第1 章 简介

Linux 具有诱人的魅力，它是一个由全世界不同民族、不同信仰、不同性别的人共同参与和协作的国际性项目。免费提供源代码 以及容易理解的类 UNIX 应用程序编程环境，促成了 Linux 的重大成功。 通过因特网从 专家 处即时获得的 高 质量的免费 支持发挥了重要 的 作用， 这促使了一个 庞大的 Linux 社区 的形成 。

由于在技术方面 开发 人员可以获得所有源码，并由此给出一些 创新方案 ，因此他们感到无比振奋 。 例如， 你可以 hack[[ ① ]](http://blog.donews.com/21cnbao/archive/2008/09/03/1342283.aspx" \l "_ftn1) Linux 的源码，并对其进行 定制 ，以让其在你的设备上于 几秒钟 之内启动 ， 而 使用一个专有操作 系统 则 很 难完成这样的 壮举。

1.1 演化

Linux 于 1991 年起源于一位名为 Linus Torvalds 的芬兰大学生的业余爱好，但很快就发展成为受全球欢迎的先进的操作系统。 Linux 第一次发布时仅支持 Intel 386 处理器，但是后来，内核在复杂性上逐步增加，可以支持众多的体系结构、多处理机系统和高性能集群。 Linux 支持很多 CPU ，主要支持的一些体系结构有 x86 、 IA64 、 ARM 、 PowerPC 、 Alpha 、 s390 、 MIPS 和 SPARC 。 Linux 已经被移植到成千上万个基于这些处理器的硬件平台之上。与此同时，内核还在不断完善，并以飞快的脚步发展。

虽然开始的时候只是一个桌面操作系统，但目前 Linux 已经进入嵌入式和企业领域，并渗入我们的日常生活。当你按掌上电脑的按键、调节你的遥控器到天气频道或者在医院接受体检的时候，很有可能某些 Linux 代码正在为你提供服务。一方面， Linux 有很大的技术优势，另一方面， Linux 可以免费获得，这两方面对它的变革发挥着重大的作用。由于能够降低消费类电子产品的价格， Linux 已成为该领域一个很好的选择，因为专有操作系统的价格有时候比硬件本身的价格还贵。

1.2 GNU copyleft

GNU 项目（ GNU 是 GNU ’ s Not UNIX 的递归缩写，即 GNU 不是 UNIX ） 比 Linux 诞生得更早，它 发起 的目标是定制 一个免费的 类 UNIX 操作系统。一个完整的 GNU 操作系统包含 Linux 内核，但也包含 一些其他 组件，如 库、 编译器和 实用工具（ utility ） 。 因此， 基于 Linux 的计算机 的 更准确 称呼应该是 GNU / Linux 系统。 GNU / Linux 系统 的 所有组成部分 都 建立 在 自由软件 之上 。

自由软件 的种类有很多， 其中一种是 公共领域 （ public domain ） 的软件。公共领域 发布的 软件没有版权， 对它的使用也不会强加 任何限制。 你 可以免费使用它 ，随意修改 它，甚至限制 你修改后代码的发布 。 你会发现 ， “ 没有限制 ”在下行过程中反而引入了限制 。

GNU 项目 的主要支撑者—— 自由软件基金会，创造了 GNU 公共许可 证 （ GNU Public License ， GPL ）， 它 也 被 称为 copyleft[[ ② ]](http://blog.donews.com/21cnbao/archive/2008/09/03/1342283.aspx" \l "_ftn2) ， 以 防止有人中途将 自由软件 转化为专有软件 。 如果某人 修改 了 copyleft 的软件 ，就必须以 copyleft 的方式分享他的软件 。 GNU 系统中的 Linux 内核 以及像 GNU 编译器（ GNU Compiler Collection ， GCC ） 等大部分组件都以 GPL 发布 。 因此，如果你修改 了 内核， 你就必须在社区分享此修改 。 基本上，你必须以 copyleft 的形式将授予你的权利传递出去。

