嵌入式系统译丛

链接器和加载器

Linkers and Loaders

[美] John R. Levine 著
李勇 译

北京航空航天大学出版社

嵌入式系统译丛

链接器和加载器
Linkers and Loaders

[美] John R. Levine 著
李 勇 译

北京航空航天大学出版社

内 容 简 介

本书讲述构建程序的关键工具——链接器和加载器，内容包括链接和加载、体系结构、目标文件、存储分配、符号管理、库、重定位、加载和覆盖、共享库、动态链接和加载、动态链接的共享库，以及着眼于成熟的现代链接器所做的一些变化；并介绍一个持续的实践项目，即使用 Perl 语言开发一个可用的小链接器。

本书适合高校计算机相关专业的学生、实习程序员、语言设计者和开发人员阅读参考。

图书在版编目(CIP)数据

链接器和加载器/(美)莱文(Levine,J. R.)著;李勇译. —北京：北京航空航天大学出版社，2009.9

ISBN 978-7-81124-571-4

Ⅰ.链… Ⅱ.①莱…②李… Ⅲ.Linux 操作系统 Ⅳ.TP316.89

中国版本图书馆 CIP 数据核字(2009)第 154609 号

链接器和加载器

Linkers and Loaders

[美] John R. Levine 著

李 勇 译

责任编辑 张少扬 纪宁宁

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(100191) 发行部电话：(010)82317024 传真：(010)82328026

http://www.buaapress.com.cn E-mail:emsbook@gmail.com

涿州市新华印刷有限公司印装 各地书店经销

*

开本：787 mm×960 mm 1/16 印张：12.5 字数：280 千字

2009 年 9 月第 1 版 2009 年 9 月第 1 次印刷 印数：5 000 册

ISBN 978-7-81124-571-4 定价：28.00 元

版 权 声 明

北京市版权局著作权登记号：图字：01－2008－3397

Linkers and Loaders
John R. Levine
ISBN－13：978－1－55860－496－4

Authorized Simplified Chinese translation edition published by the Proprietor.
ISBN：978－981－272－180－8

Elsevier (Singapore) Pte Ltd.
3 Killiney Road
#08－01 Winsland House I
Singapore 239519
Tel：(65) 6349－0200
Fax：(65) 6733－1817

First Published 2009
2009 年初版

Printed in China by Beijing University of Aeronautics and Astronautics Press under special arrangement with Elsevier (Singapore) Pte Ltd.. This edition is authorized for sale in China only, excluding Hong Kong SAR and Taiwan. Unauthorized export of this edition is a violation of the Copyright Act. Violation of this Law is subject to Civil and Criminal Penalties.

本书简体中文版由北京航空航天大学出版社与 Elsevier (Singapore) Pte Ltd. 在中国大陆境内合作出版。本版仅限在中国境内(不包括香港和澳门特别行政区及台湾)出版及标价销售。未经许可之出口，视为违反著作权法，将受法律之制裁。

To Tonia and Sarah

译者序

最初接触《链接器和加载器》这本书是在2002年底，当时无意中在网络上找到了原作者公开的电子版，在非常欣喜和激动之余，用了一个多月时间将其读完。这本书系统地介绍了系统软件中程序的链接和加载技术在几种主要的操作系统和处理器平台上的实现和差异。对于系统软件开发人员，本书是不可多得甚至是唯一比较全面的参考资料。

2003年下半年，在China Linux Forum论坛上，由开源软件技术爱好者teawater和madsys发起了一个项目将这本书翻译为英文。考虑到之前我看过这本书，于是就加入到校对者的行列中。

项目开始半年多后，我逐渐从校对者转变为该项目唯一活跃的翻译者。在2004年11月，我将所有项目人员(teawater，madsys，coly，crazyd，keyboard，hgzw，dotemail，ddd)所提供的翻译文档整合成了一个版本，称为beta1，发布在China Linux Froum的C/C++编程版。beta1版本只是在数量上完成了所有章节，但是翻译质量尚有很大的改进空间。于是不久之后，我就开始着手将所有章节重新翻译一遍，力求进一步提高翻译质量。大概10个多月后，beta2即将完成之时，我的电脑硬盘坏掉了。很多翻译稿都没有及时备份，于是只好重新翻译。在2006年8月下旬，终于将所有章节重新翻译完毕，推出了beta2版本。

