C10X & Kernel Bypass

浙江大学
计算机系统结构实验室
Computer Architecture Laboratory
of Zhejiang University

徐天宇
周会 2021.12.15

```
$ sudo iptables -t raw -I PREROUTING -p udp --dport 4321 --dst 192.168.254.1 -j DROP
$ sudo ethtool -X eth2 weight 1
$ watch 'ethtool -S eth2|grep rx'
    rx_packets:      12.2m/s
    rx-0.rx_packets: 1.4m/s
    rx-1.rx_packets:   0/s
    ...
```

将网卡上所有数据包引向0 RX队列

```
$ sudo ethtool -X eth2 weight 1 1 1 1
$ watch 'ethtool -S eth2|grep rx'
    rx_packets:      12.1m/s
    rx-0.rx_packets: 477.8k/s
    rx-1.rx_packets: 447.5k/s
    rx-2.rx_packets: 482.6k/s
    rx-3.rx_packets: 455.9k/s
```

将网卡上所有数据包分散到4个RX队列

假设增加多个核心不会进一步地造成性能的下降，处理数据包的核心也要多达 20 个才能达到Line Rate

不只是网络，当CPU处理数据的速度跟不上Line Rate的的时候，Kernel Bypass都是必要的

C10K & C10M

C10K问题：concurrent 10000 connection，单机 10000 个并发连接问题

WHY C10K?

- 最早由 Dan Kegel 在 1999 年提出。那时的服务器还只是 32 位系统，运行着 Linux 2.2 版本（后来又升级到了 2.4 和 2.6，而 2.6 才支持 x86_64），只配置了很少的内存（2GB）和千兆网卡
- 从资源上来说，对 2GB 内存和千兆网卡的服务器来说，同时处理 10000 个请求，只要每个请求处理占用不到 200KB（2GB/10000）的内存和 100Kbit（1000Mbit/10000）的网络带宽就可以。所以，物理资源是足够的，接下来自然是软件的问题，特别是网络的 I/O 模型问题。
- 在 C10K 以前，Linux 中网络处理都用同步阻塞的方式，也就是每个请求都分配一个进程或者线程。请求数只有 100 个时，这种方式自然没问题，但增加到 10000 个请求时，10000 个进程或线程的调度、上下文切换乃至它们占用的内存，都会成为瓶颈。

C10K & C10M

C10K问题：单机 10000 个并发连接问题

Two Questions

- 怎样在一个线程内处理多个请求，也就是要在一个线程内响应多个网络 I/O。以前的同步阻塞方式下，一个线程只能处理一个请求，到这里不再适用，是不是可以用非阻塞 I/O 或者异步 I/O 来处理多个网络请求呢？

- 怎么更节省资源地处理客户请求，也就是要用更少的线程来服务这些请求。是不是可以继续用原来的 100 个或者更少的线程，来服务现在的 10000 个请求呢？。

C10K & C10M

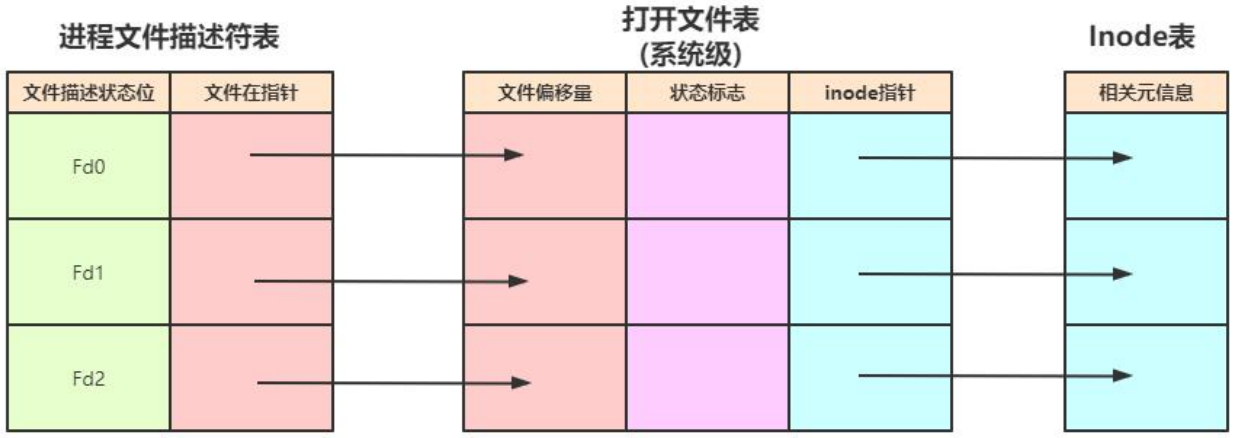
Solution

非阻塞 I/O 的解决思路—— I/O 多路复用（I/O Multiplexing）

I/O 事件通知的方式：水平触发和边缘触发

水平触发：**只要文件描述符可以非阻塞地执行 I/O ，就会触发通知**。也就是说，应用程序可以随时检查文件描述符的状态，然后再根据状态，进行 I/O 操作。

边缘触发：**只有在文件描述符的状态发生改变（也就是 I/O 请求达到）时，才发送一次通知**。这时候，应用程序需要尽可能多地执行 I/O，直到无法继续读写，才可以停止。如果 I/O 没执行完，或者因为某种原因没来得及处理，那么这次通知也就丢失了。



C10K & C10M

Solution

非阻塞 I/O 的解决思路—— I/O 多路复用（I/O Multiplexing）

水平触发解决思路

select 和 poll 需要从文件描述符列表中，找出哪些可以执行 I/O ，然后进行真正的网络 I/O 读写。由于 I/O 是非阻塞的，一个线程中就可以同时监控一批套接字的文件描述符，这样就达到了单线程处理多请求的目的。

