# virtio前端通知机制分析

2018-08-27 | 2.虚拟化， virtio | 1592 | 2
4.8k | 4 分钟

## 前言

Virtio驱动采用IO的方式来notify后端，下面我们来分析从driver开始通知到后端感知到中间都经历了哪些过程。

## 正文

### Guest->KVM

还记得Virtio网络发包过程分析中，driver在**start_xmit**的最后调用了**virtqueue_kick**函数来通知device，我们从这里开始分析。代码如下，可以看到函数首先调用了**virtqueue_kick_prepare**判断当前是否需要kick，如果需要则**virtqueue_notify**。

```
1  bool virtqueue_kick(struct virtqueue *vq)
2  {
3      if (virtqueue_kick_prepare(vq))
4          return virtqueue_notify(vq);
5      return true;
6  }
```

先分析**virtqueue_kick_prepare**函数，函数获取了old(上次kick后的vring.avail->idx)和new(当前idx)，vq->num_added用来表示两者差值。如果vq->event=1则**vring_need_event**。

```
1  bool virtqueue_kick_prepare(struct virtqueue *_vq)
2  {
3      struct vring_virtqueue *vq = to_vvq(_vq);
4  /*old表示上次kick后的vring.avail->idx;new表示当前idx*/
5      u16 new, old;
6      bool needs_kick;
7
8  ↑ 81%  START_USE(vq);
```

```
9         /* We need to expose available array entries before checking avail
10         * event. */
11        virtio_mb(vq->weak_barriers);
12
13        old = virtio16_to_cpu(_vq->vdev, vq->vring.avail->idx) - vq->num_added;
14        new = virtio16_to_cpu(_vq->vdev, vq->vring.avail->idx);
15        vq->num_added = 0;
16
17    #ifdef DEBUG
18        if (vq->last_add_time_valid) {
19            WARN_ON(ktime_to_ms(ktime_sub(ktime_get(),
20                                vq->last_add_time)) > 100);
21        }
22        vq->last_add_time_valid = false;
23    #endif
24    /*当VIRTIO_RING_F_EVENT_IDX被置位vq->event为1*/
25        if (vq->event) {
26            needs_kick = vring_need_event(virtio16_to_cpu(_vq->vdev, vring_avail_event(&vq->vring)),
27                                new, old);
28        } else {
29            needs_kick = !(vq->vring.used->flags & cpu_to_virtio16(_vq->vdev, VRING_USED_F_NO_NOT
30        }
31        END_USE(vq);
32        return needs_kick;
33    }
```

> vq->event 的值与 VIRTIO_RING_F_EVENT_IDX （Virtio PCI Card Specification Version 0.9.5中定义此字段）有关。
> Virtual I/O Device (VIRTIO) Version1.0 规范中 VIRTIO_RING_F_EVENT_IDX 变为 VIRTIO_F_EVENT_IDX，1.0规范中对此字段介绍的更为详细，具体可以看VirtIO SPEC文档2.4.9节。

下面来看**vring_need_event**，先看一下传进来的参数event_idx的含义，event_idx实际上为used ring的最后一个元素的值。代码如下，如果**(u16)(new_idx - event_idx - 1) < (u16)**(new_idx - old)成立说明backend的处理速度够快，那么返回true表示可以kick backend，否则说明backend当前处理的位置event_idx落后于old，此时backend处理速度较慢，返回false等待下次一起kick backend。

> 对于VirtIO的机制来说，backend一直消耗avail ring，frontend一直消耗used ring，因此backend用used ring的last entry告诉frontend自己当前处理到哪了。

81%

```
#define vring_avail_event(vr) (*(__virtio16 *)&(vr)->used->ring[(vr)->num])
```

```
static inline int vring_need_event(__u16 event_idx, __u16 new_idx, __u16 old)
{
    /* Note: Xen has similar logic for notification hold-off
     * in include/xen/interface/io/ring.h with req_event and req_prod
     * corresponding to event_idx + 1 and new_idx respectively.
     * Note also that req_event and req_prod in Xen start at 1,
     * event indexes in virtio start at 0. */
    return (__u16)(new_idx - event_idx - 1) < (__u16)(new_idx - old);
}
```

回到**virtqueue_kick**函数，我们接着来分析**virtqueue_notify**，**virtqueue_notify**调用了vq->notify(_vq)，notify定义在struct ving_virtqueue中，**notify**具体是哪个函数是在**setup_vq**中创建virtqueue时绑定的。

```
bool (*notify)(struct virtqueue *vq);
```

```
bool virtqueue_notify(struct virtqueue *_vq)
{
    struct vring_virtqueue *vq = to_vvq(_vq);

    if (unlikely(vq->broken))
        return false;

    /* Prod other side to tell it about changes. */
    if (!vq->notify(_vq)) {
        vq->broken = true;
        return false;
    }
    return true;
}
```

