输出下一个字符，应答中断，并且返回到中断之前正在运行的进程，该进程将从其停止的地方继续运行。

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count - 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

b)

图5-9 使用中断驱动I/O将一个字符串写到打印机：a) 当打印系统调用被发出时执行的代码；b) 打印机的中断服务过程

5.2.4 使用DMA的I/O

中断驱动I/O的一个明显缺点是中断发生在每个字符上。中断要花费时间，所以这一方法将浪费一定数量的CPU时间。这一问题的一种解决方法是使用DMA。此处的思路是让DMA控制器一次给打印机提供一个字符，而不必打扰CPU。本质上，DMA是程序控制I/O，只是由DMA控制器而不是主CPU做全部工作。这一策略需要特殊的硬件（DMA控制器），但是使CPU获得自由从而可以在I/O期间做其他工作。使用DMA的代码概要如图5-10所示。

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

b)

图5-10 使用DMA打印一个字符串：a) 当打印系统调用被发出时执行的代码；b) 中断服务过程

DMA重大的成功是将中断的次数从打印每个字符一次减少到打印每个缓冲区一次。如果有许多字符并且中断十分缓慢，那么采用DMA可能是重要的改进。另一方面，DMA控制器通常比主CPU要慢很多。如果DMA控制器不能以全速驱动设备，或者CPU在等待DMA中断的同时没有其他事情要做，那么采用中断驱动I/O甚至采用程序控制I/O也许更好。

5.3 I/O软件层次

I/O软件通常组织成四个层次，如图5-11所示。每一层具有一个要执行的定义明确的功能和一个的定义明确的与邻近层次的接口。功能与接口随系统的不同而不同，所以下面的讨论并不针对一种特定的机器。我们将从底层开始讨论每一层。

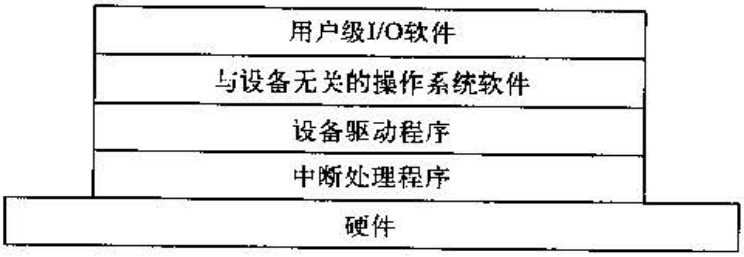

图5-11 I/O软件系统的层次

5.3.1 中断处理程序

虽然程序控制I/O偶尔是有益的，但是对于大多数I/O而言，中断是令人不愉快的事情并且无法避免。应当将其深深地隐藏在操作系统内部，以便系统的其他部分尽量不与它发生联系。隐藏它们的最好办法是将启动一个I/O操作的驱动程序阻塞起来，直到I/O操作完成且产生一个中断。驱动程序阻塞自己的手段有：在一个信号量上执行down操作、在一个条件变量上执行wait操作、在一个消息上执行receive操作或者某些类似的操作。

当中断发生时，中断处理程序将做它必须要做的全部工作以便对中断进行处理。然后，它可以将启动中断的驱动程序解除阻塞。在一些情形中，它只是在一个信号量上执行up操作；其他情形中，是对管程中的条件变量执行signal操作，还有一些情形中，是向被阻塞的驱动程序发一个消息。在所有这些情形中，中断最终的结果是使先前被阻塞的驱动程序现在能够继续运行。如果驱动程序构造为内核进程，