

Lecture 1: 知识 - 操作系统概述

大纲

第一章：操作系统概述

- (一)操作系统的基本概念
- (二)操作系统的发展历程
- (三)程序运行环境
 - 1、CPU 运行模式：内核模式，用户模式。
 - 2、中断和异常的处理
 - 3、系统调用
 - 4、程序的链接与装入
 - 5、程序运行时内存映像与地址空间
- (四)操作系统结构：分层，极块化，宏内核，锅内核，外核。
- (五)操作系统引导
- (六)虚拟机

操作系统的基本概念

功能和目标

概念：作为系统资源的管理者

- 提供的功能：
 - 处理机管理（CPU 调度）
 - 进程控制、进程同步、进程通信、死锁处理、处理机调度
 - 存储器管理
 - 提高内存利用率，内存的分配与回收、地址映射、内存保护与共享、内存扩充
 - 文件管理
 - 设备管理（驱动）
- 目标：
 - 安全，高效

概念：作为用户和计算机硬件之间的接口

- 提供的功能：
 - 命令接口
 - 联机命令接口：交互式命令接口
 - 脱机命令接口
 - 程序接口（使用 `syscall`）
 - 系统调用：程序 = 状态机 = 计算 -> `syscall` -> 计算 -> `syscall`
 - `GUI`（图形用户界面）

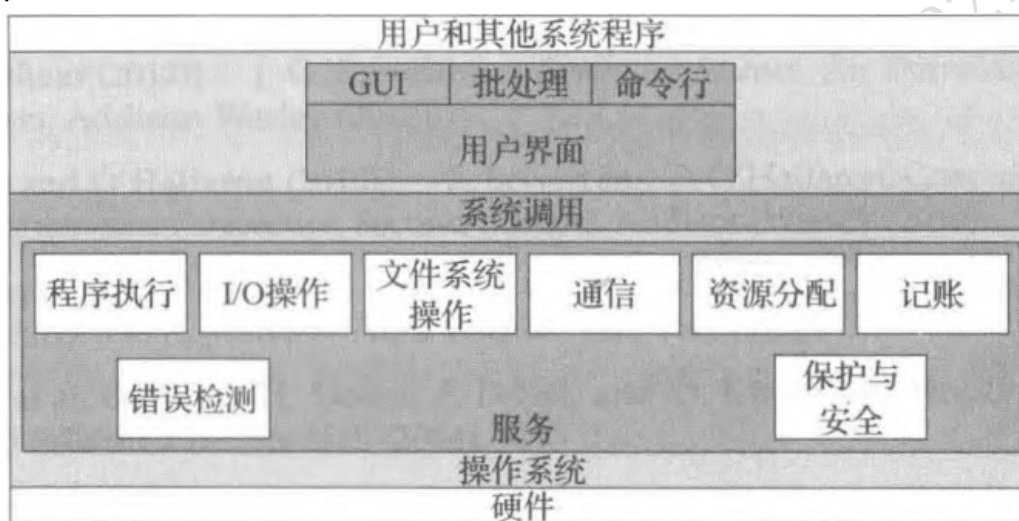
- 目标：
 - 方便用户使用

概念： 作为最接近硬件的层次

- 目标：
 - 实现对硬件机器的拓展
- 概念：
 - 没有任何软件支持的计算机称为裸机
 - 在裸机上安装的操作系统，可以提供资源管理功能和方便用户的服务功能，将裸机改造成功能更强，使用更方便的机器

补充： 从服务的视角

- 图示：



操作系统的特征

概念： 并发与并行

- 并发：
 - 指两个或多个事件在同一时间间隔内发生；
 - 这些事件**宏观上是同时发生的，但微观上是交替发生的**；
- 并行：
 - 指两个或多个事件在同一时刻同时发生；
- 真题：2009年
 - 单处理机系统：
 - 在单处理机系统中，单处理机系统只有一个处理机，因此只有一条指令流水线，每个时钟周期只能完成一条指令，所以**不能并行**执行多个进程，只能实现**并发**执行；
 - 单处理机下，可并行的设备：
 - 处理机与设备之间是并行工作的；
 - 处理机与通道之间是并行工作的；
 - 设备与设备之间是并行工作的；
 - 注意：

- 单处理机下，进程和进程之间不能并行工作
- 真题：2018年
 - 在多任务操作系统中：
 - “并行”指的是多个任务之间并行，在多任务操作系统中，CPU 可以与外设并行工作，CPU 可以与内存并发工作；
 - 注意：单核 CPU 的计算机也可以多任务工作

概念：共享

- 共享：
 - 即资源共享，是指系统中的资源可供内存中多个并发执行的进程共同使用；
- 分类：两种共享方式
 - 互斥共享方式：
 - 系统中的某些资源，虽然可以提供给多个进程使用，但一个时间段内只允许一个进程访问该资源；
 - 同时共享方式：
 - 系统中的某些资源，允许一个时间段内由多个进程同时，对它们进行访问
- 注意：
 - 所谓的同时共享往往是宏观上的，而在微观上，这些进程可能是交替地对资源进行访问（即分时共享）
- 重点：
 - 并发性和共享性互为存在条件
 - 并发和共享是操作系统的最基本特征
 - 如果失去并发性，则当前 OS 当中只能有一个进程进行执行，此时共享性也就没有存在的意义；
 - 如果失去了共享性，则当前 OS 当中多个程序也不能并发进入某区域；
- 真题：2018年
 - 在多任务操作系统中，因同时有多个任务执行，因此要对共享资源进行保护；

概念：虚拟

- 概念：
 - 指把一个物理上的实体变为若干个逻辑上的对应物；
 - 物理实体（前者）是实际存在的，而逻辑上对应（后者）是用户感受到的；
- 分类：虚拟技术
 - 空分复用技术
 - 如：虚拟存储器技术
 - 时分复用技术
 - 如：虚拟处理器
 - 没有并发性，虚拟性也就没有存在意义

概念：异步

- 在多道程序环境下，允许多个程序并发执行，但由于资源有限，进程的执行不是一贯到底的，而是走走停停，以不可预知的速度向前推进，这就是进程的异步性；
- 只有系统有了并发性，才有可能导致异步性；

OS 的发展与分类

概念：发展阶段

- 1. 手工操作阶段
 - 用户独占全机，人机速度矛盾导致资源利用率极低（大部分时间在空闲 OR 等待 IO 的状态）
- 2. 单道批处理阶段
 - 别名：
 - 单道程序系统；
 - 概念：
 - 引入脱机输入/输出技术（用外围机，把程序提前存到磁带当中），并监督程序负责控制作业的输入，输出；
 - 比如磁带机编写好程序，自动的去读取数据；
- 3. 多道批处理系统（重点）
 - 别名：
 - 多道程序系统；
 - 概念：
 - 正式诞生操作系统，用于支持多道程序的并发运行；
 - 每次往内存中读入多道程序，依次执行多道程序；
 - 在多道批处理系统中，各程序轮流使用 CPU，程序的执行过程存在不确定性；
 - 主要优点：
 - 真题：2017
 - 相比于单道程序系统，多道程序系统的主要优点：
 - 1. CPU 利用率高；
 - 2. 多道程序可以并发执行，共享计算机资源 -> 因此可以和 IO 设备并行工作 - > IO 设备利用率高；
 - 3. 资源利用率大幅提升，CPU 和其他资源更能保持“忙碌”状态，系统吞吐量增大；
 - 注意：多道程序系统相比于单道程序系统并没有减少系统开销；
 - 多道程序系统与 IO 设备
 - 真题：2016年
 - 批处理系统不能和用户交互，因为其执行逻辑是：用户向系统提交若干个作业，这些作业位于外存中，等待系统调度，逐一调入内存执行。
 - 使用中断技术后，可以使得多道批处理系统的 IO 设备可以与 CPU 并行工作
 - 注意：是完全同时工作，因为多道程序可以并发执行，因此 CPU 可以在忙一个进程的同时、把 IO 的进程挂在后台，然后 IO 设备本身可以和 CPU 并行的工作；
 - 多道程序系统与虚拟存储管理
 - 真题：2022年

- 早期的多道批处理操作系统会将所有进程的数据全部调入主存，再让多道程序并发执行；
- 因此即使不支持虚拟存储管理，也能实现多道程序并发；
- 4. 分时操作系统
 - 概念：
 - 计算机以**时间片**为单位轮流为各个用户/作业服务，各个用户可通过终端与计算机进行交互
 - 主要优点：
 - 1. 用户请求可以被即时响应，**解决了人机交互问题**；
 - 2. 允许多个用户同时使用一台计算机，并且用户对计算机的操作相互独立，感受不到别人的存在；
 - 主要缺点：
 - 不能优先处理一些紧急任务；
 - 操作系统对各个用户/作业都是完全公平的，循环地为每个用户作业服务一个时间片，不区分任务的紧急性；
- 5. 实时操作系统
 - 概念：
 - **真题**：2022年
 - 现代操作系统都是多任务的，允许用户把程序分为若干个任务，使它们并发执行；
 - 在实时操作系统的控制下，计算机系统接收到外部信号后及时进行处理，并且要在严格的时限内处理完事件；
 - 优点：
 - 实时操作系统的主要特点是**及时性和可靠性**；
 - 能够优先响应一些紧急任务，某些紧急任务不需时间片排队；
 - 两种操作系统
 - 1. 硬实时操作系统：必须在绝对严格的规定时间内完成处理；
 - 2. 软实时操作系统：能接受偶尔违反时间规定；
- 6. 其他几种操作系统（了解）
 - 网络操作系统
 - 是伴随着计算机网络的发展而诞生的，能把网络中各个计算机有机地结合起来，实现数据传送等功能，实现网络中各种资源的共享（如文件共享）和各台计算机之间的通信；
 - 分布式操作系统
 - 特点
 - 分布性和并行性
 - 概念
 - 系统中的各计算机地位相同，任何工作都可以分布在这些计算机上，由它们并行，协同完成这个任务；
 - 个人计算机操作系统
 - 如：Windows XP, MacOS, 方便个人使用；

