奔跑吧 linux 内核 勘误

(更新到 2017-11-5)

To 亲爱的奔跑吧小伙伴：

感谢各位观看《奔跑吧 Linux 内核》，上市以来得到广大小伙伴的喜欢，上市两个月已经第四次印刷了，感谢各路小伙伴和叔叔们对奔跑吧的支持！**笨叔叔作为 Linux 内核的吃瓜叔叔，能力和水平实在有限，对大家阅读造成困扰表示深深歉意**。非常感谢众多小伙伴认真阅读，并且提出了很多很棒的勘误和吊打意见。笨叔叔再次对这些小伙伴表示深深感激。他们是彭东林、陈俊、朱凌宇、蔡琛、马学跃、刘金保、赵亚坤、郭建、郭任、王叔叔、宋叔叔。。。

《奔跑吧》最新的勘误会在异步社区（http://www.epubit.com.cn/book/details/4835）中，pdf 版本的勘误会在不定期更新，并且 pdf 版本和需要修改的图片会上传到笨叔叔的 github 上：https://github.com/figozhang/Running-LinuxKernel

欢迎各位小伙伴继续对笨叔叔和《奔跑吧》进行尖锐严厉严肃严苛严格的吊打、批评以及嘲笑，也期待大家对《奔跑吧》后续的改版和优化提出有意思意见和建议。

微信公众号/微博：奔跑吧 linux 内核

微信：runninglinuxkernel

另外：有的小伙伴反馈笨叔叔的 github 上的实验代码很难 git clone 下来，为此我压缩了一个不带 gitlog 信息的包上传到网盘上，只有 120MB.

http://pan.baidu.com/s/1i5y4bid

P1 勘误

P1 勘误主要是一些技术错误、书写错误以及排版错误可能会对原文的理解产生困扰。

页数	行数	原文	更正
3		git reset v4.0 -hard	git reset v4.0 --hard
7		经典处理器架构的流水线是五级流水线：取指、译码、发射、执行和写回	经典处理器架构的流水线是五级流水线：取指（IF）、译码（ID）、执行（EX）、数据内存访问（MEM）和写回.
8		在寄存器重名阶段（Register rename stage）会做寄存器重命名	在寄存器重命名阶段（Register rename stage）会做寄存器重命名

9		储指令会计算有效地址并发射到内存系统中的 LSU 部件（Load Store Unit）	储指令会计算有效地址并发**送**到内存系统中的 LSU 部件（Load Store Unit）
11		编译时的乱序访问可以通过 volatile 关键字来规避	编译时的乱序访问可以通过 **barrier()函数**来规避
40		第 40 页描述“区间 1”和“区间 2” 多写了一个 0	正确应为： （1） 区间 1 物理地址：0x60000000～0x60800000。虚拟地址：0xc0000000～0xc0800000。 （2） 区间 2 物理地址：0x60800000～0x8f800000。虚拟地址：0xc0800000～0xef800000。
45		线性映射的物理地址等于虚拟地址 vaddr 减去 PAGE_OFFSET(0xC000_0000) 再减去 PHYS_OFFSET	线性映射的物理地址等于虚拟地址 vaddr 减去 PAGE_OFFSET(0xC000_0000) 再**加上** PHYS_OFFSET”
47		一个 pageblock 的大小通常是（MAX_ORDER - 1）个页面	一个 pageblock 的大小通常是 **2 的（MAX_ORDER - 1）**次幂个页面
60		下面是页面大小为 4K，地址宽度为 48 位，3 级映射的内存分布图:	下面是页面大小为 4K，地址宽度为 **39** 位，3 级映射的内存分布图：
61		如果 bit[63]等于 0，那么这个虚拟地址属于用户空间，页表基地址寄存器用 TTBR0.	改为：如果 bit[63]等于 0，那么这个虚拟地址属于用户空间，页表基地址寄存器用 **TTBR0_EL1**.
66		通过 pmd_offset()宏来获取相应的 PUD 表项。这里会通过 pud_index 来计算索引值	通过 pmd_offset()宏来获取相应的 **PMD** 表项。这里会通过 **pmd_index** 来计算索引值
71			图 2.7 需要更正，其中 用户空间的地址范围：0x0 ~ 0x0000_7fff_ffff_ffff 改为：用户空间的地址范围：0x0 ~ 0x0000_ffff_ffff_ffff
72		相比于多次分配离散的物理页面，分配连续的物理页面有利于提高系统内存的碎片化	相比于多次分配离散的物理页面，分配连续的物理页面有利于**缓解**系统内存的碎片化
97		所以一个 slab 的大小最大为 2^25 个页面，即 32MB 大小。	所以一个 slab 的大小最大为 2^25 个页面，即 32MB 大小，**但是不能大于页面分配器所能分配的最大的内存块。**
107		page-freelist 是内存块开始地址减去 cache colour 后的地址	page-freelist 是内存块开始地址**加上** cache colour 后的地址
107		page->s_mem 是 slab 中第一个对象的开始地址，内存块开始地址减去 cache colour 和 freelist_size	page->s_mem 是 slab 中第一个对象的开始地址，内存块开始地址**加上** cache colour 和 freelist_size
109		ac->avail 大于 ac->limit 阀值时	ac->avail 大于**或等于** ac->limit 阀值时
110		假设共享对象缓冲池中的空闲对象数量大于 limit 阈值	假设共享对象缓冲池中的空闲对象数量**等于** limit 阈值
113		ac->avail 大于缓冲池的极限值 ac-limit 时	ac->avail **等于**缓冲池的极限值 ac-limit 时
121		vm_pgoff：指定文件映射的偏移量，这个变	vm_pgoff：指定文件映射的偏移量，这个变

