

一、实验内容

1.1. 内存分配器

1.1.1 实验步骤

a. 对所有cpu添加kmem

```
struct kmem kmems[NCPU];
```

b. 修改kinit函数，初始化每个kmem的锁，由于该函数只会在cpuid=0时调用，所以初始时所有内存块都被分配给cpu0。

```
void
kinit()
{
    for(int i = 0; i < NCPU; i++){
        initlock(&kmems[i].lock, "kmem");
    }
    freerange(end, (void*)PHYSTOP);
}
```

c. 修改kfree函数，在获取cpuid前需要调用push_off函数防止中断切换cpu(后续还需调用pop_off)，而后将pa加入到相应的kmem即可。

```
void
kfree(void *pa)
{
    struct run *r;

    if(((uint64)pa % PGSIZE) != 0 || (char*)pa < end || (uint64)pa >= PHYSTOP)
        panic("kfree");

    // Fill with junk to catch dangling refs.
    memset(pa, 1, PGSIZE);

    r = (struct run*)pa;

    push_off();
    int id = cpuid();
```

```

    acquire(&kmems[id].lock);
    r->next = kmems[id].freelist;
    kmems[id].freelist = r;
    release(&kmems[id].lock);
    pop_off();
}

```

d. 修改kalloc函数，首先判断cpuid对应的kmems中是否还有空余块，若没有则需要遍历其他kmems以寻找空余块并返回，若有则执行先前相同的操作。

```

void *kalloc(void) {
    struct run *r;
    push_off();
    int id = cpuid();
    acquire(&kmems[id].lock);
    r = kmems[id].freelist;
    if(r){
        kmems[id].freelist = r->next;
        release(&kmems[id].lock);
    }else{
        release(&kmems[id].lock);
        for(int i = 0; i < NCPU; i++){
            if(id == i){
                continue;
            }
            acquire(&kmems[i].lock);
            if(kmems[i].freelist){
                r = kmems[i].freelist;
                kmems[i].freelist = r->next;
                release(&kmems[i].lock);
                break;
            }
            release(&kmems[i].lock);
        }
    }

    pop_off();
    if(r)
        memset((char*)r, 5, PGSIZE); // fill with junk
    return (void*)r;
}

```

1.1.2 实验测试结果

- kalloc test

```
$ kalloc test
start test1
test1 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 158569
lock: kmem: #fetch-and-add 0 #acquire() 153665
lock: kmem: #fetch-and-add 0 #acquire() 120807
lock: bcache: #fetch-and-add 0 #acquire() 9
lock: bcache: #fetch-and-add 0 #acquire() 11
lock: bcache: #fetch-and-add 0 #acquire() 284
lock: bcache: #fetch-and-add 0 #acquire() 12
lock: bcache: #fetch-and-add 0 #acquire() 6
lock: bcache: #fetch-and-add 0 #acquire() 4
lock: bcache: #fetch-and-add 0 #acquire() 4
lock: bcache: #fetch-and-add 0 #acquire() 4
lock: bcache: #fetch-and-add 0 #acquire() 16
lock: bcache: #fetch-and-add 0 #acquire() 4
lock: bcache: #fetch-and-add 0 #acquire() 4
--- top 5 contended locks:
lock: proc: #fetch-and-add 237672 #acquire() 124511
lock: proc: #fetch-and-add 210273 #acquire() 124565
lock: proc: #fetch-and-add 208869 #acquire() 124510
lock: proc: #fetch-and-add 161498 #acquire() 124510
lock: proc: #fetch-and-add 159713 #acquire() 124510
tot= 0
test1 OK
start test2
total free number of pages: 32496 (out of 32768)
.....
test2 OK
```

```
$ usertests sbrkmuch
usertests starting
test sbrkmuch: OK
ALL TESTS PASSED
```

- sbrkmuch

```
test opentest: OK
test writetest: OK
test writebig: OK
test createtest: OK
test openinput: 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
```

- usertests

1.2 磁盘缓存

1.2.1 实验步骤

- a. 修改bcache，将所有buf分配到NBUCKETS个桶中，并对每个桶维护一个锁。同时还需要保存原本的lock，用于保证只有一个线程由于对应的桶中为空，对其他桶进行查找，以避免死锁。

```
struct {
    struct buf buf[NBUF];
    struct spinlock lock;

    // Linked list of all buffers, through prev/next.
    // Sorted by how recently the buffer was used.
    // head.next is most recent, head.prev is least.
    struct buf heads[NBUCKETS];
    struct spinlock locks[NBUCKETS];
} bcache;
```

- b. 添加哈希函数将blockno映射到0到NBUCKETS-1。

```

int hash(uint blockno) {
    uint prime = 31;
    uint hash = blockno * prime;
    return hash % NBUCKETS;
}

```

- c. 修改binit，对各个桶以及锁进行初始化，并将所有的块先分配给第一个桶。

```

void binit(void)
{
    // TODO: modify here
    struct buf *b;

    initlock(&bcache.lock, "bcache");

    // initlock(&bcache.lock, "bcache");
    for(int i = 0; i < NBUCKETS; i++){
        initlock(&bcache.locks[i], "bcache");
        // Create linked list of buffers
        bcache.heads[i].prev = &bcache.heads[i];
        bcache.heads[i].next = &bcache.heads[i];
    }

    for(b = bcache.buf; b < bcache.buf+NBUF; b++){
        b->next = bcache.heads[0].next;
        b->prev = &bcache.heads[0];
        initsleeplock(&b->lock, "buffer");
        bcache.heads[0].next->prev = b;
        bcache.heads[0].next = b;
    }
}

