qemu网络虚拟化之数据流向分析二

邮箱：zhunxun@gmail.com

2016-09-27

上篇文章大致介绍了qemu网络虚拟化相关的数据结构，本篇就结合qemu-kvm源代码分析下各个数据结构是如何初始化以及建立联系的。

这里还是分为三个部分：

1、Tap设备区

2、Hub区

3、NIC区

1、Tap设备区

在net.c中有数组记录下net client 初始化的相关函数

```
1  static int (* const net_client_init_fun[NET_CLIENT_OPTIONS_KIND_MAX])(
2      const NetClientOptions *opts,
3      const char *name,
4      NetClientState *peer) = {
5          [NET_CLIENT_OPTIONS_KIND_NIC]       = net_init_nic,
6  #ifdef CONFIG_SLIRP
7          [NET_CLIENT_OPTIONS_KIND_USER]      = net_init_slirp,
8  #endif
9          [NET_CLIENT_OPTIONS_KIND_TAP]       = net_init_tap,
10         [NET_CLIENT_OPTIONS_KIND_SOCKET]    = net_init_socket,
11 #ifdef CONFIG_VDE
12         [NET_CLIENT_OPTIONS_KIND_VDE]       = net_init_vde,
13 #endif
14         [NET_CLIENT_OPTIONS_KIND_DUMP]      = net_init_dump,
15 #ifdef CONFIG_NET_BRIDGE
16         [NET_CLIENT_OPTIONS_KIND_BRIDGE]    = net_init_bridge,
17 #endif
18         [NET_CLIENT_OPTIONS_KIND_HUBPORT]   = net_init_hubport,
19 };
```

这里我们就从net_init_tap开始，位于tap.c中这里暂且忽略下其他无关的代码

在该函数中关键是调用了net_init_tap_one，因为qemu中的vlan不支持多TAP。

```
1  static int net_init_tap_one(const NetdevTapOptions *tap, NetClientState *peer,
2                              const char *model, const char *name,
3                              const char *ifname, const char *script,
4                              const char *downscript, const char *vhostfdname,
5                              int vnet_hdr, int fd)
6  {
7      TAPState *s;
8
9      s = net_tap_fd_init(peer, model, name, fd, vnet_hdr);
10     if (!s) {
11         close(fd);
12         return -1;
13     }
14
15     if (tap_set_sndbuf(s->fd, tap) < 0) {
16         return -1;
17     }
18
19     if (tap->has_fd || tap->has_fds) {
```


< 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

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)

```
20          snprintf(s->nc.info_str, sizeof(s->nc.info_str), "fd=%d", fd);
21      } else if (tap->has_helper) {
22          snprintf(s->nc.info_str, sizeof(s->nc.info_str), "helper=%s",
23                   tap->helper);
24      } else {
25          snprintf(s->nc.info_str, sizeof(s->nc.info_str),
26                   "ifname=%s,script=%s,downscript=%s", ifname, script,
27                   downscript);
28
29          if (strcmp(downscript, "no") != 0) {
30              snprintf(s->down_script, sizeof(s->down_script), "%s", downscript);
31              snprintf(s->down_script_arg, sizeof(s->down_script_arg),
32                       "%s", ifname);
33          }
34      }
35
36      if (tap->has_vhost ? tap->vhost :
37          vhostfdname || (tap->has_vhostforce && tap->vhostforce)) {
38          int vhostfd;
39
40          if (tap->has_vhostfd || tap->has_vhostfds) {
41              vhostfd = monitor_handle_fd_param(cur_mon, vhostfdname);
42              if (vhostfd == -1) {
43                  return -1;
44              }
45          } else {
46              vhostfd = -1;
47          }
48
49          s->vhost_net = vhost_net_init(&s->nc, vhostfd,
50                                        tap->has_vhostforce && tap->vhostforce);
51          if (!s->vhost_net) {
52              error_report("vhost-net requested but could not be initialized");
53              return -1;
54          }
55      } else if (tap->has_vhostfd || tap->has_vhostfds) {
56          error_report("vhostfd= is not valid without vhost");
57          return -1;
58      }
59
60      return 0;
61  }
```

这里首先就创建了一个NetClientState结构，前文分析过其实作为逻辑连接点，这里称之为net client.

```
1  NetClientState *qemu_new_net_client(NetClientInfo *info,
2                                      NetClientState *peer,
3                                      const char *model,
4                                      const char *name)
5  {
6      NetClientState *nc;
7
8      assert(info->size >= sizeof(NetClientState));
9
10     nc = g_malloc0(info->size);//这里申请的空间是info->size,回想在网卡端申请的是info->size+num*sizeof(NetClientState)
11     //在增加端口的时候peer还是null
12     qemu_net_client_setup(nc, info, peer, model, name,
13                           qemu_net_client_destructor);
14
15     return nc;
16 }
```

