Springer

计算机科学本科核心课程教材

Introduction to Operating System Design and Implementation
The OSP 2 Approach

操作系统设计与实现
(OSP 2 方法)

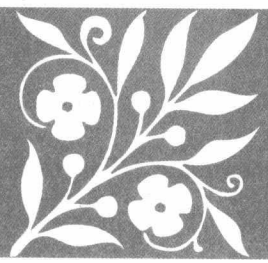
Michael Kifer　Scott A. Smolka　著
王俊华　等译

清华大学出版社

Springer

计算机科学本科核心课程教材

Introduction to Operating System Design and Implementation
The OSP 2 Approach



操作系统设计与实现
(OSP 2 方法)

Michael Kifer　Scott A. Smolka　著
王俊华　等译

清华大学出版社
北京

图书在版编目（CIP）数据

操作系统设计与实现（OSP 2 方法）/（美）基夫尔（Kifer，M.），（美）斯莫尔卡（Smolka，S. A.）著；王俊华等译．—北京：清华大学出版社，2010.2
书名原文：Introduction to Operating System Design and Implementation, The OSP 2 Approach
ISBN 978-7-302-21610-0

Ⅰ．①操…　Ⅱ．①基…②斯…③王…　Ⅲ．①操作系统—高等学校—教学参考资料　Ⅳ．①TP316

中国版本图书馆 CIP 数据核字（2009）第 228167 号

责任编辑：龙啟铭
责任校对：徐俊伟
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 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185×230　　**印　　张**：8.75　　**字　　数**：185 千字
版　　次：2010 年 2 月第 1 版　　**印　　次**：2010 年 2 月第 1 次印刷
印　　数：1～3000
定　　价：21.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：029951-01

译序

现代操作系统的基本原理和算法十分复杂，没有丰富的实践经验是难以掌握的。本书的目的在于利用用户友好而且灵活的 OSP 2 教学软件环境，教授操作系统设计与实现的相关概念。通过本书的学习，可以掌握操作系统的诸多重要特性，而且可以同时避免接触底层的、与机器相关的问题。

本书讲解如何进行 OSP 2 项目程序设计。主要内容如下：

- 第 1 章介绍 OSP 2 的重要概念，这是使用 OSP 2 系统的基础。内容包括 OSP 2 的组织结构、编译方法、运行和提交 OSP 2 项目以及 OSP 2 编程的一般规则。
- 第 2 章给出一个 OSP 2 的示例教程，为学生提供指导。
- 第 3 章讲解和实现现代操作系统的任务管理，并提供结构清晰的编程环境，实现任务管理的方法。
- 第 4 章讲解线程管理和调度，实现对线程的最常用操作以及定时器中断处理程序，完成基于时间配额的线程调度算法。
- 第 5 章讲解虚拟存储的知识、现代存储管理的技术，并提供实现这些技术的编程环境。
- 第 6 章讲解设备 I/O 的知识以及相关的设备驱动方面的知识。重点是磁盘 I/O 请求的调度上。
- 第 7 章、第 8 章和第 9 章分别介绍文件系统、进程间通信和资源管理。

本书在真实、灵活、易用的系统编程环境中，让学生练习操作系统设计与实现技能以及面向对象方法。

本书主要由王俊华翻译，参加翻译的还有李志云、李晓春、陈安华、侯佳宜、许伟、戴文雅、于樊鹏、刘朋、王嘉佳、邓卫、邓凡平、李波、程云建、许晓哲、朱珂等。

前言

OSP 2 不仅是现代操作系统的一种实现，同时也是开发实现项目的灵活环境，可用于操作系统设计的入门课程。本书的写作意图在于满足操作系统入门教材的使用需求，书中包含了三个学期的设计项目。通过这些设计项目，向学生讲解操作系统的诸多重要的本质特性，同时，也使他们不必关心与机器相关的底层属性。这样，即便只学习一个学期，学生也能学会虚拟存储管理的页面替换策略、CPU 调度策略、磁盘寻道时间优化以及操作系统设计的其他问题。

