|  |
| --- |
|  |
| Lab7实习报告 |
|  |

|  |
| --- |
| 姓名：吴悦欣 学号：1900012946  日期：2021/11/27 |

目录

[内容一：实验工作总结 3](#_Toc66996796)

[内容二：遇到的困难以及收获 3](#_Toc66996797)

[内容三：对课程或Lab的意见和建议 3](#_Toc66996798)

[内容四：参考文献 3](#_Toc66996799)

## 内容一：实验总结

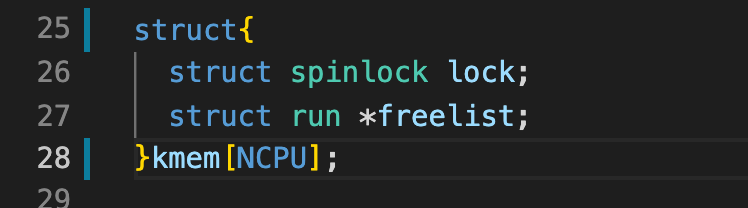
1. Lab7

这个实验叫Locks，是关于使用锁的机制使得程序正确并行的实验。xv6课本的第六章介绍了锁如何保护数据、可能产生的竞争以及xv6中的两种锁——spinlock自旋锁和sleeplock睡眠锁，两种锁的主要区别睡眠锁，在拥有锁的同时允许释放CPU并且开放中断，常常长操作的时候使用，改进了自旋锁会一直持有锁导致的效率问题。第八章的前三节介绍了文件系统的整体布局和buffer相关的文件结构，重点分析了kernel/bio.c文件。

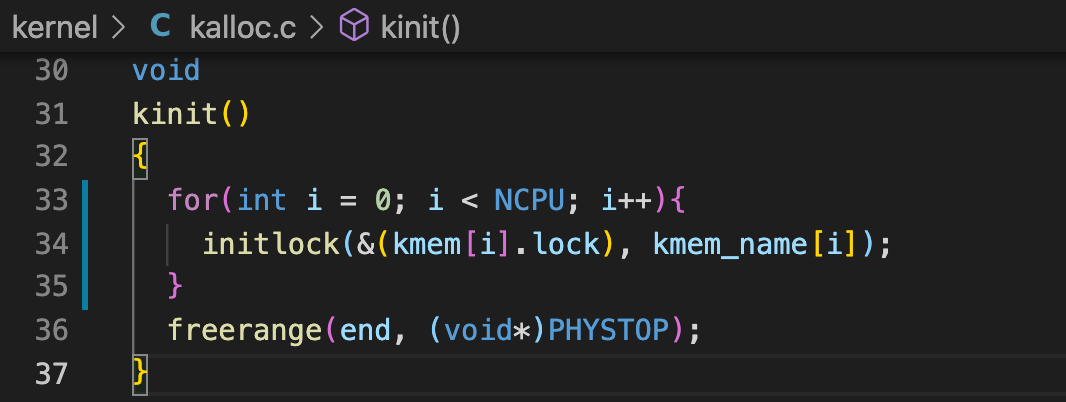
实验有两部分，核心思想/目的是一致的，将原本的“全局”的锁修改成粒度更小的锁，解决其中的并发可能产生的问题，以提高系统的效率。

第一部分是修改kernel/kalloc.c文件，目标是将原本全局的kmem.lock修改拆成多个锁分管空闲链表，直观的想法是每个CPU分配一个空闲链表及其对应的自旋锁。

