哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验设计报告

开课学期：2022 年秋季

课程名称：操作系统

实验名称：锁机制的应用

实验性质：课内实验

实验时间：10.22 地点：T2608

学生班级：20 级 08 班

学生学号：200210231

学生姓名：王木一

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2022 年 9 月

一、 回答问题

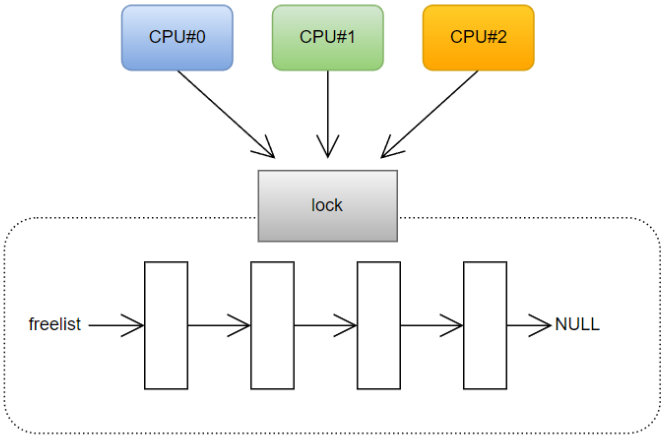
1、 内存分配器

a. 什么是内存分配器？它的作用是？

内存分配器是内核用于管理用户进程内存的机制。
作用包括初始化内存页，分配新内存空间，释放（回收）内存空间。

b. 内存分配器的数据结构是什么？它有哪些操作（函数），分别完成了什么功能？

内存分配器核心数据结构为有空闲物理页组成的链表，此链表是通过将物理内存划分为大小 4KB 的页帧，链表的每一节点都代表一个空闲页。同时使用自旋锁对链表的并发访问进行保护。



- void kinit() 初始化分配器（包括初始化自旋锁，构造空闲页链表）
- void freerange(void *pa_start, void *pa_end) 根据物理空闲内存空间（即 pa_start 和 pa_end 的范围）划分页帧，将其加入链表
- void kfree(void *pa) 释放一页内存，将其加入空闲页链表
- void *kalloc(void) 分配一页内存，从空闲页链表中取出

c. 为什么指导书提及的优化方法可以提升性能？

原有方案将所有空闲页用一个链表管理，同时使用一个自旋锁保护。当多个进程（CPU）同时访问这一链表时势必会造成锁争抢。

直觉地，为何不为每个进程（CPU）单独配置一个空闲页表呢？指导书提供的优化方案正是如此。每个进程（CPU）拥有单独的空闲链表，就不用争抢整块共同的资源，也就减少了锁的争用。只有自己对应的空闲页分配完，才会去窃取其他 CPU 的内存页，才会发生锁的争抢。

2、 磁盘缓存

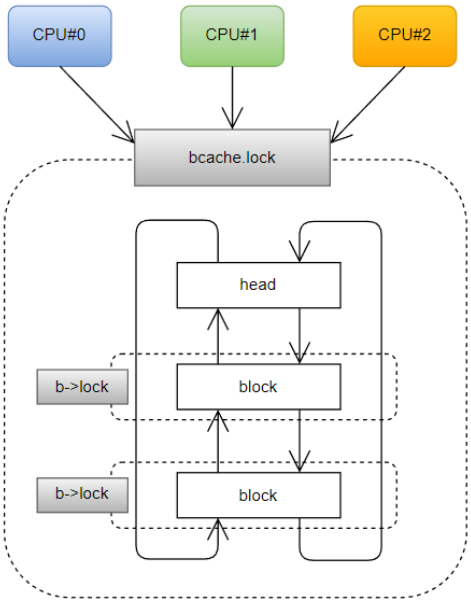
a. 什么是磁盘缓存？它的作用是？

磁盘缓存是磁盘和文件系统交互的中间层。

作用解决文件系统中磁盘和内存读取速度不匹配的问题。通过将经常访问的磁盘数据块缓存在内存中提升系统性能。

b. buf 结构体为什么有 prev 和 next 两个成员，而不是只保留其中一个？请从这样做的优点分析（提示：结合通过这两种指针遍历链表的具体场景进行思考）。

