**操作系统设计原理概述**

2024.05.15

*【教学目标】 从软件工程的角度，帮助学生系统性地认知操作系统，并在此基础上帮助学生进一步认知诸如作业、任务、进程、线程、虚拟存储器、文件以及共行、共享、空间隔离、访问控制、特权保护、微内核结构等操作系统基本概念。*

**目 录**

**§1** 操作系统的开发步骤

**§2** 操作系统的基本作用

**§3** 操作系统的系统界面

**§4** 操作系统的现代特征

**§5** 操作系统的内部功能

**§6** 操作系统的安全措施

**§7** 操作系统的体系结构

**§8** 操作系统的常见分类

**操作系统设计原理概述**

（讲义正文）

**§1 操作系统的开发步骤**

大家知道，现代计算机系统由硬件和软件两部分组成。其中，中央处理器（CPU）是硬件部分的核心，而操作系统（OS）则是软件部分的核心。中央处理器和操作系统常被人们分别称为现代计算机系统的“心”与“魂”。由此可见，操作系统在现代计算机系统中具有重要的地位。

作为计算机系统中的一个核心软件，操作系统的设计与实现同样遵循着软件工程的步骤、思想和原则：

**需求 系统 编码 产品**

**分析 设计 实现 测试**

**体系结构 部件**

**设计 算法设计**

**图1 软件开发过程示意图**

**1.1操作系统的系统需求**

【软件系统的系统需求】

所谓软件系统的系统需求是指，人们从软件系统的外部对软件系统提出的诸多期望。这些期望包含两种类型：

* 一是，期望软件系统为人们自己或者为别的系统成份提供某些服务；
* 二是，期望软件系统在提供这些服务时能够满足某些限制性要求。

人们常把前一类系统需求称为软件系统的功能性需求，把后一类系统需求称为软件系统的非功能性需求。

【操作系统的系统需求】

* 功能性需求：操作系统在计算机系统中的作用是什么？这个问题的答案通常具体表述为计算机用户需要的用户命令以及应用软件需要的系统调用。
* 非功能性需求：方便性、有效性（效率、性能）、灵活性（可扩展、可裁剪、可配置、可定制）、开放性（可兼容、可移植）、安全性（可靠性）等。

**1.2操作系统的硬件依赖**

大多数现代操作系统需要硬件平台提供以下一些硬件设施：

* 陷阱、异常、中断机制
* 计时装置
* CPU上下文切换机制
* 存储管理部件（MMU）
* 存储保护机制
* I/O控制器
* 特权机制等等

**§2 操作系统的基本作用**

计算机用户

应用软件 *应用程序员*

*操作系统* ***操作系统***

*开发者*

计算机硬件平台

**图2 现代计算机系统组成示意图**

操作系统在计算机系统中究竟发挥着什么样的作用呢？对于这个问题，人们很难给出一个统一的答案；这是因为这个问题的答案“因人而异”。这里所谓的“因人而异”是指，与操作系统打交道的各类人员对操作系统作用的描述是不同的。

我们可以把与操作系统打交道的人员大致分为三类：计算机用户、应用程序员以及操作系统开发者。我们把上述三类人员对操作系统作用的描述归纳为四种基本观点。

*（计算机用户的观点）：用户环境观点*

*（从外部看OS）*

*（应用程序员的观点）：虚拟机器观点*

*关于OS的*

*四种基本观点*

*（OS开发者观点之一）：作业组织观点*

*（从内部看OS）*

*（OS开发者观点之二）：资源管理观点*

**用户环境观点**：该观点认为，操作系统是计算机用户使用计算机系统的接口，它为计算机用户使用计算机系统提供了方便的操作环境。

**虚拟机器观点**：该观点认为，操作系统是建立在计算机硬件平台上的虚拟机器，它为应用软件提供了许多比计算机硬件功能更强或计算机硬件平台所没有的设施。

**作业组织观点**：该观点认为，操作系统是计算机系统计算活动的组织者，它负责协调在系统中运行的各个应用软件的运行秩序，为应用软件提供良好的运行环境。

**资源管理观点**：该观点认为，操作系统是计算机系统中各类软硬件资源的管理者，它负责构建、保护各种软件资源以及分配、回收、驱动各种硬件资源。

**§3 操作系统的系统界面**

习惯上，人们把由操作系统实现的、供计算机用户以及应用软件与操作系统进行交互的通道称为操作系统的系统界面或系统接口。其中，供计算机用户与操作系统进行交互的系统接口叫做用户界面或用户接口，供应用软件与操作系统进行交互的系统接口叫做程序接口。

