# 《Linux设备驱动程序》

## 图书基本信息

书名：《Linux设备驱动程序》

13位ISBN编号：9787508338637

10位ISBN编号：7508338634

出版时间：2006-1-1

出版社：中国电力出版社

作者：科波特

页数：569

译者：魏永明,耿岳,钟书毅

# 《Linux设备驱动程序》

## 内容概要

本书是经典著作《Linux设备驱动程序》的第三版。如果您希望在Linux操作系统上支持计算机外部设备，或者在Linux上运行新的硬件，或者只是希望一般性地了解Linux内核的编程，就一定要阅读本书。本书描述了如何针对各种设备编写驱动程序，而在过去，这些内容仅仅以口头形式交流，或者零星出现在神秘的代码注释中。
本书的作者均是Linux社区的领导者。Jonathan Corbet虽不是专职的内核代码贡献者，但他是备受关注的LWN.net新闻及信息网站的执行编辑。Alessandro Rubini是一名Linux代码贡献者，也是活跃的意大利Linux社区的灵魂人物。Greg Kroah-Hartman是目前内核中USB、PCI和驱动程序核心子系统（本书均有讲述）的维护者。
本书的这个版本已针对Linux内核的2.6.10版本彻底更新过了。内核的这个版本针对常见任务完成了合理化设计及相应的简化，如即插即用、利用sysfs文件系统和用户空间交互，以及标准总线上的多设备管理等等。
要阅读并理解本书，您不必首先成为内核黑客；只要您理解C语言并具有Unix系统调用的一些背景知识即可。您将学到如何为字符设备、块设备和网络接口编写驱动程序。为此，本书提供了完整的示例程序，您不需要特殊的硬件即可编译和运行这些示例程序。本书还在单独的章节中讲述了PCI、USB和tty（终端）子系统。对期望了解操作系统内部工作原理的读者来讲，本书也深入阐述了地址空间、异步事件以及I/O等方面的内容。

# 《Linux设备驱动程序》

## 作者简介

Jonahan Corbet早在1981年就开始接触BSD Unix的源代码。那时，科罗拉多大学的一名讲师让他“修正”其中的分页算法。从那时起直到现在。他深入研究了他所遇到的每一个系统，其中包括VAX、Sun、Arden以及x86系统的驱动程序。他在1993年第一次接触Linux系统，从此以后一直从事Linux的开发。Corbet先生是《Linux Weekly News》的创始人和执行主编。他和妻子及两个孩子生活在科罗多州的玻尔得市。

# 《Linux设备驱动程序》

## 书籍目录

# 《Linux设备驱动程序》

## 编辑推荐

《LINUX设备驱动程序(第3版)》详细介绍了Linux。如果您希望在Linux操作系统上支持计算机外部设备，或者在Linux上运行新的硬件，或者只是希望一般性地了解Linux内核的编程，就一定要阅读本书。本书描述了如何针对各种设备编写驱动程序，而在过去，这些内容仅仅以口头形式交流，或者零星出现在神秘的代码注释中。

# 《Linux设备驱动程序》

## 精彩短评

1、既有对源代码的解释，又有一些原理性的说教，另外一本《精通Linux设备驱动程序》有很多例子，但条理方面还是本书更好。
2、正好赶上o'relly搞活动特价，一口气买了三本经典书TCP/IP2;深入理解linux网络内幕；还有这本。买前两本书是想好好学习一下网络协议栈在linux中是如何实现的；而这本书则是为学习协议栈下面的物理层（device）驱动的实现。看了前几章，感觉翻译的还不错，但是这本书不适合入门。
3、经典
4、遥想当年，觉得写个driver很了不起
5、深入浅出
6、适合中低水平的人。Linux 设备驱动模型真心复杂！
7、非常不错，讲的linux设备驱动，i***eit
8、还在看，暂时不做评价！
9、一本非常不错的驱动入门书，看完了，写各种驱动基本没问题了。
10、经典中的经典，偶尔还会拿起来翻一番，现在回看有点过时，如若只是写驱动，自己总结下不需要这么厚的参考书。。。经典膜拜。
11、对于linux设备驱动非常专业的一本书
12、不适合设备驱动初学者，解释了设备驱动的关键点，但不够深入浅出。
13、Linux设备驱动开发的经典之作。
14、这是每一个想深入学习嵌入式驱动的爱好者必读的一本书，书中深入浅出&严谨的教你如何为字符设备、块设备、网络接口编写驱动程序。即使你不是很懂内核，只要你理解C语言而且具有unix方面的基础知识都可以从本书中受益匪浅。外国人毕竟在这方面技术比我们发展的早，所以技术的理解的成熟度也比我们透彻，希望从牛人的理解中获得更加全面深入的对嵌入式驱动的理解，即站在巨人的肩膀上走自己的路。。。
15、我算是没读好，打开方式不对吧
16、经典linux驱动开发书籍。
17、不用多说，学习linux设备驱动必读，理论讲得清晰，只是少了些实际的操作。
18、这本书不是针对我的，我读起来很累而且不知道在讲什么，从来不知道翻译重要的我，这次深刻感觉到，这本书翻译很有问题，感觉自己太渣了，实践实践再读
19、学习LINUX驱动开发，一般而言，这本书是必不可少的，现在这本书已经是2.6内核版本的，可以说书还是很不错的，不过需要一定的基础，尤其是操作系统和LINUX下C语言设计的基本知识
20、这个是在大三看的,记得当看得很細,还做了很多的纸质笔记.
21、这本书非常好，对于初学linux设备驱动开发的人
22、总觉还可以，虽然很多不够详细，但对于刚入们的来说够了，配合linux内核一起看容易看得懂。书中提供的代码，需要修改，因为内核不一样。我是在2.6.37内核上做实验的，而本书的基于的是2.6.10，还是有点区别。所以要配合自己的内核来看此书。
23、虽然从ldd2到ldd3买了若干本，但一直不好意思说自己看了。这次借培训机会对许多章节有了更深入的理解和研究，就算读完了罢。
24、粗略第一遍，纯看比较难懂，还得多结合实践
25、与深入理解LINUX内核一起，是绝配
26、如果要学习linux设备驱动的话，这是一本值得一看的书。一直在等这本第三版的中译本，终于出来了，呵呵
27、原版是非常好的一本书，可惜翻译的太烂了！！googlle翻译的吧！！
28、应该这一本书是学习Linux驱动的经典教材！推荐大家想了解Linux驱动的看看这本书！
29、这本书详细的介绍了linux内核中驱动程序的各种接口，结合了一些范例，非常易读易学，还可以作为手册。很好的一本书
30、目前，随着android系统的普及，linux系统的开源，造就了驱动的无限发展。