OSP 2 是用 Java 程序设计语言编写的，学生也将使用 Java 编写其 OSP 2 项目。因此，作为使用 OSP 2 的先决条件，学生应该具备扎实的 Java 编程能力和技巧；并深谙面向对象编程思想，如类、对象、方法和继承等；学过大学“计算机科学”课程的数据结构部分；还必须具备 Java 编程环境的实用知识，即 javac、java、文本编辑等；OSP 2 是最初的 OSP 软件的后继版本，最初的 OSP 软件发布于 1990 年，是用 C 语言编写的。

OSP 2 由一系列的模块组成，每一模块执行一项操作系统服务，例如设备调度、CPU 调度、中断处理、文件管理、存储管理、进程管理、资源管理和进程间通信。可以按照希望的顺序来组织项目，以便与课堂讲义的内容保持同步。OSP 2 软件在销售时附带了每一模块的 Java 实现参考，该参考是提供给课程指导教师使用的。

OSP 2 项目都有清晰定义的 API(应用程序编程接口)，学生必须实现它们才能成功完成项目。这样，OSP 2 就可以锻炼学生在“开放”的环境下工作。在这种环境下，学生必须进行程序设计，以满足具体的项目需求，并使用 API 实现与其他子系统的接口。

OSP 2 项目由标准 OSP 2 模块的“部分装载模块”(partial load module)组成，学生实现自己所分配到的模块，然后将其与部分装载模块链接起来。最终的结果就是一个部分由学生实现的、新的、完整的操作系统。另外，每个项目还包括一个或多个“*.java”文件，其中存储了各分配模块用到的类和方法的声明。这些文件提供了模板(template)，学生只需在其中填写所需方

法的代码。这种方式保证了与 OSP 2 接口的一致，而且减轻了原先指导教师和学生都要做的许多常规代码的输入劳动。

OSP 2 的核心是一个模拟器，它使用动态演化的、多程序运行的用户进程集来虚拟计算机系统。所编译的其他所有 OSP 2 模块则用来恰当地响应模拟器生成的操作系统驱动事件。模拟器能够“理解”它与其他模块之间的交互作用，因为通常情况下，它能够检测到其他模块对模拟事件的错误响应。在这种情况下，模拟器会恰当地结束问题程序的执行，并向用户发送有意义的消息，指示可能的错误根源。这一功能不仅可以作为学生的调试工具，也可以作为指导教师的教学工具，因为该功能实质上确保学生编写的程序必须是完美无缺的，才能被模拟器所接受。（当然，模拟器的检验并不能取代教师对学生所写程序的检查，从软件工程的角度考虑，教师的检查工作是为了确保程序设计的恰当性和合理性。）

通过操作模拟参数（simulation parameters），可以动态调整模拟器生成的作业流的复杂度。实际上，这为指导教师提供了一种测试学生所写程序质量的简单而有效的方式。另外，还有可以让学生自己调试程序的功能，包括详细的系统事件记录器和各种系统钩子程序。当检测到 OSP 2 警告或错误时，可以用这些钩子程序调用学生提供的方法。此外，图形用户界面（GUI）为学生和指导教师提供了输入模拟参数和查看与 OSP 2 执行有关的各种统计数字的便捷途径。

OSP 2 的底层模型不是任何操作系统的克隆。相反，它是多个操作系统公共特性的抽象（尽管时常可以察觉 OSP 2 向 Unix 和 Mach 操作系统的倾斜）。而且，OSP 2 模块的设计隐藏了许多底层需要关心的东西，却仍然包含了与现代操作系统真实对应的最突出的方面。将这些方面的实现作为操作系统入门课程的项目是非常合适的。

如何使用本书

本书主要是作为一本手册，讲解如何进行 OSP 2 项目程序设计。第 1 章讲解 OSP 2 的整体组织结构。第 2 章带领学生使用 OSP 2 完成一个实例。每一后继章节包含一个详细的 OSP 2 项目，首先陈述项目的目标，然后简短介绍与该章主题相关的基本操作系统概念，接下来则尽量缩小 OSP 2 手册与教科书的差距。在第一个作业发布之前，学生应该仔细阅读前言和第 1、2 章的内容。等到具体的项目分配下来（例如，一个线程管理项目），则应该仔细阅读相应章节（如果是关于线程的项目，则应该阅读第 4 章）。每一项目章节都详尽地讲解了要求学生实现的 OSP 2 模块的 API，包括对项目中每一方法功能的清晰说明。实现项目任务可能需要来自其他项目模块的方法，因此还提供了这些方法的列表。学生需要参考相关章节，以获得对这些方法更详细的说明。