**3.1 用户界面**

用户界面由一组命令及其解释程序组成，因此也叫做命令接口。所谓命令是指，计算机用户要求计算机系统为其工作的指示。一般说来，不同的操作系统其命令接口有所不同。这不仅可能体现在命令的数量、种类以及功能方面，也可能体现在命令的用法和形式方面。

从用法上看，传统的字符串命令分为批处理和交互式两种类型。批处理命令又叫做脱机命令，主要用来描述脱机作业。交互式命令又叫做联机命令，主要用来描述联机作业。

这里所谓作业（Job）是指，计算机用户为某种特定目标要求计算机系统所做工作的集合；其中，作业中的每项相对独立的工作称为作业步。脱机作业和联机作业的区别在于：计算机用户不能干涉脱机作业的处理过程，但能够随时调整联机作业的处理步骤（即可以随时改变作业步）。通常，人们用一组命令来描述作业；其中，每个命令定义一个作业步。

从形式上看，批处理命令通常为字符式，而交互式命令不仅有字符式，还有菜单式、图形式、语音式等多种形式。字符式命令是传统的命令形式。虽然对于缺乏经验的计算机用户来说，字符式命令十分繁琐、难以记忆，但是对于有经验的计算机用户而言，字符式命令使用起来十分灵活，所以至今仍有许多操作系统支持这种命令形式。图形式命令是目前最流行的命令形式。它非常直观而且易于使用，因而受到计算机用户的普遍欢迎，得到当下所有主流操作系统的广泛支持。菜单式命令是在图形显示器被广泛使用之前操作系统为用户提供的一种过渡性质的命令形式。由于与字符式命令相比它缺乏灵活性，与图形式命令相比它缺少直观性，因此菜单式命令现在主要作为图形命令接口的补充。语音式命令是当下最时髦的命令形式，它利用语音识别和语音合成技术，使计算机用户可以用自然语言来操纵计算机系统。

随着近些年来个人计算机系统尤其是手机的广泛流行，计算机系统的应用范围得到了进一步扩大，缺乏计算机专业知识的用户也随之增多。如何不断更新技术，为这些用户提供简洁易用、功能强大的命令接口，便成为操作系统领域的一个热门研究方向。我们相信，随着人工智能技术的发展，不久的将来操作系统用户界面将会更加人性化、个性化。

**3.2 程序接口**

程序接口也叫应用程序编程接口（Application Programming Interface，API）。它由一组系统调用（System Call）及其引用机制组成。系统调用是指，由操作系统实现的、供应用软件引用的系统服务。系统调用的引用机制涉及两个方面：一是，应用软件与系统调用之间的参数传递方式；二是，应用软件与系统调用之间的控制转移方式。通常，应用软件与系统调用之间可以用寄存器来传递参数。这种方式既简单又快捷，但是传递的参数不多。如果要传递许多参数，可以利用堆栈；当然，也可以使用专门设计的参数表。在现代操作系统中，自陷/软中断和过程调用是两种常用的应用软件到系统调用的控制转移方式。

对应用程序员而言，程序接口事实上定义了一台虚拟机器。该虚拟机器包含一组抽象概念以及与这组抽象概念相关的系统服务。其中，抽象概念定义了虚拟机器的基础设施，系统服务定义了相关基础设施的基本功能。对应用程序员来说，在这种虚拟机器上开发应用软件要比直接在硬件裸机上开发应用软件简单方便高效得多。这是因为，比起硬件裸机，这种虚拟机器提供了易于理解的设施、更加强大的功能。