在beta2完成后的相当长一段时间内，翻译稿被放在网络上以收集大家的修改意见。在此期间，不少热心同行通过电子邮件或者China Linux Forum，指出了beta2中数量众多的翻译错误。这些同行朋友们，没有一个人留下自己的名字，我只知道他们在网络上的ID或电子邮件地址：Chen Wei <smartchenwei@sohu.com>，Li Lin <yjh521@gmail.com>，solarsea<solarsea@bupt.org>，shuyong <shuyong@linuxforum.net>，ZXJ <jameszxj@gmail.com>。正是由于上述热心同行们的鼎力支持，才大大提高了beta2版本的整体翻译质量。在这里请允许我表达对他们这种热心奉献的最诚挚敬意！

在北京航空航天大学出版社决定将《链接器和加载器》一书引入国内市场后，出版社的张少扬编辑又对beta2翻译稿进行了非常仔细和专业的编辑审校，修正了翻译稿中大

量隐藏许久的翻译错误，将整个文档的翻译质量提高到了一个新的层面。正是由于出版社的编辑和相关工作人员高水平的专业工作，才使得这个翻译稿达到了可以面市的程度。在此请允许我向张少扬等编辑表示衷心的感谢！

从最初开始翻译这本书，到正式出版，一转眼6年过去。经过无数IT同行的指点、帮助，这本书的翻译稿总算可以和广大读者见面了。但是有一点事实应当清晰地认识到，就是原作者John R. Levine先生对工具链领域有非常深刻的理解，本书在链接和加载方面的广度和深度，是独树一帜的。所有参与翻译本书的人在这方面知识的总和，也无法覆盖本书所有的内容。即便经过这些年的校对，本翻译稿中必然还存在大量这样或那样的翻译错误。如果读者认为该翻译稿中有吸引您的内容，这是本书原作者和众多翻译人员及编辑的功劳；如果读者在翻译稿中发现翻译错误或任何可以继续改进的地方，非常期待您将修改意见反馈给北京航空航天大学出版社(emsbook@gmail.com)。我们会在后续版本中持续地提高本书的翻译质量。

译　者

2009年8月

前　言

几乎从有计算机以来，链接器和加载器就是软件开发工具包中的一部分，因为它们允许使用模块（而不是一个单独的大文件）来构建程序的关键工具。

早在1947年，程序员们就开始使用原始的加载器：将程序的例程存储在多个不同的磁带上，并将它们合并、重定位为一个程序。在20世纪60年代早期，这些加载器就已经发展得相当完善了。由于那时内存很贵且容量有限，计算机的速度很慢（以今天的标准），为了创建复杂的内存覆盖策略（以将大容量的程序加载到小容量内存中），以及重新编辑先前链接过的文件（以节省重新创建程序的时间），这些链接器都包含了很多复杂的特性。

20世纪七八十年代，链接技术几乎没有什么进展。链接器趋向于更加简单，虚拟内存技术将应用程序和覆盖机制中的大多数存储管理工作都转移给了操作系统，越来越快的计算机和越来越大的磁盘也使得重新链接一个程序或替换个别模块比仅仅链接改变过的地方更加容易了。从20世纪90年代起，链接器又开始变得复杂起来，增加了诸多现代特性，包括对动态链接共享库的支持和对C++独特要求的支持。同时，像IA64那样具有宽指令字和编译时访存调度特性的先进处理器架构，也需要将一些新的特性加入到链接器中，以确保在被链接的程序中可以满足代码的这些复杂需求。

本书的读者是哪些人？

本书可供下述几类读者使用：

- 高校学生。通常，由于链接过程看起来似乎是微不足道的或显而易见的，所以在讲述编译器构建和操作系统的课程中对链接和加载都不够重视。这对于以前的语言Fortran、Pascal和C以及不使用内存映射或共享库的操作系统而言可能是对的。但是现在的C++、Java和其他的面向对象语言需要更加完善的链接环境，使用内存映射的可执行程序、共享库和动态链接将影响一个操作系统的很多部分，所以，一个忽略链接问题的操作系统设计者将面临巨大的风险。

