

virtio后端方案vhost

原创 majieyue 最后发布于2016-05-09 11:00:10 阅读数 7768 ☆ 收藏

分类专栏: KVM/QEMU

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/majieyue/article/details/51262510>

vhost是virtio的另一种方案，用于跳过qemu，减少qemu和内核之间上下文切换的开销，对于网络IO而言提升尤其明显。vhost分为两种实现方式，本文重点讨论内核态的vhost

vhost内核模块主要处理数据面的事情，控制面上还是交给qemu，vhost的数据结构如下

```
1 struct vhost_dev {
2     MemoryListener memory_listener; /* MemoryListener是物理内存操作的回调函数集合 */
3     struct vhost_memory *mem;
4     int n_mem_sections;
5     MemoryRegionSection *mem_sections;
6     struct vhost_virtqueue *vqs; /* vhost_virtqueue列表和个数 */
7     int nvqs;
8     /* the first virtqueue which would be used by this vhost dev */
9     int vq_index;
10    unsigned long long features; /* vhost设备支持的features */
11    unsigned long long acked_features; /* guest acked的features */
12    unsigned long long backend_features; /* backend, e.g. tap设备, 支持的features */
13    bool started;
14    bool log_enabled;
15    vhost_log_chunk_t *log;
16    unsigned long long log_size;
17    Error *migration_blocker;
18    bool force;
19    bool memory_changed;
20    hwaddr mem_changed_start_addr;
21    hwaddr mem_changed_end_addr;
22    const VhostOps *vhost_ops; /* VhostOps基于kernel和user两种形态的vhost有不同的实现，内核的实现最终调用ioctl完成 */
23    void *opaque;
24 };
25
26 struct vhost_virtqueue {
27     int kick;
28     int call;
29     void *desc;
30     void *avail;
31     void *used;
32     int num;
33     unsigned long long used_phys;
34     unsigned used_size;
35     void *ring;
36     unsigned long long ring_phys;
37     unsigned ring_size;
38     EventNotifier masked_notifier;
39 };
```

vhost的内存布局，也是由一组vhost_memory_region构成，

```
1 struct vhost_memory_region {
2     __u64 guest_phys_addr;
3     __u64 memory_size; /* bytes */
4     __u64 userspace_addr;
5     __u64 flags_padding; /* No flags are currently specified. */
6 };
7
8 /* All region addresses and sizes must be 4K aligned. */
```



2



举报

```

9 | #define VHOST_PAGE_SIZE 0x1000 |
11 | struct vhost_memory {
12 |     __u32 nregions;
13 |     __u32 padding;
14 |     struct vhost_memory_region regions[0];
15 | };

```

vhost的控制面由qemu来控制，通过ioctl操作vhost_xxx的内核模块，e.g.

```

1 | long vhost_dev_ioctl(struct vhost_dev *d, unsigned int ioctl, unsigned long arg)
2 | {
3 |     void __user *argp = (void __user *)arg;
4 |     struct file *eventfp, *filep = NULL;
5 |     struct eventfd_ctx *ctx = NULL;
6 |     u64 p;
7 |     long r;
8 |     int i, fd;
9 |
10 |    /* If you are not the owner, you can become one */
11 |    if (ioctl == VHOST_SET_OWNER) {
12 |        r = vhost_dev_set_owner(d);
13 |        goto done;
14 |    }
15 |
16 |    /* You must be the owner to do anything else */
17 |    r = vhost_dev_check_owner(d);
18 |    if (r)
19 |        goto done;
20 |
21 |    switch (ioctl) {
22 |    case VHOST_SET_MEM_TABLE:
23 |        r = vhost_set_memory(d, argp);
24 |        break;
25 |    ...
26 |    default:
27 |        r = vhost_set_vring(d, ioctl, argp);
28 |        break;
29 |    }
30 | done:
31 |     return r;
32 | }

```

VHOST_SET_OWNER，用于把当前guest对应的qemu进程和vhost内核线程关联起来

```

1 | VHOST_SET_OWNER
2 | /* Caller should have device mutex */
3 | static long vhost_dev_set_owner(struct vhost_dev *dev)
4 | {
5 |     struct task_struct *worker;
6 |     int err;
7 |     /* Is there an owner already? */
8 |     if (dev->mm) {
9 |         err = -EBUSY;
10 |        goto err_mm;
11 |    }
12 |    /* No owner, become one */
13 |    dev->mm = get_task_mm(current); /* 拿到qemu进程的mm_struct, 即guest的内存分布结构 */
14 |    worker = kthread_create(vhost_worker, dev, "vhost-%d", current->pid); /* 创建vhost线程 */
15 |    if (IS_ERR(worker)) {
16 |        err = PTR_ERR(worker);
17 |        goto err_worker;
18 |    }
19 |
20 |    dev->worker = worker;
21 |    wake_up_process(worker); /* avoid contributing to loadavg */
22 |
23 |    err = vhost_attach_cgroups(dev);

```

```

24 | if (err)25 |     goto err_cgroup;
26 |
27 | err = vhost_dev_alloc_iovecs(dev); /* 为vhost_virtqueue分配iovec内存空间 */
28 | if (err)
29 |     goto err_cgroup;
30 |
31 | return 0;
32 err_cgroup:
33     kthread_stop(worker);
34     dev->worker = NULL;
35 err_worker:
36     if (dev->mm)
37         mmput(dev->mm);
38     dev->mm = NULL;
39 err_mm:
40     return err;
41 }

```



2



VHOST_SET_MEM_TABLE, 初始化vhost_dev的vhost_memory内存成员

```

1 | static long vhost_set_memory(struct vhost_dev *d, struct vhost_memory __user *m)
2 | {
3 |     struct vhost_memory mem, *newmem, *oldmem;
4 |     unsigned long size = offsetof(struct vhost_memory, regions);
5 |     if (copy_from_user(&mem, m, size))
6 |         return -EFAULT;
7 |     if (mem.padding)
8 |         return -EOPNOTSUPP;
9 |     if (mem.nregions > VHOST_MEMORY_MAX_NREGIONS)
10 |         return -E2BIG;
11 |     newmem = kmalloc(size + mem.nregions * sizeof *m->regions, GFP_KERNEL); /* 分配多个vhost_memory_region */
12 |     if (!newmem)
13 |         return -ENOMEM;
14 |
15 |     memcpy(newmem, &mem, size);
16 |     if (copy_from_user(newmem->regions, m->regions,
17 |         mem.nregions * sizeof *m->regions)) {
18 |         kfree(newmem);
19 |         return -EFAULT;
20 |     }
21 |
22 |     if (!memory_access_ok(d, newmem, vhost_has_feature(d, VHOST_F_LOG_ALL))) {
23 |         kfree(newmem);
24 |         return -EFAULT;
25 |     }
26 |     oldmem = d->memory;
27 |     rcu_assign_pointer(d->memory, newmem);
28 |     synchronize_rcu();
29 |     kfree(oldmem);
30 |     return 0;
31 | }

```

VHOST_GET_FEATURES, VHOST_SET_FEATURES, 用于读写vhost支持的features, 目前只有vhost_net模块用到,

```

1 | enum {
2 |     VHOST_FEATURES = (1ULL << VIRTIO_F_NOTIFY_ON_EMPTY) |
3 |         (1ULL << VIRTIO_RING_F_INDIRECT_DESC) |
4 |         (1ULL << VIRTIO_RING_F_EVENT_IDX) |
5 |         (1ULL << VHOST_F_LOG_ALL) |
6 |         (1ULL << VHOST_NET_F_VIRTIO_NET_HDR) |
7 |         (1ULL << VIRTIO_NET_F_MRG_RXBUF),
8 | };
9 |
10 | static long vhost_net_ioctl(struct file *f, unsigned int ioctl,
11 |     unsigned long arg)
12 | {
13 |     ....
14 |     case VHOST_GET_FEATURES:
15 |         features = VHOST_FEATURES;

