Data Structures & Conditional Variables

Data Structures & Conditional Variables
- 一、Data Structures
  - 1. counter
  - 2. Linked List
  - 3. Queue
  - 4. Hash table
  - 5. 小结
- 二、Conditional Variables
  - 0. spin
  - 1. Conditional Variables
  - 2. 实现producer-consumer
  - 3. Signal和wait之间的语义
    - Mesa
    - Hoare
  - 4. 需要broadcast

一、Data Structures

1. counter

- counter++需要保护，用锁。
- poor performance：拿锁放锁的时间比counter++多

Scalable Counting：可拓展的计数器

- 用很多local counter和一个global
- 高并发的write：write的时候各自写自己的
- 定期将local更新到global
- read
  - 等一等，读个精确的
  - 立马读个不精确的
- 阈值的设置sloppiness（平衡扩展性和精确性）

2. Linked List

Attention：在所有出口记得放锁

- 简单来说：来个global lock
- 并行度不高

Scaling Linked List：可扩展的

- 在每个node上面放个锁：允许一个list上有多个thread
- hand over hand locking：要把相关的node都锁上，比如插入时要抢到下一个结点的锁
- 遍历的时候很糟糕：每到一个node都要拿锁放锁

更多的并发不一定更快!

3. Queue

- 需要锁住head和tail：保护入列、出列，但是入和出两者之间不影响。
- 极端：queue为空的时候，head和tail一个地方。增加一个假节点guard

4. Hash table

Hash table+link list

使用上述的link list

5. 小结

用别人写好的数据结构

二、Conditional Variables

lock是保护critical code

conditional variable保护顺序：条件满足后才能执行

0. spin

```
while (done == 0) // done要到内存里读，防止reg缓存导致访问结果不一致
    ; // spin
```

1. Conditional Variables

定义：

- 显式的队列：可以一对一通知、按需通知、广播
- 等待、唤醒：wait()，signal()

```
void thr_exit() {
    Pthread_mutex_lock(&m);
    done = 1; //
    Pthread_cond_signal(&c);
    Pthread_mutex_unlock(&m);
}
void thr_join() {
    Pthread_mutex_lock(&m);
    while (done == 0)
        Pthread_cond_wait(&c, &m);
    Pthread_mutex_unlock(&m);
}

int main(int argc, char *argv[]) {
    printf("parent: begin\n");
    pthread_t p;
    Pthread_create(&p, NULL, child, NULL);
    thr_join();
    printf("parent: end\n");
```

```
    return 0;
}

void *child(void *arg) {
    printf("child\n");
    thr_exit();
    return NULL;
}
```

- 如果没有done：
  - 调度是随机的：可能先通知了，main才进入到wait。
  - 有done的话可以在睡觉前拦住（已通知的情况）
- 如果没有锁：
  - 在while的条件检查后，被context switch了

2. 实现producer-consumer

问题代码：

```
cond_t cond; //一个cv
mutex_t mutex;

void *producer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        Pthread_mutex_lock(&mutex);             // p1
        if (count == 1)                         // p2
              Pthread_cond_wait(&cond, &mutex);   // p3
        put(i);                                 // p4
        Pthread_cond_signal(&cond);              // p5
        Pthread_mutex_unlock(&mutex);            // p6
    }
}

void *consumer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        Pthread_mutex_lock(&mutex);             // c1
        if (count == 0)                         // c2
              Pthread_cond_wait(&cond, &mutex);   // c3
        int tmp = get();                         // c4
        Pthread_cond_signal(&cond);              // c5
        Pthread_mutex_unlock(&mutex);            // c6
        printf("%d\n", tmp);
    }
}
```

- 在1个consumer和1个producer下没问题
- p1~p3等待缓冲区有空位
- c1~c3等待缓冲区有东西

- 假设2个consumer和1个producer：
  - Tc1已经在c3等待了
  - 生产者唤醒Tc1
  - 但是Tc2把抢先执行，把东西拿走了。
  - Tc1再去get，出错
- fix：把line8，line20的if改成while，醒来后再去check一次

但是依然有问题

- 只有一个cv，没有指定唤醒谁
  - 两个消费者都等在c3
  - producer装好东西后，唤醒了Tc1，然后等在p3
  - Tc1消费完了，结果唤醒Tc2，自己等在c3
  - Tc2看了一眼，是空的，等在c3
  - 大家都睡着了……
- fix：两个cv，区分signal。指定producer要去叫consumer，consumer要去叫producer

```
1  cond_t empty, fill; // fix2
2  mutex_t mutex;
3
4  void *producer(void *arg) {
5      int i;
6      for (i = 0; i < loops; i++) {
7          Pthread_mutex_lock(&mutex);
8          while (count == 1) // fix1
9                Pthread_cond_wait(&empty, &mutex);
10         put(i);
11         Pthread_cond_signal(&fill);
12         Pthread_mutex_unlock(&mutex);
13     }
14 }
15
16 void *consumer(void *arg) {
17     int i;
18     for (i = 0; i < loops; i++) {
19         Pthread_mutex_lock(&mutex);
20          while (count == 0)
21               Pthread_cond_wait(&fill, &mutex);
22         int tmp = get();
23         Pthread_cond_signal(&empty);
24         Pthread_mutex_unlock(&mutex);
25         printf("%d\n", tmp);
26     }
27 }
```

如果是多个slot，只要修改get和put函数

3. Signal和wait之间的语义

Mesa

叫醒后，状态可能变化了。（大部分都是Mesa）

> 叫你的时候，buffer满了，但是醒来执行时，可能已经空了

需要醒来后再去check。

Hoare

保证叫醒到执行，状态不变（难实现）

4. 需要broadcast

在分配空间和使用空间的时候：

- 分配好了2MB，要叫醒
- 叫醒了需要4MB的人，需要1MB的人还在睡
- 4MB的人醒来发现不够，又睡了

区分不出来到底叫谁（或者必要麻烦）：`Pthread_cond_broadcast()`