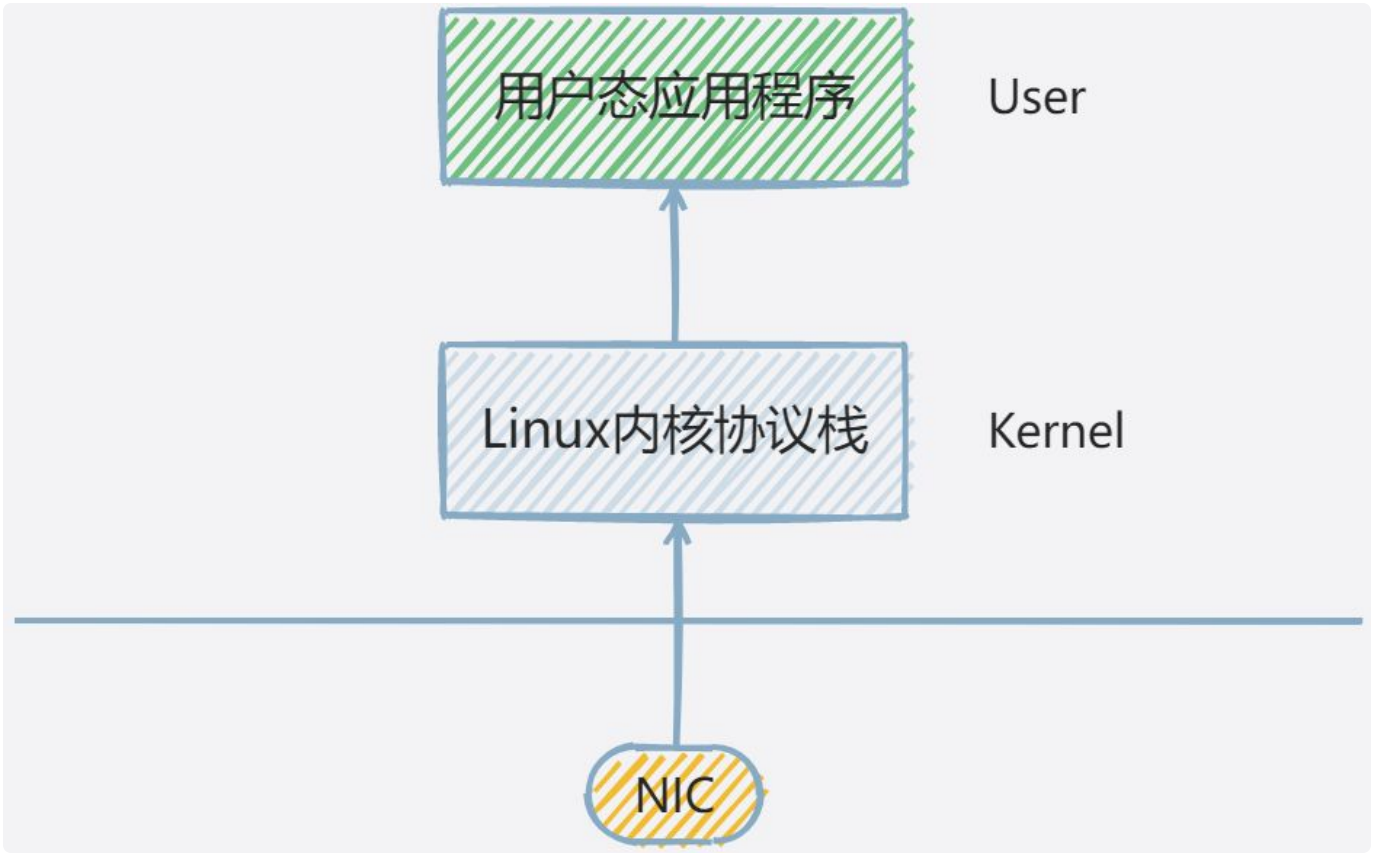
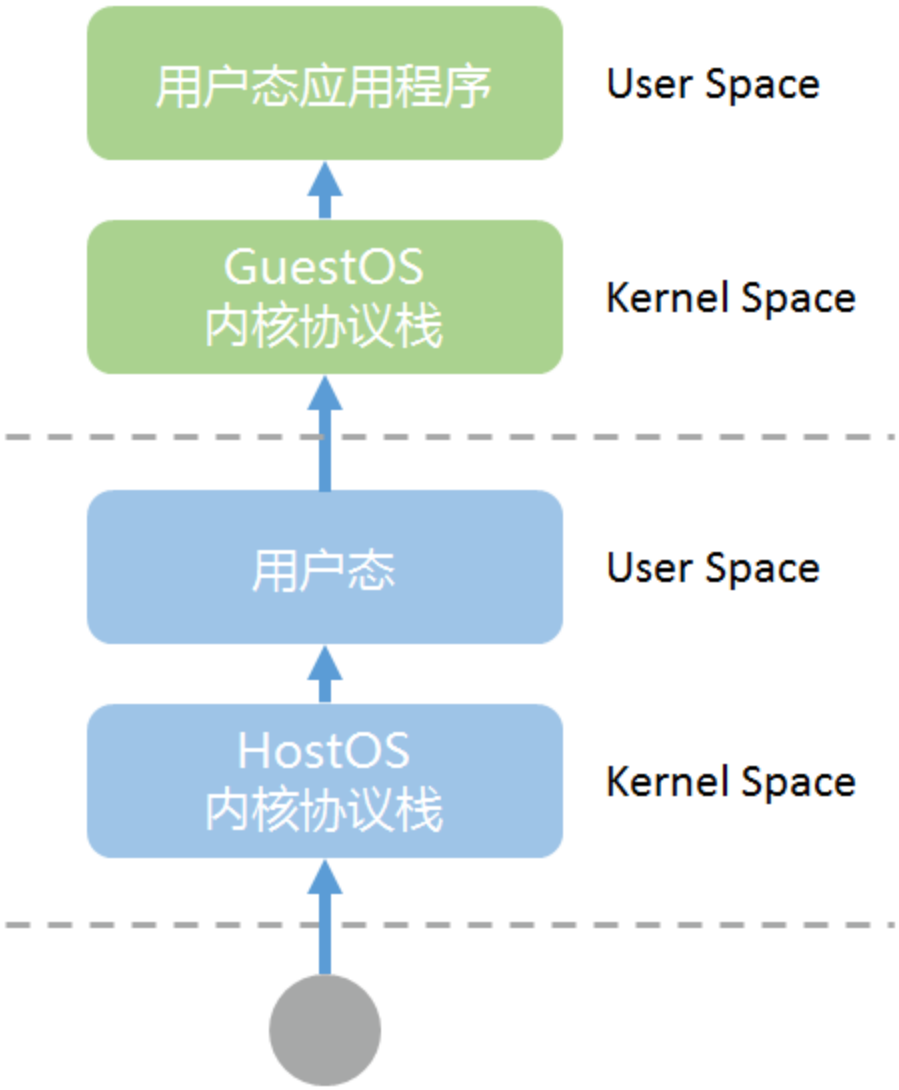
008-了解DPDK

