Linux 内核与设备驱动程序

第2章

- Linux操作系统
- Linux 内核
- 设备驱动程序

如果使用Linux只是用于一般的用途，则大可不必了解设备驱动程序。但是，如果想访问硬件，则必须与设备驱动程序打交道。在Linux中有很多与硬件有关的部分，以库函数的形式提供给用户。因此，程序员即使不了解硬件的具体细节也可以通过相关库函数来访问和控制硬件。但是，如果为一个新的设备开发相关驱动程序，或者欲找出设备驱动程序中存在的错误，则应对设备驱动程序的结构有相当的了解。

本章简单介绍在学习设备驱动程序之前必须掌握的基本概念。从第3章开始详细介绍各个细节。在这里，首先介绍设备驱动程序的运行环境，即Linux的特点，并介绍什么是设备驱动程序及Linux 的设备驱动程序具有什么样的特点，此外，为了搜索学习设备驱动程序时必要的内核源代码，还介绍与设备驱动程序有关的重要内核目录树，控制设备驱动程序的目录及相应目录可驱动的设备类型。

本章内容适合：

- 2.4内核
- 2.6内核

2.1 Linux操作系统

设备驱动程序（device driver）是Linux的一部分，因此需要理解Linux内核。本章简单介绍Linux操作系统的特点。

作者使用两台电脑，其中一台安装了Windows XP，用于编辑本书。另外一台则安装了Linux，用于编译代码和保存代码文件。

虽然这两种体系在不断地、认真地完成各自的任务，作者并不关心这样的操作系统。即只关心利用键盘输入文字时所用到的Word等软件，对运行中的系统并不好奇。操作系统便是这样存在着。多数人都在忽视它，但是每时每刻都在利用它。然而，还是有一部分人关注着操作系统，他们思考操作系统是怎么运行的？操作系统的任务是什么？怎么制作操作系统？进而，他们学习并研究操作系统。

学习操作系统需要操作系统的相关文档和源代码。如果没有这些，那么就很难真正地理解和掌握一个操作系统。尽管成千上万人正在使用微软公司的Windows操作系统，但是很少有人对Windows操作系统的内部了解得非常透彻。

相比之下，Linux的出现是一种莫大的幸运！与UNIX不同的是 ，Linux是一种以公开源代码的姿态登录的操作系统。它使希望学习操作系统或制作操作系统的多数人发狂，进而参与了开发工作。Linux内核并不仅是公开了源代码，而且把使用方法和各种疑难问题也编辑成文字资料公开于众人。很多开发者开始接近Linux，并在自己的系统上安装、修整后，再次公布于世人。因此，尽管不是商用软件，它的功能一样优秀，可安装在无数台电脑上，作为Mail服务器或Web服务器系统，最近嵌入式系统也在使用该体系。

正如前面提到的，Linux是公开了源代码的操作系统，因此Linux的开发人员并不是某个特定的集团，而是不确定的多数人。此外，从内核源代码的大小50MB（由纯text构成）就能猜测到，它是由多个内容构成的，即由系统的启动、进程调度、虚拟内存管理器、文件系统、设备驱动程序等多个组成的集合便是Linux的操作系统。此外，其他大量的GNU的软件的加盟，更使Linux如虎添翼。

2.2 Linux内核

设备驱动程序中使用的函数也用在内核上，并且影响着内核，因此离开内核就无法制作设备驱动程序。由于内核本身所具有的内容过多，本节就简单整理了学习设备驱动程序时需要掌握的内核的基本内容。

2.2.1 Linux的历史

如前述内容，Linux并不是某个特定公司独立开发的操作系统，而是由芬兰赫尔辛基大学的学生Linux-B.Torvalds在大学期间开发的操作系统。刚开始Linux-B.Torvalds对MINIX（一种小型VNIX系统）非常感兴趣，后来他决心开发出比MINIX更加优秀的操作系统。不久，1991年推出了0.02版本，到1994年开发了kernel 1.0。之后，全世界的许多黑客参与了Linux的开发。目前以开发出了2.6版本（开始编辑本书时，kernel版本已达到了2.6.4，并且持续发展）。

其实，Linux并非完全由Linux内核构成。Linux和理查德.斯托曼所创建的自由软件基金会（Free Software Foundation，FSF）的GNU项目联姻，Linux负责内核，而其他则由GNU项目的程序员开发。GNU项目的目的就是开发一个类似于UNIX，但比UNIX更为强大、可自由地使用的操作系统。为此，他们开发了一个叫做GCC的C/C++编译器和强大的编辑器Emacs以及许多GNU版本的UNIX工具软件。当Linux推出基于80386体系的内核版本之后，GCC也移植到该版本的Linux。随即，许多GNU软件和其他开源软件也被GCC编译后融入Linux中了。Linux遵循通用公共许可证（General Public License，GPL）规则，其源代码可自由地发布。任何人都可以按照自己的意愿去修改它，但是，必须按照GPL精神公开修改后的内容。

Linux与相关的发布版并不都是免费的。很多公司或开发者可以在Linux源代码的基础上开发商用软件。

由于Linux具有很强的性能和很好的灵活性，很多开发者根据自己的需要修改了Linux源代码以满足应用的需求。目前，Linux支持很多各种不同的硬件，各种移植到其他平台的项目也开展得非常活跃。

2.2.2 Linux内核的功能

内核是有效管理系统资源的资源管理者，它的功能包括以下几种。

（1）处理器管理

通常，一台PC机有一个CPU，即使用一个处理器也是如此。作为服务器的系统需要高速处理功能，而为了提高处理速度并联了多个处理器。管理系统中运作的处理器也是内核所管理的资源之一。通过管理处理器，使处理器满足操作系统的处理要求，使其有效分担和执行各个进程中必要的处理器功能。

（2）进程管理

Linux操作系统启动后，系统中至少有一个以上的进程在运行。进程又称为任务（task），是程序执行的基本单位。内核通过进程调度器负责进程的创建、撤销，并管理进程和外部环境的交互。

