强化C的方法

严晓东　师钧　冯志春

H

北京中国科学院希望高级电脑公司

一九八八年十一月

172

订购：北京中国科学院希望高级电脑公司
电话：2562329
乘车：乘320、332路汽车至海淀黄庄站下车

自　序

本书的写作出于三个目的。首先是为了表明如何加速C编码、如何改进C 编译程序，并揭示C程序的运行缓慢和局限的问题所在。我们希望通过具体而又非同寻常的例子来说明各种复杂的方法，进而分析并增强硬件性能。第二个目的是为为将所有与MS-DOS及C相关的信息同8086汇编语言结合起来。通过多年的教学实践，我们发现了许多与ROM BIOS列表、Intel外部芯片专用页以及显示适配器图表等有关的秘诀。不过，要想取出这些信息并转换成实用程序却相当困难。第三个目的是向所有对增强C语言功能感兴趣的人们奉献出这本资料。我们希望本书能成功地向IBM PC编程者展示：如何从使用C 语言进行基础编程到C语言高级编程！

序

作为一个IBM PC软件应用开发人员，我每周要花30个小时为用户编写程序。我经常在不同的机器上运行某一程序或寻求加速其运行的方法。但有些事情往往是C语言本身力所不及的，这时，我就把注意力转向汇编语言和MS—DOS内部环境。

我曾寻找过有关调试、改进并增强C代码秘诀一类的专著，拜读了 Harry. Chesley 和 Waite小组所著的《强化C的方法》一书的手稿之后，我的目的达到了。

作者用“强化一词来概括”的全部含义。书中采用以Microsoft C书写的示例程序和MASM的片段，向读者演示如何从各种角度对某一程序进行强化。Chesley 不仅揭示了在C代码中加入汇编算法的技巧，而且还专门阐明了这种例行程序的使用时机。作者说明了程序的哪些部位不便加入汇编代码、什么时候可能出错而影响程序执行速度。在了解如何使用汇编语言进行优化之后，读者还进一步学习直接利用ROM BIOS强化的示例。

这些示例程序说明了如何发掘PC 屏幕、键盘、串行端口的功能。作者详细地描述了对EGA的存取方法、分解程序以及用于串行终端的程序。

书中的最后一部分论述了直接存取硬件的方法。尽管人们常常认为这是软件可移植性的主要问题，但对硬件直接存取确实是一种必要的手段。例如，有关通过ROM BIOS向EGA作比特映射的窗口程序与对芯片的直接存取程序相比，其性能就要低劣得多。作者书中有关如何先深入了解硬件为从代码中取出所有性能的方法，给我留下了极深的印象。书中涉及了所有主要的芯片和相应的概念：直接屏幕存取、高速动画、中断驱动的串行I/O、直接声音存取。

最后一章给出了IP直方图程序。包括计算出代码的哪个部分所花的执行时间最长。有了IP直方图，就可以确定代码的低速部分所处的位置，进而可以用前几章中所描述的各种方法对这一部分程序进行优化。

该书的一个不可多得的部分就是C原语集。Harry给出的这一部分是为了屏幕作图之用途。这些函数的编码极为成功、使用起来也十分方便，以至于使我作出放弃原有的商用图形库的决定：从现在起，我将在程序中使用这些函数。

TP312/0P

译 者 话

目前，C语言已广为流行，我国拥有众多的C语言使用者。由于应用的不断深入，新的需求陆续出现。例如在用C语言进行图形设计时，C语言的速度显得很慢；要访问一些硬件设备时，我们会发现C语言没有提供直接访问的功能。如何更为有效地使用C语言已是一个急于解决的问题了。

本书正是解决上述问题的一本教科书，书中详细的讨论了如何优化C代码，如何将C语言同汇编结合起来使用，如何调用ROM BIOS，如何直接访问硬件等等，书中列举了大量的例子并通过对例子的研究，读者便可掌握许多实用的程序设计技术。书中还有一定量的练习题，这些练习题为读者提供了典型的现实问题，解决了这些问题，对于今后使用C语言有着极大帮助。

由于时间仓促，加之译者水平有限，一定会有不少错误，望读者指正。

译 者

一九八八年七月

此为试读，需要完整PDF请访问：www.ertongbook.com

目　录

第一章 引 言

强化是改进和扩充程序功能的同义语，本书要告诉读者如何增强C语言程序的功能，以及如何通过优化处理机和磁盘I/O的操作来减少程序的执行时间。读者将学到在C语言程序中，如何调用汇编语言子程序以及如何编写这些汇编语言子程序。然后，本书介绍如何直接调用ROM BIOS，利用它程序就可以访问以前不能访问的设备。还要介绍如何通过减少MS-DOS的开销来加快程序的执行。最后，将说明如何直接访问和控制硬件设备。直接访问硬件，象直接访问ROM BIOS一样，它使程序员可以做更多的事，并且效率更高。在此过程中，读者将遇到大量有用的，来自实际的程序，我们将利用它们来介绍所要讨论的技术。这些程序本身是非常有价值的，它们能够形成一个基础，在此基础上能开发出更复杂的应用程序。而且，为了提高兴趣，也可以同时开发一些既有趣又体现主要原理的游戏程序。

