

信息科学与技术丛书

冯国进 编著

Linux

驱动程序开发实例

第②版

- 基于 Linux 4.5 内核与 ARM 嵌入式系统
- 全面剖析 Linux 驱动程序开发精髓
- 涵盖多种硬件接口驱动程序
- 附赠完整实例源代码



 机械工业出版社
CHINA MACHINE PRESS

信息科学与技术丛书

Linux 驱动程序开发实例

第2版

冯国进 编著



机械工业出版社

设备驱动程序是应用程序与硬件设备之间的桥梁，驱动程序开发是软硬件结合的技术。本书深入介绍 Linux 设备驱动程序开发，涵盖了 Linux 驱动程序基础、驱动模型、内存管理、内核同步机制、I2C 驱动程序、串口驱动程序、LCD 驱动程序、网络驱动程序、USB 驱动程序、输入子系统驱动程序、块设备驱动程序、音频设备驱动程序等内容。全书以实例为主线，是为 Linux 设备驱动程序开发人员量身打造的学习书籍和实战指南。本书基于 Linux 4.5 内核，提供了丰富的实例代码和详细的注释，并附赠完整源代码供读者下载。本书主要面向各种层次的嵌入式 Linux 软硬件开发工程师，也可以作为各类嵌入式系统培训机构的培训教材和高校计算机课程教辅书籍。

图书在版编目 (CIP) 数据

Linux 驱动程序开发实例 / 冯国进编著. —2 版. —北京: 机械工业出版社, 2017.5
(信息科学与技术丛书)
ISBN 978-7-111-56706-6

I. ①L… II. ①冯… 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 (2017) 第 092047 号

机械工业出版社 (北京市百万庄大街 22 号 邮政编码 100037)

责任编辑: 车 忱

责任校对: 张艳霞

责任印制: 李 昂

三河市国英印务有限公司印刷

2017 年 7 月第 2 版 · 第 1 次印刷

184mm×260mm · 27 印张 · 652 千字

0001—3000 册

标准书号: ISBN 978-7-111-56706-6

定价: 89.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

电话服务

网络服务

服务咨询热线: (010) 88361066

机工官网: www.cmpbook.com

读者购书热线: (010) 68326294

机工官博: weibo.com/cmp1952

(010) 88379203

教育服务网: www.cmpedu.com

封面无防伪标均为盗版

金书网: www.golden-book.com

出版说明

随着信息科学与技术的迅速发展，人类每时每刻都会面对层出不穷的新技术和新概念。毫无疑问，在节奏越来越快的工作和生活中，人们需要通过阅读和学习大量信息丰富、具备实践指导意义的图书来获取新知识和新技能，从而不断提高自身素质，紧跟信息化时代发展的步伐。

众所周知，在计算机硬件方面，高性价比的解决方案和新型技术的应用一直备受青睐；在软件技术方面，随着计算机软件的规模和复杂性与日俱增，软件技术不断地受到挑战，人们一直在为寻求更先进的软件技术而奋斗不止。目前，计算机和互联网在社会生活中日益普及，掌握计算机网络技术和理论已成为大众的文化需求。由于信息科学与技术在电工、电子、通信、工业控制、智能建筑、工业产品设计与制造等专业领域中已经得到充分、广泛的应用，所以这些专业领域中的研究人员和工程技术人员越来越迫切需要汲取自身领域信息化所带来的新理念和新方法。

针对人们了解和掌握新知识、新技能的热切期待，以及由此促成的人们对语言简洁、内容充实、融合实践经验的图书迫切需要的现状，机械工业出版社适时推出了“信息科学与技术丛书”。这套丛书涉及计算机软件、硬件、网络和工程应用等内容，注重理论与实践的结合，内容实用、层次分明、语言流畅，是信息科学与技术领域专业人员不可或缺的参考书。

