Lecture19 Virtualization Intro

1. Overview

计算设备集中与分散的变化

- 大型机时代
  - 集中式计算资源，所有用户通过网络连接大型机，共享计算资源
  - 20世纪70年代，虚拟化技术已经兴起
- PC时代
  - 分布式计算资源，每个PC用户独占计算资源
  - 20实际90年代，虚拟化技术沉寂
- 云时代
  - 集中式计算资源，所有人通过网络连接，共享计算资源
  - 21世纪，虚拟化技术再次兴起

现代公司的IT部署方式：云

云计算为云租户带来的优势

- 按需租赁、无需机房租赁费
- 无需雇佣物理服务器管理人员
- 可以快速低成本地升级服务器

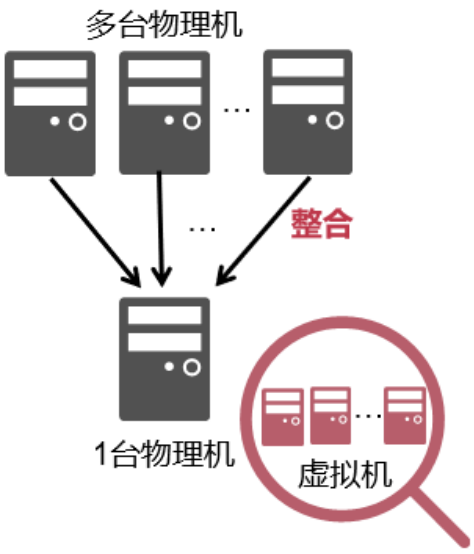
系统虚拟化是云计算的核心支撑技术

- **新引入的一个软件层**
  - 上层是操作系统（虚拟机）
  - 底层硬件与上层软件解耦
  - 上层软件可在不同硬件之间切换

虚拟化带来的优势

1. 服务器整合：提高资源利用率

- **单个物理机资源利用率低**
  - CPU利用率通常仅<20%
- **利用系统虚拟化进行资源整合**
  - 一台物理机同时运行多台虚拟机
- **显著提升物理机资源利用率**
- **显著降低云服务提供商的成本**



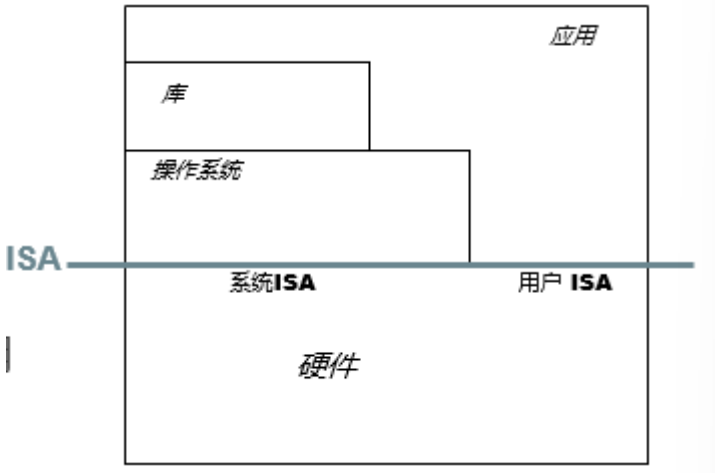
2. 方便程序开发
   - 调整操作系统
     - 单步调试操作系统
     - 查看当前虚拟硬件的状态
       - 寄存器中的值是否正确
       - 内存映射是否正确

        - **随时**修改虚拟硬件的状态
    - 测试应用程序的兼容性
        - 可以在一台物理机上同时运行不同的操作系统
        - 测试应用程序在不同操作系统上的兼容性
3. 简化服务器管理
    - 通过软件接口管理虚拟机
        - 创建、开机、关机、销毁
        - 方便高效
    - 虚拟机热迁移
        - 方便物理极其的维护和升级

