第1章 Linux 简介

让用户很详细地了解现有操作系统的实际工作方式是不可能的，因为大多数操作系统的源代码都是严格保密的。其例外是一些研究用的系统，另外一些是明确为操作系统教学而设计的系统。（还有一些系统则是同时出于这两种目的。）尽管研究和教学这两个目的都很好，但是这类系统很少能够通过对正式操作系统的小部分实现来体现操作系统的实际功能。对于操作系统的一些特殊问题，这种折衷系统所能够表现的就更是少得可怜了。

在以实际使用为目标的操作系统中，让任何人都可以自由获取系统源代码，无论目的是要了解、学习还是改进，这样的现实系统并不多。本书的主题就是这些少数操作系统中的一个：Linux。

Linux 的工作方式类似于 Uinx，是免费的，源代码也是开放的，符合标准规范的 32 位（在 64 位 CPU 上是 64 位）操作系统。Linux 拥有现代操作系统的所具有的内容，例如：

- 真正的抢先式多任务处理，支持多用户
- 内存保护
- 虚拟内存
- 支持对称多处理机 SMP（symmetric multiprocessing），即多个 CPU 机器，以及通常的单 CPU（UP）机器
- 符合 POSIX 标准
- 联网
- 图形用户接口和桌面环境（实际上桌面环境并不只一个）
- 速度和稳定性

严格说来，Linux 并不是一个完整的操作系统。当我们在安装通常所说的 Linux 时，我们实际安装的是很多工具的集合。这些工具协同工作以组成一个功能强大的实用系统。Linux 本身只是这个操作系统的内核，是操作系统的心脏、灵魂、指挥中心。（整个系统应该称为 GNU/Linux，其原因在本章的后续内容中将会给以介绍。）内核以独占的方式执行最底层任务，保证系统正常运行——协调多个并发进程，管理进程使用的内存，使它们相互之间不产生冲突，满足进程访问磁盘的请求等等。

在本书中，我们给大家揭示的就是 Linux 是如何完成这一具有挑战性的工作的。

Linux（和 Unix）的简明历史

为了让大家对本书所讨论的内容有更清楚的了解，让我们先来简要回顾一下 Linux 的历史。由于 Linux 是在 Unix 的基础上发展而来的，我们的话题就从 Unix 开始。

Unix 是由 AT&T 贝尔实验室的 Ken Thompson 和 Dennis Ritchie 于 1969 年在一台已经废弃了的 PDP-7 上开发的；它最初是一个用汇编语言写成的单用户操作系统。不久，Thompson 和 Ritchie 成功地说服管理部门为他们购买更新的机器，以便该开发小组可以实现一个文本处理系统，Unix 就在 PDP-11 上用 C 语言重新编写（发明 C 语言的部分目的就在于此）。它果真变成了一个文本处理系统——不久之后。只不过问题是他们先实现了一个操作系统而已…

最终，他们实现了该文本处理工具，而且 Unix（以及 Unix 上运行的工具）也在 AT&T 得到广泛应用。在 1973 年，Thompson 和 Ritchie 在一个操作系统会议上就这个系统发表了一篇论文，该论文引起了学术界对 Unix 系统的极大兴趣。

由于 1956 年反托拉斯法案的限制，AT&T 不能涉足计算机业务，但允许它可以以象征性的费用发售该系统。就这样，Unix 被广泛发布，首先是学术科研用户，后来又扩展到政府和商业用户。

伯克利（Berkeley）的加州大学是学术用户中的一个。在这里 Unix 得到了计算机系统研究小组（CSRG）的广泛应用。并且在这里所进行的修改引发了 Unix 的一大系列，这就是广为人知的伯克利软件开发（BSD）Unix。除了 AT&T 所提供的 Unix 系列之外，BSD 是最有影响力的 Unix 系列。BSD 在 Unix 中增加了很多显著特性，例如 TCP/IP 网络，更好的用户文件系统（UFS），工作控制，并且改进了 AT&T 的内存管理代码。