		量的单位不是 Byte，而是页面的大小（PAGE_SIZE）。	量的单位不是 Byte，而是页面的大小（PAGE_SIZE）。**对于匿名页面来说，它的值可以是 0 或者 vm_addr/PAGE_SIZE。(这里补充一句)**

127		int main ()
		{
		struct s_node root= {0, NULL};
		slist_insert(root, 2);
		slist_insert(root, 5);
			int main ()
			{
			struct s_node head = {0, NULL};
			struct s_node *root = &head;
			slist_insert(root, 2);
			slist_insert(root, 5);

137		第 12 行代码，get_unmapped_area()函数用来判断虚拟内存空间是否有足够的空间，返回一段没有映射过的空间的起始地址，这个函数会调用到具体的体系结构中实现。	这里描述不严谨，后面增加一句： 第 12 行代码，get_unmapped_area()函数用来判断虚拟内存空间是否有足够的空间，返回一段没有映射过的空间的起始地址，这个函数会调用到具体的体系结构中实现。**注意这里 flags 参数是 MAP_FIXED，表示使用指定的虚拟地址对应的空间。**
138		表示需要马上为这块进程地址空间 VMA 的分配物理页面并建立映射关系	表示需要马上为描述这块进程地址空间的 **VMA 分配物理页面并建立映射关系**
164		第 7 行的注释说明有的处理器体系结构会大于 8Byte 的 pte 表项	第 7 行的注释说明有的处理器体系结构的 pte 页表项**会大于字长(word size)**
165			图 2.19 需要修改
173		第 13 行代码，为 GFP_HIGHUSER | __GFP_MOVABLE 的新页面 new_page 分配一个分配掩码	第 13 行代码，**以** GFP_HIGHUSER | __GFP_MOVABLE **为分配掩码为 new_page 分配一个新的物理页面**
177		如果 page 的 PG_locked 位已经置位，那么当前进程调用 trylock_lock()返回 false	如果 page 的 PG_locked 位已经置位，那么当前进程调用 trylock_**page**()返回 false
177		第 34 行代码，判断当前页面是否为不属于 KSM 的匿名页面。利用 page-mapping 成员的最低 2 个比特位来判断匿名页面使用 PageAnon()宏，定义在 include/linux/mm.h 文件中。	第 34 行代码，判断当前页面是否为不属于 KSM 的匿名页面。**使用 PageAnon()这个宏来判断匿名页面，其定义在 include/linux/mm.h 文件中，它是利用 page-mapping 成员的最低 2 个比特位来做判断。**
183			图 2.22 需要修改
190		copy_pte_range()->copy_one_te()函数。	copy_pte_range()->copy_one_**pte**()函数。
197		__page_set_anon_rmap()函数中的第 20 行代码，linear_page_index()函数计算当前地址 address 是在 VMA 中的第几个页面，然后把 offset 值赋值到 page->index 中，详见第 2.17.2 节中关于 page->index 的问题。	**这句后面增加一个脚注：** 对于匿名页面来说，vma->vm_pgoff 这个成员的值是 0 或者 vm_addr/PAGE_SIZE，比如 mmap 中 MAP_SHARED 映射时 vm_pgoff 为 0，MAP_PRIVATE 映射时为 vm_addr/PAGE_SIZE。vm_addr/PAGE_SIZE 表示匿名页面在整个进程地址空间中的 offset。vm_pgoff 这个值在匿名页面的生命周期中只有 RMAP 反向映射时才用到。笔者认

