SYSU 18364012 陈鑫锐

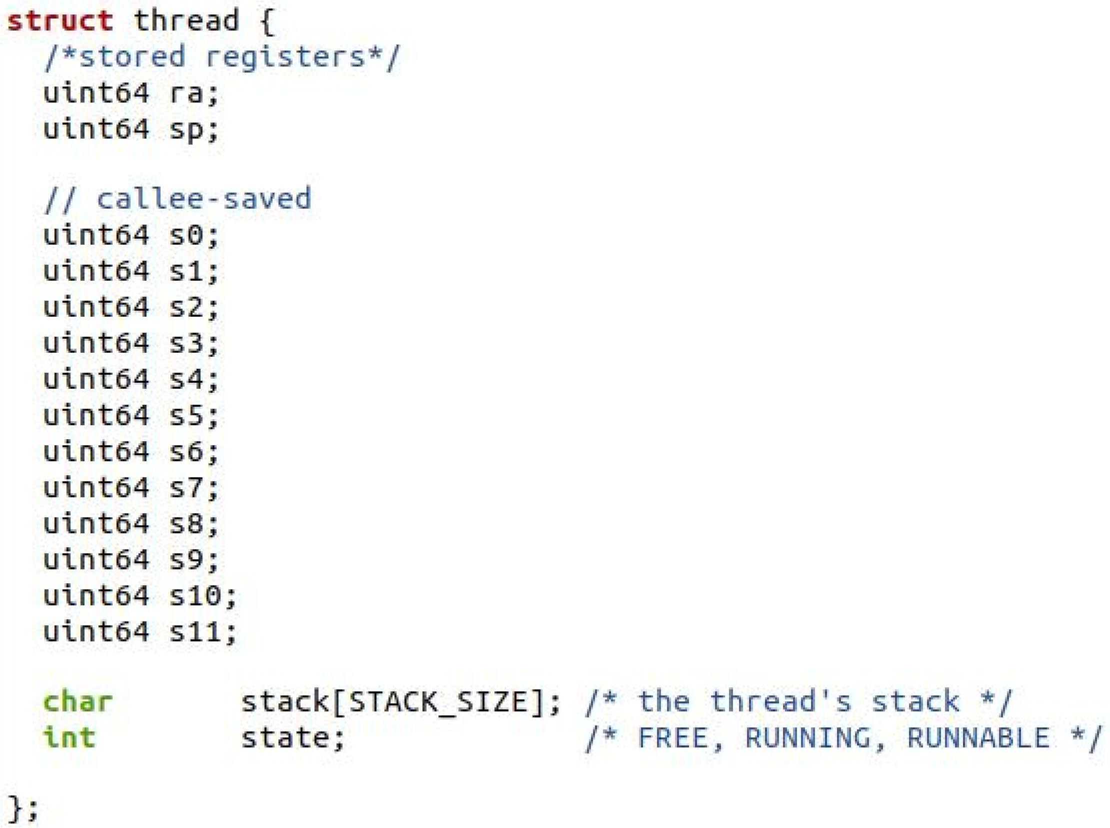
OS ASSIGNMENT3

1. Xv6 lab: Multithreading/Uthread: switching between threads

第一步：在 uthread_switch.S 中实现 thread_switch 函数，参考 kernel/switch.S，保存当前线程的寄存器，恢复即将要切换到线程的寄存器

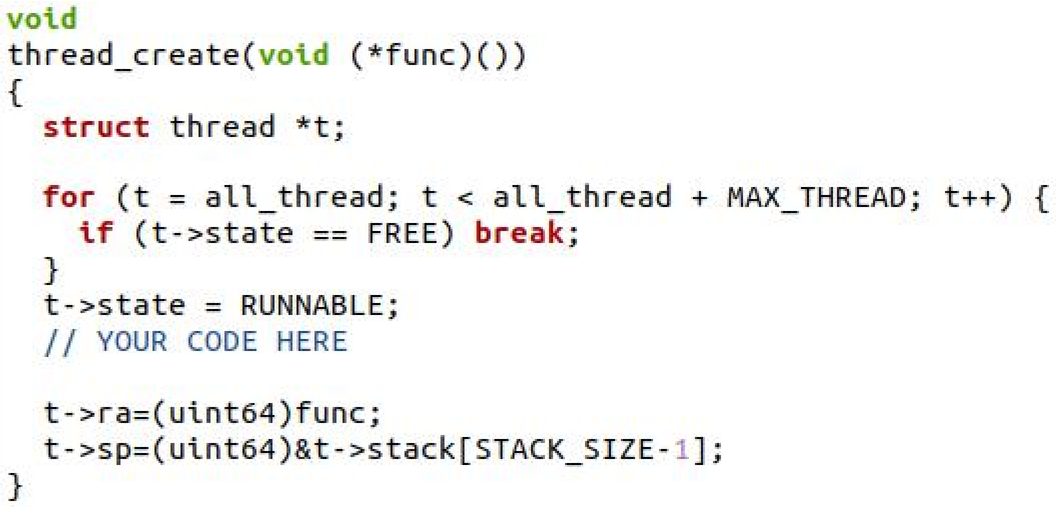
```
thread_switch:
         /* YOUR CODE HERE */
       sd ra, 0(a0)
       sd sp, 8(a0)
       sd s0, 16(a0)
       sd s1, 24(a0)
       sd s2, 32(a0)
       sd s3, 40(a0)
       sd s4, 48(a0)
       sd s5, 56(a0)
       sd s6, 64(a0)
       sd s7, 72(a0)
       sd s8, 80(a0)
       sd s9, 88(a0)
       sd s10, 96(a0)
       sd s11, 104(a0)

       ld ra, 0(a1)
       ld sp, 8(a1)
       ld s0, 16(a1)
       ld s1, 24(a1)
       ld s2, 32(a1)
       ld s3, 40(a1)
       ld s4, 48(a1)
       ld s5, 56(a1)
       ld s6, 64(a1)
       ld s7, 72(a1)
       ld s8, 80(a1)
       ld s9, 88(a1)
       ld s10, 96(a1)
       ld s11, 104(a1)
       ret
```