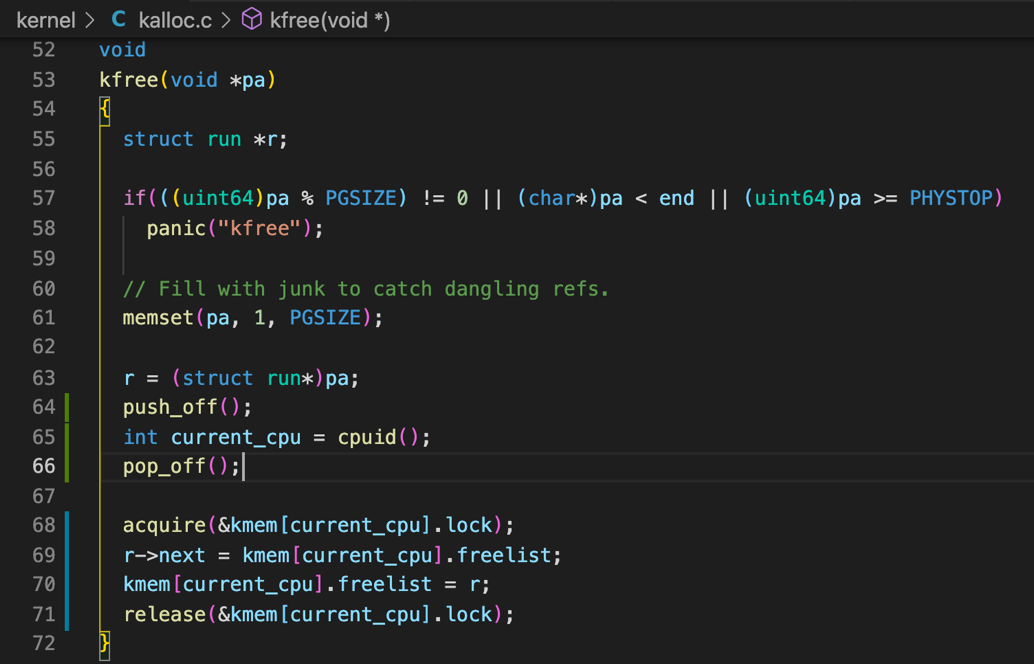
1. 修改kmem结构体定义，定义数组，数量跟CPU数量相同



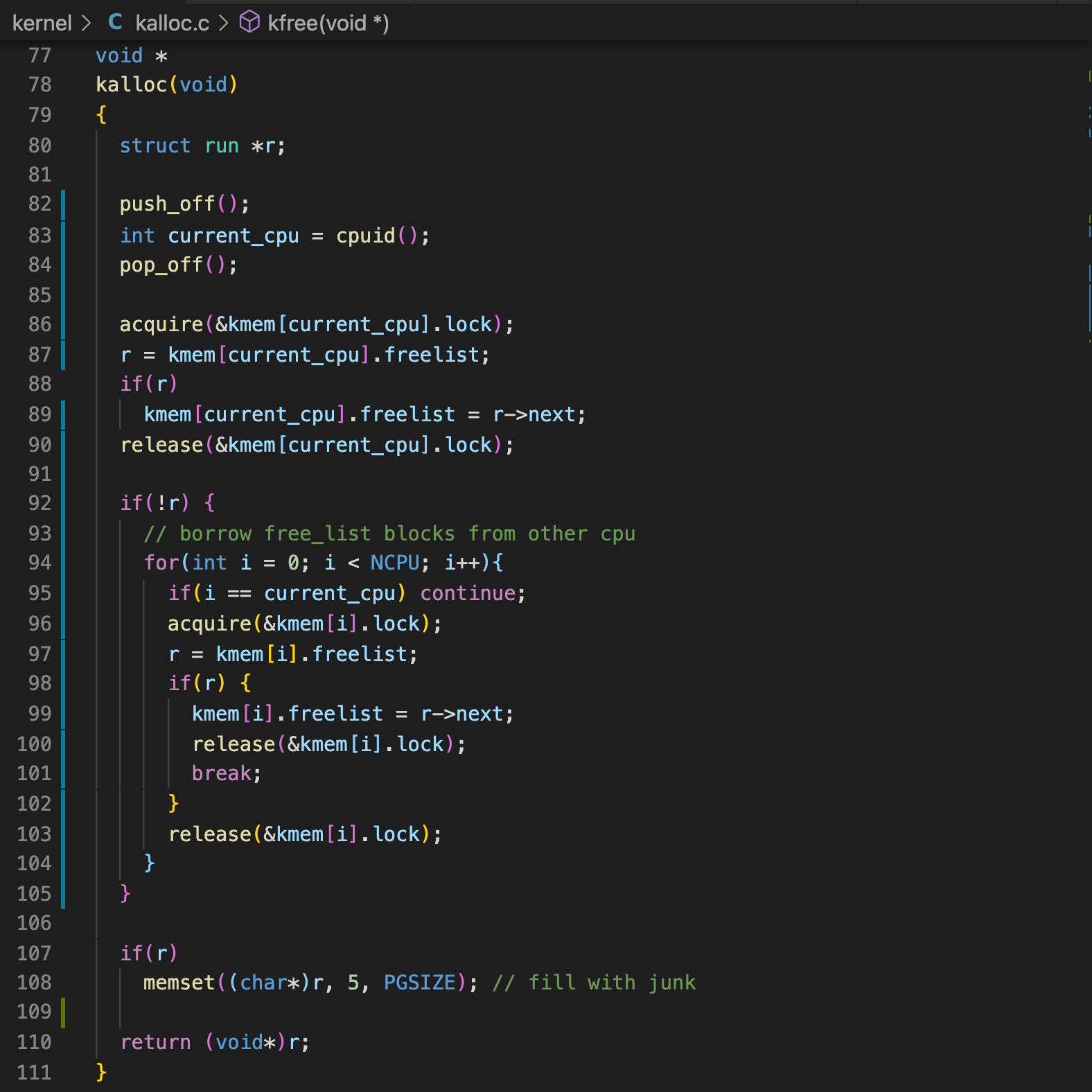
1. 修改初始化函数kinit



1. 修改kfree函数，当释放一页的时候将该页插入到当前CPU对应的空闲页链表中，需要注意的是，根据writeup的提示，在调用cpuid函数的时候应当禁止中断

