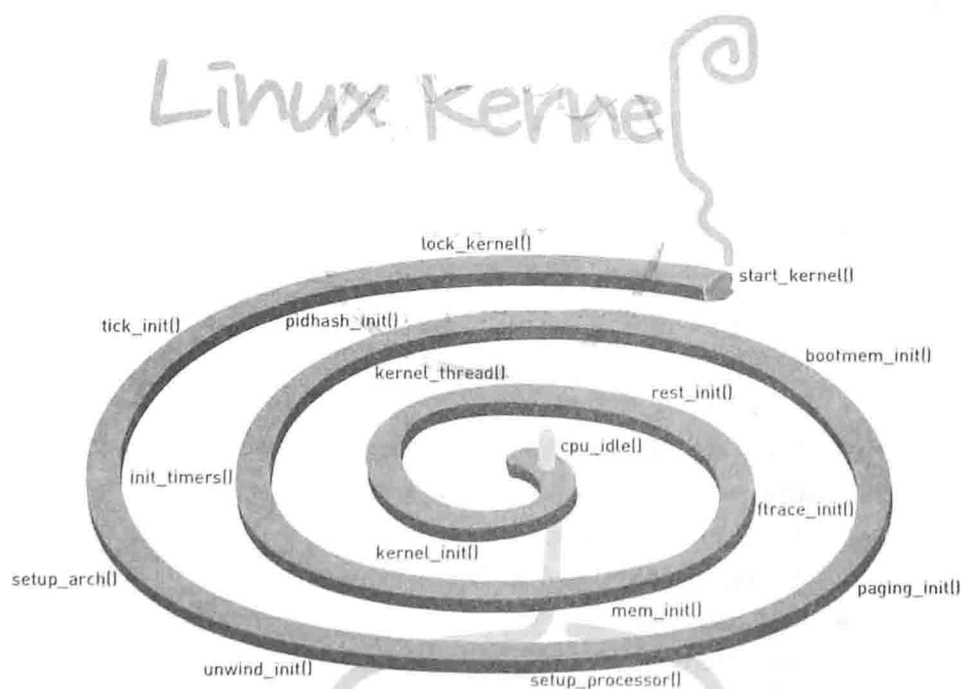


TURING

图灵程序设计丛书

ARM Linux 内核源码剖析

【韩】尹锡训等 著 崔范松 译



人民邮电出版社
北京

图书在版编目 (CIP) 数据

ARM Linux内核源码剖析 / (韩) 尹锡训等著 ; 崔范松译. -- 北京 : 人民邮电出版社, 2014.7

(图灵程序设计丛书)

ISBN 978-7-115-35910-0

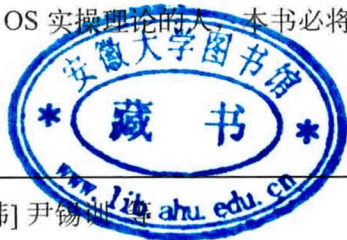
I. ①A… II. ①尹… ②崔… III. ①微处理器—系统设计②Linux操作系统—系统设计 IV. ①TP332
②TP316.89

中国版本图书馆CIP数据核字(2014)第117494号

内 容 提 要

本书是多位作者在3年Linux内核分析经验和庞大资料基础上写成的,收录了其他同类书未曾讲解的内容并进行逐行分析,一扫当前市场中其他理论书带给读者的郁闷。书中详细的代码分析与大量插图能够使读者对Linux内核及ARM获得正确认识,自然而然习得如何有效分析定期发布的Linux内核。

本书适合想从Linux内核启动开始透彻分析全部启动过程的读者,因Linux代码量庞大而束手无策的人、想要了解Linux实际运行过程的人、渴求OS实操理论的人,本书必将成为他们不可或缺的参考书。



-
- ◆ 著 [韩] 尹锡训等
 - 译 崔范松
 - 责任编辑 傅志红
 - 执行编辑 陈曦
 - 责任印制 焦志炜
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京鑫正大印刷有限公司印刷
 - ◆ 开本: 800×1000 1/16
印张: 33.25 插页: 2
字数: 786千字 2014年7月第1版
印数: 1-4 000册 2014年7月北京第1次印刷
- 著作权合同登记号 图字: 01-2013-8469号

定价: 99.00元

读者服务热线: (010)51095186转600 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京崇工商广字第 0021 号