1. 为什么要DPDK

在公有云、NFV等应用场景下，基础设施以CPU为运算核心，往往不具备专用的NP（Network Process）处理器，操作系统也以Linux为主，网络数据包的收发处理路径如下所示：



如果是虚拟化环境则路径更长

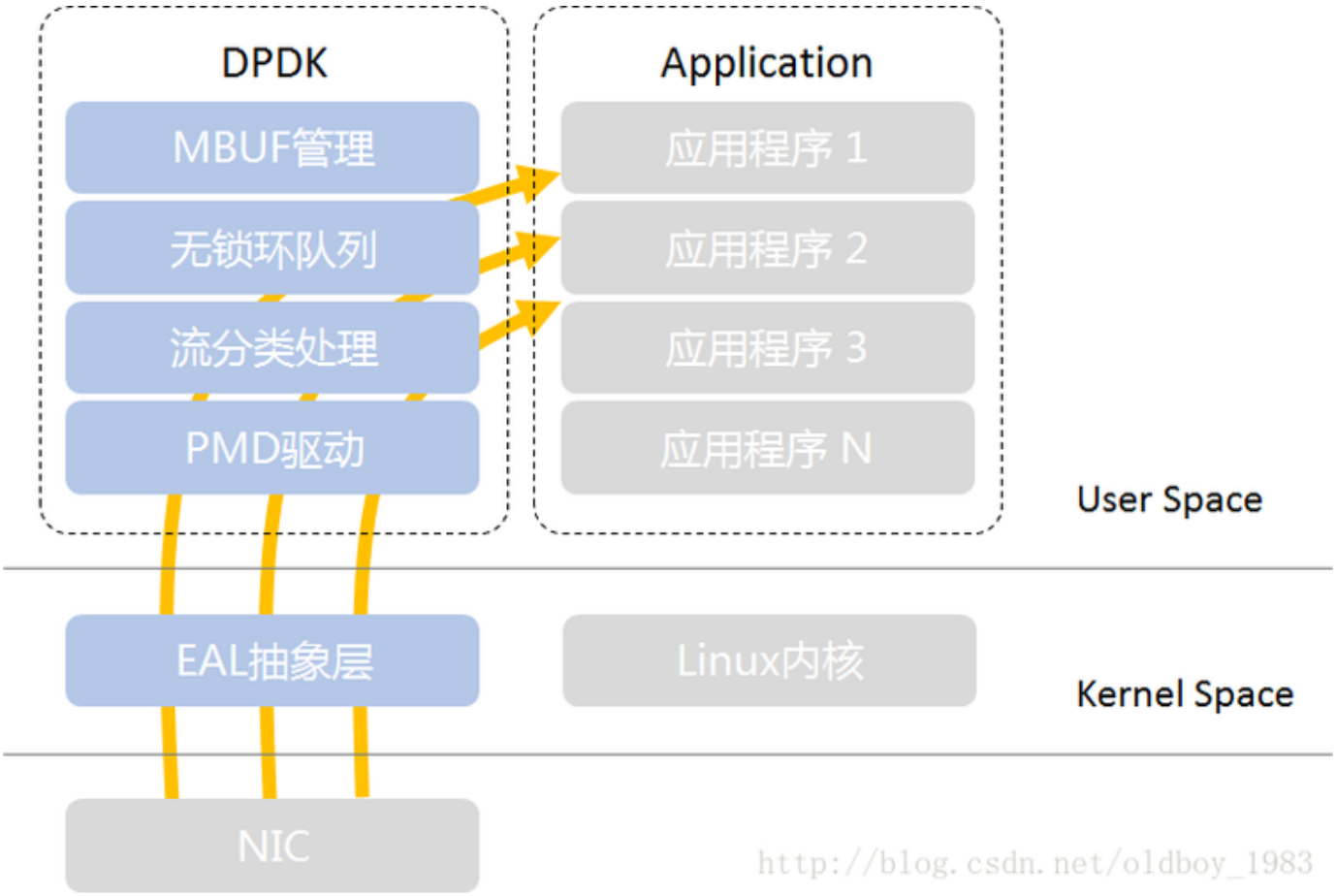


由于包处理任务存在内核态与用户态的切换，以及多次的内存拷贝，系统消耗变大，以CPU为核心的系统存在很大的处理瓶颈。为了提升在通用服务器的数据处理能力和效能，Intel推出了服务与IA（Intel Arichitechure）系统的DPDK技术。

2. DPDK的原理

DPDK是Data Plane Development Kit的缩写，简单说，**DPDK应用程序运行在操作系统的User Space，利用自身提供的数据平面库来收发包处理，绕过了Linux内核协议栈对数据包处理过程，以及提升报文处理效率。**