本书，非常的专业，讲解非常透彻，购买的同学很多，源于本书的结构安排和内容的详尽。

推荐给大家了。
31、Linux设备驱动程序对我来说正是需要的书，不错，挺满意的。
32、重新看一遍
33、使用内核API设计驱动的方方面面...
34、买的第一本Linux书
35、学习linux内核非常不错的书
36、Linux driver 编程入门
37、由于包装的问题，书上边角的折痕并没有影响到此次购书的愉悦心情。《Linux设备驱动开发》很早之前就买了一本，中途linux的工作撂了好几年，现在用又找不到书，只好再买一本，竟然是11年1月第11次印刷的，喜出望外！
38、书是好书，系统全面地讲解linux设备驱动开发。但是不适合初学者，建议有一定开发基础后再看此书效果应该更好。
39、这本书对于学习Linux设备驱动来说首选， 发货速度也快~
40、经典书籍吧，同学推荐看的，其实现在接触的主要还是Android的内核（不过也非常少有），在有需要它的时候才回来翻翻它看看~~。
41、网上看到学习linux内核的经典书就属LDD、ULK了，假期的时间好好看看
42、经典的书，必《深入理解linux内核》薄很多，但书的质量不错。
43、Linux设备驱动程序 这本书不错，不过对于初学者而言的话应该是有点难度，但是对于刚入行的工作者而言会有不少帮助。
44、开始linux下驱动程序开发的必备工具书
45、编写linux设备驱动入门书籍，很不错。
46、对于LINUX设备驱动的介绍比较详细，读了之后受益匪浅。是一本好书，非常值得一读。
47、用于linux内核入门，不错
48、linux设备驱动的经典书籍，老师说他都看几遍了，强烈推荐的书，虽然本来还没打算现在看，但是买其他的书就顺便买了，还有优惠
49、已经邮购了 粗略翻了下电子版 似乎会很有趣
50、kernel 3.0+需要很多改动
51、想了解驱动程序开发的不二选择。
52、linux驱动开发怎能少的了它
53、是linux驱动开发的必备书，也是学习linux内核的经典必读书之一
54、这本书不错，详尽讲述了linux驱动程序的知识。
55、只看完了字符驱动，此书是根据linux内核2.6.10写的，里边很多的内容与目前的内核实现不一样了，这类书有效期太短，以后要慎重购买，最好能看电子版的（这本书英文版的电子版是免费的）
56、没有实践，读了也没用
57、linux开发者必读，有些地方翻译的不好
58、就着源码啃这本LDD，再时不时嘬一口LKD，逐渐消化的感觉简直爽 <- 这是说的内核部分，后面驱动部分燥燥燥燥燥得直想撸
59、原文精彩，看了前几章，翻译的也不错。很优秀的图书。
60、书收到了，翻看了一下，没有问题，希望能借助它把Linux设备驱动学得好点。
61、经典书籍，非常有用处的Linux设备驱动程序
62、这书糟糕的翻译让我记忆犹新...也许，是我不适合这种翻译风格吧
63、详细介绍了2.6内核下驱动程序开发，是一本好书！
64、这本书对linux驱动程序进行了经典的叙述
65、Linux设备驱动，经典！
66、如果想大概的了解linux 的设备驱动，这本书是可以满足的
如果还想具体的知道如何写一个设备驱动，还需要自己去看代码
67、Linux内核必读之书，很多都是工作过以后才读的书，足以可见此书的使用价值！
68、适合Linux驱动开发者阅读。不过像所有驱动程序开发方面的书一样，该书也是有些沉闷，需要大

家耐下心来慢慢读。
69、原书肯定是好书，但是中文翻译的太差！！！！！
70、还没看多少，如果要研究Linux内核那么推荐一下这本书
71、Linux设备驱动程序（第三版）是目前翻译的最准确和忠实于LinuxDeviceDriverVersion3原文的了，搞Linux的都应该买一本来作为手边书。推荐购买。
72、老外写的书，就是经典，条理和逻辑都非常清晰。
73、就时常翻着参考吧。
74、现在正在学驱动程序，有些地方还看不太懂，不过会坚持看下去，只是现在的内核发生变化了，不知道其中的知识有没有过时的了呢
75、那是相当的不错，对入门linux内核开发，作用很重要
76、对poll的理解是从这本书开始的
77、写的比较抽象，第一遍可能不太好懂，可以配合Linux设备驱动开发详解来看
78、写写驱动还是很好玩的。可惜没有实战机会~
79、读过一遍，云里雾里。直到听完一期培训视频，然后看懂了。最近一直在看。
80、linux内核驱动开发权威书籍
81、书确实写的还可以，但很多东西都不是看一遍就能看明白的，建议真的要研究linux的驱动的人先把Linux的操作系统内的一些基本功练好，再看这本书
82、需要仔细研究Linux设备驱动了。
83、质量很好，对于搞Linux设备驱动程序这一块的同仁来说，绝对是当之无愧的经典书籍
84、很好的驱动编程入门书籍
85、不管你的兴趣是在内核层还是在应用层，只要你在LINUX领域，这本书是必看的。这本书就是窥探LINUX内核真相的一道门缝，请从***.kernel.org网站上下载最新的LINUX内核源代码，再结合这本书来看的话会使印象更加深刻。
还有一本书是《深入理解LINUX内核》，这两本书是学习LINUX必备的，需要反复琢磨。