学习程序设计最好的方法是先学习其中的原理，学习示范代码，编译和运行代码，最后将其功能扩充。因此，书中的程序例子都是完整的。它们易于装入机器中运行，并且它们的功能都是独立的。此外，每章后面都有以某种方法扩展程序功能的练习。书有如下完整的，可供运行的程序：

ShowFile	显一个正文文件，允许用户用光标来控制以显示文件的不同部分
RAMSort	对RAM中正文文件排序（与MS—DOS SORT相似）
Encript	对文件加密和解密
TicTac	一个完整的tic-tac-toe程序游戏
Term	用串行端口仿真一个简单的终端
Fract	显示分数纸图案程序
Pong	电子游戏程序Ping-Pong
Noise	计算机音响程序
IP Histogram	测量程序执行时间

1.1 读本书应具备的条件

本书主要是对已经能在IBMPC上熟练运用C语言进行程设计的人员编写的。它用来帮助编程人员在C语言程序中加入汇编语言子程序；直接使用ROM BIOS和机器硬件；尽最大可能缩短程序的执行时间。

优化C程序的技术对新的C语言程序编写人员来说也是一样有用的，虽然这一技术不是针对初学者的，但是大量的例子能帮助新手理解C语言，学会如何用它编程。如果读者已熟悉IBM PC的汇编，那就更好了。

本书中的每个程序都可在IBM PC，PC/XT和PC/AT上运行，所有这些程序都在这些机器上运行过。大部分程序也可在IBM PCjr和大量的PC兼容机上运行。ROM BIOS部分

可能不能用到与IBM PC ROM BIOS不兼容的PC“兼容机”上。然而，所有这些程序都可在与IBM PC真正兼容的任何机器上运行。这里给出的程序与操作系统版本无关，因为它大部分绕过了操作系统。本书中，MS-DOS一词用来指操作系统，但所有程序在PC-DOS同样正确。

全书使用了Microsoft C（版本3.0或4.0）和Microsoft宏汇编MASM。所有这些理论和大部分代码能用在其它编译程序／汇编程序上。每次当介绍与特殊编译程序或汇编程序有关的事项时，我们都将着重指出其特殊性。因此使用其它编译程序或汇编程序的读者也会发现这本书很有用，但当用特殊信息或编译书例子时，应当小心。

1.2 本书是如何组织的

期一、二章介绍和解释与优化有关的概念。本书主要内容划分成三个部分：

第一部分　优化执行速度

第二部分　直接访问ROM BIOS

第三部分　直接访问硬设备

每部分开始有一导言性质的材料用来解释所要描述的优化技术，然后讲解例子，每个例子都是独立的应用程序。读者可以条用两种方来使用本书：(1) 按书中的顺序学习，先读理论，然后学习应用理论的例子。(2) 直接读程序例子，在有问题时再回来读理论。

第一部分从C如何调用汇编子程序开始讲（第三章），它描述了C语言和汇编的接口，介绍了如何编写能被C程序调用的汇编子程序。第四章利用方法优化基于RAM的正文文件排序程序。其中，对选择的端口用汇编语言进行了重新编码。在第五章中也类似地对Tic—tac-toe游戏程序做了同样的优化。第一部分以第六章结尾，这一章介绍了如何优化一个程序并说明了优化磁盘I/O比优化处理机速度更有意义。

第二部分以如何调用ROM BIOS开始（第七章），描述了ROM BIOS，介绍了如何编写能被C程序调用的ROM BIOS汇编语言接口子程序、第八章利用这种方法开发了一个有效的正文文件显示程序。第九章也利用这种方法介绍了彩色／图形适配器改变显示背景色的能力。第十章利用ROM BIOS与图形显示器（释色／图形适配器或增强的图形适配器）的接口来显示fractals。第十一章使用BIOS串行接口提供的工具，仿真终端。

第三部分讲述如何直接访问硬设备（第十二章），解释如何通过汇编语言直接访问硬件。这章包括对存贮器，输入／输出端口和中断的直接访问。第十三章使用这种方法来把要显示的文本直接放入显示缓冲区去，这个程序是对第二部分开发程序的改进。第十四章的图形显示使用了同样的技术。第十五章扩充了第二部分用硬件中断开发的Tew程序。第十六章利用输入／输出端口，直接控制音响硬件，从而生成音响效果。然后，第十七章开发了一个程序，使用时钟中断，记录某个程序在什么地方花的时间有多长。这些信息与第一部分开发出的技术相结合，可用来优化处理机执行程序的速度。