- 实习程序员。实习程序员也需要知道链接器都做了什么(尤其是对现代语言)。C++的链接器中具有不少独特的特性,而大型C++程序之所以容易发生难以诊断的bug,也是由于在链接时发生了意外(最常见的情况是静态构造函数没有按照程序员预计的顺序执行)。当正确使用链接器时,其诸如共享库和动态链接一类的特性将给程序员的工作带来很大的灵活性,并给予强有力的支持。
- 语言设计者和开发人员。语言设计者和开发人员应该在构建语言和编译器时知道链接器做什么和能做什么。由于可以用链接器处理某些细节,那些手工进行了30多年的编程任务今天在C++中可以自动处理了(想一想程序员在C语言中为了获取和C++中的模板相同的功能,或为了保证在程序主体执行之前使成百个C源文件中的初始化例程可以执行,而不得不做的那堆事情吧)。有了功能更强大的链接器的支持,未来的语言将更加自动化而不仅限于程序范畴内的常规任务。由于链接是编译过程中将整个程序的代码放在一起处理,并可对程序作为一个整体施加影响的唯一阶段,因此链接器还将被加入更多的全局程序优化功能。

编写链接器的人当然也需要这本书。不过世界上所有的链接器设计者大概刚好能坐满一个房间,而他们中的半数因为审阅手稿已经拥有本书的副本了。

概 述

每章的关键点都用页边空白处的圆点标记出来,就像此段的标记一样。如果你想跳过一二章的话,那么在浏览中请至少阅读一下那些标记出来的重点。

第1章对链接的过程进行了一个简短的历史回顾,并对链接过程中的各个阶段进行了讨论。最后以一个由输入目标文件可产生"Hello,world"显示的简单却完整的链接器实例结束本章。

第2章从链接器设计的角度来回顾计算机体系结构。这里会以典型的精简指令集体系结构的SPARC,古老而富有活力的寄存器-内存体系结构的IBM 360/370,以及自成一派的Intel x86体系结构为例。在每种体系结构中,都讨论包括内存架构、程序寻址架构和地址布局等重要方面。

第3章探究目标文件和可执行文件的内部结构。本章从最简单的DOS的COM文件开始,进而分析更加复杂的文件,比如DOS的EXE、Wondows的COFF和PE(EXE和DLL)、UNIX的a.out和ELF,以及Intel/Microsoft的OMF。

第4章涵盖了链接过程的第一阶段(即为被链接程序的各段分配存储空间),并以实际的链接器来举例说明。

第5章涵盖了符号的绑定和解析，这是将某个文件中引用的另一个文件的符号解析为机器地址的过程。

第6章涵盖了目标代码库的创建和使用，以及库的结构和性能。

第7章涵盖了地址重定位，即将程序内的目标代码调整为可以反映其运行时实际地址的过程。这里谈到了位置无关代码(PIC)(这种代码是通过无需重定位的方法建立的)，并讨论了这种方法的优势和代价。

第8章涵盖了加载过程，即将程序从文件中取出装入计算机内存中运行。本章还讨论了一种古老而有效的节省存储空间的树结构覆盖技术。

第9章讨论了在不同程序中共享一个库代码的单一拷贝需要做哪些工作。本章将注意力放在静态链接的共享库上。

第10章将第9章的讨论延伸至动态链接的共享库。本章详细说明了两个实例，Win32的动态链接库(DLL)和UNIX/Linux的ELF共享库。

第11章着眼于成熟的现代链接器所做的一些变化。本章讨论C++需要的新特性，包括编译命名格式(name mangling)、全局构造与析构函数、模板扩充和冗余代码的消除，此外还包括如增量链接、链接时垃圾收集、链接时代码生成和优化、加载时代码生成以及统计和性能监视。本章对Java的链接模式进行了简要阐述，相比于本书涉及的其他链接器，该模式更具有语义上的复杂性。

