第13章　中断和中断处理

本章介绍Linux系统内核处理中断的方式。

13.1　中断

Linux系统中有很多不同的硬件设备。你可以同步使用这些设备，也就是说你可以发出一个请求，然后等待一直到设备完成操作以后再进行其他的工作。但这种方法的效率却非常的低，因为操作系统要花费很多的等待时间。一个更为有效的方法是发出请求以后，操作系统继续其他的工作，等设备完成操作以后，给操作系统发送一个中断，操作系统再继续处理和此设备有关的操作。

在将多个设备的中断信号送往CPU的中断插脚之前，系统经常使用中断控制器来综合多个设备的中断。这样即可以节约CPU的中断插脚，也可以提高系统设计的灵活性。中断控制器用来控制系统的中断，它包括屏蔽和状态寄存器。设置屏蔽寄存器的各个位可以允许或屏蔽某一个中断，状态寄存器则用来返回系统中正在使用的中断。

大多数处理器处理中断的过程都相同。当一个设备发出中段请求时， CPU停止正在执行的指令，转而跳到包括中断处理代码或者包括指向中断处理代码的转移指令所在的内存区域。这些代码一般在CPU的中断方式下运行。在此方式下，将不会再有中断发生。但有些 CPU的中断有自己的优先权，这样，更高优先权的中断则可以发生。这意味着第一级的中断处理程序必须拥有自己的堆栈，以便在处理更高级别的中断前保存CPU的执行状态。

当中断处理完毕以后， CPU将恢复到以前的状态，继续执行中断处理前正在执行的指令。中断处理程序十分简单有效，这样，操作系统就不会花太长的时间屏蔽其他的中断。

13.2　可编程中断控制器

系统设计者可以随意选择中断结构，但IBM PC使用Intel 82C59A-2 CMOS可编程控制器。

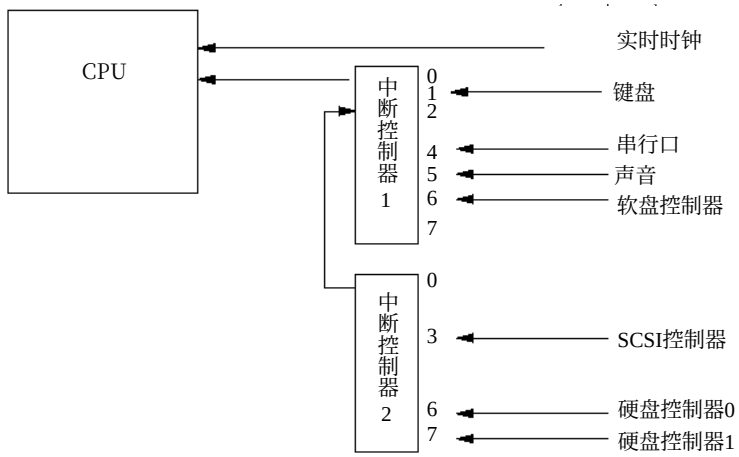

图13-1　中断控制示意图

此控制器的寄存器在ISA系统的内存空间中占有固定位置。

图13-1显示有两个8位的中断控制器级联在一起，每一个都有自己的屏蔽寄存器和中断状态寄存器。屏蔽寄存器的地址是0x21 和 0xA1，而状态寄存器的地址是0x20 和 0xA0。往屏蔽寄存器中的某一位写入1，则允许相应的中断；如果写入0，则会禁止相应的中断。所以，向屏蔽寄存器中的第三位写入1，将允许中断3；写入0，则将禁止中断3。但不幸的是，屏蔽寄存器是只写的，你无法读取该寄存器。这样，Linux系统自己必须保留一份写入到寄存器中的内容。

当发生中断信号时，中断处理程序读取两个中断状态寄存器（ ISR）的值。它将位于0x20的ISR放入一个16位的中断寄存器的低8位，同时将位于0xA0的ISR放入高8位。PIC1中的第二位用于中断控制器的级连，所以任何来自PIC2的中断都将导致PIC1中的第二位置1。

13.3 初始化中断处理的数据结构

系统内核有关中断处理的数据结构是在设备驱动程序要求对系统中断控制时创建起来的。在此过程中，设备驱动程序使用了一系列用于请求中断、允许中断以及禁止中断的 Linux系统内核服务。每一个设备驱动程序都将调用这些子过程来登记它们的中断处理过程的地址。

在PC机中，有些中断是固定的，所以当初始化时，驱动程序只需简单地请求中断即可。系统中的软盘驱动程序就是这样，它请求中断 6。但有时一个设备驱动程序不知道设备将使用哪一个中断。在 PCI结构中这不会成为一个问题，因为 PCI的设备驱动程序总是知道它们的中断号。但对于ISA结构而言，一个设备驱动程序找到自己使用的中断号却并不容易。 Linux系统通过允许设备驱动程序探测自己的中断来解决这个问题。

首先，设备驱动程序使得设备产生一个中断。然后，允许系统中所有没有指定的中断，这意味着设备挂起的中断将会通过中断控制器传送。 Linux系统读取中断状态寄存器然后将它的值返回到设备驱动程序。一个非 0的结果意味着在探测期间发生了一个或者多个的中断。设备驱动程序现在可以关闭探测，这时所有还未被指定的中断将继续被禁止。

