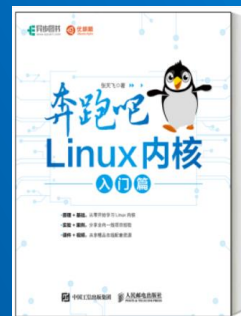




奔跑吧Linux内核*入门篇

第十章 中断管理

笨叔叔



目 录

- 中断管理背景介绍
- ARMv7上的中断处理
- 中断上下文
- 中断控制器
- Linux中断处理过程
- 中断下半部机制
- 实验

中断管理背景介绍

为什么需要中断?

- 轮询 和 中断的区别
- 轮询的优缺点?
- 中断的优缺点?

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

什么是同步和异步？

➤ ARM芯片手册里关于同步和异步的概念

D1.2.5 Definitions of synchronous and asynchronous exceptions

An exception is described as synchronous if all of the following apply:

- The exception is generated as a result of direct execution or attempted execution of an instruction.
- The return address presented to the exception handler is guaranteed to indicate the instruction that caused the exception.
- The exception is precise.

For more information about synchronous exceptions, see *Synchronous exception types, routing and priorities* on page D1-1894.

An exception is described as *asynchronous* if any of the following apply:

- The exception is not generated as a result of direct execution or attempted execution of the instruction stream.
- The return address presented to the exception handler is not guaranteed to indicate the instruction that caused the exception.
- The exception is imprecise.

For more information about asynchronous exceptions, see *Asynchronous exception types, routing, masking and priorities* on page D1-1902.

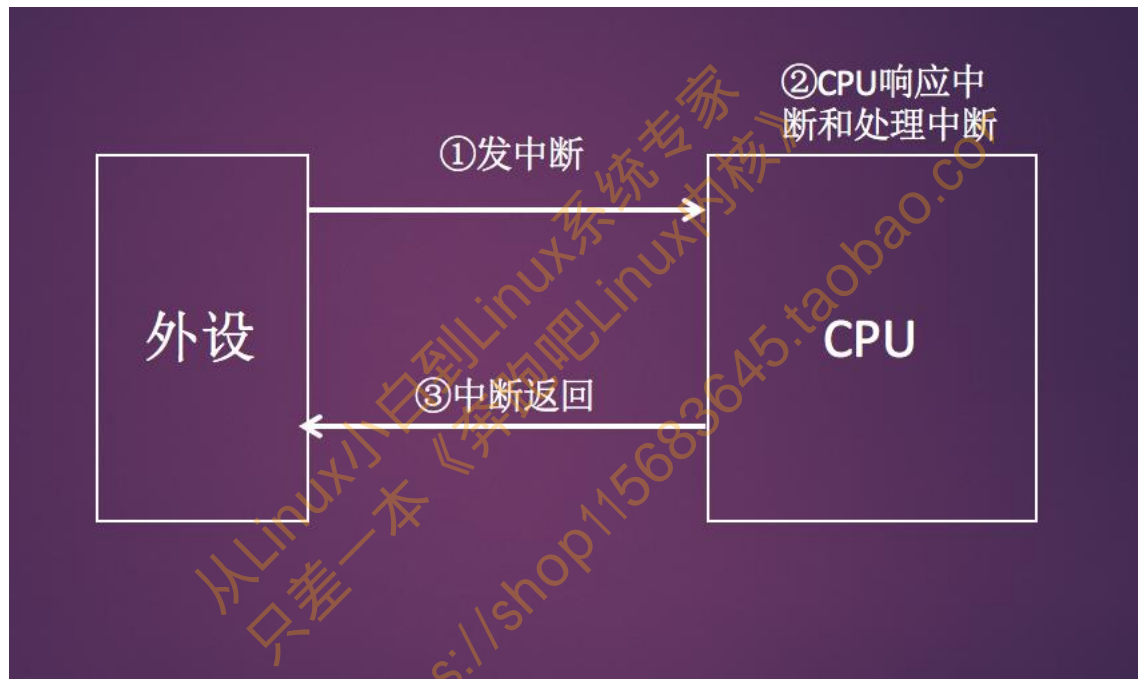
ARM v8手册第D1.2.5章

中断的类型

- 中断：系统计划外发生的事情，类似突发事件。属于异步。
- 异常：通常是执行到某条非法指令，是计划内的事情，有点类似同步。
- 自陷：程序自愿请求陷入内核态。比如系统调用。

从Linux小白到Linux系统内核
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

中断处理总体概述



Linux内核中断使用快速入门

- 注册中断函数：request_irq()

```
134 static inline int __must_check
135 request_irq(unsigned int irq, irq_handler_t handler, unsigned long flags,
136             ? const char *name, void *dev)
137 {
138     return request_threaded_irq(irq, handler, NULL, flags, name, dev);
139 }
140
```

- 写自己的中断处理函数。
- Linux源代码的driver目录有很多例子可以参考
- 例如：drivers/misc/bmp085.c
- drivers/char/tpm/tpm_tis.c

ARM v7上中断处 理

从Linux小白到linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683649.mobao.com>

ARM v7上的中断向量表

- 啥是中断向量表
- ARM v7上的中断向量表

Table B1-3 The vector tables

Offset	Vector tables			
	Hyp ^a	Monitor ^b	Secure	Non-secure
0x00	Not used	Not used	Reset	Not used
0x04	Undefined Instruction, from Hyp mode	Not used	Undefined Instruction	Undefined Instruction
0x08	Hypervisor Call, from Hyp mode	Secure Monitor Call	Supervisor Call	Supervisor Call
0x0C	Prefetch Abort, from Hyp mode	Prefetch Abort	Prefetch Abort	Prefetch Abort
0x10	Data Abort, from Hyp mode	Data Abort	Data Abort	Data Abort
0x14	Hyp Trap, or Hyp mode entry ^c	Not used	Not used	Not used
0x18	IRQ interrupt	IRQ interrupt	IRQ interrupt	IRQ interrupt
0x1C	FIQ interrupt	FIQ interrupt	FIQ interrupt	FIQ interrupt

ARM v7手册第B1.8.1章

ARM v7中断发生硬件做了那些？

- 当硬件异常发生了，这时候处理器要识别出是哪个模式的异常。
- 把当前CPSR寄存器的值存放在SPSR里。
- 异常模式的lr寄存器保存着异常发生时候那个地址，这个就是异常返回的链接地址
- 硬件会修改当前模式下的CPSR寄存器，然后进入到对应的异常模式里。
- 然后PC指针指向异常向量表对应的表项里
- 然后执行跳转。

从Linux内核到Linux系统编程
只差一本《奔跑吧Linux内核》
<https://shop11566645.taobao.com>

```

1218 __vectors_start:
1219     W(b)    vector_rst
1220     W(b)    vector_und
1221     W(ldr)   pc, __vectors_start + 0x1000
1222     W(b)    vector_pabt
1223     W(b)    vector_dabt
1224     W(b)    vector_addrxcptn
1225     W(b)    vector_irq
1226     W(b)    vector_fiq
1227

