生产者消费者问题

杨朝辉

PB17071433

2019/11/24

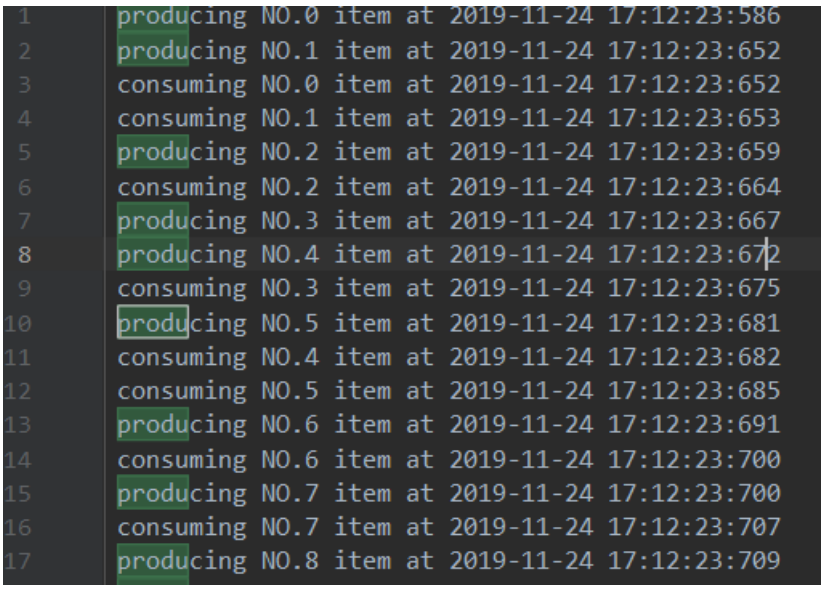
模拟方式

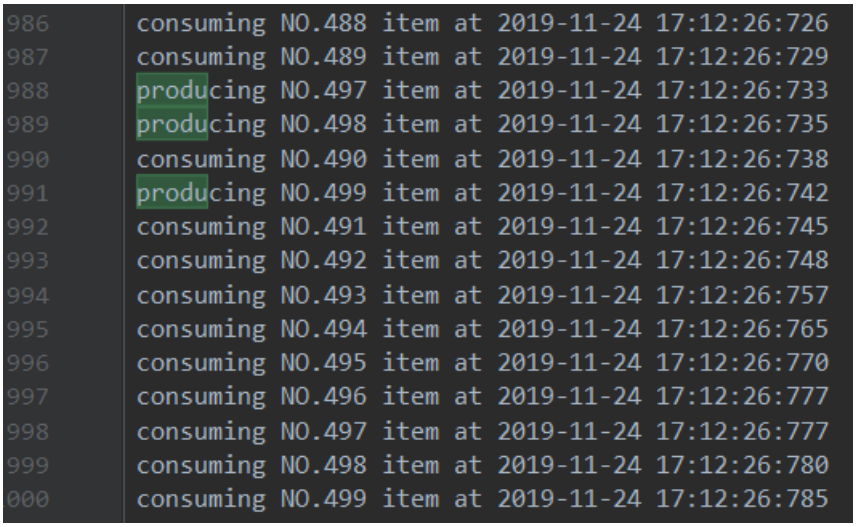
- 利用Java中对“管程”这一系统原语的支持实现了临界区避免竞争条件，成功模拟了该问题。该程序是借助Java中的synchronized关键字实现的。
- 另外，也通过信号量的方式来控制线程间的同步互斥问题，同样成功模拟了生产消费问题。实际中是借助Java内置的处理多线程并发的concurrent包中的Semaphore实现的。

运行结果

管程方式

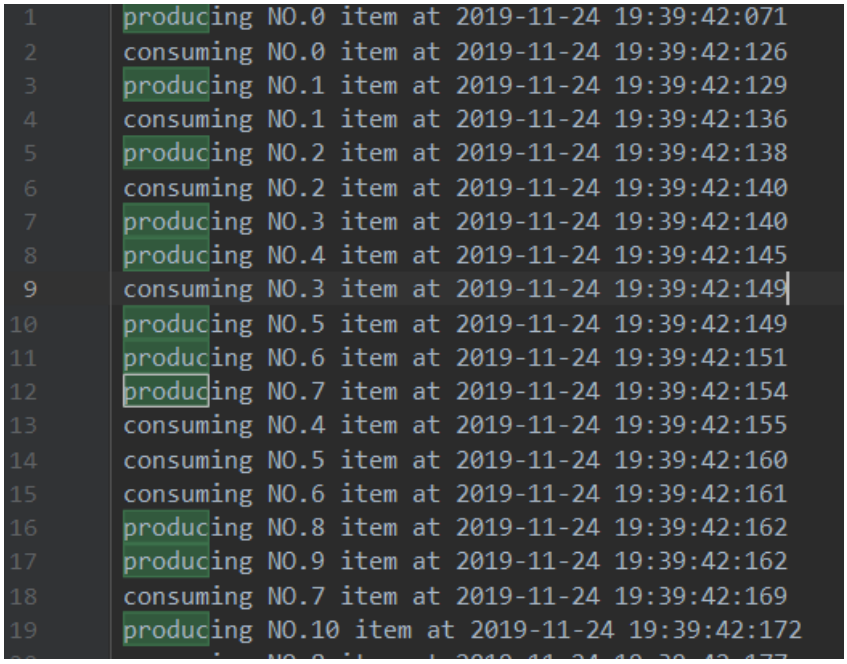
以下为基于管程原语模拟的缓冲区大小50、生产商品数量上限500时该问题的运行结果（生产的生产/消费记录文件），其中按照时间（精确到毫秒）先后记录了生产和消费记录。





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。



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。

```
static class Producer extends Thread{
    @Override
    public void run(){
        int item = 0;
        while (id < LIMIT) {
            item = produceItem();
            empty.acquire();// 1
            mutex.acquire();// 2

            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。