有志同道合者加我QQ425442742一起交流。
86、又浅入深，经典
87、当初看这本书是干什么的……现在已经看不懂了。
88、直系师兄的书，能不好好看嘛，况且这本书确实经典-,-
89、耐心点看建议先看《LINUX内核设计与实现》再看这本书，有大成
90、小部分章节未看完（USB\块设备）
91、值得一读！此本书基本上涵盖了Linux驱动有关的技术。可以作为一本参考书。
92、还有14，17，18三章没看
93、书是正版，刚刚看完了前沿和简介部分，作者和译者语言叙述非常让人易于接受，通俗易懂，是linux内核和驱动开发的经典必读著作，很高兴我又找到了技术的方向，好好研究。
94、一本经典的linux驱动及内核书籍. 经典不一定适合您. 但留一本参考还是行的.
95、大四毕设时读的
96、我是学嵌入式的，路线是从硬件到软件，而非从软件到硬件。总之以前没玩过操作系统，看这书的确是相当的费劲，看到一半了，但对往嵌入式linux里写驱动还是觉得有点力不从心啊。
97、此书是linux驱动的经典之作，朴实无华却又深邃权威，必备之书。这本书讲得又全又深，但例子有点少，建议找一本例程丰富、浅显一些的书一起看...
98、也没有完全理解，但是对于linux内核实现理解还是很有帮助的
99、满足
100、Wanting for 4th Edition...
101、书的质量不错，Linux设备驱动开发必备的参考书

# 《Linux设备驱动程序》

## 精彩书评

1、更正一下，之前看到的是网上电子版的翻译很差。后来看了纸质的正版图书，翻译还不错，是本好书！----------书不错，但翻译太差了。知道将信号量翻译成什么吗，翻译成"旗标"(见5.3节)，我算服了，完全是外行翻译得嘛。“ 不可中断操作是一个创建不可杀死的进程( 在 ps 中见到的可怕的 "D 状态" )和惹恼你的用户的好方法, 使用down_interruptible 需要一些格外的小心, 但是, 如果操作是可中断的, 函数返回一个非零值, 并且调用者不持有旗标. ” 这段是文中的一段翻译，"不可中断操作是一个创建不可杀死的进程( 在 ps 中见到的可怕的 "D 状态" )和惹恼你的用户的好方法" 还好理解，后面翻译得是什么东西哦，"需要一些格外的小心"翻译成需要格外小心不好吗？ "但是"表示转折，看不出来转了什么折；这样的地方比比皆是，翻译太垃圾了。
2、大家好，《linux设备驱动程序》的代码基于2.6.10，在当前流行的内核上已不能编译。我已移植到3.0以上的所有longterm stable分支并测试正确运行。见https://github.com/duxing2007/ldd3-examples-3.x。
3、ldd3被堪称设备驱动学习金典的确不假，但并不是每个新手都能吭下来的。新人学习还是推荐实战性强点的书，宋宝华老师的驱动开发或者精通Linux设备驱动程序开发都可以，这三本书我都稍微翻过...对于有点驱动开发积累的人，看看这本书我觉得还是蛮好的，很多点都已经点到，知道了大体概念，后续可以根据概念自己去google上扩展就好了...
4、该书对字符设备驱动和linux内核同步的介绍还是很详细的。虽然其中关于usb和pci的介绍有点笼统和不够清楚，不过也可以使读者有一定的了解。另外本书还介绍了网络设备驱动，块设备和TTY驱动，覆盖面还是挺广的。貌似驱动程序的范式就是注册设备，然后实现xxxx_operations的成员。因为是2.6的内核，所以不必太执着细节，花点时间玩玩例程，应该更容易理解书中所讲的内容。
5、不晓得是因为我看的翻译版本原因还是什么问题，反正我觉得这本书特别烂，烂到让我根本看不下去。整本书给人的感觉不知所云，知识点很不集中，逻辑很不清晰，理论讲解十分不透彻，主要就是讲了要用哪些内核函数。还不如改名叫做《内核函数运用》。也没有什么太有内涵的代码。因为知道这本书被评为经典，所以好几次我看不下去的时候忍住了接着看，但是实在是受不了这本书了，推荐看《精通linux设备驱动程序开发》这本书，讲解的逻辑要清晰多了。
6、这本书注重讲实践，如果你看过操作系统或者深入理解Linux之类的书再边看此书边实践的话，你会觉得很舒服。书的作者是Linux驱动的主要维护者，他以一个实践者的角色带领我们进入Linux内核编程。
7、怀着无比朝圣即对经典的憧憬的精神开始看这本书学习Linux驱动，当看到这本书只要求读者熟悉C语言编程及拥有Unix的使用经验就无比的兴奋以为自己将能够顺利地完成学习目标。但是，想的太天真了。书的内容没有循序渐进，一上来就给你讲一些比较专业的东西但是又没有介绍细节背景没有给你将个清楚，例如第二章构造内核源代码树什么的、第三章的各种数据结构但是又没有给你一个好的过度及之间的联系。还有，也许是翻译的原因，语句读起来费劲吃力让人昏昏欲睡。本来就是不太接地气的东西他却讲解得如此生硬。总的来说，千万不要以为你看过C语言的语法及用过几天Linux就以为能驾驭这边书了，那就太天真了。呵呵！
8、我有英文版的电子书，也另外买了一本中文版，因为觉得一天到晚对着电脑太辛苦。不过看了一半下来觉得不如看英文版。虽然对自己的英文水平并不是太有自信，不过中文版翻译的实在是不敢恭维，看着中文版翻译的字字句句自己要推敲半天才搞清除意思，这样折腾下来不见得比读英文版进度快。也许是我汉语水平实在是不行。拜一拜该书的几位译者……
9、手头有两本，一本原版的，一本中译的。阅读原版的冲动屡屡受到my pool English的打击，所以特别感谢热心人整出来的中译本，都是熟悉的方块字，翻起顺手多了，呵呵。鄙人期间花了不少精力，在研究其中式表达想要说明的东东，但是这样也好，多花点时间没坏处，有助于深刻理解。这等书没有最好的，只有更好的，选择实用、适用的，一家之言！^_^
10、这本书的内容偏简洁，介绍了内核API和驱动程序设计中一些基本的概念，对于真实驱动程序设计中涉及到的一些技巧还是比较欠缺。对于PCI和USB的驱动，觉得本书论述得不够具体，看起来有难度；对于kobject的论述，也觉得不是很直观，不过前面关于字符设备的论述还不错，对内存分配、中断还是有一些细致的介绍。关于本书的翻译和排版，真是不敢恭维。翻译者对概念混淆不清，把entry

