PEARSON

现代体系结构上的UNIX系统

内核程序员的对称多处理和缓存技术

（修订版）

[美] Curt Schimmel 著
张辉 译
田春 审校

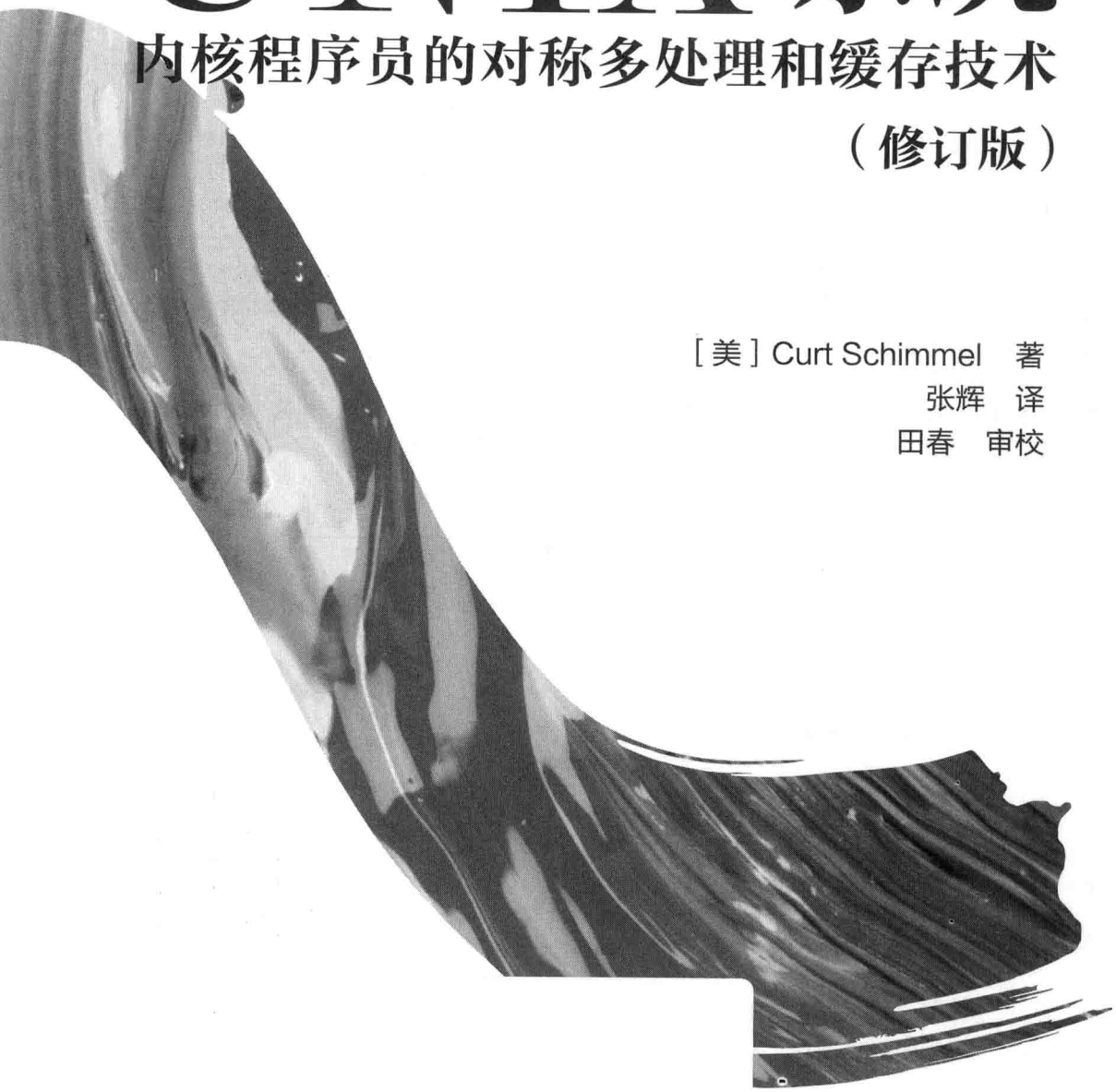
人民邮电出版社
POSTS & TELECOM PRESS

UNIX Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers

现代体系结构上的UNIX系统

内核程序员的对称多处理和缓存技术

（修订版）



[美] Curt Schimmel 著
张辉 译
田春 审校

人民邮电出版社
北京

图书在版编目（CIP）数据

现代体系结构上的UNIX系统 : 内核程序员的对称多处理和缓存技术 / (美) 希梅尔 (Schimmel,C.) 著 ; 张辉译. -- 2版（修订本）. -- 北京 : 人民邮电出版社, 2015.1
ISBN 978-7-115-35758-8

Ⅰ. ①现… Ⅱ. ①希… ②张… Ⅲ. ①UNIX操作系统 Ⅳ. ①TP316.81

中国版本图书馆CIP数据核字(2014)第238916号

内 容 提 要

本书首先回顾了与全书其他内容切实相关的 UNIX 系统内幕。回顾的目的是增进读者对 UNIX 操作系统概念的了解，并且定义随后使用的术语。本书接下来的内容分为 3 个部分。第一部分介绍了高速缓存体系结构、术语和概念，详细考察了 4 种常见的高速缓存实现——3 种虚拟高速缓存的变体和物理高速缓存。第二部分讨论了调整单处理器内核的实现，使之适合于紧密耦合、共享存储多处理器上运行时所面临的问题和设计事宜，还研究了几种不同的实现。最后一部分介绍多处理器高速缓存一致性，这一部分通过研究高速缓存加入到一个紧密耦合、共享存储器多处理器系统时出现在操作系统和高速缓存体系结构上的问题，从而将前两个部分的内容结合到一起。

本书适合于大学计算机及相关专业高年级本科生或者研究生使用。每一章都包含有一组练习题，问题都需要采用这一章所提供的信息以及一些额外学到的知识来解答，习题大都建立在这一章中所出现的例子的基础之上。在本书的末尾有选择地给出了习题的答案。

◆ 著　[美] Curt Schimmel
译　张　辉
审　校　田　春
