具有它们自己的状态、堆栈和程序计数器，那么这一模型运转得最好。

当然，现实没有如此简单。对一个中断进行处理并不只是简单地捕获中断，在某个信号量上执行up操作，然后执行一条IRET指令从中断返回到先前的进程。对操作系统而言，还涉及更多的工作。我们将按一系列步骤给出这一工作的轮廓，这些步骤是硬件中断完成之后必须在软件中执行的。应该注意的是，细节是非常依赖于系统的，所以下面列出的某些步骤在一个特定的机器上可能是不必要的，而没有列出的步骤可能是必需的。此外，确实发生的步骤在某些机器上也可能有不同的顺序。

1) 保存没有被中断硬件保存的所有寄存器（包括PSW）。

2) 为中断服务过程设置上下文，可能包括设置TLB、MMU和页表。

3) 为中断服务过程设置堆栈。

4) 应答中断控制器，如果不存在集中的中断控制器，则再次开放中断。

5) 将寄存器从它们被保存的地方（可能是某个堆栈）复制到进程表中。

6) 运行中断服务过程，从发出中断的设备控制器的寄存器中提取信息。

7) 选择下一次运行哪个进程，如果中断导致某个被阻塞的高优先级进程变为就绪，则可能选择它现在就运行。

8) 为下一次要运行的进程设置MMU上下文，也许还需要设置某个TLB。

9) 装入新进程的寄存器，包括其PSW。

10) 开始运行新进程。

由此可见，中断处理远不是无足轻重的小事。它要花费相当多的CPU指令，特别是在存在虚拟内存并且必须设置页表或者必须保存MMU状态（例如*R*和*M*位）的机器上。在某些机器上，当在用户态与核心态之间切换时，可能还需要管理TLB和CPU高速缓存，这就要花费额外的机器周期。

5.3.2 设备驱动程序

在本章前面的内容中，我们介绍了设备控制器所做的工作。我们注意到每一个控制器都设有某些设备寄存器用来向设备发出命令，或者设有某些设备寄存器用来读出设备的状态，或者设有这两种设备寄存器。设备寄存器的数量和命令的性质在不同设备之间有着根本性的不同。例如，鼠标驱动程序必须从鼠标接收信息，以识别鼠标移动了多远的距离以及当前哪一个键被按下。相反，磁盘驱动程序可能必须要了解扇区、磁道、柱面、磁头、磁盘臂移动、电机驱动器、磁头定位时间以及所有其他保证磁盘正常工作的机制。显然，这些驱动程序是有很大区别的。

因而，每个连接到计算机上的I/O设备都需要某些设备特定的代码来对其进行控制。这样的代码称为*设备驱动程序*（device driver），它一般由设备的制造商编写并随同设备一起交付。因为每一个操作系统都需要自己的驱动程序，所以设备制造商通常要为若干流行的操作系统提供驱动程序。

每个设备驱动程序通常处理一种类型的设备，或者至多处理一类紧密相关的设备。例如，SCSI磁盘驱动程序通常可以处理不同大小和不同速度的多个SCSI磁盘，或许还可以处理SCSI CD-ROM。而另一方面，鼠标和游戏操纵杆是如此的不同，以至于它们通常需要不同的驱动程序。然而，对于一个设备驱动程序控制多个不相关的设备并不存在技术上的限制，只是这样做并不是一个好主意。

为了访问设备的硬件（意味着访问设备控制器的寄存器），设备驱动程序通常必须是操作系统内核的一部分，至少对目前的体系结构是如此。实际上，有可能构造运行在用户空间的驱动程序，使用系统调用来读写设备寄存器。这一设计使内核与驱动程序相隔离，并且使驱动程序之间相互隔离，这样做可以消除系统崩溃的一个主要源头——有问题的驱动程序以这样或那样的方式干扰内核。对于建立高度可靠的系统而言，这绝对是正确的方向。MINIX 3就是一个这样的系统，其中设备驱动程序就作为用户进程而运行。然而，因为大多数其他桌面操作系统要求驱动程序运行在内核中，所以我们在这里只考虑这样的模型。

因为操作系统的设计者知道由外人编写的驱动程序代码片断将被安装在操作系统的内部，所以需要有一个体系结构来允许这样的安装。这意味着要有一个定义明确的模型，规定驱动程序做什么事情以及如何与操作系统的其余部分相互作用。设备驱动程序通常位于操作系统其余部分的下面，如图5-12所示。