第二步：在 uthread.c 的 struct thread 添加寄存器字段



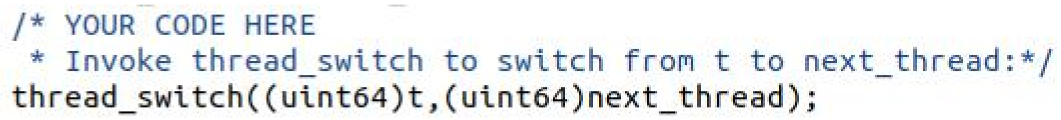
```
struct thread {
  /*stored registers*/
  uint64 ra;
  uint64 sp;

  // callee-saved
  uint64 s0;
  uint64 s1;
  uint64 s2;
  uint64 s3;
  uint64 s4;
  uint64 s5;
  uint64 s6;
  uint64 s7;
  uint64 s8;
  uint64 s9;
  uint64 s10;
  uint64 s11;

  char       stack[STACK_SIZE]; /* the thread's stack */
  int        state;             /* FREE, RUNNING, RUNNABLE */

};
```

第三步：在 thread_create()中添加保存新线程返回地址和栈指针，返回地址就是输入的函数指针 func，栈指针指向 struct thread->stack 的最后一个元素的地址，因为栈指针是从高地址向低地址增长的



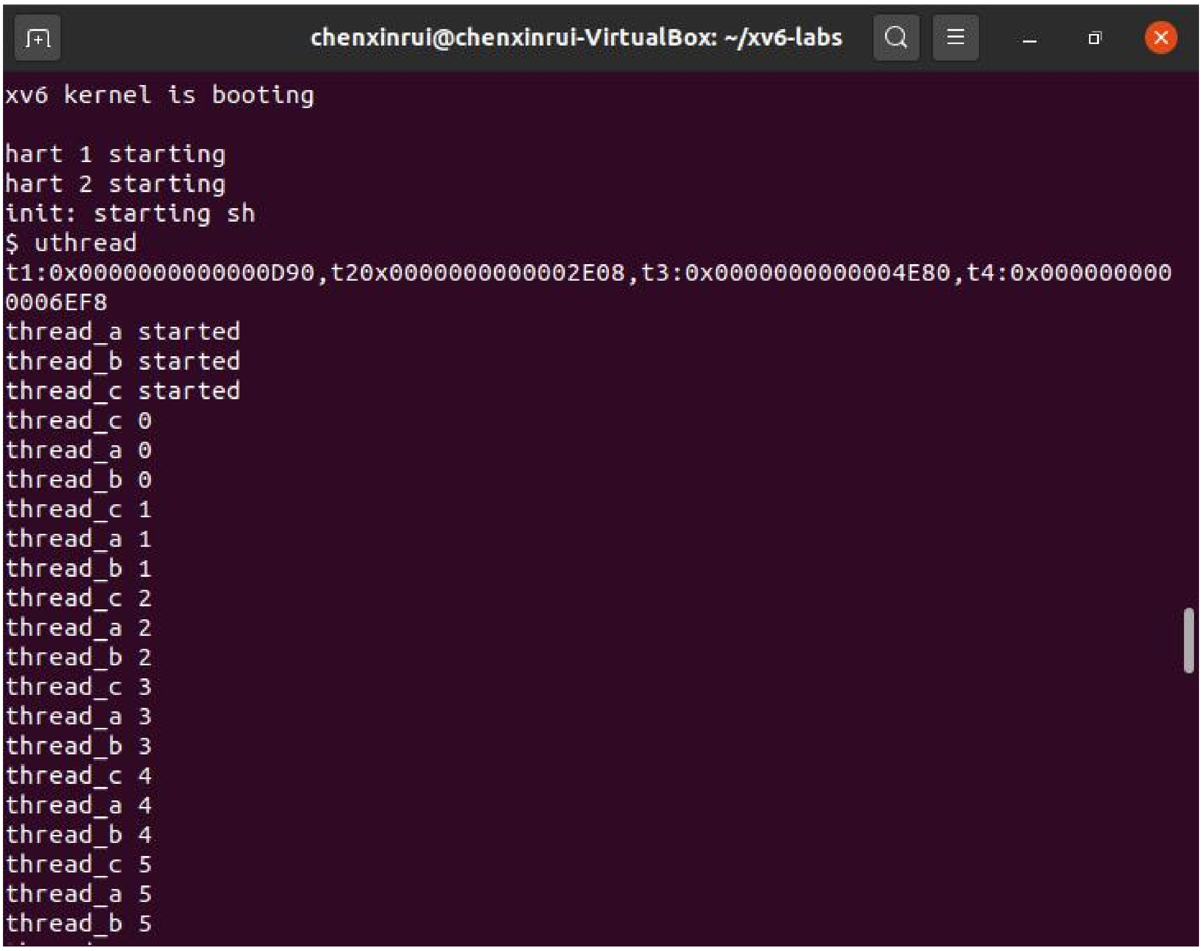
```
void
thread_create(void (*func)())
{
  struct thread *t;

  for (t = all_thread; t < all_thread + MAX_THREAD; t++) {
    if (t->state == FREE) break;
  }
  t->state = RUNNABLE;
  // YOUR CODE HERE

  t->ra=(uint64)func;
  t->sp=(uint64)&t->stack[STACK_SIZE-1];
}
```