DPDK是一组lib库和工具包的集合，最简单的架构描述如下图所示：



上图蓝色部分是DPDK的主要组件

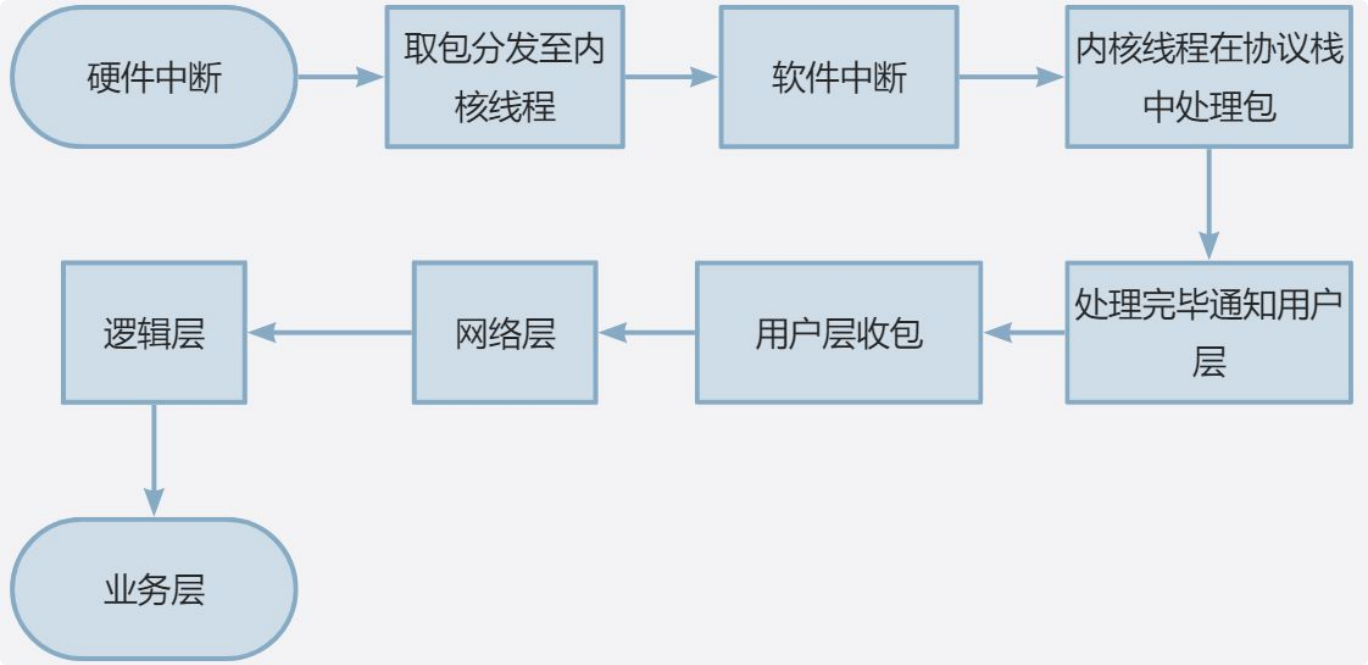
- EAL： Environment Abstract Layer 环境抽象（适配）层，PMD初始化，CPU内核和DPDK线程配置/绑定、设置HugePage大页内存等系统初始化。
- PMD：Pool Mode Driver，轮训模式驱动，通过非中断，以及数据帧进出应用缓冲区内存的零拷贝机制，提高发送、接收数据帧的效率。
- 流分类：Flow Classification，为N元组匹配和LPM（最长前缀匹配）提供优化的查找算法。
- 环队列：Ring Queue，针对单个或多个数据包生产者，单个数据包消费者的出入队列提供无锁机制，有效减少系统开销。
- MBUF缓冲区管理：分配内存创建缓冲区，并通过建立MBUF对象，封装实际数据帧，供应用程序使用。