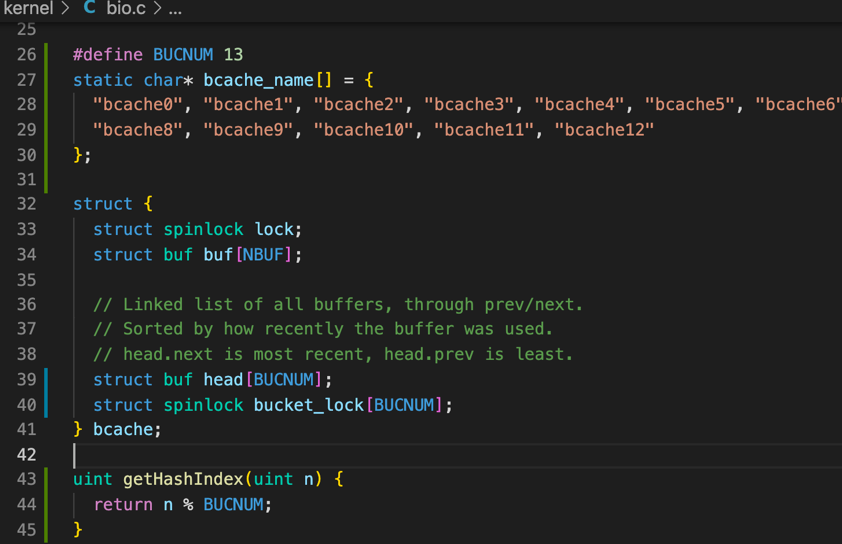


1. 修改kalloc函数，在当前CPU的空闲页链表中找一页，如果没有了，就从别的CPU的空闲页链表中拿一个过来，这个过程只需要记得加锁，问题不大

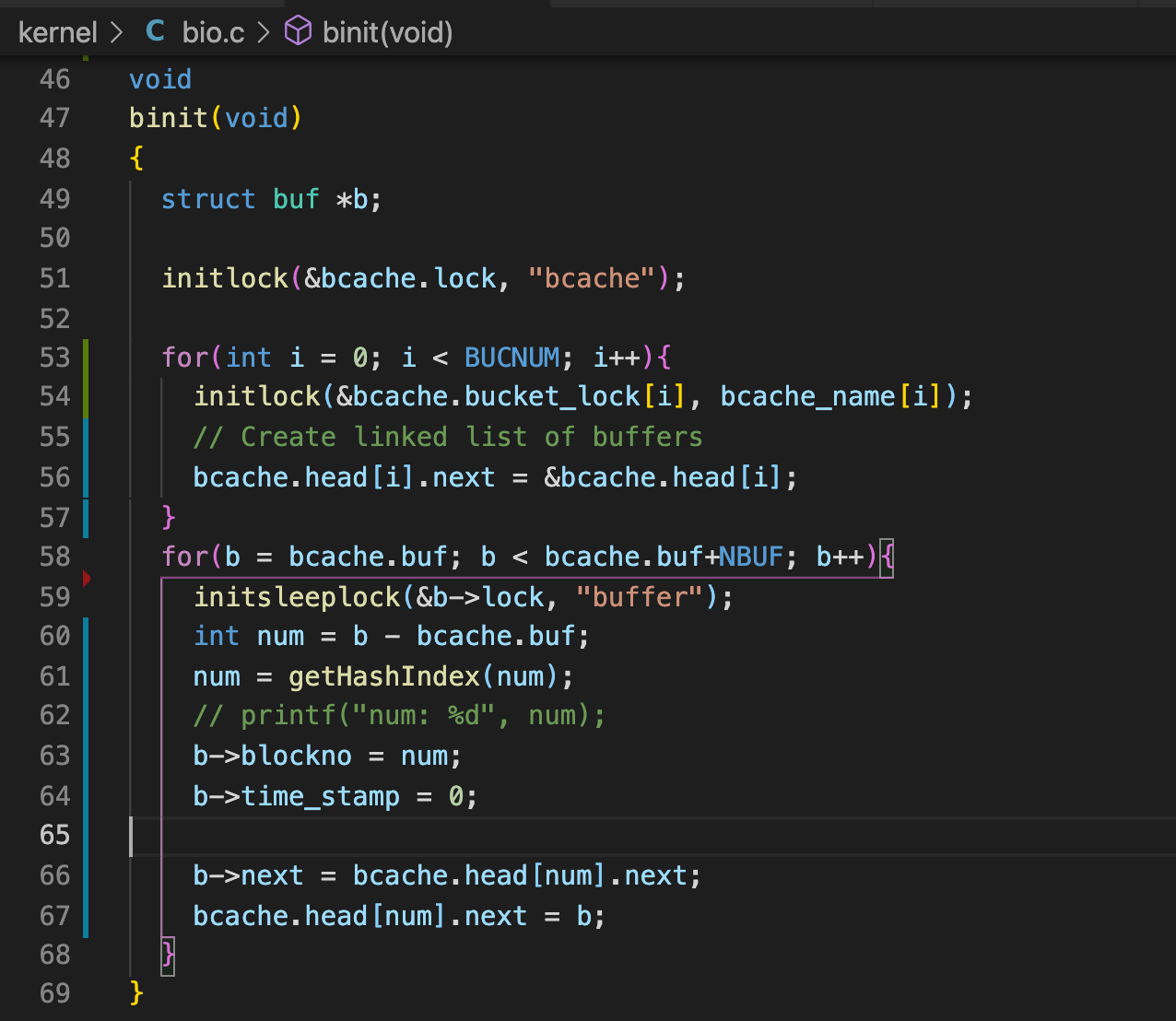


第二部分是修改kernel/bio.c中的bcache结构，原本的结构是