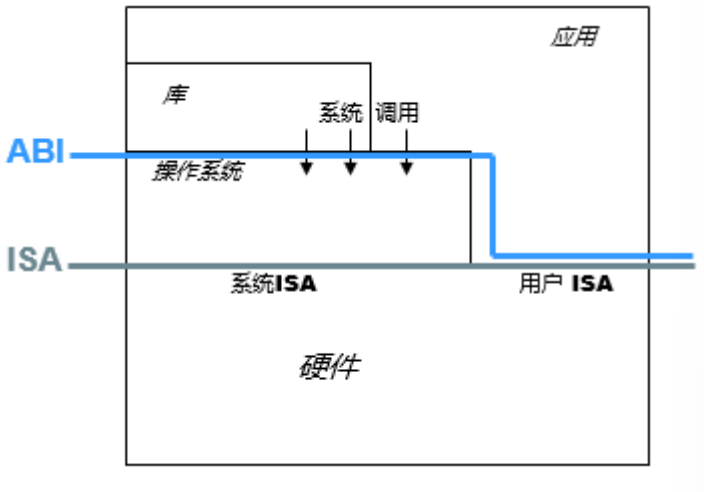
2. 什么是系统虚拟化?

操作系统中的接口层次：ISA

- **ISA层**
    - Instruction Set Architecture
    - 区分硬件和软件
    - 用户ISA：用户态和内核态程序都可以使用
    - 系统ISA：只有内核态程序可以使用



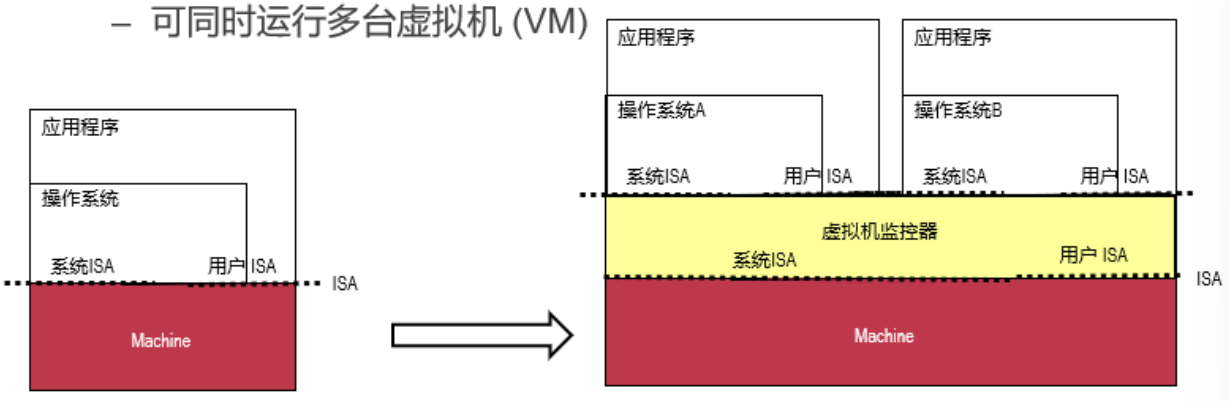
- **ABI层**
    - Application Binary Interface
    - 提供操作系统服务或者硬件功能
    - 包含用户ISA和系统调用



- **API层**
  - Application Programming Interface
  - 不同用户态库提供的接口
  - 包含库的接口和用户ISA
  - UNIX环境中的**clib**
    - 支持UNIX/C语言

虚拟机和虚拟机监控器

- **虚拟机监控器 (VMM/Hypervisor)**
  - 向上层虚拟机暴露其所需要的ISA
  - 可同时运行多台虚拟机 (VM)



系统虚拟化的标准

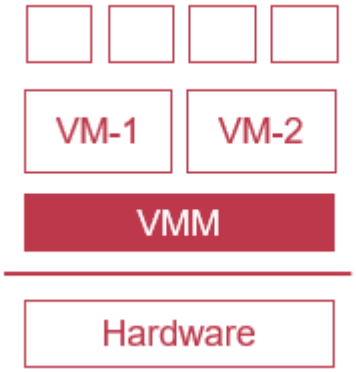
- **高效**系统虚拟化的三个特性（1974）
  1. 为虚拟机内程序提供与该程序原先执行的硬件**完全一样的接口**
  2. 虚拟机只比在无虚拟化的情况下**性能略差一点**
  3. 虚拟机监控器**控制所有物理资源**

