目录

[3.1 X86CPU对中断的硬件支持 1](#_Toc399661278)

[3.2 中断向量表IDT的初始化 1](#_Toc399661279)

[3.3 中断请求队列的初始化 1](#_Toc399661280)

[3.4 中断的响应和服务 2](#_Toc399661281)

[3.5 软中断与Bottom Half 2](#_Toc399661282)

[3.6 页面异常的进入和返回 3](#_Toc399661283)

[3.8 系统调用 3](#_Toc399661284)

3.1 X86CPU对中断的硬件支持  
 Intel中的中断向量表项为“门”，这些“门”不光为中断而设，只要想切换CPU的运行状态，即优先级别，都要通过这个门。

“任务状态段”TSS用来保存任务运行“现场”的数据结构，包括CPU中所有与具体进程有关的寄存器的内容（包括页面目录指针CR3），还包括三个堆栈指针。其中“任务寄存器”TR，用来指向当前任务的TSS

中断门进入中断服务程序时cpu会自动将中断关闭，而陷阱门进入时不会将中断关闭

## 3.2 中断向量表IDT的初始化

当中断是由外部产生或者是CPU异常产生，中断门的DPL忽略不顾set\_system\_gate()

set\_trap\_gate()设置的都是陷阱门，而前者的DPL为3，用户可以在用户空间通过“int 0x80”进行系统调用。

系统初始化时，在trap\_init()中设置了一些为CPU保留专用的IDT表项以及系统调用所用的陷阱门以后，就通过init\_IRQ()设置大量用于外设的通用中断门了

interrupt[]数组是一个函数指针数组，依次是IRQ0x00\_interrupt，IRQ0x01\_interrupt……

该函数在将中断向量相关的数值压入堆栈后就进入公共的程序common\_interrupt中。

IRQ0x00\_interrupt是一个时钟中断服务程序队列，一旦时钟中断开始，进程调度也随之开始，因此，要完成对进程调度的初始化才将时钟中断的服务程序挂上去。

## 3.3 中断请求队列的初始化

中断向量表IDT有两种表项：一种是保留专用于CPU本身的中断门，主要用于CPU产生的异常以及用户程序通过int指令产生的中断。另一种就是从0x20开始的224项用于外设的通用中断门。二者的区别在于通用中断门可以为多个中断源共享。

对于外设的中断门，系统为每个中断向量设置一个队列，而根据每个中断源使用的中断向量，将中断服务程序挂在相应的队列中，而数组irq\_desc[]中的每一个元素都是这样的队列头以及控制结构。

irq\_desc\_t中的指针action用来维持一个中断服务程序描述项的单链队列，指针handler用于对“中断通道”的控制，具体的函数取决于中断控制器。

在IDT初始化完成之初，每个中断服务队列都是空的，真正的中断服务要到具体设备的初始化程序将其中断服务程序通过request\_irq()向系统登记，挂入某个中断请求队列以后才发生。

## 3.4 中断的响应和服务

当发生外部中断时，由于当前远行级别CPL为3，中断服务程序的运行级别DPL为0。所以CPU要从TR寄存器所指的当前TSS中取出用于内核的堆栈指针，即当前进程的系统空间堆栈，且总为空。当CPU接入IRQ0X03\_interrupt时，当前进程的系统空间堆栈就会有EFLAGS以及返回地址。

在IRQ0X03\_interrupt压入系统堆栈的是中断请求号-256是由于在因系统调用而进入内核时这个位置要用来存放系统调用号，而系统调用和中断服务又共用一部分子程序。

判断一个数大于小于0只需要一条指令，而与一个常数比较，至少要多访问一次内存。

common\_interrupt在保存完所有寄存器的内容后，将返回地址ret\_from\_intr压入堆栈，然后就跳到do\_IRQ()中，放佛对do\_IRQ()的调用就发生在ret\_from\_intr地址处第一条指令之前，模拟了从ret\_from\_intr处的一次函数调用。

关于系统堆栈的内容：前面所做的一切，包括CPU在进入中断时自动做的，实际上都是在为do\_IRQ()建立一个模拟的子程序调用环境，使得在do\_IRQ()中既可以方便的知道中断前夕各寄存器的内容，也可以在执行完毕返回到ret\_from\_intr，并且从那里执行中断返回。

do\_IRQ()在从栈中找到中断号后，将中断通道描述符中的IRQ\_PENDING设为1，

如果中断请求队列的服务是关闭的(IRQ\_DISABLED为1)，或者IRQ\_INPROGRESS标志位为1，或者队列是空的，那么action为NULL，无法执行，只好返回，但是在这几种情况下，IRQ\_PENDING位会设为1，以后当开启队列的服务时，会看到这个标志位而补上一次服务。称为IRQ\_REPLAY

