humjb_1983

博客访问：9500
博文数量：80
博客积分：0
博客等级：民兵
技术积分：685
用 户 组：普通用户
注册时间：2014-02-20 08:27

加关注 短消息
论坛 加好友

文章分类

全部博文（80）
硬件相关（5）
虚拟化（13）
其他（1）
Linux其他方面（3）
Linux内核（57）
未分配的博文（1）

文章存档

2014年（80）

我的朋友

asuka20 321leon

最近访客

arm-linu 码出一片 pisming
jeppeter 刘一痕 SCvsCS

KVM基本原理及架构七-KVM内核模块中重要的数据结构

2014-07-18 19:54:46

分类：LINUX

KVM内核模块中重要数据结构

7.1 kvm

Kvm结构体代表一个具体的虚拟机，当通过KVM_CREATE_VM指令字创建一个虚拟机后，就会创建一个新的kvm结构体对象。Kvm结构体中包括了VCPU、内存、APIC、IRQ、MMU、Event事件等相关信息，该结构体主要在KVM虚拟机内部使用，用于跟踪虚拟机状态。

重要的结构体成员说明如下:

点击(此处)折叠或打开

```
struct kvm {
        …
struct kvm_memslots *memslots; //KVM虚拟机分配的内存slot，用于GPAàHVA的转换，内存虚拟化使用
        …
        struct kvm_vcpu *vcpus[KVM_MAX_VCPUS]; //KVM虚拟机中包含的VCPU结构体数组，一个VCPU对应一个数组成员。
        …
        struct kvm_io_bus *buses[KVM_NR_BUSES]; //KVM虚拟机中包括的IO总线结构体数组，一条总线对应一个kvm_io_bus结构体，如ISA总线、PCI总线。
        …
        struct kvm_vm_stat stat; //KVM虚拟机中的运行时状态信息，比如页表、MMU等状态。
        struct kvm_arch arch; //KVM中跟arch相关的参数。
        …
}

7.2 kvm_run
kvm_run结构体记录了KVM的内部运行状态，如VM-Exit发生的原因等。重要字段说明如下:
```

7.2 kvm_run

kvm_run结构体记录了KVM的内部运行状态，如VM-Exit发生的原因等。重要字段说明如下:

点击(此处)折叠或打开

```
struct kvm_run {
    __u8 request_interrupt_window; //向VCPU注入一个中断，让VCPU做好相关准备工作
    …
    __u8 ready_for_interrupt_injection; //响应request_interrupt_window的中断请求，当设置时，说明VCPU可以接收中断。
    __u8 if_flag; //中断使能标识，如果使用了APIC，则无效
    struct {
            __u64 hardware_exit_reason; //当发生VM-Exit时，该字段保存了由于硬件原因导致VM-Exit的相关信息。
        } hw;
    struct {
#define KVM_EXIT_IO_IN 0
#define KVM_EXIT_IO_OUT 1
            __u8 direction;
            __u8 size; /* bytes */
            __u16 port;
            __u32 count;
            __u64 data_offset; /* relative to kvm_run start */
        } io; //当由于IO操作导致发生VM-Exit时，该结构体保存IO相关信息。
        …
};
```

7.3 kvm_vcpu

在通过KVM_CREATE_VCPU指令字为VM创建VCPU后，KVM模块将创建kvm_vcpu结构体，其中包含了VCPU相关的信息，重要字段说明如下:

点击(此处)折叠或打开

```
struct kvm_run {
    __u8 request_interrupt_window; //向VCPU注入一个中断，让VCPU做好相关准备工作
    …
    __u8 ready_for_interrupt_injection; //响应request_interrupt_window的中断请求，当设置时，说明VCPU可以接收中断。
    __u8 if_flag; //中断使能标识，如果使用了APIC，则无效
    struct {
            __u64 hardware_exit_reason; //当发生VM-Exit时，该字段保存了由于硬件原因导致VM-Exit的相关信息。
        } hw;
    struct {
#define KVM_EXIT_IO_IN 0
#define KVM_EXIT_IO_OUT 1
            __u8 direction;
            __u8 size; /* bytes */
            __u16 port;
            __u32 count;
            __u64 data_offset; /* relative to kvm_run start */
        } io; //当由于IO操作导致发生VM-Exit时，该结构体保存IO相关信息。
        …
};
```