责任编辑　杨海玲
责任印制　彭志环

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：787×1092　1/16
印张：19
字数：452 千字　2015 年 1 月第 2 版
印数：4 001-7 000 册　2015 年 1 月河北第 1 次印刷
著作权合同登记号　图字：01-2014-6465 号

定价：59.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

版权声明

修订版序

本书是对多年前出版时采用的译文参照英文原著进行审校后重新出版的产物。通常，作为一个审校者是不应该留下序的，而且我也不能算是该领域的专家，顶多只能说是在开发并行软件的工作中用到了一些来自本书的知识，然后推动了出版社下决心再版此书的过程。按照国内的出版传统，再版一本已出版十年之久且英文原著并无更新的书是罕见的。我的看法是，这本书的内容既没有过时，也没有被任何后续出版的其他教材所取代，因此为了让广大读者继续得以通过正规渠道学习这些知识，唯一的途径就是再版此书。因此我谨代表广大读者真心感谢人民邮电出版社能够做出这个努力。

当今关于计算机软件和硬件的书本知识正在愈发脱节。关于硬件的知识和汇编语言的教材都还停留在早期的处理器型号上，学完以后对日常编写软件帮助不大（因为编译器的优化能力日益提高，很少需要插入手工的汇编代码），而面向软件和高级语言的知识则越发抽象，层层封装，学来学去，最后只学会了使用各种库而已。计算机的现代体系结构从未改变过，各种语言和虚拟机的内存模型本质上都来源于硬件在性能和可靠性上的权衡。这些权衡就体现在对高速缓存的广泛使用上，可惜关于该领域的知识却很少被提及。

如果读者的最终目标是实现最高性能的多线程程序，那么我认为本书的阅读至少可以起到打基础的良好作用。一旦从硬件层面理解了高速缓存和原子指令的本质，那么接下来就可以从著名的《多处理器编程的艺术》一书中继续汲取营养，或者通过另一本网上可以免费找到的《Is Parallel Programming Hard, And, If So, What Can You Do About It?》进一步了解在某个具体的操作系统（Linux）中，内核和应用程序是如何有效利用对称多处理器和各种高速缓存的了。