审阅者评语

整整3年的写作……

事实上，这本书的正式编写时间大约是1年。但我说用了3年，原因在于编写团队整整用了2年时间分析Linux代码，在此基础上，用1年完成了整个写作过程。为了这本书，他们在3年内不分昼夜地专注于Linux内核这一庞大领域，细心钻研，提高实力，并使辛劳努力的成果最终面世！

正是由于了解这些情况，所以，我收到原稿进行审阅时，比作者们更感慨万千。看着这本凝聚着编写团队辛勤汗水的书，我感到无比自豪。

简言之，这本书既是属于广大读者的书，更是作者们自己的书。因为书中是他们用2年时间分析Linux内核并不断提高实力的亲身经历，所以比现有的其他书籍水平更高。而且，编写团队已经完全掌握了书中内容，应该可以解答读者们的大部分疑惑。

这本书如同一把锋利的手术刀，大家读了就能感到，书中的每个部分都充满了鲜活的细节。这恰恰反映出2年间诸位作者一直保持着自己的实力。另外，由实力雄厚的编写团队逐一解开该主题的所有谜团，这使该书通顺流畅。

鉴于以上这些原因，我认为这本佳作比市面上出现的任何一本内核书都更出色。

我为这3年来付出最大努力的作者们喝彩！同时也为能够阅读此书进行学习的读者们感到幸运。因为我很久以前接触过Linux内核，但当时还没有这本书作参考，所以经历了无数失误。我向希望学习Linux内核的各位读者强烈推荐这本辅导书。

白昌佑

曾在三星电子、三星 SDS、(株) Nusco 公司主持开发各类 RTOS、在三星综合技术院开发编译器，在 (株) Nusco 开发调试器和管理程序及各种系统 S/W。目前在 (株) Nusco 担任 CEO、软件开发团队指导。10 年来一直负责管理离线系统 S/W 学习小组 <http://www.iamroot.org>。

前 言

分析 Linux 内核并将其编撰成册，这对我们来说是一项巨大的挑战。通过 iamroot.org，我们在 2009 年 5 月举办了第一次以分析 ARM Linux 内核为目的的聚会。而且，完成第一次分析前的 2 年时间里，我们每周都是单休。身为工程师，能够亲自分析以前在书上接触过的操作系统代码，这让我们感到无比喜悦，可实际的分析过程却更要求我们耐心。

我们通过数年来对内核源代码分析的学习，依据整理好的 iamroot 的指导进行源代码分析。这本书主要讲述 ARM 使用的 Linux 内核源代码，逐行分析源代码和内核的实际工作内容，主要面向希望深层次了解内核内部机制的广大读者。逐行分析内核源代码主要是分析接通电源后如何调用内核的起始代码，以及调用起始代码内各种函数并最终显示命令行提示符的一系列过程。从这一点上讲，本书结构与其他 Linux 内核教程有所不同，涵盖了其他教材上没有涉及的内容。Linux 内核的基本概念可通过许多资料获得，但不操作内核源代码，就无法满足工程师们了解操作系统实现方法的求知欲。这正是本书最大的价值所在。

在分析初期，甚至到第一次分析结束时，写书这件事都不在我们的计划之中。大家开始分析内核时还兴致勃勃，逐步深入后才明白，我们的能力和知识是有限的，而编书这一重任更是远远超出了自己的能力范围。我们明知自身还有很多不足之处却依然坚持编写此书，就是希望将分析过程中遇到的困惑和辛辛苦苦理解到的内容与广大喜欢 Linux 的工程师、对内核感兴趣的读者分享。因为我们相信，正是有了许多工程师对 Linux 做出的贡献，才能使大家学习 Linux 内核代码。

我们通过本书与大家分享经验并不是因为自己掌握的知识多，也不因为这本书涉及的内容完美无缺。就像腼腆的林纳斯（Linus）1991 年用邮件与世界分享自己的内核代码那样，我们也只希望自己整理的内容能够给一些人带来帮助，也希望另一些人以这本书为基础，写出更多、更好的著作。所以，其实是我们在大家的帮助下更加深入地理解 Linux 内核。希望这本书能为想要了解 Linux 内核的广大读者带来些许帮助。