```



举报

```

16 | if (copy_to_user(featurep, &features, sizeof features))17 | return -EFAULT;
18 | return 0;
19 | case VHOST_SET_FEATURES:
20 | if (copy_from_user(&features, featurep, sizeof features))
21 | return -EFAULT;
22 | if (features & ~VHOST_FEATURES)
23 | return -EOPNOTSUPP;
24 | return vhost_net_set_features(n, features);
25 | ....
26 | }

```

VHOST_SET_VRING_CALL, 设置irqfd, 把中断注入guest

VHOST_SET_VRING_KICK, 设置ioeventfd, 获取guest notify

```

1 | case VHOST_SET_VRING_KICK:
2 | if (copy_from_user(&f, argp, sizeof f)) {
3 | r = -EFAULT;
4 | break;
5 | }
6 | eventfp = f.fd == -1 ? NULL : eventfd_fget(f.fd);
7 | if (IS_ERR(eventfp)) {
8 | r = PTR_ERR(eventfp);
9 | break;
10 | }
11 | if (eventfp != vq->kick) { /* eventfp不同于vq->kick, 此时需要stop vq->kick同时start eventfp */
12 | pollstop = filep = vq->kick;
13 | pollstart = vq->kick = eventfp;
14 | } else
15 | filep = eventfp; /* 两者相同, 无需stop & start */
16 | break;
17 | case VHOST_SET_VRING_CALL:
18 | if (copy_from_user(&f, argp, sizeof f)) {
19 | r = -EFAULT;
20 | break;
21 | }
22 | eventfp = f.fd == -1 ? NULL : eventfd_fget(f.fd);
23 | if (IS_ERR(eventfp)) {
24 | r = PTR_ERR(eventfp);
25 | break;
26 | }
27 | if (eventfp != vq->call) { /* eventfp不同于vq->call, 此时需要stop vq->call同时start eventfp */
28 | filep = vq->call;
29 | ctx = vq->call_ctx;
30 | vq->call = eventfp;
31 | vq->call_ctx = eventfp ?
32 | eventfd_ctx_fileget(eventfp) : NULL;
33 | } else
34 | filep = eventfp;
35 | break;
36 | if (pollstop && vq->handle_kick)
37 | vhost_poll_stop(&vq->poll);
38 |
39 | if (ctx)
40 | eventfd_ctx_put(ctx); /* pollstop之后, 释放之前占用的ctx */
41 | if (filep)
42 | fput(filep); /* pollstop之后, 释放之前占用的filep */
43 |
44 | if (pollstart && vq->handle_kick)
45 | vhost_poll_start(&vq->poll, vq->kick);
46 |
47 | mutex_unlock(&vq->mutex);
48 |
49 | if (pollstop && vq->handle_kick)
50 | vhost_poll_flush(&vq->poll);
51 | return r;

```



2



下面来看下vhost的数据流, vhost与kvm模块之间通过eventfd来实现, guest到host方向的kick event, 通过ioeventfd实现, host到guest方向的call event, 通过irqf

host到guest方向

首先host处理used ring，然后判断如果KVM_IRQFD成功设置，kvm模块会通过irqfd把中断注入guest。qemu是通过virtio_pci_

> kvm_virtio_pci_vector_use -> kvm_virtio_pci_irqfd_use -> kvm_irqchip_add_irqfd_notifier -> kvm_irqchip_assign_irqfd

kvm模块的irqfd的，包含write fd和read fd（可选）

```
1 static int kvm_virtio_pci_vector_use(VirtIOPCIProxy *proxy, int nvqs)
2 {
3     PCIDevice *dev = &proxy->pci_dev;
4     VirtIODevice *vdev = virtio_bus_get_device(&proxy->bus);
5     VirtIODeviceClass *k = VIRTIO_DEVICE_GET_CLASS(vdev);
6     unsigned int vector;
7     int ret, queue_no;
8     MSIMessage msg;
9
10    for (queue_no = 0; queue_no < nvqs; queue_no++) {
11        if (!virtio_queue_get_num(vdev, queue_no)) {
12            break;
13        }
14        vector = virtio_queue_vector(vdev, queue_no);
15        if (vector >= msix_nr_vectors_allocated(dev)) {
16            continue;
17        }
18        msg = msix_get_message(dev, vector);
19        ret = kvm_virtio_pci_vq_vector_use(proxy, queue_no, vector, msg);
20        if (ret < 0) {
21            goto undo;
22        }
23        /* If guest supports masking, set up irqfd now.
24         * Otherwise, delay until unmasked in the frontend.
25         */
26        if (k->guest_notifier_mask) {
27            ret = kvm_virtio_pci_irqfd_use(proxy, queue_no, vector);
28            if (ret < 0) {
29                kvm_virtio_pci_vq_vector_release(proxy, vector);
30                goto undo;
31            }
32        }
33    }
34    return 0;
35
36    undo:
37    while (--queue_no >= 0) {
38        vector = virtio_queue_vector(vdev, queue_no);
39        if (vector >= msix_nr_vectors_allocated(dev)) {
40            continue;
41        }
42        if (k->guest_notifier_mask) {
43            kvm_virtio_pci_irqfd_release(proxy, queue_no, vector);
44        }
45        kvm_virtio_pci_vq_vector_release(proxy, vector);
46    }
47    return ret;
48 }
```

如果没有设置irqfd，则guest notifier fd会通知到等待fd的qemu进程，进入注册函数virtio_queue_guest_notifier_read，调用virtio_irq，最终调用到virtio_pci_noti

```
1 static void virtio_queue_guest_notifier_read(EventNotifier *n)
2 {
3     VirtQueue *vq = container_of(n, VirtQueue, guest_notifier);
4     if (event_notifier_test_and_clear(n)) {
5         virtio_irq(vq);
6     }
7 }
8
9 void virtio_irq(VirtQueue *vq)
10 {
11     trace_virtio_irq(vq);
```

```

12 | vq->vdev->isr |= 0x01;
    | 13 | virtio_notify_vector(vq->vdev, vq->vector);
14 | }
15 |
16 | static void virtio_notify_vector(VirtIODevice *vdev, uint16_t vector)
17 | {
18 |     BusState *qbus = qdev_get_parent_bus(DEVICE(vdev));
19 |     VirtioBusClass *k = VIRTIO_BUS_GET_CLASS(qbus);
20 |
21 |     if (k->notify) {
22 |         k->notify(qbus->parent, vector);
23 |     }
24 | }
25 |
26 | static void virtio_pci_notify(DeviceState *d, uint16_t vector)
27 | {
28 |     VirtIOPCIProxy *proxy = to_virtio_pci_proxy_fast(d);
29 |
30 |     if (msix_enabled(&proxy->pci_dev))
31 |         msix_notify(&proxy->pci_dev, vector);
32 |     else {
33 |         VirtIODevice *vdev = virtio_bus_get_device(&proxy->bus);
34 |         pci_set_irq(&proxy->pci_dev, vdev->isr & 1);
35 |     }
36 | }

