Linux 平台下中断的实现

The Relization of Interruption Based On Linux Platform

李学忠　（武汉工业学院 武汉 430023）
阳道善　（华中科技大学 武汉 430074）

摘要：中断机制是数控系统实现实时控制的基础，本文介绍了 Linux 平台下中断的产生和处理过程，并通过一个实例进行了说明。

关键词：中断 Linux

1　引言

基于 PC 机的数控系统是数控系统的发展趋势。基于 DOS 的数控系统经过十几年的发展，技术已经成熟，但由于 DOS 本质上是一个单用户的操作系统，因而不能达到资源共享，并且其程序长度受限于 640K 的内存，不适于较复杂的数控程序的运行；Linux 是一个多用户、多任务的操作系统，具有良好的开放性和可靠性，在 Linux 平台下开发数控系统具有重要意义。数控系统控制软件的重要特征之一是实时中断处理，数控系统的多任务性和实时性决定了系统中断成为整个系统必不可少的重要组成部分。那么，在 Linux 平台下，中断的产生和处理过程是怎样的呢？

在 PC 中，外部设备的中断通过 Intel 8259A 可编程中断控制器（PIC）来管理。PC 机采用两片 8259A 控制中断，CPU 通过一组 I/O 端口控制 8259A，而 8259A 则通过 INTR 管脚给 CPU 传送中断信号。在 CPU 中断程序中，每个中断都有一个中断类型号，而且每个中断对应一个中断处理程序的地址，这些地址组成一张中断向量表。当一个外部中断发生时，系统硬件决定 CPU 从该中断向量表中取出该中断事件对应的中断处理程序的地址，然后转移到该地址执行中断服务程序。

2　Linux 中断机制

Linux 在 X86 保护模式下，系统用中断描述符表（IDT）来建立中断向量。中断描述符由中断门组成，中断门数据的结构如图 1 所示。

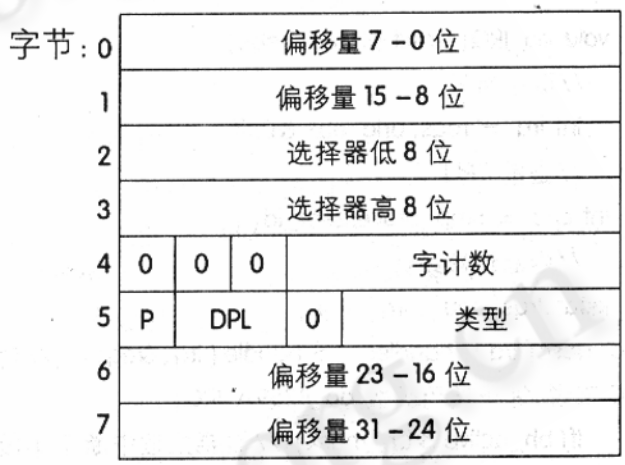

图 1　中断门的数据结构

图中，DPL 是任务能存取该门的最低特权级，即任务的特权级必须高于门的特权级。P 是有效位，P 为 0 表示描述符内容无效，为 1 表示有效。字计数的值为 0 – 31，表示从调用者堆栈复制到所调用的过程堆栈的参数数量（这里参数是 32 位的量）。偏移量是指相对于段基址的偏移量，这是目标码的入口点，段基地址由选择器指出，选择器实际上是一个 16 位的寄存器。

Linux – i386 系统在中断时，硬件平台会将下述寄存器压入堆栈：

```
push ss
push esp
push eflags
push cs
push eip
```

然后按中断向量地址调用中断函数。对于通用

PC，Linux 有 16 个外部中断向量。可以认为 Linux 的所有外部中断向量是一样的，其指向的地址等同于：

```
LinuxInterrupt:
    push irq-256 ;压入堆栈一个负数，使内核能够区别是中断调用及其中断号，
    jum common_interrupt ;跳转到通用中断代码
```

common_interrupt 即为通用的中断处理代码：

```
common_interrupt:
    SAVE ALL ;保存所有寄存器
    call do_IRQ ;调用 do_IRQ( )
    jmp ret_from_intr; 跳转 ret_from_intr( )
```

其中 do_IRQ( ) 函数是中断处理的核心，它的定义如下：

```
void do_IRQ( struct pt_regs regs)
{ //取中断号
  int irq = regs. orig_eax &0xff;
  //当前 CPU
int cpu = smp_processsor_id( );
  //中断计数统计
kstat. irqs[ cpu][ irq] + +;
irq_desc[ irq]. handler - > handle( irq, &regs) ;//对 PC 来说，等于调用函数 do_8259A_IRQ( )
  if( bh_active &bh_mask) //需要处理中断下半段 bottom_half
  do_bottom_half( ); //处理下半段
}
```

