最新UNIX系统V丛书（共五册）

程序员编程指南

（二）

H

北京中国科学院希望高级电脑技术公司

前　言

1.目的

本指南提供了关于UNIX系统环境中程序设计的资料，本书并不试图教会读者如何编写程序，而是希望把重点放在使程序运行的其它一些因素上来补充程序设计语言文本。

2.读者和预备知识

正如书名所说，本指南为程序设计者，尤其是那些尚未广泛地使用UNIX系统的程序设计者们所编写，并假设没有特殊的程序设计内容。希望本书不仅适用那些从事或管理大型的应用发展项目的人们，而且对那些仅编写个别临时程序的人们也有用。

程序设计专家、以及那些从事于发展系统软件的程序设计员们可能会发现本指南与他们的要求相比，该指南的内容深度不够，建议他们参阅《程序员参考手册》（P—H）。

本书假设读者已具有终端使用，UNIX系统编辑以及UNIX系统目录／文件结构方面的知识，如对这些基础工具掌握得不扎实，在接触本书前应先看完《用户指南》（P—H）。

3.编排

本指南编排为两部分，共十七章。内容如下：

第一部分　第一章　概况

定义建立程序设计环境的UNIX系统的特殊功能：建立块、管道、特殊文件、SHELL程序设计等等概念。作为今后内容的一个框架，定义了UNIX系统程序设计中的三个不同层次，单用户、应用和系统程序设计。

第二章　程序设计基础

介绍了使程序运行所需的最基本的设备。

第三章　应用程序设计

详述了许多在前面章节中叙述的内容，强调了当项目发展时，怎样进行改变，介绍了使程序设计有条理的工具。

第二部分　第四至第十七章　支撑工具及其说明和使用指导

包括了许多关于UNIX系统工具应用的详细资料

本书末尾附有AT＆T3B2计算机的命令表、术语表及索引。

4.与C语言的关系

UNIX系统支撑许多程序设计语言，而C编译程序可用于许多不同的操作系统，尽管如此，UNIX操作系统与C语言之间自始至终保持着密切的关系。在UNIX操作系统中，绝大多数代码是C语言，而且几年来许多采用UNIX系统的组织在他们的应用代码中也增加了使用C语言的比例。因而，你将发现，本指南希望在对你所用何种语言无关紧要时，所举例子

均采用C语言，切非指定了某特定的相关语言。

5.硬件／软件相关性

本书反映的是用3.0版本，在AT&T3B2计算机上运行UNIX系统的情况，如果发现在你的UNIX系统环境下运行命令有所不同，可能是由于在不同的软件版本中运行。如果某些命令象是根本不存在，在你的系统上可能没安装命令包中的一部分。附录A介绍了在AT&T3B2计算机上可以使用的命令包，如确实发现你自己在试图执行一个不存在的命令时，对照附录A，并与系统管理员对话。

6.符号约定

全书任何从计算机中输出和／或由用户输入的命令的例子，都遵循UNIX系统文件公共的标准符号方法：

·从终端键入的命令用黑体字表示

·在终端上由计算机打印的文字用定宽字型表示。定宽字型还用于代码样本，因为它能最精确地代表空格，空格常常是一种代码方式，有时是很重要的。

·加到显示上的注释用斜体字，以表示这一部分在显示上没有，以便把它们从表示计算机的输入与输出的文字中区分出来，解释输入或输出的注释用和其它显示相同的字体表示。

当命令格式出现时，斜体字还用作表示替换值，如filename（文件名）。

·在每一个输入的命令和菜单响应的末尾有一个隐示的RETURN，在需要只输入一个RETURN时（如在选取菜单的缺省值的情况下），用符号〈CR〉。

·在需要输入一个控制字符的位置，正如CTRL—D表示，其意义是在按CTRL键的同时再按下D键。

·美元符号$和英镑符号#是对普通用户和各自根（rout）的标准缺省提示符。$意味着被联机为普通用户，#意味着被联机为根用户。

·当#提示符用于一个例子中时，意味着这一被说明的命令可能仅供根（rout）用户使用。

7.命令参考文献

8.举例中的信息

虽然已尽全力使信息的显示与在终端上的相同，但有可能你的系统会产生出有微小差异的输出。某些显示依赖于某一具体特定的机器，这可能与你所用的不同，UNIX系统软件的版本间的变化可能引起在终端上显示的微小差异。

