Lauren·weblog

Virtio网络发包过程分析

2018-08-24 | 2.虚拟化 , virtio | 1596 | 1

19k | 18 分钟

前言

本文将分析Virtio网络发包过程，包括Frontend如何发送数据给Backend，以及Backend收到数据后的处理。

正文

Frontend： Transmit packet

查看driver注册的ops函数，其中指定的发送函数为**start_xmit**，我们从这里开始分析发包过程。

> virtio_net.c : virtnet_netdev

```
static const struct net_device_ops virtnet_netdev = {
	.ndo_open            = virtnet_open,
	.ndo_stop            = virtnet_close,
	.ndo_start_xmit      = start_xmit,
	.ndo_validate_addr   = eth_validate_addr,
	.ndo_set_mac_address = virtnet_set_mac_address,
	.ndo_set_rx_mode     = virtnet_set_rx_mode,
	.ndo_change_mtu      = virtnet_change_mtu,
	.ndo_get_stats64     = virtnet_stats,
	.ndo_vlan_rx_add_vid = virtnet_vlan_rx_add_vid,
	.ndo_vlan_rx_kill_vid = virtnet_vlan_rx_kill_vid,
	.ndo_select_queue    = virtnet_select_queue,
#ifdef CONFIG_NET_POLL_CONTROLLER
	.ndo_poll_controller = virtnet_netpoll,
#endif
	.ndo_features_check  = passthru_features_check,
};
```

90%

start_xmit函数首先获取sk_buff对应的send_queue，**free_old_xmit_skbs**释放backend处理过的desc，然后调用**xmit_skb**发包，之后调用**skb_orphan**孤立skb,因为对于guest来说**xmit_skb**之后skb的内容已经是发送的了，最后调用**virtqueue_kick**通知Host。

> virtio_net.c : start_xmit

```
static netdev_tx_t start_xmit(struct sk_buff *skb, struct net_device *dev)
{
    struct virtnet_info *vi = netdev_priv(dev);
    int qnum = skb_get_queue_mapping(skb);
    struct send_queue *sq = &vi->sq[qnum];  //对应的send_queue
    int err;
    struct netdev_queue *txq = netdev_get_tx_queue(dev, qnum);
    bool kick = !skb->xmit_more;

    /* Free up any pending old buffers before queueing new ones. */
/*释放backend处理过的desc*/
    free_old_xmit_skbs(sq);

    /* Try to transmit */
/*发包*/
    err = xmit_skb(sq, skb);

    /* This should not happen! */
    if (unlikely(err)) {
        dev->stats.tx_fifo_errors++;
        if (net_ratelimit())
            dev_warn(&dev->dev,
                 "Unexpected TXQ (%d) queue failure: %d\n", qnum, err);
        dev->stats.tx_dropped++;
        kfree_skb(skb);
        return NETDEV_TX_OK;
    }

    /* Don't wait up for transmitted skbs to be freed. */
/*孤立skb,因为此时对guest来说此skb的内容已发送*/
    skb_orphan(skb);
    nf_reset(skb);

    /* Apparently nice girls don't return TX_BUSY; stop the queue
     * before it gets out of hand.  Naturally, this wastes entries. */
    if (sq->vq->num_free < 2+MAX_SKB_FRAGS) {
        netif_stop_subqueue(dev, qnum);
        if (unlikely(!virtqueue_enable_cb_delayed(sq->vq))) {
            /* More just got used, free them then recheck. */
            free_old_xmit_skbs(sq);
            if (sq->vq->num_free >= 2+MAX_SKB_FRAGS) {
                netif_start_subqueue(dev, qnum);
                virtqueue_disable_cb(sq->vq);
            }
        }
    }

    if (kick || netif_xmit_stopped(txq))
/*通知host端*/
        virtqueue_kick(sq->vq);

    return NETDEV_TX_OK;
}
```

下面我们来分析**xmit_skb**函数是如何发包的，主要就是将数据包头部和数据包填入scatterlist，然后调用**virtqueue_add_outbuf**将sg table 写入desc描述符表，并将head desc信息写入vring.avail。