下面我们分析**setup_vq**来看看**notify**到底是什么，**vring_create_virtqueue**函数绑定**notify**为**vp_notify**，注意一下vq->priv的值下文分析要用到。

```
static struct virtqueue *setup_vq(struct virtio_pci_device *vp_dev,
                  struct virtio_pci_vq_info *info,
                  unsigned index,
                  void (*callback)(struct virtqueue *vq),
                  const char *name,
                  u16 msix_vec)
{
```

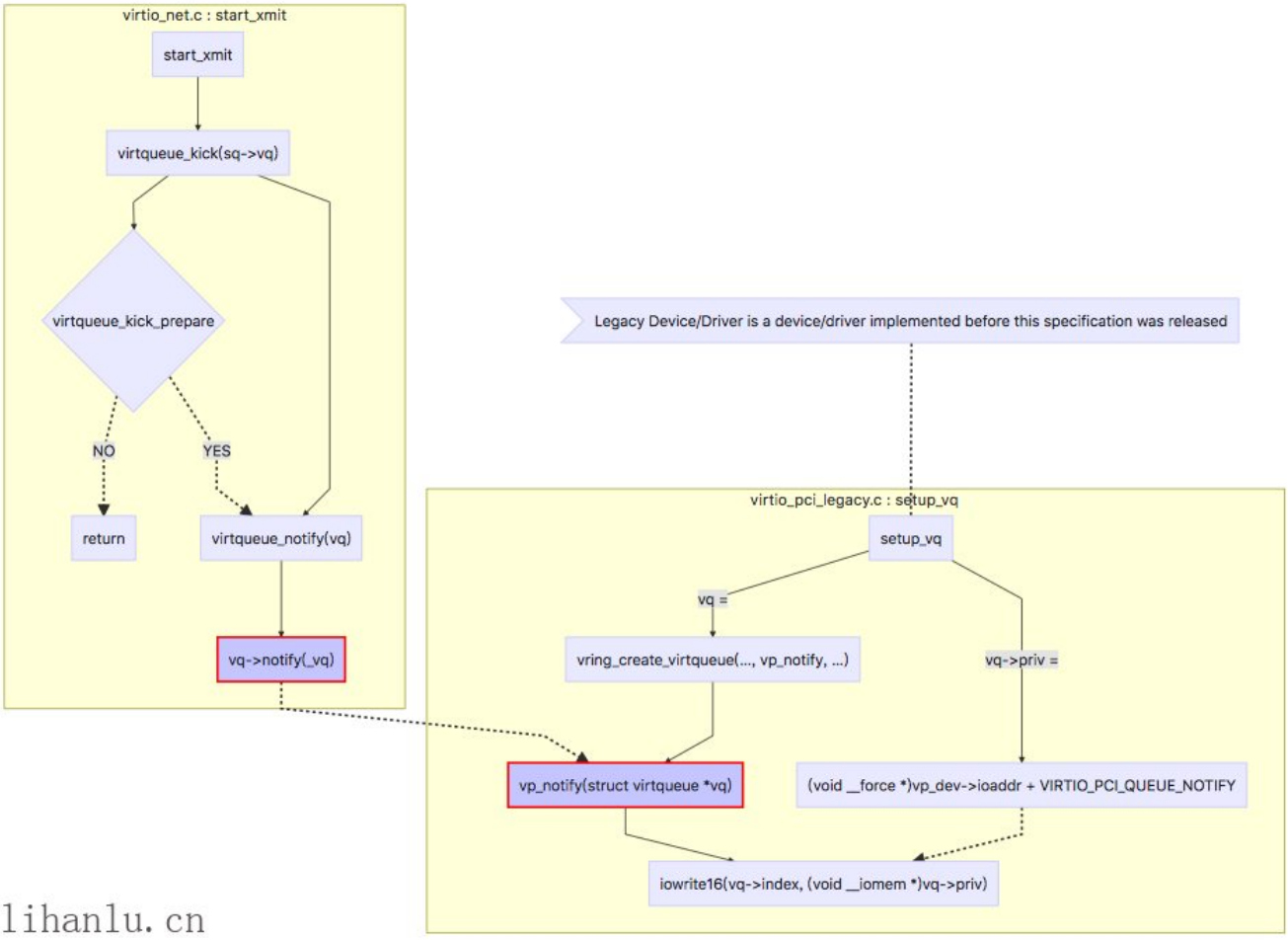
```
8
9   ……
10      /* create the vring */
11      vq = vring_create_virtqueue(index, num,
12                      VIRTIO_PCI_VRING_ALIGN, &vp_dev->vdev,
13                      true, false, vp_notify, callback, name);
14  ……
15
16      vq->priv = (void __force *)vp_dev->ioaddr + VIRTIO_PCI_QUEUE_NOTIFY;
17  ……
18  }
```

