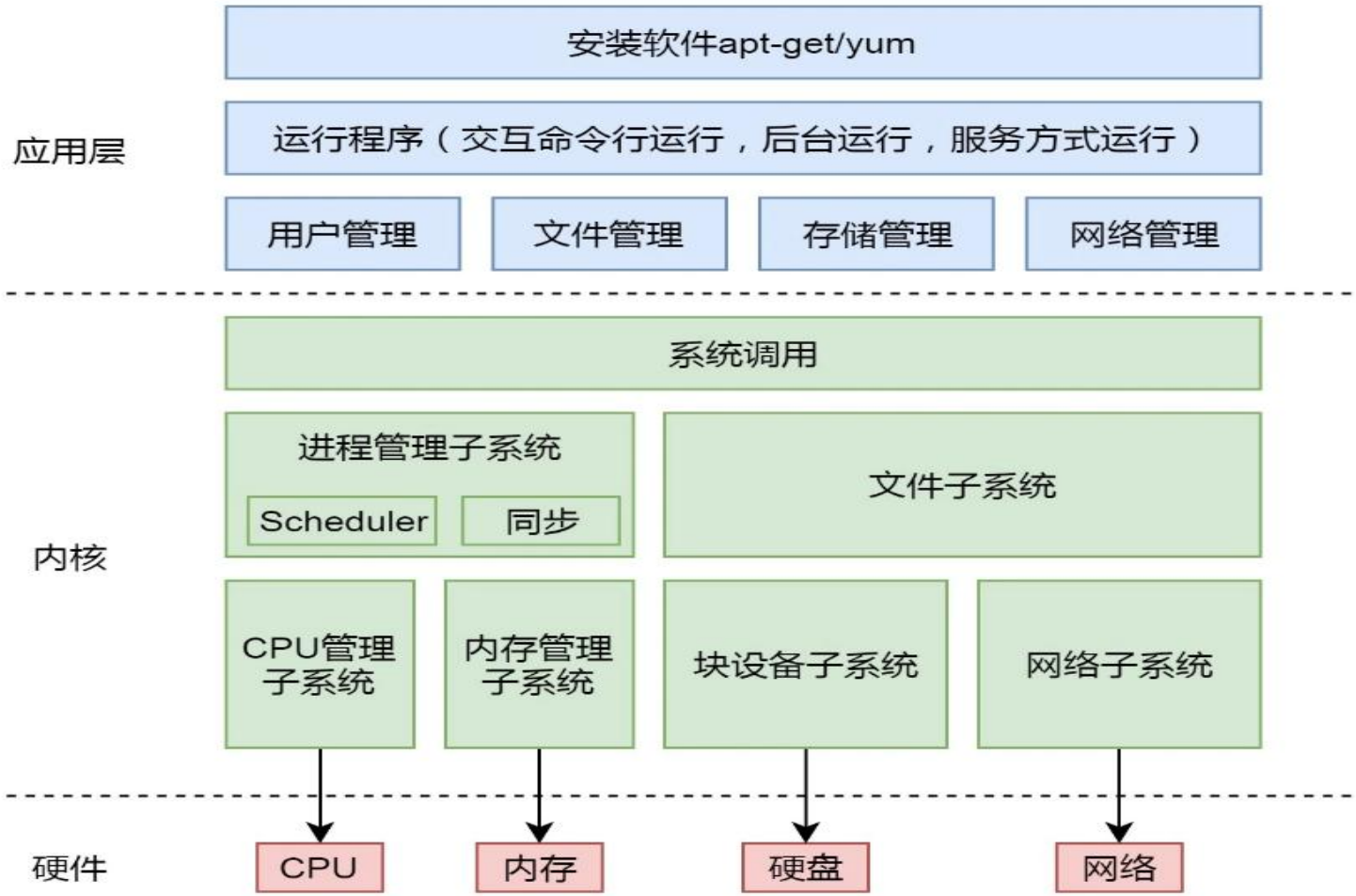
操作系统-网络子系统

By 公众号@极客重生

大师兄Alex

操作系统宏观视角



Linux操作系统

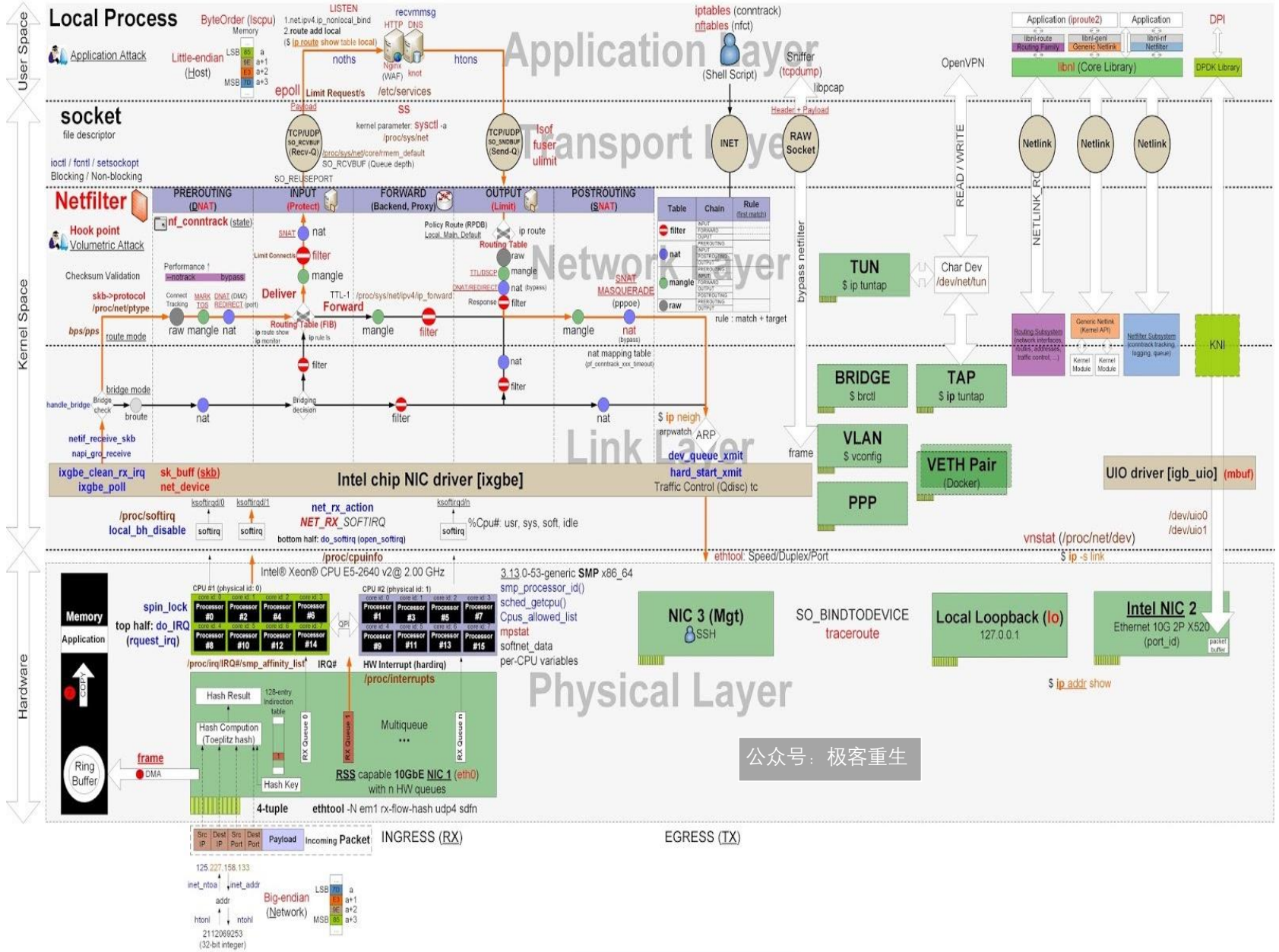
操作系统各大子系统

云计算开发	IaaS：虚拟化，计算，网络，存储 PaaS：虚拟化，容器，计算，网络，存储
Linux后台服务器开发	高性能，高并发
Linux嵌入式开发	内核裁剪，各种硬件接口适配 ，性能优化，手机，Iot
Linux SRE方向	运维，解决Linux稳定性问题。