磁盘缓存的核心数据结构是一个带头结点的双向链表，所以需要同时有两个分别指向前后的指针。



双向链表中的数据块是按最近访问的时间进行排序的，即最近访问过的内存块在最前面，最早访问的数据块在最后。这样在无论是否命中，在遍历查找链表时都会大大减少时间。

我们假设进程会频繁访问某一数据块，即当前想访问的数据块在不久之前才被访问过，所以顺序遍历链表可以快速找到目标。当未命中，需要新分配一个数据块时，要从最不常访问的数据块中寻找合适的，逆序遍历链表来寻找也会快速找到目标。

c. 为什么哈希表可以提升磁盘缓存的性能？可以使用内存分配器的优化方法优化磁盘缓存吗？请说明原因。

哈希表（桶）将缓存数据块分组，就是将一个大锁分解成若干个小锁（想法和内存分配器类似）。借助需要访问数据块块号的哈希值，让不同进程访问不同的哈希桶，避免对一个大锁的争用，降低进程忙等待的概率，提升性能。（当然若有两个进程访问数据块块号的哈希值相同，不可避免地会争夺同一个哈希桶锁）

不可以，不可为每个 CPU 分配各自的磁盘缓存（即双向链表），与内存资源独占不同，磁盘缓存是共享访问的（需保证数据块的唯一）。同时磁盘缓存较大，为每个 CPU 分配缓存会造成空间浪费。

