计 算 机 科 学 丛 书

现代操作系统

Modern Operating Systems

Andrew S. Tanenbaum 著
陈向群 等译

机械工业出版社
China Machine Press
Prentice Hall

计算机科学丛书

现代操作系统

Andrew S. Tanenbaum 著

陈向群 等译

机械工业出版社
China Machine Press

本书共分两部分，第一部分详尽讲述了传统操作系统知识，包括进程、存储器管理、文件系统、I/O设备管理、死锁等内容；第二部分主要介绍了分布式操作系统，包括层次协议、远程过程调用、互斥操作、分布式文件系统等专题。

为加深概念的理解，本书还详细介绍了四个操作系统，包括两个传统的系统UNIX和MS-DOS；两个分布式系统Amoeba和Mach。此外还简要介绍了NFS、AFS、ISIS等其他几个系统。

本书体系完整、内容丰富、叙述清晰，是大学计算机及相关专业学生不可多得的教科书，对于从事计算机管理、开发、系统分析等职业的专业人员也是优秀的参考书。

本书版权登记号：图字：01-98-0720

图书在版编目(CIP)数据

现代操作系统/坦尼鲍姆(Tanenbaum, A. S.)著；陈向群等译. - 北京：机械工业出版社，1999.11
(计算机科学丛书)
书名原文：Modern Operating Systems

ISBN 7-111-07117-4

Ⅰ.现… Ⅱ.①坦… ②陈… Ⅲ.操作系统 Ⅳ.TN316

中国版本图书馆CIP数据核字(1999)第44617号

机械工业出版社(北京市西城区百万庄大街22号 邮政编码 100037)
责任编辑：陈 谊
北京第二外国语学院印刷厂印刷 · 新华书店北京发行所发行
1999年11月第1版 · 2000年4月第2次印刷
787mm × 1092mm 1/16 · 32印张
印 数：6 001-9 000册
定 价：40.00元

译 者 序

当今，在知识经济蓬勃发展的浪潮中，软件产业占据着核心地位，而操作系统则是所有软件的基础，是软件的根本。正是由于这个原因，长久以来世界软件业巨头们从不放弃对操作系统的控制争霸。

操作系统是计算机科学与技术专业的一门专业基础课，是大学学生的必修课程。在学习过程中，教材起着相当大的作用。因此，要学好这门课，首先应该选择一本好的教材。

本书的作者Andrew S. Tanenbaum是一位经验丰富的教授，从事计算机教学多年，在操作系统、计算机网络方面颇有研究，出版了多本教材。本书是作者关于操作系统系列教材中的一本，书中既讨论了传统操作系统的原理，又系统介绍了分布式操作系统的方方面面。本书体系完整、结构合理、内容丰富、叙述清晰、适用面广。美国不少主要大学的计算机软件及相关专业都采用本书作为教材或主要教学参考书。

通过本书的学习，相信学生一定会对操作系统的功能、实现技术有全面的了解。

本书在翻译的过程中，得到很多人的支持帮助。参加翻译工作的还有张杰、张乃琳、叶松、李茹、翁念龙、王梦秋、何新明、高卫、胡英琨、陈宇、朱克耀，北京大学计算机科学技术系的部分学生也参加了翻译工作，其中徐颖、高磊、刘西川、潘颖、章苏、董方鹏等做了很多工作。

译 者

前　言

过去，大多数的计算机都是独立工作的，大多数的操作系统也设计为在单处理机上运行。然而，这种状况正在发生迅速的变化：许多计算机都联网工作，因而分布式操作系统也变得越来越重要。作为操作系统课程的教材，本书的特别之处在于认识到了这个变化，并将分布式操作系统提高到与传统的单CPU操作系统同等地位来讲述。