译成文件；内容粗糙，把两段不同的文字写成一模一样。排版也不好，章节号也不标明，仅用字体区分，甚至有几处地方还出现了英文原文。如果不参照英文电子版的书看，有的翻译内容真的很难理解。

11、第一次阅读。2012年3月8日读完，历时一个月。书本的开篇就说，Linux内核很复杂很庞大，入门较难，但是驱动程序是进入到Linux内核世界的大门。不过我看完之后没有这个感觉，只是觉得每一章都会涉及到很多内核的内容，并不会知道内核总体来说到底是怎么样的。开始接触驱动程序也是近两三个月的事情，到现在，我对开发驱动程序到底是怎么样的一回事还不是很清楚。什么情况下需要开发一个驱动程序？不同的Linux版本（包括嵌入式的）下的驱动程序会有怎么样的不同？还有很多东西不明白。而且最重要的是，还没有写过一个Linux设备驱动。连demo也没有写过。书本涵盖了设备驱动的各方面内容。不过觉得每一章里面的内容组织都比较散，读着的时候就觉得不是很连贯，具有跳跃性的。再加上它的内容都讲得比较深，就导致了我得出这样的一个结论：这本书不太适合初学者，看完会觉得收获很少而且很辛苦；但它还是一本神书，适合有一定经验的人去阅读，去全面提升自己的“ 功力 ”。

# 《Linux设备驱动程序》

## 章节试读

1、《Linux设备驱动程序》的笔记-第8章：分配内存

第8章：分配内存

第一节-kmalloc 函数的内幕。Kmalloc 的原型是： void *kmalloc(size_t size, int flags) 。其中 size 表示要申请的内存的大小，但是内核只能分配一些预定义的、固定大小的字节数组。Flags 是内存分配标志，不同的标志指定了几种不同的分配方式。

第二节-后备高速缓存。为了提高系统性能，内核使用了一个“ slab 分配器 ”来管理高速缓存，也就是后备高速缓存，也称为内存池。Slab 分配器实现的高速缓存具有 kmem_cache_t 类型，可通过调用 kmem_cache_create 创建。一旦某个对象的高速缓存被创建，就可以调用 kmem_cache_alloc 从中分配内存对象，而释放则调用 kmem_cache_free 。

第三节-get_free_page 和相关函数。如果模块需要分配大块的内存，使用面向页的分配技术会更好一些。分配页面可以使用下面的函数：get_zeroed_page()、__get_free_page()、__get_free_pages() 。

第四节-vmalloc 及其辅助函数。Vmalloc 函数的执行比较慢，不鼓励使用，但是它能够分配虚拟地址空间的连续区域，即使这些地址对应的物理地址不连续。当驱动程序要使用真正的物理地址时，就不能使用vmalloc 。

第五节-per-CPU 变量。当建立一个 per-CPU变量时，系统中的每个处理器都会拥有该变量的特有副本。它的优点在于访问快速，因为不需要锁定。常用于网络子系统中。

2、《Linux设备驱动程序》的笔记-第468页

struct gendisk is a dynamically allocated structure that requires special kernel
manipulation to be initialized; drivers cannot allocate the structure on their own.
Instead, you must call:struct gendisk *alloc_disk(int minors);The minors argument should be the number of minor numbers this disk uses; note
that you cannot change the minors field later and expect things to work properly.
When a disk is no longer needed, it should be freed with:void del_gendisk(struct gendisk *gd);A gendisk is a reference-counted structure (it contains a kobject). There are get_disk
and put_disk functions available to manipulate the reference count, but drivers
should never need to do that. Normally, the call to del_gendisk removes the final reference
to a gendisk, but there are no guarantees of that. Thus, it is possible that the
structure could continue to exist (and your methods could be called) after a call to
del_gendisk. If you delete the structure when there are no users (that is, after the final
release or in your module cleanup function), however, you can be sure that you will
not hear from it again.
Allocating a gendisk structure does not make the disk available to the system. To do
that, you must initialize the structure and call add_disk:void add_disk(struct gendisk *gd);

在ldd3中，作者认为应该“ drivers should never need to ”调用get_disk和put_disk，单纯del_gendisk就可以：这是不正确的。具体请参照：
https://www.sao.ru/hq/sts/linux/doc/porting_to_26/25711.html

3、《Linux设备驱动程序》的笔记-第7章：时间、延迟及延缓操作

第7章：时间、延迟及延缓操作

第一节-度量时间差。时钟中断由系统定时硬件以周期性的间隔产生，这个间隔由内核根据HZ的值设定，HZ是一个与体系结构有关的常数。内核使用变量 jiffies 来记录自系统启动以来产生的间隔数。所以通过两次获取 jiffies 就可以算出已经流逝的时间的长度。

第二节-获取当前时间。内核一般通过jiffies 值来获取当前时间。但是要注意这里所指的“ 当前时间 ”和“ 墙钟时间 ”是有区别的。

第三节-延迟执行。长于时钟滴答的延迟并且不会因为它的低分辨率而导致问题，则可以使用系统时钟，而非常短的延迟通常必须用软件循环的方式实现。Ndelay、udelay 和 mdelay 这几个内核函数可很好地完成短延迟任务，它们分别延迟指定数量的纳秒、微秒和毫秒时间。如上所说，这三个函数都是忙等待函数。

第四节-内核定时器。一个内核定时器是一个数据结构，它告诉内核在用户定义的时间点使用用户定义的参数来执行一个用户定义的函数。这些函数访问到的数据都必须针对并发进行保护。