目前，信息科学与技术的发展可谓一日千里，机械工业出版社欢迎从事信息技术方面工作的科研人员、工程技术人员积极参与我们的工作，为推进我国的信息化建设做出贡献。

机械工业出版社

前 言

写作背景

自 1991 年问世以来，Linux 操作系统一直在创造着开源世界的神话，它已经在服务器、嵌入式系统、智能手机等领域大放异彩，当之无愧地成为了当前最重量级的操作系统。从最初的 Linux 0.01 版到现在的 Linux 4.x 版，让我们看到了 Linux 强大的生命力。我们有理由相信，Linux 操作系统将健康地发展下去。

自十多年前在 Linux 平台上开发第一个应用开始，我便喜爱上了 Linux 平台上的软件开发。从那之后，我有幸能够长期从事嵌入式 Linux 的驱动与应用开发，今后也将在 Linux 驱动开发领域持续耕耘。Linux 带给我无穷的乐趣，我也希望向读者介绍 Linux 平台的驱动开发技术，为 Linux 的发展贡献一点绵薄之力。本书上一版出版之后，很多热心读者发来建议，也促使我创作本书第 2 版。

设备驱动程序依然是 Linux 这个伟大的操作系统的最重要的部分，设备驱动程序开发也是实际项目中非常重要的任务。设备驱动程序关系到系统的稳定可靠，这就要求工程师具备严谨的工作态度。设备驱动程序开发是软件与硬件相结合的领域，希望读者能先了解一些硬件方面的知识，为学习本书打下基础。

“操千曲而后晓声，观千剑而后识器。”我始终认为要成为一个领域的专家，就需要长时间不断地练习以及总结，在实践中不断深入探索是最便捷的学习方法，所以本书遵循了实例驱动的学习模式。希望读者能够认真钻研每一个例程，并举一反三，早日成为一名合格的驱动开发工程师。

本书特点

- 实战性：本书提供多达三十多个驱动程序例程，非常适合各种层次的驱动程序开发人员。书中例子全部基于 Linux 4.5.2 内核。本书附赠代码包含了书中大部分实例的相关代码，读者可以免费下载。
- 全面性：本书涵盖了 Linux 驱动程序基础、驱动模型、内存管理、内核同步机制、I2C 驱动程序、LCD 驱动程序、网络驱动程序、USB 驱动程序、输入子系统驱动程序、块设备驱动程序、音频设备驱动等内容，是驱动程序开发人员的完整参考书。
- 易读性：本书以实例为主线，代码注释丰富，带领读者由浅入深掌握 Linux 驱动程序开发的精髓。

内容结构

本书内容丰富全面，涵盖了 Linux 4.5 下的三类驱动设备，包括字符设备、块设备、网络设备的开发技术。本书第 1~5 章为 Linux 驱动程序开发入门基础知识；第 6 章介绍基本的硬件设备驱动开发；第 7~15 章介绍各种硬件接口的驱动程序体系，包括 I2C、LCD、USB、输入设备、网络、TTY、音频等接口。

读者对象

本书是一本专门介绍嵌入式 Linux 驱动程序开发的书，读者应具备 C 语言编程和操作系

统方面的基础知识。本书主要面向嵌入式 Linux 系统的内核、设备驱动程序、应用程序的开发工程师以及 ARM 嵌入式系统的硬件设计工程师，也可以作为各类嵌入式系统培训机构的培训实验教材和高校操作系统课程的辅导书籍。

特别致谢

在朋友、家人和机械工业出版社的帮助和支持下，本书终于得以问世，在此对他们表示衷心的感谢。特别是责任编辑车忱老师，在本书编写过程中提出了大量合理的建议，使本书得以顺利出版。

本书大部分例程基于深圳友坚恒天的 idea6410 开发板，在此对他们表示特别的感谢。本人希望能够和读者一起努力，扩大交流，共同进步。由于 Linux 驱动程序开发相当博大精深，加之本人水平有限，本书错误在所难免，请各位读者原谅并指正。读者可把修改建议发送到 fgjnew@163.com，以便再版时修正。