本书给出完整的代码样本，已试验过并确保象表示的那样编译和运行。对给出的代码片断，虽不能说对他们编译过，但对它们尽量保持了相同的准确编码的标准。

目　录

第一部分　程序设计

第一章　UNIX系统环境下的程序设计

1.1 引　　言

1983年计算机协会（Association for Computing Machinery）的图灵奖（Turing Award）授与了Ken Thompson和Dennis Ritchie。这两位是首先设计并发展了UNIX操作系统的人，在授奖会上有这样一段话："UNIX系统的成功在于它对几个关键观念的有鉴赏力的选择以及他们的精巧应用，UNIX系统的模式把一代软件设计者引到了考虑程序设计的新途径上，UNIX系统的才华是它能使程序员以其他成果为根据作为它本身的结构。"

作为在UNIX系统环境中工作的程序员，为什么应该关心Thompson和Ritchie做了什么呢？它与它们当前有何关系呢？

有的，因为如果我们懂得了这一系统设计的思想和它成熟的环境，可以帮助我们更快地成为多产的UNIX系统程序员。

1.1.1 回顾过去

你可能已经读过关于Ken Thompson在AT&T公司贝尔实验室(Bell laboratories)的过道里如何碰见一台不用的DEC PDP—7计算机，他和Dennis Ritchie以及他们的几个同事又如何把这台计算机作为原始机发展了一个成为UNIX的新的操作系统的故事。

然而，他们认识到重要的事情是所做的努力是为他们自己创造了一个愉快的计算环境，而不是"让我们到一起，来创立一个将引起全世界都关注的操作系统。"

最有意义的是系统元素放入的序列。第一层为文件系统，文件系统很快地把它放入文件和目录的一层里，作为某一类型的文件它包含所有的内容、数据库、程序、命令、目录甚至设备都是很重要的，一个文件的概念是和字节的一个一维数组一样，而没有其他隐示结构，这种查阅文件方式的清晰与简单性是使程序员和其他用户工作在舒适的计算机环境里的主要因素。

下一个是进程的概念，一个进程能产生另一个进程并与之通讯。运行作为进程的程序，这是一创造性的工作方式，这使得靠调用另外一个进程来重复使用代码很容易（UNIX的精华）。利用附加的命令操作文件和汇编程序来产生可执行程序，基本上系统本身就能够起作用。

另一个主要的发展是在DEC PDP—11机上获得的，是在该机上安装了新的系统。Ritchie把这说成是走了好运，因为PDP—11已成为非常成功的机器，它的成功，在某种程度上扩大了UNIX的名声而促进人们接受这一系统。

到1972年，管道（pipes）（在进程之间，把一个进程的输出变成另一进程的输入的连接部分）的概念被引入到这一系统，操作系统重新用更高级的语言编码（先是B，后是C），

并取名为UNIX（由Brian Kernighan定名）。到了这时，Thompson和Ritchie所寻求的“愉快的计算机环境”成为现实。但无论当时还是现在，某些其它的事情继续对产品的特性产生很强的影响。

从1971年开始，系统开始在AT&T贝尔实验室内应用，短短几年后（1974年），在低成本和没有院校支持下变得可用了。这些版本被称为研究版本，并有1～7的阿拉伯字定义。通常由它们自己生成并反馈到主系统添加到更新工具中，例如：广泛使用的屏幕编辑Vi(1)，就是由伯克利的加利福尼亚大学的William Joy添加到UNIX系统中去的。

