# Linux 中断驱动总结（单个与多个中断处理）

## 一、什么是 Linux 中断？

Linux 中断是一种异步事件机制，当硬件发生事件时，通过中断控制器通知 CPU ，请求执行已经注册好的中断服务函数。

## 二、中断服务函数是回调函数？

### 是。

* 因为它不是自己直接调用，而是把地址传给系统，系统事件发生时会“回调”调用它
* 原型：

irqreturn\_t my\_isr(int irq, void \*dev\_id);

## 三、中断注册相关函数 & 结构体

### 1. request\_irq()

int request\_irq(unsigned int irq,  
 irq\_handler\_t handler,  
 unsigned long flags,  
 const char \*name,  
 void \*dev);

| 参数 | 含义 |
| --- | --- |
| irq | 需注册的中断号，可用 gpio\_to\_irq() 转换 |
| handler | 中断处理函数（ISR） |
| flags | 触发方式：IRQF\_TRIGGER\_FALLING 等 |
| name | 在 /proc/interrupts 中显示名 |
| dev | 传给 handler 的私有数据指针，通常是设备结构 |

### 2. free\_irq()

释放中断：

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

需要和 request\_irq 中使用的 dev 一致，才能正常释放

### 3. irq\_handler\_t 类型

定义于 <linux/interrupt.h>：

typedef irqreturn\_t (\*irq\_handler\_t)(int, void \*);

## 四、单个 GPIO 中断处理

### 1. gpio\_to\_irq()

将 GPIO 转换成中断号：

int gpio\_irq = gpio\_to\_irq(PAD\_GPIO\_A+28);

### 2. 注册

request\_irq(gpio\_irq, mykey\_irq\_handler, IRQF\_TRIGGER\_FALLING, "key\_irq", NULL);

### 3. 处理函数

static irqreturn\_t mykey\_irq\_handler(int irq, void \*dev\_id)  
{  
 printk(KERN\_INFO "key interrupt\n");  
 return IRQ\_HANDLED;  
}

## 五、多 GPIO 中断处理（如你编写的代码）

### 1. 自定义结构体：

struct gpio\_irq\_t {  
 unsigned int irq\_n;  
 irq\_handler\_t fun;  
 unsigned long irq\_f;  
 const char \*name;  
 void \*dev;  
};

### 2. 列表化多个 GPIO

static struct gpio\_irq\_t keys\_irq[] = {  
 {PB\_PIO\_IRQ(PAD\_GPIO\_A+28), handler, IRQF\_TRIGGER\_FALLING, "a28", NULL},  
 {PB\_PIO\_IRQ(PAD\_GPIO\_B+30), handler, IRQF\_TRIGGER\_FALLING, "b30", NULL},  
 ...  
};

### 3. 注册循环

for (i = 0; i < ARRAY\_SIZE(keys\_irq); i++) {  
 request\_irq(keys\_irq[i].irq\_n, keys\_irq[i].fun, keys\_irq[i].irq\_f,  
 keys\_irq[i].name, keys\_irq[i].dev);  
}

### 4. 处理函数

static irqreturn\_t handler(int irq, void \*dev\_id) {  
 int i;  
 for (i = 0; i < ARRAY\_SIZE(keys\_irq); i++) {  
 if (irq == keys\_irq[i].irq\_n)  
 printk("%s triggered\n", keys\_irq[i].name);  
 }  
 return IRQ\_HANDLED;  
}

## 六、中断触发时机规则

* 一次注册，再多个小时内按键，中断会一直有效
* IRQF\_TRIGGER\_FALLING 为下降沿触发（默认按键接地或下拉时触发）
* 不是当时就错过，而是在注册后任何时间都可以触发

## 七、需要注意的点

| 编程点 | 说明 |
| --- | --- |
| GPIO 必须配置为输入 | 否则不会触发中断 |
| 硬件有一定投线要求 | 下拉拥有性最好 |
| 同一中断号可被多个 driver 分享（dev\_id 用于区分） | 需配 IRQF\_SHARED |
| request\_irq 和 free\_irq 的 dev\_id 必须匹配 | 否则释放不了 |
| 中断不能做太大操作 | 常用 tasklet/workqueue 延迟处理 |

## 八、总结

* 中断是一种异步回调机制
* ISR 是回调函数，根据中断号同一处理函数可区分多个 GPIO
* 通过结构体列表，可编写多 GPIO 处理应用
* 无论是单个还是多个中断，主要思想是“注册后触发时依据 irq 分析”