第四步：最后在 thread_schedule()中添加调用

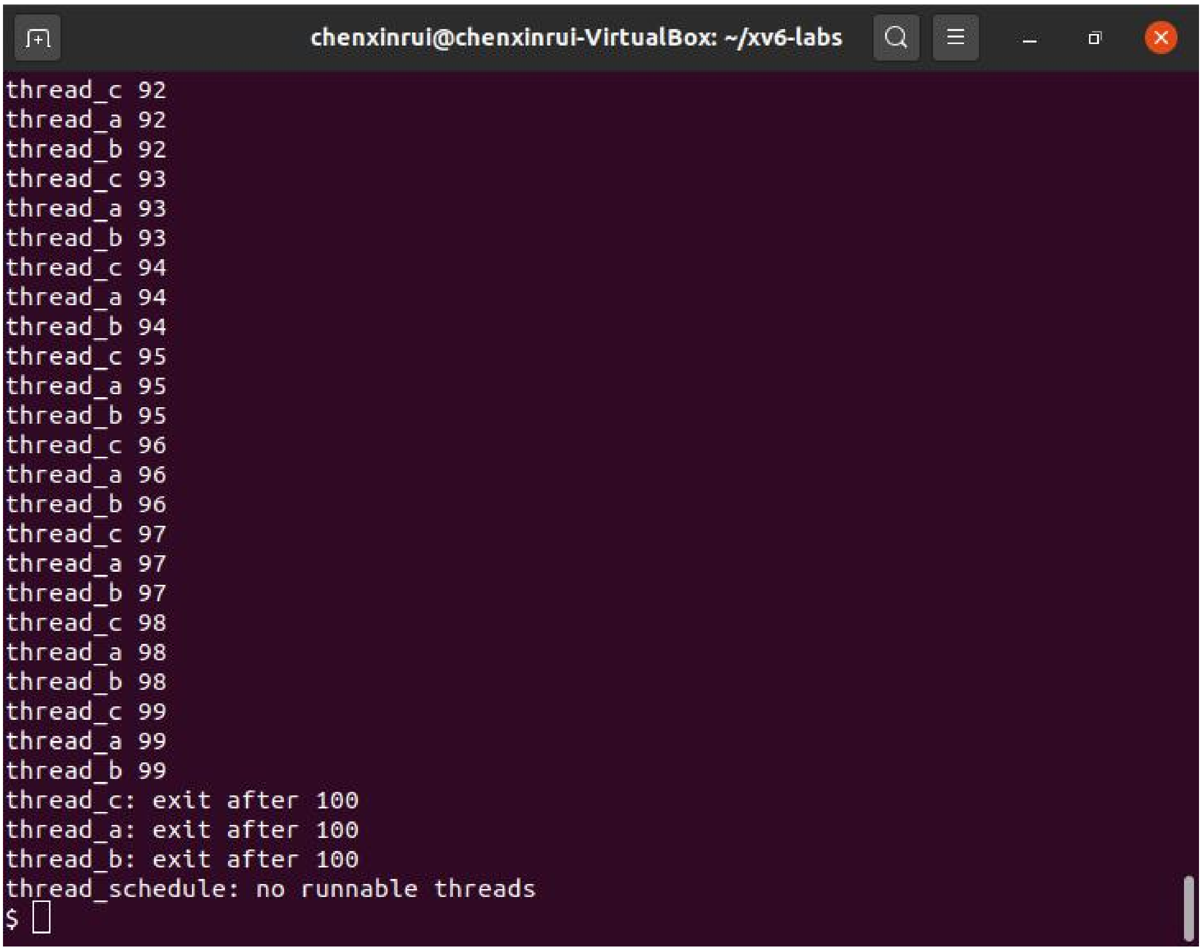


```
/* YOUR CODE HERE
 * Invoke thread_switch to switch from t to next_thread:*/
thread_switch((uint64)t,(uint64)next_thread);
```

运行结果：



```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ uthread
t1:0x0000000000000D90,t20x0000000000002E08,t3:0x0000000000004E80,t4:0x000000000
0006EF8
thread_a started
thread_b started
thread_c started
thread_c 0
thread_a 0
thread_b 0
thread_c 1
thread_a 1
thread_b 1
thread_c 2
thread_a 2
thread_b 2
thread_c 3
thread_a 3
thread_b 3
thread_c 4
thread_a 4
thread_b 4
thread_c 5
thread_a 5
thread_b 5
```



```
thread_c 92
thread_a 92
thread_b 92
thread_c 93
thread_a 93
thread_b 93
thread_c 94
thread_a 94
thread_b 94
thread_c 95
thread_a 95
thread_b 95
thread_c 96
thread_a 96
thread_b 96
thread_c 97
thread_a 97
thread_b 97
thread_c 98
thread_a 98
thread_b 98
thread_c 99
thread_a 99
thread_b 99
thread_c: exit after 100
thread_a: exit after 100
thread_b: exit after 100
thread_schedule: no runnable threads
$ 
```

2. Xv6 lab: Lock/Memory allocator

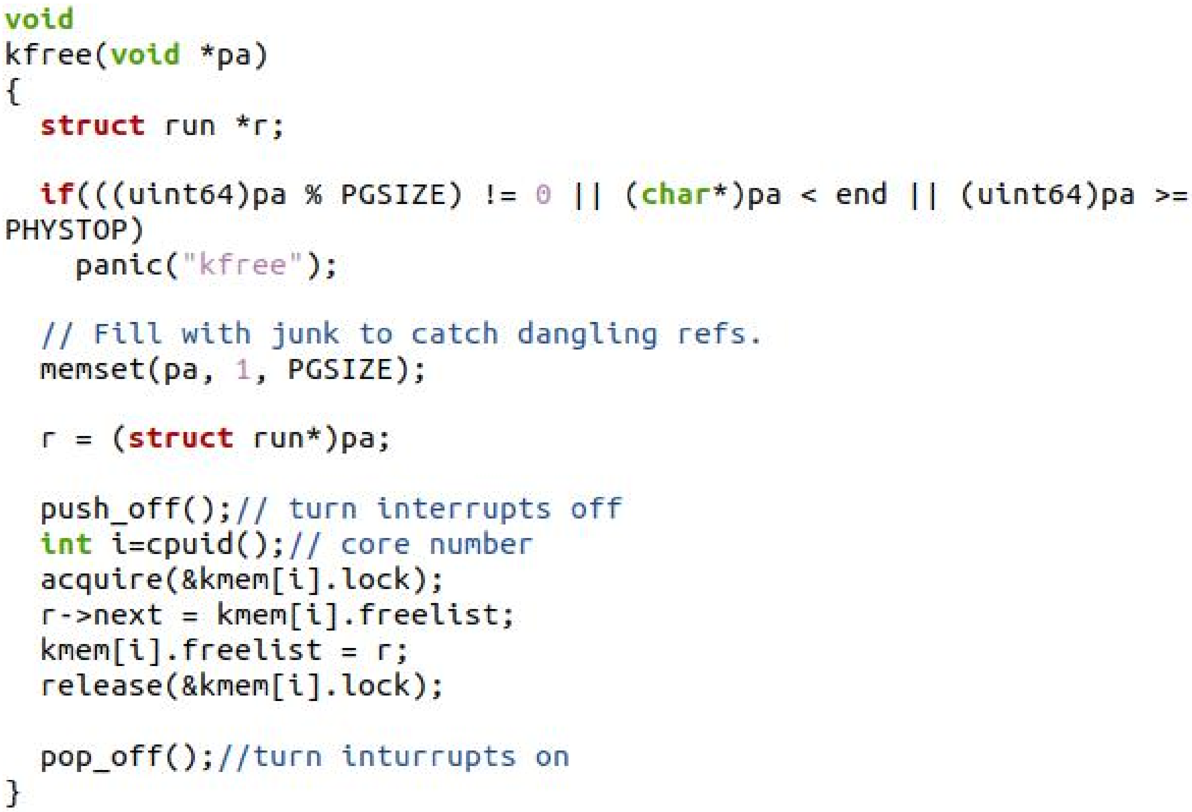
第一步：在 kalloc.c 中首先将 kmem 修改为数组，这样每个 cpu 对应一份 freelist 和 lock

```
struct {
  struct spinlock lock;
  struct run *freelist;
} kmem[NCPU];
```