3.数据处理流程对比

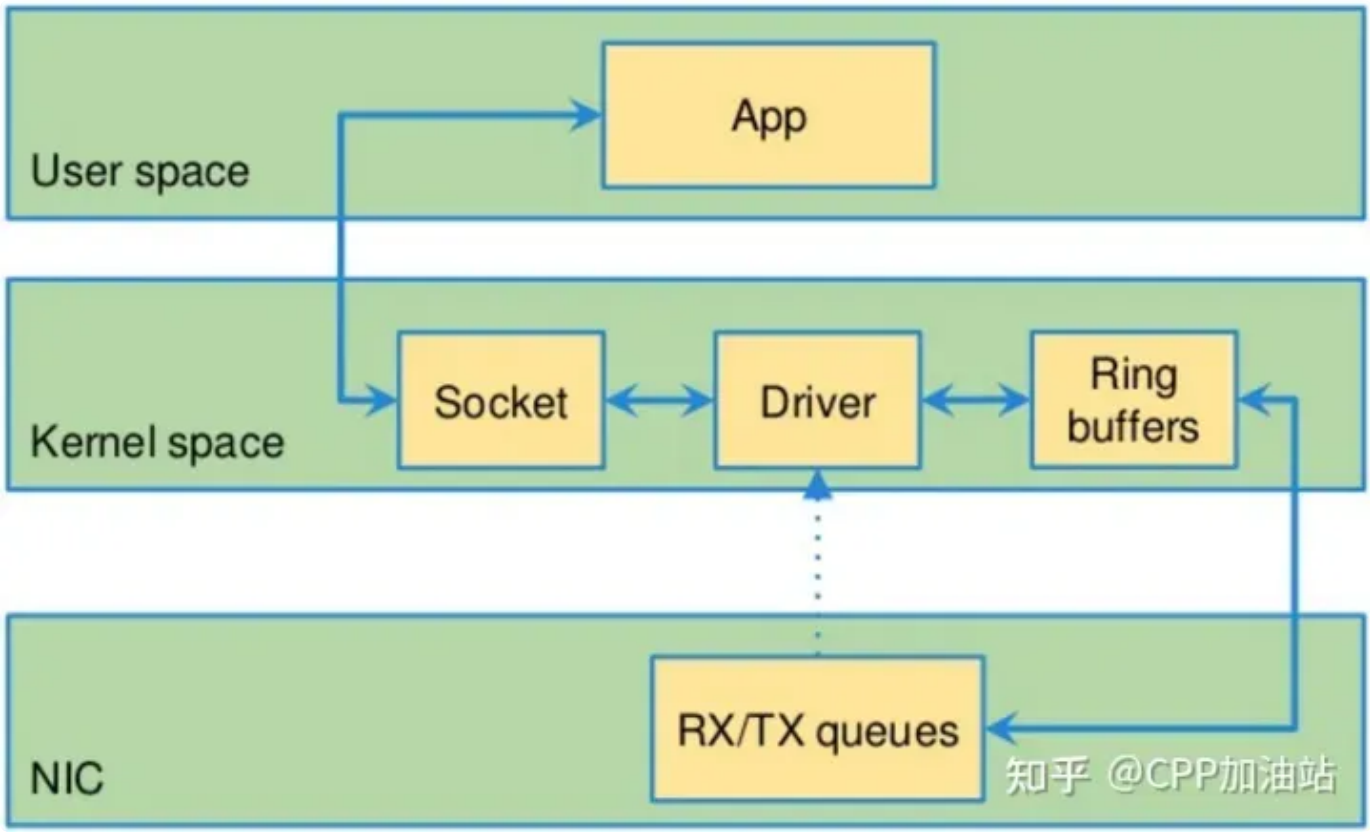
不同于传统的网络协议栈，DPDK对从内核态到用户态的网络处理流程进行了特殊的处理