而该函数的核心是调用 irq_desc[irq]. handler - > handle( )。对于通用 PC 来说，是调用 do_8259A_IRQ( ) 函数。另外，在 Linux 中，给定的中断处理程序从概念上可以被分为上半部分（top half）和下半部分（bottom half）。在中断发生时，上半部分的处理过程立即执行，但是下半部分（如果有的话）却推迟执行，上半部分决定其下半部分是否需要执行，这是通过把上半部分和下半部分处理为独立的函数并对其区别对待实现的。因此，不能推迟执行的显然应放在上半部，但是可以推迟执行的只是可能属于下半部分，如果用户把所有的中断处理代码都放入上半部，将会影响系统的运行效率。

通用中断处理代码中的函数 ret_from_intr( )，是从中断返回的函数，该函数的等价 C 语言形式为：

```
ret_from_intr:
if( 在用户态运行时被中断)
reschedule( ) //调用调度程序
restore_all( ) //恢复寄存器并调用 IRET 指令返回中断
```

因此从总体上说，单个中断处理过程可描述如下：

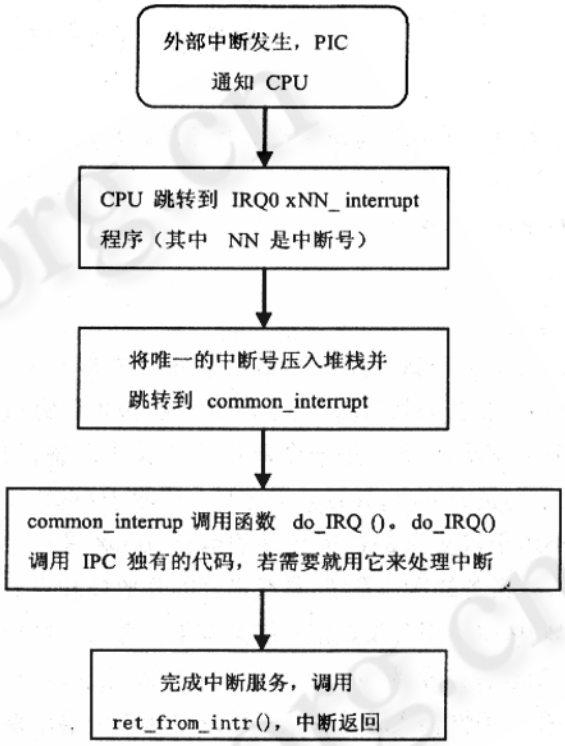

图 2　中断处理流程图

3　Linux 中断的实例

下面我们以定时器中断的工作方式为例，来进一步说明 Linux 中断和下半部分的情况。

Linux 定时器中断的安装可以通过以下几个步骤进行：

（1）确定中断号 irq；

（2）编写中断处理函数；

（3）定义 struct irqaction 变量，并调用 setup_x86_irq( ) 函数（该函数给指定中断号 irq 的中断控制队列 irq_desc[irq]. action 增加一个元素）安装中断处理程序；如果不定义 struct irqaction 变量，则必须调用 request_irq( ) 函数通过调用 setup_x86_irq( ) 函数来安装中断处理程序。

因此，我们首先定义定时器中断处理函数 timer_

interrup( ),然后定义 struct_irqaction 变量:

```
static struct irqaction irq( ) = {timer_interrupt,SA_
INTERRUPT,0"timer",NULL,NULL};
```

最后在系统初始化调用函数 time_init( ) 时,调用 setup_x86_irq( ) 函数设备定时器中断响应函数:

```
_initfunc(void time_init(void))
{
.....
setup_x86_irq(0,&irq0);//调用本函数之后,定时器将开始工作。
.....
}
```

以下是定时器中断响应函数的定义:

```
static void timer_interrupt(int irq,void * dev_id,
struct pt_regs * regs)
{
    int count;
    write_lock(&xtime_lock);
    do_timer_interrupt(irq,NULL,REGS); //该函数定义见下文
    write_unlock(&xtime_lock);
}
    static inline void do_timer_interrupt(int irq,void
* dev_id,struct pt_egs * regs)
{
    do_timer(regs); //该函数定义见下文
    if(! user_mode(regs)) //系统态时,进行代码运行统计
        x86_do_profile(regs -> eip);
}
void do_timer(struct pt_regs * regs)
{ //系统时间计数器
  (*(unsigned long *)&jiffies) + +;
  lost_ticks + +;
  mark_bh(TIMER_BH); //标志下半部分必须执行
if(! user_mode(regs)) //系统态
        lost_ticks_system + +;
      if(tq_timer) //标志定时器下半部分必须执行
        mark_bh(TQUEUE_BH);
}
```

4　结束语

目前,Linux 操作系统因其具有的稳定、健壮和完全开放的特性,在我国得到了快速发展和广泛应用,本文介绍的 Linux 平台下的中断机制,是在该环境下实现数控系统实时中断控制的基础。

参考文献

1 (美) Scott Maxwell 著,Linux 内核源代码分析,冯锐 等译,机械工业出版社,2000.

2 陈莉君,Linux 操作系统内核分析,北京:人民邮电出版社,2000.