

太初有道，道与神同在，道就是神.....

CnBlogs Home New Post Contact Admin Rss  Posts - 92 Articles - 4 Comments - 45

Linux下桥接模式详解三

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 sele
c...
--Linux-inside
2. Re:virtIO之VHOST工作原理简析

上篇文章介绍了Linux内核桥接模式涉及到的几个结构, 本节就重点放在数据包的处理上!

本节所有代码参考Linux 3.10.1内核!

前面已经提到一个数据包从网卡流到Linux内核中的L2层, 最终被交付到__netif_receive_skb_core函数中, 看下该函数中引用rx_handler的片段

```
1 rx_handler = rcu_dereference(skb->dev->rx_handler);
2 if (rx_handler) {
3     if (pt_prev) {
4         ret = deliver_skb(skb, pt_prev, orig_dev);
5         pt_prev = NULL;
6     }
7     switch (rx_handler(&skb)) {
8     case RX_HANDLER_CONSUMED:
9         ret = NET_RX_SUCCESS;
10        goto unlock;
11    case RX_HANDLER_ANOTHER:
12        goto another_round;
13    case RX_HANDLER_EXACT:
14        deliver_exact = true;
15    }
16    case RX_HANDLER_PASS:
17        break;
18    default:
19        BUG();
20 }
```

可以看到这里首先从设备结构net_device中获取其rx_handler指针, 该指针在网卡的混杂模式下指向一个处理函数叫做br_handle_frame, 即网桥的处理流程

```
1 rx_handler_result_t br_handle_frame(struct sk_buff **pskb)
2 {
3     struct net_bridge_port *p;
4     struct sk_buff *skb = *pskb;
5     const unsigned char *dest = eth_hdr(skb)->h_dest; //获取skb的目的MAC
6     br_should_route_hook_t *rhook;
7
8     if (unlikely(skb->pkt_type == PACKET_LOOPBACK))
9         return RX_HANDLER_PASS;
10
11    if (!is_valid_ether_addr(eth_hdr(skb)->h_source))
12        goto drop;
13
14    skb = skb_share_check(skb, GFP_ATOMIC);
15    /*只有skb是共享的且clone的时候分配内存出错skb才会是null*/
16    if (!skb)
17        return RX_HANDLER_CONSUMED;
18
19    p = br_port_get_rcu(skb->dev);
20
21    if (unlikely(is_link_local_ether_addr(dest))) {
22        /*
23         * See IEEE 802.1D Table 7-10 Reserved addresses
24         *
25         * Assignment Value
26         * Bridge Group Address 01-80-C2-00-00-00
27         * (MAC Control) 802.3 01-80-C2-00-00-01
28         * (Link Aggregation) 802.3 01-80-C2-00-00-02
29         * 802.1X PAE address 01-80-C2-00-00-03
30         */
31    }
```

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)

```
31      * 802.1AB LLDP          01-80-C2-00-00-0E
32      *
33      * Others reserved for future standardization
34      */
35      /*目的MAC 地址 ， 判断是否是特殊的目的MAC地址*/
36      switch (dest[5]) {
37      case 0x00: /* Bridge Group Address */
38          /* If STP is turned off,
39             then must forward to keep loop detection */
40          if (p->br->stp_enabled == BR_NO_STP)
41              goto forward;
42          break;
43
44      case 0x01: /* IEEE MAC (Pause) */
45          goto drop;
46
47      default:
48          /* Allow selective forwarding for most other protocols */
49          if (p->br->group_fwd_mask & (1u << dest[5]))
50              goto forward;
51      }
52
53      /* Deliver packet to local host only */
54      if (NF_HOOK(NFPROTO_BRIDGE, NF_BR_LOCAL_IN, skb, skb->dev,
55                 NULL, br_handle_local_finish)) {
56          return RX_HANDLER_CONSUMED; /* consumed by filter */
57      } else {
58          *pskb = skb;
59          return RX_HANDLER_PASS; /* continue processing */
60      }
61  }
62  //开始转发
63  forward:
64      switch (p->state) {
65      case BR_STATE_FORWARDING:
66          rhook = rcu_dereference(br_should_route_hook);
67          if (rhook) {
68              if ((*rhook)(skb)) {
69                  *pskb = skb;
70                  return RX_HANDLER_PASS;
71              }
72              dest = eth_hdr(skb)->h_dest;
73          }
74          /* fall through */
75      case BR_STATE_LEARNING:
76          if (ether_addr_equal(p->br->dev->dev_addr, dest))//如果数据包进入的端口的MAC和数据包的目的MAC相同
77              skb->pkt_type = PACKET_HOST;//表明这是host的数据，需要直接上缴给协议栈
78
79          NF_HOOK(NFPROTO_BRIDGE, NF_BR_PRE_ROUTING, skb, skb->dev, NULL,
80                 br_handle_frame_finish);
81          break;
82      default:
83      drop:
84          kfree_skb(skb);
85      }
86      return RX_HANDLER_CONSUMED;
87  }
```