冯国进

2016 年 10 月 1 日

目 录

出版说明

前言

第 1 章 Linux 设备驱动程序入门	1
1.1 设备驱动程序基础	1
1.1.1 驱动程序的概念	1
1.1.2 驱动程序的加载方式	2
1.1.3 编写可加载模块	3
1.1.4 带参数的可加载模块	4
1.1.5 模块依赖	5
1.1.6 printk 的等级	7
1.1.7 设备驱动程序类别	8
1.2 字符设备驱动程序原理	9
1.2.1 file_operations 结构	9
1.2.2 使用 register_chrdev 注册字符设备	11
1.2.3 使用 cdev_add 注册字符设备	14
1.2.4 字符设备的读写	16
1.2.5 IOCTL 接口	17
1.2.6 seek 接口	20
1.2.7 poll 接口	22
1.2.8 异步通知	26
1.3 seq_file 机制	28
1.3.1 seq_file 原理	28
1.3.2 seq_file 实例	29
1.4 /proc 文件系统	35
1.4.1 /proc 文件系统概述	35
1.4.2 /proc 文件系统接口	36
1.5 Linux 内核导读	40
1.5.1 Linux 内核组成	40
1.5.2 Linux 的代码结构	42
1.5.3 内核 Makefile	43
第 2 章 Linux 设备驱动模型	44
2.1 内核对象	44
2.1.1 kobject	44
2.1.2 kobj_type	45
2.1.3 kset	45
2.2 设备模型层次	46

2.3	sysfs 文件系统	49
2.4	platform 概念	51
2.5	Attributes	56
2.6	设备事件通知	60
2.6.1	kobject uevent	60
2.6.2	uevent helper	61
2.6.3	udev	63
2.7	设备树	64
第 3 章	Linux 内核同步机制	67
3.1	原子操作	67
3.2	锁机制	68
3.2.1	自旋锁	68
3.2.2	读写锁	70
3.2.3	RCU	71
3.2.4	信号量	75
3.2.5	读写信号量	77
3.2.6	互斥量	77
3.3	等待队列	78
3.3.1	等待队列原理	78
3.3.2	阻塞模式读实例	78
3.3.3	完成事件	81
3.4	通知链	83
第 4 章	内存管理与链表	86
4.1	物理地址和虚拟地址	86
4.2	内存分配与释放	87
4.3	cache	88
4.4	IO 端口到虚拟地址的映射	88
4.4.1	静态映射	88
4.4.2	ioremap	89
4.5	内核空间到用户空间的映射	90
4.5.1	mmap 接口	90
4.5.2	mmap 系统调用	91
4.6	DMA 映射	93
4.7	内核链表	93
4.7.1	Linux 内核中的链表	93
4.7.2	内核链表实例	95
第 5 章	任务与调度	98
5.1	schedule	98
5.2	内核线程	99

5.3	内核调用应用程序	101
5.4	软中断机制	103
5.4.1	软中断原理	103
5.4.2	tasklet	106
5.5	工作队列	108
5.5.1	工作队列原理	108
5.5.2	延迟工作队列	110
5.6	内核时间	110
5.6.1	Linux 下的时间概念	110
5.6.2	Linux 下的延迟	111
5.6.3	内核定时器	112
第 6 章	简单硬件设备驱动程序	115
6.1	硬件基础知识	115
6.1.1	硬件设备原理	115
6.1.2	时序图原理	116
6.1.3	嵌入式 Linux 系统构成	117
6.1.4	硬件初始化	117
6.1.5	clk 体系	120
6.2	dev/mem 与 dev/kmem	121
6.3	寄存器访问	124
6.3.1	S3C6410X 地址映射	124
6.3.2	S3C6410X 看门狗驱动程序实例	128
6.4	电平控制	131
6.4.1	S3C6410X LED 驱动程序实例	132
6.4.2	扫描型按键驱动程序实例	135
6.5	硬件中断处理	137
6.5.1	硬件中断处理原理	137
6.5.2	中断型按键驱动程序实例	141
6.6	看门狗驱动架构	146
6.7	RTC 驱动	148
6.8	LED 类设备	153
第 7 章	I2C 设备驱动程序	157
7.1	I2C 接口原理	157
7.2	Linux 的 I2C 驱动程序架构	159
7.2.1	I2C 适配器	160
7.2.2	I2C 算法	161
7.2.3	I2C 从设备	161
7.2.4	I2C 从设备驱动	162
7.2.5	I2C 从设备驱动开发	163