项 目

第3～第11章有一个持续的实践项目，是一个用Perl语言开发的小而可用的链接器。尽管Perl不太适合实现产品级链接器时所使用，但对一个学期内要完成的项目而言却是很好的选择。Perl处理了很多在C/C++编程中会拖延编程进度的低层编程细节，好让学生把精力集中在当前项目的算法和数据结构上。在当前很多的计算机上，Perl是免费的，包括Windows 95/98和NT、UNIX及Linux，并且还有很多非常优秀的书籍指导新手使用Perl(可参看参考文献中的书籍)。

第3章最初的项目构建了一个以简单而完整的目标代码格式读取和写入文件的链接器框架。后续章节不断地向它添加功能，直到最后变成一个支持共享库并产生可动态链接目标代码的功能完善的链接器。Perl善于处理二进制文件和数据结构，如果愿意的话，项目链接器可以用来处理它自己的目标代码格式。

项目中的目标文件可以从本书的支持网站linker.iecc.com中找到。

致 谢

有许许多多人慷慨地牺牲了自己的时间来阅读和评论本书的手稿，包括出版商的评论家和 comp. compilers usenet 新闻组中阅读和评注本书在线版本的读者们。下面以姓氏字母顺序列出他们的名字：Mike Albaugh、Rod Bates、Gunnar Blomberg、Robert Bowdidge、Keith Breinholt、Brad Brisco、Andreas Buschmann、David S. Cargo、John Carr、David Chase、Ben Combee、Ralph Corderoy、Paul Curtis、Bjorn De Sutter、Lars Duening、Phil Edwards、Oisin Feeley、Mary Fernandez、Michael Lee Finney、Peter H. Froehlich、Robert Goldberg、James Grosbach、Rohit Grover、Quinn Tyler Jackson、Colin Jensen、Glenn Kasten、Louis Krupp、Terry Lambert、Doug Landauer、Jim Larus、Len Lattanzi、Greg Lindahl、Peter Ludemann、Steven D. Majewski、John McEnerney、Larry Meadows、Jason Merrill、Carl Montgomery、Cyril Muerillon、Sameer Nanajkar、Jacob Navia、Simon Peyton-Jones、Allan Porterfield、Charles Randall、Thomas David Rivers、Ken Rose、Alex Rosenberg、Raymond Roth、Timur Safin、Kenneth G Salter、Donn Seeley、Aaron F. Stanton、Harlan Stenn、Mark Stone、Robert Strandh、Ian Taylor、Michael Trofimov、Hans Walheim 和 Roger Wong。

本书中正确部分是他们努力的结果。而本书中错误部分都由作者负责（如果您发现了书中错误，请用下面的地址联系本人以便在后续版本中更正这些错误）。

我特别要感谢出版商 Morgan Kaufmann 的两位编辑 Tim Cox 和 Sarah Luger，他们容忍我写作过程中无限期的拖延，并把本书中支离破碎的章节拼凑在一起。

联系我们

本书有一个支持网站 http://linker.iecc.com，上面包含了本书的样章、工程项目的 Perl 代码和目标文件示例，还有本书的更新和勘误。

您可以通过 linker@iecc.com 给作者发送 e-mail。作者会阅读所有的来信，但可能因收信量过大而无法及时回复所有的问题。

目　录

第6章　库

第7章　重定位

第8章　加载和覆盖

第9章　共享库

第1章

链接和加载

1.1 链接器和加载器做什么?

任何一个链接器和加载器的基本工作都非常简单:将更抽象的名字与更底层的名字绑定起来,好让程序员使用更抽象的名字编写代码。也就是说,它可以将程序员编写的一个诸如getline的名字绑定到"iosys模块内可执行代码的612字节处",或者可以采用诸如"这个模块的静态数据开始的第450字节处"这样更抽象的数字地址,并将其绑定到数字地址上。