（3）内存管理

与处理器相同，系统的内存是最核心的部分，也是需要管理的重要资源。内存管理是决定系统性能的重要要素，提供虚拟地址空间的方式，使各个进程在独立的空间上执行任务。虚拟内存以管理为基础，结合存储器可提供大于实际物理内存的内存空间。

（4）文件系统管理

Linux内核是以Unix系统中使用的文件系统为基础设计的。因此，文件系统呈现给用户统一的程序界面，使Linux内核上运行的应用程序以文件形式管理系统中运行的所有资源。

此外，Linux内核利用虚拟文件系统（VFS），支持当前的大部分文件系统类型。

（5）设备的控制

应用程序必须通过硬件才能与用户交互。除了CPU和内存以外，应用程序必须得到硬件的支持才能够完成各种操作。设备驱动程序负责与硬件相关的各种操作，而设备驱动程序则

是内核必须实现的功能之一。在Linux系统中设备驱动程序的设计遵循文件系统的结构，因而能够通过标准的接口来访问和控制硬盘、键盘和网络等外部设备。

（6）网络管理

目前，大多数操作系统都支持网络功能。Linux操作系统在设计之初就已经考虑到了网络的重要性，因此，具备了优秀的网络管理系统。应用程序可通过内核中的网络协议栈高效地访问网络设备。此外，网络系统所提供的安全特性和加密功能增强了系统的可靠性。

2.2.3 Linux内核的特点

Linux内核是Linux操作系统的核心。Linux内核常驻在系统内存中，负责管理内存、硬件等各种系统资源。Linux内核还通过进程调度有效地支持多个进程的并发运行，并负责输入输出操作等操作系统的核心功能。

与其他操作系统相比，Linux的最大特点在于用户可以随意制作、完善内核。因此，以源代码配置了Linux内核，被配置成执行文件时又包含在配置包里。多数用户直接使用配置包，很少有人下载内核的源代码后进行编译。但是需要优化配置系统的运作时，必须下载源代码，再通过内核的编译选项设定进行优化处理后，才能使用。此外，已存在的内核不适合自己的系统时，可以直接修改，再与其他人共享修改后的内容。

Linux与其他操作系统不同，用户可以按照自己的需要定制Linux内核。Linux通常以源代码的形式发布，如果以可执行代码的形式发布，则通常在软件包中包含内核的源代码。在大多数情况下，用户会安装和使用可执行代码形式的Linux版本。因此，很少有人下载Linux内核源代码，编译并安装使用。但是，如果用户想定制Linux内核以便更好地适应应用的需求，则需要获得Linux内核源代码，选择适当的内核选项，并编译和使用。

Linux内核不仅可以根据自己的需要来定制，修改后的内核还可以通过各种渠道发布，以便其他用户在需要时下载和使用。

用户可使用Linux内核源代码所提供的若干个脚本命令轻松地完成内核的定制，而有关Linux内核的文档也很容易得到。Linux内核的定制和自由发布对Linux及其用户的迅速发展产生了巨大的影响。在其他操作系统因商业利益而裹足不前时，Linux却以其自由精神获得了巨大的发展。

目前，Linux系统在支持各种CPU和硬件平台方面可谓首屈一指。从嵌入式系统到企业计算环境中随处可见Linux的身影。因为，Linux得到了很多开发者的支持和不断的改进，在性能和稳定性等方面显示了非凡的能力。不仅如此，Linux具有与UNIX类似的结构，从2.4版本开始与IEEE POSIX标准兼容。因此，很多UNIX程序稍经修改便可在Linux系统中编译和运行。这种特点在2.6版本中也得到了充分的支持。2.4版本下的程序不经修改就可以在2.6版本的环境中运行。

Linux的具有代表性的特点可归纳为以下的几个方面：

• 单一的（Monolithic）内核　Linux内核采用了与大部分UNIX类似的单一结构。因此，Linux内核的各个组成部分之间有非常复杂的关系。

• 非抢占型（2.4内核）和抢占型（2.6内核）2.6内核之前的版本都是不可抢占型内核。在非抢占式内核中，一旦进程从用户模式转换到内核模式，从外部就无法中断进程的运行。只能等待它们自己主动释放CPU。相反，抢占型内核在进程以内核模式运行时，也允许调度程序中断当前进程而调用更高优先级的进程，通过对抢占点的测试避免不合理的系统调用延时，2.6内核就支持该功能。2.6内核在内核编译选项中设置了抢占型和非抢占型两个选项。

• 虚拟内存系统（VM） Linux内核是在多个平台上运行的操作系统，因此将在i386微处理器上运行的内存管理系统标准化，使其适应多种MMU（内存管理单元）设备。Linux虚拟内存管理还可以利用辅助存储器管理比实际的物理内存大的内存。

• 支持无MMU Linux内核是在i386的基础上发展起来的系统，因此运行利用MMU存储器管理单元的内存管理方法。然而，对于嵌入式系统中经常使用的处理器，也有可能不存在MMU（存储器管理单元）。2.6内核可以支持此类处理器功能，即没有MMU的系统。

• 虚拟文件系统（VFS） Linux内核的最大特点之一就是虚拟文件系统。与UNIX系统不同，Linux在开发之初就支持虚拟文件系统（VFS）。几个UNIX系统只能支持特定文件系统。Linux内核不只继承了UNIX的文件系统，还可以处理其他的多种文件系统，真正成为了虚拟文件系统。Linux中可以使用ext2甚至日志文件系统jfs的多种文件系统，也可以处理在Windows中运行的NTFS文件系统和DOS文件系统。

• 利用模块扩展内核 Windows等最新的内核在操作系统运行中还可以添加或删除内核代码，特别对于支持hotplug随插即用的硬件更是必需的。Linux内核利用模块来实现对内核的扩展。

• 内核线程 2.4内核之前的版本只能支持有限的内核线程，2.6内核开始支持NPTL（Native POSIX Threading Library）和NGPT（Next Generation POSIX Threading Package）。

• 支持多线程 过去采用同时运行多个进程的模式设计了应用程序。然而，生成进程的费用过高，因此以一个进程中使用多个线程的方式替换了同时运行多个进程的方式。Linux内核简化了进程管理方法，支持多线程方式。