优点：

- 对应用程序比较友好， API 非常简单。

缺点：

- 应用软件使用 select 和 poll 时，需要对这些文件描述符列表进行**轮询**，请求数多的时候就会比较耗时。
- select 使用**固定长度的位相量**，表示文件描述符的集合，因此会有最大描述符数量的限制。比如，在 32 位系统中，默认限制是 1024。并且，在 select 内部，检查套接字状态是用**轮询**的方法，再加上应用软件使用时的**轮询**，耗时 O(n^2) 。
- poll 改进了 select 的表示方法，换成了一个**没有固定长度的数组**，这样就没有了最大描述符数量的限制（当然还会受到系统文件描述符限制）。但应用程序在使用 poll 时，同样需要对文件描述符列表进行**轮询**，这样，处理耗时跟描述符数量就是 O(N) 的关系。
- 此外，应用程序每次调用 select 和 poll 时，还需要把文件描述符的集合，从**用户空间传入内核空间**，由**内核修改后，再传出到用户空间**中。

C10K & C10M

Solution

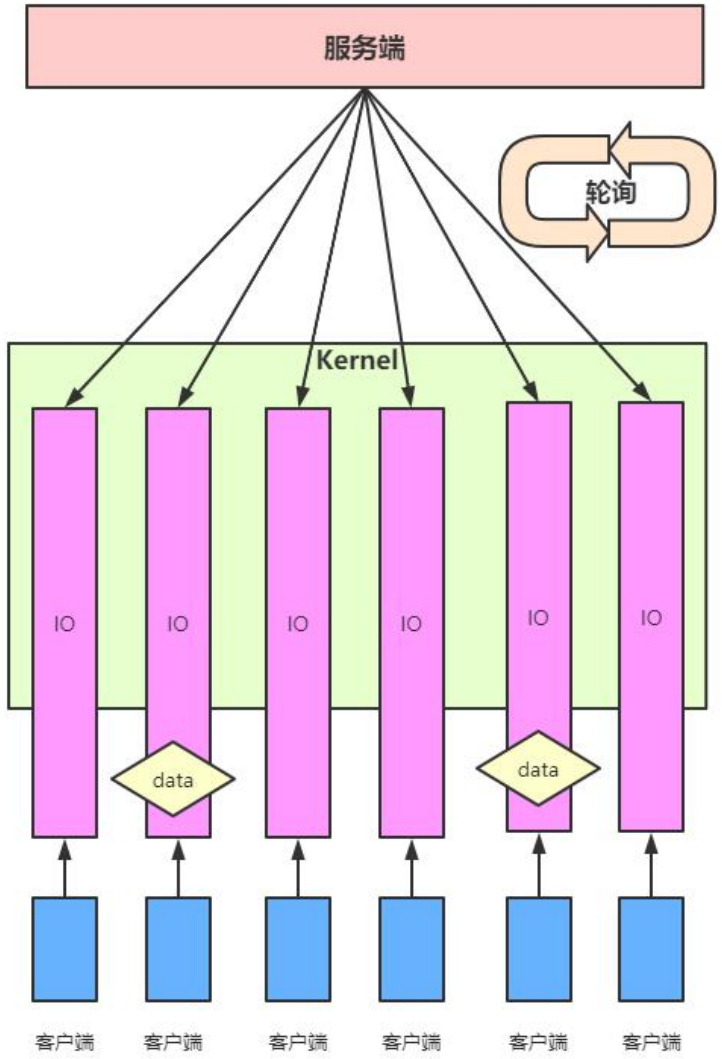
非阻塞 I/O 的解决思路—— I/O 多路复用（I/O Multiplexing）

