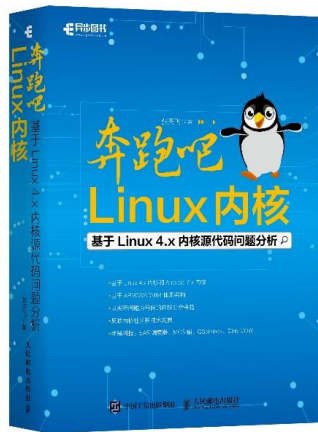
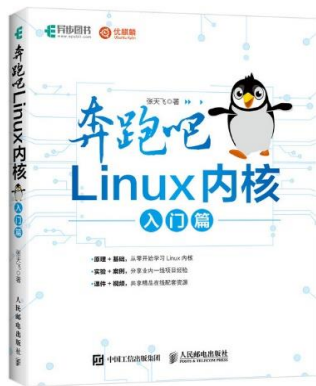


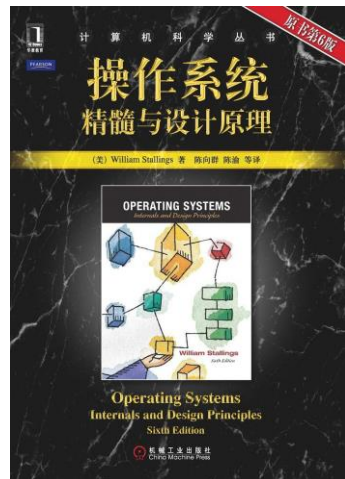
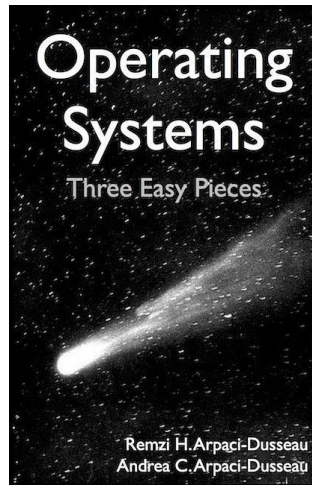
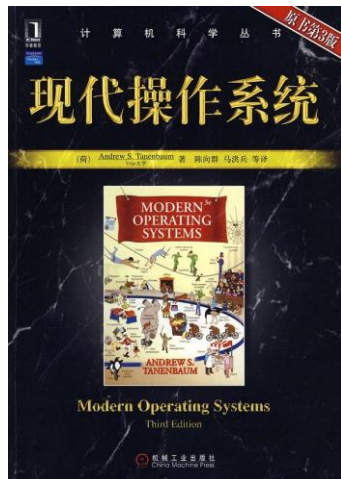
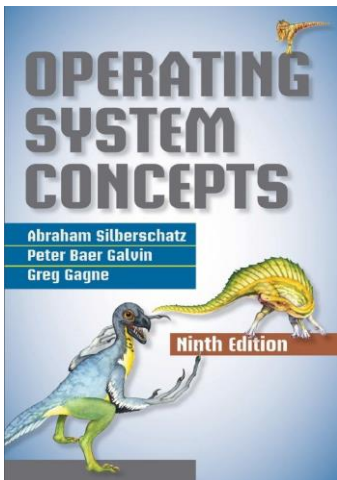
操作系统课程实验新尝试和探索