7.4 kvm_x86_ops

kvm_x86_ops结构体只包含了针对具体CPU架构进行虚拟化时的函数指针，在kvm-intel.ko和kvm-amd.ko模块中，提供了不同的接口和实现。在KVM初始化和正常运行过程中，将使用相关接口进行实际的硬件操作。主要包括如下几种类型的操作：

- ü VMM状态初始化
- ü VCPU管理
- ü 中断管理
- ü 寄存器管理
- ü 时钟管理

Kvm_x86_ops结构体在intel架构中初始化为：

点击(此处)折叠或打开

```
static struct kvm_x86_ops vmx_x86_ops = {
        .cpu_has_kvm_support = cpu_has_kvm_support,
        .disabled_by_bios = vmx_disabled_by_bios,
        .hardware_setup = hardware_setup,
        .hardware_unsetup = hardware_unsetup,
        .check_processor_compatibility = vmx_check_processor_compat,
        .hardware_enable = hardware_enable,
        .hardware_disable = hardware_disable,
        .cpu_has_accelerated_tpr = report_flexpriority,
        .vcpu_create = vmx_create_vcpu,
        .vcpu_free = vmx_free_vcpu,
        .vcpu_reset = vmx_vcpu_reset,
        .prepare_guest_switch = vmx_save_host_state,
        .vcpu_load = vmx_vcpu_load,
        .vcpu_put = vmx_vcpu_put,
        .set_guest_debug = set_guest_debug,
        .get_msr = vmx_get_msr,
        .set_msr = vmx_set_msr,
        .get_segment_base = vmx_get_segment_base,
        .get_segment = vmx_get_segment,
        .set_segment = vmx_set_segment,
        .get_cpl = vmx_get_cpl,
        .get_cs_db_l_bits = vmx_get_cs_db_l_bits,
        .decache_cr0_guest_bits = vmx_decache_cr0_guest_bits,
        .decache_cr4_guest_bits = vmx_decache_cr4_guest_bits,
        .set_cr0 = vmx_set_cr0,
        .set_cr3 = vmx_set_cr3,
        .set_cr4 = vmx_set_cr4,
        .set_efer = vmx_set_efer,
        .get_idt = vmx_get_idt,
        .set_idt = vmx_set_idt,
        .get_gdt = vmx_get_gdt,
        .set_gdt = vmx_set_gdt,
        .cache_reg = vmx_cache_reg,
        .get_rflags = vmx_get_rflags,
        .set_rflags = vmx_set_rflags,
        .fpu_deactivate = vmx_fpu_deactivate,
        .tlb_flush = vmx_flush_tlb,
        .run = vmx_vcpu_run,
        .handle_exit = vmx_handle_exit,
        .skip_emulated_instruction = skip_emulated_instruction,
        .set_interrupt_shadow = vmx_set_interrupt_shadow,
        .get_interrupt_shadow = vmx_get_interrupt_shadow,
        .patch_hypercall = vmx_patch_hypercall,
        .set_irq = vmx_inject_irq,
        .set_nmi = vmx_inject_nmi,
        .queue_exception = vmx_queue_exception,
        .interrupt_allowed = vmx_interrupt_allowed,
```

```
49.             .nmi_allowed = vmx_nmi_allowed,
50.             .get_nmi_mask = vmx_get_nmi_mask,
51.             .set_nmi_mask = vmx_set_nmi_mask,
52.             .enable_nmi_window = enable_nmi_window,
53.             .enable_irq_window = enable_irq_window,
54.             .update_cr8_intercept = update_cr8_intercept,
55.             .set_tss_addr = vmx_set_tss_addr,
56.             .get_tdp_level = get_ept_level,
57.             .get_mt_mask = vmx_get_mt_mask,
58.             .exit_reasons_str = vmx_exit_reasons_str,
59.             .gb_page_enable = vmx_gb_page_enable,
60.             .set_tsc_khz = vmx_set_tsc_khz,
61.             .write_tsc_offset = vmx_write_tsc_offset,
62.             .adjust_tsc_offset = vmx_adjust_tsc_offset,
63.             .compute_tsc_offset = vmx_compute_tsc_offset,
64.     };
```


上一篇：KVM基本原理及架构六-KVM API
下一篇：KVM基本原理及架构八-KVM内核模块重要流程分析