程序运行环境

CPU 运行模式：内核模式，用户模式

指令分类与上下文状态

概念：指令与上下文状态

- 指令分类
 - 特权指令
 - 如：内存清零指令
 - 不允许用户程序使用
 - 真题：2014年
 - 关中断指令是特权指令；
 - 真题：2021年
 - IO 操作的指令为特权指令；
 - 访问程序状态的指令为特权指令；
 - 存储特殊寄存器的指令；
 - 非特权指令
 - 如：普通的运算指令
 - 真题：2014年
 - trap 指令可以在用户态执行，负责将用户态转化为内核态；
 - 跳转指令和压栈指令均可以在用户态执行；
 - 注意：需要区分题干中 执行 和 导致 的概念
 - 除法指令可能会 导致 系统从用户态变成内核态，但除法指令本身是可以在用户态执行的；
- 上下文状态
 - 概念：
 - CPU 根据当前处于内核态还是用户态，判断当前是否执行特权指令；
 - 分类：
 - 用户态（目态）：此时 CPU 只能执行非特权指令
 - 内核态（管态、核心态）：特权指令和非特权指令都可执行
 - 实现
 - 用程序状态寄存器（PSW）中的某标志位来识别当前处理器处于什么状态
 - 如：1 为用户态，0 为核心态
 - 注意：
 - 真题：2012年
 - 注意区分题干中所提的某一事件，是不能发生在某一状态，还是不涉及某一状态；
 - 比如：系统调用、外部中断以及缺页都会使得当前程序陷入到内核态，但是它们可以在用户态发生，即用户态下可以发生系统调用、外部中断以及缺页的事件；
 - 而进程切换是在内核态执行，并且进程切换事件的发生不可以在用户态，其完全使用操作系统控制的；

面试题：用户态和内核态的区别？

- 内核态和用户态是操作系统中的两种运行模式。它们的主要区别在于权限和可执行的操作：
 - 内核态 (Kernel Mode): 在内核态下, CPU 可以执行所有的指令和访问所有的硬件资源。这种模式下的操作具有更高的权限, 主要用于操作系统内核的运行。
 - 用户态 (User Mode): 在用户态下, CPU 只能执行部分指令集, 无法直接访问硬件资源。这种模式下的操作权限较低, 主要用于运行用户程序。
 - 内核态的底层操作主要包括: 内存管理、进程管理、设备驱动程序控制、系统调用等。这些操作涉及到操作系统的核心功能, 需要较高的权限来执行。
- 分为内核态和用户态的原因主要有以下几点:
 - 安全性: 通过对权限的划分, 用户程序无法直接访问硬件资源, 从而避免了恶意程序对系统资源的破坏。
 - 稳定性: 用户态程序出现问题时, 不会影响到整个系统, 避免了程序故障导致系统崩溃的风险。
 - 隔离性: 内核态和用户态的划分使得操作系统内核与用户程序之间有了明确的边界, 有利于系统的模块化和维护。
 - 内核态和用户态的划分有助于保证操作系统的安全性、稳定性和易维护性。

用户态切换

概念：用户态切换

- 概念：内核态 -> 用户态
 - 概念：
 1. 更改标志寄存器：
 - 执行一条特权指令：修改 PSW 的标志位为“用户态”，这个动作意味着操作系统将主动让出 CPU 使用权；
 2. 指向新指令：
 - 更改指令寄存器的值，将其指向新的指令；
 3. 保存上下文：
 - 指令进入解码器进行解码后，其中标志位取出，检测到 PSW（或其他体系结构上的对应符号）发生更改时，将内核态的栈保存，将程序的执行流切换至用户态程序的代码地址处，恢复用户态进程的执行环境，继续进行用户态进程执行；
 4. 更多的设计：
 - CLR 中可以将进程设计为进程池、或者线程设计为线程池，同时进程池和线程池还分为用户态进程池、线程池和内核态进程池、线程池，在进行上下文切换时，只需要由系统自动激活某一个池中的进程、线程，减少上下文切换与进程开启的开销，减少了操作系统进入内核态；
 - 应用：会导致内核态到用户态的情况
 - 真题：2023年
 - 正确选项：执行系统调用完成后，会导致从内核态进入到用户态
 - 错误选项：执行 CPU 调度全部是在内核态进行，因此在执行 CPU 调度后、CPU 依然处于内核态；

- 错误选项：唤醒进程。唤醒进程也属于 CPU 调度，因此也一直处于内核态；
- 错误选项：阻塞进程的事件完成后，CPU 会从用户态切换为内核态；
- 概念：用户态 -> 内核态
 - 分析：
 - 由“中断”引发，硬件自动完成变态过程，触发中断信号意味着操作系统将强行夺回 CPU 的使用权；
 - 注意：
 - 发出系统调用请求是在用户态，而对系统调用的相应处理在内核态下进行；
 - 应用：会导致用户态到内核态的情况
 - 真题：2013年
 - 1. 中断的产生：
 - 比如整数除以零会发生中断，中断必须在内核态进行；
 - 比如软中断可以触发一个中断，并跳转到相应的中断处理程序；
 - 比如进行内存访存的时候，可能会产生缺页异常，进而导致变态；
 - 2. 系统调用的执行：
 - read 系统调用将用户态切换为内核态
 - 注意：
 - 调用系统的库函数不一定会导致变态，sin() 函数调用是在用户态对库函数的调用；

中断和异常的处理

中断的概念

概念：中断机制介绍

- 原因
 - 为了解决系统资源利用率、为实现多道程序并发执行而引入的一种技术；
 - 发生中断就意味着需要操作系统介入，开展管理工作；
 - 中断可以使 CPU 从用户态切换为核心态，使操作系统获得计算机的控制权；
 - 硬件可以（常常是通过系统总线）随时发送信号到 CPU 以触发中断；它是 OS 和硬件交互的关键部分；
- 中断过程原理：
 - 1. 当中断发生时，当中断发生后，当前运行的进程暂停运行，并由操作系统内核对中断进行处理，CPU 立即进入核心态；
 - 2. 检查中断信号的时机：
 - 对于内中断，CPU 在执行指令时会检查是否有异常发生；
 - 对于外中断，CPU 会在每个指令周期的末尾检查是否有外中断信号需要处理；
 - 3. 根据中断信号，检查中断向量表：
 - 当 CPU 检测到中断信号后，会根据中断信号的类型去查询“中断向量表”，以此来找到相应的中断处理程序在内存中的存放位置；
 - 中断将控制转移到合适的中断服务程序，该程序调用特定的中断处理程序；

- 补充：
 - 为了提高效率，部分设计中也可以绕过中断服务程序，直接通过指针表来访问到中断处理程序；指针表存储在地址的前 100 位左右位置；
- 4. 保存现场
 - 概念：程序状态字
 - 真题：2012年
 - 程序状态字（PSW）寄存器用于记录当前处理器的状态和控制指令的执行顺序，并且保留与运行程序相关的各种信息，主要作用是实现程序状态的保护和恢复；
 - 注意：中断处理与子程序调度的区别；
 - 中断处理：中断处理程序要将 PSW 保存，通用寄存器(包括数据、地址)、程序计数器 PC；
 - 子程序调用：子程序调用在进程内部执行，不会更改 PSW，需要在进程的栈空间中压入通用寄存器 (包括数据、地址)、程序计数器 PC 等信息；
 - 概念：外部中断与程序计数器
 - 真题：2015年
 - 外部中断处理过程，PC 值由中断隐指令自动保存，而通用寄存器内容由操作系统保存；
- 5. 执行中断服务
- 6. 恢复现场；
- 总结：中断操作与操作系统
 - 真题：2020年
 - 以下中断相关的操作是由操作系统完成的：
 - (1) 提供执行中断服务；
 - (2) 初始化中断向量表；
 - (3) 保存中断屏蔽字；
 - 以下中断相关的操作不是由操作系统完成的：
 - (1) 中断服务寻址；
 - (2) 关中断；
 - (3) 保存被中断程序的中断点（PC 值）；

中断分类

概念：中断分类

- 概念：内中断
 - 别称：
 - 异常
 - 概念：
 - 与当前执行的指令有关，中断信号来源于 CPU 内部
 - 举例：
 - (1) 试图在用户态下执行特权指令

- (2) 执行除法指令时发现除数为 0
- (3) 陷入指令 `trap`
- (4) 缺页；
- (5) 地址越界；
- (6) 存储保护错；

- 注意：

- 1. 大多数的教材、试卷中，“中断”特指狭义的中断，即外中断。而内中断一般称为“异常”
- 2. 缺页不属于外部中断，其属于指令执行中产生的中断。当涉及主存地址访存时，就有可能发生缺页；