3. 虚拟机监控器的分类

Type-1虚拟机监控器

> 提供虚拟的ISA，必须直接支持硬件驱动

- VMM直接运行在硬件上
  - 充当操作系统的角色
  - 直接管理所有的物理资源
    - 实现调度、内存管理、驱动等功能
- 性能损失较少
- 例如Xen, VMware ESX Server



Type-2虚拟机监控器（基于Host OS）

> 目前最主流的实现方式

- VMM依托于主机操作系统
  - 主机操作系统管理物理资源
  - 虚拟机监控器以进程/内核模块的形态运行
  - 易于实现和安装
  - 例如：QEMU/KVM

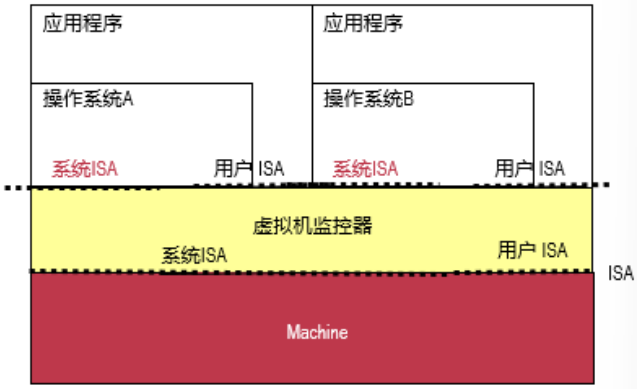
> Q：Type-2类型有什么优势?
>
> A：易于安装；可复用性强

4. 如何实现系统虚拟化

系统ISA：操作系统运行环境

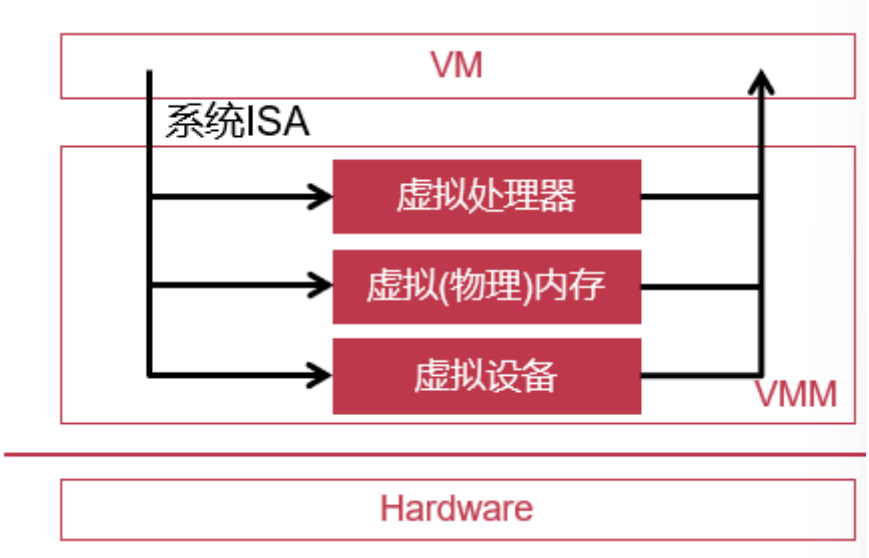
> 用户ISA不用做特别处理，可以直接用，并不会影响到运行环境

- **读写敏感寄存器**
  - sctrl_el1、ttbr0_el1/ttbr1_el1…
- **控制处理器行为**
  - 例如: WFI (陷入低功耗状态)
- **控制虚拟/物理内存**
  - 打开、配置、安装页表
- **控制外设**
  - DMA、中断



系统虚拟化的流程：Trap&Emulate

- **第一步（Trap）**
  - 捕捉所有系统ISA并陷入
- **第二步（Emulate）**
  - 由具体指令实现响应虚拟化
    - 控制虚拟处理器行为
    - 控制虚拟内存行为
    - 控制虚拟设备行为
- **第三步**
  - 回到虚拟机继续执行

