KVM及其调度

by 胡小康

资料整理
火龙果软件
荐

目录

1 虚拟化技术

2 KVM简介

3 vCPU模型

4 KVM调度

5 CFS调度器

资料整理
火龙果软件
荐

第一部分

1 虚拟化技术

2 KVM简介

3 vCPU模型

4 KVM调度

5 CFS调度器

资料整理
火龙果软件
荐

1. 虚拟化

什么是虚拟化

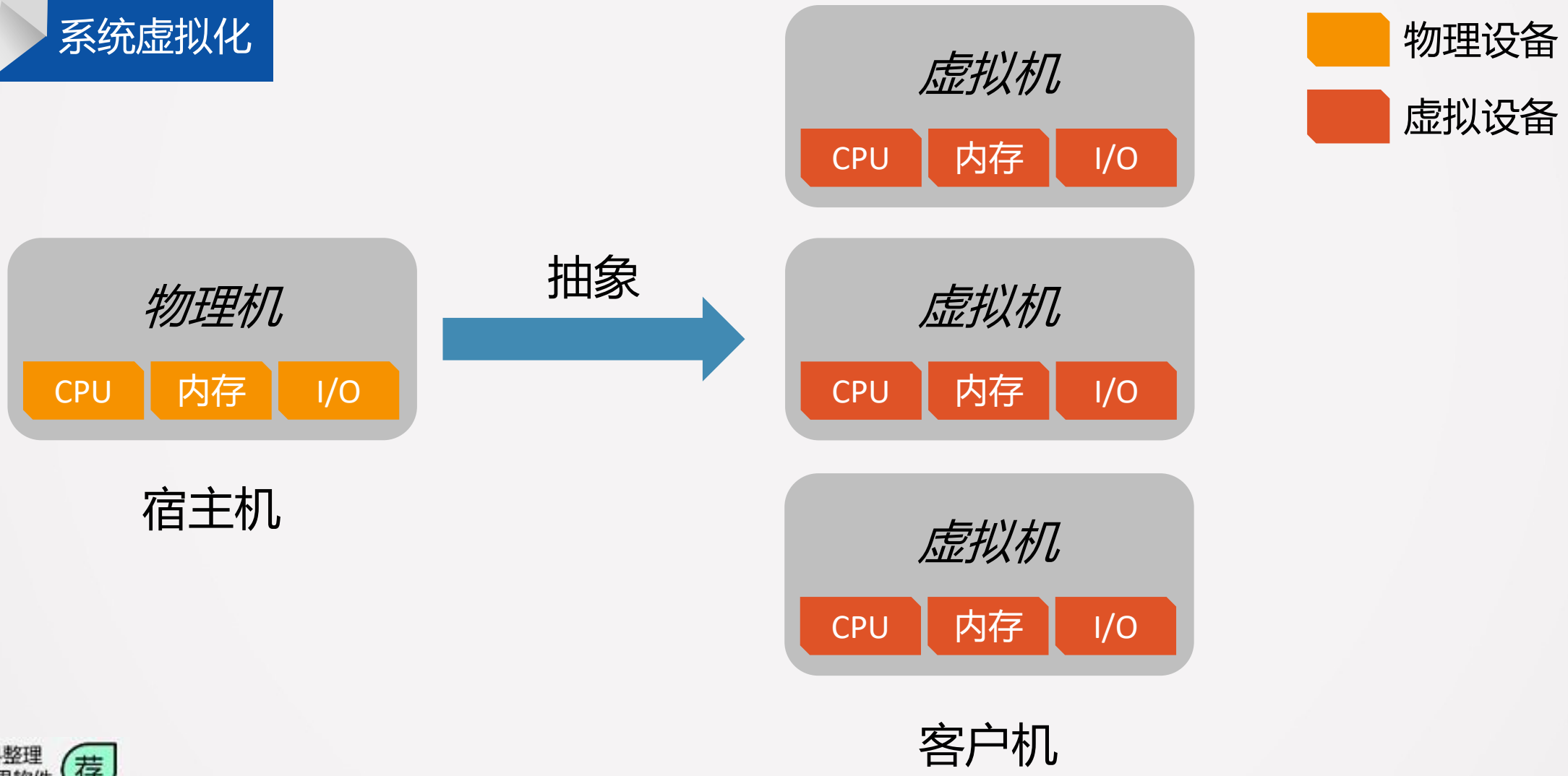
将一种形式的资源抽象成为另外一种形式的资源
下层为上层提供一个与它原先所期待的环境完全一致的虚拟环境

