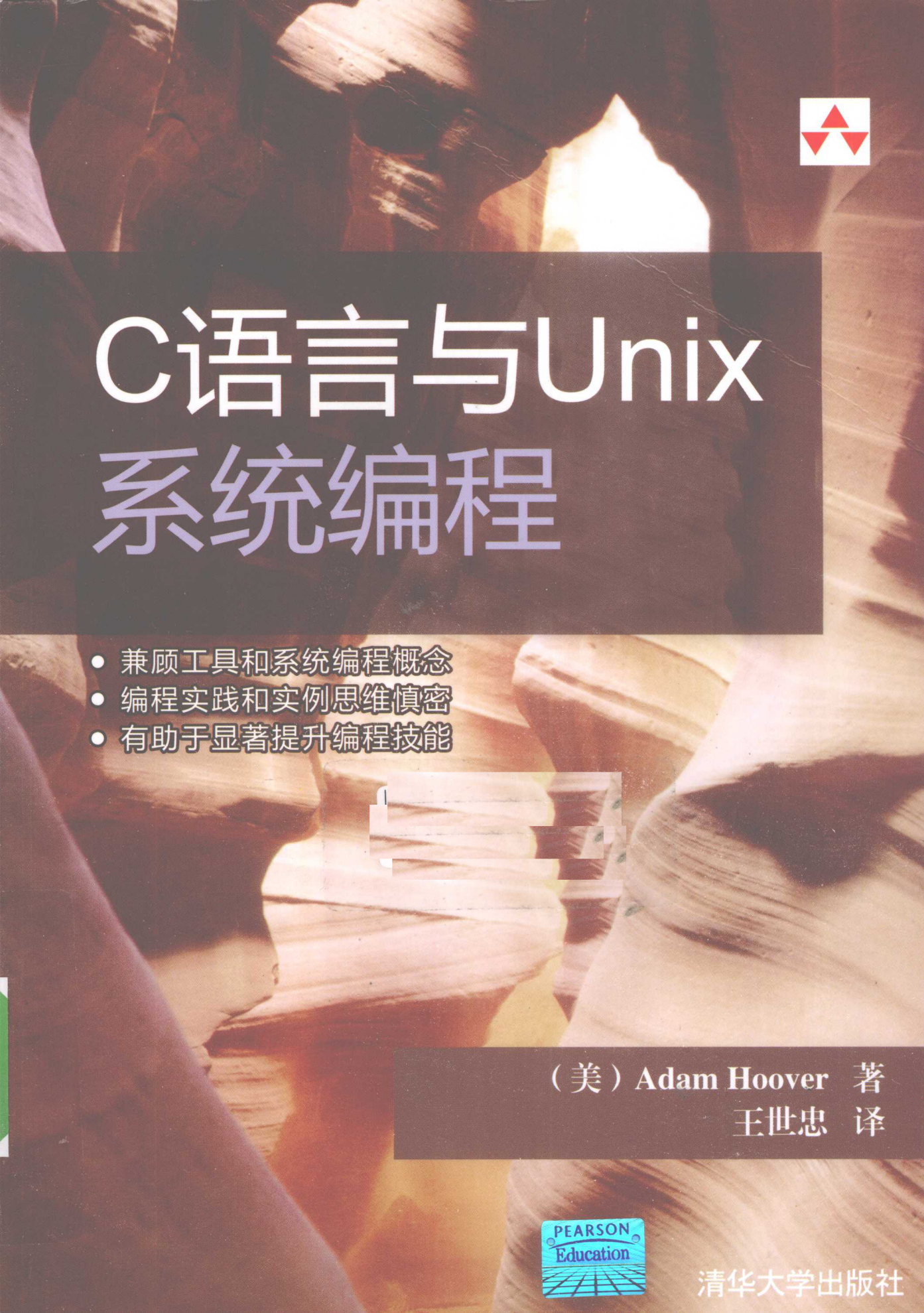
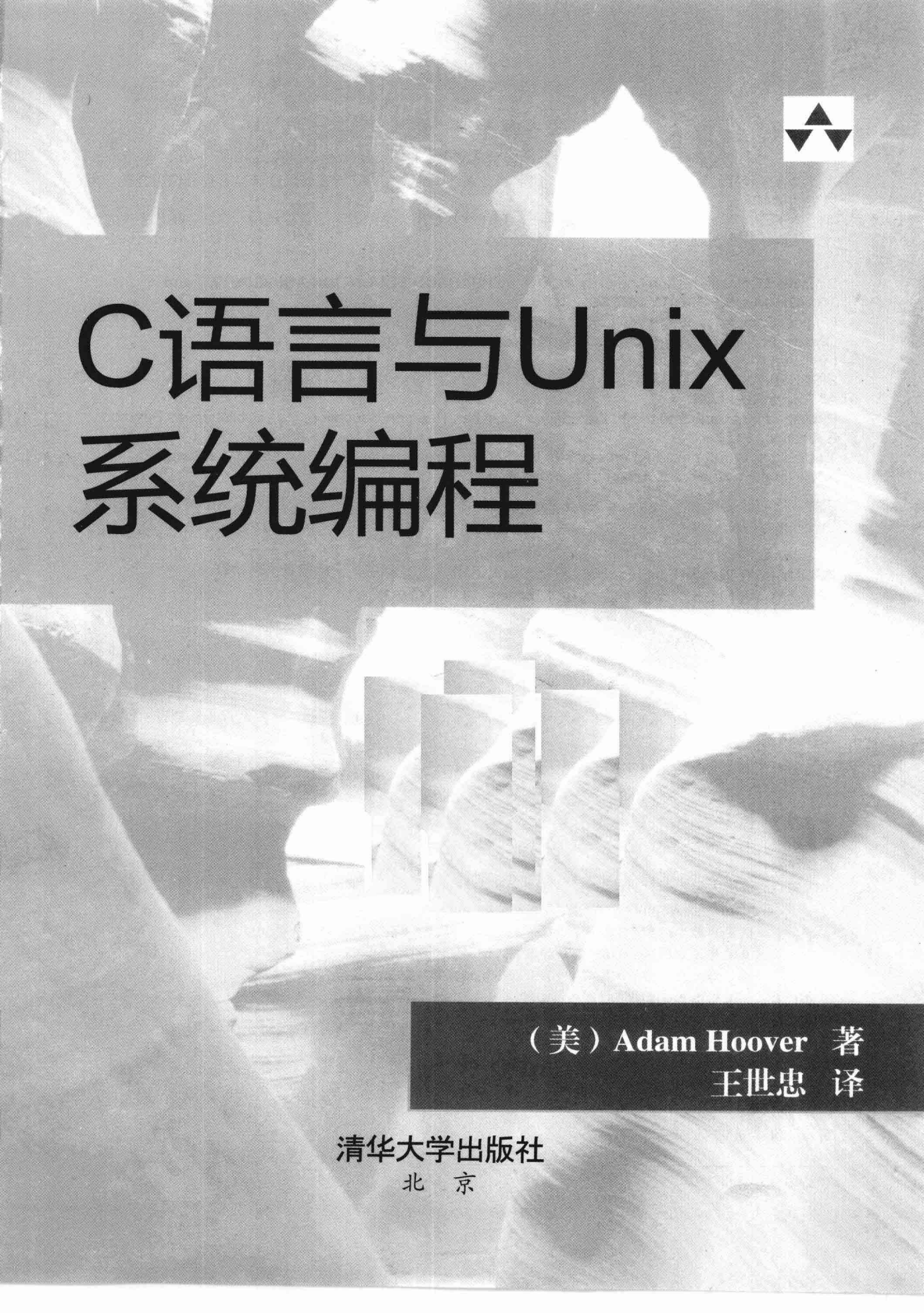