- 调度子系统
- 内存子系统
- IO子系统
- 网络子系统
- 驱动（设备管理）子系统

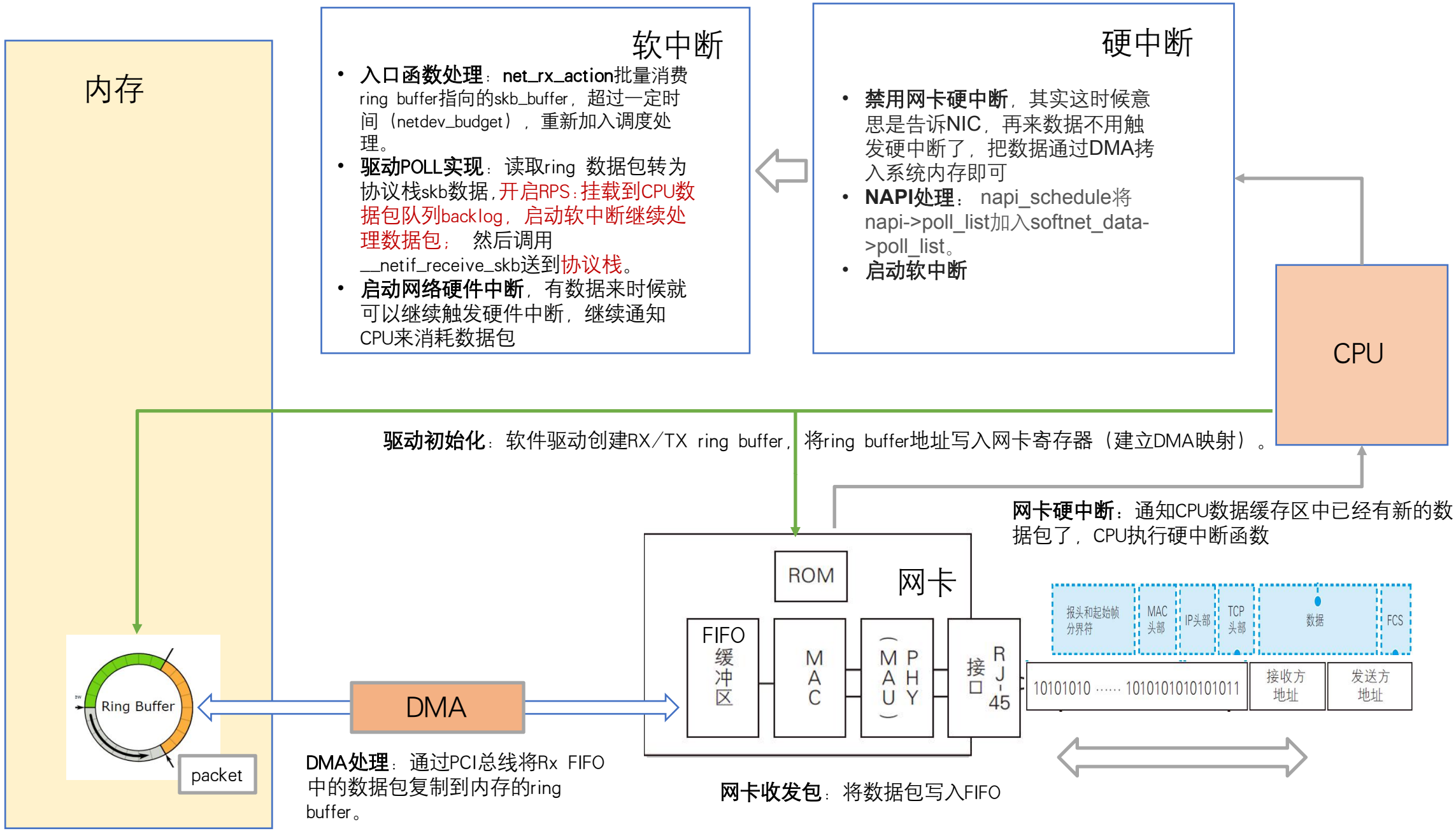
网络子系统

- 网卡驱动
- 数据包管理
- 网络驱动
- 邻居系统
- netfilter框架
- 路由模块
- 网络协议栈
- 数据处理路径

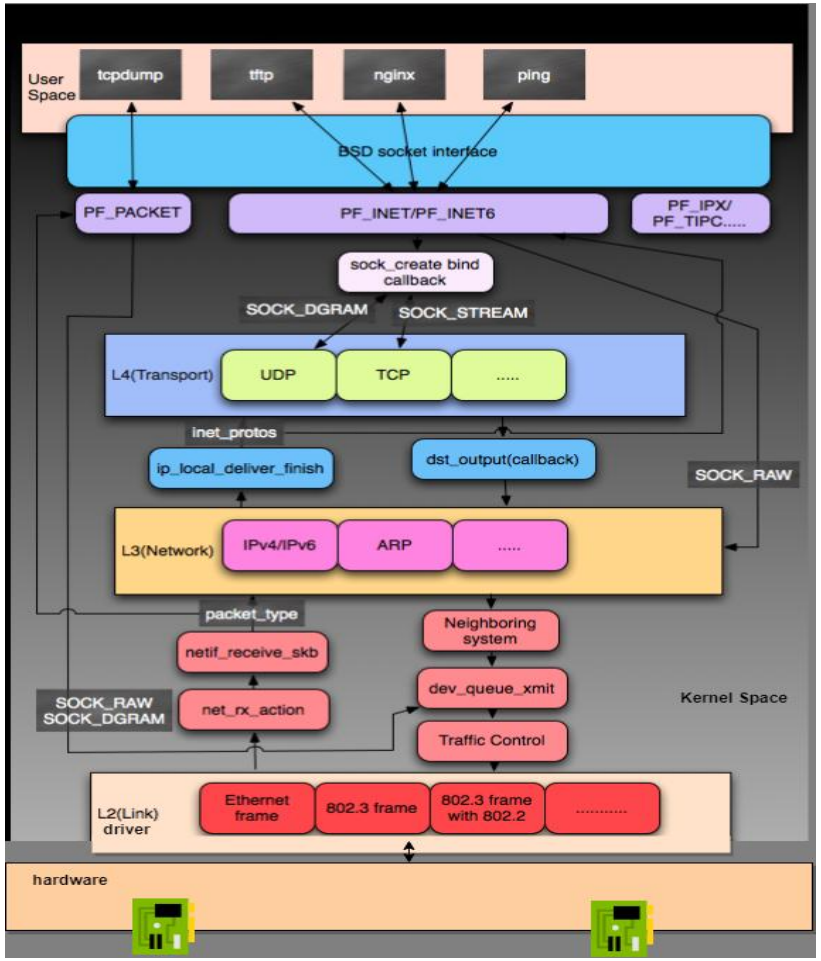


Linux网络全景图

网络子系统—网卡驱动


内存
软中断
• 入口函数处理：net_rx_action批量消费ring buffer指向的skb_buffer，超过一定时间（netdev_budget），重新加入调度处理。
• 驱动POLL实现：读取ring 数据包转为协议栈skb数据，开启RPS：挂载到CPU数据包队列backlog，启动软中断继续处理数据包； 然后调用__netif_receive_skb送到协议栈。
• 启动网络硬件中断，有数据来时候就可以继续触发硬件中断，继续通知CPU来消耗数据包
硬中断
• 禁用网卡硬中断，其实这时候意思是告诉NIC，再来数据不用触发硬中断了，把数据通过DMA拷入系统内存即可
• NAPI处理： napi_schedule将napi->poll_list加入softnet_data->poll_list。
• 启动软中断
CPU
驱动初始化：软件驱动创建RX/TX ring buffer，将ring buffer地址写入网卡寄存器（建立DMA映射）。
网卡硬中断：通知CPU数据缓存区中已经有新的数据包了，CPU执行硬中断函数
ROM
网卡
FIFO缓冲区
MAC
MAU
PHY
接口
RJ-45
报头和起始帧分界符
MAC头部
IP头部
TCP头部
数据
FCS
10101010 …… 1010101010101011
接收方地址
发送方地址
Ring Buffer
packet
DMA
DMA处理：通过PCI总线将Rx FIFO中的数据包复制到内存的ring buffer。
网卡收发包：将数据包写入FIFO

网络子系统—网络协议栈链路层—L3层

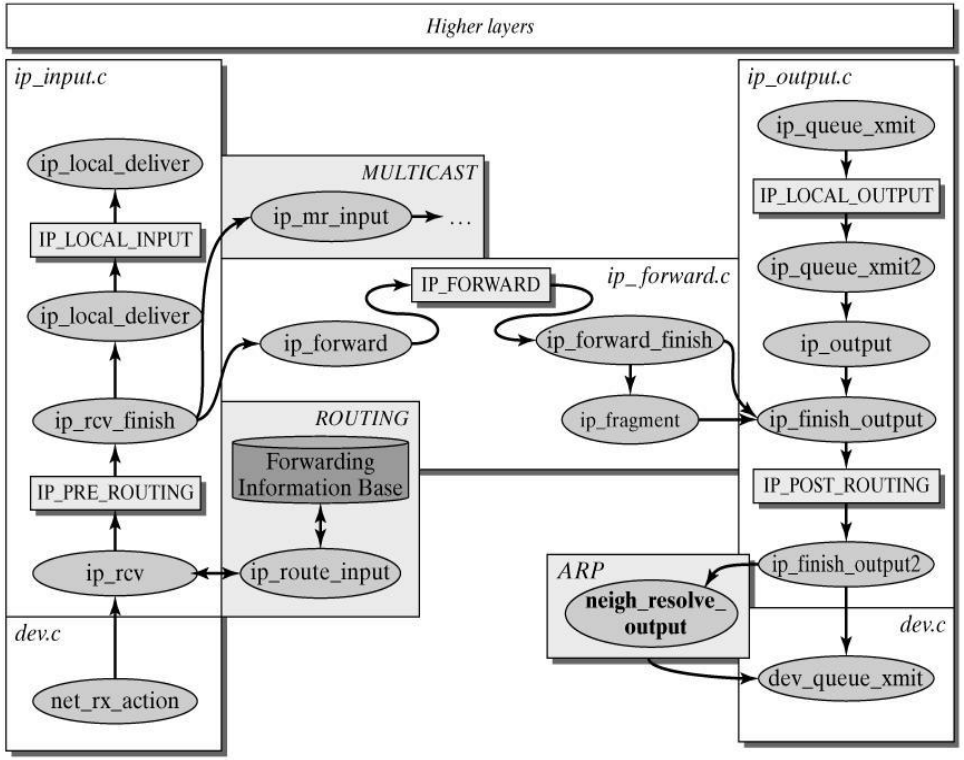


LinuxTCP/IP 网络协议栈

IP层处理

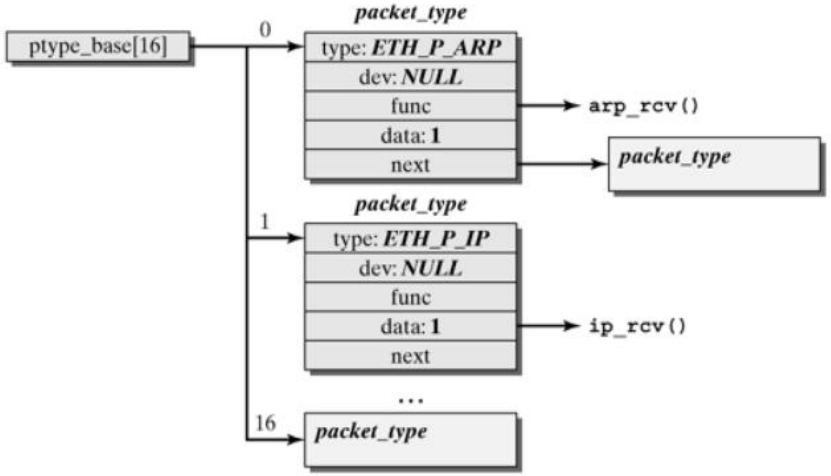
IP报文处理

- 报文分片和重组
- IP协议字段处理，IP选项，Qos，TTL，校验等处理
- 报文接收（解封装）和发送(IP协议封装，提供给上层接口）
- 组播，ICMP协议处理等