在过去的15年中，我参与设计和实现了三个不同的操作系统：TSS-11(PDP-11)，MINIX(IBM PC、Atari、Amiga、Macintosh和SPARC)，还有Amoeba(80386、Sun-3、SPARC和VAX)。经过这些实践，我认识到应该在教材中强调那些在实际的操作系统中真正有用的东西。本书包含了所有在本科生的操作系统课程中应该包含的内容，包括进程、进程间通信、信号量、管程、消息传递、经典IPC问题、调度、交换技术、虚拟存储技术、分页算法、段式管理、文件系统、安全性、保护机制、I/O硬件与软件、死锁等。不过，本书分配给各专题的篇幅与一般的教材不尽相同，因为我认为学生应该学习那些在实践中真正有用的概念，而不仅仅是寻求理论上的完善。例如，CPU调度只占了一节，而不是整个一章。许多著作中都提出并分析了很多复杂的调度算法，而实际的操作系统中采用的一般都是最简单的优先级法或轮转法。

本书的第二部分主要介绍分布式系统。我们先用一章的篇幅简单介绍了分布式系统的软硬件配置，然后依次介绍各个专题：层次协议、客户机-服务器(Client-Server)模型、远程过程调用、组内通信、时钟同步、互斥操作、选举算法、原子事务、线程、分布式文件系统等等。在一门本科生课程中包含这么多分布式系统的内容可能有些特别，不过计算机行业发展得如此迅速，分布式系统会很快成为操作系统的主流，因此每个学生都应该熟悉它。

为了加深对这些基本概念的理解，本书还详细介绍了四个操作系统，两个传统的，两个分布式的。其中两个传统的，即非分布式的系统是UNIX和MS-DOS；分布式操作系统是Amoeba和Mach。这四个例子每个各占一章，除此之外，还简要介绍了NFS、AFS、ISIS等其他几个系统。学完本书之后，读者会对操作系统有一个较完整的了解。

在此，我想指出，开始时我只想修订我的一本旧作“Operating System:Design and Implementation”，将其中有关MINIX的内容去掉，使之适合传统的“理论”教学，而不是“实验”课程。在修订的过程中，我意识到分布式系统的重要性，因而又加了七章这方面的内容。我还想在近期内修订那本MINIX的书，这样就会有两本更新后的书，一本适用于理论学习，一本适用于实践。

在本书的写作过程中，很多朋友审阅了初稿并提出了宝贵的意见，然而，本书还是存在着一些错误和不妥之处。错误似乎是不可避免的，无论有多少人审阅过书稿。欢迎发现错误的读者通过E-mail与我联系，地址是 ast@CS.VU.NL。另外，对Amoeba系统(见14章)感兴趣的公司或大学也请通过E-mail与我联系。

Andrew S. Tanenbaum (安德鲁．S．坦尼鲍姆)

目　录

第二部分 分布式操作系统

第一部分 传统操作系统

第1章 引 言

离开了软件的计算机就成为一堆废铜烂铁。有了软件，计算机可以对信息进行存储、处理和检索，检查文档拼写错误、玩探险游戏以及处理许多其他有意义的事务。计算机软件大致分为两类：系统软件和应用软件。系统软件用于管理计算机本身及应用程序；应用软件实现用户所需要的功能。操作系统(operating system)是最基本的系统软件，它控制计算机的所有资源并提供应用程序开发的基础。

现代计算机系统包含一个或多个处理器、若干主存(常称为“磁芯存储器”，尽管磁芯存储器早在十多年前就不再使用了)、时钟、终端、磁盘、网络接口及其他输入/输出设备，总而言之是一个非常复杂的系统。为了正确地使用它们，需要编写一个程序来管理所有这些部件，编写这样的程序，即使不考虑优化也是一件很困难的事情。如果每个程序员都必须处理诸如磁盘如何工作，以及每读一个磁盘块时，可能导致操作出错的有几十种因素等硬件控制问题，那么很多程序几乎没法写成。

在许多年以前，人们就认识到必须要找到某种方法，将程序员从复杂的硬件控制中脱离出来。解决的方法是在硬件的基础之上加载一层软件来管理整个系统。并为用户提供一个接口或虚拟机(Virtual machine)，从而更容易理解和进行程序设计。这一层软件就是操作系统，也就是本书所论述的主题。

这一种处理方式如图1-1所示。其底层是硬件，它本身可能由两层或多层构成。最底一层是物理设备，包括集成电路芯片、连线、电源、阴极射线管和相关的物理装置。至于它们是如何构造的和如何工作的则属于电子工程师的范围。