关于我

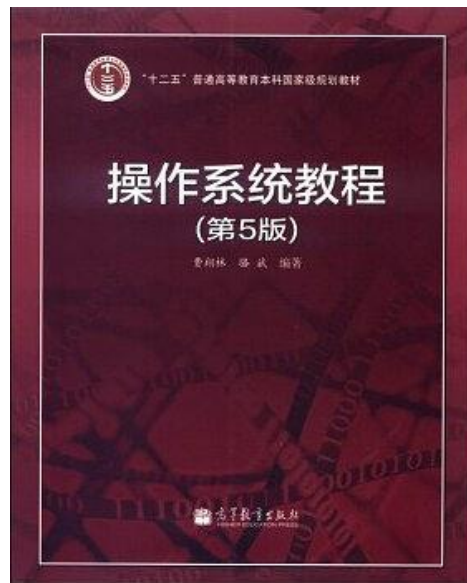
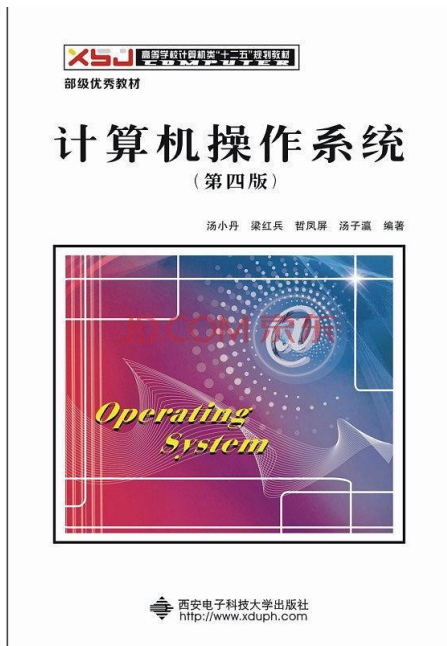
- 网名：笨叔
- 从事10几年Linux驱动和内核相关工作
- 2017年出版《奔跑吧Linux内核》
- 2019年出版《奔跑吧Linux内核 * 入门篇》
- 2019年公开小册子《Linux实验指导手册》



国外经典操作系统教材



国内经典操作系统教材



操作系统课程实验

- 大部分985高校的操作系统课程采用xv6作为实验素材 - 基于MIT 6.828课程
- 2019年的6.828课程 实验采用RISC-V的xv6
- <https://pdos.csail.mit.edu/6.828/2019/index.html>

The screenshot shows the MIT 6.S081 course website. The navigation bar includes links for "6.S081: Operating System Engineering", "Schedule", "Class", "Labs", "xv6", "References", and "Piazza". The "Labs" menu is open, listing various lab topics such as "Tools", "Lab Utilities", "Lab Shell", "Lab Allocator", "Lab Lazy allocation", "Lab Copy on-write", "Lab Uthread and alarm", "Lab Lock", "Lab File system", "Lab mmap", and "Lab network driver". Below the navigation bar, the page content includes a section for "Selection of Operating System" with a link to the "6.828 schedule", a "UNIX" section with links to "Youtube Unix intro", "The UNIX Time-Sharing System" by Dennis M. Ritchie, "The Evolution of the Unix Time-sharing System", and "The C programming language (second edition)", and a "RISC-V Emulation" section with a link to "QEMU - A fast and popular RISC-V platform and CPU emulator" and its "User manual".

6.S081: Operating System Engineering Schedule Class ▾ Labs ▾ xv6 ▾ References Piazza

Tools
Lab Utilities
Lab Shell
Lab Allocator
Lab Lazy allocation
Lab Copy on-write
Lab Uthread and alarm
Lab Lock
Lab File system
Lab mmap
Lab network driver

Selection of Operating System
Available on [the 6.828 schedule](#).

UNIX

- [Youtube Unix intro](#)
- [The UNIX Time-Sharing System](#), Dennis M. Ritchie
- [The Evolution of the Unix Time-sharing System](#), Dennis M. Ritchie
- [The C programming language \(second edition\)](#) by K. R. Kernighan and D. M. Ritchie, Prentice-Hall, Inc., 1988. ISBN 0-13-110362-8, 1998.

RISC-V Emulation

- [QEMU](#) - A fast and popular RISC-V platform and CPU emulator.
 - [User manual](#)

materials

em Technical Journal 57, number 6, part 2 (July 1981)

Hall, Inc., 1988. ISBN 0-13-110362-8, 1998.

xv6实验优缺点

➤ 优点:

- ✓ 搭建了一个小os，循序渐进地引导学生实现一些核心的功能
- ✓ 对体系结构（比如RSIC-V, X86）会有深入理解
- ✓ 对操作系统基本原理和实现有更深入理解

➤ 缺点:

- ✓ 对初学者难度比较大，比如cow实验
- ✓ 对操作系统不感兴趣的同学可以直接放弃了
- ✓ 有些学生从github直接拷贝和粘贴答案

xv6的不足

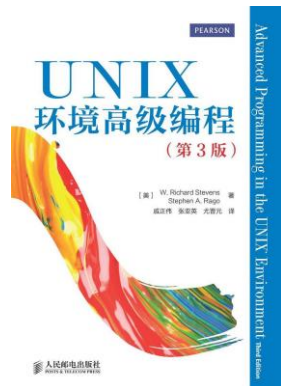
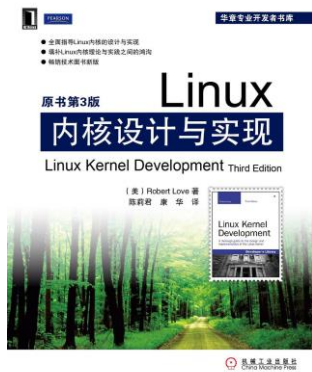
- 基于教学的小型os，类似Linux 0.11
- 离现在工业界使用的操作系统成熟度还差很远
- 离实际工业界需求有点远
- 不能立马学以致用，到企业之后还要重新学习

