与弧线中心相连的两条线段所围成的区域进行；而 ArcChord 方式则对弧线和弧线两端连线（弦）所围成的区域进行填充。可用函数 XSetArcMode() 来改变。

（3）与颜色和图案有关的成员

包括 fill_style、title、stipple、ts_x_origin 和 ts_y_origin。其中 fill_style 确定颜色和图案策略，可取 FillSolid、FillTiled、FillStippled 和 FillOpaqueStippled。

fill_style 取 FillSolid 是系统缺省的颜色策略，也是最简单的方法。这种方式下 XGCValues 成员 foreground（前景色）就是被画象素点的颜色，可用 XSetForeground() 来设定前景色。

fill_style 取 FillTiled 时为 Tile 绘图方式。所谓 Tile 绘图方式就是象铺瓷砖那样将成员 tile 所指的象素图的图案规则地重复铺满绘图目标区。成员 tile 的取值可以用 XSetTtile() 来设定或修改。第一个象素图在绘图区中的位置是另外两个成员 ts_x_origin 和 ts_y_origin 确定，可用 XSetTSOrigin() 来设定。

fill_style 取 FillStippled 和 FillOpaqueStippled 时为 Stipple 绘图方式。Stipple 的英文本意为点画或点刻，指的是在绘图或雕刻时按照一定的控制图案逐点操作的方式。在 X 中，stipple 可译为点画模板。同 tile 一样，stipple 在 XGCValues 结构中也是一个 Pixmap 类型的数据。与 tile 不同的是它的颜色深度是 1，也就是说作为点画模板的象素图中只有设定和未设定两种象素点。绘图区中各个象素点上由前景色和点画模板共同决定，点画模板也同样重复铺满整个绘图区。在 FillStippled 方式下，所绘图形是前景与点画模板的与，即在点画模板设定的象素点上画出前景的颜色；而在 FillOpaqueStippled 方式下，在点画模板设定的象素点上画出前景颜色，而未设定的象素点上面背景颜色，因为所有象素点将或者被画上了前景颜色，或者被画上了背景颜色，绘图区中原来的图案（如果有的话）被完全覆盖，因此称为不透明（Opaque）点画方式。stipple 可以用 XSetStipple() 进行设定或修改。

2. 图元

所谓的图元就是绘图用的函数。下面简要介绍一下常用的图元。

（1）画点

画一个点：

```
XDrawPoint(display, drawable, gc, x, y)
    Display display;
    Drawable drawable;
    GC  gc;
    int  x, y;
```

画多个点：

```
XDrawPoints(display, drawable, gc, points, num_points, mode)
    Display display;
    Drawable drawable;
    GC  gc;
    XPoint * points;
    int num_points;
    int mode;
```

（2）画线

画一条直线：

```
XDrawLine(display, drawable, gc, x1, y1, X2, y2)
    Display display;
    Drawable drawable;
```

```
    GC  gc;
    int  x1, y1;
```

画多条直线：

```
XDrawSegments(display, drawable, gc, segs, num_segs)
    Display display;
    Drawable drawable;
    GC  gc;
    XSegment * segs;
    int num_segs;
```

画折线：

```
XDrawLines(display, drawable, gc, points, num_points, mode)
    Display display;
    Drawable drawable;
    GC  gc;
    XPoint * points;
    int num_points;
    int mode;
```

画矩形：

```
XDrawRectangle(display, drawable, gc, x, y, width, height)
    Display display;
    Drawable drawable;
    GC  gc;
    int x, y;
    int width, height;
```

（3）画文字

画 ASCII 字符串：

```
XDrawString(display, drawable, gc, x, y, string, strlength)
    Display * display;
    Drawable drawable;
    GC  gc;
    int  x, y;
    char * string;
    int  strlength;
```

画汉字字符串：

```
XDrawText16(display, drawable, gc, x, y, items, nitems)
    Display * display;
    Drawable  drawable;
    GC  gc;
    int  x, y;
    XTextItem16 * items;
    int  nitems;
```

（4）填充

矩形填充：

```
XFillRectangle(display, drawable, gc, x, y, width, height)
    Display display;
    Drawable drawable;
    GC  gc;
    int x, y;
```

```
    int width, height;
```

多边形填充：

```
XFillPolygon(display, drawable, gc, points, npoints, shape, mode)
    Display display;
    Drawable drawable;
    GC  gc;
    XPoint * Points;
    int npoint;
    int shape, mode;
```

22.2.11 出错处理

在 X 应用程序运行中，应仔细留意可能出错的地方。在这些地方加上一些处理语句，以免导致更严重的错误，并可以快速查出出错的原因。比如在 XOpenDisplay() 时，就应该检查返回值，确认服务器连接成功，若不成功则应打印出有关信息然后退出程序，不再继续运行：

```
display = XOpenDisplay("");
if (display == NULL )   {
    fprintf(stderr, "%s:cannot open display. \n", argv[0]);
    exit(-1);
    }
```

对其他错误，要依靠有关错误处理机制进行检查和处理。通常，任何重要的函数返回状态都需要检查，并做相应处理。程序退出时，要注意释放各种加载的资源。

22.3 源程序

此例子程序摘自清华大学自动化系张学工，刘业新编著的《X Window/Motif 编程速成》。笔者对其进行了必要的修改，并在 Red Hat 5.1 环境中调试通过。

X 应用程序的编译方法十分简单，只需将系统的 X11 库连接上即可：

```
% gcc XBasicExample.C  -lX11 -o XBasicExample
```