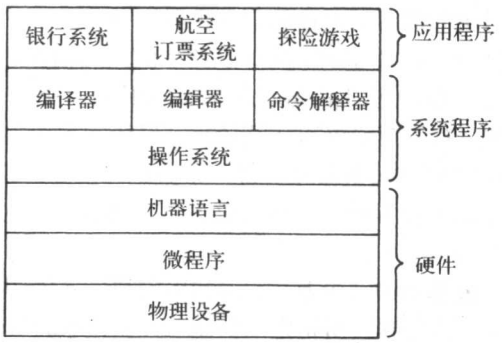

图1-1 计算机系统由硬件、系统程序和应用程序组成

其次是直接控制设备并向上一层提供更清晰的一个接口的很原始的软件。该软件被称为微程序(microprogram)，通常存放在只读存储器中，它实际上是一个解释器，先取得诸如ADD，MOVE和JUMP等机器语言指令，然后通过一序列的细小动作执行这些指令。例如，为了执行一条ADD指令，微程序必须先确定要做加法的数据的位置，取数，然后相加，最后在某处存放结果。由微程序解释执行的这一套指令集称为机器语言(machine language)。机器语言并不真正是硬件的组成部分，但硬件制造商通常在手册中给出有关机器语言的完整描述，所以不少人将它看作真正的“计算机”。在有些机器里，微程序由硬件实现，它并不是明晰的一层。

典型的机器语言有50到300条指令，大多数在机器里从事数据传送、算术运算和数值比较

等操作。在这个层次上，通过向特定的设备寄存器(devive register)写入数值来控制输入/输出设备。例如，硬盘读操作可通过把磁盘地址、内存地址、字节数和操作类型(读/写)等值装入特定的寄存器来完成。实际上，往往需要更多的操作参数，而操作完成后设备返回的状态也非常复杂。进一步说，对许多I/O设备而言，在程序设计中时序的作用非常重要。

操作系统的主要功能之一就是将这些所有的复杂性隐藏起来，并为程序员的工作提供更加方便的一套指令。例如，在概念上“从文件中读数据块”比考虑“移动磁头，等待放置”之类的细节来得简单、方便。

在操作系统的顶层是其他系统软件。其中有命令解释器(shell)、编译器、编辑器及类似的独立于应用的程序。重要的是这些程序本身并不是操作系统的组成部分，尽管通常由计算机厂商提供它们。这一点很重要，也很微妙。操作系统是专指在核心态(kernel mode)，或称管态(supervisor mode)下运行的软件，它受硬件保护而免遭用户的篡改(不考虑某些老式微处理器，它们根本没有硬件的保护)。编译器和编辑器运行在用户态(user mode)。如果某一个编译器用户不喜欢，他尽可以自己重写一个，但是用户却不可以自己换一个磁盘中断处理程序，因为这是操作系统的一部分，通常由硬件保护，以防止用户试图对它进行修改。

最后，在系统软件的上面是应用软件。这些软件是用户编写的，用来解决诸如商业数据处理、工程计算或者电子游戏等特定的问题。

1.1 什么是操作系统

多数计算机用户都有一些使用操作系统的体验，但要给出操作系统的准确定义却很困难。部分原因是操作系统执行两个相对独立的任务，并取决于从什么角度看待，读者多半听说过其中一个或另一个的功能。下面我们逐项进行讨论。

1.1.1 作为扩展机器的操作系统

正如前面所叙述的，在机器语言一级上，多数计算机的体系结构(architecture)(指令集、存储组织、I/O和总线结构)是原始的而且编程是很困难的，尤其是输入输出操作。要更细致地考察这一点，可以简要地考虑如何用NEC PD765控制器芯片来进行软盘I/O操作，这个芯片是IBM PC和多数PC机采用的。(尽管在本书中，我们交叉使用“软盘”与“磁盘”这两个名词)PD765有16条命令，每一条命令向一个设备寄存器装入长度从1 到9 字节的特定数据。这些命令用于读写数据、移动磁头臂、格式化磁道，以及初始化、检测状态、复位、校准控制器及设备等。

