多处理机操作系统

PASCAL

PASCAL

PASCAL

〔印〕M. 约瑟夫　V. R. 普拉萨德　N. 纳塔拉贾 著

科学出版社

多处理机操作系统

〔印〕M. 约瑟夫　V. R. 普拉萨德　N. 纳塔拉贾 著

承　安　焦　达　阎　铭 译

刘顾英 校

科学出版社

内 容 简 介

本书主要阐述了如何运用并发 Pascal 语言设计多处理机操作系统以及设计中所必须解决的问题. 全书共分十三章,内容包括: 绪论、CCNPascal 语言、多处理机的体系结构、操作系统概论、基本存储器分配程序、主存储器分配程序、磁盘空间分配程序、文件系统、输入/输出处理、作业管理、内核、回顾与总结.

本书的特色是将一个多处理机操作系统的复杂问题化简成若干个程序模块的设计和连接,为读者提供一种简单易行而又实用的多处理机操作系统的设计方法.

本书可作为大专院校有关专业的教学参考书,亦可供研究和设计多处理机系统的科技人员参阅.

Mathai Joseph V. R. Prasad N. Natarajan
A MULTIPROCESSOR OPERATING SYSTEM
Prentice-Hall International, Inc., 1984

多处理机操作系统
〔印〕M. 约瑟夫 V. R. 普拉萨德 N. 纳塔拉贾 著
承 安 焦 达 阎 铭 译
刘顾英 校
责任编辑 孙月湘 李淑兰
科学出版社出版
北京东黄城根北街 16 号
邮政编码: 100707
中国科学院印刷厂印刷
新华书店北京发行所发行 各地新华书店经售
*
1991 年 5 月第 一 版 开本: 787×1092 1/16
1991 年 5 月第一次印刷 印张: 23 3/4
印数: 0001—2 200 字数: 545 000
ISBN 7-03-002033-2/TP · 151
定价: 19.50元

序

出于三个原因，我对 Prentice-Hall 国际公司作为计算机科学丛书之一而出版本书表示欢迎.

首先，本书为实现该丛书的目标作出了积极贡献，它把主要的计算机系统程序的设计和实现作为一个专门的研究课题，通过研读精心编写的代码就可以完全理解整个程序.

其次，本书使人们更广泛地了解孟买 Tata 基础研究所在计算机科学方面所进行的重要而实际的研究工作.

最后，本书引起了我个人在操作系统和多处理机网络程序设计方面的浓厚兴趣.

若是所有从事系统程序设计的人都认真阅读这类专著，就不难避免 60 年代以来一直困扰我们的一连串技术故障问题.

由于我个人也负责解决过这些早期的技术故障问题，所以出版一本能有效地解决技术问题的专著使我感到特别满意.

C. A. R. 霍尔

前　言

讲授操作系统与讲授其他课程一样，通常是先介绍各种原理和技术，随后辅之以练习。练习中要求学生编写操作系统的各个部分，以说明所学技术的用途。但由于现实因素迫使多数操作系统课程无法引导学生编写一个真正的操作系统，因而只有少数精力充沛的学生才对难以看懂的商用操作系统资料感兴趣。这就造成了一种普遍看法，认为课堂上讲授的原理和技术不能付诸实践，而许多实际问题既没有在课程中讲过，又不能用所讲的技术来解决。

本书旨在介绍"实际"操作系统的设计，使读者能够了解在编写操作系统时必须解决的实际问题。本书通过分段编写各个程序来描述操作系统主要组成部分的设计，最后把各部分程序综合成一个操作系统。因此，本书的大部分文本(其中有些附加代码与具体系统有关)用扩充型 Pascal 语言编写，这些程序实际上可组成一个能工作的操作系统。这里介绍的内容说明：在操作系统领域里学习如何编制各种新程序(或类似程序)时，研究结构精良的程序是多么重要。