```

Linux 内核里定义的中断向量表

```

12 __reset:
13     b _startup      /* 0x00 Reset */
14     b _e_undef      /* 0x04 Undefined instruction */
15     b _e_svc        /* 0x08 Supervisor call */
16     b _e_pref_abort /* 0x0c Prefetch abort */
17     b _e_data_abort /* 0x10 Data abort */
18     b _e_unused     /* 0x14 Unused */
19     b _e_irq        /* 0x18 IRQ Interrupt */
20     b _e_fiq        /* 0x1c FIQ Interrupt */
21

```

小OS里定义的中断向量表

中断上下文

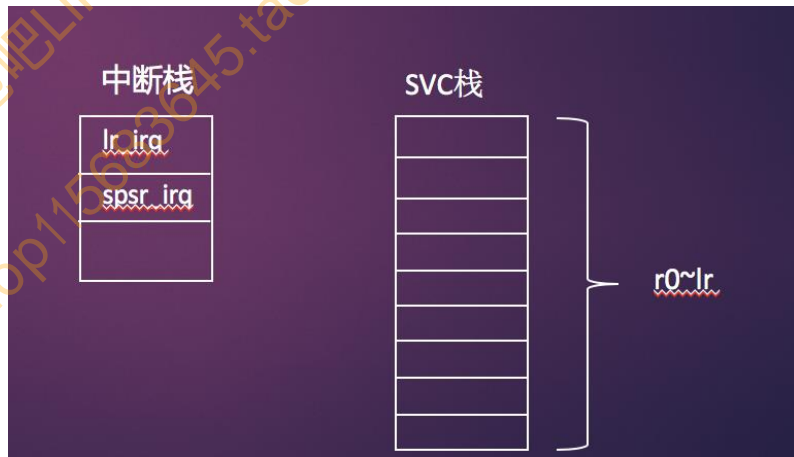
从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

中断上下文保存

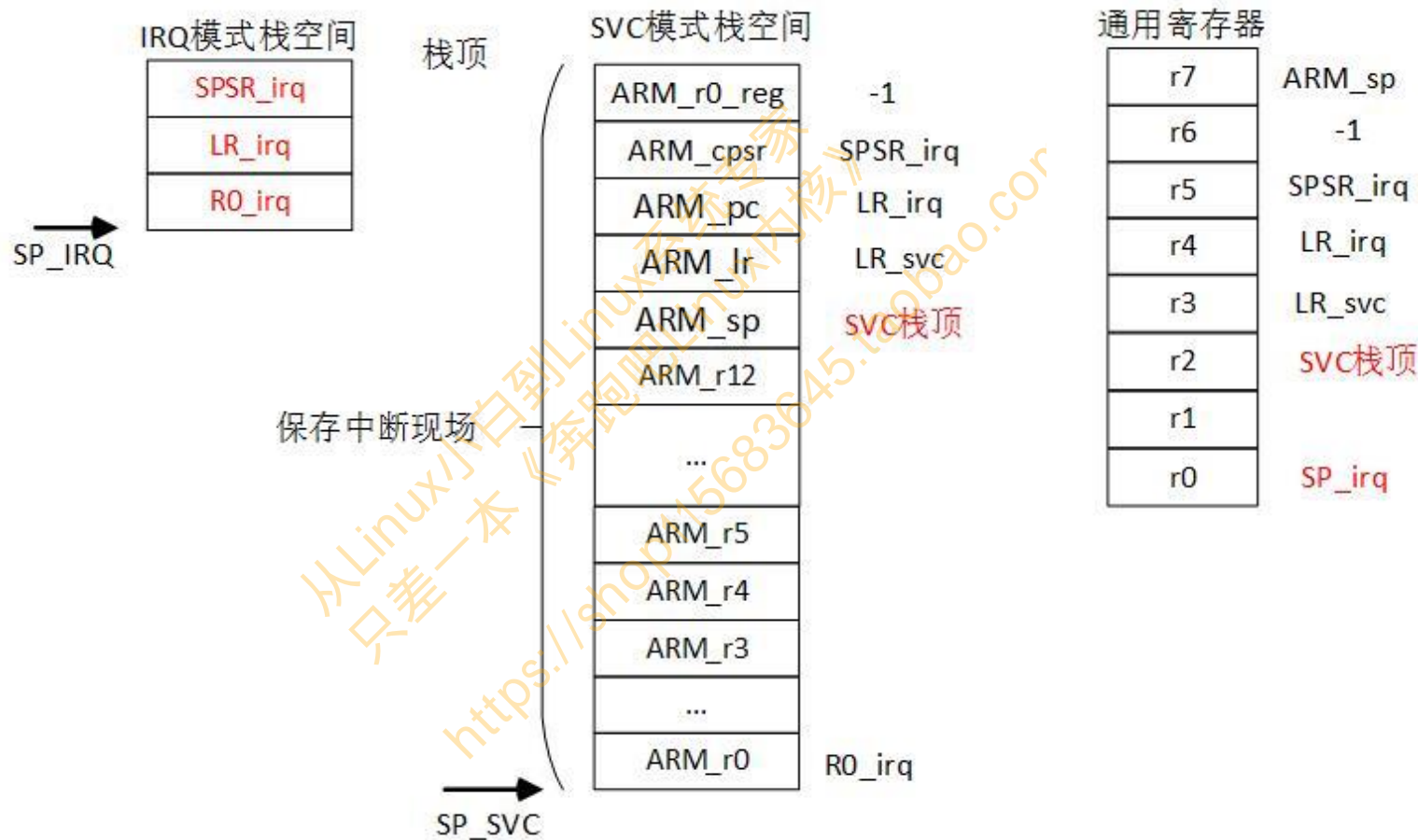
- 中断发生之后，软件需要做的事情：
 - ✓ 保存中断现场
 - ✓ 跳转到中断处理函数里
 - ✓ 恢复中断线程，返回到被中断点运行