本书的目标

本书的目标除了为学生提供一本 OSP 2 项目手册之外，更广泛地说，是讲述 OSP 2 环境，如下所述：

- ✧ 在以下方面，教会学生基础的操作系统概念：
  - ——进程和线程管理
  - ——存储管理
  - ——文件系统
  - ——进程通信
  - ——I/O 设备管理
  - ——资源管理
- ✧ 给学生创造在逼真的操作系统编程环境中练习上述技能的机会。
- ✧ 给学生布置具有挑战性的个人或团队编程任务，激发学生的“主动学习”热情，强化和扩展课堂讲义。
- ✧ 给学生布置对真实、正常运行的操作系统做出重大改动的编程任务，从而使学生熟悉操作系统实现的本质。
- ✧ 给指导教师提供灵活的操作系统编程项目，指导教师可以方便地调整自己的授课计划。

致谢

十分感激 OSP 2 开发团队的老一辈成员！例如，Sanford Barr 完成了事件引擎最初的设计和实现；William Ries、Adam Sah 和 Tomek Retelewski 与 Sanford 一起设计和实现了用 C++ 编写的 OSP 2 的早期版本；Fang Yang 负责将事件引擎和几种其他模块从 C++ 版本移植到 Java 版本；Kevin McDonnell 和 Peter Litskevitch 负责设计和实现当前版本的大多数模块，并为之编写文档；Jingjing Wei 负责实现最新的可定制 GUI 版本；Eric Nuzzi 为 OSP 2 代码设计了系统的测试协议；Martin Bruggink 负责实现 PORTS 模块；Xiaohua Wu 负责实现 RESOURCES 模块；David McManamon 负责实现供学生通过电子方式提交 OSP 2 作业方案的软件程序。

OSP 2 的某些部分依赖第三方软件程序。特别感谢 Retrologic 开发了优秀的 Java 混编器，并在 Lesser Gnu Public License(LGPL)的授权下发布。

最后，感谢 Springer London Ltd 的 Wayne Wheeler 和 Catherine Brett，他们为本书提供了编辑方面的专家意见。

目录

第1章

OSP 2 的组织结构

1.1 本章学习目标

本章的学习目标是理解 OSP 2 的一些重要概念，它们是使用 OSP 2 系统所必须具备的基础知识。具体内容包括(分成几个模块)讲解 OSP 2 的组织结构，介绍 OSP 2 的编译方法，如何运行和提交 OSP 2 项目以及 OSP 2 编程的一般规则。由于本章内容的基础性，在进行指导老师布置的 OSP 2 项目之前，都应该仔细阅读或复习。

1.2 操作系统基础

正如前言所述，OSP 2 是由众多模块集合在一起而组织起来的，每一模块都与一类 OSP 2 负责管理的资源相对应。针对学生的 OSP 2 编程作业，指导教师会布置其中的一个或多个模块，要求学生实现它们，并将这些模块放入整个系统中，通过模拟运行来确保学生所开发的代码能够正确而有效地运行。本章略微详细地讲解 OSP 2 的这种模块划分，同时也将提供学生完成作业所需的其他有用信息。不过，首先还是让我们自问一下：何谓操作系统？OSP 2 属于哪种操作系统？

何谓操作系统？ 为了准确理解什么是 OSP 2 以及它是如何组织起来的。首先考虑以下问题是不无裨益的：何谓操作系统？关于这一问题，有两种广泛流行的观点：一种观点认为操作系统是“扩展机”(extended machine)，另一种观点认为操作系统是“资源管理器”(resource manager)。根据第一种观点，操作系统的功能是向用户展示比底层硬件更加容易编程的“扩展机”或“虚拟机”等价物。程序设计通过操作系统的系统调用接口(system call interface)来实现，系统调用接口是系统调用的集合，应用程序可以通过调用它们来获得这种或那种服务。例如，有些系统调用是用来读写文件，也有些系统调用是用来更改定时器的数值。而且，通过唤醒系统调用来获得系统服务与查阅硬件规格说明书和机器寄存器相比简单多了，而如果没有操作系统，用户就不得不这样做。

