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

实验设计报告

开课学期：2021 年秋季学期

课程名称：操作系统实验课

实验名称：第三次实验——锁

实验性质：课内实验

实验时间：10.21 地点：T2210

学生班级：1901102

学生学号：190110209

学生姓名：蒋晨阳

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2018 年 12 月

一、 回答问题

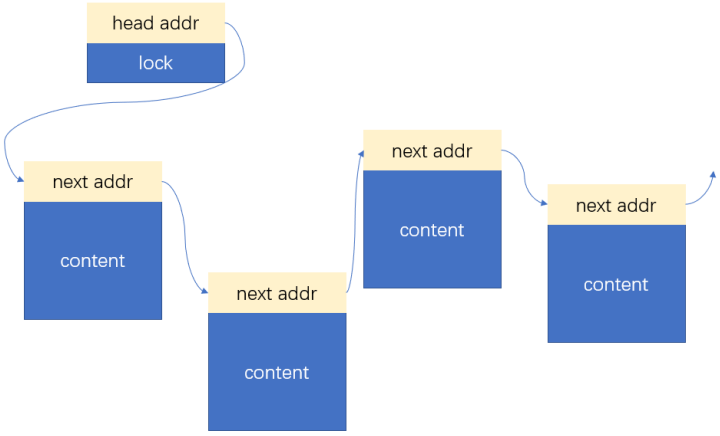
1. 内存分配器

a. 什么是内存分配器？它的作用是？

答：内存分配器是负责管理物理内存的模块。其提供了 kalloc 和 kfree 函数接口来管理物理内存，可以初始化内存，分配出当前空闲内存，回收内存。

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？

答：内存分配器的数据结构是链表，并在其上有一个配对的锁。其中黄色部分代表 run 结构体。



函数：

kinit():

初始化内存链表以及其上的锁。在本实验中分别初始化每个 cpu 所拥有的链表以及锁。

freearange(void *pa_start, void *pa_end):

指定从 pa_start 到 pa_end 的内存物理地址，按照页大小将其分割成若干页，并在页中放入垃圾数据，随后将每一页放入内存链表中。

kfree():

释放内存页。将内存页重新填入垃圾数据，并放入内存链表中。在本实验中将其分别放入对应 cpu 的链表中。

kalloc():

分配出一个内存页，并返回内存页的首地址。同时将该内存页从维护的链表中取出。

c. 为什么指导书提及的优化方法可以提升性能？

原来的程序中，不同 cpu 使用同一个内存链表，这就导致当一个 cpu 正在访问内存链表时，其余 cpu 无法访问该链表，也就无法进行内存相关操作。而如果每个 cpu 都有一个独立的内存链表，则除了窃取其他 cpu 的内存块外，cpu 之间不会出现内存链表竞争的现象。

2. 磁盘缓存

a. 什么是磁盘缓存？它的作用是？

磁盘缓存是磁盘内容在内存中的映像。cpu 对于磁盘的读写较慢，因此需要先将磁盘内容保存在缓冲区，当 cpu 需要访问磁盘时，若内存中存在相应的块，则只需要访问内存中的缓存即可，加快访问速度。

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）。

可以加快访问速度。调用 bget 时，首先使用 next 指针从队首向队尾遍历，寻找最近已使用并释放过的 block，随后使用 prev 指针，从队尾向队首遍历，寻找最久未使用

（refcnt=0）的块。调用 brelse 时，将待释放的 block 放入队首，也即最近使用过的 block 放在队首。
这样在寻找有无缓存过的块时，从队首向队尾查找，优先查找道最近缓存到的块。查找未使用的块时，从队尾向队首找，优先寻找已经很久没有使用过的块。增加命中率，减少查找次数。