全体作者

2012 年 8 月

目 录

第一部分 ARM Linux 内核——分析内核前需要做的准备

第 1 章 内核介绍及 2.6 版和 3.2 版 之间的差异	2
1.1 内核的诞生、作用以及内部结构	2
1.1.1 Linus 创造的 Linux	2
1.1.2 由多种子系统集成运行的单内核	3
1.1.3 全世界最著名的通用操作系统	5
1.2 内核 2.6 版和 3.2 版之间的差异	5
第 2 章 内核构建系统	8
2.1 内核初始化	8
2.2 内核配置	9
2.3 内核构建	11
2.4 内核安装	17
第 3 章 了解 ARM 处理器	19
3.1 处理器概要和特征	19
3.2 处理器架构与核心	19
3.3 处理器命名规则	21
3.4 处理器内部结构	21
3.5 处理器模式和寄存器	23
3.6 处理器异常	25
3.7 硬件扩展功能	26
3.7.1 缓存	26
3.7.2 内存管理装置	26
3.7.3 协处理器	26
第 4 章 构建分析环境	28
4.1 下载并安装 Linux 源内核	28

4.1.1	下载源内核	28
4.1.2	安装源内核	30
4.2	安装 ctags+ctags	31
4.2.1	用 ctags 制作源代码标签	31
4.2.2	制作 cscope 标签数据库	33
4.3	vim 插件下载及环境设置	34
4.3.1	下载 vim 插件	34
4.3.2	vim+plugin 的环境结构	37
4.3.3	vim 环境设置	38
4.4	查看源码分析环境工具	40

第二部分 内核的启动——start_kernel 调用方法

第 5 章	准备解压内核	48
5.1	进入启动加载后结束首个启动——start 标签	49
5.2	BSS 系统域初始化——not_relocated 标签	50
5.3	激活缓存——cache_on 标签	53
5.4	页目录项初始化——__setup_mmu 标签	56
5.5	指令缓存激活及缓存策略适用——__common_mmu_cache_on 标签	58
第 6 章	从压缩的内核 zImage 还原内核映像	60
6.1	解压内核并避免覆写——wont_overwrite、decompress_kernel 标签	61
6.2	调用已解压内核——call_kernel 标签	62
6.3	缓存清理及清除——cache_clean_flush 标签	62
6.4	缓存禁用——cache_off 标签	64
第 7 章	调用 start_kernel()	65
7.1	初始化指向——stext 标签	65
7.2	处理器信息搜寻——__lookup_processor_type	69
7.2.1	__lookup_processor_type 标签	69
7.2.2	__proc_info_begin 和 __proc_info_end 中保存的信息	71
7.2.3	在 MMU 禁用状态下将虚拟地址转换为物理地址	73
7.2.4	查找 proc_info_list 结构体并比较处理器信息	74
7.3	搜寻我的机型——__lookup_machine_type	75
7.3.1	__lookup_machine_type 标签	75
7.3.2	保存在 __arch_info_begin 和 __arch_info_end 中的 machine_desc 信息及访问路径	76
7.3.3	查找 machine_desc 结构体并比较机器信息	77
7.4	源自启动加载项的 atags——__vet_atags 标签	78

7.5	对虚拟内存进行基础创建—— <code>_create_page_tables</code> 标签	81
7.6	设置核心 (core) —— <code>v6_setup</code> 标签	85
7.7	打开 MMU 并使用虚拟地址—— <code>_enable_mmu/_turn_mmu_on</code> 标签	86
7.8	跳转至 <code>start_kernel</code> —— <code>_mmap_switched</code> 标签	90