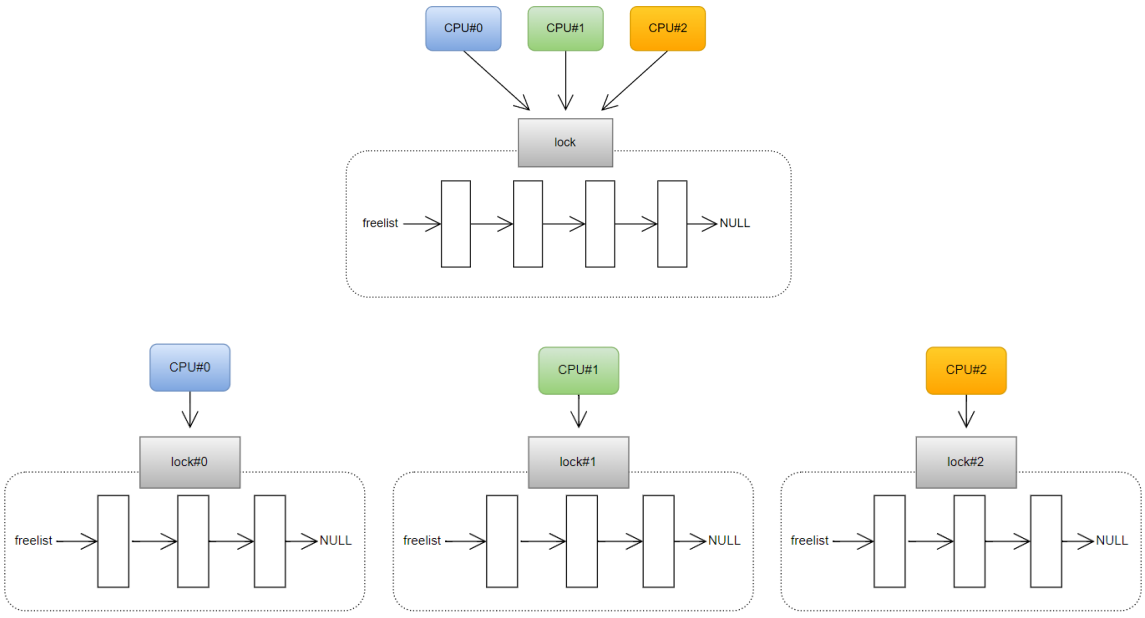
二、 实验详细设计

注意不要照搬实验指导书上的内容，请根据你自己的设计方案来填写

1. 内存分配器

1.1 数据结构更改

为每个 CPU 分配一个空闲页链表。（上图：更改前；下图：更改后）

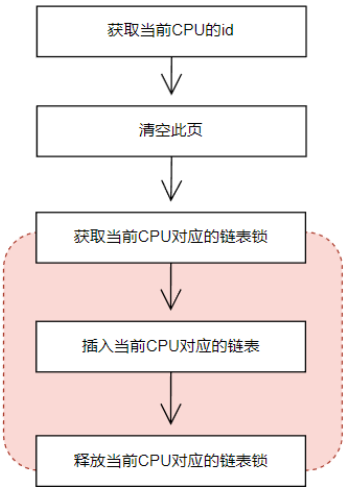


1.2 内存分配器初始化

修改 kinit()和 freepage()，初始化每个链表和每个锁。轮流将空闲页插入各个链表，使每个 CPU 所得的页数相同。

1.3 释放（回收）空闲页 kfree()

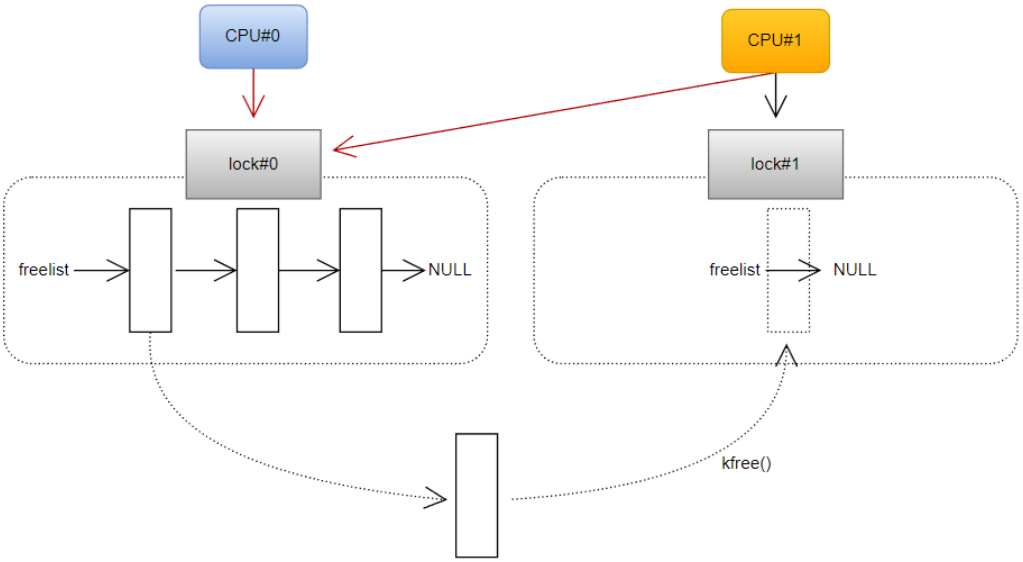
回收空闲页即对当前 CPU 对应的链表完成插入操作，流程如下图，红色区域即为临界区。