附录A是一个为那些已懂得C语言的程序员编写的关于汇编语言的简短介绍。过去没有汇编语言经验的程序员应该读读这部分，至少要浏览一下，而且应在读本书的其余部分之前读它。附录B解释了如何将一个程序编译或汇编，然后和其它子程序连接起来以产生一个可运行的应用程序。附录C介绍了经常使用的ROM BIOS调用及调用中应传递的参数和返回

的参数。附录D给出了当按不同键时从键盘上返回来的ASCⅡ和扫描码。

利用本书描述的强化技术，读者能在IBM PC/XT/AT或兼容机上获得更强的功能。强化能更全面地访问硬件和软件，增强程序的功能。使用优化技术，可以全面改进程序的性能。

第二章　强化：概念

什么是“强化”？所谓强化就是使程序能够完成它以前不能够完成的任务或使之更快地完成。例如直接访问PC的部分基础硬件；访问硬盘；更新屏幕等。在这一章中我们要讨论各种强化方法。本书的目的是在读完各章后能学到增强和优化自己的程序的方法。我们还要讨论强化一个程序的利与弊，即应该不应该进行强化及什么时候进行强化。我们还要解释如何编写易于强化的程序。最后，我们要谈谈如何将C语言和汇编语言结合起来，通过直接改进程序执行速度或者允许程序访问C语言难于访问的软硬件来强化程序。

2.1 IBM PC概述

大部分强化技术都利用了在IBM PC上可获得但又不易访问的资源。其他强化技术包含绕过PC操作系统的某些部分来编写子程序，这些子程序与那些利用硬件，操作系统或编译程序来编写的程序相比虽通用性差些但效率更高。了解IBM PC硬件和软件的总体知识对理解强化来说是必须的。

IBM PC由一个系统板（主机板）和几个插入扩展槽中的扩展板组成。如图2—1所示。系统板包括处理器和外围芯片及存贮器，键盘和扬声的接口，扩展槽等。在扩展板上，可安放更多的存贮器，显示器接口，并行和串行外部设备的接口及其他多种非标准的硬件。IBM PC有两类存贮器：RAM（随机存取存贮器）和ROM（只读存贮器）。RAM的内容可通过执行程序来改变，而ROM是用户不能改变的存贮器。

PC中运行的软件有两种来源：一部分存贮在ROM中，另一部分从磁盘装入RAM。ROM中软件称为基本输入／输出系统，即BIOS，或ROM BIOS。从磁盘中装入的软件或者是操作系统（MS—DOS），或者是应用程序。用C语言写的应用程序要引用MS-DOS，ROM BIOS，及编译程序运行时提供的库。在强化一个C语言程序时，应记住：所有这些部分合在一起才产生一个IBM PC上可运行的应用程序。图2-2显示了这些部分的层次。

应用程序要对执行的特定任务负责：它们知道任务是什么，怎么实现它，这是最高的层次。运行库提供了C和系统其他部分的接口。在某些情况下，这些接口是与机器无关的，因此程序可以很容易地从一个系统移植到另一个系统上。MS-DOS维护文件系统的结构和保持它的一致性，它提供了一个公用接口，通过它，应用程序能访问机器的基础硬件。ROM BIOS提供了一个与硬件通信的低级接口，对于较高一层次来说，它将硬件的详细操作情况隐蔽起来，ROM BIOS从字面上来看是PC的基本输入／输出系统。所有的工作最终要通过它驱动硬件动作以产生预期结果。

这些部分在本书后面的章节中会详细讨论的，现在，要记住：应用程序是在软件和硬件的顶层。当调用一个函数时，比如从磁盘上读入数据或在显示屏幕上显示字符，机器会执行一系列复杂的动作，以完成这些功能。有时，人们很难明白这一点。因为即使包含上千个动作，PC执行起来也只不过是一瞬间的事情。但当程序连续执行这样的动作系环上千万次时