1979年根据商品化的要求，AT&T开始提供支持UNIX系统的版本（称为开发版本），这些由罗马数字定义，并经常有临时发行版的数字扩充。例如：当前开发的版本是UNIX系统V3.0版。

现在由AT&T提供的UNIX系统版本来自于一个标准软件工厂，根据增加新的发行版本的特点来说明市场的需要，然而使得UNIX系统的基本特性，仍然是创新的产品，研究它的起因和它们工作的学院环境。有时这些特性被看作为UNIX的基本原理，不管怎样，这意味着高级程序员已开始工作于UNIX系统中。

1.1.2 简述UNIX系统基本原理

如果你正用UNIX系统编写程序，你就应该将这个格言挂在墙上：

把工作建立于别人之上！

与计算机环境不同，每一个新标题用小黑块开始，在一个UNIX系统上，程序设计好的有效部分在于bins、lbins和/usr/bins，更不用说可扩充的编译程序语言等都是很好使用。

UNIX系统的性能（管道进程和文件系统）在于这种重新使用能力，共享和进行扩充的历史可追朔到1969年，如果不把这些用于工作上，你将冒失去UNIX系统基本性质的风险。

1.2 UNIX系统工具及其查阅

对“UNIX系统工具”这个术语要进行一些阐述，最狭义的含意是：应用存在的软件作为一个新任务的组成部分。较广义地理解，从上下文关系所决定的含意是：通常看作UNIX系统的元素，被称为特征性、实用性、程序、筛选程序、命令、语言、函数等等。因为被称为这些名字的有一种或多种事物，从而使它变得混乱，所以通常用狭义理解作为程序设计中的一部分。

1.2.1 本指南中包括和不包括的工具

《程序员指南》一本是关于在UNIX系统环境中进行程序处理工具的使用。因此，首先让我们谈谈关于工具的含义，在这本手册中某些将不被包括，而不包括的这些资料在其他手册中都能找到。实际上，某些对你来讲可能是比较重要的工具，在本指南中也没包括，因为不可能在本手册中全部包括你需要的关于UNIX系统工具的每一件事。

1.2.2 本书中不包括的工具

1. login过程
2. UNIX系统编辑及其使用
3. 如何组织文件系统及其运行
4. shell程序设计

关于这些标题的资料在《用户指南》及许多有关的商业性书中都能找到。

这里包括的工具分类如下：

1.获得程序运行的实用性

2.组织软件开发项目的实用性

3.特殊语言

4.调试与分析工具

5.编译不是语句部分的语言成分，如：标准库、系统调用、函数等。

1.2.3 作为样机工具的shell

任何时候都应对UNIX系统机使用的shell注册。shell是作为用户与UNIX系统核之间的互相作用，解释这种状态下的命令。但这仅仅是对它的部分描述，因为它的性能可作用于进程、直接控制流向、段中断和间接的输入与输出，它是一种成熟的程序设计语言，使用这些性能的程序就是著名的shell过程或shell原本。

shell的大多数创新用途包括在shell原本控制下运行集合在一起的命令串，该方法使用的很多命令在《用户参考手册》中提供，花费时间阅读《用户参考手册》是值得的。当你试图找一条用正确的可选开关处理一个难解决的程序设计问题时，就请仔细查阅该手册。你越熟悉该手册中介绍的命令，就越能够掌握UNIX系统环境中的全部特征。

这里，介绍shell程序设计并不是我们的目的，我们强调的重点是：shell过程能运行的全部应用开发样板，当懂得所有shell程序设计的细微差异时，就能清楚一个复杂的任务，得到一个shell过程及运行，它比编写、编译及调试编译码所花费的大量时间要远远少得多。

这种使程序快速地变成产品的能力，使shell成为程序开发一个有价值的工具。shell程序设计允许你最大可能地“建立在其他人工作之上”，因为它允许你拼合简单和有效的主要成份。多次甚至较大的应用是作为测试目的而开发的一种样板系统，也愿花费几十个月的工作使它成为产品。