3.1 传统处理方式

Linux网络协议处理过程如下

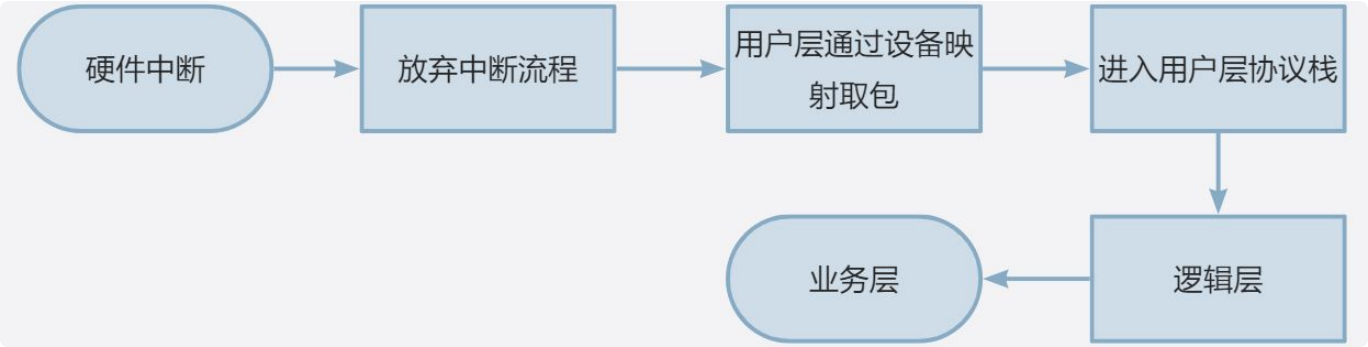


首先网卡通过中断方式通知协议栈对数据包进行处理，协议栈先会对数据包进行合法性等必要的检验，然后判断数据包目标是否为本机的socket，满足条件则会将数据包拷贝一份向上递交给用户socket处理，**不仅处理路径冗长，还需要从内核到应用层的一次拷贝过程**。