两种著名的系统调用接口分别是用于 Microsoft Windows 各种版本（Windows 2000/XP/Vista）的 Win32 API（应用程序编程接口）和用于 Unix 操作系统（如 System V、BSD 和 Linux）的 POSIX。OSP 2 有自己的系统调用接口，本书的后续章节将会讲解组成这一接口的系统调用（Java 方法）。

根据第二种观点，操作系统负责有效和公平地管理计算机系统的资源。这些资源包括处理器（CPU）、存储器（真实的或虚拟的）、磁盘等设备、文件和目录以及网络连接（或端口）。所谓有效，指的是只要可能，操作系统就应该最大化资源利用；所谓公平，指的是在用户程序执行过程中，保证公平地分配系统资源。注意，这里举例给出的大部分资源属于物理硬件资源，只有文件和目录除外。操作系统中负责管理这些“逻辑资源”（logical resources）的部分称为文件系统（file system）。

本章接下来的部分将澄清把操作系统作为资源管理器看待的观点更加适合于 OSP 2，因为 OSP 2 的系统调用接口是以各种资源的形式组织起来的，OSP 2 负责管理这些资源。更准确地说，OSP 2 是被组织成许多模块——确切地说是 Java 程序包，对于要求 OSP 2 管理的任一类型的资源，都有一个这样的模块。例如，每一存储器、设备、端口等资源都分别有一个 OSP 2 模块，各模块存储（定义）了与该模块相关的多个 Java 方法。综合起来，这些模块构成了 OSP 2 的系统调用接口。

操作系统的不同版本。为更好地理解 OSP 2，明白可以在多个不同版本的操作系统之间进行选择是大有好处的。对此学生的脑海中会立即浮现出几个操作系统版本，相信学生也都听说过，它们是 Unix、Linux、Windows 和 MacOS。这些操作系统的主要区别在于其结构，当然，系统调用接口也不同。Windows XP/Vista、Solaris——太阳微系统公司（SUN Microsystems）的一种 Unix 版本和 Mach——20 世纪 80 年代在卡耐基梅隆大学（Carnegie Mellon University）开发的一款操作系统，后来影响了一大批商业操作系统，如 MacOS X，这类操作系统在如下意义上可以视作面向对象的系统：它们将基本的系统资源描述为对象，对象之间有清晰的消息传递接口。

尽管 OSP 2 并不是根据任何操作系统构建的，但在其结构的某些部分，的确能够看到其向 Unix 和 Mach 的倾斜。向 Unix 的倾斜在 FileSys 程序包中表现得最为明显，它使用索引节点（i-node）描述保存在磁盘上的文件，目录则将文件名映射到索引数字（i-numbers，inode indices）。在 PORTS 程序包中可以发觉 Mach 的影响，该程序包将 Mach 式的端口用于进程间通信。Mach 还将端口用于异常处理（每一进程有一个异常端口），这一点 OSP 2 并未涉及。

OSP 2 是名副其实的面向对象的操作系统。它是用面向对象的 Java 编程语言编写的。系统资源和数据结构按类描述，从而在对象之间提供了清晰的方法调用接口。而且，OSP 2 使用子类指定对象。例如，I/O 请求模块（I/O Request Block，IORB）就是 Event 的一个子类。这样，线程就可以在操作系统中等待，并在事件发生时被通知。

另一种区分操作系统的方法，某种意义上说，也是区分新旧操作系统的方法，就是这些操作系统是否支持线程(thread)。在较旧的操作系统中，如 Unix，正在执行的程序是以进程的形式组织起来的：操作系统的任务是调度 CPU 上的进程，将 CPU 从一个进程切换到另一个进程，以实现多程序运行(multiprogramming)的目的。多程序运行是一种以最大化资源利用为目标的技术。其基本思想是同时有多个进程的存储驻留，并将 CPU 从已经进入阻塞状态而等待某个事件(例如，I/O 操作完成)的进程切换到已准备好执行的进程。这样，就可以让 CPU 大部分时间保持在忙碌状态进行有用的工作，这正是资源管理器应该争取做到的。