最基本的命令是READ和WRITE。它们均需要13个参数，所有这13个参数被封装在9个字节中。这些参数所指定的信息有：欲读取的磁盘块地址、磁道的扇区数、物理介质的记录格式、扇区间隙，以及对已删除数据地址标识的处理方法等。如果不懂这些晦涩的指令也没关系，它本身相当深奥。当操作结束时，控制器芯片在7个字节中返回23个状态及出错字段。这样好象还不够，编写软盘控制的程序员还要保持注意步进电机的开关状态。如果电机关闭则在读写数据前要先启动它(有一段较长的启动延时)。电机不能长时间处于开启状态，否则将损坏软盘。程序员还必须在较长的启动延迟和可能对软盘造成损坏(和丢失数据)之间作出折衷。

很清楚，一般程序员并不想涉足软盘编程的这些具体细节(也包括硬盘，它也非常复杂且与软盘不同)。程序员需要的是一种简单的，高度抽象的可以与之打交道的设备。对磁盘而言，典型的抽象是包含了一组已命名文件的一张磁盘。每个文件可以打开、读写，最后关闭。至于其中的一些细节，如是否应使用修改的频率模块化数据记录格式、当前步进电机的状态等则在

对用户的抽象中隐藏。

将硬件细节与程序员隔离开来，并提供一个可以读写的命名文件的简洁的方式，当然就是操作系统。与操作系统屏蔽了磁盘硬件，提供了一个简明的、面向文件的接口类似，其他许多包括定时器、存储器管理等低层硬件的特性也被隐藏了。在每种情况中，操作系统提供的抽象都较底层硬件本身更简单、更易使用。

从这个角度看，操作系统的作用是为用户提供一台等价的扩展机器(extended machine)，或称虚拟机，它比底层硬件更容易编程。至于操作系统是如何做到这一点的，这正是本书所讨论的内容。

1.1.2 作为资源管理器的操作系统

把操作系统看作是提供给用户的基本的方便接口的概念是一种自顶向下的观点。按照另一种自底向上的观点，操作系统则用来管理一个复杂系统的各个部分。现代计算机包含处理器、存储器、时钟、磁盘、终端、磁带设备、网络接口、激光打印机以及许多其他设备。从这个角度看，操作系统的任务是在相互竞争的程序之间有序地控制对处理器、存储器以及其他I/O接口设备的分配。

假设在一台计算机上运行的三个程序试图同时在同一台打印机上输出计算结果。那么头几行可能是程序1的输出，下几行是程序2的输出，然后又是程序3的输出等等。最终结果将是一团糟。操作系统采用将打印输出送到磁盘上缓冲区的方法，可以避免这种混乱。在一个程序结束后，操作系统可以将暂存在磁盘上的文件送到打印机输出，同时其他程序可以继续产生新的输出结果，很明显，这些程序的输出并没有真正送到打印机。

当计算机有多个用户时，特别是用户还需要共享诸如磁带机和照片输出机等昂贵的资源。就更有必要管理和保护存储器、I/O设备以及其他设备。经济考虑是一方面，还要让用户共享信息。总之，这种观点认为操作系统的主要任务是跟踪谁在使用什么资源、满足资源请求、记录使用状况，以及协调各程序和用户对资源使用请求的冲突。

1.2 操作系统历史

操作系统经历了多年的发展过程，对此我们进行一下简要的回顾。在历史上由于操作系统与其运行的计算机系统结构联系非常密切，我们将按照计算机的换代历程介绍操作系统的发展。

第一台真正的数字计算机是英国数学家Charles Babbage(1792-1871)设计的。尽管Babbage为建造他的“分析机”投入了毕生精力，但却没能让它成功地运行起来，因为它是纯机械式的，当时的技术不可能使分析机的零部件达到需要的精度。当然，这个分析机没有操作系统。

1.2.1 第一代计算机(1945~1955)：真空管和插件板

从Babbage失败之后一直到第二次世界大战，数字计算机的建造几乎没有什么进展。40年代中期，哈佛大学的Howard Aiken、普林斯顿高等研究院的John Neumann(冯·诺依曼)、宾夕法尼亚大学的J.Presper Eckert和William Mauchley、德国的Konrad Zuse、以及其他一些人，都使用真空管成功地建造了运算机器。这些非常巨大的机器，使用了数万个真空管，占据了几个房间，然而其运算速度却比现在最便宜的家用计算机还要慢得多。