> virtio_net.c : xmit_skb

```
static int xmit_skb(struct send_queue *sq, struct sk_buff *skb)
{
    struct virtio_net_hdr_mrg_rxbuf *hdr;  //数据包头部
    const unsigned char *dest = ((struct ethhdr *)skb->data)->h_dest;
    struct virtnet_info *vi = sq->vq->vdev->priv;
    unsigned num_sg;
    unsigned hdr_len = vi->hdr_len;       //Packet virtio header size
    bool can_push;

    pr_debug("%s: xmit %p %pM\n", vi->dev->name, skb, dest);

    can_push = vi->any_header_sg &&
        !((unsigned long)skb->data & (__alignof__(*hdr) - 1)) &&
        !skb_header_cloned(skb) && skb_headroom(skb) >= hdr_len;
    /* Even if we can, don't push here yet as this would skew
     * csum_start offset below. */
    if (can_push)
        hdr = (struct virtio_net_hdr_mrg_rxbuf *)(skb->data - hdr_len);
    else
        hdr = skb_vnet_hdr(skb);
/*校验和相关设置*/
    if (skb->ip_summed == CHECKSUM_PARTIAL) {
        hdr->hdr.flags = VIRTIO_NET_HDR_F_NEEDS_CSUM;
        hdr->hdr.csum_start = cpu_to_virtio16(vi->vdev,
```

↑ 90%

```
                        skb_checksum_start_offset(skb));
        hdr->hdr.csum_offset = cpu_to_virtio16(vi->vdev,
                             skb->csum_offset);
    } else {
        hdr->hdr.flags = 0;
        hdr->hdr.csum_offset = hdr->hdr.csum_start = 0;
    }
/*如果是GSO数据包需要重新设置包头*/
    if (skb_is_gso(skb)) {
        hdr->hdr.hdr_len = cpu_to_virtio16(vi->vdev, skb_headlen(skb));
        hdr->hdr.gso_size = cpu_to_virtio16(vi->vdev,
                          skb_shinfo(skb)->gso_size);
        if (skb_shinfo(skb)->gso_type & SKB_GSO_TCPV4)
            hdr->hdr.gso_type = VIRTIO_NET_HDR_GSO_TCPV4;
        else if (skb_shinfo(skb)->gso_type & SKB_GSO_TCPV6)
            hdr->hdr.gso_type = VIRTIO_NET_HDR_GSO_TCPV6;
        else if (skb_shinfo(skb)->gso_type & SKB_GSO_UDP)
            hdr->hdr.gso_type = VIRTIO_NET_HDR_GSO_UDP;
        else
            BUG();
        if (skb_shinfo(skb)->gso_type & SKB_GSO_TCP_ECN)
            hdr->hdr.gso_type |= VIRTIO_NET_HDR_GSO_ECN;
    } else {
        hdr->hdr.gso_type = VIRTIO_NET_HDR_GSO_NONE;
        hdr->hdr.gso_size = hdr->hdr.hdr_len = 0;
    }
/*判断设备是否支持合并buffer*/
    if (vi->mergeable_rx_bufs)
        hdr->num_buffers = 0;
    sg_init_table(sq->sg, MAX_SKB_FRAGS + 2);
    if (can_push) {
        __skb_push(skb, hdr_len);
        num_sg = skb_to_sgvec(skb, sq->sg, 0, skb->len);
        /* Pull header back to avoid skew in tx bytes calculations. */
        __skb_pull(skb, hdr_len);
    } else {
/*数据包头部填入scatterlist*/
        sg_set_buf(sq->sg, hdr, hdr_len);
/*数据包填入scatterlist*/
        num_sg = skb_to_sgvec(skb, sq->sg + 1, 0, skb->len) + 1;
    }
/*sg table 写入desc描述符表，head desc信息写vring.avail*/
    return virtqueue_add_outbuf(sq->vq, sq->sg, num_sg, skb, GFP_ATOMIC);
}
```