运行“XBasicExample”的结果如图 22-1 所示。

源程序清单如下：

```
/*
* file XbasicExample.c
*/
```

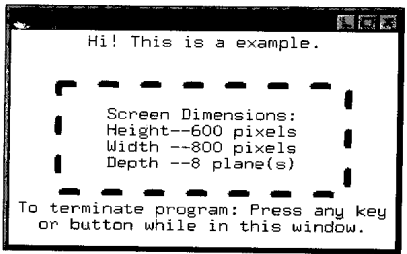


图 22-1 XBasicExample 的运行结果

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>

#include <stdio.h>

#include "/usr/X11R6/include/X11/bitmaps/icon"
#define BITMAPDEPTH 1
#define TOO_SMALL   0
#define BIG_ENOUGH  1
/* These are used as arguments to nearly every Xlib routine, so it saves
 * routine arguments to declare them global. If there were
 * additional source files, they would be declared extern there */
Display  * display;
int screen_num;

/* * * * *  The Main  * * * * */
void main(argc, argv)
    int argc;
    char ** argv;
{
Window win;
unsigned int width, height;
int  x, y;
unsigned int border_width = 4;
unsigned int display_width, display_height;
unsigned int icon_wwidth;
unsigned int icon_wheight;
char * window_name = "Basic Window Program";
char * icon_name = "basicwin";
Pixmap icon_pixmap;
XSizeHints size_hints;
XIconSize * size_list;
int count;
XEvent report;
GC gc;
XFontStruct * font_info;
```

```
int window_size = BIG_ENOUGH; /* or TOO_SMALL to display contents */

icon_wwidth=icon_width;
icon_wheight=icon_height;

/* connect to X server */
display = XOpenDisplay("");
if (display == NULL )   {
    fprintf(stderr, "%s:cannot open display. \n", argv[0]);
    exit(-1);
    }

/* get screen size from display structure macro */
screen_num = DefaultScreen(display);
display_width = DisplayWidth(display, screen_num);
display_height = DisplayHeight(display, screen_num);

/* Set the initial window position. */
x=0;
y=0;

/* size window with enough room for text  */
width = display_width/2;
height = display_height/2;

/* create opaque window  */
win=XCreateSimpleWindow(display, RootWindow(display, screen_num),
                    x, y, width, height, border_width, BlackPixel(display,
                    screen_num), WhitePixel(display, screen_num));

/* get available icon sizes from Window manager  */
if(XGetIconSizes(display, RootWindow(display, screen_num),
                        &size_list, &count) ==0)
  fprintf(stderr, "%s:Window manager didn't set icon sizes -- use default. \n
,                   argv[0]);
else  {
    ;
/* A real application would search through size-list
 * here to find an acceptable icon size, and then
 * create a pixmap of that size. This requires
 * that the application have data for several sizes
 * of icons. */
    }

/* Create pixmap of depth 1 (bitmap) from icon */
icon_pixmap=XCreateBitmapFromData(display, win,
                    icon_bits, icon_wwidth, icon_wheight);

/* Set size hints for window manager.
 * The window manager may override these settings. */

/* x, y, width, and height hints are now taken from
 * the actual settings of the window when mapped. Note
 * that PPosition and PSize must be specified anyway. */

size_hints.flags = PPosition | PSize | PMinSize;
size_hints.min_width = 300;
size_hints.min_height = 200;
```

```
{
XWMHints wm_hints;
XClassHint class_hints;
XTextProperty windowName, iconName;

/* These calls store window_name and icon_name into
 * XTextproperty structures and set their other
 * fields properly. */

if (XStringListToTextProperty(&window_name, 1, &windowName) == 0) {
        fprintf(stderr, "%s:structure allocation for windowName failed. /n",
                argv[0]);
        exit(-1);
   }

if (XStringListToTextProperty(&icon_name, 1, &iconName) == 0) {
        fprintf(stderr, "%s:structure allocation for iconName failed. /n",
                argv[0]);
        exit(-1);
   }

wm_hints.initial_state = NormalState;
wm_hints.input = True;
wm_hints.icon_pixmap = icon_pixmap;
wm_hints.flags = StateHint | IconPixmapHint | InputHint;

class_hints.res_name = argv[0];
class_hints.res_class = "Basicwin";

XSetWMProperties(display, win, &windowName, &iconName,
            argv, argc, &size_hints, &wm_hints, &class_hints);
}

/* Select event types wanted */
XSelectInput(display, win, ExposureMask | KeyPressMask |
                ButtonPressMask   StructureNotifyMask);
load_font(&font_info);

/* create GC for text and drawing */
getGC(win, &gc, font_info);

/* Display window */
XMapWindow(display, win);

/* get events, use first to display text and graphics */
while(1) {
XNextEvent(display, &report);
switch(report.type) {
   case Expose:
            /* unless this is the last contiguous expose,
             * don't draw the window */
      if (report.xexpose.count != 0)
            break;
            /* if window too small to use */
      if (window_size == TOO_SMALL)
            TooSmall(win, gc, font_info);
      else {
            /* place text in window */
            draw_text(win, gc, font_info, width, height);
```

```
                /* place graphics in window */
                draw_graphics(win, gc, width, height);
            }
        break;
    case ConfigureNotify:
                /* window has been resized, change width and
                 * height to send to draw_text and draw_graphics
                 *in next Expose */

        width = report.xconfigure.width;
        height = report.xconfigure.height;
        if((width < size_hints.min_width) || (height < size_hints.min_height))
                window_size=TOO_SMALL;
        else
                window_size=BIG_ENOUGH;
        break;
    case ButtonPress:
                /* trickle down into KeyPress(no break) */
    case KeyPress:
        XUnloadFont(display, font_info->fid);
        XFreeGC(display, gc);
        XCloseDisplay(display);
        exit(1);
    default:
                /* all events selected by StructureNotifyMask
                 * except ConfigureNotify are thrown away here,
                 * since nothing is done with them */
        break;
    } /* end switch */
  } /*  end while */