个人建议

- 对于学有余力或者对OS方向感兴趣的同学，独立完成xv6实验后，可以选择一个工业界主流的操作系统来研究：比如开源的Linux内核
- 对于没有采用xv6实验的高校，也可以直接采用Linux为实验对象

本科Linux内核和应用编程课程（选修）

- 部分高校本科有Linux内核和应用编程课程（西邮、上交大等）
- 以介绍Linux内核基本框架和基本概念为主要内容
- 介绍Linux应用编程接口



高级操作系统课程（某985高校研究生课程）

高级操作系统主要教学计划：（八周，每周4节课，32学时，2学分）

1. 操作系统相关理论，1周；
2. Linux操作系统内核结构、进程管理、存储管理、中断、系统调用、设备驱动，2周；
3. 虚拟化与容器，分布式系统，1周；
4. 第5到8周，学生分组做报告，主要涉及：Linux内核近2年新增模块/功能分析，嵌入式开源OS分析，操作系统前沿技术研究（针对OSDI、SOSP、FAST、USENIX Security等国际知名OS会议上的新技术动向开展研究）。

采用Linux作为实验对象的好处

➤ 优点:

- ✓ 学以致用，到企业里立马上手
- ✓ 学习最新最先进的操作系统理念和技术，比如EAS调度器，NUMA，异构计算等
- ✓ 学习开源编程规范、理念、文化

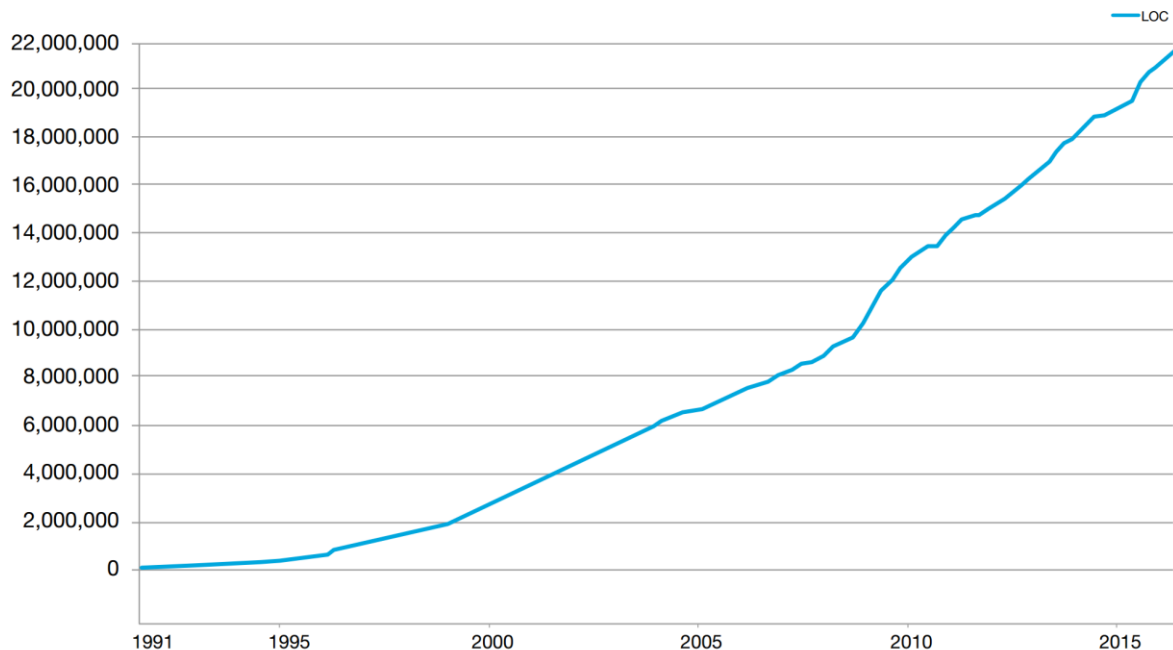
➤ 缺点:

- ✓ Linux内核代码太庞大，学习曲线骤增
- ✓ Linux内核更新太频繁
- ✓ **没有很好的入门教程和实验素材**

Linux内核行数的变化

- Linux内核从1991年的1w行代码发展到超过200w行代码

Total Lines of Code in the Linux Kernel



搭建一个开放的实验平台

- 基于QEMU + Linux 4.x (Linux 5.x)
- 支持x86_64、ARM64、RISC-V等主流体系结构
- 基于busybox小文件系统或者Debian Rootfs文件系统
- “00” 编译Linux内核
- Eclipse + GDB 图形化调试内核
- 70多个实验案例，部分实验案例源自实际项目
- Git仓库：https://github.com/figozhang/runninglinuxkernel_4.0
- 400多页实验指导手册免费下载
- 提供统一的实验平台：vmware镜像

我们想做的事情…

- 免费和开放的平台
- 入门实验 和免费的实验指导手册（目前提供70多个入门实验）
- 持续不断增加有趣的实验和修订实验指导手册
- 从企业中实际项目中抽闲出有趣的实验（比如新增的特工队实验）
- 从广大一线工程师中收集有趣的实验
- …

特工队实验例子

7.9 实验 9：特工队：修改敌军计算机的系统调用（新增）

1. 实验目的

- 通过本实验可以深刻理解操作系统中的页表是怎么构建的。
- 如何去遍历页表。
- 如何修改页表。
- 深刻理解各级页表中页表项的各个字段是什么含义。
- 系统调用在内核是如何被调用的。
- 当编写的驱动发生 crash 和 panic 时，如何去 debug?

注意：本实验有相当难度，学有余力的同学可以尝试做本实验。

2. 实验要求

假设你是一名安全特工，正在执行一项秘密任务，这项秘密任务就是要深入到敌军的作战指挥中心的计算机里安装一个窃听的程序，简单来说就是把计算机的系统调用动态替换掉。假设你的同事已经帮你把敌军计算机的 root 密码给破解了，接下来就看你如何动态修改系统调用了。**注意：编写的内核模块不能让敌军的计算机重启、crash/panic，否则就暴露行踪，秘密行动失败。**

1) 编写一个内核模块。

实验环境：ARM64 体系结构，Linux 5.0 内核。

要求替换系统调用表 (sys_call_table) 中某一项系统调用，替换成自己编写的系统调用处理函数 (例如：my_new_syscall())，在新的系统调用函数中打印一句“hello, I have hacked this syscall”，然后再调用回原来的系统调用处理函数。

比如以 ioctl 系统调用为例，它在系统调用表中的编号就是 __NR_ioctl。那么需要修改系统调用表 sys_call_table[__NR_ioctl] 的指向，让其指向 my_new_syscall() 函数，然后在 my_new_syscall() 函数中打印一句话，调用原来的 sys_call_table[__NR_ioctl] 指向的处理函数。

