RISC-V Hypervisor Extension

李宇

2023 年 6 月 14 日

自我介绍

李宇

- 字节跳动 虚拟化研发工程师
  - 目前专注于 Kata Containers 相关研发
  - 技术栈以 x86, C, Rust 为主
- 2020 年参加 OS Tutorial Summer of Code 2020
  - 成功在 zCore 中跑起 GCC
- 试图混入今年的活动，尝试在 ArceOS 中实现 Hypervisor
  - 但由于工作比较忙~~而且很懒~~，导致进度缓慢

基本术语

术语	解释
Hypervisor (VMM)	用于创建与执行虚拟机的软件、固件或硬件
Host	用于运行虚拟机的机器（宿主机）
Guest	运行在 VMM 中的虚拟机
vCPU	虚拟出的 CPU，一般对应宿主机上的线程
PA & VA	Physical Address & Virtual Address
HPA & HVA	Host Physical Address & Host Virtual Address
GPA & GVA	Guest Physical Address & Guest Virtual Address

CPU 虚拟化

CPU 虚拟化大致分为以下三类

- 通过软件翻译指令实现（软件虚拟化）
  - QEMU (TCG)
  - Spike
- 通过修改虚拟机内核协同实现（半虚拟化）
  - Xen（早期版本）
- 通过硬件虚拟化技术实现（硬件虚拟化）
  - KVM (Linux)

需要说明的是，现在主流的实现方式如 KVM, Hyper-V 在硬件虚拟化的基础上也使用了部分半虚拟化的特性，以此提高性能

VM entry & VM exit

半虚拟化和硬件虚拟化会将指令运行在硬件上以保证性能。

对于一般的运算、访存、跳转指令可以直接运行，但一些较敏感的指令或者一些特殊的指令可能无法直接运行在 Guest 中，这时会发生 VM exit 陷入到 Host 中，由 Host 来处理，处理完成后再返回到 Guest。

比如，RISC-V 虚拟机中的

- `ECALL` 指令，执行时会陷入到 Host
- `WFI` 指令，可以通过设置 `hstatus.VTW` 来控制是否陷入到 Host。若虚拟机空闲，宿主机可以将其调度走
- `SFENCE.VMA` 指令，可以通过设置 `hstatus.VTVM` 来控制是否陷入到 Host

内存虚拟化的需求

内存虚拟化的需求

- 由于内存是 Host 提供的，所以 Guest 最终也需要访问 Host 的内存
- 保证隔离性，确保 Guest 无法访问 Host 内存，也不能访问其他 Guest 的内存
- 为确保 Guest 可以无感知地正常运行，需要让 Guest （认为的）物理地址从 0 开始

内存虚拟化的实现方式

影子页表 (Shadow Page Table)

- 在硬件还不支持两级地址翻译时出现的技术
- 宿主机需要拦截虚拟机部分对页表的修改操作，并把虚拟机的页表偷换掉
- Guest 认为自己填写的是 VA $\rightarrow$ PA 的映射
- Host 将页表从 GVA $\rightarrow$ GPA 的映射替换成 GVA $\rightarrow$ HPA 的映射
- 由于 Guest 对页表的操作经常会被 Host 劫持，导致性能相对较低

内存虚拟化的实现方式

两级地址翻译

- 通过硬件实现，比较常见的有
  - Intel 的 Extended Page Tables (EPT)
  - AMD 的 Nested Page Tables (NPT)
- RISC-V 中的实现被称为 G-stage Page Tables
- Guest 正常填写 VA $\rightarrow$ PA 的映射（GVA $\rightarrow$ GPA）
- Host 会在第二级地址的 Page Fault 时填写 GPA $\rightarrow$ HPA 的映射
- 虚拟机运行时，硬件会进行 GVA $\rightarrow$ GPA $\rightarrow$ HPA 的翻译，以此实现内存虚拟化

内存虚拟化的实现方式

Direct Paging（Xen 的半虚拟化实现）

- Xen 会为虚拟机提供一个 Physical to Machine (P2M) 的映射
- Guest 在填写页表时会主动查询 P2M 映射，然后向页表中写入 GVA $\rightarrow$ HPA 的映射
- 整个过程中不会发生频繁的 VM exit，而且 MMU 翻译地址的开销也较小，整体性能较高

IO 虚拟化的发展

传统方式: trap & emulate

- Host 将设备地址所在的页配置成 MMIO
- Guest 访问时发生 VM exit
- Host 模拟该设备行为后再返回到 Guest

经典案例: QEMU 中的 E1000 网卡

IO 虚拟化的发展

VirtIO: 宿主机和虚拟机间通过共享内存进行通信

Guest Kernel (Network)	↔	Host Kernel (Virtualization)	↔	VMM	↔	Host Kernel (Network)