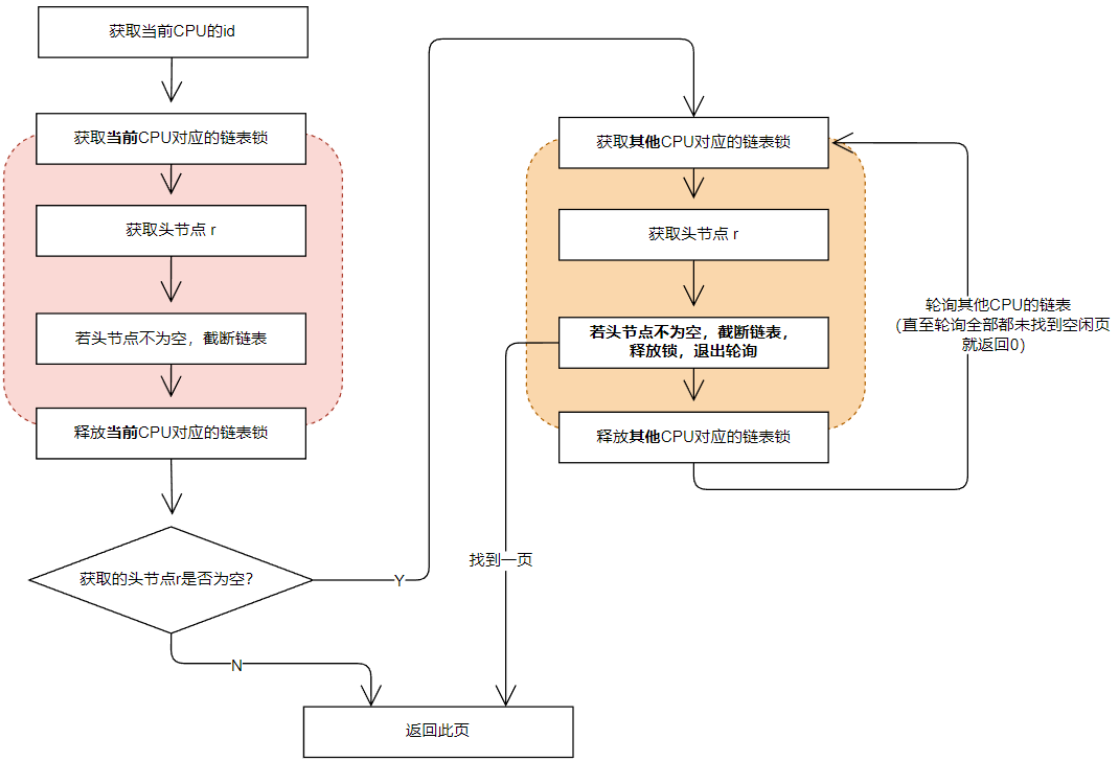
1.4 分配空闲页 kalloc()

若此时的 CPU 对应的链表不为空时，分配页表只涉及本 CPU 对应的链表操作（从链表中获取头节点，更新链表头节点）。

而当前链表为空，需要从其他 CPU 的空闲链表中“窃取”一个空闲页，从而可能会引起锁争抢。如下图：



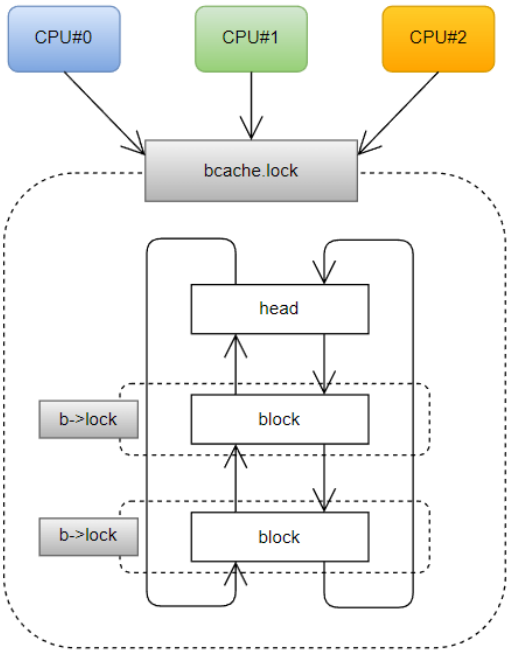
由于空闲页属于非共享资源，且 freelist 存储的是**空闲页表**，故在 kalloc()时窃取到的空闲页不用再插入自己的链表中，待释放页表时 kfree()再插入自己的页表。流程如下：