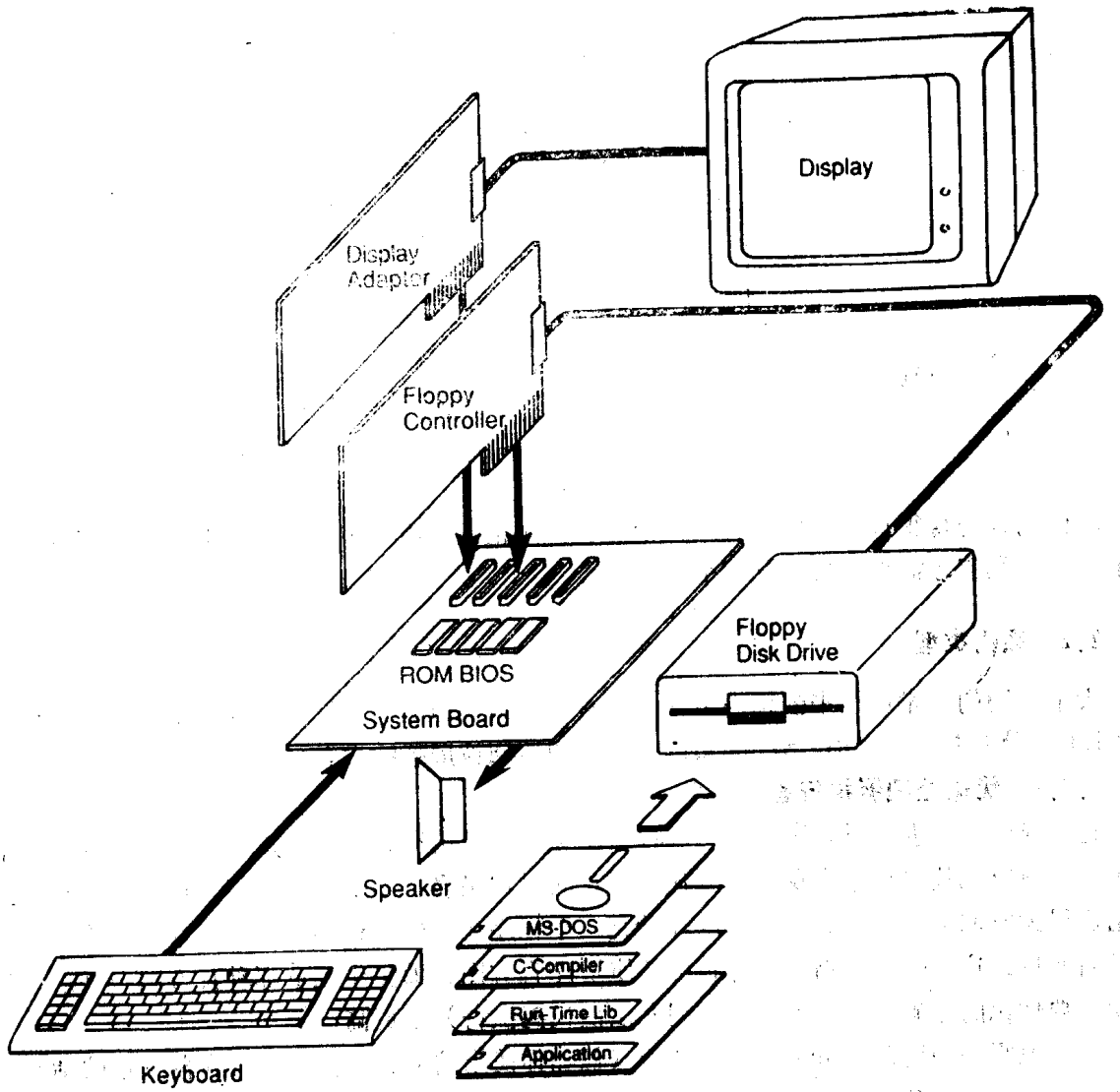

Figure 2-1. Components of the IBM PC

图2—1 IBM PC计算机构成

此为试读,需要完整PDF请访问: www.ertongbook.com

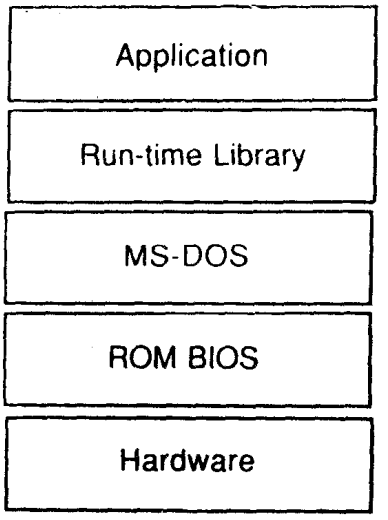

Figure 2-2. Components of a Running Application

图? -2 一个正在运行的应用程序的构成

执行时间以致变得很长，人们才感到这个过程。而正是使用程序的人们需要等待程序及早完结的时候，我们才需要考虑强化一个程序。

2.2 强化类型

大量不同的技术可用来强化一个程序。它们可归纳为五类：优化处理器执行速度，优化磁盘I/O，绕过MS—DOS，绕过ROM BIOS和访问被屏蔽的设备。

2.2.1 优化处理器执行速度

这类强化要改进代码效率以使其运行速度更快，但从提高处理器速度的角度来优化一个程序，必须首先找出程序在哪一部分代码上花的时间最多。不知道这一点，可能去优化那些只花费很少时间运行的代码。典型的经验是一个程序在10%的代码上花去了90%的时间，关键是如何找到这10%的代码。一种技术是使用IP直方图，即第十七章那种直方图一样。一旦找到了要优化的正确区域，可采取下列方法使代码执行效率更高：