本书前三章完全是基础知识。第一章是绪论。第二章介绍编写本书中所有程序所用的 CCNPascal 语言。第三章扼要介绍多处理机体系结构。第四章开始讨论操作系统的设计，提出了一些简单的要求，概述了一些编写操作系统时必须构成的成分。第一个要介绍的操作系统成分是基本存储器分配程序，其设计在第五章中予以介绍。第六章和第七章分别讨论了该分配程序在编制主存储器分配程序和磁盘空间分配程序中的应用。第八章和第九章描述整个文件系统。第八章从用户观点出发比较简单地讨论了文件系统；而第九章则阐述文件系统结构的细节。第十章讨论使用实际设备(与虚拟文件"设备"不同)的输入和输出处理。第十一章讨论把这些操作系统成分综合起来的过程，并将其用于作业管理；本章还涉及到调度和存储器管理，这一章是操作系统设计的要点所在。第十二章阐述构成操作系统基础的内核，介绍实际硬件的抽象描述，以及实现 CCNPascal 程序时所需的支持。第十三章综述本书所用的各种技术，以及操作系统的设计。

本书虽不是一本专门教程，但已在各类讲习班上用作教材。读者要理解本书的内容必须具备一些基础知识，比如熟悉简单的操作系统，通晓程序设计的抽象概念：监督程序、类程和进程等的使用。掌握了这些基础知识，就可以几种方式使用本书：

1. 作为某门课程的辅助教材时，书中的某些部分可用作研究实例。

2. 第一至八章、第十和第十一章的第一部分可作为一学期的操作系统中级课程。

3. 用作两学期的操作系统课程。

书中安排了多组练习，有的用来修改某章中给出的程序，有的用来编写能满足不同要求的程序。还有一些用作小组练习，适合一起学习的一组学生使用。

然而，本书的内容并不都是容易讲授的。如第九章中的文件系统结构，第十一章第二部分关于相当复杂的作业管理，以及第十二章中描述内核的大部分内容，无疑都是有一定难度的。但考虑到实际工作中专业程序员的需要，我们还是加进了这部分内容。尤其是

考虑到有些学生潜心于商业操作系统文件的毅力和耐心，更觉得这部分内容对这些学生会有用处.

尽管本书有一定篇幅，但还有很多内容无法介绍，如文件系统的相容性和事故恢复，交换策略和交换程序，系统参数的操作员控制，虚拟存储器的使用，以及其他与操作系统性能和使用有关的许多问题，在本书中只作了扼要的介绍. 这些问题本身都相当重要，但总得有一条界线，确定哪些该写，哪些该略. 我们是根据本书的篇幅和单元来安排内容的，凡需要大量增加篇幅的内容和突破单元的细节都予以略去. 本书的宗旨就是鼓励读者能够为操作系统的这些部分编制结构优良的程序.

历史背景

本书讨论的操作系统以一个为实际多处理机系统编制的操作系统为蓝本（该专题的书目在参考文献中给出）. 由于研制操作系统要涉及一些重要课题，所以下面扼要介绍一下历史背景.

1975 年初的几个星期里，印度国家软件开发和计算技术中心在联合国开发计划的资助下，组织了一次专题学术讨论会，我们有幸参加了这次讨论会. 会上除了许多其他内容外，还介绍了两项新的重要研究成果：P. Brinch Hansen 作了一系列专题讲座，介绍并发 Pascal 语言；W. A. Wulf 介绍卡内基-梅隆大学研制的 C. mmp 多处理机系统及其 Hydra 操作系统的设计. 当时，我们中有几位已经谈到有可能利用印度研制的 TDC 316 计算机构成多处理机系统，而这次讨论会恰好成为一种动力，促成把这些想法变成具体的计划，并在几个月后提交给印度政府的电子委员会，后经有关部门批准. 到了年底，第一台 TDC316 机就交付使用了. 其间，我们分析了如何修改并发 Pascal 语言，以满足我们的要求，同时研究了几种简单的互连方案，以便将几台 TDC 316 计算机连成所谓的紧耦合网络（CCN).

到 1976 年底，我们已研制成一个实验性编译程序. 该程序用 DEC System 10 上的 Pascal 编写，以 Pascal 编译程序作为模型构成我们的并发 Pascal 版本. 这时，我们已经完成了对原始语言的修改，因此最好另取一个合适的名字，故得到 CCNPascal 这个名字. 另外三台 TDC 316 计算机也用与实验机相同的方式连在一起. 由于众所周知的实验性设计特点，无论是编译程序还是系统都没有长期运行的可靠性. 因而 1977 年的大部分时间对其进行改进，以便能开始进行其他实验(如操作系统部分的实现). 对操作系统的各种设计方案进行了比较、筛选，两年以后终于完成了一个较为简单的操作系统，并执行了第一个“用户”程序.