任务（Task）是程序接口中最常见的、也是最重要的抽象概念之一。它常被应用程序员用来表达各种程序级共行计算单元。在传统操作系统中，任务等同于进程（Process）。传统上，进程是指程序在某个数据集上的一次相对独立的运行过程。在传统操作系统中，进程既是系统中独立运行的最小单位，也是系统中资源分配的基本对象。在现代操作系统中，任务的内涵被分解为两个概念：线程（Thread）和进程。其中，线程是指计算机系统中处理机所执行的一个相对独立的指令流。在现代操作系统中，线程才是独立运行的最小单位，而进程仅仅是资源分配的基本对象。现代操作系统允许同一进程中的多个线程共享进程所拥有的各种资源，包括共享可重入程序代码。

虚拟存储器（Virtual Memory，VM）也是程序接口中最常见、最重要的抽象概念。所谓虚拟存储器，简单地说，就是任务的逻辑地址空间。它是操作系统对计算机系统中多级物理存储体系进行高度抽象的结果。虚拟存储器使得在计算机系统中运行的程序其尺寸不再受限于计算机系统实际装配的物理内存的大小。

文件（File）是程序接口中另外一个最常见、最重要的抽象概念。所谓文件是指命名了的字节流。它是现代操作系统对计算机系统中种类繁多的外部设备进行高度抽象的结果。它屏蔽了各种外部设备千差万别的构造细节，用几种简洁的相关操作取代了各种外部设备多种多样的专有功能，为应用软件提供了一种统一的、规范的抽象设施及其功能。文件及其相关操作的引入极大地方便了应用程序员，但是对某些应用软件来说，这种统一的、规范的抽象设施未必合乎其用，比如多媒体应用软件。

上述任务、虚拟存储器、文件之间的关系非常类似于物理处理机、物理内存、物理设备之间的关系：物理处理机可以直接寻址（按物理地址访问）物理内存，但只能间接存取物理设备；类似地，任务可以直接寻址（按逻辑地址访问）虚拟存储器，但只能间接访问文件。

顺便指出，对应用程序员而言，程序接口通常还定义了应用软件框架以及编程范式，最典型最常见的编程范式就是并发程序设计。

**§4 操作系统的现代特征**

支持任务共行与资源共享是现代操作系统的两个最基本的设计目标，也是现代操作系统的两个最基本的特征。

**4.1 什么是任务共行和资源共享**

* **任务共行**

**本质上讲，共行这一概念是指，计算机系统中同时存在多个相对独立的计算活动。**

大家知道，程序这一概念表述的是，计算活动所遵循的静态的计算逻辑。因此，我们可以尝试从程序的角度来理解共行的含义。

从程序的角度看，**共行**意味着三种情形：一是，多道程序一起共同运行（程序之间的共行）；二是，单道程序各个部分一起共同运行（程序内部的共行）；三是，单道程序同时多次运行（程序自身的共行）。

第一种情形有赖于操作系统采用多道程序设计（Multiprogramming）技术。采用多道程序设计技术的操作系统允许多道程序同时准备运行；当正在运行的那道程序因为某种原因（比如等待输入或输出数据）暂时不能继续运行时，系统将自动地启动另一道程序运行；一旦原因消除（比如数据已经到达或数据已经输出完毕），暂时停止运行的那道程序在将来某个时候还可以被系统重新启动继续运行。

第二种情形有赖于程序本身采用并发程序设计（concurrent programming）范式。采用这一编程范式的程序常被称为并发程序。并发程序由多个可以独立运行的计算单元组成，每个计算单元只承担整个程序所定义的计算工作的一部分。

第三种情形有赖于程序本身具备可重入性（Reenterability）。具备可重入性的程序常被称为可重入程序。可重入程序被启动开启一次运行过程后，在该次运行过程尚未结束前，可以再次被启动开启另一次运行过程；这样的重复启动可以进行多次；可重入程序的多次运行过程互不干扰，运行结果也互不影响。

一般说来，不同的计算活动可以有相同的计算逻辑。也就是说，不同的计算活动可以使用同一个程序。这意味着，我们从程序的角度很难区分两个不同的计算活动，因而用程序这一概念也就很难准确地表述共行的内涵。为此，人们引入了任务这一概念。

