仓库链接： https://github.com/zhaoyihui233/mit-6.S081-lab-2021/tree/main/lab8%20Locks

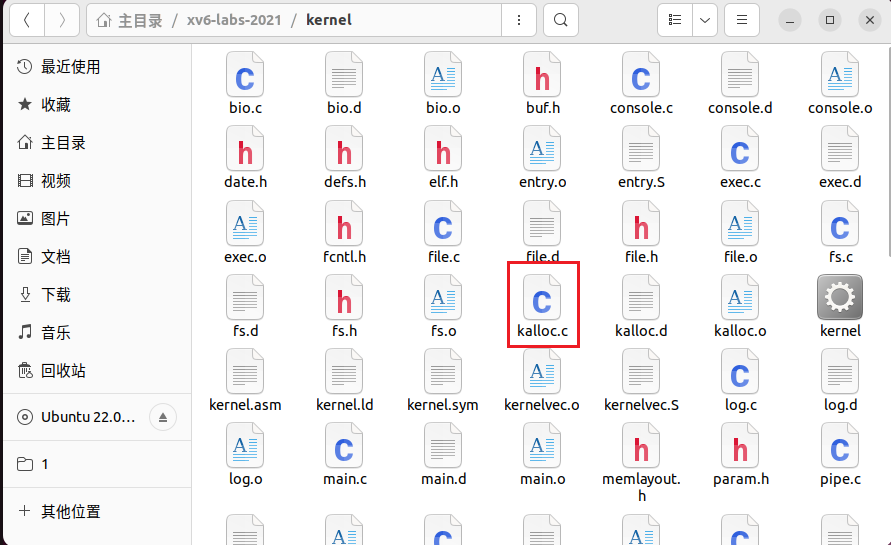
# 1 Memory allocator

## 1.1实验内容

该lab为每个 CPU 分配独立的空闲页表链表，这样多个 CPU 并发分配物理页就不会互相排斥，提高了并行性。于是本次实验，需要我们完善物理页的分配与释放。

在kerkel/kalloc.c文件里完成我们的代码。这里的代码还是单CPU时的代码，我们需要将之修改为多CPU、多空闲页表链表的代码。

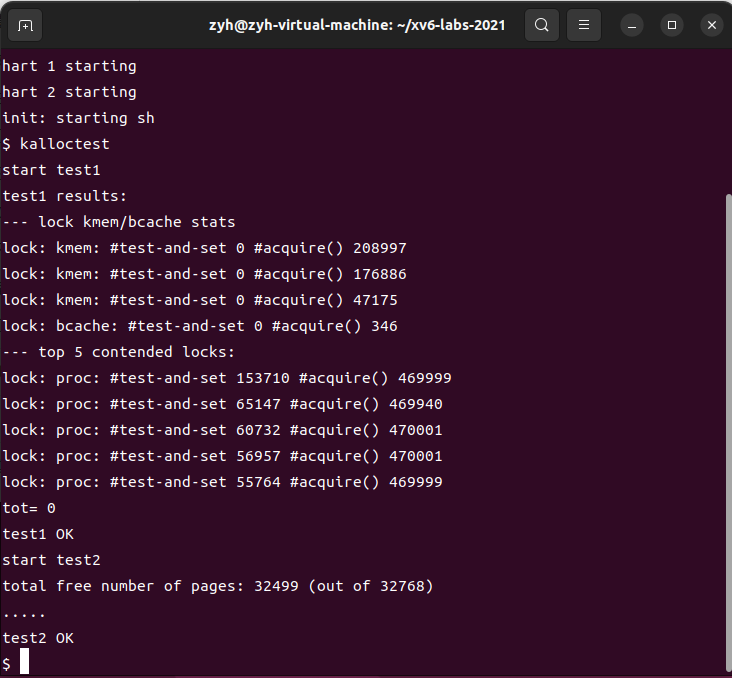
## 1.2代码位置与截图



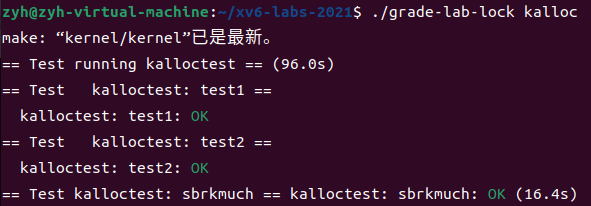


## 1.3实验结果

make qemu -> kalloctest



./grade-lab-lock kalloc



make grade:

## 1.4代码实现

### 1.4.1实现思路

提示已经基本点明了代码框架。

首先，就是将单个kmem结构体改为结构体数组，数组的长度是CPU的个数，通过常量NCPU获取。每个CPU都有自己的kmem结构体，kmem结构体有两个属性，一个是空闲链表的头指针，一个是互斥量。正好，本次lab就是需要每个CPU有自己的空闲链表。至于每个CPU有各自的互斥量，则可以实现CPU有各自独立的资源，而不是多个CPU共用一个锁。

kinit函数需要修改，因为原来的代码只对一个kmem进行初始化，这里需要对所有kmem进行初始化。具体来说，需要对多个互斥量进行初始化。

释放空间的kfree函数也需要修改，要将原来的共享锁修改为使用各自独立的锁。为获取CPU的锁，就需要获取当前CPU的编号，可以通过调用cpuid函数获取，但是该函数仅在中断关闭时可以调用它并使用它的结果，于是根据提示使用push\_off函数与pop\_off进入中断和离开中断。

最后要修改kalloc函数，这是代码修改最多的地方。首先就像在kfree函数里修改的那样，要把原来单CPU单空闲页表链表的代码改好，还要实现当一个CPU的空闲页表不足时，能够从其他CPU的空闲页表链表中获取空闲页表。

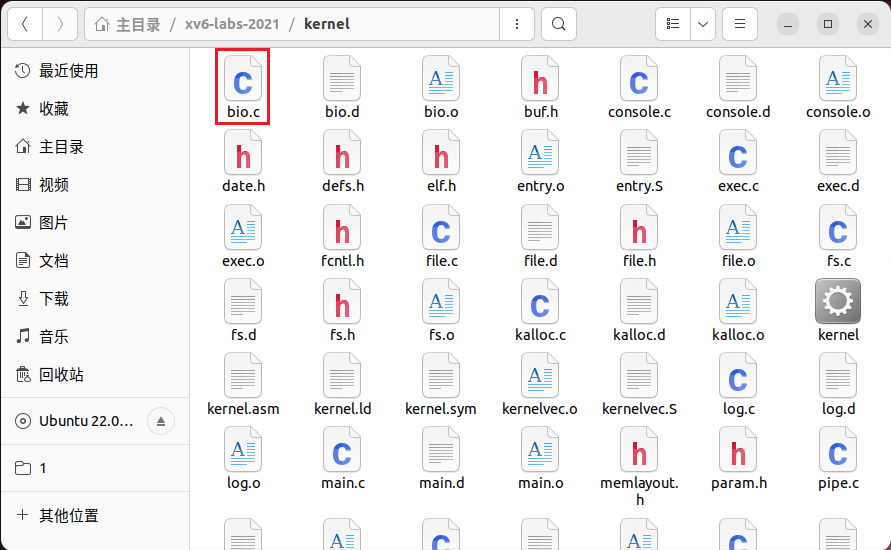
# 2 Buffer cache

## 2.1实验内容

如果多个进程密集地使用文件系统，会争夺bcache.lock，发生锁竞争。我们的任务，就是通过修改kernel/bio.c里的代码，解决上述锁竞争问题。原始版本的buffer cache由一个大锁bcache.lock保护，限制了并行运行的效率，我们要把它拆解为更精细的锁。

