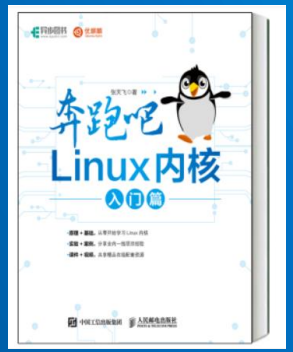
奔跑吧LINUX内核 * 入门篇

第四章 内核模块

笨叔叔



目 录

- Linux内核模块
- 实验

Linux内核模块

内核模块

- 内核模块Loadable Kernel Module（LKM）
  - ✓ Linux内核在运行时加载一组目标代码来实现某个特定的功能，这样在实际使用Linux的过程中可以不需要重新编译内核代码来实现动态扩展。
  - ✓ Linux内核采用宏内核架构，即操作系统的大部分功能都在内核中实现，比如进程管理、内存管理、进程调度、设备管理等，并且都在特权模式下（内核空间）运行

从一个简单内核模块开始

- 最简单的内核模块，打印“my first kernel module init”

```
0     #include <linux/init.h>
1     #include <linux/module.h>
2
3     static int __init my_test_init(void)
4     {
5         printk("my first kernel module init\n");
6         return 0;
7     }
8
9     static void __exit my_test_exit(void)
10    {
11        printk("goodbye\n");
12    }
13
14    module_init(my_test_init);
15    module_exit(my_test_exit);
16
17    MODULE_LICENSE("GPL");
18    MODULE_AUTHOR("Ben Shushu");
19    MODULE_DESCRIPTION("my test kernel module");
20    MODULE_ALIAS("mytest");
```

- Makefile文件
  - ✓ 编写一个对应的Makefile文件

```
0     BASEINCLUDE ?= /lib/modules/`uname -r`/build
1
2     mytest-objs := my_test.o
3     obj-m    :=   mytest.o
4
5     all :
6     $(MAKE) -C $(BASEINCLUDE) M=$(PWD) modules;
7
8     clean:
9     $(MAKE) -C $(BASEINCLUDE) SUBDIRS=$(PWD) clean;
10    rm -f *.ko;
```

- 运行make命令来编译内核模块
- 通过file命令来查看是内核模块是否编译正确

```
$file mytest.ko
mytest.ko: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), BuildID[
sha1]=57aa8267c3049e08ac8f7e47b4e378c284c8d5c3, not stripped
```

- 通过modinfo命令查看模块信息

```
$modinfo mytest.ko
filename:       /home/figo/work/runninglinuxkernel_4.0/module_test/module_test_case/
simple_module/mytest.ko
alias:          mytest
description:    my test kernel module
author:         Ben Shushu
license:        GPL
srcversion:     BE0A0951E6B195A8836CB99
depends:
retpoline:      Y
name:           mytest
vermagic:       4.15.0-20-generic SMP mod_unload modversions
```

- 使用insmod命令加载模块

```
$sudo insmod mytest.ko
```

- dmesg命令查看内核log信息

```
$dmesg
…
[258.575353] my first kernel module init
```

- lsmod命令查看已加载模块依赖关系

```
$ lsmod
Module                  Size  Used by
mytest                 16384  0
bnep                   24576  2
xt_CHECKSUM            16384  1
```

- 系统会在/sys/modules目录下新建一个目录

```
figo@figo-OptiPlex-9020:/sys/module/mytest$ tree  -a
.
├── coresize
├── holders
├── initsize
├── initstate
├── notes
│   └── .note.gnu.build-id
├── refcnt
├── sections
│   ├── .exit.text
│   ├── .gnu.linkonce.this_module
│   ├── .init.text
│   ├── __mcount_loc
│   ├── .note.gnu.build-id
│   ├── .orc_unwind
│   ├── .orc_unwind_ip
│   ├── .rodata.str1.1
│   ├── .strtab
│   └── .symtab
├── srcversion
├── taint
└── uevent

3 directories, 18 files
figo@figo-OptiPlex-9020:/sys/module/mytest$
```