在介绍程序接口时我们已经指出，任务常被应用程序员用来表达应用软件各种程序级共行计算单元。那么，本质上讲，任务究竟是什么呢？要理解任务的本质内涵，就必须从共行计算控制的角度来理解任务的含义。从共行计算控制的角度看，所谓任务是指，计算机系统在某个资源集合上所做的一次相对独立的计算过程。就任务而言，包括应用软件在内的各种程序也是其使用的一类资源，它们被用于表述任务内在的计算逻辑。

不难理解，任务表述的是计算活动动态的计算过程。由于任何一个计算活动其计算过程都是独一无二的，因此可以用计算过程来区分不同的计算活动，故而用任务这一概念可以更好地诠释共行的真实含义。

从任务的角度看，**共行**有两层含义：在宏观层面上，共行是指系统中有多个任务同时运行；在微观层面上，共行是指单处理机系统中的任务并发（Task Concurrency：即多个任务在单个处理机上交替运行）或多处理机系统中的任务并行（Task Parallelism：即多个任务在多个处理机上同时运行）。

在现代计算机系统中，任务以“走走停停”的方式运行，其推进速度是不可预知的。人们常把任务的这种运行特性称为异步性。

* **资源共享**

资源共享也有两层含义：从宏观上看，资源共享是指多个任务可以同时使用系统中的某个资源；从微观上看，资源共享是指多个任务可以交替互斥地使用系统中的某个资源。

在支持资源共享的系统中，被共享的资源既可以是硬件资源，也可以是软件资源。软件资源也叫虚拟资源，这是因为软件资源是用虚拟技术实现的。

经常地，人们用虚拟技术将当前系统中已经装配的某个物理设施（即硬件设施）变换为多个功能对等的逻辑设施（即软件设施），或者仿真实现当前系统中没有装配的某类设施。

虚拟技术的本质就是“抽象”。其实，“抽象”也是软件工程的重要思想和原则。这一思想和原则始终贯穿于操作系统的设计与实现过程。毕竟操作系统也是一个软件，它的设计与实现必然要遵循软件工程的思想和原则。

在现代计算机系统中，大多数物理设施以及用于定义软件设施的变量、堆栈、表格等数据结构，在一段时间内只允许一个任务占有，即使这个任务当下并不实际使用这些资源。人们常把具有这种使用特性的资源称为临界资源（或独占资源）。

**4.2 为何要任务共行和资源共享**

提高计算效率历来是人们对计算机系统提出的一个最基本需求。计算机系统软硬件开发者们早已发现，充分利用计算机系统中各种计算活动内在的共行特性是提高计算效率的一条重要途径。因此，设计并实现各种共行计算控制模型一直是计算机系统软硬件开发者们的一项主要工作。操作系统开发者们也不例外，设计并实现任务级共行计算控制模型已经成为现代操作系统开发者们的一项重要工作。

大家知道，任何计算机算法的设计者终将面临这样一个基本问题：算法的运行效率始终受制于有限的计算资源。从运行效率的角度看，要保持高效率，最好的办法就是让所有的算法代码都在运行。然而，客观现实是，支撑算法运行的计算资源总是有限的，不足以支撑全部代码同时运行。不难理解，在任务共行的情景下，这一问题更加严重。解决这一问题的一个基本思路是，让当下必须运行的算法代码使用计算资源，其它暂时无需运行的算法代码处于等待状态。这一思路就是我们前面所说的资源共享。

任务共行与资源共享二者互为存在前提：没有任务共行，就无需资源共享；而不能有效地实现资源共享，也就无法有效地实现任务共行。

**§5 操作系统的内部功能**

*任务控制*

*任务管理*

*低级通信*

*任务通信 本机通信*

*高级通信*

*远程通信*

*OS内部功能 可按地址存取的软件资源管理*

*（比如虚存管理）*

*软件资源管理*

*可按句柄访问的软件资源管理*

*（比如文件管理）*

*资源管理*

*处理机管理*

*主存储器管理*

*硬件资源管理 存储器管理*

*辅助存储器管理*

*I/O设备管理*

不管各个实际的操作系统在功能上有多大差别，它们的功能总是可以分为两类：一是启动、终止、控制应用软件的运行，二是构建、保护系统中的软件资源以及分配、回收、驱动系统中的硬件资源。前者称为任务管理，包括任务控制和任务通信。后者称为资源管理，包括软件资源管理和硬件资源管理。

