ARM嵌入式Linux系统开发丛书

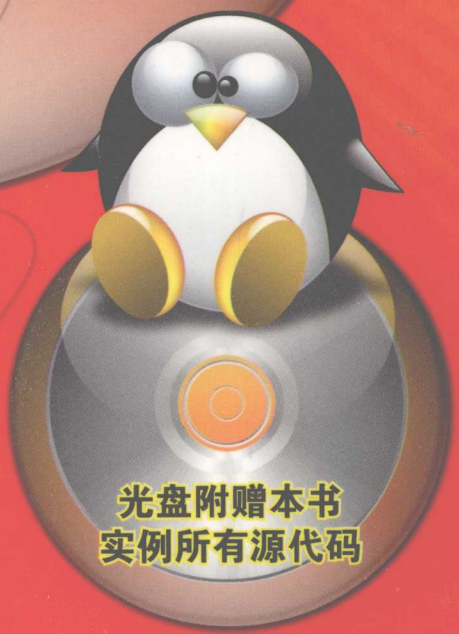
引领ARM嵌入式Linux编程应用开发新潮流！

ARM嵌入式Linux设备驱动实例开发

李亚锋 编著

本书内容包括：

- 按键设备驱动
- 触摸屏设备驱动
- MMC/SD卡设备驱动
- 网卡设备驱动
- LCD设备驱动
- USB设备驱动
- NAND Flash设备驱动
- PCI设备驱动