分类

1. 系统虚拟化
2. 存储虚拟化
3. 网络虚拟化
4. 桌面虚拟化

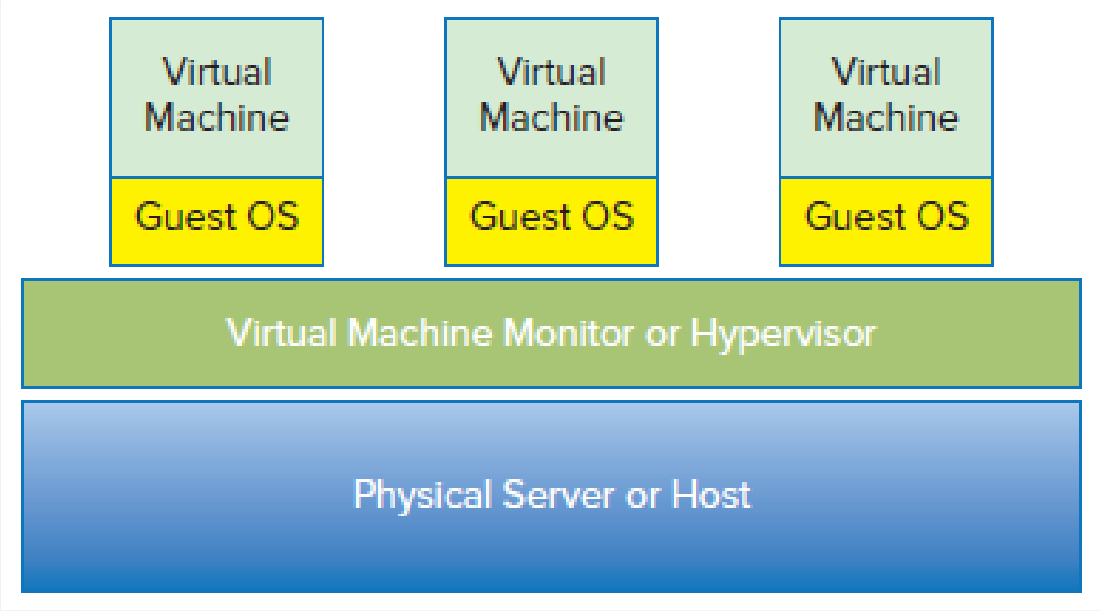
资料整理
火龙果软件
荐

2. 系统虚拟化



资料整理
火龙果软件
荐

2. 系统虚拟化



系统虚拟化基本架构

VMM需要创建虚拟运行环境，让客户机以为自己是独占硬件的，而实际上，它所看到的硬件都是VMM虚拟出来的资源。

资料整理
火龙果软件
荐

2. VMM

职责

- 取代传统的操作系统，管理系统中的物理硬件资源。
- 为各个虚拟机提供虚拟资源，包括虚拟CPU、虚拟内存、虚拟I/O设备等。
- 管理虚拟环境，维护虚拟机的正常运行。

分类

- 类型一VMM，直接运行在硬件上，控制硬件资源，管理虚拟机。
- 类型二VMM，运行在传统操作系统中，类似于一个应用程序，物理资源仍由操作系统管理，VMM通过调用操作系统的服务来获取资源。

资料整理
火龙果软件
荐

3. 分类

完全虚拟化

- 全面模拟了底层硬件资源，为虚拟机提供的虚拟环境与真实物理机完全相同。
- Guest OS无需做出任何修改就可以直接安装并运行。
- Guest OS意识不到自己是处于虚拟环境中。

类虚拟化

- 提供给虚拟机的环境与真实硬件平台不同，而是一个修改过的硬件抽象。
- Guest OS的源代码需要经过修改，从而与VMM配合，完成系统虚拟化。

资料整理
火龙果软件
荐

3. 实现

- CPU虚拟化
  - 传统做法：“优先级压缩”加“陷入再模拟”；虚拟化漏洞
  - 现在做法：硬件辅助CPU虚拟化，如Intel VT-x技术，引入“根操作”和“非根操作”
- 内存虚拟化
  - 传统做法：影子页表，复杂度高，内存开销大
  - 现在做法：从硬件层面解决，如Intel的EPT技术
- I/O虚拟化
  - 纯模拟设备，半虚拟化设备，直接分配设备