• 支持多处理器 Linux内核支持SMP方式的多处理器系统。尤其从2.6版本开始支持NUMA（Non-Uniform Memory Access）体系结构。

• 强大的网络功能 Linux内核继承了BSD网络处理方式，因此网络处理的稳定性及效率极高。

• GPL许可 Linux内核的特点之一就是GPL许可方式。因此，所有人都可以参考源代码，修改并再配置源代码。

2.2.4 内核源代码的结构

Linux内核源代码按照所实现的功能被分布在各个目录下。由于Linux内核所涉及的内容过分庞大，因此，表2-1只列出与设备驱动程序相关的目录。

表2-1 内核目录的清单

目录	说明
Documentation/	关于内核的各种文档
arch/	与平台有关的代码
crypto/(2.6)	加密代码
drivers/	设备驱动程序
fs/	文件系统
include/	内核代码的头文件
init/	内核的初始化代码
ipc/	System V 进程间通信
kernel/	进程、timing、程序运行、信号、模块等核心代码
lib	内核内部使用的库函数
mm/	内存管理
net/	网络协议栈
scripts/	编译内核时用到的shell脚本和程序
security/(2.6)	安全
usr/(2.6)	initramfs的实例

1. Documentation/

在该目录下有很多设备驱动程序开发者需要参考的各种文档。文档的内容包括在使用内核代码时，不同平台需要注意的各种事项以及设备驱动程序的标准开发方法等。在config.help文件中保存着2.4内核版本的有关设置内核选项的帮助，如果想在内核中增加一些选项，则需修改此文档。如果想了解编译内核时所用的语法，则应仔细查看kbuild。

2. arch/

考虑到移植的方便，Linux内核将平台有关的代码和平台无关的代码分别存放在不同的目录下。尽管Linux设备驱动程序独立于平台，但是，由于有关硬件本身在一定程度上受制于平台，因此不能完全独立。尤其涉及平台相关的寄存器时，经常需要引用arch目录下的源代码。表2-2是该目录下的各个子目录清单（2.6内核）。

表2-2 arch目录的子目录清单

目录	说明
alpha/	HP的alpha微处理器
arm/	ARM微处理器
arm26/	ARM 26位微处理器
cris/	axis communication的CRIS体系结构
h8300/	
i386	Intel i386系列的微处理器
ia64/	Intel 64位Itanium微处理器
m68k/	摩托罗拉MC680×0微处理器
m68knommu/	没有MMU的MC680×0系列微处理器
mips/	Mips微处理器
parisc/	HP9000 Parisc工作站
ppc/	摩托罗拉-IBM Power PC微处理器
ppc64/	64位Power PC微处理器
s390/	IBM ESA/390 z 系列

续表

目录	说明
sh/	SuperH微处理器
sparc/	Sun Sparc微处理器
sparc64/	64位Sun Sparc微处理器
um/	用户模式内核
v850/	NEC的V850系列微处理器
x86_64/	64位i386系列的微处理器

各个体系结构中包含着内核启动至定时器 、中断信号、MMU、处理器结构有关的程序代码。通常，在不同的平台上分别管理不同处理器目录下的子目录。

3. include/

编写设备驱动程序时，需要搜索内核源代码，也要搜索头文件。Linux内核源代码把头文件分为与体系结构密切相关的头文件和独立的头文件两大类进行管理。Include/asm目录中存在与体系结构密切相关的头文件，include/linux中集合了在Linux内核中不受体系结构影响的头文件。以下是开发设备驱动程序时经常参考的目录。

• include/asm 符号连接的目录。因此，实际位置连接在“asm-”开头的实际体系结构目录上。该目录中的头文件中定义了符合体系结构的特性的各种变量类型或结构体以及其他宏函数。对于使用相同处理器的系统而言，它的结构也不一定相同。连接到设备驱动程序上的部分上更加突出了这样的差异。此类Machine体系结构相关部分再次细分到下一级目录上进行管理。Machine体系结构的头文件名称开头为“mach-”。

• include/linux 作为编写设备驱动程序的过程中参照最多的目录，收集了与标准linux内核相关的所有头文件。该目录是采用了POSIX标准的基本头文件，虽然应用程序也能参照其文件，主要集合了内核所能参照的内容。

• include/media/、include/pcmcia/、include/scsi/、include/sound/、include/video/这些目录主要包含与设备驱动程序有关的为数不多的头文件，而大部分的内容则在include/linux目录下。包含了控制设备的设备驱动程序中必要的头文件。

2.2.5 浏览内核源代码

在开发设备驱动程序或在分析已有的设备驱动程序时，需要浏览内核源代码以便了解有关函数和数据的定义。对于初学者来说，浏览庞大的内核源代码并非一件易事。下面就推荐几种我经常使用的方法供大家参考。这些也并非绝对的，大家可以寻找适合自己的其他方法。

1. http://lxr.linux.no/

访问该网站是浏览Linux内核源代码最有效的方法。可以通过浏览器浏览源代码，查看所要的函数、变量和结构体等。也可以在自己的计算机上安装该网站的帮助系统，但是安装方法比较难，因此，建议大家避开美国的访问高峰期访问该网站。该网站提供identifier search和freetext search两种搜索方式。identifier search用于搜索函数、变量等可以用文法区分的单词，而freetext search则用于搜索指定的字符串。

显示的结果包括两个方面的内容。一是被定义的头文件和源代码的位置，二是引用相应内容的源代码列表。在设备驱动程序中包含头文件时，经常用到定义位置。在设备驱动程序上添加头文件时经常用到定义位置。查找相应函数或变量的使用方法时可以使用引用位置（由于缺乏有关内核函数的参考文档，查阅这些函数在其他源代码中的调用方法，也许是了解它们的唯一途径）。

该网站的帮助系统存在的问题是，当使用结构体的函数指针调用函数时，难以继续跟踪。

2. Grep命令的组合

Linux内核中grep的命令最为有效，是搜索内核源代码时最有用的命令。该命令的用法如下。

```
使用方法：grep[选项]. . . 模式[文件]. . .
```