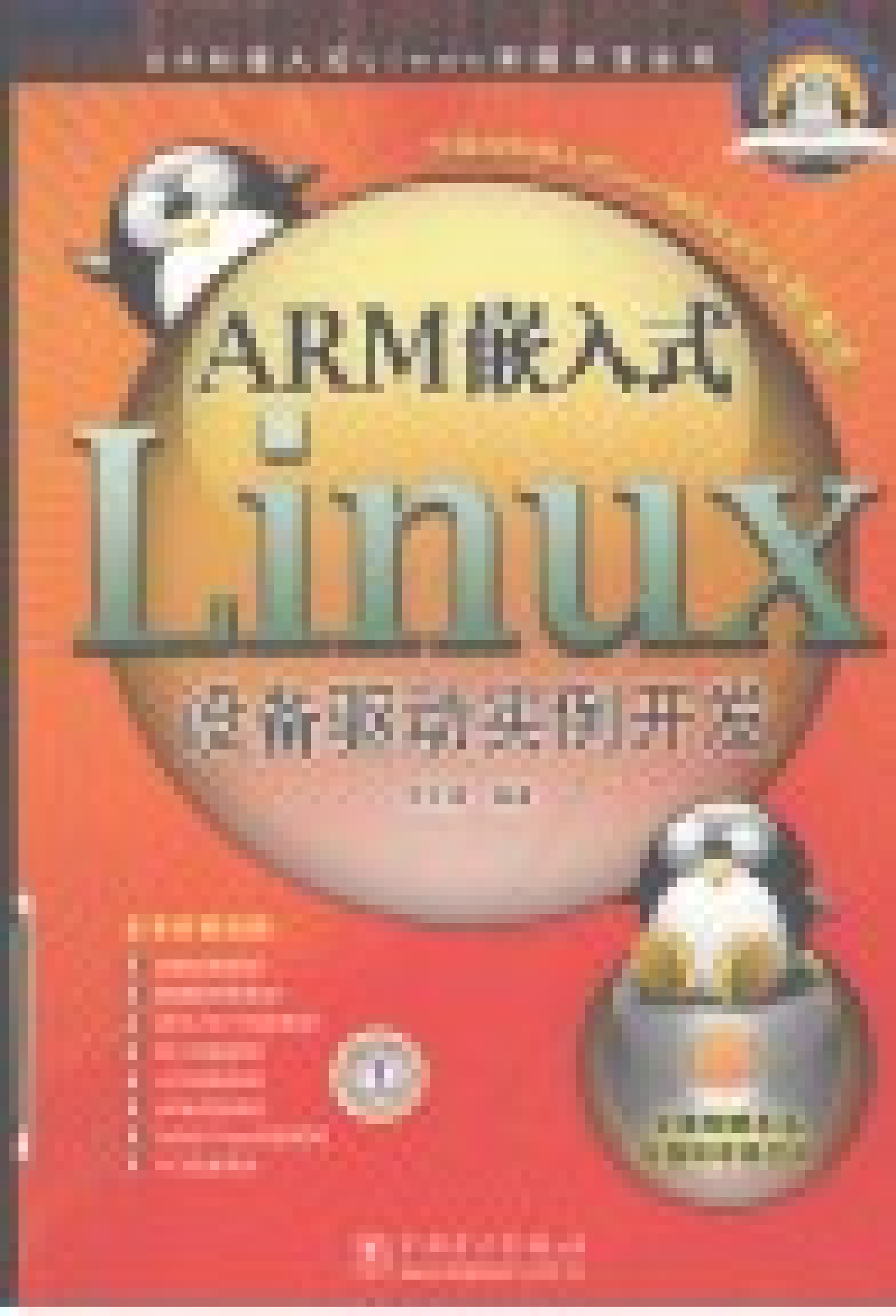


光盘附赠本书
实例所有源代码

中国电力出版社
www.infopower.com.cn

ARM嵌入式Linux

设备驱动实例开发



ARM嵌入式 Linux系统开发丛书

ARM嵌入式 Linux 设备驱动实例开发

李亚锋 编著

中国电力出版社
www.infopower.com.cn

内 容 提 要

Linux 是一个成熟而稳定的开放源代码操作系统，将 Linux 植入嵌入式设备具有众多的优点。本书以应用最广泛的新一代 ARM9 处理器为对象，专门分析 Linux 系统下设备驱动的设计和实现方法。本书以 8 个典型实例为研究对象，讲述基于 Linux 2.6.20 内核的设备驱动开发过程。这些实例几乎覆盖了 Linux 系统下常见的设备类型，其中包括按键设备、触摸屏、MMC/SD、网卡、Framebuffer、USB 和 PCI 设备等，本书提供了实例的所有源代码，便于读者分析和学习。

本书通俗易懂，可作为高等院校电子类、电气类、控制类、计算机类等专业本科生、研究生学习嵌入式 Linux 设备驱动开发的参考书或自学教材，也可供广大希望转入嵌入式领域的科研和工程技术人员参考使用，还可作为嵌入式开发培训班的教材或教辅材料。

图书在版编目（CIP）数据

ARM 嵌入式 Linux 设备驱动实例开发 / 李亚锋编著 .—北京：中国电力出版社，2008
(ARM 嵌入式 Linux 系统开发丛书)
ISBN 978-7-5083-7435-2

I. A… Ⅱ. 李… Ⅲ. ①微处理器，ARM – 系统设计② Linux 操作系统 – 系统设计 Ⅳ. TP332 TP316.89

中国版本图书馆 CIP 数据核字（2008）第 077264 号

责任编辑：王杏芸
责任校对：崔燕菊
责任印制：郭华清

书　　名：ARM 嵌入式 Linux 设备驱动实例开发
编　　著：李亚锋
出版发行：中国电力出版社
　　地址：北京市三里河路 6 号　邮政编码：100044
　　电话：（010）68362602　　传真：（010）68316497
印　　刷：北京同江印刷厂
开本尺寸：185mm × 260mm　　印　张：16　　字　数：337 千字
书　　号：ISBN 978-7-5083-7435-2
版　　次：2008 年 7 月北京第 1 版
印　　次：2008 年 7 月第 1 次印刷
印　　数：0001—4000 册
定　　价：30.00 元（含 1CD）

PREFACE 前言

开放的Linux受到广泛的欢迎，得到越来越多公司的支持，但是阻碍Linux在各个领域广泛应用的主要因素就是内核和驱动开发高端人才极度缺乏。Linux源代码中85%是设备驱动，在嵌入式系统中驱动程序更为重要，几乎每一个嵌入式系统都是从驱动程序员手中调试出来的。面对巨大的市场需求，将有越来越多的人员加入到Linux设备驱动开发行业中来。但许多初学者根本不知道如何入手学习Linux设备驱动开发，本书以实用简洁为宗旨给读者提供了一个纲领性的参考，希望读者能快速地掌握嵌入式Linux设备驱动开发的核心。