整个bcache使用一个锁，实验的目的是将里面的buf分多个部分进行管理，建议的方式是使用哈希表进行映射到“桶”中，每个桶用一个锁。具体实现中的修改如下

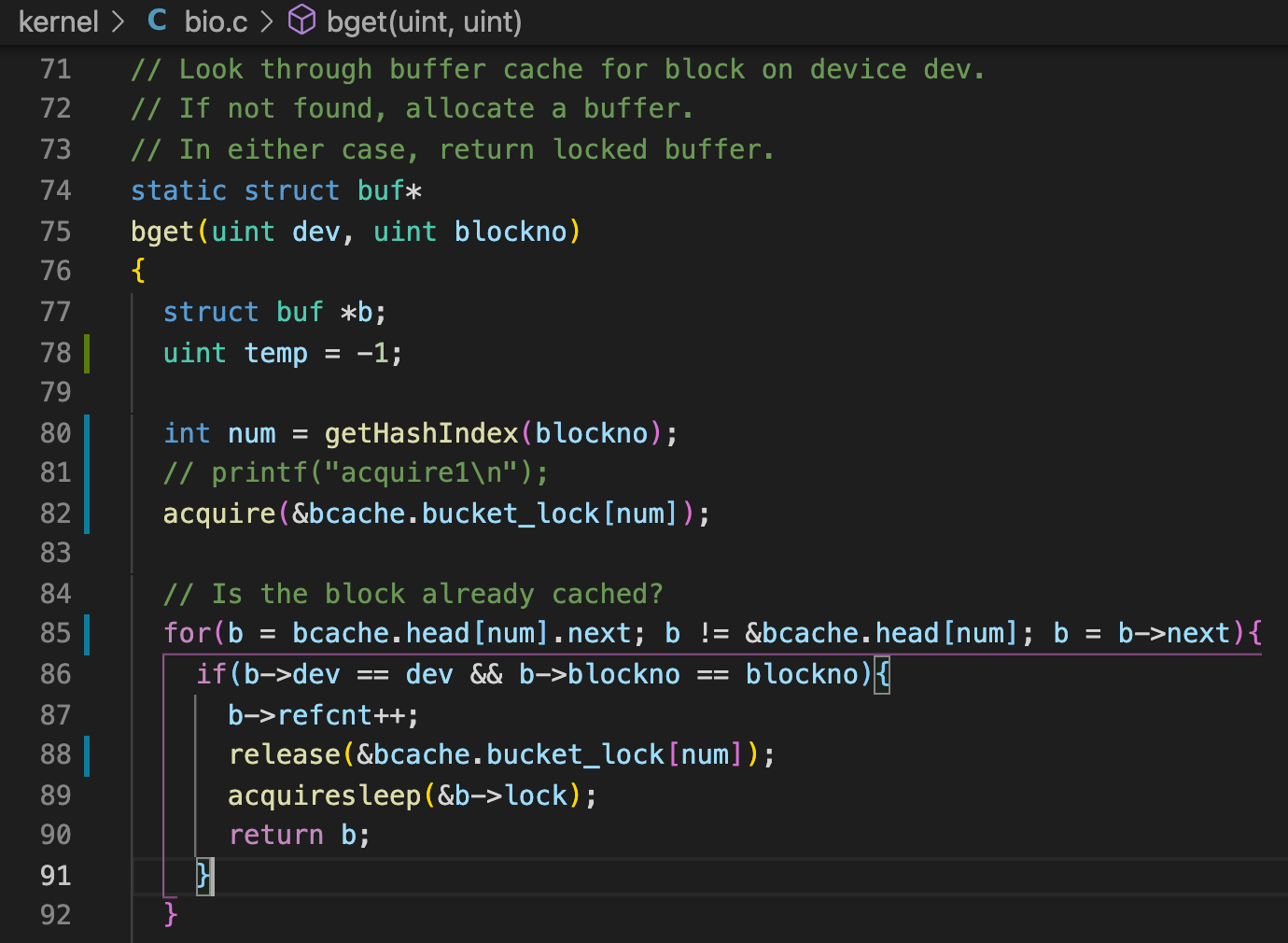
1. 使用质数13作为哈希的除数，修改bcache的结构体，将原本的一个head改成13个head，作为之后buf分配放置的列表，并为每个链表添加锁