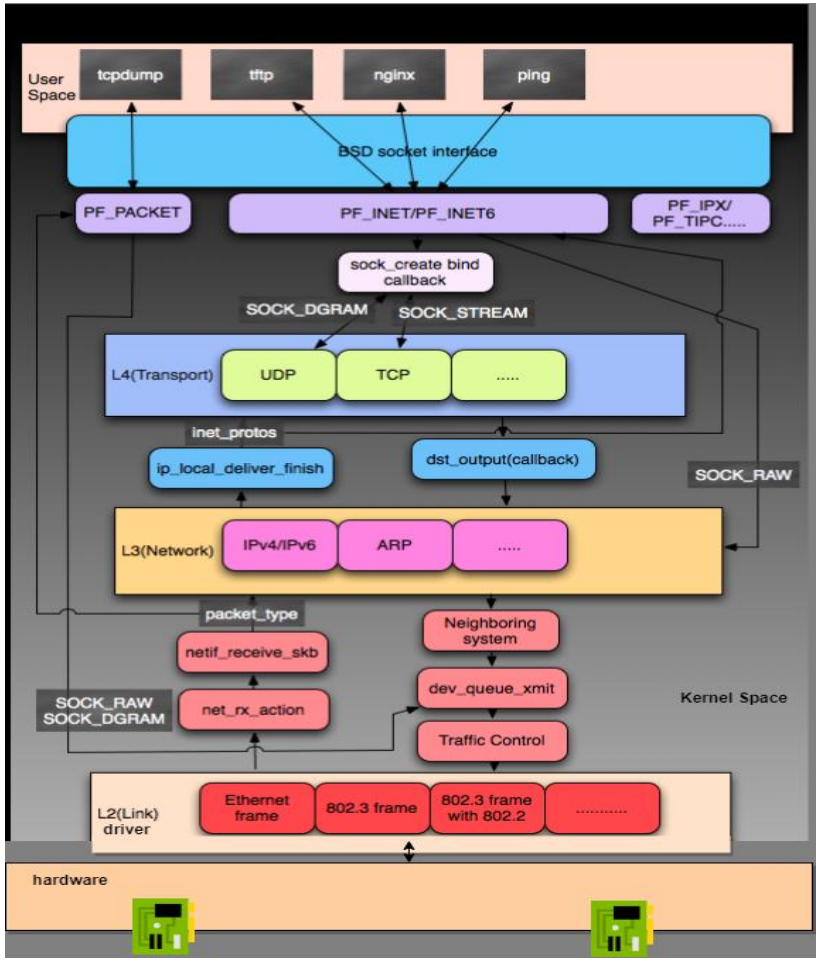


链路报文分发处理模块

网络层的协议（IP、ARP、IPv6、IPX）通过这个接口被添加到Linux网络架构中， 根据报文的以太网头的ether type选择对应协议处理函数



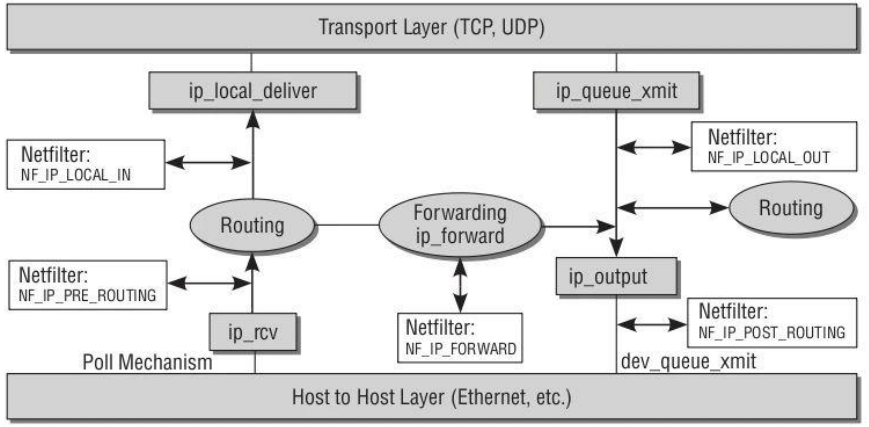
网络子系统—网络协议栈L3处理



LinuxTCP/IP 网络协议栈

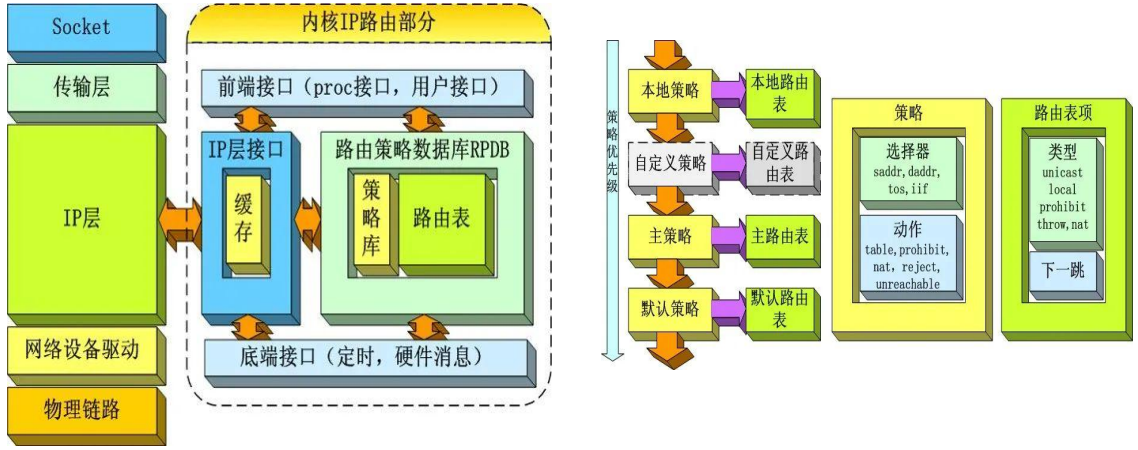
Netfilter框架

防火墙，报文过滤系统，报文HOOK处理框架



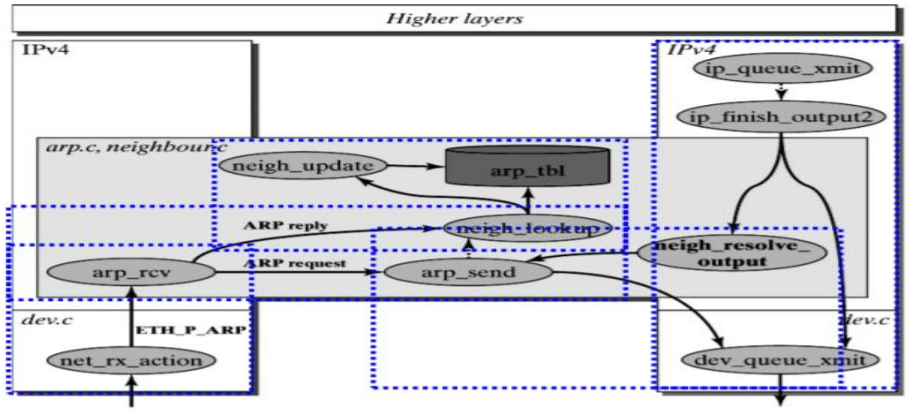
路由模块

IP三层路由转发，最长掩码匹配。

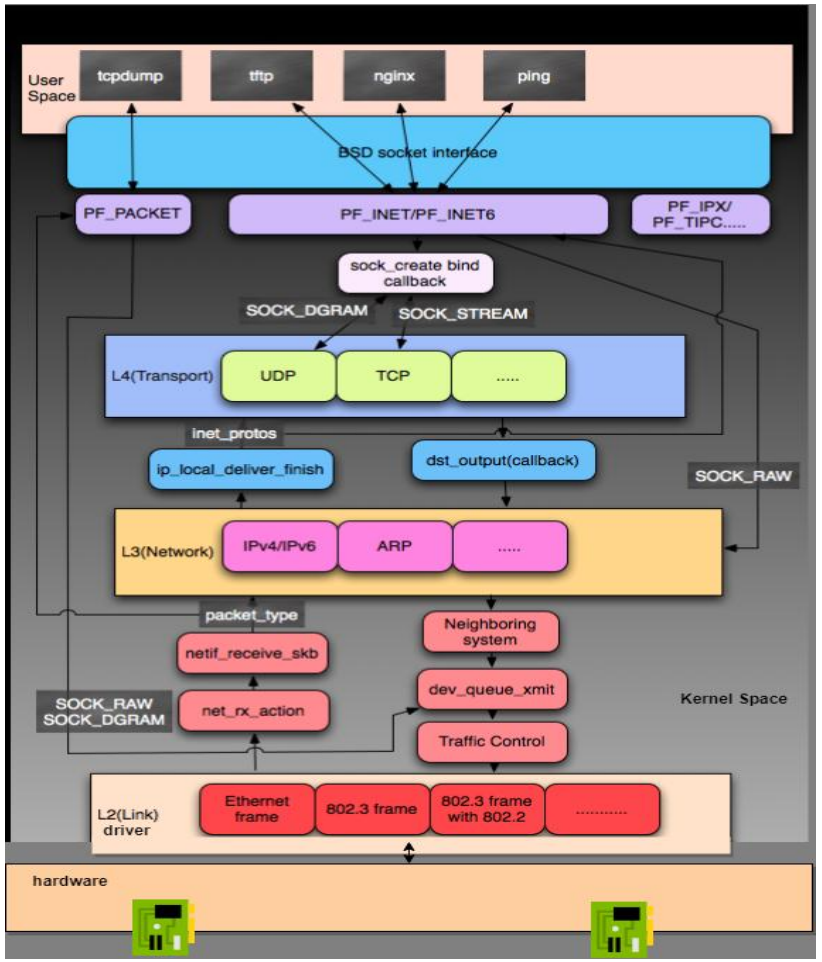


邻居模块

获取根据IP获取对端mac地址，IPv4的ARP协议和IPv6 NDP协议



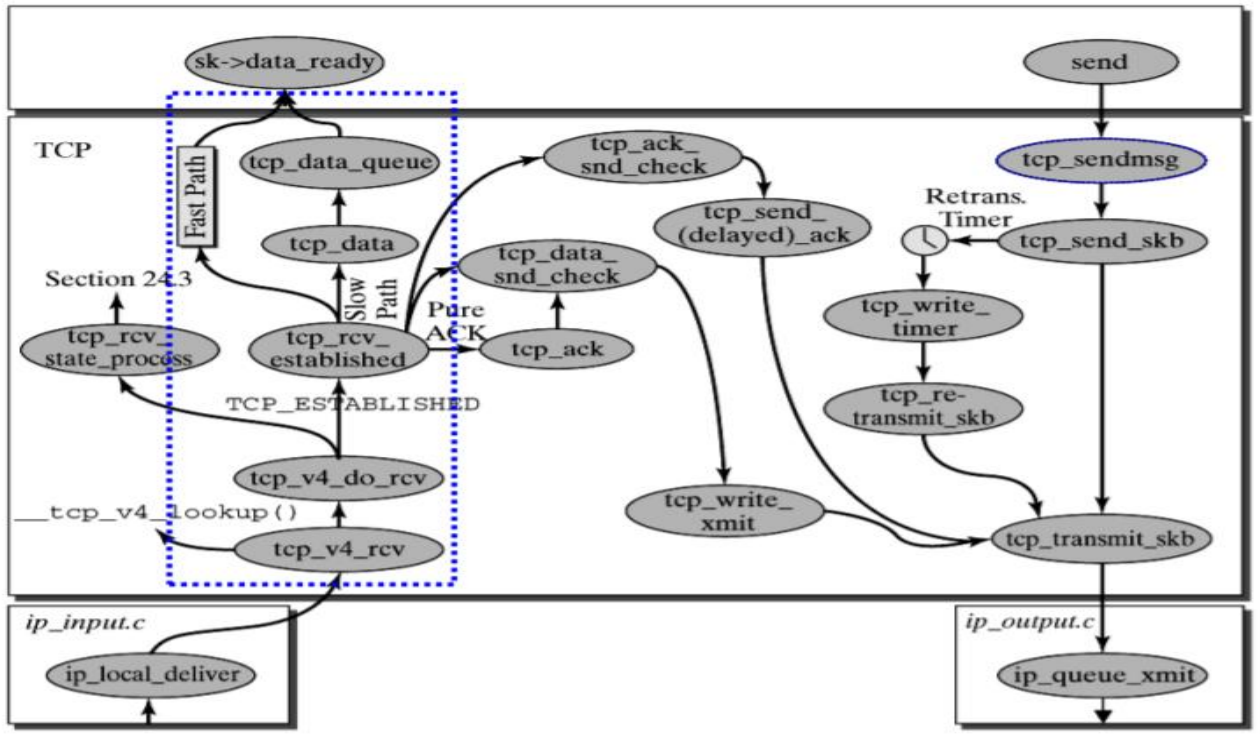
网络子系统—网络协议栈TCP



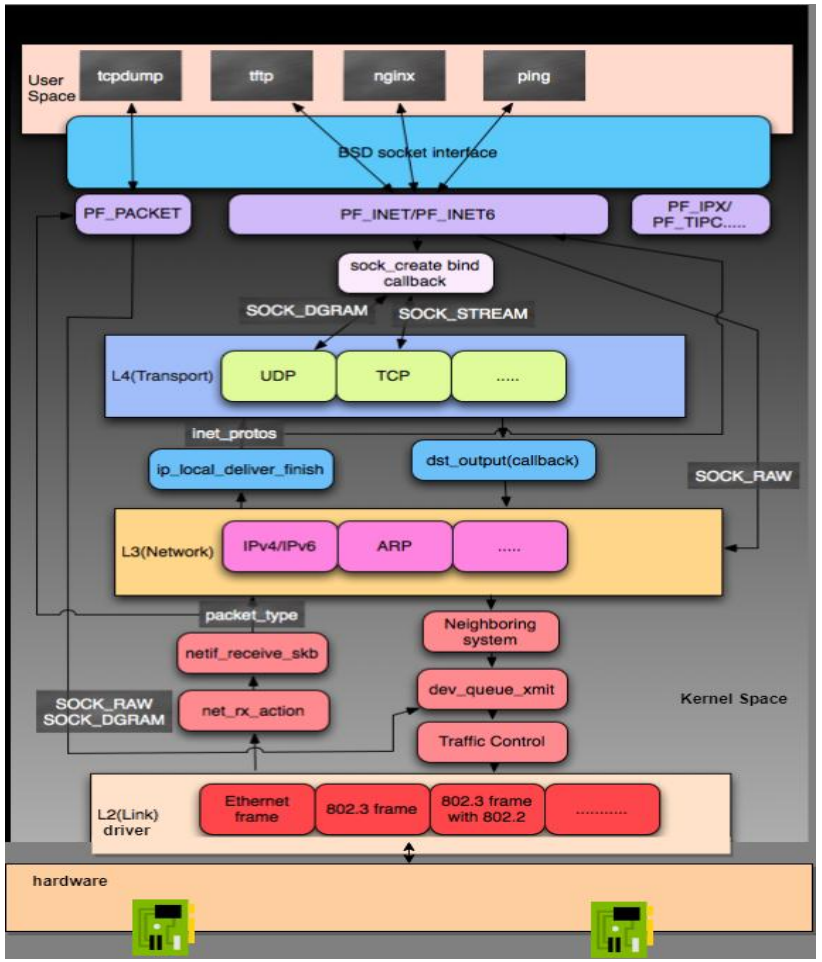
LinuxTCP/IP 网络协议栈