该函数中申请空间后就调用了qemu_net_client_setup函数设置net client。还有一点需要注意，这里申请的空间是info->size,可以看下

```
1 static NetClientInfo net_tap_info = {
2     .type = NET_CLIENT_OPTIONS_KIND_TAP,
3     .size = sizeof(TAPState),
4     .receive = tap_receive,
5     .receive_raw = tap_receive_raw,
6     .receive_iov = tap_receive_iov,
```

```
7     .poll = tap_poll,
8     .cleanup = tap_cleanup,
9 };
```

可见这里的大小是TAPState的大小。这也就解释了上面的函数中DO_UPCAST(TAPState, nc, nc);下面看qemu_net_client_setup函数

```
1 static void qemu_net_client_setup(NetClientState *nc,
2                                    NetClientInfo *info,
3                                    NetClientState *peer,
4                                    const char *model,
5                                    const char *name,
6                                    NetClientDestructor *destructor)
7 {
8     nc->info = info;//建立NetClientState到port的连接
9     nc->model = g_strdup(model);
10    if (name) {
11        nc->name = g_strdup(name);
12    } else {
13        nc->name = assign_name(nc, model);
14    }
15
16    if (peer) {//相互指向
17        assert(!peer->peer);
18        nc->peer = peer;
19        peer->peer = nc;
20    }
21    QTAILQ_INSERT_TAIL(&net_clients, nc, next);//加入全局的net_clients链表中
22
23    nc->incoming_queue = qemu_new_net_queue(nc);//设置接收队列
24    nc->destructor = destructor;
25 }
```

该函数设置net client,完成最主要的功能就是TAPState和Hub进行关联。函数体并不难理解，设置了下info,model,name,peer等字段，peer用于指向传递进来的peer指针，通知也设置对端的peer指向。然后把NetClientState结构加入到全局的net_clients链表中(从尾部加入)，之后再设置接收队列incoming_queue和析构函数。

2、HUb端

和前面类似，这里也从net_init_hubport开始，位于hub.c中

```
1 int net_init_hubport(const NetClientOptions *opts, const char *name,
2                       NetClientState *peer)
3 {
4     const NetdevHubPortOptions *hubport;
5
6     assert(opts->kind == NET_CLIENT_OPTIONS_KIND_HUBPORT);
7     hubport = opts->hubport;
8
9     /* Treat hub port like a backend, NIC must be the one to peer */
10    if (peer) {
11        return -EINVAL;
12    }
13
14    net_hub_add_port(hubport->hubid, name);
15    return 0;
16 }
```

函数体很简单，做了简单的验证后就调用net_hub_add_port函数给指定的Hub增加一个port,有两个参数，分别为hubid和name,前面提到过qemu中用Hub来实现vlan，这里实际上也可以理解为vlan id.

看net_hub_add_port函数

```
NetClientState *net_hub_add_port(int hub_id, const char *name)
{
    NetHub *hub;
    NetHubPort *port;

    QLIST_FOREACH(hub, &hubs, next) {
        if (hub->id == hub_id) {
            break;
        }
    }
    if (!hub) {
        hub = net_hub_new(hub_id);
    }
    port = net_hub_port_new(hub, name);
    return &port->nc;
}
```

这里首先要从全局的Hub链表hubs遍历到指定的Hub，如果没有则创建一个新的，然后调用net_hub_port_new函数创建port，返回port->nc

```
static NetHubPort *net_hub_port_new(NetHub *hub, const char *name)
{
    NetClientState *nc;
    NetHubPort *port;
    int id = hub->num_ports++;
    char default_name[128];

    if (!name) {
        snprintf(default_name, sizeof(default_name),
                 "hub%dport%d", hub->id, id);
        name = default_name;
    }

    nc = qemu_new_net_client(&net_hub_port_info, NULL, "hub", name);
    port = DO_UPCAST(NetHubPort, nc, nc);
    port->id = id;
    port->hub = hub;

    QLIST_INSERT_HEAD(&hub->ports, port, next);

    return port;
}
```

这里会通过qemu_new_net_client函数创建一个net client即NetClientState，和上面一样，这里申请的空间是sizeof(NetHubPort),然后转换指针到NetHubPort，做一些其他的设置，如指定port id和所属的hub，然后加入port到Hub下属的port链表（从头插入）。

3、NIC端

这里我们还是先从net_init_nic函数说起。但是该函数 具体的作用我还真没有分析到。结合具体的网卡没有发现调用此函数的，相比之下该函数好像有点独立了！后面有机会在看吧，先结合e1000网卡的初始化过程进行分析。

下面从e1000网卡初始化开始，主要的文件在e1000.c文件中

```
static void e1000_class_init(ObjectClass *klass, void *data)
{
    DeviceClass *dc = DEVICE_CLASS(klass);
    PCIDeviceClass *k = PCI_DEVICE_CLASS(klass);

    k->init = pci_e1000_init;
```

```
    k->exit = pci_e1000_uninit;
    k->romfile = "efi-e1000.rom";
    k->vendor_id = PCI_VENDOR_ID_INTEL;
    k->device_id = E1000_DEVID;
    k->revision = 0x03;
    k->class_id = PCI_CLASS_NETWORK_ETHERNET;
    set_bit(DEVICE_CATEGORY_NETWORK, dc->categories);
    dc->desc = "Intel Gigabit Ethernet";
    dc->reset = qdev_e1000_reset;
    dc->vmsd = &vmstate_e1000;
    dc->props = e1000_properties;
}
```