- 概念：外中断

- 别称：

- 中断（狭义的中断）

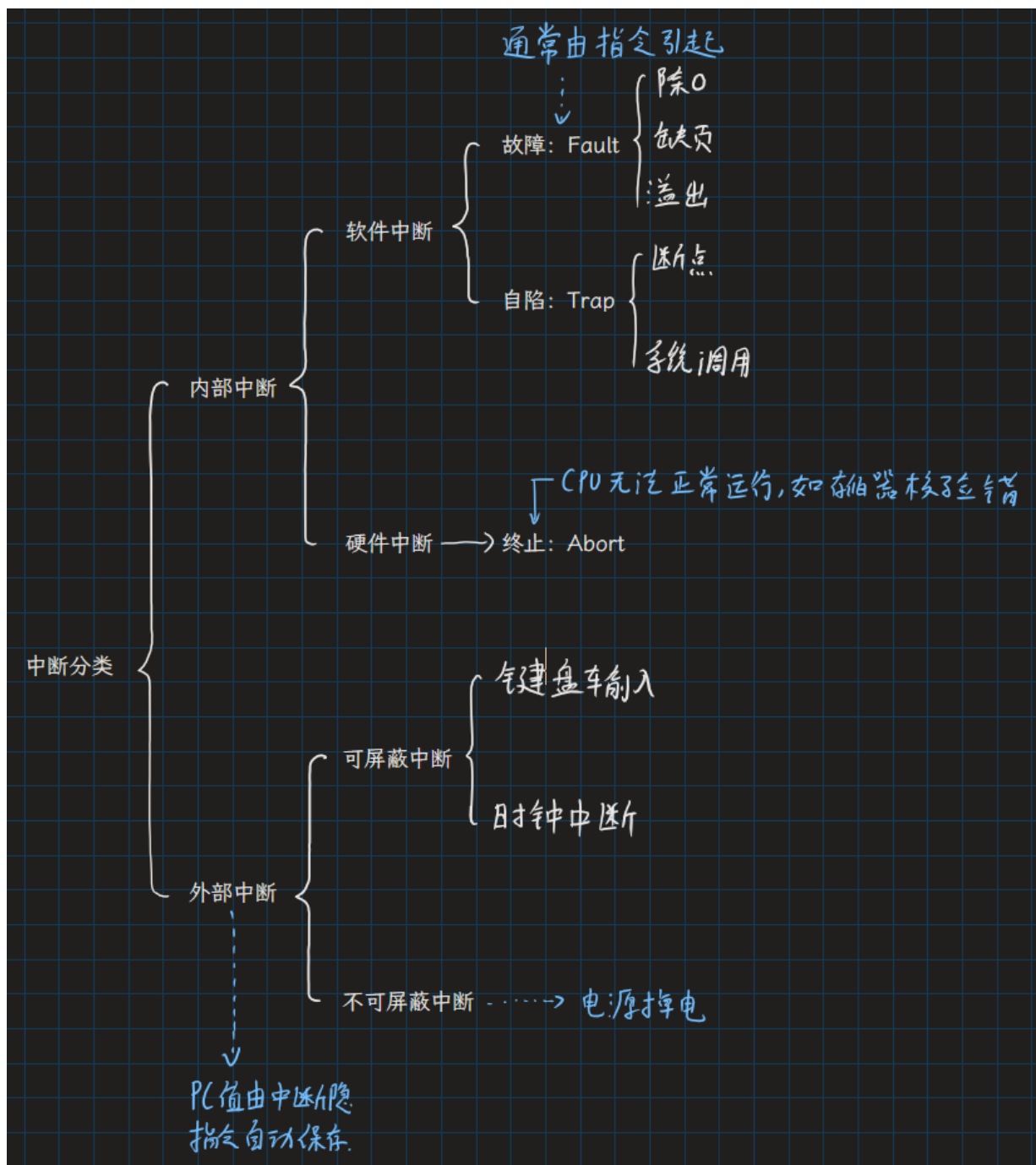
- 概念：

- 与当前**执行的指令无关**，中断信号来自于 `CPU` 外部；

- 举例：

- (1) 时钟中断（定时器到期）；
 - 时钟部件每隔一个时间片（如 `50 ms`）会给 `CPU` 发送一个时钟中断信号，此时先暂停执行的程序，进入内核程序，处理中断程序（时钟中断）；
 - 此时如果发现当前进程 1 的执行时间超过了时钟片的最大时长，则此时将程序的执行权切换为进程 2，交替进行执行；
 - (2) `IO` 中断
 - 打印机完成打印之后，发送一个外中断，表示打印已经结束，可以结束任务；
 - (3) 网络数据报到达
 - 网络数据包到达后，网络控制器会发出中断信号来通知 `CPU`，该中断事件与 `CPU` 正在执行的指令无关，属于外部中断；
 - (4) 不可屏蔽中断，如掉电

- 总结：



概念：陷入指令

概念：

- 有时候应用程序想请求操作系统内核的服务，此时会执行一条特殊的指令：**陷入指令**，该指令会引发一个内部中断信号；
- 自陷是一种事先安排的“异常”事件，通过在程序中显式地调用自陷指令触发异常，执行“**陷入指令**”意味着应用程序主动地将 **CPU** 控制权还给操作系统内核，**系统调用**就是通过陷入指令完成的；

应用

真题：2020年

- 1. 系统调用：用于在用户态调用操作系统内核程序，如系统调用，条件陷阱指令；
- 2. 程序断点：可通过“自陷”方式实现程序调试的“断点设置”功能，并且“自陷”指令执行完毕后，会返回自陷指令的**下一条指令**执行（注意不是当前指令，而是下一条）；

注意：

- 陷入指令是在**用户态**执行的，执行陷入指令之后立即引发一个内中断，使 **CPU** 进入核心态
- 补充：陷入指令与访管指令
 - 从指令集的角度定义，或者说从硬件角度 (**cpu** 状态)，访管指令，是用户程序自愿进管的指令
 - 进管同时也意味着程序放弃 **cpu** 的控制权
 - 该指令本身属于非特权指令，可在用户态执行，执行后进入核心态；
 - 核心态是通过 **cpu** 置相应标志表明当前处于核心态，**cpu** 进入核心态后，可以执行指令集中的所有指令（包括特权指令和非特权指令，但不执行访管指令）；

概念：中断原语

- 陷入： **trap**
- 故障： **fault**
 - 比如缺页故障
- 终止： **abort**
 - 由致命错误引起，内核程序无法修复该错误，因此一般不再将 **CPU** 使用权还给引发终止的应用程序，而是直接终止该应用程序；
 - 如：整数除 0、非法使用特权指令

系统调用

系统调用的概念与分类

概念：系统调用

- 介绍：
 - 是操作系统提供给应用程序（程序员/编程人员）使用的接口；
 - 一种可供内核外程序调用的系统指令，应用程序可以发出系统调用请求来获得操作系统的服务；
- 补充：
 - 这些调用通常以 **C** 或 **C++** 编写，对某些底层任务（如需直接访问硬件的任务），也可能以汇编语言指令编写
- 作用：
 - 应用程序通过系统调用请求操作系统的服务；
 - 系统中的**各种共享资源**都由操作系统统一掌管，因此在用户程序中，凡是与资源有关的操作（如存储分配，**I/O** 操作，文件管理等），都必须通过系统调用的方式向操作系统提出服务请求；
 - 这样可以保证系统的**稳定性和安全性**，防止用户进行非法操作；
- 举例：
 - **UNIX** 下的 **read()** API