回想起来，我们把一种全新而且没有经过测试的计算机用于这样一种系统真是冒了不少风险. 特别值得一提的是，只有一个仅有四五个人的小组在设计 CCNPascal 语言的新功能，研制其第一个编译程序，且用这种语言编制操作系统及其内核，每次都要编写好几个版本. 尽管我们没有完全解决样机系统硬件的可靠性问题，但大量的程序开发工作没有遇到不可克服的困难. 使用一种优秀的编程语言和系统设计技术具有无法估量的价值，这个经验永远值得我们牢记.

致谢

CCN 计划得到印度政府电子委员会的资助，我们的工作在很大程度上应归功于这一资助。本书作者所在的国家软件开发和计算技术中心（NCSDCT）为 CCN 计划及本书的写作提供了优惠支持。我们要感谢该中心的主任 R. Narasimhan 和我们的许多（过去和现在的）同事对我们的工作所给予的帮助。印度电子有限公司（ECIL）在制造连接该公司的 TDC316 计算机的特殊硬件和帮助我们维修硬件方面做了大量的份外工作。

许多 NCSDCT 的成员以各种方式多次参与了 CCN 软件的设计和实现：R. Visewanathan 和 K. V. S. Prasad 参与了内核的早期设计；Satish Thatta 设计了文件系统的第一个版本；Sandhyn Desai 编写了连接操作系统各个模块的系统装入程序；K. T. Narayane 对 CCNPascal 编译程序的第一个版本做了大量工作。在形成内核和操作系统的最终形式及其实现的过程中，许多工作应归功于 Mukul Sinha 和 I.V.Ramakrishnam。前者还参加了硬件的系统设计。在印度电子有限公司，A. K. Kaul 和 P. V. S. Nayak 参加了专用硬件的设计。在开发过程中，M. V. Wilkes 每年来视察一次，对设计提出了宝贵意见；W.A.Wulf 也是这样，他花了不少时间和精力研究我们的设计，并为 C. mmp 计划的进度提出了许多具体意见。

C. A. R. 霍尔建议我们写一本关于操作系统的书。他不仅亲自编辑这套丛书，首创了我们书中用到的许多技术，而且长期从事程序的出版工作。我们之所以采纳他的建议，在很大程度上是因为他劝告我们作出这一努力是值得的。我们还要深切感谢 Prentice-Hall 国际公司的 H. Hirschberg 和 R. Decent，以及 R. M. Mckeag 和 R. Gimson，他们极其耐心和认真地审阅了本书的初稿。

好几位读者对我们的手稿提出了意见，他们是 H. N. Mahabala, V. K. Joglekar, R. Chandrasekhar 和 K. Lodaya 等。对他们以及与本书有关的读者和其他人员，在此一并致谢。

本书文稿的打字、编辑和编排是在印度国家软件开发和计算技术中心的 DEC System“10”上用标准文本编辑程序和 NCSDCT 文本排写系统 DIP 完成的。

M. 约瑟夫　V.R. 普拉萨德　N. 纳塔拉贾

1982 年 11 月于 NCSDCT

目　　录

第一章 绪 论

一旦好运气或某个慈善的基金会给我们提供了一个新的计算机系统，我们首先需要的是一本操作系统手册．迅速学习一下这本手册之后，我们就能坐到一台分时终端旁“试试各种操作”．尔后，我们中有些人如果不怕复杂的命令格式、作业控制语言和费解的错误信息，就会再去仔细阅读该手册，以彻底搞清系统的使用方法．这一过程将一直进行到每个用户获得使用该系统解决实际问题所需要的信息为止．如出现新的问题，则再去阅读手册或请教当地的专家，看看能否找到一个简单的解决办法；或者回忆一下手册中的有关内容，查阅能告诉我们如何解决这个问题的章节．