7.3	I2C 控制器驱动	163
7.3.1	S3C2410X 的 I2C 控制器	163
7.3.2	S3C2410X 的 I2C 控制器驱动	164
7.4	通用 I2C 从设备	172
7.4.1	通用 I2C 从设备驱动	172
7.4.2	通过 read 与 write 接口读写	174
7.4.3	通过 I2C_RDWR 命令读写	177
7.4.4	I2Ctools	180
7.5	个性化 I2C 从设备驱动	181
第 8 章	TTY 与串口驱动程序	185
8.1	TTY 概念	185
8.2	Linux TTY 驱动程序体系	185
8.2.1	TTY 驱动程序架构	185
8.2.2	TTY 文件层	186
8.2.3	线路规程层	188
8.2.4	TTY 驱动层	190
8.2.5	TTY 数据链路分析	193
8.3	串口驱动层	194
8.3.1	uart_driver	194
8.3.2	uart_port	195
8.4	S3C6410X 串口设备驱动程序	197
8.5	TTY 应用层	201
第 9 章	Framebuffer 驱动程序	203
9.1	Linux Framebuffer 驱动程序原理	203
9.1.1	Framebuffer 核心数据结构	203
9.1.2	Framebuffer 操作接口	206
9.1.3	Framebuffer 驱动的文件接口	207
9.1.4	Framebuffer 驱动框架代码分析	209
9.2	S3C6410X 显示控制器	210
9.3	S3C6410X LCD 驱动程序实例	215
9.3.1	注册与初始化	215
9.3.2	fb_ops 实现	220
9.3.3	DMA 传输机制	222
9.3.4	内核配置	227
9.4	Framebuffer 应用层	227
9.5	Qt 界面系统移植	229
第 10 章	输入子系统	231
10.1	Linux 输入子系统概述	231
10.2	Linux 输入子系统原理	231

10.2.1	输入设备	232
10.2.2	输入事件	233
10.2.3	input Handler 层	234
10.2.4	常用的 Input Handler	236
10.3	输入设备应用层	241
10.4	键盘输入设备驱动程序实例	243
10.5	Event 接口实例	249
10.6	触摸屏驱动程序实例	253
10.6.1	S3C6410X 触摸屏控制器	253
10.6.2	S3C6410X 触摸屏驱动程序	255
10.7	Linux 红外遥控驱动	263
第 11 章	块设备驱动与文件系统	268
11.1	块设备驱动原理	268
11.1.1	block_device	268
11.1.2	gendisk	269
11.1.3	bio	270
11.1.4	请求队列	271
11.2	Linux 文件系统概述	276
11.2.1	虚拟文件系统	277
11.2.2	日志文件系统和非日志文件系统	278
11.2.3	根文件系统	279
11.2.4	文件系统总结	280
11.2.5	文件系统挂载	280
11.3	虚拟文件系统接口	281
11.3.1	VFS 文件接口	281
11.3.2	VFS 目录接口	283
11.4	根文件系统制作	284
11.4.1	Busybox	284
11.4.2	shell 基础	286
11.4.3	根文件系统构建实例	288
11.4.4	添加 mdev	288
11.5	NFS 根文件系统搭建	289
第 12 章	NAND Flash 驱动	293
12.1	MTD 设备层	293
12.1.1	MTD 架构	293
12.1.2	MTD 字符设备	295
12.1.3	MTD 块设备	300
12.2	NAND Flash 驱动层概述	304
12.2.1	硬件原理	304