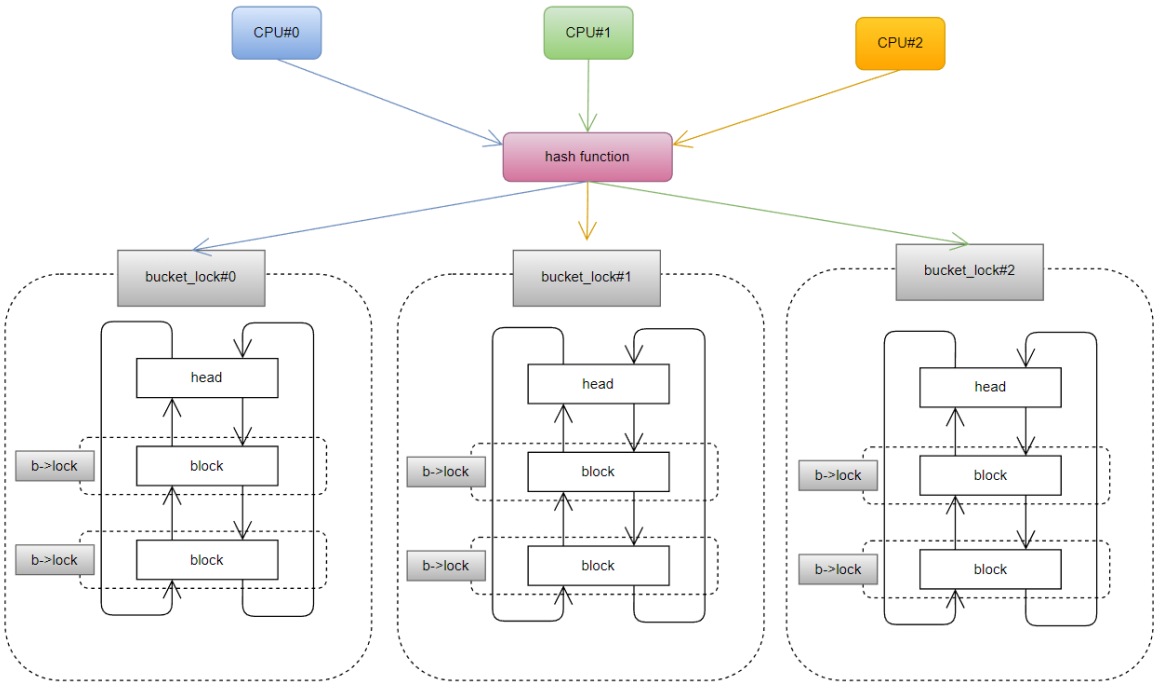


2. 磁盘缓存

2.1 数据结构更改

构造 13 个哈希桶，每个桶中都有一个存储数据块的双向链表。每次根据要访问的块号的哈希值分配到不同的桶中操作。上图：更改前；下图：更改后