Linux 内核 基于 GPL 第2 版。 在内核社区，人们一直在争论是否应该支持GPL 的最新版本GPL v3 。目前的 趋势似乎是反对采取GPL v3 。

通过 系统调用访问内核服务的 Linux 应用 程序 没有 被看作 衍生的工作， 因此并不受限于 GPL 。 而库则采用 GNU 轻量级 通用公共许可证 （ GNU Lesser General Public License ， LGPL ） ，其限制要少于 GPL 。 专有软件也 允许 与 LGPL 下的库 动态链接 。

1.3 kernel.org

Linux 内核源代码的主要存放仓库是 www.kernel.org 。该网站包含所有已经发布的内核版本，世界上许多地方都有 kernel.org 的镜像网站。

除了已经发布的版本以外， kernel.org 还包含了由一线开发人员提供的补丁，这些补丁可以作为未来稳定版本的试验床（ test bed ）。补丁是一种文本文件，其包含了开发树中原始版本和开发人员提供的新版本之间的源码差异。由 Linux 内核维护领袖 Andrew Morton 定期提供的 -mm 补丁是一种流行的补丁（可在 kernel.org 上获得），在该补丁中，我们可以找到在主线源代码树中尚未提供的实验性的功能。 kernel.org 上另一个会定期公布的补丁是由 Ingo Molnar 维护的 -rt （ real time ，实时）补丁。 -rt 补丁的一些功能已经被融入 Linux 内核主线中。

1.4 邮件列表和论坛

Linux 内核邮件列表 （ Linux Kernel Mailing List ， LKML ）是开发人员就设计问题进行辩论并决定 Linux 未来要包含的功能的论坛。你可以在 [www.lkml.org](http://www.lkml.org/) 看到实时的邮件列表。 Linux 内核目前包含由遍布于世界各地的成千上万的开发人员贡献的数百万行代码，正是 lkml 将他们连在一起。

LKML 并不是为了解答一般的 Linux 问题，其基本规则是只能张贴与内核有关的、以前没有被回答过并且在众所周知的文档中没有提及的问题。如果在你编译 Linux 应用程序的时候 C 编译器崩溃了，你应该在其他地方张贴这样的问题。

LKML 中讨论的一些问题线索甚至比畅销的《纽约时报》更有趣，花几个小时浏览 LKML 的压缩包将有助于你洞察 Linux 内核背后的理念。

内核的大部分的子项目都拥有自己的邮件列表。因此，如果你正在开发 Flash 设备的驱动，就可以订阅 linux-mtd 邮件列表；如果你发现了 Linux USB 存储设备驱动的 bug ，就可以在 linux-usb-devel 邮件列表发起一个线索。在以后的几章中，我们就参考了相关的邮件列表。

在各种论坛上，来自世界各地的内核专家会聚集于同一个屋檐下共同商讨 Linux 技术。加拿大渥太华每年举行一次的 Linux Symposium 就是这样的一个会议。其他的还包括在德国举行的 Linux Kongress 和在澳大利亚组织举行的 linux.conf.au 。也有一些 Linux 论坛聚集了众多的商界领袖，他们在论坛上分享真知灼见，其中的一个例子就是每年在北美举行的 LinuxWorld Conference and Expo 。

在 <http://lwn.net/>上可以获得 Linux 开发社区的最新消息。如果你只是想简单地了解内核的最新发布版，不想阅读太多的资料， <http://lwn.net/>可能是一个好地方。另一个网络社区 <http://kerneltrap.org/>则讨论当前的内核议题。

在每个主线 Linux 内核版本发布中，你都会看到重大的改进，如内核抢占、不受限于锁（ lock-free ）的读操作、分担中断处理工作的新服务或者新体系结构的支持。因此，请一直跟踪邮件列表、网站和论坛，以保证自己在 Linux 技术上厚重的份量。

1.5 Linux 发行版

一个 GNU/Linux 系统除了内核以外，还由大量的实用工具（ utility ）、程序、库和工具（ tool ）组成，因此，获得和正确安装所有的组件将是一项艰巨的任务。而 Linux 发行版则有序地将这些组件进行了分类，并捆绑成相应的包。一个典型的发行版包含了数以千计现成的包。这使得用户无需担心下载不到正确版本的程序，也无需关心程序间的依赖问题。