资料整理
火龙果软件
荐

第二部分

1 虚拟化技术

2 KVM简介

3 vCPU模型

4 KVM调度

5 CFS调度器

资料整理
火龙果软件
荐

基础知识

什么是KVM

Kernel-based Virtual Machine, 属于类型二VMM，采用完全虚拟化
内核中的一个模块，按需加载，与QEMU协同工作，构建虚拟化环境

KVM和QEMU的分工

KVM负责CPU虚拟化，内存虚拟化，以及一些对性能要求比较高的虚拟设备；大部分外设（网卡、硬盘等）交由用户空间的QEMU来模拟

资料整理
火龙果软件
荐

QEMU

- 开源软件，不属于KVM，包含整套虚拟机实现技术。
- 采用纯软件方式实现虚拟机，性能很低。
- 为了简化开发，KVM没有选择从零起步，而是对QEMU进行了修改和利用。

KVM

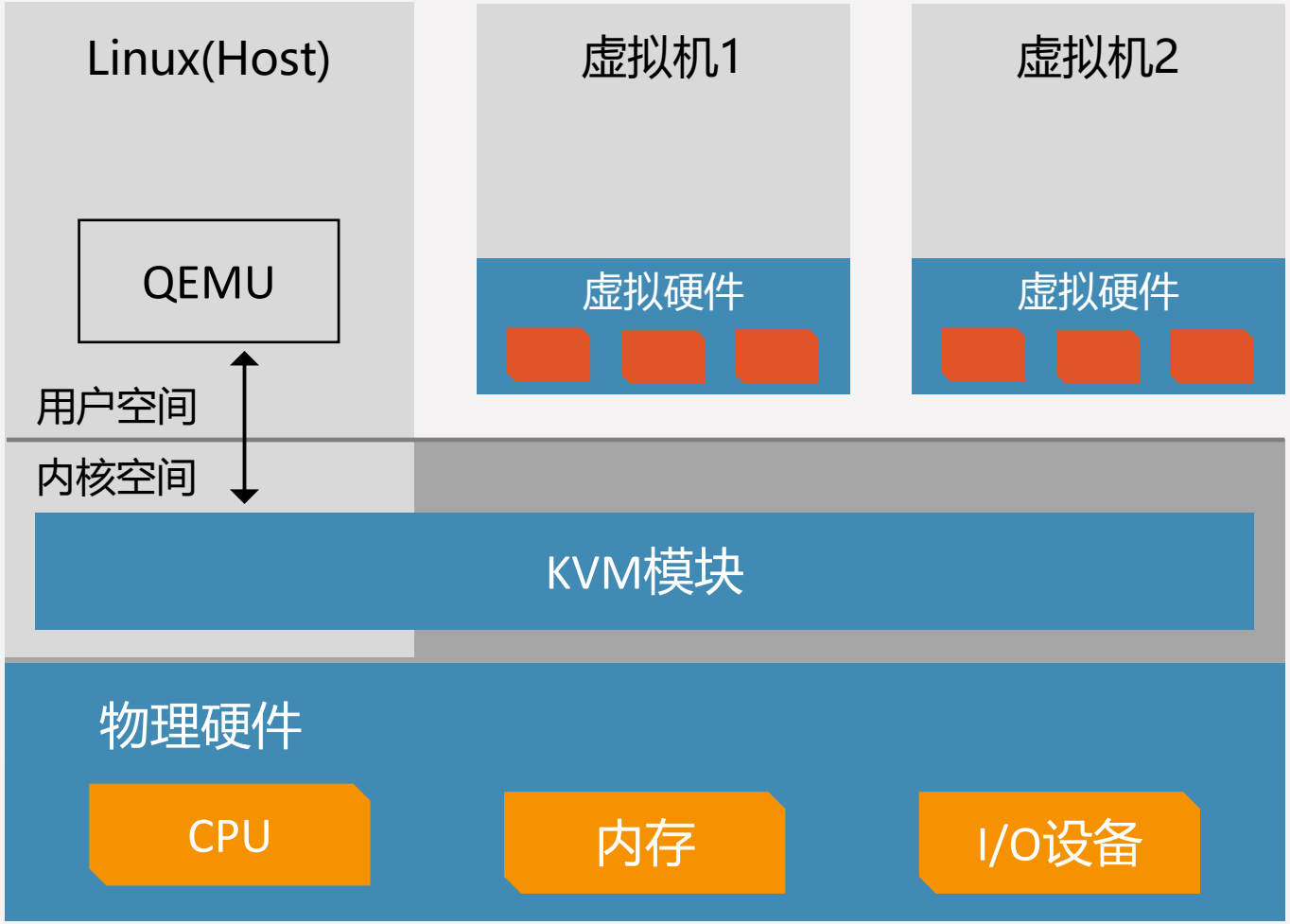
借用了现有的QEMU来完成设备模拟，仅需专注于对性能要求较高的CPU虚拟化、内存虚拟化