如果想查找readb，则执行以下grep命令。

```
[root@]# grep readb * -r
```

此命令会查找包含readb字符串的所有文件（包括当前目录和子目录下的文件）。如果显示的结果太多，那么可以使用grep命令的组合方式。例如，想查找声明readb的地方则只需查找相关头文件即可。这时就执行以下命令。

```
[root@]# grep readb * -r |grep include
```

该命令使用了grep命令与管道命令的组合，将显示在第一个grep命令的结果中只包含include的部分。查找指定函数被调用的位置也可以使用类似的命令。重复使用多个管道命令也可查找包含多个单词的文件。

3. Include/linux目录的浏览习惯

在编写设备驱动程序的过程中寻找内核源代码上使用的函数时，能够看到不少宏函数。因此，在查找某些函数时，从include/linux或include/asm目录开始查找是很有必要的。

提示

本书主要讨论Linux设备驱动程序的相关内容，至于Linux内核，很难在本书中做详细的讨论。但是，学习Linux设备驱动程序需要了解Linux内核的运行机制，因此，在这里给大家推荐《运行Linux（修订4版）》（Hanbit Media，2003）一书，以便在需要时参考。

2.3 设备驱动程序

前一节中简单介绍了Linux设备驱动程序的基础知识，即Linux的特点和内核相关的内容。在这里首先介绍Linux设备驱动程序的特点，并介绍与设备驱动程序相关的重要内核目录树，还有运行设备驱动程序的目录和相应目录所能控制的设备类型等内容。

2.3.1 向内核请求资源处理的方法

设备驱动程序（device driver）使系统支持的硬件能够在应用程序中使用，它是内核所提供的数据库。与大多数操作系统相同，Linux也把硬件看作内核所管理的系统资源。应用程序要想控制硬件，必须向内核发出资源使用请求，而内核根据这个请求管理系统。应用程序向内核发出资源处理请求的方法大致分为两大类。第一是系统调用方式，第二是利用文件输出输入方式使用设备驱动程序的方法。

1. 系统调用方式

所谓系统调用方式，指制定系统控制所需的调用规则后，利用软件中断服务功能，使内核执行应用程序发出的处理请求。内核为了区分需要处理的功能，依据功能分配序号，并在内核里定义相应序号的控制程序。应用程序把相应的功能序号存储到寄存器中，然后调用软件中断服务，从而把控制权提交到内核。被中断（interrupt）调用后，由内核内部的服务中断再确认功能序号，再调用与控制有关的服务程序。处理了所有的服务程序后，内核再向应用程序交还控制权。编写Linux应用程序的人们把所有系统调用隐藏为数据包函数，因此很难认识到这样的系统调用方式。

最初设计的操作系统中只存在系统调用方式，因此所谓控制硬件其实就是系统调用。然而系统中使用的硬件数量逐渐增多，利用系统调用的应用程序的功能也不断发展，已经出现了诸多问题。在编写应用程序的开发者立场上看，需要掌握更多的控制硬件的系统调用相关知识，相反，在编写内核服务函数的开发者立场上看，又增加了硬件开发的负担。

因此，没有标准化的硬件控制接触方式时，偶尔会处于瘫痪状态。为了有效处理系统调用，必须减少条件处理的分支数量，从而合理分配CPU占用时间。然而硬件的数量过多，早已超出了分配资源的限度。另外，早先设置的变量类型也无法满足需求。因此，当前的操作系统中除了系统调用功能外，还定义了另一种硬件控制的方法。

2. 文件形态的设备驱动程序

UNIX系统为了控制硬件，使用系统调用方式的同时，又引入了以文件输入输出函数控制硬件的概念。该方式与普通文件的处理过程相似。向表示硬件的设备文件指定应用程序的输入输出，从而调用内核内部连接设备文件的设备驱动程序服务程序，处理完请求再把控制流交还给应用程序。Linux内核在继承UNIX文件方式的基础上增加了几种特性，从而改善了功能。由应用程序的开发人员将设备接触方式标准化后形成了利用设备文件的硬件控制方式，而由编写设备驱动程序的开发人员制作了适合界面的服务程序。

2.3.2 模块和设备文件

1. 模块

在内核开发初期，Linux开发人员要想测试编好的设备驱动程序，必须在内核源代码里生成模块。该方式需要较长的内核编译时间。并且每次修改内容时，都要再次启动系统。为了改善这样的低效率方式，人们开发了在系统内核被启动运行的同时可修改设备驱动程序的

方法。从而被Linux开发人员提出的概念就是模块，它是在内核被启动后运行的状态下即动态加载或清除设备驱动程序的一种概念。

模块方式不仅可以缩短Linux设备驱动程序的开发时间，而且内核上没有设置多余的功能，合理配置了内核的资源。

只有模块方式才能支持连接在PCI、USB、PCMCIA上的设备PNP功能。例如，PCMCIA支持设备的热插拔，利用模块可以使内核对应设备的添加和删除。

只有带MMU的CPU才能支持模块方式，并且内核的版本要一致。

2. 设备文件

Linux中运行的很多应用程序都不能直接控制硬件。采用stdin和stdout标准输入输出文件格式隐藏了用户经常使用的输入设备——键盘。另外，对于编写Linux应用程序的程序员，硬件并不是需要关注的对象，多数情况下没有必要掌握Linux中硬件的运行方法。编写Linux的音频生成程序或打印控制程序时以及利用设备驱动程序直接控制硬件时，都要掌握硬件和设备驱动程序的连接方法。此时需要掌握的内容便是设备文件。