} /* end of main( )  */

getGC(win, gc, font_info)
        Window win;
        GC * gc;
        XFontStruct  *font_info;
{
unsigned long valuemask=0; /* ignore XGCvalues and use defaults */
XGCValues values;
unsigned int line_width = 6;
int line_style = LineOnOffDash;
int cap_style = CapRound;
int join_style = JoinRound;
int dash_offset = 0;
static char dash_list[] = {12, 24};
int list_length = 2;

/* Create default Graphics Context */
*gc = XCreateGC(display, win, valuemask, &values);

/* specify font */
XSetFont(display, *gc, font_info->fid);

/* specify black foreground since default window background is
 * white and default foreground is undefined. */
XSetForeground(display, *gc, BlackPixel(display, screen_num));

/* set line attributes */
```

```
XSetLineAttributes(display, *gc, line_width, line_style, cap_style, join_style
);

/* set dashes */
XSetDashes(display, *gc, dash_offset, dash_list, list_length);
; /* end of getGC */

load_font(font_info)
        XFontStruct ** font_info;
{
/* Load font and get font information structure. */
if ((*font_info = XLoadQueryFont(display, '9x15')) == NULL)
    {
     /*(fprintf(stderr,"%s:cannot open 9x15 font /n", argv[0]);*/
    printf('can not open font /n');
    exit(-1);
    }
} /* end of load_font */

draw_text (win, gc, font_info, win_width, win_height)
        Window win;
        GC gc;
        XFontStruct * font_info;
        unsigned int win_width, win_height;
{
char * string1="Hi! This is a example.";
char * string2="To terminate program: Press any key";
char * string3="or button while in this window.";
char * string4="Screen Dimensions:";
int    len1, len2, len3, len4;
int    width1, width2, width3;
char   cd_height[50], cd_width[50], cd_depth[50];
int    font_height;
int    initial_y_offset, x_offset;

/* need length for both XTextWidth and XDrawString */
len1 = strlen(string1);
len2 = strlen(string2);
len3 = strlen(string3);

/* get string widths for centering */
width1 = XTextWidth(font_info, string1, len1);
width2 = XTextWidth(font_info, string2, len2);
width3 = XTextWidth(font_info, string3, len3);

font_height = font_info->ascent + font_info->descent;

/* output text, centered on each line */
XDrawString(display, win, gc, (win_width - width1)/2,
                              font_height, string1, len1);
XDrawString(display, win, gc, (win_width - width2)/2,
        (int) (win_height - (2 * font_height)), string2, len2);
XDrawString(display, win, gc, (win_width - width3)/2,
        (int) (win_height - font_height), string3, len3);

/* copy numbers into string variables */
sprintf(cd_height, 'Height--%d pixels', DisplayHeight(display, screen_num));
sprintf(cd_width, 'Width --%d pixels', DisplayWidth(display, screen_num));
sprintf(cd_depth, 'Depth --%d plane(s)', DefaultDepth(display, screen_num));
```

```
/* reuse these for same purpose */
len4 = strlen(string4);
len1 = strlen(cd_height);
len2 = strlen(cd_width);
len3 = strlen(cd_depth);

/* To center strings vertically, we place the first string
 * so that the top of it is two font_heights above the center
 * of the window. Since the baseline of the string is what we
 * need to locate for XDrawString, and the baseline is one
 * font_info->ascent below the top of the character,
 * the final offset of the origin up from the center of
 * the window is one font_ height + one descent.  */

initial_y_offset = win_height/2-font_height-font_info->descent;
x_offset = (int)win_width/4;
XDrawString(display, win, gc, x_offset, (int) initial_y_offset, string4, len4);
XDrawString(display, win, gc, x_offset, (int) initial_y_offset + font_height,
                    cd_height, len1);
XDrawString(display, win, gc, x_offset, (int) initial_y_offset + 2*font_height,
                    cd_width, len2);
XDrawString(display, win, gc, x_offset, (int) initial_y_offset + 3*font_height,
                    cd_depth, len3);
} /* end of draw_text */

draw_graphics(win, gc, window_width, window_height)
       Window  win;
       GC gc;
       unsigned int window_width, window_height;
{
int x, y ;
int width, height;

height = window_height/2;
width = 3 * window_width/4;
x = window_width/2 - width/2; /* center */
y = window_height/2 - height/2;
XDrawRectangle(display, win, gc, x, y, width, height);
} /* end of draw_graphics */

TooSmall(win, gc, font_info)
       Window win;
       GC gc;
       XFontStruct * font_info;
{
char * string1 = "Too Small";
int     y_offset, x_offset;

y_offset = font_info->ascent + 2;
x_offset = 2;