可以看到**vp_notify**函数中写**VIRTIO_PCI_QUEUE_NOTIFY**(A 16-bit r/w queue notifier)来达到通知的目的.

```
1   bool vp_notify(struct virtqueue *vq)
2   {
3       /* we write the queue's selector into the notification register to
4        * signal the other end */
5       iowrite16(vq->index, (void __iomem *)vq->priv);
6       return true;
7   }
```

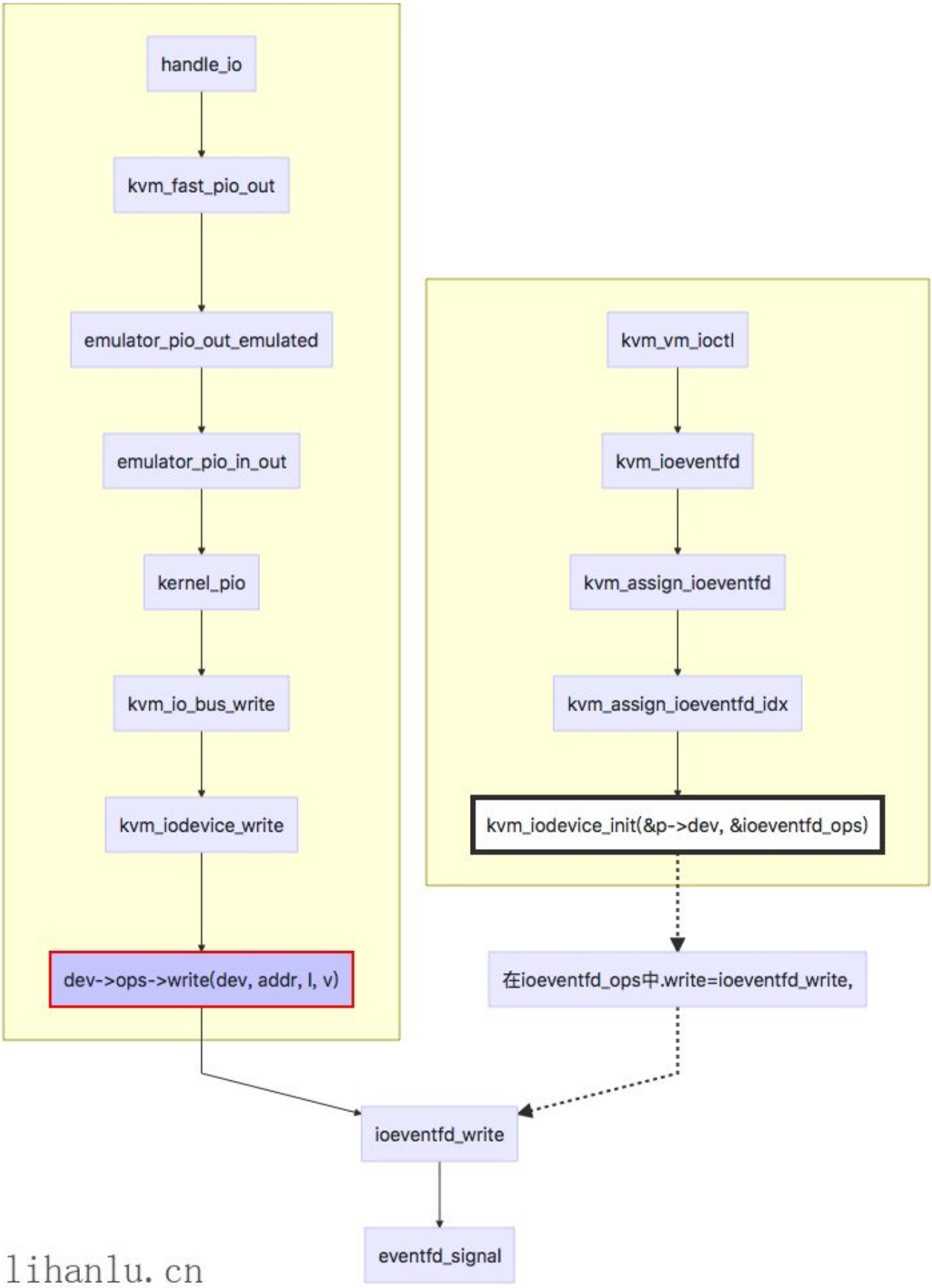
## 本节总结

上述过程总结如下图，首先**virtqueue_kick_prepare**根据feature bit以及后端的处理速度来判断时候需要通知，如果需要则调用**vp_notify**,在其中iowrite VIRTIO_PCI_QUEUE_NOTIFY




## KVM->QEMU

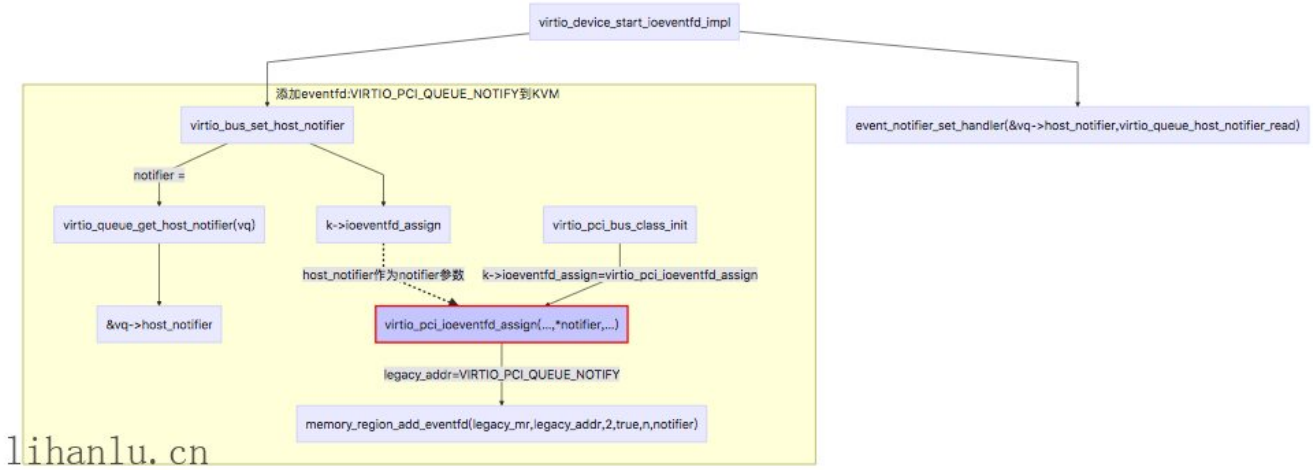
iowrite VIRTIO_PCI_QUEUE_NOTIFY后会产生一个**VM exit**，KVM会判断exit_reason，IO操作对应的执行函数是**handle_io()**，此部分的代码比较多，就不占用篇幅进行详细分析了，下面列举出函数的调用流程，感兴趣的小伙伴可以结合代码看一看，如图，最后会调用**eventfd_signal**唤醒QEMU中的poll。