多年以来，BSD 版本的 Unix 一直在学术环境中占据主导地位，但最终发展成为 System V 版本的 AT&T 的 Unix 则成为商业领域的主宰。从某种程度上来说，这是有社会原因的：学校倾向于使用非正式但通常更好用的 BSD 风格的 Unix，而商业界则倾向于从 AT&T 获取 Unix。

在用户需求驱动和用户编程改进特性的促进下，BSD 风格的 Unix 一般要比 AT&T 的 Unix 更具有创新性，而且改进也更为迅速。但是，在 AT&T 发布最后一个正式版本 System V Release 4（SVR4）时，System V Unix 已经吸收了 BSD 的大多数重要的优点，并且还增加了一些自己的优势。这种现象的部分原因在于从 1984 年开始，AT&T 逐渐可以将 Unix 商业化，而伯克利 Unix 的开发工作在 1993 年 BSD4.4 版本完成以后就逐渐收缩以至终止了。然而，BSD 的进一步改进由外界开发者延续下来，到今天还在继续进行。正在进行的 Unix 系列开发中至少有四个独立的版本是直接起源于 BSD4.4，这还不包括几个厂商的 Unix 版本，例如惠普的 HP-UX，都是部分地或者全部地基于 BSD 而发展起来的。

实际上 Unix 的变种并不止 BSD 和 System V。由于 Unix 主要使用 C 语言来编写，这就使得它相对比较容易地移植到新的机器上，它的简单性也使其相对比较容易重新设计与开发。Unix 的这些特点大受商业界硬件供应商的欢迎，比如 Sun、SGI、惠普、IBM、DEC（数字设备公司）、Amdahl 等等；IBM 还不止一次对 Unix 进行了再开发。厂商们设计开发出新的硬件并简单地将 Unix 移植到新的硬件上，这样新的硬件一经发布便具备一定的功能。经过一段时间之后，这些厂商都拥有了自己的专有 Unix 版本。而且为了占有市场，这些版本故意以不同的侧重点发布出来以更好的占有用户。

版本混乱的状态促进了标准化工作的进行。其中最主要的就是 POSIX 系列标准，它定义了一套标准的操作系统接口和工具。从理论上说，POSIX 标准代码很容易移植到任何遵守 POSIX 标准的操作系统中，而且严格的 POSIX 测试已经把这种理论上的可移植性转化为现实。直到今天，几乎所有的正式操作系统都以支持 POSIX 标准为目标。

现在让我们回顾一下，在 1984 年，杰出的电脑黑客 Richard Stallman 独立开发出一个类 Unix 的操作系统，该操作系统具有完全的内核、开发工具和终端用户应用程序。在 GNU（“GNU’s Not Unix”首字母的缩写）计划的配合下，Stallman 开发这个产品有自己的技术理想：他想开发出一个质量高而且自由的操作系统。Stallman 使用了“自由”（free）这个词，不仅意味着用户可以免费的获取软件；而且更重要的是，它将意味着某种程度的“解放”：用户可以自由使用、拷贝、查询、重用、修改甚至是分发这份软件，完全没有软件使用协议的限制。这也正是 Stallman 创建自由软件基金会（FSF）资助 GNU 软件开发的本意（FSF 也在资助其它科研方面的开发工作）。

15 年以来，GNU 工程已经吸收、产生了大量的程序，这不仅包括 Emacs，gcc（GNU 的 C 编译器），bash（shell 命令），还有大部分 Linux 用户所熟知的许多应用程序。现在正在进行开发的项目是 GNU Hurd 内核，这是 GNU 操作系统的最后一个主要部件（实际上 Hurd 内核早已能够使用了，不过当前的版本号为 0.3 的系统在什么时候能够完成，还是未知数）。

尽管 Linux 大受欢迎，但是 Hurd 内核还在继续开发。这种情况的原因有几个方面，其

一是 Hurd 的体系结构十分清晰的体现了 Stallman 关于操作系统工作方式的思想，例如，在运行期间，任何用户都可以部分的改变或替换 Hurd（这种替换不是对每个用户都是可见的，而是只对申请修改的用户可见，而且还必须符合安全规范）。另一个原因是据介绍 Hurd 对于多处理器的支持比 Linux 本身的内核要好。还有一个简单的原因是兴趣的驱动，因为程序员们希望能够自由地进行自己所喜欢的工作。只要有人希望为 Hurd 工作，Hurd 的开发就不会停止。如果他们能够如愿以偿，Hurd 有朝一日将成为 Linux 的强劲对手。不过在今天，Linux 还是自由内核王国里无可争议的主宰。