边缘触发解决思路

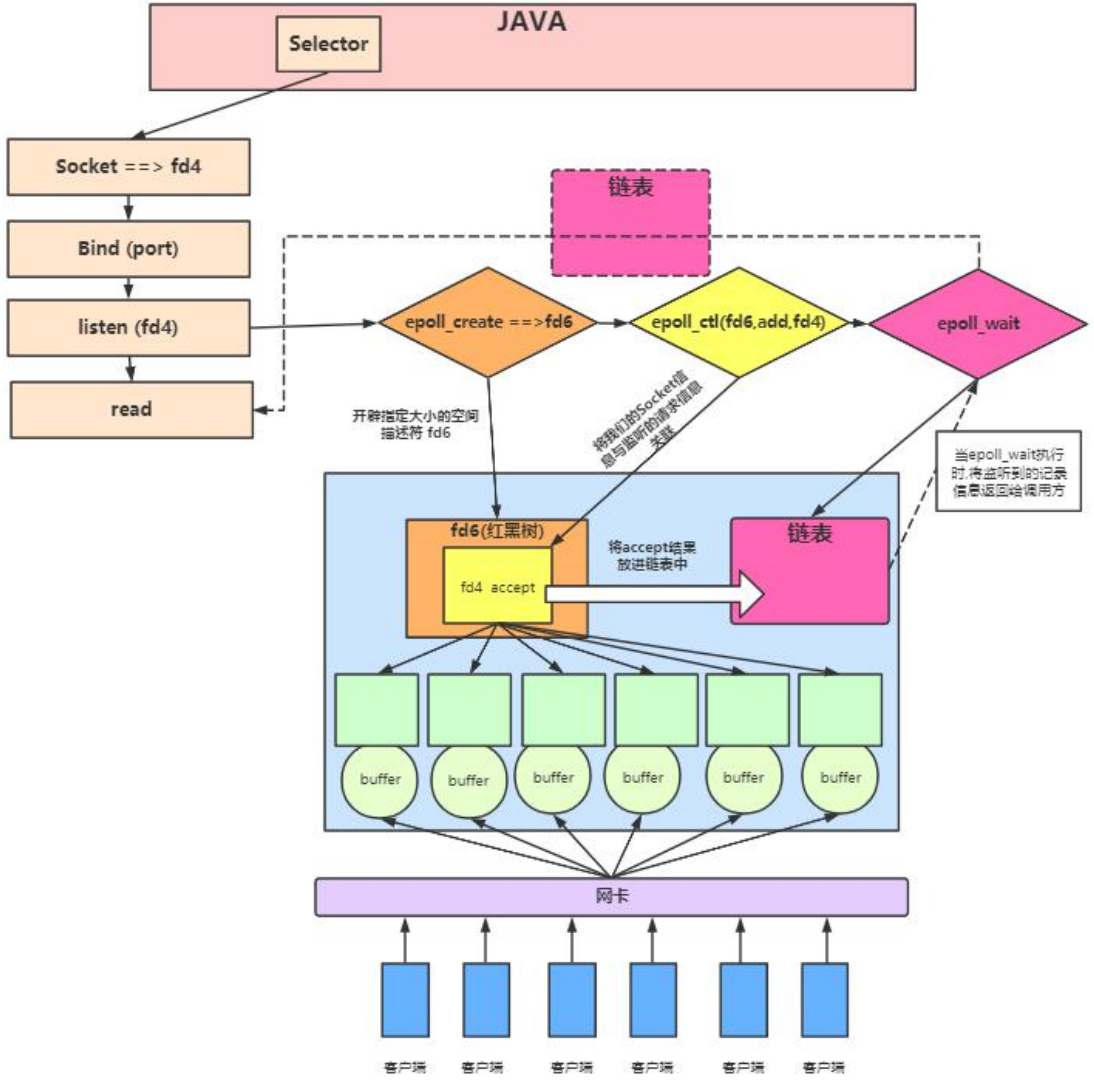
使用epoll

- epoll 使用**红黑树**在内核中管理文件描述符的集合，不需要应用程序在每次操作时都传入、传出这个集合。
- epoll 使用**事件驱动**的机制，只关注有 I/O 事件发生的文件描述符，不需要轮询扫描整个集合。

C10K & C10M



poll/select



java 中的epoll

C10K & C10M

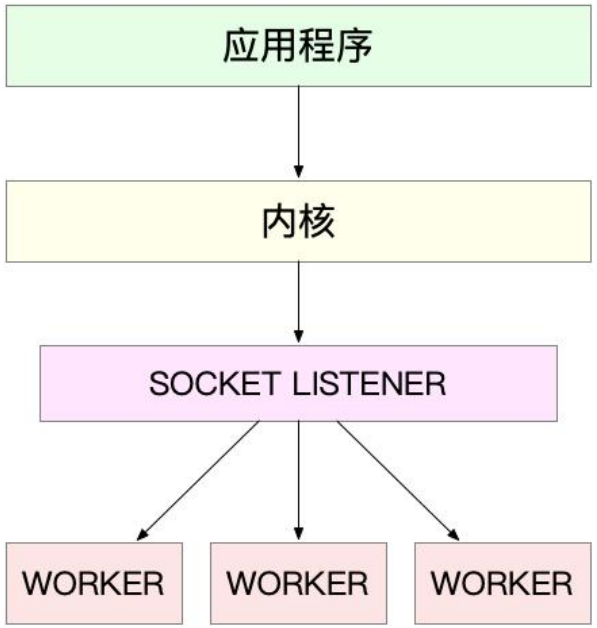
Solution

异步 I/O 的解决思路—— Asynchronous I/O，简称为 AIO

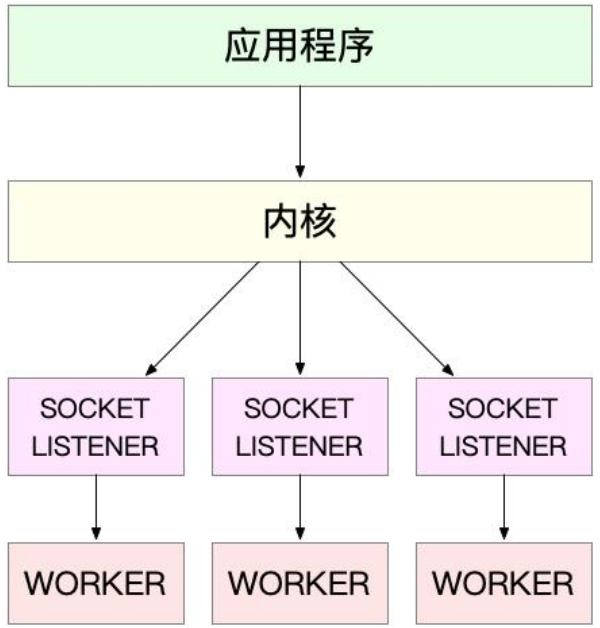
异步 I/O 允许应用程序同时发起很多 I/O 操作，而不用等待这些操作完成。而在 I/O 完成后，系统会用事件通知（比如信号或者回调函数）的方式，告诉应用程序。这时，应用程序才会去查询 I/O 操作的结果

工作模型

- 惊群问题-->锁竞争



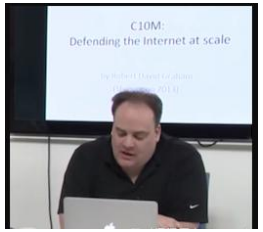
- 主进程执行 bind() + listen() 后，创建多个子进程；
- 然后，在每个子进程中，都通过 accept() 或 epoll_wait() ，来处理相同的套接字