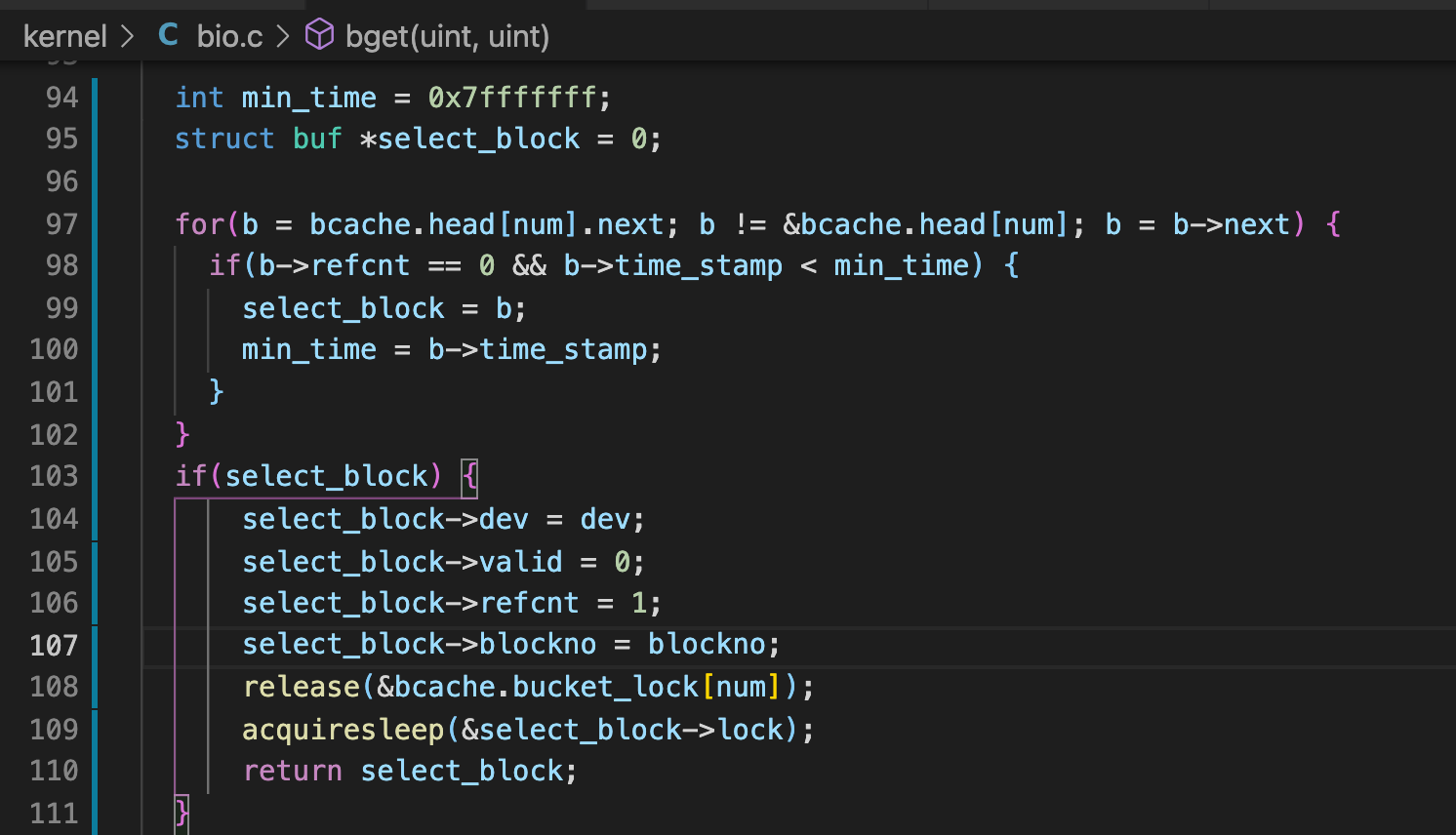
1. 修改binit函数，为每个链表的锁进行初始化，并将buf尽可能平均地分配到链表中，按照它们的地址进行初始的标号



