【嵌入式应用技术丛书】

嵌入式Linux

驱动模板精讲与项目实践

林锡龙◎编著

- 一线研发实战经验积累，所有技巧及讲解使用的工具都可以直接应用到实际开发工作中。
- 采用模板的方式对分散的各种驱动知识点进行讲解，所有模板都可以直接引用。
- 对每个知识点都提供实际案例，从模块的原理介绍，到系统层次的分析，图文并茂，分析透彻。
- 提供大量的驱动例程，读者可以直接运行调试，快速应用到实际开发中。

例程源代码、目标文件，各种相关工具

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

嵌入式应用技术丛书

嵌入式 Linux 驱动模板精讲与项目实践

林锡龙　编著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书从实用的角度对嵌入式 Linux 驱动设计开发从理论到实践做了比较深入的介绍，以 Mini2440 开发板为基础，对各种常见驱动开发进行模板化设计训练，各种模板可以直接应用到实际项目开发中，其中的一线研发技巧和案例分析可以供专业特训和广大嵌入式 Linux 开发者实战使用。

本书共分为 13 章，书中介绍的各种开发技巧对实际应用有很大的借鉴意义，在各个知识点介绍中穿插实际项目的经验分享，包括实际研发的管理经验介绍，对驱动开发初学者和实际设计开发人员有很大益处，也可作为一线研发公司的岗前培训教程。

本书重点在于内核驱动的讲解，以及模板式开发的训练，力求完全揭晓各种 Linux 开发中的技巧和模糊点，是一本比较实用的驱动开发训练教程。

图书在版编目（CIP）数据

嵌入式 Linux 驱动模板精讲与项目实践 / 林锡龙编著. —北京：电子工业出版社，2014.5
（嵌入式应用技术丛书）
ISBN 978-7-121-23082-0

Ⅰ.①嵌… Ⅱ.①林… Ⅲ.①Linux 操作系统—程序设计 Ⅳ.①TP316.89

中国版本图书馆 CIP 数据核字（2014）第 083991 号

策划编辑：陈韦凯
责任编辑：康 霞
印 刷：北京京师印务有限公司
装 订：北京京师印务有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1 092 1/16 印张：19.75 字数：506 千字
版 次：2014 年 5 月第 1 版
印 次：2014 年 5 月第 1 次印刷
印 数：3 500 册 定价：59.00 元（含光盘 1 张）

前　言

一、行业背景

嵌入式 Linux 驱动开发涉及的知识点比较多，要求开发者掌握的技能也比较多，且内核知识点比较分散，对于初学者来说门槛比较高，而嵌入式 Linux 在各行各业中已被广泛应用，在物联网、通信行业、航空航天、消费电子、汽车电子等行业中急需掌握嵌入式 Linux 软件开发的研发人员。近年来，随着嵌入式应用越来越广泛，嵌入式 Linux 开发变得更加复杂，嵌入式 Linux 驱动开发已经成为嵌入式应用领域的一个重大课题。

二、关于本书

本书大部分内容基于专业培训机构特训的讲稿。在专业培训提倡的嵌入式 Linux 驱动的模板式教学中取得了很好的效果，在此之上结合一线研发经验对驱动开发进行战略性指导，其中很多关键点是作者花了很长时间实实在在整理出来的，旨在揭晓嵌入式 Linux 驱动中的各种机制，达到知其然且知其所以然的目的。