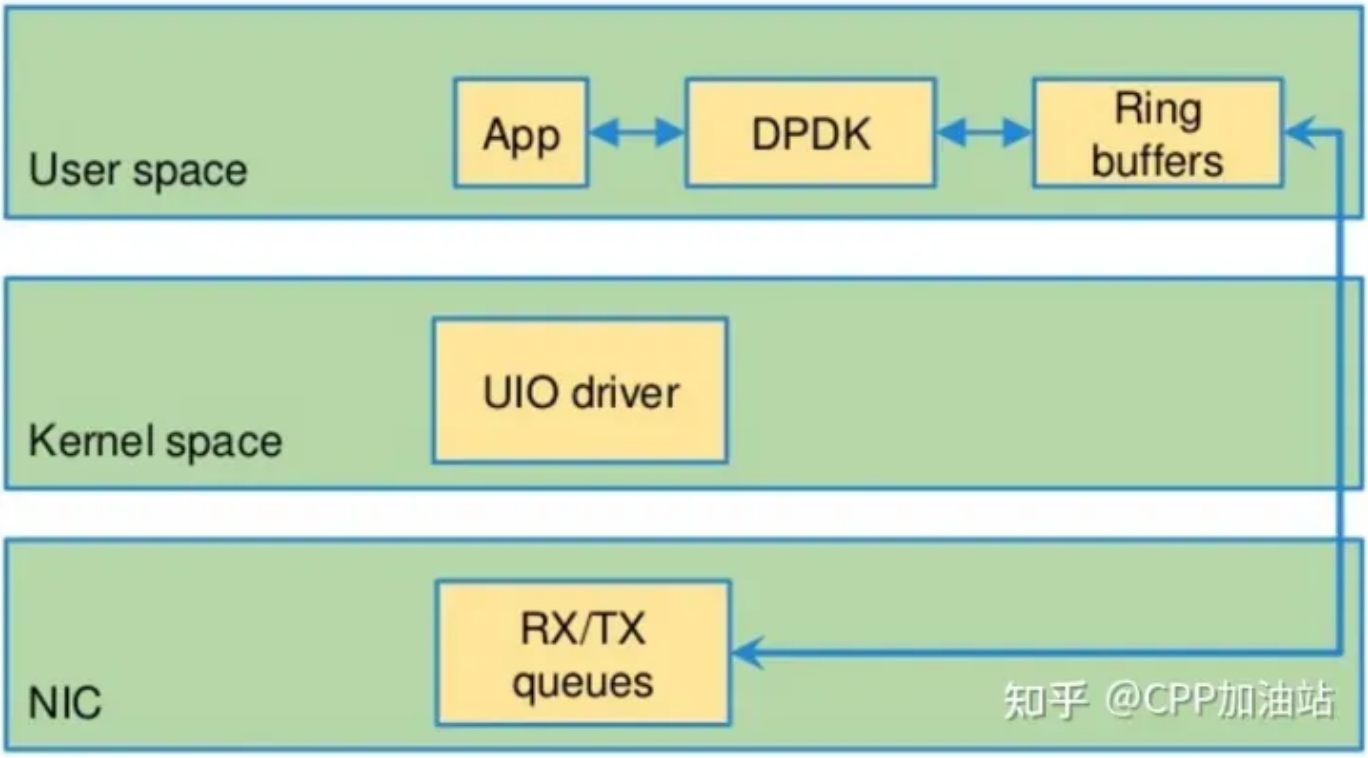


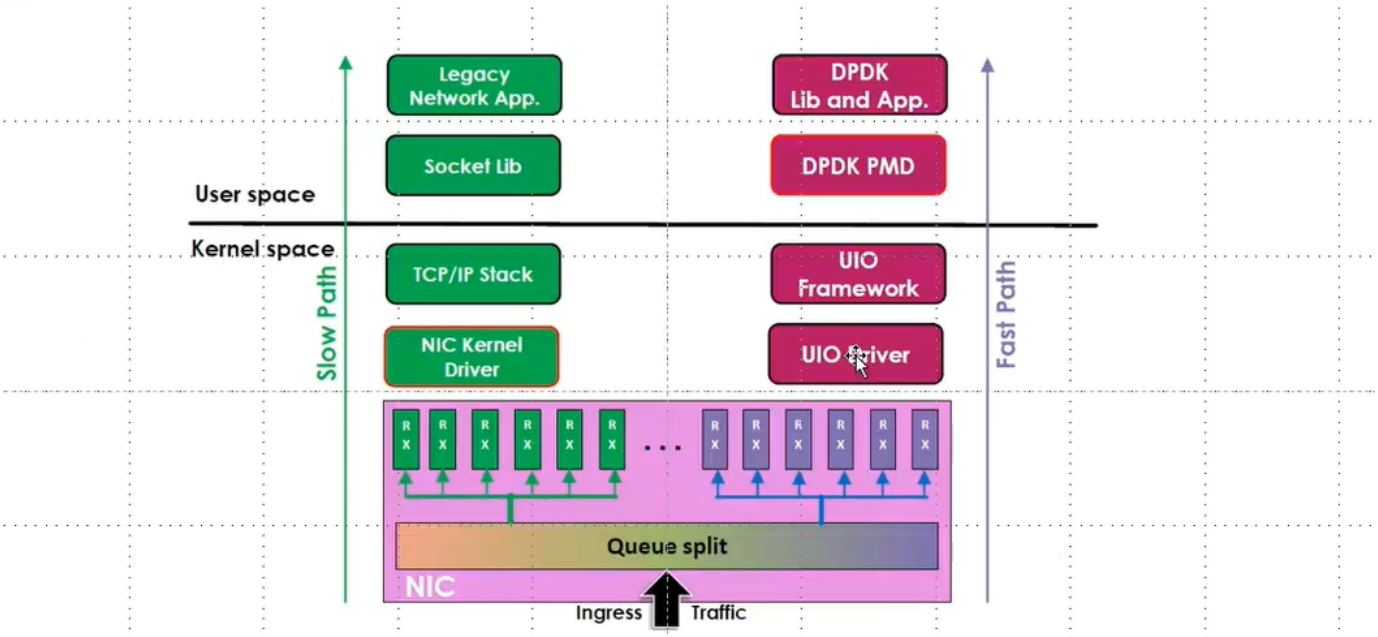
3.2 DPDK处理方式

DPDK网络协议处理过程



DPDK针对Intel网卡实现了基于轮训方式的PMD驱动，该驱动由API，用户空间运行的驱动程序构成，该驱动使用无中断操作网卡的接收和发送队列（除了链路状态通知必须采用中断方式外）。PMD驱动从网卡上接收数据包后，会直接通过DMA方式传输到预分配的内存中，同时更新无锁环形队列中的数据包指针，不断轮训的应用程序很快就能感知收到数据包，并在预分配的内存地址上直接处理数据包。