Vhost: 将需要经过内核的 IO（网络等）直接 offload 到内核，不再经过用户态

Guest Kernel (Network)	↔	Host Kernel (Virtualization)	↔	Host Kernel (Network)

Vhost-user: 用户态程序接管网络或存储设备

Guest Kernel (Network)	↔	Host Kernel (Virtualization)	↔	User Process (Network)

IO 虚拟化的发展

VFIO

- 基于硬件提供的虚拟化 IO 直通技术、IO 设备的 DMA 和中断重映射功能以及 IOMMU 实现
- 通过将 IO 设备的 DMA 和中断重定向到虚拟机中，达到一种虚拟机直接使用宿主机设备的效果
- 可以通过硬件提供的 SR-IOV 功能，在硬件层面切分成多个虚拟设备

VFIO-mdev

- VFIO-mdev 是一套内核的框架，厂商的驱动可以在 VFIO 基础上进行一些软件层面的切分

IO 虚拟化的发展

vDPA (virtio Data Path Acceleration)

- vDPA 定义了一种设备，这种设备的数据路径 (datapath) 严格遵守 VirtIO 规范，控制路径 (control path) 可以由厂商自定义
- vDPA 设备可以通过硬件实现，也可以由软件模拟
- vDPA 设备可以给虚拟机使用，也可以给宿主机使用

VDUSE (vDPA Device in Userspace)

- VDUSE 可以在用户态模拟 vDPA 设备
- 为了保证安全性，vDPA 的控制路径在内核中实现，数据路径在用户态实现

特权级[1]

在不支持 RVH 的系统中

P	Abbr	Name
0	U-mode	User mode
1	S-mode	Supervisor mode
3	M-mode	Machine mode

在支持 RVH 的系统中

V	P	Abbr	Name
0	0	U-mode	User mode
0	1	HS-mode	Hypervisor-extended supervisor mode
0	3	M-mode	Machine mode
1	0	VU-mode	Virtual user mode
1	1	VS-mode	Virtual supervisor mode

[1] 本节内容基于 The RISC-V Instruction Set Manual Volume II: Privileged Architecture (Document Version 20211203)

Hypervisor CSRs

RVH 中新增了一些可以在 HS-mode 使用的，用于控制 Hypervisor 行为的 CSR。

其中大部分 CSR 的行为与同名但不同前缀的 CSR 接近，需要重点关注的有

- `htimedelta`: 虚拟机中 CSR `time` 的值与宿主机 CSR `time` 之间的差值
- `hgatp`: 用于存储控制 GPA $\rightarrow$ HPA 映射的页表
- `hvip`: 用于向 VS-mode 注入中断

Hypervisor CSRs

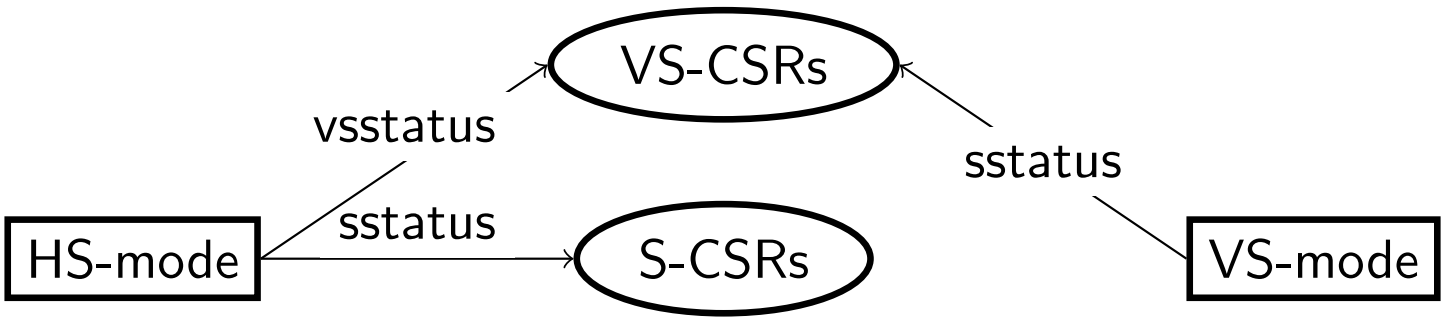
- `hstatus`
  - `hstatus.SPV` (Supervisor Previous Virtualization mode)
    之前是否处于虚拟化的特权级 (VS-mode 或 VU-mode)
  - `hstatus.SPVP` (Supervisor Previous Virtual Privilege)
    之前的虚拟特权级
  - `hstatus.VTW` (Virtual Timeout Wait)
    VS-mode 执行 `WFI` 指令时是否会 trap 到 HS-mode
  - `hstatus.VTVM` (Virtual Trap Virtual Memory)
    VS-mode 执行 `SFENCE.VMA`, `SINVAL.VMA` 或访问 CSR `satp` 时是否会 trap 到 HS-mode
  - `hstatus.HU` (Hypervisor in U-mode)
    U-mode 是否可以使用 `HLV`, `HSV` 等指令
  - `hstatus.GVA` (Guest Virtual Address)
    陷入到 HS-mode 时，如果 `stval` 中的是 GVA，则该 bit 会被设为 1

