在本实验，你将在 重构代码以提升并行 上获得经验。多核机器低并行的常见症状是high lock contention。提高并行，通常包括改变数据结构和锁策略来减少争用。你将为xv6 memory allocator和block cache这么做。

在写代码之前，确保阅读xv6 book的以下部分：

Chapter6：”Locking”和对应代码

Section3.5：”Code：Physical memory allocator”

Section8.1到8.3：”Overview”,”Buffer cache layer”,”Code:Buffer cache”

git fetch、git checkout lock、make clean

# Memory allocator

程序user/kalloctest压测xv6的memory allocator：3个进程grow、shrink它们的地址空间，导致一些调用：kalloc和kfree。kalloc和kfree获取kmem.lock。kalloctest打印（”#fetch-and-add”）acquire中的循环迭代编号，由于尝试获取另外一个核已经持有的锁，kem lock和少量其它锁。acquire循环迭代编号是一个lock contention的粗略测量。kalloctest的输出看起来类似之前完成的实验。

$ **kalloctest**

start test1

test1 results:

--- lock kmem/bcache stats

lock: kmem: #fetch-and-add 83375 #acquire() 433015

lock: bcache: #fetch-and-add 0 #acquire() 1260

--- top 5 contended locks:

lock: kmem: #fetch-and-add 83375 #acquire() 433015

lock: proc: #fetch-and-add 23737 #acquire() 130718

lock: virtio\_disk: #fetch-and-add 11159 #acquire() 114

lock: proc: #fetch-and-add 5937 #acquire() 130786

lock: proc: #fetch-and-add 4080 #acquire() 130786

tot= 83375

test1 FAIL

对于每个锁，acquire持有：该锁acquire的调用次数；acquire循环中，尝试获取但没能设置锁的次数。kalloctest调用一个system call，让kernel打印那些对于kmem、bcache lock计数，（这是本实验的关注点），也打印5个争用最强烈的锁。如果存在锁争用，acquire循环迭代中的计数将很多大。system call返回为了获取kmem和bcache lock的循环迭代总数。

对于本实验，你必须使用一个专用unloaded多核机器。如果你使用一个正在做其他事的机器，kalloctest打印的计数将是无用的。你可以使用专用Athena workstation，或你的笔记本电脑，但不要使用dialup机器。

kalloctest中锁争用的根本原因是kalloc()只有一个free list，受单lock保护。为了移除锁争用，你将不得不重新设计memory allocator来避免一个单独lock和list。基本想法是：每个CPU持有一个free list，每个list有其自己的锁。不同CPU上的分配和释放可以并行执行，因为每个CPU将在不同的list上操作。主要挑战将是处理这样的场景：某个CPU上的free list空了，但另外CPU list有free memory。在这种情况下，cpu必须steal另外CPU free list的一部分。stealing可能引发锁争用，但不常见。

你的工作是实现每个CPU freelist，当CPU free list为空时stealing。你必须让所有锁名以kmem开头。你应该为每把锁调用initlock，传入一个以kmem开头的名字。执行kalloctest来看是否你的实现已经减少了锁争用。为了检查仍可以分配所有内存，执行usertests sbrkmuch。你的输出将和下面显示的相似，减少了kmem lock争用总数，尽管具体数字不同。确保usertests中所有测试通过。make grade应该表明kalloctests通过。

$ **kalloctest**

start test1

test1 results:

--- lock kmem/bcache stats

lock: kmem: #fetch-and-add 0 #acquire() 42843

lock: kmem: #fetch-and-add 0 #acquire() 198674

lock: kmem: #fetch-and-add 0 #acquire() 191534

lock: bcache: #fetch-and-add 0 #acquire() 1242

--- top 5 contended locks:

lock: proc: #fetch-and-add 43861 #acquire() 117281

lock: virtio\_disk: #fetch-and-add 5347 #acquire() 114

lock: proc: #fetch-and-add 4856 #acquire() 117312

lock: proc: #fetch-and-add 4168 #acquire() 117316

lock: proc: #fetch-and-add 2797 #acquire() 117266

tot= 0

test1 OK

start test2

total free number of pages: 32499 (out of 32768)

.....

test2 OK

$ **usertests sbrkmuch**

usertests starting

test sbrkmuch: OK

ALL TESTS PASSED

$ **usertests**

...

ALL TESTS PASSED

一些提示：

你可以使用kernel/param.h中常量NCPU。

让freerange把所有空闲内存给正在执行freerange的CPU。

函数cpuid返回当前cpu核编号，但仅在中断关闭时可以调用它并使用它的结果。你应该使用push\_off()和pop\_off()来关闭、开启中断。

看kernel/sprintf.c中的snprintf函数，为string formatting找灵感。可以将所有锁命名为kmem。

# Buffer cache

这半个作业独立于上半个；不管你是否完成了上半个，你都可以处理这半个并通过测试。

如果多进程集中地使用文件系统，它们可能会竞争获取bcache.lock，保护kernel/bio.c中的磁盘block cache。bcachetest创建多个进程重复读取不同文件，为了产生bcache.lock上的竞争。它的输出如下（在你完成本实验之前）：

$ **bcachetest**

start test0

test0 results:

--- lock kmem/bcache stats

lock: kmem: #fetch-and-add 0 #acquire() 33035

lock: bcache: #fetch-and-add 16142 #acquire() 65978

--- top 5 contended locks:

lock: virtio\_disk: #fetch-and-add 162870 #acquire() 1188

lock: proc: #fetch-and-add 51936 #acquire() 73732

lock: bcache: #fetch-and-add 16142 #acquire() 65978

lock: uart: #fetch-and-add 7505 #acquire() 117

lock: proc: #fetch-and-add 6937 #acquire() 73420

tot= 16142

test0: FAIL

start test1

test1 OK

你可能会看到不同的输出，但为获取bcache lock的acquire循环迭代次数将会很高。如果你看到kernel/bio.c中的代码，你将看到bacache.lock保护cached block buffers list，每个block buffer中的引用数量（b->refcnt），cached block的标识（b->dev和b->blockno）。

修改block cache，以致于执行bcachetest时，对bcache所有锁的acquire循环迭代次数接近0。与block cache相关的所有锁计数之和应该为0，但如果和小于500也是可以的。更改bget和brelse，以致于对bcache不同block的并发查看和释放不会再锁上冲突（例如，不得不等待bcache.lock）。你必须保证不变性：每个block最多一个副本被缓存。当你完成时，你的输出应该和下面所示相似（尽管不完全一样）。确保usertests仍然通过。当你完成时，make grade应该通过所有测试。

$ **bcachetest**

start test0

test0 results:

--- lock kmem/bcache stats

lock: kmem: #fetch-and-add 0 #acquire() 32954

lock: kmem: #fetch-and-add 0 #acquire() 75

lock: kmem: #fetch-and-add 0 #acquire() 73

lock: bcache: #fetch-and-add 0 #acquire() 85

lock: bcache.bucket: #fetch-and-add 0 #acquire() 4159

lock: bcache.bucket: #fetch-and-add 0 #acquire() 2118

lock: bcache.bucket: #fetch-and-add 0 #acquire() 4274

lock: bcache.bucket: #fetch-and-add 0 #acquire() 4326

lock: bcache.bucket: #fetch-and-add 0 #acquire() 6334

lock: bcache.bucket: #fetch-and-add 0 #acquire() 6321

lock: bcache.bucket: #fetch-and-add 0 #acquire() 6704

lock: bcache.bucket: #fetch-and-add 0 #acquire() 6696

lock: bcache.bucket: #fetch-and-add 0 #acquire() 7757

lock: bcache.bucket: #fetch-and-add 0 #acquire() 6199

lock: bcache.bucket: #fetch-and-add 0 #acquire() 4136

lock: bcache.bucket: #fetch-and-add 0 #acquire() 4136

lock: bcache.bucket: #fetch-and-add 0 #acquire() 2123

--- top 5 contended locks:

lock: virtio\_disk: #fetch-and-add 158235 #acquire() 1193

lock: proc: #fetch-and-add 117563 #acquire() 3708493

lock: proc: #fetch-and-add 65921 #acquire() 3710254

lock: proc: #fetch-and-add 44090 #acquire() 3708607

lock: proc: #fetch-and-add 43252 #acquire() 3708521

tot= 128

test0: OK

start test1

test1 OK

$ **usertests**

...

ALL TESTS PASSED

$

请你把所有锁名以“bcache”开头。即你该为每个锁调用initlock，传入一个以“bcache”开头的名字。

减少block cache中的争用比kalloc更困难，因为bcache buffers确实在多进程间共享。对于kalloc，给每个CPU它自己的allocator可以减少绝大多数争用；它们不会对block cache起作用。我们建议你用哈希表（每个bucket一把锁）在cache中查看block序号。

某些情况下，你的方案有lock conflict是可以的：

1，当两个进程并发地使用相同block序号。bcachetest test0从不会这么做。

2，当两个进程同时在cache中miss时，需要找到一个未使用的block来替换。bcachetest test0从不会这么做。

3，无论你使用什么方案来区分blocks和locks，两个进程并发使用blocks都会产生conflict；例如，两个进程使用blocks（它的block号哈希到哈希表同一位置）。bcachetest.test0()可能这么做，取决于你的设计，但你应该尝试调整策略细节来避免confilcts（示例，改变哈希表的尺寸）。

bcachetest的test1使用块比缓冲区，练习一些文件系统代码路径。

这是一些提示：

1，阅读xv6书中block cache的描述（章节：8.1-8.3）。

2，使用固定数量的buckets，并不动态扩容hasmb表是可以的。使用素数（例如13）个buckets来减少哈希碰撞的可能性。

3，当buffer没有被找到，在哈希表中找一个buffer，并给该buffer分配一个entry必须是原子的

4，移除list的所有buffers（bcache.head等等），不再用最后使用时间来标定buffers。（如，kernel/trap.c中用的ticks）。因为这个改变，brelse无需获取bcache lock，bget可以基于时间标定选择LRU block。

5，bget中串行化删除是可以的。（例如，当在cache中没查到时，bget中选择一个buffer来重用）

6，你的方案，在一些常见中可能需要持有两个锁；例如，在移除期间，你可能需要持有bcache lock和每个bucket的lock。确保你避免死锁。

7，当取代一个block，你可能移动一个struct buf从一个bucket到另外一个bucket，因为新block哈希到一个不同的bucket。你可能有一个困难情况：新block可能哈希到与旧block一样的bucket。确保你在那种情况下避免死锁。

8，一些调试提示：实现bucket锁，但在bget开始、结束时保留全局bcache.lock acquire/release来串行化代码。一旦确保无竞争情况正确时，移除全局锁，处理并发问题。你也可以执行make CPUS=1 qemu来测试单核。