一个ISA 设备驱动程序知道了它的中断号以后，就可以请求对中断的控制了。

PCI 结构的系统中断比ISA结构的系统中断要灵活得多。ISA设备使用中断插脚经常使用跳线设置，所以在设备驱动程序中是固定的。但 PCI设备是在系统启动过程中PCI初始化时由PCI BIOS 或 PCI 子系统分配的。每一个PCI设备都有可能使用A、B、C 或者 D这4个中断插脚中的一个。缺省情况下设备使用插脚A。

每个PCI插槽的PCI中断A、B、C 和 D是通过路由选择连接到中断控制器上的。所以PCI插槽4的插脚A可能连接到中断控制器的插脚 6，PCI插槽4的插脚B可能连接到中断控制器的插脚7,以此类推。

PCI中断具体如何进行路由一般依照系统的不同而不同，但系统中一定存在 PCI中断路由拓扑结构的设置代码。在Intel PC机中，系统的BIOS代码负责中断的路由设置。对于没有BIOS的系统，Linux系统内核负责设置。

PCI的设置代码将中断控制器的插脚号写入到每个设备的PCI 设置头中。PCI的设置代码根据所知道的PCI中断路由拓扑结构、 PCI设备使用的插槽，以及正在使用的PCI中断的插脚号来决定中断号，也就是IRQ号。

系统中可以有很多的PCI中断源，例如当系统使用了PCI-PCI桥时。这时，中断源的数目可能超过了系统可编程中断控制器上插脚的数目。在这种情况下，某些 PCI设备之间就不得不共享一个中断，也就是说，中断控制器上的某一个插脚可以接收来自几个设备的中断。 Linux系统通过让第一个中断源请求者宣布它使用的中断是否可以被共享来实现中断在几个设备之间共

享的。中断共享使得irq_action数组中的同一个入口指向几个设备的irqaction结构。当一个共享的中断有中断发生时，Linux系统将会调用和此中断有关的所有中断处理程序。所有可以共享中断的设备驱动程序的中断处理程序都可能在任何时候被调用，即使在自身没有中断需要处理时。

13.4 中断处理

Linux系统处理中断所需要的数据结构参见图13-2。

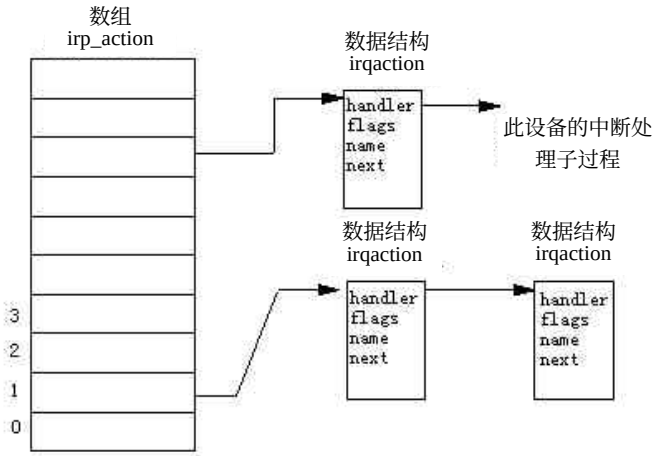

图13-2 中断处理数据结构示意图

Linux系统中断处理程序的一个主要任务就是将中断定位到中断处理程序适当部分的代码上，此代码必须了解中断的拓扑结构。例如，如果软盘控制器的中断在中断控制器的插脚为 6，那么系统的中断处理程序必须能够识别此中断是来自软盘控制器，并且可以将中断定位到软盘驱动程序的中断处理代码中。Linux系统使用一个指针数组指向包括系统中断处理程序地址的数据结构。这些处理程序属于设备驱动程序的一部分，并且每个设备驱动程序自己负责在初始化时请求中断。图13-2显示irq_action是指向irqaction结构的指针的数组。每一个irqaction结构都包含有关此中断处理程序的信息，包括中断处理程序的地址。由于各个系统之间的中断数目和具体如何处理中断并不完全相同，所以Linux系统的中断处理代码是随系统而异的。

当中断发生时，Linux系统首先通过读取系统中可编程中断控制器的中断状态寄存器来确定中断的来源。然后将此来源转换成irq_action数组中的位移。例如，来自软盘的中断控制器插脚6的中断将会转换成irq_action数组中第七个指针。如果发生的中断不存在中断处理程序，那么Linux系统内核将记录一个错误。否则它将调用此中断来源的所有 irqaction结构所指向的中断处理程序。

当Linux系统内核调用设备驱动程序的中断处理代码时，它必须能够迅速找出中断的原因和处理办法。设备驱动程序可以通过读取被中断的设备的状态寄存器的值来确定中断的原因。一旦找出中断的原因，设备驱动程序可能需要做更多的工作。如果是这样，Linux系统内核可以将工作推迟到以后再完成。这样就避免了CPU在中断方式下花费太多的时间。