本书各章内容说明如下。

第 1 章为驱动总论，是驱动开发在高层次上的介绍。

第 2 章介绍的内核编程及基础知识点是驱动开发的基础，对驱动开发涉及的 Linux 内核中的各个知识点进行介绍，旨在扫清对 Linux 内核阅读的障碍。

第 3 章介绍驱动模块开发，涉及各种模块移植中常见的设备方法，其中各个模板可以直接应用到实际研发中。

第 4 章结合 Linux 操作系统讲解互斥机制在各种情况下的使用，重点分析各种机制的使用注意事项。

第 5 章重点介绍 Linux 中断的上下部机制及 Linux 提供的各种接口，强调中断程序设计的基本理念及设计手段。

第 6 章介绍 Linux 驱动中字符设备驱动的高级设备方法。

第 7 章在 Linux 子系统的层面上介绍各种高级设备驱动。

第 8 章重点介绍如何使用虚拟文件系统，这些实用技巧是一线研发的经验总结。

第 9 章对设备驱动模型各个元素进行讲解，并介绍如何一步步手动建立设备驱动模型模板。

第 10 章手把手带领读者建立最基本的文件系统，并制作各种常见的文件系统，其中穿插介绍各种实用技巧及实际研发工具。

第 11 章介绍一线研发人员使用的开发调试技巧，重点强调实用。

第 12 章结合 V 字形开发模型在嵌入式 Linux 驱动开发中的具体实施进行总结。

第 13 章介绍嵌入式 Linux 设备驱动编程规范。

本书附带光盘一张，包含书中例程的源代码、目标文件，以及各种相关的工具，光盘按照书中各个知识点建立相应文件夹存放。其中各个知识点配套的例程可以直接在 Mini2440 开发板上运行。驱动相应例程含有相应的例程配套使用。各个例程也作为相应 Linux 驱动开发的模版，可以直接修改并应用到具体项目开发中。

三、本书特色

- 一线研发实战经验积累，所有技巧及讲解使用的工具都可以直接应用到实际开发工作中。
- 采用模板的方式对分散的各种驱动知识点进行讲解，所有模板都可以直接引用。
- 对每个知识点都提供实际案例，从模块的原理介绍，到系统层次的分析，图文并茂，力求分析透彻。
- 提供了大量的驱动例程，这些例程可以快速应用在实际开发中，读者可以直接运行调试。
- 结合实际研发工作对开发过程中的思考进行总结，重在实用。

四、作者介绍

本书由林锡龙编著，编著者在写作过程中查阅了大量开源工具及互联网上的资料，对各种资料的作者不能一一列举，在此表示谢意。

由于时间仓促，书中程序和图表较多，错误之处在所难免，请广大读者批评指正。作者电子邮箱：wlxl_1204@163.com。

编 著 者

目　录

第 1 章　驱动总论

1.1　总论

1.1.1　驱动在内核中的比例

在 Linux 内核中，驱动程序的代码量占有相当大的比重。图 1.1 是一幅 Linux 内核代码量的统计图（单位：行数），对应的内核版本是 2.6.29。从图中可以很明显地看到，在 Linux 内核中驱动程序（drivers）的代码超过了 500 万行，所占的比例最高。