为了测试一台样机，确定用户出错的可能范围——当设计一种应用时，有些事情不是那么容易安排，采用处理陌生用户输入的方法可设计出避免大量的再生码方法。

在UNIX系统环境里一个经常的事就是找到一个有效的UNIX系统工具，利用一页和编译码的一半完成两行指令。shell过程能混合编译参数和有规律的UNIX系统命令，让你利用以前做的工作。

1.3 三个程序设计环境

我们用强调信息需要和在不同环境下使用UNIX系统工具的方法来区别三个程序的设计环境，我们不打算介绍技巧或经验性内容。在“单用户”范畴里，能找到有几十年经验的高级程序员以及相对于新手可能是一个应用开发或系统程序设计组的成员。

1.3.1 单用户程序员

在这环境中的程序员仅编写容易完成他们的初步工作的程序，结果程序正好被加到程序员协同工作的有效程序贮存里，这相似于UNIX系统繁荣时的环境，人们开发一个有用的工具并让其它组织一起共享它。单用户程序员不会有外界强加的要求或合作者，或涉及到项目管理。程序设计任务的本身是非常直接地驱动代码，分时系统的优点之一，如UNIX是人们

用程序设计技巧能自由地安排他自己的工作，不须要通过一般的计划批准通道而为一个程序设计开发解决他们的问题来等候几个月。

单用户程序员需要懂得：

1.选择一种适当的语言

2.编译和运行程序

3.应用系统库

4.分析程序

5.调试程序

6.记录程序版本

完成这些功能的绝大多数信息在第二章的“单用户水平”节中能找到。

1.3.2 应用程序设计

在这环境里工作的程序员是为其他的非程序设计的用户而开发系统，绝大多数大型计算机的应用仍将包含一群应用开发的程序员，他们会是目的用户组织的雇员或成为软件开发公司工作。在这环境中工作的某些人的确是个程序员，只是在计划管理范围内。

在这环境里的人们所需要的资料包括第二章的所有标题，附加下列资料：

1.软件控制系统

2.文件和记录锁定

3.进程间的通讯

4.共用存贮器

5.先进的调试技术

这些标题在第三章中讨论

1.3.3 系统程序员

系统程序员是那些从事于编写部分的或同操作系统本身很相近的软件工具的程序员，其目的可包括编写一个新的设备驱动程序、一个数据库管理系统或增强了UNIX系统核心部份。除懂得用操作系统周围的原码的方法以及怎样使它变化与增强外，他们还必须详尽地熟悉第二与第三章中的所有标题。

1.4 摘 要

在这概述章里，我们已介绍了UNIX系统开发状况和影响程序员现在用它工作的因素。我们已介绍了对于程序员从本指南其他章中什么是能找到的，什么不能找到。我们还建议在许多情况下的程序设计问题靠利用UNIX系统与众所周知的shell互相作用的命令解释程序容易地解决，最后，我们定义了三个程序设计环境，希望它会帮助读者适应以后章节的编制。

第二章　程序设计基础

2.1 引　言

本章的内容是为那些仅学习编写在UNIX系统环境中运行程序的人们安排的。在第一章，我们已定义了UNIX系统用户中的一部分为单用户程序员，属这一类的用户，特别是那些对程序设计不是十分感兴趣的，将发现本章（加上相关的文件手册）介绍了他们所需要了解的关于在UNIX系统计算机上编码和运行程序足够的信息。

2.2 选择程序设计语言

在一个给定的状态，怎样确定采用那一种程序语言呢？一种回答可能是“我总是用HAIRBOL编码，因为它是我最熟的语言”。的确，在某些情况下，这是一个合理的回答。但是，假定你可以采用的程序设计语言不只一种，而不同的程序设计语言有他们的优点和缺点，再设想一旦你学会了应用一种程序设计语言，再学习应用另一种相应地变简单了，问你自己下面的问题，你可能会接近语言的选择这一问题：