为给多程序运行下一个论断，则要更细致地考虑一下将 CPU 从一个进程切换到另一个进程意味着什么，这一操作通常称为上下文切换(context switch)。其中牵涉到多个步骤。首先，当前正在执行的进程必须从 CPU 移除，放入该进程正在等待的事件的相关队列中。然后，将 CPU 决定调度执行的下一个进程分派给(dispatch)CPU。这又牵涉到重设许多机器寄存器(例如，程序计数器、通用功能寄存器、存储管理寄存器等)，将这些寄存器的值设置为新分派的进程上次运行时的值。此时，就可以恢复该进程运行了。这是简化了的关于上下文切换背后运行机制的公认不误的观点；有关这一话题的更多内容将在第 4 章进一步讲解。

在较新型的系统中，例如 Mac、Solaris 和 Windows 2000/XP/Vista，调度和分派的执行单元已经不再是进程了，而是线程；简单地讲，进程是线程的容器，可以容纳一个或多个线程。这种进程通常又称为任务(task)，这也是本书约定采用的称谓。那么，任务作为线程的“容器”又是什么意思呢？意思是作为任务组成要素的线程共享分配给任务的资源，这些资源包括存储器、文件和通信端口。结果，将 CPU 从一个线程切换到另一个线程要比将 CPU 从一个进程切换到另一个进程简单得多，而对于不支持线程的操作系统，只能采用后一种方式。正如我们将看到的，OSP 2 支持进程和线程。

操作系统是事件驱动的。操作系统是所谓的事件驱动系统(event-driven system)的一个绝佳的例子。正如名称所反映的，事件驱动系统响应它所熟悉的某种事件而触发动作。例如，GUI(图形用户界面)程序就是一个事件驱动系统，它响应用户鼠标的单击；最终执行哪块代码则取决于用户所单击的界面元素(工具栏项目、普通按钮、单选按钮等)。而对于操作系统的情况，OS 所响应的事件包括用户(或事件系统)程序所做的系统调用、硬件中断和机器错误。典型的事件驱动系统是用包含在 while 循环中的大块 case 语句构造的，while 循环“捕捉”系统打算响应的各种事件。一旦捕捉到事件，则执行与该事件相应的 case 语句。

这种事件循环结构在操作系统中确实存在。例如，不妨考虑一下系统调用在一般的操作系统中是如何执行起来的。调用系统调用的程序首先将系统调用的参数推入系统堆栈中。系统调用的编码保存在寄存器中，并执行陷阱指令(trap instruction)从用户模式

切换到内核模式。内核检查系统调用的编码，跳转到正确的系统调用处理程序。这一跳转过程通常是通过一张被系统调用编码索引的系统调用处理程序指针表完成的。在这个时候，系统调用处理程序开始运行。当运行结束时，在紧跟陷阱指令的指令处，控制权可能重新交给调用程序。

操作系统对硬件中断的处理采用与事件驱动类似的方式。对于硬件中断的情况，一部分系统存储器被保留，用于保存中断向量。使用触发中断的设备的设备号，并通过中断向量的索引可以找到该设备的中断处理程序的地址。

OSP 2 也是事件驱动的，毕竟，作为一种操作系统，这一点不足为奇。但是，OSP 2 响应的是模拟事件(simulated event)。也就是说，OSP 2 的核心部分是一个称为事件引擎(event engine)的模拟器，见图 1.1，该模拟器半随机地产生前面讨论过的各种事件(系统调用、硬件中断等)。响应某个事件时，就调用合适的 Java 方法。例如，假定事件引擎产生了一个与打开文件的系统调用实例相对应的事件，则调用 FILESYS 模块中的 open()方法。而且，假如指导教师为将 FILESYS 模块作为作业项目分配给学生，那么，响应该事件时执行的就是学生为 open()方法编写的代码。这实际上就是对 OSP 2 内部工作机制的简化描述。第 1.9 节将更详尽地讲解 OSP 2 的事件处理。

