# Linux中断处理程序设计

## 中断概念

[中断](http://baike.baidu.com/item/%E4%B8%AD%E6%96%AD/3933007)指当出现需要时，CPU暂时停止当前程序的执行转而执行处理新情况的程序和执行过程。

为什么需要中断？

1. 外设的处理速度一般慢于CPU
2. CPU不能一直等待外部事件

所以设备必须有一种方法来通知CPU它的工作进度，这种方法就是中断。

## 中断实现

在Linux驱动程序设计中，为设备实现一个中断包含两个步骤：

1. 向内核注册中断
2. 实现中断处理函数

## 中断注册

request\_irq用于实现中断的注册功能：

int request\_irq(unsigned int irq,

void (\*handler)(int, void\*, struct

pt\_regs \*),

unsigned long flags,

const char \*devname,

void \*dev\_id)

返回0为成功，或者返回一个错误码

unsigned int irq 中断号

void (\*handler)(int,void \*,struct pt\_regs \*) 中断处理函数

unsigned long flags 与中断管理相关的各种选项

const char \* devname 设备名

void \*dev\_id 共享中断时使用

中断标志

IRQF\_DISABLED （SA\_INTERRUPT） 如果设置该位，表示为快速中断；如果没有设置该位，那么就是个慢速中断

IRQF\_SHARED （SA\_SHIRQ） 该位表明中断可以在设备间共享

快速中断与慢速中断

快速中断不会被其他中断打断；而慢速中断会被其他中断打断。

中断处理程序就是普通的C代码。特别之处在于它是在中断上下文中运行的，它的行为受到某些限制：

1. 不能向用户空间发送或者接收数据
2. 不能使用可能引起阻塞的函数
3. 不能使用可能引起调度的函数

中断处理函数流程

void short\_sh\_interrupt(int irq, void \*dev\_id, struct pt\_regs \*regs)

{

/\* 判断是否是本设备产生了中断(为什么要做这样的检测?) \*/

value = inb(short\_base);

if (!(value & 0x80)) return;

/\* 清除中断位(如果设备支持自动清除,则不需要这步) \*/

outb(value & 0x7F, short\_base);

/\* 中断处理,通常是数据接收\*/

。。。。。。。。。

/\*唤醒等待数据的进程\*/

wake\_up\_interruptible(&short\_queue);

}

释放中断

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

## 中断分层

中断可分为上半部与下半部。

上半部：当中断发生时，它进行对应的硬件读写，并登记该中断。

下半部：在系统空闲的时候对上半部登记的中断进行后续处理。

工作队列是一种将任务推后执行的形式，它把推后的任务交由一个内核线程去执行。这样下半部会在进程上下文执行，它允许重新调度甚至睡眠。每个被推后的任务叫做工作，由这些工作组成的队列称为工作队列。

工作队列执行步骤：

1、创建工作队列 create\_workqueue

2、创建工作 INIT\_WORK

3、提交工作 queue\_work 或者schedule\_work