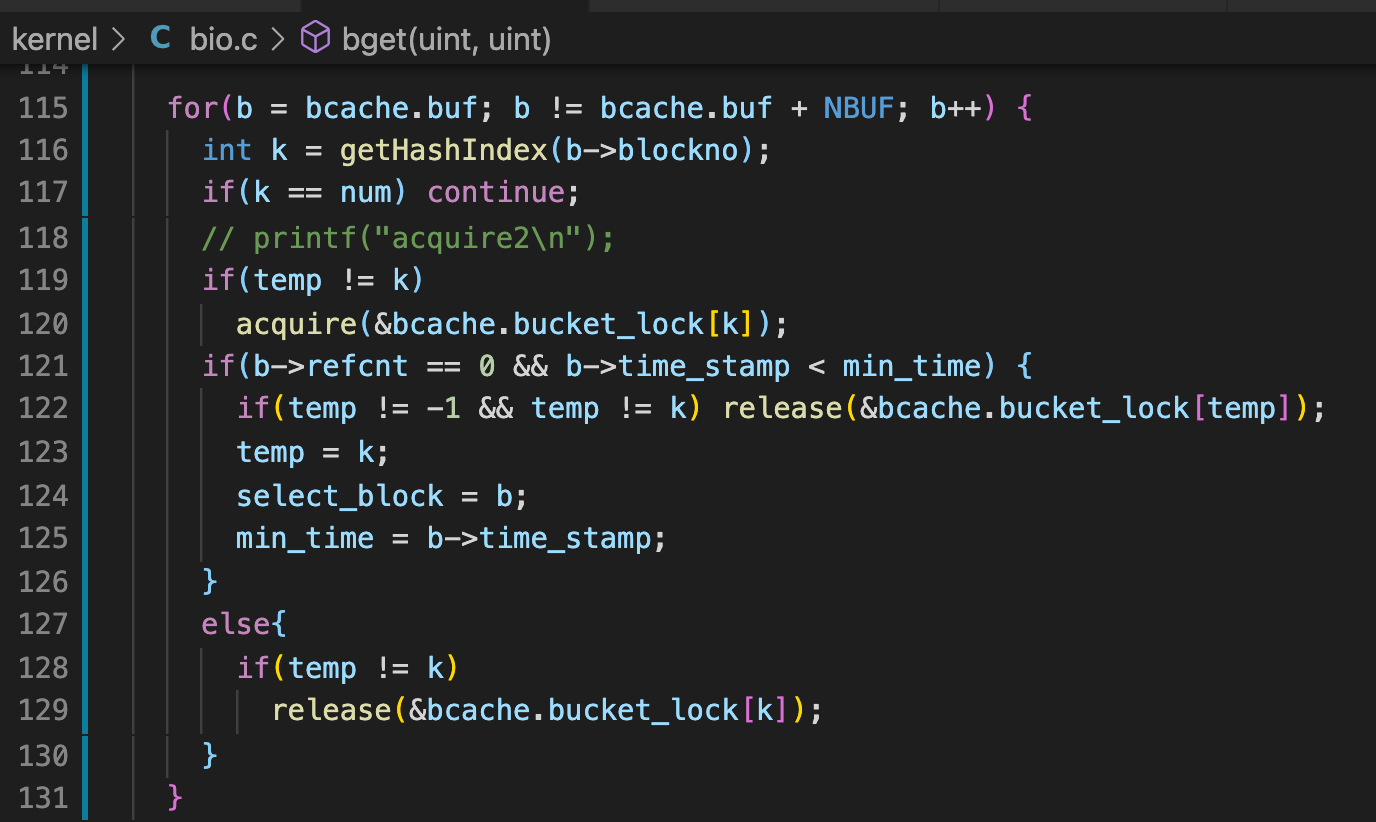
1. 修改bget函数，首先确认是否已经缓存了这个块；



如果没有找到则在blockno指定的列表中寻找是否有空闲的buffer，缓存到其中；



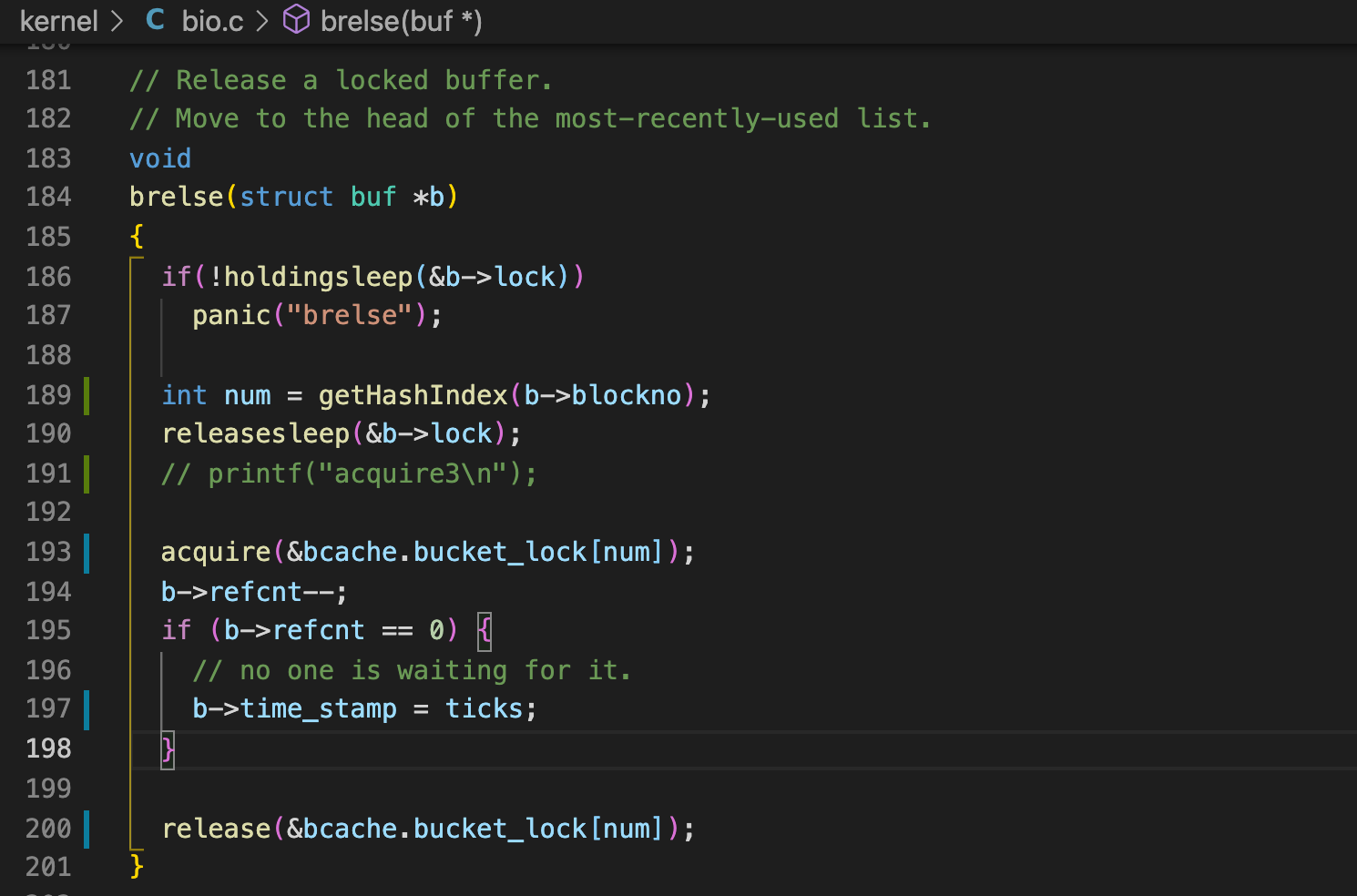
如果没有，则去其他列表中“偷”一个空闲的buffer。其中需要注意的细节是在偷其他链表的buffer的时候锁的处理需要格外小心，需要先获取该链表的锁之后再进行是否符合条件判断，如果找到了一个更优的（time\_stamp更小的）buffer块，就要释放上一个（temp记录）锁，同时不同的buffer是可能映射到同一个桶中的，所以还要判断二者是否相等。



