## virtIO之VHOST工作原理简析

2017-07-19

一、前言

之前有分析过虚拟化环境下virtIO的实现，virtIO相关于传统的虚拟IO在性能方面的确提高了不少，但是按照virtIO虚拟网卡为例，每次虚拟机接收数据包的时候，数据包从linux bridge经过tap设备发送到用户空间，这是一层数据的复制并且伴有内核到用户层的切换，而在用户空间virtIO后端驱动把数据写入到虚拟机内存后还需要退到KVM中，从KVM进入虚拟机，又增加了一次模式的切换。在IO比较频繁的情况下，会造成模式切换次数过多从而降低性能。而vhost便解决了这个问题。把后端驱动从qemu中迁移到内核中，作为一个独立的内核模块存在，这样在数据到来的时候，该模块直接监听tap设备，在内核中直接把数据写入到虚拟机内存中，然后通知虚拟机即可，这样就和qemu解耦和，减少了模式切换次数和数据复制次数，提高了性能。下面介绍下vhost的初始化流程。

二、 整体框架

介绍VHOST主要从三个部分入手：vHOST内核模块，qemu部分、KVM部分。而qemu部分主要是virtIO部分。本节不打算分析具体的工作代码，因为基本原理和VIRTIO类似，且要线性的描述vhost也并非易事。

1、vHOST 内核模块

vhost内核模块主要是把virtiO后端驱动的数据平面迁移到了内核中，而控制平面还在qemu中，所以就需要一些列的注册把相关信息记录在内核中，如虚拟机内存布局，设备关联的eventfd等。虽然KVM中有虚拟机的内存布局，但是由于vhost并非在KVM中，而是单独的一个内核模块，所以需要qemu单独处理。且目前vhost只支持网络部分，块设备等其他部分尚不支持。内核中两个文件比较重要：vhost.c和vhost-net.c。其中前者实现的是脱离具体功能的vhost核心实现，后者实现网络方面的功能。内核模块加载主要是初始化vhost-net，起始于vhost_net_init（vhost/net.c）

```
static const struct file_operations vhost_net_fops = {
    .owner          = THIS_MODULE,
    .release        = vhost_net_release,
    .unlocked_ioctl = vhost_net_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl   = vhost_net_compat_ioctl,
#endif
    .open           = vhost_net_open,
    .llseek         = noop_llseek,
};
```

函数表中vhost_net_open和vhost_net_ioctl两个函数需要注意，简单来讲，前者初始化，后者控制，当然是qemu通过ioctl进行控制。那么初始化主要是初始化啥呢?

主要有vhost_net(抽象代表vhost net部分)、vhost_dev（抽象的vhost设备），vhost_virtqueue。基本初始化的流程我们就不介绍，感兴趣可以参考代码，一个VM即一个qemu进程只有一个vhost-net和一个vhost-dev，而一个vhost-dev可以关联多个vhost_virtqueue。一般而言vhost_virtqueue作为一个结构嵌入到具体实现的驱动中，就网络而言vhost_virtqueue嵌入到了vhost_net_virtqueue。初始化最重要的任务就是初始化vhost_poll。在vhost_net_open的尾部，有如下两行代码

```
vhost_poll_init(n->poll + VHOST_NET_VQ_TX, handle_tx_net, POLLOUT, dev);
vhost_poll_init(n->poll + VHOST_NET_VQ_RX, handle_rx_net, POLLIN, dev);
```

在分析函数代码之前，先看下vhost_poll结构

```
struct vhost_poll {
    poll_table                table;
    wait_queue_head_t        *wqh;
    wait_queue_t              wait;
    struct vhost_work      work;
    unsigned long          mask;
```


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)

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)

```
    struct vhost_dev     *dev;
};
```

结合上篇poll机制的文章，这些字段就不难理解，table是包含一个函数指针，在驱动的poll函数中被调用，主要用于把当前进程加入到等待队列。wqh是一个等待队列头。wait是一个等待实体，其包含一个函数作为唤醒函数，vhost_work是poll机制处理的核心任务，参考上面就是处理网络数据包，其中有函数指针指向用户设置的处理函数，这里就是handle_tx_net和handle_rx_net,mask指定什么情况下进行处理，主要是POLL_IN和POLL_OUT，dev就指向依附的vhost-dev设备。结合这些介绍分析vhost_poll_init就无压力了。

看下vhost_poll_init函数的代码

```
void vhost_poll_init(struct vhost_poll *poll, vhost_work_fn_t fn,
             unsigned long mask, struct vhost_dev *dev)
{
    init_waitqueue_func_entry(&poll->wait, vhost_poll_wakeup);
    init_poll_funcptr(&poll->table, vhost_poll_func);
    poll->mask = mask;
    poll->dev = dev;
    poll->wqh = NULL;
    /*设置处理函数*/
    vhost_work_init(&poll->work, fn);
}
```

