技术之美

lvyilong316

博客访问： 1966180
博文数量： 167
博客积分： 0
博客等级： 民兵
技术积分： 6734
用 户 组： 普通用户
注册时间： 2013-01-23 18:56

加关注 短消息
论坛 加好友

个人简介

将晦涩难懂的技术讲的通俗易懂

文章分类

全部博文（167）
容器技术（3）
spdk（1）
SDN（0）
论文（1）
论文（0）
网络大二层（1）
性能优化（4）
虚拟化（13）
体系结构（2）
dpdk（17）
链接、装载与库（1）
python（1）
内核网络（12）
分布式存储（1）
Nginx（1）
经典算法（1）
学习生活（1）
Web（2）
网络编程（8）
疑难杂症（5）
Linux系统使用（5）
算法（2）
ASP.NET（1）

qemu和vhost-user前后端协商过程

分类： LINUX 2018-06-24 21:39:48

qemu和vhost-user前后端协商过程

这篇文章主要从qemu的角度分析虚拟机启动前后端的协商过程。虚拟机当后端使用dpdk vhost-user时整个前后端过程可以分为三个阶段：qemu启动阶段，前端驱动加载写VIRTIO_PCI_GUEST_FEATURES寄存器，前端驱动加载完成写VIRTIO_PCI_STATUS寄存器。我们

我们这里主要分析qemu和dpdk vhost_user的交互逻辑**（代码：qemu2.10）**。

qemu启动阶段

qemu启动后，dpdk vhost_user会和qemu建立vhost socket链接，连接建立成功后qemu会调用net_vhost_user_event函数。

net_vhost_user_event

点击(此处)折叠或打开

```
static void net_vhost_user_event(void *opaque, int event)
{
    const char *name = opaque;
                    /* 定义多个NetClientState结构，每个队列一个 */
    NetClientState *ncs[MAX_QUEUE_NUM];
    VhostUserState *s;
    Chardev *chr;
    Error *err = NULL;
    int queues;
    /* 从qemu参数中获取后端设备定义的队列个数 */
    queues = qemu_find_net_clients_except(name, ncs,
                                          NET_CLIENT_DRIVER_NIC,
                                          MAX_QUEUE_NUM);
    assert(queues < MAX_QUEUE_NUM);

    s = DO_UPCAST(VhostUserState, nc, ncs[0]);
    chr = qemu_chr_fe_get_driver(&s->chr);
    trace_vhost_user_event(chr->label, event);
    switch (event) {
    case CHR_EVENT_OPENED: /* 链接建立的逻辑 */
        if (vhost_user_start(queues, ncs, &s->chr) < 0) {
            qemu_chr_fe_disconnect(&s->chr);
            return;
        }
        s->watch = qemu_chr_fe_add_watch(&s->chr, G_IO_HUP,
                                         net_vhost_user_watch, s);
        qmp_set_link(name, true, &err);
        s->started = true;
        break;
    case CHR_EVENT_CLOSED: /* 链接断开的逻辑 */
        /* a close event may happen during a read/write, but vhost
         * code assumes the vhost_dev remains setup, so delay the
         * stop & clear to idle.
         * FIXME: better handle failure in vhost code, remove bh
         */
        if (s->watch) {
            AioContext *ctx = qemu_get_current_aio_context();

            g_source_remove(s->watch);
            s->watch = 0;
            qemu_chr_fe_set_handlers(&s->chr, NULL, NULL, NULL, NULL,
                                     NULL, NULL, false);

            aio_bh_schedule_oneshot(ctx, chr_closed_bh, opaque);
        }
```

linux系统编程（28）
Linuc/unix（36）
C/C++（18）
未分配的博文（2）

文章存档
2020年（3）
2019年（18）
2018年（19）
2017年（9）
2016年（26）
2015年（18）
2014年（54）
2013年（20）