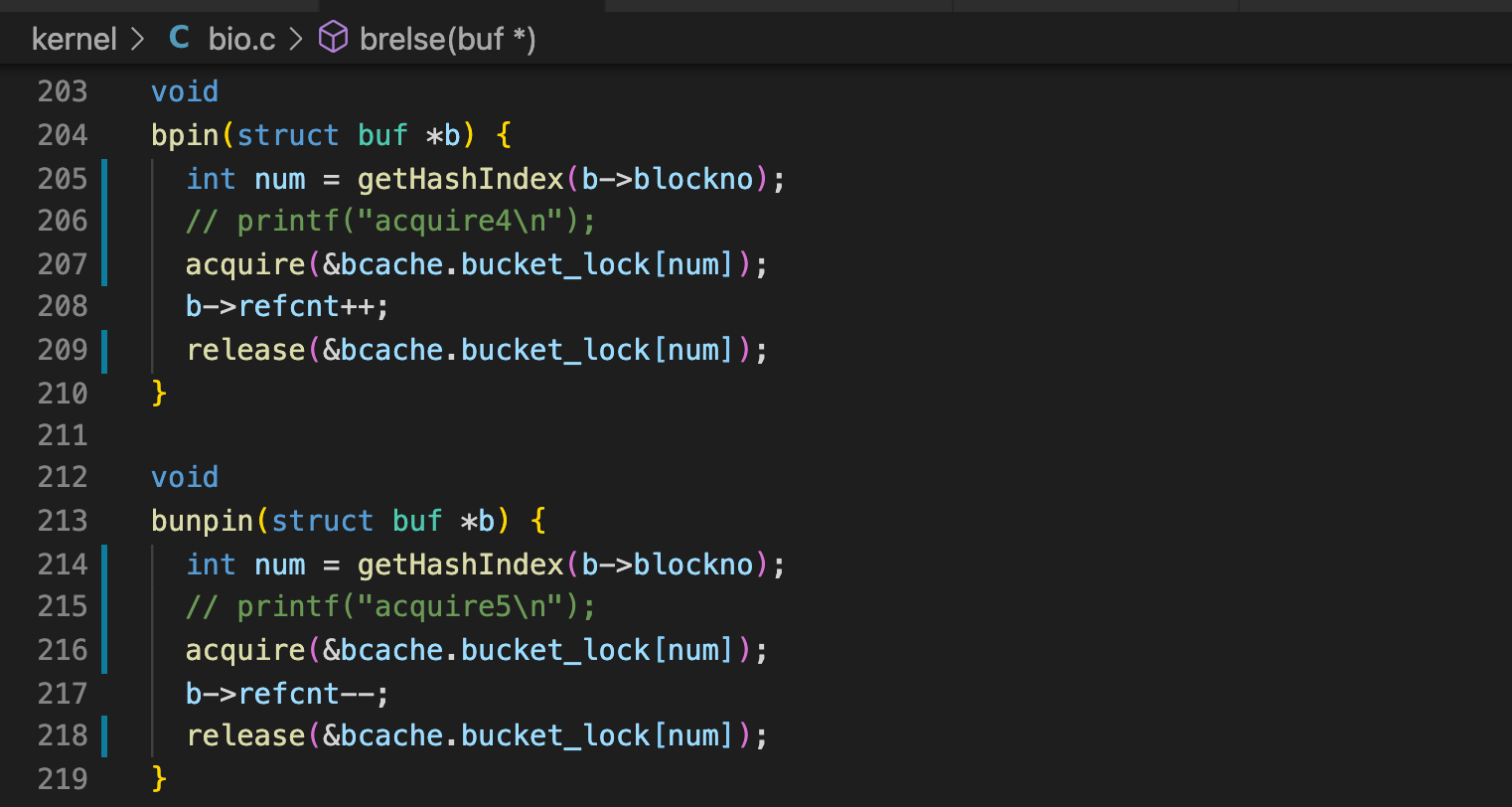
然后再将这个buffer插入到发起“偷”的行为的桶中



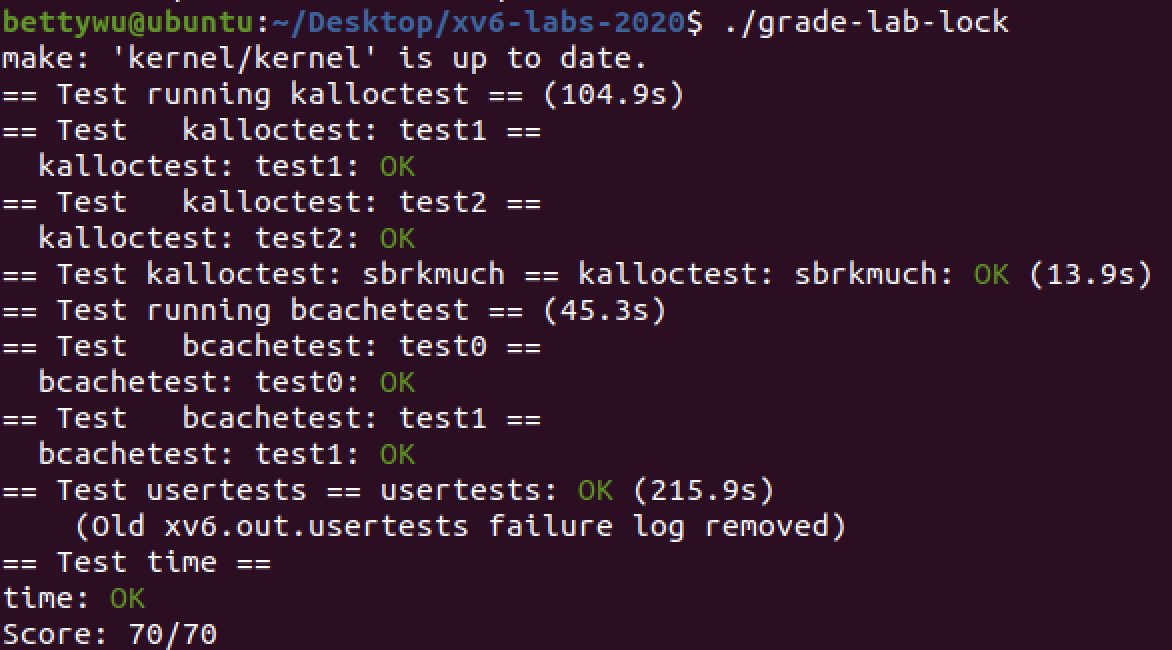
1. 修改brelse函数，更新time\_stamp即可



1. 修改bpin和bunpin函数



实验结果如下：



## 内容二：遇到的困难以及收获

1. Lab7

kalloc的部分实现比较简单，但是bcache的改造就比较tricky，主要遇到的问题是在修改bget函数的过程中的考虑，在调试的过程中遇到了多次panic: acquire，要把这一部分想明白需要耐心。实验前后kalloctest和bcachetest的一些对比表明了细粒度的锁能够在很大程度上提高程序的效率。

简要描述自己在实现Lab中遇到的难题和解决方法，以及在Lab实现中的收获与感想。内容要具体，按实际情况撰写。

## 内容三：对课程或Lab的意见和建议

欢迎同学们对本课程或Lab提出宝贵意见和建议。

## 内容四：参考文献

列出参考的文章或者网址等。

注意：在实习报告完成后，请同学们记得在目录部分按右键更新域。