C语言与Unix系统编程

- 兼顾工具和系统编程概念
- 编程实践和实例思维慎密
- 有助于显著提升编程技能

（美）Adam Hoover 著
王世忠 译

PEARSON Education
清华大学出版社



C语言与Unix系统编程

（美）Adam Hoover 著
王世忠 译

清华大学出版社
北 京

内 容 简 介

《C 语言与 Unix 系统编程》针对系统编程所涉及的问题，深入探究底层数据类型，以内存管理为重点，结合编程实践和实例，繁简得当地介绍了系统编程工具和资源，旨在帮助读者显著提升编程技能，为以后的学习和工作奠定良好的基础。

本书作为系统编程的入门教材，适合一学期的教学使用，是读者学习数据结构、算法、操作系统和编译器等高级编程主题的理想基础。

图书在版编目(CIP)数据
C 语言与 Unix 系统编程/(美)胡佛(Hoover, A.)著；王世忠译. --北京：清华大学出版社，2010.7
书名原文：System Programming with C and Unix
ISBN 978-7-302-23049-6
Ⅰ. ①C… Ⅱ. ①胡… ②王… Ⅲ. ①C 语言—程序设计 ②UNIX 操作系统—程序设计 Ⅳ. ①TP312 ②TP316.81

中国版本图书馆 CIP 数据核字(2010)第 113392 号

责任编辑：汤斌浩
责任校对：王 晖
责任印制：杨 艳
出版发行：清华大学出版社 **地 址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn **邮 编**：100084
社 总 机：010-62770175 **邮 购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京市世界知识印刷厂
装 订 者：三河市兴旺装订有限公司
经 销：全国新华书店
开 本：185×260 **印 张**：17.75 **字 数**：420 千字
版 次：2010 年 7 月第 1 版 **印 次**：2010 年 7 月第1 次印刷
印 数：1～4000
定 价：39.00 元

产品编号：035378-01

译 者 序

翻译《C 语言与 Unix 系统编程》一书，当属机缘巧合。

2009 年秋天，从成都返回北京后，我的工作不是很忙，于是精力稍显过剩的我在好友汤斌浩的建议下，接下了翻译《C 语言与 Unix 系统编程》的任务。

此后的三个月时间里，我把自己的日程安排得满满的。10 余年前，我曾翻译过几本计算机专业的英文书籍，但那毕竟是陈年往事。此番重操旧业，对于科技图书的翻译，对于计算机技术的发展，对于计算机教材的写作风格，又有了新的认识。