第三部分 内核的执行——内核的起始与结束位置

第 8 章	<code>start_setup_processor_id()</code> ~ <code>lock_kernel()</code>	94
8.1	<code>smp_setup_processor_id()</code> 、 <code>lockdep_init()</code> 、 <code>debug_objects_early_init()</code>	95
8.1.1	<code>smp_setup_processor_id()</code>	95
8.1.2	<code>lockdep_init()</code>	95
8.1.3	<code>debug_objects_early_init()</code>	96
8.2	栈溢出感应—— <code>_boot_init_stack_canary</code>	98
8.3	初始化提供进程集成方法的 cgroup—— <code>_cgroup_init_early()</code>	98
8.3.1	<code>cgroupfs_root</code> 和 cgroup 的关联初始化—— <code>init_cgroup_root()</code>	102
8.3.2	初始化子系统—— <code>cgroup_init_subsys()</code>	103
8.4	禁用 IRQ	104
8.5	<code>early_boot_irqs_off()</code> 、 <code>early_init_irq_lock_class()</code>	104
8.6	大内核锁—— <code>lock_kernel()</code>	106
第 9 章	注册针对时钟事件的处理器	111
9.1	函数的声明和定义—— <code>tick_init()</code>	111
9.2	注册处理事件的处理器—— <code>_clockevents_register_notifier()</code>	113
9.2.1	为 <code>clockevents_lock</code> 添加自旋锁	114
9.2.2	<code>clockevents_chain</code> 生成原理	115
9.2.3	在 <code>clockevents_chain</code> 中注册 <code>tick_notifier</code> 的方法	116
9.2.4	对 <code>clockevents_lock</code> 解除自旋锁的原理	117
第 10 章	在 CPU 位图中注册当前运行 CPU/初始化 HIGHMEM 管理	119
10.1	在包含热插拔信息的位图上添加执行 <code>init_task</code> 的 CPU—— <code>boot_cpu_init()</code>	119
10.2	管理高端内存—— <code>page_address_init()</code>	121
第 11 章	整体指向—— <code>setup_arch</code>	123
第 12 章	<code>unwind_init()</code> ~ <code>early_trap_init()</code>	126
12.1	栈回溯—— <code>unwind_init()</code>	126
12.2	求出包含机器信息的 <code>machine_desc</code> 结构体—— <code>setup_machine()</code>	126
12.3	处理 ATAG 信息—— <code>setup_arch()</code>	127
12.4	处理启动参数—— <code>parse_cmdline()</code>	129

12.5	构建源代码树——request_standard_resources()	131
12.6	初始化 cpu possible 位图——smp_init_cpus()	136
12.7	用栈指定各 ARM 异常模式——cpu_init()	137
12.8	初始化以处理异常——early_trap_init()	138
12.9	查看中断处理器函数	143
12.9.1	调用 IRQ 处理器——asm_do_IRQ()	147
12.9.2	返回中断之前——ret_to_user 标签	147
第 13 章 设置处理器——setup_processor()		150
13.1	查看 setup_processor()结构	150
13.2	查找 CPU ID——read_cpuid_id()	151
13.3	查找处理器信息——lookup_processor_type()	153
13.4	查找处理器结构信息——cpu_architecture()	153
13.5	查找处理器缓存类型_cacheid_init()	156
13.6	调用处理器初始化函数——cpu_proc_init()	160
第 14 章 准备内存分页——paging_init()		163
14.1	查看 paging_init()的整体结构	163
14.2	设置内存类型表——build_mem_type_table()	165
14.3	检验内存信息——sanity_check_meminfo()	166
14.4	准备页表——prepare_page_table()	168
14.4.1	prepare_page_table()	168
14.4.2	Linux 的分页结构	170
14.4.3	求出页目录项	170
14.4.4	pmd_clear()	172
14.5	设备区域映射准备——devicemaps_init()	174
14.6	准备使用高端内存——kmap_init()	177
14.7	初始化零页	178
14.7.1	分配内存——__alloc_bootmem_nopanic()	179
14.7.2	在指定节点使用 fallback 分配内存——alloc_bootmem_core	180
14.7.3	将虚拟地址变换为 page 结构体——virt_to_page	182
14.8	保持数据缓存一致性——flush_dcache_page()	182
第 15 章 在启动时初始化内存分配器		184
15.1	bootmem 函数流和数据结构	185
15.2	查看 bootmem_init()结构	188
15.3	查找虚拟内存盘位置——check_initrd()	189
15.4	将节点的 BANK 信息反映到页目录——bootmem_init_node()	191
15.4.1	map_memory_bank()	192
15.4.2	bootmem_bootmap_pages()	195

