ToBeDone

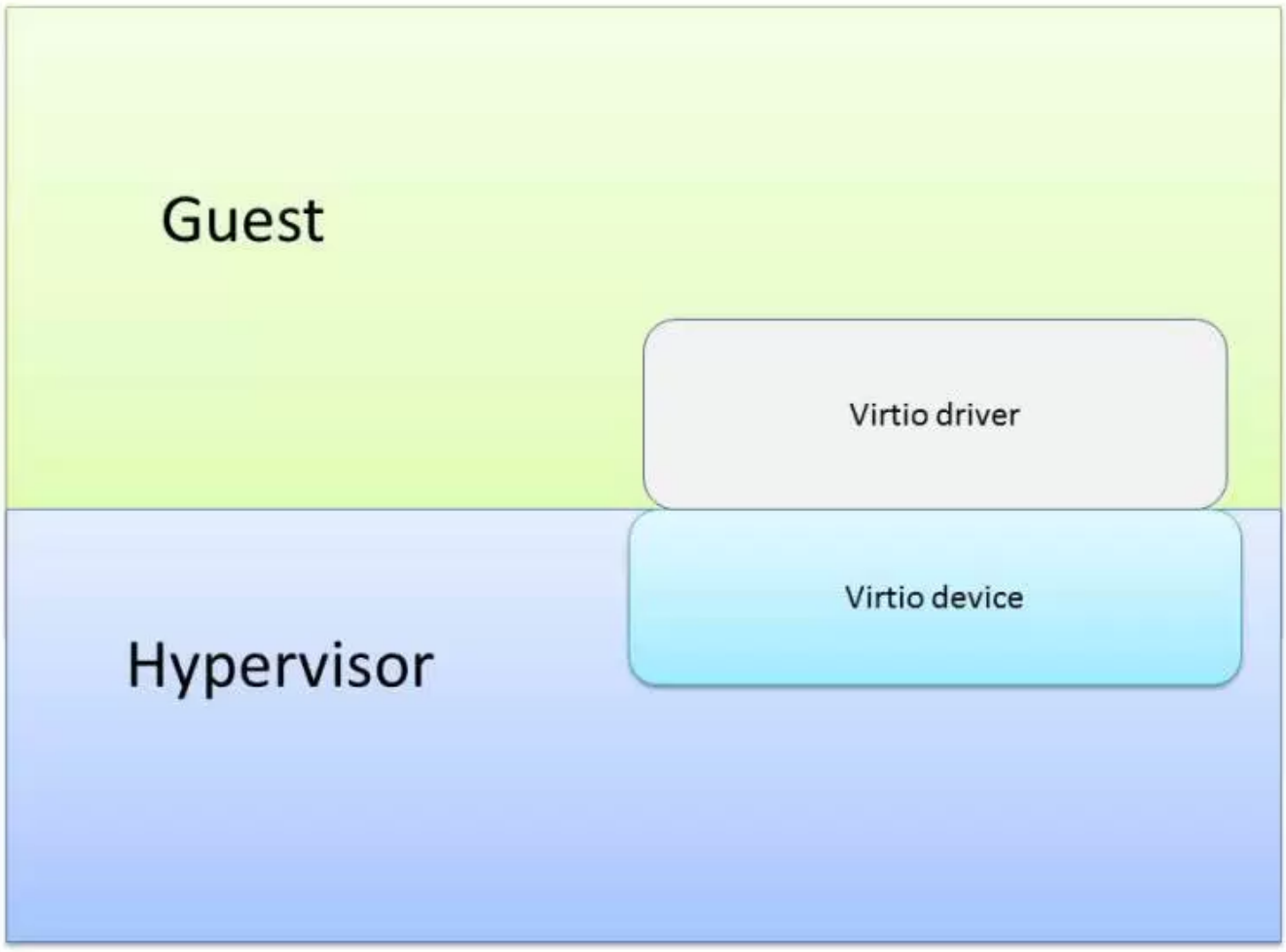
Programming, linux, network, algorithm and more

27OCT2014

virtio guest side implementation: PCI, virtio device, virtio net and virtqueue

posted in driver, Linux, virtio, virtualization by Jipan Yang

With the publishing of OASIS virtio specification version 1.0, virtio made another big step in becoming an official standard from a De-Facto standard for virtual i/o device in paravirtualization environment.



(https://jipanyang.files.wordpress.com/2014/10/virtio_device_driver1.jpg)

virtio device

For virtio, device emulation is done in hypervisor/host. They are commonly implemented as PCI devices, virtio over memory mapped device (MMIO) or channel i/o is also seen.

Take QEMU as example, it emulates the control plane of virtio PCI device like device status, feature bits and device configuration space, while the implementation of virtqueue backend data plane has three options so far:

- **Virtio backend running inside QEMU**

virtqueue notification and actual data access are done directly by QEMU. virtio.c contains the major implementation.