因为打包是 GNU 许可证范围内的一种有效的赚钱方式，因此，目前的市场上诞生了不少 Linux 发行版。其中， Red Hat/Fedora 、 Debian 、 SuSE 、 Slackware 、 Gentoo 、 Ubuntu 和 Mandriva 这些发行版定位于桌面用户；而 MontaVista 、 TimeSys 和 Wind River 发行版则面向嵌入式系统开发。通常，嵌入式 Linux 的发行版还包括一套可动态配置的紧凑的应用程序集，以便针对资源的限制为系统进行量体裁衣。

除了打包以外，发行版还为内核的开发提供了增值服务。因此，许多项目都开始于发行版提供的内核而非 kernel.org 发布的官方内核，之所以要这样做，理由有如下几个。

         Linux 发行版遵守设备工业领域的标准，因此是开发的更好起点。 特殊兴趣组 （ Special Interest Group ， SIG ）已经成立，其目的是促进 Linux 在各个领域的应用。 消费电子产品 Linux 论坛 （ Consumer Electronics Linux Forum ， CELF ，网址为 [www.celinuxforum.org](http://www.celinuxforum.org/) ）集中于消费类电子领域的 Linux 应用。 CELF 规范定义了一些功能的支持等级，如可扩展性、快速启动、片上执行以及电源管理等。 开源开发实验室 （ Open Source Development Lab ， OSDL ，网址为 [www.osdl.org](http://www.osdl.org/) ）则集中于电信级设备。 OSDL 的 电信级 Linux （ Carrier Grade Linux ， CGL ）规范包含了对稳定性、高可用性、运行时补丁以及增强的错误恢复能力的诠释，这些问题在电信领域非常重要。

         主线内核版本可能并未包含对用户所选择的嵌入式控制器的充分支持，即使控制器建立在内核所支持的 CPU 核心之上。但是， Linux 发行版可能包含了控制器内部所有外设的设备驱动程序。

         在 内核开发过程中你计划使用的调试工具可能不包含在主线内核中。例如，内核并不包含内建的调试器支持。如果想在内核开发过程中使用内建的调试器，用户必须单 独下载并打上相应的补丁。如果针对用户内核版本的测试过的补丁并不易用，用户必须忍受更多的麻烦。而发行版则包含了很多有用的调试功能，所以你可以立即开 始使用它们。

         一些发行版提供了法律补偿，让你的公司无须为任何由于内核 bug 所产生的诉讼承担责任。

         发行版往往会对他们发布的内核进行较多的测试 [[ ③ ]](http://blog.donews.com/21cnbao/archive/2008/09/03/1342283.aspx" \l "_ftn3) 。

         用户可以从内核发行版的供应商处购买到他们提供的服务以及软件包支持。

1.6 查看源代码

在进入内核领域进行热身之前，让我们先下载 Linux 源代码，学会打补丁，并查看内核源码树的分布。

首先，到 [www.kernel.org](http://www.kernel.org/) 下载最新的稳定的源代码（源代码以 gzip （ .zip ）和 bzip2 （ .bz2 ）两种压缩格式提供），然后进行解压缩。在下列命令中，请用最新的内核版本号（例如 2.6.23 ）代替 X.Y.Z ：

bash> cd /usr/src

bash> wget www.kernel.org/pub/linux/kernel/vX.Y/linux-X.Y.Z.tar.bz2

...

bash> tar xvfj linux-X.Y.Z.tar.bz2

现在，你已经拥有位于 /usr/src/linux-X.Y.Z/ 目录的源代码树， 下面将 通过打 -mm 补丁 （ Andrew Morton ） 进行一项增加一些实验性功能的实验。

运行如下命令：

bash> cd /usr/src

bash> wget www.kernel.org/pub/linux/kernel/people/akpm/patches/X.Y/X.Y.Z/X.Y.Z-

mm2/X.Y.Z-mm2.bz2

打上这个补丁：

bash> cd /usr/src/linux-X.Y.Z/

bash> bzip2 -dc ../X.Y.Z-mm2.bz2 | patch -p1

命令中的 -dc 选项意味着让 bzip2 将指定的文件解压缩到标准输出。它将以管道方式输送到补丁程序，补丁程序会将补丁中的代码改变应用到源码树中每个需要修改的文件。