Linux中运行的应用程序没有用于管理硬件的特定函数或方法。有一种形式叫系统调用（call），它实质上是连接内核和应用程序的界面，并不是控制硬件的通道。Linux直接继承了UNIX的哲学，因此它以文件形式体现系统中的所有资源。RAM、进程及任务都可以表示为文件，甚至可以把辅助存储设备——硬盘也表示为文件。普通的文件可以保存程序中写入的数据，并且相应地扩展文件大小。此类文件属于信息存储类型文件。但是/dev/目录下的文件具有一些特殊的功能。通常把Linux中/dev/目录下的文件称作设备文件。这里的每一个设备文件真正表示硬件。例如连接系统的输入设备——鼠标表示为/dev/mouse设备文件。处理显屏输入输出的设备文件为/dev/console。/dev/目录底下存在很多尺寸较小的设备文件，可以使用文件删除语句控制设备文件。另外，还有系统中没有的大量设备文件。

设备文件是一种信息文件。普通文件的目的在于保存数据，而设备文件的目的在于提供系统或硬件的信息。此类设备文件上保存的信息包括3种类型，分别为设备类型信息和主设备号（major）、次设备号（minor）。

其他的信息没有太大的意义。设备类型的信息可以区分设备是文字还是块。主设备号是连接应用程序和设备驱动程序的纽带。应用程序以open()函数打开设备文件，内核在相应的设备文件上得到主设备号，并寻找相应设备号所处理的设备驱动程序。使用主设备号连接完应用程序和设备驱动程序，再由次设备号显示实质的设备。当然，对于不同的设备驱动程序，次设备号的意义也不同，因此不一定遵守上述的运行规则，但是大多数设备驱动程序中的次设备号仍表示实质的设备。

也有不存在设备文件的设备驱动程序，包括网络设备驱动程序。

Linux系统中运行的程序只能使用设备文件控制硬件。通常，此类设备文件上使用低级输入输出函数。最有趣的是，生成设备文件时不使用creat()函数，而是使用mknod等实用程序（utility）。当然，mknod也是利用了系统调用的实用程序。由mknod生成的设备文件不一定要配置在/dev目录下，可以随意指定它的位置。然而，为了形成标准化的文件系统，最好

指定到/dev目录下。部分文件系统无法形成设备文件系统，即网络文件系统或支持Linux标准化inode类型的文件系统中无法生成设备文件。

2.3.3 设备驱动程序的类型

Linux中备有大量设备驱动程序，可以使用大量的设备。设备驱动程序处理各个硬件的方式与硬件的类型有关。Linux内核为了控制多种类型的硬件，制作了几种标准化模式，以该模式为基础编写了各式各样的设备驱动程序，从而控制大量硬件。Linux中使用的设备驱动程序改变了UNIX的派生结构。因此，部分内容直接继承了UNIX的特性，而部分内容的形态又与UNIX有差异。特别是，随着版本的提高，Linux内核进一步得到改善，逐步发展为与UNIX不同的设备驱动程序。

Linux中使用的设备驱动程序大体上包括字符结构、块结构和网络结构。它们是基本结构，与Linux内核的使用方式有着密不可分的关系。

由基本结构变换为适合各种硬件的设备驱动程序结构。Linux中使用的基本设备驱动程序分成3大类。

• 字符　控制长短不一致字符列的设备驱动程序，应用程序直接呼叫的、没有缓存的设备驱动程序。

• 块　通过一定大小的缓存处理数据的设备，由内核内部的文件系统进行管理，没有内部缓存的设备驱动程序。

• 网络　与网络层连接的设备驱动程序。

这是基本的分类方法，为了理解设备驱动程序的结构必须掌握上述概念。查看设备驱动程序的源代码后，我们能够发现上述的基本分类法还是不够完善，上述的分类法的依据只不过是内核中控制设备驱动程序的方法而已。在硬件的立场上，依据内核源代码中出现的设备驱动程序目录结构进行分类的方法更为科学。不采用内核控制设备驱动程序的方式，而是采用依据硬件的连接与类型分类的方式，这便是2.6内核中新增加的整体型设备驱动程序的概念。

1. 字符设备驱动程序

字符设备驱动程序是最接近普通文件方式的设备驱动程序。应用程序中使用open( )、close( )、read( )、write( )等文件处理函数，并且以普通文件方式处理设备文件，从而控制硬件。常见的文件结构为流（stream），与字符设备驱动程序连接的设备文件也是流。即利用文件节点指定设备文件的特定处理位置，并且可以读取设备的处理位置。字符设备驱动程序的结构非常简单，与应用程序的呼叫具有1：1的对应关系，因此多数硬件表现为字符设备驱动程序。总之，编写嵌入式Linux设备驱动程序的开发人员一定要理解并且能够熟练使用字符设备驱动程序。

2. 块设备驱动程序

除了特殊情况（分区），应用程序中不直接使用块设备驱动程序。块设备驱动程序是支持文件系统的结构，以文件系统实现与应用程序的连接。

为了提高块设备的处理效率，内核利用内部缓存来进行处理。块设备驱动程序称做块的原因在于体现文件系统的硬件不使用流处理方式，而是采取块单位的处理方式。通常使用扇区（sector）单位，为了实现多个块设备之间的互换，Linux经常使用的块大小单位为1kByte。块设备驱动程序的特点之一是具有字符设备驱动程序的特点，即可进行块处理，也能够进行流处理。它与字符设备驱动程序相同，利用open( )、close( )、read( )、write( )等文件处理用函数进行连接。字符设备驱动程序中存在大量控制硬件输入输出的函数，多数的块设备驱动程序中既存在处理硬件输出输入的函数，也存在块设备驱动程序的固有方式。

3. 网络设备驱动程序

字符设备驱动程序和块设备驱动程序都可以采取普通文件形式，然而应用程序却不能直接处理网络设备驱动程序。它被设计成与内核内部的网络协议栈挂接使用的结构。因此网络设备驱动程序没有设备文件。网络设备驱动程序的结构只在网络设备驱动程序中使用，这是以文件系统为基础的设备驱动程序结构和网络设备驱动程序之间的差异。应用程序不能直接使用网络设备驱动程序。只有ifconfig等特殊的程序才能利用系统呼叫方式，呼叫和控制网络设备驱动程序的部分函数。另外，网络设备驱动程序中不仅是硬件设备，还存在软件类型的设备驱动程序，还可以与其他设备驱动程序联合使用。