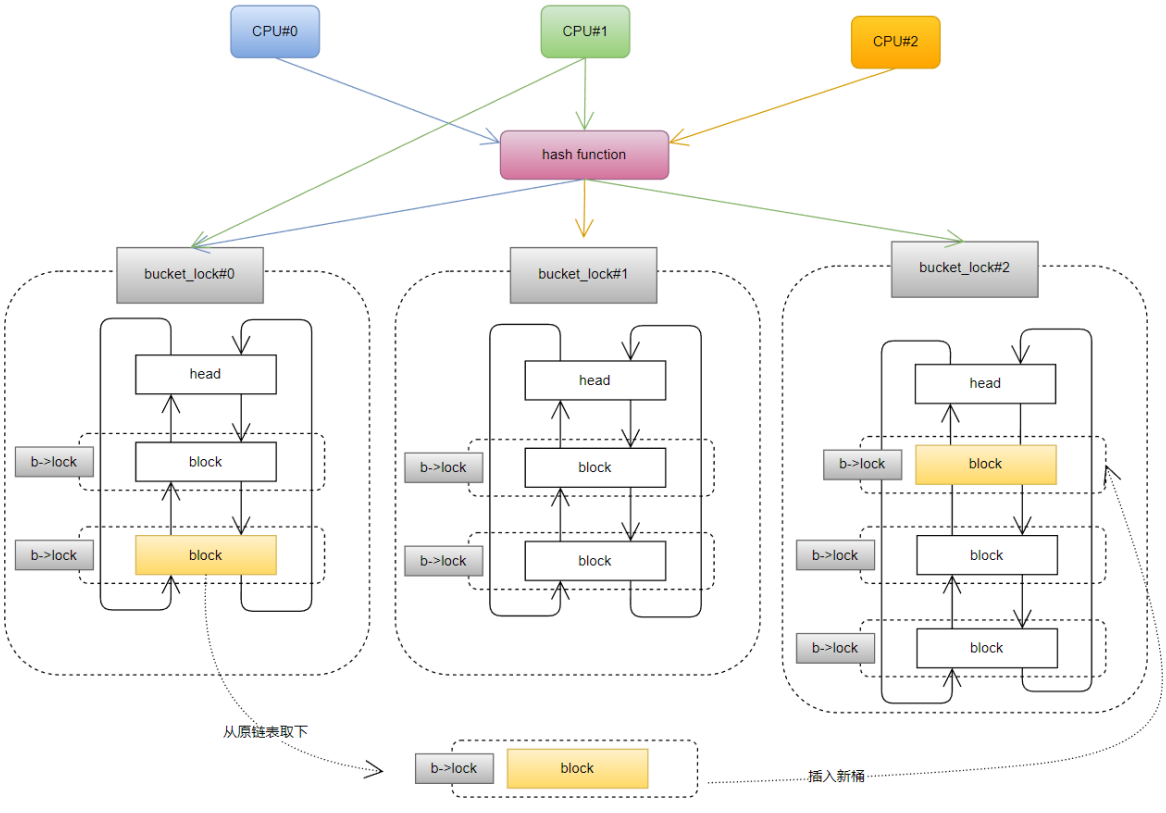
2.2 磁盘缓存初始化

修改 binit()，初始化每个双向链表和每个锁。轮流将数据块插入各个链表，使每个桶所得的块数相同。

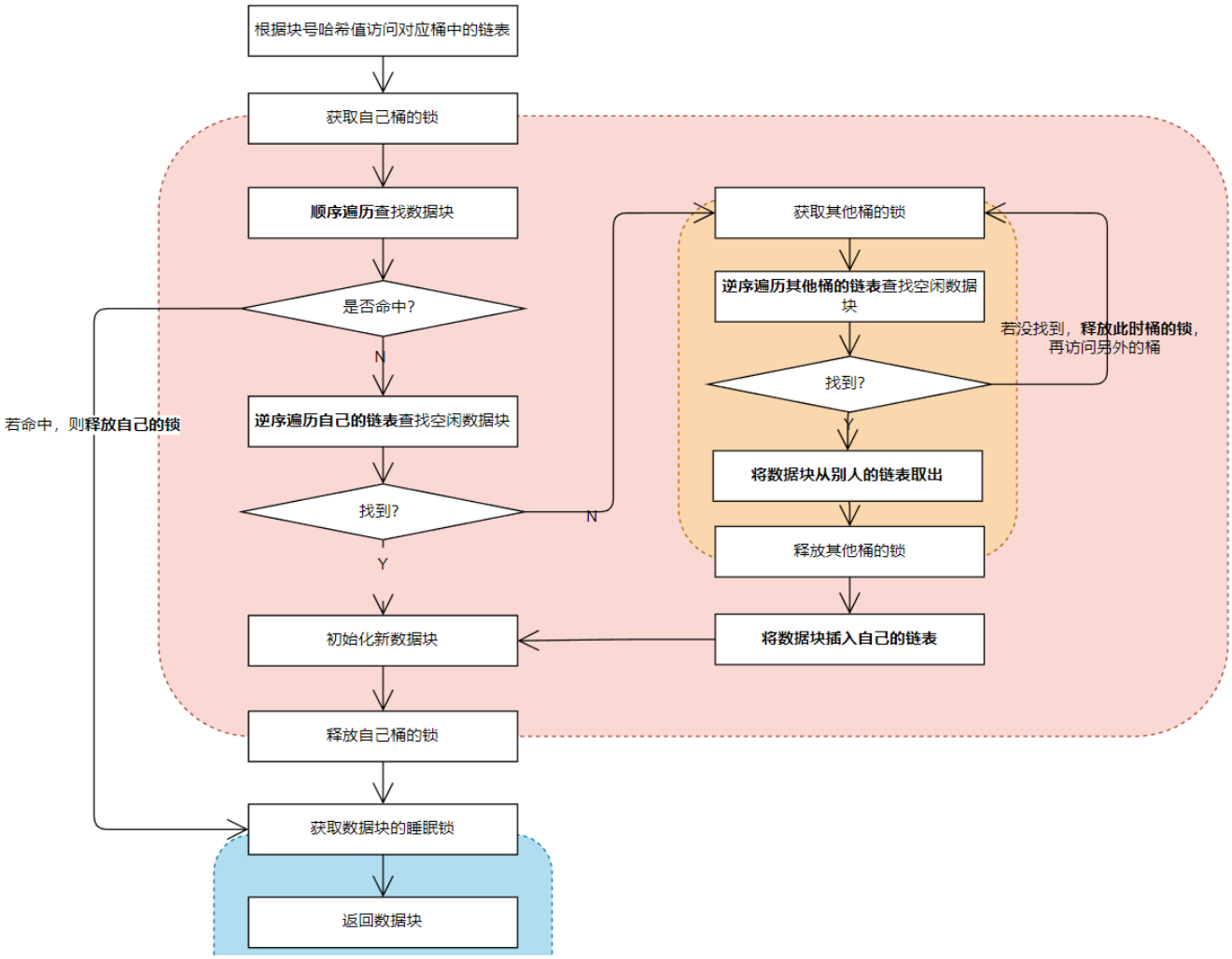
2.3 查询数据块 bget()