如果你需要打多个补丁，则要注意打补丁的顺序。为了生成一个包含 X.Y.Z-aa-bb 补丁的 内核，应首先下载 X.Y.Z 内核的完整源代码，再打上 X.Y.Z-aa 补丁，最后打上 X.Y.Z-aa-bb 补丁。

|  |
| --- |
| 补丁提交  使用 diff 命令可以为你完成的内核更改产生补丁：  bash> diff –Nur /path/to/original/kernel /path/to/your/kernel > changes.patch  要注意的是，在 diff 命令中，原始内核的路径应该放在修改后内核路径的前面。基于 2.6 内核补丁提交约定，你需要在补丁的最后加上这样的一行：  Signed-off-by: Name <Email>  有了这一行，你阐明了这些代码是由你编写的，你拥有贡献它的权利。  你现在就可以在相关的邮件列表（如 LKML ）中张贴你的补丁了 。  文档 Documentation/SubmittingPatches 包含了一个创建和提交补丁的向导，而 Documentation/applying-patches.txt 是一个教你如何打补丁的教程。 |

现在，打好补丁后的 /usr/src/linux-X.Y.Z/ 已经准备好投入使用了，接下来，我们花一些时间来查看内核源代码树的结构。进入内核源代码树的根目录并列出它的子目录：

(1) arch 。该目录包含了体系结构相关的文件。可以在 arch/ 目录下看到针对 ARM 、 Motorola 68K 、 s390 、 MIPS 、 Alpha 、 SPARC 、 IA64 等处理器的子目录。

(2) block. 。该目录主要包含块存储设备 I/O 调度算法的实现。

(3) cryto 。该目录实现了密码操作以及加密相关的 API ，它们可被应用于 WiFi 设备驱动的加密算法等场合。

(4) Documentation 。 该目录包含了内核中各个子系统的简 要 描述， 它将是你挖掘内核相关问题答案的第一站。

(5) drivers 。这个目录包含了大量设备类和外设控制器的驱动，包括字符、串口、 内置集成电路 （ Inter-Integrated Circuit ， I 2 C ）、 个人计算机存储卡国际联盟 （ Personal Computer Memory Card International Assciation ， PCMCIA ）、 外围组件互连 （ Peripheral Component Interconnect ， PCI ）、 通用串行总线 （ Universal Serial Bus ， USB ）、视频、音频、块、 集成驱动电子设备 （ Integrated Drive Electronics ， IDE ）、 小型计算机系统接口 （ Small Computer System Interface ， SCSI ）、 CD-ROM 、网络适配器、 异步传输模式 （ Asynchronous Transfer Mode ， ATM ）、蓝牙和 内存技术设备 （ Memory Technology Device ， MTD ）等。每一类设备对应 drivers/ 下面的一个子目录，例如 PCMCIA 驱动的源代码位于 drivers/pcmcia/ 目录， MTD 驱动位于 drivers/mtd/ 目录。 drivers/ 下的这些子目录是本书的主要议题。

(6) fs 。这个目录包含了 EXT3 、 EXT4 、 reiserfs 、 FAT 、 VFAT 、 sysfs 、 procfs 、 isofs 、 JFFS2 、 XFS 、 NTFS 和 NFS 等文件系统的实现。

(7) include 。内核头文件位于此目录。该目录下以 asm 开头的子目录包含了体系结构相关的头文件，比如 include/asm-x86/ 子目录包含了 X86 体系结构的头文件， include/asm-arm/ 包含了 ARM 体系结构的头文件。

(8) init 。这个目录包含了从高层角度看到的初始化和启动代码。

(9) ipc 。这个目录包含了对消息队列、信号、共享内存等进程间通信（ IPC ）机制的支持。

(10) kernel 。本目录包含基本内核中体系结构无关的部分。

(11) lib 。通用内核对象（ kobject ）处理、循环冗余校验（ CRC ）算法等库函数的实现位于此目录。

(12) mm 。这个目录包含了内存管理的实现。

(13) net 。该目录包含了网络协议，包括 Internet 协议第 4 版（ IPv4 ）、 IPv6 、 Internet 网络分组交换协议（ IPX ）、蓝牙、 ATM 、红外、链路访问过程平衡（ LAPB ）以及逻辑链路控制（ LLC ）。