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count)
```

返回值 函数名称 参数

概念：系统调用分类

- 分类：

- 1. 设备管理：
 - 完成设备的请求/释放/启动等功能
- 2. 文件管理：
 - 完成文件的读/写/创建/删除等功能
- 3. 进程控制：
 - 完成进程的创建/撤销/阻塞/唤醒等功能
- 4. 通信（进程通信）：
 - 完成进程之间的消息传递/信号传递等功能；
 - 两种模式：
 - 1. 消息传递；
 - 2. 共享内存；
- 5. 内存管理：
 - 完成内存的分配/回收等功能
- 其他：
 - 6. 信息维护：系统调用可以返回系统的其他信息，如当前用户数、操作系统版本、内存或磁盘的可用量等；
 - 7. 保护：保护提供控制访问计算机的系统资源的机制。包括 `set_permission()` 和 `get_permission()`，用于设置资源（如文件和磁盘）权限；

- 图示：

进程管理

- fork, execve, exit

内存管理

- mmap, munmap, mprotect

文件管理

- open, close, read, write, lseek, dup

进程间通信

- pipe, wait

系统调用与库函数

概念： 库函数和系统调用的关系

- 概念：
 - 操作系统内核对外面提供的 API 就是 Syscall，而当中和用户交互的就是 Shell；
 - 而在 Shell 的外面，OS 提供了一个框架性程序，帮助开发者实现应用程序（因为没有人愿意在没有库函数的情况下，直接使用汇编、系统调用来制作应用程序），这个框架性的程序就是库函数；
- 分析：系统调用与库函数的区别
 - 不涉及系统调用的库函数：
 - 如：取绝对值的函数，计算三角函数的函数；
 - 涉及系统调用的库函数：
 - 如：创建一个新文件的函数
- 补充：如何在编程语言上使用系统调用：
 - Windows 系统：WindowsAPI；
 - POSIX 系统：POSIX API（这包括几乎所有版本的 UNIX、Linux 和 MacOSX）；
 - Java 虚拟机：Java API；
 - C 语言：在系统调用之上构建可以普遍使用的标准库：libc
- 总结：基于 API 使用系统调用的好处；
 - 好处 1：程序具有更好的可移植性；
 - 任何支持同一 API 的系统上编译并执行；
 - 好处 2：易用性；
 - 实际系统调用比 API 更为注重细节且更加难用；

分析： 判断事件是否通过系统调用完成

- 可通过系统调用完成：

- 参考：上面的五大分类，设备管理、文件管理、进程控制、进程通信、内存管理；
- 参考：Linux 的常见命令；
- 举例：进程创建可以通过系统调用完成；
- 不通过系统调用完成的操作：
 - 常见第一类：库函数完成的、不需要系统介入操作
 - 常见第二类：完全由操作系统完成的操作
 - 真题：2021年 页置换相关
 - 页置换不通过系统调用完成；
 - 页置换是操作系统内核管理内存时使用的一种机制，它不需要用户级别的系统调用，由操作系统内核在内存管理过程中自动执行的；
 - 用户级别的程序通常通过系统调用请求内存，例如在 Linux 系统中使用 malloc、mmap 等函数，但页置换的具体实现细节对用户程序是不透明的，用户程序不需要直接进行页置换操作，操作系统会负责处理内存页的加载和替换，以确保程序的运行效率和系统资源的有效利用；
 - 真题：2021年 进程调度
 - 进程调度是在操作系统内部自行完成的，此过程对用户完全隐藏，不能通过系统调用操作；

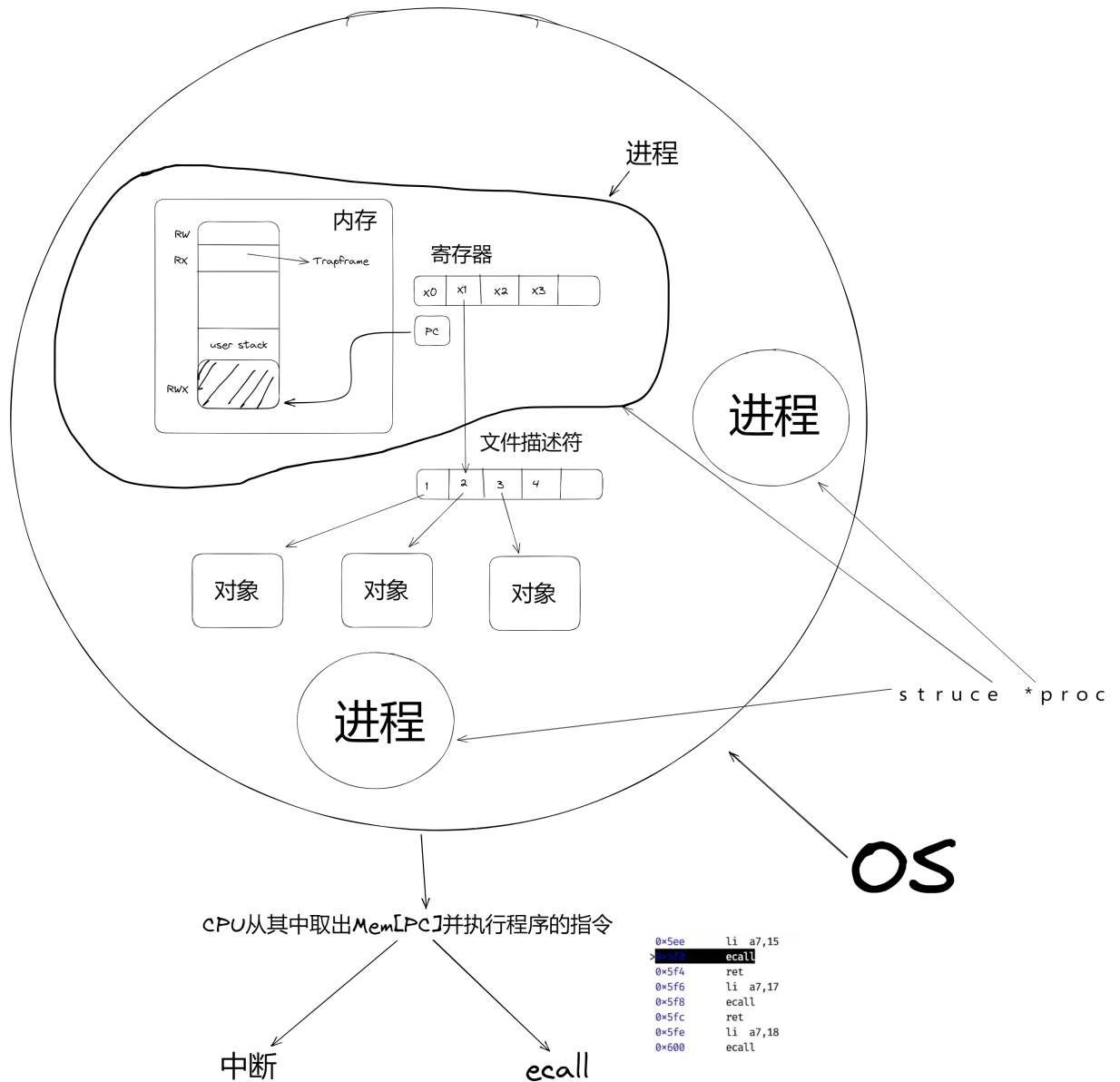
系统调用的过程

概念：系统调用背后的过程

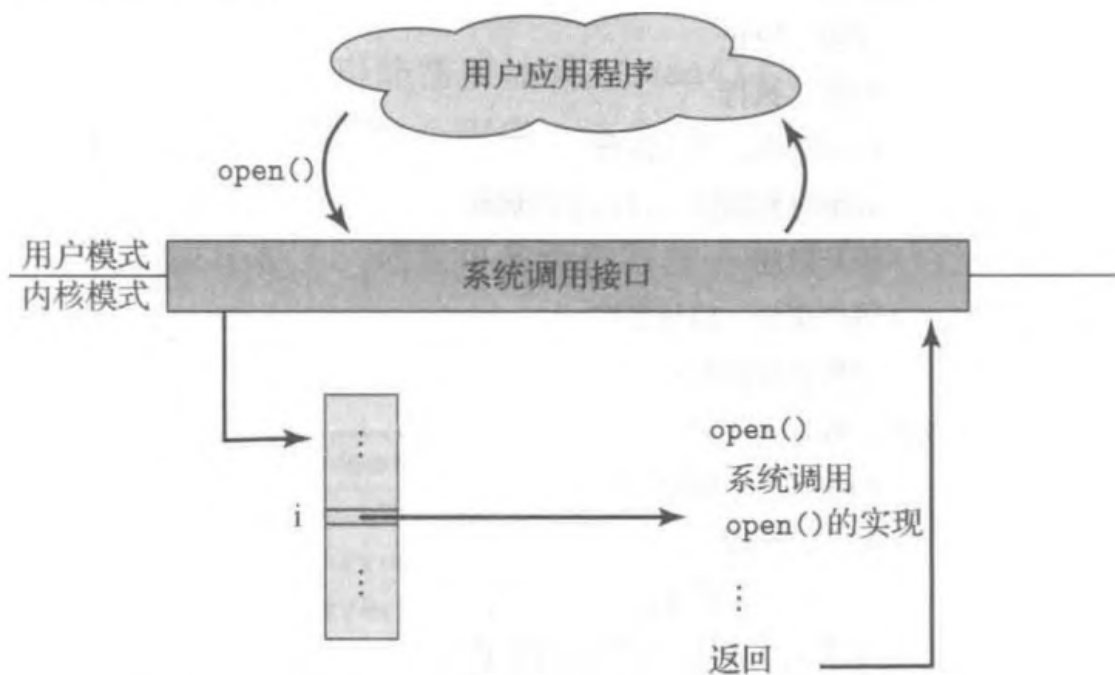
- 过程：
 - 0. 用户程序通过特定的系统调用接口
 - 如在 Linux 系统中使用 int 0x80 指令或 syscall 指令发起调用；
 - 1. 触发中断或异常，比如执行陷入指令，导致控制权从用户态转移到内核态；
 - 执行中断时，由硬件负责保存断点、关中断，并将 CPU 模式改为内核态；
 - 因此，保存断点、变态这三个操作不是由操纵系统完成的，而是硬件完成；
 - 真题：2022年
 - 注意：将 CPU 模式改为内核态这一操作不是由操作系统完成的。CPU 进入内核态是通过中断实现的，并且是在中断的硬件阶段由硬件完成的变态；
 - 2. 传递系统调用参数，保存当前程序上下文状态；
 - 比如保存用户的通用寄存器的值；
 - 在 OS 当中有一个名字为 proc 的结构体链表，当中存储了所有的指向进程的指针，在 OS 视角可以来统一的管理他们；
 - OS 通过调用这些指针，就可以切换不同当前寄存器状态、执行不同的程序，执行不同的调度；
 - 图示见下方；
 - 3. 调用、执行系统调用服务例程，当前 CPU 的执行流切换至内核态的进程中；
 - 4. 恢复上下文环境，返回用户程序，切换回用户态；