编写目的

嵌入式Linux属于一个交叉学科，并且也是一个高起点的学科，它涵盖了微电子技术、电子信息技术、计算机软件和硬件等多项技术领域。另外，学习嵌入式Linux最好具备相应的嵌入式开发板和软件，还需要有经验的人进行指导开发。目前国内大部分高校都很难达到这种要求，这也是造成目前国内嵌入式Linux开发人才极其缺乏局面的原因之一。

很多希望学习嵌入式Linux设备驱动的人已经具备了一定的硬件知识，并且对操作系统原理、数据结构等都很了解，但在Linux系统下进行设备驱动开发仍然没有合适的入口点。编写本书的主要目的就是对那些Linux设备驱动开发的初学者有个很好的指导作用，让他们少走弯路。

此外，笔者希望通过编写本书来总结这几年在工作中的项目经验，与更多的读者分享自己的技术，也是对自己所做项目的巩固；通过编写本书，笔者更加清楚了实践与理论之间的联系，从而将自己的亲身经验和教训寄托在书中的每个章节。

主要内容

本书共由10章组成，提供8个典型实例，以下是每章的内容概要：

- 第1章：介绍嵌入式Linux设备驱动开发基础知识，包括Linux设备驱动概念以及分类、Linux中断处理、内存与I/O端口、并发控制、阻塞与非阻塞概念。此外，以最简单的内核模块Hello world为例，讲述Linux设备驱动的编译与加载。接着介绍了ARM处理器及其选型，还专门介绍了常用的ARM处理器S3C2410。最后介绍了北京蓝海微芯科技发展有限公司提供的ARM9开发板，本书所有实验代码都是基于该开发板进行的。
- 第2章：介绍本书的第一个典型实例：按键设备模块的驱动开发。首先介绍按键

设备的硬件接口和相关寄存器，接着介绍字符设备相关的核心数据结构，最后重点分析按键设备驱动程序的代码实现，并且给出了该设备驱动的测试程序。

- 第 3 章：以触摸屏设备驱动为实例，首先对触摸屏设备进行了讲述，包括四线式触摸屏工作原理以及 S3C2410 触摸屏接口的工作原理；然后讲述触摸屏设备的接口电路以及相关的寄存器定义；接着重点分析了触摸屏设备驱动程序的代码实现；最后讲述了如何编译和测试触摸屏设备驱动程序。
- 第 4 章：以MMC/SD卡设备驱动为实例，首先对MMC/SD卡分别进行了介绍，然后讲述S3C2410 芯片的SDI接口以及相关的寄存器定义，接着重点分析了MMC/SD卡设备驱动程序的代码实现，最后讲述了如何测试和编译MMC/SD卡设备驱动程序。
- 第 5 章：以 CS8900A 网卡设备驱动为实例，首先对网络设备驱动进行了介绍，然后讲述 CS8900A 网卡设备，接着重点分析 CS8900A 网卡设备驱动程序的代码实现，最后讲述了如何测试和编译 CS8900A 网卡设备驱动程序。
- 第 6 章：介绍基于 Framebuffer 的 LCD 设备驱动。首先对 Framebuffer 进行了介绍，然后讲述 Framebuffer 内部的 API，接着重点分析基于 Framebuffer 机制的 LCD 设备驱动程序的代码实现，最后讲述了如何编译和测试 LCD 设备驱动程序。
- 第 7 章：介绍 USB 驱动程序的实现。首先对 USB 进行了概要介绍；然后介绍 USB 驱动，其中包括 USB 主机驱动、Hub 驱动和设备驱动；最后重点分析 USB 主控制器驱动程序和 USB 设备驱动程序。
- 第 8 章：介绍 NAND Flash 驱动程序的实现。首先对 NAND Flash 进行了概要介绍，然后讲述 MTD 设备驱动，最后重点分析 S3C2410 NAND Flash 设备驱动程序。此外，还讲述了如何使用 fdisk 工具在 Linux 系统下为 CF 卡进行分区。
- 第 9 章：介绍 PCI 设备驱动程序的实现。首先对 PCI 总线进行了概要介绍，然后讲述 PCI 设备驱动，最后重点分析了 PCI 设备驱动程序的代码实现。
- 第 10 章：主要介绍 Linux 下常用调试方法和调试工具。首先介绍打印调试技术；然后介绍查询调试技术；最后介绍常见的内核调试工具，分别介绍了 GDB、KGDB、KDB、strace 和 OOPS。

