```
linux> gcc -c main2.c
linux> gcc -static -o prog2c main2.o ./libvector.a
```

或者等价地使用：

```
linux> gcc -c main2.c
linux> gcc -static -o prog2c main2.o -L. -lvector
```

图 7-8 概括了链接器的行为。-static 参数告诉编译器驱动程序，链接器应该构建一个完全链接的可执行目标文件，它可以加载到内存并运行，在加载时无须更进一步的链接。-lvector 参数是 libvector.a 的缩写，-L.参数告诉链接器在当前目录下查找 libvector.a。

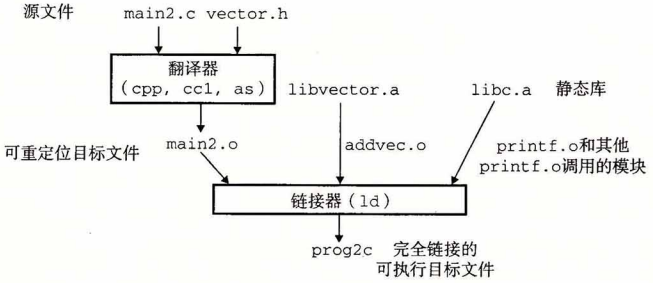

图 7-8 与静态库链接

当链接器运行时，它判定 main2.o 引用了 addvec.o 定义的 addvec 符号，所以复制 addvec.o 到可执行文件。因为程序不引用任何由 multvec.o 定义的符号，所以链接器就不会复制这个模块到可执行文件。链接器还会复制 libc.a 中的 printf.o 模块，以及许多C运行时系统中的其他模块。

7.6.3 链接器如何使用静态库来解析引用

虽然静态库很有用，但是它们同时也是一个程序员迷惑的源头，原因在于 Linux 链接器使用它们解析外部引用的方式。在符号解析阶段，链接器从左到右按照它们在编译器驱动程序命令行上出现的顺序来扫描可重定位目标文件和存档文件。(驱动程序自动将命令行中所有的.c 文件翻译为.o 文件。)在这次扫描中，链接器维护一个可重定位目标文件的集合 E(这个集合中的文件会被合并起来形成可执行文件)，一个未解析的符号(即引用了但是尚未定义的符号)集合 U，以及一个在前面输入文件中已定义的符号集合 D。初始时，E、U 和 D 均为空。

- 对于命令行上的每个输入文件 f，链接器会判断 f 是一个目标文件还是一个存档文件。如果 f 是一个目标文件，那么链接器把 f 添加到 E，修改 U 和 D 来反映 f 中的符号定义和引用，并继续下一个输入文件。
- 如果 f 是一个存档文件，那么链接器就尝试匹配 U 中未解析的符号和由存档文件成员定义的符号。如果某个存档文件成员 m，定义了一个符号来解析 U 中的一个引用，那么就将 m 加到 E 中，并且链接器修改 U 和 D 来反映 m 中的符号定义和引用。对存档文件中所有的成员目标文件都依次进行这个过程，直到 U 和 D 都不再发生变化。此时，任何不包含在 E 中的成员目标文件都简单地被丢弃，而链接器将继续处理下一个输入文件。