2) 卸载模块时候把系统系统表恢复原样。

3) 用 clone 系统调用来验证你的驱动，clone 系统调用号是 __NR_clone。

提高初学者兴趣

➤ 源自实际项目的案例抽象

➤ 历经4次Linux宕机

➤ 涉及Linux多个核心模块

- ARM64体系结构
- ARM64内存管理
- Linux内核内存管理
- 系统调用
- Linux宕机分析

```
root@benshushu:/mnt# insmod testsyscall_issue.ko
[ 191.649281] testsyscall_issue: loading out-of-tree module taints kernel.
[ 191.679779] testsyscall_issue: module verification failed: signature and/or required key missing - tainting kernel
[ 191.817883] Found the sys_call table at ffff000011732b20.
[ 191.964776] replace system call ...
[ 191.965620] walk_pagetable get pte=0x7FFFC003
[ 191.965807] mkwrite pte=0x800007FFFC403
[ 191.967559] walk_pagetable: pte=0x800007fffc403
[ 191.968034] got sys_call_table[29] at 0.
[ 191.984000] Unable to handle kernel write to read-only memory at virtual address ffff000011732c08
[ 191.984900] Mem abort info:
[ 191.985070]   ESR = 0x96000004f
[ 191.985330]   Exception class = DABT (current EL), IL = 32 bits
[ 191.985566]   SET = 0, FnV = 0
[ 191.985720]   EA = 0, S1PTW = 0
[ 191.985911] Data abort info:
[ 191.986069]   ISV = 0, ISS = 0x00000004f
[ 191.987128]   CM = 0, WnR = 1
[ 191.988815] swapper pgtable: 4k pages, 48-bit VAs, pgdp = (____ptrval____)
[ 191.989668] [ffff000011732c08] pgd=000000007ffff003, pud=000000007fffe003, pmd=000800007fffc403, pte=00e000004173279
3
[ 191.994840] Internal error: Oops: 9600004f [#1] SMP
[ 191.996028] Modules linked in: testsyscall_issue(OE+)
[ 191.998209] CPU: 1 PID: 565 Comm: insmod Kdump: loaded Tainted: G          OE      5.0.0 #1
[ 191.999715] Hardware name: linux,dummy-virt (DT)
[ 192.000771] pstate: 60000005 (nZCv daif -PAN -UA0)
[ 192.003707] pc : hack_syscall_init+0x49c/0x1000 [testsyscall_issue]
[ 192.004347] lr : hack_syscall_init+0x438/0x1000 [testsyscall_issue]
[ 192.004973] sp : ffff80002451f5b0
[ 192.005440] x29: ffff80002451f5b0 x28: ffff800024549c00
[ 192.006139] x27: 0000000000000000 x26: 0000000000000000
[ 192.007305] x25: 0000000056000000 x24: 0000000000000015
[ 192.008007] x23: 0000000040001000 x22: 0000ffff88ca4d44
```


谢谢观赏

Backup

第1章 LINUX 系统入门

- [1.1 实验 1: 在虚拟机中安装优麒麟 Linux 18.04 系统](#)
- [1.2 实验 2: 给优麒麟 Linux 系统更换心脏](#)
- [1.3 实验 3: 使用 O0 编译的内核 - runninglinuxkernel](#)
- [1.4 实验 4: 如何编译和运行一个 ARM Linux 内核](#)
- [1.5 实验 5: 运行 Debian+ARM32 系统 \(新增\)](#)
- [1.6 实验 6: 运行 Debian+ARM64 系统 \(新增\)](#)
- [1.7 实验 7: 运行 Debian+X86_64 系统 \(新增\)](#)
- [1.8 实验 8: 手把手制作一个 Debian rootfs 系统 \(新增\)](#)
- [1.9 实验 9: 配置 QEMU 虚拟机的桥接网络 \(新增\)](#)
- [1.10 实验 10: 动手 DIY 一个 RISC-V 的 Debian 系统 \(新增\)](#)

第2章 LINUX 内核基础知识

- [2.1 实验 1: GCC 编译](#)
- [2.2 实验 2: 内核链表](#)
- [2.3 实验 3: 红黑树](#)
- [2.4 实验 4: 使用 Vim 工具](#)
- [2.5 实验 5: 把 Vim 打成一个强大的 IDE 编辑工具](#)
- [2.6 实验 6: 建立一个 git 本地仓库](#)
- [2.7 实验 7: 解决合并分支冲突](#)
- [2.8 实验 8: 利用 git 来管理 Linux 内核开发](#)
- [2.9 实验 9: 利用 git 来管理项目代码](#)

第3章 内核编译和调试

- [3.1 使用O0优化等级编译内核的好处](#)
- [3.2 实验1：通过QEMU调试ARM Linux内核](#)
- [3.3 实验2：通过QEMU调试ARMv8的Linux内核](#)
- [3.4 实验3：通过Eclipse+QEMU单步调试内核](#)
- [3.5 实验4：在QEMU中添加文件系统的支持](#)
- [3.6 实验5：使用DS-5单步调试arm64内核](#)

第4章 内核模块

- [4.1 实验1：编写一个简单的内核模块](#)
- [4.2 实验2：向内核模块传递参数](#)
- [4.3 实验3：在模块之间导出符号](#)
- [4.4 实验4：在优麒麟系统中编译内核模块（新增）](#)