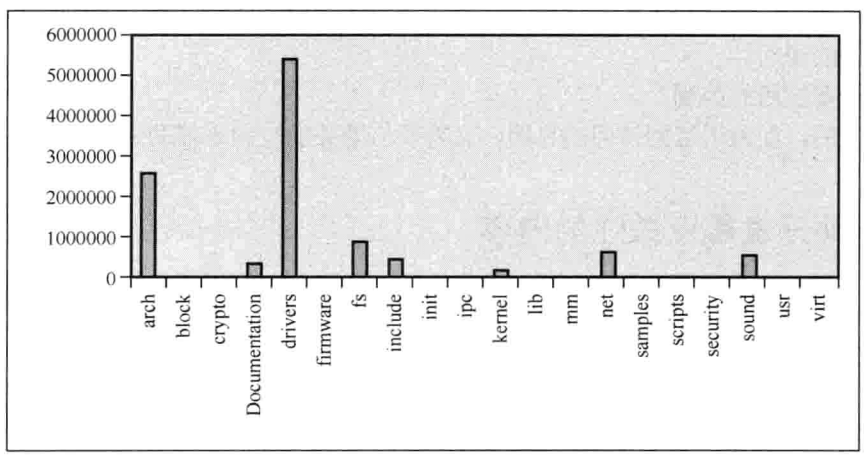

图 1.1　Linux 内核代码量统计图

1.1.2　驱动开发需要具备的能力

目前，Linux 软件工程师大致可分为两个层次。

（1）Linux 应用软件工程师。Linux 应用软件工程师主要利用 C 库函数和 Linux API 进行应用软件的编写。

（2）Linux 固件工程师。Linux 固件工程师主要进行 Bootloader、Linux 的移植及 Linux 设备驱动程序的设计。

一般而言，对固件工程师的要求要高于应用软件工程师，而其中的 Linux 设备驱动编程又是 Linux 程序设计中比较复杂的部分，究其原因，主要包括如下几个方面。

（1）设备驱动属于 Linux 内核的部分，编写 Linux 设备驱动需要有一定的 Linux 操作系统内核基础。

（2）编写 Linux 设备驱动需要对硬件原理有相当的了解，大多数情况下是针对一个特定的嵌入式硬件平台编写驱动的。

（3）Linux 设备驱动中广泛涉及多进程并发的同步、互斥等控制，容易出现 bug。

（4）由于 Linux 设备驱动属于内核的一部分，所以它的调试也相当复杂。

Linux 设备驱动的开发要求比较高，要求开发人员掌握一定的硬件知识、Linux 内核技能、操作系统并发概念和较高的软件编程驾驭能力，Linux 设备驱动程序作为内核的一部分运行，像其他内核代码一样，如果出错将导致系统严重损伤。一个编写不当的驱动程序甚至会导致系统崩溃，导致文件系统破坏和数据丢失，所以对 Linux 设备驱动开发比对应用程序要求高得多。

Linux 驱动的编写涉及如下主题。

（1）内核模块、驱动程序的结构。

（2）驱动程序中的并发控制。

（3）驱动程序中的中断处理。

（4）驱动程序中的定时器。

（5）驱动程序中的 I/O 与内存访问。

（6）驱动程序与用户程序的通信。

实际内容错综复杂，掌握起来也有难度，但从本质上来说，这些内容仅分为两类。

（1）设备的访问。

（2）对设备访问的控制。

前者是目的，而为了达到访问的目的，又需要借助并发控制等辅助手段。

1.1.3 驱动开发重点关注的内容

初看起来 Linux 设备驱动开发涉及的内容很多，而需要实现驱动的设备又千差万别，其实质主要包括以下几点内容。

（1）对驱动进行分类，先归纳为是哪种类型的驱动，归类正确再利用内核提供的子系统进行开发，这时往往会发现很多通用的事情内核已经做了。一个优秀的驱动工程师应该最大限度地利用内核的资源，因为内核已经实现的毕竟稳定性强、可移植性高。

（2）找到内核提供的子系统后，接下来就是要制作该子系统对该类设备提供的表征，也就是描述该类驱动的结构体，然后定义这个结构体，把必要的数据进行初始化，最后调用该内核子系统提供的接口函数提交给内核管理。这是大部分驱动程序开发的战略流程。

（3）明确子系统已经做了什么，需要在自己驱动中实现哪些内容，通常的做法是找一个接近的驱动程序进行修改，而不是一行一行地对代码进行编写。到内核中找接近的驱动例程是一个又快又好的方法。这些例程基本上都提供接口如何使用、调用流程等，借鉴已有例子可以避免低级错误。

（4）以上都是与内核接口有关的，驱动另一个涉及的就是芯片手册，这个与做其他嵌入式软件一致，故对从单片机软件开发或者其他操作系统软件开发转过来做 Linux 驱动开发的人员来说，这部分是一个强项。

（5）驱动的另外一个内容就是协议，包括各种嵌入式总线协议，从简单的 SPI 到复杂的 PCI

或者 USB 等。协议的基本知识是需要掌握的，好在内核对各种常见协议都是以子系统的形式提供的，在子系统中做了大部分共性工作，所以大大降低了驱动开发的工作量。

综上所述，学好驱动开发，一个重要的方面就是对内核的学习，熟悉内核的组织和思维方式。

1.2 驱动理论与思想

Linux 对驱动的管理有自身的一套方法并提供相应的机制，但其具体实现可以由开发者自己决定。这一点可能比较抽象，但真正接触到实践之后，你会发现相同的一个功能，如点灯驱动，有很多实现方式——简单字符方式、cdev 方式、mics 实现、设备驱动模型等，对于设备的管理实现也有多种方法，只要你喜欢你可以写出很多种方式的驱动。正因为存在这么多种形式的驱动，初学者最初会比较茫然，进而觉得复杂，再后来就会感觉很灵活，然后感觉很多事情 Linux 已经做好了，你要做的就是熟悉一下它所提供的机制，然后正确调用其所提供的接口实现功能。

