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

实验设计报告

开课学期：大三秋季学期

课程名称：操作系统实验

实验名称：Lab:3 锁机制应用

实验性质：课内实验

实验时间：10/28 地点：T2608

学生班级：2019 级 4 班

学生学号：190110419

学生姓名：李怡凯

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？

一个根据用户需求实现屏蔽物理地址，根据虚拟地址进行内存分配回收等功能的结构；
作用是为用户提供 kalloc 和 kfree 接口，用于内存的申请与释放。

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？

由一个空闲页链表 freelist 和一个自旋锁构成；
kinit()：初始化内存分配器；
freerange(void *pa_start, void* pa_end):将 pa_start 到 pa_end 范围内的页表清理并加入 freelist；
kfree(void* pa):清理 pa 对应的页表并加入 freelist；
kalloc(void):在 freelist 中取出一个页表，覆盖其原有内容并返回；

c. 为什么指导书提及的优化方法可以提升性能？

原来的 kalloc 与 kfree 功能都需要使用内存分配器中唯一的锁，从而导致在多 CPU 环境下将出现多个进程争抢锁的情况；现在通过将原来的 freelist 拆分为多个，每个 CPU 拥有自己的 freelist，在执行 kalloc 与 kfree 函数时不必要争抢唯一的锁，从而提升性能。

2. 磁盘缓存

a. 什么是磁盘缓存？它的作用是？

磁盘缓存是位于 CPU 和磁盘之前的一层能够实现更快速访问的存储区域；它能够将最近使用的磁盘块缓存起来，在下一次访问磁盘前可以首先查找该区域，如果命中则直接使用该缓存块，未命中再继续查找磁盘。

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）。

在查找 bcache 中是否存在某一缓存块时，根据局部性原理，其更有可能是一个新插入的缓存块，而新插入的缓存块是紧跟在 head 后面，所以从前往后遍历，通过 next 指针实现；而在未命中时，需要找到一块未使用的缓存块，此时根据局部性原理，应该选择一块最近最少使用的块，这个块更应该被存放在靠后的位置，所以从后往前遍历，通过 prev 指针实现。

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。

在未使用哈希表的时候，磁盘缓存的性能瓶颈在于多个进程争抢磁盘缓存的锁，而通过哈希表将每次访问映射到不同的哈希桶，并将原来的锁拆分，从而减少锁的争用，从而提升性能；

不能；磁盘缓存不同于内存页表，磁盘缓存的每一块都对应磁盘上的一个块，如果采用内存分配器的优化方式，将原来的磁盘缓存区拆分，给每个 CPU 一个磁盘缓存区，则可能出现一个磁盘块同时出现在多个 CPU 的磁盘缓存区的情况，进而造成写错误或读错误。

二、 实验详细设计

1. 内存分配器优化

1.1 工作原理

内存分配器封装了内存分配，对外提供 kalloc 和 kfree 两个接口用于申请与释放页表空间。通过调用 kalloc，内存分配器在存放空闲页表的链表 freelist 中寻找一个未被使用的页返回，为避免线程错误需要在这个过程中加锁；通过调用 kfree，将页表清空并重新加入 freelist 中，这个过程同样需要加锁。

1.2 性能瓶颈分析

在多 CPU 环境下，当多个 CPU 同时申请或释放页表的时候，就会发生锁争用的情况，造成大量 CPU 阻塞等待，造成性能下降。

1.3 优化策略

将原来的 freelist 链表拆分，让每个 CPU 拥有自己的空闲页表，在申请内存时优先从自己对应的空闲页表中寻找，如果空闲页表已满再从别的 CPU 处窃取，从而减少多个 CPU 争抢一个锁的情况。