作为操作系统的管理对象，任务是一类实体。组成此类实体的内容常被称为任务映像（Task Image）。任务映像大致包含以下四个部分的内容：

* 控制信息：包括任务的标识、状态、优先数等；
* CPU现场：也叫处理器上下文(CPU Context)，由CPU中的各个寄存器的内容组成；
* 地址空间：由任务拥有的可按地址存取的堆栈、代码、数据等组成；
* 资源集合：由任务拥有的可按句柄访问的命名存储区、信号量、文件等组成。

在现代操作系统中，任务都是多线程的。因此，CPU现场有多个副本，用于描述不同的线程其运行进度。

任务的地址空间中装配有各种可以直接寻址的软件资源。一般说来，任务地址空间的管理功能包括***装配***、***映射***、***保护***三种类型。其中，装配功能可进一步分为创建时装配和运行时装配两种类型。所谓创建时装配是指，在任务被创建过程中将任务按地址可能存取的全部软件资源装配到任务的地址空间中；该功能常常包含可执行程序的装入时动态链接功能。所谓运行时装配是指，在任务运行过程中将任务按地址即将存取的软件资源装配到任务的地址空间中；该功能总是包含可执行程序的运行时动态链接功能。映射功能也可进一步分为两种类型：单级映射和多级映射。单级映射是指，将任务地址空间映射到由主存储器构成的单级物理存储体系中；多级映射则指，将任务地址空间映射到由主存储器、辅助存储器等构成的多级物理存储体系中。保护功能意指防止非法存取任务地址空间中的软件资源。空间隔离、访问控制以及同步互斥机制是此类软件资源常用的保护技术。

句柄（Handle）是现代操作系统用来表示任务地址空间之外的某种软件资源的内部标识。文件是我们最熟悉的一种任务可按句柄访问的软件资源。事实上，在现代操作系统中还有一些软件资源可被任务按句柄访问，比如用作任务通信机制的信号量、命名存储区等。这些软件资源的共同特征包括：

* 有一个外部名字，通常是一个ASCII码字符串；
* 可以被多个任务共享；
* 受到系统的保护。

一般说来，系统为管理此类软件资源提供了***创建***、***打开***、***关闭***、***删除***等操作。其中，打开操作的作用是，将该类软件资源的名字转换成相应的句柄。显然，任何任务对该类软件资源进行访问之前都必须执行打开操作。由于命名存储区、信号量、文件等均有自己特有的属性，因此上述操作对它们将分别表现出不同的语义。

操作系统的硬件资源管理功能可进一步分解为处理机管理、存储器管理以及设备管理三种类型。尽管这几种硬件资源管理功能管理的对象各不相同，操作系统在其中扮演的角色却是一样的。操作系统硬件资源管理功能的实现遵循“*申请*­—***分配***—*使用*—*释放*—***回收***”这一模式。在这种模式中，操作系统扮演着资源分配者和回收者的角色。

除分配与回收功能外，所有操作系统其I/O设备管理模型都必须包含设备控制功能，该功能由设备驱动程序完成。在现代操作系统中，设备驱动程序常常是访问设备的唯一接口，它实际控制设备实现真正的I/O操作。

**§6 操作系统的安全措施**

在现代计算机系统中有多个任务在同时运行。这些任务或者由于相互合作或者因为共享资源而存在着种种制约关系。因此，它们的任何一个病态行为对整个系统的安全性都是一种威胁。大多数现代操作系统采用以下四种措施来防范这种威胁。

* 身份认证

身份认证是现代计算机系统的第一道安全防线。这里所谓的身份认证是指，操作系统利用计算机用户提供的信息，鉴别用户的真实性、验证真实用户的合法性、标识合法用户的访问权限等。由于通常情况下现代计算机系统中的各项计算活动是由计算机用户激发的，因此通过身份认证操作系统可以规范计算机系统中各项任务的行为。

目前，操作系统验证真实用户合法性的信息主要有三类。一是，用户掌握的秘密信息，比如口令（所知）；二是，用户持有的专属物品，比如U盾（所有）；三是，用户自身的生物特征，比如指纹（所属）。

