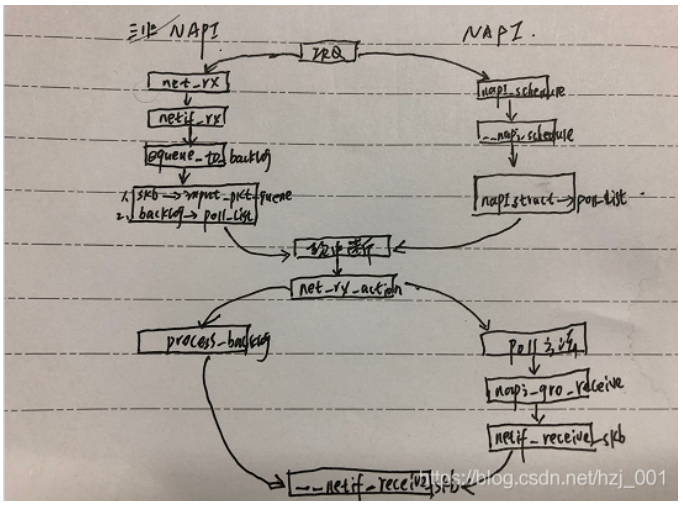
网络收包流程-报文从网卡驱动到网络层（或者网桥)的流程（非NAPI、NAPI）(一)

原创 菜鸟别浪 于 2019-08-26 20:23:05 发布 693 收藏 8 版权
分类专栏： linux 网络 tcp/ip 文章标签： 网络收报中断处理 NAPI process_backlog poll_list

linux 2 订阅 81 篇文章 订阅专栏
网络 0 订阅 25 篇文章 订阅专栏
tcp/ip 2 订阅 13 篇文章 订阅专栏

1.上图(网上的没我这个详细，哈哈)：



2.具体说明NAPI和非NAPI的区别：

(1) 支持NAPI的网卡驱动必须提供轮询方法poll()。

(2) 非NAPI的内核接口为netif_rx()，NAPI的内核接口为napi_schedule()。

(3) 非NAPI使用共享的CPU队列softnet_data->input_pkt_queue，NAPI使用设备内存(或者设备驱动程序的接收环)。

net_rx函数的作用：

a.为skb分配内存

b.从网卡copy内存到skb空间

netif_rx的作用：

a.将net_rx从网卡获取的skb，放到当前cpu的softnet_data公共等待队列input_pkt_queue，即enqueue_to_backlog-->__skb_queue_tail(&sd->input_pkt_queue, skb);

b.将process_backlog加入到当前cpu的softnet_data的poll_list中，即enqueue_to_backlog-->____napi_schedule->list_add_tail(&napi->poll_list, &sd->poll_list);

c.触发软中断，即__raise_softirq_irqoff(NET_RX_SOFTIRQ);

napi_schedule的作用：

a.将特定于硬件的poll_list加入到当前cpu的softnet_data的poll_list（container_of函数即可通过poll_list获得对应的napi_struct结构体，从而可以获得poll函数）

b.触发软中断，即__raise_softirq_irqoff(NET_RX_SOFTIRQ);

注意：NAPI方式的skb直接从设备内存获得，不用自己维护内存

3.中断方式与NAPI的结合：

NAPI 的核心在于：在一个繁忙网络，每次有网络数据包到达时，不需要都引发中断，因为高频率的中断可能会影响系统的整体效率，假象一个场景，我们此时使用标准的 100M 网卡，可能实际达到的接收速率为 80MBits/s，而此时数据包平均长度为 1500Bytes，则每秒产生的中断数目为：

80M bits/s / (8 * 1500 Byte) = (80 * 1024 * 1024) bit/s / (8 * 1500 Byte) = 6990 个中断 /s

每秒6990个中断，对于系统是个很大的压力，此时其实可以转为使用轮询 (polling) 来处理，而不是中断；但轮询在网络流量较小的时没有效率，因此低流量时，基于中断的方式则比较合适，这就是 NAPI 出现的原因，在低流量时候使用中断接收数据包，而在高流量时候则使用基于轮询的方式接收。

4.NAPI方式的优势：NAPI 适合处理高速率数据包的处理，而带来的好处则是：

1.中断缓和 (Interrupt mitigation)，由上面的例子可以看到，在高流量下，网卡产生的中断

可能达到每秒几千次，而如果每次中断都需要系统来处理，是一个很大的压力，而 NAPI 使用轮询时是禁止了网卡的接收中断的，这样会减小系统处理中断的压力。

2.数据包节流 (Packet throttling)，NAPI 之前的 Linux NIC 驱动总在接收到数据包之后产生一个 IRQ，接着在中断服务例程里将这个 skb 加入本地的 softnet_data，然后触发本地 NET_RX_SOFTIRQ 软中断后续处理。如果包速过高，因为 IRQ 的优先级高于 SoftIRQ，导致系统的大部分资源都在响应中断，但 softnet 的队列大小有限，接收到的超额数据包也只能丢掉，所以这时这个模型是在用宝贵的系统资源做无用功。而 NAPI 则在这样的情况下，直接把包丢掉，不会继续将需要丢掉的数据包扔给内核去处理，这样，网卡将需要丢掉的数据包尽可能的早丢弃掉，内核将不可见需要丢掉的数据包，这样也减少了内核的压力。