在早期，都有一个小组专门设计、制造、编程、操作和维护每台机器。程序设计全部采用

机器语言，通过在一些插板上的硬连线来控制其基本功能。没有程序设计语言(甚至没有汇编语言)，操作系统更是前所未闻。使用机器的方式是程序员提前在墙上机时表上预约一段时间，然后到机房将他的插件板插到计算机里，在接下来的几小时里计算自己的题目，期盼着在这段时间中，二万多个真空管没有烧断的。这时实际上所有的题目都是数值计算问题，例如计算正弦和余弦函数表。

到了50年代早期，有了改进，出现了穿孔卡片，这时就可以将程序写在卡片上然后读入计算机而不用插板，但其他过程则依然如旧。

1.2.2 第二代计算机(1955~1965)：晶体管和批处理系统

50年代晶体管的发明极大地改变了整个状况。计算机已经很可靠，厂商可以成批地生产计算机并卖给用户。用户可以指望计算机长时间运行，完成一些有用的工作。此时，设计人员、生产人员、操作人员、程序人员和维护人员之间第一次有了明确的分工。

这些计算机安装在专门的空调房间里，由专人操作。只有少数大公司、主要的政府部门或大学才承受得了数百万美元的标价。要运行一个作业(job，即一个或一组程序)，程序员首先将程序写在纸上(用FORTRAN语言或汇编语言)，然后穿孔成卡片。再将卡片盒带到输入室，交给操作员。

计算机运行完当前任务后，其计算结果从打印机上输出，操作员到打印机上撕下运算结果并送到输出室，稍后程序员就可取到结果。然后，操作员从已送到输入室的卡片盒中读入另一个任务。如果需要FORTRAN编译器，操作员还要从文件柜把它取来读入计算机。许多机时被操作员在机房里走来走去而浪费掉了。

由于当时计算机非常昂贵，人们很自然地想办法减少机时的浪费。通常采用的方案就是批处理系统(batch system)。其思想是：在输入室收集全部的作业，然后用一台相对便宜的计算机，将它们读到磁带上，如IBM 1401计算机，它适用于读卡片、拷贝磁带和输出打印，但不适用于作数值运算。另外用较昂贵的计算机，如IBM 7094来完成真正的计算。情况如图1-2所示。

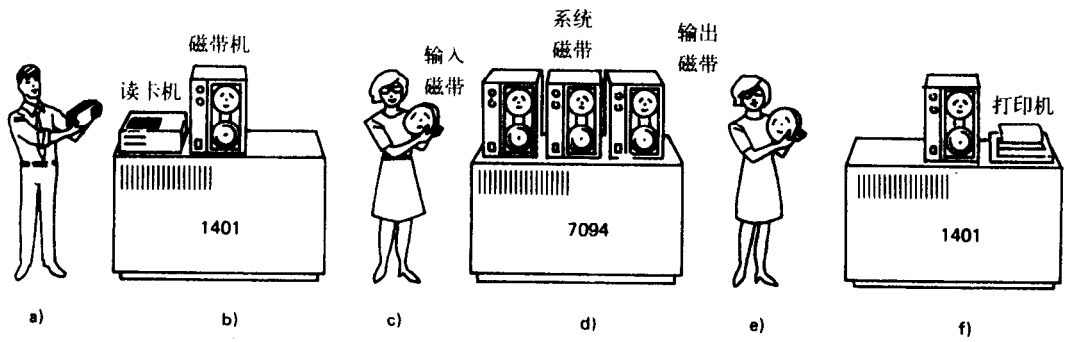

图1-2　一种早期的批处理系统

a)程序员将卡片拿到1401机处　b)1401机将批处理作业读到磁带上　c)操作员将输入带送至7094机
d)7094机进行计算　e)操作员将输出带送到1401机 (f)1401机打印输出

在一个小时的批量作业收集之后，磁带倒转后被送到机房里并装到磁带机上。随后操作员装入一个特殊的程序(现代操作系统的前身)，从磁带上它读入第一个作业并运行。其输出写到第二盘磁带上，并不打印。每个作业结束后，操作系统自动地从磁带上读入下一个作业并运行。当一批作业完全结束后，操作员取下输入和输出磁带，将输入磁带换成下一批作业，并把输出