(14) scripts 。内核编译过程中要使用的脚本位于此目录。

(15) security 。这个目录包含了针对安全的框架。

(16) sound 。 Linux 音频子系统位于此目录。

(17) usr 。此目录包含了 initramfs 的实现。

|  |
| --- |
| **统一的 x86 体系结构源码树**  从 2.6.24 内核开始， i386 和 x86\_64 （ 32 位的 i386 系统的 64 位的堂弟）体系结构源码树已被统一纳入共同的 arch/x86/ 目录。如果你使用的是比 2.6.24 老的内核，请用 arch/i386 / 代替本书中所说的 arch/x86 / 目录。同样地，用 include/asm-x86/ 代替 include/asm-i386 / 。此外，这些目录中的一些文件名也发生了变化。 |

阅读这些庞大的目录是一项艰巨的任务，表 1-1 中的一些工具可以帮助你更方便地浏览内核源码。

表1-1 源码浏览工具

|  |  |
| --- | --- |
| 工具 | 描述 |
| lxr | Linux 交叉引用（ lxr ），可从 <http://lxr.sourceforge.net/>下载，它可以让你通过 Web 浏览器浏览源代码，它为内核符号的定义和使用提供了超链接。 |
| cscope | cscope （网址为 <http://cscope.sourceforge.net/>）可用于为内核源码树建立一个符号数据库，通过它，你可以快速地定位到声明、定义以及正则表达式等。 cscope 可能不如 lxr 这般多才多艺，但是它很灵活，使用你最喜欢的文本编辑器（而不是浏览器）你就可以使用搜索功能。在内核源代码的根目录，运行 cscope - qkRv 命令就可建立交叉引用数据库。 - q 选项将产生更多的索引信息以加快搜索速度，但是它会消耗更多的额外的初始启动时间。 -k 要求 cscope 调整它的行为以适应内核源代码， - R 选项意味着递归遍历子目录。通过 man 手册页，你可以获得详细的调用语法。 |
| ctags/etags | Ctags 实用工具（网址为 <http://ctags.sourceforge.net/>）可用于为许多语言产生交叉引用的标签。通过它，你可以在 vi 等编辑器中定位源码树中的符号和函数定义。从内核源码树的根目录运行“ make tags ”可以为所有源文件建立标签。  Etags 实用工具为 emacs 编辑器产生相似的索引信息。运行“ make TAGS ” 可以为内核源文件创建标签。 |
| 实用工具 | grep 、 find 、 sdiff 、 strace 、 od 、 dd 、 make 、 tar 、 file 和 objdump 等工具。 |
| GCC 选项 | 实用 -E 选项可以让 GCC 产生预处理源代码。预处理代码包含头文件的扩展，并减少了为了扩展多层宏定义在多个嵌套的头文件间进行跳跃的需要。下面是一个预处理 drivers/char/mydrv.c 并产生扩展后的输出文件 mydrv.i 的例子：  bash> gcc -E drivers/char/mydrv.c -D\_\_KERNEL\_\_ -Iinclude  -Iinclude/asm-x86/mach-default > mydrv.i  使用 -I 选项可以指定你的代码说依赖的 include 的路径。  使用 -S 选项可以让 GCC 产生汇编列表。下面的命令可以为 drivers/char/mydrv.c 产生汇编文件 mydrv.s ：  bash> gcc -S drivers/char/mydrv.c -D\_\_KERNEL\_\_ -Iinclude  -Ianother/include/path |

1.7 编译内核

现在，你已经拥有了内核源代码布局的理念，现在，让我们进行一项细小的内核代码修改，并编译和运行 Linux 。进入顶层的 init / 目录，对初始化文件 main.c 进行一项小的修改，即在 start\_kernel() 函数的开头加上一个打印语句，宣布你对北极熊的喜爱：

asmlinkage void \_\_init start\_kernel(void)

{

    char \*command\_line;

    extern struct kernel\_param \_\_start\_\_\_param[],

           \_\_stop\_\_\_param[];

+   printk("Penguins are cute, but so are polar bears\n");

    /\* ... \*/

    rest\_init();