第二步：初始化 kmem 时将每个 cpu 对应 kmem[i]都初始化

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

第三步：修改 kfree 代码



```
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();// turn interrupts off
  int i=cpuid();// core number
  acquire(&kmem[i].lock);
  r->next = kmem[i].freelist;
  kmem[i].freelist = r;
  release(&kmem[i].lock);

  pop_off();//turn inturrupts on
}
```

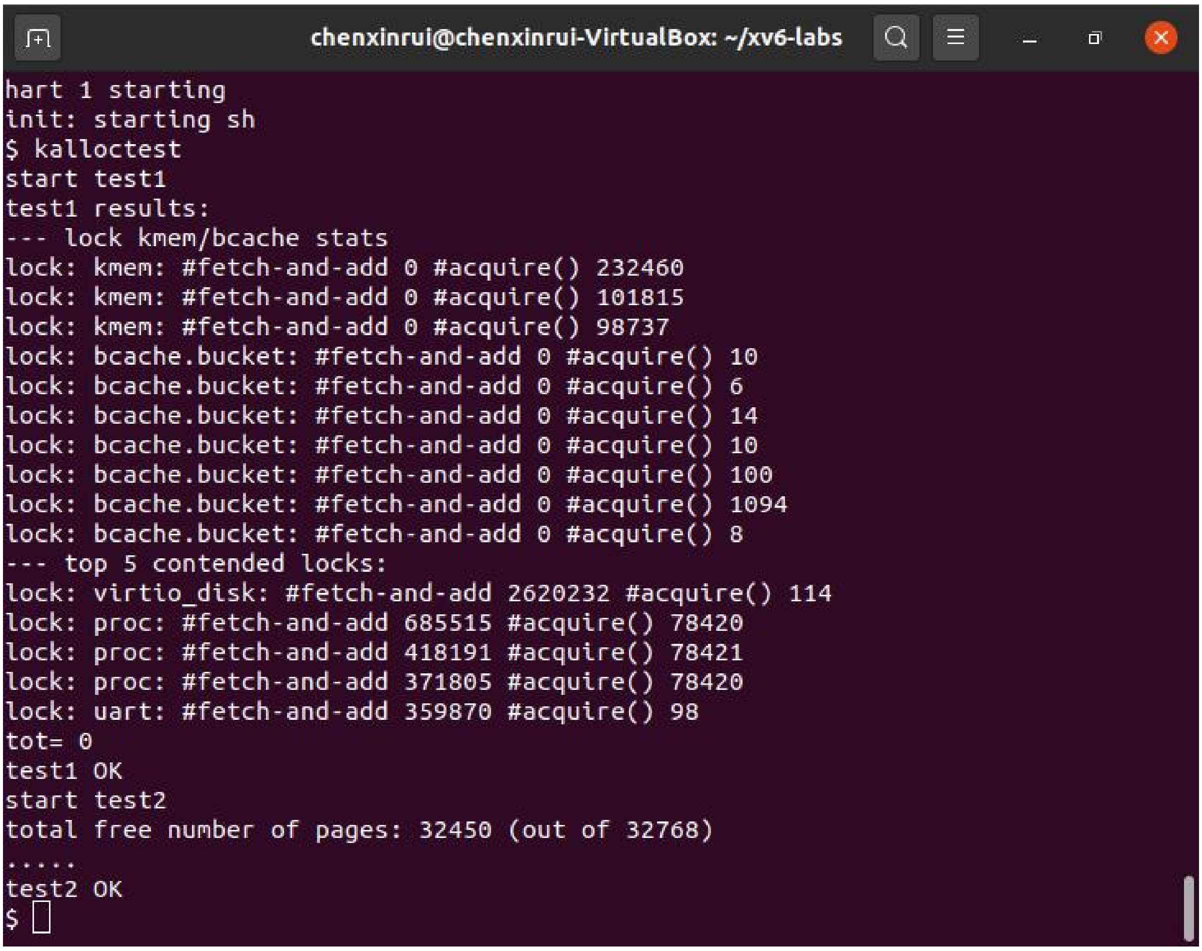
第四步：修改 kalloc 代码

```
void *kalloc(void){
  struct run *r;
  push_off();// turn interrupts off
  int i=cpuid();// core number
  acquire(&kmem[i].lock);
  r = kmem[i].freelist;
  if(r)
    kmem[i].freelist = r->next;
  release(&kmem[i].lock);
  if(!r){//current cpu->freelist is empty
     for(int j=0;j<NCPU;j++){//borrow from other cpu->freelist
       if(j!=i){
          acquire(&kmem[j].lock);
          if(kmem[j].freelist){
              r=kmem[j].freelist;
              kmem[j].freelist=r->next;
              release(&kmem[j].lock);
              break;
          }
          release(&kmem[j].lock);
```