TCP处理模块

- TCP报文收发
- TCP Socket连接管理
- TCP协议状态机，定时器处理
- TCP滑动窗口，拥塞控制框架



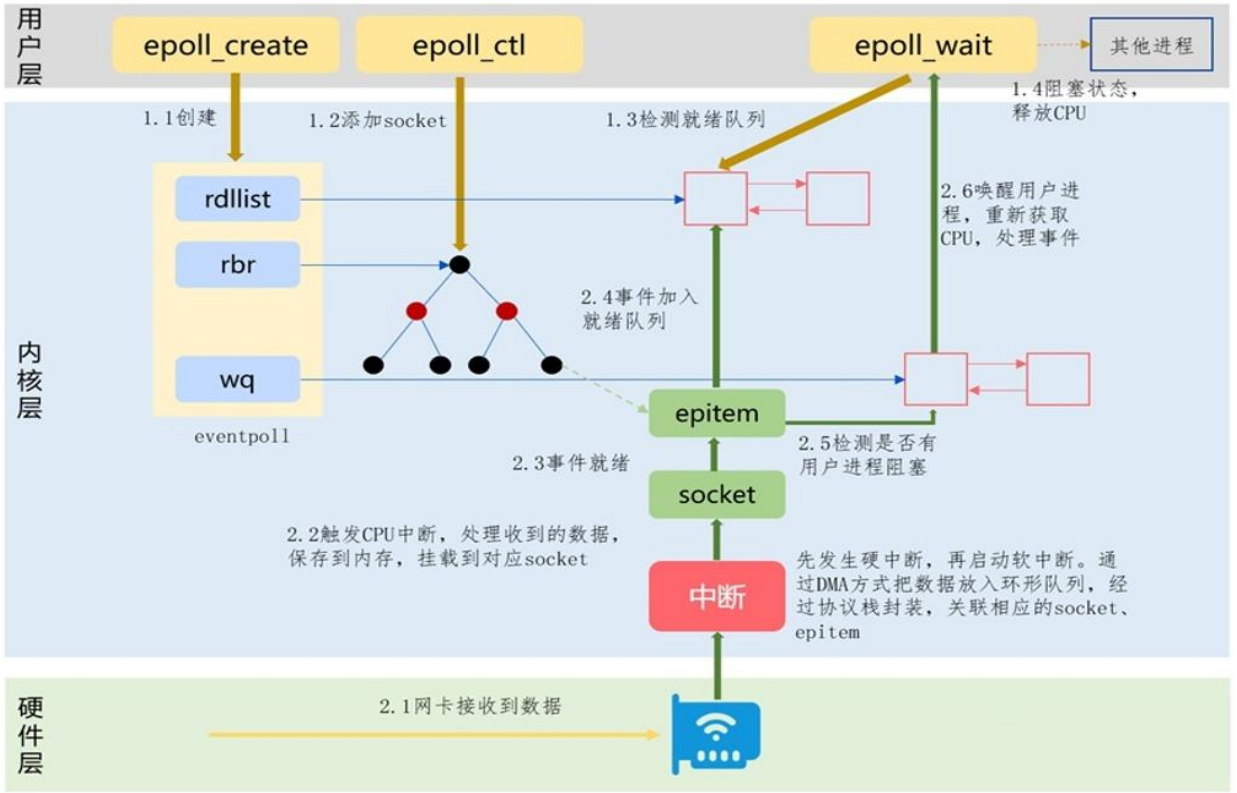
网络子系统—网络协议栈socket接口



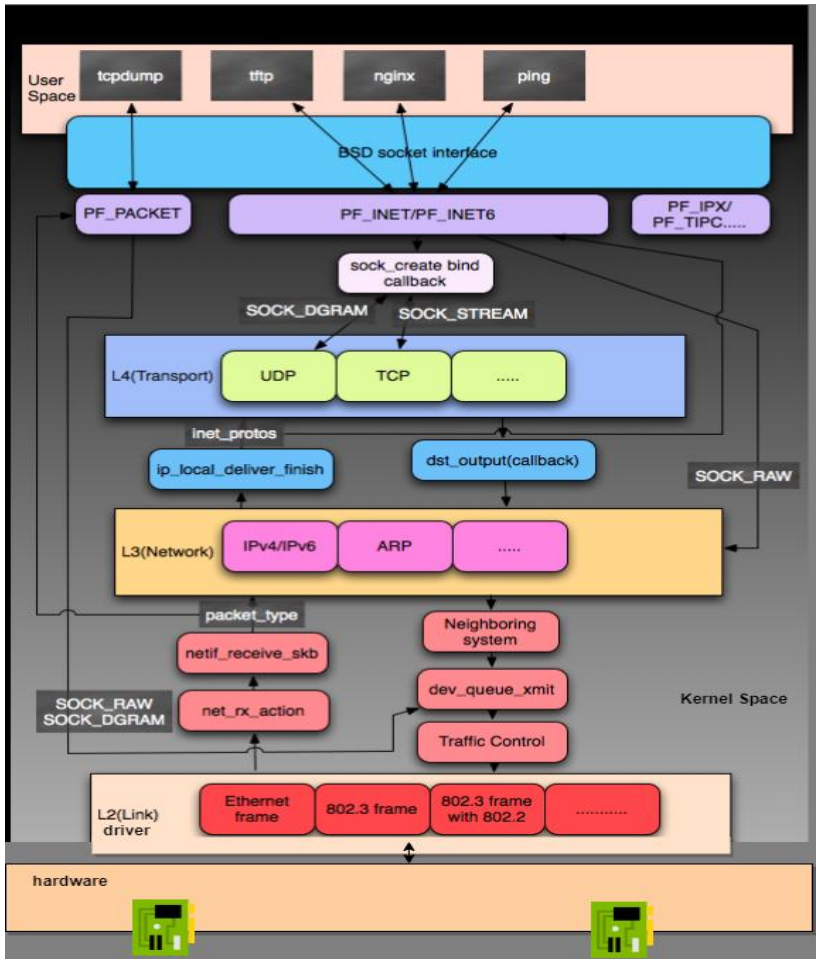
LinuxTCP/IP 网络协议栈

epoll模块

- epoll句柄采用红黑树管理，提升socket句柄CURD性能
- 唤醒相关文件句柄睡眠队列的entry，调用其回调
- 就绪事件用rdllist链表记录。
- 唤醒epoll睡眠队列的task，上报就绪socket



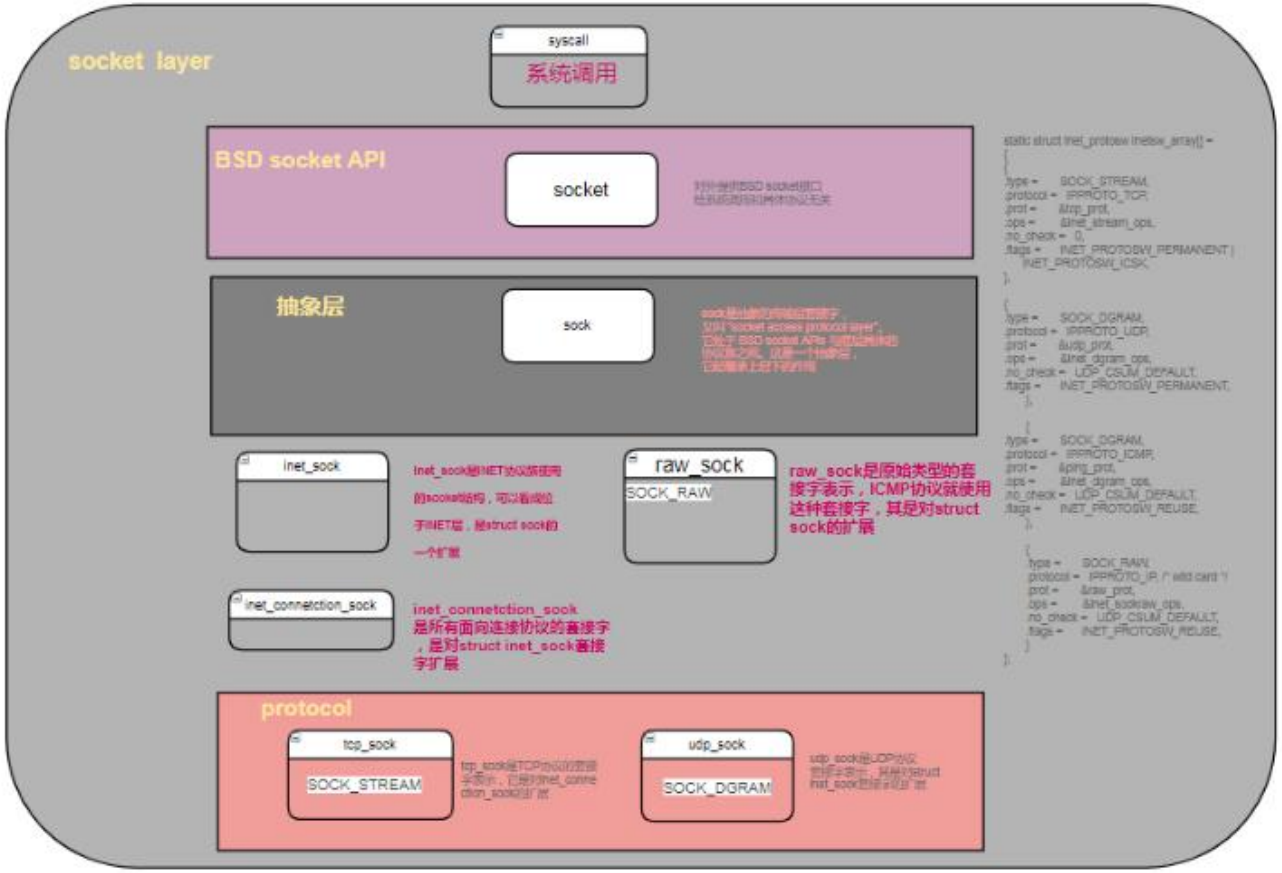
网络子系统—网络协议栈socket接口



LinuxTCP/IP 网络协议栈

Socket框架

- socket系统调用（socket，bind，listen，accept，send，recv等）
- BSD socket API
- 协议栈sock抽象适配层
- tcp/udp/icmp/raw/packet/netlink/... socket管理
- socket选项