内核模块总结

- 模块加载函数：加载模块时，该函数会被自动执行，通常做一些初始化工作。
- 模块卸载函数：卸载模块时，该函数也会被自动执行，做一些清理工作。
- 模块许可声明：内核模块必须声明许可证，否则内核会发出被污染的警告。
- 模块参数：根据需求来添加，为可选项。
- 模块作者和描述声明：一般都需要完善这些信息。
- 模块导出符号：根据需求来添加，为可选项。

模块参数

- 模块可以传递参数
  - ✓ 内核模块作为一个可扩展的动态模块，为Linux内核提供了灵活性。但是有时我们需要根据不同的应用场景给内核模块传递不同的参数，Linux内核提供一个宏来实现模块的参数传递。

```
#define module_param(name, type, perm)                \
    module_param_named(name, name, type, perm)

#define MODULE_PARM_DESC(_parm, desc) \
    __MODULE_INFO(parm, _parm, #_parm ":" desc)
```

- ✓ module_param()宏由3个参数组成，name表示参数名，type表示参数类型，perm表示参数的读写等权限。
- ✓ MODULE_PARM_DESC()宏为这个参数的简单说明，参数类型可以是byte、short、ushort、int、uint、long、ulong、char和bool等类型。

模块参数

- 例子

```
36 static int debug = 1;
37 module_param(debug, int, 0644);
38 MODULE_PARM_DESC(debug, "enable debugging information");
39
40 MODULE_DESCRIPTION("altera FPGA kernel module");
41 MODULE_AUTHOR("Igor M. Liplianin  <liplianin@netup.ru>");
42 MODULE_LICENSE("GPL");
43
44 #define dprintk(args...) \
45         if (debug) { \
46                 printk(KERN_DEBUG args); \
47         }
48
```

driver/misc/altera-stapl/altera.c

这个例子定义了一个模块参数debug，类型是int，初始化值为1，权限访问为0644。也就是说root权限用户可以修改这个值，这个参数的用途是打开调试信息。其实这是一个比较常用的内核调试方法，可以通过模块参数使用调试功能。

符号共享

- 我们在为一个设备编写驱动程序时，会把驱动按照功能分成好几个内核模块，这些内核模块之间有一些接口函数需要相互调用，这怎么实现呢？Linux内核为我们提供两个宏来解决这个问题。

```
EXPORT_SYMBOL( )
EXPORT_SYMBOL_GPL( )
```

- EXPORT_SYMBOL()把函数或者符号对全部内核代码公开，也就是将一个函数以符号的方式导出给内核中的其他模块使用。
- EXPORT_SYMBOL_GPL()只能包含GPL许可的模块，内核核心的大部分模块导出来的符号都是使用GPL()这种形式的。
- 如果要使用EXPORT_SYMBOL_GPL()导出函数，那么需要显式地通过模块申明为“GPL”，如MODULE_LICENSE("GPL")。
- 内核导出的符号表可以通过/proc/kallsyms来查看。

实验

实验1：编写一个简单的内核模块

- 实验目的
  - ✓ 了解和熟悉编译一个基本的内核模块需要包含的元素。
- 实验步骤
  - ✓ 1）编写一个简单的内核模块程序。
  - ✓ 2）编写对应的Makefile文件。
  - ✓ 3）在优麒麟Linux机器上编译和加载运行该内核模块。
  - ✓ 4）在QEMU上运行ARM32的Linux系统，编译该内核模块并运行