QEMU

使用了KVM的虚拟化技术，为自己的虚拟机提供硬件虚拟化的加速，极大的提高性能

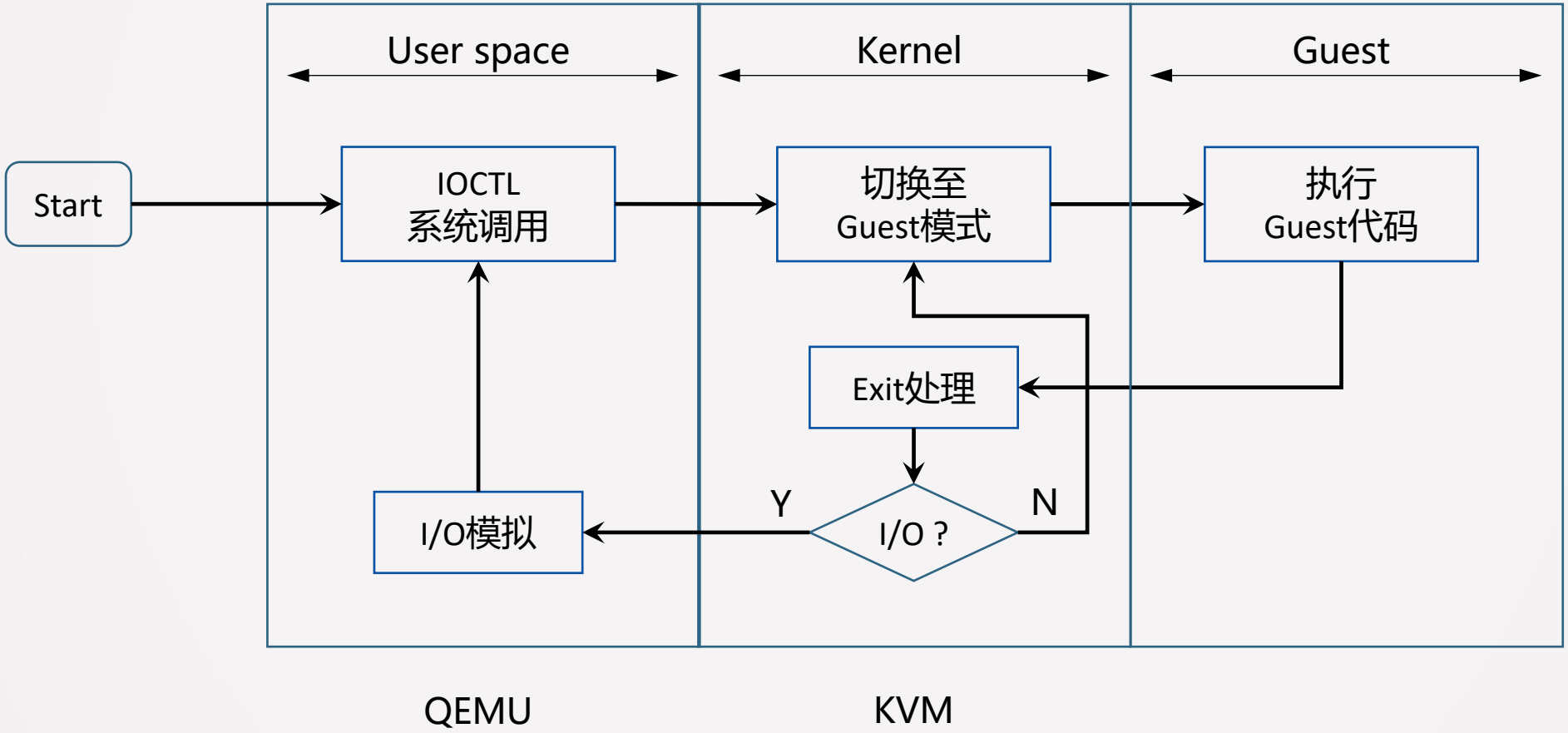
资料整理
火龙果软件
荐

基本架构



资料整理
火龙果软件
荐

工作原理



资料整理
火龙果软件
荐

第三部分

1 虚拟化技术

2 KVM简介

3 vCPU模型

4 KVM调度

5 CFS调度器

资料整理
火龙果软件
荐

vCPU模型

基础知识

虚拟机和vCPU

每个虚拟机都是一个标准的Linux进程（QEMU创建）
每个vCPU是QEMU进程派生出的一个普通线程

vCPU调度

vCPU是可调度的执行实体，VMM把vCPU调度至pCPU上运行，