对于中断程序而言，对于同一个CPU不允许中断服务嵌套，而对于不同CPU而言不允许并发地进入同一个中断服务程序。

在do\_IRQ()中的handle\_IRQ\_event()会判断中断通道是否共享，如果共享会重新开启中断，并将服务队列的函数依次跑一遍。

## 3.5 软中断与Bottom Half

在内核中，softirq\_init对为bh机制设置的数据结构bh\_task\_vec[]完成了初始化，bh函数的执行是建立在tasklet的基础上完成的，不过添加了一些限制。

内核中为软中断设置了一个以“软中断号”为下标的数组softirq\_vec[]，类似于中断机制的irq\_desc[]，这是一个全局量，系统中每个CPU看到的是同一个数组。但是每个CPU各有其自己的“软中断控制/状况结构”，所以这些数据结构形成以CPU编号为下标的irq\_stat[]。

对执行bh函数的要求是在哪一个CPU提出来的，就把它调度到哪一个CPU上执行

内核在do\_IRQ()完成一个通道的中断程序以后，以及每当从系统调用返回时，都要检查是否有软中断请求在等待。

需要执行一个特定bh函数时，通过mark\_bh()->tasklet\_hi\_schedule(bh\_task\_vec+nr)将bh\_task\_vec[nr]即tasklet\_struct结构链入tasklet\_hi\_vec[cpu]中，并通过\_\_cpu\_raise\_softirq()正式发出软中断请求。最终的执行需要等到中断服务程序或系统调用结束后的do\_softirq()

注意：tasklet允许在同一时刻不同tasklet运行在不同CPU上，而bh不允许，因此bh\_action()中有一个全局性量lobal\_bh\_lock，保证只有一个CPU上运行bh

## 3.6 页面异常的进入和返回

在异常发生时，除了将中断发生时需要压入的压入堆栈，如果异常产生错误码，也需要把错误码压入堆栈。对于没有错误码的异常，就压入数值0

时钟中断保证了进程运行一段时间后能够被强制调度

系统时间是指内核中的两个全局量：一个是数据结构xtime由get\_cmos\_time()从CMOS中获取从历史上某一时刻开始的时间的“绝对值”；另一个是jiffies，记录开机以来时钟中断的次数。

jiffies++ (\*(unsigned long \*)&jiffies)++ 后者可以在一条指令中完成，保证了操作的原子性

time\_interrupt()->do\_time\_interrupt()->do\_timer->mark\_bh将bh\_task\_vec[TIMER\_BH]挂入tasklet\_hi\_vec队列中，使CPU在中断返回之前执行与TIMER\_BH对应的函数timer\_bh()

timer\_bh()第一件事是update\_wall\_time()，处理xtime；第二件事是calc\_load()，目的是计算CPU负荷的统计信息；最后就是run\_timer\_list()，检查系统中各个“计时器”，如果到点就执行相应的函数

## 3.8 系统调用

Linux内核在系统调用时是通过寄存器来传递参数

系统调用返回时，在\_\_syscall\_error中，%eax中的返回值会保存入全局变量errono，并将其自身内容改为-1，这也是大部分系统调用返回值的约定。

进入系统内核时，首先将%eax即系统调用号压入orig\_ax位置，由于SAVE\_ALL保存的寄存器数量有限，实际上只有5个寄存器可以用来传递参数。

当检查到进程的PT\_TRACESYS位为1时，，在服务程序之前和之后都要调用syscall\_trace()，向父进程报告具体系统调用的返回与进入。

最后会通过%eax系统调用号为下标在系统调用跳转表找到对应的函数，不支持的系统调用号均指向sys\_ni\_syscall()

由于传递的参数经常是指针，所以从用户空间到系统空间之间的复制数据操作很重要。

copy\_from\_user()一般情况下通过\_\_generic\_copy\_from\_user()下的\_\_copy\_user\_zeroing()来完成。复制时先按长整数进行，然后对于剩余的部分（不超过3个）字节按字节进行。按理说要检查待复制页面的“合法”性，可是由于发生概率小，极其影响效率，故将可能的错误交给页面异常处理程序了。发生这样的异常时，如果能够在“异常表”中找到发生异常的指令所在地址，并得到相应的“修复”地址，就将异常返回重新执行的地址换为“修复”地址。

GUN的gcc和ld除了text和data段还支持两个段。一个是fixup，专门用于异常发生后的修复，另一个是\_\_ex\_table，专门用于异常地址表。