可以看到**sg_set_buf**和**skb_to_sgvec**最后都调用了**sg_set_page**，**sg_set_page**中把sk_buffer(逻辑buffer)中的物理块的page信息、offset、len信息放入sg中。

> scatterlist.h : sg_set_buf

```
static inline void sg_set_buf(struct scatterlist *sg, const void *buf,
			      unsigned int buflen)
{
#ifdef CONFIG_DEBUG_SG
	BUG_ON(!virt_addr_valid(buf));
#endif
	sg_set_page(sg, virt_to_page(buf), buflen, offset_in_page(buf));
}
```

skb_to_sgvec

> skbuff.c : skb_to_sgvec

```
int skb_to_sgvec(struct sk_buff *skb, struct scatterlist *sg, int offset, int len)
{
	int nsg = __skb_to_sgvec(skb, sg, offset, len);

	sg_mark_end(&sg[nsg - 1]);

	return nsg;
}
```

> skbuff.c : __skb_to_sgvec

```
static int
__skb_to_sgvec(struct sk_buff *skb, struct scatterlist *sg, int offset, int len)
{
	int start = skb_headlen(skb);
	int i, copy = start - offset;
	struct sk_buff *frag_iter;
	int elt = 0;

	if (copy > 0) {
		if (copy > len)
			copy = len;
		sg_set_buf(sg, skb->data + offset, copy);
		elt++;
		if ((len -= copy) == 0)
			return elt;
```

```
            offset += copy;
        }

        for (i = 0; i < skb_shinfo(skb)->nr_frags; i++) {
            int end;

            WARN_ON(start > offset + len);

            end = start + skb_frag_size(&skb_shinfo(skb)->frags[i]);
            if ((copy = end - offset) > 0) {
                skb_frag_t *frag = &skb_shinfo(skb)->frags[i];

                if (copy > len)
                    copy = len;
                sg_set_page(&sg[elt], skb_frag_page(frag), copy,
                        frag->page_offset+offset-start);
                elt++;
                if (!(len -= copy))
                    return elt;
                offset += copy;
            }
            start = end;
        }

        skb_walk_frags(skb, frag_iter) {
            int end;

            WARN_ON(start > offset + len);

            end = start + frag_iter->len;
            if ((copy = end - offset) > 0) {
                if (copy > len)
                    copy = len;
                elt += __skb_to_sgvec(frag_iter, sg+elt, offset - start,
                           copy);
                if ((len -= copy) == 0)
                    return elt;
                offset += copy;
            }
            start = end;
        }
        BUG_ON(len);
        return elt;
    }
```

↑ 90% terlist.h : sg_set_page

```
static inline void sg_set_page(struct scatterlist *sg, struct page *page,
                               unsigned int len, unsigned int offset)
{
       sg_assign_page(sg, page);
       sg->offset = offset;
       sg->length = len;
}
```