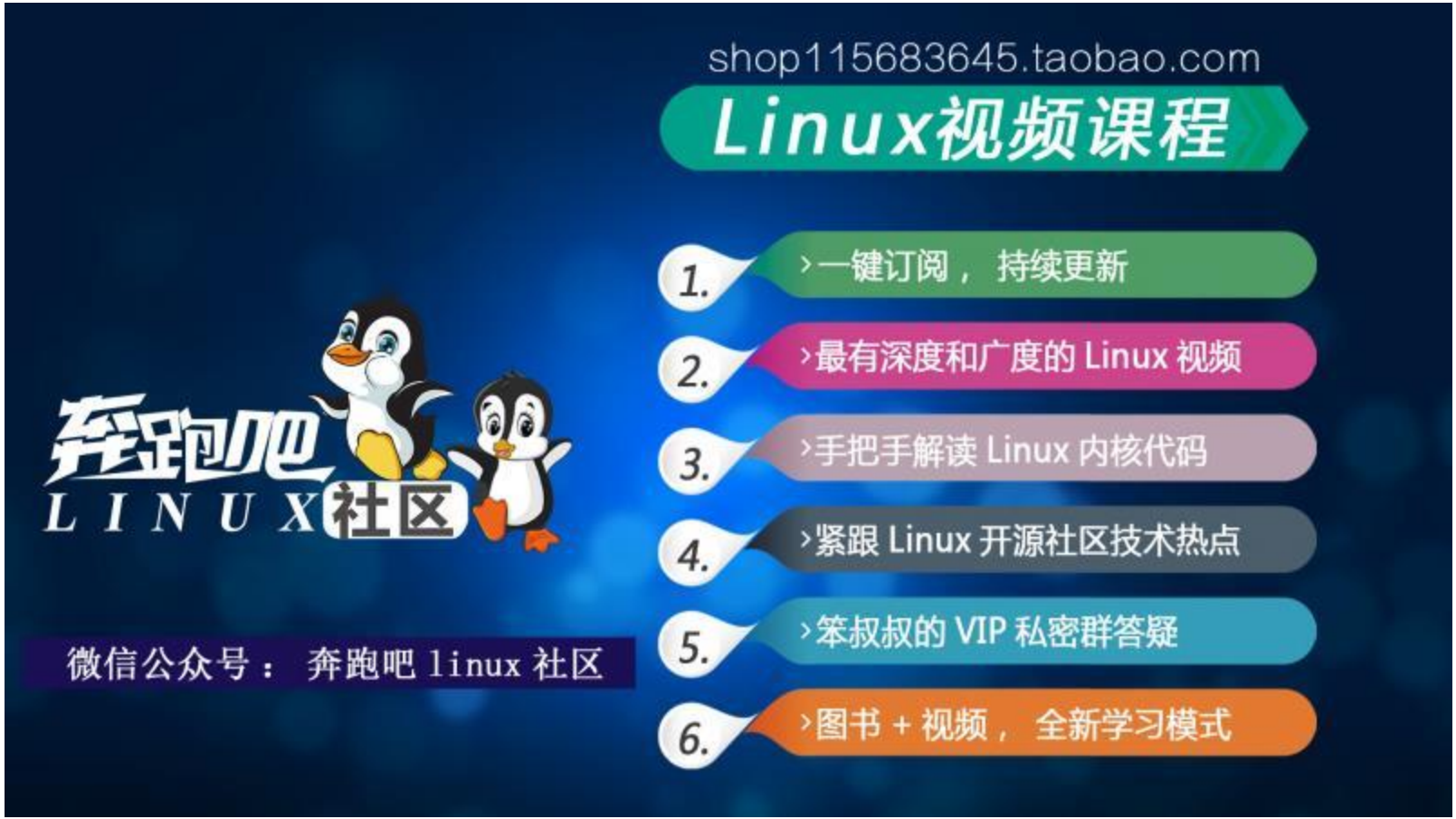
实验2：向内核模块传递参数

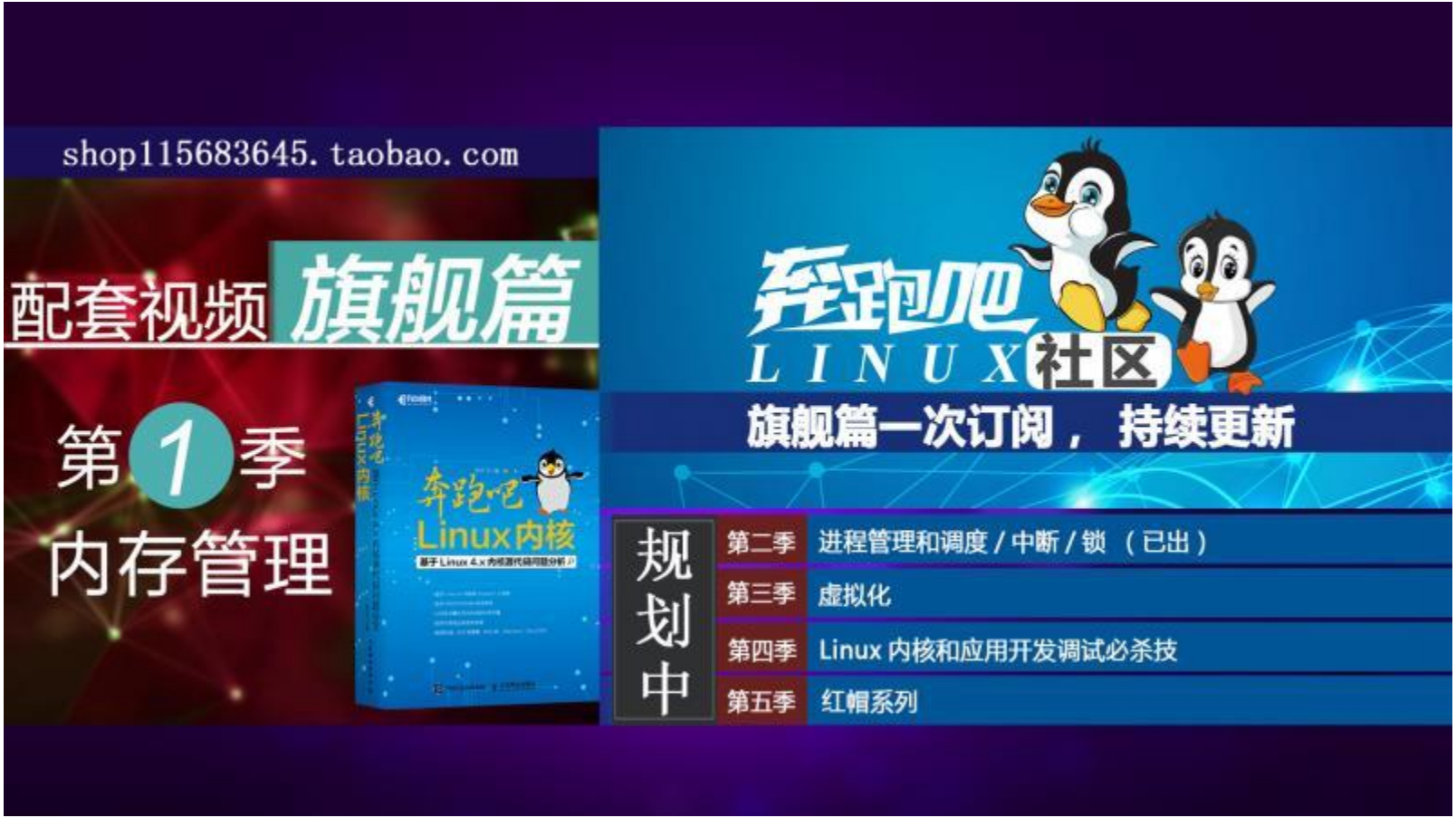
- 实验目的
  - ✓ 学会如何向内核模块传递参数。
- 实验步骤
  - ✓ 1）编写一个内核模块，通过模块参数的方式向内核模块传递参数。

