5. ring library

Ring 可以管理队列，rte\_ring管理一个有限长度的链表，有如下特性：

1. FIFO
2. 长度固定，指针存在一个数组（表）中
3. 无锁队列
4. 同时支持单消费者和多消费出队列
5. 同时支持单生产者和多生产者进队列
6. 批量出列，可以从队列中取指定数目的元素，如果不成功则不做操作
7. 批量入队，可以将指定数目的元素一起插入到队列中，如果不成功则不做操作
8. Burst出列，如果无法从队列中取到指定数目的元素，则取尽量多的元素出队列
9. Burst入队，如果无法将指定数目的元素入队列，则能入多少是多少。

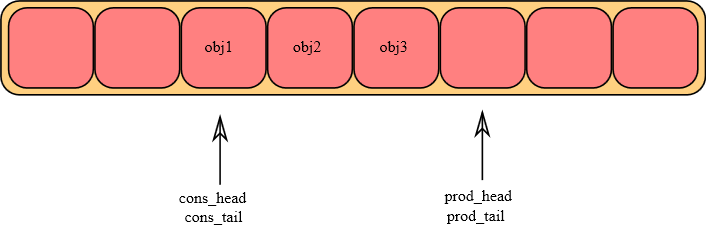
Ring通过一个数据结构来管理一个有限长度的链表，好处有：

1. 速度快：只需要一个compare&Swap的指令
2. 比一个完全的无所队列简单
3. 适用于批量进/出队列的操作。因为指针是存放在一个table中，所以批量出队列时，并不会产生大量的cache miss，批量入队列的效率和单独一个元素入队列的效率差异不大。

缺点有：

1. 链表长度固定
2. 由于ring中有存放指向链表元素指针的数组，当ring多了之后，比较耗费内存。

Ring的一个简单的示意图如下，其中cons\_head、cons\_tail、prod\_head和prod\_tail分别表示消费队列头指针，消费队列尾指针，生产队列头指针，生产队列尾指针。



5.1 ring在freebsd中实现参考

FreeBSD8.0中实现了类似的无锁队列，具体请参考freebsd中的bufring.h和bufring.c

5.2 linux中的无锁队列

参考文档Linux Lockless Ring Buffer Design

5.3 其他特征

5.3.1 名字

每个ring由唯一的名字标识，无法创建名字相同的两个ring

5.3.2 阀值

每个ring有一个标识ring快满了的阀值，一旦进队列的数目超过了该阀值，可以立即通知生产者。

该机制可用在如局域网中来不及收包时通知对端PAUSE

5.3.3 Debug

当debug开关开启时（CONFIG\_RTE\_LIBRTE\_RING\_DEBUG）,程序会记录每个ring进出队列的统计。这些统计是在每个core上单独进行的，以避免并发访问和原子操作。

5.4 用户场景

（1）用于dpdk进程间的通信

（2）用于memory pool的分配

5.5 ring的详解

这部分描述了ring上的操作过程。Ring中包含了两对head和tail游标，一对游标用于生产者，另一对用于消费者，这两对游标分别是prod\_head, prod\_tail, cons\_head 和 cons\_tail。

Ring中的指针数组是一个环形数组。

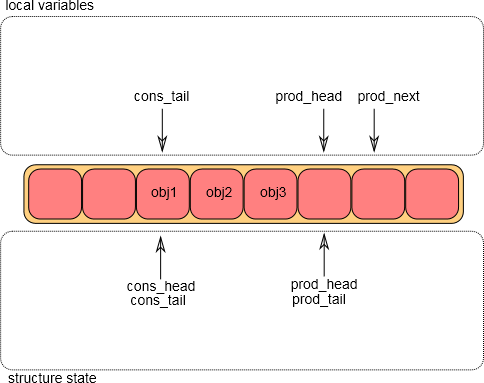
5.5.1 单生产者入队

这部分详细描述了单生产者向ring中添加元素的过程。在这个过程中，只有prod\_head和prod\_tail有改动。

初始状态是prod\_head和prod\_tail指向相同的位置。

5.5.1.1 进队列步骤一

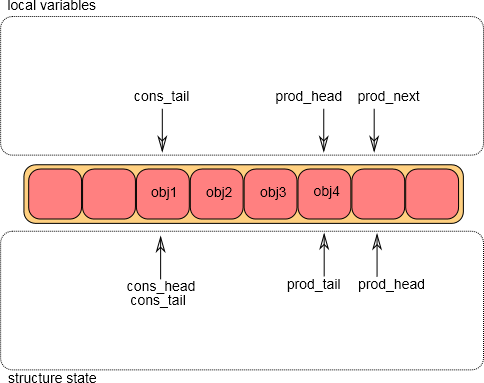
首先，将prod\_head和cons\_tail拷贝到本地，将prod\_next指向数组中的下一个元素（批量入队中则是批量入队之后的元素），如果队列空间不足（通过与cons\_tail比较），则返回错误。



