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

实验设计报告

开课学期：2022 年秋季

课程名称：操作系统

实验名称：锁机制的应用

实验性质：课内实验

实验时间：2022.10.21 地点：T2507

学生班级：6

学生学号：200110631

学生姓名：张景昊

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2022 年 9 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？

答：
定义：
内存分配器是一种负责内存的分配与管理的容器；
作用：
它对上层提供 kalloc()和 kfree()两个接口来管理操作系统中剩余的空闲物理内存：
当需要分配新的内存空间时调用 kalloc()函数，需要释放内存空间的时候调用 kfree()函数。通过这两个接口实现对空闲物理内存的分配和释放
（接口名称在不同的操作系统中有不同定义，但内存分配器实现的功能大同小异）

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？

答：
数据结构：
内存分配器维护了由空闲物理页组成的链表和一个内存池。链表由空闲物理页组成，将物理内存划分为 4kb 大小来管理（传统的页面大小）并使用自旋锁进行保护。每个空闲页都在链表中指向下一个空闲物理页。内存池由两个指针来描述，在使用过程中会被划分为内存块，配合链表实现内存空间管理
函数及功能：
freerange():将内存划分为 1 页大小的空间
kinit():初始化所有 kmem 锁，并调用 freerange()函数实现内存初始划分
kfree():将指针指向的页的内存全部设置为 1，并将其添加到 kmem.freelist 链表中，以此实现对特定物理内存页的回收
kalloc():从 kmem.freelist 中找到一个空闲页并将他拿出来，将这一页的内存值全部设置为 5 并返回指向该页的指针。以此实现分配一页空闲的物理内存

c．为什么指导书提及的优化方法可以提升性能？

答：

在实验的原始实现中，由于所有内存块由一个锁管理，因此若出现多个进程并发获取内存，就会造成非常多的锁冲突。

指导书上给出的优化方法是将 freelist 等分，让不同的 CPU 核使用独立的链表，每个链表由各自的锁管理，这样在多进程并发获取内存的时候，就不会出现多个 CPU 核争抢同一个空闲区域的现象。由此可以大大减少并发获取内存时产生的锁冲突

2、 磁盘缓存

a．什么是磁盘缓存？它的作用是？

答：
定义：
将最近经常访问的磁盘块缓存在内存当中，这块内存块被称为磁盘缓存
作用：
操作系统对不同位置的存储空间访问速度不同，读取磁盘的速度很慢而读取内存的速度则快很多。利用磁盘缓存，将最常访问的磁盘块缓存在内存当中，产生一个磁盘与文件系统交互的中间层，就能够极大提高数据访存效率

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）。

答：
在 buf 结构体中，最近访问过的内存块会被放在指针的下一个，即 next 指向的结点，最远访问过的会被存在指针的上一个节点，即 prev 指向的节点。在对链表进行遍历时，在有些情况下需要寻找最远访问过的结点，这时只需要访问 prev 指向的节点即可。这样设置指针可以减少开销

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。

答：

磁盘缓存出现性能问题是由于，所有的 buffer 都被写入了同一个链表中，因此若出现多个进程并发请求 buffer，它们的请求只能被顺序处理，并行性不足。通过哈希表的方式，将各个块号 blockno 的散列值作为 key 对块进行分组，在需要获取或者释放缓存块时，只需要对某个哈希桶加锁，就可以在各个哈希桶之间实现并行操作。既然实现了并行的操作，自然会比顺序操作有更好的性能。

可以。对磁盘缓存的优化本质上是一种以提高并行性为目的的优化，利用内存分配器优化内存，本质上实现的也是提高并行性能的优化。既然能够做到提高并行性能，用来对磁盘缓存做优化应该也是没有问题的