5.NAPI的实现原理：

假定某个网络适配器此前没有分组到达，但从现在开始，分组将以高频率频繁到达。这就是NAPI设备的情况，如下所述。

(1) 第一个分组将导致网络适配器发出IRQ。为防止进一步的分组导致发出更多的IRQ，驱动程序会在**硬中断处理流程中**关闭该适配器的Rx IRQ。并将该适配器放置到一个轮询表上。

(2) 只要适配器上还有分组需要处理，内核就一直对轮询表上的设备进行轮询。

(3) 重新启用Rx中断。

如果在新的分组到达时，旧的分组仍然处于处理过程中，工作不会因额外的中断而减速。虽然对设备驱动程序（和一般意义上的内核代码）来说轮询通常是一个很差的方法，但在这里该方法没有什么不利之处：在没有分组还需要处理时，将停止轮询，设备将回复到通常的IRQ驱动的运行方式。在没有中断支持的情况下，轮询空的接收队列将不必要地浪费时间，但NAPI并非如此。

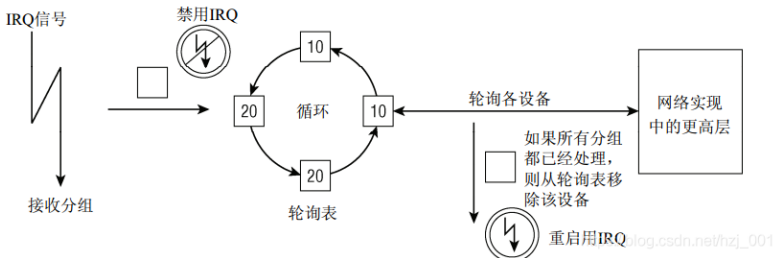
NAPI的另一个优点是可以高效地丢弃分组。如果内核确信因为有很多其他工作需要处理，而导致无法处理任何新的分组，那么网络适配器可以直接丢弃分组，无须复制到内核。

只有设备满足如下两个条件时，才能实现NAPI方法。

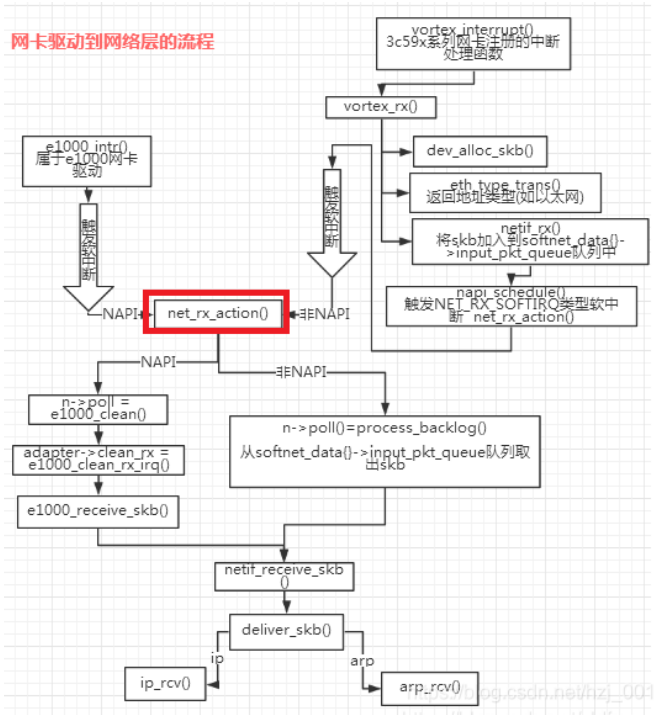
(1) 设备必须能够保留多个接收的分组，例如保存到DMA环形缓冲区中。下文将该缓冲区称为Rx缓冲区。

(2) 该设备必须能够禁用用于分组接收的IRQ。而且，发送分组或其他可能通过IRQ进行的操作，都仍然必须是启用的。

如果系统中有多个设备，会怎么样呢？这是通过循环轮询各个设备来解决的。



6.具体网卡的举例说明：



显示推荐内容

参与评论 您还未登录，请先 登录 后发表或查看评论