- 哪些东西算是中断现场？

- ✓ lr_irq寄存器
- ✓ spsr_irq寄存器



Linux中断管理之中断上下文保存



中断控制器

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop1156836.taobao.com>

三星2140上的中断控制器

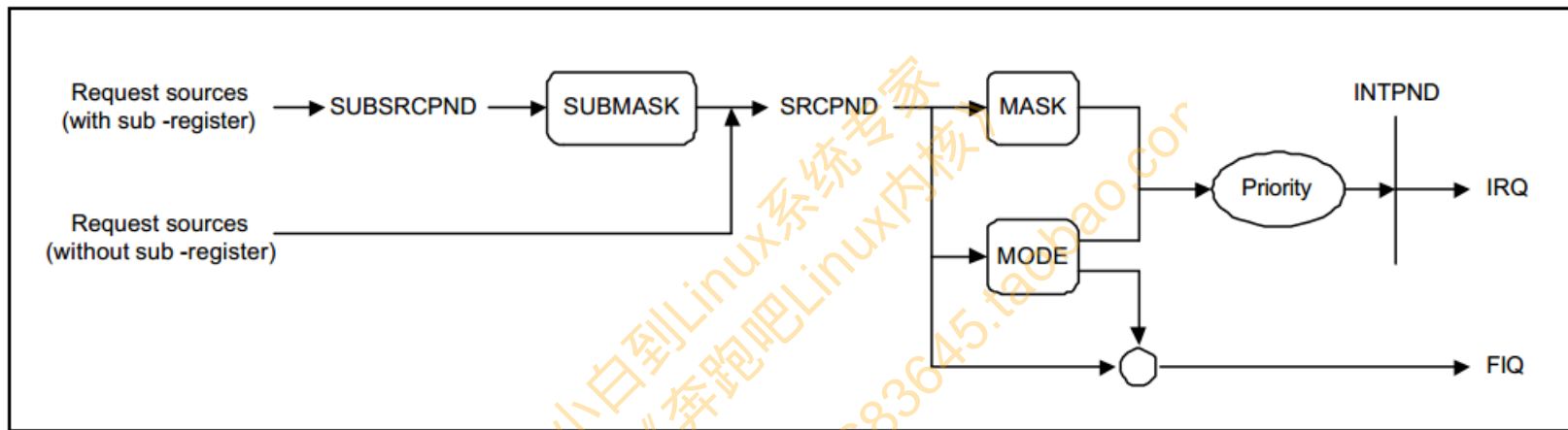
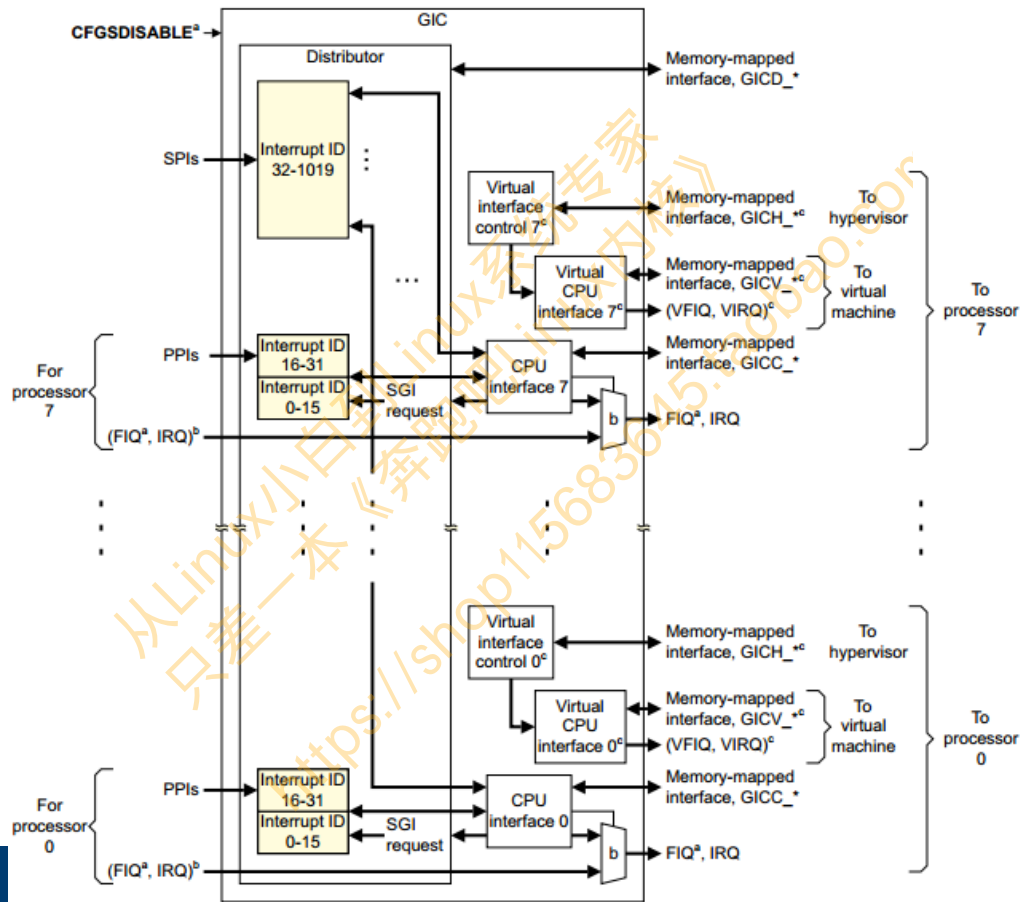


Figure 14-1. Interrupt Process Diagram

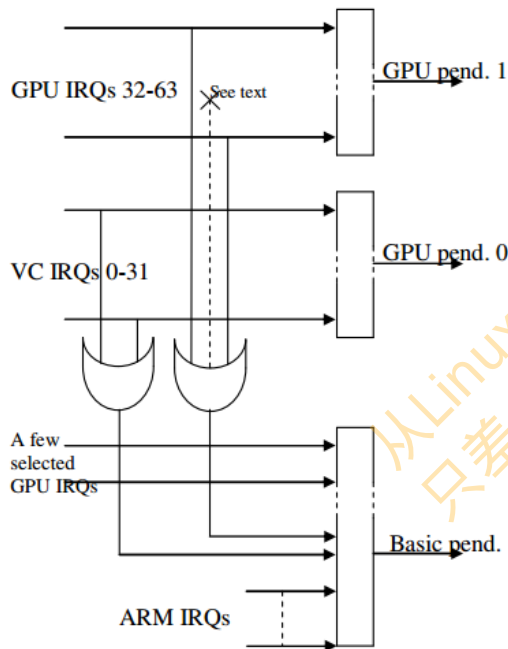
通过几个简单的寄存器来控制这个中断控制器：
中断pending寄存器：每个bit位表示一个中断源
中断mask寄存器：用来屏蔽某个中断源

ARM v7上的GIC v2中断控制器



树莓派3B+上的中断控制器

- 类似之前的ARM 9的，简单实用的
- 不支持中断优先级



Registers overview:

Address offset ⁷	Name	Notes
0x200	IRQ basic pending	
0x204	IRQ pending 1	
0x208	IRQ pending 2	
0x20C	FIQ control	
0x210	Enable IRQs 1	
0x214	Enable IRQs 2	
0x218	Enable Basic IRQs	
0x21C	Disable IRQs 1	
0x220	Disable IRQs 2	
0x224	Disable Basic IRQs	