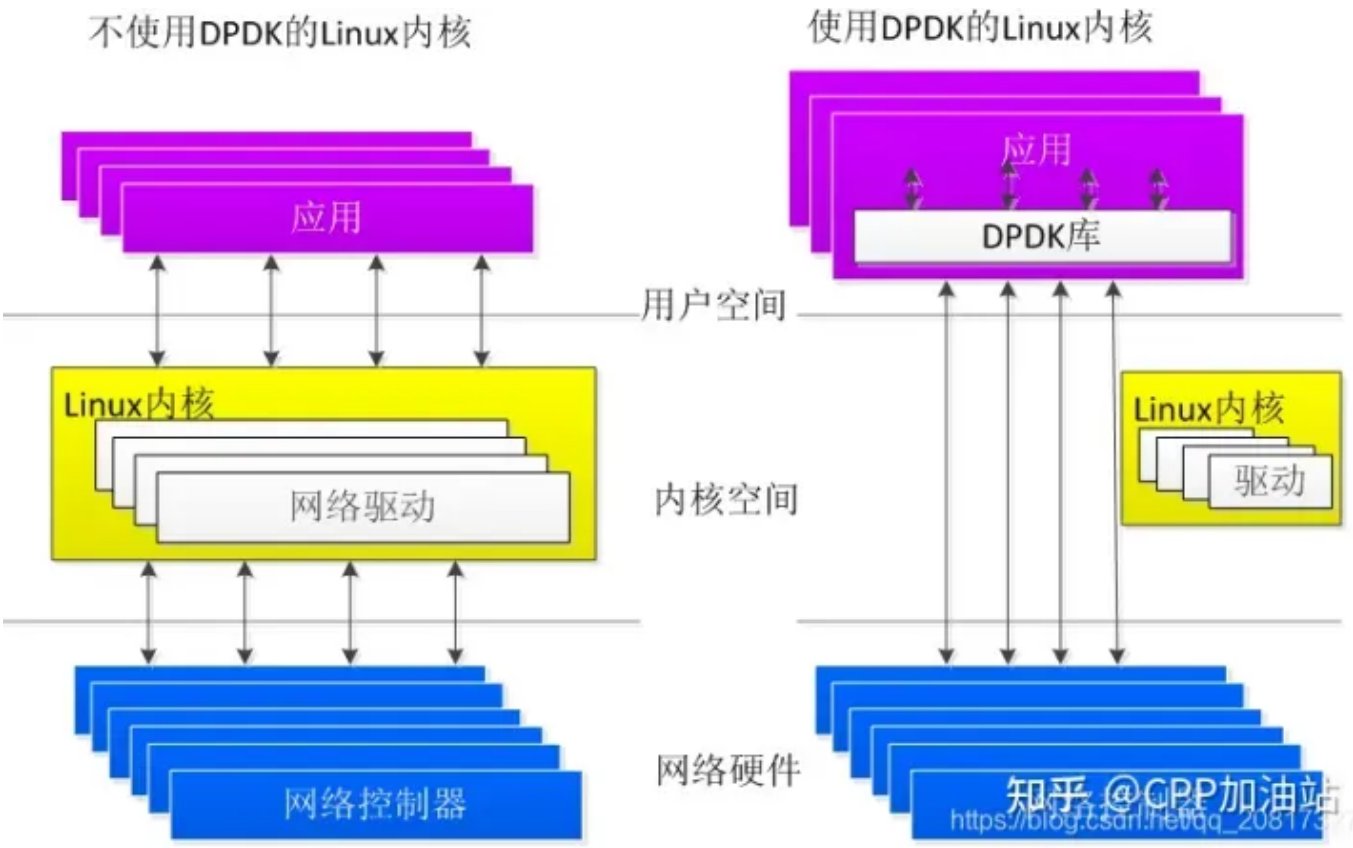
左边是原来的方式数据从 网卡 -> 驱动 -> 协议栈 -> Socket接口 -> 业务
右边是DPDK的方式，基于UIO（UserSpace I/O）旁路数据。数据从 网卡 -> DPDK轮询模式-> DPDK基础库 -> 业务
用户态的好处是易用开发和维护，灵活性好。并且Crash也不影响内核运行，鲁棒性强。
DPDK支持的CPU体系架构：x86、ARM、PowerPC（PPC）

DPDK拦截中断，不触发后续中断流程，并绕过协议栈，通过UIO技术将网卡收到的报文拷贝到应用层处理，报文不再经过内核协议栈。减少了中断，DPDK的包全部在用户控件使用内存池管理，内核控件与用户空间的内存交互不用进行拷贝，只做控制权限转移，减少报文拷贝过程，提高报文的转发效率。