12.2.2	NAND 核心层架构	305
12.2.3	NAND Flash 坏块处理	308
12.3	S3C6410X NAND Flash 驱动	310
12.4	Ubifs 文件系统实例	315
第 13 章	网络设备驱动程序	319
13.1	网络设备程序概述	319
13.1.1	网络设备的特殊性	319
13.1.2	sk_buff 结构	320
13.1.3	网络设备驱动程序架构	321
13.1.4	虚拟网络设备驱动程序实例	325
13.1.5	网络硬件接口的分层结构	329
13.2	DM9000A 网卡驱动程序开发	329
13.2.1	DM9000A 原理	329
13.2.2	DM9000A 驱动程序分析	331
13.2.3	DM9000A 网卡驱动程序移植	341
13.4	ethtool	344
13.5	PHY 芯片驱动	347
13.6	Netlink Socket	352
13.6.1	Netlink 机制	352
13.6.2	Netlink 应用层编程	357
13.6.3	Netlink 驱动程序实例	357
第 14 章	USB 驱动程序	361
14.1	USB 体系概述	361
14.1.1	USB 系统组成	361
14.1.2	USB 主机	361
14.1.3	USB 设备逻辑层次	362
14.2	Linux USB 驱动程序体系	364
14.2.1	USB 总体结构	364
14.2.2	USB 设备驱动	364
14.2.3	USB 设备	365
14.2.4	主机控制器驱动	366
14.2.5	USB 请求块 urb	367
14.3	USB 设备枚举	370
14.4	S3C6410X USB 主机控制器驱动程序	372
14.4.1	驱动程序原理分析	372
14.4.2	S3C6410X 加载 U 盘实例	374
14.5	USB 键盘设备驱动程序分析	375
第 15 章	音频设备驱动程序	380
15.1	ALSA 音频体系	380

15.2	ALSA 核心层	381
15.2.1	声卡	381
15.2.2	音频设备	382
15.2.3	PCM	382
15.2.4	音频控制接口	384
15.2.5	AC97 声卡	387
15.3	ALSA SOC 架构	388
15.3.1	SOC 声卡	389
15.3.2	DAI	392
15.3.3	codec	393
15.3.4	SOC 平台	394
15.3.5	PCM 运行时配置	394
15.3.6	DAPM	397
15.4	ALSA 驱动程序实例	400
15.4.1	S3C6410X 的 AC97 控制单元	401
15.4.2	Machine Driver	402
15.4.3	Platform Driver	403
15.4.4	Codec Driver	408
15.5	ALSA 音频缓冲逻辑	409
15.6	ALSA 应用编程接口	413
	参考文献	418

第 1 章 Linux 设备驱动程序入门

到目前为止，设备驱动程序代码仍然是 Linux 内核代码中最多的部分。操作系统最重要的功能之一就是支持各类设备的访问，承担硬件与应用软件之间的桥梁作用。Linux 操作系统中主要包含字符设备、块设备、网络设备等三类基本的设备驱动程序，内核中的设备驱动程序大部分基于这三类设备驱动。本章主要介绍 Linux 设备驱动程序的入门知识，包含模块基础、字符设备驱动、proc 文件系统等内容。

1.1 设备驱动程序基础

1.1.1 驱动程序的概念

所谓设备驱动程序，就是驱使设备按照用户的预期进行工作的软件，它是应用程序与设备沟通的桥梁。从本质上讲，设备驱动程序主要负责硬件设备的参数配置、数据读写与中断处理。Linux 的运行空间分为内核空间与用户空间。为了保护系统的安全，这两个空间各自运行在不同的级别，不能相互直接访问和共享数据。Linux 内核为应用层提供了一系列系统调用接口，应用程序可以通过这组接口来获得操作系统内核提供的服务。应用层程序运行在用户态，而设备驱动程序是操作系统的一部分，运行在内核态。应用程序要控制硬件设备，首先通过系统调用访问内核，内核层根据系统调用号来调用驱动程序对应的接口函数来访问设备。图 1-1 说明了 Linux 驱动程序的运行原理。