/* output text, centered on each line */
XDrawString(display, win, gc, x_offset, y_offset, string1, strlen(string1));
} /* end of TooSmall */
/* enf of the file */
```

附录 A

佳文共赏

Linux——自由而奔放的黑马[1]

在制作电影《泰坦尼克号》所用的 160 台 Alpha 图形工作站中，有 105 台运行的是 Linux 操作系统

电影《泰坦尼克号》的导演卡梅隆在奥斯卡奖的颁奖大会上接过最佳导演奖时相当狂妄地对着全场的观众重复了男主角的一句台词："我是世界之王"。不过这也不奇怪，耗资两亿多美元，一举夺得 11 项奥斯卡大奖，赢得全球最高票房，它的导演是有资格狂妄一番的。然而真正有资格睥睨世界的主角应该是银幕后面最先进的计算机技术！毫不夸张地说，这部影片如果离开了电脑，不要说获奥斯卡奖，就是真正要开机拍摄都是不可能的。但是在笔者看来，在这一切的后面，还有一个默默无闻的大主角，这就是上百台图形工作站所使用的一套操作系统。它的名字虽然没有 UNIX 或者 NT 那么响亮，但它却有可能成为世界软件史上的一个奇迹，它的名字叫 Linux（英文读音为"LINN－ooks"）。

Linux 是一个极年轻的操作系统，它的诞生日期从 1991 年算起，迄今还不满八年，但是它的发展和成长却迅捷无比，成为操作系统领域中一匹名符其实的黑马。迄今为止，对 Linux 在全球范围内的装机台数的估计各有说法，最低的估计为 300 万，最高的估计数字为 900 万。而 1997 年，MacOS 的装机台数为 380 万，IBM OS/2 为 120 万，Windows NT 则为 700 多万。虽说 Linux 还无法与拥有一亿多用户的 Windows 相比，但是它确立自身地位和影响力所花费的时间却只有 Windows 的一半。作为一种 UNIX 操作系统，Linux 的强大性能显然使得其他品牌的 UNIX 黯然失色。有分析家认为，"Linux 的广泛普及已使其成为 UNIX 市场上最具活力的一只新军。"甚至连 UNIX 之父 Dennis Ritchie 也认为 Linux "确实不错"。有一些分析家甚至认为，在未来数年间，Linux 将成为 NT 真正强有力的对手，也是唯一可以冲破微软垄断性文化圈的出路所在。

芬兰、《卡勒瓦拉》、赫尔辛基和 Linus

芬兰可以说是世界上唯一一个国土面积按比例来说处在北极圈内最多的国家（约 1/4 的国土），在这个寒冷的国度里，遍布着大大小小约六万多个湖泊，芬兰也因此被人们称

---

[1] 本文选自水木清华 BBS 站 Linux 板，原作者文山。略有修改。

为“千湖之国”。在芬兰，一年中实际上只有三个季度，即春冬、夏季和秋冬。“严寒的冬天”长达 8 个月之久，而夏天却只有 60 天左右。芬兰的历史可以说是芬兰人与自然、与寒冷做艰苦卓绝的斗争的一个神话。

芬兰人的民族史诗《卡勒瓦拉》就记载了这个民族从远古时代起直到圣母玛丽亚生下英雄卡勒利亚王为止的所有神话。这部史诗的作者从丰富的民间传说、神话及歌谣中汲取了一切养料和精彩篇章，将它们收集、改编并润色整理，1835 年初版时有 35 篇长诗，共 12000 多行；而 1849 年再版时，篇幅几乎增加了一倍，共 50 篇长诗，23000 余行，最终成为芬兰人的“荷马史诗”。140 多年后，又有一位芬兰人创造了另一部伟大的“史诗”，不过这一次他用的是计算机语言，他收集这部“史诗”创作素材的地方是覆盖全球的因特网。这使得他的创作从一开始便具有了国际性，使得他的这部“史诗”成了一部国际性的作品。这部“史诗”的问世，很有可能在本世纪以及下一个世纪成为芬兰人对于世界的最巨大的贡献。这部“史诗”的创作是从芬兰首都赫尔辛基开始的。

芬兰首都赫尔辛基是一座三面环海、风景秀丽、大小湖泊星罗棋布的城市；它还是世界闻名的大学城和国际性的政治、文化及会议中心，这里曾举行过多次西方国家的首脑会议。在距市中心约 10 分钟脚程的地方，有一条以卡勒瓦拉命名的大街，街道两旁，19 世纪的旧式住宅和现代化的建筑相映成趣。卡勒瓦拉大街上，有一座大学生寄宿公寓，我们这部史诗的缔造者 Linus Torvalds 就住在这座公寓里。

现在看来，Linux 并非深思熟虑的惊人之作，而是一个逐渐扩展的过程。它综合了许多次的试验、各种各样的概念和一小段一小段的程序，在不知不觉中逐渐凝聚成了一个有机的整体。这个过程与史诗《卡勒瓦拉》的成书过程极为相似。它最初的生成动机应当追溯到 1990 年的秋天。那时的 Linus 正在赫尔辛基大学学习 UNIX 课程，所用的教材是 Andrew Tanenbaum 的《操作系统：设计与实施》。因为在学校上机需要长时间排队等待，于是“一气之下，我干脆自己掏钱买了一台 PC 机”，Linus 回忆说。

Linus 在自己的 PC 机上，利用 Tanenbaum 教授自行设计的微型 UNIX 操作系统 MINIX 为开发平台，开发了属于他自己的第一个程序。“这个程序包括两个进程，都是向屏幕上写字母，然后用一个定时器来切换这两个进程。”他回忆说，“一个进程写 A，另一个进程写 B，所以我就在屏幕上看到了 AAAA，BBBB，如此循环重复的输出结果。”

Linus 说刚开始的时候他根本没有想到要编写一个操作系统内核。1991 年，他需要一个简单的终端仿真程序来存取 Usenet 新闻组的内容，于是他就在前两个草草编写的进程的基础上又写了一个程序。当然，他把那些个 A 和 B 改成了别的东西。“一个进程是从键盘上阅读输入然后发送给调制解调器，另一个进程则是从调制解调器上阅读发送来的信息然后送到屏幕上供人阅读。”然而要实现这两个新的进程，他显然还需要一些别的东西，这就是驱动程序。他必须为不同的显示器、键盘和调制解调器编写驱动程序。1991 年的夏季，也就是在他购买了第一台 PC 之后 6 个月，Linus 觉得他还需要从网上下载某些文件，为此他必须读写某个磁盘。“于是我又不得不写一个磁盘驱动程序，然后是一个文件系统。而一旦当你有了任务切换器、文件系统和设备驱动程序之后，你当然就拥有了一个 UNIX，”或者至少是它的一个内核。Linux 就以这样一种极其古怪但也极其自然的方式问世了。

这个羽毛未丰的操作系统很有可能马上夭折，所以 Linus 并没有在 MINIX 新闻组中公布它。他只是在赫尔辛基技术大学的一台 FTP 服务器上发了一则消息，说用户可以