下面看**virtqueue_add_outbuf**如何将sg table 写入desc描述符表。

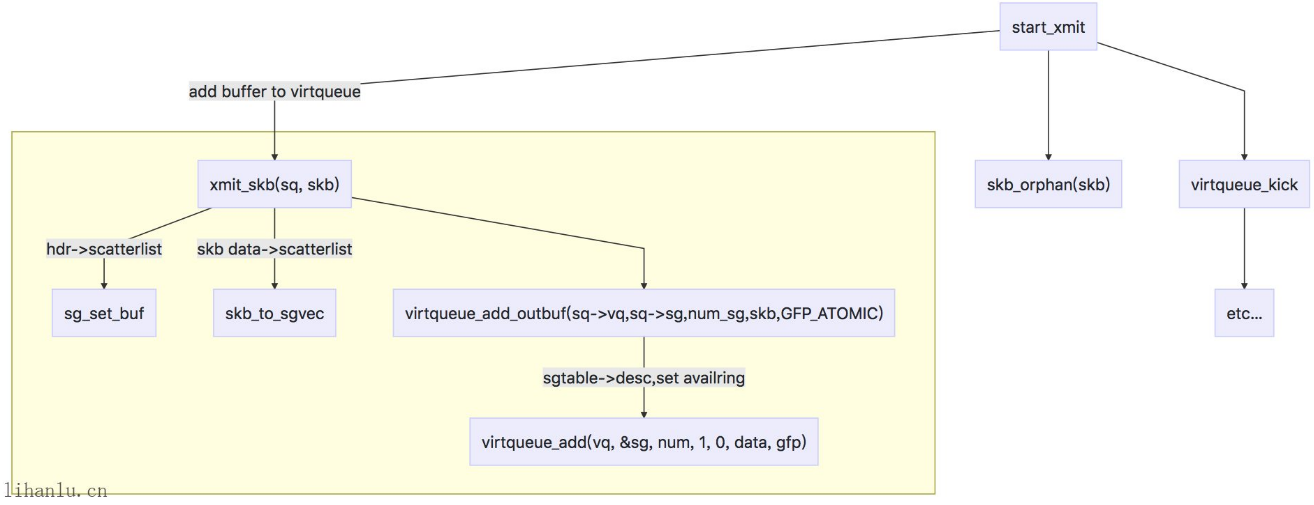
```
int virtqueue_add_outbuf(struct virtqueue *vq,
                         struct scatterlist *sg, unsigned int num,
                         void *data,
                         gfp_t gfp)
{
       return virtqueue_add(vq, &sg, num, 1, 0, data, gfp);
}
```