第五节-tasklet 。Tasklet 是一个小任务，它始终在中断期间运行，也就是在进程上下文之外运行，且始终被调度在同一个CPU上运行。这样一个小任务被定义在将来的某个时间点上运行，但是当CPU忙于运行某个进程时，tasklet 会被稍微推迟运行。

第六节-工作队列。工作队列类似于tasklet，但是它运行在一个特殊的内核进程上下文中，而且可以休眠。对于设备驱动程序，很多情况下都不需要有自己的工作队列，这时可以使用内核提供的共享的默认工作队列。

4、《Linux设备驱动程序》的笔记-第5章：并发和竞态

第5章：并发和竞态

设备驱动程序开发者必须在开始设计时就考虑到并发因素，并且还必须对内核提供的并发管理设施有坚实的理解。因为并发无处不在，而且很容易引发重大错误。

第一节-并发及其管理。当对资源进行共享访问时特别容易出现问题，所以应当尽量避免资源的共享。在不得不实现资源共享时，当多个执行线程共享硬件或软件资源的任何时候，其中一个线程可能会发现该资源出现不一致状态，所以必须显式地管理对该资源的访问。

第二节-信号量和互斥体。其实互斥体也就是信号量用于互斥时的叫法。要使用信号量，内核代码必须包括 <asm/semaphore.h> 。如果在拥有一个信号量时发生错误，必须在将错误状态返回给调用者之前释放该信号量。

第三节-自旋锁。自旋锁原语所需要包含的文件是 <linux/spinlock.h> 。适用于自旋锁的两条重要规则：任何拥有自旋锁的代码都必须是原子的，也就是不可打断的；自旋锁必须在可能的最短时间内被占有。

第四节-锁陷阱。这节介绍了一些由锁引发的问题和一些原则。①如果某个获得锁的函数要调用其他同样试图获取这个锁的函数，我们的代码就会死锁。②在必须获取多个锁时，不同的进程都应该始终以相同的顺序获得。③如果我们必须获得一个局部锁，以及一个位于内核更中心处的锁，则应该首先获取自己的局部锁。

第五节-除了锁之外的办法。有些时候我们可以重新构造算法，以从根本上避免使用锁。例如使用原子变量 atomic_t 类型，原子位操作，seqlock，RCU 机制。

5、《Linux设备驱动程序》的笔记-第13章：USB 驱动程序

第13章：USB 驱动程序

1、从拓扑上来看，一个USB 子系统并不是以总线的方式来布置的；它是一棵由几个点对点的连接构建而成的树。从技术层面来说，USB 只担当设备和主控制器之间通信的角色。其实USB 驱动程序包括两种：宿主（host）系统上的驱动程序和设备（device）上的驱动程序。

2、内核提供了一个称为USB 核心（USB core）的子系统来处理USB 相关的大部分复杂性，使得驱动程序的开发变得容易。三个最基本的概念是端点、接口和配置。端点用于建立主机和设备之间的单向传

输管道，有控制、中断、批量和等时四种。接口是对端点进行捆绑后的结果，提供了更高一层的逻辑功能。而配置，又是对接口进行捆绑，实现不同的设备状态，方便转换。
3、USB 代码通过一个称为urb （USB 请求块）的东西和所有的USB 设备通信。Urb 和端点的关系：可能会为单个端点分配多个urb，也可能对多个端点重用单个urb 。Urb 由struct urb 结构体描述。

6、《Linux设备驱动程序》的笔记-第一章&第二章

第1章：设备驱动程序简介
1、“ 通常，设备驱动程序就是这个进入Linux内核世界的大门 ”，“ 设备驱动程序在Linux内核中扮演着特殊的角色，它们是一个个独立的黑盒子，使某个特定硬件响应一个定义良好的内部编程接口，这些接口完全隐藏了设备的工作细节。用户的操作通过一组标准化的调用执行，而这些调用独立于特定的驱动程序 ”。
2、Linux系统将设备分成三种基本类型：字符设备、块设备和网络设备。

第2章：构造和运行模块
1、“ 内核黑客通常拥有一个‘ 牺牲用的 ’系统，用于测试新的代码 ”。
2、模块在被使用之前需要注册，而退出时要仔细撤销初始化函数所做的一切。驱动模块只能调用由内核导出的那些函数。
3、公共内核符号表中包含了所有的全局内核项（即函数和变量）的地址。当模块被装入内核后，它所导出的任何符号都会变成内核符号表的一部分。

7、《Linux设备驱动程序》的笔记-第三章 - 字符设备驱动程序

第一节-主设备号和次设备号。对字符设备的访问都是通过文件系统内的设备名称进行的。通常而言，主设备号标识设备对应的驱动程序，而次设备号用于正确确定设备文件所指的设备。对应的数据结构为dev_t 类型。分配设备号使用函数alloc_chrdev_region() ，释放就使用unregister_chrdev_region() 函数。
第二节-一些重要的数据结构。大部分基本的驱动程序的操作都要涉及到三个重要的内核数据结构，分别是 file_operations、file 和 inode。内核用inode 结构来表示文件，而用 file 结构来表示打开的文件。
第三节-字符设备的注册。内核内部使用 struct cdev 结构来表示字符设备，在内核调用设备的操作之前，必须分配并注册一个或者多个这种结构。调用函数 cdev_add() 告诉内核该结构的信息，而调用cdev_del()函数从系统中移除一个字符设备。
第四节-open 和 release。open方法提供给驱动程序以初始化的能力，而 release 方法的作用正好与open相反。
第五节-read 和 write。read 和 write 方法完成的任务是相似的，拷贝数据到应用程序空间，或反过来从应用程序空间拷贝数据。

8、《Linux设备驱动程序》的笔记-第15章：内存映射和 DMA

第15章：内存映射和 DMA
1、第一部分讲述了mmap 系统调用的实现过程。该系统调用直接将设备内存映射到用户进程的地址空间里。对驱动程序来说，这可以提供给用户直接访问设备内存的能力。
2、执行直接I/O 访问也就是不通过内核空间缓存，可以大大提高速度。但这只适合于非字符设备、设置缓冲I/O 开销特别大的情况。实现的关键是使用get_user_pages 函数。