该程序要完成的任务的本质是什么？

这一任务需要发展一个复杂的算法，还是一个在大量的记录上完成的一个简单的过程？这一程序设计任务是否有许多分离的部分这个程序能再分成分别可编译程序的几个函数、还是只有一个模型？何时要这个程序必须编完？是现在就需要呢，还是有足够的时间来作出可能最有效的处理？程序的应用范围是什么？将使用这个程序的仅仅是你一个人呢，还是这个程序将要传遍于全世界？这个程序是否有可能将要移植到另外的系统上？这个程序的预计寿命为多少？将采用几次呢？是从现在起用足五年吗？

2.3 UNIX系统环境中的支持语言

用“支持的语言”，指的是那些由AT&T提供的用在AT&T3B2运行UNIX系统V3.0版本的计算机上的语言。由于这些是可以分别买到的项目，所以不必要所有这些都装配在你的机器上。另一方面，或许在你的机器上可以使用来自其他来源而这里没有提到的语言。尽管如此，在本节和下一节，我们将对a)六种全功能程序设计语言，和b)一些特殊用途语言的特征作一简要的介绍。

2.3.1 C 语 言

由于C语言最初是被用来重新编译UNIX系统内核的，所以它是与UNIX系统紧密相联系的。如果需要使用大量的UNIX系统功能调用，低级I/O，存贮或设备管理，或进程相互间通讯，C语言理所当然是第一选择。然而，绝大多数程序不要求直接这样与操作系统联接，因此，在选择C语言时最好根据下面的一个或几个性质：

1. 多种的数据类型：字符、整数、长整数、浮点、双精度。
2. 低级结构（绝大多数UNIX系统内核用C语言写成）

3. 派生数据类型，诸如数组、函数、指针、结构和组织。

4. 多维数组

5. 标记了的指针和作指针运算的能力。

6. 按位算符

7. 流程控制语句的变化：if，if—else,switch,while,do—while和for。

8. 高度的可移植性

C语言早已是适合于结构化程序设计的一种语言。在C语言中根据功能考虑是自然的。逻辑上的下一步是把每一个功能看作一个隔离开的可编译单元，这一方法（用些小程序块编译一个程序）简化了修改的工作。如果这种开始听起来象把新程序建立在已有的手段之上的UNIX系统之哲学，并不仅仅是巧合。在你为某一程序创造某些功能的同时你会发现，这些功能的许多可以被舍取或迅速修改，用到另外一个程序上。

利用C语言的一个最大难题是，必须使C语言程序员达到所需的足够能力，这需用几个月的时间非常集中地使用该语言。如果你是个暂时程序员，选择一种要求低一些的语言可能使你少遇到些困难。

2.3.2 FORTRAN

FORTRAN是最早的高级程序设计语言，由于它的运算功能的多变性而现在仍受到人们的赞赏。如果要为统计分析或其它科技应用编写一个程序，FORTRAN语言是一个很好的选择。最初的设计目标是要产生一种具有操作高效益的语言，以牺牲类型定义和数据抽象领域里的灵活性为代价而达到了这一目标。

例如，仅仅只有一种形式的迭代语句。FORTRAN还要求用一种几乎固定的格式输入源编码行。通过利用为使FORTRAN语言更灵活而设计的一种UNIX系统手段，可以克服这一不足。

2.3.3 Pascal

Pascal最初是为块结构化程序设计而设计的一种教学手段。由于Pascal的一直向前的风格它已得到了很广泛地接受。Pascal是一种高度结构化的允许系统级调用（和C语言相同的特点）的语言。然而，由于该语言的开发者们的目的是产生一种教给人们设计程序的语言，所以它可能最适合于一些小的项目。它不方便的地方还有缺少对变量指定初始值的功能和有限的文件处理能力。

