Lab1

姓名: 陈锐林, 学号:21307130148

2023 年 9 月 26 日

实验一、内存分配实验

一、举例说明：

根据题意，如果没有锁可能会导致较严重的后果。例如，如果两个进程同时对同一资源进行修改，又没有锁保护，会导致数据不一致。

二、锁竞争：

未修改前，kalloc 只维护单个空闲链表，只有一个全局锁保护；虽然比较安全，但是也很可能出现锁竞争。同样地，如果多个线程同时分配一块内存，虽有锁的保护，但是会导致这些进程排队等候，导致性能下降。并且还可能导致死锁的问题出现；并且即使不涉及对同一内存访问，锁粒度太大也要很久等待。

三、push_off() and pop_off：

通过查看 xv6 的代码，以及查阅相关资料；得知这俩个函数的作用在于保证锁正常进行。push_off 函数用于禁用中断，保存 CPU 的中断状态，防止锁的获取被中断打断；pop_off 用于锁释放后，恢复之前保存的中断状态，允许重新中断。

四、实验：

1. 实验思路：

通过题目的 hint，可从 kmem 出发；并且利用变量 NCPU 和 cpuid() 函数。首先，要为每个 CPU 分配一个列表，所以要将 kmem 改成数组形式，而大小即为 NCPU；其次，在 kinit() 中要依次初始化；然后先解决简单的 kfree()，这里只要使用 cpuid() 函数得到将要 free 的 cpu 即可，其他和之前一样。最后看 kalloc() 函数，这里分配内存时，首先考虑当前 cpu(cpuid 得到) 的 freelist 是否有空间，有就直接插入；如果没有空间时，就要考察能不能利用其他 cpu 下的空间，遍历其余 cpu，如果某个页表还有空间就挪一半出来。

2. 代码设计：

(1) 对 kmem/kinit()/kfree 的修改：

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

(a) kmem

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

(b) kinit

```
push_off();
int id = cpuid();
acquire(&kmem[id].lock);
r->next = kmem[id].freelist;
kmem[id].freelist = r;
release(&kmem[id].lock);
pop_off();
```

(c) kfree

(2) 对 kalloc 的修改：这里需要说明的是 p1/p2/p3，其余就是正常的锁/开锁以及一些逻辑判断，基于有没有找到对应的内存。p1 是原始指针，p3 用来存储 p1 的值；而 p2 一次跑两格，意思是只取一半的空闲页表。

```
push_off();
int nowcpu = cpuid();
acquire(&kmem[nowcpu].lock);
r = kmem[nowcpu].freelist;
if(r)   //先考察有没有空间
  kmem[nowcpu].freelist = r->next;
else{
  int flag = 0; //表示是否成功找到
  int i = 0;  //遍历
  for(;i < NCPU;i++){
    if(i == nowcpu) continue;
    acquire(&kmem[i].lock); //锁!
    struct run *p1 = kmem[i].freelist;
    if(p1){  //如果还有剩
      struct run *p2 = p1;  //出头鸟 跑得快
      struct run *p3 = p1;
      while(p2 && p2->next){  //这里主要是借些来
        p2 = p2->next->next;
        p3 = p1;
        p1 = p1->next;
      }
```

(a) part1

```
      kmem[nowcpu].freelist = kmem[i].freelist;
      if(p1 == kmem[i].freelist){  //只有一页咯
        kmem[i].freelist = 0;
      }
      else{
        kmem[i].freelist = p1;
        p3->next = 0;
      }
      flag = 1;
    }
    release(&kmem[i].lock); //开锁
    if(flag){  //如果找到了就break
      r = kmem[nowcpu].freelist;
      kmem[nowcpu].freelist = r->next;
      break;
    }
  }
}
release(&kmem[nowcpu].lock);
pop_off();
```

(b) part2

3. 测试结果：

```
test1 OK
start test2
total free number of pages: 32497 (out of 32768)
.....
test2 OK
start test3
child done 1
child done 100000
test3 OK
```

实验二：同步互斥实验

1. 复旦早餐店：

(1) 请写出题目中的互斥与同步关系：

同步 1：鸡蛋灌饼进程和煎饼果子进程同时负责制作和售卖各自的早餐。

同步 2：顾客队伍 1 和顾客队伍 2 分别同步排队，确保了顾客按照自己的需求购买早餐。

互斥 1：老板和老板娘把做好的吃的放进篮子里这两件事是互斥的。

互斥 2：两个顾客队列中只有一个能买到吃的，这是互斥的。

(2) 伪代码：初始情况是篮子里已有一个吃的，以及假设是两列队伍都很长，所以老板和老板娘交替放鸡蛋灌饼和煎饼果子。并且顾客很多，不考虑顾客队列的人数变化（假设无穷无尽）

```
Process 1 #(鸡蛋灌饼):
    while(true):
    wait(3):
        if 篮子为空:
            make 鸡蛋灌饼, 放入篮子
            call(2)

Process 2 #(煎饼果子):
    while(true):
        wait(4)
        if 篮子为空:
            make 煎饼果子, 放入篮子
            call(1)
```

Process 3 #(买鸡蛋灌饼):

```
while(true):
    wait(1)
    if 篮子有鸡蛋灌饼:
        买走
```

Process 4 #(买煎饼果子):

```
while(true):
    wait(2)
    if 篮子有煎饼果子:
        买走
```

2. 哲学家就餐问题：

(1) 实现思路：

这个题目要求完成的部分其实不多，只要完成 philosopher 函数的部分即可。首先我们要定义当前索引为 i 的哲学家要拿起的筷子；其左手边为 i 号筷子，右手边为 (i%NUMBER_OF_PHILOSOPHERS)。第一部分是 pickUp() 函数，主要实现拿起筷子这一环节。可以直接调用 pthread 库中的函数，对于 i 号哲学家，先试着拿起左筷子（即锁住左筷子）；接着他进一步尝试锁住右筷子，如果返回错误，说明筷子正被人占用，那么他要主动放手（左右都要）；过段时间后继续试着拿起左筷子，再右筷子。长此往复，直到吃完一波了才进入到第二部分。第二部分为 putDown 函数，即吃完了主动放下筷子，解开锁，然后等待会进入下一阶段。

(2) 代码实现：(主要是 philosopher 函数)

```
void philosopher(int philosopherNumber) {
    while (1) {
        int i = (int)philosopherNumber;
        int left = i;
        int right = (i + 1) % NUMBER_OF_PHILOSOPHERS;
        think(philosopherNumber);
        pthread_mutex_lock(&chopsticks[left]);
        while(!pthread_mutex_trylock(&chopsticks[right])){
            pthread_mutex_unlock(&chopsticks[left]);
            sleep(rand()%3 + 1);
            pthread_mutex_lock(&chopsticks[left]);
        }
        eat(philosopherNumber);
        pthread_mutex_unlock(&chopsticks[left]);
        pthread_mutex_unlock(&chopsticks[right]);
        sleep(rand()%3 + 1);
    }
}
```

(3) 测试结果:

```
chenrl959@SK-20210701MSSI:~/NEWHW/lab1$ ./DiningPhilosopher
Philosopher 0 will think for 2 seconds
Philosopher 2 will think for 2 seconds
Philosopher 4 will think for 1 seconds
Philosopher 3 will think for 1 seconds
Philosopher 1 will think for 3 seconds
Philosopher 1 will eat for 3 seconds
Philosopher 2 will eat for 1 seconds
Philosopher 3 will eat for 1 seconds
Philosopher 1 will think for 2 seconds
Philosopher 4 will eat for 2 seconds
Philosopher 2 will think for 1 seconds
```