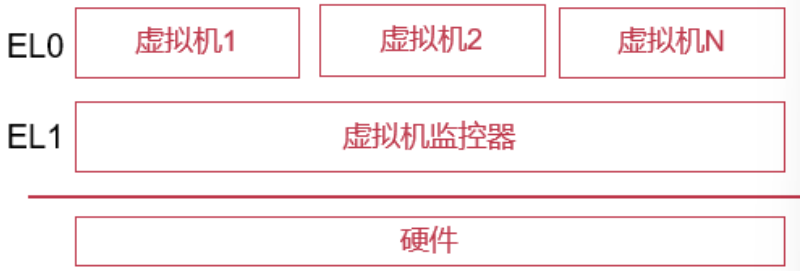


系统虚拟化技术

- **处理器虚拟化**
  - 捕捉系统ISA
  - 控制虚拟处理器的行为
- **内存虚拟化**
  - 提供“假”物理内存的抽象
- **设备虚拟化**
  - 提供虚拟的I/O设备

虚拟化：一种直接的实现方法

- **把虚拟机当做应用程序**
  - 将虚拟机监控器运行在EL1
  - 将客户操作系统和其上的进程都运行在EL0
  - 当操作系统执行系统ISA指令时下陷
    - 写入TTBR0_EL1
    - 执行WFI指令
    - …

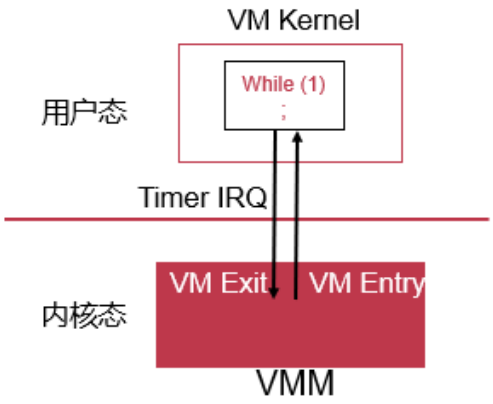


虚拟化功能：迭代演进、分布理解

- **第一版：支持只有内核态的虚拟机**
- **第二版：支持虚拟机内的时钟中断**
- **第三版：支持虚拟机内单一用户态线程**
- **第四版：支持虚拟机内多个用户态线程**
- **第五版：支持多个虚拟机间的分时复用**
- **第六版：支持多个物理CPU**
- **第七版：支持多个虚拟CPU**

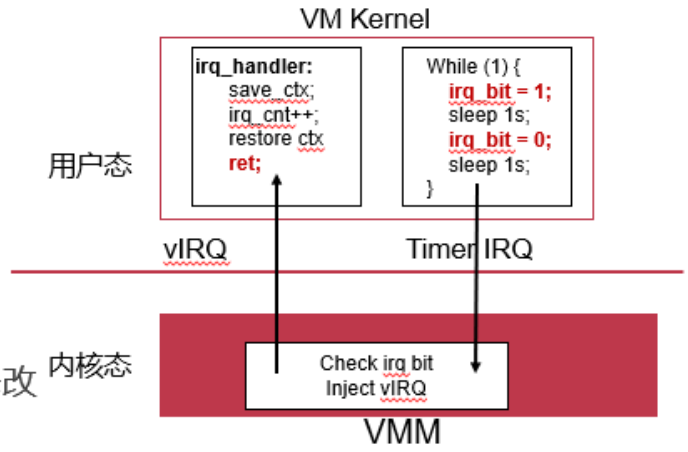
第一版：虚拟机只在内核态运行简单代码

- **VM的能力**
  - 只支持一个VM
  - 没有内核态与用户态的切换
  - 只有内核态，且仅运行用户ISA的指令（与用户态没有区别）
- **VMM的实现**
  - 处理时钟中断造成的VM Exit



