邮箱：zhunxun@gmail.com

< 2020年5月 >

日	一	二	三	四	五	六
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

搜索

找找看

谷歌搜索

PostCategories

C语言(2)
IO Virtualization(3)
KVM虚拟化技术(26)
linux 内核源码分析(61)
Linux日常应用(3)
linux时间子系统(3)
qemu(10)
seLinux(1)
windows内核(5)
调试技巧(2)
内存管理(8)
日常技能(3)
容器技术(2)
生活杂谈(1)
网络(5)
文件系统(4)
硬件(4)

PostArchives

2018/4(1)
2018/2(1)
2018/1(3)
2017/12(2)
2017/11(4)
2017/9(3)
2017/8(1)
2017/7(8)
2017/6(6)
2017/5(9)
2017/4(15)
2017/3(5)
2017/2(1)
2016/12(1)
2016/11(11)
2016/10(8)
2016/9(13)

ArticleCategories

时态分析(1)

Recent Comments

1. Re:virtio前端驱动详解
我看了下，Linux-4.18.2中的vp_notify()函数。bool vp_notify(struct virtqueue *vq){ /* we write the queue's selec...
--Linux-inside
2. Re:virtIO之VHOST工作原理简析

KVm中EPT逆向映射机制分析

2017-05-30

前几天简要分析了linux remap机制，虽然还有些许瑕疵，但总算大致分析的比较清楚。今天分析下EPT下的逆向映射机制。EPT具体的工作流程可参考前面博文，本文对于EPT以及其工作流程不做过多介绍，重点介绍逆向映射机制。其实逆向映射机制在最主要的作用就是映射的逆向，说了等于白说，但也不无道理。linux下根据虚拟地址经过页表转换得到物理地址。怎么根据物理地址得到对应的虚拟地址呢？这里便用到了逆向映射。逆向映射有什么用呢？最重要的，在页面换出时，由于物理内存的管理由一套相对独立的机制在负责，根据物理页面的活跃程度，对物理页面进行换出，而此时就需要更新引用了此页面的页表了，否则造成不同步而出错。如果获取对应的物理页面对应的pte的地址呢？内核的做法是先通过逆向映射得到虚拟地址，根据虚拟地址遍历页表得到pte地址。

在KVM中，逆向映射机制的作用是类似的，但是完成的却不是从HPA到对应的EPT页表项的定位，而是从gfn到对应的页表项的定位。理论上讲根据gfn一步步遍历EPT也未尝不可，但是效率较低；况且在EPT所维护的页面不同于host的页表，理论上讲是虚拟机之间是禁止主动的共享内存的，为了提高效率，就有了当前的逆向映射机制。

我们都知道虚拟机的物理内存由多个slot构成，每个slot都是一个kvm_memory_slot结构，表示虚拟机物理内存的一段空间，为了说明问题，不妨先看下该结构：

```
struct kvm_memory_slot {
    gfn_t base_gfn;
    unsigned long npages;
    /*一个slot有许多客户机虚拟页面组成，通过dirty_bitmap标记每一个页是否可用,一个页面对应一个位*/
    unsigned long *dirty_bitmap;
    struct kvm_arch_memory_slot arch;
    unsigned long userspace_addr;//对应的HVA 地址
    u32 flags;
    short id;
};
```

slot本质是qemu进程用户空间的hva,紧急你是qemu进程的虚拟地址空间，并没有对应物理地址，各个字段的意义不言自明了。其中有一个kvm_arch_memory_slot结构，我们重点描述。

```
struct kvm_arch_memory_slot {
    unsigned long *rmap[KVM_NR_PAGE_SIZES];
    struct kvm_lpage_info *lpage_info[KVM_NR_PAGE_SIZES - 1];
};
```

该结构的rmap字段是指针数组，每种页面大小对应一项，截止3.10.1版本，KVM的大页面仅仅支持2M而并没有考虑1G的页面，普通的页面就是4KB了。所以默认状态下，提到大页面就是指的2M的页面。结合上面的kvm_memory_slot结构可以发现，kvm_arch_memory_slot其实是kvm_memory_slot的一个内嵌结构，所以每个slot都关联一个kvm_arch_memory_slot，也就有一个rmap数组。其实在虚拟机中，qemu为虚拟机分配的页面主要是大页面，但是这里为了方面，按照4KB的普通页面做介绍。

初始化阶段

在qemu为虚拟机注册各个slot的时候，在KVM中会初始化逆向映射的相关内存区。
__kvm_set_memory_region-->kvm_arch_create_memslot

在该函数中，用一个for循环为每种页面类型的rmap分配空间，具体分配代码如下

```
lpages = gfn_to_index(slot->base_gfn + npages - 1,
                      slot->base_gfn, level) + 1;

        slot->arch.rmap[i] =
            kvm_kvzalloc(lpages * sizeof(*slot->arch.rmap[i]));
        if (!slot->arch.rmap[i])
            goto out_free;
```

再问一个问题，从设置ioeventfd那个流程来看的话是guest发起一个IO，首先会陷入到kvm中，然后由kvm向qemu发送一个IO到来的event，最后IO才被处理，是这样的吗?