- 真题：2017年
 - 系统调用的过程：

- (1) 传递系统调用参数，保存当前用户态进程的上下文环境，比如系统根据进程上下文环境中，存储寄存器数值；
- (2) 执行陷入指令，OS 进入内核态；
- (3) 执行响应的服务程序；
 - 真题：2019年
 - 执行系统调用服务程序时，CPU 处于内核态；
- (4) 返回用户态；

- 图示：上下文状态的保存



- 图示：



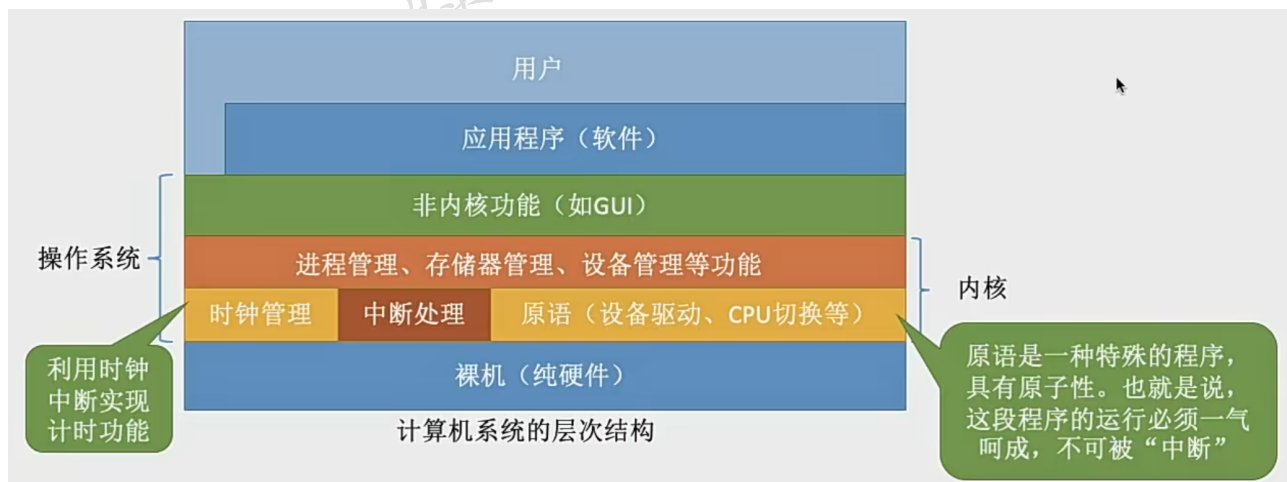
• 注意

- 1. 陷入指令是在用户态执行的，执行陷入指令之后立即引发一个内中断，从而 CPU 进入核心态；
- 2. 发出系统调用请求是在用户态，而对系统调用的相应处理在核心态下进行；
- 3. 陷入指令是唯一一个**只能在用户态执行**，而不可在核心态执行的指令；

操作系统结构

操作系统的内核

概念：体系结构示意图



概念：层次结构构成

- 概念：内核
 - 是计算机上配置的底层软件，是操作系统最基本，最核心的部分；
- 概念：内核程序
 - 实现操作系统内核功能的那些程序就是内核程序；
- 概念：操作系统内核模块

- 时钟管理：
 - 实现计时功能
- 中断处理：
 - 负责实现中断机制
- 原语
 - 是一种特殊的程序
 - 处于操作系统最底层，是最接近硬件的部分
 - 这种**程序的运行具有原子性**：其运行只能一气呵成，不可中断，即使有中断信号，也需要先执行完原语才可以进行中断；
 - **运行时间较短，调用频繁**；
- 其他模块（有些操作系统不含有或更多）：
 - 对系统资源进行管理的功能：
 - 进程管理
 - 存储器管理
 - 设备管理

操作系统的体系结构

总结

	特性、思想	优点	缺点
分层结构	内核分多层，每层可单向调用更低一层提供的接口	1. 便于调试和验证，自底向上逐层调试验证 2. 易扩充和易维护，各层之间调用接口清晰固定	1. 仅可调用相邻低层，难以合理定义各层的边界 2. 效率低，不可跨层调用，系统调用执行时间长
模块化	将内核划分为多个模块，各模块之间相互协作。 内核 = 主模块 + 可加载内核模块 主模块：只负责核心功能，如进程调度、内存管理 可加载内核模块：可以动态加载新模块到内核，而无需重新编译整个内核	1. 模块间逻辑清晰易于维护，确定模块间接口后即可多模块同时开发 2. 支持动态加载新的内核模块（如：安装设备驱动程序、安装新的文件系统模块到内核），增强OS适应性 3. 任何模块都可以直接调用其他模块，无需采用消息传递进行通信，效率高	1. 模块间的接口定义未必合理、实用 2. 模块间相互依赖，更难调试和验证
宏内核（大内核）	所有的系统功能都放在内核里（大内核结构的OS通常也采用了“模块化”的设计思想）	1. 性能高，内核内部各种功能都可以直接相互调用	1. 内核庞大功能复杂，难以维护 2. 大内核中某个功能模块出错，就可能导致整个系统崩溃
微内核	只把中断、原语、进程通信等最核心的功能放入内核，进程管理、文件管理、设备管理等功能以用户进程的形式运行在用户态	1. 内核小功能少、易于维护，内核可靠性高 2. 内核外的某个功能模块出错不会导致整个系统崩溃	1. 性能低，需要频繁的切换用户态/核心态。用户态下的各功能模块不可以直接相互调用，只能通过内核的“消息传递”来间接通信 2. 用户态下的各功能模块不可以直接相互调用，只能通过内核的“消息传递”来间接通信
外核（exokernel）	内核负责进程调度、进程通信等功能，外核负责为用户进程分配未经抽象的硬件资源，且由外核负责保证资源使用安全	1. 外核可直接给用户进程分配“不虚拟、不抽象”的硬件资源，使用户进程可以更灵活的使用硬件资源 2. 减少了虚拟硬件资源的“映射层”，提升效率	1. 降低了系统的一致性 2. 使系统变得更复杂

大内核与微内核

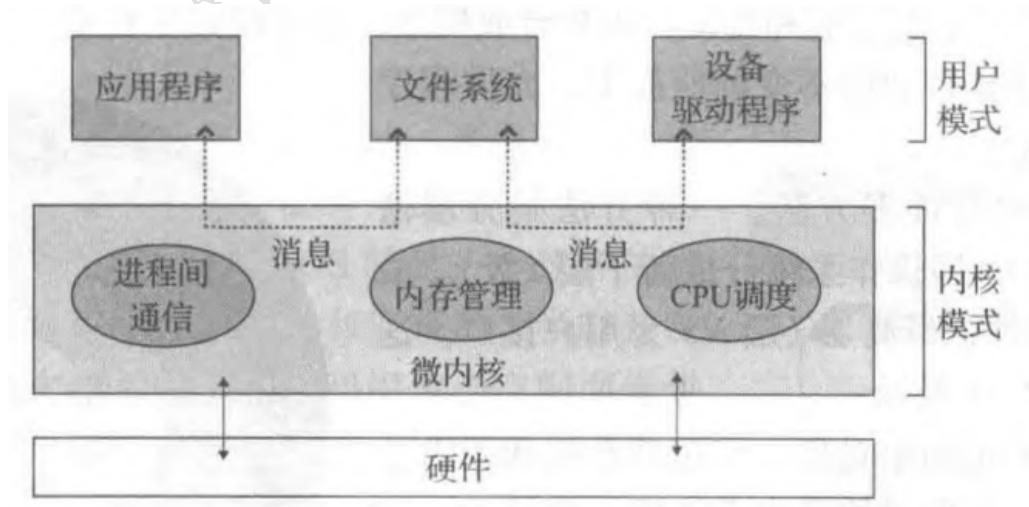
概念：大内核

- 概念：
 - 将操作系统的主要功能模块都作为系统内核，**运行在核心态**；
 - 比如进程管理、存储管理、设备管理、时钟管理、中断处理、原语；
- 优点：

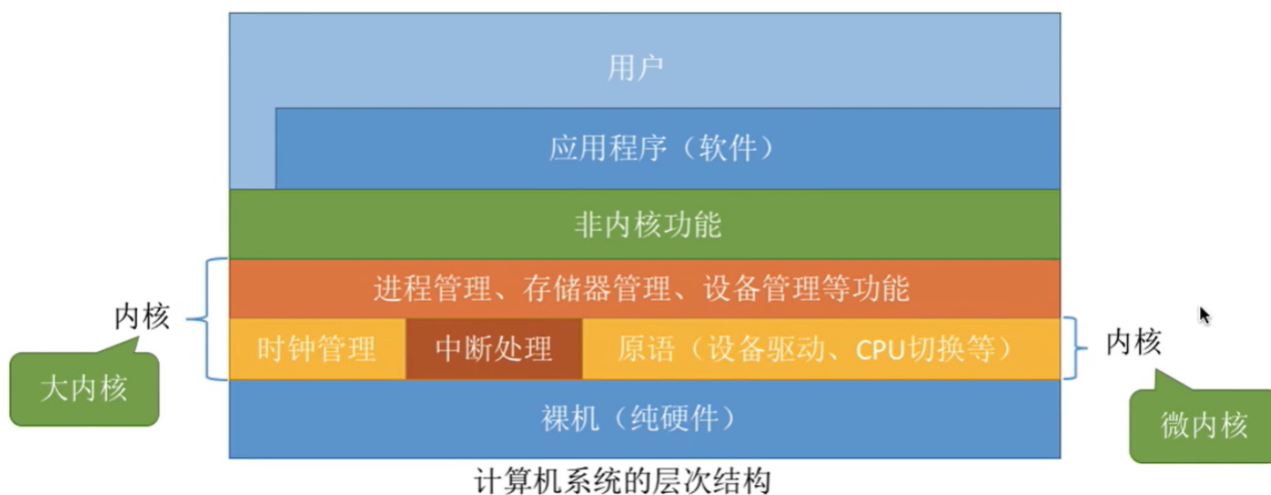
- 高性能：因为原本程序执行可能会进行 **更多的上下文切换**（CPU 状态的转换），大内核下因为功能都在内核态内，因此减少了上下文切换的数量；
- 缺点：
 - 内核代码庞大，结构混乱，难以维护；

概念：微内核

- 概念：
 - 从内核中删除所有不必要的部件，而将它们当作系统级与用户级的程序来实现；
 - 通常微内核会提供**最小化的进程管理、内存管理以及通信**的功能；
- 优点：
 1. 内核功能少，结构清晰，方便维护；
 2. 便于扩展操作系统；
 - 新的服务可以在用户空间增加，而无需修改内核；
 3. 更好的安全性和可靠性；
 - 大部分的服务是以用户进程而非内核进程来运行。
- 缺点：
 - 需要频繁地在核心态和用户态之间切换，**性能低**；
- 真题：2023年
 - 与宏内核操作系统相比，微内核具有：
 - （1）较高的可靠性；
 - （2）较高的安全性；