在 GNU 发展的中期，也就是 1991 年，一个名叫 Linus Torvalds 的芬兰大学生想要了解 Intel 的新 CPU——80386。他认为比较好的学习方法是自己编写一个操作系统的内核。出于这种目的，加上他对当时 Unix 变种版本对于 80386 类机器的脆弱支持十分不满，他决定要开发出一个全功能的、支持 POSIX 标准的、类 Unix 的操作系统内核，该系统吸收了 BSD 和 System V 的优点，同时摒弃了它们的缺点。Linus（虽然我知道我应该称他为 Torvalds，但是所有人都称他为 Linus）独立把这个内核开发到 0.02 版，这个版本已经可以运行 gcc，bash 和很少的一些应用程序。这些就是他开始的全部工作了。后来，他又开始在因特网络上寻求广泛的帮助。

不到三年，Linus 的 Unix—Linux—已经升级到 1.0 版本。它的源代码量也呈指数形式增长，实现了基本的 TCP/IP 功能（网络部分的代码后来重写过，而且还可能会再次重写）。此时 Linux 就已经拥有大约 10 万用户了。

现在的 Linux 内核由 150 多万行代码组成，Linux 也已经拥有了大约 1000 万用户（由于 Linux 可以自由获取和拷贝，获取具体的统计数字是不可能的）。Linux 内核 GNU/Linux 附同 GNU 工具已经占据了 Unix 50%的市场。一些公司正在把内核和一些应用程序同安装软件打包在一起，生产出 Linux 的 distribution（发行版本），这些公司包括 Red Hat 和 Calera prominent 公司。现在的 GNU/Linux 已经备受注目，得到了诸如 Sun、IBM、SGI 等公司的广泛支持。SGI 最近决定在其基于 Intel 的 Merced 的系列机器上不再搭载自己的 Unix 变种版本 IRIX，而是直接采用 GNU/Linux；Linux 甚至被指定为 Amiga 将要发布的新操作系统的基础。

GNU 通用公共许可证

这样一个如此流行大受欢迎的操作系统当然值得我们学习。按照通用公共许可证(GPL，（General Public License）)的规定，Linux 的源代码可以自由获取，这使得我们学习该系统的强烈愿望得以实现。GPL 这份非同寻常的软件许可证，充分体现了上面提到的 Stallman 的思想：只要用户所做的修改是同等自由的，用户可以自由地使用、拷贝、查询、重用、修改甚至重新发布这个软件。通过这种方式，GPL 保证了 Linux（以及同一许可证保证下的大量其它软件）不仅现在自由可用，而且以后经过任何修改之后都仍然可以自由使用。

请注意这里的自由并不是说没有人靠这个软件盈利，有一些日益兴起的公司，比如发行最流行的 Linux 发行版本的 Red Hat，就是一个例子。（Red Hat 自从面世以来，市值已经突破数十亿美元，每年盈利数十万美元，而且这些数字还在不断增长）。但是任何人都不能限制其它用户涉足本软件领域，而且所作的修改不能减少其自由程度。

本书的附录 B 中收录有 GNU 通用公共许可证协议的全文。

Linux 开发过程

如上所述，由于 Linux 是一款自由软件，它可以免费获取以供学习研究。Linux 之所以值得学习研究，是因为它是相当优秀的操作系统。如果 Linux 操作系统相当糟糕，那它就根本不值得被我们使用，也就没有必要去研究相关的书籍。（除非一种可能，为了追求刺激）。Linux 是一款十分优秀的操作系统还在于几个相互关联的原因。