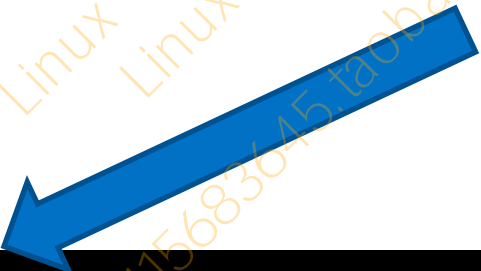
Linux中断处理过程

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

硬件中断号和软件中断号

- 硬件中断号：硬件SoC设计的时候就确定下来的中断号
- Linux软件中断号：Linux系统映射的软件中断号

Linux系统软件分配和映射的中断号



```
134 static inline int __must_check
135 request_irq(unsigned int irq, irq_handler_t handler, unsigned long flags,
136             ? const char *name, void *dev)
137 {
138     return request_threaded_irq(irq, handler, NULL, flags, name, dev);
139 }
140
```

ARM Vexpress 开发板

- cat /proc/interrupts 查看中断分配情况

硬件中断号

软件中断号

```
/ # cat /proc/interrupts
CPU0
18: 749 GIC 27 arch_timer
20: 0 GIC 34 timer
21: 0 GIC 127 vexpress-spc
38: 0 GIC 47 eth0
41: 0 GIC 41 mmci-pl18x (cmd)
42: 0 GIC 42 mmci-pl18x (pio)
43: 8 GIC 44 kmi-pl050
44: 100 GIC 45 kmi-pl050
45: 75 GIC 37 uart-pl011
51: 0 GIC 36 rtc-pl031
IPI0: 0 CPU wakeup interrupts
IPI1: 0 Timer broadcast interrupts
IPI2: 0 Rescheduling interrupts
IPI3: 0 Function call interrupts
IPI4: 0 Single function call interrupts
IPI5: 0 CPU stop interrupts
IPI6: 0 IRQ work interrupts
IPI7: 0 completion interrupts
Err: 0
```

以uart0为例：
硬件中断号：37
Linux中断号：45

硬件资料

➤ Cortex-A15_A7 MPCore test chip

(1) 内部中断。

- 32个内部中断用于连接CPU核和GIC中断控制器。

(2) 外部中断。

- 30个外部中断连接到主板的IOFPGA
- Cortex-A15 cluster连接8个外部中断
- Cortex-A7 cluster连接12个外部中断
- 芯片外部连接21个外设中断
- 还有一些保留未使用的中断

从Linux内核源码入门
只差一本《Linux内核入门》
<https://shop115683643.taobao.com>

表 5.1 Vexpress V2P-CA15_CA7 平台中断分配表

GIC 中断号	主板中断序号	中断源	信号	描述
0:31	—	MPCore cluster	—	CPU核和GIC的内部私有中断
32	0	IOFPGA	WDOG0INT	Watchdog timer
33	1	IOFPGA	SWINT	Software interrupt
34	2	IOFPGA	TIM01INT	Dual timer 0/1 interrupt
35	3	IOFPGA	TIM23INT	Dual timer 2/3 interrupt
36	4	IOFPGA	RTCINTR	Real time clock interrupt
37	5	IOFPGA	UART0INTR	串口0中断
38	6	IOFPGA	UART1INTR	串口1中断
39	7	IOFPGA	UART2INTR	串口2中断
40	8	IOFPGA	UART3INTR	串口3中断
42:41	10	IOFPGA	MCI_INTR[1: 0]	Media Card中断[1:0]
47	15	IOFPGA	ETH_INTR	以太网中断

硬件中断号和Linux中断号映射

- Linux注册中断函数使用Linux中断号，俗称软件中断号或者IRQ中断号

- 宏NR_IRQS

```
131 #define IRQ_GPIO3_START (IRQ_GPIO2_END + 1)
132 #define IRQ_GPIO3_END (IRQ_GPIO3_START + 31)
133
134 #define NR_IRQS (IRQ_GPIO3_END + 1)
```

- 位图变量allocated_irqs用来分配软件中断号

```
97 int nr_irqs = NR_IRQS;
98 EXPORT_SYMBOL_GPL(nr_irqs);
99
100 static DEFINE_MUTEX(sparse_irq_lock);
101 static DECLARE_BITMAP(allocated_irqs, IRQ_BITMAP_BITS);
102
```

硬件中断号和Linux中断号映射

- irq domain管理框架，每个中断控制器用一个irq domain数据结构来描述
- 中断映射过程是在：irq_domain_alloc_irqs()函数里
 - ✓ 从allocated_irqs位图里分配一个空间的比特位
 - ✓ 分配一个struct irq_irq_desc数据结构