1.4 具体实现

实现上主要修改 kalloc 与 kfree 函数实现，在 kinit 初始化时首先将所有空闲页表分配给 CPU0，后续其他 CPU 再从 CPU0 抢占获得。

优化后的 kalloc 流程图如图 1 所示。

kfree 主要修改释放页表的部分逻辑，原来的逻辑为获取 kmem 锁→将页表加入 freelist→释放 kmem 锁，修改为获取本 CPU 的 kmem

锁→将页表加入本 CPUfreelist→释放本 CPUkmem 锁。

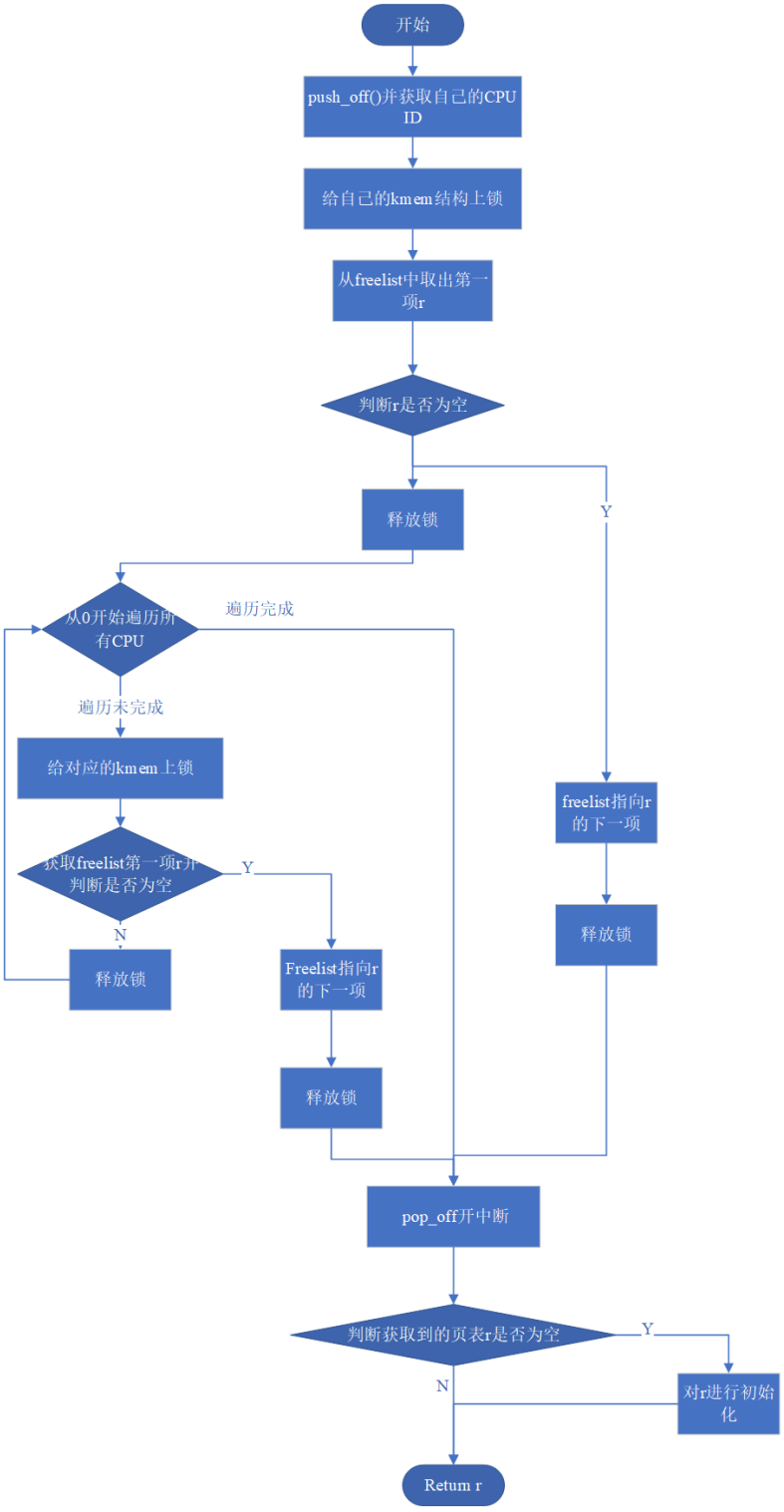

图 1 kalloc 流程图

2. 磁盘缓存器优化

2.1 工作原理

磁盘缓存器是介于磁盘和 CPU 之间的存储介质，在 CPU 需要访

问磁盘时，首先通过 bget 接口在磁盘缓存器中查找块号 blockno 和设备号 dev 的磁盘块是否在缓存器中，如果在则直接获取该块，否则寻找一个空闲块并将磁盘内容读取到该块中，当 CPU 不再使用该块时调用 brelse 接口将块释放。

2.2 性能瓶颈分析

为了确保线程安全，bget 函数在函数开始就为磁盘缓存器上锁，直到块命中或找到一个空闲块；brelse 函数同样会在函数开始就给磁盘缓存器上锁。当多个进程需要使用磁盘缓存器时，这样的策略会导致大量的 CPU 等待，造成性能下降。

2.3 优化策略

将原来的磁盘缓存区分区，通过块号进行哈希映射，对应到相应的区域中，再对相应的区域上锁并寻找，如果命中则释放锁并返回，如果未命中则给其他分区上锁，再到其他分区中寻找空闲块窃取，最后释放锁返回。

