在哪个设备上，上面提到的命令就可以变成

```
cp /b/x /a/d/x
```

和所有文件都在硬盘上是一样的。

Linux文件系统的另一个有趣的性质是*加锁*（locking）。在一些应用中会出现两个或更多的进程同时使用同一个文件的情况，可能导致竞争条件（race condition）。有一种解决方法是使用临界区，但是如果这些进程属于相互不认识的独立的用户，这种解决方法是不方便的。

考虑这样的一个例子，一个数据库组织许多文件在一个或多个目录中，它们可以被不相关的用户访问。可以通过设置信号量来解决互斥的问题，在每个目录或文件上设置一个信号量，当程序需要访问相应的数据时，在相应的信号量上做一个down操作。但这样做的缺点是，尽管进程只需要访问一条记录却使得整个目录或文件都不能访问。

由于这种原因，POSIX提供了一种灵活的、细粒度的机制，允许一个进程使用一个不可分割的操作对小到一个字节、大到整个文件加锁。加锁机制要求加锁者标识要加锁的文件、开始位置以及要加锁的字节数。如果操作成功，系统会在表格中添加记录说明要求加锁的字节（如数据库的一条记录）已被锁住。

系统提供了两种锁，*共享锁*和*互斥锁*。如果文件的一部分已经被加了共享锁，那么在上面尝试加共享锁是允许的，但是加互斥锁是不会成功的；如果文件的一部分已经被加了互斥锁，那么在互斥锁解除之前加任何锁都不会成功。为了成功地加锁，请求加锁的部分的所有字节都必须是可用的。

在加锁时，进程必须指出当加锁不成功时是否阻塞。如果选择阻塞，则当已经存在的锁被删除时，进程被放行并在文件上加锁；如果选择不阻塞，系统调用在加锁失败时立即返回，并设置状态码表明加锁是否成功，如果不成功，由调用者决定下一步动作（比如，等待或者继续尝试）。

加锁区域可以是重叠的。如图10-26a所示，进程A在第4字节到第7字节的区域加了共享锁，之后，进程B在第6字节到第9字节加了共享锁，如图10-26b所示，最后，进程C在第2字节到第11字节加了共享锁。由于这些锁都是共享锁，是可以同时存在的。

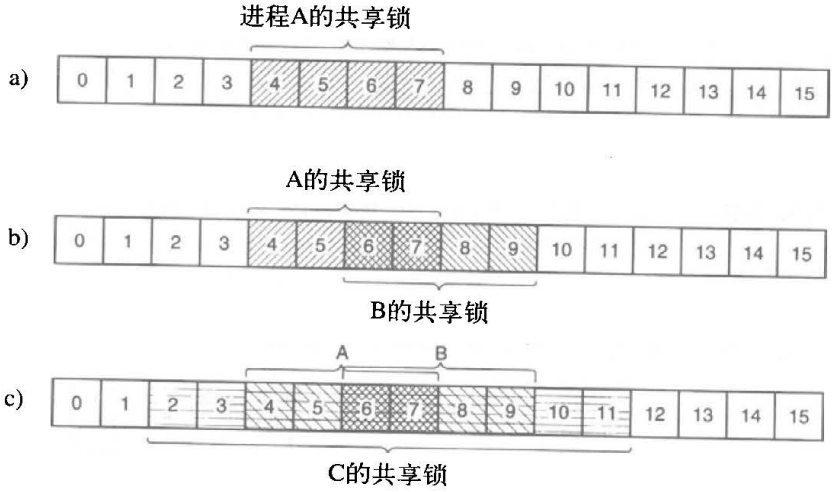

图10-26 a) 加了一个锁的文件；b) 增加了第二个锁；c) 增加了第三个锁

此时，如果一个进程试图在图10-26c中文件的第9个字节加互斥锁，并设置加锁失败时阻塞，那么会发生什么？由于该区域已经被进程B和进程C两个进程加锁，这个进程将会被阻塞，直到进程B和进程C释放它们的锁为止。

10.6.2 Linux的文件系统调用

许多系统调用与文件和文件系统有关。在本节中，首先研究对单个文件进行操作的系统调用，之后我们会研究针对目录和文件系统的系统调用。要创建一个文件时，可以使用creat系统调用。（曾经有人问Ken Thompson，如果给他一次重新发明UNIX的机会，他会做什么不同事情，他回答说他要把这个系统调用的拼写改成create，而不是现在的creat。）这个系统调用的参数是文件名和保护模式。于是，