奔跑吧LINUX内核* 入门篇

第六章系统调用

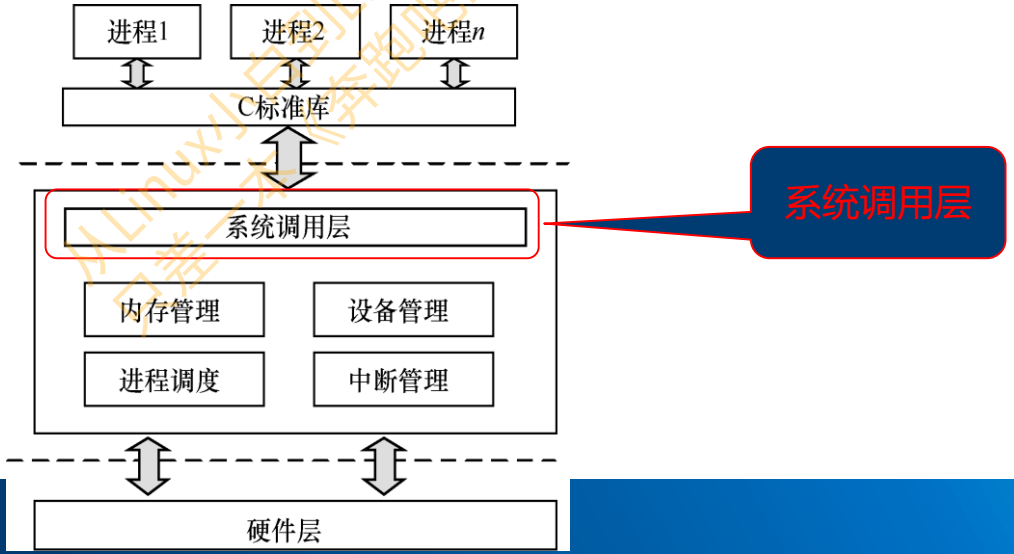
笨叔叔

目 录

Linux系统调用

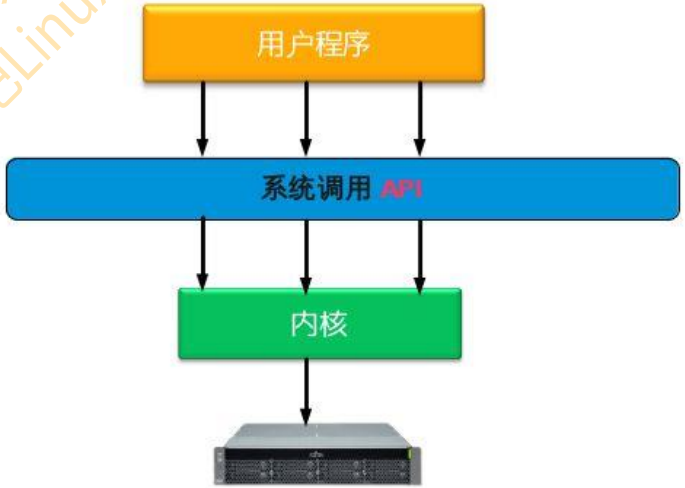
系统调用层

- 内核空间和用户空间
  - ✓ 处理器的运行模式通常分成两个空间：一个是内核空间，另一个是用户空间。大部分的应用程序运行在用户空间，而内核和设备驱动运行在内核空间。
- 系统调用层
  - ✓ 由操作系统实现提供的所有系统调用所构成的集合即程序接口或应用编程接口(Application Programming Interface，API)。是应用程序同系统之间的接口



系统调用层

- 系统调用层作用：
  - 为用户空间程序提供一层硬件抽象接口。
  - 保证系统稳定和安全。
  - 可移植性。



系统调用和POSIX标准

- 应用编程接口（API）：
  - ✓ 应用程序调用用户空间实现的应用编程接口来编程，而不是直接调用系统调用。
  - ✓ 一个API接口函数可以由一个系统调用实现，也可以由多个系统调用来实现，甚至完全不使用任何系统调用。因此，一个API接口没有必要对应一个特定的系统调用

- POSIX（Portable Operating System Interface of UNIX）标准
  - ✓ POSIX的诞生和UNIX的发展密不可分。UNIX系统诞生于20世纪70年代的贝尔实验室，很多商业厂商基于UNIX发展自己的UNIX系统，但是标准不统一。后来IEEE制定了POSIX标准
  - ✓ POSIX标准针对的是API而不是系统调用
  - ✓ Linux以libc来实现POSIX标准

系统调用表

- Linux系统为每一个系统调用赋予一个系统调用号。当应用程序执行一个系统调用时，应用程序就可以知道执行和调用到哪个系统调用了，从而不会造成混乱。
- 对于ARM32系统来说，其系统调用号定义在arch/arm/include/uapi/asm/unistd.h头文件中。

```
24 /*
25  * This file contains the system call numbers.
26  */
27
28 #define __NR_restart_syscall          (__NR_SYSCALL_BASE+  0)
29 #define __NR_exit                     (__NR_SYSCALL_BASE+  1)
30 #define __NR_fork                     (__NR_SYSCALL_BASE+  2)
31 #define __NR_read                     (__NR_SYSCALL_BASE+  3)
32 #define __NR_write                    (__NR_SYSCALL_BASE+  4)
33 #define __NR_open                     (__NR_SYSCALL_BASE+  5)
34 #define __NR_close                    (__NR_SYSCALL_BASE+  6)
35                                       /* 7 was sys_waitpid */
36 #define __NR_creat                    (__NR_SYSCALL_BASE+  8)
37 #define __NR_link                     (__NR_SYSCALL_BASE+  9)
38 #define __NR_unlink                   (__NR_SYSCALL_BASE+ 10)
39 #define __NR_execve                   (__NR_SYSCALL_BASE+ 11)
40 #define __NR_chdir                    (__NR_SYSCALL_BASE+ 12)
41 #define __NR_time                     (__NR_SYSCALL_BASE+ 13)
42 #define __NR_mknod                    (__NR_SYSCALL_BASE+ 14)
43 #define __NR_chmod                    (__NR_SYSCALL_BASE+ 15)
44 #define __NR_lchown                   (__NR_SYSCALL_BASE+ 16)
```

实验1：在ARM32机器上新增一个系统调用

- 实验目的
  - ✓ 通过新增一个系统调用，理解系统调用的实现过程。
- 实验步骤
  - ✓ 1）在ARM Vexpress平台上新增一个系统调用，该系统调用不用传递任何参数，在该系统调用里输出当前进程的PID和UID值。
  - ✓ 2）编写一个应用程序来调用这个新增的系统调用。

实验2：在优麒麟Linux机器上新增一个系统调用

- 实验目的
  - ✓ 通过新增一个系统调用，理解系统调用的实现过程。
- 实验步骤
  - ✓ 1）在优麒麟Linux平台上新增一个系统调用，该系统调用不用传递任何参数，在该系统调用里输出当前进程的PID和UID值。该实验的目的是让读者学会如何在x86_64里添加一个系统调用，并比较和ARM32系统的区别。
  - ✓ 2）编写一个应用程序来调用这个新增的系统调用。

BACKUP

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