致谢

首先，感谢我的妻子，是她精心照顾家庭才使得我能专心从事撰写工作，这本书的完成离不开她对我的默默支持。其次，感谢我的岳父、岳母，是他们对我们孩子的精心照顾，才使得我有更多的时间投入到写作中。

鉴于作者水平有限，加之时间仓促，本书一定有不少疏漏与不妥之处，希望得到广大读者的指正与建议。有兴趣的读者可以登录笔者的个人 Blog 来做技术上的交流：http://mike2linus.blog.com.cn/。

作　者

2008 年 4 月

CONTENTS 目 录

第 1 章

嵌入式 Linux 设备驱动开发基础

1.1 设备驱动介绍

设备驱动是任何计算机系统（包括嵌入式系统）硬件与软件工作的接口，它是软件系统与硬件系统交互的桥梁。设备驱动往往指设备驱动程序，也就是说它属于软件。设备驱动可以理解为一类程序，而这类程序的功能一般就是驱动硬件正常工作，提供给软件良好的工作接口。设备驱动程序既可以工作在有操作系统的环境下，也可以工作在无操作系统的环境中。通常在实现一些简单的硬件控制时，由于系统功能比较单一，不需要操作系统来管理，所以针对这种情况的设备驱动实现相对来说比较简单，因为它只完成控制特定硬件的功能，而不需要考虑其他的并发任务等复杂情况。但是往往作为一个大型的、多任务的复杂系统，需要有操作系统来对它进行管理。在这种情况下，编写驱动程序就要考虑到许多其他任务的并发、任务的优先级以及出现中断情况的处理等，所以通常在带有操作系统的环境下编写设备驱动相对比较复杂，但是这也是实际中应用最广的类型，所以对想从事设备驱动程序开发的读者来说，这部分内容是必须掌握的。

1.1.1 Linux 设备驱动

本书是以应用最广泛的 Linux 操作系统为研究对象，讲述如何在 Linux 系统下开发设备驱动程序。通常设备驱动程序在 Linux 系统中的位置如图 1.1 所示，最底层为硬件抽象层，其上面有文件系统、IPC（进程间通信）、网络协议栈、设备驱动、内核调度和内存管理，这部分都属于系统内核空间，系统调用层是内核空间与用户空间的接口层，这样应用程序不会直接访问内核程序，从而增强了内核的安全性，同时应用程序也不能直接操作硬件设备，而是通过设备驱动提供的接口来访问硬件，从而保证了硬件设备的安全性。