--Linux-inside

3. Re:virtIO之VHOST工作原理简析

你好。设置ioeventfd这个部分和guest里面的virtio前端驱动有关系吗?
设置ioeventfd和virtio前端驱动是如何发生联系起来的？谢谢。

--Linux-inside

4. Re:QEMU IO事件处理框架

良心博主，怎么停跟了，太可惜了。

--黄铁牛

5. Re:linux 逆向映射机制浅析

小哥哥520脱单了么

--黄铁牛

Top Posts

1. 详解操作系统中断(21154)
2. PCI 设备详解一(15808)
3. 进程的挂起、阻塞和睡眠(13714)
4. Linux下桥接模式详解一(13467)
5. virtio后端驱动详解(10539)

推荐排行榜

1. 进程的挂起、阻塞和睡眠(6)
2. qemu-kvm内存虚拟化1(2)
3. 为何要写博客(2)
4. virtIO前后端notify机制详解(2)
5. 详解操作系统中断(2)

gfn_to_index把一个gfn转化成该gfn在整个slot中的索引，而这里获取的其实就是整个slot包含的不同level的页面数。然后为slot->arch.rmap[i]分配内存，每个页面对应一个unsigned Long.

建立阶段

建立阶段自然是在填充EPT的时候了，在KVM中维护EPT的核心函数是tdp_page_fault函数。该函数的处理在之前的文章中也有介绍，在函数尾部会调用rmap_add函数建立逆向映射

```
static int rmap_add(struct kvm_vcpu *vcpu, u64 *spte, gfn_t gfn)
{
    struct kvm_mmu_page *sp;
    unsigned long *rmapp;

    sp = page_header(__pa(spte));
    kvm_mmu_page_set_gfn(sp, spte - sp->spt, gfn);
    rmapp = gfn_to_rmap(vcpu->kvm, gfn, sp->role.level);
    return pte_list_add(vcpu, spte, rmapp);
}
```

page_header是一个内联函数，主要目的在于获取kvm_mmu_page，一个该结构描述一个层级的页表，地址保存在page结构的private字段，然后调用kvm_mmu_page_set_gfn，对kvm_mmu_page进行设置。这不是重点，接着就获取了gfn对应的rmap的地址，重点看下

```
static unsigned long *gfn_to_rmap(struct kvm *kvm, gfn_t gfn, int level)
{
    struct kvm_memory_slot *slot;

    slot = gfn_to_memslot(kvm, gfn);
    return __gfn_to_rmap(gfn, level, slot);
}
```

首先转化成到对应的slot，然后调用了__gfn_to_rmap

```
static unsigned long *__gfn_to_rmap(gfn_t gfn, int level,
                    struct kvm_memory_slot *slot)
{
    unsigned long idx;
    /*gfn在slot中的index*/
    idx = gfn_to_index(gfn, slot->base_gfn, level);
    /*rmap是一个指针数组，每个项记录对应层级的gfn对应的逆向映射，index就是下标*/
    return &slot->arch.rmap[level - PT_PAGE_TABLE_LEVEL][idx];
}
```

额。。。到这里就很明确了，我们再次看到了gfn_to_index函数，这里就根据指定的gfn转化成索引，同时也是在rmap数组的下标，然后就返回对应的表项的地址，没啥好说的吧.....现在地址已经获取到了，还等什么呢？设置吧，调用pte_list_add函数，该函数也值得一说

```
static int pte_list_add(struct kvm_vcpu *vcpu, u64 *spte,
            unsigned long *pte_list)
{
    struct pte_list_desc *desc;
    int i, count = 0;
    /*如果*pte_list为空，直接设置逆向映射即可 */
    if (!*pte_list) {
        rmap_printk("pte_list_add: %p %llx 0->1\n", spte, *spte);
        *pte_list = (unsigned long)spte;
    } else if (!(*pte_list & 1)) {
        rmap_printk("pte_list_add: %p %llx 1->many\n", spte, *spte);
        desc = mmu_alloc_pte_list_desc(vcpu);
        desc->sptes[0] = (u64 *)*pte_list;
        desc->sptes[1] = spte;
        *pte_list = (unsigned long)desc | 1;
```

```
        ++count;
    } else {
        rmap_printk("pte_list_add: %p %llx many->many\n", spte, *spte);
        desc = (struct pte_list_desc *)(*pte_list & ~1ul);
        while (desc->sptes[PTE_LIST_EXT-1] && desc->more) {
            desc = desc->more;
            count += PTE_LIST_EXT;
        }
        /*如果已经满了，就再次扩展more*/
        if (desc->sptes[PTE_LIST_EXT-1]) {
            desc->more = mmu_alloc_pte_list_desc(vcpu);
            desc = desc->more;
        }
        /*找到首个为空的项，进行填充*/
        for (i = 0; desc->sptes[i]; ++i)
            ++count;
        desc->sptes[i] = spte;
    }
    return count;
}
```