* 空间隔离

现代操作系统常常为不同的任务规定不同的地址变换函数，使得不同的任务其逻辑地址空间被映射到物理存储体系的不同区域，从而实现不同的任务其逻辑地址空间彼此隔离。这样做将避免任务之间的相互干扰。

* 访问控制

若多个任务共享某些资源（比如代码或数据），那么空间隔离技术将无法避免这些任务之间相互干扰。此时，需要使用访问控制技术。访问控制技术分别就系统中的各种资源规定了在什么情况下它们可以被什么样的任务访问以及可以被这些任务施加什么样的操作。通常所谓的存储保护、文件保护、I/O保护等都属于访问控制的概念范畴。

* 特权保护

在现代操作系统中，访问控制并不限于简单的读写控制，还包括特权保护。在提供特权保护的系统中，所有的资源均被赋予某种特权级别，对它们的访问均按照某种特权规则进行。

特权保护并不仅仅是访问控制的一种。计算机硬件平台的某些指令­——诸如改变地址变换函数的指令、改变访问控制权限的指令、改变代码和数据特权级别的指令等——如果不加以控制使用，对那些蓄意制造麻烦的任务来说，空间隔离技术以及访问控制技术都将失去效用。这些指令常常被人们称为特权指令。特权保护也意味着对特权指令的使用加以控制。

尽管一些计算机硬件平台支持多级特权模式，但大多数现代操作系统只使用两级特权模式；其中一个用于应用软件（相应的特权模式被称为用户模式或用户态，User Mode），另一个用于操作系统（相应的特权模式被称为内核模式或内核态，Kernel Mode）。

一般地，为了保护操作系统，我们可以为操作系统规定一个专用的地址变换函数，使得操作系统的逻辑地址空间与各个任务的逻辑地址空间隔离开来。然而，由于在系统中运行的各个任务经常要引用操作系统提供的系统服务，因此这样做将导致在任务与操作系统之间进行频繁的跨越地址空间的访问，从而增加系统开销。所以绝大多数现代操作系统将其代码和数据作为任务逻辑地址空间的一部分，为系统中所有任务共享。在这种情况下，操作系统的保护只能利用访问控制技术和特权保护技术来完成。

**§7 操作系统的体系结构**

**7.1 现代操作系统总体功能结构**

计算机用户

***用户接口*** 应用

***子系统*** 软件

***基础平台子系统***

计算机硬件平台

**图3 现代操作系统总体功能结构示意图**

从功能的角度看，大多数现代操作系统其总体结构包含两类子系统：用户接口子系统和基础平台子系统。其中，用户接口子系统提供计算机用户需要的用户命令，基础平台子系统提供应用软件需要的系统调用。操作系统的内部功能都是在基础平台子系统中实现的。

用户接口子系统与基础平台子系统之间的相互关系具有单向性。具体地说，用户接口子系统在实现各种用户命令时能够引用基础平台子系统所提供的各种系统调用，但基础平台子系统在实现各种系统调用时不会引用用户接口子系统所提供的各种用户命令。

从原理上讲，一个操作系统可以实现多个用户接口子系统供不同的用户选用。从基础平台子系统的角度看，用户接口子系统与应用软件没有什么不同：它们与基础平台子系统的交互关系是一致的。毫无疑问，计算机用户可以象开发应用软件那样基于基础平台子系统所提供的系统调用来开发自己的用户接口子系统。在后续的讨论中，我们把用户接口子系统归类为应用软件而不再单独提起。

一般说来，基础平台子系统是大多数计算机系统中唯一与硬件平台直接交互的软件成份。因此，对于应用软件来说，基础平台子系统隐藏了硬件平台所提供的各种硬件设施及其功能。

**7.2 基础平台子系统的组织结构**

效率永远是操作系统的一项十分重要的非功能性需求。灵活性、安全性等则是现代操作系统的不可忽略的非功能性需求。这些非功能性需求均与操作系统的编程组织结构以及运行组织结构密切相关。