3、直接内存访问（DMA)I/O 操作允许外围设备直接和主存进行数据交换，而不需要系统处理器的参与。内核实现了一个与总线、体系架构无关的DMA层，隐藏了大量的问题，使得编写驱动程序变得容易。

9、《Linux设备驱动程序》的笔记-第18章：TTY 驱动程序

第18章：TTY 驱动程序

1、物理tty设备的例子有串口、USB到串口的转换器，还有需要特殊处理才能正常工作的调制解调器。而tty驱动程序有三类：控制台、串口和pty 。通常来说，tty核心从用户那里得到将被发往tty设备的数据，然后把数据发送给tty线路规程驱动程序，该驱动程序负责把数据传递给tty驱动程序，tty驱动程序对数据进行格式化，然后才能发送给硬件。

2、数据结构。Tty_driver用来向tty核心注册一个tty驱动程序。Tty_operations包含所有的回调函数，它们被tty驱动程序设置，并被tty核心调用。Tty_struct被tty核心用来保存当前特定tty端口的状态。

3、当用户使用open打开由驱动程序分配的设备节点时，tty核心将调用open函数。当用户使用先前由open函数创建的文件句柄作为参数调用close函数时，tty核心调用close函数指针。当数据要被发送给硬件时，用户调用write函数，核心收到该调用后把数据发送给tty驱动程序的write函数。至于read函数，驱动程序没有实现，而是使用设备把数据传到核心管理的缓冲区，再由用户获取的办法。

10、《Linux设备驱动程序》的笔记-第17章：网络驱动程序

第17章：网络驱动程序

1、网络接口是第三类标准Linux设备，它与字符设备和块设备都有较大的不同，不能体现“一切都是文件”的思想。网络接口存在于它们自己的名字空间中，并导出一系列不同的操作。

2、net_device结构处于网络驱动程序层的最核心地位，当其被初始化完成后，就可以传递给register_netdev函数进行注册。驱动程序可在装载阶段或内核引导阶段探测接口，若有，就会在设备链表中插入一个数据结构。

3、网络接口所执行的最重要的任务是数据的传输和接收，调用函数hard_start_transmit来把数据放入外发队列。而接受数据有中断驱动方式和轮询方式。

4、套接字缓冲区sk_buff结构处于内核网络子系统的核心地位。这个结构对于驱动程序开发人员来说不是必须了解的，但是对跟踪问题和优化代码很有帮助。

11、《Linux设备驱动程序》的笔记-第八章 分配内存

一些注意事项

1.kmalloc:

不是基于页的分配，可能会产生内存碎片而浪费。

分配的区间不会清零，需要显示的清零，且在物理内存中是连续的。分配区域最小为32或者64，不大于128KB。

当设置标志为GFP_KERNEL时，表示是内核空间的进程执行的。可引起休眠。GFP_ATOMIC不会引起休眠。

内存区段中的高端内存是针对大容量的内存而存在的机制，其在内核空间没有对应不变的地址。kmalloc不可分配高端内存。

分配的是虚拟地址，要通过MMU转换成物理地址。

2.后备高速缓存
slab分配器。为了不需要反复的初始化而设定的特殊功能的内存池。
基于slab的高速缓存 scullc
声明高速缓存指针kmem_cache_t---->创建一个高速缓存kmem_cache_create---->为高速缓存分配量子kmem_cache_alloc---->释放量子kmem_cache_free---->释放高速缓存kmem_cache_destroy
使用高速缓存运行速度略高，并且对内存的利用率更佳。
内存池是保留空闲区域供紧急使用，一般情况下不用内存池。

3.get_free_page相关函数
面向页的内存分配。
常用__get_free_pages(GFP_ATOMIC,0);
分配时要提供相应的出错函数。
由__get_free_pages函数的最大优点是这些分配的页面完全属于我们自己，而且在理论上可以通过适当的调整页表将他们合并成一个线性区域。
分配的是虚拟地址，要通过MMU转换成物理地址。

4.vmalloc
分配连续的虚拟地址，但是物理地址不一定连续。
效率不高。和kmalloc分配的地址范围不同。
分配的是虚拟地址，不能再微处理器之外使用。vmalloc使用在分配一大块连续的只在软件中存在的用于缓冲的内存区域，比如用来获得装在模块所需要的内存空间。
ioremap和vmalloc都是面向页分配，会修改页表。
vmalloc不可用在原子上下文中，因为内部调用了kmalloc(GFP_KERNEL)可能会引起休眠。

5.获取大的缓冲区
大的连续的缓冲区分配容易失败，导致内存碎片化。
执行大的IO操作的最好方式是离散/聚集操作。

6.引导时获得专用缓冲区
需要大的连续缓冲区也可以在系统引导期间通过请求内存实现。
模块不能再引导时分配，而只能连接到内核的设备驱动才能在引导时分配内存。

12、《Linux设备驱动程序》的笔记-第4章：调试技术

由于是一个不与特定进程相关的功能集合，所以内核代码无法轻易地放在调试器中执行，而且也很难跟踪。所以单从调试手段上来说，内核调试比起应用程序要难得多。
第一节-内核中的调试支持列出了十几个用于内核开发的配置选项并说明其作用。
第二节-通过打印调试。最普通的调试技术就是监视，即在应用程序编程中，在一些适当的地点调用printf 显示监视信息。调试内核代码的时候，可以用 printk 来完成相同的工作。接着讲解了 printk 的8种日志级别，不同的级别会把打印信息输送到不同的地方。
第三节-通过查询调试。这里的“ 查询 ”是指向 /proc 文件系统进行查询。/proc 文件系统是一种特殊的、由软件创建的文件系统，内核使用它向外界导出信息。其下面的没一个文件都绑定与一个内核函数，用户读取其中的文件时，该函数动态地生成文件的“ 内容 ”。所以每一个内核模块想要在调试时能够查询到一些有用的信息，就要实现相应的能够生成信息的函数。
第四节-通过监视调试。这里所说的“ 监视 ”就不是像上面提到的打印那么简单了。而是利用 strace 命令，它可以显示由用户空间程序所发出的所有系统调用，而且还能显示调用参数以及用符号形式表示的返回值。
第五节-调试系统故障。大部分错误都是因为对NULL指针取值或因为使用了其他不正确的指针值。这

