的库可能具有冲突的 p_vaddr 值，系统必须在运行时找到备选虚拟地址以解决任何冲突。

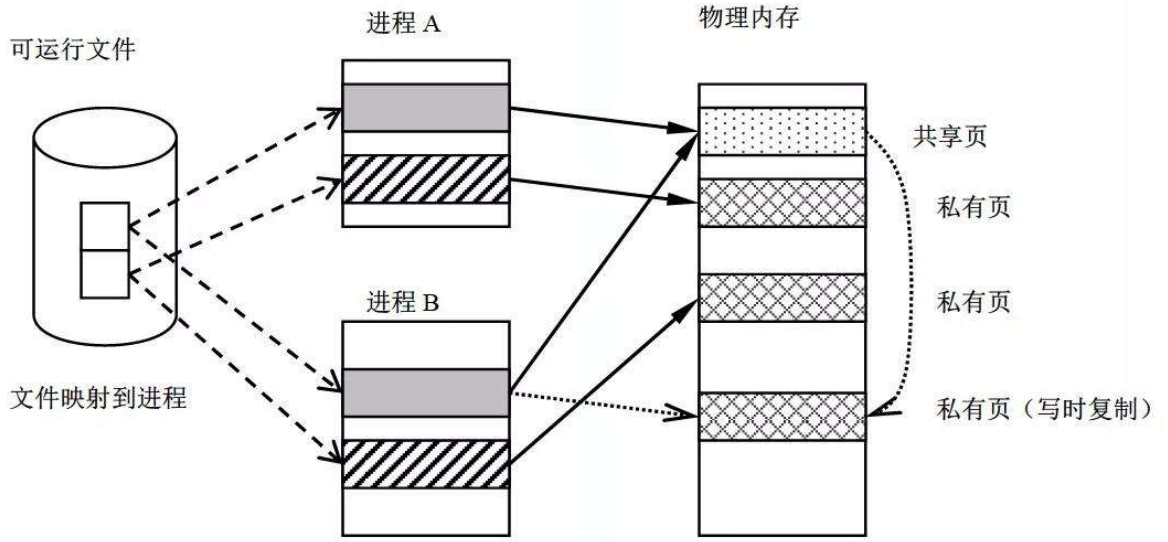

图 6-2　文件映射和物理内存共享

当一个库被加载到内存中时，它可能会在不同的程序或同一程序的不同实例中被分配到不同的虚拟地址。这个虚拟地址和库文件内的程序头指定的地址之间的差值被称为“基地址”。基地址用于修正库中的符号和引用地址。但库中的不同段的相对位置是固定的，所以跨段的引用并不受实际加载位置的影响。

假设我们有一个名为 libMath.so 的共享库，该库有一个名为 add()的函数。当这个库被链接器编译时，它可能会为 add()函数分配一个默认的虚拟地址，比如 0x1000。这个地址是在 libMath.so 的程序头中由 p_vaddr 指定的。

现在，我们的程序 mainApp 要使用这个库。当 mainApp 启动并加载 libMath.so 时，系统加载器可能会发现 0x1000 这个地址已经被其他程序或库使用了。于是，系统加载器决定将 libMath.so 加载到另一个地址，比如 0x5000。这时，基地址（即差值）就是 0x5000 − 0x1000 = 0x4000。

因此，当 mainApp 调用 add()函数时，它不会去默认的 0x1000 地址查找，而是会加上基地址 0x4000，从而在 0x5000 处找到并执行 add()函数。

同时，libMath.so 库内部如果有任何对其他函数或变量的引用，它们的地址也需要根据这个基地址进行调整。例如，如果 libMath.so 内部有一个全局变量 globalVar，其默认地址是 0x1100，那么在实际运行时，其地址将被重定位为 0x1100 + 0x4000 = 0x5100。

一般来说，程序的大多数库都是按照固定的顺序加载的，因此它们在不同的程序实例中的加载地址通常是相同的。这有助于调试，因为可以在固定的指令地址上设置断点。为了提高安全性，许多现代的 Linux 发行版默认启用了地址空间随机布局（Address Space Layout Randomization，ASLR）。ASLR 会使链接器随机选择地址来加载共享库，但这个功能可以被关闭。

当进程启动时，其可执行文件总是首先被加载到内存中，并且总是加载到固定的地址，除非启用了 ASLR，但共享库可能会被加载到不同的地址。因此它们需要使用位置独立的代码