1.2 地址绑定:从历史的角度

链接器和加载器是做什么的?深入理解的方法就是看看它们在计算机编程系统的发展中充当了什么角色。

最早的计算机完全是用机器语言进行编程的。程序员需要在纸质表格上写下符号化的程序,然后手工将其汇编为机器码,通过开关、纸带或卡片将其输入到计算机中(真正的高手可以在开关上直接编码)。如果程序员使用符号化的地址,那他就得手工完成符号到地址的绑定。如果后来发现需要添加或删除一条指令,那么整个程序都必须手工检查一遍,并将所有被添加或删除指令影响的地址都进行修改。

这个问题就在于名字和地址绑定得过早了。汇编器通过让程序员使用符号化名字编写程序,然后由程序将名字绑定到机器地址的方法解决了这个问题。如果程序被改变了,那么程序员必须重新汇编它,但是地址分配的工作已经从程序员转给计算机了。

代码的库使得地址分配工作更加复杂。由于计算机可以执行的基本操作极其简单,有用的程序都是由那些可以执行更高级、更复杂操作的子程序组成的。计算机在安装时都带有一些预先编写好、调试好的子程序库,程序员可以将它们用在自己编写的新程序中,而勿需编写所有的子程序。然后程序员可以将这些子程序加载到主程序中以构成一个完整的可以工作的

程序。

程序员们甚至在使用汇编语言之前就使用子程序库了。在 1947 年，领导 ENIAC 项目的 John Mauchly 就写文章描述了如何将主程序和磁带中一系列选定的子程序一起加载到计算机中，并通过将子程序代码重定位以反映实际被加载的地址。鉴于 Mauchly 认为程序和子程序都是由机器语言编写的，因此我们可能会惊奇地发现，甚至在汇编语言出现之前，链接器的两个基本功能重定位和库查询就已经出现了。可重定位的加载器允许子程序的作者或用户在编写子程序时认为它们都起始于地址 0，并将实际的地址绑定延迟到这些程序被链接到某个特定的程序中时。

随着操作系统的出现，有必要将可重定位的加载器从链接器和库中分离出来。在有操作系统之前，一个程序可以支配机器所有的内存，由于知道计算机中所有的地址都是可用的，因此它能以固定的内存地址来汇编和链接。但是有了操作系统以后，程序就必须和操作系统甚至其他程序共享计算机的内存。这意味着在操作系统将程序加载到内存之前是无法确定程序运行的确切地址的，并将最终的地址绑定从链接时推延到了加载时。现在链接器和加载器已经将这个工作划分开了，链接器对每一个程序的部分地址进行绑定并分配相对地址，加载器完成最后的重定位步骤并分配实际地址。

随着计算机系统变得越来越复杂，链接器被用来做了更多、更复杂的名字管理和地址绑定的工作。Fortran 程序使用了多个子程序和公共块（被多个子程序共享的数据区域），并由链接器来为这些子程序和公共块进行存储布局和地址分配。链接器不得不更多地处理目标代码库，包括用 Fortran 或其他语言编写的应用程序库，以及编译器支持的库（这些可以从被编译好的处理 I/O 或其他高级操作的代码中隐含调用）。

程序占用的空间很快就变得比可用的内存大了，因此链接器提供了覆盖技术，它可以让程序员安排程序的不同部分来分享相同的内存，当程序的某一部分被其他部分调用时可以按需加载。1960 年前后磁盘出现后，覆盖技术在大型主机系统上得到了广泛的应用，一直持续到 20 世纪 70 年代中期虚拟内存技术出现；在 80 年代早期，以几乎相同的形式在微型计算机上出现，并在 90 年代由于 PC 上采用虚拟内存后逐渐没落。现在它们仍用于内存受限的嵌入式环境中，并且为了提高性能，当程序员或者编译器需要精确的控制内存使用时可能会再次出现。

随着硬件重定位和虚拟内存的出现，每一个程序可以再次拥有整个地址空间，因此链接器和加载器变得不那么复杂了。由于硬件（而不是软件）重定位可以对任何加载时重定位进行处理，程序可以按照被加载到固定地址的方式来链接。但是具有硬件重定位功能的计算机往往不只运行一个程序，而且经常会运行同一个程序的多个副本。当计算机运行一个程序的多个实例时，程序中的某些部分在所有的运行实例中都是相同的（尤其是可执行代码），而另一些部