些错误通常会导致一个 oops 消息。
第六节-调试器和相关工具。本节介绍的调试器包括 gdb 、kdb 和 kgdb 。

13、《Linux设备驱动程序》的笔记-第9章：与硬件通信

第9章 ：与硬件通信
第1节-I/O 端口和 I/O 内存。每种外设都通过读写寄存器进行控制。需要注意的是，在对这些寄存器进行读写时，不能使用高速缓存，避免读写指令的重新排序。设置内存屏障是一种可行的解决办法。
第2节-使用 I/O 端口。使用之前需要先分配 I/O 端口，内核为我们提供了一个注册用的接口，它允许驱动程序声明自己需要操作的端口，其核心函数是 request_region 。对于 C 语言程序，访问不同大小的端口就要使用不同的函数。
第3节-使用 I/O 内存。I/O 内存仅仅是类似 RAM 的一个区域，在那里处理器可以通过总线访问设备。为安全起见，访问 I/O 内存时不鼓励直接使用指向 I/O 内存的指针，而是应该使用包装函数。

ps:这一节的笔记写得好差。。

14、《Linux设备驱动程序》的笔记-第409页

今天只看了地址类型以及一点点内存映射。后面的还不是很明白。
我觉得需要注意的:
linux是虚拟内存系统，用户程序与硬件使用的物理地址是不同的。虚拟内存可以使进程看到比实际大得多的内存。
几种常见地址:
用户虚拟地址---->用户空间所能看到的常规地址。32位线性地址中，31--22为页目录表，21--12为页表，11--0为页内偏移。
物理地址:处理器----系统内存
总线地址:外围总线----内存。通常与物理地址相同。
注意区分内核逻辑地址与内核虚拟地址。
逻辑地址+偏移量=虚拟地址。逻辑地址与物理地址的映射是线性的和一一对应的。kmalloc返回的是内核逻辑地址。
内核虚拟地址>内核逻辑地址，它与物理地址不必一一对应，但是也是将内核空间的地址映射到物理地址上。vmalloc返回的是虚拟地址，它不存在直接的物理映射。
“ 内核代码和数据结构必须与3GB用户空间1GB内核空间相互对应，占用内核地址空间最大的部分是物理内存的虚拟映射。内核无法直接操作没有映射到内核地址空间的内存。 ”为什么？？？
linux这样是为了简化内核中虚拟地址和物理地址之间相互转化的工作，__va()、__pa()轻松的就变换过来了，并不是说物理内存映射到内核空间就全部被内核占了。对于内核中非动态数据使用的物理内存是不可以变了，但是动态的部分，是可以释放掉的，而释放掉后这一部分对应的物理内存又可以影射到用户空间。
为什么“ 实际使用的内核空间会小于1GB ”
因为要保留一段空间供动态映射。
低端内存依赖硬件与内核的设置，为逻辑地址，通常存放的是内核数据结构。
高端内存为用户空间使用。
什么事虚拟内存区VMA
VMA用于管理进程地址空间中不同区域的内核数据结构。
每一个运行的进程，都会获得一个4G的内存地址空间，即所谓的虚拟内存，这里面的所有地址都是虚拟的，和物理内存啥的并不直接挂钩。而在操作系统那头，这些虚拟地址所映射到的实际地址，可以是物理内存地址，也可以是页面文件的地址。如果是物理内存小于这个虚拟地址的范围的话，映射的

物理内存还可能是重复的物理内存地址片段，使用的时候可以通过清空内存数据，将内存数据写入页面文件这样的方式进行物理内存的重新利用，以提高物理内存的利用效率。
可以通过cat /proc/1/maps查看进程1的内存区域

至于内存映射处理，还不太明白。

15、《Linux设备驱动程序》的笔记-第14章：Linux 设备模型

第14章：Linux 设备模型
1、这一章所讲的设备模型主要是内核为了方便、高效地管理设备而设计出来的，对许多驱动程序开发者来说是可以忽略的。
2、组成设备模型的基本是kobject 结构。Kobject 能够处理对象的引用计数、在sysfs中表述一个对象、把各种需要的数据结构关联起来、处理热插拔事件。一个kset 是嵌入相同类型结构的kobject 集合，可以认为是其顶层容器。Kobject 是隐藏在sysfs 虚拟文件系统后的机制，对于sysfs 中的每个目录，内核中都会存在一个对应的kobject 。
3、设备模型展示了总线和它们所控制的设备之间的连接。Bus_type 结构用于表示总线。而设备就用device 结构来表示。
4、每当产生热插拔事件，内核就会调用用户空间程序/sbin/hotplug 。

16、《Linux设备驱动程序》的笔记-第16章：块设备驱动程序

第16章：块设备驱动程序
1、对大多数块设备驱动程序来说，第一步是向内核注册自己，使用函数register_blkdev 来执行。而注销就使用unregister_blkdev 函数。对于磁盘设备，使用gendisk 结构来表示。
2、块设备驱动程序的核心是请求函数。请求函数原型：void request(request_queue_t *queue); ，驱动程序处理读取、写入以及其他对设备的操作时，就会调用该函数。请求队列跟踪未完成的块设备的I/O请求，使用blk_init_queue 函数来创建和初始化一个请求队列。返回队列中下一个要处理的请求的函数为 elv_next_request()。

17、《Linux设备驱动程序》的笔记-第30页

insmod 将模块的代码和数据装入内核，然后使用内核的符号表解析模块中未解析的符号

18、《Linux设备驱动程序》的笔记-第100页

参考

19、《Linux设备驱动程序》的笔记-第1页

花了整整一天时间，终于把内核树建立起来了，环境是：VMWare+ uBuntu10.10-i386+Linux-source-2.6.32.60+drm33.26, 现在可以开始我的kernel device driver之路了。

>下载内核：download from http://kernel.org
for ubuntu, it's better to use "sudo apt-get intall linux-sourcexxxx".
you can use 'apt-cache search linux-source' to search it, it's better to install a source with ubuntu patches.