所以，本书算是填补了软件和硬件知识之间的空白，把一切都关联起来了。另外，本书是操作系统无关的，并非书名所暗示的仅适用于 UNIX。不过类 UNIX 系统普遍具有一种简单性：极少量的稳定的系统接口用来处理几乎所有的事务，因此以 UNIX 为蓝本来讲述硬件知识有助于读者迅速地拨云见日，看到真正有用的东西，这是好事。相比之下，即便非常熟悉 Windows 系统和各种 Win32 API 的人也未必真正了解 Windows 内核原理，按照一本极其有趣的书《Rootkit：系统灰色地带的潜伏者》中的说法，微软公司在 Windows 内核中所设计的虚拟内存模型，其复杂度令英特尔的虚拟内存机制相形见绌。

本书审校过程中的主要改动如下：对于一些计算机术语的译文，参照目前流行的译法以及微软的计算机术语库做了调整。例如，context 原本翻译成“现场”，现译成“上下文”；transaction 原本翻译成“交易”，现改为“事务”。最明显的改动是几乎所有的“多处理机”全部改为了“多处理器”，因为按照百度百科的解释，“多处理机”是更为广义的概念，相比之下本书只讨论单台传统意义计算机上的对称多处理器系统。此外，一些原本翻译得不准确的句子也有改动。虽然不敢说比原译者理解得更准确，但有了前人的译文再考虑更贴切的译法，总比自己完全重译得到的文字要好一些，所以绝不敢贪功。不过也不排除一些原本翻译不够准确的句子在此次审校过程中

竟没能发现，对于这类情况就只能希望读者见谅了。

总而言之，希望所有各方为了再版此书的所作的努力都是值得的，也希望这本书能够真正对广大读者起到帮助作用。

田春

2014 年 9 月 21 日

于意大利博洛尼亚

序

本书的目标在于，为了让操作系统能够在采用了高速缓存（cache memory）以及（或者）多处理器（multiprocessor）技术的现代计算机系统上运行，就若干必须要解决的问题提供实用的信息。在撰写本书的时候，已经有一些讲述 UNIX 系统实现的书籍，但是却没有一本书详细描述应该如何管理高速缓存和多处理器。许多计算机体系结构方面的书籍都是从硬件方面介绍高速缓存和多处理器的，但是却没有一本书能够成功地解决这些现代体系结构给操作系统所带来的问题。本书的意图就是通过建立计算机体系结构和操作系统之间的桥梁来填补这些缺憾。

本书是为操作系统开发人员编写的，它从系统程序员的角度阐述了高速缓存和多处理器的操作。虽然本书的读者对象是 UNIX 系统程序员，但是本书的编写形式使之能够适用于任何操作系统，其中包括所有的 UNIX 变体。通过概念层次上阐述的问题和解决方案，并且以使用 UNIX 系统服务（system service）为例来展示这些问题出现的地方，从而达到了这一效果。所以本书提供的解决方案也可以应用到相应环境下的其他操作系统上。

本书打算采取两种途径向操作系统开发人员提供帮助。首先，读者将学习如何调整现有的操作系统，使之能够在现代体系结构上运行。为了做到这一点，本书从操作系统的角度来详细研究这些体系结构的操作，并且阐述了操作系统要管理这些体系结构必须做什么。其次，读者将学习现代体系结构在不同方法之间进行抉择时所涉及的若干权衡考虑。在投身于采用高速缓存和多处理器的新型计算机系统设计工作时，这会给予操作系统开发人员所需要的背景知识。

本书假定读者熟悉 UNIX 系统调用接口（system call interface）和 UNIX 内核原理的高级概念。读者还应该熟悉计算机体系结构（computer architecture）和计算机系统组织（computer system organization），应具备计算机科学系本科生水平的相应课程所教授的知识。