 - （3）较高的可扩展性；
 - 其缺点有：
 - 更低的性能；
 - 如下图所示，由于采用微内核，进一步分功能移至用户模式，因此需要使用 IPC（进程间通信）机制来进行内进程间通信，这带来了额外的开销；
- 图示：



分析：两种内核对比



分层结构、模块化与外核

概念：分层结构

- 概念：
 - 分成不同的层次，最底层是硬件，最高层是用户接口；
 - 每层只可以调用相邻低层；
- 优点：
 - 1. 便于功能的调试与验证；
 - 第一层可先调试而无需考虑系统其他部分，这是因为根据定义，它只使用了基本硬件(假设是正确的)，以便实现功能；
 - 一旦第一层调试后可认为它能正确运行，这样就可调试第二层，如此向上。如果在调试某层时发现错误，那么错误应在这层上，这是因为其低层都已调试好了。因此，系统的设计和实现得以简化；
 - 2. 易于扩充和易维护，各层次之间的调用接口清晰；
- 缺点：
 - 1. 每层只可以调用相邻低层，难以合理的定义各层之间的边界；
 - 2. **效率低**，不能跨层调用，系统如果调用一个底层的指令，此时也得需要穿过所有层次；
 - 当一个用户程序执行 IO 操作时，它执行系统调用并陷入 IO 层，IO 层会调用内存管理层，内存管理层接着调用 CPU 调度层最后传递到硬件；
 - 在每一层，参数可能会被修改，数据可能需要传递，等等。每层都为系统调用增加额外开销；

概念：模块化

- 组成：
 - 内核 = 主模块 + 可加载模块；
 - 主模块：
 - 进程模块、内存模块；
 - 主模块只有核心功能，并知道如何加载模块以及如何让模块进行通信；
 - 可加载模块：

- 比如驱动程序

- 解释:

- 内核提供核心服务, 而其他服务可在内核运行时**动态链接实现**, 添加新功能时**无需重新编译**内核, 而是通过可加载模块;
- 内核有一组核心组件, 无论在启动或运行时, 内核都可通过模块链入额外服务;
- 常见于现代 `UNIX` (`Solaris`、`Linux` 和 `Mac OS X`)以及 `Windows` 的实现;

- 优点:

- 1. 模块间**结构清晰**, 可以多个模块并行的开发;
- 2. 支持**动态加载**新的内核模块 (如: 安装设备驱动程序、安装新的文件系统模块到内核), 增强 `OS` **适应性**;
- 3. 任何模块都可以直接调用其他模块, **无需采用消息传递**进行通信, **效率高**;

- 缺点:

- 1. 模块间相互依赖、难以验证;
- 2. 模块间的定义难以清晰;

概念: 宏内核

- 概念:

- 所有的系统功能都放在内核里 (大内核结构的 `OS` 通常也采用了"模块化"的设计思想);

- 优点:

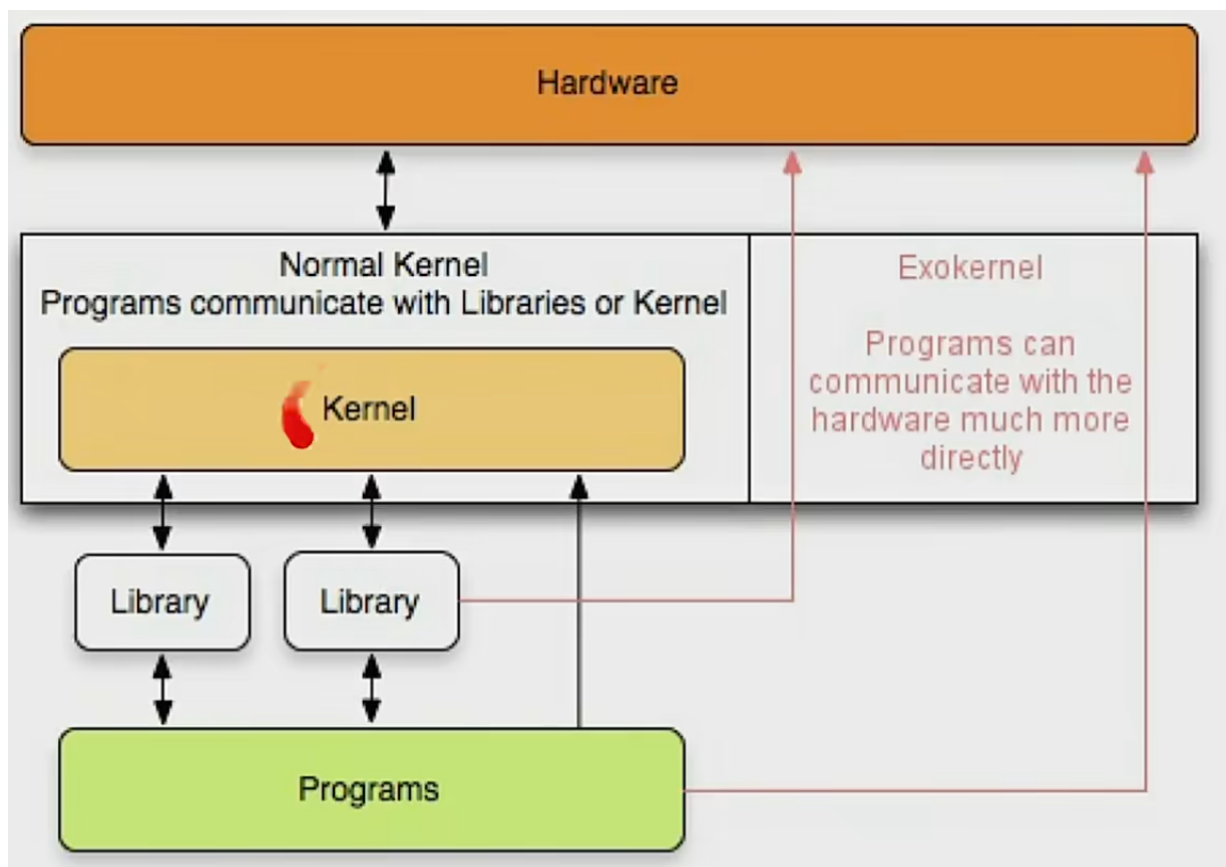
- 性能高, 内核内部各种功能都可以直接相互调用;

- 缺点:

- 1. 内核庞大功能复杂, 难以维护;
- 2. 大内核中某个功能模块出错, 就可能导致整个系统崩溃;

概念: 外核

- 图示

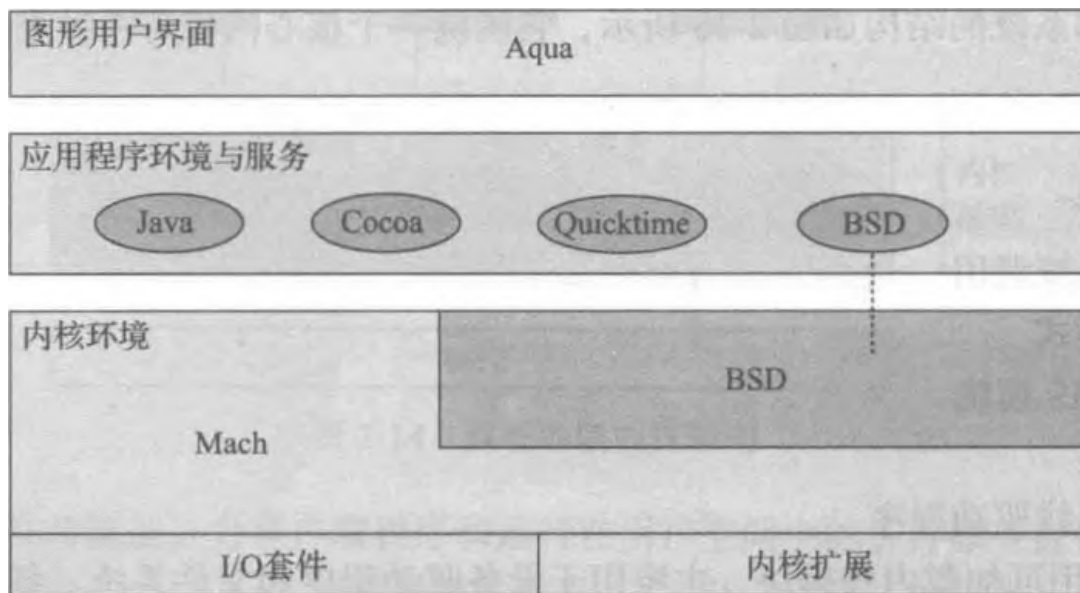


- 内外核：
 - 1. 内核负责进程调度、进程通信等功能；
 - 2. 外核负责为用户进程分配未经抽象的硬件资源，且由**外核**负责保证资源**使用安全**；
- 举例：
 - 有一些需要经常访问的硬件（内存区域），此时直接使用外核，可以直接在硬件上分配连续的物理内存的地址空间，外存负责这个部分的访问安全；
- 优点：
 - 1. 外核可直接给用户进程分配"不虚拟、不抽象"的硬件资源，使用户进程可以更灵活的使用硬件资源；
 - 2. 减少了虚拟硬件资源的"映射层"，提升效率；
- 缺点：
 - 1. 降低了系统的一致性；
 - 2. 使系统变得更复杂；

补充：其他常见内核结构

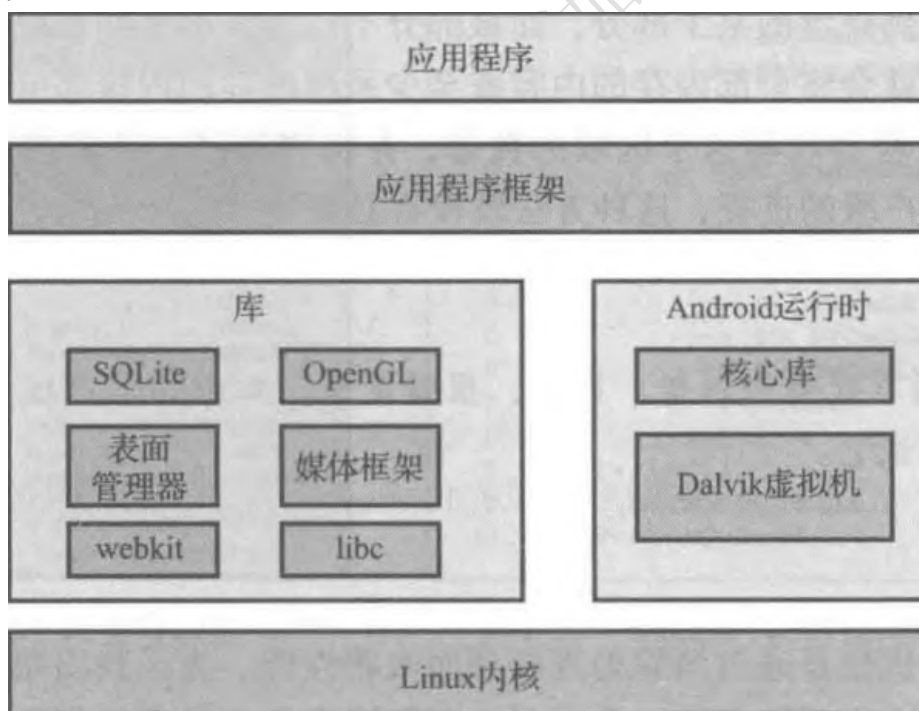
Mac OS X 结构

- Apple MacOS X 操作系统采用混合结构；
- 其中包括了：分层结构、微内核结构等；
 - Mach 为微内核结构，提供内存管理、RPC 和进程间通信等功能；
- 图示：



Android 结构

- 分析：
 - 采用分层结构；
 - 1. 底层为 **Linux** 内核，提供进程、内存以及硬件设备的驱动程序；
 - 2. **Android** 运行环境：核心库+虚拟机，虚拟机针对内存和部分移动设备进行优化；
 - 3. 库
- 图示：



操作系统中的初始化

操作系统的引导

概念：引导程序

- 引导程序代码能够定位内核，并加载到内存以开始执行；

- 有的计算机系统采用两个步骤：一个简单引导程序从磁盘上调入一个更复杂的引导程序，而后再加载内核；

概念：OS 启动执行

过程：

• 第一部分：系统初始化

• 注意：

- 系统初始化不等于操作系统初始化，操作系统的初始化需要在磁盘初始化后、选择磁盘上某个分区的操作系统后才可以进行；

• 第一步：CPU 接收到重置事件

- 如上电开机或重新启动。

• 第二步：指令寄存器加载 ROM 中的引导程序