- 所有的进程都监听相同的接口，并且开启 SO_REUSEPORT 选项，由内核负责将请求负载均衡到这些监听进程中去

C10K & C10M

Robert Graham Shmoocon 2013



C10M问题： 8 核 CPU、64G 内存，在 10GBPS 的网络上保持 1000万 的并发连接

物理资源:1000 万个请求需要大量的系统资源。比如，

- 假设每个请求需要 16KB 内存的话，那么总共就需要大约 150 GB 内存。
- 而从带宽上来说，假设只有 20% 活跃连接，即使每个连接只需要 1KB/s 的吞吐量，总共也需要 16 Gb/s 的吞吐量。千兆网卡显然满足不了这么大的吞吐量，所以还需要配置十万兆网卡，或者基于多网卡 Bonding 承载更大的吞吐量。

软件资源

- 大量的连接也会占用大量的**软件资源**，比如文件描述符的数量、连接状态的跟踪（CONNTRACK）、网络协议栈的缓存大小（比如套接字读写缓存、TCP 读写缓存）等等。
- 最后，大量请求带来的**中断处理**，也会带来非常高的处理成本。这样，就需要多队列网卡、中断负载均衡、CPU 绑定、RPS/RFS（软中断负载均衡到多个 CPU 核上），以及将网络包的处理卸载（Offload）到网络设备（如 TSO/GSO、LRO/GRO、VXLAN OFFLOAD）等各种硬件和软件的优化。
- C10M 的**解决方法**，本质上还是构建在 epoll 的非阻塞 I/O 模型上。只不过，除了 I/O 模型之外，还需要从应用程序到 Linux 内核、再到 CPU、内存和网络等各个层次的深度优化，有的解决方案还需要借助硬件，来卸载那些原来通过软件处理的大量功能。

C10K & C10M

Robert Graham Shmoocon 2013

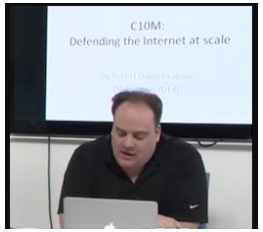
C10M问题： 8 核 CPU、64G 内存，在 10GBPS 的网络上保持 1000万 的并发连接

Robert Graham——“Unix的设计初衷并不是一般的服务器操作系统，而是电话网络的控制系统。由于是实际传送数据的电话网络，所以在控制层和数据层之间有明确的界限。问题是我们现在根本不应该使用Unix服务器作为数据层的一部分。正如设计只运行一个应用程序的服务器内核，肯定和设计多用户的服务器内核是不同的。“

Robert Graham——**关键要理解内核不是解决办法，内核是问题所在。**

C10K & C10M

Robert Graham Shmoocon 2013



C10M问题： 8 核 CPU、64G 内存，在 10GBPS 的网络上保持 1000万 的并发连接

挑战不在硬件而在于软件

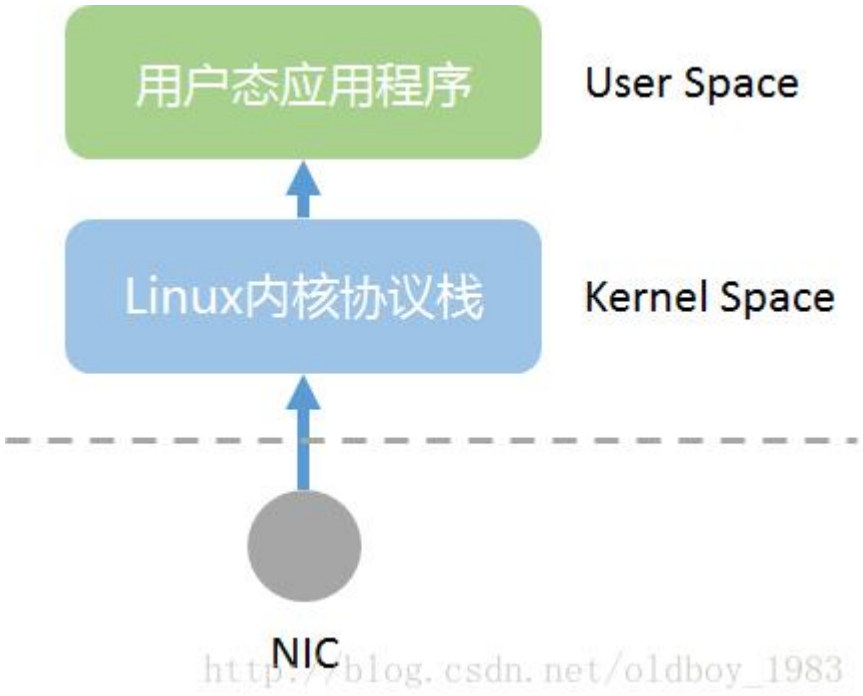
- 最初的设计是让Unix成为一个电话网络的控制系统，而不是成为一个服务器操作系统。对于控制系统而言，针对的主要目标是用户和任务，而并没有针对作为协助功能的数据处理做特别设计，也就是既没有所谓的**快速路径、慢速路径**，也没有各种数据服务处理的优先级差别。
- 其次：传统的CPU，因为只有一个核，操作系统代码以多线程或多任务的形式来提升整体性能。而现在，4核、8核、32核、64核和100核，都已经是真实存在的CPU芯片，如何**提高多核的性能可扩展性**，是一个必须面对的问题。比如让同一任务分割在多个核心上执行，以**避免CPU的空闲浪费**，当然，这里面要解决的技术点有任务分割、任务同步和异步等。
- 再次：**核心缓存大小与内存速度**是一个关键问题。现在，内存已经变得非常的便宜，随便一台普通的笔记本电脑，内存至少也就是8G以上，高端服务器的内存上24G那是相当的平常。但是，内存的访问速度仍然很慢，CPU访问一次内存需要约60~100纳秒，相比很久以前的内存访问速度，这基本没有增长多少。对于在一个带有1GHZ主频CPU的电脑硬件里，如果要实现10M性能，那么平均每一个包只有100纳秒，如果存在两次CPU访问内存，那么10M性能就达不到了。核心缓存，也就是CPU L1/L2/LL Cache，虽然访问速度会快些，但大小仍然不够。

