主要内容：

* 什么是中断
* 中断类型
* 中断相关函数
* 中断处理机制
* 中断控制方法
* 总结

### **1. 什么是中断**

没有中断的话，CPU和外围设备之间协同工作可能只有轮询这个方法：CPU定期检查硬件状态，需要处理时就处理，否则就跳过。

中断机制是硬件在需要的时候向CPU发出信号，CPU暂时停止正在进行的工作，来处理硬件请求的一种机制。

### **2. 中断类型**

中断一般分为异步中断(一般由硬件引起)和同步中断(一般由处理器本身引起)。

异步中断：CPU处理中断的时间过长，所以先将硬件复位，使硬件可以继续自己的工作，然后在适当时候处理中断请求中耗时的部分。

举个例子：网卡的工作原理

网卡收到数据包后，向CPU发出中断信号，请求处理接收到的数据包

CPU将收到的数据包拷贝到内存后，即通知网卡继续工作

至于数据包拷贝至内存后的处理会在适当的时候进行

同步中断：CPU处理完中断请求的所有工作后才反馈硬件

举个例子：系统异常处理(比如运算中的除0操作)

应用程序出现异常后，需要内核来处理

内核调用相应的异常处理函数来处理异常

处理完后终了应用程序或者给出message

### **3. 中断相关函数**

实现一个中断，主要需要知道3个函数：

* 注册中断的函数
* 释放中断的函数
* 中断处理程序的声明

#### **3.1 注册中断的函数**

    位置：<linux/interrupt.h>  include/linux/interrupt.h

定义如下：

/\*

 \* irg     - 表示要分配的中断号（也称中断线）

 \* handler - 实际的中断处理程序

 \* flags   - 标志位，表示此中断的具有特性

 \* name    - 中断设备名称的ASCII 表示，这些会被/proc/irq和/proc/interrupts文件使用

 \* dev     - 用于共享中断线，多个中断程序共享一个中断线时(共用一个中断号)，依靠dev来区别各个中断程序

 \* 返回值：

 \* 执行成功：0

 \* 执行失败：非0

 \*/

int request\_irq(unsigned int irq,

                irq\_handler\_t handler,

                unsigned long flags,

                const char \*name,

                void \*dev)

#### **3.2 释放中断的函数**

定义比较简单：

void free\_irq(unsigned int irq, void \*dev)

如果不是共享中断线，则直接删除irq对应的中断线。

如果是共享中断线，则判断此中断处理程序是否中断线上的最后一个中断处理程序，

    是最后一个中断处理程序 -> 删除中断线和中断处理程序

    不是最后一个中断处理程序 -> 删除中断处理程序

#### **3.3 中断处理程序的声明**

声明格式如下：

/\*

 \* 中断处理程序的声明

 \* @irp  - 中断处理程序(即request\_irq()中handler)关联的中断号

 \* @dev  - 与 request\_irq()中的dev一样，表示一个设备的结构体

 \* 返回值：

 \* irqreturn\_t -  执行成功：IRQ\_HANDLED  执行失败：IRQ\_NONE

 \*/

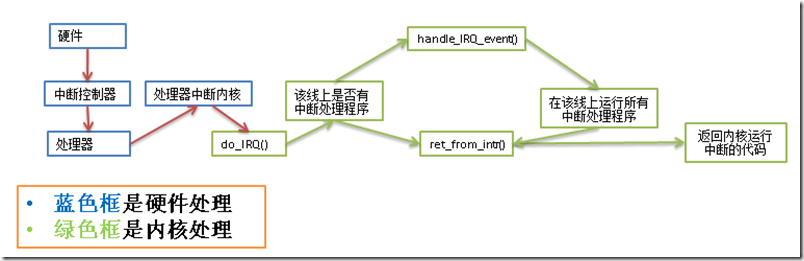
static irqreturn\_t intr\_handler(int, irq, void \*dev)

### **4. 中断处理机制**

中断处理的过程主要涉及3函数：

* do\_IRQ 与体系结构有关，对所接收的中断进行应答
* handle\_IRQ\_event 调用中断线上所有中断处理
* ret\_from\_intr 恢复寄存器，将内核恢复到中断前的状态

处理流程可以参见书中的图，如下：

[](https://images0.cnblogs.com/blog/83005/201304/19113723-21463cd0f94f455480d282ac1b38138d.png)

### **5. 中断控制方法**

常用的中断控制方法见下表：

|  |  |
| --- | --- |
| **函数** | **说明** |
| local\_irq\_disable() | 禁止本地中断传递 |
| local\_irq\_enable() | 激活本地中断传递 |
| local\_irq\_save() | 保存本地中断传递的当前状态，然后禁止本地中断传递 |
| local\_irq\_restore() | 恢复本地中断传递到给定的状态 |
| disable\_irq() | 禁止给定中断线，并确保该函数返回之前在该中断线上没有处理程序在运行 |
| disable\_irq\_nosync() | 禁止给定中断线 |
| enable\_irq() | 激活给定中断线 |
| irqs\_disabled() | 如果本地中断传递被禁止，则返回非0；否则返回0 |
| in\_interrupt() | 如果在中断上下文中，则返回非0；如果在进程上下文中，则返回0 |
| in\_irq() | 如果当前正在执行中断处理程序，则返回非0；否则返回0 |

### **总结**

中断处理对处理时间的要求很高，如果一个中断要花费较长时间，那么中断处理一般分为2部分。

上半部只做一些必要的工作后，立即通知硬件继续自己的工作。

中断处理中耗时的部分，也就是下半部的工作，CPU会在适当的时候去完成。