我的朋友
xiaobing qwunkee 岸边的莎
Klose lr011415 dd8924
可怜的猪 vv1133 Garfield

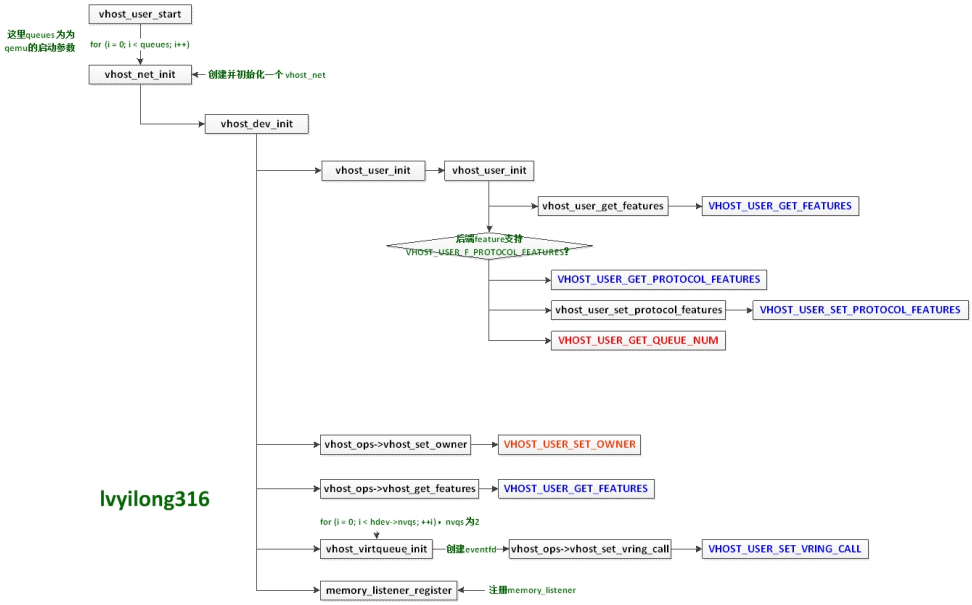
最近访客
forestmo QingSha 小小城御
丽斑麻蜥 格伯纳 nxy_123
dafuzi liucf399 sharsafe

推荐博文
·gdb相关调试命令
·来看看基于Kubernetes的Spark...
·JavaScript正则表达式
·switch_root 命令
·在VMware下实现主机与虚拟主...

相关博文
·手把手教你怎么激活windows10...
·CentOS中zip压缩和unzip解压...
·帮您解决win7系统执行bat批处...
·win8系统固态硬盘优化设置的...
·win7注册表拒绝访问和修改的...
·xp升级到win7系统怎么保留原...
·win8解除网速限制的操作方法...
·Linux应用领域多，具有包含哪...
·Ubuntu18.04更换国内源（阿里...
·win10安装Linux子系统 + ROS(...

```
46.             break;
47.         }
48.
49.         if (err) {
50.             error_report_err(err);
51.         }
52.     }
```

其中分别有链接建立的逻辑，和链接断开的逻辑。我们主要关注链接建立的逻辑，也就是vhost_user_start。下面是vhost_user_start涉及和vhost_user交互的注意流程，具体代码不再展开。



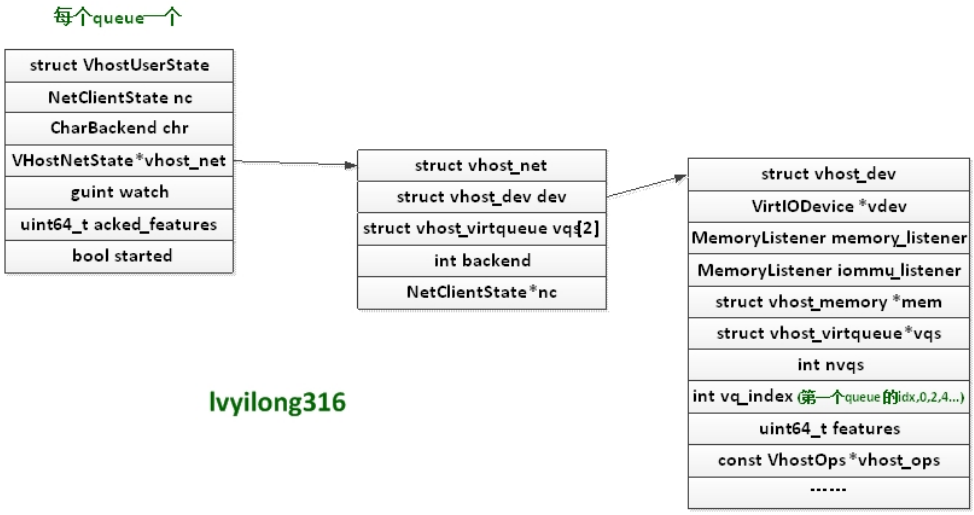
其中需要说明的一点是qemu向vhost_user发送消息是通过vhost_user_write函数进行的，而其中又会调用vhost_user_one_time_request对所发消息进行判断，如果当前消息属于只发一次的消息，且之前已经发送过了，就不在发送了。那么那些消息是只需要发送一次的呢？我们看下vhost_user_one_time_request的实现就清楚了。