1. 说明一些局部变量为寄存器变量（假如你使用的 C编译器支持寄存器变量的话），这样没有改变程序功能而加快了程序执行，因此，这是最安全的优化。

2. 重新编写高效的C代码。经常花很少一点时间就能重写一个更为有效的子程序。

3. 找一个更有效的算法。一个高效的算法可以提高执行的效率。

4. 用汇语言重写子程序。通常用汇编写的代码比C编译器根据 C 源代码产生的代码更紧凑，更有效（尽管C编译器也能产生紧凑的代码）。

加快代码执行速度的技术主要包含在本书的第一部分。第三章我们将解释如何用汇编语言子程序替换C语言子程序。在第四，第五章中，我们给出用汇编语言重写的子程序来优化程序的例子。第十七章，我们将讨论如何找出模块，把这个模块加到程序中，就会产生一个直方图，用它记录了程序实际上花费最多时间的地方。

2.2.2 优化磁盘I/O

使程序运行快的一种方法是优化磁盘I/O。需要从磁盘上读出写入数据的应用程序，磁盘I/O往往花费了应用程序的大部分执行时间。通过仔细改进访问磁盘的方法，适当选择缓冲区的大小或改变存取磁盘上数据的顺序，可能比优化处理器速度更有效。仅仅改变延冲区大小有时就比花一周时间把C语言子程序编成汇编代码所产生的效果更好。这种优化技术将在第六章中讲述，在这一章中，我们要开发和优化一个多个磁盘操作的程序。

2.2.3 绕过MS-DOS

第三种改进应用程序执行速度的方法是绕过MS-DOS。MS-DOS给在PC上的硬件提供了一个通用的接口，但这个接口对完成一个特定的功能来说并不总是最有效的。更通用的子程序，象MS-DOS所提供的那些子程序，比起某些不很通用的但更专用的子程序来说，在某一特定的应用中，效率是不高的。在每个PC中，都有一个直接操作硬件的子程序的集合，这就是所谓ROM BIOS。通过直接调用ROM BIOS，绕过MS-DOS，这样可以省掉MSDOS通用子程序的开销。

本书第二部分讨论ROM BIOS。第七章解释如何调用它，第二部分其余章节给出了一些使用ROM BIOS的例子。特别是第八章指出了一个应用程序应如何通过绕过MS-DOS，直接访问BIOS从而加快它的运行速度。

2.2.4 绕过ROM BIOS

提高一个应用程序速度的第四种方法是绕过MS-DOS和ROM BIOS，直接面对PC的硬件进行程序设计。这种方法免除了引入机器或MS-DOS提供的通用子程序，或运行库所带来的开销。事实上，假如程序员很熟练，能写出很高效的汇编语言程序，那么，同时使用两种优化技术——用汇编语言编写程序和直接访问硬件，便会得到在IBM PC上运行的最高效程序。

第三部分讨论硬件的直接访问。第十二章给出了其中的理论，其余章节给出了直接使用硬件的例子，特别是第十三章的例子，它显示了这些技术应用，如何使程序运行得更快。

2.2.5 访问被屏蔽的设备

这种强化技术与功能，不是直接与速度相关连它允许程序使用ROM BIOS提供的手段和询问编译运行库或MS-DOS不能访问的机器硬件。没有一个系统允许应用程序访问所有的基础硬件，相反，它试图选择那些应用程序最需要的，给它们提供一个相关联的有序接口，而把硬件详情对使用它的应用程序屏蔽起来。通常，这正是所期望的，但也有些应用程序恰好需要访由操作系统没有提供访问手段的硬件。在IBM PC上，应用程序可以调用BIOS，它比MS-DOS提供了对硬件更复杂的访问。如果BIOS不提供这种功能，便需要直接访问硬件。

这方面的强化技术包括在第二，第三部分中。第二部分的第九章，十，十一章和第三部分的第十四章，第十五、十六、十七章提供了在运行库和MS-DOS中没有提供的访问手段的例子。

这五种类型的强化也可结合起来以产生更强有力，更高效的程序。特别是，当把这些能提供以前没能获得的设备的访问能力的技术结合起来时，优化速度的作用便更加明显。当然，C语言编程人员也就能拥有一个功能更强有力的语言了。当把本书中给出的技术结合在一起时，便可以用C语言编写出在PC上运行得既快速、高效又强有力的应用程序

2.3 强化的利与弊

决定花时间和精力来强化一个程序并不是十分显然的事。强化能使程序运行更快，完成更多的任务，但也是有代价的。必须仔细权衡利弊。有时别无选择，必须进行强化。例如，当程序为完成任务必须进行直接访问硬件时就是这种情况。然而，这仍有潜在的付作用。从根本上说，是否进行强化取决于这个程序如何使用，在何处使用，使用频率如何，以及它要求什么特别的功能。在这一节中，我们要讨论强化一个应用程序的好处和不利之处。