实验3：在模块之间导出符号

- 实验目的
  - ✓ 学会如何在模块之间导出符号。
  - ✓ 在设计模块时考虑其层次结构。
- 实验步骤

BACKUP

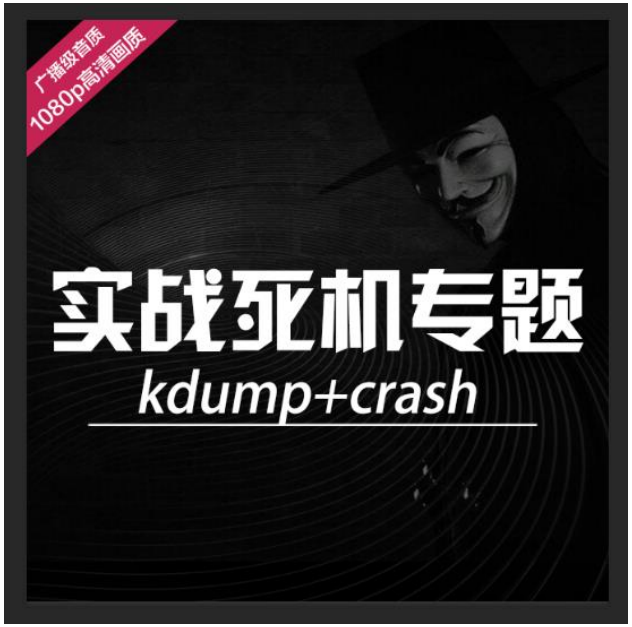

shop115683645.taobao.com
Linux视频课程
奔跑吧 LINUX社区
微信公众号： 奔跑吧 linux 社区
1. 一键订阅， 持续更新
2. 最有深度和广度的 Linux 视频
3. 手把手解读 Linux 内核代码
4. 紧跟 Linux 开源社区技术热点
5. 笨叔叔的 VIP 私密群答疑
6. 图书 + 视频， 全新学习模式


shop115683645.taobao.com
配套视频 旗舰篇
第1季
内存管理
奔跑吧Linux内核
基于Linux 4.x内核源代码问题分析
奔跑吧 LINUX社区
旗舰篇一次订阅， 持续更新
规划中
第二季 进程管理和调度 / 中断 / 锁 （已出）
第三季 虚拟化
第四季 Linux 内核和应用开发调试必杀技
第五季 红帽系列

