操作系统的三个容易片段

REMZI H.ARPACI-DUSSEAU
ANDREA C. ARPACI-DUSSEAU
UNIVERSITY OF WISCONSIN-MADISON

May 14, 2016

献给 *Vedat S. Arpaci*，一位人生典范

前言

给所有人

欢迎来到本书。我们希望你会像我们享受写作本书一样享受阅读它。这本书叫做**操作系统 Three Easy Pieces**，书名显然是向费曼在物理学上的一部有史以来最伟大的讲义 (F96) 致敬的。虽然本书毫无疑问达不到像那位著名物理学家的高水准，但也许它对寻求理解整个操作系统（或更广义地，整个系统）的你来说也许是足够的。

三个简单部分指本书围绕的三个主要专题：**虚拟化**，**并发性** 和 **持久化**。在讲这些概念时，我们会讨论大多数操作系统执行的重要事情；希望你在这过程中也会感到有趣。学习新东西应该很有趣对吧，至少应该有趣。

每个主要的概念都被分为好几个章节，其中大部分章节是提出一个具体问题然后展示如何解决它。每章都很短，并且尽量参考那些最初提供那些想法（概念的原型）的材料。我们写作本书的一个目标就是使历史的路径尽可能清晰，因为这样将帮助学生更清楚的理解"它现在是什么、它曾经是什么、它将来会是什么"。在这里，知道香肠是怎么制作的和理解香肠适合做什么同样重要[1]。

有一些本书通篇会使用的方法值得在这里提一下。第一个就是问题的 **症结**。任何时候我们试图解决一个问题时，首先就是尝试陈述问题最重要的焦点是什么；在行文中想这样的**问题的症结** 会被直接指出，而且希望通过后面呈现的一些技术、算法和想法来解决。

文中也有许多为正文提供一些色彩的**旁白** 和 **提示**。旁白通常讨论和正文相

[1]提示：吃！或者如果你是一个素食主义者，走开

关的东西（但可能不是本质的）；提示通常是可以应用到系统的一般经验。为了方便，本书末尾有一个列出了所有旁白、提示和症结的索引。

我们在本书中还使用了一种古老的说教方法：**对话**，作为呈现一些材料的不同方式。对话被用来引入主要概念（以一种很自然的方式，你会看到）以及时不时回顾一下材料。对话也可以作为写的更幽默一些的机会，不够你发现它是有用还是幽默，恩，这是另一回事了。

在每个主要部分的开始处，我们首先给出操作系统所提供的 **抽象**，然后在后续章节里探索为提供这个抽象所需要的机制、策略和其他需要的支持。抽象是计算机科学所有方面的基础，所以也毫无疑问是操作系统的本质。

在各个章节里，我们在可能的地方都使用**实际代码**（而不是**伪代码**），所以基本上所有例子，你都可以能够输入和运行它们。在真实系统上运行实际代码会是学习操作系统最好的方法。所以我们也鼓励你可以这样做时就去做。

在本书的不同地方，我们都搞了点**家庭作业** 来保证你理解文中内容。许多这样的家庭作业都是操作系统部件的小仿真；你应该下载这些作业，运行他们来测验自己。家庭作业的仿真器有一下特征：通过给他们不同的随机数种子，你几乎可以产生无穷的问题集；仿真器也可以被要求解决你的问题。因此，你可以测试、再测试自己直到你达到一个比较好的理解程度。

本书最重要的补充是一些**工程**，通过设计、实现和测试你自己的代码，你可以学到真实系统是如何运行的。所有的工程（包括代码示例，前面提到过）用的都是 C 语言 **C 语言** (KR88)；C 是大多数操作系统底层采用的一种简单强大的语言，很值得你将它加入你自己的语言工具箱。有两种类型的工程可供选择（到在线附录里去）。第一种是 **系统编程** 类项目；这种工程很适合想学底层 C 编程的 C 和 UNIX 新手。第二种是基于 MIT 开发的一个叫 xv6(CK+08) 的真实操作系统内核；这种工程很适合那些想深入到操作系统内部的已经具备了 C 语言基础的同学。在华盛顿，我们以三种不同方式开过这个课：全是系统编程，全是 xv6 编程，或者这两者混合。