```



2



整个过程如图所示 (摘自<http://royluo.org/2014/08/22/vhost/>)

guest到host方向

guest通过向pci配置空间写入从而产生VMEXIT，被kvm截获之后触发注册fd的notification

```

1 | kvm_init:
2 |     memory_listener_register(&kvm_memory_listener, &address_space_memory);
3 |     memory_listener_register(&kvm_io_listener, &address_space_io);
4 |
5 | static MemoryListener kvm_memory_listener = {
6 |     .region_add = kvm_region_add,
7 |     .region_del = kvm_region_del,
8 |     .log_start = kvm_log_start,
9 |     .log_stop = kvm_log_stop,
10 |    .log_sync = kvm_log_sync,
11 |    .log_global_start = kvm_log_global_start,
12 |    .log_global_stop = kvm_log_global_stop,
13 |    .eventfd_add = kvm_mem_ioeventfd_add,
14 |    .eventfd_del = kvm_mem_ioeventfd_del,
15 |    .coalesced_mmio_add = kvm_coalesce_mmio_region,
16 |    .coalesced_mmio_del = kvm_uncoalesce_mmio_region,
17 |    .priority = 10,
18 | };
19 |
20 | static MemoryListener kvm_io_listener = {
21 |     .eventfd_add = kvm_io_ioeventfd_add,
22 |     .eventfd_del = kvm_io_ioeventfd_del,
23 |     .priority = 10,
24 | };
25 |
26 | static void kvm_io_ioeventfd_add(MemoryListener *listener,
27 |     MemoryRegionSection *section,
28 |     bool match_data, uint64_t data,
29 |     EventNotifier *e)
30 | {
31 |     int fd = event_notifier_get_fd(e);
32 |     int r;
33 |
34 |     r = kvm_set_ioeventfd_pio(fd, section->offset_within_address_space,
35 |         data, true, int128_get64(section->size),
36 |         match_data);
37 |     if (r < 0) {
38 |         fprintf(stderr, "%s: error adding ioeventfd: %s\n",
39 |             __func__, strerror(-r));

```



举报

```
40 |     abort();
    |         41 | }
42 | }
```

而kvm_io_ioeventfd_add最终调用了kvm_set_ioeventfd_pio，后者调用了kvm_vm_ioctl(kvm_state, KVM_IOEVENTFD, &kick) 进入到了kvm.

```
1 static int kvm_set_ioeventfd_pio(int fd, uint16_t addr, uint16_t val,
2     bool assign, uint32_t size, bool datamatch)
3 {
4     struct kvm_ioeventfd kick = {
5         .datamatch = datamatch ? val : 0,
6         .addr = addr,
7         .flags = KVM_IOEVENTFD_FLAG_PIO,
8         .len = size,
9         .fd = fd,
10    };
11    int r;
12    if (!kvm_enabled()) {
13        return -ENOSYS;
14    }
15    if (datamatch) {
16        kick.flags |= KVM_IOEVENTFD_FLAG_DATAMATCH;
17    }
18    if (!assign) {
19        kick.flags |= KVM_IOEVENTFD_FLAG_DEASSIGN;
20    }
21    r = kvm_vm_ioctl(kvm_state, KVM_IOEVENTFD, &kick);
22    if (r < 0) {
23        return r;
24    }
25    return 0;
26 }
```

KVM_IOEVENTFD的ioctl最终调用了kvm的kvm_ioeventfd函数，后者会调用到kvm_assign_ioeventfd或者kvm_deassign_ioeventfd

```
1 int
2 kvm_ioeventfd(struct kvm *kvm, struct kvm_ioeventfd *args)
3 {
4     if (args->flags & KVM_IOEVENTFD_FLAG_DEASSIGN)
5         return kvm_deassign_ioeventfd(kvm, args);
6
7     return kvm_assign_ioeventfd(kvm, args);
8 }
9
10 static int
11 kvm_assign_ioeventfd(struct kvm *kvm, struct kvm_ioeventfd *args)
12 {
13     int pio = args->flags & KVM_IOEVENTFD_FLAG_PIO;
14     enum kvm_bus bus_idx = pio ? KVM_PIO_BUS : KVM_MMIO_BUS;
15     struct _ioeventfd *p; /* ioeventfd: translate a PIO/MMIO memory write to an eventfd signal. */
16     struct eventfd_ctx *eventfd; /* mostly wait_queue_head_t */
17     int ret;
18
19     /* must be natural-word sized */
20     switch (args->len) {
21     case 1:
22     case 2:
23     case 4:
24     case 8:
25         break;
26     default:
27         return -EINVAL;
28     }
29
30     /* check for range overflow */
31     if (args->addr + args->len < args->addr)
32         return -EINVAL;
33 }
```



2



举报

```

34  /* check for extra flags that we don't understand */35 | if (args->flags & ~KVM_IOEVENTFD_VALID_FLAG_MASK)
36      return -EINVAL;
37
38  eventfd = eventfd_ctx_fdget(args->fd); /* file->private_data */
39  if (IS_ERR(eventfd))
40      return PTR_ERR(eventfd);
41
42  p = kzalloc(sizeof(*p), GFP_KERNEL); /* 分配一个_ioeventfd, 并把内存地址, 长度, eventfd_ctx与其关联起来 */
43  if (!p) {
44      ret = -ENOMEM;
45      goto fail;
46  }
47
48  INIT_LIST_HEAD(&p->list);
49  p->addr = args->addr;
50  p->length = args->len;
51  p->eventfd = eventfd;
52
53  /* The datamatch feature is optional, otherwise this is a wildcard */
54  if (args->flags & KVM_IOEVENTFD_FLAG_DATAMATCH)
55      p->datamatch = args->datamatch;
56  else
57      p->wildcard = true;
58
59  mutex_lock(&kvm->slots_lock);
60
61  /* Verify that there isnt a match already */
62  if (ioeventfd_check_collision(kvm, p)) {
63      ret = -EEXIST;
64      goto unlock_fail;
65  }
66
67  kvm_iodevice_init(&p->dev, &ioeventfd_ops);
68
69  ret = kvm_io_bus_register_dev(kvm, bus_idx, &p->dev); /* 注册到kvm的pio bus或者mmio bus上 */
70  if (ret < 0)
71      goto unlock_fail;
72
73  list_add_tail(&p->list, &kvm->ioeventfds); /* 添加到kvm.ko的ioeventfds的list中 */
74
75  mutex_unlock(&kvm->slots_lock);
76
77  return 0;
78
79 unlock_fail:
80  mutex_unlock(&kvm->slots_lock);
81
82 fail:
83  kfree(p);
84  eventfd_ctx_put(eventfd);
85
86  return ret;
87 }

```



kvm_assign_ioeventfd中, 通过注册一个pio/mmio的地址段和一个fd, 当访问这块内存产生的VMEXIT就会在kvm.ko中被转化成为fd的event notification,

```

1  static const struct kvm_io_device_ops ioeventfd_ops = {
2      .write = ioeventfd_write,
3      .destructor = ioeventfd_destructor,
4  };
5
6  /* MMIO/PIO writes trigger an event if the addr/val match */
7  static int
8  ioeventfd_write(struct kvm_io_device *this, gpa_t addr, int len,
9      const void *val)
10 {
11     struct _ioeventfd *p = to_ioeventfd(this);
12
13     if (!ioeventfd_in_range(p, addr, len, val))
14         return -EOPNOTSUPP;

```



举报


```

15 | 16 | eventfd_signal(p->eventfd, 1);
17 | return 0;
18 | }

```

最终event notification通过eventfd_signal，唤醒vhost线程，整体的流程如下图所示

vhost的控制面和数据面如下图所示

最后，以vhost-net为例说明下vhost网络报文的初始化以及收发流程，e.g.

