《操作系统》

课程讲稿

第4章 第3讲

中断处理

软件所制

第六章 第3讲 寄存器

**学时：1**学时

**教学目的：**深入学习linux下中断处理的过程……

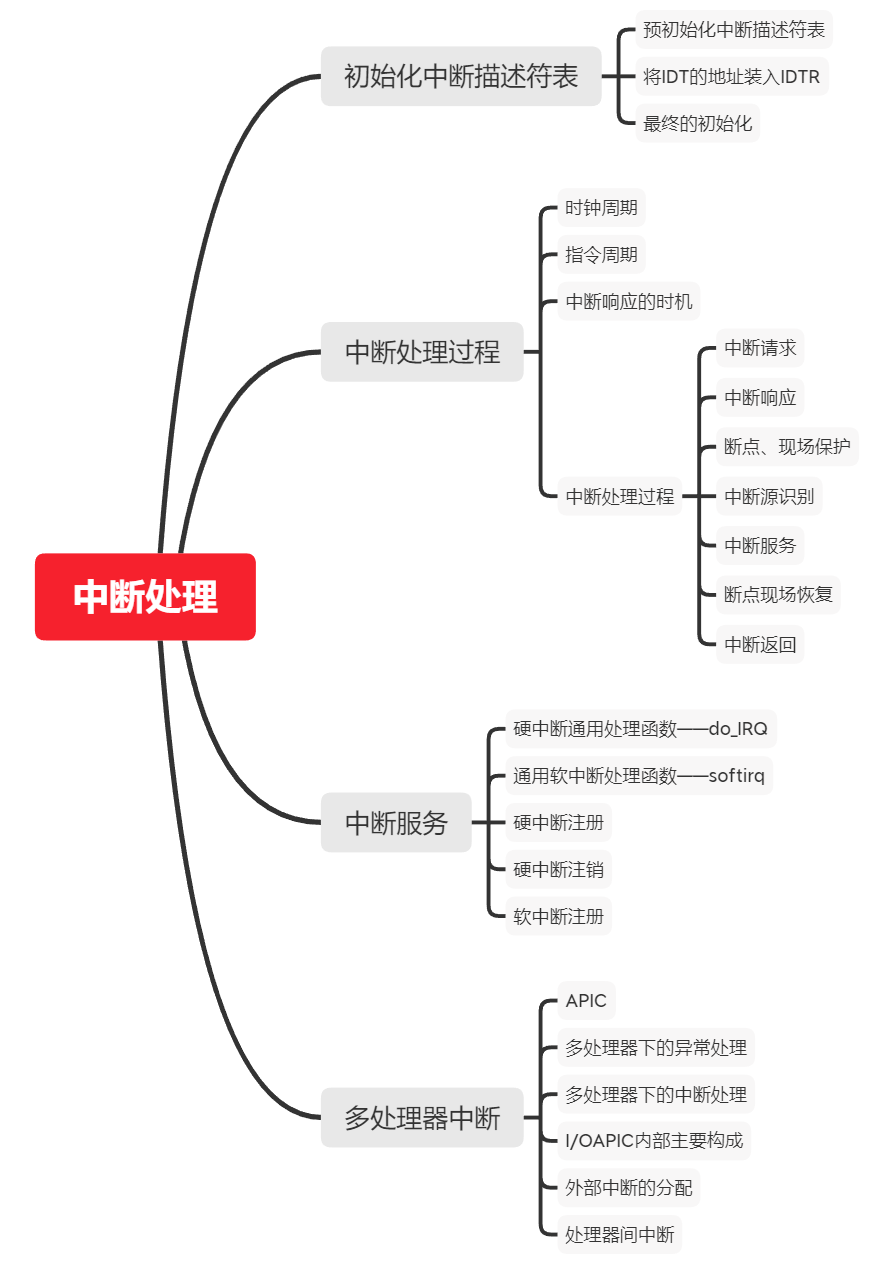
**课程时间线：**



**课外参考读物：**

《LINUX设备驱动程序（第三版）》

**知识框图：**

****

**PPT讲稿：**

1. 今天我们来学习中断机制中最重要的中断处理部分。
2. 中断处理这个部分学习完基本上就掌握了整个中断机制的工作流程了，因此这一节是本章的重中之重。
3. 本节中我们将分为四块内容来介绍中断处理，分别是初始化中断描述符表、中断处理过程、中断服务和多处理器中断，我们先来看看初始化过程。
4. 整个中断的过程是一个又长又复杂的过程，为了让中断机制正常运行，首先就要对这个机制进行初始化，其中首先就要对中断描述符表进行初始化。整个IDT表的初始化过程主要分为三个部分，
   1. 预初始化中断描述符表：填充0~255个中断描述符表的内容，这些内容只在整个初始化的过程中有效，但是在中断机制的正常运行中是会被替换掉的。
   2. 将IDT的地址装入IDTR：使用lidt汇编指令将IDT表的首地址装入idtr寄存器，这样系统就可以快速访问到IDT表了，提升了中断机制的效率。
   3. 最终的初始化：0~255个中断描述符分别指向不同的处理程序，这里就会替换掉第一步所填充的内容。
5. 在整个初始化的过程中涉及到几个重要的数据结构，先来看看最重要的门描述符结构。门描述符就是中断描述符表的表项，它们并不描述某种内存段，而是描述控制转移的入口点，也就是目标代码的门，通过这个门就可以进入目标代码，例如中断处理函数。看一下门描述符的结构，offset\_high,offset\_middle和offset\_low三个字段合起来就是中断处理函数地址在段中的偏移量；segment就是相应的段选择子，根据它在GDT中查找可以最终获取到段基地址；bits是该中断描述符的一些属性值，刚好是16位，每个字段各有代表不同的意义：
   1. ist表示此中断处理函数是使用pre-cpu的中断栈，还是使用IST的中断栈;
   2. Zero字段保留未使用
   3. type表示所中断是何种类型，目前有以下五种：
      1. 中断门
      2. 系统中断门
      3. 陷阱门
   4. dpl用于管理前门可被访问的权限

在结构体定义后面的\_attribute\_(packed)关键字是用于告诉编译器如何布局这个结构体，这样一来门描述符结构体一般就会布局成右图所示这样。

1. 下面就是中断描述符表的定义了，中断描述符表就是很多个门描述符的数组，其中IDT\_ENTRIES宏定义为256，即对应了256个中断向量。下图说明了idtr寄存器和中断描述符表之间的关系，寄存器高32位确定了idt的首地址，而低16位则确定了idt 的表长。
2. 下面就是idt表的初步初始化，在右边这个函数里面，分别将idt表的前32项指向了early\_idt\_handlers\_array这个默认异常处理函数，接着将剩下的32~255个表项都指向early\_ignore\_irq这个默认中断处理函数