5.5.1.2 进队列步骤二

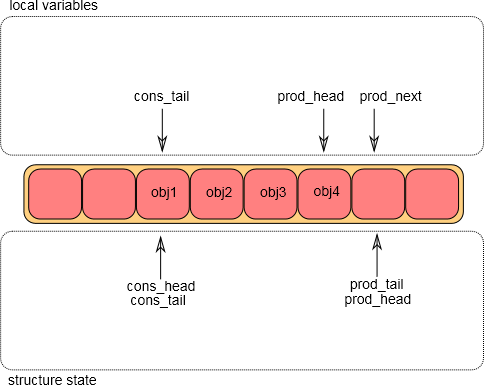
第二步是将prod\_head的值修改到prod\_next指向位置相同

然后将入队元素的指针拷贝到队列中（obj4），如图：



5.5.1.3 进队最后一步

一旦元素进队列，就将prod\_tail的值修改成与Prod\_head指向相同的位置，进队操作完成。如图：



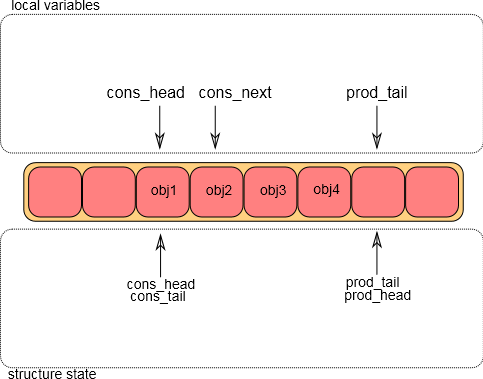
5.5.2 单消费者队列

下面展示单个元素出队列的操作。这个过程中，只修改cons\_head和cons\_tail的值，并且只有一个消费者。

Cons\_head和cons\_tail在初始状态指向相同的位置。

5.5.2.1出队列步骤一

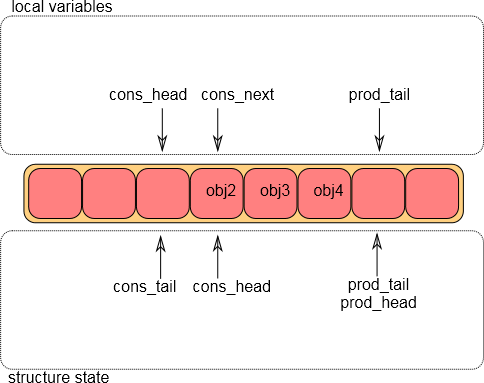
首先，cons\_head 和prod\_tail被拷贝到本地，cons\_next 变量指向下一个元素（如果是批量出队列则是指向这一批出队列的元素的下一个元素）。如果队列中没有足够的元素（与prod\_tail比较），则返回错误。



5.5.2.2 出队列步骤二

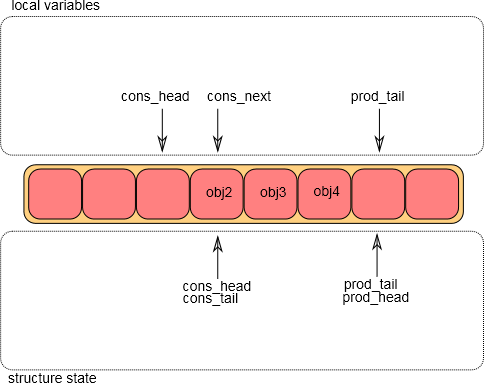
修改cons\_head的值指向cons\_next指向的位置

将元素的指针拷贝出去（obj1）



5.5.2.3 出队列最后一步

最后，将cons\_tail修改成与cons\_head指向相同的值



5.5.3多生产者队列

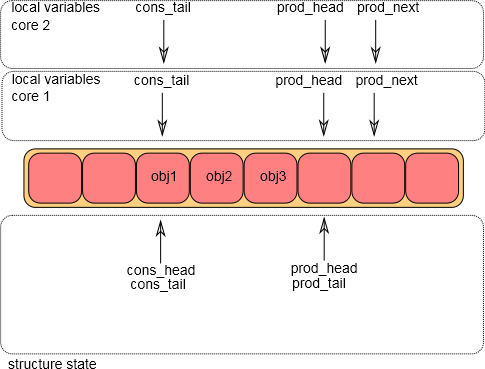
这部分主要是讲两个生产者（多生产者队列）同时入队列的操作，多生产者进队时，主要需要修改prod\_head和prod\_tail两个游标。