vhost_user_one_time_request

点击(此处)折叠或打开

```
1.  static bool vhost_user_one_time_request(VhostUserRequest request)
2.  {
3.      switch (request) {
4.      case VHOST_USER_SET_OWNER:
5.      case VHOST_USER_RESET_OWNER:
6.      case VHOST_USER_SET_MEM_TABLE:
7.      case VHOST_USER_GET_QUEUE_NUM:
8.      case VHOST_USER_NET_SET_MTU:
9.          return true;
10.     default:
11.         return false;
12.     }
13. }
```

这些消息反应在图中使用红色表示。另外需要注意的就是图中的两个循环，一个是vhost_net_init的调用。这个函数会初始化一个vhost_net结构，每个queue都会调用一次（每个queue都对应一个vhost_net结构），例如qemu启动参数定义设备有20个queue，则这个函数就会调用20次。下面是vhost_net相关数据结构关系。

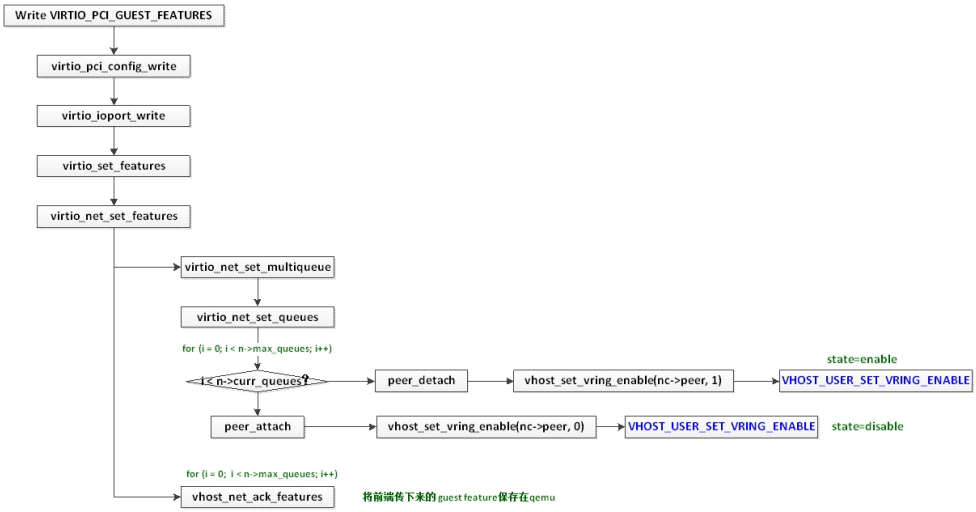


另一个循环是vhost_virtqueue_init的调用，这个调用是当前queue (对应结构体vhost_net)的每个virtqueue(也就是ring)调用一次，其中nvqs是固定的2（每个queue有两个ring）。

所以这一步的协商过程就是有一个个以VHOST_USER_GET_FEATURES开始的循环构成，其中VHOST_USER_SET_VRING_CALL又会被内部循环调用两次。

