LInux下桥接模式详解二

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

上篇文章导入博客园的比较早，而这篇自己在写的时候才发现内部复杂的很，以至于没能按时完成，造成两篇文章的间隔时间有点长！

话不多说，言归正传！

前面的文章介绍了桥接模式下的基础理论知识，其实本节想结合LInux源代码分析下桥接模式下的数据包的转发流程，但是看了源码才发现，这部分内容太多，非一篇文章可以描述的清楚的，所以决定本篇文章主要介绍下linux网络相关的主要结构体，以及各个结构之间的关当一个网络数据包来到host的物理网卡，由于此时网卡已经是混杂模式，所以此数据包的目的地不一定就是host本身。那么此时网卡的　设备控制器就会向host的APIC发送中断信号。CPU收到中断信号后，会自动进入处理该中断的流程，调用IDT中网卡驱动注册的中断处理函数进行处理。

而最终数据包会__netif_receive_skb_core函数进行处理，在进入此函数之前，我们有必要了解下相关的数据结构。

struct net_device 网络设备结构，这里只列举了和我们要分析的相关的信息

```
struct net_device{
    ……
    unsigned long    state;

    ……
    unsigned int    flags;    /* interface flags (a la BSD)    */
    unsigned int    priv_flags; /* Like 'flags' but invisible to userspace.

    ……

    #if IS_ENABLED(CONFIG_VLAN_8021Q)
    struct vlan_info __rcu    *vlan_info;    /* VLAN info */
    #endif

    ……
    unsigned char *dev_addr
    rx_handler_func_t __rcu *rx_handler;
    void __rcu    *rx_handler_data;

    ……

}
```

net_device结构代表一个网络设备，每一个物理网卡还有linux内部都有一个独立的net_device结构与之对应。

state表示设备的状态

flag表示设备的特性，而priv_flag则表示设备的私有特性，对用户空间是不可见的。

dev_addr表示设备的mac地址

rx_handler代表一个钩子函数，在网卡混杂模式开启时，此函数会被初始化成一个转发数据包的函数br_handle_frame

rx_hander_data 指向一个net_bridge_port结构，该端口是正是skb->dev在开启网桥特性后的代表的端口结构。说起来由点绕，还是看下代码

```
static inline struct net_bridge_port *br_port_get_rcu(const struct net_device *dev)
{
    struct net_bridge_port *port =
            rcu_dereference_rtnl(dev->rx_handler_data);

```

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(15806)
3. 进程的挂起、阻塞和睡眠(13713)
4. Linux下桥接模式详解一(13465)
5. virtio后端驱动详解(10538)

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

```
6     return br_port_exists(dev) ? port : NULL;
7 }
```

该函数从一个net_device获取桥接端口，可以看到正是通过其rx_handler_data指针。而dev是否存在这个端口即其结构里面是否设置了此指针就要看dev的私有特性了

```
1 #define br_port_exists(dev) (dev->priv_flags & IFF_BRIDGE_PORT)
```

struct sk_buff 应用层的数据包结构

```
1 struct sk_buffer{
2     struct sk_buff *next;
3     struct sk_buff    *prev;
4
5     ……
6
7     struct net_device *dev;
8
9     ……
1115     __u16    transport_header;//传输层头部偏移
16     __u16    network_header; //IP头部偏移
17     __u16    mac_header;//MAC地址偏移
18     /* These elements must be at the end, see alloc_skb() for details. */
19     sk_buff_data_t    tail;
20     sk_buff_data_t    end;
21     unsigned char    *head,//buffer header指针
22                      *data; //数据指针
23     unsigned int    truesize;
24     atomic_t    users;
25
26 }
```

该结构是数据包逐层交付所必须的结构，其中Next和prev分别指向下一个和上一个buffer，dev表示这个buffer从哪个设备进入，data指向buffer的数据区，head指向buffer的最开始的头部，

mac_header是以太网头部到head指针的偏移，network_header是Ip数据包头部到head指针的偏移，transport_header是传送层头部到head指针的偏移，tail指向数据部分的结束，end指向buffer的结束。truesize是buffer实际的大小，user记录用户数，主要表明是否共享。

struct net_bridge 网桥结构

```
1 struct net_bridge{
2 struct list_head port_list;//所有端口组成的链表头
3 struct net_device    *dev; //对应的物理设备
4
5 ……
6
7 struct net_bridge_mdb_htable __rcu *mdb;
8 ……
9 }
```

