

# 生产者消费者问题

杨朝辉

PB17071433

2019/11/24

## 模拟方式

- 利用Java中对“管程”这一系统原语的支持实现了临界区避免竞争条件，成功模拟了该问题。该程序是借助Java中的synchronized关键字实现的。
- 另外，也通过信号量的方式来控制线程间的同步互斥问题，同样成功模拟了生产消费问题。实际中是借助Java内置的处理多线程并发的concurrent包中的Semaphore实现的。

## 运行结果

### 管程方式

以下为基于管程原语模拟的缓冲区大小50、生产商品数量上限500时该问题的运行结果（生产的生产/消费记录文件），其中按照时间（精确到毫秒）先后记录了生产和消费记录。

```
1 producing NO.0 item at 2019-11-24 17:12:23:586
2 producing NO.1 item at 2019-11-24 17:12:23:652
3 consuming NO.0 item at 2019-11-24 17:12:23:652
4 consuming NO.1 item at 2019-11-24 17:12:23:653
5 producing NO.2 item at 2019-11-24 17:12:23:659
6 consuming NO.2 item at 2019-11-24 17:12:23:664
7 producing NO.3 item at 2019-11-24 17:12:23:667
8 producing NO.4 item at 2019-11-24 17:12:23:672
9 consuming NO.3 item at 2019-11-24 17:12:23:675
10 producing NO.5 item at 2019-11-24 17:12:23:681
11 consuming NO.4 item at 2019-11-24 17:12:23:682
12 consuming NO.5 item at 2019-11-24 17:12:23:685
13 producing NO.6 item at 2019-11-24 17:12:23:691
14 consuming NO.6 item at 2019-11-24 17:12:23:700
15 producing NO.7 item at 2019-11-24 17:12:23:700
16 consuming NO.7 item at 2019-11-24 17:12:23:707
17 producing NO.8 item at 2019-11-24 17:12:23:709
```

```

986      consuming NO.488 item at 2019-11-24 17:12:26:726
987      consuming NO.489 item at 2019-11-24 17:12:26:729
988      producing NO.497 item at 2019-11-24 17:12:26:733
989      producing NO.498 item at 2019-11-24 17:12:26:735
990      consuming NO.490 item at 2019-11-24 17:12:26:738
991      producing NO.499 item at 2019-11-24 17:12:26:742
992      consuming NO.491 item at 2019-11-24 17:12:26:745
993      consuming NO.492 item at 2019-11-24 17:12:26:748
994      consuming NO.493 item at 2019-11-24 17:12:26:757
995      consuming NO.494 item at 2019-11-24 17:12:26:765
996      consuming NO.495 item at 2019-11-24 17:12:26:770
997      consuming NO.496 item at 2019-11-24 17:12:26:777
998      consuming NO.497 item at 2019-11-24 17:12:26:777
999      consuming NO.498 item at 2019-11-24 17:12:26:780
000      consuming NO.499 item at 2019-11-24 17:12:26:785

```

## 信号量方式

以下为基于信号量同步原语的同样参数的模拟结果。

```

1      producing NO.0 item at 2019-11-24 19:39:42:071
2      consuming NO.0 item at 2019-11-24 19:39:42:126
3      producing NO.1 item at 2019-11-24 19:39:42:129
4      consuming NO.1 item at 2019-11-24 19:39:42:136
5      producing NO.2 item at 2019-11-24 19:39:42:138
6      consuming NO.2 item at 2019-11-24 19:39:42:140
7      producing NO.3 item at 2019-11-24 19:39:42:140
8      producing NO.4 item at 2019-11-24 19:39:42:145
9      consuming NO.3 item at 2019-11-24 19:39:42:149
10     producing NO.5 item at 2019-11-24 19:39:42:149
11     producing NO.6 item at 2019-11-24 19:39:42:151
12     producing NO.7 item at 2019-11-24 19:39:42:154
13     consuming NO.4 item at 2019-11-24 19:39:42:155
14     consuming NO.5 item at 2019-11-24 19:39:42:160
15     consuming NO.6 item at 2019-11-24 19:39:42:161
16     producing NO.8 item at 2019-11-24 19:39:42:162
17     producing NO.9 item at 2019-11-24 19:39:42:162
18     consuming NO.7 item at 2019-11-24 19:39:42:169
19     producing NO.10 item at 2019-11-24 19:39:42:172

```

## 其他结论

信号量实现方式中，核心程序结构如下，这样运行结果正常，并未发生竞争条件。

```

1  static class Producer extends Thread{
2      @Override
3      public void run(){
4          int item = 0;
5          while (id < LIMIT) {
6              item = produceItem();
7              empty.acquire();// 1
8              mutex.acquire();// 2
9
10             insertItem(item);

```

```

11
12         mutex.release();// 1
13         full.release();// 2
14     }
15 }
16 // ...
17 }
18
19 static class Consumer extends Thread {
20     @Override
21     public void run() {
22         int item;
23         while (true) {
24             full.acquire();// 1
25             mutex.acquire();// 2
26
27             item = removeItem();
28
29             mutex.release();// 1
30             empty.release();// 2
31
32             if (item == LIMIT - 1) break;
33         }
34     }
35 // ...
36 }

```

当把如上 1、2 代码段互换位置时，运行结果就会存在竞争条件。如下图，写入生产消费记录的第一段后程序便发生阻塞，不再运行，即Producer和Consumer线程发生死锁。此处，当Producer运行生产记录时，该过程随机sleep了0-10毫秒，CPU被Consumer线程占用，Consumer并不是先检测缓冲区是否已经存在item，便先执行down(mutex)进入临界区，接着down(full)时发生阻塞；如此一来Producer线程down(mutex)时便不能访问缓冲区，从而造成死锁。

```

1   producing NO.0 item at 2019-11-24 19:36:04:790
2   |

```

可见Java实现线程并发确实是较为容易和安全的，但对管程的设计、对信号量访问的把控能力同样对编程者提出了要求。