			为，对于匿名页面来说把 vma-vm_pgoff 看成 0 可能更好的体现出 page->index 的含义来，把 page->index 看成是一个 VMA 里面的 offset，而不是整个进程地址空间的 offset，也许更容易理解一些。
197			**图 2.24 少了一根虚线**
199		第 8 行代码，分配一个属于子进程的 avc 数据结构。 第 16 行代码，通过 pavc 找到父进程 VMA 中的 anon_vma。 第 18 行代码，anon_vma_chain_link()函数把属于子进程的 avc 挂入子进程的 VMA 的 anon_vma_chain 链表中，同时也把 avc 添加到属于父进程的 anon_vma-rb_root 的红黑树中，	第 8 行代码，**分配一个新的 avc 数据结构，这里称为 avc 枢纽**。 第 16 行代码，通过 pavc 找到父进程 VMA 中的 anon_vma。 第 18 行代码，anon_vma_chain_link()函数把这个 **avc 枢纽挂入子进程的 VMA 的 anon_vma_chain 链表中，同时也把 avc 枢纽添加到属于父进程的 anon_vma->rb_root 的红黑树中，**
203		因为在父进程的 AVp 队列中会有 100 万个匿名页面，扫描这个队列要耗费很长的时间。	有 100 万个匿名页面指向父进程的 AVp 数据结构。每个匿名页面在做反向映射时，最糟糕的情况下需要扫描这个 AVp 队列全部成员，但是 AVp 队列里大部分的成员（VMA）并没有映射这个匿名页面。这个扫描的过程是需要全程持有锁的，锁的争用变得激烈，导致有一些性能测试中出现问题。
235		在 get_scan_count()计算匿名页面和文件缓存页面分别扫描数量时会用到	**它们**在 get_scan_count()**分别**计算匿名页面和文件缓存页面的扫描数量时会用到
238		第 34 行代码，get_page_unless_zero()是为 page-_count 引用计数加 1，并且判断加 1 之后是否等于 0，也就是说，这个 page 不能是空闲页面，否则返回-EBUSY。	第 34 行代码，get_page_unless_zero()是为 page->_count 引用计数加 1，**先判断是否为非 0 然后再加 1**，也就是说，这个 page 不能是空闲页面，否则返回-EBUSY。
243		第 70～81 行代码，page 有多个用户映射(page-_mapcount=0)	第 70～81 行代码，page 有**一个或多个**用户映射(page-_mapcount=0)
249			**图 2.31 排版时候弄错了，在第二次印刷的已经修改了**
255		migrate_pages()函数的参数 from 表示将要迁移的页面链表，get_new_page 是内存函数指针，put_new_page 是迁移失败时释放目标页面的函数指针，private 是传递给 get_new_page 的参数，mode 是迁移模式，reason 表示迁移的原因。第 11 行代码，for 循环表示这里会尝试 10 次。	migrate_pages()函数的参数 from 表示将要迁移的页面链表，get_new_page 是**申请新内存页面的函数指针**，put_new_page 是迁移失败时释放目标页面的函数指针，private 是传递给 get_new_page 的参数，mode 是迁移模式，reason 表示迁移的原因。第 12 行代码，for 循环表示这里会尝试 10 次。
258		第 58~59 行，对于迁移失败的页面，调用 remove_migration_ptes()删除迁移的 pte。	第 58~59 行，对于迁移失败的页面，调用 remove_migration_ptes()**把迁移失败页面的 pte 设置回原来的位置。**
296		原文：然后查询 stable 树，还需要多次的 memcpy 次比较，合并 10000 次 pte 页表项也	然后查询 stable 树，还需要多次的 **memcmp** 次比较，合并 10000 次 pte 页表项也就意味