给执教者

如果你是想使用本书的教师或教授 [1]，请自由使用。你也许注意到，它们可以在下面的网页上自由获取：

`http://www.ostep.org`

你也可以在`lulu.com`上买打印的副本。在上面的页面上可以找到。

目前可以像这样来引用本书：

Operating Systems: Three Easy Pieces
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
Arpaci-Dusseau Books, Inc.
May, 2014 (Version 0.8)
`http://www.ostep.org`

这个课程分成 15 周的学期比较好，这样你可以以一个合理的深度讲完大部分主题。将课程压缩到一个 10 周的季度可能要丢掉一些细节。也有些章节是关于虚拟机监视器的，我们通常将其挤到一个学期的不同时候，要么在虚拟化部分结束时，要么在接近结束时作为旁白。

本书一个稍微不同的地方是并发性部分，大多数操作系统的书籍将这个主题放到前面，本书将其推后到学生已经对 CPU 和内存的虚拟化有了一定理解之后。教授这门课的将近 15 年的经验显示，如果不理解什么是地址空间，什么是进程，或者为什么上下文切换可以发生在任意时间点，那么学生将很难理解并发问题是怎么产生的，或者为什么他们会尝试去解决并发问题。而一旦他们理解了那些概念，引入线程记号和由此引发的问题就会变得相对容易，或者至少更容易些。

你也许注意到本书没有附带幻灯片。这个缺失的主要原因是我们相信最老式教学方法：粉笔和一块黑板。所以，当教授这门课时，我们带着几个主要观念和几个例子，然后用黑板呈现它们；讲义文稿和少许代码示例也是很有用的。以我们的经验，使用过多的幻灯片会使学生们仅仅“登记”一下课程（然后就去刷脸书），因为它们知道材料就在那等着他们稍后消化；使用黑板使课具有生动的体验，因而（希望）对课上的学生来说更具有交互性、动态性和有趣。

如果你想要一份我们备课的笔记，请发封邮件索取。我们已经将其分享给了全世界的很多人了。

最后一个请求：如果你使用在线免费章节，请不要做本地拷贝，直接**链接**到章节就是。这将帮助我们追踪使用情况（过去几年有超过一百万次下载！）也会保证学生们得到最新的版本。

给学生

如果你是阅读本书的学生，非常感谢！我们很荣幸能提供一些资料帮助你追寻有关操作系统的知识。我们都深情的回忆起我们本科阶段的一些教科书（例如，Hennessy and Patterson(HP90), 计算机架构的经典教科书）并希望本书会成为你具有良好回忆的书中的一本。

你也许注意到本书可以在线免费获取。这样做的一个主要的原因是：教科书通常都太贵了。这本书，我们希望，会是帮助那些在追寻良好教育的人的第一波免费材料，不管他们是来自世界的那个地方或他们愿意为书本花多少钱。如果做不到那样，这本免费的书，也总好过没有。

我们同时也希望，在可能的地方，向你指出本书许多材料的最初来源：那些伟大的论文和那些在操作系统领域耕耘多年的人。想法不是凭空产生的；它们来自那些聪明并且努力工作的人（包括许多图灵奖获得者[2]），因此我们在可能时应该努力赞扬那些想法和那些人。这样，我们希望能更好的理解那时发生的变革，而不是像那些观点从来就存在一样书写 (K62)。更进一步地，这样的引用文献也许会鼓励你自己更进一步的深究；阅读本领域内的著名论文的确会是一种最好的学习方法。

[2]图灵奖是计算机科学的最高奖项，就像诺贝尔奖，只不过你可能从来没听说过。

致谢

本节包含所有帮助过本书写作的人的感谢。最重要的事是：**你的名字也可以在这里！**但是，你必须先提供帮助。所以请给我们发送反馈并帮助本书除错。然后你就可以出名了！或者，至少将你的名字写入一些书里。

