Linux网络栈

原创转载请注明出处：https://www.cnblogs.com/agilestyle/p/11394930.html

OSI模型

OSI 模型把网络互联的框架分为应用层、表示层、会话层、传输层、网络层、数据链路层以及物理层等七层，每个层负责不同的功能。其中，

应用层，负责为应用程序提供统一的接口。
表示层，负责把数据转换成兼容接收系统的格式。
会话层，负责维护计算机之间的通信连接。
传输层，负责为数据加上传输表头，形成数据包。
网络层，负责数据的路由和转发。
数据链路层，负责MAC寻址、错误侦测和改错。
物理层，负责在物理网络中传输数据帧。

TCP/IP 模型

TCP/IP 模型把网络互联的框架分为应用层、传输层、网络层、网络接口层等四层，其中，

应用层，负责向用户提供一组应用程序，比如 HTTP、FTP、DNS 等。
传输层，负责端到端的通信，比如 TCP、UDP 等。
网络层，负责网络包的封装、寻址和路由，比如 IP、ICMP 等。
网络接口层，负责网络包在物理网络中的传输，比如 MAC 寻址、错误侦测以及通过网卡传输网络帧等。

OSI 模型 与 TCP/IP 模型 的关系

随笔档案

2020年5月(8)

2020年4月(28)

2020年3月(2)

2020年2月(13)

2020年1月(24)

2019年12月(9)

2019年11月(11)

2019年10月(15)

2019年9月(80)

2019年8月(45)

阅读排行榜

1. Docker安装ES(1949)

2. ConcurrentLinkedQueue和LinkedBlockingQueue区别(1518)

3. Java Thread之start和run方法的区别(1515)

4. Docker安装Kibana(955)

5. Java文件拷贝方式(952)

推荐排行榜

1. Docker安装ES(1)

2. 高可用之降级、限流、拒绝服务(1)

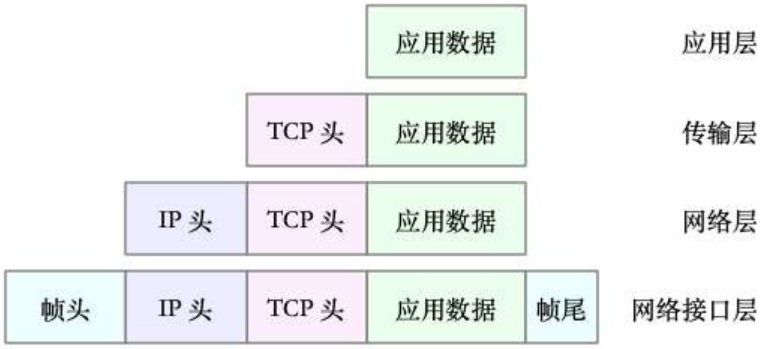
3. 内存工作原理(1)

Linux网络栈

有了 TCP/IP 模型后，在进行网络传输时，数据包就会按照协议栈，对上一层发来的数据进行逐层处理；然后封装上该层的协议头，再发送给下一层。

当然，网络包在每一层的处理逻辑，都取决于各层采用的网络协议。比如在应用层，一个提供 REST API 的应用，可以使用 HTTP 协议，把它需要传输的 JSON 数据封装到 HTTP 协议中，然后向下传递给 TCP 层。

而封装做的事情就很简单了，只是在原来的负载前后，增加固定格式的元数据，原始的负载数据并不会被修改。比如，以通过 TCP 协议通信的网络包为例，通过下面这张图，可以看到，应用程序数据在每个层的封装格式。