我们在基本了解了系统之后，就想知道操作系统是如何完成其功能的(操作系统的最新教程教会我们许多有用的原理，但并未告诉我们操作系统的工作方式)．仅与专家一起进行讨论是远远不够的，除了阅读“系统人员”产生的文件之外别无他法．我们的问题就是从这里开始的．书架上摆满了内部文件手册，每本手册似乎都指望我们了解其他手册中的信息．有许多详尽的流程图，说明如何将二进位数字组成表项的方块图，以及旨在修改手册内容的改正文件，从而使情况变得更加复杂．系统程序员在学完制造厂商提供的培训课程后，就可以对这些文件运用自如．不过，像其他专业人员一样，他们用“你到底要知道些什么?”这样的话来回答希望全面了解情况的人．这个问题似乎很合理，但是实际情况是，在我们能确切地知道从详细文件中寻找什么内容之前必须了解系统的一般结构!

这种情况许多人都很熟悉，只是最后的结局不同而已．有的人认为操作系统的理论和实践之间的鸿沟太大，使许多人对操作系统不太感兴趣，这一点不足为怪．于是，有些人认为理论足够好，就是不切合实际，真正实用的操作系统提供的内容要丰富得多．也有些人走向另一个极端，心安理得地等待理论付诸实践．最后，还有少数果敢的人在一大堆文件中寻找操作系统的结构，然后对照现有的理论加以检查．

对这种情况或其结局来说，并没有具体的是非标准．但是本书旨在说明，同时了解一个具有相当规模的操作系统的结构和细节是可能的．为此，我们要做一个编制多处理机操作系统的练习．我们将系统地编制该操作系统的各程序成分，最后将这些成分组合成一个操作系统．

1.1 操作系统的编写

操作系统的设计和其他大型程序的设计一样，有技术规范和正确性、测试和性能评价，以及文件编制等问题．本书将逐一讨论这些问题，因为要编制一个正确、高效和实用的程序，这些问题都应解决．但是由于我们着重探讨的是操作系统的设计问题，因此还要讨论并行性(即许多活动同时发生的)问题，而且我们也无法求助于其他一些程序来控制计算机系统及其外围设备，或处理执行过程中出现的错误．这两个因素在决定操作系统的

设计方式时起着重要的作用.

出于各种理由,设计一个新的操作系统,与研究或修改现有的操作系统一样,是一项令人兴奋的练习. 最简单的一个理由(尽管很可能把人引入歧途)就是操作系统是一种执行时"靠近硬件"的程序. (当人们说"软件指挥硬件"时,他们经常是想到了操作系统!)了解一个高效的操作系统怎样按步就班地从卡片输入机或终端上接受输入信息,并实际完成不同任务间的和谐协调的各个动作,是令人振奋的事. 不过,我们还得深入研究操作系统设计中更基本的一些问题,正是这些问题才使操作系统值得人们进行研究.

操作系统不仅是一个大型程序,而且是一个必须协调不同的并行任务的操作,使程序能在所有条件下正确执行的程序. 同时,操作系统必须提供各种设施,以方便使用,提高执行效率和可靠性. 其基本结构必须精心设计,以使操作系统随使用情况的发展而发展时,能扩充新的功能来满足新的需要. 计算机系统的每个认真的用户很快就会认识到其操作系统可能具备的功能和受到的限制;但对操作系统设计者来说,这些功能和限制是程序开发过程中各种选择的结果.

人们早就了解了操作系统的许多基本性质("早"是相对于计算机科学这门新兴学科而言的). 人们已对设计中出现的各种独特的问题加以分离和研究,找出了各种解决办法,而且许多现代操作系统表明解决研究问题的各项方案具有实用价值. 但是,对于打算设计一个先进的操作系统的程序员来说,如果遇到的实际细节过多,以至于很难确定问题的症结,则上述方法的实际作用就十分有限了.