15.4.3	find_bootmap_pfn()	196
15.4.4	node_set_online()	197
15.4.5	NODE_DATA 宏	198
15.4.6	init_bootmem_node()	200
15.4.7	free_bootmem_node()	202
15.4.8	reserve_bootmem_node()	202
15.5	排除 0 号节点——reserve_node_zero()	203
15.6	排除虚拟内存盘节点——bootmem_reserve_initrd()	204
15.7	设置为无可分页——bootmem_free_node()	205
15.8	初始化 free_area 区域	207
15.8.1	free_area 结构体	207
15.8.2	free_area_init_node()	208
15.8.3	free_area_init_core()	209
15.8.4	init_currently_empty_zone()	211
15.8.5	memmap_init()	212
第 16 章 mm_init_owner()~preempt_disable()		217
16.1	设置内存拥有者——mm_init_owner()	217
16.2	保存命令行——setup_command_line()	218
16.3	初始化 per-cpu 数据——setup_per_cpu_areas()	219
16.4	求 CPU 个数——setup_nr_cpu_ids()	221
16.5	注册 SMP 上的启动进程——smp_prepare_boot_cpu()	222
16.6	初始化数据结构以使用调度程序——sched_init()	224
16.6.1	为集合调度中使用的 task_group 的 sched_entity 结构体和 runqueue 结构体分配内存	224
16.6.2	初始化 root_domain、rt_bandwidth、task_group 相关数据结构	227
16.6.3	初始化系统上所有可用 CPU 的就绪队列	229
16.6.4	初始化当前任务的调度相关值与注册针对负载均衡的中断处理器	230
16.7	允许内核抢占和阻止抢占——preempt_enable()/preempt_disable()	231
第 17 章 构建借用内存的后台		233
17.1	在 build_all_zonelists()中操作的一些数据结构	233
17.2	查看 build_all_zonelists()结构	235
17.3	决定 zone 的列表方式——set_zonelist_order()	236
17.4	构建备用列表和备用位图——__build_all_zonelists()	238
17.4.1	build_zonelists()	239
17.4.2	build_zonelist_in_node_order()	241
17.4.3	build_zonelists_in_zone_order()	243
17.4.4	build_thisnode_zonelists()	244
17.4.5	build_zonelists_cache()	244

17.5	输出备用列表信息—— <code>mminit_verify_zonelist()</code>	246
17.6	指定处理页分配请求的节点—— <code>cpuset_init_current_mems_allowed()</code>	246
17.7	求空页数—— <code>nr_free_pagecache_pages()</code>	247
17.8	页移动性.....	250
第 18 章	<code>page_alloc_init()~pidhash_init()</code>	253
18.1	处理用于热插拔 CPU 的页—— <code>page_alloc_init()</code>	253
18.2	处理 console 参数—— <code>parse_early_param()</code>	255
18.3	处理特殊参数—— <code>parse_args()</code>	257
18.4	确认中断处理是否激活—— <code>irqs_disable()</code>	260
18.5	内核异常列表定义—— <code>sort_main_extable()</code>	261
18.6	初始化 RCU 机制—— <code>rcu_init()</code>	263
18.7	准备使用 IRQ—— <code>early_irq_init()</code>	266
18.8	初始化中断—— <code>init_IRQ()</code>	269
18.9	构建迅速搜寻进程信息的结构—— <code>pidhash_init()</code>	271
第 19 章	<code>init_timers()~page_cgroup_init()</code>	273
19.1	初始化计时器—— <code>init_timers()</code>	274
19.1.1	<code>timers_cpu_notify()</code>	275
19.1.2	<code>register_cpu_notifier()</code>	275
19.1.3	<code>open_softirq()</code>	276
19.2	初始化高分辨率计时器—— <code>hrtimers_init()</code>	277
19.3	注册 <code>softirq</code> 的回调函数—— <code>softirq_init()</code>	280
19.4	设置 <code>xtime</code> —— <code>timekeeping_init()</code>	282
19.5	初始化硬件计时器—— <code>time_init()</code>	285
19.6	初始化时钟时间—— <code>sched_clock_init()</code>	286
19.7	激活 CPU 的中断处理—— <code>local_irq_enable()</code>	288
19.8	检测用作根文件系统的 <code>init</code> 虚拟内存盘.....	288
19.9	初始化以分配动态内存—— <code>vmalloc_init()</code>	289
19.10	预先初始化目录项和索引节点缓存—— <code>vfs_caches_init_early()</code>	290
19.11	初始化 <code>cpuset</code> 子系统—— <code>cpuset_init_early()</code>	293
19.12	初始化内存子系统—— <code>page_cgroup_init()</code>	294
第 20 章	终止 <code>bootmem</code> 分配器并替换为伙伴系统.....	297
20.1	<code>mem_init()</code> 函数的调用关系及其与数据结构的相互关系.....	297
20.2	查看 <code>mem_init()</code> 结构.....	298
20.3	记录到不存在的内存位图—— <code>free_unused_mmap_node()</code>	300
20.4	移交至普通空白页伙伴系统—— <code>free_all_bootmem_node()</code>	301
20.4.1	<code>register_page_bootmem_info_node()</code>	301
20.4.2	<code>free_all_bootmem_core()</code>	303