强化的第一个弊病在于它限制了程序运行的机器环境。以前描述的大多数技术绕过IBM PC的某些部分：操作系统，ROM BIOS或C编译器从改进程序效率。但是，提供给PC的软件和编译程序的一个作用就是要使应用程序可以不考虑它所运行的特定机器的硬件情况，这被称为对高层次程序屏蔽低层次的细节。在使用通用的C编译器的情况下，对应用程序，可以屏蔽了处理器类型，许多C程序可以在从IBM PC微处理机到DEC VAX小型机甚至Cray巨型计算机上都可以运行。如果一个程序是使用汇编语言，它只能在使用8086/8088/80286系列的微处理机上运行。类似地，如果一个应用程序使用ROM BIOS，它只能在与IBM PC ROM BIOS完全兼容的机器上运行。下面的表描述了因为强化要绕过IBM PC部分软件而产生的影响。

正如表中所指出的那样，有几类不同的兼容性。与IBM PC兼容并不保证一个机器能运行你的程序。我们可以把IBM PC兼容机分成三组：第一组包括硬件10%兼容的机器，它们可运行在IBM PC上运行的任何程序。第二组包括BIOS与IBM PC100%兼容的机器，但它们的低级硬件可能有细微差别。这种机器能运行通过MS-DOS或BIOS访问硬件的程序。第

部件名	应用程序与……无关	通过……	程序只能运行在……机器上
C编译器	处理器	用汇编编写	8086/8088/80286
C文件I/O库	局部文件系统	直接调用MS-DOS	MS-DOS机器
MS-DOS	ROM BIOS	直接调用ROMBIOS	IBM PC ROM BIO兼容机
ROM BIOS	硬件	访问硬件	IBM PC硬件兼容机器

三组包括能运行DOS，但有不同的ROM BIOS和基本硬件的机器。它们仅能运行那些只通过MS-DOS访问硬件的程序，由于上述的各种不同兼容，所以某些程序不能在所有IBM PC兼容机上运行。

从兼容的观点来看，编写仅调用MS-DOS的程序最好，这样程序可以在每种有MS-DOS的机器上运行。但实际上，有各种各样的理由要求绕过MS-DOS甚至绕过BIOS。绕过与不绕过有着很大的差别：就象一个快速响应的编辑路和一个慢到甚至不能尽快显示输入字符的编辑器的差别；一个100%可靠的通信程序和偶尔还要丢失一些字符的通信程序的差别；一个具有音响效果的游戏程序和一个蹩脚的游戏程序的差别。另一方面，绕过也意味着一个程序只能在与IBM100%兼容的机器上运行，如果你购买的是不兼容的机器，则这种技术不是很有用。为IBM PC编写应用程序的人员必须作出一系列的决定，权衡程序特点和它所能

运行的机器范围。对于所有的IBM PC的程序设计人员，都必须作出同样的决定。

强化有第二个弊端：它经常要把C代码重写为汇编代码。因为汇编语言很难理解和改动，这会增加将来维护程序和添加新功能的费用。用汇编语言重写是否实际增大一个程序的费用取决于程序本身，取决于程序中多大部分要用汇编重写及C和汇编代码之间的划分是否得当。

我们已经讨论了强化的某些弊病和在什么情况下，对一个特定的应用程序应使用什么样的强化技术。另一方面，通常还有两个原因需要强化一个程序：程序运行不够快或者其功能不能满足要求。这两个原因都是主观的——如，多快才算是足够快。如果是一千人而不是十几个人使用程序，完全值得花额外时间彻底强化这个程序。如果是一个程序每天都使用，而不是一月一次，那么强化也是值得的。只有你和使用程序的用户才能决定强化是否恰当。

某些类型的强化是无害的或者基本上是无害的。优化磁盘I/O和说明变量为寄存器变量并不改变程序的功能，也不限制程序能运行的机器。但如果程序只是在IBM PC上运行，也就是说，它不会被装到VAX或Macintosh上，用汇编语言有选择地重写某些部分也是无害的。直接访问ROM BIOS限制了程序只能在ROM BIOS兼容机上运行，直接访问硬件限制了程序只能在硬件兼容的机器上运行。这里也一样，如果讨论的程序只在PC而不会在BIOS不兼容的机器上运行，那么这些强化技术也不会引起问题。

有时别无选择，必须进行强化，而不管它有什么不利因素。如果应用程序需要访问硬件，但是MS-DOS不能获得，甚至通过ROM BIOS也不能获得，此时就必须采用强化技术。这已不再是用移植性换取效率的事，而是可移植性和根本不能工作之间的差别——如果程序根本就不能工作，就谈不上可移植性问题。这种类型的强化技术可以增加新功能或者增加现存程序以前不可能有的功能。