科技图书的翻译，仰仗于扎实的专业背景和中文素养。有些术语，在不同的语境下，有不同的含义，如果翻译错了，必然会贻笑大方，严重的时候甚至误人子弟。因此，在翻译过程中，虽然也能像以前那样流畅地进行翻译，但对于遣词用句的斟酌，显然更加谨慎，同时也保留了自己的一种风格。

翻译的同时，也是一种学习。学习知识，很多人都喜欢通过实例入手，当年作为学生的我也不例外。《C 语言与 Unix 系统编程》是一本非常优秀的计算机编程教材，作者基于自己多年的编程经验深入浅出地介绍了 Unix 系统环境下 C 语言编程的主要技巧，并提供了很多实例，用以说明每一项基本的编程技术和容易出现错误的地方。本书的作者没有拘泥于 C 语言，也没有拘泥于传统的 Unix 环境，而是结合现代计算机编程技术的发展来阐述 Unix 环境下的 C 语言编程，非常适合学生及 Unix 环境下编程的初学者学习。在翻译的过程中，能够体会到作者的风格，就好像有良师在旁进行指导，帮助你循序渐进地掌握一些貌似艰深的知识。

在翻译本书的过程中，得到了我的家人、朋友和公司同事的太多帮助。李耘和韩冠楠协助我处理了在翻译过程中遇到的大多数语言问题，陈华亮、段玉明和方俊斌帮助我审阅了大部分章节的译稿，刘海鹏、薛红、杨晶晶、叶果等协助我借阅、查找了大量的计算机、数学等领域的英文原著，在部分专业词汇的翻译上给予了我很大的帮助。感谢我的同事张建军、谢军霞、王琦、王国悦、张利军、张婷婷、张永静、张媛媛，他们为我分担了许多工作，并对每章的译稿提出了很多宝贵的意见。

同时，感谢晁丹和金琰——我工作上的得力助手，在我翻译本书的过程中帮助我做了大量的琐碎工作，感谢他们无私的付出和努力！

最后，感谢好友 Tung 为本书所花的心血以及所有认真、细致的工作。

王世忠

前　言

本书的目标是向读者介绍系统编程工具及资源，从而使读者成为一名更优秀的程序员。考虑以下问题：什么是库，如何使用它们？什么是调试器，它们在程序开发过程中是如何起到帮助作用的？什么是脚本描述语言，它们对什么类型的编程工作有用？什么是系统调用，它们用在什么地方？为什么有人宁愿从一个外壳程序中运行程序而不使用图形用户界面？就像本书大纲所勾画的一样，在学习“系统编程”的过程中，无疑会提出这些问题。

除了系统编程这个概念外，本书还深入研究了底层的数据类型：位和字节、位操作、数组、字符串、结构和指针。有关这些内容，本书重点介绍了内存，并着重使读者了解为什么要使用以及如何使用不同的数据类型。理解内存级别的代码能够使读者清楚地了解哪怕是最困难的编程概念。对一名学生而言，会觉得这些问题比循环和条件之类的其他基本编程概念更不容易理解。本书之所以涉及底层数据类型的内容，目的在于增强读者对以前所学基础知识的理解。本书的目标是进一步提高读者的编程技能，使其能从中级水平上升到高级水平，达到深入理解并自如使用上述内容的境界。

本书的目标读者

本书的内容被设计成用于有关系统编程的一学期课程，供完成了编程入门课程的读者使用。因此，本书的另一个通俗名称可以是“第二学期编程”。在设计本书内容时，是希望作为读者学习数据结构、算法、操作系统和编译器等高级编程专题之前的铺垫。然而，并不一定要按照这种顺序来学习本书的内容，系统编程的学习加强的是学生有效完成更高级主题的能力。本书着重强调读者实际的编程技巧，这对学生后续课程的学习非常有益。对具有较强的编程技能的学生而言，他们简单浏览一下本书前几章即可，以便能将更多的时间花在系统编程的内容上。虽然本书是为一学期课程的学习而写的，但书中有足够多的内容可以让教师根据学生的知识水平灵活定制本书的学习内容。

授课教师注意事项

在本书每一章的内容中，都包含许多示例代码，用以说明当前章所讨论的概念。所有的示例均已经过课堂授课过程中反复使用的全面测试。在本书出版商的网站（http://www.pearsonhighered.com/irc），可以下载这些示例代码。为便于编码、显示和在课堂上进行讨论，我们特意编写了这些简短的示例代码。本书作者在授课过程中，一般使用一台连接到投影仪的笔记本电脑来执行大多数示例代码，并以不同的方式对它们进行修改和测试，以说明所学习的概念。在课堂上，鼓励学生带笔记本电脑来上课，执行示例代码，或讨论每一章“问题与练习”中一些比较简单的问题。这些交互式技术旨在鼓励学生的学习，并且在设计本书内容的时候，我们也倾向于使这种教学方式变得更加容易一些。