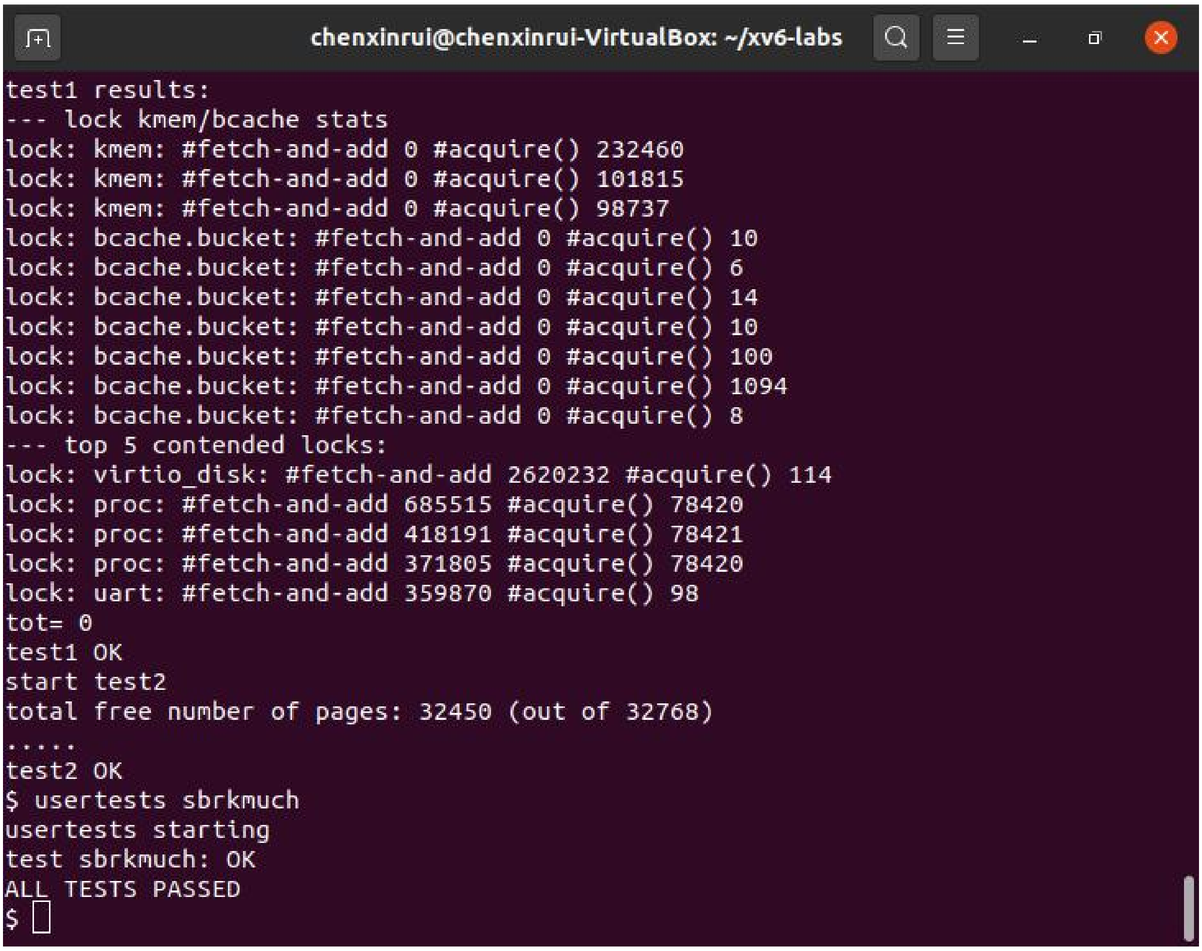
```
      }
    }
  }
  pop_off();//turn on inturrupt
  if(r)
    memset((char*)r, 5, PGSIZE); // fill with junk
  return (void*)r;
}
```

运行结果:

运行 kalloctest 以查看您的实现是否减少了锁争用，并运行 usertests sbrkmuch 来检查它是否仍可以分配所有内存。



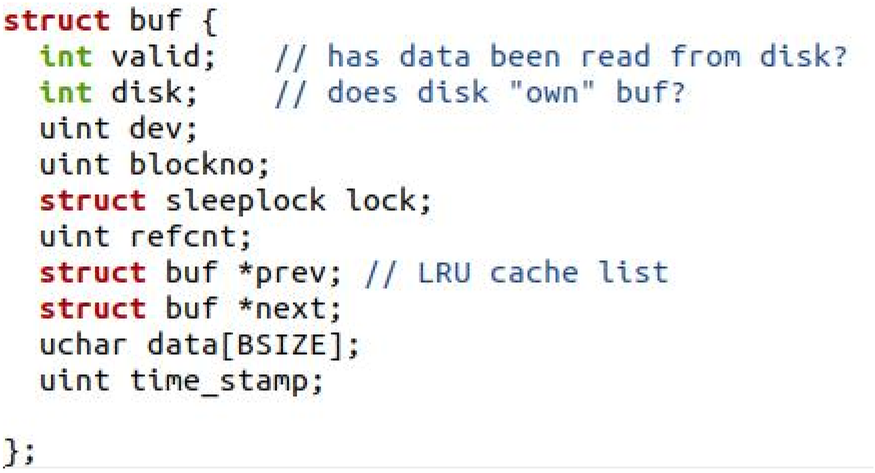
```
hart 1 starting
init: starting sh
$ kalloctest
start test1
test1 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 232460
lock: kmem: #fetch-and-add 0 #acquire() 101815
lock: kmem: #fetch-and-add 0 #acquire() 98737
lock: bcache.bucket: #fetch-and-add 0 #acquire() 10
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6
lock: bcache.bucket: #fetch-and-add 0 #acquire() 14
lock: bcache.bucket: #fetch-and-add 0 #acquire() 10
lock: bcache.bucket: #fetch-and-add 0 #acquire() 100
lock: bcache.bucket: #fetch-and-add 0 #acquire() 1094
lock: bcache.bucket: #fetch-and-add 0 #acquire() 8
--- top 5 contended locks:
lock: virtio_disk: #fetch-and-add 2620232 #acquire() 114
lock: proc: #fetch-and-add 685515 #acquire() 78420
lock: proc: #fetch-and-add 418191 #acquire() 78421
lock: proc: #fetch-and-add 371805 #acquire() 78420
lock: uart: #fetch-and-add 359870 #acquire() 98
tot= 0
test1 OK
start test2
total free number of pages: 32450 (out of 32768)
.....
test2 OK
$ 
```



```
test1 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 232460
lock: kmem: #fetch-and-add 0 #acquire() 101815
lock: kmem: #fetch-and-add 0 #acquire() 98737
lock: bcache.bucket: #fetch-and-add 0 #acquire() 10
lock: bcache.bucket: #fetch-and-add 0 #acquire() 6
lock: bcache.bucket: #fetch-and-add 0 #acquire() 14
lock: bcache.bucket: #fetch-and-add 0 #acquire() 10
lock: bcache.bucket: #fetch-and-add 0 #acquire() 100
lock: bcache.bucket: #fetch-and-add 0 #acquire() 1094
lock: bcache.bucket: #fetch-and-add 0 #acquire() 8
--- top 5 contended locks:
lock: virtio_disk: #fetch-and-add 2620232 #acquire() 114
lock: proc: #fetch-and-add 685515 #acquire() 78420
lock: proc: #fetch-and-add 418191 #acquire() 78421
lock: proc: #fetch-and-add 371805 #acquire() 78420
lock: uart: #fetch-and-add 359870 #acquire() 98
tot= 0
test1 OK
start test2
total free number of pages: 32450 (out of 32768)
.....
test2 OK
$ usertests sbrkmuch
usertests starting
test sbrkmuch: OK
ALL TESTS PASSED
$ 
```

