Lab4: locks

在本实验中，您将获得重新设计代码以提高并行性的经验。多核机器上并行性差的一个常见症状是频繁的锁争用。提高并行性通常涉及更改数据结构和锁定策略以减少争用。您将对xv6内存分配器和块缓存执行此操作。

> Attention
>
> 在编写代码之前，请确保阅读xv6手册中的以下部分：
>
> - 第6章：《锁》和相应的代码。
> - 第3.5节：《代码：物理内存分配》
> - 第8.1节至第8.3节：《概述》、《Buffer cache层》和《代码：Buffer cache》

要开始本实验，请将代码切换到`lock`分支

```
$ git checkout lock
$ make clean
```

Memory allocator

程序*user/kalloctest.c*强调了xv6的内存分配器：三个进程增长和缩小地址空间，导致对`kalloc`和`kfree`的多次调用。`kalloc`和`kfree`获得`kmem.lock`。`kalloctest`打印（作为“#fetch-and-add”）在`acquire`中由于尝试获取另一个内核已经持有的锁而进行的循环迭代次数，如`kmem`锁和一些其他锁。`acquire`中的循环迭代次数是锁争用的粗略度量。完成实验前，`kalloctest`的输出与此类似：

```
$ 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
```

`acquire` 为每个锁维护要获取该锁的 `acquire` 调用计数，以及 `acquire` 中循环尝试但未能设置锁的次数。 `kalloctest` 调用一个系统调用，使内核打印 `kmem` 和 `bcache` 锁（这是本实验的重点）以及5个最有具竞争的锁的计数。如果存在锁争用，则 `acquire` 循环迭代的次数将很大。系统调用返回 `kmem` 和 `bcache` 锁的循环迭代次数之和。

对于本实验，您必须使用具有多个内核的专用空载机器。如果你使用一台正在做其他事情的机器， `kalloctest` 打印的计数将毫无意义。你可以使用专用的 Athena 工作站或你自己的笔记本电脑，但不要使用拨号机。

`kalloctest` 中锁争用的根本原因是 `kalloc()` 有一个空闲列表，由一个锁保护。要消除锁争用，您必须重新设计内存分配器，以避免使用单个锁和列表。基本思想是为每个CPU维护一个空闲列表，每个列表都有自己的锁。因为每个CPU将在不同的列表上运行，不同CPU上的分配和释放可以并行运行。主要的挑战将是处理一个CPU的空闲列表为空，而另一个CPU的列表有空闲内存的情况；在这种情况下，一个CPU必须“窃取”另一个CPU空闲列表的一部分。窃取可能会引入锁争用，但这种情况希望不会经常发生。

YOUR JOB

您的工作是实现每个CPU的空闲列表，并在CPU的空闲列表为空时进行窃取。所有锁的命名必须以“ `kmem` ”开头。也就是说，您应该为每个锁调用 `initlock` ，并传递一个以“ `kmem` ”开头的名称。运行 `kalloctest` 以查看您的实现是否减少了锁争用。要检查它是否仍然可以分配所有内存，请运行 `usertests sbrkmuch` 。您的输出将与下面所示的类似，在 `kmem` 锁上的争用总数将大大减少，尽管具体的数字会有所不同。确保 `usertests` 中的所有测试都通过。评分应该表明考试通过。

```
 $ 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
$
```

提示：

- 您可以使用*kernel/param.h*中的常量`NCPU`
- 让`freerange`将所有可用内存分配给运行`freerange`的CPU。
- 函数`cpuid`返回当前的核心编号，但只有在中断关闭时调用它并使用其结果才是安全的。您应该使用`push_off()`和`pop_off()`来关闭和打开中断。
- 看看*kernel/sprintf.c*中的`snprintf`函数，了解字符串如何进行格式化。尽管可以将所有锁命名为“`kmem`”。

Buffer cache(hard

这一半作业独立于前一半；不管你是否完成了前半部分，你都可以完成这半部分（并通过测试）。

如果多个进程密集地使用文件系统，它们可能会争夺 `bcache.lock` ，它保护*kernel/bio.c*中的磁盘块缓存。 `bcachetest` 创建多个进程，这些进程重复读取不同的文件，以便在 `bcache.lock` 上生成争用；（在完成本实验之前）其输出如下所示：

```
$ 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
```