本书扩充了为计算机产业界的 UNIX 系统专业人员所开发的 I 级课程（course I）。在过去的 4 年中，这门课一直在美国的 USENIX 大会和欧洲的 UKUUG 大会上讲授。由于它是一门为期一天的辅导课，因而在所含材料的数量上就有所限制。本书更为细致地涵盖了有关高速缓存和多处理器课程的所有材料，并且还包含了其他主题。

本书适合于大学高年级本科生或者研究生使用。每一章都包含有一组练习题。选出的问题都需要采用这一章所提供的信息以及一些额外学到的知识来解答，并不是简单地模仿他人的材料来出题。在许多情况下，习题都建立在这一章中所出现的例子的基础之上。答案往往采取一小段话的形式（大多数情况下有 4~5 句话，有时会长一些）。希望读者能够尝试解答所有的问题，以此增进对所学概念的掌握。在本书的末尾有选择地给出了习题的答案。

我们首先回顾了与本书其他内容切实相关的 UNIX 系统内幕。回顾的目的是增进读者对 UNIX 操作系统概念的了解，并且定义随后使用的术语。本书接下来分成 3 个部分：高速缓存存储系统（cache memory system）、多处理器 UNIX 实现以及多处理器高速缓存一致性（cache consistency）。第一部分“高速缓存存储系统”介绍了高速缓存体系结构、术语和概念。接下来详

细讲述了 4 种常见的高速缓存实现：3 种虚拟高速缓存（virtual cache）的变体和物理高速缓存（physical cache）。第二部分“多处理器系统”讨论了调整单处理器（uniprocessor）内核的实现，使之适合于在紧密耦合（tightly-coupled）、共享存储多处理器（shared memory multiprocessor）上运行时所面临的问题和设计事宜，还研究了几种不同的实现。最后一部分介绍多处理器高速缓存一致性，这一部分通过研究高速缓存加入到一个紧密耦合、共享存储器多处理器系统时所出现的操作系统和高速缓存体系结构上的问题，从而将前两个部分的内容结合到一起。

本书因地制宜地使用一组经过挑选的现代多处理器体系结构来举例说明相关的概念。其中 Motorola 68040 和 Intel 80X86 系列（80386、80486 和 Pentium）代表了传统的 CISC（complex instruction set computer，复杂指令集计算机）处理器。RISC（reduced instruction set computer，精简指令集计算机）方法则以 MIPS 系列（R2000、R3000 和 R4000）、Motorola 88000 以及来自德州仪器公司（Texas Instruments，TI）的 SPARC v8 兼容处理器（MicroSPARC 和 SuperSPARC）为代表。另外还出现了其他几个例子，包括 Sun 和 Apollo 工作站和 Intel i860。在附录 A 中可以找到这些处理器特性的一份汇总介绍。

我要向在本书付梓之前耗费时间评审书稿的人士表示我的感激之情。特别要感谢下面这些人：Steve Albert、Paul Borman、Steve Buroff、Clement Cole、Peter Collinson、Geoff Collyer、Bruce Curtis、Mukesh Kacker、Brian Kernighan、Steve Rago、Mike Scheer、Brian Silverio、Rich Stevens、Manu Thapar、Chris Walquist 和 Erez Zarok。我也要向 Addison-Wesley 公司的员工们表示感谢，感谢他们在本书出版的过程中所提供的帮助和建议，特别要感谢 Kim Dawley、Kathleen Duff、Tiffany Moore、Simone Payment、Marty Rabinowitz 和 John Wait。他们的帮助使得这本书比我一个人努力的结果要好得多。我还要感谢许多在上辅导课期间花时间填写课程评语，从而提供其深思熟虑后的反馈意见的人士。

前　言

在计算机系统发展历史中的许多时期，构建整体上速度更快的系统的愿望都集中在系统的三大组成部分——CPU、存储子系统和 I/O 子系统——的速度都更快上面。通过提高时钟速度就可以制造出更快的 CPU。通过降低存取时间就可以制造出更快的存储子系统。通过提高数据传输速率就可以制造出更快的 I/O 子系统。但是，随着时钟速度和传输速率的提高，要提高系统的整体速度就变得越来越困难了，因此要设计和构建这样的系统，成本也变得越来越高。随着速度的提高，传输延迟（propagation delay）、信号上升时间（signal rise time）、时钟同步和分发（clock synchronization and distribution）等都变得越发重要起来。这类高速设计的高成本更难获得有效的性能价格比。