3. Xv6 lab: Lock/Buffer cache

第一步：在 kernel/buf.h 的 struct 中添加字段 time_stamp，用以标记 buf 的时间戳

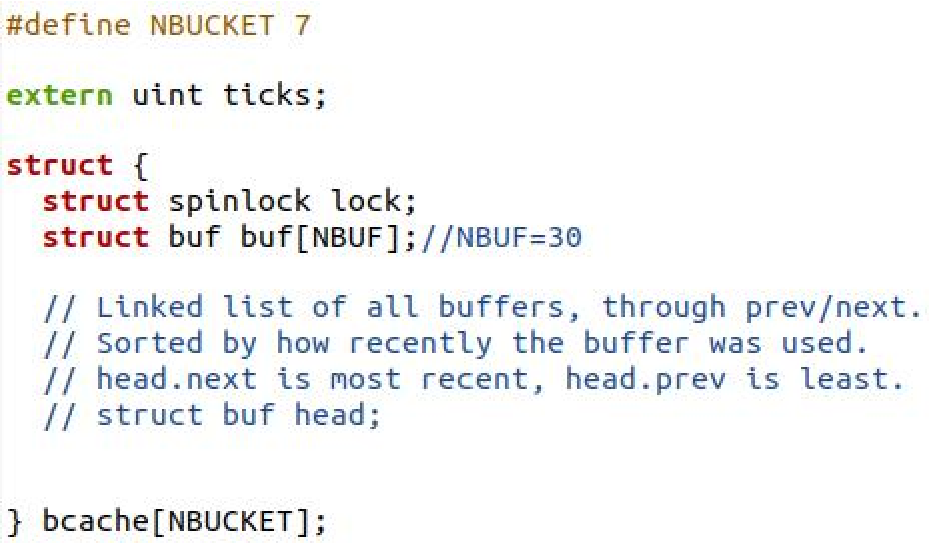


```
struct buf {
  int valid;   // has data been read from disk?
  int disk;    // does disk "own" buf?
  uint dev;
  uint blockno;
  struct sleeplock lock;
  uint refcnt;
  struct buf *prev; // LRU cache list
  struct buf *next;
  uchar data[BSIZE];
  uint time_stamp;

};
```

第二步：bget()中可能出现两种情况：hit(命中)，eviction（驱逐之前的替换新的）。

在这里将 bcache 设计成如下形式，其中 NBUCKET 需要小一点，因为锁的数目有限，

而 binit 至少需要 NBUF*NBUCKET+NBUCKET 个锁

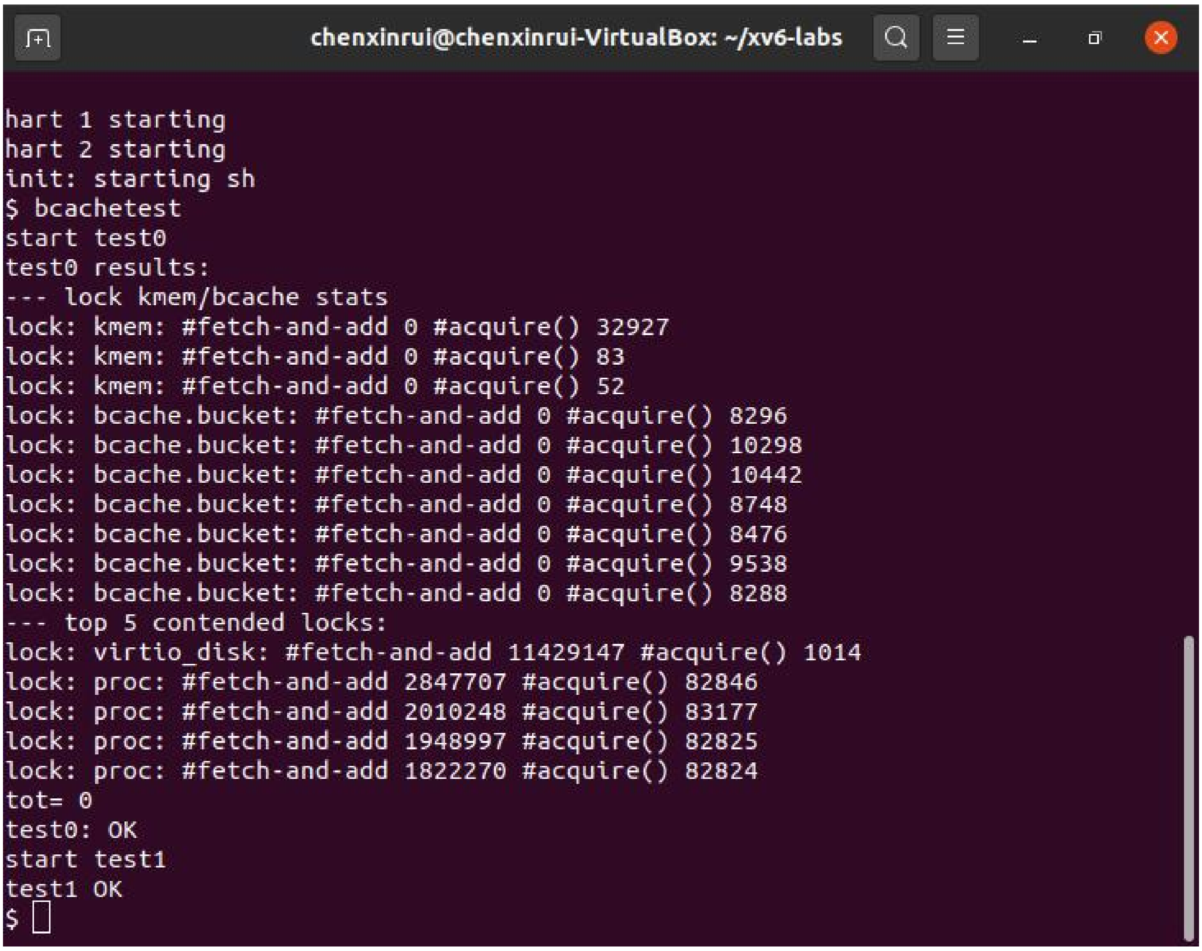


```
#define NBUCKET 7

extern uint ticks;

struct {
  struct spinlock lock;
  struct buf buf[NBUF];//NBUF=30

  // 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 head;


} bcache[NBUCKET];
```