代码来看很简单，意义需要解释下，每个vhost_net_virtqueue都有自己的vhost_poll，该poll是监控数据的核心机制，而现阶段仅仅是初始化。vhost_poll_wakeup是自定义的等待队列唤醒函数，在对某个描述符poll的时候会把vhost_poll加入到对应描述符的等待队列中，而该函数就是描述符有信号时的唤醒函数，唤醒函数中会验证当前信号是否满足vhost_poll对应的请求掩码，如果满足调用vhost_poll_queue->vhost_work_queue,该函数如下

```
void vhost_work_queue(struct vhost_dev *dev, struct vhost_work *work)
{
    unsigned long flags;

    spin_lock_irqsave(&dev->work_lock, flags);
    if (list_empty(&work->node)) {
        /*把vhost_work加入到设备的工作链表，该链表会在后台线程中遍历处理*/
        list_add_tail(&work->node, &dev->work_list);
        work->queue_seq++;
        /*唤醒工作线程*/
        wake_up_process(dev->worker);
    }
    spin_unlock_irqrestore(&dev->work_lock, flags);
}
```

该函数会把vhost_work加入到设备的工作队列，然后唤醒vhost后台线程vhost_worker，vhost_worker会遍历设备的工作队列，调用work->fn即之前我们注册的处理函数handle_tx_net和handle_rx_net，这样数据包就得到了处理。

**vhost_net_ioctl控制信息**

vhost控制接口通过一系列的API指定相应的操作，下面列举一部分

VHOST_GET_FEATURES

VHOST_SET_FEATURES

这两个用于获取设置vhost一些特性

VHOST_SET_OWNER //设置vhost后台线程，主要是创建一个线程绑定到vhost_dev,而线程的处理函数就是vhost_worker

VHOST_RESET_OWNER //重置OWNER

VHOST_SET_MEM_TABLE //设置guest内存布局信息

VHOST_NET_SET_BACKEND //

VHOST_SET_VRING_KICK //设置guest notify guest->host

VHOST_SET_VRING_CALL //设置host notify host->guest

2、qemu部分

前面介绍的都是内核的任务，而内核是为用户提供服务的，除了vhost内核模块加载时候主动执行一些初始化函数，后续的都是由qemu中发起请求，内核才去响应。这也正是qemu维持控制平面的表现之一。qemu中相关代码的介绍不介绍太多，只给出相关主线，感兴趣可以自行参考。这里我们主要通过qemu讨论下host和guest的通知机制，即irqfd和IOeventfd的初始化。先介绍下irqfd和IOeventfd的任务。

irqfd是KVM为host通知guest提供的中断注入机制，vhost使用此机制通知客户机某些任务已经完成，需要客户机着手处理。而IOevnetfd是guest通知host的方式，qemu会注册一段IO地址区间，PIO或者MMIO，这段地址区间的读写操作都会发生VM-exit，继而在KVM中处理。详细内容下面介绍

irqfd的初始化流程如下：

virtio_net_class_init

virtio_net_device_init virtio-net.c

virtio_init virtio.c

virtio_vmstate_change

virtio_set_status

virtio_net_set_status

virtio_net_vhost_status

vhost_net_start

virtio_pci_set_guest_notifiers //为guest_notify设置eventfd

kvm_virtio_pci_vector_use

kvm_virtio_pci_irqfd_use

kvm_irqchip_add_irqfd_notifier

kvm_irqchip_assign_irqfd

kvm_vm_ioctl(s, KVM_IRQFD, &irqfd);

//向kvm发起ioctl请求

IOeventfd工作流程如下：

virtio_ioport_write

virtio_pci_start_ioeventfd

virtio_pci_set_host_notifier_internal //

memory_region_add_eventfd

memory_region_transaction_commit

address_space_update_topology

address_space_update_ioeventfds

address_space_add_del_ioeventfds

eventfd_add=kvm_mem_ioeventfd_add

kvm_all.c

kvm_set_ioeventfd_mmio

kvm_vm_ioctl(kvm_state,

KVM_IOEVENTFD, &iofd);

3、KVM部分

KVM部分实现对上面ioctl的响应，在kvm_main.c的kvm_vm_ioctl里面，先看KVM_IRQFD的处理

kvm_irqfd->kvm_irqfd_assign,kvm_irqfd_assign函数比较长，我们主要介绍下核心功能

函数在内核生成一个_irqfd结构，首先介绍下_irqfd的工作机制

```
struct _irqfd {
    /* Used for MSI fast-path */
    struct kvm *kvm;
    wait_queue_t wait;
    /* Update side is protected by irqfds.lock */
    struct kvm_kernel_irq_routing_entry __rcu *irq_entry;
    /* Used for level IRQ fast-path */
    int gsi;
    struct work_struct inject;
    /* The resampler used by this irqfd (resampler-only) */
    struct _irqfd_resampler *resampler;
    /* Eventfd notified on resample (resampler-only) */
```

```
    struct eventfd_ctx *resamplefd;
    /* Entry in list of irqfds for a resampler (resampler-only) */
    struct list_head resampler_link;
    /* Used for setup/shutdown */
    struct eventfd_ctx *eventfd;
    struct list_head list;
    poll_table pt;
    struct work_struct shutdown;
};
```