到目前为止帮助过我们的人包括：Abhirami Senthilkumaran*, Adam Drescher* (WUSTL), Adam Eggum, Ahmed Fikri*, Ajaykrishna Raghavan, Akiel Khan, Alex Wyler, Anand Mundada, B. Brahmananda Reddy (Minnesota), Bala SubrahmanyamKambala, Benita Bose, BiswajitMazumder (Clemson), Bobby Jack, Bj ¨orn Lindberg, Brennan Payne, Brian Kroth, Cara Lauritzen, Charlotte Kissinger, Chien-Chung Shen (Delaware)*, Christoph Jaeger, Cody Hanson, Dan Soendergaard (U. Aarhus), David Hanle (Grinnell), Deepika Muthukumar, Dorian Arnold (NewMexico), DustinMetzler, Dustin Passofaro, Emily Jacobson, EmmettWitchel (Texas), Ernst Biersack (France), Finn Kuusisto*, Guilherme Baptista, Hamid Reza Ghasemi, Henry Abbey, Hrishikesh Amur, Huanchen Zhang*, Hugo Diaz, Jake Gillberg, James Perry (U.Michigan-Dearborn)*, Jan Reineke (Universit¨at des Saarlandes), Jay Lim, Jerod Weinman (Grinnell), Joel Sommers (Colgate), Jonathan Perry (MIT), Jun He, Karl Wallinger, Kartik Singhal, Kaushik Kannan, Kevin Liu*, Lei Tian (U.Nebraska-Lincoln), Leslie Schultz, LihaoWang, Martha-Ferris,Masashi Kishikawa (Sony), Matt Reichoff, Matty Williams, Meng Huang, Mike Griepentrog, Ming Chen (Stonybrook), Mohammed Alali (Delaware),Murugan Kandaswamy, Natasha Eilbert, Nathan Dipiazza, Nathan Sullivan, Neeraj Badlani (N.C. State), Nelson Gomez, NghiaHuynh (Texas), Patricio Jara, Radford Smith, RiccardoMutschlechner, Ripudaman Singh, Ross Aiken, Ruslan Kiselev, Ryland Herrick, Samer AlKiswany, SandeepUmmadi (Minnesota), Satish Chebrolu (NetApp), Satyanarayana Shanmugam*, Seth Pollen, Sharad Punuganti, Shreevatsa R., Sivaraman Sivaraman*, Srinivasan Thirunarayanan*, Suriyhaprakhas BalaramSankari, SyJinCheah, Thomas Griebel, Tongxin Zheng, Tony Adkins, Torin Rudeen (Princeton), Tuo Wang, Varun Vats, Xiang Peng, Xu Di, Yue Zhuo (Texas A&M), Yufui Ren, Zef RosnBrick, Zuyu Zhang. 特别感谢以上那些标注了星号的人，他们不仅仅提供建议帮助提高本书。

特别感谢 Joe Meehean(Lynchburg) 教授和他的每章详细笔记，Jerod Weinman(Grinnell) 教授和他的全班的超赞的小册子，Chien-Chung Shen(Delaware) 和他的宝贵详细的阅读和注释。所有的三人都极大的帮助了作者们改善本书的材料。

此外，还要感谢这些年来上 537 课的上百位学生。特别要感谢那些鼓励本书从笔记成型为书本的 08 秋季班学生（他们特别讨厌没有一本教科书来读–执意的学生们！）他们还给我们足够的赞扬让我们保持前进（其中在今年的课上还有一个滑稽的评论"哎呀我的天啊，你应该写一本教科书啊！"）。

我们也亏欠了几个上 xv6 实验课（大部分现已纳入 537 主课程里了）的勇士。09 年春季班：Justin Cherniak, Patrick Deline, Matt Czech, Tony Gregerson, Michael Griepentrog, Tyler Harter, Ryan Kroiss, Eric Radzikowski, Wesley Reardan, Rajiv Vaidyanathan, 和 Christopher Waclawik。09 年秋季班：Nick Bearson, Aaron Brown, Alex Bird, David Capel, Keith Gould, Tom Grim, Jeffrey Hugo, Brandon Johnson, John Kjell, Boyan Li, James Loethen,Will McCardell, Ryan Szaroletta, Simon Tso, and Ben Yule. 10 年春季班：Patrick Blesi, Aidan Dennis-Oehling, Paras Doshi, Jake Friedman, Benjamin Frisch, Evan Hanson, Pikkili Hemanth, Michael Jeung, Alex Langenfeld, Scott Rick, Mike Treffert, Garret Staus, Brennan Wall, Hans Werner, Soo-Young Yang, 和 Carlos Griffin (almost).