下载 Linux 的公开版本。“Linux 是我的笔名，但是我要是真用它来命名的话，我担心有人会认为我狂妄自大，而且不会去认真地对待它。所以我当时选了一个很糟糕的名字：Freax，”这个字是由 free（自由）+ freak（怪胎）+ x 构成的，“我知道这听起来令人恶心。”幸好，管理这台 FTP 服务器的 Ari Lemmke 根本不喜欢 Freax 这个名字，他最后还是选择了 Linux。

到 1992 年 1 月止，全世界大约只有 100 个左右的人在使用 Linux，但正是他们为 Linux 做了关键性的在线洗礼。他们所提供的所有初期的上载代码和评论后来证明对 Linux 的发展至关重要，尤为重要的是那些网上黑客们为了解决 Linux 的错误而编写的许多插入代码段。Linus 就是如此这般脚步蹒跚跌跌撞撞地创建了一个网上的“卡勒利亚王国”，并开始为他的“卡勒瓦拉”收集并组织各种有用的素材。网上的任何人在任何地方都可以得到 Linux 的基本文件，并可通过电子邮件发表评论或者提供修正代码，Usenet 还专门为它开辟了一个论坛。于是，Linux 就从最开始的一个人思想的产品变成了一副巨大的织锦，变成了由无数志同道合的黑客们发起的一场运动。

GNU、GPL、Linux

在我们这个世界上流行的软件按其提供方式和是否赢利可以划分为三种模式，即商业软件（Commercial software）、共享软件（Shareware）和自由软件（Freeware 或 Free software）。

商业软件由开发者出售拷贝并提供技术服务，用户只有使用权，但不得进行非法拷贝、发行和修改；共享软件由开发者提供软件试用程序拷贝授权，用户在试用该程序拷贝一段时间之后，必须向开发者交纳使用费用，开发者则提供相应的升级和技术服务；而自由软件则由开发者提供软件全部源代码，任何用户都有权使用、拷贝、发行、修改该软件，同时用户也有义务将自己修改过的程序代码公开。

1984 年，自由软件的积极倡导者 Richard Stallman 组织开发了一个完全基于自由软件的软件体系——GNU，并拟定了一份通用公共许可证（General Public License，简称 GPL）。目前人们已很熟悉的一些软件如 BIND、Perl、Apache 等实际上都是自由软件的经典之作，现在又有了 Netscape 的加盟。可以想象，如果没有了它们，那么 Internet 的真实面貌大概会令人惨不忍睹。

Richard Stallman 像一个神态庄严的传教士一样喋喋不休地到处宣讲自由软件的福音，阐述他创立 GNU 的梦想：“自由的思想，但不是免费的午餐”。然而同是自由软件的积极倡导者，Linus 就显得轻松自在得多，他从不对自由软件应该是什么或者自由软件对于我们有什么样的意义等重大问题妄加评论。但是他却毫不犹豫地把 Linux 奉献给了自由软件，奉献给了 GNU，从而最终使自由软件有了一个发展的根基——基于 Linux 的 GNU。

从本质上讲，Linus 是个理想主义者，但同时他又非常实际。1993 年，Linux 的第一个“产品”版 Linux 1.0 问世的时候，是按完全自由发行版权进行发行的。它要求所有的源码必须公开，而且任何人均不得从 Linux 交易中获利。显然他还记得在他还是个穷学生的时候，由于买不起 UNIX 商业版时的尴尬和苦恼，即使后来他使用的 MINIX 在他看来也仍然太贵。然而半年以后，他开始意识到这种纯粹的自由软件的理想对于 Linux 的发行和发展来说实际上是一种障碍而不是一股推动力，因为它限制了 Linux 以磁盘拷

贝或者 CD-ROM 等媒体形式进行发行的可能，也限制了一些商业公司参与 Linux 的进一步开发并提供技术支持的良好愿望。于是 Linus 决定转向 GPL 版权，这一版权除了规定有自由软件的各项许可权之外，还允许用户出售自己的程序拷贝。

这一版权上的转变后来证明对于 Linux 的进一步发展而言确实极为重要。从此以后，便有多家技术力量雄厚又善于市场运作的商业软件公司加入了原先完全由业余爱好者和网络黑客所参与的这场自由软件运动，开发出了多种 Linux 的发行版本，磨光了纯粹自由软件许多粗糙不平的棱角，增加了更易于用户使用的图形界面和众多的软件开发工具，极大地拓展了 Linux 的全球用户基础。Linus 本人也认为："使 Linux 成为 GPL 的一员是我一生中所做过的最漂亮的一件事"。

举例来说，目前市面上流行的一种"CD-ROM 大餐"就包含了三家商业软件公司即 Red Hat、Slackware 和 Caldera 所包装的 Linux 发行版本，售价仅为 150 美元，内容却极为丰富。比如说 Slackware 的标准 Linux 发行版中包括有以下的内容：操作系统本身、X Free86、X Windows、NTeX、TeX、GNU C 和 C++ 编译器、Objective C、FORTRAN 77、Tcl、TclX、make、bya cc、GNU Bison、flex、C 库、GNU common LISP、TCP/IP 网络、SLIPP/PPP、IP accounting、防火墙、Java 内核支持、BSD 邮件发送、cnews、nn、tin、trn、inn、fvwm95、GNU chess、Apache HTTP server、Arena 和 Lynx Web 浏览器。与直接从 Internet 上下载数百兆字节相比，CD-ROM 版本其实更为便宜，而且安装起来也更为方便快捷。

商业软件公司的加盟也使大多数 Linux 的普通用户吃了定心丸。因为在很多人看来，"可自由发行"的软件好像总是和"缺乏技术支持"以及"业余水平"划等号的，其实不然。Linux 从一开始就主要是在一些软件行业中的高手之间流行的，并且很快就在全球范围内网罗了一大批职业的和业余的技术专家，形成了一个数量庞大而且非常主动热心的支持者群体。它们能够通过网络很快地响应你所遇到的任何问题。举例来说，当 Pentium II 设计上的 Bug 刚一被发现，Linux 是最早一个提供了解决方案的操作系统。1997 年，Linux 支持者群体在众多的软件公司中一举胜出，荣获了美国《InfoWorld》杂志的最佳技术支持奖，而这一奖项原本只是为商业公司而设立的。

但是不管怎么说，商业软件公司所提供的技术支持总是显得更为正规一些。比如 Caldera 可以和用户签订一年 1500 美元或者一次 60 美元的技术支持合同；Workgroup Solutions 的支持合同是一年 1,000 美元或一小时 150 美元或一次 50 美元；Red Hat 的 AnswerDesk 则可提供每天 24 小时，每周七天的电话支持，用户可以用信用卡按小时或按分钟付费。这些正规的技术支持服务对于把 Linux 更快地推向企业计算领域无疑是大有帮助的。

NT、UNIX、Linux