二、 实验详细设计

注意不要照搬实验指导书上的内容，请根据你自己的设计方案来填写

锁争用：

为不同的 CPU 分别创建一个内存池，分别使用锁管理，每个 CPU 从各自的内存池获取内存，调用内存分配器进行内存管理；当某个内存池为空的时候，允许该 CPU 从其他内存池偷取内存块，以此在一定程度上缓解划分内存带来的内存不足现象。

磁盘缓存：

创建一个大小为 13 的哈希表，表中元素为 cache 池，每个池由自己的锁管理，cache 块根据 blockno 哈希到不同的 cache 池中。在读取新的磁盘块时，先到对应的 cache 池中寻找是否已缓存，若没有，则现在自己的池中寻找空闲块，若无，则到其他池寻找空闲块，找到后将其转移到本池。在这个过程中，要为 cache 块打上时间戳，每次使用该块时更新，在寻找空闲块时优先选择最久未被使用的空闲块。

三、 实验结果截图

第一部分锁争用，修改之前 kalloctest 的运行结果如下：

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ kalloctest
start test1
test1 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 23691 #acquire() 433016
lock: bcache: #fetch-and-add 0 #acquire() 1242
--- top 5 contended locks:
lock: kmem: #fetch-and-add 23691 #acquire() 433016
lock: proc: #fetch-and-add 22372 #acquire() 173331
lock: virtio_disk: #fetch-and-add 15919 #acquire() 114
lock: proc: #fetch-and-add 5505 #acquire() 173408
lock: proc: #fetch-and-add 3284 #acquire() 173408
tot= 23691
test1 FAIL
start test2
total free number of pages: 32499 (out of 32768)
.....
test2 OK
$
```

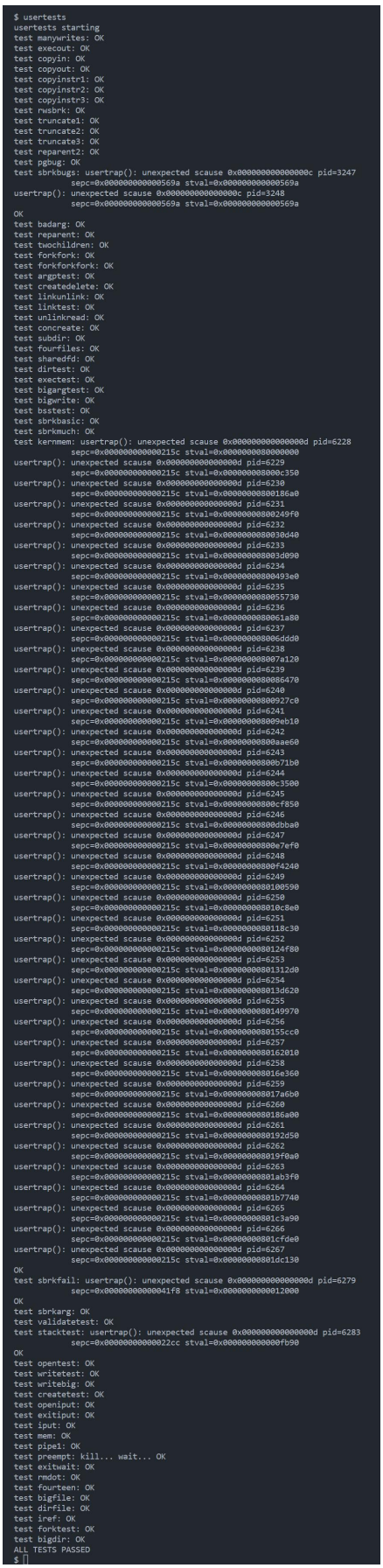
可见此时锁争用的数量较多

修改之后 kalloctest 的运行结果如下：