因为受这样和那样的因素影响，系统设计人员扩大了他们的关注范围，以找出提高系统整体性能的其他途径。精简指令集计算机（Reduced Instruction Set Computer，RISC）系统的概念就是其成果之一，在 RISC 系统中对 CPU 指令集进行了简化，从而让一个低成本的快速硬件实现就能完成这些指令。另一项成果是高速缓存存储系统的开发。高速缓存通过把程序中引用最频繁的数据和指令保存在一小块高速存储器中，以此降低主存储系统的负载，从而提高系统的性能。通过增加一小块成本划算、高速度的高速缓存（而不是一个成本高、规模大的高速主存储子系统）就能加速存储器整体存取速度。采用高速缓存存储器有可能带来更快的存储器整体存取时间，这对于 RISC 系统来说尤为重要，因为要完成相同的任务，使用精简的指令集通常要求 RISC CPU 比传统的 CPU 体系结构取得和执行更多的指令。一般而言，RISC 系统需要更高的带宽来以峰值性能运行。

通过并行运行更多台设备而不是提高任何单台设备的速度就能获得更高的 I/O 传输速率。这就导致了诸如廉价磁盘冗余阵列（Redundant Arrays of Inexpensive Disks，RAID）之类设备的发展，在 RAID 中，多块磁盘并行运行以提供更高的整体传输率。通过增加一个系统中 CPU 的数量来构建多处理器，这样的技术也可以用于提高 CPU 的速度。多处理器把系统负载分散到多个处理器上来增加整个系统的吞吐率。

多处理器和高速缓存是密切相关的。紧密耦合多处理器系统（tightly coupled multiprocessor system）有一个共享的主存储子系统，随着处理器数量的增加，它需要更高的主存储带宽，因为每个处理器在取得和执行一条独立的指令流的同时，都必须在存储器中访问一组独立的数据。将一块高速缓存和每个处理器进行耦合，从高速缓存而不是共享的主存储器来满足处理器大部分的存储器访问请求，就可以降低主存储器的负载。这是一种颇为划算的提高系统性能的途径。

虽然高速缓存能够在多处理器中增加有效的存储器带宽，但是高速缓存结构对于管理它所需要的操作系统开销有很大的影响，这又反过来影响了系统的整体性能。

总而言之，构建速度更快的计算机系统不仅仅是一件诸如提高 CPU 时钟速度这样的事。虽然这样的技术实际上造就了更快的系统，但是它们不一定就是经济上划算的解决方法。通过集中

研究如何利用现有的系统部件来提供更高的系统性能，人们已经发现高速缓存和多处理器是划算的解决方案。因此，我们就从这里开始研究高速缓存和多处理器的体系结构，以及它们给操作系统带来的问题。

符号约定

本部分举例说明在本书的示例中所采用的符号约定。

常数

十进制、八进制和十六进制常数都以 C 编程语言的标准记法来表示。十进制常数总是以数字 1~9 中的一个开头。八进制常数以 0 开头。十六进制常数以字符序列“0x”开头。这里是一些例子：

12345	十进制常数
07654	八进制常数
0x3af	十六进制常数

字长

存储器中的一个字（word）为 4 字节（byte）。每字节为 8 位（bit），因此每一个字是 32 位。

字节顺序

存储器最容易想成是字的数组（array）。要引用一个字中的单个字节，需要有一种给它们编号的约定。所有展示存储器中字节排列的例子都使用了大端（big-endian）的字节顺序。这意味着每个字的最高有效字节（Most Significant Byte，MSB）拥有最低的地址，而最低有效字节（Least Significant Byte，LSB）拥有最高的地址。在字中的字节是从左到右读取的，首先读取最高有效字节。例如，下面的字节编号方式会运用到字长为 4 字节的机器上。

	MSB			LSB
字 0	字节 0	字节 1	字节 2	字节 3
字 1	字节 4	字节 5	字节 6	字节 7
字 2	字节 8	字节 9	字节 10	字节 11

