```
sort <input> output
```

并且无论输入来自任意类型的存储盘或者键盘，输出送往任意类型的存储盘或者屏幕，上述命令都可以工作。尽管这些设备实际上差别很大，需要非常不同的命令序列来读或写，但这一事实所带来的问题将由操作系统负责处理。

与设备独立性密切相关的是统一命名（uniform naming）这一目标。一个文件或一个设备的名字应该是一个简单的字符串或一个整数，它不应依赖于设备。在UNIX系统中，所有存储盘都能以任意方式集成到文件系统层次结构中，因此，用户不必知道哪个名字对应于哪台设备。例如，一个USB盘可以安装（mount）到目录/usr/ast/backup下，这样复制一个文件到/usr/ast/backup/monday就是将文件复制到USB盘上。用这种方法，所有文件和设备都采用相同的方式——路径名进行寻址。

I/O软件的另一个重要问题是错误处理（error handling）。一般来说，错误应该尽可能地在接近硬件的层面得到处理。当控制器发现了一个读错误时，如果它能够处理那么就应该自己设法纠正这一错误。如果控制器处理不了，那么设备驱动程序应当予以处理，可能只需重读一次这块数据就正确了。很多错误是偶然性的，例如，磁盘读写头上的灰尘导致读写错误时，重复该操作，错误经常就会消失。只有在低层软件处理不了的情况下，才将错误上交高层处理。在许多情况下，错误恢复可以在低层透明地得到解决，而高层软件甚至不知道存在这一错误。

另一个关键问题是同步（synchronous）（即阻塞）和异步（asynchronous）（即中断驱动）传输。大多数物理I/O是异步的——CPU启动传输后便转去做其他工作，直到中断发生。如果I/O操作是阻塞的，那么用户程序就更加容易编写——在read系统调用之后，程序将自动被挂起，直到缓冲区中的数据准备好。正是操作系统使实际上是中断驱动的操作变为在用户程序看来是阻塞式的操作。

I/O软件的另一个问题是缓冲（buffering）。数据离开一个设备之后通常并不能直接存放到其最终的目的地。例如，从网络上进来一个数据包时，直到将该数据包存放在某个地方并对其进行检查，操作系统才知道要将其置于何处。此外，某些设备具有严格的实时约束（例如，数字音频设备），所以数据必须预先放置到输出缓冲区之中，从而消除缓冲区填满速率和缓冲区清空速率之间的相互影响，以避免缓冲区欠载。缓冲涉及大量的复制工作，并且经常对I/O性能有重大影响。

此处我们将提到的最后一个概念是共享设备和独占设备的问题。有些I/O设备（如磁盘）能够同时让多个用户使用。多个用户同时在同一磁盘上打开文件不会引起什么问题。其他设备（如磁带机）则必须由单个用户独占使用，直到该用户使用完，另一个用户才能拥有该磁带机。让两个或更多的用户随机地将交叉混杂的数据块写入相同的磁带是注定不能工作的。独占（非共享）设备的引入也带来了各种各样的问题，如死锁。同样，操作系统必须能够处理共享设备和独占设备以避免问题发生。

5.2.2 程序控制I/O

I/O可以以三种根本不同的方式实现。在本小节中我们将介绍第一种（程序控制I/O），在后面两小节中我们将研究另外两种（中断驱动I/O和使用DMA的I/O）。I/O的最简单形式是让CPU做全部工作，这一方法称为*程序控制I/O*（programmed I/O）。

借助于例子来说明程序控制I/O是最简单的。考虑一个用户进程，该进程想在打印机上打印8个字符的字符串“ABCDEFGH”。它首先要在用户空间的一个缓冲区中组装字符串，如图5-7a所示。

然后，用户进程通过发出系统调用打开打印机来获得打印机以便进行写操作。如果打印机当前被另一个进程占用，该系统调用将失败并返回一个错误代码，或者将阻塞直到打印机可用，具体情况取决于操作系统和调用参数。一旦拥有打印机，用户进程就发出一个系统调用通知操作系统在打印机上打印字符串。

然后，操作系统（通常）将字符串缓冲区复制到内核空间中的一个数组（如*p*）中，在这里访问更加容易（因为内核可能必须修改内存映射才能到达用户空间）。然后操作系统要查看打印机当前是否可用。如果不可用，就要等待直到它可用。一旦打印机可用，操作系统就复制第一个字符到打印机的数据寄存器中，在这个例子中使用了内存映射I/O。这一操作将激活打印机。字符也许还不会出现在打印机上，因为某些打印机在打印任何东西之前要先缓冲一行或一页。然而，在图5-7b中，我们看到第一个字符已经打印出来，并且系统已经将“B”标记为下一个待打印的字符。