在释放缓存块时同样只需要给对应的分区上锁即可，然后进行释放，最后解锁。

通过上述改造，降低锁争用，从而提高性能。

2.4 具体实现

优化后的磁盘缓存器在 binit，bget 和 brelse 函数中发生改动。

binit 函数在初始化时将所有的磁盘缓存块分配到哈希桶中，并设置 blockno 为对应的哈希桶号。

经过优化后的 bget 函数流程如图 2 所示。

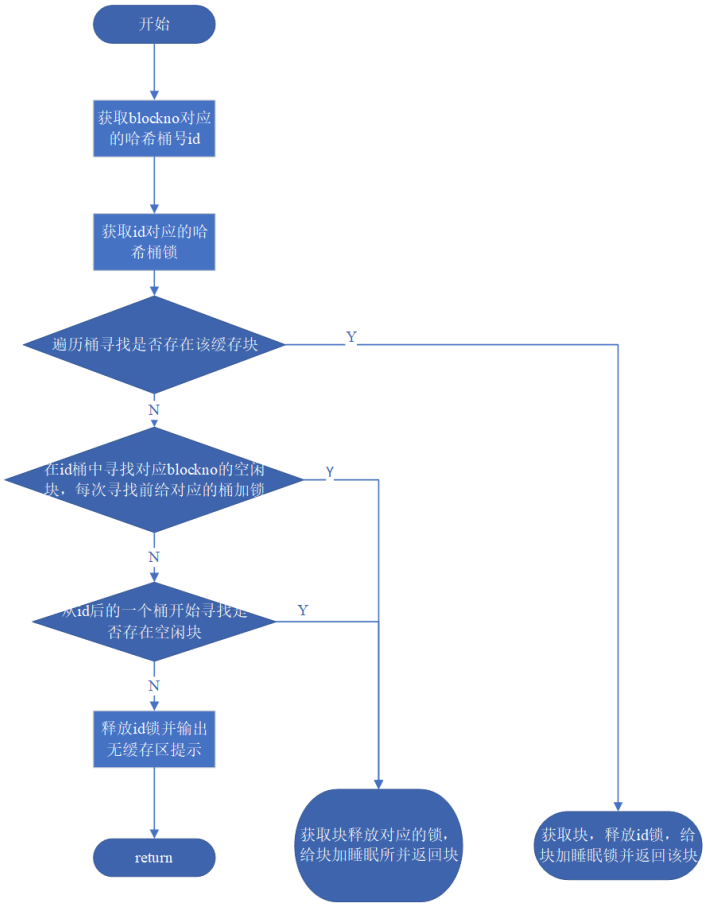

图 2 bget 函数流程图

brelse 函数在此前的基础上进行修改，获取对应桶锁，然后对 refcnt 进行自减，不同的是如果 refcnt 为 0，不需要将缓存块重新加入空闲块列表，而是直接保留在桶中等待被使用。

三、 实验结果截图

实验截图如图 3 所示。

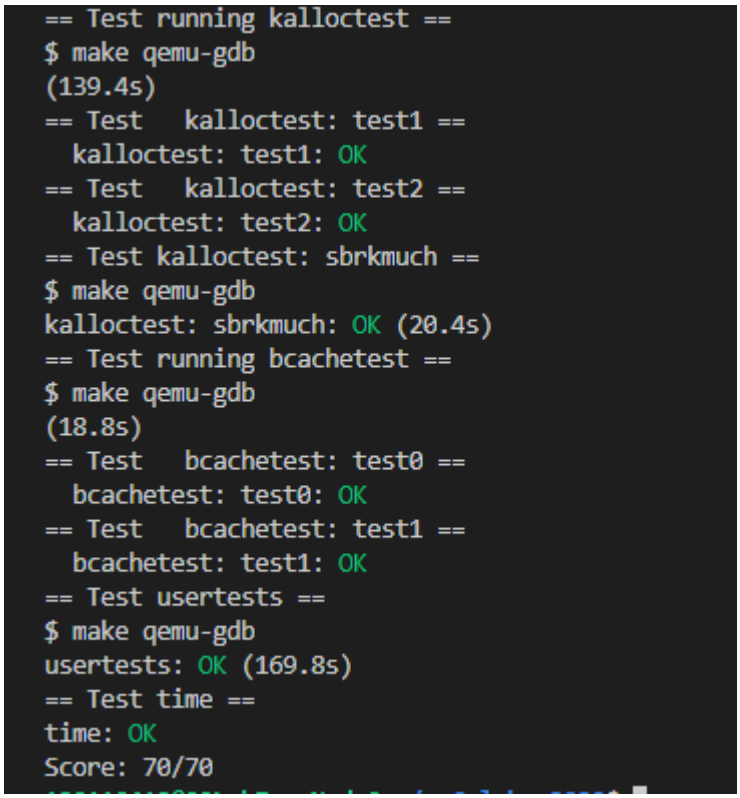

图 3 实验结果截图