我们实际上无法告诉程序员怎样才能不考虑具体要求和所使用的计算机系统而设计出一个实用的操作系统. 换句话说,在设计一个具体的操作系统时,有许多选择是因具体问题而异的. 本书旨在为设计一类相当有用的操作系统提供一个范例. 我们将根据一组实际的要求和实际的硬件来设计一个完整的操作系统. 我们也象其他程序员一样,需要作出各种选择,并证明这些选择是合理的. 还可能出现资源限制之类的问题,而且要考虑到硬件可能不够完善,即可能出现故障,必须安全地进行处理. 我们在每个阶段都将产生操作系统的一个部分的程序文本,并且根据这一部分的一组特点来设计其他部分. 如果把所有这些程序文本集中在一起,就可构成一个能在计算机上执行的完整的操作系统.

程序员在设计操作系统时要深入到什么程度呢? 如果技术要求和硬件与本书介绍的类似,那么这项任务就完成得差不多了,因为本书中设计的操作系统是可以实施的. 但是我们还是希望聪敏的程序员能进一步改进该项设计. 对于大多数程序员来说,无论是想构成实际的操作系统,还是要学习操作系统的设计原理,本书中的练习都应被看作为一个把设计要求变为最终的程序文本的例子. 有的程序员可能会发现该操作系统中的有些程序成分可以直接利用,有些则需要进行修改,还有些则需要重写. 有的程序员可能会发现基本结构需要根据具体要求加以修改. 还有的程序员只想学习本书介绍的设计,以便着手设计满足其他要求的完全不同的操作系统. 本书的目的之一就是为这些程序员提供帮助,使操作系统的设计任务能遵循系统的方法进行,以产生预定的结果.

1.2 设 计 问 题

操作系统这种程序的基本功能就是使一组互相竞争的用户程序能有效地共享计算机

系统的资源．这一功能已延续了二十多年．现代操作系统和原始操作系统的真正区别在于：目前的系统资源不仅包括存储器、处理机和外围设备等实际设备，而且包括用户产生的抽象资源，即逻辑资源(例如文件)．现代操作系统之所以复杂，主要是因为各级用户要求共享抽象资源，同时要防止对抽象资源的越权存取和不相容存取．

给用户程序分配资源和从用户程序收回资源的方法，长期以来一直是操作系统研究的最重要课题之一．人们已经开发出几种"资源管理"技术．为具体的操作系统选择合适的资源管理技术取决于计算机系统的种类和在该系统上执行的用户程序的类型．例如，小型计算机系统的操作系统一般较简单，而且规模较小，因为系统的资源很少，而且分配方式较固定．而大型计算机系统的资源常远远超过任何单用户程序所需的资源，因而其操作系统要能有效地在几个用户之间分配资源．使用大型计算机系统的一个重要的理由就是许多用户无法预测的要求能由一个优异的操作系统进行平均，只有当用户需要资源时才分配给它，而把其余的资源分给其他用户．

操作系统的一个目的就是分配资源，并对各用户程序的要求作出"仲裁"．然而，对每个用户程序来说，操作系统的外部特征稍有不同：它是使用系统功能的一种工具．例如，实时操作系统将外部生产过程(如化工厂、造纸厂)产生的信号提供给应用程序，以完成计算；然后，应用程序又送出控制信号，以改变生产过程的运行．分时操作系统允许不同终端上的用户执行独立的程序，并共享程序和数据．即使是袖珍计算器，也可认为它有一个"极其"简单的操作系统，从键盘读入命令和数据，并在显示器上把结果显示出来．因此，操作系统也是系统和用户之间的接口．

操作系统像所有的接口一样，使用户不必考虑操作环境的某些细节．比如，程序员使用分时系统时，可以运行他们的程序而不必考虑系统中处理机的个数，也不必考虑其他终端上的用户在干什么．同样，操作系统也使用户不必考虑系统中的实际存储容量．因而，即使存储器的一部分出了问题，用户程序仍可继续执行(尽管要慢一些)．但是，操作系统也可以让用户使用计算机系统的一些特殊功能．用户不一定非得使用系统任意选定的打印机，也可以规定在具有特殊字符的打印机上打印其输出信息，甚至可以规定在某一台打印机上打印其输出信息．