虽然我们的研究生们没有直接帮助本书，但他们教了我们很多关于我们已知的系统。他们在威斯康星州（Wisconsin）时我们定期与他们交谈，但他们确实是在做实际的工作，而且通过告诉我们他们在做什么让我们每周都能学到些新东西。这个列表包括共同发表过文章的当前的和以前的学生；星号标记的是那些在我们的引导下获得了博士学位的人：Abhishek Rajimwale, Ao Ma, Brian Forney, Chris Dragga, Deepak Ramamurthi, Florentina Popovici*, Haryadi S. Gunawi*, James Nugent, John Bent*, Lanyue Lu, Lakshmi Bairavasundaram*, Laxman Visampalli, Leo Arulraj, Meenali Rungta,Muthian Sivathanu*, Nathan Burnett*, Nitin Agrawal*, Sriram Subramanian*, Stephen Todd Jones*, Suli Yang, Swaminathan Sundararaman*, Swetha Krishnan, Thanh Do, Thanumalayan S. Pillai, Timothy Denehy*, Tyler Harter, Venkat Venkataramani, Vijay Chidambaram, Vijayan Prabhakaran*, Yiying Zhang*, Yupu Zhang*, Zev Weiss.

最后还要感激 Aaron Brown，它多年前（09 春节班开始）第一次开这个课，然后开 xv6 实验课（09 秋季班），还要感激为本课做了差不多两年（10 秋季班到 12 春季班）的助教。他的勤奋工作对工程进展（特别是 xv6 的那一块）很有帮助，也让威斯康星州（Wisconsin）数不清的本科和研究生们获得了更好的学习体验。就像 Aaron 会说的（以他一贯的简洁风格）:“谢了”。

最后一句

叶芝 (William Butler Yeats) 的名句说“教育并不是填桶，而是点亮一把火。”他是对的，同时也是错的[3]。你其实还是要做一些“填桶”工作的，而这些笔记在这里无疑会对你的学业有帮助的；毕竟，当到 Google 面试，他们问你一些有关信号量的刁钻问题时，你最好还是要知道什么是信号量，对吧？

但是叶芝的主要观点也是明确的：教育的真实意思是让你对某事物产生兴趣，自动的去学习有关主题的更多知识，而不仅仅是消化已学的并在课上得个好分数。就像我们其中的一位父亲 (Remzi 的父亲, Vedat Arpaci) 说过 :“学习要超越课堂”。

我们写了这些笔记来激发你对操作系统的兴趣，进一步在这个主题下阅读，和你的教授谈这个领域中进行的令人兴奋的研究，甚至是亲自参与到研究中去。这是一个很棒的领域，到处都是对计算历史有着重要而深远的影响，而且令人兴奋又精彩的想法。虽然我们知道这把火不会点燃你们全部，但我们希望他能点燃你们中的大部分，即使是一些也很值得。因为一旦这把火亮了，那就是你真正能够做一些伟大事情的时候了。因此教育过程的真正要点就是：前进，去学习很多新的迷人的主题，学习，成熟，然后最重要的，找到那把能够点燃你的火。

Andrea and Remzi
Married couple
Professors of Computer Science at the University of Wisconsin
Chief Lighters of Fires, hopefully.[4]

[3]假如他确实说过这话；就像很多名言一样，这个名言的历史也是模糊的。

[4]如果这听起来像是我们接受了过去一些纵火犯们的历史的话，那你可能会错意了。也许。如果这听起来很勉强，那是因为他就是很勉强，但你要原谅我们。

引用文献

[1] Matthew Sands Richard P. Feynman Robert B. Leighton. *Six Easy Pieces*. Perseus Distribution, Mar. 11, 2011. 176 Seiten. ISBN: 0465025277. URL: http://www.ebook.de/de/product/13622985/richard_p_feynman_robert_b_leighton_matthew_sands_six_easy_pieces.html.

Contents

List of Figures

List of Tables

Chapter 1

关于本书的对话

教授：欢迎来到本书！这本书名叫**操作系统：Three Easy Pieces**, 我在这是来教你有关操作系统的东西的。我叫"教授"，你是？

学生：你好教授！你也许猜到了，我是学生。我已经准备好学习了！

教授：很好。有任何问题吗？

学生：当然！为什么叫做"Three Easy Pieces"?