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。

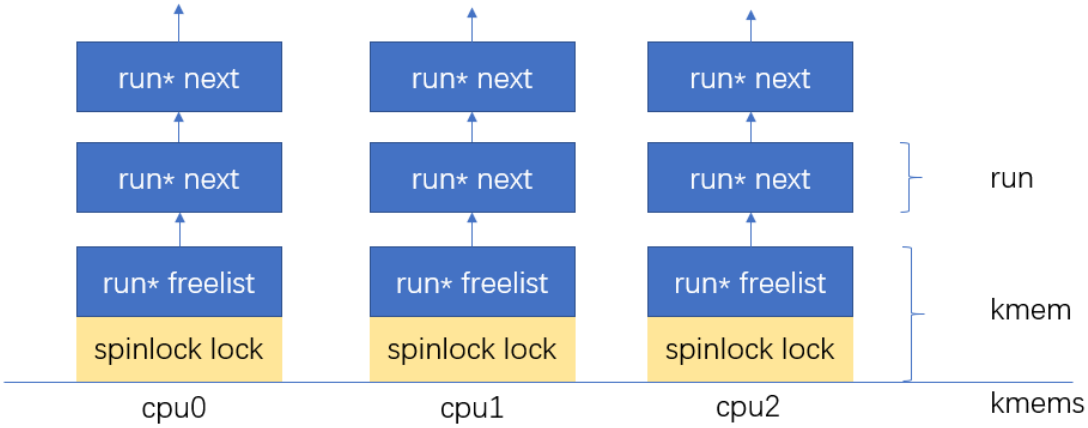
通过采用哈希表，可以并行的访问不同的桶，提高并行性。无法使用内存分配器的优化方法优化磁盘缓存。两者的优化方法中都采用了桶的思想，但是内存分配器是每一个 cpu 拥有一个链表，并访问自己的链表，而每一个 cpu 都可以访问每一个磁盘块。同时 kalloc 无需保证原子性，而由于可能存在多个进程同时访问同一个磁盘块，因此 bget 必须保证操作的原子性。

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. 实验 1

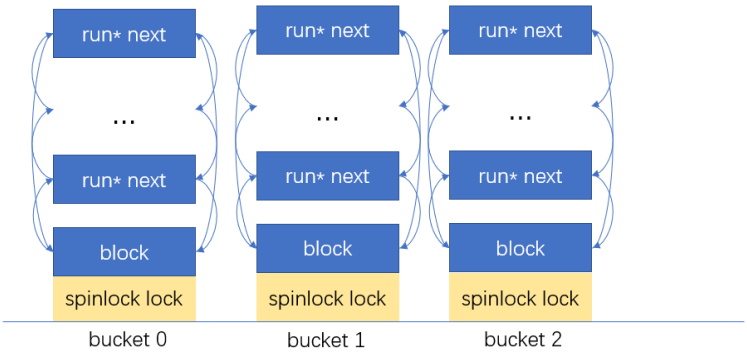
为每一个 cpu 建立一个桶，每个桶都存储一个内存链表



在初始化时（kinit()和 freerange()函数），初始化每一个链表上的锁。在为每个桶分配内存时，准备一个计数器，每得到一个新的页地址，计数器加一并对桶的个数取模，得到桶号，将新页地址存入对应的桶中。在将内存放入桶中时，需要将对应链表（kmem）加锁，完成操作后再解锁。
在释放内存时（kfree()函数），首先得到当前 cpu 号，随后将该内存块放入桶号为 cpu 号的链表的头部，再放入之前对桶加锁，放入之后解锁。
在分配内存时（kalloc()函数），首先需要获取当前 cpu 号，随后查看 cpu 对应的链表中有无空闲页，如果有则将对应页从链表中取出并返回，如果没有，则遍历其他 cpu 对应的链表，寻找空闲页，若有空闲页则将对应页取出。如果都没有空闲页则返回 0。其中取出空闲页的操作需要加锁来保证原子性。
该问题相对容易，较易解决。