这里的关键是：首先要考虑好了强化的结果，不然就不要进行强化。否则只能得到一个没有多大用场的程序——一个只能在你自己开发所用的机器上运行．但不能在打算使用你的程序的用户的机器上运行的程序。如果你方方面面都考虑到了，就可以开发出一个吸引人的．运行得又快的好程序。记住，程序的优化的结果，是要程序使用更加方便，而且用户也无须再做一些额外的工作。

2.4 模块化程序设计

重新写一个也许将来要强化的程序的第一条规则是：程序应模块化编写。当一个程序采用模块化技术写成时，其中的每一个模块都容易修改，如果一个程序不是模块化的，强化会引起很长的混乱，加大了复杂性，可能完成一个不可维护的程序。

模块化程序设计的好处也可体现在通常的程序开发和维护上，而不只是体现在强化程序这点上。用模块构成程序，是是一种好的方法。

设计一个模块化程序化程序并不是把代码分成块块以及限到每块的长度。学会如何编写模块化程序的最好方法是研究别人写的程序，然后写你自己的程序。不幸的是，没有一个普遍的、快速的规则来说明如何进行模块化程序设计，但有一些一般性的指导原则，兹讨论如下。

2.4.1 把功能与实现区别开

每个模块有两个方面：一方面是它的功能，这是给应用程序的其它部分使用的；另一方

面，是它的实现，由只能为程序员和编译器所识别的内部细节构成。外面部分告诉我们一个子程序做什么，而内部则告诉我们怎么做。图2—3用电视机作为例子说明了它们之间的差别。外面包括了控制部件和显示屏，这是和电视机使用者的接口；内部包括管子和导线，它们对控制起作用，产生要显示的图像。当设计模块时，重要地是把模块的功能和实现分开，如果模块功能不变，那么改变实现而不会影响程序的其它模块，因此我们可以重新写一个模块以提高其效率，而不用担心对程序其它部分产生不良的影响。这样就加快了子程序的执行速度，而程序作为一个整体并不改变功能。

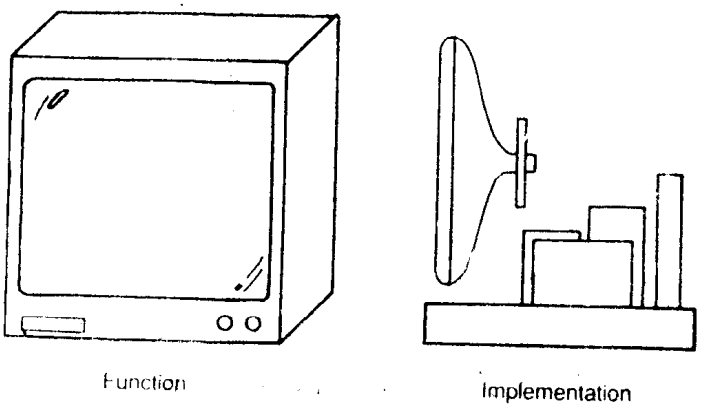

Figure 2-3. Function vs Implementation

图2—3　功能与实现

2.4.2　设计适当大小的模块

每个模块的大小最好能既能理解它的外部功能，又能理解它的内部算法。模块化的一个理由是，当程序大而复杂，从整体上不便理解时，把程序划分成理解的块。模块本身又可由几个更小的功能块即函数组成。

要编制的大多数程序都较复杂，不能一次完全理解，但可以把大的复杂的系统分成若干个小问题来解决（见图2—4）。

2.4.3　使模块间通信量极小

在程序划分成模块时，要使模块间要求的通信量极少。图2-5介绍了如何把一个模块划分成几个子模块，并使模块间联系减少。而划分不好，就有可能使设计复杂化。每个问题或算法都有这样几行，使它们的划分既容易又很自然。找出这些行是程序设计技巧的一部分。它经常与机器或应用程序的原始概念有关，例如，把显示器存取模块和磁盘存取模块分开是一种很自然的划分。采用模块化设计程序也是出于同样的原因。硬件也是模块化设计的。显示器和磁盘是IBMPC硬件模块中的两个模块。设计者知道这比把PC系统设计成单一的复杂部件要好。类似地，大多数应用程序从使用它们的外界继承了一些特性。外部环境通常也按一定的方法分成一些模块，这些模块即可作为我们划分程序模块的一种根据。

到目前为止，我们讨论的划分都是水平方向的，即把程序分成几个部分，而几个部分处于同一层——例如，从文件中读入一数据块，然后把它显示在屏幕上。垂直划分也是可能的：这样可以把混杂在一起的东西分成不同的层次。例如，先从物理一级上控制磁盘驱动器，然