网络子系统网络加速—offload（网卡卸载）

TCP 分段卸载 (TSO)
使用 TCP 协议发送大数据包。使用 NIC 处理分段，然后将 TCP、IP和数据链路层头协议添加到每个分段。
UDP 分片卸载 (UFO)
使用 UDP 协议发送大数据包。使用 NIC 将 IP 分段处理成 MTU 大小的数据包以用于大型 UDP 数据报。
通用分段卸载 (GSO)
使用 TCP 或 UDP 协议发送大数据包。如果 NIC 无法处理分段/分段，GSO 会执行相同的操作，绕过 NIC 硬件。这是通过将分段延迟到尽可能晚来实现的，例如，当设备驱动程序处理数据包时。
大型接收卸载 (LRO)
使用 TCP 协议。所有传入的数据包在收到时都会重新分段，从而减少系统必须处理的分段数量。它们可以在驱动程序中或使用 NIC 合并。LRO 的一个问题是它倾向于对所有传入的数据包进行重新分段，通常会忽略标头和其他可能导致错误的信息的差异。启用 IP 转发时，通常无法使用 LRO。LRO 与 IP 转发相结合会导致校验和错误。
通用接收卸载 (GRO)
使用 TCP 或 UDP 协议。在重新分割数据包时，GRO 比 LRO 更严格。例如，它检查每个数据包的 MAC 标头，必须匹配，只有有限数量的 TCP 或 IP 标头可以不同，并且 TCP 时间戳必须匹配。重新分段可以由 NIC 或 GSO 代码处理。
应用卸载：
使用内核 TLS 和 SSL_sendfile() 提高 NGINX 性能

卸载设置：要检查当前的卸载设置，请使用该ethtool命令: ethtool –k eth1

网络子系统网络加速—并发优化

网络并发优化

- **中断亲和**

适当设置网卡中断和CPU绑定，可以最大限度的提升网络性能： /proc/irq/xxx(网卡中断)/smp_affinity_list

- **网卡多队列（RSS）**

Receive Side Scaling（RSS）是所述机构具有多个RX / TX队列过程的数据包。当带有RSS 的网卡接收到数据包时，它会对数据包应用过滤器并将数据包分发到RX队列（每个RX队列可以绑定一个CPU）。过滤器通常是一个哈希函数.

- **RPS**

Receive Packet Steering （RPS）的基本思想是根据每个队列的rps_map 将同一流的数据包发送到特定的 CPU. 软件分流

- **RFS**

Receive Flow Steering（RFS）进一步延伸到应用程序，保证应用的CPU和协议栈一致，减少进程上下文切换。

- **XPS**

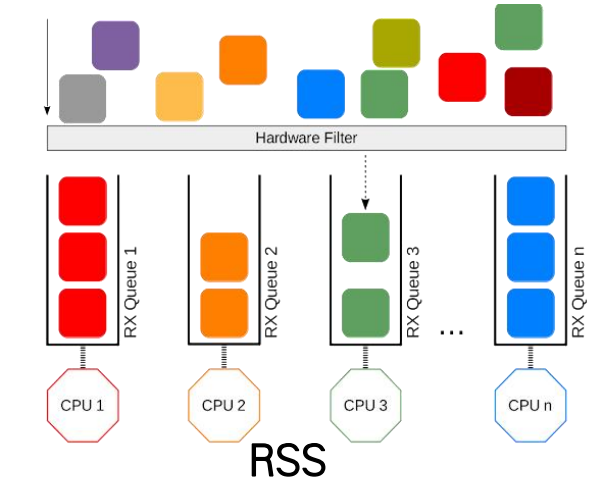
Transmit Packet Steering(XPS），类似RPS，在发送时候，设置好发送队列和CPU一一映射，减少资源争抢，提高性能，

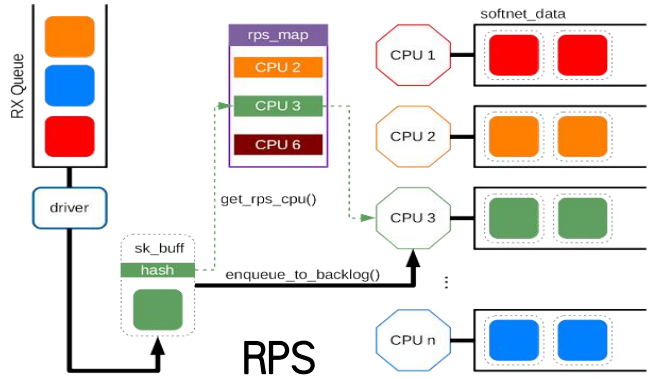
- **SO_REUSEPORT**

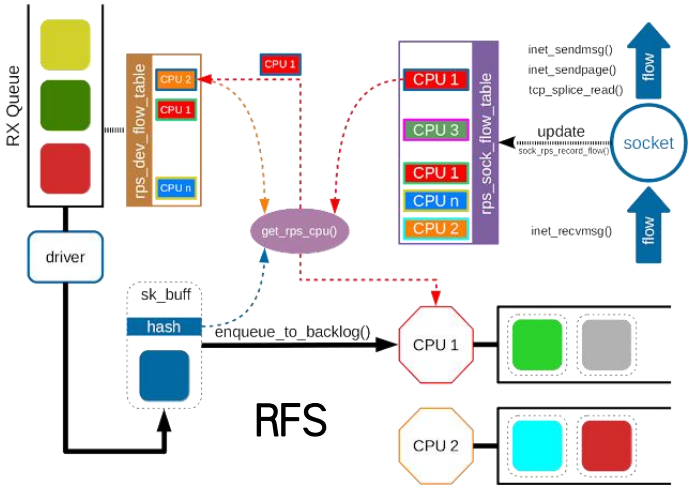
SO_REUSEPORT支持多个进程或者线程绑定到同一端口：允许多个套接字bind()/listen() 同一个TCP/UDP端口

- 每一个线程拥有自己的服务器套接字。
- 在服务器套接字上没有了锁的竞争。

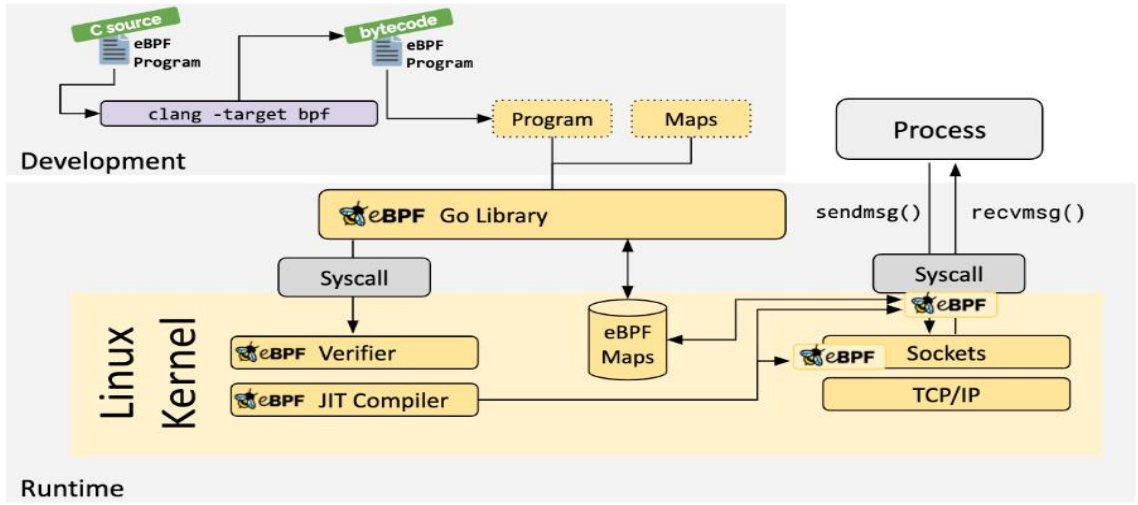
实现socket的负载均衡，热备，热更新等
可以充分利用多核的优势。





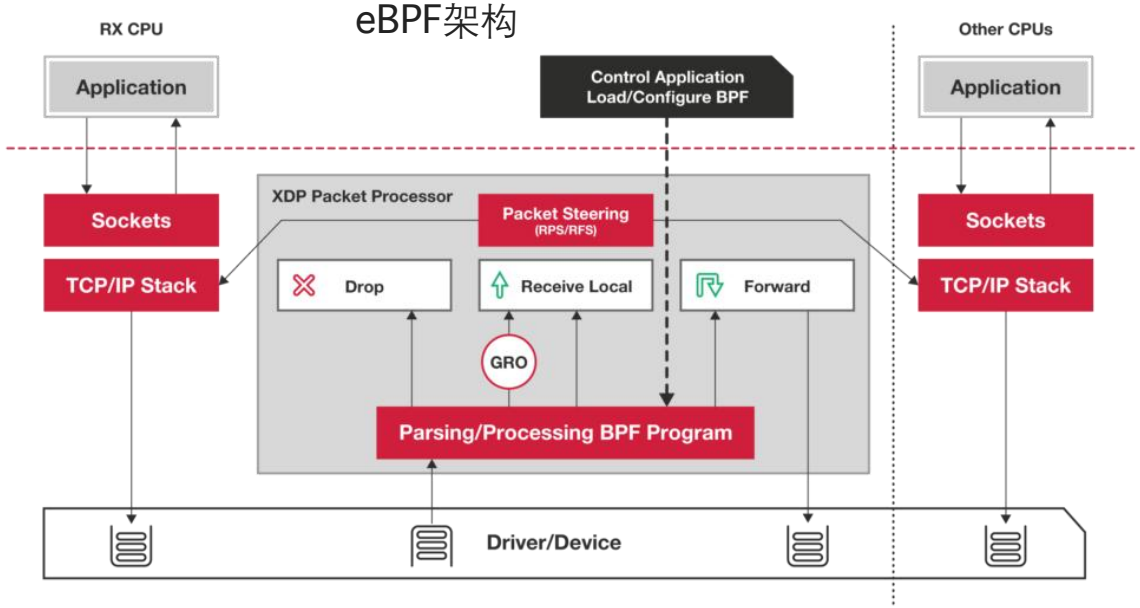


网络子系统网络加速—ebpf && xdp



eBPF架构

- eBPF是一个能够在内核运行虚拟机的技术，提供了一种在内核事件和用户程序事件发生时安全注入代码的机制，使得非内核开发人员也可以对内核进行控制。随着内核的发展，eBPF 逐步从最初的数据包过滤扩展到了网络、内核、安全、跟踪等，而且它的功能特性还在快速发展中。
- 高性能，jit指令优化。
- 降低内核编程门槛，用户态写内核代码，可以用Python，GO，C++等高级语言。
- 高可扩展，可以在线更新内核功能和扩展内核功能，内核可编程。