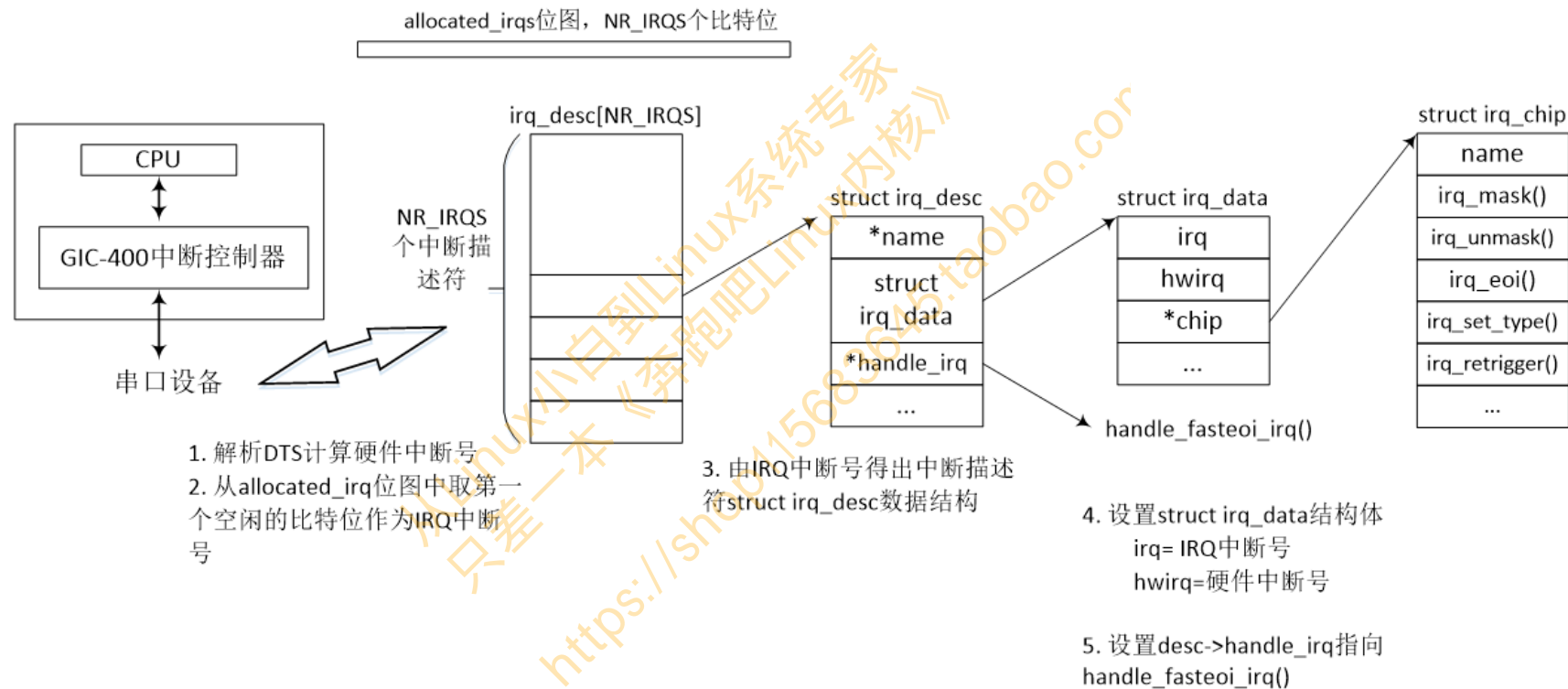
irq_desc描述符

- irq_desc[]定义成一个NR_IRQS个中断描述符
- 每个Linux中断号 对应一个irq_desc数据结构。

```
260 struct irq_desc irq_desc[NR_IRQS] __cacheline_aligned_in_smp = {  
261     [0 ... NR_IRQS-1] = {  
262         .handle_irq      = handle_bad_irq,  
263         .depth            = 1,  
264         .lock             = _RAW_SPIN_LOCK_UNLOCKED(irq_desc->lock),  
265     }  
266 };
```

- 数组下标表示IRQ中断号，通过IRQ中断号找到相应中断描述符

分配中断号总结



注册中断函数

- 中断处理程序 (interrupt handler) 或中断服务例程
- 注册中断API:

```
static inline int request_irq(unsigned int irq, irq_handler_t handler, unsigned long flags,  
                             const char *name, void *dev)
```

- 注册中断线程化API:

```
int request_threaded_irq(unsigned int irq, irq_handler_t handler,  
                         irq_handler_t thread_fn, unsigned long irqflags,  
                         const char *devname, void *dev_id)
```

中断标志位

表 5.2 中断标志位

中断标志位	描 述
IRQF_TRIGGER_*	中断触发的类型，有上升沿触发、下降沿触发、高电平触发和低电平触发
IRQF_DISABLED	此标志位已废弃，不建议继续使用
IRQF_SHARED	多个设备共享一个中断号。需要外设硬件支持，因为在中断处理程序中要查询是哪个外设发生了中断，会给中断处理带来一定的延迟，不推荐使用
IRQF_PROBE_SHARED	中断处理程序允许sharing mismatch发生
IRQF_TIMER	标记一个时钟中断
IRQF_PERCPU	属于特定某个CPU的中断
IRQF_NOBALANCING	禁止多CPU之间的中断均衡
IRQF_IRQPOLL	中断被用作轮询
IRQF_ONESHOT	One shot表示一次性触发的中断，不能嵌套。 (1) 在硬件中断处理完成之后才能打开中断； (2) 在中断线程化中保持中断关闭状态，直到该中断源上所有的thread_fn完成之后才能打开中断； (3) 如果request_threaded_irq()时primary handler为NULL且中断控制器不支持硬件ONESHOT功能，那应该显示地设置该标志位
IRQF_NO_SUSPEND	在系统睡眠过程中(suspend)不要关闭该中断
IRQF_FORCE_RESUME	在系统唤醒过程中必须强制打开该中断
IRQF_NO_THREAD	表示该中断不会被线程化