```

- d. 修改bget，首先计算哈希得到对应的桶，然后在该桶中查找是否已经缓存。若没有缓存，则需要在该桶中查找空块。若该桶中不存在空块，则需要到其余桶中获取空块，此时需要先获取lock锁以避免死锁(如a中所述)，找到后将其加入该桶中并返回。

```

static struct buf* bget(uint dev, uint blockno) {
    // TODO: 修改bget() 和 brelse() 使得缓存区并发的查询和释放不容易发生锁争用
    struct buf *b;
    int id = hash(blockno);

    acquire(&bcache.locks[id]);

    // Is the block already cached?
    for(b = bcache.heads[id].next; b != &bcache.heads[id]; b = b->next){
        if(b->dev == dev && b->blockno == blockno){
            b->refcnt++;

```

```

        release(&bcache.locks[id]);
        acquiresleep(&b->lock);
        return b;
    }
}

// Not cached.
// Recycle the least recently used (LRU) unused buffer.
for(b = bcache.heads[id].prev; b != &bcache.heads[id]; b = b->prev){
    if(b->refcnt == 0) {
        b->dev = dev;
        b->blockno = blockno;
        b->valid = 0;
        b->refcnt = 1;
        release(&bcache.locks[id]);
        acquiresleep(&b->lock);
        return b;
    }
}
acquire(&bcache.lock);
for(int i = 0; i < NBUCKETS; i++){
    if(i == id){
        continue;
    }
    acquire(&bcache.locks[i]);
    for(b = bcache.heads[i].prev; b != &bcache.heads[i]; b = b->prev){
        if(b->refcnt == 0){
            b->dev = dev;
            b->blockno = blockno;
            b->valid = 0;
            b->refcnt = 1;

            b->prev->next = b->next;
            b->next->prev = b->prev;

            b->next = bcache.heads[id].next;
            b->prev = &bcache.heads[id];
            bcache.heads[id].next->prev = b;
            bcache.heads[id].next = b;

            release(&bcache.locks[id]);
            release(&bcache.locks[i]);
            release(&bcache.lock);
            acquiresleep(&b->lock);
            return b;
        }
    }
    release(&bcache.locks[i]);
}
release(&bcache.locks[id]);
release(&bcache.lock);
panic("bget: no buffers");
}

```

- e. 修改brelse、bpin、bunpin, 将head替换为对应的桶。

```

void
brelse(struct buf *b)
{
    if(!holdingsleep(&b->lock))
        panic("brelse");

    releasesleep(&b->lock);

    int id = hash(b->blockno);

    acquire(&bcache.locks[id]);
    b->refcnt--;
    if (b->refcnt == 0) {
        // no one is waiting for it.
        b->next->prev = b->prev;
        b->prev->next = b->next;
        b->next = bcache.heads[id].next;
        b->prev = &bcache.heads[id];
        bcache.heads[id].next->prev = b;
        bcache.heads[id].next = b;
    }

    release(&bcache.locks[id]);
}

void
bpin(struct buf *b) {
    int id = hash(b->blockno);
    acquire(&bcache.locks[id]);
    b->refcnt++;
    release(&bcache.locks[id]);
}

void
bunpin(struct buf *b) {
    int id = hash(b->blockno);
    acquire(&bcache.locks[id]);
    b->refcnt--;
    release(&bcache.locks[id]);
}

```

1.2.2 实验测试结果

- bcachetest

```
$ bcachetest
start test0
test0 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 32965
lock: kmem: #fetch-and-add 0 #acquire() 58
lock: kmem: #fetch-and-add 0 #acquire() 61
lock: bcache: #fetch-and-add 0 #acquire() 19
lock: bcache: #fetch-and-add 0 #acquire() 6195
lock: bcache: #fetch-and-add 0 #acquire() 5650
lock: bcache: #fetch-and-add 0 #acquire() 6464
lock: bcache: #fetch-and-add 0 #acquire() 4170
lock: bcache: #fetch-and-add 0 #acquire() 4530
lock: bcache: #fetch-and-add 29 #acquire() 6178
lock: bcache: #fetch-and-add 0 #acquire() 2208
lock: bcache: #fetch-and-add 0 #acquire() 6318
lock: bcache: #fetch-and-add 0 #acquire() 6184
lock: bcache: #fetch-and-add 0 #acquire() 2220
lock: bcache: #fetch-and-add 0 #acquire() 6324
lock: bcache: #fetch-and-add 0 #acquire() 4528
lock: bcache: #fetch-and-add 0 #acquire() 4248
--- top 5 contended locks:
lock: virtio_disk: #fetch-and-add 7615564 #acquire() 1209
lock: proc: #fetch-and-add 1089171 #acquire() 112595
lock: proc: #fetch-and-add 466924 #acquire() 112222
lock: proc: #fetch-and-add 464485 #acquire() 112222
lock: proc: #fetch-and-add 450513 #acquire() 112223
tot= 29
test0: OK
start test1
test1 OK
```


- usertests

```
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
```

1.3 实验测试结果

```
== Test running kallocetest ==
$ make qemu-gdb
(71.5s)
== Test    kallocetest: test1 ==
    kallocetest: test1: OK
== Test    kallocetest: test2 ==
    kallocetest: test2: OK
== Test kallocetest: sbrkmuch ==
$ make qemu-gdb
kallocetest: sbrkmuch: OK (9.8s)
== Test running bcachetest ==
$ make qemu-gdb
(21.4s)
== Test    bcachetest: test0 ==
    bcachetest: test0: OK
== Test    bcachetest: test1 ==
    bcachetest: test1: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (173.0s)
== Test time ==
time: OK
Score: 70/70
```