- **Virtio backend inside host kernel, vhost**

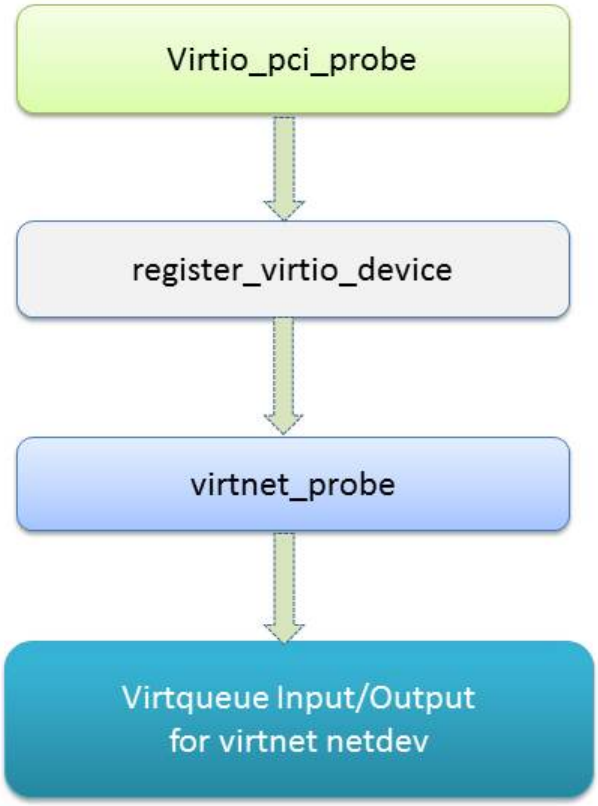
QEMU help setup kick/irq eventfd, vhost utilizes them to communicates with drivers in guest directly for virtqueue notification. Linux kernel module vhost sends/receives data via virtqueue with guest without exiting to host user space. vhost-worker is the kernel thread handling the notification and data buffer, the arrangement that enables it to access whole QEMU/guest address space is that: QEMU issues VHOST_SET_OWNER ioctl call to saves the mm context of qemu process in vhost_dev, then vhost-worker thread take on the specified mm context. Check use_mm() (http://lxr.free-electrons.com/source/mm/mmu_context.c#L20) kernel function.

- **Virtio backend running in separate userspace process**

With the development and increasing popularity of user space application/SDK like snabbswitch and dpdk-ovs for network switching, there is need for host user space applications to perform direct virtio data transfer with guest OS. vhost-user or user space vhost is feature in QEMU for addressing this request. The key for this solution is to have shared memory between guest OS and the host user application, and ioeventfd/irqfd for virtqueue event notification. snabswitch probably is the first application taking advantage of this feature, dpdk-ovs is picking up though it already has another implementation of user space vhost which relies on user space cuse driver to emulate kernel vhost functions.

virtio device driver

Guest OS implements the drivers for driving virtio device presented by underlying hypervisor. Here we walk through the virtio PCI network card driver example in Linux kernel.



(https://jipanyang.files.wordpress.com/2014/10/virtio_probe1.jpg)

Virtio PCI

As that for a regular PCI device, virtio pci driver fills data and call back functions into standard pci_driver structure, among them the most important parts are the pci_device_id table (http://lxr.free-electrons.com/source/drivers/virtio/virtio_pci.c#L94) and probe function. All virtio devices use vendor id of ox1af4. virtio_pci_probe() (http://lxr.free-electrons.com/source/drivers/virtio/virtio_pci.c#L679) will be called if a virtio pci device with that specific vendor id is detected on PCI bus. The probe function allocates a virtio_pci_device structure which saves the necessary information like pci_dev pointer, MSI-X config and virtqueues list. virtio_device structure is also part of virtio_pci_device, "struct virtio_config_ops virtio_pci_config_ops (http://lxr.free-electrons.com/source/drivers/virtio/virtio_pci.c#L655)" provides a set of virtio device status/feature/configuration/virtqueue callback functions and it will be linked into virtio_device.

At the end of virtio PCI probe processing, PCI subsystem vendor and device id will be assigned to the new virtio_device as its vendor and device id, then the newly created virtio_device which is inside virtio_pci_device structure will be register onto virtio bus: "int register_virtio_device(struct virtio_device *dev)" (http://lxr.free-electrons.com/source/drivers/virtio/virtio.c#L200)

Virtio netdev

virtio_net_driver is a Linux kernel driver module registered with virtio_bus. Note that virtio_bus (http://lxr.free-electrons.com/source/drivers/virtio/virtio.c#L176) has been registered during kernel initialization : core_initcall(virtio_init); and the code is built in kernel and not a loadable module.