并通常是在不同vCPU之间时分共享pCPU。

资料整理
火龙果软件
荐

工作示意图

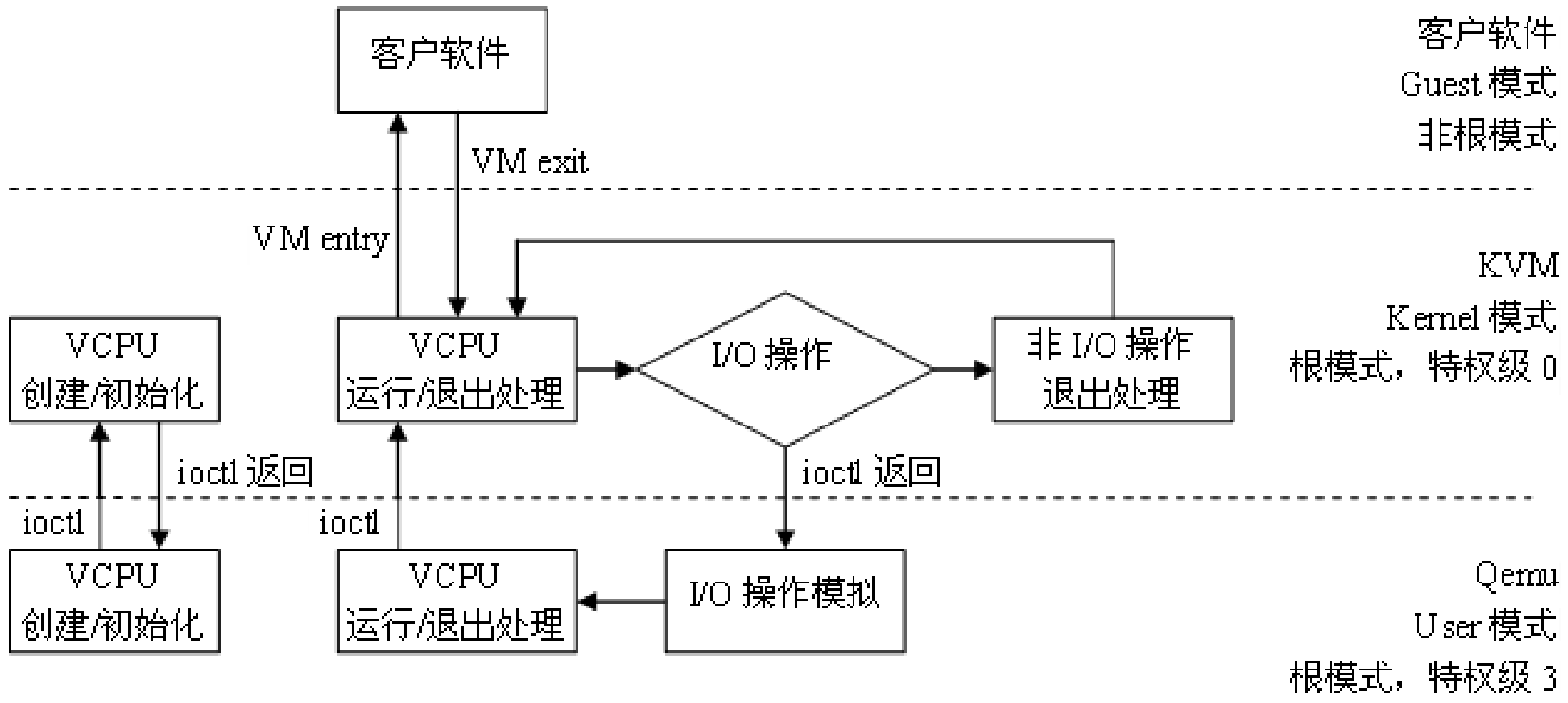


图 2.1　VCPU 工作模型

资料整理
火龙果软件
荐

vCPU模型

创建与运行

- Qemu-kvm启动一个子线程（vCPU线程），该线程创建并初始化vCPU，主线程等待；
- 子线程创建并初始化vCPU完毕，进入等待状态，等候主线程的运行通知；
- 主线程继续初始化虚拟机工作，初始化完成后，通知子线程继续运行；
- 子线程调用kvm_vcpu_ioctl(vcpu, KVM_RUN, ...)来运行。