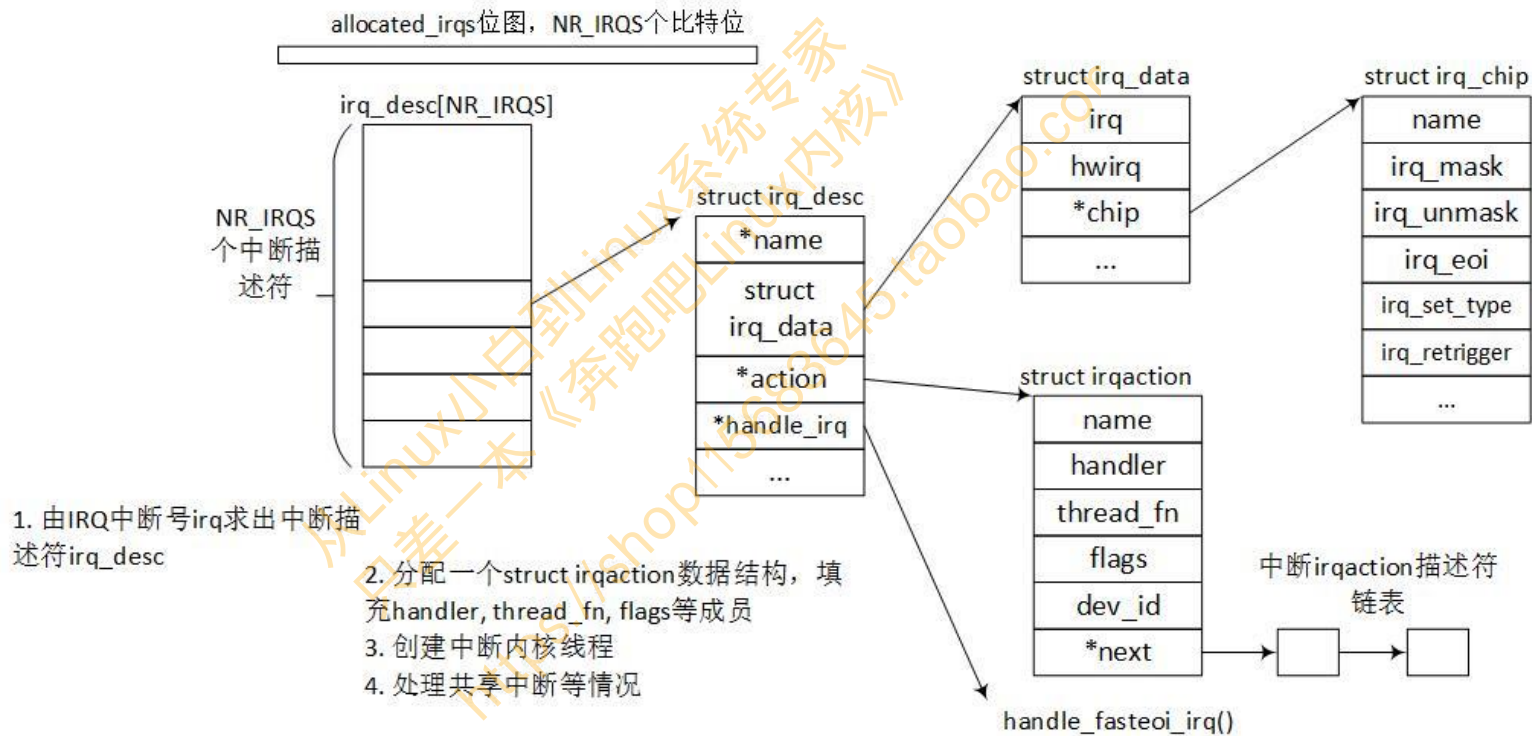
中断irqaction描述符

- 每个请求的中断都有一个中断action描述符
- 对于共享中断，多个irqaction描述符会串成一个链表



注册中断小结

`request_irq(irq, handler, flags, dev_id, name)` 注册中断



中断上半部处理

- 中断上半部和中断下半部
- 为啥要有中断上半部：
 - ✓ 硬件中断处理程序以异步方式执行，它会打断其他重要的代码执行，因此为了避免被打断的程序停止时间太长，硬件中断处理程序必须尽快执行完成。
 - ✓ 硬件中断处理程序通常在关中断的情况下执行。

从Linux小白到Linux内核专家
只差一本《奔跑吧Linux内核》
<https://shop11568345.taobao.com>

中断线程化

- 为啥有中断线程化?
- 中断线程化注册函数

```
int request_threaded_irq(unsigned int irq, irq_handler_t handler,  
                        irq_handler_t thread_fn, unsigned long irqflags,  
                        const char *devname, void *dev_id)
```

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

中断affinity

- SMP IRQ Affinity: 把中断均衡分布在不同的CPU核心上

```
/proc/irq # ls
16          21          44
17          38          45
18          41          51
19          42          default_smp_affinity
20          43
```

```
/proc/irq/45 # cat smp_affinity
01
/proc/irq/45 #
```

中断下半部机制

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop11568364.taobao.com>

中断上半部和下半部

- 为什么要有中断上半部和下半部机制?

- ✓ 硬件中断是异步发生的
- ✓ 关中断时间越短越好

- 什么是关中断?

- Linux内核有哪些下半部机制?

从Linux入门到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

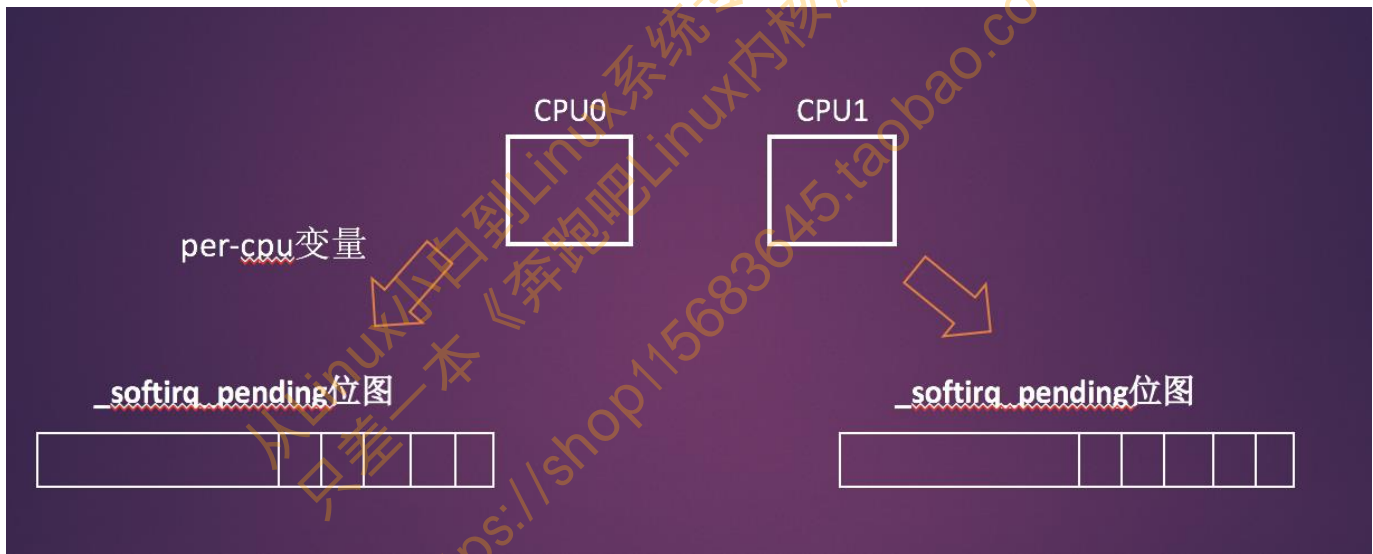
中断下半部执行的时机点

- 延迟执行的代码究竟放在什么时机点来运行？
- 方案一：中断处理函数（上半部）完成后，即将返回中断现场时
- 方案二：把下半部要执行的部分放到一个内核线程里，在中断返回之后，由调度器来调度这个内核线程来执行。

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683605.taobao.com>

软中断

- 软中断 SoftIRQ: 软件实现的一种机制，而非硬件实现的中断



CPU0

CPU1



per-cpu变量

_softirq_pending位图_softirq_pending位图softirq_vec[]

action()
action()
action()
...

中断处理函数完成之后，返回中断现场之前：

- 检查本地CPU的软中断状态寄存器 _softirq_pending 是否有没有处理的软中断？
- 处理对应的软中断 softirq_vec[] -> action()

软中断API

- 软中断定义的类型:
- 注册软中断

```
void open_softirq(int nr, void (*action)(struct softirq_action *))
```