- 以 ROM 形式存储，也称为固件（firmware）；
- 激活的 CPU 读取 ROM 中的 boot 程序，将指令寄存器指向 BIOS（基本输入/输出系统）的第一条指令，即开始执行 BIOS 指令

• 第三步：构建中断向量表

• 真题：2022年

- BIOS 程序在内存最开始的空间构建中断向量表，因为接下来的 POST（通电自检，Power Self Test）过程要用到中断功能；

• 第四步：进行通电自检，检测硬件是否有故障

- 完成 POST 后，BIOS 将控制权交给排在首位的启动设备，接着，CPU 将该设备主引导扇区的内容（主引导记录 MBR）加载到内存中；
- MBR 的作用是告诉 CPU 去硬盘的哪个主分区找操作系统；

• 第二部分：磁盘初始化

• 第一步：BIOS 将磁盘的第一块，即主引导记录读入内存

- 执行磁盘引导程序（MBR），读取硬盘中 MBR 中的数据；

• 第二步：MBR 指明分区，执行分区引导程序

- MBR 指明操作系统所在的分区，因此找到 OS 所在的分区，执行此分区中的分区引导程序；
- 由 MBR 指明操作系统所在的分区，即找含有操作系统的主分区（活动分区），并将活动分区的引导扇区的内容（分区引导记录 PBR）加载到内存加以执行；
- 活动分区的引导扇区就是活动分区的第一个扇区；
- 2. 初始化存储空间管理所用的数据结构（如位示图、空闲分区表）

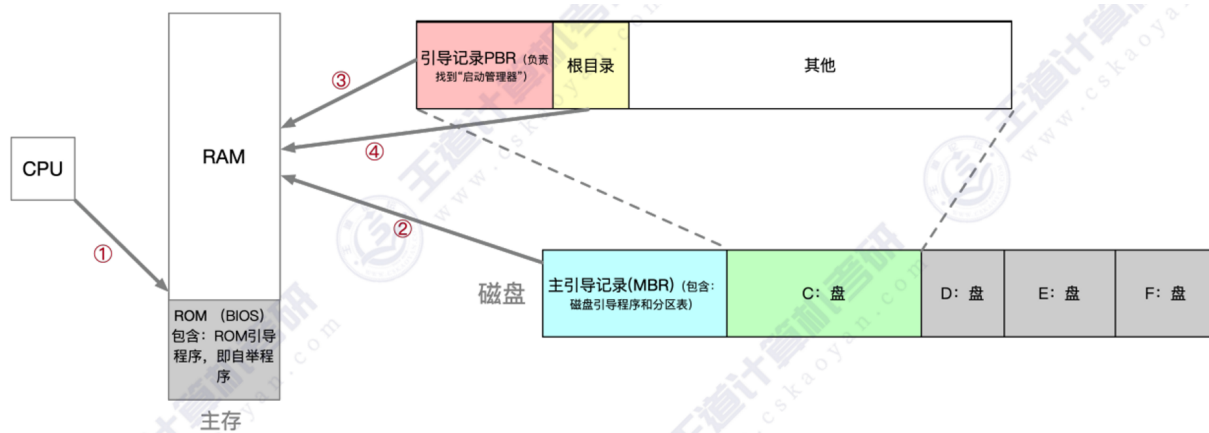
• 第三部分：操作系统初始化

• 第一步：将该分区内操作系统的初始化程序加载到内存 RAM

• 第二步：运行操作系统的初始化程序

- CPU 执行启动管理器中的启动管理程序；

• 图示：



总结:

- 重置事件 -> 读取 ROM 中固件, PC 指向 BIOS 第一条指令 -> BIOS 读取硬盘中的 MBR -> 将这些数据加载至 RAM -> CPU 执行读取的磁盘引导程序 + 找到 C 盘的位置 -> 读取 C 盘中的 PBR -> CPU 执行启动管理程序 -> 磁盘物理初始化 -> 将磁盘分区 -> 逻辑格式化 -> 操作系统初始化, 执行开机一系列命令;

举例:

- GRUB 用于引导 Linux 系统;

分析: 安装操作系统后硬件使用

硬盘视角:

1. 主引导记录 (MBR)
 - 包含: 磁盘引导程序和分区表
2. C 盘 (活动盘):
 - 引导记录 PBR (负责找到“启动管理器”)
 - 根目录
 - 其他
3. 其他盘

主存视角:

1. ROM
 - 包含 ROM 引导程序 BIOS, 即自举程序;
 - BIOS 是由一系列程序构成, 即使 CPU 断电, 这其中的数据也不会丢失;
2. RAM

补充: 另一种表述

1. CPU 从一个特定主存地址开始 > 取指令, 执行 ROM 中的引导程序 (先进行硬件自检, 再开机);
2. 将磁盘的第一块: 主引导记录读入内存, 执行磁盘引导程序, 扫描分区表;
3. 从活动分区 (又称主分区, 即安装了操作系统的分区)读入分区引导记录, 执行其中的程序;
4. 从根目录下找到完整的操作系统初始化程序 (即启动管理器)并执行, 完成“开机”的一系列动作;

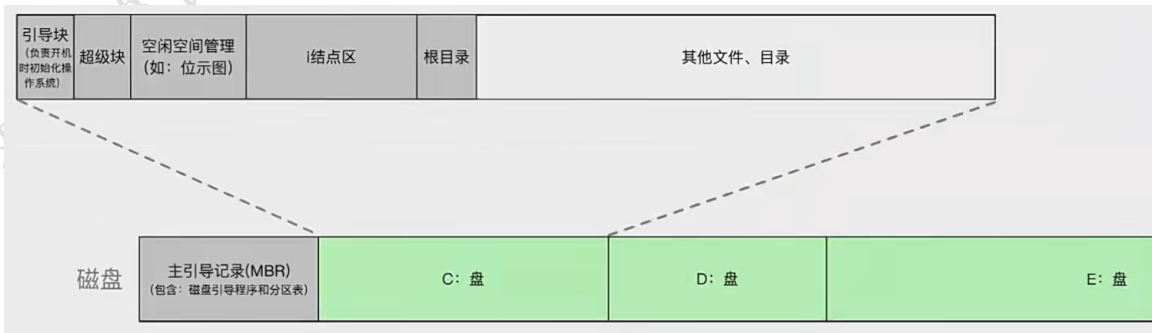
磁盘初始化

概念：三类初始化

- 真题：2022年
- 顺序：
 - 第一步：低级格式化 (物理格式化)
 - 将磁盘的各个磁道划分为扇区；
 - 一个扇区通常可分为头、数据区域 (如 512 B 大小)、尾三个部分组成；
 - 管理扇区所需要的各种数据结构一般存放在头、尾两个部分，包括扇区校验码 (如奇偶校验、CRC 循环冗余校验码等，校验码用于校验扇区中的数据是否发生错误)；
 - 第二步：将磁盘分区
 - 将磁盘分区，每个分区由若干柱面组成 (即分为我们熟悉的 C 盘、D 盘、E 盘)
 - 将磁盘分区 (分卷 Volume)，完成各分区的文件系统初始化；
 - 注意：逻辑格式化后，灰色部分就有实际数据了，白色部分还没有数据；



- 第三步：逻辑格式化
 - 逻辑格式化中，创建文件系统，写入文件管理所需的数据结构（初始化保存空闲磁盘块信息的数据结构）；
 - 文件系统的根目录的创建是在文件系统初始化的时候创建的；
 - 建立文件系统的索引结点表和空闲分区管理的数据结构
 - 检查位视图：
 - 判断哪些盘块是否空闲；
 - 超级块：
 - 迅速的找到若干个空闲的盘块；
 - 索引区 (i 节点区)：
 - 存放若干的索引节点；
 - 通过一个索引结点的下标，迅速的定位到一个索引节点；
 - 根目录；



真题：2021年

- 题目

【2021 统考真题】某计算机用硬盘作为启动盘，硬盘的第一个扇区存放主引导记录，其中包含磁盘引导程序和分区表。磁盘引导程序用于选择引导哪个分区的操作系统，分区表记录硬盘上各分区的位置等描述信息。硬盘被划分成若干分区，每个分区的第一个扇区存放分区引导程序，用于引导该分区中的操作系统。系统采用多阶段引导方式，除了执行磁盘引导程序和分区引导程序，还需要执行 ROM 中的引导程序。回答下列问题：