【基础平台子系统编程组织结构】

* 无结构

早期的操作系统，其实现内部功能的各种过程被无序地堆积在一起，毫无结构可言，因此被人们称为无结构操作系统。

无结构操作系统的开发者们常常关注操作系统的运行效率而忽视了操作系统的可维护性。在编写操作系统程序时，他们竭尽所能降低系统运行开销并为此牺牲了操作系统程序的条理性和可读性。这就导致无结构操作系统中各种过程的调用关系虽紧凑但杂乱无章。

* 模块化结构

随着时间的推移，操作系统的内部功能越来越多，相应地操作系统程序越来越庞大。在这种情况下，降低操作系统程序的复杂度，减少操作系统的设计与编程错误，也就不可避免地成为了操作系统开发者们另一个关注焦点。维持操作系统程序的条理性和可读性，进而增强操作系统的可维护性，也就理所当然地成为操作系统开发者们新的追求目标。

模块化程序设计技术的出现为实现这一新的目标提供了一条可行路径。一些操作系统开发者运用这一技术将实现操作系统内部功能的各种过程分门别类地封装在不同的模块中。他们努力实现，封装在模块内部的各个过程彼此之间的调用关系仍然十分紧凑，而模块之间的引用关系则相对松散。这样做的结果是，操作系统程序的可理解性大为提高，操作系统的可维护性大为增强，操作系统的开发进度也大为加快。

* 分层式结构

为了进一步提高操作系统程序的可理解性、增强操作系统的可维护性，一些操作系统开发者在开发模块化结构操作系统时，对操作系统模块之间的引用关系进行了规范，使其呈现出层次结构。这种结构的一个重要特点是：所有各层其实现方案不会引用其以上各层所提供的服务而是只引用其以下各层所提供的服务。

分层式结构有利于满足操作系统的可维护性、可裁剪性、可扩展性、可定制性、可移植性以及部件可重用性等非功能性需求，但不利于生成紧凑的高效的操作系统可执行代码（操作系统映像），而且构造一个纯粹的分层式结构将非常困难。

【基础平台子系统运行组织结构】

* 单模式运行结构

一些操作系统，尤其是早期的操作系统，其基础平台子系统是以单一可执行程序的形式运行的。人们常把这些操作系统其基础平台子系统称为单体系统。

取决于单体系统开发者的设计，单体系统在运行过程中，或者使用应用软件所用的计算机硬件特权模式（用户模式），或者使用专用的计算机硬件特权模式（内核模式）。无论哪一种设计决定，运行中的单体系统都只使用一种计算机硬件特权模式。基于这一特点，我们把单体系统的运行组织结构称为单模式运行结构。

* 双模式运行结构

经常地，现代操作系统开发者将基础平台子系统从功能上分解为扩展模块和内核模块两大部分，并把它们置于不同级别的计算机硬件平台提供的特权模式下运行。通常，大多数现代操作系统只用了两级特权模式：一是较低级别的用户模式或用户态，用于运行应用软件以及扩展模块；二是较高级别的内核模式或内核态，用于运行内核模块。我们把采用两级特权模式的基础平台子系统运行组织结构称为双模式运行结构。

计算机用户

应用软件

***扩展***

用户模式  ***模块*** 低特权级别

内核模式 ***内核模块*** 高特权级别

计算机硬件平台

**图4 基础平台子系统双模式运行结构示意图**

* 微内核结构设计思想

若操作系统开发者为基础平台子系统选用双模式运行结构，一个基本问题就不可避免地出现在他们面前：如何划分扩展模块和内核模块的功能？微内核结构设计思想为解决该问题提供了一种思路。我们可以把微内核结构设计思想表述为：尽最大努力剔除内核模块中的多余成份并把它们移到扩展模块中实现，内核模块只实现一些必要的简单的设施及其服务，从而保持内核模块的简洁高效可靠。

* 机制与策略分离原则

必须指出，尽管微内核结构设计思想为解决上述问题提供了一种思路，但在具体实施时仍然令人迷茫。何为内核模块中的多余成份？在操作系统工程实践中，这个新的问题仍然令人困扰。为此，有人提出了一个设计原则：操作系统机制与策略分离。通俗点说，策略决定“做什么”，而机制定义“如何做”。机制与策略分离原则就是，留下机制在内核模块中实现，把策略移到扩展模块中去。