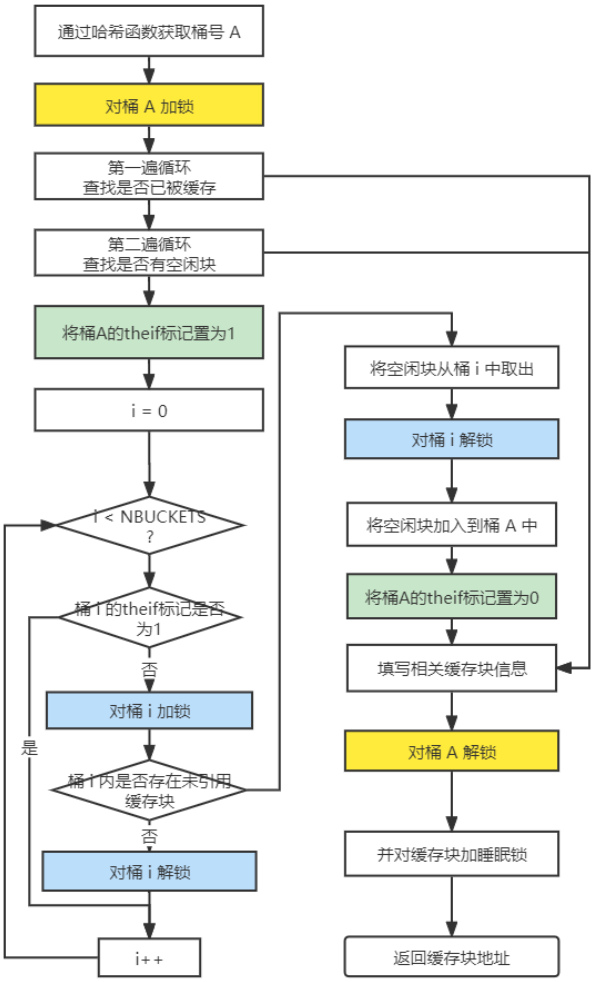
2. 实验 2

此实验较困难，很容易出现各种各样的 panic
与实验 1 思路一样，也是使用桶的方法，但是采用哈希函数来得到每一个块号对应的桶号，将对应 buf 加入到链表中。



初始化（binit）的思路与内存块初始化思路一样，区别在于一个是将缓存块加入到双向链表中，一个是将内存块添加到单向链表中。
释放（brelse）的思路也与内存块释放的思路一样。区别在于在通过块号的哈希值获取到桶号之后，将该块从原有位置移动到链表头部（bcache.hashbucket[hash_value]指向的位置），而内存块是将内存块加入到链表头部（获取到内存块之后会将内存块从链表中取出）。

获取（bget）的主体思路也一样，但是需要小心锁的应用。首先对目标桶加锁，第一遍循环在其中查找是否有目标磁盘块的缓存，第二遍循环在其中查找是否有引用数目为 0 的缓存，一旦查找到之后，释放锁，填充缓存相应信息，对缓存块加睡眠锁，返回该缓存块。如果没有找到锁，则需要去其他的链表中进行查找。每遍历到一个链表，先对其进行加锁，随后在其中查找空闲块，若能找到，则取出空闲块，解锁，对空闲块初始化，将空闲块加入到目标桶中，目标桶解锁，返回该空闲块；若不能找到，则解锁当前链表，继续遍历查找其他桶。流程图大概如下：



其中黄色区域是对目标桶 A 加锁，用于保证 bget 操作的原子性，防止出现多个进程同时访问同一个磁盘块的情况，蓝色部分是为了对桶 i 加锁，防止出现读写冲突，而绿色部分是为了避免（or 减少）死锁产生的情况。由于在对桶 A 加锁的同时对桶 i 加锁，极有可能出现死锁，因此引入 theif 标记，使得桶 A 访问到的桶 i 此时不可能访问其他的桶（桶 i 的 theif 标记需为 0）。本地测试并没有出现过死锁的情况。

三、 实验结果截图

请填写

```
[cs@localhost xv6-labs-2020]$ ./grade-lab-lock
make: 'kernel/kernel' is up to date.
== Test running kalloctest == (106.4s)
== Test   kalloctest: test1 ==
  kalloctest: test1: OK
== Test   kalloctest: test2 ==
  kalloctest: test2: OK
== Test kalloctest: sbrkmuch == kalloctest: sbrkmuch: OK (11.3s)
== Test running bcachetest == (9.1s)
== Test   bcachetest: test0 ==
  bcachetest: test0: OK
== Test   bcachetest: test1 ==
  bcachetest: test1: OK
== Test usertests == usertests: OK (203.2s)
== Test time ==
time: OK
Score: 70/70
```