qemu通过netdev tap,vhost=on在创建网络设备时指定后端基于vhost，net_init_tap会对vhost的每个queue，调用net_init_tap_one初始化vhost通过vhost_net_init完成

```

1 | typedef struct VhostNetOptions {
2 |     VhostBackendType backend_type; /* vhost kernel or userspace */
3 |     NetClientState *net_backend; /* TAPState device */
4 |     void *opaque; /* ioctl vhostfd, /dev/vhost-net */
5 |     bool force;
6 | } VhostNetOptions;
7 |
8 | static int net_init_tap_one(const NetdevTapOptions *tap, NetClientState *peer,
9 |     const char *model, const char *name,
10 |     const char *ifname, const char *script,
11 |     const char *downscript, const char *vhostfdname,
12 |     int vnet_hdr, int fd)
13 | {
14 | ...
15 | if (tap->has_vhost ? tap->vhost :
16 |     vhostfdname || (tap->has_vhostforce && tap->vhostforce)) {
17 |     VhostNetOptions options;
18 |
19 |     options.backend_type = VHOST_BACKEND_TYPE_KERNEL;
20 |     options.net_backend = &s->nc;
21 |     options.force = tap->has_vhostforce && tap->vhostforce;
22 |
23 |     if ((tap->has_vhostfd || tap->has_vhostfds)) {
24 |         vhostfd = monitor_handle_fd_param(cur_mon, vhostfdname);
25 |         if (vhostfd == -1) {
26 |             return -1;
27 |         }
28 |     } else {
29 |         vhostfd = open("/dev/vhost-net", O_RDWR); /* open /dev/vhost-net for ioctl usage */
30 |         if (vhostfd < 0) {
31 |             error_report("tap: open vhost char device failed: %s",
32 |                 strerror(errno));
33 |             return -1;
34 |         }
35 |     }
36 |     qemu_set_cloexec(vhostfd);
37 |     options.opaque = (void *) (uintptr_t) vhostfd;
38 |     s->vhost_net = vhost_net_init(&options); /* 初始化struct vhost_net */
39 |     if (!s->vhost_net) {
40 |         error_report("vhost-net requested but could not be initialized");
41 |         return -1;
42 |     }
43 | }
44 | ...
45 | }
46 |
47 | struct vhost_net {

```



2



举报

```

48 | struct vhost_dev dev; 49 | struct vhost_virtqueue vqs[2];
50 | int backend;
51 | NetClientState *nc;
52 | };
53 |
54 | struct vhost_net *vhost_net_init(VhostNetOptions *options)
55 | {
56 |     int r;
57 |     bool backend_kernel = options->backend_type == VHOST_BACKEND_TYPE_KERNEL;
58 |     struct vhost_net *net = g_malloc(sizeof *net);
59 |
60 |     if (!options->net_backend) {
61 |         fprintf(stderr, "vhost-net requires net backend to be setup\n");
62 |         goto fail;
63 |     }
64 |
65 |     if (backend_kernel) {
66 |         r = vhost_net_get_fd(options->net_backend);
67 |         if (r < 0) {
68 |             goto fail;
69 |         }
70 |         net->dev.backend_features = qemu_has_vnet_hdr(options->net_backend)
71 |             ? 0 : (1 << VHOST_NET_F_VIRTIO_NET_HDR);
72 |         net->backend = r; /* backend设置为NetClientState对应的fd */
73 |     } else {
74 |         net->dev.backend_features = 0;
75 |         net->backend = -1;
76 |     }
77 |     net->nc = options->net_backend; /* nc设置为NetClientState */
78 |
79 |     net->dev.nvqs = 2; /* TX queue和RX queue */
80 |     net->dev.vqs = net->vqs; /* vhost_dev, vhost_net公用vhost_virtqueue */
81 |
82 |     r = vhost_dev_init(&net->dev, options->opaque,
83 |         options->backend_type, options->force); /* 初始化vhost_dev, 这里通过VHOST_SET_OWNER的ioctl创建vhost kthread */
84 |     if (r < 0) {
85 |         goto fail;
86 |     }
87 |     if (!qemu_has_vnet_hdr_len(options->net_backend,
88 |         sizeof(struct virtio_net_hdr_mrg_rxbuf))) {
89 |         net->dev.features &= ~(1 << VIRTIO_NET_F_MRG_RXBUF);
90 |     }
91 |     if (backend_kernel) {
92 |         if (~net->dev.features & net->dev.backend_features) {
93 |             fprintf(stderr, "vhost lacks feature mask %" PRIu64
94 |                 " for backend\n",
95 |                 (uint64_t)(~net->dev.features & net->dev.backend_features));
96 |             vhost_dev_cleanup(&net->dev);
97 |             goto fail;
98 |         }
99 |     }
100 |     /* Set sane init value. Override when guest acks. */
101 |     vhost_net_ack_features(net, 0);
102 |     return net;
103 | fail:
104 |     g_free(net);
105 |     return NULL;
106 | }

```



2



当guest启动成功, qemu会配置相应的vhost, 调用virtio_net_set_status用于开启/关闭virtio-net设备及队列, virtio_net_set_status会调用到vhost_net_start用于启动vhost_net_stop用于关闭vhost队列

```

1 | int vhost_net_start(VirtIODevice *dev, NetClientState *ncs,
2 |     int total_queues)
3 | {
4 |     BusState *qbus = BUS(qdev_get_parent_bus(DEVICE(dev)));
5 |     VirtioBusState *vbus = VIRTIO_BUS(qbus);
6 |     VirtioBusClass *k = VIRTIO_BUS_GET_CLASS(vbus);
7 |     int r, e, i;

```



举报

```

8 |
9 |     if (!vhost_net_device_endian_ok(dev)) {
10 |         error_report("vhost-net does not support cross-endian");
11 |         r = -ENOSYS;
12 |         goto err;
13 |     }
14 |
15 |     if (!k->set_guest_notifiers) {
16 |         error_report("binding does not support guest notifiers");
17 |         r = -ENOSYS;
18 |         goto err;
19 |     }
20 |
21 |     for (i = 0; i < total_queues; i++) {
22 |         vhost_net_set_vq_index(get_vhost_net(ncs[i].peer), i * 2);
23 |     }
24 |     /* 调用virtio_pci_set_guest_notifiers来配置irqfd等信息; 如果没有enable vhost, qemu同样会调用到这里 */
25 |     r = k->set_guest_notifiers(qbus->parent, total_queues * 2, true);
26 |     if (r < 0) {
27 |         error_report("Error binding guest notifier: %d", -r);
28 |         goto err;
29 |     }
30 |     /* 如果tun支持多队列的场景, 会有多个NetClientState, 分别代表tap设备的一个队列, 每个NetClientState都会对应一个vhost_net结构 */
31 |     for (i = 0; i < total_queues; i++) {
32 |         r = vhost_net_start_one(get_vhost_net(ncs[i].peer), dev); /* 对每个队列调用vhost_net_start_one */
33 |
34 |         if (r < 0) {
35 |             goto err_start;
36 |         }
37 |     }
38 |
39 |     return 0;
40 |
41 | err_start:
42 |     while (--i >= 0) {
43 |         vhost_net_stop_one(get_vhost_net(ncs[i].peer), dev);
44 |     }
45 |     e = k->set_guest_notifiers(qbus->parent, total_queues * 2, false);
46 |     if (e < 0) {
47 |         fprintf(stderr, "vhost guest notifier cleanup failed: %d\n", e);
48 |         fflush(stderr);
49 |     }
50 | err:
51 |     return r;
52 | }

```



2



```

1 | static int vhost_net_start_one(struct vhost_net *net,
2 |                               VirtIODevice *dev)
3 | {
4 |     struct vhost_vring_file file = {};
5 |     int r;
6 |
7 |     if (net->dev.started) {
8 |         return 0;
9 |     }
10 |
11 |     net->dev.nvqs = 2; /* vqs包含一个TX virtqueue和一个RX virtqueue */
12 |     net->dev.vqs = net->vqs;
13 |     /* 调用<span style="font-family: Arial, Helvetica, sans-serif;">virtio_pci_set_guest_notifiers来enable vhost ioeventfd */</span>
14 |     r = vhost_dev_enable_notifiers(&net->dev, dev); /* 停止在qemu中处理guest的IO通知, 开始在vhost里处理guest的IO通知 */
15 |     if (r < 0) {
16 |         goto fail_notifiers;
17 |     }
18 |
19 |     r = vhost_dev_start(&net->dev, dev);
20 |     if (r < 0) {
21 |         goto fail_start;
22 |     }
23 |
24 |     if (net->nc->info->poll) {