教授：这个简单。你看这有费曼的一份极棒的物理学讲义...

学生：哦！那个写了"别闹了，费曼先生"的人，对吧？确实是好书。那这本书也会想那本一样幽默吗？

教授：恩...不会。那本书是很棒，我很高兴你读过它。希望这本书更像他的物理学笔记。其中的一些基础总结起来了，称为"Six Easy Pieces"。他谈论的是物理学；而我们会讲操作系统相关的"Three Easy Pieces"。这就好像说操作系统大概有物理学一半那么难。

学生：我喜欢物理学，所以应该挺好的。有哪些小块呢？

教授：他们是我们将要学的三个关键思想：虚拟化，并发性和持久化。在学习这些思想的过程中，我们会学到所有关于操作系统如何工作的知识，包括如何决定下一个在 CPU 上运行的程序，在虚拟内存系统中如何处理内存过载，虚拟机监视器是怎么工作的，如果管理磁盘信息，甚至还有一些怎么构建一个当部分失效

时仍然能工作的分布式系统。就是像这样的东西。

学生：我完全不知道你在说些什么，真的。

教授：很好，这说明你上对课了。

学生：我还有一个问题：学这些东西的最好方法是什么？

教授：好问题！每个人都应该自己想出这个问题的答案。但这是我会做的：上一门课，听听教授对这些材料的介绍。然后，不如说每周末吧，阅读这些笔记，让自己对这些思想有更深刻的印象。当然，过一段时间后（提示：在考试前！），再次阅读这些笔记巩固你的知识。毫无疑问，你的教授也会布置一些家庭作业和工程，你应该完成它们；在写实际代码解决实际问题的做工程过程是将笔记中的思想付诸实践的最佳方式。就像孔子说过…

学生：啊我知道！'我听我往，我看我记，我做我理解'或像这样的什么吧。

教授：（惊奇）你怎么知道我要说什么？

学生：那看起来很搭。另外，我是孔子的粉丝。

教授：我想我们这一路上会处的很好的。

学生：教授，还有一个问题。这些对话有什么用呢？我的意思是，这不是一本书吗？为什么吧直接放上写材料呢？

教授：恩，好问题，真是好问题！我想适当的把你自己拉出书本叙述有时是有好处的；这些对话就是那适当的时候。所以你和我将会一起把所有这些相当复制的思想搞明白的。你赞同吗？

学生：所以我们必须要思考？恩，我同意。我的意思是，要不然我还要做什么呢？在本书之外我好像又没什么其他事。

教授：可惜我也是。让我们开始吧！

Chapter 2

操作系统介绍

如果你在上本科操作系统的课程，那你应该具有计算机程序运行是做什么的基本概念了。如果不是，这本书（和相应的课程）对你就会相当难。—所以你也许应该停止阅读本书了，或者是在继续之前跑到附近的书店补充必要的背景知识（Patt/Patel(PP03) 和 Bryant/O'Hallaron(BOH10) 都是很棒的书）。

所以，当程序运行是究竟发生了什么？

一个运行的程序会做一件相当简单的事情：它执行指令。每秒执行几百万条（当今，甚至是十几亿条）指令，处理器从内存中**取指**，**解码** 指令（例如：搞清这是条什么指令），然后 **执行** 指令（例如：做被设定好要做的事，比如把两个数加到一起，访问内存，检查条件，跳到一个函数出，等等）。在执行完这条指令，处理器就接着执行下一条指令，然后如此这般，直到整个程序最终运行完毕[1]

在这里，我们刚刚描述了基本的 **冯诺依曼** 计算模型[2] 听起来挺简单的，是吧？但是在这门课上，我们会学习到当一个程序运行时，为了达到让系统 **易于使用** 的基本目标，有许多不平凡的事情在进行着。

事实上，有一个软件主体负责让运行程序更简单（甚至允许你在同一时间无

[1]当然，现代处理器在这面罩下还做着相当多的奇怪骇人的事情来使得程序运行的更快，例如，一次运行多条指令，甚至不按序发出和完成这些指令！但这不是我们在这里关心的重点；我们只需关心大多数程序所假设的简单模型：指令在表面上有序线性地一条接一条的执行，