您可能会看到不同的输出，但 `bcache` 锁的 `acquire` 循环迭代次数将很高。如果查看*kernel/bio.c*中的代码，您将看到 `bcache.lock` 保护已缓存的块缓冲区的列表、每个块缓冲区中的引用计数（ `b->refcnt` ）以及缓存块的标识（ `b->dev` 和 `b->blockno` ）。

YOUR JOB

修改块缓存，以便在运行 `bcachetest` 时，bcache（buffer cache的缩写）中所有锁的 `acquire` 循环迭代次数接近于零。理想情况下，块缓存中涉及的所有锁的计数总和应为零，但只要总和小于500就可以。修改 `bget` 和 `brelse` ，以便bcache中不同块的并发查找和释放不太可能在锁上发生冲突（例如，不必全部等待 `bcache.lock` ）。你必须保护每个块最多缓存一个副本的不变量。完成后，您的输出应该与下面显示的类似（尽管不完全相同）。确保 `usertests` 仍然通过。完成后， `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
$
```

请将你所有的锁以“`bcache`”开头进行命名。也就是说，您应该为每个锁调用`initlock`，并传递一个以“`bcache`”开头的名称。

减少块缓存中的争用比`kalloc`更复杂，因为bcache缓冲区真正的在进程（以及CPU）之间共享。对于`kalloc`，可以通过给每个CPU设置自己的分配器来消除大部分争用；这对块缓存不起作用。我们建议您使用每个哈希桶都有一个锁的哈希表在缓存中查找块号。

在您的解决方案中，以下是一些存在锁冲突但可以接受的情形：

- 当两个进程同时使用相同的块号时。`bcachetest test0`始终不会这样做。
- 当两个进程同时在cache中未命中时，需要找到一个未使用的块进行替换。`bcachetest test0`始终不会这样做。
- 在你用来划分块和锁的方案中某些块可能会发生冲突，当两个进程同时使用冲突的块时。例如，如果两个进程使用的块，其块号散列到哈希表中相同的槽。`bcachetest test0`可能会执行此操作，具体取决于您的设计，但您应该尝试调整方案的细节以避免冲突（例如，更改哈希表的大小）。

`bcachetest`的`test1`使用的块比缓冲区更多，并且执行大量文件系统代码路径。

提示：

- 请阅读xv6手册中对块缓存的描述（第8.1-8.3节）。
- 可以使用固定数量的散列桶，而不动态调整哈希表的大小。使用素数个存储桶（例如13）来降低散列冲突的可能性。
- 在哈希表中搜索缓冲区并在找不到缓冲区时为该缓冲区分配条目必须是原子的。
- 删除保存了所有缓冲区的列表（`bcache.head`等），改为标记上次使用时间的时间戳缓冲区（即使用***kernel/trap.c***中的`ticks`）。通过此更改，`brelse`不需要获取bcache锁，并且`bget`可以根据时间戳选择最近使用最少的块。
- 可以在`bget`中串行化回收（即`bget`中的一部分：当缓存中的查找未命中时，它选择要复用的缓冲区）。
- 在某些情况下，您的解决方案可能需要持有两个锁；例如，在回收过程中，您可能需要持有bcache锁和每个bucket（散列桶）一个锁。确保避免死锁。
- 替换块时，您可能会将`struct buf`从一个bucket移动到另一个bucket，因为新块散列到不同的bucket。您可能会遇到一个棘手的情况：新块可能会散列到与旧块相同的bucket中。在这种情况下，请确保避免死锁。
- 一些调试技巧：实现bucket锁，但将全局`bcache.lock`的`acquire`/`release`保留在`bget`的开头/结尾，以串行化代码。一旦您确定它在没有竞争条件的

情况下是正确的，请移除全局锁并处理并发性问题。您还可以运行 `make CPUS=1 qemu` 以使用一个内核进行测试。