2.3.4　整合型设备驱动程序（2.6内核）

前面介绍了区分Linux设备驱动程序类型的最基本的方法。然而，仔细观察Linux内核中配置的设备驱动程序，除了字符、块及网络外，由硬件的运行方式形成了新的管理层。以串行（serial）设备为例子，不是简单注册了字符设备，而是按照标准化串行设备的结构体类型注册了串行设备。因此，很难看到字符设备驱动程序的痕迹。

接着查看注册串口设备的结构体。内核内部存在控制串行设备的主核心函数。用于控制串行设备的多种功能函数被注册在该结构体的各种域（field）——成员变量上。设备驱动程序编写响应该调用的函数。即使不利用上述结构体形式和函数进行注册，也可以编写执行串行处理的设备驱动程序，但是不能使用应用程序上运行标准化串行设备的方法，否则会使设备驱动程序进入死循环。

为了轻松控制多种类型的硬件，除了Linux中的基本类型，还进一步体系化调整了设备驱动程序的结构，2.6内核中形成了整合型设备驱动程序。整合型设备驱动程序中定义了下面列出的几种格式设备驱动程序和相关的概念。所谓整合型设备驱动程序指统一了内核内部管理体系中的处理方式，而不是统一了设备驱动程序的组成。

- device　指实际存在的硬件；
- driver　指控制设备的软件；
- bus　指连接硬件的总线；
- class　指硬件的类型；
- interface　指理论上的接口。

内核源代码的Documentation/driver-model/中详细说明了整合型设备驱动程序的技术内

容，与整合型设备驱动程序相关的结构体和函数声明在include/linux/devices.h上。

（1）Sysfs

Sysfs是2.6内核中新增的与整合型设备驱动程序相关的文件系统。在Shell程序上可以查看该文件系统中设备驱动程序的结构相关信息，并且间接显示内核对设备驱动程序的目录树管理结构。通常，使用下列的mount结构。

```
[root@]#mount -t sysfs sysfs /sys
```

（2）Device

设备指内核中运行的硬件对象。一个设备驱动程序可以控制一个设备，也可以控制多个设备。此时要分别管理设备和设备驱动程序。驱动程序的实际控制对象称为设备。由struct device定义和登记设备，由内核管理信息。

（3）Driver

驱动程序指真正控制设备的软件。一个驱动程序可以控制一个以上的设备，同样一个设备也可以被多个驱动程序所控制。为了运行设备，利用struct device_driver定义和注册驱动程序的特性，而由内核来管理该信息。

（4）Bus

所有设备都通过称做总线（bus）的电子信号线连接到处理器上。Bus包括直接连接到处理器的bus和间接连接的bus。在一个系统中包含复合运行的多个总线，而设备归属一条总线。驱动程序为了占有各个设备，占有控制总线的设备，通过总线占有设备。常见的总线类型包括PCI、IDE、USB、PCMCIA、SCSI、I2C等。内核也需管理总线，它利用称做struct bus_type的结构体定义和登记总线，而由内核来管理该信息。

（5）Class

与设备和驱动程序不同，依据处理方式对硬件进行分类的称做类（class）。例如，USB键盘是一种输入设备，也是USB设备。一个设备也要根据使用目的分别进行管理。为了分类管理设备，使用struct device_class结构体定义和登记特性，而由内核来管理该信息。

（6）Interface

不仅要按类型对设备进行分类，也要根据输出输入的处理方式进行分类。例如，接收用户输入信息的代表性设备为鼠标。应用程序中使用鼠标。类似的设备还包括触摸屏（touch screen），在应用程序中把触摸屏处理为鼠标。但是这两种设备的体现方式和输入方式都不同。为了管理不同的输入输出处理方式，使用struct device_interface结构体定义和登记特性，而由内核来管理该信息。

2.3.5 设备驱动程序的层次

前面已提到，设备驱动程序以基本的设备驱动程序结构为基础，根据硬件的类型分为多种注册方式。因此要想编写设备驱动程序，必须理解已形成的设备驱动程序。目前，相关的文字资料较少，要想学习设备驱动程序具有一定的难度。幸运的是内核源代码中的设备驱动程序源代码都依据上述特点形成了目录。因此，通过查看各个目录，就能够理解已搭建的设备驱动程序层。在这里简单介绍已形成的几种类型设备驱动程序层。

1. 各种类型共同体现的内容

通常，较为完整的设备驱动程序包括以下几个基本要素。

（1）接口（interface）

体现连接系统的其他系统相关接口。整合型设备驱动程序中表示为总线，也表示为接口。例如，IDE、PCI、USB及I2C等。

（2）接口十进制

利用接口连接，体现处理外部设备和内核之间相互通信的十进制。

（3）连接在接口运行的设备

体现接口的客户端，IDE等设备的对象为HDD、CDROM，对于PCMCIA成为PCMCIA卡的客户设备体现位置。该位置多连接到其他类型的设备驱动程序上。

（4）硬件检测层

体现接口控制器的检测和接口连接设备的检测内容。

（5）Proc文件系统

支持/proc文件系统。

（6）支持Devfs 文件系统

不是盲目分配设备文件，而是实际存在某种设备时，使设备驱动程序支持相应设备文件。

（7）PROCESS API

体现支持其他应用程序或其他设备驱动程序之间的相互连接或相应设备驱动程序的API函数（也有不存在该部分的时候）。

（8）核心

整体上体现上述所有项的中心位置，在这里注册各个设备的相应驱动程序。

当然，上述分类方法并不是绝对的。也可以说是作者的私人看法。但是，仔细分析Linux源代码，基本符合上述分类法。另外，为了将上述项系统化而形成的便是前面所提到的整合型设备驱动程序模型。

2. 具有代表性的设备驱动程序层

Linux内核提供了非常多的设备驱动程序层，在这里列举了几种最具有代表性的例子。此外，还有不少设备驱动程序层，设备驱动程序的编写作者决定该驱动程序的特性，并利用相应层所提供的注册函数处理硬件。