从 2.4 到 2.6，Linux 内核在可装载模块机制、设备模型、一些核心 API 等方面发生较大改变。特别是引入设备模型，设备模型是一个难点，将由后续专门章节介绍。本章作为驱动理论将在一个更高的角度来总结内核对各种驱动所提供的相似点。之所以先提及设备模型，个人觉得真正理解 Linux 的驱动应该以设备模型为主线，设备驱动模型的提出涉及驱动管理方式的变化，如从原来手动创建设备文件到自动创建设备文件这个变化 Linux 引入了 Udev 系统，再展开来说，从设备模型中引入各种子系统、总线等概念，对这些进一步深入可以发现 Linux 提供各种驱动的 core，如 I2C-core，input-core，rtc-core，serial_core 等，如果打开阅读内核代码，你会发现在内核代码 driver 下对每种设备都进行了分类，而且每个文件夹下基本都有一个 core 文件，这些 core 文件已经对相应类型的驱动进行通用属性和行为的封装。市场上很多 Linux 驱动的书籍基本上都会对每种驱动以一个章节进行描述。在某种程度上讲，随着内核的发展，在驱动上尽量将各种类型的设备进行抽象，由内核里底层的代码，如总线驱动或此类设备共用的核心模块来实现共性的工作，从而简化设备驱动开发。

设备模型的引入既在驱动设备管理上进行了分类又在文件系统上进行了统一。“一切皆文件”是 Linux 的设计思想，可以在 Linux 下的 sys 虚拟文件系统中看到 class 文件夹下有各种类型设备的相应文件夹。系统对设备的管理系统 Udev 也是按照 class 下的文件进行设备文件创建的。当然这也只是众多主线之一，还有总线、设备、驱动等概念，提出这些概念及响应机制的目的是为了简化设备驱动的开发、管理与维护。Linux 博大精深，且不断发展，新的特性不断涌现，知识点众多，为开发者和用户不断提供新的模块，我们必须抓住主线，逐个深入，长期积累。

现在不少嵌入式驱动开发者在项目中只采用基本的属性行为，很少或者基本用不上高级特性功能，采用 IOCTL 方式进行用户与驱动之间的对话式交互，通过编制上下对应的 IOCTL 命令即可完成绝大部分工作。在对驱动开发应聘者的面试中可以发现，很多应聘者对设备模型理解不多或者不清楚，这对深入理解新内核驱动显然是不够的。

1.2.1 分类思想

Linux 对各种各样的设备进行分类，一般分为字符型设备、块设备和网络设备 3 种。但在内核中对于一个具体的设备还有细分，而且整个内核是按照不同细分种类的设备来提供支持的。在系统运行之后查看一下 sys/class，图 1.2 所示为显示 sys/class 目录情况。

```
[xl.lin@localhost class]$ ls
backlight  i2c-adapter       iscsi_endpoint   misc           scsi_device   usb_device
bluetooth  iLO               iscsi_host       net            scsi_disk     usb_endpoint
dma_v3     infiniband        iscsi_session    pci_bus        scsi_generic  usb_host
firmware   infiniband_cm     iscsi_transport  pcmcia_socket  scsi_host     vc
graphics   input             leds             printer        tty           vtconsole
hwmon      iscsi connection  mem              raw            uio
```

图 1.2 sys/class 目录

并且可以看到各种类型的驱动。对于具体驱动来说，Linux 内核还保留了一些固定的主设备号，在内核代码 include\linux/Major.h 中定义了一些默认的类型主设备号。比如：

```
#define MISC_MAJOR          10  /*混杂型设备*/
#define SCSI_CDROM_MAJOR    11
#define MUX_MAJOR           11
#define XT_DISK_MAJOR       13
#define INPUT_MAJOR         13  /*输入型设备*/
```

