相反，图5-14b所示为一种不同的设计，在这种设计中所有驱动程序具有相同的接口。这样一来，倘若符合驱动程序接口，那么添加一个新的驱动程序就变得容易多了。这还意味着驱动程序的编写人员知道驱动程序的接口应该是什么样子的。实际上，虽然并非所有的设备都是绝对一样的，但是通常只存在少数设备类型，而它们的确大体上是相同的。

这种设计的工作方式如下。对于每一种设备类型，例如磁盘或打印机，操作系统定义一组驱动程序必须支持的函数。对于磁盘而言，这些函数自然地包含读和写，除此之外还包含开启和关闭电源、格式化以及其他与磁盘有关的事情。驱动程序通常包含一张表格，这张表格具有针对这些函数指向驱动程序自身的指针。当驱动程序装载时，操作系统记录下这张函数指针表的地址，所以当操作系统需要调用一个函数时，它可以通过这张表格发出间接调用。这张函数指针表定义了驱动程序与操作系统其余部分之间的接口。给定类型（磁盘、打印机等）的所有设备都必须服从这一要求。

如何给I/O设备命名是统一接口问题的另一个方面。与设备无关的软件要负责把符号化的设备名映射到适当的驱动程序上。例如，在UNIX系统中，像/dev/disk0这样的设备名惟一确定了一个特殊文件的i节点，这个i节点包含了*主设备号*（major device number），主设备号用于定位相应的驱动程序。i节点还包含*次设备号*（minor device number），次设备号作为参数传递给驱动程序，用来确定要读或写的具体单元。所有设备都具有主设备号和次设备号，并且所有驱动程序都是通过使用主设备号来选择驱动程序而得到访问。

与设备命名密切相关的是设备保护。系统如何防止无权访问设备的用户访问设备呢？在UNIX和Windows中，设备是作为命名对象出现在文件系统中的，这意味着针对文件的常规的保护规则也适用于I/O设备。系统管理员可以为每一个设备设置适当的访问权限。

2. *缓冲*

无论对于块设备还是对于字符设备，由于种种原因，缓冲也是一个重要的问题。作为例子，我们考虑一个想要从调制解调器读入数据的进程。让用户进程执行read系统调用并阻塞自己以等待字符的到来，这是对到来的字符进行处理的一种可能的策略。每个字符的到来都将引起中断，中断服务过程负责将字符递交给用户进程并且将其解除阻塞。用户进程把字符放到某个地方之后可以对另一个字符执行读操作并且再次阻塞。这一模型如图5-15a所示。

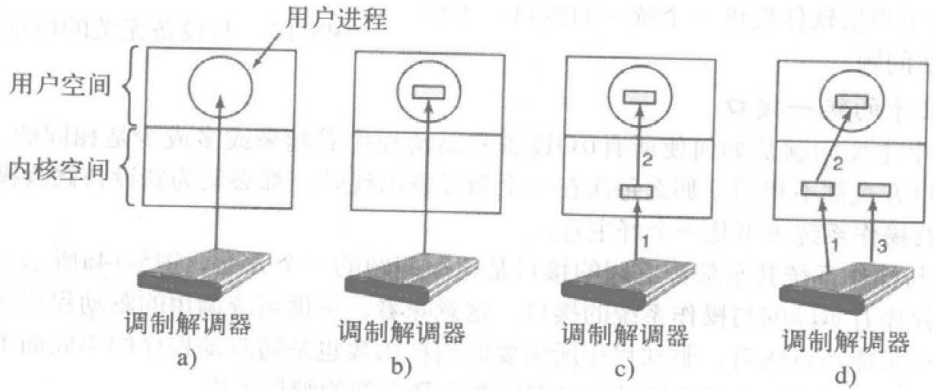

图5-15 a) 无缓冲的输入；b) 用户空间中的缓冲；c) 内核空间中的缓冲接着复制到用户空间；d) 内核空间中的双缓冲

这种处理方式的问题在于：对于每个到来的字符，都必须启动用户进程。对于短暂的数据流量让一个进程运行许多次效率会很低，所以这不是一个良好的设计。

图5-15b所示为一种改进措施。此处，用户进程在用户空间中提供了一个包含n个字符的缓冲区，并且执行读入n个字符的读操作。中断服务过程负责将到来的字符放入该缓冲区中直到缓冲区填满，然后唤醒用户进程。这一方案比前一种方案的效率要高很多，但是它也有一个缺点：当一个字符到来时，如果缓冲区被分页而调出了内存会出现什么问题呢？解决方法是将缓冲区锁定在内存中，但是如果许多进程都在内存中锁定页面，那么可用页面池就会收缩并且系统性能将下降。

另一种方法是在内核空间中创建一个缓冲区并且让中断处理程序将字符放到这个缓冲区中，如图