```



举报

```

25 | net->nc->info->poll(net->nc, false);
    | 26 | }
27 |
28 | if (net->nc->info->type == NET_CLIENT_OPTIONS_KIND_TAP) {
29 |     qemu_set_fd_handler(net->backend, NULL, NULL, NULL);
30 |     file.fd = net->backend;
31 |     for (file.index = 0; file.index < net->dev.nvqs; ++file.index) {
32 |         const VhostOps *vhost_ops = net->dev.vhost_ops;
33 |         r = vhost_ops->vhost_call(&net->dev, VHOST_NET_SET_BACKEND,
34 |             &file);
35 |         if (r < 0) {
36 |             r = -errno;
37 |             goto fail;
38 |         }
39 |     }
40 | }
41 | return 0;
42 | fail:
43 | file.fd = -1;
44 | if (net->nc->info->type == NET_CLIENT_OPTIONS_KIND_TAP) {
45 |     while (file.index-- > 0) {
46 |         const VhostOps *vhost_ops = net->dev.vhost_ops;
47 |         int r = vhost_ops->vhost_call(&net->dev, VHOST_NET_SET_BACKEND,
48 |             &file);
49 |         assert(r >= 0);
50 |     }
51 | }
52 | if (net->nc->info->poll) {
53 |     net->nc->info->poll(net->nc, true);
54 | }
55 | vhost_dev_stop(&net->dev, dev);
56 | fail_start:
57 |     vhost_dev_disable_notifiers(&net->dev, dev);
58 | fail_notifiers:
59 |     return r;
60 | }

```



2



vhost net的各个数据结构之间关系如下图所示

我们来看下内核对vhost_net的定义，e.g.

```

1 | static const struct file_operations vhost_net_fops = {
2 |     .owner      = THIS_MODULE,
3 |     .release    = vhost_net_release,
4 |     .unlocked_ioctl = vhost_net_ioctl,
5 | #ifdef CONFIG_COMPAT
6 |     .compat_ioctl = vhost_net_compat_ioctl,
7 | #endif
8 |     .open      = vhost_net_open,
9 | };
10 |
11 | static struct miscdevice vhost_net_misc = {
12 |     MISC_DYNAMIC_MINOR,
13 |     "vhost-net",
14 |     &vhost_net_fops,
15 | };
16 |
17 | enum {
18 |     VHOST_NET_VQ_RX = 0,
19 |     VHOST_NET_VQ_TX = 1,
20 |     VHOST_NET_VQ_MAX = 2,
21 | };
22 |
23 | enum vhost_net_poll_state {
24 |     VHOST_NET_POLL_DISABLED = 0,
25 |     VHOST_NET_POLL_STARTED = 1,
26 |     VHOST_NET_POLL_STOPPED = 2,
27 | };

```



举报

```

28 |
29 | struct vhost_net {
30 |     struct vhost_dev dev;
31 |     struct vhost_virtqueue vqs[VHOST_NET_VQ_MAX]; /* vhost的virtqueue封装，其handle_kick的回调函数会被ioeventfd唤醒 */
32 |     struct vhost_poll poll[VHOST_NET_VQ_MAX]; /* 对应于NetClientState的socket IO，分别用两个vhost_poll结构体 */
33 |     /* Tells us whether we are polling a socket for TX.
34 |      * We only do this when socket buffer fills up.
35 |      * Protected by tx vq lock. */
36 |     enum vhost_net_poll_state tx_poll_state;
37 | };
38 |
39 | static int vhost_net_open(struct inode *inode, struct file *f)
40 | {
41 |     struct vhost_net *n = kmalloc(sizeof *n, GFP_KERNEL);
42 |     struct vhost_dev *dev;
43 |     int r;
44 |
45 |     if (!n)
46 |         return -ENOMEM;
47 |
48 |     dev = &n->dev;
49 |     n->vqs[VHOST_NET_VQ_TX].handle_kick = handle_tx_kick; /* TX virtqueue->kick的callback函数 */
50 |     n->vqs[VHOST_NET_VQ_RX].handle_kick = handle_rx_kick; /* RX virtqueue->kick的callback函数 */
51 |     r = vhost_dev_init(dev, n->vqs, VHOST_NET_VQ_MAX);
52 |     if (r < 0) {
53 |         kfree(n);
54 |         return r;
55 |     }
56 |
57 |     vhost_poll_init(n->poll + VHOST_NET_VQ_TX, handle_tx_net, POLLOUT, dev); /* 初始化vhost_net的TX vhost_poll */
58 |     vhost_poll_init(n->poll + VHOST_NET_VQ_RX, handle_rx_net, POLLIN, dev); /* 初始化vhost_net的RX vhost_poll */
59 |     n->tx_poll_state = VHOST_NET_POLL_DISABLED;
60 |
61 |     f->private_data = n;
62 |
63 |     return 0;
64 | }

```



2



handle_tx_kick/handle_rx_kick的实现和handle_tx_net/handle_rx_net完全一致，这里为什么要有两个不同的函数呢？看完下面的代码分析后1
不过我先在这里剧透下，handle_tx_kick/handle_rx_kick是阻塞在TX queue/RX queue的kick fd上的回调函数，handle_tx_net/handle_rx_net:
vhost_net TX poll/RX poll上的阻塞函数，无论对于TX还是RX而言，报文的路径都是一个两阶段的过程，e.g.

TX首先是kick virtqueue的fd，之后进行vring的buffer传递，最后通过NetClientState的socket fd发送，但socket有可能会出现缓冲区不足，或者
不够等情况，此时需要poll在socket的fd上阻塞等待。同理RX也是如此，一阶段阻塞在socket fd上，二阶段阻塞在virtqueue kick fd上

前面分析时已经提到过，qemu在vhost_virtqueue_start时，会取得VirtQueue的host_notifier的rfd，并把fd通过VHOST_SET_VRING_KICK传入kvm.ko，这样kvm.k
eventfd_signal通知这个fd。vhost模块会把这个fd和vhost_virtqueue->kick关联起来，并最终调用vhost_poll_start阻塞在这个poll fd上。

当guest发送报文时，ioeventfd触发了vhost_virtqueue的kick fd，POLLIN事件导致vhost_poll_wakeup被调用，最后唤醒了vhost worker线程，
的handle_kick函数，即handle_tx_kick

```

1 | static void handle_tx_kick(struct vhost_work *work)
2 | {
3 |     struct vhost_virtqueue *vq = container_of(work, struct vhost_virtqueue,
4 |         poll.work);
5 |     struct vhost_net *net = container_of(vq->dev, struct vhost_net, dev);
6 |
7 |     handle_tx(net);
8 | }
9 |
10 | static void handle_tx(struct vhost_net *net)
11 | {
12 |     struct vhost_virtqueue *vq = &net->dev.vqs[VHOST_NET_VQ_TX];
13 |     unsigned out, in, s;
14 |     int head;
15 |     struct msghdr msg = {
16 |         .msg_name = NULL,
17 |         .msg_namelen = 0,