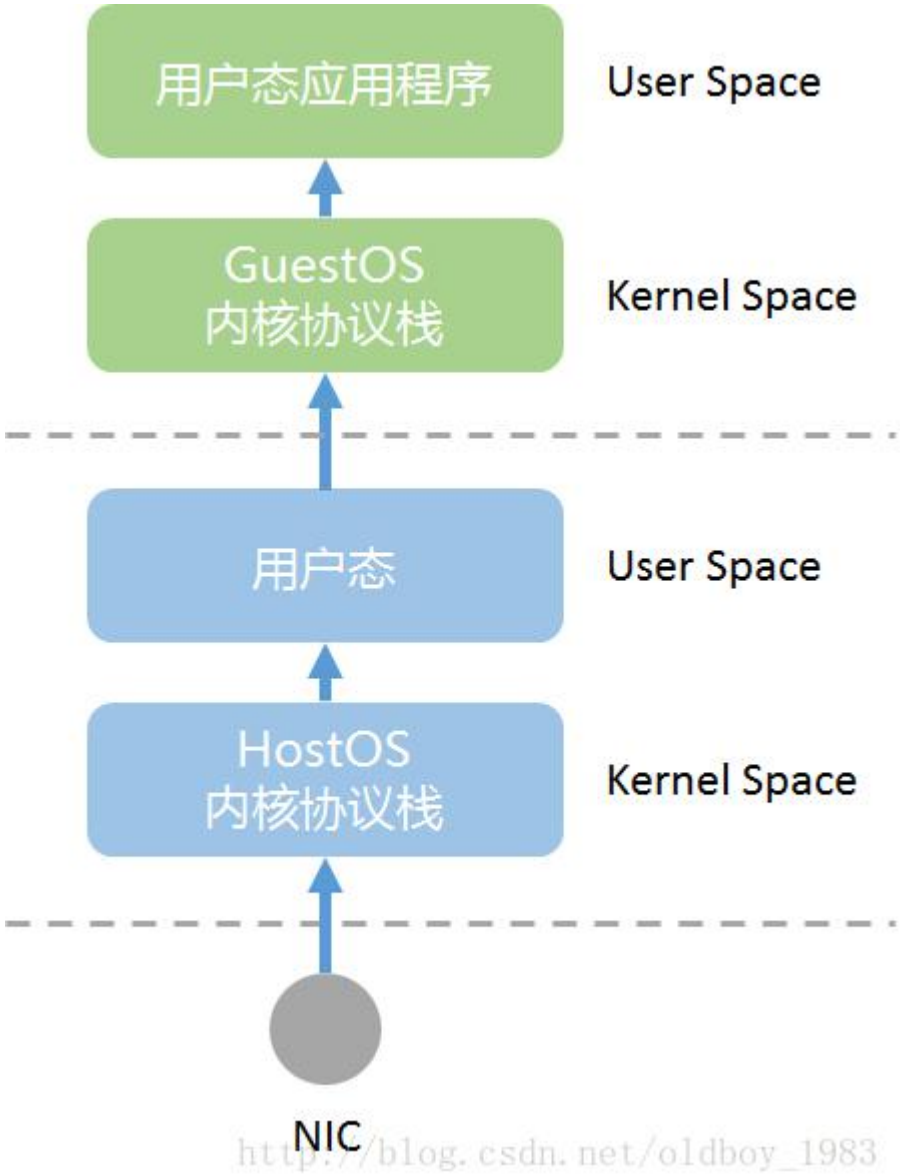
Kernel Bypass

- Packet_mmap:PACKET_MMAP在内核空间中分配一块内核缓冲区，然后用户空间程序调用mmap映射到用户空间。将接收到的数据包拷贝到那块内核缓冲区中，这样用户空间的程序就可以直接读到捕获的数据包，从而减少了数据从内核copy到用户的性能消耗。严格来讲并不属于内核旁路技术。
- PF_RING:数据包不经过内核网络协议栈。
- Snabbswitch:在用户空间实现硬件驱动。
- DPDK：用户态进程轮询。
- RDMA：数据从一台计算机传输到另一台计算机时，无需双方操作系统介入。
- XDP：网络数据包进入内核协议栈之前就开始处理。