[2]冯诺依曼是早期计算机系统的先驱。他也在博弈论和原子弹方面做了先驱工作，在 NBA 还打了 6 年的球。好吧，其中的一些事不是真的。

缝的运行多个程序），允许程序共享内存，使程序能和设备互动，以及其他的一些像这样的有趣事情。这个程序主体被称作 **操作系统**（或简写为 **OS**）[3]，因为它确保系统以易于使用的方式正确有效的运行。

症结：如何虚拟化资源

一个我们要在本书回答的中心问题很简单：操作系统是怎么虚拟化资源的？这是我们问题的症结。操作系统为什么要这么做（虚拟化资源）不是主要问题。因为答案是显然的：这会使系统更易于使用。因此，我们将注意力集中在怎么做上面：为了实现虚拟化，操作系统实现了什么机制和策略？操作系统是怎么做的这么高效的？需要什么硬件支持？

我们将会在想这样的阴影盒子里面写“问题的症结”，作为一种突出在解决构建操作系统时遇到的问题的方式。因此，在一个特定的主题下的笔记里，你可能会发现一个或多个这样用来突出问题的症结盒子。当然，章节里面的详细内容将会给出解决方法，至少是给出解决方法的规范。

操作系统做到这个的基本方法是采用一种叫做**虚拟化** 的通用技术。换句话说，操作系统接手**物理** 资源（像处理器，内存，磁盘）然后将它们转换为一种更一般、更强大更易于使用的**虚拟** 资源形式。正因为这个，我们有时也把操作系统叫做**虚拟机** 。

当然，为了让用户能告诉操作系统做什么，让用户能使用虚拟机提供的功能（如运行一个程序，分配一些内存，访问一个文件），操作系统必须提供一些能让你调用的接口 (APIs)。事实上，一个典型的操作系统公开几百个**系统调用** 供应用程序使用。因为操作系统提供了这些运行程序、访问内存和设备以及相关的行为的调用接口，我们有时也说操作系统提供了一个**标准库** 给应用程序。

最后，因为虚拟化允许多个程序运行（因此共享 CPU），多个程序并行访问他们的结构和数据（因此共享内存），多个程序访问设备（因此共享磁盘等），操作系统有时也叫作 **资源管理器** 。所有这些 CPU，内存，磁盘都是系统的 **资源** ；操作系统的角色就是去**管理** 这些资源，而且尽可能做的高效，公平或满足许多其他的可能目标。为了更进一步了解操作系统的角色，让我们看看具体的例子。

[3]操作系统的一个早期名字是**监管器** (supervisor) 或者甚至叫做 **主控制程序** 。

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include "common.h"

int
main(int argc, char * argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char * str = argv[1];
    while (1) {
        Spin(1);
        printf("%s\n", str);
    }
    return 0;
}
```

Figure 2.1: 示例：循环并打印的一段代码

2.1 虚拟化 CPU

图2.1呈现了我们第一个程序。这个程序没做很多事情。事实上，它就是调用了 Spin()，一个不停查看时间，并在满了 1 秒钟后返回。然后打印出一个用户通过命令行传入的字符串，如此这样重复再重复。

假设我们把这个文件保存为 cpu.c 然后在单 CPU 的系统上编译运行它，那我们就可以看到：

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
A
^C
prompt>
```

并不是一次很有趣的运行 — 系统开始运行程序，这个程序不停的检查时间，直到时间流逝了一秒钟。一旦时间过去了一秒，程序吧用户传入的字符串（这里是字母“A”）打印出来，然后继续。这个程序将会不停的运行下去，直到按下“Ctrl-C”（这个快捷键是 UNIX 类系统终止前台运行程序用的）我们才能终止这个程序。[1]

引用文献

[1] Matthew Sands Richard P. Feynman Robert B. Leighton. *Six Easy Pieces*. Perseus Distribution, Mar. 11, 2011. 176 Seiten. ISBN: 0465025277. URL: http://www.ebook.de/de/product/13622985/richard_p_feynman_robert_b_leighton_matthew_sands_six_easy_pieces.html.

索引

症结索引