然而，这些技术并不需要使用本书的内容。这些示例代码足够短，有利于在课堂上以其他方式来显示。教师可以有足够的信心来使用它们，并知道它们确实可以像在本书内容中所演示的一样执行。

在本书的某些章中，提供了某个专题的多个示例供读者讨论。例如，脚本语言编程就是通过外壳脚本编程（shell scripting）、Perl 和 MATLAB 进行演示的。所有这三种语言并不需要进行深度的讨论，研究几个示例就可以搞清楚脚本编程语言的整个目的和使用方法。同样的手段也在介绍库（示例是 C 标准语言、curse 和 X）和系统调用（示例是进程、信号和套接字）的内容中采用。根据在课堂上着重介绍的示例的不同，教师可以在教学实践过程中灵活地介绍和讨论涉及这些专题的内容。

在本书中，Unix 指的是任意一种类 Unix 系统，包括 Linux、Mac OS X、BSD 变种、Solaris、HP-UX、AIX 等。作者在写作本书的时候，使用了 Linux 来开发示例程序，但这些示例可以在任何 Unix 或类 Unix 系统中正常运行。

补充内容

针对通过正规渠道购买本书的读者，我们提供以下补充内容。

- 丰富的授课幻灯片（PPT 和 PDF 格式），在这些幻灯片中，包含本书所有的图。
- 本书所有可以编译和执行的示例程序源代码。
- 包含每章末尾“问题与练习”中所有问题及练习的解决方案的“教师指导手册”。
- 关于软件授权的额外附录。

如果要获取这些补充内容，请访问 http://www.pearsonhighered.com/irc。登陆后，先注册等待回复。对于希望得到用于获取教师专用材料密码的教师，可以与当地 Pearson Education 公司的代表机构进行联系(或发送邮件到 coo@netease.com)，或者发邮件至 computing@aw.com。

本书与其他书的区别

本书的重点在于系统编程这个概念，而不是对它涉及的任何一个专题进行全面介绍和讨论，这些专题包括文本编辑器、外壳程序、库、脚本编程语言、系统调用和程序生成等。全面介绍这些专题会使本书变得过度庞大和笨重，并且，市面上已经有不少介绍和讨论上述专题的优秀图书。本书将所有的系统编程概念组合在一起，希望能从教材的角度为系统编程问题提供一个良好的解决方案。

致　谢

作者谨在此感谢本书写作过程中审阅本书初稿的以下各位：

- Old Dominion 大学的 Hussein Abdel-Wahab
- Lehigh 大学的 Brian D. Davision
- Delaware 大学和 Hologic 公司的 Christopher R.Fischer
- Cal Poly 大学的 Gene Fisher
- Northern Kentucky 大学的 Richard Fox
- Baylor 大学的 Cindy Fry
- Arizona 大学的 Patrick T. Homer
- Florida Atlantic 大学的 Sam Hsu
- Oberlin 学院的 Benjamin A. Kuperman
- 位于 Fredonia 的 SUNY 学院的 Natalie A. Nazarenko
- Clemson 大学的 Bill Reid
- Idaho 大学的 Bob Rinker
- 芝加哥大学的 Nicholas A. Russo
- Clemson 大学的 Melissa Smith
- Villanova 大学的 Tom Way
- Massachusetts 大学的 Tilman Wolf

他们对本书所做的努力和反馈具有极高的价值并帮助我完成了本书的写作。特别感谢本书的审阅者 Melissa Smith 和 Bill Reid，在本书的写作过程中，他们使用本书的内容进行了教学。最后，感谢作者的妻子和孩子，Adair、Austin 和 Gabrielle。书写完了，让我们一起去远足吧！

目　录

第1章 简 介

本章提示

- 什么是系统编程
- 三个工具
- 如何进行调试
- 程序开发
- C语言回顾

1.1 什么是系统编程

计算机系统是由硬件和软件组成的。一般而言，软件指的是在计算机上运行的程序。尽管硬件和软件都是可以修改和升级的，但修改和升级更多发生在软件上。事实上，计算机拥有软件的主要原因是要改变在计算机上执行的指令流。这意味着，在一个计算机系统文件的生命周期中，经常需要编写新的程序或修改、改进旧的程序。

基于这种需求，人们很自然地就需要寻找计算机系统支持程序开发的手段或方法。在过去的30年中，人们开发了一些工具和资源，用来辅助程序的开发，这些工具和资源包括标准库(也称系统库)、系统调用、调试器、外壳环境、系统程序和脚本编程语言。对这些工具的了解和使用大大加强了程序员的能力。然而，尽管了解某些专门工具的细节很重要，但更重要的是，还要了解什么时候以及如何最有效地使用这些工具。有一些不错的参考手册提供了有关某种专门语言或工具的详尽信息，但本书计划向读者介绍有关系统编程概念的更广泛内容。

我们可能会将系统编程定义为在程序开发过程中对系统工具的使用。恰当地使用这些工具有几个目的。

第一，同时也是最重要的，它节省了大量的时间和精力。使用系统库，可以节省程序员用来独立开发那些相同的函数的时间。使用调试器，可以节省用来查找并改正程序中错误的大量时间。在现在的系统程序中，都方便地提供了诸如用于文件集的文本搜索或程序执行计时这样的常用工具。