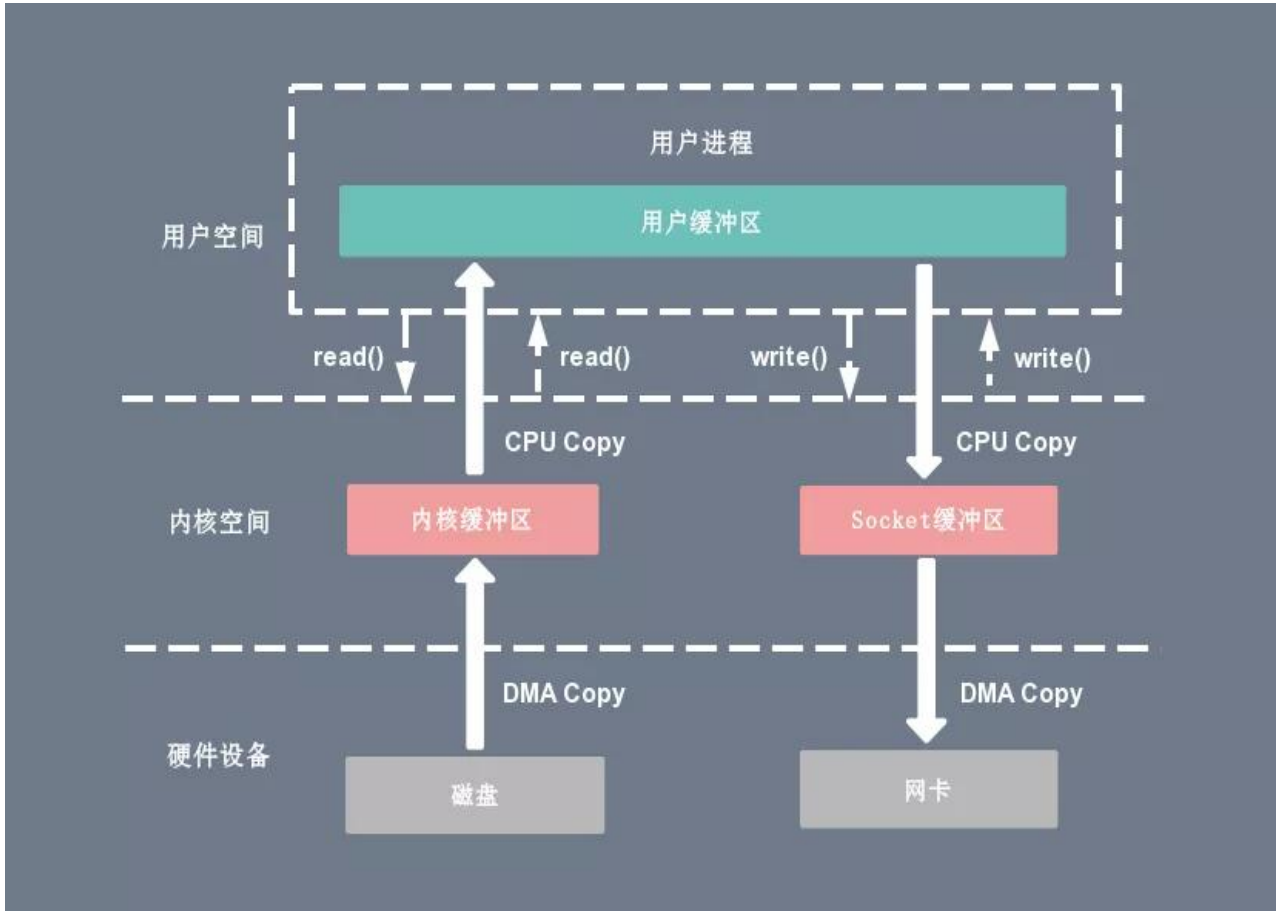


XDP架构

- XDP（eXpress Data Path）为Linux内核提供了高性能、可编程的网络数据路径。由于网络包在还未进入网络协议栈之前就处理，它给Linux网络带来了巨大的性能提升
- 在网络协议栈前处理
- 无锁设计
- 批量I/O操作
- 轮询式
- 直接队列访问
- DDIO（网卡直接IO），支持硬件offload加速
- 支持eBPF，高效开发，安全可靠，性能好
- 和内核耦合紧密，适合基于内核网络组件平滑演进高性能方案，比如DDOS防护，网络采样，高性能防火墙；

网络子系统网络加速-零拷贝

当一个网络数据包通过网卡进入内核，然后再进入用户空间的时候，**至少会经过2次data copy**。



在Linux中零拷贝技术主要实现思路：

- **用户态直接 I/O（DPDK)**：应用程序可以直接访问硬件存储，操作系统内核只是辅助数据传输。这种方式依旧存在用户空间和内核空间的上下文切换，硬件上的数据直接拷贝至了用户空间，不经过内核空间。因此，直接 I/O 不存在内核空间缓冲区和用户空间缓冲区之间的数据拷贝。
- **减少数据拷贝次数(**共享内存mmap等）：在数据传输过程中，避免数据在用户空间缓冲区和系统内核空间缓冲区之间的CPU拷贝，以及数据在系统内核空间内的CPU拷贝，这也是当前主流零拷贝技术的实现思路,。
- **写时复制技术**：写时复制指的是当多个进程共享同一块数据时，如果其中一个进程需要对这份数据进行修改，那么将其拷贝到自己的进程地址空间中，如果只是数据读取操作则不需要进行拷贝操作。

网络子系统网络加速—IO加速技术演进

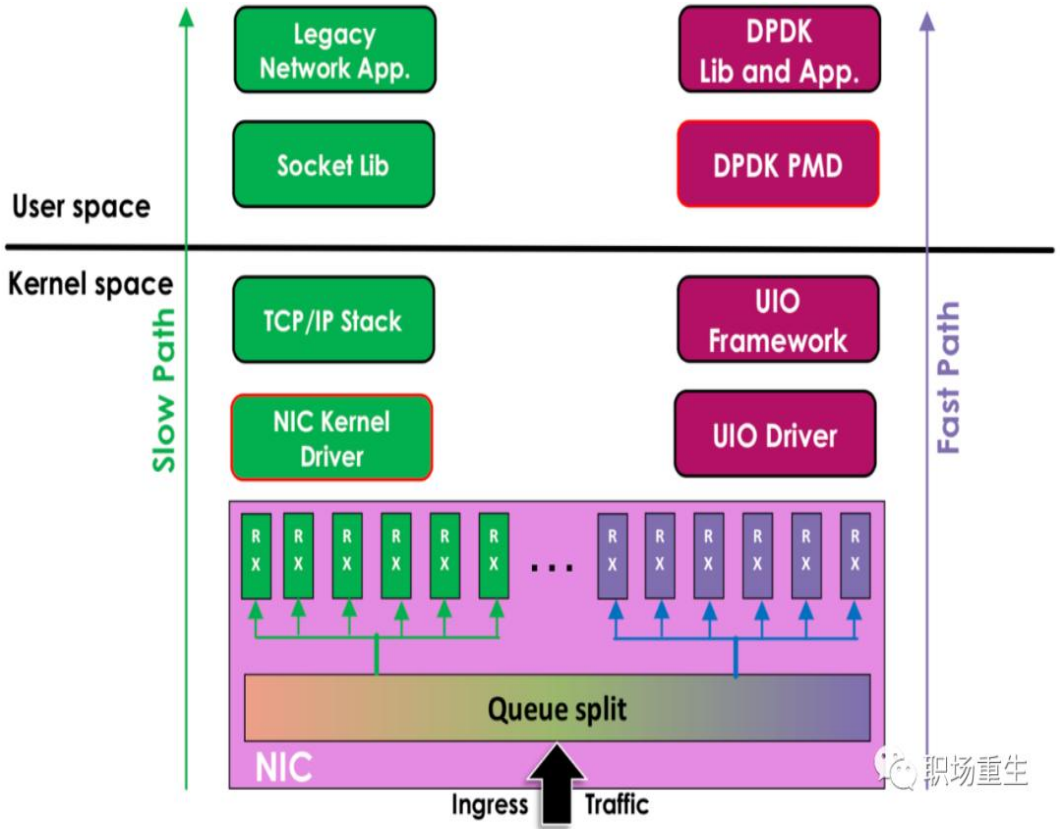
- 百万级PPS
- 协议栈功能全面
- 支持更多网卡

- 千万级PPS
- Bypass kernel，极致优化
- 用户态驱动，协议栈
- 智能网卡

- 亿级PPS
- 高带宽，低时延
- FPGA／P4

网络子系统网络加速--kernelbypass

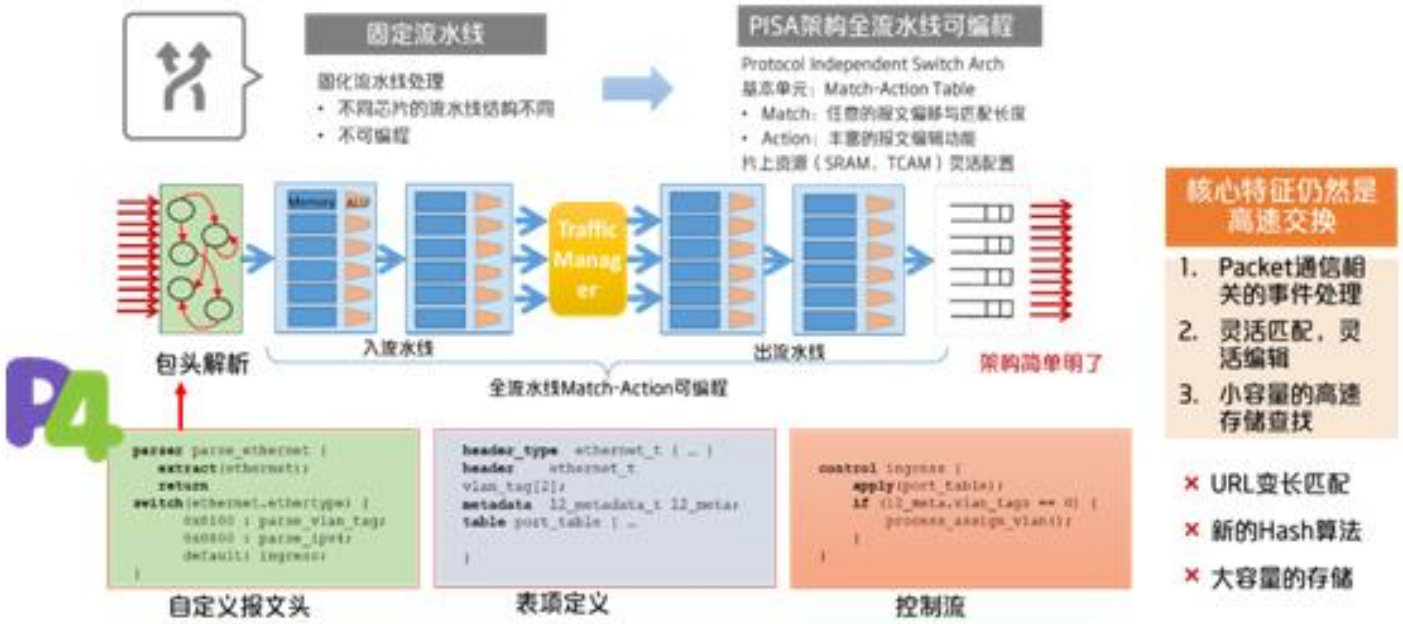
kernelbypass：绕过内核协议栈（路径长，多核性能差）



DPDK优化

- Intel DPDK全称Intel Data Plane Development Kit，是intel提供的数据平面开发工具集，为Intel architecture（IA）处理器架构下用户空间高效的数据包处理提供库函数和驱动的支持，它不同于Linux系统以通用性设计为目的，而是专注于网络应用中数据包的高性能处理，适合高性能网关(IO需求大）场景；
- PMD用户态驱动，使用无中断方式直接操作网卡的接收和发送队列；
- 采用HugePage减少TLB Miss；
- DPDK采用向量SIMD指令优化性能；
- CPU亲缘性和独占；
- 内存对齐：根据不同存储硬件的配置来优化程序，确保对象位于不同channel和rank的起始地址，这样能保证对象并并行加载，性能也能够得到极大的提升；
- Cache对齐，提高cache访问效率；
- NUMA亲和，提高numa内存访问性能；
- 减少进程上下文切换：保证活跃进程数目不超过CPU个数；减少堵塞函数的调用，尽量采样无锁数据结构；
- 利用空间局部性，采用预取Prefetch，在数据被用到之前就将其调入缓存，增加缓存命中率；
- 充分挖掘网卡的潜能：借助现代网卡支持的分流（RSS, FDIR）和卸载（TSO，chksum）等特性；