20.4.3	__free_pages_bootmem()	305
20.4.4	__free_pages()	308
20.4.5	free_hot_cold_page()	308
20.4.6	__free_pages_ok()	309
20.5	移交到高端内存空白页伙伴系统——free_area()	314
第 21 章	初始化以支持 CPU 热插拔	315
21.1	初始化 cpu_hotplug 成员变量——cpu_hotplug_init()	315
21.2	CPU 的联机→脱机转换处理	316
第 22 章	激活 slab 内存分配器——kmem_cache_init()	318
22.1	slab 分配器的概念及结构体	318
22.2	slab 分配器的重要结构体——kmem_cache 和 kmem_list3	319
22.3	查看 kmem_cache_init()结构	322
22.4	初始化 initkmem_list3[]、cache_cache、nodelist[]	326
22.5	连接 kmem_list3 数组并决定 cache 压缩时间——set_up_list3s()	328
22.6	求出用于 cache 扩展/压缩的页顺序——cache_estimate()	329
22.7	malloc_sizes 和 cache_names	332
22.8	生成 cache——kmem_cache_create()	334
22.8.1	kmem_cache_zalloc()	336
22.8.2	calculate_slab_order()	337
22.8.3	setup_cpu_cache()	337
22.8.4	enable_cpucache()	339
22.9	生成 arraycache_init,kmem_list3 cache	340
22.10	用 kmalloc()函数分配的内存替代静态分配的内存	342
第 23 章	kmem_trace_init()~security_init()	344
23.1	生成 ID allocator 缓存——idr_init_cache()	345
23.2	初始化 pageset——setup_per_cpu_pageset()	345
23.3	指定交叉节点——numa_policy_init()	350
23.4	结束计时器初始化——late_time_init()	353
23.5	测定 BogoMIPS——calibrate_delay()	353
23.6	制作位图以分配进程识别符 (ID) ——pidmap_init()	354
23.7	初始化优先树的数据结构——prio_tree_init()	356
23.8	生成 anon_vma slab 缓存——anon_vma_init()	356
23.9	为对象的每个用户赋予资格——cred_init()	357
23.10	初始化数据结构以使用 fork()函数——fork_init()	358
23.11	初始化生成进程的缓存——proc_caches_init()	359
23.12	初始化缓冲缓存——buffer_init()	361
23.13	准备密钥——key_init()	364

第 24 章 初始化 VFS 中使用的多种缓存—— <code>vfs_cache_init()</code>	367
第 25 章 <code>radix_tree_init()~ftrace_init()</code>	382
25.1 基数树相关数据结构初始化—— <code>radix_tree_init()</code>	383
25.2 准备使用信号—— <code>signals_init()</code>	383
25.3 注册并挂载 <code>proc</code> 文件系统—— <code>proc_root_init()</code>	384
25.4 注册未能初始化的子系统—— <code>cgroup_init()</code>	385
25.5 重置 <code>top_cpuset</code> 并注册 <code>cpuset</code> 文件系统—— <code>cpuset_init()</code>	386
25.6 初始化任务统计信息接口—— <code>delayacct_init()</code>	387
25.7 为管理延迟信息做准备—— <code>delayacct_init()</code>	388
25.7.1 延迟审计.....	388
25.7.2 <code>delayacct_init</code>	389
25.7.3 <code>task_delay_info</code> 结构体和 <code>delayacct_tsk_init()</code>	390
25.8 检查写缓冲一致性—— <code>check_bugs()</code>	392
第 26 章 同步内存与后备存储—— <code>page write back</code>	394
26.1 页回写机制.....	394
26.2 激活页回写—— <code>pdflush_init()</code>	395
26.3 <code>pdflush</code> 线程.....	397
26.4 指定页回写函数.....	398
26.5 周期性页回写和强制性页回写回调函数调用方法.....	399
26.5.1 周期性页回写函数—— <code>wb_kupdate()</code>	399
26.5.2 强制性页回写函数—— <code>background_writeout()</code>	401
26.6 初始化周期性页回写.....	403
第 27 章 查看启动内核的最终函数结构—— <code>rest_init()</code>	405
第 28 章 生成执行函数的内核线程—— <code>kernel_thread()</code>	407
28.1 查看 <code>kernel_thread()</code> 结构.....	407
28.2 生成处理器的网关—— <code>do_fork()</code>	408
28.3 复制父进程—— <code>copy_process()</code>	411
第 29 章 唤醒新生成的任务.....	419
29.1 查看 <code>wake_up_new_task</code> 结构.....	419
29.2 获取任务的就绪队列—— <code>task_rq_lock()</code>	421
29.3 改善任务的优先顺序—— <code>effective_prio()</code>	422
第 30 章 准备使用内核.....	426
30.1 将当前进程转移到其他 CPU—— <code>sched_init_smp()</code>	427
30.2 结束系统整体初始化—— <code>do_basic_setup()</code>	429
30.2.1 生成执行 <code>rcu_sched_grace_period()</code> 的线程—— <code>rcu_init_sched()</code>	430