第二版：虚拟机内部支持时钟中断

- **VM的能力**
  - **设置irq_handler**
  - **开关时钟中断 (irq_bit)**
  - **运行时钟中断处理函数**
- **VMM的实现**
  - 捕捉VM对irq_handler的修改
  - 捕捉VM对irq_bit的修改
  - 根据irq_bit决定插入虚拟时钟中断vIRQ并调用irq_handler

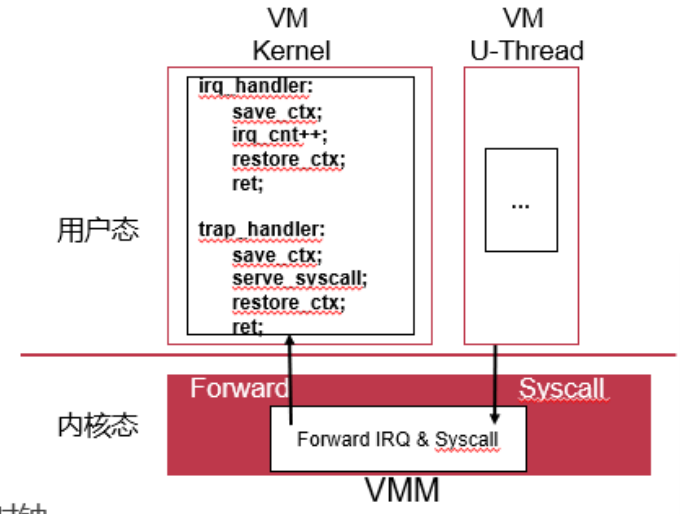


> Q：为什么要保存`save_ctx`？
>
> A：handler中的ctx是下陷之前进程的ctx，需要保证在回去时不变

> - 时钟中断发生之后，发生下陷，先进入VMM的时间中断handler函数中：
>
> 1. Save Context
> 2. 处理自己的中断
> 3. 检查Vitual Timer，插入虚拟中断，进入irq_handler(恢复下陷前的ctx)
> 4. 由于handler不知道自己是否运行在虚拟态，所以还是要保存ctx
> 5. 回去的时候还要save(save还是下陷前的ctx)
> 6. Restore Context

第三版：虚拟机内支持运行单一用户态线程

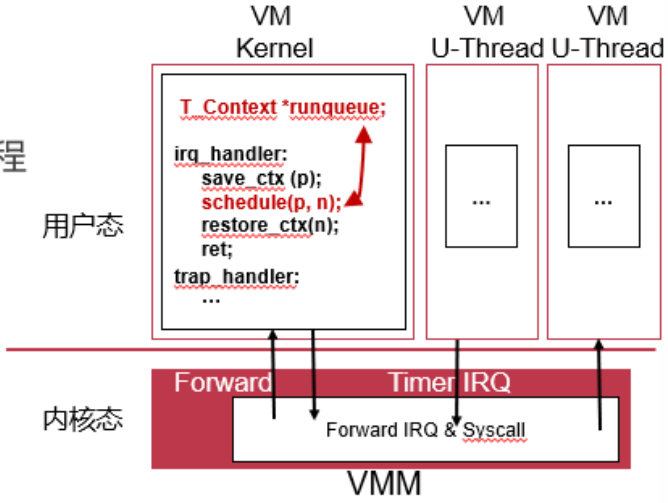
- **VM的能力**
  - **虚拟机包含内核态与用户态**
  - **用户态运行一个用户态线程**
    - **U-Thread**
  - **用户态线程可调用内核syscall**
  - **用户态线程可被时钟中断打断**
- **VMM的实现**
  - 捕捉并转发U-Thread系统调用syscall
  - 转发syscall至VM内核
  - 捕捉并转发U-Thread执行时的时钟中断