3. 接着就使用x86下的LIDT指令将IDT表在内存中的首地址放入了IDTR寄存器，这样在之后的系统运行过程中就可以快速访问到idt表了。
4. 最终会分别进入trap\_init、init\_IRQ、softirq\_init这三个函数，来分贝初始化异常、外部中断和软中断。
5. 接下来我们进入中断处理过程的学习
6. 为了进行后面的学习，我们还要对一些基本的概念进行一下铺垫，比如计算机内部的时钟周期和指令周期。对于时钟周期，我们只需要记住他是cpu进行操作的最小时间单位就行了
7. 指令周期则是值两个指令执行时间的间隔。
8. 在对前两个概念进行了学习之后，我们来思考一下，cpu是在哪个时机才能响应中断呢？是在指令周期的结束阶段，为什么呢？
9. 除了ppt上列举的两种原因，还有一种原因就是：指令的执行在cpu内部是一种原子操作，也就是在指令的执行过程中不能被打断，否则肯定会引起程序执行的错误。同学们可以仔细想想为啥会引起执行错误。（硬件和软件两方面）
10. 下面就要详细介绍中断处理的过程了 。如右图所示，外部中断发出中断信号后会首先经过中断控制器，然后再由中断控制器的一系列处理才将中断请求发送给cpu，而异常和非屏蔽中断则直接由发出的设备直接发送给cpu。然后cpu开始查中断描述符表，找到中断处理函数，最后进入处理函数运行。
11. 发出中断请求的设备有两种，一种是外部中断源，发出的是外部中断，另一种是内部中断源，发出的是内部中断或异常。外设发出中断请求之后，由中断控制器将请求信号输入CPU的引脚，这样cpu才能知道有外部中断到来。而内部中断源发出的请求则不用通过中断控制器，直接发送给cpu。
12. CPU接受中断请求就称为中断响应。对于外部中断源，每条指令的最后一个时钟周期都会检测是否有中断请求。而对于不同的中断源，cpu则有不同的响应方式。对应内部中断源，cpu无条件响应，例如电源断点，一旦出现必须立即无条件响应，否则进行其他任何工作都是没有意义的。而外部中断源则会收到更严格的规定才能被响应，例如是否被屏蔽。
13. 为了使CPU在完成中断服务程序后能返回到源程序继续执行，需要将原程序被中断处的相关信息（EIP、CS、标志寄存器...）保存到堆栈中。待中断服务程序执行完毕后再将这些信息返回到CPU各寄存器当中。思考一下断点保护是由软件完成还是硬件自动完成呢？由硬件自动完成，因为将EIP入栈的动作不可能由显式的指令来完成。那么现场保护呢？由指令也就是软件来完成。
14. 断点保护之后就开始识别中断源了，cpu会读取中断请求发送来的中断类型码，总共有256个类型。每个中断源都有相应的服务程序，CPU识别中断源之后会取得中断服务程序的入口地址，跳到中断服务程序运行。
15. 找到对应的中断服务程序之后就可以开始运行中断服务程序了。如果到来的请求是异常或者非屏蔽中断，则直接在保存断点、保存现场之后就进入响应的中断处理程序了。而如果是外部中断，则会进入C，在do\_IRQ里面根据中断号irq再找到对应的中断服务例程，从而进入中断服务例程来运行。在调用do\_IRQ之前，会先调用SAVE\_ALL将各寄存器的值放入栈中，其中中断号就存放于这里面。