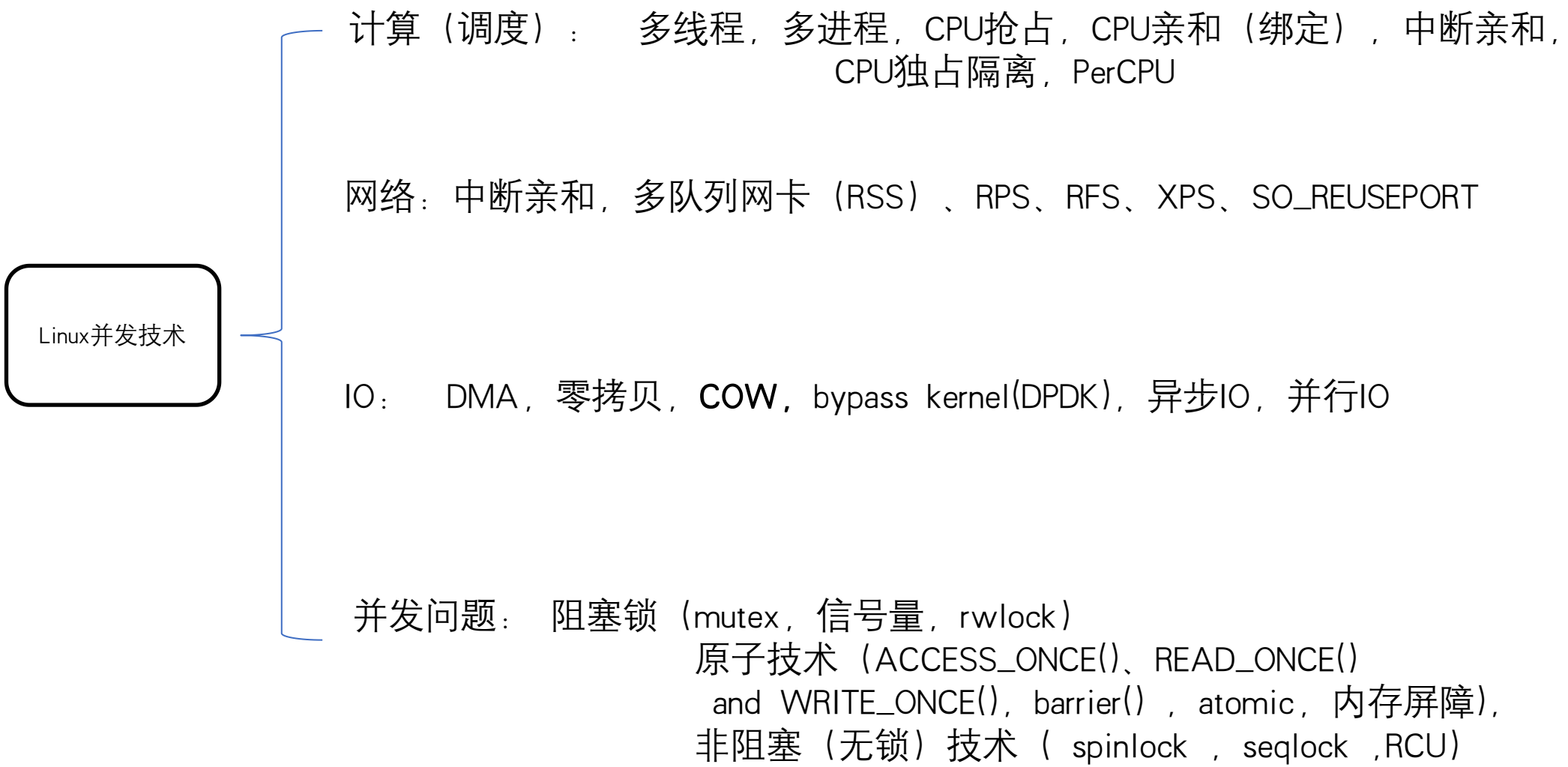
网络子系统网络加速—硬件加速P4



P4架构

- p4 为一种高级可编程协议无关处理语言，结合可编程交换机芯片，编程能力强，可以实现业务offload 到硬件，转发面 p4lang 定制开发，控制面可通过 Apache Thrift、gRPC 接口远程管理，生态繁荣包括P4 Runtime、Stratum；
- 性能高，**1.8T ~ 6.5T** 线速转发，更低时延；
- 每Tbps设备成本大幅降低；
- 主要应用场景是大流量的边界网关，大流量无状态网关，大流量状态网关（当前P4交换机对内存容量支持有限，对配置量有一定的限制）；

深入理解操作系统(Linux)并行技术



深入理解操作系统(Linux)并行技术—如何解决并发问题

有'锁'技术

Mutex：互斥等待。

信号量：多对1等待。

Rwlock：读多写少，读优先，写等待。

原子技术

ACCESS_ONCE()/READ_ONCE()/WRITE_ONCE()：禁止编译器对数据访问的优化，强制从内存而不是缓存中获取数据；

atomic：硬件级加锁（粒度很小）

atomic_inc()/atomic_read()等：整型原子操作；

CAS：原子方式对内存执行读-改-写操作，无锁技术的基础；

无'锁'技术

自旋锁（spinlock）：临界区短，无堵塞。

Seqlock：读多写少，写优先，读重试。

RCU：读多写少，读不影响写，复制更新

内存屏障

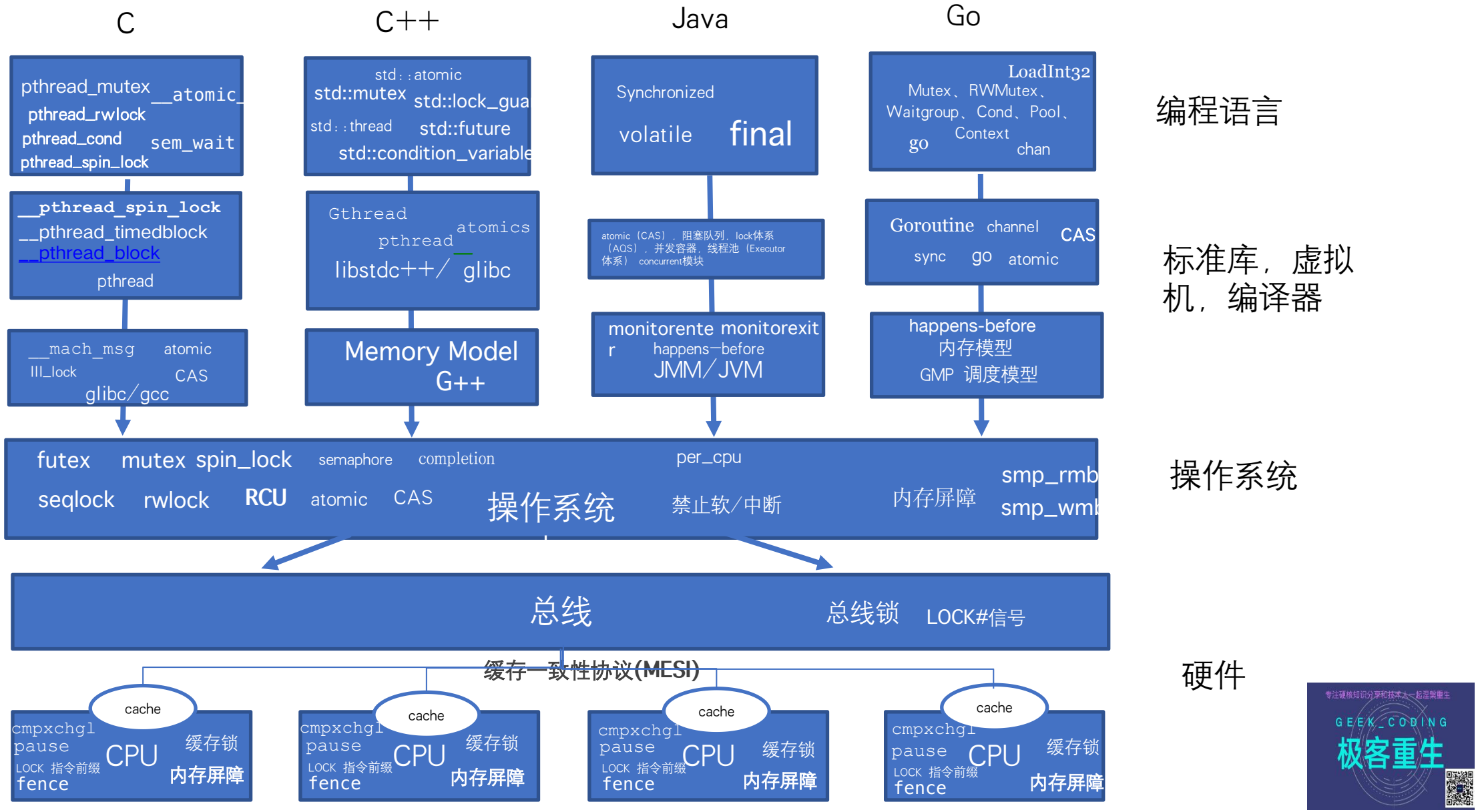
smb_wmb()：写内存屏障，刷新store buffer，同时限制编译器和CPU的乱序优化；