## Backend

最后看一下Backend的处理，首先在后端是如何**把VIRTIO_PCI_QUEUE_NOTIFY**和eventfd联系起来的？如下图，**memory_region_add_eventfd**函数会和**kvm**来协商使用哪一个

eventfd进行kick通信，而event_notifier_set_handler函数把IOHandler函数virtio_queue_host_notifier_read和host_notifier添加到AioHandler中，也就是说当前端kick后，后端会执行virtio_queue_host_notifier_read函数，到这里我们就和上一篇文章的发包过程结合起来了。




## 联系我

你可以直接在下方留言，也可以E-Mail联系我。

打赏


**本文作者：** Lauren

**本文链接：** http://lihanlu.cn/virtio-frontend-kick/

**版权声明：** 本博客所有文章除特别声明外，均采用 BY-NC-SA 许可协议。转载请注明出处！


virtio


Virtio网络发包过程分析

利用Qemu GDB调试Kernel


昵称 邮箱 网址(http://)

81%

Just go go

提交

## 2 评论

Anonymous Chrome 73.0.3683.75 Linux

回复

写得很好

Anonymous Chrome 71.0.3578.98 Windows 7

回复

分析的很详细，感谢

Powered By Valine
v1.4.4

京ICP备19012335号-1
© 2017 – 2020 Lauren | 92k | 1:23
由 Hexo & NexT.Mist 强力驱动