对于一个新的驱动来说，首先必须要确定一下要把它当哪类设备来处理，也就是归为哪类设备，然后再看内核在这类设备中提供的支持，最后调用这类设备的接口函数进行处理。当然，如果一个新驱动有接近类似的，那么还是在接近的驱动基础上进行修改。

1.2.2 分层思想

对 Linux 的 input、RTC、MTD、I2C、SPI、TTY、USB 等诸多设备驱动进行分析，可以看到大致都是按照分层次来设计的。市面上很多介绍 Linux 驱动的书籍在章节编排上一般先介绍该类型的硬件知识，再介绍协议和相关操作，然后再对实际例子进行分析。实际上，协议和相关操作是归纳在内核相对应的 core 或者类似 core 相关的文件中的。这样对于一个具体设备来说并不需要对该部分再进行一次编写。这就是分层思想带来的好处。

Linux 内核分层的框架设计用到了面向对象的设计思想。在设备驱动方面，往往为同类设备设计了一个框架，而框架中的核心层则实现了该设备通用的一些功能。如果具体设备不想使用核心层的函数则可以对其进行实现重载。图 1.3 所示为驱动核心层与实例之间的关系图。

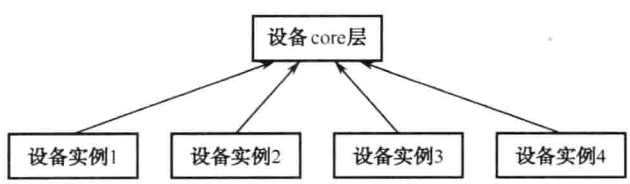

图 1.3 驱动核心层与实例之间的关系图

1. input 子系统层次

input 子系统整体系统框架如图 1.4 所示，整个系统是按一定层次进行划分和组织的。