smb_rmb()：读内存屏障，刷新invalidate queue，同时限制编译器和CPU的乱序优化；

smb_mb()：读写内存屏障，同时刷新store buffer和invalidate queue，同时限制编译器和CPU的乱序优化；

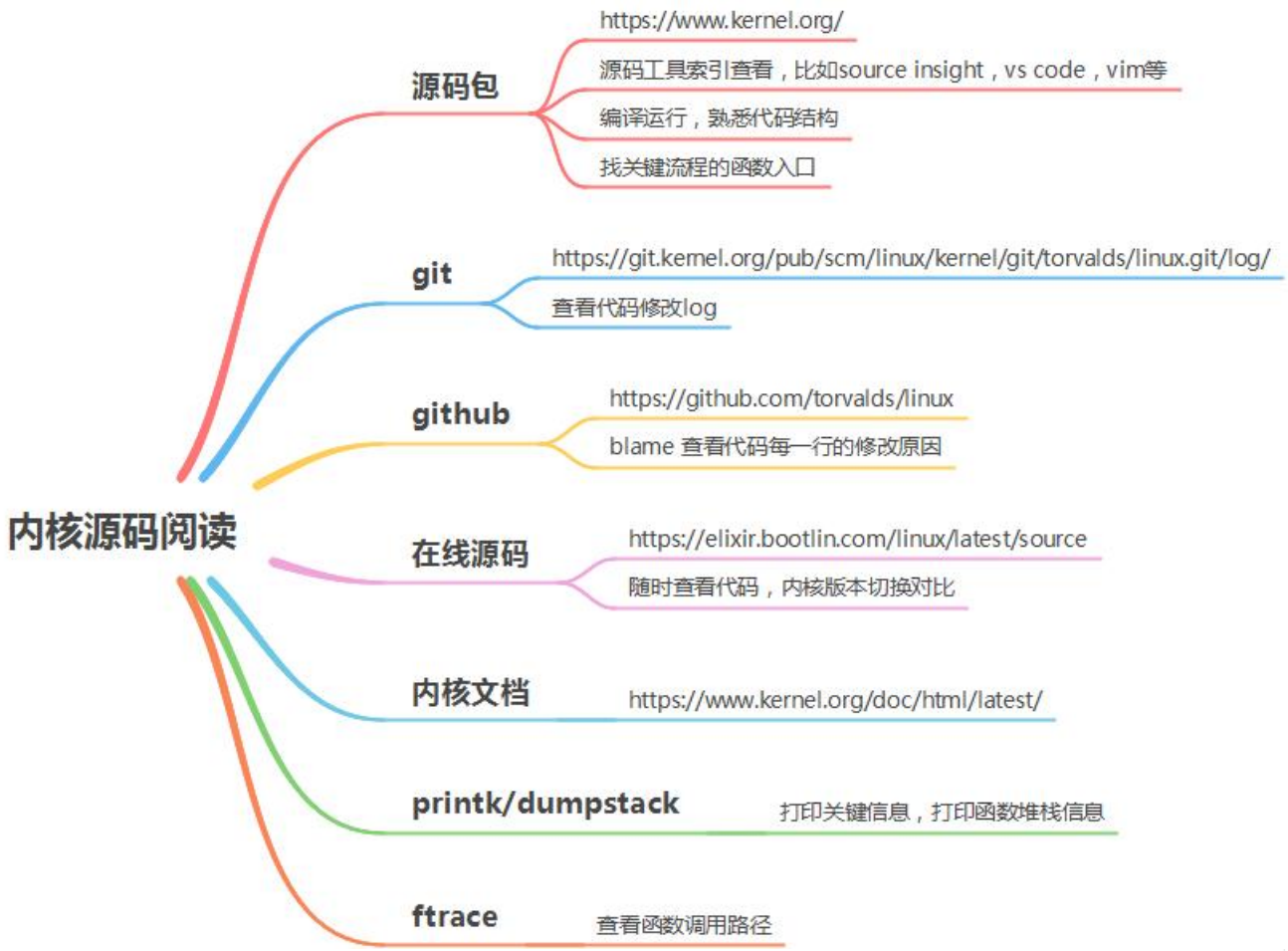
并发/并行问题技术大局观

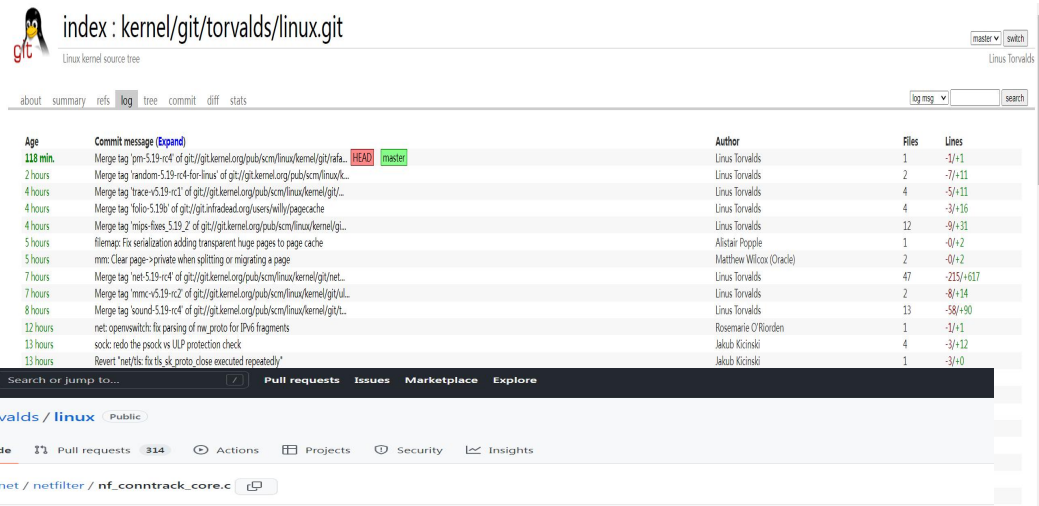
公众号：极客重生

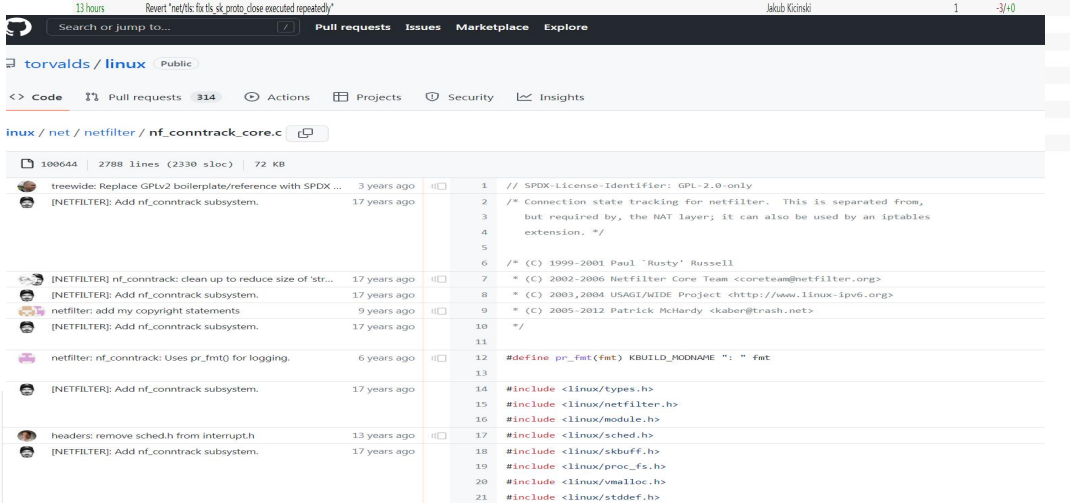


By 公众号@极客重生

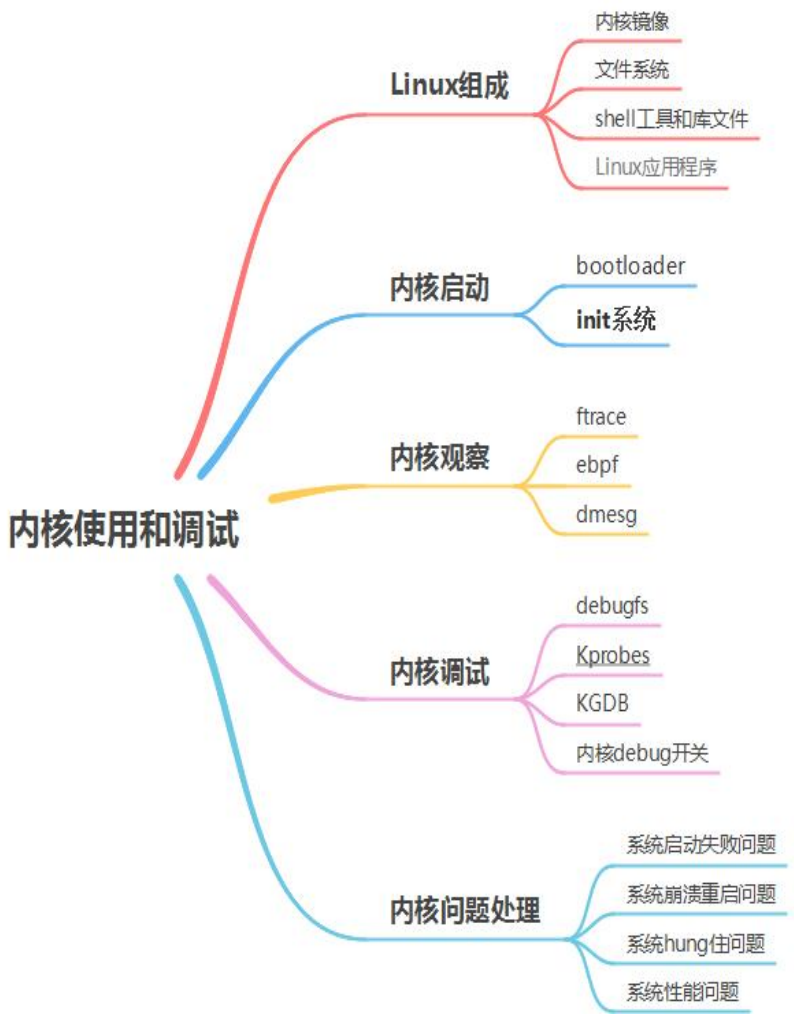
Linux内核细节视角 —— 如何看内核源码



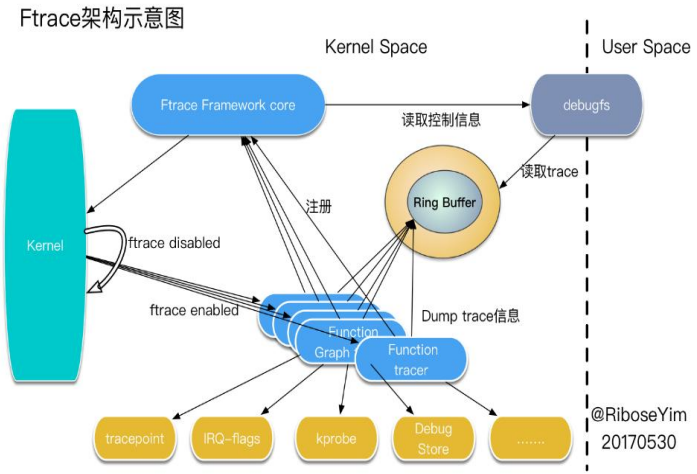


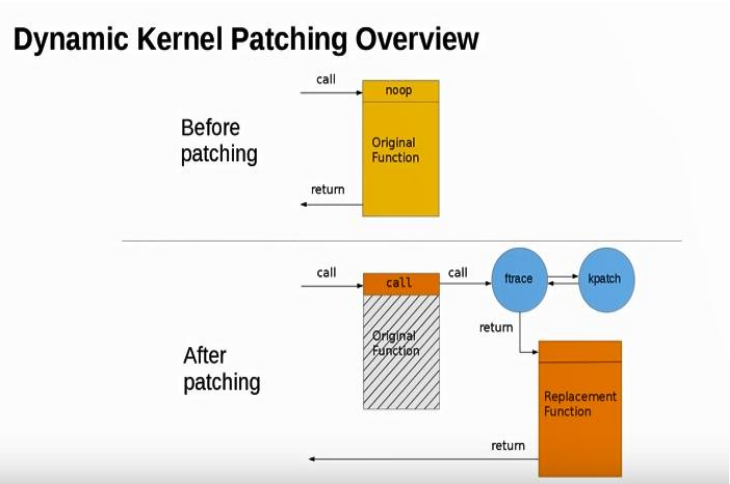


Linux内核实战视角—— 内核如何使用（观察，诊断，解决问题）



- Printk/dmesg
- DUMP_STACK()/WARN_ON(1)
- kgdb/kdb
- debugfs等文件系统
- kprobe/systemTap
- ftrace/trace-cmd
- ebpf
- 热补丁
- 性能分析(perf)
- 死锁检测（Lockup Detector）
- 内存泄漏检测（kmemleak）





推荐学习资料

书籍

- 《操作系统导论》
- 《操作系统：设计与实现》
- 《操作系统—精髓与设计原理》
- 《Linux内核设计与实现》
- 《深入Linux内核架构》
- 《Linux 内核情景分析》
- 《深入理解Linux内核》
- 《深入理解计算机系统》

经典文章

虚拟内存精粹
深入理解 Linux的 I/O 系统
深入理解Linux内存子系统
深入理解虚拟化
Linux网络子系统
Linux Kernel TCP/IP Stack
Linux调度系统全景指南(上篇)
Linux问题分析与性能优化
深入理解Linux 的Page Cache

课程

极客时间：

- 趣谈Linux操作系统
- 极客时间—操作系统45讲(实战)
- 极客时间—性能优化实战

路线

极客星球：

- Linux内核学习路线
- 后台开发基础修炼路线（操作系统部分）