virtqueue_add首先判断是否支持间接描述符表并且total_sg > 1(否则也不需要使用间接描述符表了)，如果支持indirect把descs_used设置为1，否则设为实际的sg entry总数total_sg。如果主描述符表的空闲表项数小于total_sg就错误返回。然后在for循环中先调用**vring_map_one_sg**函数得到scatterlist中数据的GPA,然后将sg table中entry信息分别对应到描述符表中。标记last desc信息，更新空闲desc数量以及free_head，最后将head desc信息写入vring.avail->ring[]并更新idx。

```
static inline int virtqueue_add(struct virtqueue *_vq,
                                struct scatterlist *sgs[],
                                unsigned int total_sg,
                                unsigned int out_sgs,
                                unsigned int in_sgs,
                                void *data,
                                gfp_t gfp)
{
       struct vring_virtqueue *vq = to_vvq(_vq);
       struct scatterlist *sg;
       struct vring_desc *desc;
       unsigned int i, n, avail, descs_used, uninitialized_var(prev), err_idx;
       int head;
       bool indirect;

       START_USE(vq);
    ……
       head = vq->free_head;
```

```
//判断是否支持间接描述符并且total_sg > 1
    /* If the host supports indirect descriptor tables, and we have multiple
     * buffers, then go indirect. FIXME: tune this threshold */
    if (vq->indirect && total_sg > 1 && vq->vq.num_free)
        desc = alloc_indirect(_vq, total_sg, gfp);
    else
        desc = NULL;

    if (desc) {
        /* Use a single buffer which doesn't continue */
        indirect = true;
        /* Set up rest to use this indirect table. */
        i = 0;
        descs_used = 1;
    } else {
        indirect = false;
        desc = vq->vring.desc;
        i = head;
        descs_used = total_sg;
    }
/*主描述符表的空闲表项数小于total_sg，错误*/
    if (vq->vq.num_free < descs_used) {
        ……
        return -ENOSPC;
    }
/*sg table中entry信息记录到对应的desc表中*/
    for (n = 0; n < out_sgs; n++) {
        for (sg = sgs[n]; sg; sg = sg_next(sg)) {
/*得到总线地址addr，也就是GPA*/
            dma_addr_t addr = vring_map_one_sg(vq, sg, DMA_TO_DEVICE);
            if (vring_mapping_error(vq, addr))
                goto unmap_release;
/*GPA等信息填入desc[]中*/
            desc[i].flags = cpu_to_virtio16(_vq->vdev, VRING_DESC_F_NEXT);
            desc[i].addr = cpu_to_virtio64(_vq->vdev, addr);
            desc[i].len = cpu_to_virtio32(_vq->vdev, sg->length);
            prev = i;
            i = virtio16_to_cpu(_vq->vdev, desc[i].next);
        }
    }
    for (; n < (out_sgs + in_sgs); n++) {
        for (sg = sgs[n]; sg; sg = sg_next(sg)) {
            dma_addr_t addr = vring_map_one_sg(vq, sg, DMA_FROM_DEVICE);
            if (vring_mapping_error(vq, addr))
                goto unmap_release;

            desc[i].flags = cpu_to_virtio16(_vq->vdev, VRING_DESC_F_NEXT | VRING_DESC_F_WRIT
            desc[i].addr = cpu_to_virtio64(_vq->vdev, addr);
            desc[i].len = cpu_to_virtio32(_vq->vdev, sg->length);
            prev = i;
            i = virtio16_to_cpu(_vq->vdev, desc[i].next);
        }
    }
/*标记last desc信息*/
    /* Last one doesn't continue. */
    desc[prev].flags &= cpu_to_virtio16(_vq->vdev, ~VRING_DESC_F_NEXT);

    if (indirect) {
        /* Now that the indirect table is filled in, map it. */
        dma_addr_t addr = vring_map_single(
            vq, desc, total_sg * sizeof(struct vring_desc),
            DMA_TO_DEVICE);
        if (vring_mapping_error(vq, addr))
            goto unmap_release;
        vq->vring.desc[head].flags = cpu_to_virtio16(_vq->vdev, VRING_DESC_F_INDIRECT);
        vq->vring.desc[head].addr = cpu_to_virtio64(_vq->vdev, addr);
        vq->vring.desc[head].len = cpu_to_virtio32(_vq->vdev, total_sg * sizeof(struct vring_desc));
    }
/*更新空闲desc数量*/
    /* We're using some buffers from the free list. */
    vq->vq.num_free -= descs_used;
/*更新free_head指针*/
    /* Update free pointer */
    if (indirect)
        vq->free_head = virtio16_to_cpu(_vq->vdev, vq->vring.desc[head].next);
    else
        vq->free_head = i;

    /* Store token and indirect buffer state. */
    vq->desc_state[head].data = data;
    if (indirect)
        vq->desc_state[head].indir_desc = desc;
    vq->vq.num_free -= descs_used;
/*desc链中第一个desc放入vring_avail中的ring[]*/
    /* Put entry in available array (but don't update avail->idx until they
     * do sync). */
    avail = virtio16_to_cpu(_vq->vdev, vq->vring.avail->idx) & (vq->vring.num - 1);
    vq->vring.avail->ring[avail] = cpu_to_virtio16(_vq->vdev, head);

    /* Descriptors and available array need to be set before we expose the
     * new available array entries. */
    virtio_wmb(vq->weak_barriers);
/*更新vring_avail中idx*/
    vq->vring.avail->idx = cpu_to_virtio16(_vq->vdev, virtio16_to_cpu(_vq->vdev, vq->vring.avail->i
    vq->num_added++;
……
}
```

回到**xmit_skb**之后调用**virtqueue_kick**通知Host(具体的通知过程会另写一篇文章分析),上述过程总结如下图。



Backend

Backend收到通知后会执行**virtio_queue_host_notifier_read**函数，我们从这里开始分析

```
void virtio_queue_host_notifier_read(EventNotifier *n)
{
    VirtQueue *vq = container_of(n, VirtQueue, host_notifier);
    if (event_notifier_test_and_clear(n)) {
        virtio_queue_notify_vq(vq);
    }
}
```

virtio_queue_notify_vq中调用了**virtio_net_device_realize**中**virtio_net_add_queue**时绑定的处理函数**handle_output**，该函数根据不同的设备有不同的实现，以网卡为例看看创建VirtQueue的时候给绑定的是哪个函数。

```
static void virtio_queue_notify_vq(VirtQueue *vq)
{
    if (vq->vring.desc && vq->handle_output) {
        VirtIODevice *vdev = vq->vdev;

        if (unlikely(vdev->broken)) {
            return;
        }
```