（1）串行，tty

可轻松处理终端设备和串行设备的标准化设备驱动程序层。

（2）声音

使轻松处理与音响相关的声卡或MIDI等设备的标准化设备驱动程序层。

（3）USB

使轻松处理USB主机控制器和USB相连客户端设备的标准化设备驱动程序层。

（4）帧缓存器（Frame bumper）

使轻松处理VGA等Graph设备的标准化设备驱动程序层。

（5）IDE

使轻松处理连接在IDE接口上的块设备的标准化设备驱动程序层。

（6）PCI

使轻松处理PCI主机接口的标准化设备驱动程序层。

2.3.6 设备驱动程序源代码的结构及介绍

Linux内核包含了大量的设备驱动程序源代码，它几乎可以控制目前运行在其他系统中的所有硬件。特别是使用嵌入式Linux系统的人们，既使不去设计Linux内核，也可以通过分析相应硬件的设备驱动程序源代码减少误操作，快速达到开发目的。对于Linux内核源代码中的设备驱动程序，以功能系统分类，并创建目录后进行管理。

1. 设备驱动程序源代码

表2-3 设备驱动程序目录

目录	说明
drivers/acorn/	ACORN系统使用ARM核心的32位处理器，与普通的PC相似，包含了控制ACORN系统用硬件的设备驱动程序源代码
drivers/acpi/	ACPI就是Advanced Configuration and Power Interface的缩写，意思是“高级配置与电源接口”，这是英特尔、微软和东芝共同开发的一种电源管理标准，包含使其他设备驱动程序和内核使用ACPI功能的标准化函数的源代码和控制ACPI相关硬件的设备驱动程序源代码
drivers/atm/	ATM把传输数据分为53字节的信元或数据包，是通过虚拟通信信道传输数据的专用异步传送模式（dedicatedconn ection）交换技术，该目录中包含了控制ATM功能硬件的设备驱动程序源代码
drivers/base/(2.6)	2.6内核中新增的目录，包含管理设备和设备驱动程序的驱动程序新模式源代码，整合型设备结构（unified device structure）源代码
drivers/block/	包含Linux内核中支持块设备的基本函数源代码，还有类似于具有块设备特性的硬盘等控制硬件的设备驱动程序源代码
drivers/bluetooth/	蓝牙，一种无线通信的标准，由英特尔、IBM、诺基亚、爱立信、东芝五家公司联合开发，蓝牙的目标是要提供一种通用的无线接口标准，多用在手机、电脑、PDA等的通信上。该目录中包含蓝牙的网络函数源代码和控制蓝牙通信硬件的设备驱动程序源代码
drivers/cdrom/	带有块设备驱动程序特性的辅助存储器中，包含可控制CD-ROM驱动器的设备驱动程序源代码。该目录中只控制除ATAPI或SCSI接口外的CDROM专用设备
drivers/char/	包含Linux内核中支持字符设备的基本函数源代码，具有字符设备特性的多数硬件设备驱动程序的源代码
drivers/char/agp	包含支持AGP用显卡的设备驱动程序源代码
drivers/char/drm	DRM（Direct Rendering Infrastructure）是Xfree86中使用的设备驱动程序，包含支持DRM的设备驱动程序的源代码
drivers/char/drm-4.0(2.4)	2.6内核中删除了该目录，包括支持DRM的另一设备驱动程序的源代码
drivers/char/ftape	包含磁带流设备用设备驱动程序源代码
drivers/char/ip2	包含IntelliPort的多端口串行设备驱动程序的源代码
drivers/char/ipmi	包含支持IPMI（Intelligent Platform Management Interface）的设备驱动程序的源代码
drivers/char/joystick (2.4)	2.6内核中移到drivers/input目录上，包含支持joystic的其他设备驱动程序的源代码
drivers/char/mwave	包含与IBM的Winmodem相似的Linux用设备驱动程序的源代码
drivers/char/pcmcia	包含PCMAIA总线上运行的串行设备驱动程序的源代码