Linux 优秀的原因之一在于它是基于天才的思想开发而成的。在学生时代就开始推动整个系统开发的 Linus Torvads 是一个天才，他的才能不仅展现在编程能力方面，而且组织技巧也相当杰出。Linux 的内核是由世界上一些最优秀的程序员开发并不断完善的，他们通过 Internet 相互协作，开发理想的操作系统；他们享受着工作中的乐趣，而且也获得了充分的自豪感。

Linux 优秀的另外一个原因在于它是基于一组优秀的概念。Unix 是一个简单却非常优秀的模型。在 Linux 创建之前，Unix 已经有 20 年的发展历史。Linux 从 Unix 的各个流派中不断吸取成功经验，模仿 Unix 的优点，抛弃 Unix 的缺点。这样做的结果是 Linux 成为了 Unix 系列中的佼佼者：高速、健壮、完整，而且抛弃了历史包袱。

然而，Linux 最强大的生命力还在于其公开的开发过程。每个人都可以自由获取内核源程序，每个人都可以对源程序加以修改，而后他人也可以自由获取你修改后的源程序。如果你发现了缺陷（bug），你可以对它进行修正，而不用去乞求不知名的公司来为你修正。如果你有什么最优化或者新特点的创意，你也可以直接在系统中增加功能，而不用向操作系统供应商解释你的想法，指望他们将来会增加相应的功能。当发现一个安全漏洞后，你可以通过编程来弥补这个漏洞，而不用关闭系统直到你的供应商为你提供修补程序。由于你拥有直接访问源代码的能力，你也可以直接阅读代码来寻找缺陷，或是效率不高的代码，或是安全漏洞，以防患于未然。

除非你是一个程序员，否则这一点听起来仿佛没有多少吸引力。实际上即使你不是程序员，这种开发模型也将使你受益匪浅，这主要体现在以下两个方面：

- 可以间接受益于世界各地成千上万的程序员随时进行的改进工作。
- 如果你需要对系统进行修改，你可以雇用程序员为你完成工作。这部分人将根据你的需求定义单独为你服务。可以设想，这在源程序不公开的操作系统中它将是什么样子。

Linux 这种独特的自由流畅的开发模型已被命名为 bazaar（集市模型），它是相对于 cathedral（教堂）模型而言的。在 cathedral 模型中，源程序代码被锁定在一个保密的小范围内。只有开发者（很多情况下是市场）认为能够发行一个新版本，这个新版本才会被推向市场。这些术语在 Eric S. Raymond 的 The Cathedral and the Bazaar 一文中有所介绍，大家可以在 **http://www.tuxedo.org/~esr/writings/**找到这篇文章。Bazaar 开发模型通过重视实验，征集并充分利用早期的反馈，对巨大数量的脑力资源进行平衡配置，可以开发出更优秀的软件。（顺便说一下，虽然 Linux 是最为明显的使用 bazaar 开发模型的例子，但是它却远不是第一个使用这个模型的系统。）

为了确保这些无序的开发过程能够有序地进行，Linux 采用了双树系统。一个树是稳定树（stable tree），另一个树是非稳定树（unstable tree）或者开发树（development tree）。一些新特性、实验性改进等都将首先在开发树中进行。如果在开发树中所做的改进也可以应用于稳定树，那么在开发树中经过测试以后，在稳定树中将进行相同的改进。按照 Linus 的观点，一旦开发树经过了足够的发展，开发树就会成为新的稳定树，如此周而复始的进行下去。

源程序版本号的形式为 x.y.z。对于稳定树来说，y 是偶数；对于开发树来说，y 比相应

的稳定树大一（因此，是奇数）。截至到本书截稿时，最新的稳定内核版本号是 2.2.10，最新的开发内核的版本号是 2.3.12。对 2.3 树的缺陷修正会回溯影响（back-propagated）2.2 树，而当 2.3 树足够成熟的时候会发展成为 2.4.0。（顺便说一下，这种开发会比常规惯例要快，因为每一版本所包含的改变比以前更少了，内核开发人员只需花很短的时间就能够完成一个实验开发周期。）

http://www.kernel.org 及其镜像站点提供了最新的可供下载的内核版本，而且同时包括稳定和开发版本。如果你愿意的话，不需要很长时间，这些站点所提供的最新版本中就可能包含了你的一部分源程序代码。