```



举报

```

18     .msg_control = NULL,
19     .msg_controllen = 0,
20     .msg_iov = vq->iov,
21     .msg_flags = MSG_DONTWAIT,
22 };
23 size_t len, total_len = 0;
24 int err, wmem;
25 size_t hdr_size;
26 struct vhost_ubuf_ref *uninitialized_var(ubufs);
27 bool zcopy;
28 struct socket *sock = rcu_dereference(vq->private_data); /* NetClientState对应的socket以private_data的形式保存在vhost_virtque
29 if (!sock)
30     return;
31
32 wmem = atomic_read(&sock->sk->sk_wmem_alloc);
33 if (wmem >= sock->sk->sk_sndbuf) { /* 已经申请的socket写内存, 超过了发送缓冲区 */
34     mutex_lock(&vq->mutex);
35     tx_poll_start(net, sock); /* 此时无法发送, 阻塞等待在sock上 */
36     mutex_unlock(&vq->mutex);
37     return;
38 }
39
40 mutex_lock(&vq->mutex);
41 vhost_disable_notify(&net->dev, vq); /* disable virtqueue的notify通知, 通过VRING_USED_F_NO_NOTIFY标志位 */
42
43 if (wmem < sock->sk->sk_sndbuf / 2)
44     tx_poll_stop(net);
45 hdr_size = vq->vhost_hlen;
46 zcopy = vq->ubufs;
47
48 for (;;) {
49     /* Release DMAs done buffers first */
50     if (zcopy)
51         vhost_zerocopy_signal_used(vq);
52
53     head = vhost_get_vq_desc(&net->dev, vq, vq->iov, /* 从last_avail_idx开始, 把avail_desc内容拷贝过来 */
54                             ARRAY_SIZE(vq->iov),
55                             &out, &in,
56                             NULL, NULL);
57     /* On error, stop handling until the next kick. */
58     if (unlikely(head < 0))
59         break;
60     /* Nothing new? Wait for eventfd to tell us they refilled. */
61     if (head == vq->num) { /* 此时vq->avail_idx == vq->last_avail_idx, 前端没有新buf过来 */
62         int num_pends;
63
64         wmem = atomic_read(&sock->sk->sk_wmem_alloc);
65         if (wmem >= sock->sk->sk_sndbuf * 3 / 4) {
66             tx_poll_start(net, sock);
67             set_bit(SOCK_ASYNC_NOSPACE, &sock->flags);
68             break;
69         }
70         /* If more outstanding DMAs, queue the work.
71          * Handle upend_idx wrap around
72          */
73         num_pends = likely(vq->upend_idx >= vq->done_idx) ?
74             (vq->upend_idx - vq->done_idx) :
75             (vq->upend_idx + UIO_MAXIOV - vq->done_idx);
76         if (unlikely(num_pends > VHOST_MAX_PEND)) {
77             tx_poll_start(net, sock);
78             set_bit(SOCK_ASYNC_NOSPACE, &sock->flags);
79             break;
80         }
81         if (unlikely(vhost_enable_notify(&net->dev, vq))) { /* 重新调用vhost_enable_notify打开event notify flag */
82             vhost_disable_notify(&net->dev, vq); /* vhost_enable_notify返回false, 说明avail_idx有了变化, 那么continue */
83             continue;
84         }
85         break;
86     }
87     if (in) { /* Tx应该全部是out */
88         vq_err(vq, "Unexpected descriptor format for TX: "

```



2



赏



举报

```

89     "out %d, int %d\n", out, in);
90     break;
91 }
92 /* Skip header. TODO: support TSO. */
93 s = move_iovec_hdr(vq->iov, vq->hdr, hdr_size, out); /* hdr_size是VNET_HDR的元数据, 里面没有实际报文内容 */
94 msg.msg_iovlen = out;
95 len = iov_length(vq->iov, out);
96 /* Sanity check */
97 if (!len) {
98     vq_err(vq, "Unexpected header len for TX: "
99             "%zd expected %zd\n",
100             iov_length(vq->hdr, s), hdr_size);
101     break;
102 }
103 /* use msg_control to pass vhost zerocopy ubuf info to skb */
104 if (zcopy) {
105     vq->heads[vq->upend_idx].id = head;
106     if (len < VHOST_GOODCOPY_LEN) {
107         /* copy don't need to wait for DMA done */
108         vq->heads[vq->upend_idx].len =
109             VHOST_DMA_DONE_LEN;
110         msg.msg_control = NULL;
111         msg.msg_controllen = 0;
112         ubufs = NULL;
113     } else {
114         struct ubuf_info *ubuf = &vq->ubuf_info[head];
115
116         vq->heads[vq->upend_idx].len = len;
117         ubuf->callback = vhost_zerocopy_callback;
118         ubuf->arg = vq->ubufs;
119         ubuf->desc = vq->upend_idx;
120         msg.msg_control = ubuf;
121         msg.msg_controllen = sizeof(ubuf);
122         ubufs = vq->ubufs;
123         kref_get(&ubufs->kref);
124     }
125     vq->upend_idx = (vq->upend_idx + 1) % UIO_MAXIOV;
126 }
127 /* TODO: Check specific error and bomb out unless ENOBUFS? */
128 err = sock->ops->sendmsg(NULL, sock, &msg, len);
129 if (unlikely(err < 0)) {
130     if (zcopy) {
131         if (ubufs)
132             vhost_ubuf_put(ubufs);
133         vq->upend_idx = ((unsigned)vq->upend_idx - 1) %
134             UIO_MAXIOV;
135     }
136     vhost_discard_vq_desc(vq, 1); /* 发送失败, 回退last_avail_idx */
137     if (err == -EAGAIN || err == -ENOBUFS)
138         tx_poll_start(net, sock); /* 阻塞等待vhost_net->poll之后尝试重新发送 */
139     break;
140 }
141 if (err != len)
142     pr_debug("Truncated TX packet: "
143             "len %d != %zd\n", err, len);
144 if (!zcopy)
145     vhost_add_used_and_signal(&net->dev, vq, head, 0); /* 更新virtqueue used ring部分, e.g. used_elem, last_used_idx */
146 else
147     vhost_zerocopy_signal_used(vq);
148 total_len += len;
149 if (unlikely(total_len >= VHOST_NET_WEIGHT)) {
150     vhost_poll_queue(&vq->poll); /* 超出了quota, 重新入队列等待调度 */
151     break;
152 }
153 }
154
155 mutex_unlock(&vq->mutex);
156 }

```



2



举报

收包过程首先是vhost阻塞在NetClientState的socket上, e.g.

vhost_poll_init(n->poll + VHOST_NET_VQ_RX, handle_rx_net, POLLIN, dev)

```
1 static void handle_rx_net(struct vhost_work *work)
2 {
3     struct vhost_net *net = container_of(work, struct vhost_net,
4         poll[VHOST_NET_VQ_RX].work);
5     handle_rx(net);
6 }
7
8 static void handle_rx(struct vhost_net *net)
9 {
10    struct vhost_virtqueue *vq = &net->dev.vqs[VHOST_NET_VQ_RX];
11    unsigned uninitialized_var(in), log;
12    struct vhost_log *vq_log;
13    struct msghdr msg = {
14        .msg_name = NULL,
15        .msg_namelen = 0,
16        .msg_control = NULL, /* FIXME: get and handle RX aux data. */
17        .msg_controllen = 0,
18        .msg_iov = vq->iov,
19        .msg_flags = MSG_DONTWAIT,
20    };
21
22    struct virtio_net_hdr_mrg_rxbuf hdr = {
23        .hdr.flags = 0,
24        .hdr.gso_type = VIRTIO_NET_HDR_GSO_NONE
25    };
26
27    size_t total_len = 0;
28    int err, headcount, mergeable;
29    size_t vhost_hlen, sock_hlen;
30    size_t vhost_len, sock_len;
31
32    struct socket *sock = rcu_dereference(vq->private_data);
33
34    if (!sock)
35        return;
36
37    mutex_lock(&vq->mutex);
38    vhost_disable_notify(&net->dev, vq); /* disable virtqueue event notify机制 */
39    vhost_hlen = vq->vhost_hlen;
40    sock_hlen = vq->sock_hlen;
41
42    vq_log = unlikely(vhost_has_feature(&net->dev, VHOST_F_LOG_ALL)) ?
43        vq->log : NULL;
44    mergeable = vhost_has_feature(&net->dev, VIRTIO_NET_F_MRG_RXBUF);
45
46    while ((sock_len = peek_head_len(sock->sk))) { /* 下一个报文的长度 */
47        sock_len += sock_hlen;
48        vhost_len = sock_len + vhost_hlen;
49        headcount = get_rx_bufs(vq, vq->heads, vhost_len, /* get_rx_bufs用于从virtqueue中拿到多个avail desc, */
50            &in, vq_log, &log, /* 直到满足所有这些iov加起来可以容纳下一个报文的长度 */
51            likely(mergeable) ? UIO_MAXIOV : 1); /* 相当于多次调用vhost_get_vq_desc */
52
53        /* On error, stop handling until the next kick. */
54        if (unlikely(headcount < 0))
55            break;
56        /* OK, now we need to know about added descriptors. */
57        if (!headcount) {
58            if (unlikely(vhost_enable_notify(&net->dev, vq))) {
59                /* They have slipped one in as we were
60                 * doing that: check again. */
61                vhost_disable_notify(&net->dev, vq);
62                continue;
63            }
64            /* Nothing new? Wait for eventfd to tell us
65             * they refilled. */
66            break;
67        }
```