}

你现在已经为编译内核揭开了序幕，进入内核源代码并运行清除命令：

bash> cd /usr/src/linux-X.Y.Z/

bash> make clean

接下来，进行内核配置工作。这一步的主要工作是选择你要编译的组件，可以选择需要的组件以静态还是动态链接的方式编译进内核：

bash> make menuconfig

menuconfig 是内核配置菜单的文本界面，使用 make xconfig 可以产生一个图形界面。你所选择的配置信息被存放在内核源码树根目录的 .config 文件中。如果你不想从头开始进行配置，可以使用 arch/your-arch/defconfig 或者 arch/your-arch/configs/your-machine\_defconfig （如果你的体系结构中包含 一些 可被支持的平台的话） 文件作为起点。因此，如果你正在为 32 位 x86 体系结构编译内核，则运行如下命令：

bash> cp arch/x86/configs/i386\_defconfig .config

编译内核并产生一个压缩的启动映像：

bash> make bzImage

现在，内核映像将位于 arch/x86/boot/bzImage ，更新你的启动分区：

bash> cp arch/x86/boot/bzImage /boot/vmlinuz

你也许需要修改根据新的启动映像更新你的 bootloader 。如果你正在使用 GRUB 这个 bootloader ，它将自动完成配置；如果你正在使用 LILO ，则增加一个标记：

bash> /sbin/lilo

Added linux \*

最后，重新启动 Linux 并启动到你的新内核：

bash> reboot

启动后的第一条信息显示了你对北极熊的热爱。

1.8 可加载的模块

由于 Linux 可运行于各种各样的体系结构，并支持无数的 I/O 设备，因此把所有要支持的设备都直接编译进内核并不可行。 发行版通常包含一个最小的内核映像，而将其他的功能以模块的形式提供。在运行的时候，模块会按需动态加载。

要产生模块，则进入内核源码树根目录并编译：

bash> cd /usr/src/linux-X.Y.Z/

bash> make modules

运行如下命令安装编译生成的模块：

bash> make modules\_install

此命令将在 /lib/modules/X.Y.Z/kernel/ 目录下创建一个内核源代码目录结构，并将可加载的模块放入其中。它也将激活 depmod 实用工具，以便产生模块依赖文件 /lib/modules/X.Y.Z/modules.dep 。

如下实用工具可用于操纵模块： insmod 、 rmmod 、 lsmod 、 modprobe 、 modinfo 和 depmod 。前两个实用工具用于加载和移除模块，而 lsmod 用于列出目前已经加载的模块， modprobe 是 insmod 的一个更智能的版本，它通过分析 /lib/modules/X.Y.Z/modules.dep 文件来加载相关模块。 例如，假定你需要挂载一个 USB 笔驱动器上的 VFAT 分区，使用 modprobe 加载 VFAT 文件系统驱动 [[ ④ ]](http://blog.donews.com/21cnbao/archive/2008/09/03/1342283.aspx" \l "_ftn4) ：

bash> modprobe vfat

bash> lsmod

Module       Size     Used by

vfat         14208    0

fat          49052    1 vfat

nls\_base     9728     2 vfat, fat

从 lsmod 命令的输出可以看出， modprobe 加载了 3 个模块而不是 1 个。 Modprobe 首先发现它不得不加载 /lib/modules/X.Y.Z/kernel/fs/vfat/vfat. ko ，当它查看 /lib/modules/X.Y.Z/modules.dep 模块依赖文件的时候，它发现了如下内核 ， 并由此意识到自己必须首先加载另外 2 个模块：

/lib/modules/X.Y.Z/kernel/fs/vfat/vfat.ko:

/lib/modules/X.Y.Z/kernel/fs/fat/fat.ko

/lib/modules/X.Y.Z/kernel/fs/nls/nls\_base.ko

于是它首先加载了 fat.ko 和 nls\_base.ko 这 2 个模块，之后加载 vfat.ko ，这样，你挂载 VFAT 分区时所需要的所有模块都被自动加载了。

使用 modinfo 实用工具可以提取模块的详细信息：     

bash> modinfo vfat

filename:      /lib/modules/X.Y.Z/kernel/fs/vfat/vfat.ko

license:       GPL

description:   VFAT filesystem support

...

depends:       fat, nls\_base

为了将内核驱动编译为模块，在配置内核的时候，请将相应的菜单选择按钮置为 <M> 。本书中的大部分设备驱动例子以内核模块的形式实现。为了从 mymodule.c 源文件构造 mymodule.ko 模块，可以创建一个一行的 Makefile 文件，并且以如下方式执行它：

bash> cd /path/to/module-source/

bash> echo "obj-m += mymodule.ko" > Makefile

bash> make –C /path/to/kernel-sources/ M=`pwd` modules

make: Entering directory '/path/to/kernel-sources'

  Building modules, stage 2.