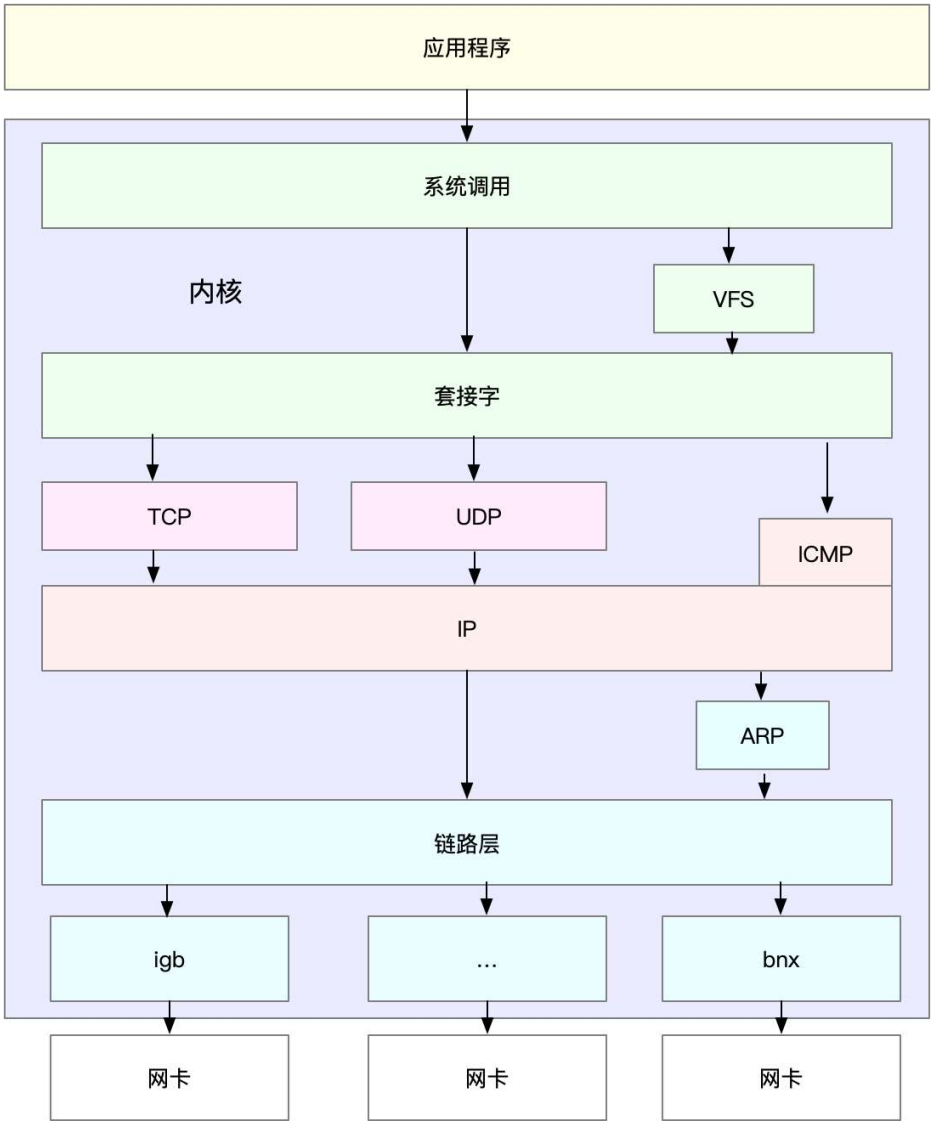
其中：

- 传输层在应用程序数据前面增加了 TCP 头；
- 网络层在 TCP 数据包前增加了 IP 头；
- 而网络接口层，又在 IP 数据包前后分别增加了帧头和帧尾。

这些新增的头部和尾部，都按照特定的协议格式填充，这些新增的头部和尾部，增加了网络包的大小，但物理链路中并不能传输任意大小的数据包。网络接口配置的最大传输单元（MTU），就规定了最大的 IP 包大小。在最常用的以太网中，MTU 默认值是 1500（这也是 Linux 的默认值）。

一旦网络包超过 MTU 的大小，就会在网络层分片，以保证分片后的 IP 包不大于MTU 值。显然，MTU 越大，需要的分包也就越少，网络吞吐能力就越好。

理解了 TCP/IP 网络模型和网络包的封装原理后，很容易能想到，Linux 内核中的网络栈，其实也类似于 TCP/IP 的四层结构。如下图所示，就是 Linux 通用 IP 网络栈的示意图：



从上到下来看这个网络栈，可以发现，

- 最上层的应用程序，需要通过系统调用，来跟套接字接口进行交互；
- 套接字的下面，就是前面提到的传输层、网络层和网络接口层；
- 最底层，则是网卡驱动程序以及物理网卡设备。

网卡是发送和接收网络包的基本设备。在系统启动过程中，网卡通过内核中的网卡驱动程序注册到系统中。而在网络收发过程中，内核通过中断跟网卡进行交互。

再结合前面提到的 Linux 网络栈，可以看出，网络包的处理非常复杂。所以，网卡硬中断只处理最核心的网卡数据读取或发送，而协议栈中的大部分逻辑，都会放到软中断中处理。