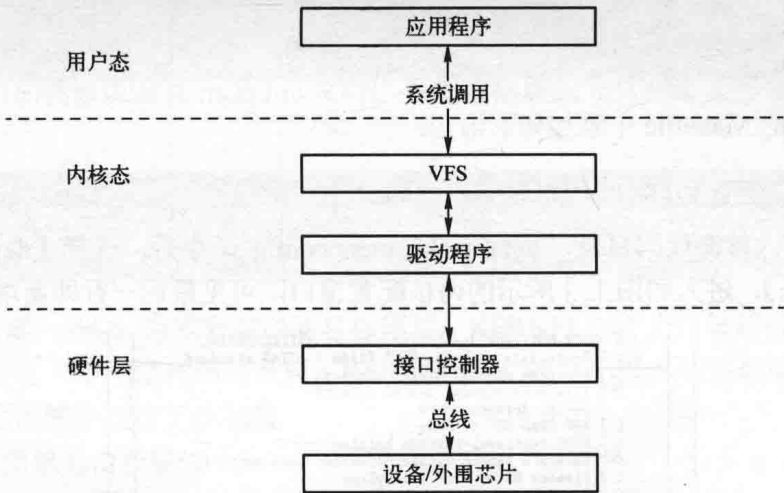


图 1-1 设备驱动程序的原理

Linux 中的大部分驱动程序是以内核模块的形式编写的。内核模块是 Linux 内核向外部提供的一个接口，其全称为动态可加载内核模块 (Loadable Kernel Module, LKM)。Linux

内核本身是一个单内核（monolithic kernel），具有效率高的优点，也具有可扩展性和可维护性差的缺陷。模块机制就是为了弥补这一缺陷而生。内核模块可以被单独编译，它在运行时被链接到内核作为内核的一部分在内核空间运行。要让内核支持可加载模块，需要配置内核的【Enable loadable module support】选项，如图 1-2 所示。

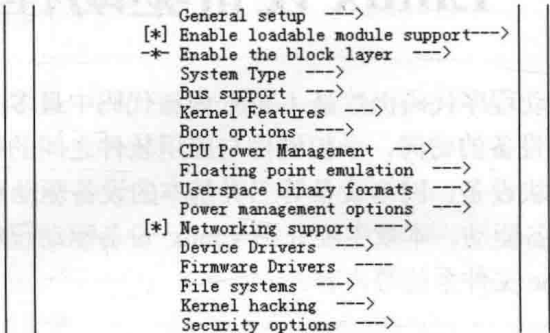


图 1-2 在内核中增加可加载模块支持

1.1.2 驱动程序的加载方式

Linux 设备驱动程序有两种加载方式。一种是直接编译进 Linux 内核，在 Linux 启动时加载；另一种是采用内核模块方式，这种模块可动态加载与卸载。

如果希望将新驱动程序编译进内核，需要修改内核代码和编译选项。下面以字符型设备为例，说明如何在 Linux 内核中添加一个新的设备驱动程序。如果驱动程序代码源文件为 infrared_s3c2410.c，将 infrared_s3c2410.c 复制到内核代码的/drivers/char 目录，并在该目录下的 Kconfig 文件最后增加如下语句：

```
config INFRARED_REMOTE
    tristate "INFRARED Driver for REMOTE"
    depends on ARCH_S3C64XX || ARCH_S3C2410
    default y
    help
```

在该目录下的 Makefile 中添加如下语句：

```
Obj-$(CONFIG_INFRARED_REMOTE)+= infrared_s3c2410.o
```

进入 Linux 内核源代码目录，执行 make menuconfig 命令后，选择【device drivers】->【character devices】，进入如图 1-3 所示的内核配置窗口，可见最后一行即新增的驱动：

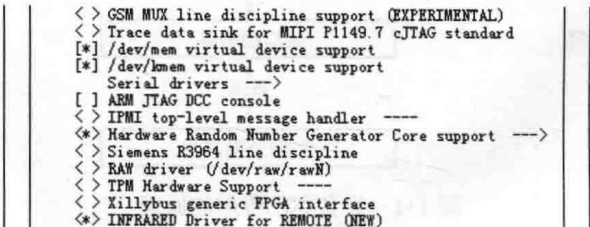


图 1-3 在内核中增加新驱动程序

在内核配置窗口中可以使用上下键、空格键和回车键进行选择、移动和取消选择。内核配置窗口中以< >开头的行是内核模块的配置，以[]开头的行是内核功能的配置。选项前如果为<*>，表示相应的模块将被编译进内核。如果选项前是< >则表示不编译进内核。这里在【INFRARED Driver for REMOTE】行前面设置为<*>，则 `infrared_s3c2410.o` 将被编译进内核。在使用 `make zImage` 命令编译内核时所有设置为<*>的项将被包含在内核映像中。