30.2.2	生成 events 工作队列——init_workqueues	430
30.2.3	初始化 cpuset 子系统的 top_cpuset——cpuset_init_smp()	437
30.2.4	生成 khelper 工作队列——usermodehelper_init()	437
30.2.5	初始化 Linux 的设备模型——driver_init()	439
30.2.6	在 proc 文件系统注册 irq 信息——init_irq_proc()	446
30.2.7	调用内核未知子系统——do_initcalls()	448
30.3	为初始化之后的操作做准备——init_post()	451
第 31 章	内核线程守护进程	453
31.1	内核线程守护进程——kthreadd()	453
31.2	忽略信号——ignore_signals()	456
31.3	设置 nice 值——set_user_nice()	458
31.4	搜索执行任务的 CPU——set_cpus_allowed_ptr()	463
31.5	搜索包含列表的实际结构体位置——list_entry()	464
31.6	生成内核线程——create_kthread()	466
第 32 章	find_task_by_pid_ns()~cpu_idle()	469
32.1	用 PID 搜索任务——find_task_by_pid_ns()	470
32.2	解除 BKL——unlock_kernel()	472
32.3	将调度类变更为 idle——init_idle_bootup_task()	473
32.4	RCU 机制激活完成通知——rcu_scheduler_starting()	473
32.5	激活内核抢占——preempt_enable_no_resched()	473
32.6	执行进程调度表——schedule()	474
32.7	Linux 启动万里长征的终点——cpu_idle()	476
附 录		
附录 A	汇编语言、gas 关键词总结	480
附录 B	内核分析常见 API	485
附录 C	浅谈 ext2 文件系统	487
附录 D	Linux 线程模型	497
附录 E	链接器脚本文件结构	500
后记		510
索引		513

第一部分

ARM Linux内核

——分析内核前需要做的准备

Linux 内核源代码即使压缩后也超过了 70 MB。对如此庞大的源代码进行逐行分析往往需要付出莫大的努力。但是，需要倾注的努力与我们具备的 Linux 知识、对 ARM 处理器的理解、代码分析工具的使用经验等或许是一种反比关系。也就是说，如果我们具备了这些丰富的知识和经验，将会大大缩短分析代码的时间。因此，正式进入源代码分析之前，需要做好充分准备。

第一部分是内核代码分析的准备阶段，主要包括对 Linux 内核的介绍、如何编译内核并生成可加载到内存的图像、必须掌握的 ARM 处理器相关内容、为提高代码分析效率而搭建分析环境等代码分析必备常识。

真正进入代码分析之前，我们利用第一部分做个热身吧！

内核介绍及2.6版和3.2版之间的差异

本书对启动Linux内核过程中调用的代码进行分析，并着重讲解Linux是怎样运行的，以及Linux运行时哪些部分需要初始化等内容。

本章在分析之前，先讲解两个内容：第一，Linux是由林纳斯发起的当前最成功的操作系统之一；第二，Linux内核是由多个子系统构成的一个单内核。

最后，了解本书中分析的内核版本和内核3.2版之间的差异。在这一过程中大家会发现，内核从2.4版升级到2.6版的过程中，结构上并没有发生太大变化。所以本书并没有把重点放在内核3.2版上，但对其的讲解有助于分析最新版的内核。