- 触发一个软中断

```
void raise_softirq(unsigned int nr)
```

- 软中断处理核心函数 do_softirq()

```
[include/linux/interrupt.h]
```

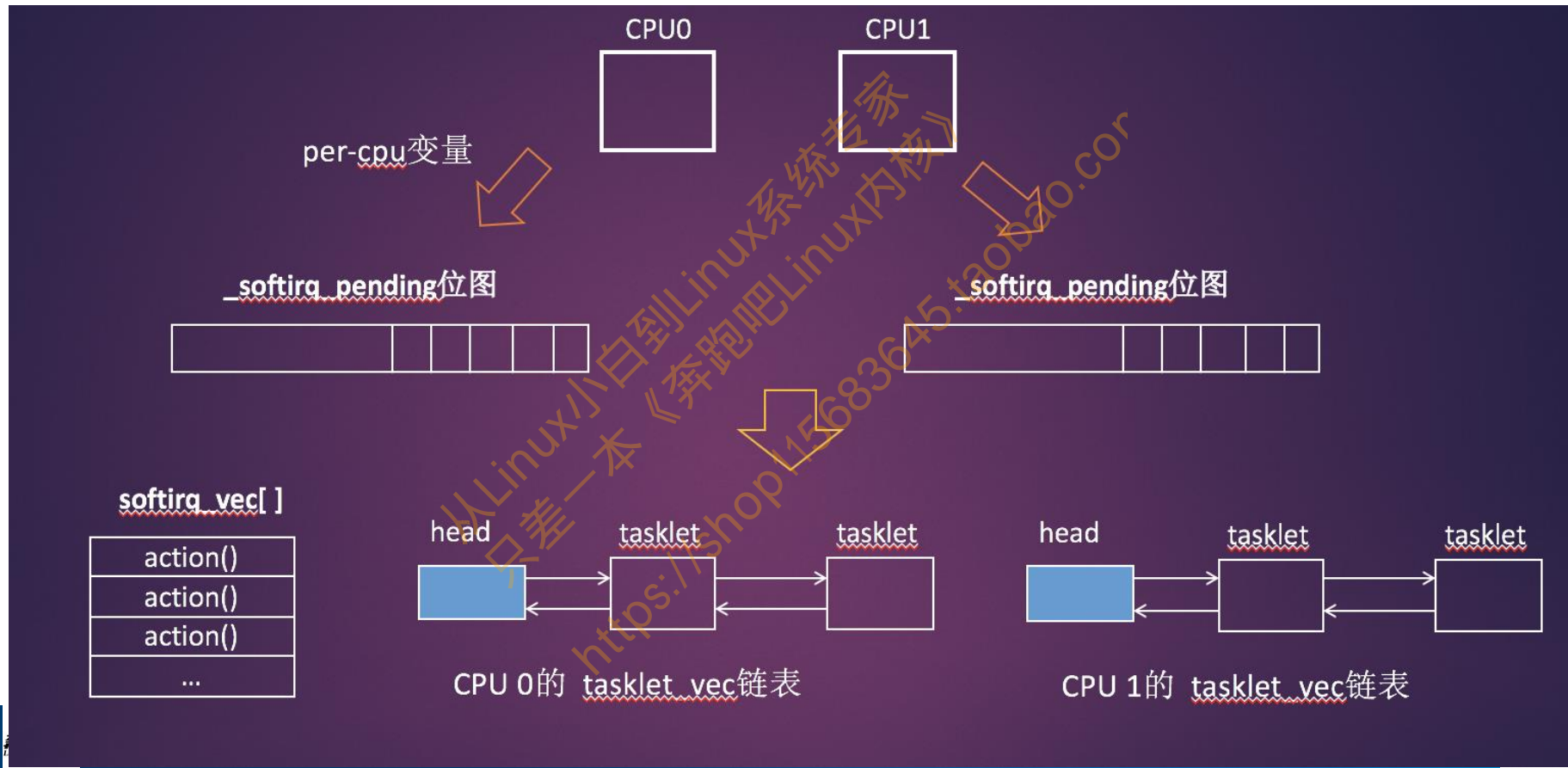
```
enum  
{  
    HI_SOFTIRQ=0,  
    TIMER_SOFTIRQ,  
    NET_TX_SOFTIRQ,  
    NET_RX_SOFTIRQ,  
    BLOCK_SOFTIRQ,  
    BLOCK_IOPOLL_SOFTIRQ,  
    TASKLET_SOFTIRQ,  
    SCHED_SOFTIRQ,  
    HRTIMER_SOFTIRQ,  
    RCU_SOFTIRQ,  
  
    NR_SOFTIRQS  
};
```

软中断小结

- 软中断类型是静态定义的，建议不新建软中断类型
- 软中断的回调函数是在开中断下执行的
- 软中断的执行点：在硬件中断处理函数返回之前
- 软中断算中断上下文，因此软中断总是抢占 进程上下文。
- 同一类型的软中断，可以在多CPU上并行执行

从Linux小白到Linux大神
只差一本《奔跑吧Linux》
<https://shop115683645.taobao.com/>

Tasklet机制



如何在驱动中实现一个tasklet?

- 要初始化一个tasklet, 比如可以通过静态定义或者动态定义。通常可以在你的驱动代码里面的数据结构中定义一个tasklet。
- 实现你的tasklet的handler函数。
- 调度自己的tasklet。

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

tasklet总结

- tasklet基于 软中断实现的
- tasklet是串行执行的
- 驱动开发者应该使用tasklet而不是软中断

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

中断上下文

- 什么是中断上下文?
- 什么是进程上下文?
- 中断上下文包括哪些?
 - ✓ 硬中断上下文
 - ✓ 软中断上下文
 - 软中断和tasklet
 - ksoftirqd内核线程
 - 使能BH, local_bh_enable()

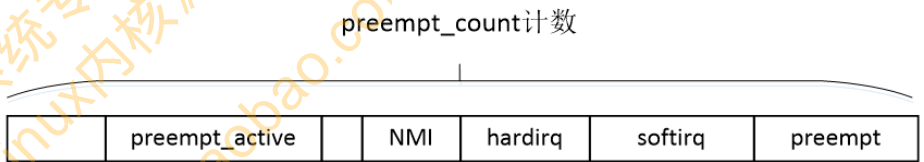


图5.6 preempt_count计数

workqueue机制

- 工作队列，workqueue机制
- 本质上就是把中断里需要延后处理的事情，放到进程上下文来执行
- 工作队列的基本原理是把work（需要推迟执行的函数）交由一个内核线程来执行

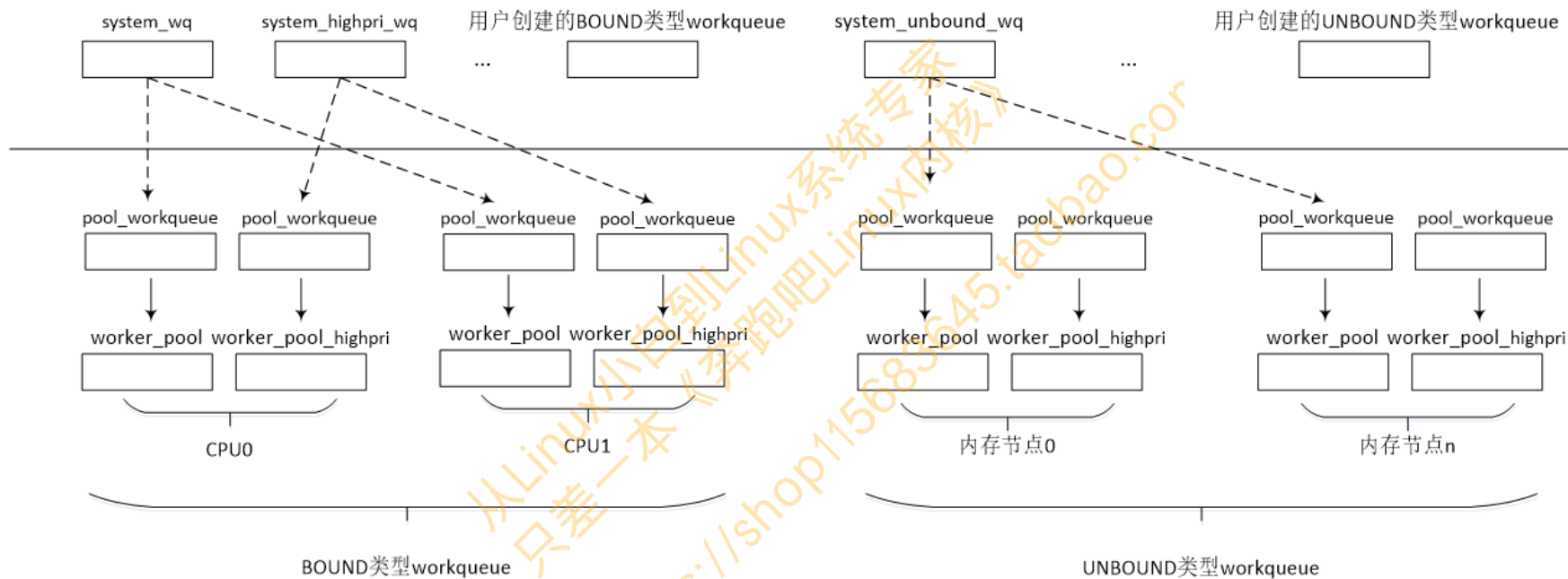
从Linux小白到Linux系统内核专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