Prod\_head和prod\_tail在初始状态时指向相同的位置。

5.5.3.1 多生产者队列入队步骤一

在两个核上，prod\_head 和 prod\_tail（官网文档中这儿有笔误）拷贝到本地，本地变量prod\_next指向下一个元素的地址（如果是入队N个元素则指向N个元素后的地址）。

如果空间不够（与cons\_tail比较），则返回错误。

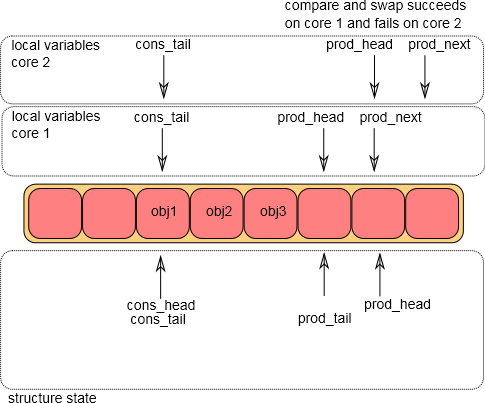


5.5.3.2 多生产者队列入队步骤二

第二步的主要工作是将prod\_head修改成与prod\_next指向相同的地址。这个操作通过compare and swap指令完成，这个操作主要有如下两步：

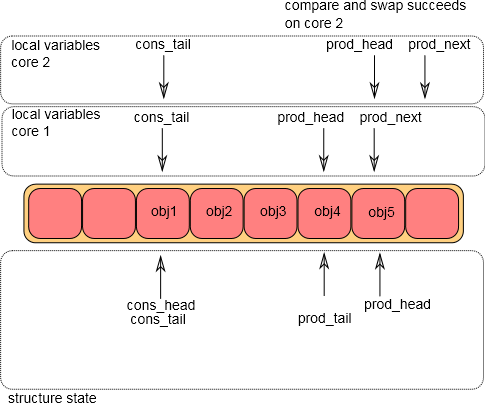
1. 如果ring->prod\_head与本地存放的prod\_head不同，则CAS操作失败，跳到步骤一重新执行
2. 否则，ring->prod\_head设置成prod\_next的值，CAS操作成功，则完成入队操作。

下图中，core1一次就完成c&s操作，core2则是第二次的时候完成。



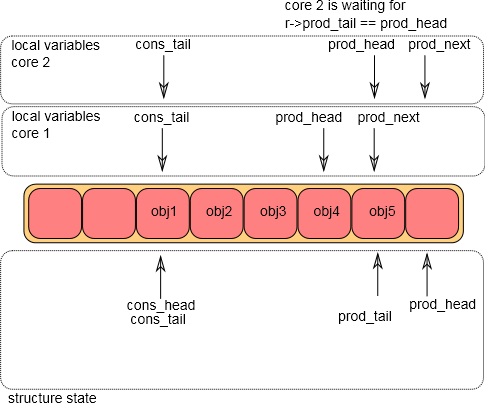
5.5.3.3 多生产者队列入队步骤三

Core1向ring中插入obj4的地址，core2插入obj5的地址。如图：



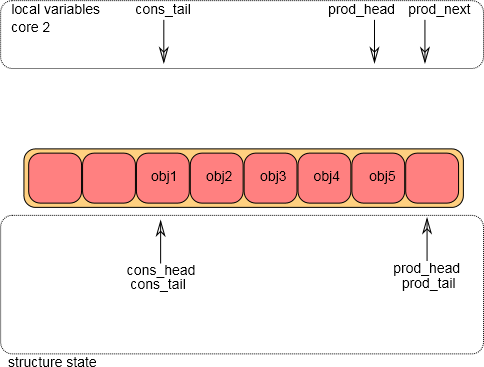
5.5.3.4 多生产者队列入队步骤四

每个core都需要更新ring->prod\_tail，只有在某个core上的本地变量prod->head与ring->prod\_tail值相同的条件下，才允许这个core更新ring->prod\_tail成该core本地存放的prod\_next的值。因此，只有core1才能更新ring->prod\_tail成prod\_next的值，如图：



5.5.3.5 多生产者队列入队最后一步

一旦core1更新ring->prod\_tail完成，core2更新ring->prod\_tail的条件也就充分了。Core2完成更新后，如图：



5.5.4 模32位索引