2.3.4 COBOL

可能更多的程序员对COBOL比对其它任何一种程序设计语言更熟悉。由于它的重点放在了输入/输出的管理以及确定记录的格式上，所以它常常被用在商业应用上。

对于复杂的算法采用COBOL语言有点繁锁，但当许多记录要通过某简单的过程时，它还是很有效的。

例如，扣留了税算值的工资单。

这是一种应用起来非常冗长的语言，因为每个程序仅仅在描述记录格式，处理环境和在编译而用的变量方面就需要很长的文字。由于COBOL语言冗长，所以编译过程常常相当复杂。一旦被用来编写并进入使用，COBOL程序将保留使用几年，而且某些人把文字认为是自己的文献在程序员时间上的投资往往使他们拒绝改变。

2.3.5 BASIC

最常听到的对BASIC的评论是容易学。随着个人微型机的普及，许多人学习了BASIC语言，因为在很短的时间里就可编成一个可运行的程序。然而，对大型程序设计课题采用BASIC是困难的。它缺乏结构化的控制流程，要求每个变量都定义在整个程序中，并且没有办法在函数与调用程序之间进行数值交换。BASIC的大多数版本执行解释编码而不是编译。它编写较慢运行的程序。然而，尽管它的局限性，对于很快地使简单的过程投入运行，BASIC语言是很有用的。

2.3.6 汇编语言

作为最接近于机器语言的汇编语言，对于某特定的在其上运行你的程序的计算机是专用的。作为编译过程，高级语言有些被翻译成为某特定处理机的汇编语言。需要用汇编语言的最常见的情况是产生于当你要作某些不在高级语言范围之内工作的时候。由于汇编语言是一种机器专用化的语言，用它写的程序不能移植。

2.3.7 特殊目的的语言

除了上面的正式程序设计语言外，UNIX系统环境常提供一种或几种下面所例举的特殊目的的语言。

注意：由于UNIX系统应用程序和命令是以功能分组装配的，有可能所提到的设备不是在所有的系统上全部都可以得到。

2.3.7.1 aWK

awk（这一名字是它的开发者名字的第一个字母构成）是查找输入的文件中符合某规范文件中描述的类型的那些行，对于所找到的符合某类型的一行，aWK所执行的动作也描述在这一规范中。一个用几行所编写的aWK程序，它所完成的功能用FORTRAN或C这样的程序设计语言要用几张纸才能描述完，这种情况并非不常见。例如，考虑这种情况，你有一批记录，这些记录由一个键区和另一个代表量的区组成。你已根据键区把这些记录进行分类，而现在你要通过完全相同的键来增大记录的量值，并要输出一个设有重复键的一个文件，这样一个程序的伪码看上去可能象这样：

```
把第一个记录读入保持区；
读附加的记录直到EOF；
{如果键与在保持区中记录的键相匹配，把这个量值加到被保持的记录的量值区域上。
如果键与保持在记录中的键不对应，写下这个保持的记录。
把新的记录移到保持区}
到EOF，从保持区写出最后的记录。一个aWK程序来实现这一任务将象这样
        {qty〔$1〕+=$2}
    END {for(keginqty)Printkeg,qty〔key〕}
```

这仅仅说明了aWK的一个特性：它用关联数组的能力。利用aWK，输入的文件不必要分类，这是程序的一个要求。

2.3.7.2 lex

lex是一个加在FORTRAN或C语言程序中的词汇分析程序。词汇分析程序关注的是语言的词汇而不是语法、语法是定义语言的结构的一系列的规则。lex能够产生用于鉴别用户指定的正规表达式的C语言子程序。

当一个正规表达式被鉴别出来时采取某些动作，并把输出流送到下一个程序。

2.3.7.3 Yacc

Yacc (Yec Another Compiler Compiler)是把一种输入语言描述成计算机程序的一种工具。Yacc产生的C语言子程序根据在规范文件中的规则分析输入流。Yacc规范文件设立了一套语法规则连同输入的特征与规则相符合时所要采取的动作。lex可以同Yacc用来控制输入过程和把特征送到应用语法规则的分析程序中。