```
hart 2 starting
hart 1 starting
init: starting sh
$ kalloctest
start test1
test1 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 51831
lock: kmem: #fetch-and-add 0 #acquire() 190227
lock: kmem: #fetch-and-add 0 #acquire() 190979
lock: bcache: #fetch-and-add 0 #acquire() 334
--- top 5 contended locks:
lock: proc: #fetch-and-add 25915 #acquire() 137210
lock: virtio_disk: #fetch-and-add 8978 #acquire() 57
lock: proc: #fetch-and-add 6155 #acquire() 137288
lock: proc: #fetch-and-add 2138 #acquire() 137291
lock: pr: #fetch-and-add 1986 #acquire() 5
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
```

由上图可见，在划分内存池的思想修改代码后，tot=0，锁争用现象大大减少，达到了预期效果

usertests 的运行结果如下：



由图可见，通过了所有测试，实验该部分完成

修改代码前 bcachetest 的运行结果如下：

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ bcachetest
start test0
test0 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 32941
lock: kmem: #fetch-and-add 0 #acquire() 38
lock: kmem: #fetch-and-add 0 #acquire() 100
lock: bcache: #fetch-and-add 62437 #acquire() 72486
--- top 5 contended locks:
lock: virtio_disk: #fetch-and-add 186423 #acquire() 1236
lock: bcache: #fetch-and-add 62437 #acquire() 72486
lock: proc: #fetch-and-add 45065 #acquire() 85797
lock: proc: #fetch-and-add 36609 #acquire() 85797
lock: proc: #fetch-and-add 34449 #acquire() 86186
tot= 62437
test0: FAIL
start test1
test1 OK
$
```

可见此时缓存块管理的锁争用次数较多

修改代码后 bcachetest 的运行结果如下：

```
$ bcachetest
start test0
test0 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 32933
lock: kmem: #fetch-and-add 0 #acquire() 57
lock: kmem: #fetch-and-add 0 #acquire() 94
lock: bcache_hash: #fetch-and-add 0 #acquire() 4140
lock: bcache_hash: #fetch-and-add 0 #acquire() 2129
lock: bcache_hash: #fetch-and-add 0 #acquire() 4282
lock: bcache_hash: #fetch-and-add 0 #acquire() 4328
lock: bcache_hash: #fetch-and-add 0 #acquire() 6340
lock: bcache_hash: #fetch-and-add 0 #acquire() 6330
lock: bcache_hash: #fetch-and-add 0 #acquire() 6608
lock: bcache_hash: #fetch-and-add 0 #acquire() 6297
lock: bcache_hash: #fetch-and-add 0 #acquire() 7861
lock: bcache_hash: #fetch-and-add 0 #acquire() 6291
lock: bcache_hash: #fetch-and-add 0 #acquire() 4542
lock: bcache_hash: #fetch-and-add 0 #acquire() 4133
lock: bcache_hash: #fetch-and-add 0 #acquire() 2125
--- top 5 contended locks:
lock: virtio_disk: #fetch-and-add 206130 #acquire() 1245
lock: proc: #fetch-and-add 42963 #acquire() 77904
lock: proc: #fetch-and-add 36838 #acquire() 78280
lock: proc: #fetch-and-add 34556 #acquire() 77904
lock: proc: #fetch-and-add 16496 #acquire() 77902
tot= 0
test0: OK
start test1
test1 OK
```