Motorola 的处理器、Sun SPARC 兼容类型的处理器以及 IBM 的处理器（IBM PC 的处理器除外）都使用大端的字节顺序。DEC 和 Intel 处理器使用小端（little-endian）的字节顺序，在这类处理器中，一个字里面的字节是按照逆序编号的。MIPS 处理器可以配置成任何一种字节顺序，在生产基于 MIPS 处理器的系统的所有厂商中，除了 DEC 之外，都选择大端的字节顺序。

位顺序

在一个字或者字节内，各个位的顺序遵循低字节结尾的顺序，这意味着最低有效位（least significant bit，LSb）的编号为 0 位，而 4 字节长的字中，最高有效位（most significant bit，MSb）是 31 位。最高有效位始终在最左边。在需要用位的编号来说清楚一个例子的时候，就在一个字节或者字的上方给出位的编号来。例如，下面的字显示出了每个字节内的最高和最低有效位，每个字节则按照字内的位顺序进行编号。

31 24	23 16	15 8	7 0
字节 0	字节 1	字节 2	字节 3

位范围

有时需要从一个字或者字节中引用一串连续的位，这就称为位范围（bit range）。位范围可以用尖括号括起范围两端的位号来表示。例如，在上面所示的一个字中，构成字节 2 的位序列被表示成“位<15..8>”。最高有效位号始终出现在左边，右边跟着最低有效位号。这种记法所表示的范围始终包括这两个位的位置本身。

存储器大小的单位

千字节（kilobyte）被缩写成 KB，它包含 1024 字节。兆字节（megabyte）被缩写成 MB，它包含 1024 KB。吉字节（gigabyte）被缩写成 GB，它包含 1024 MB。例如，4 KB 是 4096 字节，而 8 MB 是 8 388 608 字节。存储器大小始终是以字节来衡量的。KB、MB 和 GB 等缩写均会在本书中使用。

目　录

第1章 回顾 UNIX 内核原理

本章回顾了 UNIX 内核原理的有关内容，在以后各章中会用到它们。这里没有完整地讨论这个主题，而是作为那些已经熟悉基本概念和术语的人对这些内容进行的一次复习。本章的内容涉及单处理器系统。多处理器 UNIX 系统的实现是本书第二部分的主题。不熟悉 UNIX 操作系统或者 UNIX 内核原理的读者应该首先从本章末尾给出的参考文献中选出一些阅读。

1.1 引言

UNIX 系统是一种多用户、多任务操作系统，它提供了高度的程序可移植性以及丰富的开发工具集合。UNIX 系统取得成功的一部分原因在于它提供的可移植的应用程序接口集合。这一接口集合能够轻而易举地处理把应用程序从一家厂商的系统移植到另一家厂商的问题。UNIX 取得成功的另一部分原因在于操作系统、命令和库（library）本身的编写都可以轻松地移植到不同的计算机上，从而促进了市场上 UNIX 硬件平台的多样性。

UNIX 系统在逻辑上具有分层的结构，可以分成两个主要部分：内核（kernel）和用户程序（user program）。图形化的表示如图 1-1 所示。

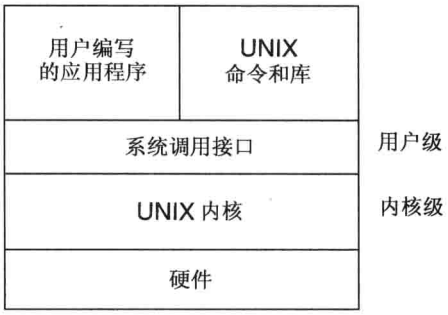


图 1-1 UNIX 系统的逻辑分层

内核的用途是与硬件接口并且控制硬件。内核还向用户程序提供一组抽象的系统服务，称为系统调用（system call），使用可移植的接口就能够访问系统调用。内核在内核级（kernel level）上运行，在这个级别上它能够执行特权操作。这能让内核完全控制硬件和用户级程序，并且提供一个让所有的程序协调共享底层硬件的环境。