这里函数的含义还比较明确，我们先看下所有数据包的类型定义



```
1 #define PACKET_HOST      0      /* To us */
2 #define PACKET_BROADCAST 1      /* To all */
3 #define PACKET_MULTICAST 2      /* To group */
4 #define PACKET_OTHERHOST 3      /* To someone else */
5 #define PACKET_OUTGOING  4      /* Outgoing of any type */
6 /* These ones are invisible by user level */
7 #define PACKET_LOOPBACK  5      /* MC/BRD frame looped back */
8 #define PACKET_FASTROUTE 6      /* Fastrouted frame */
```



数据包的这个特性记录在skb->pkt_type字段中，只是占用三个bit位。

继续看函数体

在函数中，前半部分都是一些验证，这里首先验证数据包的类型，然后验证数据包中源mac地址的合法性，

接着检查skb是否是共享的，这一些都通过后会判断目的MAC地址是否是特殊的MAC地址，虽然这一可能性不大，但是还是要判断下。这里判断的内容不是本文重点，就不在详细描述。

然后就到了forward节：

这里根据端口的state做switch

在BR_STATE_FORWARDING状态下，调用了一个hook函数。这部分内容还不是很理解。

而在BR_STATE_LEARNING状态下，首先判断了目的MAC是否和数据流入端口的mac地址是否相同，相同就表明数据包是发往本机的，设置skb的包类型为PACKET_HOST，然后调用了

br_handle_frame_finish函数

```
1 int br_handle_frame_finish(struct sk_buff *skb)
2 {
3     const unsigned char *dest = eth_hdr(skb)->h_dest;
4     struct net_bridge_port *p = br_port_get_rcu(skb->dev);
5     struct net_bridge *br;
6     struct net_bridge_fdb_entry *dst;
7     struct net_bridge_mdb_entry *mdst;
8     struct sk_buff *skb2;
9     bool unicast = true;
10    u16 vid = 0;
11    //如果端口不可用，则直接丢弃数据包
12    if (!p || p->state == BR_STATE_DISABLED)
13        goto drop;
14    //对vlan标签做相关判断，查看skb是否符合
15    if (!br_allowed_ingress(p->br, nbp_get_vlan_info(p), skb, &vid))
16        goto drop;
17
18    /* insert into forwarding database after filtering to avoid spoofing */
19    br = p->br; //获取网桥结构
20    if (p->flags & BR_LEARNING) //如果网桥具备学习能力，更新转发表
21        br_fdb_update(br, p, eth_hdr(skb)->h_source, vid);
22    //如果不是广播地址&&是多播地址&&多播发送成功
23    if (!is_broadcast_ether_addr(dest) && is_multicast_ether_addr(dest) &&
24        br_multicast_rcv(br, p, skb))
25        goto drop;
26    //此时已经更新表完毕，端口若还是处于学习状态就drop
27    if (p->state == BR_STATE_LEARNING)
28        goto drop;
29
30    BR_INPUT_SKB_CB(skb)->brdev = br->dev;
31
32    /* The packet skb2 goes to the local host (NULL to skip). */
33    skb2 = NULL;
34    //判断网卡若是处于混杂模式
35    if (br->dev->flags & IFF_PROMISC)
36        skb2 = skb;
37
38    dst = NULL;
39    //如果是广播地址
40    if (is_broadcast_ether_addr(dest)) {
41        skb2 = skb;
42        unicast = false; //设置单播标识为false
43    } else if (is_multicast_ether_addr(dest)) {
44        mdst = br_mdb_get(br, skb, vid);
45        if ((mdst || BR_INPUT_SKB_CB_MROUTERS_ONLY(skb)) &&
46            br_multicast_querier_exists(br, eth_hdr(skb))) {
47            if ((mdst && mdst->mglst) ||
48                br_multicast_is_router(br))
49                skb2 = skb;
50            br_multicast_forward(mdst, skb, skb2);
51            skb = NULL;
52            if (!skb2)
53                goto out;
54        } else
55            skb2 = skb;
56        unicast = false;
57        br->dev->stats.multicast++;
58    } //查找转发表并判断表项，如果表项存在且端口是本地端口
```

```

59     } else if ((dst = __br_fdb_get(br, dest, vid)) &&
60         dst->is_local) {
61         skb2 = skb;
62         /* Do not forward the packet since it's local. */
63         skb = NULL;
64     }
65     //fdb表中存在表项且是本地端口或者多播处理完成  skb2=skb  skb=NULL unicast=true
66     //广播或者多播未处理完成  skb2=skb  skb!=NULL unicast=false
67     //fdb表中未找到表项或者不是本地端口  skb!=NULL  skb2=NULL unicast=true
68     if (skb) {
69         if (dst) {
70             //转发表中表项存在且不是本地端口，即需要转发到其他端口
71             dst->used = jiffies;
72             //实施转发
73             br_forward(dst->dst, skb, skb2);
74         } else
75             //处理广播或者多播或者未找到端口的单播
76             br_flood_forward(br, skb, skb2, unicast);
77     }
78     //目的端口是本地端口&&多播&&广播
79     if (skb2)
80         return br_pass_frame_up(skb2);
81
82 out:
83     return 0;
84 drop:
85     kfree_skb(skb);
86     goto out;
87 }