> - 应该有一个VM_ctx
>
> 1. user_mode（标明是内核态还是用户态）
>
> 2. PCB（Process Control Block）

第四版：虚拟机内部支持多个用户态线程

- **VM的能力**
  - 用户态运行**多个**用户态线程
  - **内核可调度用户态线程**
- **VMM的实现**
  - 与第三版相同

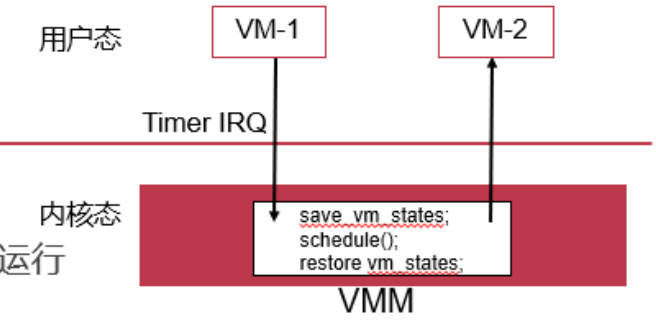


思考：Fork bomb是否会影响VMM?

> 不会影响，最多就是把分配给虚拟机的内存用完

第五版：支持多个虚拟机间的分时复用

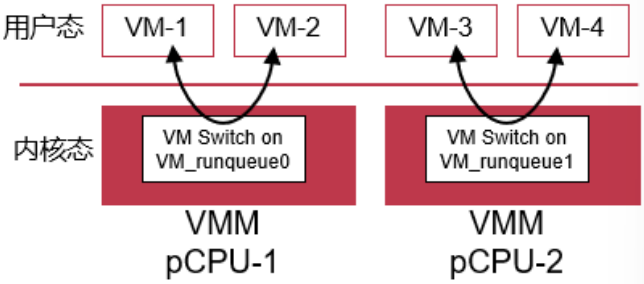
- **VM的能力**
  - **支持多个VM**
- **VMM的实现**
  - 每个VM对应一个内核线程
  - 维护VM_runqueue队列
    - 每个元素对应一个VM的运行状态
  - 由VMM实现VM间切换
    - 保存和恢复VM寄存器



> Chcore已经支持，只需在PCB中多加一个vmCtx，切换进程的时候多切换一个vmCtx

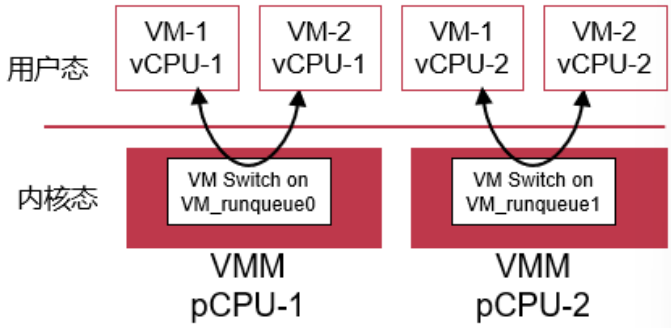
第六版：VMM支持多个物理CPU

- **VM的能力**
  - 与第五版相同
- **VMM的实现（基于第五版）**
  - 为每个pCPU维护不同的VM_runqueue



第七版：虚拟机支持多个虚拟CPU

- VM的能力（与第六版的区别）
  - 虚拟机有多个Virtual CPU (vCPU)
- VMM的实现
  - 在VM_runqueue中标记出VM和vCPU的类型



> - struct vCPU
>
> 1. user_mode
>
> 2. vcpu_ctx
>
> 调度的单位是vCPU（类似于线程），这里VM_ctx更像是进程
>
> 虚拟化中的锁会带来巨大的性能开销(Virtualization Lock Preemption)

ARM不是严格的可虚拟化架构

- 敏感指令
  - 读写特殊寄存器或更改处理器状态
  - 读写敏感内存：例如访问未映射内存、写入只读内存
  - I/O指令
- 特权指令
  - 在用户态执行会触发**异常**，并陷入内核态

> 在ARM中：不是所有敏感指令都属于特权指令

- 例子: CPSID/CPSIE指令
  - CPSID和CPSIE分别可以关闭和打开中断
  - 内核态执行：PSTATE.{A, I, F} 可以被CPS指令修改
  - 在用户态执行：CPS 被当做NOP指令，不产生任何效果
    - 不是特权指令