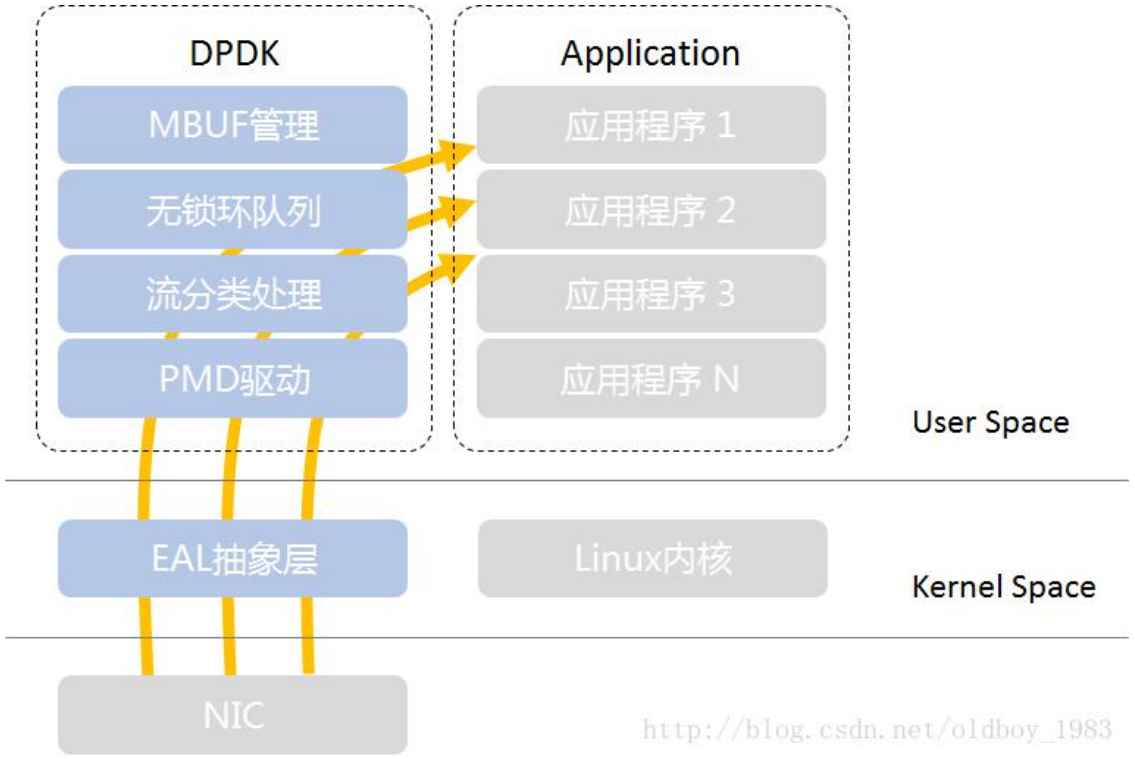
DPDK

- DPDK——Intel Data Plane Development Kit





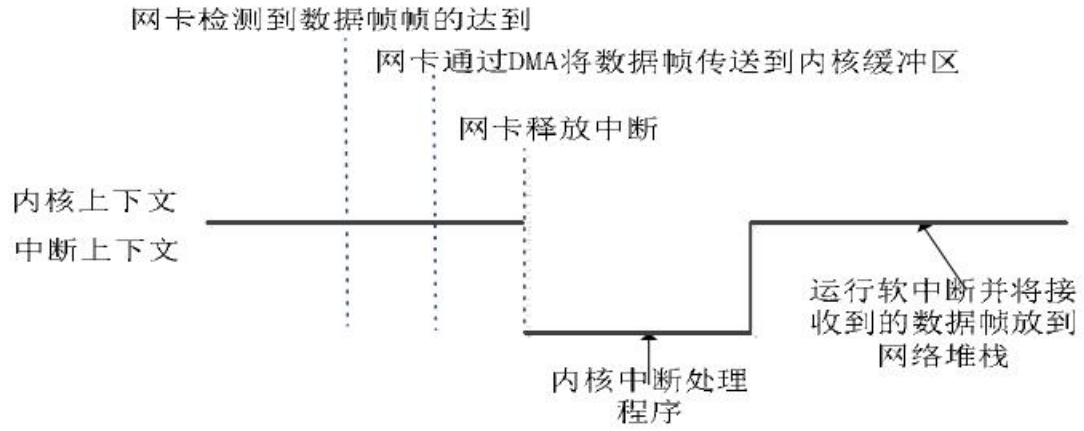
DPDK



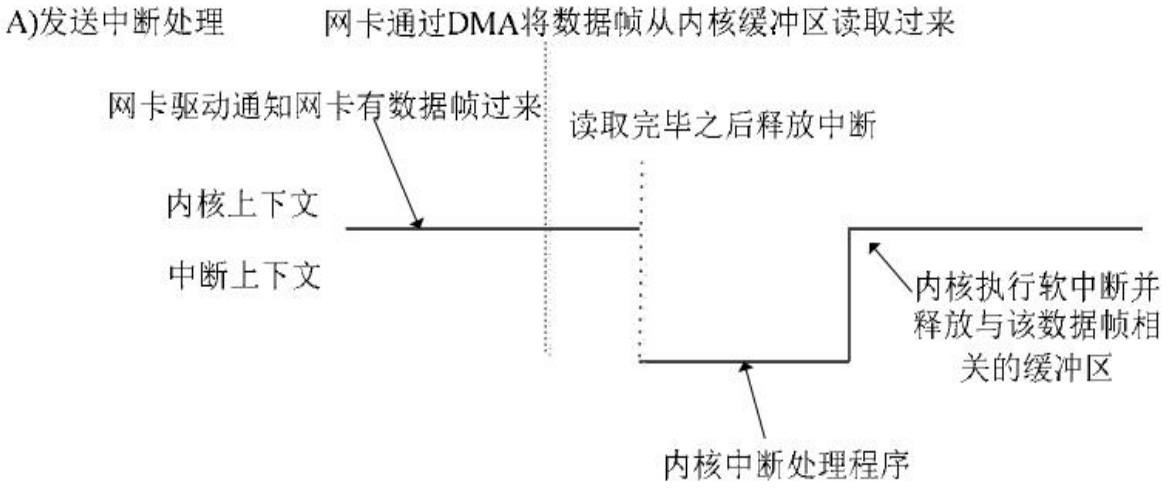
- PMD：Poll Mode Driver，轮询模式驱动，通过非中断，以及数据帧进出应用缓冲区内存的零拷贝机制，提高发送/接受数据帧的效率
- 流分类：Flow Classification，为N元组匹配和LPM（最长前缀匹配）提供优化的查找算法
- 环队列：Ring Queue，针对单个或多个数据包生产者、单个数据包消费者的出入队列提供无锁机制，有效减少系统开销
- MBUF缓冲区管理：分配内存创建缓冲区，并通过建立MBUF对象，封装实际数据帧，供应用程序使用
- EAL：Environment Abstract Layer，环境抽象（适配）层，PMD初始化、CPU内核和DPDK线程配置/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