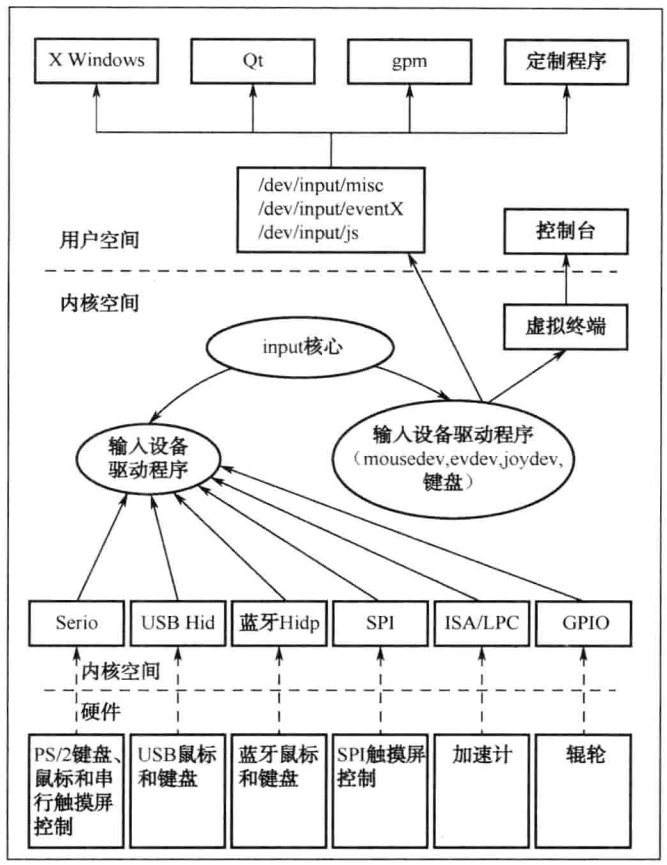

图 1.4　input 子系统整体系统框架图

再回到内核相应的路径文件夹 drivers/input 下，图 1.5 所示为看到的输入子系统相关文件。

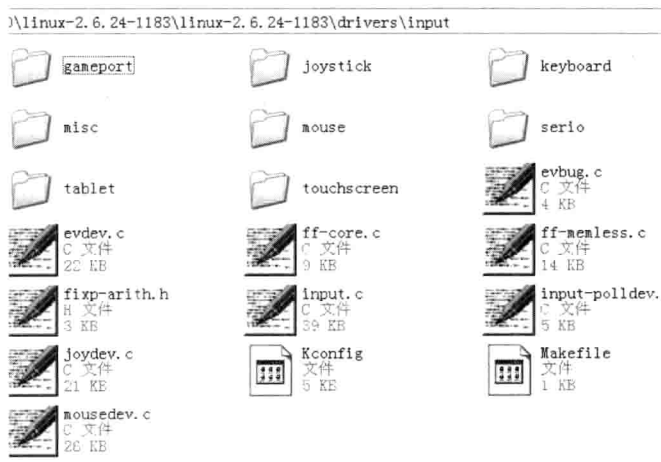

图 1.5　Linux 输入子系统相关文件

这些文件分为事件层、核心层和处理层。每个文件夹对应的是一种类型的输入设备，会产生事件报告到核心层，核心层决定调用哪些处理层对事件进行分发，最后发送到用户空间相应的设备文件下。图 1.6 所示为 input 子系统文件层次示意图。

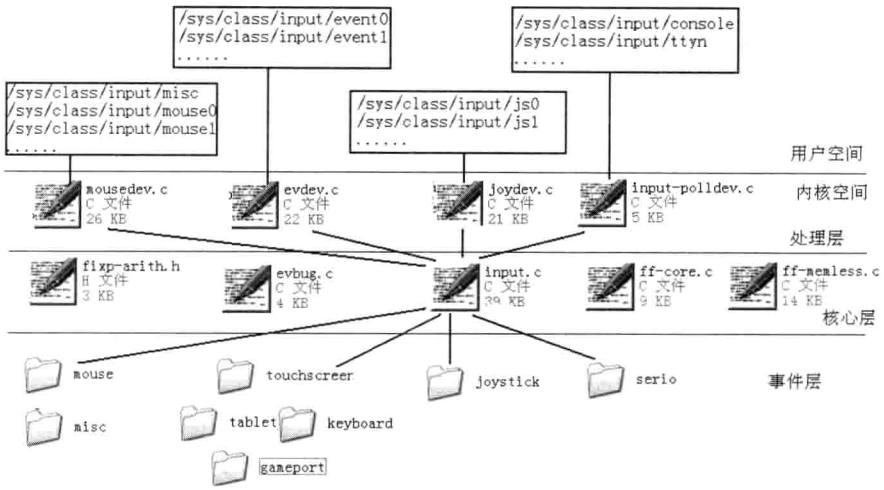

图 1.6　input 子系统文件层次示意图

这是在 Linux-2.6.24 版本下的标准内核，当然如果有厂家需要自己添加相应的驱动即可以在事件层中添加底层操作的一个文件夹，然后在处理层中添加一个相应的文件并添加到 input 系统中。

2．RTC 子系统层次

接着移到 drivers/rtc 目录下，该目录下的文件并没有像 input 下那样组织，但是按照前面提过的 Linux 很多地方采用面向对象的方法抽象共同点，然后在实例化的指导下我们可以看出，下面这些 rtc-×××.c 带有型号的文件都是实例化的驱动。图 1.7 所示为 Linux 中 RTC 实例化文件。

