# 第7章

CHAPTER 7

# 链　接

链接(linking)是将各种代码和数据片段收集并组合成为一个单一文件的过程，这个文件可被加载(复制)到内存并执行。链接可以执行于编译时(compile time)，也就是在源代码被翻译成机器代码时；也可以执行于加载时(load time)，也就是在程序被加载器(loader)加载到内存并执行时；甚至执行于运行时(run time)，也就是由应用程序来执行。在早期的计算机系统中，链接是手动执行的。在现代系统中，链接是由叫做链接器(linker)的程序自动执行的。

链接器在软件开发中扮演着一个关键的角色，因为它们使得分离编译(separate compilation)成为可能。我们不用将一个大型的应用程序组织为一个巨大的源文件，而是可以把它分解为更小、更好管理的模块，可以独立地修改和编译这些模块。当我们改变这些模块中的一个时，只需简单地重新编译它，并重新链接应用，而不必重新编译其他文件。

链接通常是由链接器来默默地处理的，对于那些在编程入门课堂上构造小程序的学生而言，链接不是一个重要的议题。那为什么还要这么麻烦地学习关于链接的知识呢？

- 理解链接器将帮助你构造大型程序。构造大型程序的程序员经常会遇到由于缺少模块、缺少库或者不兼容的库版本引起的链接器错误。除非你理解链接器是如何解析引用、什么是库以及链接器是如何使用库来解析引用的，否则这类错误将令你感到迷惑和挫败。
- 理解链接器将帮助你避免一些危险的编程错误。Linux 链接器解析符号引用时所做的决定可以不动声色地影响你程序的正确性。在默认情况下，错误地定义多个全局变量的程序将通过链接器，而不产生任何警告信息。由此得到的程序会产生令人迷惑的运行时行为，而且非常难以调试。我们将向你展示这是如何发生的，以及该如何避免它。
- 理解链接将帮助你理解语言的作用域规则是如何实现的。例如，全局和局部变量之间的区别是什么？当你定义一个具有 static 属性的变量或者函数时，实际到底意味着什么？
- 理解链接将帮助你理解其他重要的系统概念。链接器产生的可执行目标文件在重要的系统功能中扮演着关键角色，比如加载和运行程序、虚拟内存、分页、内存映射。
- 理解链接将使你能够利用共享库。多年以来，链接都被认为是相当简单和无趣的。然而，随着共享库和动态链接在现代操作系统中重要性的日益加强，链接成为一个复杂的过程，为掌握它的程序员提供了强大的能力。比如，许多软件产品在运行时使用共享库来升级压缩包装的(shrink-wrapped)二进制程序。还有，大多数 Web 服务器都依赖于共享库的动态链接来提供动态内容。

这一章提供了关于链接各方面的全面讨论，从传统静态链接到加载时的共享库的动态链接，以及到运行时的共享库的动态链接。我们将使用实际示例来描述基本的机制，而且指出链接问题在哪些情况中会影响程序的性能和正确性。为了使描述具体和便于理解，我们的讨论是基于这样的环境：一个运行 Linux 的 x86-64 系统，使用标准的 ELF-64(此后称为 ELF)