这是linux内部的网桥对应的结构，port_list连接网桥所有的端口，dev指向网桥的设备结构体，mdb指向网桥的组播数据库转发表

struct net_bridge_port 网桥端口结构

```
1 struct net_bridge_port
2 {
3     struct net_bridge    *br; //对应的网桥
4     struct net_device    *dev; //端口对应的设备
5     struct list_head    list;
6     ……
7     u8 state;
8     ……
```

```
    unsigned long flags;
    ……
    struct hlist_head mglist;
    ……
}
```

net_bridge_port结构对应于网桥的一个端口，state表明端口的状态，flags表明端口本身的特性，dev指向它关联的设备，br指向它attach的网桥，mglist连接所有port加入的组，flag记录了端口的某些特性，state表明了端口的某一个状态，如转发、学习等。

struct net_bridge_fdb_entry 网桥内部转发表表项

```
struct net_bridge_fdb_entry
{
    struct hlist_node        hlist;
    struct net_bridge_port        *dst;

    struct rcu_head            rcu;
    unsigned long            updated;
    unsigned long            used;
    mac_addr            addr;
    unsigned char            is_local;
    unsigned char            is_static;
    __u16                vlan_id;
};
```

这是网桥内部转发表的表项，hlist表明表项作为一个节点存在于某张表中，这张表就是转发表。dst指向目的端口，addr是表项的mac地址，isLocal表明是否是本地端口，本地端口我猜想是网桥的数据流入端口，即当目的mac是本地端口表明这是发往本地的数据包；isstatic表明是否是静态地址，静态地址不能自动更新。

struct net_bridge_mdb_htable /*组播组数据库转发表，该结构体将所有的组播组数据库转发项通过hash数组连接到一起*/

```
struct net_bridge_mdb_htable
{
    struct hlist_head    *mhash;
    struct rcu_head    rcu;
    struct net_bridge_mdb_htable    *old;
    u32    size;
    u32    max;
    u32    secret;
    u32    ver;
};
```

该结构表示一个组播数据库转发表，连接了所有的组播数据库转发项.size表示表的大小，max表示最大容量。

struct net_port_vlans

```
struct net_port_vlans {
    u16                port_idx;
    u16                pvid;
    union {
        struct net_bridge_port        *port;
        struct net_bridge        *br;
    }                parent;
    struct rcu_head            rcu;
    unsigned long            vlan_bitmap[BR_VLAN_BITMAP_LEN];
    unsigned long            untagged_bitmap[BR_VLAN_BITMAP_LEN];
    u16                num_vlans;
};
```

struct net_bridge_mdb_entry

```
struct net_bridge_mdb_entry

{
        struct hlist_node            hlist[2];
        struct hlist_node            mglist;
        struct net_bridge           *br;//桥
        struct net_bridge_port_group *ports;//
        struct rcu_head                      rcu;
        struct timer_list             timer;//组播组数据库项失效定时器，若超时，则会将该组播端口从
组播组数据库项的组播端口列表中删除
        struct timer_list             query_timer;//查询定时
        __be32                        addr;//组播组地址
        u32                         queries_sent;

};
```

struct net_bridge_port_group

```
1 struct net_bridge_port_group {
2     struct net_bridge_port        *port;
3     struct net_bridge_port_group __rcu *next;
4     struct hlist_node        mglist;
5     struct rcu_head            rcu;
6     struct timer_list        timer;
7     struct br_ip            addr;
8     unsigned char            state;
9 };
```

这是用于组播的组结构，一个组绑定一个组播地址addr，next指向下一个组播组，port指向组的端口,timer是定时器，mgList用于连接一个端口加入的所有的group,表头保存在port结构里

struct mac_addr MAC地址结构

```
1 struct mac_addr
2 {
3     unsigned char    addr[6];
4 };
```

可以看到内核中MAC地址用6个字节表示

牵扯到的几个结构基本都在这里了，里面好多变量我也不是很清楚，有说错的地方还请老师们多多指正！！下一篇就结合源代码分析具体的数据包处理流程了

Linux 下桥接模式3

分类: linux 内核源码分析, KVM虚拟化技术

jack.chen
关注 - 12
粉丝 - 44
+加关注

0 0

« 上一篇： 通过virt-manager 利用NFS创建、迁移虚拟机2
» 下一篇： LInux下桥接模式详解三

posted @ 2016-09-21 20:54 jack.chen Views(5423) Comments(0) Edit 收藏

最新 IT 新闻:
· 寒武纪回应问询函：3年内仍需逾30亿投入芯片研发
· 微软63.2GB源码泄露：涉及Azure、Office和Windows
· AI级体验！小牛电动发布爆款旗舰MQi2：续航超50km 4599元起
· 首批商用5G国家网速一览：比Wi-Fi快几条街
· 华为P40首发 一图看懂EMUI 10.1所有亮点
» 更多新闻...

以马内利