Virtual Supervisor CSRs

RVH 中新增了一系列 VS CSR。

除 `senvcfg`, `scounteren`, `scontext` 外，其他 S-mode 的 CSR 均有一个对应的 VS-mode CSR。

当处于非虚拟化模式 (V=0) 时，VS CSRs 不会对当前系统造成任何影响，但当切换到虚拟化模式 (V=1) 时，处于 VS-mode 访问到的 S-mode CSR 会被替换为 VS CSRs。



中断和异常

RVH 新增了一些用于虚拟化的中断和异常

Interrupt	Exception Code	Description
1	2	Virtual supervisor interrupt
1	6	Virtual supervisor timer interrupt
1	10	Virtual supervisor external interrupt
1	12	Supervisor guest external interrupt
0	10	Environment call from VS-mode
0	20	Instruction guest-page fault
0	21	Load guest-page fault
0	22	Virtual instruction[2]
0	23	Store/AMO guest-page fault

[2] 实现较复杂，不再详细展开。有兴趣可以参考 Spec 8.6.1 节

Two-Stage Address Translation

RVH 添加了一个新的 CSR `hgatp`，其使用方式与 `satp` 类似，用于存储 GPA $\rightarrow$ HPA 的映射。

`hgatp` 有四种分页模式: Sv32x4, Sv39x4, Sv48x4, Sv57x4。

GPA 到 HPA 的翻译过程被称作 G-stage translation。当 G-stage translation 失败，即访问的 GPA 没有对应的 HPA 时，会触发 Guest-Page Fault。

此外，RVH 中包含两个用于刷新 TLB 的指令:
`HFENCE.VVMA` 作用于受 `vsatp` 控制，用于 GVA $\rightarrow$ GPA 翻译的 TLB，
`HFENCE.GVMA` 作用于受 `hgatp` 控制，用于 GPA $\rightarrow$ HPA 翻译的 TLB。

其他细节

RVH 还添加了一些新的访存指令 HLV.*width*, HLVX.HU/WU, HSV.*width*。
这些指令可以使用 GVA 访问 Guest 的内存。

其中，HLV.*width* 和 HSV.*width* 仅会检查该地址的读权限或写权限，但 HLVX.HU/WU 会额外检查运行权限。

当 hstatus.HU=1 时，以上访存指令在 HS-mode 和 U-mode 均可以使用，但当该 bit 为 0 时仅 HS-mode 可以使用。

附录

Xen 的半虚拟化内存管理: X86 Paravirtualised Memory Management

IO 虚拟化的发展

- VirtIO (2008): Virtio interface
- Vhost (2010): vhost_net: a kernel-level virtio server
- VFIO (2012): VFIO
- Vhost-user (2014): Vhost and vhost-net support for userspace based backends
- VFIO-mdev (2016): Add Mediated device support
- vDPA (2020): vDPA support
- VDUSE (2021): Introduce VDUSE - vDPA Device in Userspace

RISC-V ISA Specifications: Specifications

扩展阅读

QEMU TCG 的实现

- TCG IR: qemu/tcg/tcg.c
- RISC-V 基本设定: qemu/tcg/riscv/tcg-target.c.inc
- RISC-V 指令定义: qemu/target/riscv/insn32.decode

  编译时会生成:
  `build/libqemu-riscv64-softmmu.fa.p/decode-insn32.c.inc`
- RVI 指令行为: qemu/target/riscv/insn_trans/trans_rvi.c.inc

Spike 的暴力美学

- 指令 encoding: riscv-isa-sim/riscv/encoding.h
- 所有指令行为: riscv-isa-sim/riscv/insn

Q & A