第三步：修改后的 kernel/bio.c 如下

```
uint idx(uint blockno){
    return blockno % NBUCKET;
}
void binit(void){
    struct buf *b;
    for(int i=0;i<NBUCKET;i++){
        for(b = bcache[i].buf; b < bcache[i].buf+NBUF; b++){
            initsleeplock(&b->lock, "buffer");
        }
        initlock(&bcache[i].lock, "bcache.bucket");
    }
}
static struct buf* bget(uint dev, uint blockno){
    struct buf *b=0;
    int i= idx(blockno);
    uint min_time_stamp=-1;
    struct buf *min_b=0;
    acquire(&bcache[i].lock);//only need to hold one lock to avoid competition
    // Is the block already cached?
    for(b = bcache[i].buf; b < bcache[i].buf+NBUF; b++){
        if(b->dev==dev && b->blockno == blockno){//hit
            b->refcnt++;
            release(&bcache[i].lock);
            acquiresleep(&b->lock);
            return b;
        }//find the buf according to the smallest time_stamp
        if(b->refcnt==0 && b->time_stamp<min_time_stamp){
            min_time_stamp=b->time_stamp;
            min_b=b;
        }
    }
    b=min_b;
    if(b!=0){
        b->dev = dev;
        b->blockno = blockno;
        b->valid = 0;
        b->refcnt = 1;
        release(&bcache[i].lock);
        acquiresleep(&b->lock);
        return b;
    }
    panic("bget: no buffers");}
void brelse(struct buf *b){
    if(!holdingsleep(&b->lock))
        panic("brelse");
    releasesleep(&b->lock);
    int i=idx(b->blockno);
    acquire(&bcache[i].lock);
    b->refcnt--;
    if(b->refcnt == 0){// no one is waiting for it.
        b->time_stamp=ticks;
    }
  release(&bcache[i].lock);}
void bpin(struct buf *b){
    int i=idx(b->blockno);
    acquire(&bcache[i].lock);
    b->refcnt++;
    release(&bcache[i].lock);
}
void bunpin(struct buf *b) {
    int i=idx(b->blockno);
    acquire(&bcache[i].lock);
    b->refcnt--;
    release(&bcache[i].lock);
}
```

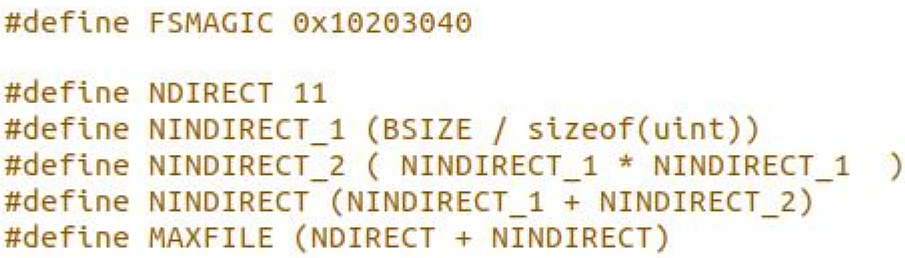
运行结果：



```
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() 32927
lock: kmem: #fetch-and-add 0 #acquire() 83
lock: kmem: #fetch-and-add 0 #acquire() 52
lock: bcache.bucket: #fetch-and-add 0 #acquire() 8296
lock: bcache.bucket: #fetch-and-add 0 #acquire() 10298
lock: bcache.bucket: #fetch-and-add 0 #acquire() 10442
lock: bcache.bucket: #fetch-and-add 0 #acquire() 8748
lock: bcache.bucket: #fetch-and-add 0 #acquire() 8476
lock: bcache.bucket: #fetch-and-add 0 #acquire() 9538
lock: bcache.bucket: #fetch-and-add 0 #acquire() 8288
--- top 5 contended locks:
lock: virtio_disk: #fetch-and-add 11429147 #acquire() 1014
lock: proc: #fetch-and-add 2847707 #acquire() 82846
lock: proc: #fetch-and-add 2010248 #acquire() 83177
lock: proc: #fetch-and-add 1948997 #acquire() 82825
lock: proc: #fetch-and-add 1822270 #acquire() 82824
tot= 0
test0: OK
start test1
test1 OK
$ 
```

4. Xv6 lab: File System/Large files

第一步：首先修改 fs.h 中的宏定义，NINDIRECT_1 是第一级间接访问的 block 数目，NINDIRECT_2 是第二级间接访问的 block 数目



```
#define FSMAGIC 0x10203040

#define NDIRECT 11
#define NINDIRECT_1 (BSIZE / sizeof(uint))
#define NINDIRECT_2 ( NINDIRECT_1 * NINDIRECT_1  )
#define NINDIRECT (NINDIRECT_1 + NINDIRECT_2)
#define MAXFILE (NDIRECT + NINDIRECT)
```