Linux网络收发流程

了解了 Linux 网络栈后，再来看看， Linux 到底是怎么收发网络包的。

注意，以下内容都以物理网卡为例。事实上，Linux 还支持众多的虚拟网络设备，而它们的网络收发流程会有一些差别。

网络包的接收流程

先来看网络包的接收流程。

当一个网络帧到达网卡后，网卡会通过 DMA 方式，把这个网络包放到收包队列中；然后通过硬中断，告诉中断处理程序已经收到了网络包。

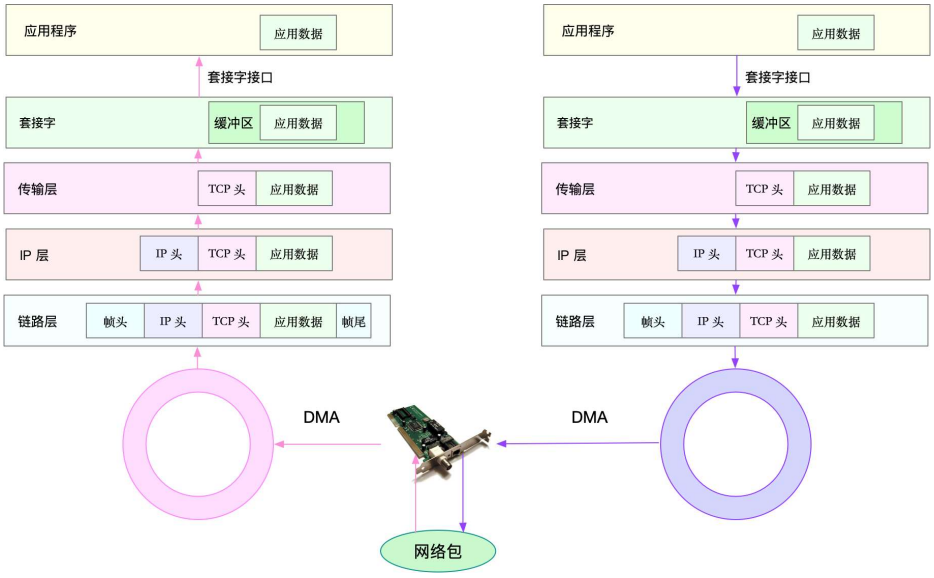
接着，网卡中断处理程序会为网络帧分配内核数据结构（sk_buff），并将其拷贝到 sk_buff 缓冲区中；然后再通过软中断，通知内核收到了新的网络帧。

接下来，内核协议栈从缓冲区中取出网络帧，并通过网络协议栈，从下到上逐层处理这个网络帧。比如，

- 在链路层检查报文的合法性，找出上层协议的类型（比如 IPv4 还是 IPv6），再去掉帧头、帧尾，然后交给网络层。
- 网络层取出 IP 头，判断网络包下一步的走向，比如是交给上层处理还是转发。当网络层确认这个包是要发送到本机后，就会取出上层协议的类型（比如 TCP 还是 UDP），去掉 IP 头，再交给传输层处理。
- 传输层取出 TCP 头或者 UDP 头后，根据 <源 IP、源端口、目的 IP、目的端口> 四元组作为标识，找出对应的 Socket，并把数据拷贝到Socket 的接收缓存中。

最后，应用程序就可以使用 Socket 接口，读取到新接收到的数据了。

为了更清晰表示这个流程，参考下图，这张图的左半部分表示接收流程，而图中的粉色箭头则表示网络包的处理路径。



网络包的发送流程

了解网络包的接收流程后，就很容易理解网络包的发送流程。网络包的发送流程就是上图的右半部分，很容易发现，网络包的发送方向，正好跟接收方向相反。

1. 首先，应用程序调用 Socket API（比如 sendmsg）发送网络包。
2. 由于这是一个系统调用，所以会陷入到内核态的套接字层中。套接字层会把数据包放到 Socket 发送缓冲区中。
3. 接下来，网络协议栈从 Socket 发送缓冲区中，取出数据包；再按照 TCP/IP 栈，从上到下逐层处理。比如，传输层和网络层，分别为其增加TCP 头和 IP 头，执行路由查找确认下一跳的 IP，并按照 MTU 大小进行分片。
4. 分片后的网络包，再送到网络接口层，进行物理地址寻址，以找到下一跳的 MAC 地址。然后添加帧头和帧尾，放到发包队列中。这一切完成后，会有软中断通知驱动程序：发包队列中有新的网络帧需要发送。
5. 最后，驱动程序通过 DMA ，从发包队列中读出网络帧，并通过物理网卡把它发送出去。

Reference

https://time.geekbang.org/column/article/80898 （强烈推荐读者购买此专栏，都是干货价值一个亿）

强者自救 圣者渡人

标签： Network

« 上一篇： ConcurrentLinkedQueue和LinkedBlockingQueue区别
» 下一篇： 公平锁 / 非公平锁

posted @ 2019-08-22 16:07 李白与酒 阅读(355) 评论(0) 编辑 收藏