磁带拿到一台1401机器上进行脱机(off line，即，不与主计算机联机)打印。

典型的输入作业结构如图1-3所示。一开始是张$JOB卡片，它标识出所需的最大运行时间(分钟)、计费帐号，以及程序员的名字。接着是$FRORTRAN卡片，通知操作系统从系统磁带上装入FORTRAN语言编译器。之后是待编译的源程序，然后是$LOAD卡片，通知操作系统装入编译好的目标程序(目标程序通常保存在临时磁带上以便于马上使用)。接着是$RUN卡片，告诉操作系统运行该程序并使用随后的数据。最后，$END卡片标识作业结束。这些基本的控制卡片是现代作业控制语言和命令解释器的先驱。

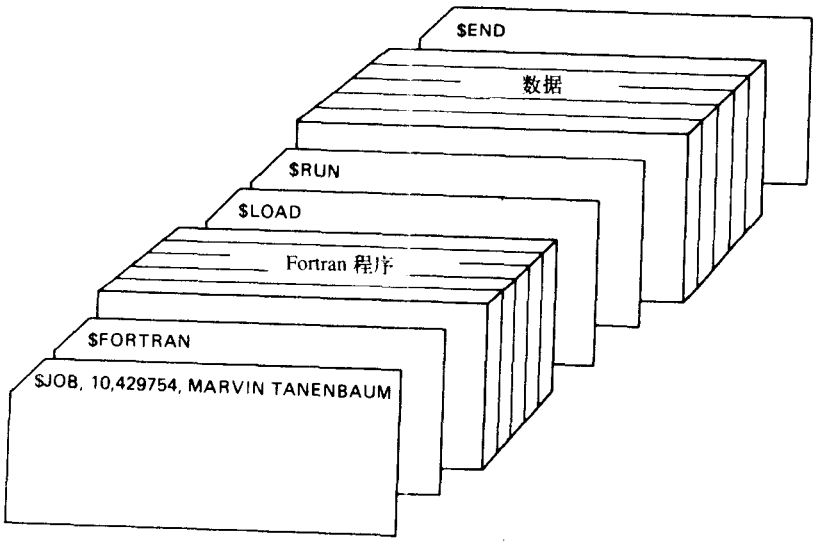

图1-3 典型的FMS作业结构

第二代计算机主要用于科学与工程计算，例如解偏微分方程。这些题目大多用FORTRAN语言和汇编语言编写。典型的操作系统是FMS(FORTRAN Monitor System，FORTRAN监控系统)和IBSYS(IBM为7094机配备的操作系统)。

1.2.3 第三代计算机(1965~1980)：集成电路芯片和多道程序

60年代初期，大多数计算机厂商都有两条完全不同并且不兼容的生产线。一条是基于字的大型的科学用计算机，诸如IBM 7094，主要用于科学和工程计算。另一条是基于字符的商用计算机，诸如IBM 1401，银行和保险公司主要用它从事磁带归档和打印服务。

对厂商来说，开发和维护两种完全不同的产品是昂贵的。另外，许多新的计算机用户一开始时只需要一台小计算机，后来可能需要一台较大的计算机，而且希望能够更快地执行原有程序。

IBM公司试图通过引入System/360来一次性地解决这两个问题。360是一个软件兼容的计算机系列，其低档机与1401相当，高档机则比7094功能强很多。这些计算机只在价格和性能(最大存储器容量、处理器速度、允许的I/O设备数量等)上有差异。由于所有的计算机都有相同的体系结构和指令集，因此在理论上，为一种型号机器编写的程序可以在其他所有型号的机器上运行。而且360被设计成既可用于科学计算，又可用于商业计算，这样一个系列的计算机便可以满足所有用户的要求。在随后的几年里，IBM使用更现代的技术陆续推出了360的后续机型，如著名的370、4300、3080和3090系列。