The virtio_driver probe fucntion virtnet_probe() (http://lxr.free-electrons.com/source/drivers/net/virtio_net.c#L1674) will be called once a virtio device with device ID VIRTIO_ID_NET has been detected. To work as a network device, net_device structure is created via alloc_etherdev_mq() , various network features are checked. Within the probe function, TX/RX virtqueue will be created/initialized, the find_vqs() callback function in virtio_pci_config_ops (http://lxr.free-electrons.com/source/drivers/virtio/virtio_pci.c#L655) of virtio_device will called during this process. virtnet_info works as the private data of virtio net_device to link net_device and virtio_device together.

net_device_ops virtnet_netdev (http://lxr.free-electrons.com/source/drivers/net/virtio_net.c#L1385) is also configured for the new net device which will be ready to function as a network interface card for sending/receiving data.

Virtio netdev RX/TX

At least 2 virtqueues will be created for virtio net_device interface, one for TX and the other one for RX. If host supports flow steering feature VIRTIO_NET_F_MQ (http://lxr.free-electrons.com/source/include/uapi/linux/virtio_net.h#L54), then more than one pair of queues may be created. Supporting of VIRTIO_NET_F_CTRL_VQ (http://lxr.free-electrons.com/source/include/uapi/linux/virtio_net.h#L48) adds another virtqueue which is used by driver to send commands to manipulate various features of device which would not easily map into the PCI configuration space.

In RX direction, virtnet_receive() (http://lxr.free-electrons.com/source/drivers/net/virtio_net.c#L731) is called by virtnet_poll() or virtnet_busy_poll() depending on kernel CONFIG_NET_RX_BUSY_POLL option setting. virtqueue_get_buf() (http://lxr.free-electrons.com/source/drivers/virtio/virtio_ring.c#L524) is the next layer function which gets the data from host. As special requirement for virtio driver, it needs to add in buffer for host to put data there. Function try_fill_recv() (http://lxr.free-electrons.com/source/drivers/net/virtio_net.c#L658) serves that purpose, virtqueue_add_inbuf() (http://lxr.free-electrons.com/source/drivers/virtio/virtio_ring.c#L380) is called eventually to expose input buffers to the other end.

For virtnet device driver, the TX processing starts with the call back function start_xmit() (http://lxr.free-electrons.com/source/drivers/net/virtio_net.c#L914) within virtnet_netdev , it first frees up any pending old buffer via free_old_xmit_skbs() (http://lxr.free-electrons.com/source/drivers/net/virtio_net.c#L828), then goes into xmit_skb() (http://lxr.free-electrons.com/source/drivers/net/virtio_net.c#L847) which calls

virtqueue_add_outbuf() (http://lxr.free-electrons.com/source/drivers/virtio/virtio_ring.c#L358) located in virtio_ring.c. virtqueue_get_buf() (http://lxr.free-electrons.com/source/drivers/virtio/virtio_ring.c#L524)is also called by free_old_xmit_skbs() (http://lxr.free-electrons.com/source/drivers/net/virtio_net.c#L828) to get used buffer and release the virtqueue ring descriptor back to desc free list.

Virtqueue

virtqueue is the fundamental building block of virtio, it is the mechanism for bulk data transport on virtio devices. From virtqueque point of view, both virtio netdev RX and TX queues are virtually the same. Each virtqueue consists of three parts: Descriptor table, Available ring and Used ring.

Each descriptor could refer to a buffer that driver uses for virtio device, the addr field of it points to a physical address of guest.

Available ring stores index of entries in descriptor table for which guest informs host/hypervisor that the buffer descriptor points to is available for use. In the perspective of guest virtio netdev TX queue, the buff is filled with data to be processed by host. While for guest virtio netdev RX queue, the buffer is empty and should be filled by host. virtqueue_add() (http://lxr.free-electrons.com/source/drivers/virtio/virtio_ring.c#L187) which is called by both virtqueue_add_outbuf() (http://lxr.free-electrons.com/source/drivers/virtio/virtio_ring.c#L358) & virtqueue_add_inbuf() (http://lxr.free-electrons.com/source/drivers/virtio/virtio_ring.c#L380) operates on the available ring. Guest performs write operation on available ring data structure, host reads it only.

Used ring is where the virtio device (host) returns buffer once it is done with them. it is only written to by the device, and read by the driver (guest). For both virtio netdev RX/TX queues, detach_buf() which is called by virtqueue_get_buf() will take the descriptors indicated by used ring and put them back to descriptor table free list.

Refer to Virtual I/O Device (VIRTIO) Version 1.0 (http://docs.oasis-open.org/virtio/virtio/v1.0/cs01/virtio-v1.0-cs01.pdf) and virtio: Towards a De-Facto Standard For Virtual I/O Devices (http://www.ozlabs.org/~rusty/virtio-spec/virtio-paper.pdf) for authentic information about virtio.

1 Comment

One thought on “virtio guest side implementation: PCI, virtio device, virtio net and virtqueue”

Pingback: How to setup virsh or libvirt or VirtIO in a custom kernel | Life in Linux Kernel