下面这则故事是一个真实的故事。1996 年度，美国林肯州内布拉斯加普雷斯大学系统部准备把他们部门中一套已经陈旧过时的 NetWare 服务器更换掉，另外安装一套新的操作系统。系统部经理 Quinn Coldiron 当时的首选目标不用说当然是 Windows NT 4.0，然而他没有料到，在安装了 NT 之后，竟会遇到那么多难以解决的麻烦，多次打电话寻求技术支持又让学校破费了很多钱。万般无奈之际，他决定试用一下 Linux，结果却令他大感意外。从 1997 年 1 月至 7 月，他们部门的 Linux 服务器仅在意外情况下当过三

次，两次是因为楼里的电源线路发生故障，另外一次则是因为操作人员的愚笨所致。更令他觉得不可思议的是，在原先那套旧的 NetWare 系统上，如果同时有五个用户登录做专业出版，系统就会崩溃；而现在，还是同样的硬件（256M 内存、2 块 150MHz CPU），但是 Linux 却可以轻松自如地支持 40 个用户同时登录做专业出版。于是他在鉴定白皮书中按耐不住兴奋地说："Linux 服务器已经证明和我所用过的其他服务器操作系统同样地可靠，而且要比其中的大多数操作系统更为可靠。"

Linux 的神奇之处不仅在于它可免费获得和它所发起的声势浩大的软件运动，更在于它本身强大的性能、卓越的稳定性和众多的功能。Linux 刚开始的时候主要是为低端 UNIX 用户而设计的，它可以使很多已经过了时的硬件重新焕发青春。它在只有 4M 内存的 Intel 386 处理器上就能非常好地运行，而这类机器即使用 Windows 3.x 也很难进行较好地管理。随着 Linux 用户基础的不断扩大、性能的不断提高、功能的不断增加、各种平台版本的不断涌现，以及越来越多商业软件公司的加盟，Linux 已经在不断地向高端发展，开始进入越来越多的公司和企业计算领域。虽然到目前为止，还没有哪家公司肯将它的全部信息系统建立在 Linux 上，但是 Linux 已经在很多企业计算领域中大显身手。

据从事 Linux 开发的 Red Hat 软件公司说，他们公司现在已拥有了许多第一流的企业用户和团体用户，其中包括 NASA、迪斯尼、洛克希德、通用电气、波音、Ernst & Young、UPS、IRS、Nasdaq，以及多家美国一流的大学机构等。Red Hat 公司的总裁 Robert Young 认为，Linux 最大的单项应用是 Internet 和 Intranet 服务器，"从防火墙到 Web 服务器，据分析家估计，Linux 已成为网上的第二大通用操作系统。" Linux 的其他应用从打印服务器到 FTP 服务器到实时的数据收集等应有尽有，目前在网上应用最多的 Apache Web 服务器也已成为各种标准的 Linux 发行版的一个部分。

即使作为一种台式机操作系统，与许多用户非常熟悉的 UNIX 相比，它的性能也显得更为优秀。一台 Linux 服务器支持 100 到 300 个用户毫无问题，一台 Linux 打印服务器支持 200 到 300 台网络打印客户更是易如反掌。而且它不大在意 CPU 的速度，它可以把每种处理器的性能发挥到极限，到时候用户就会发现，影响系统性能提高的限制因素主要是其总线和磁盘 I/O 的性能。正如一些分析家所指出的，Linux 已经成为 UNIX 市场大饼中一个重要的非常具有活力的不断扩大的一角。

但是 Linux 如何更有效地争取商业市场的支持和信任仍然是它发展中最为关键的问题。导致这一问题出现的因素主要有以下几点：

一、商业市场中的多数人仍然认为 Linux 是一种由业余爱好者及网络黑客们所开发的软件，这一点因为有不少的商业软件公司加入 Linux 的开发队伍而正在得到改善。

二、有不少人认为，由于 Linux 缺乏台式应用软件而断定它不可能进入主流操作系统行列。这一点目前也已有了很大改观，已有多家软件公司向 Linux 提供了各种性能强大的台式应用，如 Applix 和 Star 公司提供了数种字处理、电子表格、图形应用等程序；Corel WordPerfect 7、Adabas D 和 Raima Database Manager++ 数据库、Netscape Navigator 3.0 和 Fast—Track Web 服务器、Adobe Acrobat PDF 阅读器、FreeBuilder 等的 Linux 版均已问世；甚至连微软这样的软件业巨人也正在准备推出其分布式计算标准 DCOM 的 Linux 版。

三、由于 Linux 本身独具的这种分布式开发模式，有人认为它最终会乱成一团。Linux 就象《卡勒瓦拉》一样，由最初的约 10,000 行程序经过全球网络上数不清的编程人员的

不断添加，目前的规模已达 100 万行左右；由 Linus 本人所控制的主要版本现已达到 2.2 版，而由各家商业软件公司所自行开发的发行版本更是不计其数。如何对这种开发模式进行有效地控制和管理、减少软件本身不必要的膨胀，确实是决定 Linux 未来发展的一个关键性问题。

四、最后是微软。虽然 Linux 在 UNIX 市场上已成为佼佼者，但它在与微软的对抗中到底能够坚持多久？微软已经扼杀掉了许多新生的操作系统，Linux 会不会重蹈覆辙？对这一点业界中人看法不一，不过多数人认为 Linux 大有希望。其一是因为 Linux 1.0 从一出世起就是一个多少已经完全成型、性能良好、功能齐备的操作系统，这是与那些在经历痛苦而缓慢的成长过程时就需要抵御微软杀手锏的操作系统不同的。其二是在 Linux 的背后凝聚了 UNIX 过去所有的成功并抛弃了其所有缺陷后所形成的 UNIX 的全部精华，这种资源优势要比微软投入 NT 的资源优势大得多。

Linus 本人在有人问到他会不会对微软构成威胁时不无揶揄地说："我根本没有打算威胁微软，主要是因为我根本没有把微软真正视为对手。尤其没有把 Windows 视为对手——因为 Linux 和 Windows 的目标完全不一样。至于说到 Windows NT，我曾经对它发生过兴趣，但是我越深入进去就越发现它不过是一个带有较稳定内核的传统的 Windows 而已。我从中找不到任何技术上令人感兴趣的东西。依我看，微软做得更多的是怎么去挣钱，而不是去制作一个更好的操作系统。"对于 Linux 的未来，Linus 也充满信心："Linux 一直就是最棒的。我对 Linux 的未来确实一点儿都不担心，因为从技术方面看，Linux 肯定会越变越好；而从非技术方面看，我个人也看不出有什么需要担忧的。"

我们也衷心希望 Linux 在前途未卜的茫茫大洋上不要像泰坦尼克号那样去完结一个辉煌的沉没，而是去开创一个灿烂的新生！

附录 B

专业术语中英文对照表