360是第一个采用小规模集成电路的主流机型，与采用分立晶体管制造的第二代计算机相

比，其性能价格比有很大提高。360很快就获得了成功，其他主要厂商也很快采纳了系列兼容机的思想。这些计算机的后代目前仍在大型的计算中心里使用。

“单一家族”思想的最大优点同时也是其最大的缺点。原因所有的软件，包括操作系统，都要能够在所有机器上运行，从小的代替1401把卡片拷贝到磁带上的机器，到用于替代7094进行气象预报及其他繁重计算的大型机；从只能带很少外部设备的机器到有很多外设的机器；从商业领域到科学计算领域等。总之，它要有效地适用于所有的用途。

IBM(别人也不行)无法写出同时满足这些需求相互冲突的软件。其结果是一个庞大的又极其复杂的操作系统诞生了，它比FMS大了约2~3数量级规模。其中包含有数千名程序员写的数百万行汇编语言代码，不可避免地也有成千上万处错误，这就不断地导致发行新的版本试图更正这些错误。每个新版本在订正老错误的同时又引入新错误，所以随着时间的流逝，错误的数量可能大致保持不变。

OS/360的设计者之一Fred Brooks 后来写过一本书(Brooks， 1975)描述他在开发OS/360过程中的经验。我们不可能在这里复述该书的全部内容，不过其封面是一群史前动物陷入泥潭而不能自拔。1991年出版的Silberschatz等人的著作的封面也表达了类似的观点。

抛开OS/360的庞大和存在的问题，OS/360和其他公司的类似的第三代操作系统的确合理地满足了大多数用户的要求。同时，它们也使第二代操作系统所缺乏的几项关键技术得到广泛应用。其中最重要的可能是多道程序设计(multiprogramming)。在7094机上，若当前作业因等待磁带或其他I/O而暂停时，CPU就只能简单地踏步直至该I/O完成。对于CPU操作密集的科学计算问题，I/O操作较少，因此浪费的时间很少。然而对于商业数据处理，I/O操作等待的时间通常占到80%~90%，所以必须采取某种措施减少CPU时间的浪费。

解决办法是将内存分几个部分，每一部分存放不同的作业，如图1-4所示。当一个作业等待I/O操作完成时，另一个作业可以使用CPU。如果内存中可以存放足够多的作业，则CPU利用率可以接近100%。在主存中同时驻留多个作业需要特殊的硬件来对其进行保护，以避免作业的信息被窃取或受到攻击，360及其他第三代计算机都配有此类硬件。

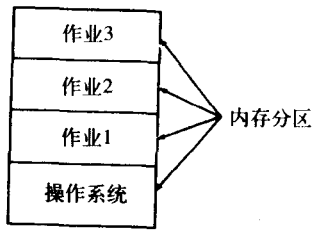

图1-4 内存中有三个作业的一个多道程序系统

第三代计算机的另一个特性是，卡片被拿到机房后能够很快地将一个作业从卡片读入磁盘。于是任何时刻当一个作业运行结束，操作系统就能将一个新作业从磁盘读出，装入空出来的内存区域运行，这种技术叫做spooling (Simultaneous Peripheral Operation On Line，同时的外部设备联机操作，亦称假脱机技术)，该技术同时也用于输出。当采用了spooling技术后，就不再需要IBM 1401机，也不必再将磁带搬来搬去。

第三代操作系统很适于大型科学计算和繁忙的商务数据处理，但其实质上仍然是批处理系统。许多程序员很怀念第一代计算机的使用方式，他们那时可以几个小时独占一台机器，可以随时调试他们的程序。而对第三代计算机，从一个作业提交到运算结果取回往往长达数小时，更有甚者，一个逗号的误用会导致编译失败，可能浪费程序员半天时间。

程序员们希望很快得到响应，这种需求导致了分时系统(timesharing)的出现。它实际上是多道程序的一个变种，每个用户都有一个联机终端。在分时系统中，假设有20个用户登录，其中17个在思考或谈论或喝咖啡，则CPU可给那三个需要的作业轮流分配服务。由于调试程序的用户常常只发出简短的命令(如编译一个五页的源文件)，而很少有长的费时命令(如上百万条记录的文件排序)，所以计算机能够为许多用户提供交互式快速的服务，同时在CPU空闲时还能