1.1 内核的诞生、作用以及内部结构

1.1.1 Linus 创造的 Linux

Linux^①是由赫尔辛基大学的研究生林纳斯·托瓦兹(Linus Torvalds)在1991年发布的操作系统。林纳斯在大学期间对MINIX的许可授权政策感到不满，从而打算开发Linux。虽然开发初期是以MINIX为基础的，但进展到一定程度后，他在运行Linux内核的Linux系统上进行了开发。^②

之后，与MINIX相关的组件全部被GNU应用程序取代^③。由于从GNU阵营中得到了许多支持和无数内核黑客的贡献，Linux成为全世界应用最广、最为成功的开放源操作系统。

① Linux的早期名称是不一样的。刚开始林纳斯将自己制作的组件命名为Freax，后来接受了上传其内核的网站管理者的提议，更名为Linux，并使用至今。Linux的意思是“林纳斯的UNIX”(Linus's UNIX)或“Linux不是UNIX”(Linux is not UNIX)，或者是“林纳斯的MINIX”(Linus's MINIX)。

② 构建Linux的编译器是linux-gcc。那么linux-gcc又是用什么编译器构建的呢？某位内核黑客与林纳斯对此进行过对话，参见如下URL：<http://groups.google.com/group/comp.os.linux/msg/4ae6db18d3f49b0e>。

③ 通过GNU获得了什么灵感或感动呢？在几乎与此同时的1992年年初，林纳斯用GPL取代了Linux的许可。

1.1.2 由多种子系统集成运行的单内核

Linux内核由许多子系统构成。图1-1表示各子系统之间的关系，请通过该图了解各子系统的作用。

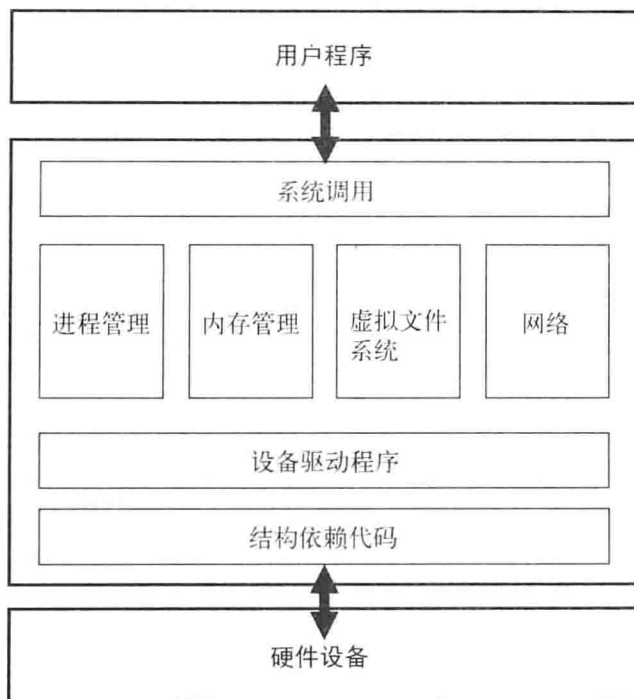


图1-1 Linux子系统结构

结构依赖代码

无论为何种结构类型，Linux均提供相同服务。但在其底层，每个结构都有需要另行控制的部分。这些部分是指用于CPU、MMU以及机载（on board）状态的低级（low-level）驱动程序，而且这些代码均位于arch目录下。

设备驱动程序

我们使用的电脑中，有很多周边设备（LCD、按钮、蓝牙、Wi-Fi等）通过多种方式（J2C、SCSI、EIDE）进行通信。设备驱动程序是指为这些高层设备制订的代码，位于drivers目录下。Linux内核中有一半以上是设备驱动程序代码，Linux通常最先封装最新设备。

进程管理

执行进程前必须先分配CPU资源。同时执行多个进程时，根据制定的政策公平分配资源，并且要管理进程的生成及状态转移的过程。进程管理就是执行这些任务的子系统，位于kernel目录下。

内存管理

与CPU同样重要的系统资源是内存。内存管理子系统负责内存分配、释放及共享，其相关代