新的工作队列机制CMWQ

- 旧的工作队列机制：
 - ✓ 内核线程数量太多
 - ✓ 并发性比较差
 - ✓ 死锁问题
- 新的解决办法：concurrency-managed workqueues (CMWQ)

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683643.taobao.com>

工作池



workqueue使用三部曲1

- 第一部曲：使用alloc_workqueue()创建新的workqueue
- 若使用系统默认的workqueue，可以省略创建workqueue

```
[include/linux/workqueue.h]

#define alloc_workqueue(fmt, flags, max_active, args...) \
    __alloc_workqueue_key((fmt), (flags), (max_active), \
        NULL, NULL, ##args)
```

workqueue使用三部曲2

- 第二部曲：使用INIT_WORK()宏声明一个work和该work的回调函数。

```
[include/linux/workqueue.h]
```

```
#define INIT_WORK(_work, _func)
```

```
__INIT_WORK((_work), (_func), 0)
```

- 写work的回调函数
- void worker_handler(void *data)

workqueue使用三部曲3

- 第三部曲：在新workqueue上调度一个work：queue_work()
- 如果使用系统默认的工作队列，可以使用schedule_work()来调度一个work

```
[include/linux/workqueue.h]

static inline bool schedule_work(struct work_struct *work)
{
    return queue_work(system_wq, work);
}
```

workqueue使用小结

- 使用系统默认的工作队列system_wq，步骤如下。
 - ✓ 使用INIT_WORK()宏声明一个工作和该工作的回调函数。
 - ✓ 调度一个工作：schedule_work()。
 - ✓ 取消一个工作：cancel_work_sync()。
- 若想创建一个专门的工作队列，步骤如下。
 - ✓ 使用alloc_workqueue()创建新的工作队列。
 - ✓ 使用INIT_WORK()宏声明一个工作和该工作的回调函数。
 - ✓ 在新工作队列上调度一个工作：queue_work()。
 - ✓ 驱动退出之前flush工作队列上所有工作：flush_workqueue()。

实验

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

实验1： tasklet

➤ 实验目的

- ✓ 了解和熟悉Linux内核的tasklet机制的使用。

➤ 实验步骤

- ✓ 1) 写一个简单的内核模块，初始化一个tasklet，在write()函数里调用该tasklet回调函数，在tasklet回调函数中输出用户程序写入的字符串。
- ✓ 2) 写一个应用程序，测试该功能。

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

实验2：工作队列

➤ 实验目的

- ✓ 通过本实验了解和熟悉Linux内核的工作队列机制的使用。

➤ 实验步骤

- ✓ 1) 写一个简单的内核模块，初始化一个工作队列，在write()函数里调用该工作队列回调函数，在回调函数中输出用户程序写入的字符串。
- ✓ 2) 写一个应用程序，测试该功能。

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

实验3：定时器和内核线程

➤ 实验目的

- ✓ 通过本实验了解和熟悉Linux内核的定时器和内核线程机制的使用。

➤ 实验步骤

- ✓ 写一个简单的内核模块，首先定义一个定时器来模拟中断，再新建一个内核线程。当定时器到来时，唤醒内核线程，然后在内核线程的主程序中输出该内核线程的相关信息，如PID、当前jiffies等信息。

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》
<https://shop115683645.taobao.com>

BACKUP

从Linux小白到Linux系统专家
只差一本《奔跑吧Linux内核》

<https://shop115683645.taobao.com>

中断管理面试必考20题

- 问题1：为啥要有中断？
- 问题2：在ARM v7处理器里，如果一个中断发生了，处理器为我们做了些什么？
- 问题3：在ARM处理器里，中断发生了，那本地CPU的中断是什么时候关闭的，又是在什么时候被开启的？
- 问题4：ARMv7和ARMv8的中断向量表是怎么样的？
- 问题5：外设中断来了，我们软件（OS）需要做些什么事情？
- 问题6：中断发生，软件如何去保存中断现场？
- 问题7：硬件中断号和软件中断号区别？它们是什么关系，又怎么勾搭上的？
- 问题8：请讲一下Linux内核里注册中断是怎么回事？
- 问题9：在中断处理函数里，发生了调度，会怎么样？请画图来阐述。Linux是怎么规避的？
- 问题10：为什么中断处理要分上半部和下半部？

中断管理面试必考20题

- 问题11：中断线程化是怎么回事？
- 问题12：请描述一下软中断的实现原理？
- 问题13：软中断的回调函数里是否允许响应本地中断？
- 问题14：什么是中断上下文？什么是软中断上下文？
- 问题15：为什么软中断上下文总是抢占进程上下文？
- 问题16：怎么判断当前是否在中断上下文或者软中断上下文？
- 问题17：软中断上下文是否允许睡眠？
- 问题18：workqueue是工作在中断上下文还是进程上下文？回调函数允许睡眠吗？
- 问题19：这么多下半部机制该如何选择？
- 问题20：请问，你在调试中断时候遇到过哪些坑？

想听笨叔详细解答中断面试必考20题，请订阅奔跑吧第二季旗舰篇视频

中断管理4

那些狗日的面试题目

笨叔叔



shop115683645.taobao.com

Linux视频课程



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



规划中

奔跑吧
Linux社区

旗舰篇一次订阅，持续更新

- | | |
|-----|----------------------|
| 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

奔跑2.1 代码导读C语言部分（二）	0:21:28
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

奔跑2.11 page数据结构	0:25:41
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！



扫码识别

微店二维码



淘宝店二维码



微信号: Runing-LinuxKernel

《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

<https://shop11538560.com>