```

这里就要做比较详细的判断了，首先判断端口的状态，然后调用**br_allowed_ingress**函数验证vlan标签，这里就不深入去查看了。接着就调用**br_fdb_update**更新网桥的转发表，对组播数据包进行预处理。

接着就开始了数据包转发前的地址判断，先判断是否是广播地址，是就令skb2=skb即复制一份数据包,并设置unicast为false。

然后判断是否是组播地址，是就从组播数据库中获取对应的**net_bridge_mdb_entry**结构，该结构中记录了组播组中的端口，在经过几个验证之后就调用**br_multicast_forward**进行组播数据包的转发，之后置空skb

最后就剩下单播地址了，从地址转发表中获取**net_bridge_fdb_entry**结构并判断其is_local属性，如果is_lcoal为true则表示这个发往host的数据包，就设置复制一份skb，然后置空skb指针。

然后就开始其他端口的转发，这里在前一部分已经根据不同的情况设置了skb指针，所以如果skb指针不为空就表示这是单播或者广播或者多播未处理的情况，然后判断前面获取的单播转发表的表项是否为空，如果不为空就表示这个发往其他端口的单播数据包，那么就调用br_forward进行转发。如果为空就表示这有可能是其他的情况，那么就调用**br_flood_forward**进行处理，注意这里还有一个参数就是unicast，这是单播标识，函数中会用到。

br_flood_forward会把单播数据包发往所有支持BR_FLOOD特性的端口，上面也许注意到了不只是单播数据包可以走到这里，广播也可以走到这里，这也难怪，单播在未找到表项的情况下只能向所有其他的支持BR_FLOOD特性的端口转发，这和广播很相似，只不过是广播的话BR_FLOOD特性也不起作用，直接全部转发了。**只是我不太明白的是未处理的组播数据包怎么办了??**

接着就处理本地数据包的情况，即数据包目的地址是host的单播数据、广播、组播都需要给host上层交付，那么这里就调用**br_pass_frame_up**函数

```

1 static int br_pass_frame_up(struct sk_buff *skb)
2 {
3     struct net_device *indev, *brdev = BR_INPUT_SKB_CB(skb)->brdev;
4     struct net_bridge *br = netdev_priv(brdev);
5     struct br_cpu_netstats *brstats = this_cpu_ptr(br->stats);
6
7     u64_stats_update_begin(&brstats->syncp);
8     brstats->rx_packets++;
9     brstats->rx_bytes += skb->len;
10    u64_stats_update_end(&brstats->syncp);

```

```

11
12  /* Bridge is just like any other port.  Make sure the
13   * packet is allowed except in promisc modue when someone
14   * may be running packet capture.
15   */
16  if (!(brdev->flags & IFF_PROMISC) &&
17      !br_allowed_egress(br, br_get_vlan_info(br), skb)) {
18      kfree_skb(skb);
19      return NET_RX_DROP;
20  }
21
22  skb = br_handle_vlan(br, br_get_vlan_info(br), skb);
23  if (!skb)
24      return NET_RX_DROP;
25
26  indev = skb->dev;
27  skb->dev = brdev;
28
29  return NF_HOOK(NFPROTO_BRIDGE, NF_BR_LOCAL_IN, skb, indev, NULL,
30                netif_receive_skb);
31 }

```



到了该函数已经要准备把数据交付给网络层了，并且已经设置数据包的设备skb->dev修改为网桥代表的设备，表明这是从网桥发出的数据包。最后会再次调用netif_receive_skb重新接受数据包但是这时skb->dev是网桥，并且网桥设备的rx_handler指针肯定为空，那么就不会再次进入网桥的处理，而是直接交付上层了。

br_flood_forward

分类: [linux 内核源码分析](#), [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下桥接模式详解二](#)

» 下一篇: [谈windows中的句柄](#)

posted @ 2016-09-22 16:19 jack.chen Views(6075) Comments(0) Edit 收藏

[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](#)。

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...](#)