DPDK——PMD

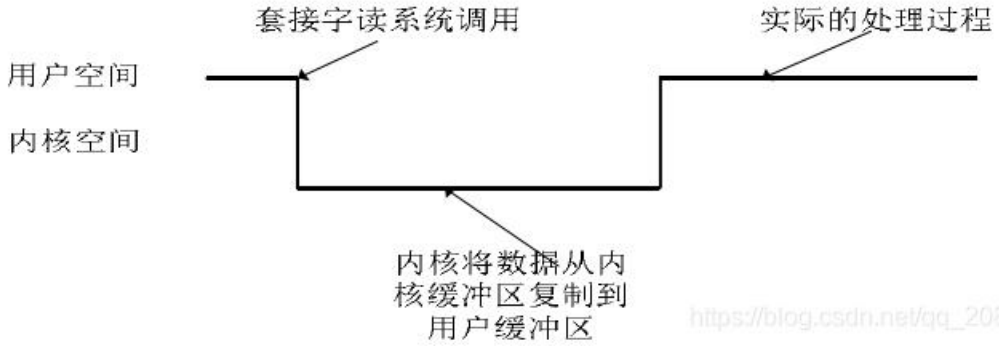
A)接收中断处理



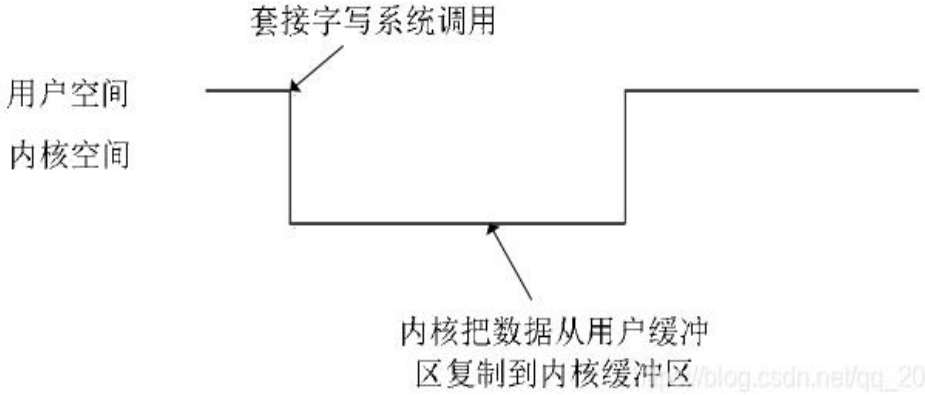
A)发送中断处理



B)接收系统调用



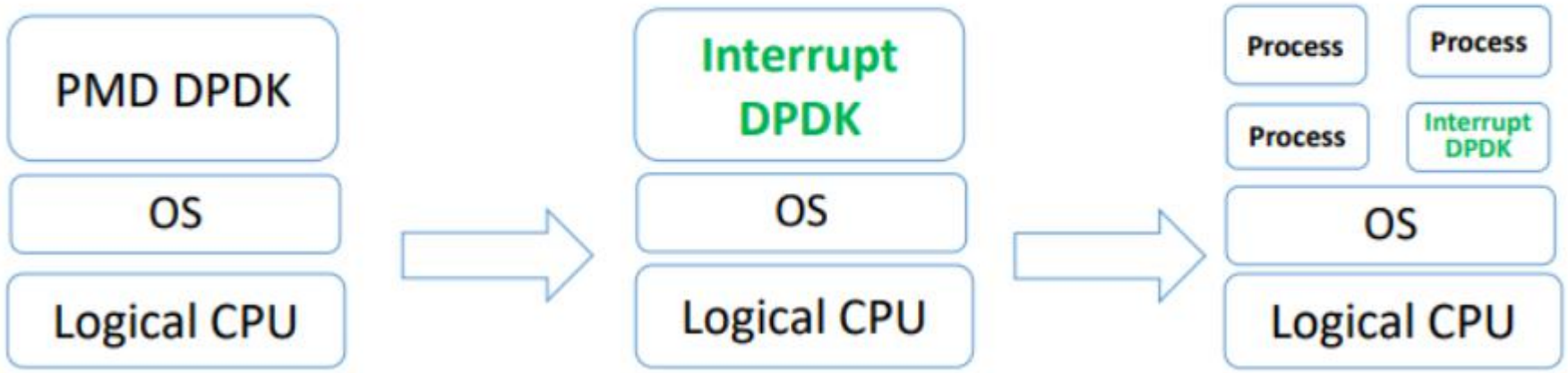
B)发送系统调用



DPDK——PMD

DPDK的UIO驱动屏蔽了硬件发出中断，然后在用户态采用主动轮询的方式。

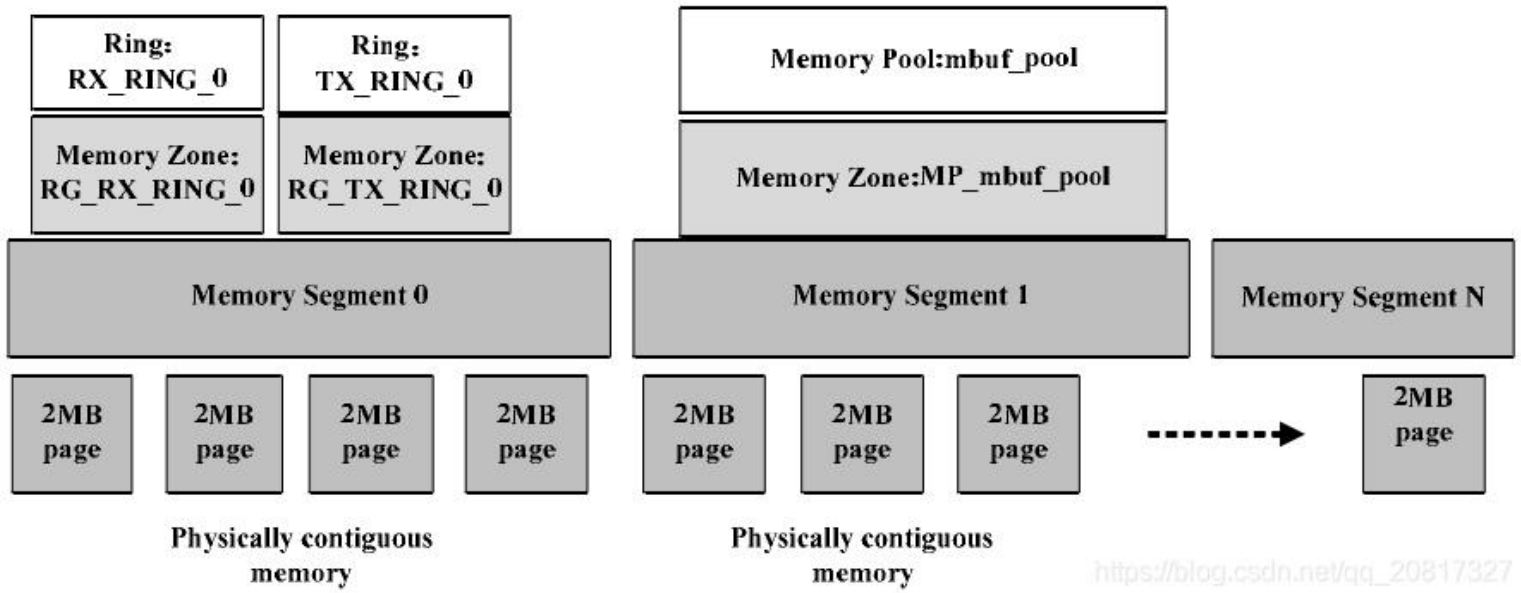
Interrupt DPDK:



没包可处理时进入睡眠，改为中断通知。并且可以和其他进程共享同个CPU Core，但是DPDK进程会有更高调度优先级

DPDK——大页内存

Linux操作系统通过**查找TLB**来实现快速的**虚拟地址到物理地址**的转化。由于TLB是一块高速缓冲cache，容量比较小，容易发生没有命中。当没有命中的时候，会触发一个中断，然后会访问内存来刷新页表，这样会造成比较大的时延，降低性能。Linux操作系统的页大小只有**4K**，所以当应用程序占用的内存比较大的时候，会需要较多的页表，开销比较大，而且容易造成未命中。相比于linux系统的4KB页，Intel DPDK缓冲区管理库提供了Hugepage大页内存，大小有**2MB和1GB**页面两种，可以得到明显性能的提升，因为采用大页内存的话，可以需要更少的页，从而需要更少的TLB，这样就减少了虚拟页地址到物理页地址的转换时间。

