操作系统　2 月 26 日　第二周

PB18151866 龚小航

1. 从系统的角度理解，操作系统主要负责哪两大功能？

操作系统是与硬件紧密相连的程序，可以将操作系统看作**资源分配器**：面对许多甚至冲突的资源请求，操作系统应考虑如何为各个程序和用户分配资源以便系统能有效公平的运行。同时，从另一个方面，强调控制各种 I/O 设备和用户程序的需求，操作系统是控制程序。**控制程序**管理用户程序的执行，以防止计算机资源的错误或不当使用。

2. 什么是系统调用？阐述系统调用与 API 的区别和逻辑关系。

系统调用提供操作系统服务接口。操作系统内核中设置了一组用于实现各种系统功能的子程序，称为系统调用。

系统调用和 API 的区别：程序的运行空间分为内核与用户空间，在逻辑上它们之间是相互隔离的，因此用户程序不能访问内核数据，也无法使用内核函数。而系统调用就是一种特殊的接口。通过这个接口，用户可以访问内核空间。系统调用规定了用户进程进入内核的具体位置；API 就是应用程序接口，是一些预定义的函数，跟内核没有必然的联系。程序员调用的是 API（API 函数），然后通过与系统调用共同完成函数的功能。因此，API 是一个提供给应用程序的接口，一组函数，是与程序员进行直接交互的；而系统调用则不与程序员进行交互，它根据 API 函数，通过一个软中断机制向内核提交请求，以获取内核服务的接口。

系统调用和 API 的联系：一个 API 可能会需要一个或多个系统调用来完成特定功能。如果某个 API 需要内核提供的功能就需要系统调用，否则不需要。

3. 阐述 Dual Mode 的工作机制，以及采用 Dual Mode 的原因。

双重模式的工作机制：计算机硬件通过一个模式位来表示当前的模式：内核模式（0）、用户模式（1）。从而计算机就可以区分操作系统执行的任务和用户执行的任务。计算机系统执行用户应用时，系统处于用户模式，当该应用请求操作系统服务时，系统从用户模式切换为内核模式以满足请求。从计算机开机开始，在系统引导阶段，硬件从内核模式开始，操作系统接着加载，然后开始在用户模式下执行用户程序。一旦有陷阱或中断，硬件会从用户模式切换到内核模式。即每次操作系统能够控制计算机时他就处于内核模式；将控制权交给用户程序前，系统会切换回用户模式。

采用双重模式的原因：双重模式的执行提供保护手段，可以防止操作系统和用户程序收到错误应用程序的影响。双重模式将可能引起损害机器的指令作为特权指令,并且硬件只有在内核模式下才允许执行特权指令；用户态下试图执行特权指令时硬件认为该指令非法并且不执行，并将以陷阱形式通知操作系统。

4. 分析 Monolithic 结构，层次化结构，模块化结构和微内核结构的优劣。

【单片结构（简单结构）】：这一类操作系统**缺乏明确定义的结构**。例如 MS-DOS 最初是小的、简单的、功能有限的系统，利用最小空间提供最多功能，因此没用被仔细地分成模块。在这种结构中，系统并没有很好的区分功能的接口和层次，应用程序能够访问基本的 I/O 程序并直接写道显示器和磁盘驱动，这种自由**易使系统遭受错误或是恶意程序的伤害**，因此用户程序出错会导致整个系统的崩溃。最初的 UNIX 也为单片结构，内核提供了大量功能，这种单片结构使得 UNIX **难以实现和设计**。单片结构也有其独特的优势：系统调用接口和内核通信的**开销非常小**。

【层次化结构】：在适当的硬件支持下，操作系统可分成许多块，这些块与单片结构相比更小且更合适。对于开发人员来说，能有更多自由，可采用自顶向下的方法改变系统内部工作或创建模块操作系统。分层结构保证程序接口不变的前提下允许程序员自由实现底层程序。在这种结构下，操作系统分成若干层：最低层为硬件，层数为 0，最高层为用户接口，层数为 N。层 M 可调用更低层的操作。这种方法**简化了构造和调试**，而且**简化了系统的调试与验证**。只要从底层开始往上层层验证，某层出现错误则该层有错，因此**操作系统的设计得到简化**。然而，分层结构也有不足：**难以合理定义各层**，且与其他方法相比**效率较差**。因为

当用户程序在最外层执行时，它每层都为系统调用增加额外开销，导致系统调用需要执行更长的时间。

【模块化结构】：现代操作系统一般采用可加载的内核模块。内核提供核心服务，其他服务可在内核运行时动态实现。由于任何模块都可以调用任何其他模块，这种结构比分层系统更加**灵活**。由于模块无需调用消息传递来进行通信，它比微内核方法更有效。

【微内核结构】：这种结构从内核中删除所以不必要的部件，而将它们当作系统级与用户级的程序来实现。这种结构**便于扩展操作系统**。所有新服务可在用户空间内增加，而不需要修改内核。就算要修改也是小修改。这种结构也提供了**更好的安全性和可靠性**，如果一个服务出错，那么操作系统的其他部分并不受影响。同时，微内核结构由于增加了系统功能的开销，微内核的**性能会受损。**

5. 举例说明采用机制与策略分离的设计原因。

机制决定**如何做**，策略决定**做什么**。策略与机制分离对于确保灵活性至关重要。策略需要随时间地点而改变，如果不将机制与策略分离，最坏情况下每次策略的改变可能都需要改变底层机制。因此对策略不敏感的通用机制更为可取，这样策略的改变只需要重新定义一些系统参数。比如定时器，这是一种保护 CPU 的机制，同时为某个特定用户应将定时器设置成多长时间就是策略问题。如果机制与策略不分离，那么对

多位用户设定计时器时间时都会设计机制的底层改变，将会影响运行速度。