很明显，用户既不必考虑系统的某些特点，又能明确利用某些功能，这两个目的是难以调和的．尽管大多数用户并不关心其程序在存储器的哪一部分执行，但总是有些用户需要利用系统的具体特点．我们如何才能使所有用户只看到他所关心的细节呢？更一般地说，我们怎样才能使系统的一些特点是"可见"的，而使另一些特点是"隐蔽"的呢？特别是，我们怎样在构成操作系统的大而复杂的程序中做到这一点呢？

这些问题没有简单的解决办法，因为在设计大型程序时可能遇到许多复杂问题，而新的要求又可能在实施过程中出现．所能办到的只是采用系统的设计和实施过程，使得程序的内部结构总能满足外部的要求．一旦有新的要求，就可以清楚地知道需对程序结构作多大修改．

＊ 首先，我们必须确定系统功能：必须精确规定系统的外部功能，以便把它作为系统设计的目标．

＊ 这些外部功能必须由操作系统的各单元来完成，而现在我们要注意的就是这些操作系统单元．对操作系统这一类程序来说，通常有多少个主要的外部功能，就得有多少个

主要的操作系统单元，二者是相对应的。

* 一旦我们获得了操作系统的总体结构，就可能发现该程序的每个单元仍嫌太大，不易编程且难以理解。因而这些单元就必须按主要功能再进行划分，使得这些功能可以用较小的程序单元来实现。

* 该过程可能需要反复多次，才能使单元越划越小直到所划分的单元易于编程，易于理解为止。最后一组单元是无须进一步划分的基本单元，用以构成系统的其余部分。

这种系统地逐步分解程序设计的方法称为“逐步求精”法，所划分成的单元称为“抽象单元”。

通常在构成程序的过程中使用两种抽象单元：设计抽象单元和语言抽象单元，前者是精心安排系统的初始化高级设计的结果，后者是编程用的。当然，在设计及实施过程中能使用一种统一的表示法是大有好处的。

1.3 语　言

不久以前，系统程序（如操作系统和编译程序）在设计时采用流程图那样的高级表达形式，而实施时则采用机器语言或汇编语言。有两条理由说明这样做是不令人满意的。首先，在设计的描述级和实现的操作之间存在着鸿沟。设计和汇编语言程序之间难以建立对应关系，而必然要进行的修改和扩充设计将对此产生很大影响。如果弄不清一段汇编语言程序代码所执行的是流程图中的哪个功能，那就更谈不上修改这段程序代码了，也难以保证它与设计描述相吻合。这个问题是不能通过提供更详细的描述来解决的，因为对于大型程序来说，这样的描述异常复杂，也得不到令人满意的结果。

其次，解决操作系统的许多程序设计问题要求始终使用抽象过程，使程序的执行总能遵循一定的规则。在汇编语言程序里，这种抽象只能产生不能和其他程序段相区别的一些代码段。可以认为这些程序段很难令人满意，因为它们必须仔细地进行编写，而修改则要冒很大风险（见过用汇编语言写的操作系统代码文本的人可能见过诸如这样一些注释：“以下十行为保证一致性所需，未经系统程序员同意切勿修改”。当最初的系统程序员转到其他任务上去时，必须加上这类注释）。并不是说不能使汇编语言程序遵守某些规则，程序是人编的，人的智慧和勤奋足以做到让程序遵守必要的规则。但是，把精力都浪费在这样的细节上，容易忽视那些更本质、更重要的设计问题。

高级程序设计语言的目的是将程序设计所需要的抽象单元变作语言的特性。例如，字符串处理语言 SNOBOL 把文本串作为完成所定义的操作的基本对象。同样，必须能识别出操作系统中特定的并行任务编程问题所需要的基本结构。这些基本结构确定之后，便可让编译程序检查（至少是在较大程度上）操作用得是否正确，并让编译程序编写者检查操作执行得是否正确。这些问题只需解决一次，语言一旦确定，每个编译程序就有了程序编译时所需遵循的一组规则。

根据这一原则，我们要求语言设计者提供一组能满足我们的使用要求的机制。一组设计良好的机制要具有多种用途，并应能在形式上证明：在所有条件下都能保证具有某些特性。没有一组机制既能灵活地满足所有用户的要求，又定义得足够好，以简化正确性的形式证明。在语言机制的灵活性和形式特性之间必然存在着某种折衷关系，但界限应