当一个程序启动的时候，它的栈并不是空的。相反，它包含了所有的环境变量以及为了调用它而向shell输入的命令行。这样，一个程序就可以发现它的参数了。比如，当输入以下命令

```
cp src dest
```

时，cp程序运行，并且栈上有字符串“cp src dest”，这样程序就可以找到源文件和目标文件的名字。这些字符串被表示为一个指针数组来指向字符串中的符号，使得解析更加容易。

当两个用户运行同样的程序，比如编辑器，可以在内存中立刻保持该编辑器程序代码的两个副本，但是并不高效。相反地，大多数Linux系统支持*共享代码段*。在图10-12a和图10-12c中，可以看到两个进程A和B拥有相同的代码段。在图10-12b中可以看到物理内存的一种可能布局，其中两个进程共享了同样的代码片段。这种映射是通过虚拟内存硬件来实现的。

数据段和栈段从来不共享，除非是在一个fork之后，并且仅仅是那些没有被修改的页面。如果二者之一要增长但是没有邻近的空间来增长，这并不会产生问题，因为在虚拟地址空间中邻近的页面并不一定要映射到邻近的物理页面上。

在有些计算机上，硬件支持指令和数据拥有不同的地址空间。如果有这个特性，Linux就可以利用它。例如，在一个32位地址的计算机上如果有这个特性，那么就有2^{32}字节的指令地址空间和2^{32}字节的数据地址空间。一个到0的跳转指令跳入到代码段的地址0，而一个从0的移动使用数据空间的地址0。这使得可用的数据空间加倍。

除了动态分配更多的内存，Linux中的进程可以通过*内存映射文件*来访问文件数据。这个特性使我们可以把一个文件映射到进程空间的一部分而该文件就可以像位于内存中的字节数组一样被读写。把一个文件映射进来使得随机读写比使用read和write之类的IO系统调用要容易的多。共享库的访问就是用这种机制映射进来后进行的。在图10-13中，我们可以看到一个文件被同时映射到两个进程中，但在不同的虚拟地址上。

把一个文件映射进来的一个附加的好处是两个或者更多的进程可以同时映射相同的文件。其中一个进程对文件的写可以被其他进程马上看到。实际上，通过映射一个临时文件（所有的进程退出之后就被丢弃），这种机制可以为多进程共享内存提供高带宽。在最极限的情况下，两个（或者更多）进程可以映射一个文件覆盖整个地址空间，从而提供了一种介于进程之间和线程之间的共享方式。这样地址空间是共享的（类似于线程），但是每个进程维护其自身的打开文件和信号，这些不同于线程。实际上，从来没有做过让两个地址空间完全相同的事情。

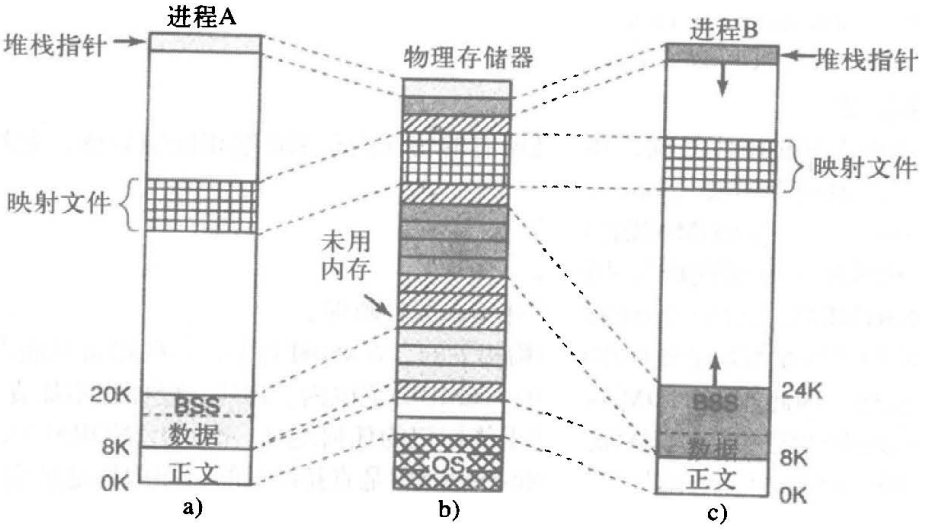

图10-13 两个进程可以共享一个映射文件

10.4.2 Linux中的内存管理系统调用

POSIX没有给内存管理指定任何系统调用。这个主题被认为是太依赖于机器而不便于标准化。可是，这个问题通过这样的说法被隐藏起来了：那些需要动态内存管理的程序可以使用malloc库函数（由ANSI