>编译内核：参考：http://listsetio090529.blog.163.com/blog/static/132732184201171735051299/
值得另外说的一点是，如果想要用当前系统版本中的.config, 当然最好当前系统内核版本跟你要编译的一样，或者相差不多，可以cp /boot/config-xxxx /usr/sur/linux-source-xxxx/.config, 然后用make oldconfig就可以了。
>安装内核，也参考：http://listsetio090529.blog.163.com/blog/static/132732184201171735051299/
另外说的一点是，它没有自动生成initdxxxx.img文件，用这个命令：mkinitramfs -o initrd.img -2.6.27.7<个性化定制的名字> <版本名字>。
再把内核加入到grub menu里面：
sudo grub update, 不行的话再edit /boot/grub/grub.cfg
>重启系统 -- 按住--shift, 进入系统选择界面。
>做自己的第一个hello world, kernel module.参考
：http://listsetio090529.blog.163.com/blog/static/132732184201171735051299/

20、《Linux设备驱动程序》的笔记-第10章：中断处理

第10章：中断处理
　一般来说，如果一个驱动程序需要用到中断，只需为对应的设备的中断注册一个处理例程，并且在中断到达时进行正确的处理。
　第一节-安装中断处理例程。首先就需要申请一个中断通道（或者中断请求IRQ），可以使用内核函数request_irq()，但应该在设备第一次打开、硬件被告知产生中断之前。使用 free_irq() 释放通道。
　第二节-实现中断处理例程。中断处理例程的编写受到一些限制：不能向用户空间发送或者接受数据；不能做如何可能发生休眠的操作；不能调用schdule 函数。
　第三节-顶半部和底半部。顶半部是实际响应中断的例程，也就是调用 request_irq 注册的中断例程。而所谓的底半部是一个被顶半部调度，并在稍后更安全的时间内执行的例程。
　第四节-中断共享。中断通道不够用的时候就要使用中断共享了。共享的中断也是通过 request_irq() 安装的，只是它的 flags 参数有所不同。

21、《Linux设备驱动程序》的笔记-第11章：内核的数据类型

第11章：内核的数据类型
　内核使用的数据类型主要被分为三大类：类似 int 这样的标准C 语言类型，类似u32 这样的有确定大小的类型，以及像pid_t 这样的用于特定内核对象的类型。在适当的地方使用准确的数据类型，可以大大提高代码的移植性。
　在不同的体系架构上，普通C 语言的数据类型所占空间的大小可能并不相同，所以当需要使用确定大小的数据类型时就要特别注意。也可以使用Linux 系统特有的数据类型 u8，u16，u32 ,u65 ,表示确定位数的数据。
　为提高代码的移植性，应当注意以下几点。避免使用显式的常量值；不要假定每一秒一定有100个jiffies；内存页的大小为 PAGE_SIZE 字节，而不是4 KB ；不要做字节序的假设，系统可能为大端也可能为小端。

22、《Linux设备驱动程序》的笔记-第一章

　简介

linux系统介绍，以及编写驱动的一些常用知识

23、《Linux设备驱动程序》的笔记-第十二章：PCI 驱动程序

第十二章：PCI 驱动程序
PCI 的全称是 Peripheral Component interconnect （外围设备互联）。通俗来讲这就是一种总线，或者说PCI 是一种现在使用的最普遍的总线标准,定义了计算机的各个不同部分之间应该如何交互。
每个PCI 外设由一个总线编号、一个设备编号以及一个功能编号来标识。在单个系统中插入多个总线，可通过桥（bridge）来完成。PCI 中的I/O 空间使用32位地址总线。配置空间利用了地理寻址（geographical addressing）。在系统引导时，每个PCI 外设由主板上的固件自动配置。调用以struct pci_driver 指针为参数的pci_register_driver 函数可以注册一个PCI 驱动程序。设备接入后，处理中断的驱动程序特定代码只需要读取PCI_INTERRUPT_LINE 即可获得设备获取到的中断线。

24、《Linux设备驱动程序》的笔记-第6章：高级字符驱动程序操作

第6章：高级字符驱动程序操作
在第3章的基础上，这章介绍了编写全功能字符设备驱动程序的几个概念。
第一节-ioctl 。Ioctl 是驱动程序要实现的一类系统调用，是一个用于设备控制的公共接口。实现 ioctl 就要选择对应不同命令的编号、返回值、ioctl 参数等。
第二节-阻塞型I/O 。当设备不可用时，阻塞相应的进程是个不错的处理策略。但是让进程进入休眠，需要遵守两条重要规则：永远不要在原子上下文中进入休眠；当进程被唤醒时它无法知道它休眠的多长时间、其间都发生了些什么事。
第三节-poll 和 select 。Poll 、select 和 epoll 的功能本质上是一样的：让进程决定其是否可以对一个或多个打开的文件做非阻塞的读取或写入。
第四节-异步通知。这是通过发送SIGIO 信号来通知进程某件事情发生了的一种技术。首先，需要指定一个进程作为文件的“ 属主 ”，然后，用户程序在设备中设置 FASYNC 标志。
第五节-定位设备。如果设备操作未定义 llseek 方法，内核默认通过修改 filp->f_pos 来执行定位。
第六节-设备文件的访问控制。最生硬的方法是一次只允许一个进程打开设备，但这样及其不灵活。也可以限制每次只由一个用户访问，这样，该用户的不同进程就可以协作使用同一个设备了。有些设备可以实现“ 虚拟设备 ”，可以使得一个物理设备同时被多个进程使用。

# 《Linux设备驱动程序》

## 版权说明