tot=0，可见经过修改代码后，缓存快管理的锁争用次数明显下降

usertests 的运行结果如下：

```
$ usertests
usertests starting
test manywrites: OK
test execout: OK
test copyin: OK
test copyout: OK
test copyinstr1: OK
test copyinstr2: OK
test copyinstr3: OK
test rwsbrk: OK
test truncate1: OK
test truncate2: OK
test truncate3: OK
test reparent2: OK
test pgbug: OK
test sbrkbugs: usertrap(): unexpected scause 0x000000000000000c pid=3246
            sepc=0x000000000000569a stval=0x000000000000569a
usertrap(): unexpected scause 0x000000000000000c pid=3247
            sepc=0x000000000000569a stval=0x000000000000569a
OK
test badarg: OK
test reparent: OK
test twochildren: OK
test forkfork: OK
test forkforkfork: OK
test argptest: OK
test createdelete: OK
test linkunlink: OK
test linktest: OK
test unlinkread: OK
test concreate: OK
test subdir: OK
test fourfiles: OK
test sharedfd: OK
test dirtest: OK
test exectest: OK
test bigargtest: OK
test bigwrite: OK
test bsstest: OK
test sbrkbasic: OK
test sbrkmuch: OK
test kernmem: usertrap(): unexpected scause 0x000000000000000d pid=6221
            sepc=0x000000000000215c stval=0x0000000080000000
usertrap(): unexpected scause 0x000000000000000d pid=6222
            sepc=0x000000000000215c stval=0x000000008000c350
usertrap(): unexpected scause 0x000000000000000d pid=6223
            sepc=0x000000000000215c stval=0x0000000080018a0
usertrap(): unexpected scause 0x000000000000000d pid=6224
            sepc=0x000000000000215c stval=0x000000008002249f0
usertrap(): unexpected scause 0x000000000000000d pid=6225
            sepc=0x000000000000215c stval=0x0000000080030d40
usertrap(): unexpected scause 0x000000000000000d pid=6226
            sepc=0x000000000000215c stval=0x000000008003d090
usertrap(): unexpected scause 0x000000000000000d pid=6227
            sepc=0x000000000000215c stval=0x0000000080493e0
usertrap(): unexpected scause 0x000000000000000d pid=6228
            sepc=0x000000000000215c stval=0x0000000080055730
usertrap(): unexpected scause 0x000000000000000d pid=6229
            sepc=0x000000000000215c stval=0x0000000080061a80
usertrap(): unexpected scause 0x000000000000000d pid=6230
            sepc=0x000000000000215c stval=0x000000008006ddd0
usertrap(): unexpected scause 0x000000000000000d pid=6231
            sepc=0x000000000000215c stval=0x000000008007a120
usertrap(): unexpected scause 0x000000000000000d pid=6232
            sepc=0x000000000000215c stval=0x0000000080086470
usertrap(): unexpected scause 0x000000000000000d pid=6233
            sepc=0x000000000000215c stval=0x00000008009027c0
usertrap(): unexpected scause 0x000000000000000d pid=6234
            sepc=0x000000000000215c stval=0x000000008009eb10
usertrap(): unexpected scause 0x000000000000000d pid=6235
            sepc=0x000000000000215c stval=0x0000000800aae60
usertrap(): unexpected scause 0x000000000000000d pid=6236
            sepc=0x000000000000215c stval=0x0000000800b71b0
usertrap(): unexpected scause 0x000000000000000d pid=6237
            sepc=0x000000000000215c stval=0x0000000800c3500
usertrap(): unexpected scause 0x000000000000000d pid=6238
            sepc=0x000000000000215c stval=0x0000000800cf850
usertrap(): unexpected scause 0x000000000000000d pid=6239
            sepc=0x000000000000215c stval=0x0000000800dbba0
usertrap(): unexpected scause 0x000000000000000d pid=6240
            sepc=0x000000000000215c stval=0x0000000800e7ef0
usertrap(): unexpected scause 0x000000000000000d pid=6241
            sepc=0x000000000000215c stval=0x0000000800f4240
usertrap(): unexpected scause 0x000000000000000d pid=6242
            sepc=0x000000000000215c stval=0x0000000080100590
usertrap(): unexpected scause 0x000000000000000d pid=6243