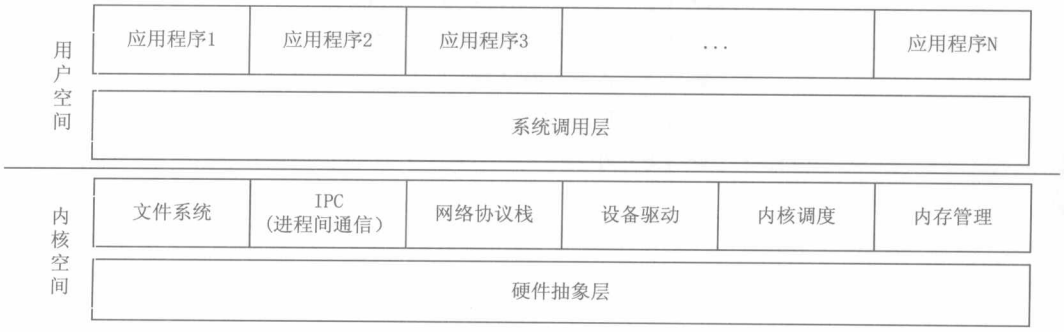


图 1.1 设备驱动在 Linux 系统中的位置

1.1.2 Linux 设备驱动分类

Linux 系统一般将设备驱动程序分为三大类：字符设备、块设备和网络设备，不过它

们之间的区分并不严格，下面将介绍这三类驱动程序各自的特点：

- 字符设备

  字符设备是 Linux 系统中应用最广泛的一类设备驱动，它能够像字节流（文件）一样被顺序访问，通常直接传输来自用户进程的数据而不经过缓存。字符设备驱动程序通常会实现 open、close、ioctl、read 和 write 等系统调用函数。本书第 2 章讲述的按键设备驱动就是一个典型的字符设备驱动实例。通过文件系统节点可以访问字符设备，例如/dev/tty1 和/dev/lp1。字符设备和普通文件系统之间的唯一区别是：普通文件允许在其上来回读写，而大多数字符设备仅仅是数据通道，只能顺序读写。此外，字符设备驱动程序不需要缓冲，它直接从用户进程传输数据，或传输数据到用户进程。

- 块设备

  所谓块设备是指对其信息的存取以“块”为单位，如常见的光盘、硬磁盘、软磁盘、磁带等，块长取 512 字节、1024 字节或 4096 字节。块设备和字符设备一样可以通过文件系统节点来访问。在大多数 UNIX/Linux 系统中，只能将块设备看作多个块进行访问，一个块设备通常是以 1024 字节数据大小的块为传输单位。字符设备和块设备主要区别是：在对字符设备发出读/写请求时，实际的硬件 I/O 操作一般就紧接着发生了；而块设备则不然，它利用一块系统内存作为缓冲区，当用户进程对设备请求能满足用户的要求时，就返回请求的数据，如果不能满足就调用请求函数来进行实际的 I/O 操作，因此，块设备主要是针对磁盘等慢速设备设计的，以免消耗过多的 CPU 时间来等待。

- 网络设备

  网络设备驱动在 Linux 系统中是比较特殊的，它不像字符设备和块设备通常实现 read 和 write 等操作，而通常是通过一种套接字（Socket）[①]接口来实现，任何网络事务处理都是通过接口实现的。通常接口是一个硬件设备，但也可以像 loopback（回路）接口一样是纯软件工具。网络接口是由内核网络子系统驱动的，它负责发送和接收数据包，而且无需了解每次事务是如何映射到实际被发送的数据包。尽管 telnet 和 ftp 连接都是面向流的，它们使用同样的设备进行传输；但设备并没有看到任何流，仅看到数据报。由于不是面向流的设备，所以网络接口不像/dev/tty1 那样简单地映射到文件系统的节点上。UNIX/Linux 调用这些接口的方式是给它们分配一个独立的名字（如 eth0）。这样的名字在文件系统中并没有对应项。内核和网络设备驱动程序之间的通信与字符设备驱动程序和块设备驱动程序与内核间的通信是完全不一样的。

① 套接字是通信的基石，是支持 TCP/IP 协议的网络通信的基本操作单元。套接字可以看作不同主机间的进程进行双向通信的端点，它构成了单个主机内及整个网络间的编程界面。套接字存在于通信域中，通信域是为了处理一般的线程通过套接字通信而引进的一种抽象概念。套接字通常和同一个域中的套接字交换数据（数据交换也可能穿越域的界限，但这时一定要执行某种解释程序）。各种进程使用这个相同的域通过 Internet 协议簇来进行相互间的通信。

其实，除了这三种类型的驱动程序外，还有许多比较特殊的设备驱动程序，如 IIC、USB、RTC、PCI 等，只不过在实际中大部分驱动程序都属于这三种类型，所以通常所说的 Linux 设备驱动程序开发一般指这三类设备驱动的开发。

1.2 设备驱动相关的重要概念

设备驱动开发对于不同的操作系统，实现机制是不一样的，比如μC/OS、VxWorks、Windows CE、Palm OS、Linux 等，其设备驱动实现方式都是不一样的。由于本书是以应用非常广泛的 Linux 操作系统为研究对象，所以本书讲述的设备驱动开发默认都是以 Linux 系统为开发对象。下面将介绍 Linux 系统下的设备驱动开发需要掌握的一些重要概念。