2



举报


```

68 | /* We don't need to be notified again. */ 69 | if (unlikely((vhost_hlen)))
70 | /* Skip header. TODO: support TSO. */
71 | move_iovec_hdr(vq->iov, vq->hdr, vhost_hlen, in);
72 | else
73 | /* Copy the header for use in VIRTIO_NET_F_MRG_RXBUF:
74 |  * needed because sendmsg can modify msg_iov. */
75 | copy_iovec_hdr(vq->iov, vq->hdr, sock_hlen, in);
76 | msg.msg_iovlen = in;
77 | err = sock->ops->recvmsg(NULL, sock, &msg,
78 |      sock_len, MSG_DONTWAIT | MSG_TRUNC); /* 报文被收到virtqueue->iov里面 */
79 | /* Userspace might have consumed the packet meanwhile:
80 |  * it's not supposed to do this usually, but might be hard
81 |  * to prevent. Discard data we got (if any) and keep going. */
82 | if (unlikely(err != sock_len)) {
83 |     pr_debug("Discarded rx packet: "
84 |         " len %d, expected %zd\n", err, sock_len);
85 |     vhost_discard_vq_desc(vq, headcount); /* 回滚used ring */
86 |     continue;
87 | }
88 | if (unlikely(vhost_hlen) &&
89 |     memcpy_toiovecend(vq->hdr, (unsigned char *)&hdr, 0,
90 |         vhost_hlen)) {
91 |     vq_err(vq, "Unable to write vnet_hdr at addr %p\n",
92 |         vq->iov->iov_base);
93 |     break;
94 | }
95 | /* TODO: Should check and handle checksum. */
96 | if (likely(mergeable) &&
97 |     memcpy_toiovecend(vq->hdr, (unsigned char *)&headcount,
98 |         offsetof(typeof(hdr), num_buffers),
99 |         sizeof hdr.num_buffers)) {
100 |     vq_err(vq, "Failed num_buffers write");
101 |     vhost_discard_vq_desc(vq, headcount);
102 |     break;
103 | }
104 | vhost_add_used_and_signal_n(&net->dev, vq, vq->heads,
105 |     headcount); /* 添加多个vring_used_elem, 并notify前端 */
106 | if (unlikely(vq_log))
107 |     vhost_log_write(vq, vq_log, log, vhost_len);
108 | total_len += vhost_len;
109 | if (unlikely(total_len >= VHOST_NET_WEIGHT)) {
110 |     vhost_poll_queue(&vq->poll); /* 超出了quota, 重新入队列等待, 注意此时加入的是vq的poll, 下次会触发调用handle_rx_kick */
111 |     break;
112 | }
113 | }
114 |
115 | mutex_unlock(&vq->mutex);
116 | }

```



2



👍 点赞 2 ☆ 收藏 🔄 分享 ...



majieyue

发布了80 篇原创文章 · 获赞 21 · 访问量 48万+



工位出租600元/月

出租工位



我要吐槽两句...



举报

virtio,vhost 和vhost-user

阅读数 6934

随着qemu2.1的发布, 可以看到, qemu支持了vhost-user. 从介绍可以看出, 这是把原来vhost-backend从kernel... 博文 来自: jincm的专栏

Qemu Vhost Block架构分析（下）

一. 概述在上半部已经将VirtIO-blk

KVM vhost与VM之间的数据流交换

vm virtio driver ...

vhost 的演变

vhost 的演变virtio-net的后端驱动经历过从virtio-net后端----到内核态vhost-net，再到用户态vhost-uservirtio-net...



传统ERP已经过时，2019流行的ERP系统是这一款！
好用的erp系统

dppk vhost研究(一)

Vhost/Virtio是一种半虚拟化的设备抽象接口规范，在Qemu和KVM中的得到了广泛的应用，在客户机操作系统中实...

vhost

vhost的用户态程序接口定义在/usr/include/linux/vhost.hvhost目前只支持tap network backendvhost.h/vhost.c---

vhost：一种 virtio 高性能的后端驱动实现

什么是 vhostvhost 是 virtio 的一种后端实现方案，在 virtio 简介中，我们已经提到 virtio 是一种半虚拟化的实现方...

探秘DPDK Virtio的不同路径，so easy!

DPDK点击蓝字关注DPDK开源社区什么是Vhost/VirtioVhost/Virtio是一种半虚拟化的设备抽象接口规范，在Qemu...

虚拟化virtio_net的网络配置问题

目前正在构建基于ARM64服务器的虚拟化环境，遇到了如下的问题virtio_net的问题，请大侠们指点 目标拓扑如下： 主机...

virtio后端方案vhost_网络_majieyue的专栏-CSDN博客

virtio后端方案vhost - majieyue的专栏 - CSDN博客



工位出租600元/月
出租工位

vhost网络设备才会添加eventfd

vhost网络设备就会从virtio_set_status中调用memory_region_add_eventfd，然后通过kvm_io_ioeventfd_add添...

...高性能的后端驱动实现 - weixin_30719711的博客 - CSDN博客

virtio,vhost 和vhost-user_运维_jincm的专栏-CSDN博客

DPDK系列之十六： Virtio技术分析之二，vhost技术对于virtio的增强

一、前言



丛林溪水鱼
205篇文章
排名:千里之外



Kewei-Yu
11篇文章
排名:千里之外



天弓 (tg)
27篇文章
排名:千里之外



zhang123ding
7篇文章
排名:千里之外

ovs virtio vhost通信流程_kklvsports的专栏-CSDN博客

virtio前端驱动能够通知后端的原理_kklvsports的专栏-CSDN博客

vhost源码分析

1、概述 vhost的大致原理就是qemu在Guest、Host之间创建一些共享buffer，Guest作为生产者往buffer填充可用描...

博文 来自: u012377033 阅读量 3350

博文 来自: softgmx的专栏 阅读量 2万+

博文 来自: zhang123ding 阅读量 47

博文 来自: 潜心修行, 阅读量 19

博文 来自: cybertan的专栏 阅读量 7068

博文 来自: weixin_34137799 阅读量 135

博文 来自: weixin_37097605 阅读量 38

博文 来自: 六六哥的博客 阅读量 774

博文 来自: cloudvtech的博客 阅读量 1767

博文 来自: zheng的博客 阅读量 414

Qemu Vhost Block架构分析（上）

阅读数 4567

一.简介一说到Vhost Block是一个很

博文 来自: u01237703

virtio的vring队列_运维_majieyue的专栏-CSDN博客

virtio-blk后端处理-请求接收、解析、提交 - 慢慢游 - CSDN博客

virtio的io路径 vhost的io路径 和vhost-user vhost-user *****网络设备

阅读数 127

随着qemu2.1的发布，可以看到，qemu支持了vhost-user。从介绍可以看出，这是把原来vhost-backend从kernel…

博文 来自: tycoon的专

virtio后端驱动详解 - sdulibh的专栏

virtio/vhost介绍

(1) Virtio Introduction--Paravirtualized I/O with KVM and lguestVirtio is an I/O virtualization fram…

博文 来自: Oliverlyn的

DPPDK-实战之Virtio/vhost（虚拟技术）

阅读数 1795

0x01 缘由 在现在云计算和大量的数据中心建设过程中，虚拟化技术快速发展，也来了解下vhost。0x02 介绍 通过…

博文 来自: 3-Number

virtio/vhost的速率机制 (by joshua)