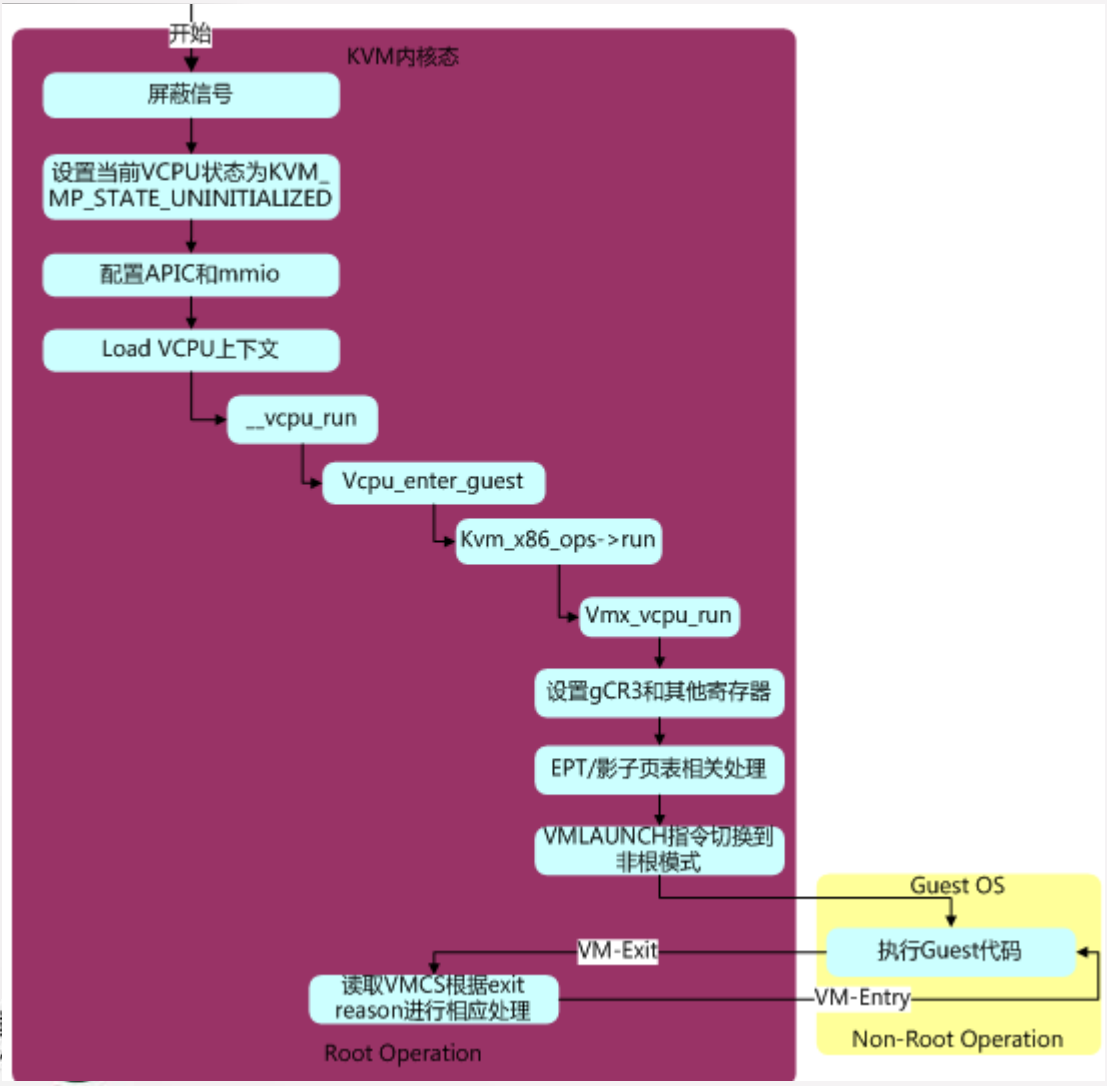
```
kvm_vcpu_ioctl(vcpu, KVM_RUN, ...) {
    vcpu_load(vcpu);
    kvm_arch_vcpu_ioctl_run(vcpu, vcpu->run);
}
```

```
vcpu_load(struct kvm_vcpu *vcpu) {
    kvm_arch_vcpu_load(vcpu, cpu);
}
```