1.2.1 Linux 中断

中断是设备驱动中非常重要的一个概念，所以这里首先讲述中断概念，然后讲述中断请求过程、中断处理程序以及中断相关的内核函数。

1.2.1.1 基本概念

“中断”一词的字面意思是中间发生阻隔、停顿或故障而断开，但在计算机术语中的定义是是指 CPU 在正常运行程序时，由于内部/外部事件或由程序预先安排的事件引起 CPU 暂时停止正在运行的程序，转到为该内部/外部事件或预先安排的事件服务的程序中去，服务完毕再返回继续运行被暂时中断的程序的过程。Linux 的中断通常分为两种：软中断和硬中断。注意，这里的“软”和“硬”的意思是指和软件相关以及和硬件相关，而不是软件实现的中断或硬件实现的中断。软中断就是通过“信号机制”实现的中断。Linux 通过信号来产生对进程的各种中断操作，就目前所知的信号共有 31 个，这里就不一一列举了。硬中断就是通常意义上的“中断处理程序”，它直接处理由硬件发过来的中断信号。当硬中断收到它应当处理的中断信号以后，就回到自己驱动的设备上去看设备的状态寄存器以了解发生了什么事情，并进行相应的操作。软中断主要是进程调度要做的事情。通常，设备驱动的中断是指硬中断，所以这里只讨论硬中断，即和硬件相关的中断。

1.2.1.2 中断请求过程

发生中断，是因为外设需要通知操作系统它那里发生了一些事情，但是中断的功能仅仅是一个设备报警灯，当灯亮的时候中断处理程序只知道有事情发生了，但发生了什么事情还要亲自到设备那里去看才行。也就是说，当中断处理程序得知设备发生了一个中断的时候，它并不知道设备发生了什么事情，只有当它访问设备上的一些状态寄存器以后，才能知道具体发生了什么，要怎样去处理。设备通过中断线向中断控制器发送高电平告诉操

作系统它产生了一个中断，而操作系统会从中断控制器的状态位知道是哪条中断线上产生了中断。注意，并不是每个设备都可以向中断线发中断信号，只有对某一条确定的中断线拥有了控制权，才可以向这条中断线发送信号。由于计算机的外部设备越来越多，通常中断线是非常宝贵的资源。要使用中断线，就得进行中断线的申请，即 IRQ（Interrupt Requirement，中断请求），我们也常把申请一条中断线称为申请一个 IRQ 或者是申请一个中断号。IRQ 是非常宝贵的，所以建议只有当设备需要中断的时候才申请占用一个 IRQ，或者是在申请 IRQ 时采用共享中断的方式，这样可以让更多的设备使用中断。无论对 IRQ 的使用方式是独占还是共享，申请 IRQ 的过程都是一样的，分为 3 步：第 1 步，将所有的中断线探测一遍，看看哪些中断还没有被占用，并从那些还没有被占用的中断中选一个作为该设备的 IRQ。第 2 步，通过中断申请函数申请选定的 IRQ，这时要指定申请的方式是独占还是共享。第 3 步，根据中断申请函数的返回值决定怎样做：如果成功了就可以正常工作，如果没成功，或者重新申请，或者放弃申请并返回错误。