其中最主要的函数pci_e1000_init函数和e1000_properties，其他的暂且忽略。前者是e1000网卡的初始化函数，后者是从命令行接收到的参数赋值到相关的网卡属性。先看后者吧

```
static Property e1000_properties[] = {
    DEFINE_NIC_PROPERTIES(E1000State, conf),
    DEFINE_PROP_BIT("autonegotiation", E1000State,
                    compat_flags, E1000_FLAG_AUTONEG_BIT, true),
    DEFINE_PROP_BIT("mitigation", E1000State,
                    compat_flags, E1000_FLAG_MIT_BIT, true),
    DEFINE_PROP_END_OF_LIST(),
};
```

宏定义DEFINE_NIC_PROPERTIES把接收到的相关信息赋值到E1000State结构中的conf字段

```
#define DEFINE_NIC_PROPERTIES(_state, _conf)                            \
    DEFINE_PROP_MACADDR("mac",   _state, _conf.macaddr),                \
    DEFINE_PROP_VLAN("vlan",     _state, _conf.peers),                   \
    DEFINE_PROP_NETDEV("netdev", _state, _conf.peers),                   \
    DEFINE_PROP_INT32("bootindex", _state, _conf.bootindex, -1)
```

然后查看pci_e1000_init函数，首先根据pci_dev获取了E1000State结构

然后调用e1000_mmio_setup函数设置网卡的MMIO地址空间，然后调用pci_register_bar函数注册ba空间。

最重要的是设置d->nic,即E1000State结构中的NICState字段，这里是调用了qemu_new_nic函数创建一个net client

```
NICState *qemu_new_nic(NetClientInfo *info,
                       NICConf *conf,
                       const char *model,
                       const char *name,
                       void *opaque)
{
//conf->peers.ncs指向一个NetClientState指针数组，即数组的每一项都指向一个NetClientState结构
    NetClientState **peers = conf->peers.ncs;
    NICState *nic;
    int i, queues = MAX(1, conf->queues);//这里的queues貌似应该是0

    assert(info->type == NET_CLIENT_OPTIONS_KIND_NIC);
    assert(info->size >= sizeof(NICState));

    nic = g_malloc0(info->size + sizeof(NetClientState) * queues);
    nic->ncs = (void *)nic + info->size;
    //nic->ncs也指向一个NetClientState数组，数组项的个数是MAX(1, conf->queues);
    nic->conf = conf;
    nic->opaque = opaque;
    //设置两个数组的NetClientState建立关系
    for (i = 0; i < queues; i++) {
        qemu_net_client_setup(&nic->ncs[i], info, peers[i], model, name,
```

```
23                                  NULL);
24          nic->ncs[i].queue_index = i;
25      }
26
27      return nic;
28  }
```

该函数就需要仔细分析一下了，其中有几点我也不是很明白，后面会表明出来。

函数体中首先获取conf->peers.ncs，结合前篇文章不难看到这里是获取了一个NetClientState地址数组的地址，peers便指向这个数组，这里应该是对端的NetClientState地址数组。

然后给nic分配地址，这里分配的空间是info->size + sizeof(NetClientState) * queues，可以看到这里虽然是给NICState申请空间，但是紧跟着NICState还有一个queues数量的NetClientState空间，这些net client是代表网卡端的net client.

然后就是一个循环，依次调用qemu_net_client_setup函数对net client 做设置，并和前面提到的对端数组中的对应NetClientState做相互的关联。

疑惑：

1、在NICConf的初始化中并为什么没有发现初始化queues字段的？难道是这里queues字段默认是0？

2、这里NICState为什么会关联一个NetClientState地址数组，从架构来看，好像和多队列相关，每个队列对应一个NetClientState结构，但是在Hub初始化端口的时候发现每个端口只有一个NetClientState，这里有点疑惑，难道NICState关联的数组里面其实只有一个表项？

总结：

本篇文章大致结合源代码分析了各个数据结构之间建立关系的过程，总体上展现了一个框架，下篇文章就在上一个层次，从数据包的流向看数据包是如何在这些结构中流动的！！

前面笔者的疑惑，还请晓得的老师多多指点，谢谢！

分类：KVM虚拟化技术, linux 内核源码分析

jack.chen
关注 - 12
粉丝 - 44
+加关注

1 0

« 上一篇： qemu网络虚拟化之数据流向分析一
» 下一篇： sVIrt概述

posted @ 2016-09-27 20:51 jack.chen Views(870) Comments(0) Edit 收藏

相关博文：
· QEMU网络配置
· KVM/QEMU桥接网络设置[zz]
· linux网络流程分析（一）---网卡驱动
· ArcGIS网络分析之发布网络分析服务(二)
· linux内核SPI总线驱动分析（二）
» 更多推荐...

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...

以马内利