90%

```
        trace_virtio_queue_notify(vdev, vq - vdev->vq, vq);
        vq->handle_output(vdev, vq);
    }
}
```

可以看到这里给rx_vq绑定的是**virtio_net_handle_rx**，tx_vq绑定的**virtio_net_handle_tx_timer**或**virtio_net_handle_tx_bh**，下面以**virtio_net_handle_tx_bh**为例看一下。

```
static void virtio_net_add_queue(VirtIONet *n, int index)
{
    VirtIODevice *vdev = VIRTIO_DEVICE(n);

    n->vqs[index].rx_vq = virtio_add_queue(vdev, n->net_conf.rx_queue_size,
                                           virtio_net_handle_rx);

    if (n->net_conf.tx && !strcmp(n->net_conf.tx, "timer")) {
        n->vqs[index].tx_vq =
            virtio_add_queue(vdev, n->net_conf.tx_queue_size,
                             virtio_net_handle_tx_timer);
        n->vqs[index].tx_timer = timer_new_ns(QEMU_CLOCK_VIRTUAL,
                                              virtio_net_tx_timer,
                                              &n->vqs[index]);
    } else {
        n->vqs[index].tx_vq =
            virtio_add_queue(vdev, n->net_conf.tx_queue_size,
                             virtio_net_handle_tx_bh);
        n->vqs[index].tx_bh = qemu_bh_new(virtio_net_tx_bh, &n->vqs[index]);
    }

    n->vqs[index].tx_waiting = 0;
    n->vqs[index].n = n;
}
```

virtio_net_handle_tx_bh中调用了**qemu_bh_schedule**(q->tx_bh)运行参数指定的函数，而参数tx_bh在**virtio_net_add_queue**中调用**qemu_bh_new**(virtio_net_tx_bh, &n->vqs[index])绑定为**virtio_net_tx_bh**

```
static void virtio_net_handle_tx_bh(VirtIODevice *vdev, VirtQueue *vq)
{
    VirtIONet *n = VIRTIO_NET(vdev);
    VirtIONetQueue *q = &n->vqs[vq2q(virtio_get_queue_index(vq))];

    if (unlikely((n->status & VIRTIO_NET_S_LINK_UP) == 0)) {
        virtio_net_drop_tx_queue_data(vdev, vq);
        return;
    }

    if (unlikely(q->tx_waiting)) {
        return;
    }
    q->tx_waiting = 1;
    /* This happens when device was stopped but VCPU wasn't. */
    if (!vdev->vm_running) {
        return;
    }
    virtio_queue_set_notification(vq, 0);
    qemu_bh_schedule(q->tx_bh);
}
```