根据提示，我们可以使用哈希表来组织缓冲区的磁盘块，这样不同bucket的磁盘块的查找就能并发执行。

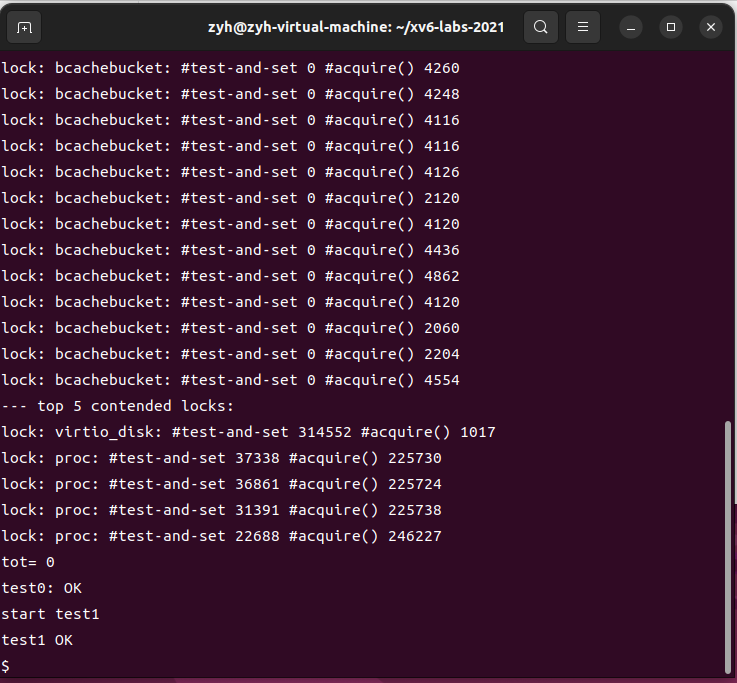
## 2.2代码位置与截图



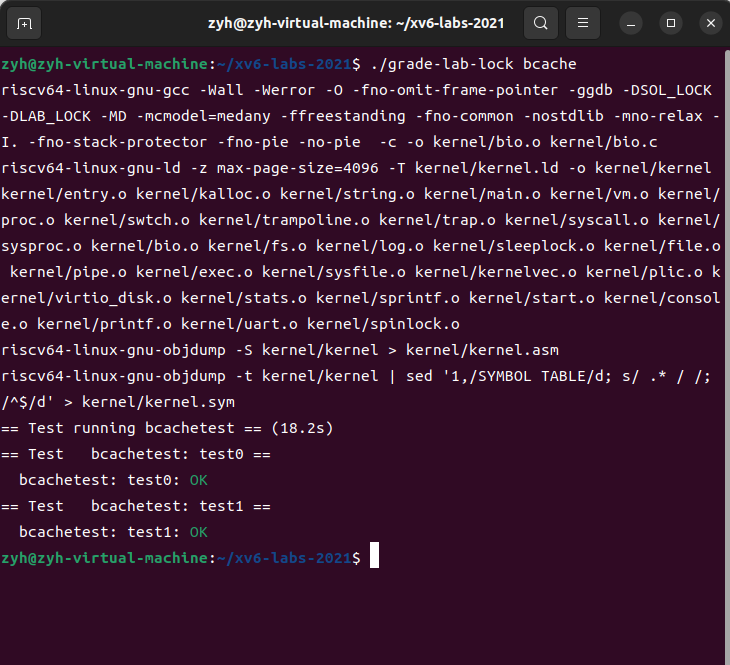


## 2.3实验结果

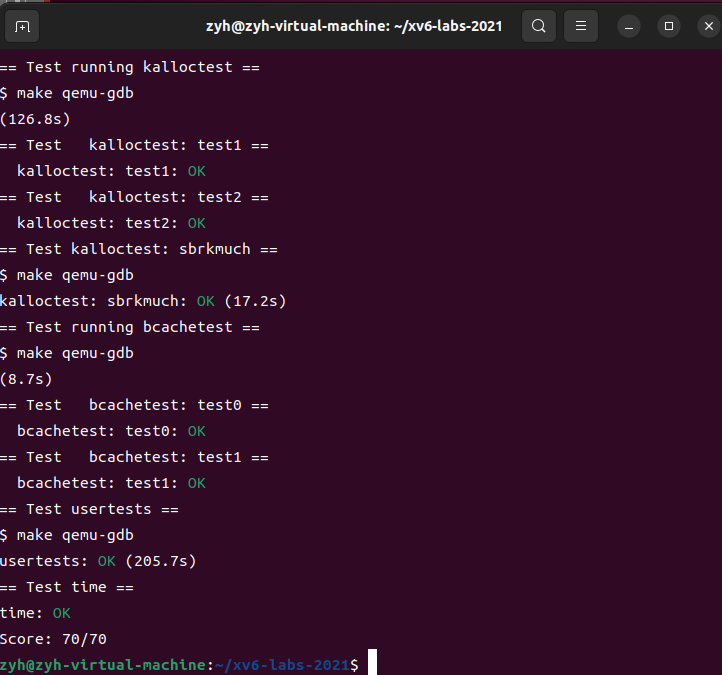
make qemu -> bcachetest



./grade-lab-lock bcache



make grade:



## 2.4代码实现

### 2.4.1实现思路

为了细化锁，原来代码里的bcache结构体已经不能满足要求了，我们需要维护一个新的数据结构。根据提示，我们可以利用哈希表来组织起缓冲区里的磁盘块，这样锁也能细化到哈希表的bucket，虽然这样仍然有锁冲突的可能，但是已经可以满足lab的要求。

我们定义如下的结构体数组：

struct

{

struct spinlock lock;

struct buf buf[NBUF\_PER\_BUC];

} bcacheBucket[NBUCKET];

根据提示，哈希表的程度最好设置为质数，例如13，因此这里的NBUCKET等于17。

于是我们要修改文件里的其他函数。

bpin函数和bunpin函数需要我们修改锁，修改较少。

binit函数是用来初始化相关数据结构的，因为我们不再使用原来的bcache结构体，所以我们需要彻底修改其代码：

void binit(void)

{

for (int i = 0; i < NBUCKET; i++)

{

initlock(&bcacheBucket[i].lock, "bcachebucket");

for (int j = 0; j < NBUF\_PER\_BUC; j++)

initsleeplock(&bcacheBucket[i].buf[j].lock, "buffer");

}

}

代码初始化数据结构里的所有锁。

最后是代码修改的重要部分，即修改bget函数和brelse函数的代码。

首先是bget函数，它的作用是返回指定的磁盘块在缓冲区里的地址。因此我们需要到对应的bucket里寻找是否包含指定的块。如果有，则返回其地址；如果没有，则在缓冲区里给其分配空间，并返回该空间的地址。然后是brelse函数，它的修改比较简单，类比着原来的代码即可。