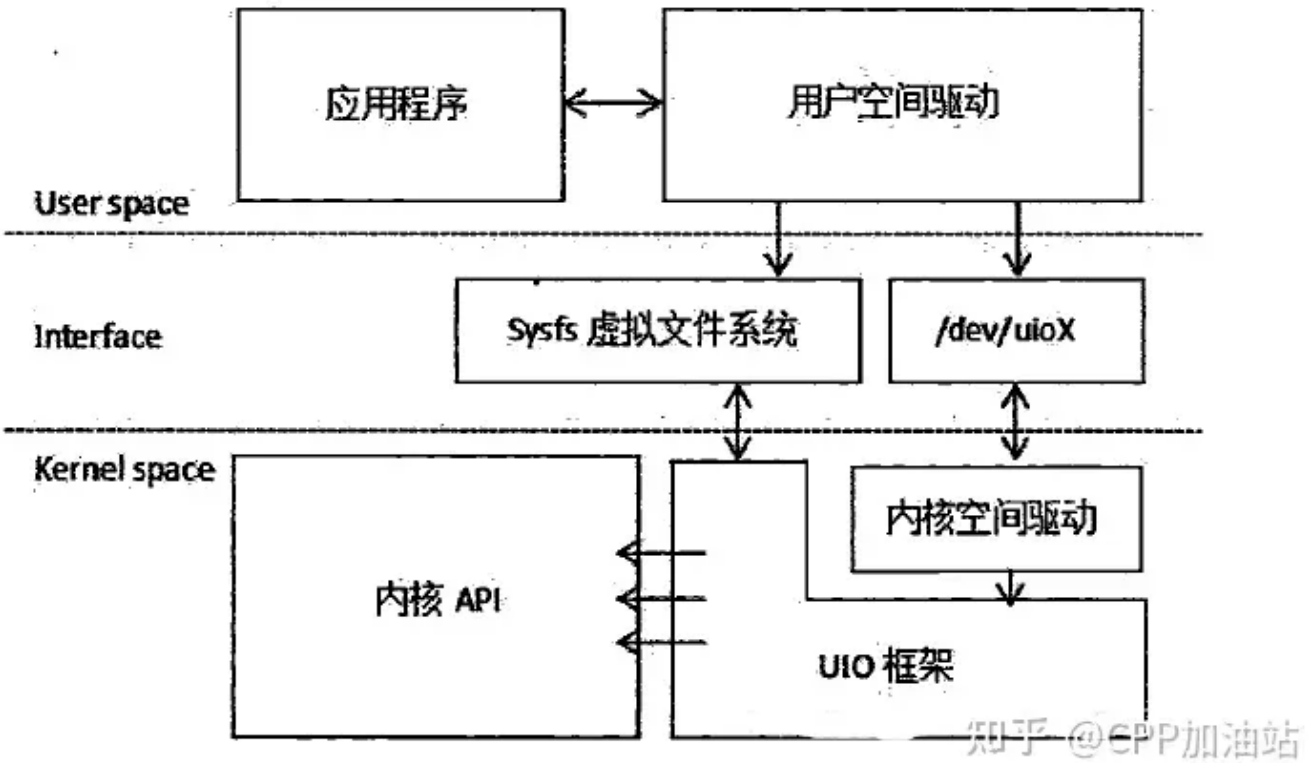
4.了解DPDK使用的技术

4.1 UIO

UIO 的全称为Linux Userspace I/O，提供应用空间下驱动程序的支持，也就是网卡驱动是运行在用户空间的，减少了报文在内核空间和用户空间的内存多次拷贝。如图：DPDK绕过了Linux内核的网络驱动模块，直接从网络硬件到达用户空间，不需要进行频繁的内存拷贝和系统调用。根据官方给出的数据，DPDK裸包反弹每个包需要80个时钟周期，而传统Linux内核协议栈每包需要2k~4k个时钟周期。DPDK能显著提升虚拟化网络设备的数据采集效率。



下图为UIO技术的工作原理图



UIO技术将设备分为用户空间驱动和内核空间驱动两部分，内核空间驱动主要负责设备资源分配，UIO设备注册以及小部分中断响应函数，驱动的大部分工作在用户空间驱动程序下完成。通过UIO框架提供的

API接口将UIO的驱动注册到内核，注册完成后将生成存有设备物理地址等信息的map文件，**用户态进程访问该文件将设备对应的内存空间地址映射到用户空间，即可直接操作设备的内存空间**，UIO技术使得应用程序可以通过用户空间驱动程序直接操作设备的内存空间，避免了数据在内核缓冲区和应用缓冲区的多次拷贝，提高数据处理的效率。

4.2 PMD

DPDK用户空间的轮询模式驱动：用户空间驱动使得应用程序不需要经过linux内核就可以访问网络设备卡。网卡设备可以通过DMA方式将数据包传输到事先分配好的缓冲区，这个缓冲区位于用户空间，应用程序通过不断轮询的方式可以读取数据包并在原地址上直接处理，不需要中断，而且也省去了内核到应用层的数据包拷贝过程。

相对于linux系统传统中断方式，Intel DPDK避免了中断处理、上下文切换、系统调用、数据复制带来的性能上的消耗，大大提升了数据包的处理性能。同时由于Intel DPDK在用户空间就可以开发驱动，与传统的在内核中开发驱动相比，安全系数大大降低。因为内核层权限比较高，操作相对比较危险，可能因为小的代码bug就会导致系统崩溃，需要仔细的开发和广泛的测试。而在应用层则相反，比较安全，且在应用层调试代码要方便的多。

5.DPDK总结

DPDK的核心技术如下：

- 通过UIO技术将报文拷贝到用户空间处理
- 通过大页内存，降低cache miss，提高命中率，进而提升cpu的访问速度
- 通过cpu的亲和性，绑定网卡，cpu绑核，减少cpu任务切换
- 通过无锁队列，减少资源竞争

DPDK的核心思想如下：

- 用户态模式的PMD驱动，去中断，避免内核态和用户态内存拷贝，减少系统开销，从而提升I/O吞吐能力
- 用户态有一个好处，一旦程序崩溃，不至于导致内核崩溃，健壮性更好，鲁棒性更强
- HugePage，通过更大的内存页（如1G内存页），减少TLB（Translation Lookaside Buffer，即快表） Miss，Miss对报文转发性能影响很大
- 多核设备上创建多线程，每个线程绑定到独立的物理核，减少线程调度的开销。同时每个线程对应着独立免锁队列，同样为了降低系统开销
- 向量指令集，提升CPU流水线效率，降低内存等待开销