第二，系统工具提供了通常情况下极其难以实现的程序开发机会。系统调用提供了对操作系统核心功能的访问，包括内存管理、文件访问、进程管理以及进程间通信等。某些标准库提供了超越大多数程序员能力的复杂函数。例如，数学库就包括三角函数和其他需要特别方法才能够解决的实值操作(real-valued operation)。

第三，对系统工具的不间断使用可以提高标准化，以方便为某台计算机系统开发的代码更容易地移植到另一台计算机系统。系统库提供了一个抽象层，从而实现了在多台计算机系统中对同一函数的调用。一个应用程序可以调用一个系统库，而不用担心有关硬件的

细节。在这种情况下，只要目标系统具有系统的系统库，该应用程序就可以进行移植。对于图形而言，这就变得更加重要了，因为数量及硬件的显示能力已经得到了扩展。

对基本的系统文件结构的了解有助于程序管理。Unix 计算机系统通常包含一万多个与系统操作相关的文件(这还没有包括用户数据文件)。现在，已经有了用来组织这些文件的标准方法，也有了用于库、系统程序、设备文件(与硬件的连接)、应用程序和用户数据的公用空间。

最后是外壳环境。外壳环境有丰富的选项、功能和配置能力，对编程新手而言，掌握它是很困难的。然而，一旦熟练地掌握了它，外壳环境对任何正式的程序员而言都是一种功能强大的工具。在进程控制、系统管理和程序开发方面，它提供了极大的便利。

编写本书的目的有三个。第一，它支持关于工具和系统编程概念的教学；第二，它能够帮助读者提高其编程技能，超越其入门水准；第三，它提供了关于编程的缜密的实践步骤、允许读者实际操作和开发的示例，以及系统编程的概念和技巧。为了帮助读者达到这些目标，在本书的内容中，我们提供了示例代码和程序。在每一章的结尾，还提供了大量的问题和练习，用来加强读者对本章内容的理解。

除了系统编程的概念之外，本书还讨论了底层的数据类型：位和字节、位操作、数组、串、结构和指针。在介绍这部分内容的时候，我们强调了内存，以及如何和为什么要使用这些不同的数据类型。很正常，对于一名学生而言，相对于诸如循环和条件之类的其他基本编程概念，学习这些主题的内容会觉得不是很容易。关于底层数据类型的内容，我们努力尽早地涉及，旨在提高读者的编程水平，使其能够很好且自如地使用这些底层数据类型。

1.1.1 需要的背景知识

本书假定读者已经具备了对编程的基本了解，如变量、循环、条件和控制流等。例如，读者可能已经完成了一学期的 C 编程入门课程。如果读者学习的是不同的编程语言，如 Java 或 C++，那么学习一下 1.5 节，可以掌握等同于 C 语言句法(syntax)的背景知识。本书还假定读者拥有一台正在工作的计算机系统，并且安装了 C 编译器、文本编辑器、外壳程序(shell)和调试器。我们还假定读者熟悉计算机系统的基本操作，如在目录或文件夹层次结构中导航以及执行程序等。最后，我们还假定读者具备一些关于外壳程序的背景知识，入门级编程课程中介绍的相关内容就可以了。

1.1.2 为什么要用 Unix

本书的大部分内容都可以在任何一台计算机系统上进行学习，并且可以使用任何操作系统。然而，在写作本书的时候，如果没有认识到两个最主流的操作系统，则是很幼稚的，那就是 Windows 和 Unix。[①]出于以下给出的原因，本书鼓励读者在 Unix 系统上学习系统编程的概念。需要说明的是，我们讨论的话题是哪一个系统更适合学习。还有其他一些一直没有间断过的争论，即哪个操作系统具有更好的商业模式、更好的开发，以及其他的争论焦点。有兴趣的读者可以去查找关于这些争论的其他资料。

Windows 操作系统被设计用来简化计算机的使用。Windows 计算机是一个封闭的系统

① 在本书中，Unix 指的是任何类 Unix 系统，包括 Linux、Mac OS X、BSD 变种等。

(closed system)，按照向普通用户提供尽可能好用的系统这一商业模式，Windows 计算机系统被设计成为一个马上就可以使用的系统。这样的设计策略必定会避免普通用户更关心机箱内的东西从而对系统造成伤害。此外，微软公司就内部工作和 Windows 设计发布了受限信息，这也是一个向竞争者保护其设计的商业策略。Windows 是一个庞大的一成不变的系统，不允许系统的各个部分断开或相互交换。再一次说明，这是一个商业决策。微软要销售他们的产品，而且只是其产品。因此，他们使其产品全面集成。作为商业动机，Windows 的封闭式设计、技术细节的较少公布以及非模块化特点，就很容易理解了。然而，也正是这些有助于普通用户的特点，会使学习系统操作的学生感到沮丧。