根据查询块号的哈希值，从对应的桶中顺序查找链表。

若命中则返回对应块，若未命中则逆序查找当前桶中的链表，寻找一个最久没用过的缓存块用于载入磁盘数据。若当前桶中都无法找到没用的缓存块，则需要去别的桶中“窃取”。如下图：



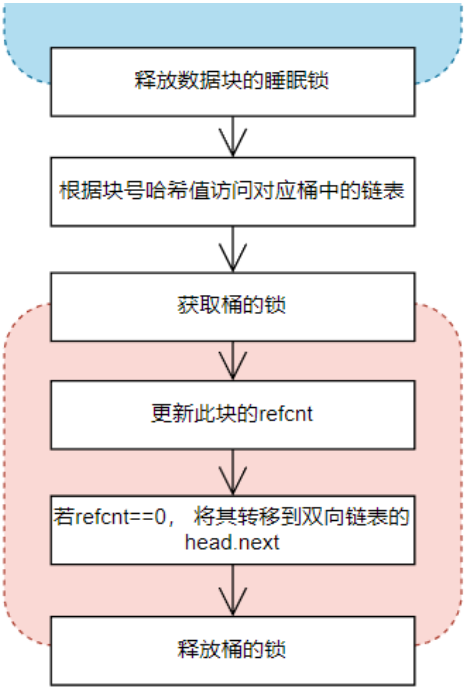
如上图，当 CPU1 在桶 2 未找到空闲块，就去桶 0 寻找，找到后将数据块从原链表取下，并将其插入自己桶的链表中（与内存分配器不同，磁盘缓存块是共享资源，同时缓存中只能由一个磁盘数据拷贝，所以窃取成功后需将数据块插入原来的链表）流程如下：



与内存分配器不同，因为窃取到后需要插回去，所以 bget()中可能会获得两个锁。

2.4 释放数据块 brelse()

释放此块的睡眠锁。根据块号的哈希值，找到要操作的桶的链表，获取锁，更新块的数据 refcnt，若没有其他进程等待使用就将此块的位置转移到 head 之后，释放锁。流程如下：