先走下函数流程，我们已经传递进来gfn对应的rmap的地址，就是pte_list，接下来主要分为三部分；if……else if ……else

首先，如果*ptelist为空，则直接*pte_list = (unsigned long)spte;直接把rmap地址的内容设置成表项地址，到这里为止，so easy……但是这并不能解决所有问题，说到这里看下函数前面的注释吧

```
/*
 * Pte mapping structures:
 *
 * If pte_list bit zero is zero, then pte_list point to the spte.
 *
 * If pte_list bit zero is one, (then pte_list & ~1) points to a struct
 * pte_list_desc containing more mappings.
 *
 * Returns the number of pte entries before the spte was added or zero if
 * the spte was not added.
 *
 */
```

根据注释判断，pte_list即我们之前的到的rmap最低一位表明这直接指向一个spte还是pte_list_desc，后者用作扩展remap.那么到了else if这里，如果*pte_list不为空且也并没有指向一个pte_list_desc，那么就坏了，根据gfn定位到了 这个remap项，但是人家已经在用了，怎么办？解决方案就是通过pte_list_desc扩展下，但是最后要表明这是一个pte_list_desc，所以要吧最后一位置1，然后设置进*pte_list。还是介绍下该结构

```
struct pte_list_desc {
    u64 *sptes[PTE_LIST_EXT];
    struct pte_list_desc *more;
};
```

结构比较简单，自身携带一个PTE_LIST_EXT大小的指针数组，PTE_LIST_EXT为3，也就是扩展一下可以增加2个表项，数量不多，所以如果还不够，就通过下面的more扩展。more又指向一个pte_list_desc。好了，接下看我们的else

如果前两种情况都不对，这就是remap项不为空，且已经指向一个pte_list_desc，同样的道理我们需要获取该结构,找到一个能用的地方啊。如何找?

如果desc->sptes已经满了，且more不为空，则递归的遍历more，while循环出来，就有两种情况

1、sptes有剩余

2、more为空

此时进行判断，如果sptes没满，直接找到一个空闲的项，进行填充；否则，申请一个pte_list_desc，通过more进行扩展，然后在寻找一个空闲的。

PS:上面是函数的大致流程，可是为何需要扩展呢？之前有提到，初始化的时候为每个页面都分配了remap空间，如果qemu进程为虚拟机分配的都是4KB的页面，那么每个页面均会对应一个位置，这样仅仅if哪里就可以了，不需要扩展。但是qemu为虚拟机分配的一般是比较大的页面，就是2M的，但是虚拟机自己分配的很可能是4KB的，这样，初始化的时候为2M的页为单位分配rmap空间，就不能保证所有的小页面都对应一个唯一的remap地址，这样就用到了扩展。

以马内利

参考：kvm 3.10.1源码

分类: KVM虚拟化技术

好文要顶 关注我 收藏该文

jack.chen
关注 - 12
粉丝 - 44
+加关注

0 0

« 上一篇： linux内核获取当前进程路径分析
» 下一篇： linux伙伴系统接口alloc_page分析1

posted @ 2017-05-30 16:52 jack.chen Views(1498) Comments(1) Edit 收藏

Post Comment

#1楼 2018-08-27 11:44 | CheneyLin

从gfn找到页表项最多只要查询访问4次(4级页表)，为啥还要用反向映射，况且建立rmap还得有一定内存开销，并没有提高太多效率吧?

支持(0) 反对(

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册， 访问 网站首页。

相关博文：
· intel EPT 机制详解
· Qemu创建KVM虚拟机内存初始化流程
· Linux内存管理 (2)页表的映射过程
· 内存虚拟化
· 【内核】Linux 2.6 内存反向映射机制 Reverse Mapping
» 更多推荐...

最新 IT 新闻:
· 腾讯在列！微软宣布超140家工作室为Xbox Series X开发游戏
· 黑客声称从微软GitHub私人数据库当中盗取500GB数据
· IBM开源用于简化AI模型开发的Elyra工具包
· 中国网民人均安装63个App：腾讯系一家独大
· Lyft颁布新规：强制要求乘客和司机佩戴口罩
» 更多新闻...

以马内利