采用可加载模块方式让驱动程序的运行更加灵活，也更利于调试。可加载模块用于扩展 Linux 操作系统的功能。使用内核模块的优点是可以按照需要进行加载，而且不需要重新编译内核。这种方式控制了内核的大小，而模块一旦被插入内核，它就和内核其他部分一样，可以访问内核的地址空间、函数和数据。可加载模块通常以 `.ko` 为扩展名。在图 1-3 中选项前如果为<M>，表示编译成可加载模块。在使用 `make modules` 命令编译内核时，所有设置为<M>的项将被编译。`make modules` 结束后可以使用下面的命令安装内核中的可加载模块文件到一个指定的目录：

```
make modules_install INSTALL_MOD_PATH=/home/usr/modules
```

使用 `make` 命令编译内核相当于执行 `make zImage` 和 `make modules` 两个命令。

1.1.3 编写可加载模块

Linux 内核模块必须包含以下两个接口：

```
module_init(your_init_func);    //模块初始化接口
module_exit(your_exit_func);    //模块卸载接口
```

加载一个内核模块的命令是 `insmod`，格式如下：

```
#insmod modulename.ko
```

卸载一个内核模块的命令是 `rmmmod`，格式如下：

```
#rmmmod modulename
```

可加载模块的源代码可以放在内核代码树中，也可以独立于内核代码树。如果是后一种情况，需要为可加载模块编写 `makefile` 文件。可加载模块的 `makefile` 文件最重要的就是设置如下几个变量：

```
CC= arm-none-linux-gnueabi-gcc
obj-m:= smodule.o
KERNELDIR ?= /root/fgj/linux-4.5.2
```

`CC` 是编译器，`obj-m` 为需要编译的目标模块，`KERNELDIR` 为内核路径。注意在编写可加载模块前先要有一个内核代码目录树。`KERNELDIR` 的内核版本必须与运行的内核版本一致，否则编译出的模块往往无法加载。

例 1.1 最简单的内核模块

代码见 `\samples\ldoor\l-simple`。核心代码如下：

```
static int demo_module_init(void)
{
    printk("demo_module_init\n");
```

```

        return 0;
    }
    static void demo_module_exit(void)
    {
        printk("demo_module_exit\n");
    }
    //模块入口
    module_init(demo_module_init);
    module_exit(demo_module_exit);
    MODULE_DESCRIPTION("simple module");
    MODULE_LICENSE("GPL");

```

模块运行在内核态，不能使用用户态 C 库函数中的 `printf` 函数，而要用 `printk` 函数打印调试信息。编写一个 Makefile 文件如下：

```

AR = ar
ARCH      = arm
CC = arm-none-linux-gnueabi-gcc
DEBFLAGS = -O2
obj-m      := smodule.o
KERNELDIR ?= /root/fgj/linux-4.5.2
PWD        := $(shell pwd)
modules:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) LDDINC=$(PWD)/../include modules
clean:
    rm -rf *.o *~ core .depend *.cmd *.ko *.mod.c .tmp_versions

```

执行 `make` 后生成 `smodule.ko`。运行结果如下：

```

[root@urbetter drivers]# insmod smodule.ko
demo_module_init
[root@urbetter drivers]# cat /proc/modules
smodule 868 0 - Live 0xbf000000 (O)
[root@urbetter drivers]# rmmod smodule
rmmod: can't change directory to '/lib/modules': No such file or directory
[root@urbetter /home]# mkdir -p /lib/modules/$(uname -r)
[root@urbetter drivers]# rmmod smodule
demo_module_exit

```

第一次运行 `rmmod smodule` 会失败，因为需要在 `/lib/modules` 目录下建立以内核版本号名称的目录，才能正确卸载模块。`uname -r` 用来得到内核版本号。建立正确的目录后，模块可以正常卸载。

1.1.4 带参数的可加载模块

宏 `MODULE_PARM(var,type,right)` 用于向模块传递命令行参数。参数类型可以是整数、长整型、字符串等类型。