另一方面，Unix，特别是 Linux，具有不同的设计理念。它是开放源代码的，因此，我们可以学习和研究它的内部工作细节。它是完全模块化的，因此，它的任何系统组件都可以进行交换或进行替代。例如，Linux 内核就是在桌面环境中完全独立地开发的。在这个核心组件中，Linux 计算机的操作者在如何配置系统的操作方面可以有很多的选择。内核本身可以进行替代或修改。有人可能会进行争论，这些特性阻碍了 Unix 在不关心便利性和开放性的普通用户之间的广泛流行。然而，也正是它的这些设计特性，使 Unix 对学习系统操作的学生具有很强的吸引力。

本书中的示例全部是在一台运行 Linux 操作系统的计算机上开发出来的。大部分情况下，这些示例也可以很容易地在 Windows 或其他 Unix 系统上运行。Unix/Linux 和 Windows 在设计上有一些重要的区别，如系统的多用户和单用户本质。在适当的地方，我将详细讨论这些不同。但除了这些考虑之外，在系统的深层，这两个系统的设计概念具有很大的相似性。

1.1.3　为什么要使用 C

在计算技术领域，特定编程语言的选择长期以来纷争不断。对应用程序开发而言，这样的争论依然如火如荼。然而，对系统编程而言，C 语言是专家们公认的编程语言。原因很简单，C 离硬件最近。所有编程语言都提供了很多抽象层来辅助程序开发。例如，与数值内存地址相比，命名变量的概念极大地简化了程序的开发工作。与所有常用编程语言不同，C 提供了最小的抽象，并且由此使 C 成为距离硬件最近的语言。多数 C 语句都可以简单地翻译成机器代码。在 C 语言中可用的数据类型被设计成反映硬件直接支持的数据类型。通过间接方式(指针)访问内存，则向程序员提供了访问系统所有组件的能力。

从历史上讲，Linux 内核的开发以及最初 Unix 操作系统的开发，都是使用 C 语言来完成的。大多数系统软件也是使用 C 语言来开发的。设备驱动程序几乎也总是使用 C 语言来完成的。距离硬件较近的一个间接好处就是速度。执行用 C 语言编写的代码要比执行用其他语言编写的代码速度快。对从事系统软件工作的人或希望开发密切与硬件(外围设备或主机系统)打交道的代码的人而言，学习使用 C 语言的概念会有机会提升他们最实用的编程技巧。

我们的这种选择并不是排斥其他编程语言的学习，或者提倡只学习 C 语言编程。本书没有提及的其他编程概念也很容易学习，并且可以使用另外一种编程语言容易地加以实现。然而，作者的观点是，充分了解最接近硬件的编程语言，有助于读者更好地理解一种更抽象的编程语言及其用法。

1.2 三个工具

系统程序员的三个主要工具是外壳程序(shell)、文本编辑器和调试器。熟悉这些工具的使用可以提高编程技巧并节省用来使程序正确运行所花费的时间。下面的内容将介绍这些工具及其相互依赖性。真正的挑战在于了解如何结合使用这三个工具。显然，编译器没有出现在这个列表中。编译器是一个强有力的工具，是程序开发过程中的好帮手。作为一个工具，我们将在本书的 6.1 节讨论程序构建的时候更全面地介绍编译器。

1.2.1 外壳程序

我们假定读者熟悉像生成一个文件列表这样的外壳程序的基本操作。有一些很好的书籍和在线参考材料，它们提供了操作某个特定外壳程序的必要内容。本书的附录 B 提供了一个常用命令列表。本节的内容将重点讨论为什么要使用外壳程序以及外壳程序对程序开发有何好处。

所谓外壳程序，就是允许用户运行其他程序的程序。通常情况下，外壳程序是在一台终端上运行的。在过去，终端就是连接到一台计算机的简单显示器和键盘，用来提供一个计算机的文本界面。现在，大多数操作系统都提供了可以有多台虚拟终端同时运行的图形界面，其中，每一台虚拟终端都运行一个独立的外壳程序。如果某台终端被用作系统管理或用来显示代表这台完整的计算机系统的错误信息，则这台终端即被用作一个控制台(console)了。在使用过程中，外壳程序、终端和控制台这几个名称经常互相交替使用。

多数 Unix 系统都提供了几种方法启动外壳程序。某些系统在登录后提供外壳程序，这没有图形桌面系统便利。在其他系统中，外壳程序必须通过一个菜单或鼠标单击界面手工启动。常见的外壳程序在启动后如图 1.1 所示。通过一个全文本界面，命令被输入到外壳程序。外壳程序通知用户，它正等待通过提示符接收下一条命令。在图 1.1 中，提示符是 `ahoover@video>`，即用户名和机器名。某些外壳程序的配置在提示符中显示了当前的目录或其他信息。在本书中，为清楚起见，外壳程序提示符今后都显示为 `ahoover@video>`。