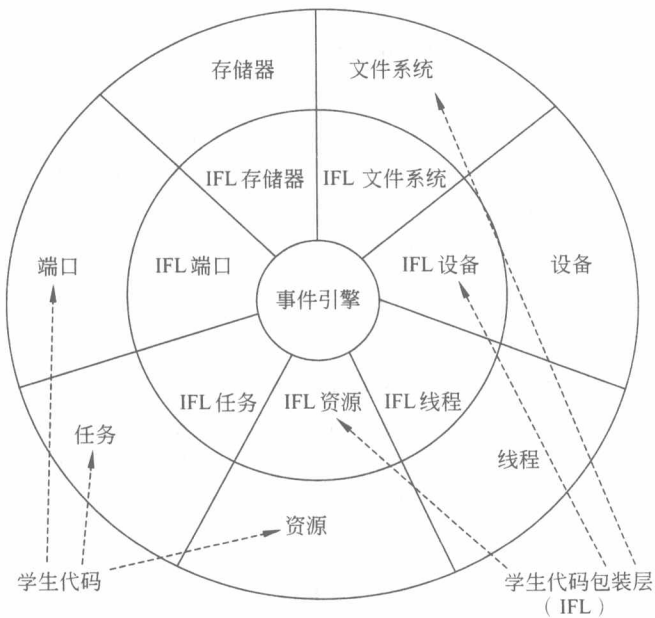


图 1.1 OSP 2 的逻辑结构

所有这些意味着，在 OSP 2 中，本质上没有执行任何用户程序，事件引擎以事件流的形式模拟全部用户程序，OSP 2 对这些事件做出响应。这种以模拟为基础的方法有多种

优点。首先，事件是通过一种所谓的 OSP 2 接口层(interface layer，IFL)传递的，OSP 2 接口层介于事件引擎和存储系统调用代码的各种 OSP 2 模块之间(见图 1.1)。因此，IFL 有机会监视系统调用方法的执行，以确保这些方法的行为在语义上是正确的。一旦在学生实现的系统调用方法中检测到错误，IFL 可以向学生返回有意义的错误消息。这些消息对于学生调试代码有很大帮助。

IFL 扮演的另一有用角色，也是学生和指导教师感兴趣的角色：它收集事件流处理过程中的系统性能统计数据。IFL 收集的统计数据的例子包括 CPU 使用率、页面错误数以及磁盘取数臂移过的轨道数。这些统计数据对于评估学生的 CPU 调度算法、页面替换方案、硬盘调度算法等的性能十分有用。

以模拟为基础的方法还有另一个优点，就是要调试学生编写的操作系统模块时，无需编写和运行用户层的测试程序(如果使用真实的操作系统，就不得不做这些)：由模拟器为测试提供事件流。而且，通过操作模拟参数(simulation parameters)，还可以动态调整事件引擎所产生的事件流的组成和密度。例如，假定指导教师将 FILESYS 作为一个作业项目布置给学生，他可以设置模拟参数，使事件流包含高百分比的文件系统相关事件。这就为测试学生的程序提供了一种简单而且有效的方式。

OSP 2 并不仅仅模拟用户程序，它还模拟底层硬件，包括 CPU、磁盘、系统时钟、硬件定时器和中断向量。OSP 2 模拟的硬件将在第 1.4 节全面讲解。

OSP 2 的微内核结构。操作系统设计中的一个有趣的话题是单内核(monolithic kernel)与微内核(microkernel)结构的区别。这里的术语“内核”用来指代操作系统中运行在内核模式(kernel mode)下的部分：内核模式是相对于用户模式(user mode)具有更高执行优先权的模式，在内核模式下，执行中的代码具有对系统数据结构和服务的访问权限。回到单内核与微内核的区别上来，单内核将全部操作系统功能组织在单个进程中，而微内核只将一小部分的重要的功能分配给内核，包括地址空间、进程通信和基本调度。通过使用定义清晰的接口，很好地封装和界定了基本服务模块，这也是微内核的典型特征。

如图 1.1 所示，也如前面所述，OSP 2 具有分层结构体系，包括三个主要层次：事件引擎、IFL 和学生模块层。一旦控制进入学生模块，可以认为系统在内核模式下运行。既然在 OSP 2 中实际上并没有用户程序，取而代之的是以事件流的形式随机模拟的用户线程，在 OSP 2 中也就谈不上用户模式。因此，文件系统、任务管理子系统、虚拟存储管理子系统等，都运行在内核模式下。

因为 OSP 2 纯粹的面向对象设计，所有操作系统子系统都使用清晰定义的方法接口封装成模块(类)。这些子系统包括那些处理诸如线程调度和进程间通信等活动的基本子系统。OSP 2 所采用的这种结构有时被称为“修正的微内核结构”。