2.3.7.4 M_4

M_4为可以用作汇编语言和C语言程序的预处理程序的宏处理程序。在《程序员参考手册》的第(1)节中介绍了M_4。

2.3.7.5 bc和dc

bc使你能够使用一个计算机终端就象使用一个可编程序计算器一样。你可以编辑一个数学计算的文件并调用bc来执行。bc程序使用dc。如果愿意你可以直接使用dc,但是由于它利用的是逆波兰表达式，所以不易习惯。直接使用dc意味着你把数字输到子栈中，而运算器跟在其后。在《用户参考手册》的第(1)节中，介绍了bc和dc。

2.3.7.5 Curses

Curs s实际上是一个C功能库。Curses被包括在这一表中是因为这一套功能几乎相当于处理终端屏幕的子语言。如果你编写的程序包含有交互用户屏幕。你将愿意熟悉这组功能。

除了上述的全部内容外，不要忽视了利用Shell过程的可能性。

2.3.8 编码之后

UNIX系统环境中，大多数编译系统的最后两步是汇编程序和连接编辑程序、编译系统产生汇编语言代码，汇编程序把代码翻译成程序所运行的计算机上的机器语言，连接编辑程序判定所有的没有定义的符号并使得目标模块可以执行。用UNIX系统上的大多数语言，汇编程序和连接编辑程序以我们所熟悉公共目标文件格式（COFF）产生文件。公共格式使得应用更简单了，这些应用取决于那些在不同的机器上运行不同版本的UNIX系统的目标文件中的信息。

在公共目标文件格式下一个目标文件中包括：

1. 文件标题
2. 可选择的副标题
3. 节标题表
4. 与节标题相对应的数据
5. 重新分配的信息
6. 行数
7. 符号表
8. 字符串表

一个目标文件由节组成。通常至少有二节：tcxe和data有些目标文件包括称为bss的一节（bss是一个汇编语言的伪操作最初代表“由符号开始的块”）。当bss出现时，保持专预置的数据，编译的多种选择导致了不同项目的信息包括在公共目标文件格式中。例如，用一种选择编译一个程序，加上行数和Sdb(Sgmbohc Debugger)命令就可以所需要的其它符号信息充分生效。

你可以从事程序设计许多年而无须过分担心公共目标文件格式的内容和组织。因此我们对这一点不再进一步作更详细的介绍。详细的材料见本指南的第11章。

2.3.9 编译和连接编辑过程

用于编译的命令取决于采用的语言：

1. 对于C语言程序，CC编译并连接编辑。
2. 对于FORTRAN语言程序，f77编译并连接编辑。

2.3.9.1 编译C语言程序

采用C语言编译系统，必须使文件中的源码有文件名并以符号C为后缀，如My code.C.（中的.C一样）。

调用这一编译程序的命令为：

```
CC My Code.C.
```

如果编译成功，过程进一步通过连接编辑那一级，其结果将是一个以a.out命令可执行的文件。

对于CC命令有几种选择来控制它的运行，最常用的选择是：

−C　引起编译系统阻止连接编辑过程。它产生一个目标文件（My code.o），该目标文件在后来的时间里可以利用不带−C选择的CC命令来连接编辑。

−g　使得编译系统产生关于符号调试程序即sdb所用的变量和语言描述的特殊信息。如果你正在通过调试程序这一步，采用这一选择。

−o　导致包含一个附加优化内容。这一选择逻辑上与−h选择互不相容。通常在程序已完成调试之后可能用−o来减小目标文件的体积，加快执行的速度。

−P　引起编译系统产生代码，产生的代码与Prof(1) 命令相连来产生程序所用时间的运行分布图，在确定那一个子程序为改进了的译码的选择中很有用。

−o　Outfile通知CC告诉连接编辑程序对执行文件用指定的名称而不用缺省值a.out。

其它可以同CC一起用的选择，查阅《程序员参考手册》。

如果你输入CC命令用一个以.S结尾的文件名。编译系统把它当作汇编语言源码并绕过汇编之前的所有步骤。

2.3.9.2 编译FORTRAN程序

命令f77调用FORTRAN编译系统。除了源码文件必须有一个f后缀外，这一命令的操作与CC命令的相似。f77命令编译源码并调入连接编辑程序来产生一个其名称为a.out的执行文件。