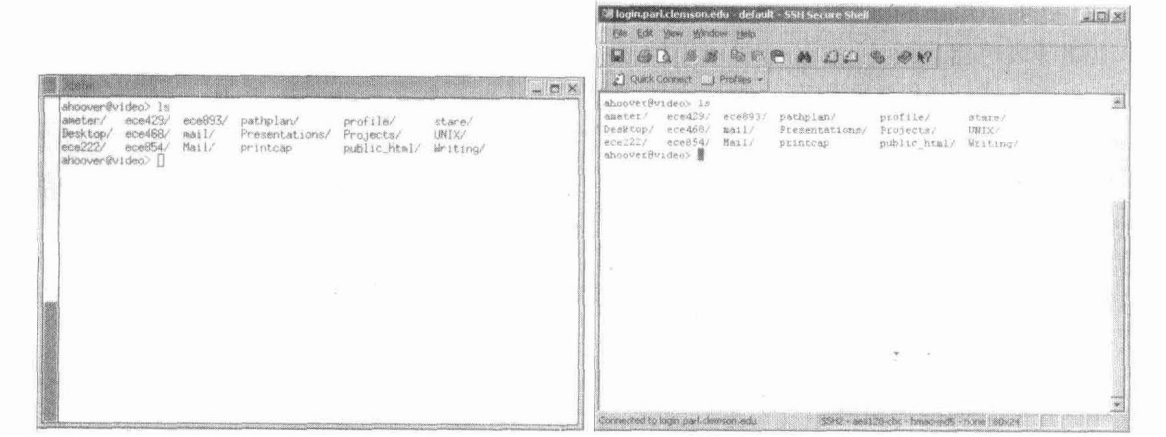


图 1.1 运行在不同终端模拟器下的两个外壳程序

典型的命令就是一个程序的名称，它开始这个程序的执行。例如，在图 1.1 的两个图中，就说明了程序 `ls` 的执行，它提供了当前目录内的文件列表。类似于 `ls`，许多在外壳中运行的程序都是全文本输入和输出。然而，也可以在外壳程序中运行基于图形(GUI)的程序。例如，在图 1.2 中，给出了在提示符下输入 `xclock` 后的结果，它启动了程序 xclock 的执行。

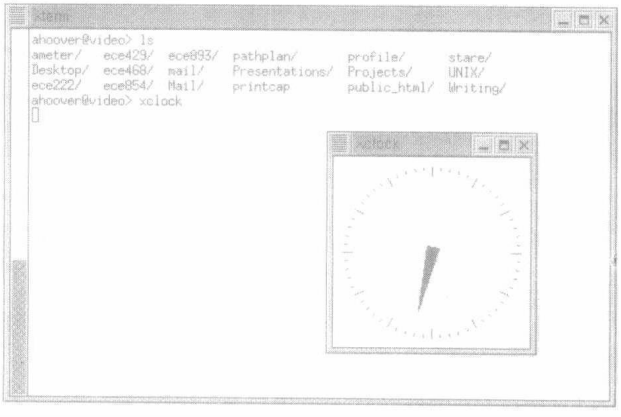


图 1.2　通过在外壳程序中输入程序的名称启动一个基于 GUI 的程序

对那些熟悉在如今桌面系统中运行程序的人而言，这种启动程序的方法可能看上去有些奇怪。单击一个图标，查看弹出的下拉菜单，这便是运行程序的典型操作。为什么外壳程序要如此启动一个程序来运行其他的程序呢？答案是为了灵活。在程序如何执行方面，桌面鼠标和菜单操作提供的选择是有限制的。一般情况下，桌面和菜单方式是以默认模式来迅速运行程序的。例如，启动一个字处理程序就会以全屏幕模式(程序充满整个屏幕)打开该程序，并打开一个空白页面，并且所有的选项都被设置成默认状态(取消粗体、文本左对齐，字体是 Times Roman 等)。如果一个人在启动该字处理程序的时候希望有些选项有所变化，那该怎么办？通过命令行参数(command line argument)，外壳程序就可以做到这一点。命令行参数指的是在外壳程序提示符下在要执行的程序名称之后输入的任何内容。它提供的是用户希望如何执行程序的信息。例如，在提示符下输入 `xclock-help`，会产生下面的显示：

```
ahoover@video> xclock -help
Usage: xclock[-analog][-bw <pixels>][-digital][-brief]
    [-utime][-fg <color>][-bg <color>][-hd <color>]
    [-hl <color>][-bd <color>]
    [-fn<font_name>][-help][-padding <pixels>]
    [-rv][-update <seconds>][-display displayname]
    [-geometry geom]
ahoover@video>
```

为响应命令行参数`-help`，程序 `xclock` 显示了其使用方法，然后停止下来并退出。程序的使用方法解释了在启动程序时可以使用的所有命令行参数。对这个特定的程序而言，大多数参数是改变时钟显示的方式。例如，`xclock -digital -bg grey` 可使程序运行后显示如图 1.3 所示的结果。

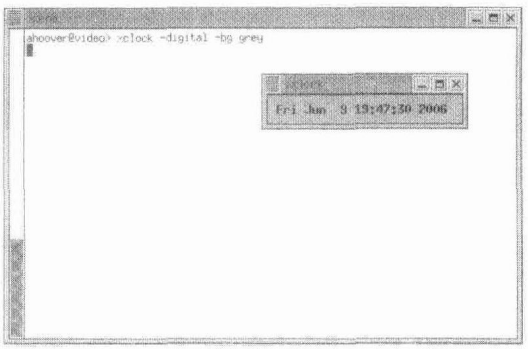


图 1.3　命令行参数改变了程序运行的方式

在程序开发和系统管理中，命令行参数所提供的控制能力及灵活性通常都非常有用。虽然很多程序在运行的时候都可以重新进行配置，但通过菜单界面选择选项需要占用时间。通过命令行参数在启动的时候对程序进行配置可以节省大量的时间，尤其是在开发过程中一个程序要运行多次的时候。