续表

目录	说明
drivers/char/rio	包含Specialix Rio 多端口串行设备驱动程序的源代码
drivers/char/watchdog	包含支持watchdog的设备驱动程序的源代码
drivers/cpufreq/(2.6)	2.6内核中新增的目录，包含可调整CPU时间控制设备驱动程序的源代码
drivers/dio/	包含支持HP300系统中使用的“DIO”扩展总线bus的设备驱动程序目录
drivers/eisa/(2.6)	EISA是为使用80386/80486及奔腾处理器的电脑把ISA标准扩展为32位接口的标准总线规则
drivers/fc4/	光纤通道是电脑与设备之间以最高1Gbit/s的速度传送数据的技术。把电脑服务器连接到共享存储器上，或者连接存储控制器和设备的规格。包含控制光纤通道的硬件设备驱动程序源代码
drivers/gsc/(2.4)	包含在HP-PARISC工作站的GSC总线上运行的硬件设备驱动程序源代码
drivers/hil/(2.4)	HIL（Human Interface Loop）用在 惠普（Hewlett Packard）的PARISC基本设计上，是与8通道型USB相似的控制器。包含支持HIL的设备驱动程序源代码。2.6内核中删除了该目录
drivers/hotplug/(2.4)	电源认可的系统上，删除或插入外设的称做热插线（hotplug）。包含支持PCI设备的热插线的设备驱动程序源代码
drivers/i2c/	I2C称做Inter-IC，是连接直接回路的双线双向直连总线。包含支持I2C总线的基本函数的源代码和控制I2C硬件的设备驱动程序源代码
drivers/ide/	IDE是主板数据总线和磁盘存储设备之间使用的总线规格，基础为IBM 电脑的ISA 16位总线标准。目前，主要使用称做EIDE的IDE功能强化版。包含支持IDE总线的基本函数源代码和支持IDE总线的硬件设备驱动程序源代码
drivers/ieee1394/	包含支持IEEE1394总线的基本函数源代码和支持IEEE1394总线的硬件设备驱动程序源代码
drivers/input/	包含驱动游戏杆、键盘及鼠标等与输入设备相关硬件的设备驱动程序源代码
drivers/isdn/	ISDN（Integrated Services Digital Network）铜类电话线外，实现高速传送的CCITT/ITU标准规格。包含驱动ISDN通信设备的设备驱动程序源代码
drivers/macintosh/	包含Apple的Macintosh设备驱动程序源代码
drivers/mca/(2.6)	MCA（Micro Channel Architecture）微通道机构是IBM PS/2计算机中的总线设计方案，是扩展卡和相关设备之间的接口。包含支持MCA总线的基本函数源代码和支持MCA总线的硬件设备驱动程序源代码
drivers/md/	包含支持软件RAID或硬件RAID及线型文件系统等多个模块设备的函数源代码和相应硬件设备驱动程序源代码
drivers/media/	包含驱动无线通信设备和视频设备的设备驱动程序源代码
drivers/message/	包含驱动Fusion MPT设备和I2O总线处理硬件的设备驱动程序源代码
drivers/misc/	包含驱动IBM RSA service processor设备的驱动程序源代码
drivers/mtd/	嵌入式设备中Flash、RAM等在芯片上体现静态文件系统的MID驱动程序源代码
drivers/net/	包含控制网卡设备的驱动程序源代码
drivers/nubus/	NuBus是MIT开放的定义为Eurocard（9U）的总线，从macintosh到Performa都可以使用。包含支持NuBus的函数源代码和控制硬件的设备驱动程序源代码
drivers/oprofile/(2.6)	OProfile是使用最新电脑内置的特殊硬件分析系统性能的监视工具。包含支持OProfile的函数源代码和控制硬件的设备驱动程序源代码。2.6内核开始支持该目录
drivers/parisc/(2.6)	整合管理PCI、PCMCIA、EISA、GSC、ISA等各种总线，是2.6内核新增的目录。包含各种总线选项处理函数的源代码
drivers/parport/	支持打印端口等并口设备的驱动程序源代码
drivers/pci/	包含支持PCI总线的函数源代码和控制PCI主机硬件的设备驱动程序源代码
drivers/pcmcia/	包含支持PCMCIA总线的函数源代码和控制PCMCIA主机硬件的设备驱动程序源代码
drivers/pnp/	包含支持Plug In Play的函数源代码
drivers/s390/	驱动IBM的ESA/390和z串行设备及虚拟TTY设备和多控制台设备的设备驱动程序
drivers/sbus/	包含支持由SUN开发的SBUS总线的设备驱动程序
drivers/scsi/	包含支持SCSI总线控制和SCSI块设备的设备驱动程序
drivers/serial/(2.6)	包含支持Serial接口设备的设备驱动程序
drivers/sgi/(2.4)	包含支持SGI的系统的设备驱动程序。2.6内核不支持该目录，已被删除

续表

目录	说明
drivers/sound/(2.4)	包含Sound和相关设备驱动程序。2.6内核中不在drivers目录中，而是在sound中进行管理
drivers/tc/	支持HP（原来的DEC）增强通道（TURBOChannel）总线的设备驱动程序
drivers/telephony/	包含与VoIP（voice-over-IP）相关的设备驱动程序
drivers/usb/	包含USB主机管理员和USB客户端设备的驱动程序
drivers/video/	包含视频卡设备的驱动程序
drivers/zorro/	支持Amiga系统Zorro总线的设备驱动程序
sound/(2.6)	包含与Sound相关的设备驱动程序。2.4内核中由drivers/sound进行管理

2. 搜索设备驱动程序源代码的方法

开发Linux设备驱动程序时，通常会修改已形成的设备驱动程序的部分内容，而不是重新编写源代码。这样可以缩短开发时间。

然而不容易找到相应的源代码。多数情况下源代码的文件名和设备名称并不是1：1配置的。在这里介绍几种作者常用的方法，供读者参考。

3. 查找设备模型名称

对于PCI设备，最好在Linux内核运行的状态下寻找代码。启动内核后，利用Ispci实用程序在Shell上查找相应设备的模型名称。通常，使用Ispci就可以找到设备开发商和设备模型名称，然后记录两个内容。

```
[root@]  #lspci
```

该命令参考了/usr/share/pci.ids，该文件维持最新状态。

不是PCI设备时，通常在嵌入设备中配置Linux内核。此时，需要预先知道控制器的模型名称。

4. 搜索Drivers目录

利用已得到的模型名称搜索drivers目录。例如，对于称做cmd980的设备，可运行下列句柄。

```
[root@] #cd drivers
[root@] #grep cmd980 * -r
```

若此时发现的位置就是实际设备驱动程序的源代码，那么相应源代码基本就是设备驱动程序。

5. 2.6内核中找到Kconfig

2.6内核中在帮助和输出的信息中包含着Kconfig。此时可以确认CONFIG_变量。对于CONFIG_XXX可以查找相应选项的Makefile。例如，对于CONFIG_BLK_DEV_SIIMAGE要确认相应的对象，该对象的源代码就是相应的设备驱动程序源代码。

6. 搜索2.4内核的Config.in

2.4内核中Config.in通常包含在输出的信息里。此时同样要确定CONFIG_变量。对于

CONFIG_XXX可以查找相应选项的Makefile。例如，首先确认CONFIG_BLK_DEV_SIIMAGE相应的对象。该对象的源代码就是相应的设备驱动程序源代码。

7. 搜索2.4内核的Documentation/Configure.help

对于2.4内核，搜索内核编译帮助文件Documentation/Configure.help，也是一种方法。

```
[root@] #cd Documentation/
[root@] #grep CMD680 Configure.help
```

此时也要确认CONFIG_变量。对于CONFIG_XXX可以查找相应选项的Makefile。例如，对于CONFIG_BLK_DEV_SIIMAGE，要确认相应的对象。该对象的源代码就是相应的设备驱动程序源代码。

8. 其他

使用前述方法还是不能搜索到源代码时，可能是不存在支持的设备驱动程序，或者只支持功能相同的其他模型。此时需要得知功能相同的控制器模型名称，再采取相同的方法查找源代码。

没有已编写的设备驱动程序时，只能参考功能相似的其他设备驱动程序直接编写代码。