在这里我们又多了一个中断服务例程的概念，其实中断服务例程就是中断处理程序的子集，处理的是不同类型的中断过程中的不同部分，它和另外一部分相同的过程（例如保存寄存器的值）合起来就叫做中断处理程序。

1. 在中断服务例程结束之后，整个中断处理就接近尾声了，这涉及到断点、现场恢复及中断返回，这个过程的目的就是为了让之前被中断的程序能回到原来的状态继续执行。
2. ppt上的这张图就形象地反映了整个中断处理的过程。（简单描述）
3. 下面就进入中断服务这一部分了，在这一部分我们将学习到中断服务程序是如何运行的。
4. 大家可能对do\_IRQ这个函数感到好奇，为什么不直接通过idt表直接找到中断服务例程，而要经过do\_IRQ呢？这是因为老的版本中外设的数目还不够多，这时所分配给外部中断的15个中断号还够用，但是之后外设数目越来越多，中断号不够用了，因此用这个方式来增加可用的外部中断号。do\_IRQ的主要作用其实就是再次识别中断号，进入中断服务例程。
5. do\_IRQ只有一个参数，那就是regs，其实就是若干个寄存器的值（cpu现场），中断服务例程所需的各个参数都存放于寄存器中，前面的SAVE\_ALL函数将这些寄存器的值保存到内存中（regs参数）里面之后将这个参数在传递给do\_IRQ。
6. do\_IRQ的主要流程就是将被中断进程的上下文先保存，通过传入的参数（\*regs）找到中断号irq的值，通过这个值找到对应的中断服务例程，然后进入中断服务例程ISA。最后调用exiting\_irq()，标志下半部的开始、ISA的结束。
7. 前面提到了上半部和下半部的概念，这里我们先简单介绍一下。在外部中断到来，进入c之后，则会进入中断服务例程，在这里系统成功许愿可以通过request\_irq函数来添加自己定义的新的中断服务例程。do\_IRQ结束之后就进入下一段也就是中断处理的后半段，这一部分由do\_softirq来管理（它在do\_IRQ的末尾部分会被调用），它对于后半段的作用就像do\_IRQ对于前半段一样，类似一个通用的后半段处理函数，而后半段的处理函数可以由open\_softirq来添加。关于上半部和下半部的细节我们在下一节课会讲。
8. 在完成中断处理之后，就要进入尾声了。在do\_IRQ开启了后半段的处理之后就可以进行中断返回了，在后面我们会知道后半段的处理是可以延后的，也就是类似于进程睡眠一样的功能。在do\_IRQ里面是调用ret\_from\_intr这个函数来进行中断返回的，有兴趣的同学可以下去研究一下这个函数的源码。
9. 接下来就是中断注册了。其中涉及了几个重要的数据结构。对于每一个外设的IRQ都用一个irq\_desc实例来描述，主要存放中断服务例程的指针以及其他关于这个类型中断的数据。而irq\_desc数组则是一组描述中断的数组，这里同学们会发现它和终端描述符表非常类似，其实在早期中断描述符表的功能就和现在的irq\_desc数组是一样的，只是后来为了扩展的需求，将中断描述符表的这个功能移出来由irq\_desc数组来实现。
10. 接着就是irq\_desc内部irqaction类型的成员了，这里面存放的最重要的数据就是指向中断服务例程的指针，其他还存放着外设的ID号等数据，具体各个成员的含义可以看注释。
11. 刚才我们讲了Linux内核用request\_irq（）函数注册中断服务例程，这个服务例程可以由我们系统程序员自己来定义。在request\_irq（）函数实际干活的其实是request\_threaded\_irq（）函数。
12. 下面来看看函数调用所需的各个参数的意义：
    1. irq： irq 中断号，是一个整型
    2. handler：handler 中断服务函数，是在发生中断时，首先要执行的code
    3. thread\_fn：放在内核线程里执行的函数，通常为NULL