在过去几年中，人们开发了一系列外壳程序，这些示例包括 sh、csh、tcsh、ksh 和 bash 等。在 Windows 系统中，有一个非常简单的被称为控制台的外壳程序，有的时候，这个外壳程序也被称为“DOS 控制台”或“命令提示符”。这些外壳程序在内部的功能上是有区别的。除了具有运行程序的功能并提供命令行参数之外，外壳程序都有一个它可以执行的内部命令的列表。例如，大多数外壳程序都具有为需要经常输入的命令建立别名的功能。在外壳程序 tcsh 中，输入以下命令：

```
ahoover@video> alias xc  xclock -digital -bg grey
ahoover@video>
```

就会使较短的命令 xc 成为较长的命令 xclock -digital -bg grey 的一个别名。这在我们需要反复运行同一命令的时候，如在程序调试过程中，就可能相当有用。在表 1.1 中列出了一些外壳程序的常用内部命令。对所有最流行的外壳程序(尤其是在执行被设计成是一个受限外壳程序的 Windows 控制台程序的时候)而言，所有这些命令都是最常用的。不幸的是，不同的外壳程序使用不同的句法来实现某些内部命令。例如，针对前面给出的用于外壳程序 tcsh 的例子，如果要使用外壳程序 bash 来创建相同的别名，就得输入以下命令：

```
ahoover@video> alias xc="xclock -digital -bg grey"
ahoover@video>
```

表 1.1　某些常用的外壳程序内部命令

命令	描述
alias	创建一个别名
cd	改变目录
pwd	打印当前工作目录
set	给一个外壳程序变量赋值
which	标识程序的完整路径

多数高级程序员只选择一个他能够熟练使用的外壳程序。幸运的是，大多数外壳程序都很相似，所以熟悉了某个特定的外壳程序之后，程序开发人员也可以很容易地使用任何一个外壳程序进行工作。

除了内部的外壳程序命令之外，还有一些可在大多数 Unix 系统中运行的预先编译好的程序。在表 1.2 中，列出了这方面的一些常用程序。我们将这些程序称为系统程序，因为一般来讲，它们都具备为计算机系统操作、探究和开发程序的功能。例如，`ls` 就是一个系统程序，它提供当前目录中的文件列表。或许，最重要的系统程序是 `man`。它访问的是存储在本地计算机系统中的帮助文件手册。通常，它们是用于所有程序的独立的“手册页”，并且经常被用作更复杂程序的支持文件。在系统中，还有一些用于各种库中所有函数的手册页。这些手册页通常被默认安装在一个 Unix 系统中，但也经常通过 Internet 多次在网络上发布，通过一个网络搜索引擎可以找到它们。在使用某个特定外壳程序时，是有可能(也是我们推荐的方法)通过网络发现相关的教程或其他帮助材料的。

表 1.2　某些常用的系统程序

命令	描述
gerp	针对指定文本检索文件
ls	列出文件和它们的属性
man	显示用于命令/程序的(帮助)手册
more	通过暂停滚动方式显示文本文件
time	测量程序的运行时间
sort	在一个文本文件中对文本行进行排序

使用外壳程序内部命令和系统程序的目的就是要帮助系统程序员的工作。记住一个特定的操作是外壳程序命令还是一个系统程序并不是很重要。有的时候，这些操作会被列在一起。重要的是，要能够自如地使用这些能够节省时间和精力的常用操作。在 5.3.1 小节我们讨论管道链(pipeline chaining)的内容时，将再次讨论这些程序。在附录 B 和附录 C 中，我们提供了更长的列表。

1.2.2　文本编辑器

我们这里讨论的第二个工具是文本编辑器。文本编辑器的基本操作可以使用户编写和编辑代码、将它们保存为一个文件，将它们从一个文件中加载进来等。这些操作与一个字处理程序所支持的那些操作并无太大区别。事实上，使用字处理程序来编写代码也是有可能的(虽然不推荐这样做)。然而，除了一个典型的字处理程序所能够提供的功能之外，文本编辑器能够提供更多额外的用来支持编程的功能。例如，图 1.4 演示的就是括号匹配查找功能。使用文本编辑器 `vi`，如果光标位于一个左括号或右括号上，按百分号键(%)，即可将光标移至匹配的括号上。再按一次该键，光标就会回到它开始的位置。同样的击键方式还可以匹配左引号和右引号(它们用来标识代码块)以及左右中括号(它们用来声明数组)。对文本编辑器 `gedit` 而言，括号匹配功能是通过一个菜单选项来激活的。对文本编辑器 `emacs` 而言，括号匹配功能会简单地在每次输入右括号时将光标移动到相应的左括号，动