2.5 其他操作

将其他涉及 bache 锁的地方（如 bpin()，bunpin()），更改为 bucket_lock 的操作

因为本次实验涉及代码较少，故没有贴代码，具体代码详见 kalloc.c bio.c

三、 实验结果截图

请填写

usertests

```
ALL TESTS PASSED
$
```

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ kalloctest
start test1
test1 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 12612
lock: kmem: #fetch-and-add 0 #acquire() 195495
lock: kmem: #fetch-and-add 0 #acquire() 204455
lock: kmem: #fetch-and-add 0 #acquire() 4091
lock: kmem: #fetch-and-add 0 #acquire() 4090
lock: kmem: #fetch-and-add 0 #acquire() 4090
lock: kmem: #fetch-and-add 0 #acquire() 4090
lock: kmem: #fetch-and-add 0 #acquire() 4090
lock: bcache.bucket: #fetch-and-add 0 #acquire() 2
lock: bcache.bucket: #fetch-and-add 0 #acquire() 4
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6
lock: bcache.bucket: #fetch-and-add 0 #acquire() 8
lock: bcache.bucket: #fetch-and-add 0 #acquire() 10
lock: bcache.bucket: #fetch-and-add 0 #acquire() 10
lock: bcache.bucket: #fetch-and-add 0 #acquire() 282
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6
lock: bcache.bucket: #fetch-and-add 0 #acquire() 4
lock: bcache.bucket: #fetch-and-add 0 #acquire() 2
--- top 5 contended locks:
lock: proc: #fetch-and-add 22295 #acquire() 152147
lock: virtio_disk: #fetch-and-add 7176 #acquire() 57
lock: proc: #fetch-and-add 5747 #acquire() 152226
lock: proc: #fetch-and-add 3349 #acquire() 152152
lock: pr: #fetch-and-add 1958 #acquire() 5
tot= 0
test1 OK
start test2
total free number of pages: 32496 (out of 32768)
.....
test2 OK
```

```
$ bcachetest
start test0
test0 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 154854
lock: kmem: #fetch-and-add 0 #acquire() 2041676
lock: kmem: #fetch-and-add 0 #acquire() 1856257
lock: kmem: #fetch-and-add 0 #acquire() 128958
lock: kmem: #fetch-and-add 0 #acquire() 124866
lock: kmem: #fetch-and-add 0 #acquire() 120776
lock: kmem: #fetch-and-add 0 #acquire() 116686
lock: kmem: #fetch-and-add 0 #acquire() 112596
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6180
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6178
lock: bcache.bucket: #fetch-and-add 0 #acquire() 4266
lock: bcache.bucket: #fetch-and-add 0 #acquire() 4262
lock: bcache.bucket: #fetch-and-add 0 #acquire() 2258
lock: bcache.bucket: #fetch-and-add 0 #acquire() 4256
lock: bcache.bucket: #fetch-and-add 0 #acquire() 2526
lock: bcache.bucket: #fetch-and-add 0 #acquire() 4552
lock: bcache.bucket: #fetch-and-add 0 #acquire() 5056
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6180
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6178
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6176
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6174
--- top 5 contended locks:
lock: proc: #fetch-and-add 270301 #acquire() 3841393
lock: proc: #fetch-and-add 263648 #acquire() 3841393
lock: proc: #fetch-and-add 220869 #acquire() 3841393
lock: proc: #fetch-and-add 215258 #acquire() 3841393
lock: proc: #fetch-and-add 215058 #acquire() 3841393
tot= 0
test0: OK
start test1
test1 OK
```

```
== Test running kalloctest ==
$ make qemu-gdb
(122.0s)
== Test   kalloctest: test1 ==
  kalloctest: test1: OK
== Test   kalloctest: test2 ==
  kalloctest: test2: OK
== Test kalloctest: sbrkmuch ==
$ make qemu-gdb
kalloctest: sbrkmuch: OK (11.6s)
== Test running bcachetest ==
$ make qemu-gdb
(8.4s)
== Test   bcachetest: test0 ==
  bcachetest: test0: OK
== Test   bcachetest: test1 ==
  bcachetest: test1: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (142.3s)
== Test time ==
time: OK
Score: 70/70
200210231@comp0:~/xv6-labs-2020$
```