* 扩展模块的运行组织

扩展模块的运行组织可以采用共享库的形式。共享库有两种形态：静态链接库和动态链接库。相较于静态链接库，动态链接库更有利于支持应用软件随着操作系统的不断升级而不断更新。不管何种形态，共享库总是在调用其服务的应用任务的上下文中运行。所谓任务的上下文(Task Context)，简单地说就是，任务在运行过程中可以访问的资源集合。

扩展模块的运行组织也可以采用任务的形式，即：扩展模块以多个任务的形式在系统中运行，应用任务通过某种任务间通信的方式访问扩展模块。远程过程调用（Remote Procedure Call，RPC）就是一种最常用的用于访问扩展模块的任务间通信方式。人们习惯上把扩展模块的这种运行组织形式称为客户/服务器(Client/Server，C/S)模式。

相较于任务组织形式，共享库组织形式在性能和效率方面具有优势；这是因为共享库组织形式避免了因地址空间切换或保护域切换而产生的系统开销。相较于共享库组织形式，任务组织形式在安全性和可靠性方面具有优势；这是因为任务组织形式为保护扩展模块而采取了诸如空间隔离、访问控制等相关措施。

* 内核模块的运行组织

陷阱指令/程序异常

***上半部***

***下半部***

硬件中断

**图5 内核模块运行组织示意图**

在大多数现代操作系统中，有三类事件会导致系统访问内核模块：陷阱指令、程序异常以及硬件中断。前两类事件的处理程序构成了内核模块的上半部，后一类事件的处理程序构成了内核模块的下半部。顺便指出：陷阱指令也叫软中断；因为它可以实现控制从低特权级别向高特权级别的转移，所以通常被用于系统调用。

内核模块的上半部与下半部有四点不同：

* 实现的功能不同

上半部实现的主要功能是，为应用软件和扩展模块提供一组系统调用以及处理当前任务运行过程中出现的异常事件。下半部专门处理由硬件平台各种设备产生的、并不与当前任务直接关联的各类中断事件。

* 激活的原因不同

当前任务在运行过程中引用上半部提供的系统调用或者产生异常事件时，上半部将被激活。当硬件平台产生中断事件时，下半部将被激活。

* 运行的方式不同

上半部以与当前任务同步的方式运行并且是可抢占的。下半部以与当前任务异步的方式运行并且是不可抢占的。

* 依赖的上下文不同

上半部在当前任务的上下文中运行。下半部在专用的系统上下文中运行。

内核模块的上半部与下半部总是共享一组数据结构和基本例程。为此，内核模块还实现了诸如信号、信号量等同步互斥机制。

*备注：Linux中断处理程序被分为Top Half和Bottom Half两部分。在我看来，Top Half应翻译成顶半部，Bottom Half应翻译成底半部。但是，的确有人把Top Half翻译成上半部，把Bottom Half翻译成下半部。这后一种译法不会影响人们对Linux中断处理程序编程模型的理解。只是这里有必要提醒：Linux中断处理程序的上半部和下半部与这里所说的内核模块的上半部和下半部不是一回事儿，因为它们所描述的对象是不同的，莫要搞混了！*

**§8 操作系统的常见分类**

**8.1 基于应用场景的分类**

（也是从历史进程的角度进行的分类，这是一种最常见的分类）

* 批处理系统（科学计算；性能指标：周转时间、吞吐率）：单道批处理系统、多道批处理系统
* 分时系统（信息查询；性能指标：响应时间）
* 实时系统（实时控制、流媒体播放；性能指标：截止时间）：硬实时系统、软实时系统

**8.2 基于硬件平台的分类**

* 单机系统
* 嵌入式操作系统
* 手机操作系统
* 个人电脑操作系统
* 多机系统：
* 多处理机操作系统
* 分布式操作系统
* 网络操作系统

**8.3 基于用户任务的分类**

* 单用户单任务操作系统
* 单用户多任务操作系统
* 多用户多任务操作系统

**思 考**

1. 当计算机系统通电开机后，操作系统是如何逐步掌控计算机系统的？（操作系统映像加载以及操作系统初始化过程）
2. 当计算机用户发出命令请求计算机系统做某项工作后，操作系统控制下的计算机系统是如何响应的？（用户命令的处理过程）