资料整理
火龙果软件
荐

vCPU模型

创建与运行




```
__vcpu_run() {
    while (r > 0) {
        r = vcpu_enter_guest(vcpu);
        if (r < 0) break;
        if (need_resched()) kvm_resched(vcpu);
    }
    return r;
}
```

```
int vcpu_enter_guest() {
    ...    // VM Entry
    vcpu->mode = IN_GUEST_MODE;
    kvm_x86_ops->run(vcpu);
    ...    // VM Exit
    r = kvm_x86_ops->handle_exit(vcpu);
    return r;
}
```

资料整理
火龙果软

第四部分

1 虚拟化技术

2 KVM简介

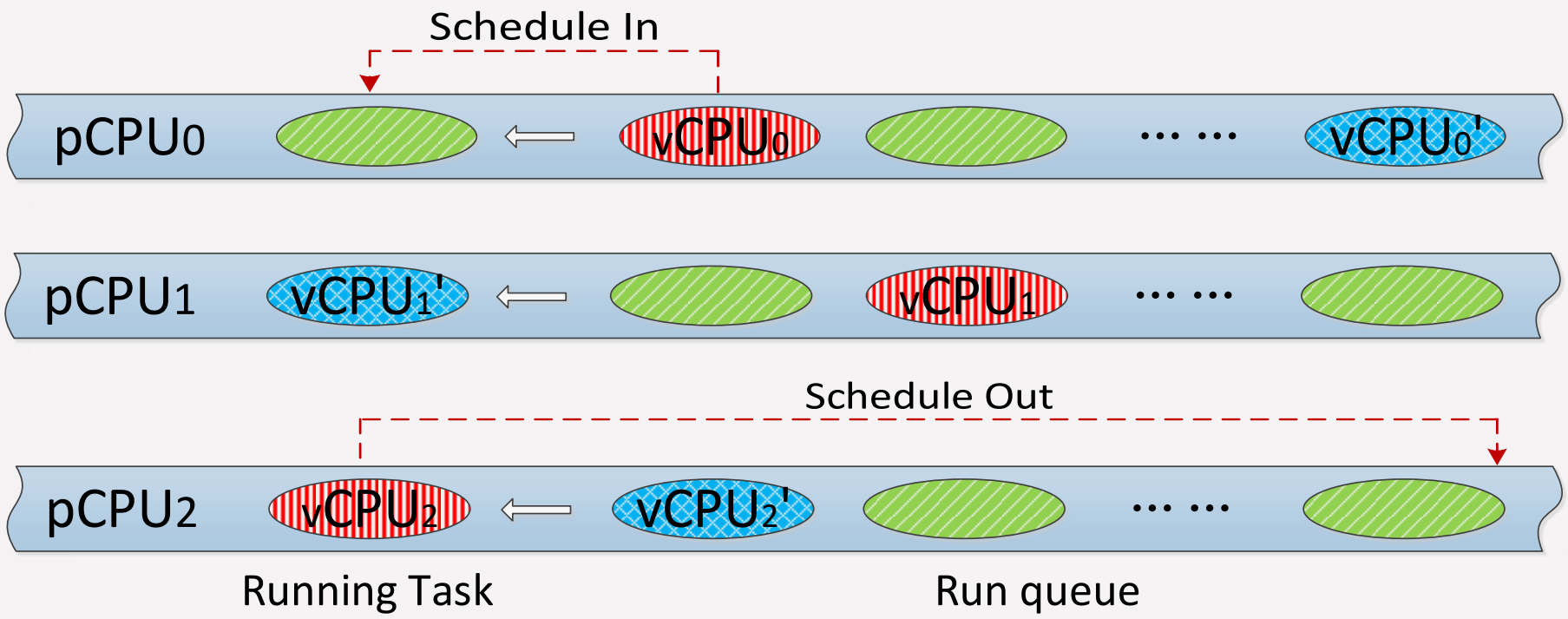
3 vCPU模型

4 KVM调度

5 CFS调度器

资料整理
火龙果软件
荐

示意图



既然vCPU是标准的Linux线程，那么对vCPU的调度就可以直接借用Linux现有的调度器

资料整理 火龙果软件 荐

基础知识

运行队列

- SMP系统中，每个CPU（这里指的是逻辑CPU）都会有自己的运行队列。
- 进程调度的任务：从各个CPU运行队列中分别选择一个进程在该CPU上运行。
- 在Linux中，进程调度的调度实体其实是线程（LWP）。

