更新情况，然后显式地将他们的程序与更新了的库重新链接。

另一个问题是几乎每个C程序都使用标准I/O函数，比如printf和scanf。在运行时，这些函数的代码会被复制到每个运行进程的文本段中。在一个运行上百个进程的典型系统上，这将是对稀缺的内存系统资源的极大浪费。(内存的一个有趣属性就是不论系统的内存有多大，它总是一种稀缺资源。磁盘空间和厨房的垃圾桶同样有这种属性。)

共享库(shared library)是致力于解决静态库缺陷的一个现代创新产物。共享库是一个目标模块，在运行或加载时，可以加载到任意的内存地址，并和一个在内存中的程序链接起来。这个过程称为*动态链接*(dynamic linking)，是由一个叫做*动态链接器*(dynamic linker)的程序来执行的。共享库也称为*共享目标*(shared object)，在Linux系统中通常用.so后缀来表示。微软的操作系统大量地使用了共享库，它们称为DLL(动态链接库)。

共享库是以两种不同的方式来“共享”的。首先，在任何给定的文件系统中，对于一个库只有一个.so文件。所有引用该库的可执行目标文件共享这个.so文件中的代码和数据，而不是像静态库的内容那样被复制和嵌入到引用它们的可执行的文件中。其次，在内存中，一个共享库的.text节的一个副本可以被不同的正在运行的进程共享。在第9章我们学习虚拟内存时将更加详细地讨论这个问题。

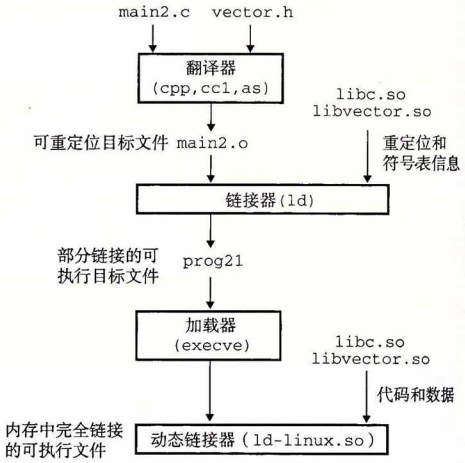

图7-16　动态链接共享库

图7-16概括了图7-7中示例程序的动态链接过程。为了构造图7-6中示例向量例程的共享库libvector.so，我们调用编译器驱动程序，给编译器和链接器如下特殊指令：

```
linux> gcc -shared -fpic -o libvector.so addvec.c multvec.c
```

-fpic选项指示编译器生成与位置无关的代码(下一节将详细讨论这个问题)。-shared选项指示链接器创建一个共享的目标文件。一旦创建了这个库，随后就要将它链接到图7-7的示例程序中：

```
linux> gcc -o prog2l main2.c ./libvector.so
```

这样就创建了一个可执行目标文件prog2l，而此文件的形式使得它在运行时可以和libvector.so链接。基本的思路是当创建可执行文件时，静态执行一些链接，然后在程序加载时，动态完成链接过程。认识到这一点是很重要的：此时，没有任何libvector.so的代码和数据节真的被复制到可执行文件prog2l中。反之，链接器复制了一些重定位和符号表信息，它们使得运行时可以解析对libvector.so中代码和数据的引用。

当加载器加载和运行可执行文件prog2l时，它利用7.9节中讨论过的技术，加载部分链接的可执行文件prog2l。接着，它注意到prog2l包含一个.interp节，这一节包含动态链接器的路径名，动态链接器本身就是一个共享目标(如在Linux系统上的ld-linux.so)。加载器不会像它通常所做地那样将控制传递给应用，而是加载和运行这个动态链接器。然