1.3 OSP 2 的组织结构

OSP 2 包含了许多可以作为编程作业分配给学生的项目。每个项目涉及一个独立 Java 包的实现，该 Java 包含有一个或者多个 Java 类及相关方法。正是因为这些 Java 包可以作为编程作业分配给学生的潜在可能性，我们常常将这些 Java 包称为学生包（student package）或者学生项目（student project）。但是，必须明白的是，这些 Java 包的参考实现是标准的 OSP 2 软件发行的一部分，必须将它们放在适当的位置才能使系统正常运行（除非某个包的参考实现已经被学生编写的实现替换掉）。每一学生包负责管理它的同类系统资源，如下所述：

- ✧ DEVICES：处理辅助存储设备（例如，磁盘驱动器）的 I/O 请求。
- ✧ FILESYS：实现文件系统，包括基本文件操作和目录结构。
- ✧ MEMORY：使用分页和分段等技术管理物理和虚拟存储器。
- ✧ RESOURCES：使用死锁检测和死锁避免算法管理抽象系统资源。
- ✧ TASKS：控制任务的创建和删除，任务是一系列线程及相关资源的容器。
- ✧ THREADS：负责创建、结束、分派、挂起和恢复线程，线程是 OSP 2 中的基本执行单元。
- ✧ PORTS：实现进程间通信功能，进程间通信允许线程之间相互发送消息。

为了说明学生项目是如何组织的，考虑 OSP 2 的 MEMORY 模块。该模块对应 Java 包 osp. Memory，其中包含 PageFaultHandler、PageTableEntry 和 FrameTableEntry 等类。每一个类都包含在自己的. java 文件 PageFaultHandler. java、PageTableEntry. java 和 FrameTableEntry. java 等中。对于 MEMORY 项目，要求学生实现与这些文件关联的各种类。

OSP 2 的核心部分是事件引擎（event engine），它是以事件为基础的模拟器，驱动着学生包的执行。事件引擎产生的事件调用学生包中的方法，这些方法表示系统调用（例如，创建任务、写文件）和硬件中断。总的来说，它们模拟多程序运行操作系统环境中执行程序流的行为。

在事件引擎和学生层之间还有一个层，称为接口层（interface layer，IFL）。IFL 监视学生包的运行，目的是捕捉学生代码中的语义错误（然后产生智能的错误或警告消息）和收集性能统计数据。这样，可以将 IFL 视为学生包外围的一种保护性的“包装层”。OSP 2 的逻辑结构如图 1.1 所示。

1.4 OSP 2 中模拟的硬件

OSP 2 的 Hardware 和 Interrupts 包模拟多程序运行操作系统中与硬件有关的部分。Hardware 包含有 4 个 Java 类[①]。

CPU：该类模拟被模拟机器的 CPU。它定义了一个方法 interrupt()，用于产生指定类型的中断(例如，磁盘中断、页面错误)。Interrupts 包支持的中断向量将在本节稍后部分讲解。

Disk：该类表示连接到系统的一块硬盘，其原型声明如下。

```
public class Disk extends Device;
```

它实现的方法提供了对磁盘物理特性及其当前操作状态的访问。该类中的方法包括：

- final public int getPlatters()

返回磁盘盘片的数量。

- final public int getTracksPerPlatter()

返回每一盘片的轨道数。

- final public int getSectorsPerTrack()

返回每一轨道的扇区数。

- final public int getBytesPerSector()

返回每一扇区的字节数。

- final public int getRevsPerTick()

返回每一周期的转数。

- final public int getSeekTimePerTrack()

返回磁头从一个轨道移动到相邻轨道的平均时间。

- final public int getHeadPosition(int track)

返回磁头的位置，也就是磁头所在的磁盘柱面。

这些方法可用于实现 I/O 调度，更多关于 OSP 2 设备的内容见第 6 章“磁盘请求调度”。

HClock：该类表示硬件时钟，可以使用下列方法访问当前的模拟时间：

- public final static long get()

---

① 注意，Hardware 和 Interrupts 包中的所有方法都声明为 final 类型的，这意味着不能创建它们的子类。这样做是出于面向对象的原因：可以认为这些类是“完美无缺”的，或者从概念上讲，它们不应该有子类。许多其他 OSP 2 包中的方法被声明为 final 类型，也是出于同样的原因。