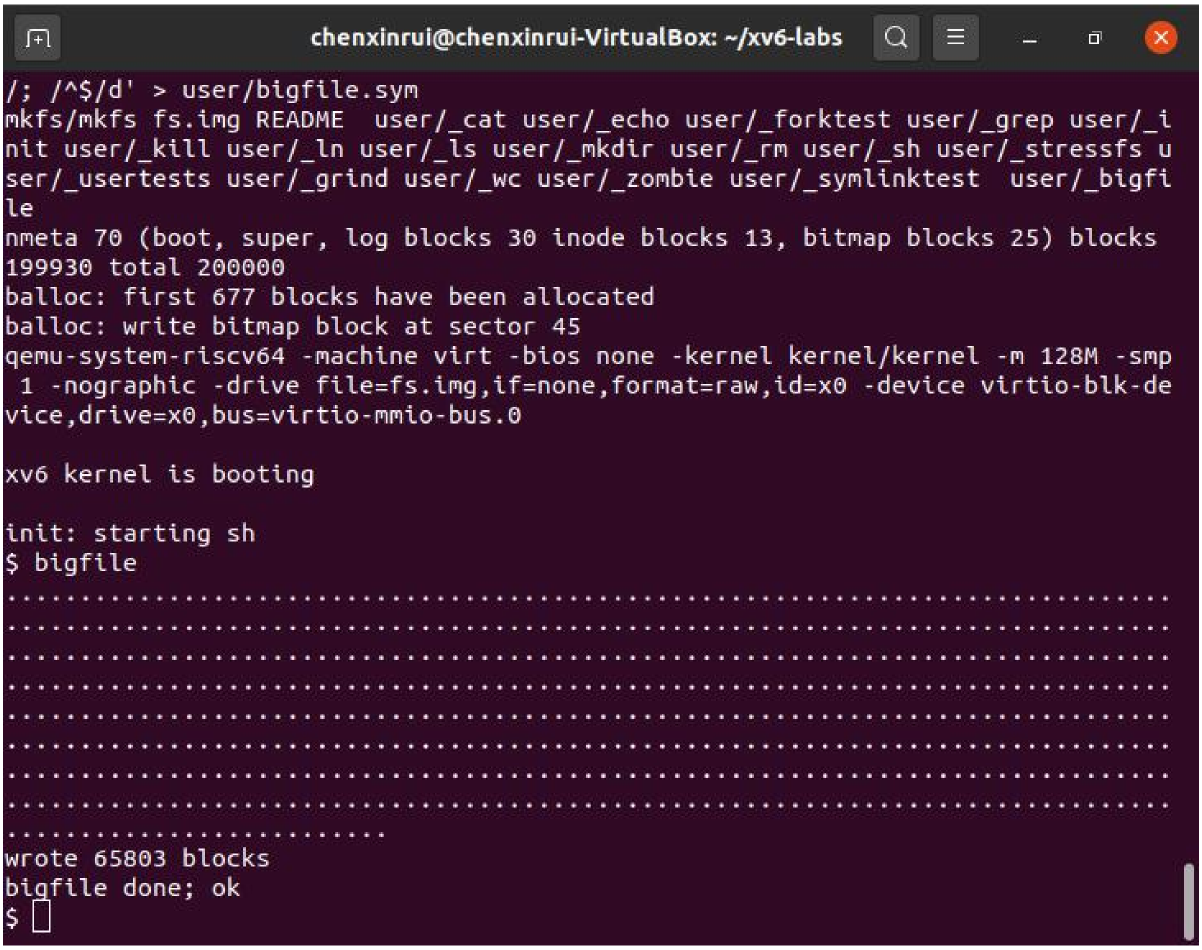
第二步：修改bmap()，使其除了直接块和单间接块之外还实现双间接块。ip->addrs[]的前11个元素应该是直接块；第十二个应该是一个间接块（就像当前的块一样）；第13个应该是一个新的双间接块。

```
static uint bmap(struct inode *ip, uint bn){
  uint addr, *a;
  struct buf *bp;
  if(bn < NDIRECT){
    if((addr = ip->addrs[bn]) == 0)
      ip->addrs[bn] = addr = balloc(ip->dev);
    return addr;
  }
  bn -= NDIRECT;
  if(bn < NINDIRECT_1){
    // Load first-level indirect block, allocating if necessary.
    if((addr = ip->addrs[NDIRECT]) == 0)
      ip->addrs[NDIRECT] = addr = balloc(ip->dev);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn]) == 0){
      a[bn] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    return addr;
  }
  bn -= NINDIRECT_1;
  if(bn < NINDIRECT_2){
    // Load second-level indirect block, allocating if necessary.
    if((addr = ip->addrs[NDIRECT+1]) == 0)
      ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    uint bn_1= ( bn & 0xff00)>>8 ;//level1
    uint bn_2= bn & 0xff;//level2
    if((addr = a[bn_1]) == 0){
      a[bn_1] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn_2]) == 0){
      a[bn_2] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    return addr;
  }
  panic("bmap: out of range");
}
```

第三步：确保 itrunc 释放文件的所有块，包括双间接块。

```
void itrunc(struct inode *ip){
  int i, j, k;
  struct buf *bp;
  struct buf *bp2;
  uint *a;
  uint *a2;
  for(i = 0; i < NDIRECT; i++){
    if(ip->addrs[i]){
      bfree(ip->dev, ip->addrs[i]);
      ip->addrs[i] = 0;
    }
  }
  if(ip->addrs[NDIRECT]){
    bp = bread(ip->dev, ip->addrs[NDIRECT]);
    a = (uint*)bp->data;
    for(j = 0; j < NINDIRECT_1; j++){
      if(a[j])
        bfree(ip->dev, a[j]);
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT]);
    ip->addrs[NDIRECT] = 0;
  }
  if(ip->addrs[NDIRECT+1]){
    bp = bread(ip->dev, ip->addrs[NDIRECT+1]);
    a = (uint*)bp->data;
    for(j = 0; j < NINDIRECT_1; j++){
      if(a[j]){
        bp2 = bread(ip->dev, a[j]);
        a2 = (uint*)bp2->data;
        for(k=0 ;k < NINDIRECT_1 ;k++){
            if(a2[k]) bfree(ip->dev,a2[k]);
        }
        brelse(bp2);
        bfree(ip->dev, a[j]);
      }
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT+1]);
    ip->addrs[NDIRECT+1] = 0;
  }
  ip->size = 0;
  iupdate(ip);
}
```

运行结果：



```
/; /^$/d' > user/bigfile.sym
mkfs/mkfs fs.img README  user/_cat user/_echo user/_forktest user/_grep user/_i
nit user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs u
ser/_usertests user/_grind user/_wc user/_zombie user/_symlinktest  user/_bigfi
le
nmeta 70 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 25) blocks
199930 total 200000
balloc: first 677 blocks have been allocated
balloc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
 1 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-de
vice,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ bigfile
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
..........................
wrote 65803 blocks
bigfile done; ok
$ 
```