guest驱动加载

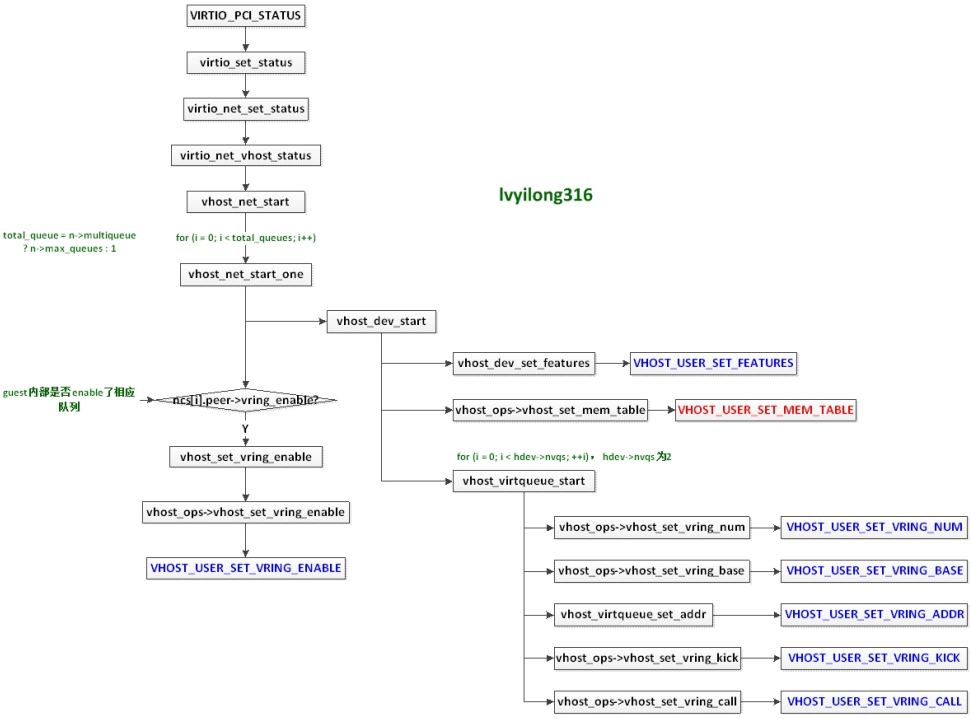
在guest启动后，加载virtio-net驱动，会写寄存器VIRTIO_PCI_GUEST_FEATURES，这个写操作会被kvm捕获传递给qemu。qemu会做如下处理。



其中有两个变量比较关键，一个是max_queues，这个就是qemu启动时后端指定的队列个数，另一个是curr_queues，这个是当前前端enable的queue。例如启动时指定20个queue，但一般guest启动默认只会enable一个queue，所以max_queues为20，curr_queues为1。另外注意vhost_set_vring_enable，最终调用的是vhost_user_set_vring_enable，这个函数会为当前queue的每个ring发送一次VHOST_USER_SET_VRING_ENABLE消息，具体在下个阶段分析。所以20个队列会为每个queue都发送两个VHOST_USER_SET_VRING_ENABLE消息，共40个，但只有小于curr_queues时，也就是只有enable的queue才会发送state为1的消息（共两个），否则state为0。以20个queue为例，会发生20个VHOST_USER_SET_VRING_ENABLE消息，但只有第一个state为enable。

guest驱动加载完成

当guest中virtio-net加载完成后会写VIRTIO_PCI_STATUS寄存器，这个操作同样会被kvm捕获传递给qemu。qemu的相应处理逻辑如下。



其中比较关键的又是两个循环，一个是vhost_net_start_one的调用。这里的循环控制变量total_queues当guest驱动支持多队列时即为qemu启动的指定的后端队列个数，当guest不支持多队列特性的时候即为1。另一处循环就是vhost_virtqueue_start的调用，这个循环和第一阶段的内部循环类似，nvqs为2，即每个queue拥有的ring的个数。hdev即为vhost_dev结构。

最后要注意的就是只有guest中enable的queue才会调用vhost_ops->vhost_set_vring_enable，也就是对于开机默认只enable 1个对列的情况只会调用一次。而vhost_ops->vhost_set_vring_enable实际上就是vhost_user_set_vring_enable，我们看下其实现。

vhost_user_set_vring_enable

点击(此处)折叠或打开

```
static int vhost_user_set_vring_enable(struct vhost_dev *dev, int enable)
{
    int i;

    if (!virtio_has_feature(dev->features, VHOST_USER_F_PROTOCOL_FEATURES)) {
        return -1;
    }

    for (i = 0; i < dev->nvqs; ++i) {
        struct vhost_vring_state state = {
            .index = dev->vq_index + i,
            .num = enable,
        };

        vhost_set_vring(dev, VHOST_USER_SET_VRING_ENABLE, &state);
    }

    return 0;
}
```

可以看到vhost_user_set_vring_enable内部是多当前queue的每个ring调用一次VHOST_USER_SET_VRING_ENABLE，所以对于一个队列enable的情况这里会发送两个VHOST_USER_SET_VRING_ENABLE。

到此为止，guest启动前后端的协商过程就完成了。**如果是后端dpdk重启，vhost_user重连过程和以上启动过程类似，区别是没有第二个阶段，因为这个时候guest内部驱动已经加载完成**。

上一篇：基于dpdk的用户态协议栈f-stack实现分析
下一篇：Intel RDT特性详解

给主人留下些什么吧！~~