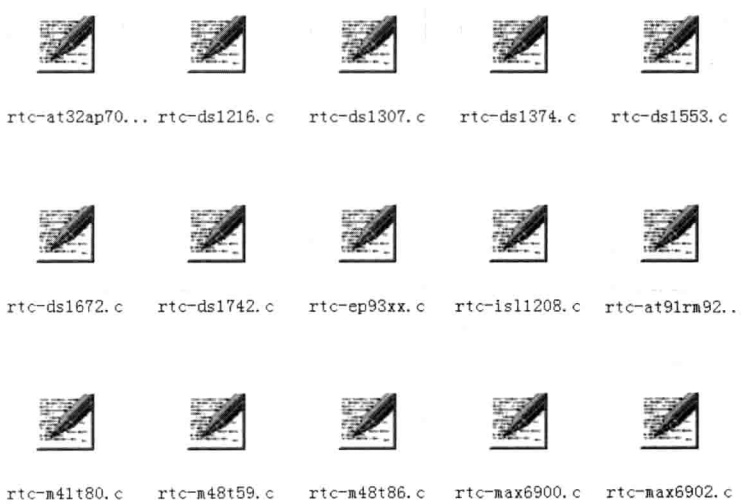

图 1.7　Linux 中 RTC 实例化文件

在 RTC 系统中使用的是 platform 总线，platform 总线在后面设备模型中会重点介绍。这里只要理解为设备模型中“总线，驱动，设备”中的一种总线即可，而这些实例化的 rtc-×××.c 文件就是模型中的驱动，需要实现的是 platform_driver 结构，设备则放在相应体系结构中对应的目录下，如 arch/arm/plat-s3c24xx/下，需要实现的是 platform_device。Linux 已经提供 platform 总线，驱动是对应某个型号的芯片，基本上可以在 drivers/rtc 目录下找到，那么驱动开发实际上就是完成系统中各个器件相应的设备描述即可，这就是驱动开发所要做的，而且熟悉之后工作量不是很大。至此，设备模型中的三个要素已经具备，只要驱动和设备注册成功且匹配，实际芯片就能正常工作。

下面看一个采用 Linux 的 RTC 系统运行的例子，如图 1.8 所示。

```
[root@FriendlyARM class]# ls
bdi             input         net           scsi_generic  vc
firmware        leds          ppp           scsi_host     video4linux
graphics        mem           pvrusb2       sound         vtconsole
hwmon           misc          rtc           tty
i2c-adapter     mmc_host      scsi_device   ubi
i2c-dev         mtd           scsi_disk     usb_device
[root@FriendlyARM class]# ls /sys/class/rtc/
rtc0
```

图 1.8 RTC 系统运行实例

这个 class 中的 rtc 是在哪儿实现的呢？看一下 drivers/rtc 下的 class.c 文件：

```
static int __init rtc_init(void)
{
    rtc_class = class_create(THIS_MODULE, "rtc");
    …
}
```

这是 RTC 系统核心层附属的 class 模块自动做的。也就是说，该子系统不但把 RTC 相关的共同操作提供了，而且还附带送上驱动模型这一套 linux-2.6 中的亮点特性。

接下来回到本节主题，图 1.9 所示为 RTC 系统的层次结构图。

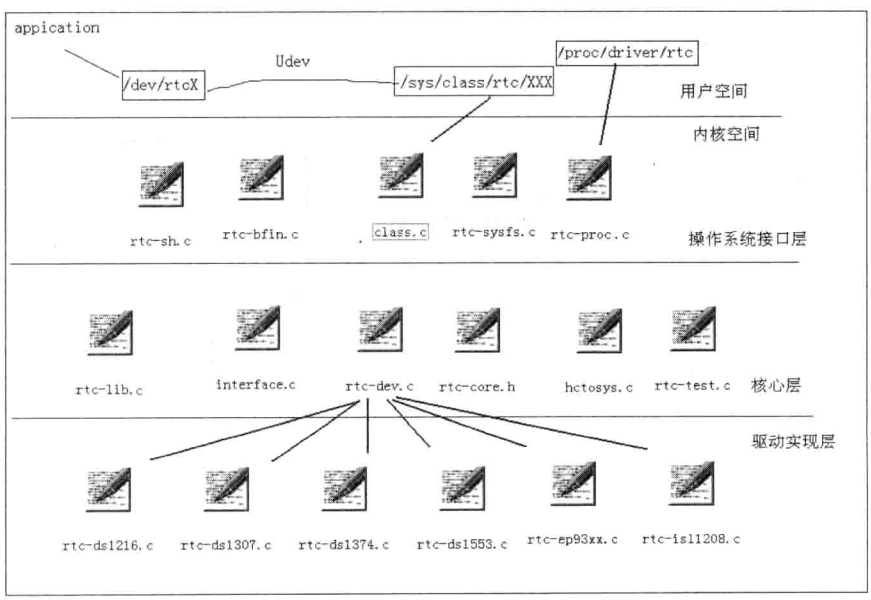

图 1.9 RTC 系统的层次结构图