            sepc=0x000000000000215c stval=0x000000008010c8e0
usertrap(): unexpected scause 0x000000000000000d pid=6244
            sepc=0x000000000000215c stval=0x0000000080118c30
usertrap(): unexpected scause 0x000000000000000d pid=6245
            sepc=0x000000000000215c stval=0x0000000080124f80
usertrap(): unexpected scause 0x000000000000000d pid=6246
            sepc=0x000000000000215c stval=0x0000000801312d0
usertrap(): unexpected scause 0x000000000000000d pid=6247
            sepc=0x000000000000215c stval=0x000000008013d620
usertrap(): unexpected scause 0x000000000000000d pid=6248
            sepc=0x000000000000215c stval=0x0000000080149970
usertrap(): unexpected scause 0x000000000000000d pid=6249
            sepc=0x000000000000215c stval=0x0000000080155cc0
usertrap(): unexpected scause 0x000000000000000d pid=6250
            sepc=0x000000000000215c stval=0x0000000080162010
usertrap(): unexpected scause 0x000000000000000d pid=6251
            sepc=0x000000000000215c stval=0x000000008016e360
usertrap(): unexpected scause 0x000000000000000d pid=6252
            sepc=0x000000000000215c stval=0x000000008017a6b0
usertrap(): unexpected scause 0x000000000000000d pid=6253
            sepc=0x000000000000215c stval=0x0000000080186a00
usertrap(): unexpected scause 0x000000000000000d pid=6254
            sepc=0x000000000000215c stval=0x0000000080192d50
usertrap(): unexpected scause 0x000000000000000d pid=6255
            sepc=0x000000000000215c stval=0x000000008019f0a0
usertrap(): unexpected scause 0x000000000000000d pid=6256
            sepc=0x000000000000215c stval=0x0000000801ab3f0
usertrap(): unexpected scause 0x000000000000000d pid=6257
            sepc=0x000000000000215c stval=0x0000000801b7740
usertrap(): unexpected scause 0x000000000000000d pid=6258
            sepc=0x000000000000215c stval=0x0000000801c3a90
usertrap(): unexpected scause 0x000000000000000d pid=6259
            sepc=0x000000000000215c stval=0x0000000801cfde0
usertrap(): unexpected scause 0x000000000000000d pid=6260
            sepc=0x000000000000215c stval=0x0000000801dc130
OK
test sbrkfail: usertrap(): unexpected scause 0x000000000000000d pid=6272
            sepc=0x00000000000041f8 stval=0x0000000000012000
OK
test sbrkarg: OK
test validatetest: OK
test stacktest: usertrap(): unexpected scause 0x000000000000000d pid=6276
            sepc=0x00000000000022cc stval=0x000000000000fb90
OK
test opentest: OK
test writetest: OK
test writebig: OK
test createtest: OK
test openiput: OK
```

```
OK
test opentest: OK
test writetest: OK
test writebig: OK
test createtest: OK
test openiput: OK
test exitiput: OK
test iput: OK
test mem: OK
test pipe1: OK
test preempt: kill... wait... OK
test exitwait: OK
test rmdot: OK
test fourteen: OK
test bigfile: OK
test dirfile: OK
test iref: OK
test forktest: OK
test bigdir: OK
ALL TESTS PASSED
$
```

由上图可见，修改后的代码通过了 usertests，该部分实验完成

最后运行 makegrade 结果如下：

```
== Test running kalloctest ==
$ make qemu-gdb
(119.2s)
== Test   kalloctest: test1 ==
  kalloctest: test1: OK
== Test   kalloctest: test2 ==
  kalloctest: test2: OK
== Test kalloctest: sbrkmuch ==
$ make qemu-gdb
kalloctest: sbrkmuch: OK (12.2s)
== Test running bcachetest ==
$ make qemu-gdb
(9.8s)
== Test   bcachetest: test0 ==
  bcachetest: test0: OK
== Test   bcachetest: test1 ==
  bcachetest: test1: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (155.3s)
== Test time ==
time: FAIL
    Cannot read time.txt
Score: 69/70
make: *** [Makefile:317: grade] Error 1
200110631@comp1:~/xv6-labs-2020$
```

由于未知原因，测试程序无法读取已经存在于目录当中的 time.txt 文件；但是实验内容的 kalloctest 和 bcachetest 两个测试均能通过，该实验达到了预期效果