1.2.1.3 中断处理程序

Linux 中的中断处理程序很有特色，一个中断处理程序通常分为两个部分：上半部（top half）和下半部（bottom half）。之所以会有上半部和下半部之分，完全是考虑中断处理的效率。上半部的功能是“登记中断”，当一个中断发生时，它就把设备驱动程序中中断例程的下半部挂到该设备的下半部执行队列中，然后等待新的中断到来。这样一来，上半部执行的速度就会很快，它就可以接收更多由它负责的设备产生的中断了。上半部之所以要快，是因为它是完全屏蔽中断的，如果它不执行完，其他中断就不能被及时处理，只能等到该中断处理程序执行完毕以后。所以要尽可能多地对设备产生的中断进行服务和处理，中断处理程序就一定要快。但是，有些中断事件的处理是比较复杂的，所以中断处理程序必须多花一点时间才能把事情做完。可怎样化解在短时间内完成复杂处理的矛盾呢？这时候 Linux 引入了下半部的概念。下半部和上半部最大的不同是下半部是可中断的，而上半部是不可中断的。下半部几乎做了中断处理程序所有的事情，因为上半部只是将下半部排到了它们所负责设备的中断处理队列中去。下半部一般所负责的工作是查看设备以获得产生中断的事件信息，并根据这些信息（一般通过读设备上的寄存器得来）进行相应的处理。由于下半部是可中断的，所以在它运行期间，如果其他设备产生了中断，下半部可以暂时中断，等到那个设备的上半部运行完了，再回头来运行它。但是有一点一定要注意，如果一个设备中断处理程序正在运行，无论是运行上半部还是下半部，只要中断处理程序还没有处理完毕，在这期间设备产生的新中断都将被忽略。因为中断处理程序是不可重入的，同一个中断处理程序是不能并行的。

1.2.1.4 中断相关函数

与 Linux 设备驱动程序中断处理相关的函数有申请和释放 IRQ（中断请求）函数，即 request_irq 和 free_irq，这两个函数在实际中非常重要，在头文件<include/linux/interrupt.h>

文件中声明。其函数原型如下：

- int request_irq(unsigned int irq, irqreturn_t (*handler)(int irq, void *dev_id, struct pt_regs *regs),unsigned long flags, const char *dev_name, void *dev_id);
  该函数的作用是注册一个 IRQ。其中参数 irq 是要申请的硬件中断号。参数 handler 是向系统登记的中断处理函数，是一个回调函数，中断发生时系统调用这个函数。参数 dev_id 是设备的 ID。参数 flags 是中断处理的属性，若设置为 SA_INTERRUPT，表明中断处理程序是 FRQ（快速中断请求），FRQ 程序被调用时屏蔽所有中断，而 IRQ 程序被调用时不屏蔽 FRQ。若设为 SA_SHIRQ，则多个设备共享中断，这时会用到 dev_id 参数。参数 dev_name 是定义传递给 request_irq 的字符串，用来在 /proc/interrupts 中显示中断的拥有者。
- void free_irq(unsigned int irq, void *dev_id);
  该函数的作用是释放一个 IRQ，一般是在退出设备或关闭设备时调用。
- void enable_irq(unsigned int irq);
  该函数的作用是打开一个 IRQ，允许该 IRQ 产生中断。
- void disable_irq(unsigned int irq);
  该函数的作用是关闭一个 IRQ，禁止该 IRQ 产生中断。

1.2.2 内存与 I/O 端口

内存与 I/O 端口也是 Linux 设备驱动开发经常用到的两个概念，编写驱动程序大多数情况下都是对内存和 I/O 端口的操作。下面将分别介绍内存和 I/O 端口在 Linux 设备驱动程序中的使用。

1.2.2.1 内存

对于运行标准的 Linux 内核，硬件平台需要提供对 MMU（内存管理单元）的支持，并且 Linux 内核提供了复杂的存储管理系统，使得进程能够访问的内存达到 4GB。这 4GB 的空间被人们分为两个部分，一部分是用户空间，另一部分是内核空间。用户空间的地址分布是从 0～3GB，3～4GB 空间定义为内核空间。编写 Linux 驱动程序必须知道如何在内核空间中申请内存，内核空间中最常用的内存分配和释放函数是 kmalloc 和 kfree，这两个函数非常类似标准 C 库中的 malloc 和 free。这两个函数原型如下：

```
void *kmalloc(size_t size, int flags);
void kfree(void *obj);
```

这两个函数被声名在内核源码<include/linux/slab.h>文件中。设备驱动程序作为内核的一部分，不能直接使用虚拟内存，必须利用内核提供的 kmalloc 与 kfree 来申请和释放内核存储空间。kmalloc 带两个参数，第一个参数（size）是要申请的内存数量，第二个参数（flags）用来控制 kmalloc 的优先权。其中 flags 参数的值经常有以下几种：