下面这一行命令选择的意义与它的在CC命令中的相同。

−C、−D、−o、−g和−0 outfile。

2.3.9.3 输入和运行BASIC程序

可以通过两种方式调用BASIC程序

1. 用命令

```
basic  bscpsm,b
```

这里bscpsm.b是保持着你的BASIC语句的文件名。它告诉UNIX系统输入并运行这个程序。如果程序中包含有一个运行另外一个程序的run语句，你会从一个程序链接到另一个程序上。通过common语句，第一个程序中定义的变量可以保持给第二个程序。

2. 利用设立一个shell正本。

2.3.9.3 编译程序诊断信息

C编译程序对不能编译的语句产生错误信息。这些信息一般相当容易理解，但与大多数语言的编译程序一样，它们有时指示的几个语句不是实际错误的所在，例如，如果你无意中在一个if语句的结尾多加了一个；其后面的else将被标为语法错误。在if后面跟着由几个语句组成的一个语句组的情况下，由else而引起的语法错误的行号数可以作为你查找前面错误的开始。不对称的大括号｛ ｝是另一个常产生的语法错误。

2.3.9.4 连接编辑

ld命令直接调用连接编辑程序。然而，典型的用户很少直接调用ld，较普遍地是用语言编辑控制命令（如CC）调用ld。连接编辑程序把几个目标文件组织成一个，完成重新放置，消除外部符号，合并启动子程序，支持sdb利用的符号表信息。当然，你可能从单个的目标文件开始而不是从几个目标文件。合成执行模块被留在以a.out命名的文件里。

在ld命令行命名的任何非目标文件（典型地0结尾的名称）都被认定为档案库或连接编辑程序的指令文件。ld命令有16种选择。我们将要介绍它们中的4种。

如果你用单一命令作两项工作，可通过CC命令行指定那些选择把它们反馈到连接编辑程序，这是通常的情况。

－0　outfile提供一个名称来代替输出文件的名称a.out。

－lx　指示连接编辑程序寻找库elbx.a，这里x多达几个字符。对于C程序，如采用CC命令，则自动寻找libc.a。－lx选择是用来带入通常不在寻找通道上的那些库如libm a－数学库。在一个命令行中－lx选择可以不止一次地出现，而x的值不相同。当遇到库的名字时便查寻这个库，因此这一选择在一命令行中的位置是很重要的。最安全的位置是把它放在一个命令行的结尾。－lx选择是与－L选择有关的。

－Ldir　在看缺省库目录前，改变libx.a查寻序列，在指定的目录中查寻，通常是lib或lusr/lib，如果有库的不同形式而你要把连接编辑程序指定到一个正确的上，这是非常有用的，它运行的前提是假定一旦已找到一个库就不必要继续再查找那个库了。因为－L转向寻找由－lx选择指定的库，所以在命令行中它必须在这样的一些选择之前。

－Usymna输入Symname（符号名）作为一个符号表中没有定义的符号。当你从一个档案库中全部输入时，它是很有用的，因为符号表初始是空的，并且需要一个不区分的引用来迫使第一个程序的加载。

当通过CC命令调用连接编辑程序时，启动程序（语言程序典型地是/lib/Crto.0）与你的程序相连接，这一程序在主程序执行之后调入exit(2)。

连接编辑程序接受一个含有连接编辑程序驱动程序的文件，连接编辑程序命令语言的详细内容见第12章。

2.4 程序设计语言与UNIX系统的接口

对于各种各样的服务，当程序在计算机中运行时依靠的是操作系统。某些服务，如把程序送入主存储器并开始执行，程序是全透明的。事实上，当连接编辑程序把一个目标模块标为可执行的目标模块时，它就先安排了这些服务。作为一个程序员，不用考虑这些事情。