下面分类给出了本书中出现的专业术语之中英文对照。

A

Accept	接受
Access permission	访问许可
Acknowledgement	确认
Adapter	适配器
Address	地址
Address resolution	地址解析
Address space	地址空间
Aging algrithm	寿命算法
Allocation	分配
Architecture	体系结构
Assembly language	汇编语言
Atomic action	原子行为
Attribute	属性
Authentication	验证

B

Backup	备份
Bit map	位图
Block device	块设备
Block group	块组
Block group descriptor	块组描述符
Block special file	块特殊文件
Boot block	引导块
Boot loader	引导装载器
Boot record	引导记录
Boot sector	引导扇区
Booting	引导
Bootstrap	引导程序
Bottom half handling	底半处理
Buffer cache	缓冲区缓存
Bus	总线

C

Cache	高速缓存，缓存
Catching signal	捕获信号
Character device	字符设备
Character special file	字符特殊文件
Child process	子进程
Client-server	客户/服务器
Clock	时钟
Clock interrupt	时钟中断
Clock tick	时钟滴答
Command interpreter	命令解释器
Connection	连接
Context	上下文，环境
Context switch	上下文切换
Controller	控制器
Copy-on-writing	写时复制
Critical region	关键区域
Critical section	关键段
Current directory	当前目录
Cylinder	柱面

D

Daemon	守护进程
Data segment	数据段
Deadlock	死锁
Deallocation	释放
Demanding paging	需求分页
Device controller	设备控制器

Device driver	设备驱动程序
Device file	设备文件
Direct memory access	直接内存访问
Directory	目录
Directory tree	目录树
Dirty	脏
Disk	磁盘
Disk hardware	磁盘硬件
Disk partition	磁盘分区
Diskette	软盘
DMA	直接内存访问
Domain	域
Domain name	域名
Dot clock	点时钟

E

Error	错误
Error handling	错误处理
Escape character	转义字符
Escape sequence	转义序列
Exception	异常
Executable image	可执行映象
Extended partition	扩展分区

F

FIFO	先进先出
File	文件
File access	文件存取
File attribute	文件属性
File backup	文件备份
File descriptor	文件描述符
File locking	文件锁定
File naming	文件命名
File operation	文件操作
File position	文件位置
File structure	文件结构
File system	文件系统
File type	文件类型
First-in First-out	先进先出
Floppy disk	软盘
Font	字体
Formatting	格式化
Forwarding information database	转发信息数据库
Fragmentation	碎片化
Free block	空闲块
Full backup	完全备份
Function key	功能键

G

General Public License	通用公共许可证

H

Halt	停止、终止
Handler	处理程序，处理过程
Hard disk	硬盘
Hard Link	硬链接
Hash table	哈希表
Header files	头文件
Hierarchical directories	层次化目录

I

I/O	输入输出
I/O adapter	I/O 适配器
I/O device	I/O 设备
I/O device controller	I/O 设备控制器
IDE disk	IDE 磁盘
Incremental backup	增量备份
Index node (inode)	索引节点
Indirect block	间接块
Initialization	初始化
Input / Output	输入/输出
Interleaving	交错
Interprocess communication	进程间通讯
Interrupt	中断
Interrupt descriptor table	中断描述符表
Interrupt handler	中断处理程序
Interrupt request	中断请求
Interrupt vector	中断向量
IPC	进程间通讯
IRQ	中断请求

J

Job	作业
Job control	作业控制

K

Kernel	内核
Kernel call	内核调用
Kernel mode	内核模式
Keyboard	键盘
Keyboard layout	键盘布局
Keymap	键映射
Kill	杀掉，删除

L

LBA	线性块寻址
Least recently used algorithm	最近最少使用算法
Lightweight process	轻量级进程
Linear address	线性地址
Linear block addressing	线性块寻址
Link to a file	到某文件的链接
Linked list	链表
Linux	Linux
Listening	监听
Loading module	装载模块
Lock	琐
Logical partition	逻辑分区
Login	登录
Logout	注销
Loopback	回环
LRU	最近最少使用算法

M

Magic number	幻数
Mailbox	邮箱
Major device number	主设备号
Massage queue	消息队列
Master boot record	主引导记录
Memory	内存，存储器
Memory management	内存管理
Memory manager	内存管理程序