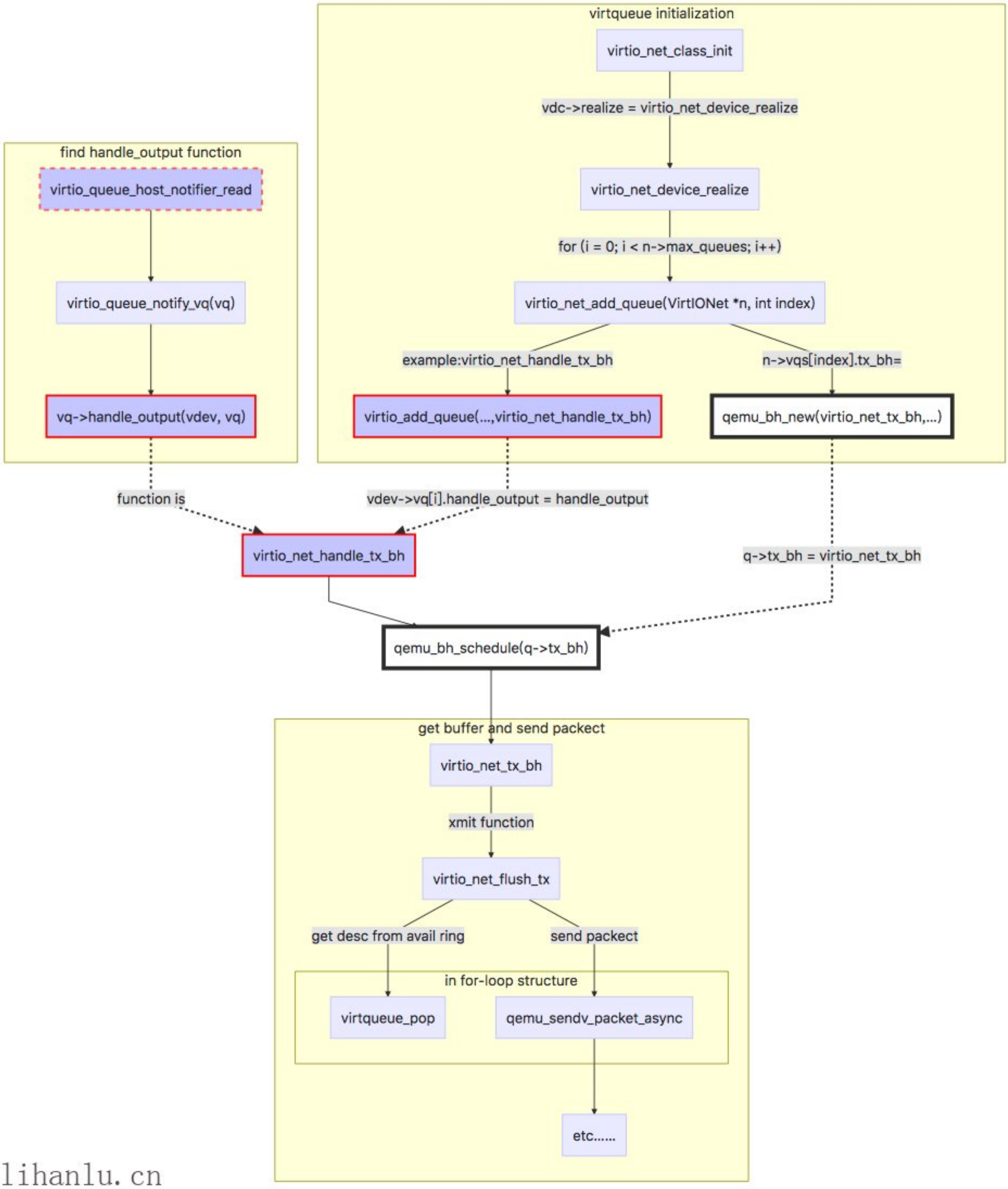
virtio_net_tx_bh函数调用发送函数**virtio_net_flush_tx**

```
static void virtio_net_tx_bh(void *opaque)
{
    ……
    ret = virtio_net_flush_tx(q);
    ……
}
```