kvm是关联的虚拟机，wait是一个等待队列对象，允许irqfd等待某个信号，irq_entry是中断路由表，属于中断虚拟化部分，本节不作介绍。gsi是全局的中断号，很重要。inject是一个工作对象，resampler是确认中断处理的，不做考虑。eventfd是其关联的evnetfd,这里就是guestnotifier.在kvm_irqfd_assign函数中，给上面inject和shutdown都关联了函数

```
INIT_WORK(&irqfd->inject, irqfd_inject);
INIT_WORK(&irqfd->shutdown, irqfd_shutdown);
```

这些函数实现了irqfd的简单功能，前者实现了中断的注入，后者禁用irqfd。irqfd初始化好后，对于irqfd关联用户空间传递的eventfd，之后忽略中间的resampler之类的处理，初始化了irqfd等待队列的唤醒函数irqfd_wakeup和核心poll函数irqfd_ptable_queue_proc，接着就调用irqfd_update更新中断路由项目，中断虚拟化的代码单独开一篇文章讲解，下面就该调用具体的poll函数了，这里是file->f_op->poll(file, &irqfd->pt);，实际对应的就是eventfd_poll函数，里面会调用poll_table->_qproc,即irqfd_ptable_queue_proc把irqfd加入到描述符的等待队列中，可以看到这里吧前面关联的eventfd加入到了poll列表，当该eventfd有状态时，唤醒函数irqfd_wakeup就得到调用，其中通过工作队列调度irqfd->inject，这样irqfd_inject得到执行，中断就被注入，具体可以参考vhost_add_used_and_signal_n函数，在从guest接收数据完毕就会调用该函数通知guest。

IOEVENTFD

内核里面起始于kvm_ioeventfd->kvm_assign_ioeventfd,这里相对于上面就比较简单了，主要是注册一个IO设备，绑定一段IO地址区间，给设备分配操作函数表，其实就两个函数

```
static const struct kvm_io_device_ops ioeventfd_ops = {
    .write      = ioeventfd_write,
    .destructor = ioeventfd_destructor,
};
```

而当guest内部完成某个操作，如填充好了skbuffer后，就需要通知host，此时在guest内部最终就归结于对设备的写操作，写操作会造成VM-exit继而陷入到VMM中进行处理，PIO直接走的IO陷入，而MMIO需要走EPT violation的处理流程，最终就调用到设备的写函数，这里就是ioeventfd_write，看下该函数的实现

```
static int
ioeventfd_write(struct kvm_io_device *this, gpa_t addr, int len,
        const void *val)
{
    struct _ioeventfd *p = to_ioeventfd(this);

    if (!ioeventfd_in_range(p, addr, len, val))
        return -EOPNOTSUPP;

    eventfd_signal(p->eventfd, 1);
    return 0;
}
```

实现很简单，就是判断地址是否在该段Io地址区间内，如果在就调用eventfd_signal，给该段IOeventfd绑定的eventfd一个信号，这样在该eventfd上等待的对象就会得到处理。

以马内利!

参考资料:

linux3.10.1源码

KVM源码

qemu源码

分类: qemu, KVM虚拟化技术, linux 内核源码分析

好文要顶 关注我 收藏该文

jack.chen
关注 - 12
粉丝 - 44
+加关注

1 0

« 上一篇： linux IO多路复用POLL机制深入分析
» 下一篇： KVM中断虚拟化浅析

posted @ 2017-08-23 09:18 jack.chen Views(6435) Comments(4) Edit 收藏

**Post Comment**

#1楼 2017-11-24 15:44 | avalanche

您好，看了您关于virtio/vhost的博客，相当收益，想请问一下，vhost在数据路径上是否就仅仅是将包转发给TAP设呢？据我所知，除了转发之后，可能还有处理virtio_net_hdr这样的virtio头部信息，除了这两者之外vhost还有什么重要的数据处理吗？

支持(0) 反对(

#2楼 [楼主 ] 2017-11-27 15:36 | jack.chen

你好，根据代码来看内核做的处理很简单，在vhost目录下的net.c文件中可以查看，在handle_tx中，仅仅是virtio包头就sendmsg了，并没有其他额外的操作!

支持(0) 反对(

#3楼 2019-12-27 13:58 | Linux-inside

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。

支持(0) 反对(

#4楼 2019-12-27 14:01 | Linux-inside

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?

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。**

以马内利