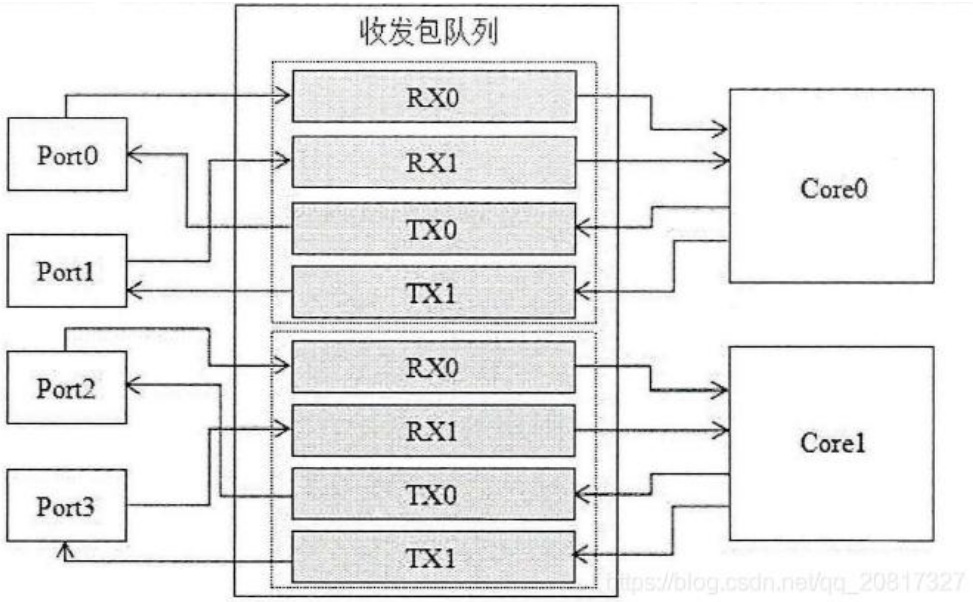


DPDK——CPU affinity

随着核心的数量越来越多，为了提高程序工作的效率必须使用多线程。但是随着CPU的核心的数目的增长，Linux的核心间的调度和共享内存争用会严重影响性能。利用Intel DPDK的CPU affinity可以将各个线程绑定到不同的cpu，可以省去来回反复调度带来的性能上的消耗。

多核轮询模式：多核轮询模式有两种，分别是**IO独占式**和**流水线式**。

IO独占式是指每个核独立完成数据包的接收、处理和发送过程，核之间相互独立，其优点是其中一个核出现问题时不影响其他核的数据收发。**流水线式**则采用多核合作的方式处理数据包，数据包的接收、处理和发送由不同的核完成。流水线式适合**面向流**的数据处理，其优点是可对数据包**按照接收的顺序**有序进行处理，缺点是当某个环境（例如接收）所涉及的核出现阻塞，则会造成收发中断。



DPDK——无锁环形缓存管理

内存池缓存区的申请和释放采用的是生产者-消费者模式无锁缓存队列进行管理，避免队列中锁的开销，在缓存区的使用过程中提高了缓冲区申请释放的效率