进程分类

- Linux中，普通进程分为“交互式进程”和“批处理进程”。
- 交互式进程，经常与用户进行交互，当接受输入后，必须被很快唤醒。
- 批处理进程，不必与用户交互，常在后台运行，因此无需快速响应。

资料整理
火龙果软件
荐

基础知识

调度时机

- 当前正在运行的进程的状态变为非可执行状态，主动或被动。
- 发生抢占，即进程运行时非预期地被剥夺CPU的使用权。可分两种情况：进程用完了时间片；出现了优先级更高的进程。

内核态抢占

- 理想情况下，只要满足“出现了优先级更高的进程”，当前进程就应该被抢占。
- 如果当前进程正处于内核态，支持抢占并不容易。
- Linux 2.6开始支持内核抢占，但在许多地方还需临时关闭。

资料整理
火龙果软件
荐

调度器发展

Linux O(N) Scheduler

- 基于优先级的设计
- 时间片递减
- 倾向于提高交互式进程的优先级
- 算法复杂度为O(N)

CFS Scheduler

- Complete Fair，完全公平思想
- 不再跟踪进程睡眠时间，也不再企图区分进程类型
- 将所有普通进程统一对待
- 对交互式进程仍然提供快速响应

1 2 3

Linux O(1) Scheduler

- 静态优先级决定基本时间片长短
- 考察平均睡眠时间，区分进程类型，设置动态优先级
- 动态优先级决定如何调度，目的是优待交互式进程
- 算法复杂度为O(1)

资料整理
火龙果软件
荐

第五部分

1 虚拟化技术

2 KVM简介

3 vCPU模型

4 KVM调度

5 CFS调度器

资料整理
火龙果软件
荐

CFS原理

设计理念	在真实硬件上模拟了一个“理想的多任务CPU” “同时”执行所有的可执行任务，并且公平地获得自己所占的CPU份额
现实情况	不可能做到真正“同时”，每个CPU在同一时刻只能运行一个进程 CFS就是尽可能去接近这样的“理想的多任务CPU”

资料整理
火龙果软件
荐

CFS原理

引入vruntime，虚拟运行时间

Essentially, vruntime is a measure of the "runtime" of the thread - the amount of time it has spent on the processor. Among the tasks on a given run queue, the task with the lowest vruntime is the task that most deserves to run, hence select it as 'next'.

资料整理
火龙果软件
荐

CFS原理

- CFS中采用红黑树构建一个进程执行的时间轴，其键值就是vruntime。
- CFS每次均挑选树中处于最左边的进程（其vruntime值最小）来执行，随着该进程的执行，它的vruntime值逐渐变大，它在树中的位置越来越向右。
- 某个时刻，这个正在被执行的进程不再处于树中最左位置，新的位于最左边的进程会抢占它，继而得到调度运行。
- 按照这种模式，所有进程都有被执行的机会。

资料整理
火龙果软件
荐

vruntime

- 每次时钟中断时，vruntime增加的值由实际运行时间换算而来，优先级高则值小，相当于增加的慢，这就保证了调度的公平性。
- 对于交互式进程，由于长时间睡眠，vruntime值会较小，当其被唤醒时，就会优先执行。

优先级越高 → vruntime增加的越慢 → 获得运行时间越多 → vruntime越来越大 → 平衡状态

资料整理 火龙果软件 荐

完全公平

系统中只有n个普通进程在运行，只有1个逻辑CPU，没有新进程，没有进程睡眠，也没有进程执行完毕，初始时n个进程的vruntime均相同。

计算任一进程的vruntime在一个调度周期内的增加值

calc_delta_fair(delta_exec, curr) + calc_delta_fair(delta_exec, curr) + ...

= delta_exec * NICE_0 / curr->weight + delta_exec * NICE_0 / curr->weight + ...

= ideal_runtime * NICE_0 / curr->weight

= period * curr->weight / cfs_rq->weight * NICE_0 / curr->weight

= period * NICD_0 / cfs_rq->weight

定值！

资料整理
火龙果软件
荐

完全公平

理想情况	每轮调度周期开始前和结束后，所有进程的vruntime值均相同，并且各个进程根据优先级获得了公平的CPU份额
现实情况	不可能每轮都保持公平，也许需要经历好几轮后才能达到公平，但从长远来看，CFS调度确实是“完全公平”的！

资料整理
火龙果软件
荐

Q&A

资料整理
火龙果软件
荐