阅读数 687

版权声明：可以任意转载，转载时请务必以超链接形式标明文章原始出处和作者信息及本版权声明 (作者：张华 发表…

博文 来自: 技术并艺术着

host_notifier, 虚拟机通过VHOST发包流程(基于kernel3.10.0 && qemu 2.0.0)

阅读数 1205

1. 系统中的eventfd_add(), 以及memory_listener的注册static MemoryListener kvm_memory_listener = { .even…

博文 来自: 六六哥的博客

Vhost Architecture(基于kernel3.10.0 & qemu2.0.0)

阅读数 1973

在前面的文章中在介绍virtio机制中，可以看到在通常的应用中一般使用QEMU用户态程序来模拟I/O访问，而Guest…

博文 来自: 六六哥的博客



工位出租600元/月

出租工位

ovs virtio vhost通信流程

阅读数 3132

转自： 点击打开链接版本说明qemu version： 2.6.0kernel version： 3.10.102简而言之vhost模块需要提前加载，注…

博文 来自: kklvsports的专栏

QEMU通过virtio接收报文处理流程（QEMU2.0.0）

阅读数 1596

1. set_guest_notifiers初始化流程static void virtio_pci_bus_class_init(ObjectClass *klass, void *data){ k-…

博文 来自: 六六哥的博客

dppdk vhost研究(二)

阅读数 1821

继续本专题的研究，关于本专题前期的内容请参考[这里](http://blog.csdn.net/me_blue/article/details/77854595)…

博文 来自: 潜心修行，独立思考

大学四年自学走来，这些私藏的实用工具/学习网站我贡献出来了

阅读数 70万+

大学四年，看课本是不可能一直看课本的了，对于学习，特别是自学，善于搜索网上的一些资源来辅助，还是非常…

博文 来自: 帅地

在中国程序员是青春饭吗？

阅读数 31万+

今年，我也32了，为了不给大家误导，咨询了猎头、圈内好友，以及年过35岁的几位老程序员………舍了老脸去揭人…

博文 来自: 启航

卸载 x 雷某度！GitHub 标星 1.5w+，从此我只用这款全能高速下载工具！

阅读数 18万+

作者 | Rocky0429来源 | Python空间大家好，我是 Rocky0429，一个喜欢在网上收集各种资源的蒟蒻…网上资源眼花…

博文 来自: Rocky0429

为什么猝死的都是程序员，基本上不见产品经理猝死呢？

阅读数 19万+

相信大家时不时听到程序员猝死的消息，但是基本上听不到产品经理猝死的消息，这是为什么呢？我们先百度搜一下…

博文 来自: 曹银飞的专栏

毕业5年，我问遍了身边的大佬，总结了他们的学习方法

阅读数 1万+

我问了身边10个大佬，总结了他们的学习方法，原来成功都是有迹可循的。

博文 来自: 敖丙

推荐10个堪称神器的学习网站

阅读数 29万+

每天都会收到很多读者的私信，问我：“二哥，有什么推荐的学习网站吗？最近很浮躁，手头的一些网站都看烦了，…

博文 来自: 沉默王二



强烈推荐10本程序员必读的书

很遗憾，这个春节注定是刻骨铭心的，新型冠状病毒让每个人的神经都是紧绷的。那些处在武汉的白衣天使们，尤其…

博文 来自： 沉默王二

阅读数 11万+

2 万+

万+

万+

万+

万+

万+

万+

为什么说程序员做外包没前途？

之前做过不到3个月的外包，2020的第一天就被释放了，2019年还剩1天，我从外包公司离职了。我就谈谈我个人的…

博文 来自： dotNet全栈

B 站上有哪些很好的学习资源？

哇说起B站，在小九眼里就是宝藏般的存在，放年假宅在家时一天刷6、7个小时不在话下，更别提今年的跨年晚会，…

博文 来自： 九章算法的

拼多多面试问了数据库基础知识，今天分享出来

一个SQL在数据库是怎么执行的，你是否了解过了呢？

博文 来自： 敖丙

新来个技术总监，禁止我们使用Lombok！

我有个学弟，在一家小型互联网公司做Java后端开发，最近他们公司新来了一个技术总监，这位技术总监对技术细节…

博文 来自： HollisChua

大学四年，因为知道这些开发工具，我成为别人眼中的大神

亲测全部都很好用，自己开发都离不开的软件，如果你是学生可以看看，提前熟悉起来。…

博文 来自： 敖丙

在三线城市工作爽吗？

我是一名程序员，从正值青春年华的 24 岁回到三线城市洛阳工作，至今已经 6 年有余。一不小心又暴露了自己的实…

博文 来自： 沉默王二

这些插件太强了，Chrome 必装！尤其程序员！

推荐 10 款我自己珍藏的 Chrome 浏览器插件

博文 来自： 沉默王二

@程序员：GitHub这个项目快薅羊毛

今天下午在朋友圈看到很多人都在发github的羊毛，一时没明白是怎么回事。后来上百度搜索了一下，原来真有这回…

博文 来自： dotNet全栈开发

没用过这些 IDEA 插件？怪不得写代码头疼

使用插件，可以提高开发效率。对于开发人员很有帮助。这篇博客介绍了IDEA中最常用的一些插件。…

博文 来自： 扬帆向海的博客

Java C语言 Python C++ C# Visual Basic .NET JavaScript PHP SQL Go语言 R语言 Assembly language Swift Ruby

MATLAB PL/SQL Perl Visual Basic Objective-C Delphi/Object Pascal Unity3D

©2019 CSDN 皮肤主题: 大白 设计师: CSDN官方博客

majieyue

TA的个人主页>

原创

粉丝

获赞

评论

访问

80

152

21

28

48万+

等级: 博客 6

周排名: 9万+

积分: 5293

总排名: 1万+

勋章:

关注

私信

最新文章

OVS datapath流表结构及匹配过程

ovs的upcall及ofproto-dpif处理细节

ovs vswitchd的启动分析

ovs的netdev, ofproto以及dpif etc.

virtio network驱动分析


分类专栏


	C++	1篇
	Linux	12篇
	Python	3篇
	服务器开发	1篇
	Xen	20篇
	服务器系统	2篇
	存储技术	4篇
	Linux内核	25篇
	Java	1篇
	工具	1篇
	erlang	
	OpenvSwitch	14篇
	KVM/QEMU	6篇
	SDN	
	用户态协议栈	


归档				
2016				
11月	10月	5月	4月	
1篇	3篇	2篇	4篇	
3月				
3篇				
2014				
10月	6月	4月	3月	
1篇	1篇	1篇	1篇	
2013				


热门文章	
open vswitch研究：ovs的安装和使用	
阅读数 45571	
linux内核网络协议栈学习笔记：关于GRO/GSO/LRO/TSO等patch的分析和测试	
阅读数 28631	
open vswitch研究: ovsdb	
阅读数 19318	
virtio的qemu总线与设备模型	
阅读数 13471	
virtio的vring队列	
阅读数 13338	


最新评论	
linux内核网络协议栈学习笔记：...	
notbaron: [reply]chinasiyu[reply] 对的，是网卡硬件特性	
linux内核网络协议栈学习笔记（...	


2



















举报

lkcool7: 赞

open vswitch研究：ov...
NK_test: 运行到rmmod bridge后开发机崩了。。


virtio的vring队列
chen_1020: 上面的notify机制，貌似讲的不太清楚，根据代码来看客户端每次add_sg都会判断 ...

open vswitch研究：vs...
zoushidexing: 赞~

 QQ客服

 kefu@csdn.net

 客服论坛

 400-660-0108

工作时间 8:30-22:00

[关于我们](#) [招聘](#) [广告服务](#) [网站地图](#)

京ICP备19004658号 经营性网站备案信息

 公安备案号 11010502030143

京网文〔2020〕1039-165号

©1999-2020 北京创新乐知网络技术有限公司

网络110报警服务

北京互联网违法和不良信息举报中心

中国互联网举报中心 家长监护

版权与免责声明 版权申诉



2



举报