  MODPOST

  CC /path/to/module-sources/mymodule.mod.o

  LD [M] /path/to/module-sources/mymodule.ko

make: Leaving directory '/path/to/kernel-sources'

bash> insmod ./mymodule.ko

内核模块减小了内核的尺码，并缩短了开发—编译—测试的周期。对于一次修改而言，你仅仅需要重新编译特定的模块并重新加载它。在第 21 章中，我们将学习模块调试技术。将驱动设计为内核模块也有一些缺陷。不像内建的驱动，模块无法在系统启动时预留资源，在加载时预留资源成功的几率会减小。

1.9 整装待发

Linux 已经跋涉了许多的地域，成为一门艺术。所以你可以基于它来学习操作系统的概念、处理器体系结构甚至工业领域。当你学习某一设备驱动子系统所使用的技术时，可以更深入地探索其背后潜在的设计由来。

在没有明确说明的情况下，书中都假定为 32 位 x86 体系结构。但是，本书也考虑到了一个事实：你更有可能为嵌入式设备而非传统的 PC 兼容的系统编写驱动程序。因此，对于串口驱动一章，讲解了 2 个设备：一个 PC 衍生器件上的触摸控制器和一个手机上的 UART 。对于 I 2 C 设备驱动程序一章，则讲解了 PC 系统中的 EEPROM 和嵌入式设备中的实时钟。本书也将介绍内核为大多数设备驱动类所提供的核心基础设施，它们隐藏了设备驱动程序的体系结构相关性。

在本书接近尾声的第 21 章讨论了设备驱动的调试技术。你可能会发现，在阅读本书的过程中，开发驱动时提前阅读该章会很有用。

本书基于 2.6 内核，它包含了对 2.4 内核的大量改变，覆盖了所有的主要的子系统。因此，希望你已经在你的系统中安装了基于 2.6 的 Linux ，并开始用内核源代码进行实验。基于以下两个主要的原因，本书的每一章都充分地指出了相关的内核源文件：

(1) 因为内核中的每个驱动子系统包含成千上万行的源代码，所以本书只可能进行一个相对简单的呈现，查看源代码中与书中例子相关的真实的驱动将为你提供更广阔的视角。

(2) 在开发一个驱动之前，参考 drivers/ 目录中现存的、与你的要求相似的驱动并以之作为起点是一个好主意。

因此，为了从本书中获得最大的益处，要通过频繁地浏览源码树和啃代码来熟悉内核。在代码探索的过程中，也要跟踪邮件列表的进展。

[[ ① ]](http://blog.donews.com/21cnbao/archive/2008/09/03/1342283.aspx" \l "_ftnref1) 意指对源码进行一些有针对性的修改。——译者注   
[[ ② ]](http://blog.donews.com/21cnbao/archive/2008/09/03/1342283.aspx" \l "_ftnref2) 版权为 copyright ，这里故意用 copyleft 。——译者注   
[[ ③ ]](http://blog.donews.com/21cnbao/archive/2008/09/03/1342283.aspx" \l "_ftnref3) 因为需要将内核冻结在一个版本上进行测试（而这个版本不是最新的），所以发行版内核经常会引入比其版本更新的官方内核的一些功能。   
[[ ④ ]](http://blog.donews.com/21cnbao/archive/2008/09/03/1342283.aspx" \l "_ftnref4) 这个例子假定这个模块没有被内核自动加载。如果你在配置过程中启用了自动内核模块加载 (CONFIG\_KMOD) 选项，那么当侦测到缺失的子系统时，内核将自动以相应的参数运行 modprobe 。在第 4 章中，你将学习到模块自动加载的知识。