- 1) 系统启动过程中操作系统的初始化程序、分区引导程序、ROM 中的引导程序、磁盘引导程序的执行顺序是什么？
- 2) 将硬盘制作为启动盘时，需要完成操作系统的安装、磁盘的物理格式化、逻辑格式化、对磁盘进行分区，执行这 4 个操作的正确顺序是什么？
- 3) 磁盘扇区的划分和文件系统根目录的建立分别是在第 2) 问的哪个操作中完成的？

• 解析

- (1)
 - 执行顺序：
 - 1. ROM 中的引导程序；
 - 2. 磁盘引导程序；
 - 3. 分区引导程序；
 - 4. 操作系统的初始化程序；
 - 系统启动过程中，首先运行 ROM 中的引导程序，由磁盘引导程序指明操作系统所在的分区，然后运行该分区的分区引导程序，最后运行该分区内操作系统的初始化程序；
- (2)
 - 执行顺序：
 - 1. 磁盘的物理格式化；
 - 2. 对磁盘进行分区；
 - 3. 逻辑格式化；
 - 4. 操作系统的安装；
 - 磁盘格式化分物理格式化和逻辑格式化两种，物理格式化对新磁盘划分磁道和扇区，并在每个扇区的地址域上记录地址信息，每个扇区又划分出标识部分、间隔区和数据区等，物理格式化由磁盘生产厂家在出厂前完成；
 - 逻辑格式化是指在磁盘物理格式化之后对磁盘进行分区、建立根目录、写入文件管理所需的数据结构的过程。对分区进行逻辑格式化后，才能在该分区安装操作系统。
- (3)
 - 在磁盘的逻辑格式化中建立文件系统，进而得到根目录；
 - 由 (2) 可知，在磁盘的物理格式化中完成对磁盘扇区的划分；

虚拟机

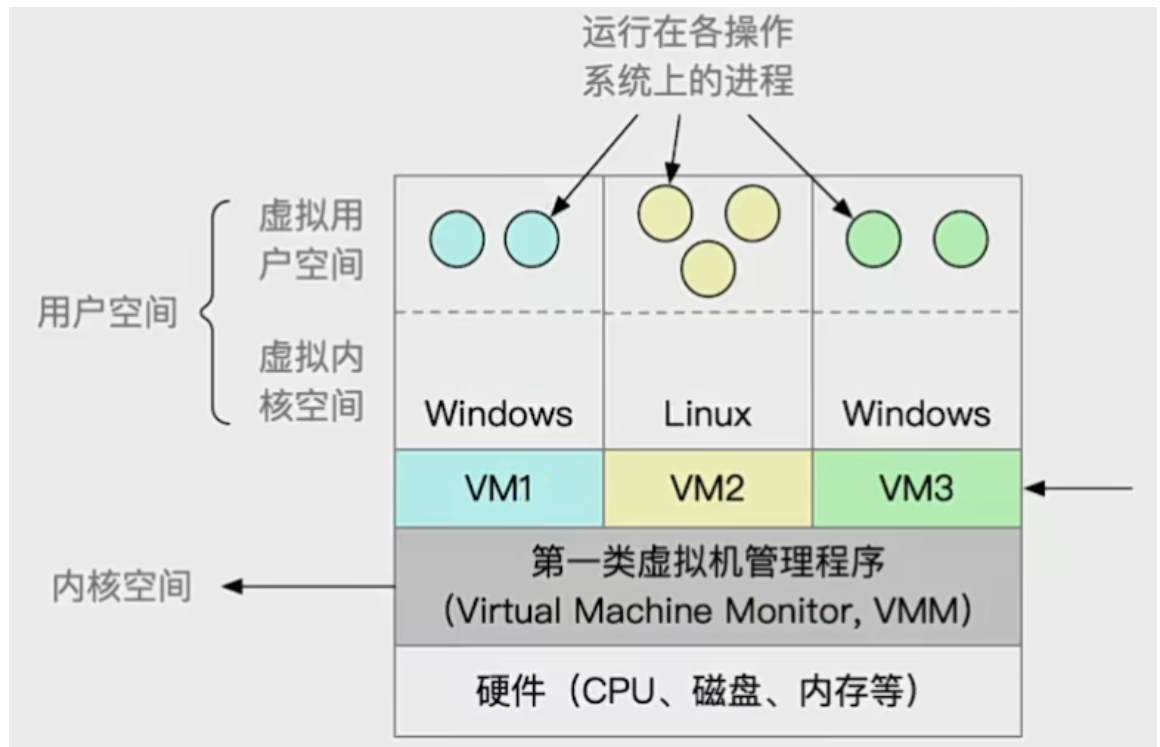
概念：什么是虚拟机

- 虚拟机
 - 使用虚拟化技术，将一台物理机器虚拟化为多台虚拟机器 (VirtualMachine, VM)，每个虚拟机器都可以独立运行一个操作系统；

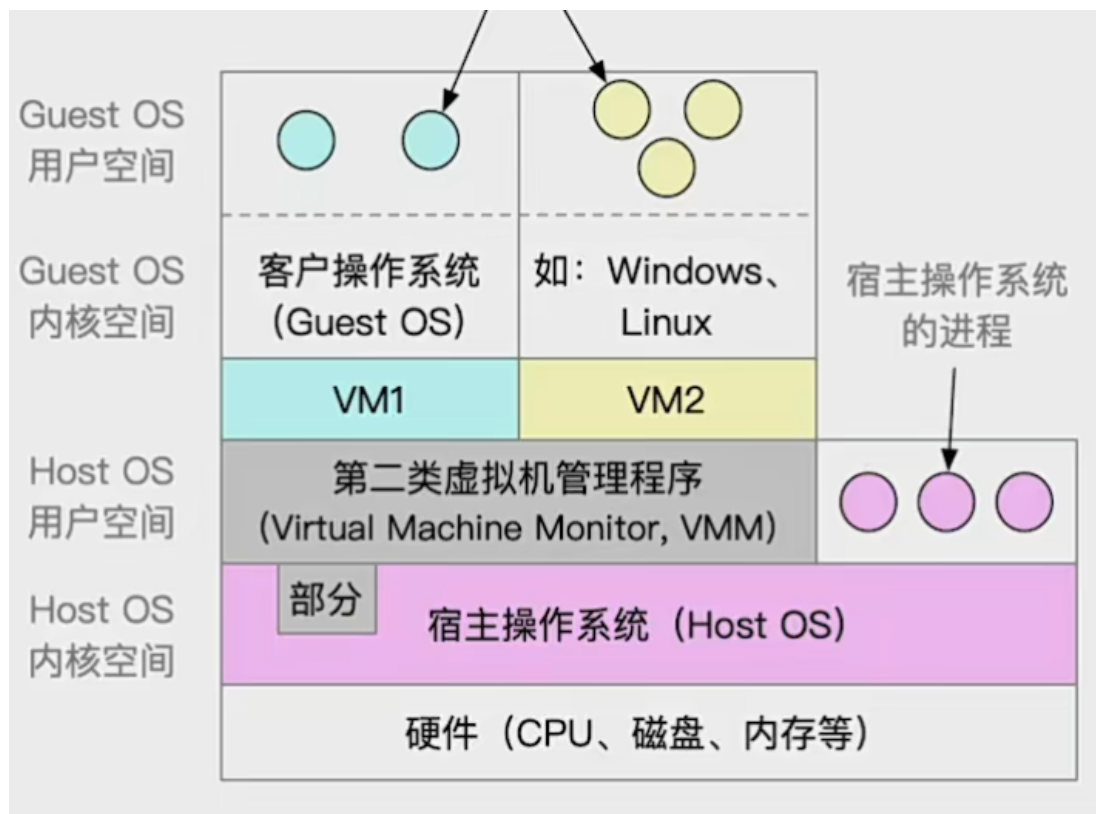
- 通过虚拟化技术，操作系统可以在其他操作系统上运行应用程序 -> 进一步引申：允许某一平台的操作系统允许在另一个平台上；
- 举例：
 - 在 `Java` 虚拟机上允许 `Java` 程序；
- 同义术语：
 - 虚拟机管理程序/虚拟机监控程序/`Virtual Machine Monitor` / `Hypervisor`；

概念：两类虚拟机

- 第一类 `VMM` - 部分运行在内核态
 - 图示：



- 概念：
 - 上层的操作系统运行在用户态，其中的内核空间为虚拟内核空间，其特权指令需要经过 `VM` 进行转化，才可以变成内核空间中的内核指令；
- 第二类 `VMM` - 完全运行在用户态
 - 图示：



- 概念：
 - 运行在用户态上，其如果需要使用特权指令，也需要向宿主操作系统进行申请；

分析：两类虚拟机架构对比

- 资源分配方式
 - VMM 1：需要在原本的硬盘上直接进行物理的分配，类似于**外核**的分配方式，分配的是物理内存；
 - VMM 2：虚拟机向宿主操作系统申请一个大文件，是 **Host OS** 为其分配的一个虚拟内存；
- 性能
 - VMM 1：性能更好；
 - VMM 2：性能更差，需要 **Host OS** 进行上下文转换；
- 可支持的虚拟机数量
 - VMM 1：可以支持更多的虚拟机；
 - VMM 2：更少；
- 虚拟机的迁移程序：
 - VMM 1：迁移起来更麻烦；
 - VMM 2：迁移起来很简单，直接打包成 **iso** 文件；
- 运行模式：
 - VMM 1：第一类 **VMM** 运行在最高特权级 (**Ring0**)，可以执行最高特权的指令；
 - VMM 2：部分运行在用户态、部分运行在内核态。**GuestOS** 发出的系统调用会被 **VMM** 截获，并转化为 **VMM** 对 **Hostos** 的系统调用；
- 图示：

	第一类VMM	第二类VMM
对物理资源的控制权	直接运行在硬件之上，能直接控制和分配物理资源	运行在Host OS之上，依赖于Host OS为其分配物理资源
资源分配方式	在安装Guest OS时，VMM要在原本的硬盘上自行分配存储空间，类似于“外核”的分配方式，分配未经抽象的物理硬件	GuestOS 拥有自己的虚拟磁盘，该盘实际上是 Host OS 文件系统中的一个文件。GuestOS 分配到的内存是虚拟内存
性能	性能更好	性能更差，需要HostOS作为“中介”
可支持的虚拟机数量	更多，不需要和 Host OS 竞争资源，相同的硬件资源可以支持更多的虚拟机	更少，Host OS 本身需要使用物理资源，Host OS 上运行的其他进程也需要物理资源
虚拟机的可迁移性	更差	更好，只需导出虚拟机镜像文件即可迁移到另一台 HostOS 上，商业化应用更广泛
运行模式	第一类VMM运行在最高特权级（Ring 0），可以执行最高特权的指令。	第二类VMM部分运行在用户态、部分运行在内核态。GuestOS 发出的系统调用会被 VMM 截获，并转化为 VMM 对 HostOS 的系统调用

补充：VMware 架构举例

• 图示：