virtio_net_flush_tx函数调用**virtqueue_pop**从avail ring中取数据buffer的head desc，然后调用**qemu_sendv_packet_async**发送packet，qemu_sendv_packet_async函数本文不做详细的分析。

```
static int32_t virtio_net_flush_tx(VirtIONetQueue *q)
{
    VirtIONet *n = q->n;
    VirtIODevice *vdev = VIRTIO_DEVICE(n);
    VirtQueueElement *elem;
    int32_t num_packets = 0;
    int queue_index = vq2q(virtio_get_queue_index(q->tx_vq));
    if (!(vdev->status & VIRTIO_CONFIG_S_DRIVER_OK)) {
        return num_packets;
    }

    if (q->async_tx.elem) {
        virtio_queue_set_notification(q->tx_vq, 0);
        return num_packets;
    }

    for (;;) {
        ssize_t ret;
        unsigned int out_num;
        struct iovec sg[VIRTQUEUE_MAX_SIZE], sg2[VIRTQUEUE_MAX_SIZE + 1], *out_sg;
        struct virtio_net_hdr_mrg_rxbuf mhdr;
//从avail ring中取数据buffer的head desc
        elem = virtqueue_pop(q->tx_vq, sizeof(VirtQueueElement));
        if (!elem) {
            break;
        }

        out_num = elem->out_num;
        out_sg = elem->out_sg;
        if (out_num < 1) {
            virtio_error(vdev, "virtio-net header not in first element");
            virtqueue_detach_element(q->tx_vq, elem, 0);
            g_free(elem);
            return -EINVAL;
        }

        if (n->has_vnet_hdr) {
            if (iov_to_buf(out_sg, out_num, 0, &mhdr, n->guest_hdr_len) <
                n->guest_hdr_len) {
                virtio_error(vdev, "virtio-net header incorrect");
                virtqueue_detach_element(q->tx_vq, elem, 0);
                g_free(elem);
                return -EINVAL;
            }
            if (n->needs_vnet_hdr_swap) {
                virtio_net_hdr_swap(vdev, (void *) &mhdr);
                sg2[0].iov_base = &mhdr;
                sg2[0].iov_len = n->guest_hdr_len;
                out_num = iov_copy(&sg2[1], ARRAY_SIZE(sg2) - 1,
                                   out_sg, out_num,
                                   n->guest_hdr_len, -1);
                if (out_num == VIRTQUEUE_MAX_SIZE) {
                    goto drop;
                }
                out_num += 1;
                out_sg = sg2;
            }
        }
        /*
         * If host wants to see the guest header as is, we can
         * pass it on unchanged. Otherwise, copy just the parts
         * that host is interested in.
         */
    assert(n->host_hdr_len <= n->guest_hdr_len);
    if (n->host_hdr_len != n->guest_hdr_len) {
      unsigned sg_num = iov_copy(sg, ARRAY_SIZE(sg),
                   out_sg, out_num,
                   0, n->host_hdr_len);
      sg_num += iov_copy(sg + sg_num, ARRAY_SIZE(sg) - sg_num,
              out_sg, out_num,
              n->guest_hdr_len, -1);
      out_num = sg_num;
      out_sg = sg;
    }
/*调用qemu发包函数，通过qemu nic发送*/
    ret = qemu_sendv_packet_async(qemu_get_subqueue(n->nic, queue_index),
                   out_sg, out_num, virtio_net_tx_complete);
    if (ret == 0) {
      virtio_queue_set_notification(q->tx_vq, 0);
      q->async_tx.elem = elem;
      return -EBUSY;
    }

drop:
/*取消内存映射，更新used vring信息*/
    virtqueue_push(q->tx_vq, elem, 0);
    virtio_notify(vdev, q->tx_vq);
    g_free(elem);

    if (++num_packets >= n->tx_burst) {
      break;
    }
  }
  return num_packets;
}
```

Host收到通知后的发包流程总结如下图。


virtqueue initialization
virtio_net_class_init
vdc->realize = virtio_net_device_realize
virtio_net_device_realize
for (i = 0; i < n->max_queues; i++)
virtio_net_add_queue(VirtIONet *n, int index)
example:virtio_net_handle_tx_bh
n->vqs[index].tx_bh=
virtio_add_queue(...,virtio_net_handle_tx_bh)
qemu_bh_new(virtio_net_tx_bh,...)
find handle_output function
virtio_queue_host_notifier_read
virtio_queue_notify_vq(vq)
vq->handle_output(vdev, vq)
function is
vdev->vq[i].handle_output = handle_output
virtio_net_handle_tx_bh
q->tx_bh = virtio_net_tx_bh
qemu_bh_schedule(q->tx_bh)
get buffer and send packect
virtio_net_tx_bh
xmit function
virtio_net_flush_tx
get desc from avail ring
send packect
in for-loop structure
virtqueue_pop
qemu_sendv_packet_async
etc......
lihanlu. cn

联系我

你可以直接在下方留言，也可以✉E-Mail联系我。

打赏

本文作者： Lauren

↑ 90%

virtio

Virtio原理简介

virtio前端通知机制分析

昵称 邮箱 网址(http://)

Just go go

提交

1 评论

Anonymous Chrome 80.0.3987.132 Windows 10.0
2020-03-19 回复

先读一下,有问题再请教

Powered By Valine
v1.4.4