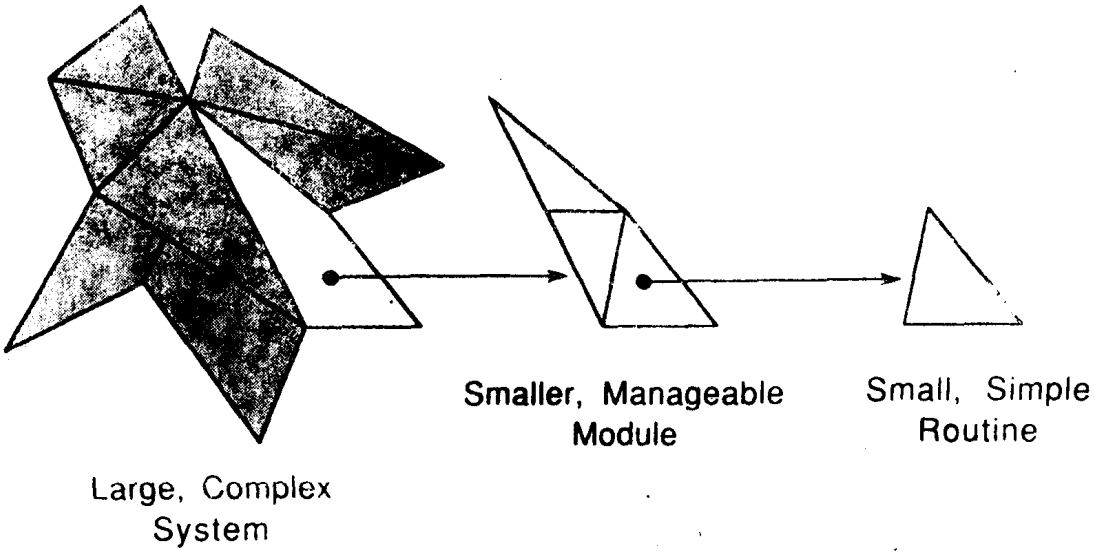

Figure 2-4. Breaking Down Complexity

图2—4　通过划分降低问题的复杂性

后当成一个文件系统来处理它，这些都是独立的模块。在IBM PC上，ROM BIOS模块控制物理的磁盘驱动构，而MS-DOS模块把磁盘看作一个文件系统（见图2-6）。高层模块通过高层应用程序增加使用功能，同时屏蔽低级操作的细节。这使高层不用关心低层细节，同时也低层模块改动而不影响高层软件。

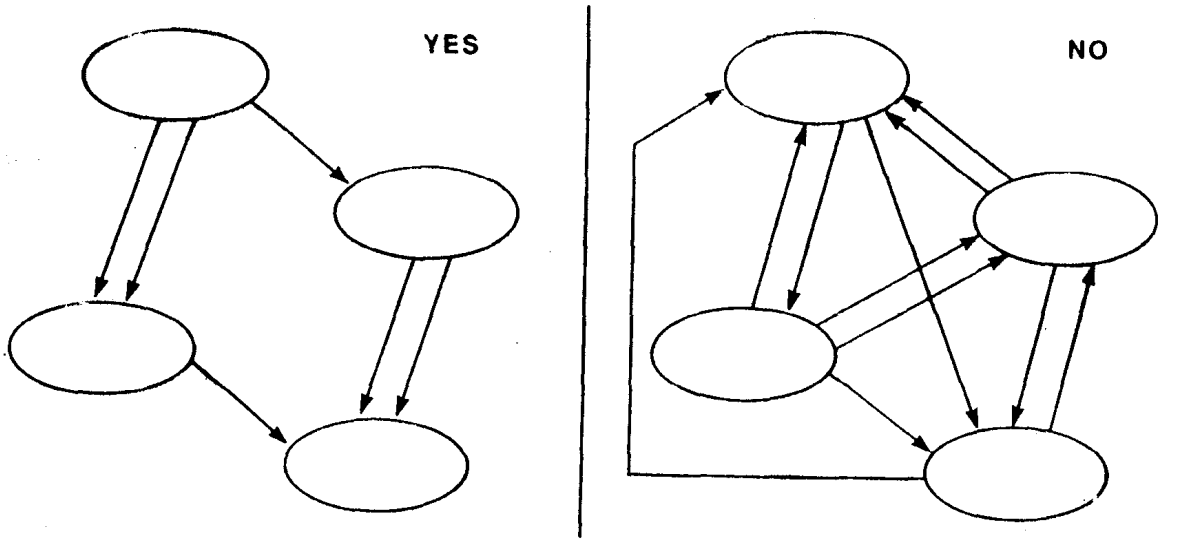

Figure 2-5. Minimize Interconnections

图2-5　模块间关联极小化

此为试读，需要完整PDF请访问：www.ertongbook.com