第1季旗舰篇课程目录

课程名称	时长
序言一： Linux内核学习方法论	0:09:13
序言二： 学习前准备	
序言2.1 Linux发行版和开发板的选择	0:13:56
序言2.2 搭建Qemu+gdb单步调试内核	0:13:51
序言2.3 搭建Eclipse图形化调试内核	0:10:59
实战运维1：查看系统内存信息的工具（一）	0:20:19
实战运维2：查看系统内存信息的工具（二）	0:16:32
实战运维3：读懂内核log中的内存管理信息	0:25:35
实战运维4：读懂 proc meminfo	0:27:59
实战运维5：Linux运维能力进阶线路图	0:09:40
实战运维6：Linux内存管理参数调优(一)	0:19:46
实战运维7：Linux内存管理参数调优(二)	0:31:20
实战运维8：Linux内存管理参数调优(三)	0:22:58
运维高级如何单步调试RHEL— CENTOS7的内核一	0:15:45
运维高级如何单步调试RHEL— CENTOS7的内核二	0:41:28
vim:打造比source insight更强更好用的IDE(一)	0:24:58
vim:打造比source insight更强更好用的IDE(二)	0:20:28
vim:打造比source insight更强更好用的IDE(三)	0:23:25
实战git项目和社区patch管理	
2.0 Linux内存管理背景知识介绍	
奔跑2.0.0 内存管理硬件知识	0:15:25
奔跑2.0.1 内存管理总览一	0:23:27
奔跑2.0.2 内存管理总览二	0:07:35
奔跑2.0.3 内存管理常用术语	0:09:49
奔跑2.0.4 内存管理究竟管些什么东西	0:28:02
奔跑2.0.5 内存管理代码框架导读	0:38:09
2.1 Linux内存初始化	
奔跑2.1.0 DDR简介	0:06:47
奔跑2.1.1 物理内存三大数据结构	0:19:39
奔跑2.1.2 物理内存初始化	0:11:13
奔跑2.1 内存初始化之代码导读一	0:43:54
奔跑2.1 内存初始化之代码导读二	0:23:31
2.2 页表的映射过程	
奔跑2.2.0 ARM32页表的映射	0:08:54
奔跑2.2.1 ARM64页表的映射	0:10:58
奔跑2.2.2 页表映射例子分析	0:11:59
奔跑2.2.3 ARM32页表映射那些奇葩的事	0:09:42
2.3 内存布局图	
奔跑2.3.1 内存布局一	0:10:35
奔跑2.3.2 内存布局二	0:13:30
2.4 分配物理页面	
奔跑2.4.1 伙伴系统原理	0:10:10
奔跑2.4.2 Linux内核中的伙伴系统和碎片化	0:11:14
奔跑2.4.3 Linux的页面分配器	0:21:37
2.5 slab分配器	
奔跑2.5.1 slab原理和核心数据结构	0:18:36
奔跑2.5.2 Linux内核中slab机制的实现	0:16:56
2.6 vmalloc分配	
奔跑2.6 vmalloc分配	0:15:48
2.7 VMA操作	
奔跑2.7 VMA操作	0:16:42
2.8 malloc分配器	
奔跑2.8.1 malloc的三个迷惑	0:17:41
奔跑2.8.2 内存管理的三个重要的函数	0:17:38
2.9 mmap分析	
奔跑2.9 mmap分析	0:23:14
2.10 缺页中断处理	
奔跑2.10.1 缺页中断一	0:31:07
奔跑2.10.2 缺页中断二	0:16:58
2.11 page数据结构	
奔跑2.11 page数据结构	0:29:41
2.12 反向映射机制	
奔跑2.12.1 反向映射机制的背景介绍	0:19:01
奔跑2.12.2 RMAP四部曲	0:07:31
奔跑2.12.3 手撕Linux2.6.11上的反向映射机制	0:07:35
奔跑2.12.4 手撕Linux4.x上的反向映射机制	0:10:08
2.13 回收页面	
奔跑2.13 页面回收一	0:16:07
奔跑2.13 页面回收二	0:11:41
2.14 匿名页面的生命周期	0:26:16
2.15 页面迁移	0:19:07
2.16 内存规整	0:24:03
2.17 KSM	0:28:17
2.20 Meltdown漏洞分析	
奔跑2.20.1 Meltdown背景知识	0:10:13
奔跑2.20.2 CPU体系结构之指令执行	0:11:25
奔跑2.20.3 CPU体系结构之乱序执行	0:11:03
奔跑2.20.4 CPU体系结构之异常处理	0:03:48
奔跑2.20.5 CPU体系结构之cache	0:10:56
奔跑2.20.6 进程地址空间和页表及TLB	0:17:39
奔跑2.20.7 Meltdown漏洞分析	0:06:04
奔跑2.20.8 Meltdown漏洞分析之x86篇	0:12:07
奔跑2.20.9 ARM64上的KPTI解决方案	0:25:39
代码导读	
奔跑2.1 内存初始化之代码导读一	0:43:54
奔跑2.1 内存初始化之代码导读二	0:23:31
奔跑2.1 代码导读C语言部分（一）	0:27:34
奔跑2.1 代码导读C语言部分（二）	0:21:28
代码导读3页表映射	1:12:40
代码导读4分配物理页面	0:55:57
git入门和实战	
git入门与实战：节目总览	0:08:48
git入门与实战1：建立本地的git仓库	0:30:53
git入门与实战2：快速入门	0:12:45
git入门与实战3：分支管理	0:24:27
git入门与实战4：冲突解决	0:20:20
git入门和实战5：提交更改	0:12:15
git入门和实战6：远程版本库	0:13:26
git入门和实战7：内核开发和实战	0:15:52
git入门和实战8：实战rebase到最新Linux内核代码	0:18:07
git入门和实战9：给内核发补丁	0:13:57

第2季旗舰篇课程目录	
课程名称	时长
进程管理	
进程管理1基本概念	0:52:16
进程管理2进程创建	0:53:24
进程管理3进程调度	0:54:51
进程管理4多核调度	0:49:38
中断管理	
中断管理1基本概念	1:04:27
中断管理2中断处理part1	0:46:28
中断管理2中断处理part2	0:10:19
中断管理3下半部机制	0:55:57
中断管理4面试题目	1:13:57
锁机制	
锁机制入门1基本概念	0:56:16
锁机制入门2-Linux常用的锁	0:54:01