		就意味着 memcpy 需要 10000 次	着 **memcmp** 需要 10000 次
333		普通进程的优先级：100～139。 实时进程的优先级：1～99。 Deadline 进程优先级：0。	普通进程的优先级：100～139。 实时进程的优先级：0～99。 Deadline 进程优先级：**-1**。
352		原文：包括运行时间或等待 CPU 时间	改为：包括运行时间**和**等待 CPU 时间
505		__asm__ __volatile__("@ atomic_add "\n"	应该改为 __asm__ __volatile__("@ atomic_add\n" **（去掉一个"号）**
507		原文：待某些事件需要睡眠，例如调用 wait_even（）。睡眠者代码片段如下：	待某些事件需要睡眠，例如调用 wait_**event**（）。睡眠者代码片段如下：
596		*如果该中断源是 pening 状态 *那么 pending 状态变成 avtive and pending *如果中断是 ative 状态，现在变成 avtive and pending	*如果该中断源是 **pending** 状态 *那么 pending 状态变成 **active** and pending *如果中断是 **active** 状态，现在变成 **active** and pending
597		中断 N 的状态从 pending 变成 avtive and pending	中断 N 的状态从 pending 变成 **active** and pending
597		原文：中断 M 的状态为 active and pinding	中断 M 的状态为 active and **pending**
609		其中__irq_set_handler()用来设置中断描述符 desc-hander_irq 的回调函数，	改为：其中__irq_set_handler()用来设置中断描述符 desc->**handle**_irq 的回调函数，
611			表 5.2 中的 RQF_ONESHOT 改为：IRQF_ONESHOT
615		原文：确保即该内核线程	确保即**使**该内核线程
617		原文： old_prt 指向 irqaction 链表末尾	改为：old_**ptr** 指向 irqaction 链表末尾
624		通常 stack_hole=0 、S_FRAME_SIZE=18 、S_FRAME_SIZE 称为寄存器框架大小，	改成：通常 stack_hole=0 、S_FRAME_SIZE=**72** ，其中 S_FRAME_SIZE 称为寄存器框架大小，
625			**图 5.5 有错误，需要修正**
625		第 4 行代码，r0 寄存器还保存着 IRQ 模式的栈指针，IRQ 模式的栈空间分别保存着 r0、LR_irq 和 CPSR_irq 寄存器的内容。通过 ldmia 指令，把 IRQ 模式的栈空间复制到 SVC 模式的 r3、r4 和 r5 寄存器中。 第 7 行代码，把 CPSR_svc 寄存器的内容赋值到 r2 寄存器中。 第 8 行代码，刚才已把 IRQ 模式的 r0 寄存器内容复制到 r3 寄存器，现在重新赋值到 SVC 模式的 r0 寄存器中。 第 22 行代码，这时 r2 寄存器存放 SPSR_svc，r3 寄存器存放 LR_svc，r4 寄存器存放 IRQ 模式的 LR_irq，r5 寄存器存放 CPSR_irq，r6 寄存器存放-1。通过 stmia 指令把这些寄存器的内容保存到 SVC 模式的栈中的 ARM_sp、ARM_lr、ARM_pc、ARM_cpsr 和 ARM_ORIG_r0 中。	第 4 行代码，r0 寄存器还保存着 IRQ 模式的栈指针，IRQ 模式的栈空间分别保存着 r0、LR_irq 和 **SPSR_irq** 寄存器的内容。通过 ldmia 指令，把 IRQ 模式的栈空间复制到 SVC 模式的 r3、r4 和 r5 寄存器中。 第 7 行代码，**把 svc 栈顶的地址赋值到 r2 寄存器中。** 第 8 行代码，刚才已把 IRQ 模式的 r0 寄存器内容复制到 r3 寄存器，现在重新赋值到 SVC 栈的 **ARM_r0** 处。 第 22 行代码，这时 r2 寄存器**存放 svc 的栈顶地址**，r3 寄存器存放 LR_svc，r4 寄存器存放 IRQ 模式的 LR_irq，r5 寄存器存放 **SPSR_irq**，r6 寄存器存放-1。通过 stmia 指令把这些寄存器的内容保存到 SVC 模式的栈中的 ARM_sp、ARM_lr、ARM_pc、ARM_cpsr 和 ARM_ORIG_r0 中。

627		原文：说明是一个外设中断或 SPI 和 PPI 类型中断	说明是一个外设中断（SPI 或 PPI 类型中断）
657		省电类型的工作队列 system_freezable_wq	省电类型的工作队列 system_power_efficient_wq
658		原文：表示 UNBOUND 类型的工作线程，名字为“kworker/u + CPU_ID + worker_id”	表示 UNBOUND 类型的工作线程，名字为“kworker/u + pool_id + worker_id”
664		原文：当 pool_workqueue-refcnt 成员计数小于 0 时，会通过	当 pool_workqueue-refcnt 成员计数等于 0 时，会通过
694		原文： 静态代码插装技术	改为： 静态代码插桩技术
729		CONFIG_MESSAGE_LOGLEVEL_DEFAULT=8 //默认打印等级设置为 0，即打开所有的打印信息这里错了，	CONFIG_MESSAGE_LOGLEVEL_DEFAULT=8 //默认打印等级设置为 8，即打开所有的打印信息
730		# echo 'file svcsock.c +p' > /sys/kernel/dynamic_debug/control	这里举的几个例子里面路径漏了 debug，正确的路径如下：/sys/kernel/debug/dynamic_debug/control
734		原文： 可以使用 objdum 工具	可以使用 objdump 工具
734		使用 gbd 中的"l"指令加上出错函数和偏移量即可。	使用 gdb 中的"list"指令加上出错函数和偏移量即可。

P2 勘误

P2 勘误主要是一些拼写错误、大小写等问题，不影响对原文的理解，这里不就列出来了，可以参考异步社区上的勘误。

http://www.epubit.com.cn/book/details/4835