    4. irqflags ：指定了 快速中断或中断共享等中断处理属性 。
    5. name ：通常是设备驱动程序的名称
    6. dev\_id 中断名称 可作为共享中断时的中断区别参数，也可以用来指定中断服务函数需要参考的数据地址
13. request\_threaded\_irq（）函数的主要作用就是创建一个新的irqaction描述符，将新的irq号和新的中断服务例程句柄写入，再将irqaction描述符加入到IRQ链表中，同时在前面讲的irq\_desc数组中填入新的irq\_desc元素来指向这个irqaction描述符，这样就完成了irq号于中断服务例程的对应。
14. 有时候系统中有的中断服务例程比较老旧或者应该被淘汰了，我们不允许它们继续存在于内核当中，因为这样会导致整个内核越来越冗长。因此内核还提供了对硬中断注销的功能，也就是删除不用了的中断服务例程。内核提供得注销函数非常简单，就是将要注销的中断号和将要注销的ISR对应的设备号，注销的过程就是和注册的过程刚好相反。
15. 再来看看软中断的注册，软中断里面也有和硬中断的irq\_desc数组或IRQ链表类似的结构soft\_vec数组，作用也是将软中断号与对应的软终端处理函数关联起来，具体我们将在后面一节课将。软中断注册函数open\_softirq()的功能很简单，就是将soft\_vec数组的第nr个元素指向所要注册的action函数。
16. 接下来开始学习多处理器中断。在学习之前我们先来考虑几个问题。既然是处理器，那么有中断到来时，我们如何确定由哪一个cpu来处理这个中断。 处理器间如何互相发送中断。这是多处理器下的中断处理
17. 这下我们就大概知道整个中断处理过程的全貌了，但是前面我们讲都是在单核的情况下的中断处理，而如今市面上有很多多核设备，例如电脑、手机，那么多喝情况下的中断处理过程是怎样的呢？本章最后我们就来看看多处理器中断。
18. 这里也有几个再设计多处理器中断时需要考虑的问题。第一，异常来源于哪儿？第二多处理器如何处理异常。下面带着这些问题和我们自己的思考，学习前人是如何解决这些问题的。
19. 在多处理器时代肯定不能使用单处理器时代的那一套方法来处理中断，因此不再使用传统的8259A来作为中断控制器，而是使用APIC。APIC分为本地APIC（LAPIC）以及I/OAPIC。本地apic在单个的cpu内部，有多少cpu就有多少个本地apic。而ioapic集成在主板上。Ioaipc负责将到来的中断分发给各个cpu上的本地aipc。
20. I/OAPIC内部主要由一组IRQ线、一张24项的中断重定向表构成。中断重定向表的作用如ppt上所讲，就是一个将中断进行重定向的功能。整个Ioaipc的功能就是将到来的中断通过先前设置好的策略分发到对应的本地aipic上。
21. Ioapic分配中断的策略有两种，静态分发和动态分发。静态意思就是在重定向表中写好了对各个中断信号进行处理的处理器，而动态分发则会根据各cpu所运行进程的优先级来确定中断在那个cpu上运行。整个分发的过程就如黄色框里面所描述的一样。
22. 前面说的是外中断到来时如何进行处理，下面就是处理器间中断该如何处理了。处理器间中断时利用IPI这种特殊的硬中断来实现的，由处理器发出被其他处理器接收，便于处理器间通信或同步。当一个处理器接收到一个中断时，如果发现另一个处理器处理该中断更加合理，则可以通过 IPI 机制将该中断传递到其他的处理器，实现处理器的负载平衡。这里我们需要注意一下，处理器间发送中断请求只用通过它们之间的本地APIC，而不用通过I/OAPIC。
23. 下面是Linux 针对 IA32 的 SMP 系统定义的 5 种 IPI中断向量。前面的三种都有它们各自对应的功能，而后面两种则保留下来用于今后的扩展。
24. 各处理器可以使用内核提供的API来进行处理器间中断。