实战死机专题课程目录	
课程名称	时长
上集x86_64	
实战死机专题（上集）part1-kdump+crash介绍	0:30:09
实战死机专题（上集）part2-crash命令详解	0:28:15
实战死机专题（上集）part3-实战lab1	0:12:38
实战死机专题（上集）part4-实战lab2	0:11:03
实战死机专题（上集）part4-实战lab3	0:06:48
实战死机专题（上集）part4-实战lab4	0:15:28
实战死机专题（上集）part4-实战lab5	0:12:21
实战死机专题（上集）part4-实战lab6	0:24:07
实战死机专题（上集）part4-实战lab7	0:59:34
下集arm64	
实战死机专题(下集)part1	0:13:19
实战死机专题(下集)part2	0:20:47
实战死机专题(下集)part3	0:11:22
实战死机专题(下集)part4	0:33:01

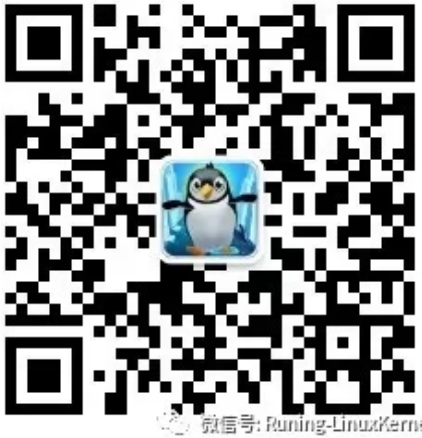
全程约5小时高清，140多页ppt，8大实验，基于x86_64的**Centos 7.6**和**arm64**，提供全套实验素材和环境。全面介绍kdump+crash在死机黑屏方面的实战应用，全部案例源自线上云服务器和嵌入式产品开发实际案例！

微店 奔跑吧Linux内核

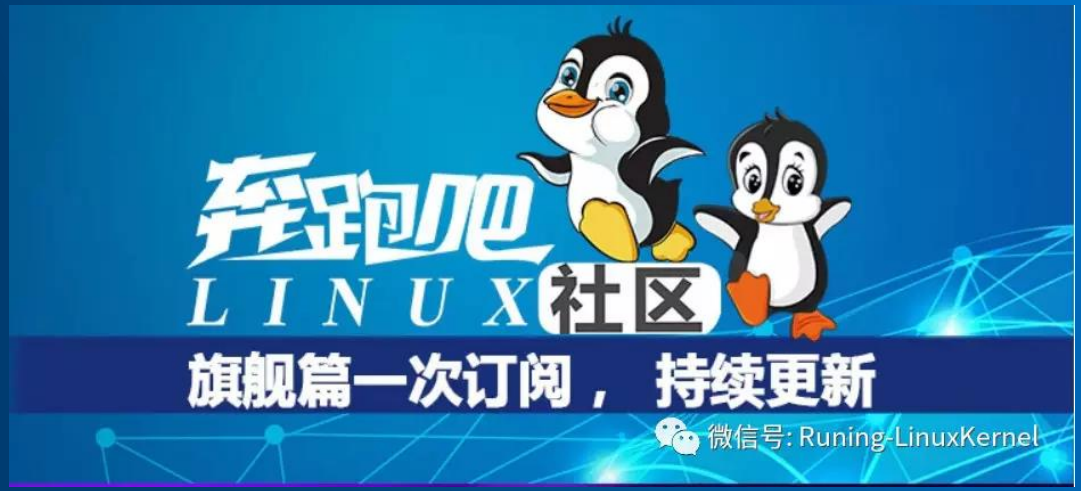
扫码识别

微店二维码

淘宝店二维码



《奔跑吧Linux内核 * 入门篇》相关的免费视频，或者更多更精彩更in的内容，请关注奔跑吧Linux社区微信公众号


奔跑吧
LINUX社区
旗舰篇一次订阅， 持续更新
微信号: Runing-LinuxKernel