Memory mapping	内存映射
Memory swapping	内存交换
Microkernel operating system	微内核操作系统
MINIX	MINIX
Minor device number	次设备号
Mirroring	镜像
Module	模块
Monitor	监视器
Monolithic operating system	单块操作系统
Mount	挂装，挂上
Mount point	挂装点
Mounted file system	已挂装文件系统
Mouse protocol	鼠标协议
MS-DOS	MS-DOS
Multilevel page table	多级页表
Multiprocessor	多处理器
Multiprogramming	多道程序设计方法
Mutual exclusion	互斥
Mutual exclusion condition	互斥条件

N

Named pipe	命名管道
Network	网络
Network device	网络设备
Nonpreemptive scheduling	非抢先式调度

O

Operating system	操作系统
Orphan	孤儿进程

P

Packet	数据包
Page	内存页
Page block	页块
Page fault	页故障
Page frame	页帧
Page table	页表
Paging	分页
Panic	应急
Parent process	父进程
Partition	分区
Partition table	分区表
Password	密码，口令
Path name	路径名
Physical memory	物理内存
Pipe	管道

Pixel	象素
Pointer	指针
Policy	策略
Port	端口
POSIX	可移植操作系统接口
Preemptive scheduling	抢先式调度
Primary partition	主分区
Printer daemon	打印机守护进程
Priority	优先级
Priority scheduling	优先级调度
Privilege level	特权级
Process	进程
Process group	进程组
Process hierarchy	进程层次
Process management	进程管理
Process scheduling	进程调度
Process state	进程状态
Process switch	进程切换
Process table	进程表
Process tree	进程树
Processor	处理器
Producer-consumer problem	生产者－消费者问题
Programm	程序
Protected mode	保护模式
Protocol	协议
Pseudoterminal	伪终端

R

Random access file	随机访问文件
Raw mode	原始模式
Real-time scheduling	实时调度
Real-time system	实时系统
Reassembly	重组
Receive	接收
Regular expression	规则表达式
Repeat delay	重复延迟
Repeat rate	重复率
Resource	资源
Root directory	根目录
Root file system	根文件系统
Round robin scheduling	循环赛调度
Routine	例程
Routing	路由
Routing cache	路由缓存
Run level	运行级别

S

Scheduler	调度程序
Scheduling algorithm	调度算法
Scheduling mechanism	调度机制
Scheduling policy	调度策略
Script file	脚本文件
SCSI device	SCSI 设备
Sector	扇区
Security	安全性
Segment	段
Semaphore	信号量
Server process	服务器进程
Session	会话
Session leader	会话领头进程
Shared memory	共享内存
Shell	命令解释程序
Shutdown	关机，当机
Signal	信号
Single user mode	单用户模式
Sleeping	休眠
Socket	套接字
Socket address family	套接字地址族
Socket buffer	套接字缓冲区
Socket type	套接字类型
Special file	特殊文件
Spooling	假脱机
Spooling directory	假脱机目录
Standard input	标准输入
Standard output	标准输出
Subnet	子网
Subnet mask	子网掩码
Super-block	超块
Super-user	超级用户
Suspend	挂起
Swapping	交换
Swapping space	交换空间
Symbol link	符号链接
Synchronization	同步
System call	系统调用

T

Task	任务
Task queue	任务队列
Terminal	终端
Thread	线程
Timer	定时器
Timesharing	分时
Time-slice	时间片
Transmission	传输

U

UNIX	UNIX
Unloading module	卸载模块
Unmount	卸挂，卸下
User accout	用户帐号
User authentication	用户验证
User group	用户组
User mode	用户模式

V

Vector	向量
Video controller	视频控制器
Video RAM	视频 RAM
Virtual address	虚拟地址
Virtual address space	虚拟地址空间
Virtual console	虚拟控制台
Virtual file system	虚拟文件系统
Virtual memory	虚拟内存
Virus	病毒

W

Waiting queue	等待队列
Wakeup	唤醒
Watchdog timer	看门狗定时器
Window manager	窗口管理器
Working directory	工作目录

X

X client	X 客户
X server	X 服务器
X terminal	X 终端
X Window	X Window

Z

Zombie state	僵死状态

[General Information]
书名=Linux 实用教程
作者=魏永明 杨飞月 吴漠霖
页数=396
SS号=10028750
出版日期=1999年5月第1版