第5章 简单的字符设备驱动

- [5.1 实验1：从一个简单的字符设备开始](#)
- [5.2 实验2：使用misc机制来创建设备](#)
- [5.3 实验3：为虚拟设备编写驱动](#)
- [5.4 实验4：使用KFIFO改进设备驱动](#)
- [5.5 实验5：把虚拟设备驱动改成非阻塞模式](#)
- [5.6 实验6：把虚拟设备驱动改成阻塞模式](#)
- [5.7 实验7：向虚拟设备中添加I/O多路复用支持](#)
- [5.8 实验8：为什么不能唤醒读写进程](#)
- [5.9 实验9：向虚拟设备中添加异步通知](#)

第6章 系统调用

- [6.1 实验1：在ARM32机器上新增一个系统调用](#)
- [6.2 实验2：在优麒麟Linux机器上新增一个系统调用](#)

第7章 内存管理

- [7.1 实验1：查看系统内存信息](#)
- [7.2 实验2：获取系统的物理内存信息](#)
- [7.3 实验3：分配内存](#)
- [7.4 实验4：slab](#)
- [7.5 实验5：VMA](#)
- [7.6 实验6：mmap](#)
- [7.7 实验7：映射用户内存](#)
- [7.8 实验8：OOM](#)
- [7.9 实验9：特工队：动态修改计算机的系统调用（新增）](#)
- [7.10 小结（新增）](#)

第8章 进程管理

- [8.1 实验1：fork和clone](#)
- [8.2 实验2：内核线程](#)
- [8.3 实验3：后台守护进程](#)
- [8.4 实验4：进程权限](#)
- [8.5 实验5：设置优先级](#)
- [8.6 实验6：per-cpu变量](#)

第9章 同步管理

- [9.1 实验 1: 自旋锁](#)
- [9.2 实验 2: 互斥锁](#)
- [9.3 实验 3: RCU](#)

第10章 中断管理

- [10.1 实验 1: tasklet](#)
- [10.2 实验 2: 工作队列](#)
- [10.3 实验 3: 定时器和内核线程](#)

第12章 开源社区

- [12.1 实验 1: 使用 cppcheck 检查代码](#)
- [12.2 实验 2: 提交第一个 Linux 内核补丁](#)
- [12.3 实验 3: 管理和提交多个补丁组成的补丁集](#)
- [12.4 实验 4: 在 github 中创建和管理一个开源项目](#)

第13章 块设备和文件系统（新增）

- [13.1 实验 1: 块设备实验](#)
- [13.2 实验 2: 动手写一个简单文件系统](#)

第11章 调试和性能优化

- [11.1 实验 1: printk](#)
- [11.2 实验 2: 动态输出](#)
- [11.3 实验 3: procfs](#)
- [11.4 实验 4: sysfs](#)
- [11.5 实验 5: debugfs](#)
- [11.6 实验 6: 使用 ftrace](#)
- [11.7 实验 7: 添加一个新的跟踪点](#)
- [11.8 实验 8: 使用示踪标志](#)
- [11.9 实验 9: 使用 kernelshark 来分析数据](#)
- [11.10 实验 10: 分析 oops 错误](#)
- [11.11 实验 11: 使用 perf 工具来进行性能分析](#)
- [11.12 实验 12: 采集 perf 数据生成火焰图](#)
- [11.13 实验 13: 使用 slub_debug 检查内存泄漏](#)
- [11.14 实验 14: 使用 kmemleak 检查内存泄漏](#)
- [11.15 实验 15: 使用 kasan 检查内存泄漏](#)
- [11.16 实验 16: 使用 valgrind 检查内存泄漏](#)
- [11.17 实验 17: 使用 lkp-tests 工具进行性能测试](#)
- [11.18 实验 18: kdump 死机实战 1: 运行和配置 kdump（新增）](#)
- [11.19 实验 19: kdump 死机实战 2: 访问已经删除的链表（新增）](#)
- [11.20 实验 20: kdump 死机实战 3: 内存 bug（新增）](#)
- [11.21 实验 21: kdump 死机实战 4: 死战驱动死机问题（新增）](#)
- [11.22 实验 22: kdump 死机实战 5: 在 Centos 7.6 上安装和配置 kdump（新增）](#)
- [11.23 实验 23: kdump 死机实战 6: 实战 arm64 死机（新增）](#)
- [11.24 死机问题进阶](#)

实验指导手册下载



登录“奔跑吧linux社区”微信公众号：

- 输入“实验指导手册”下载实验指导手册。
- 输入“全套资料”下载全套配套实验平台和资料。
- 输入“免费视频”下载免费视频。

Linux内核架构概貌

