作业 3 报告

韩云飞 SA13226297

1. Please draw a dag for one of your programs that you wrote in your last assignments, and try to explain the concurrency relation among the threads according to the dag.

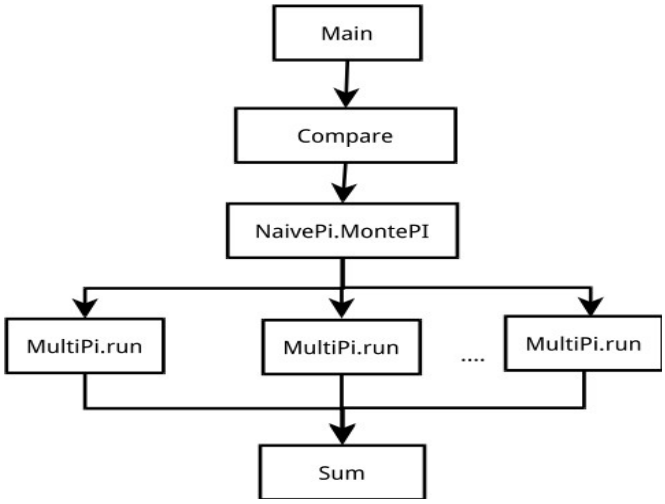
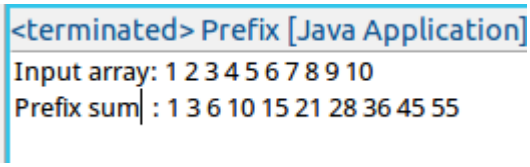


图 1

图 1 中，在每次 MultiPi.start 执行之后，都会启动一个新的 Thread 来执行 MultiPi 程序，所有的 MultiPi 线程会并行地计算各自的 sum, 等所有的线程计算完成之后，才会把所有的 sum 加起来用于最终的计算。

2. **[Optional]** Please try to implement the parallel prefix algorithm you learned from the class.

实现见 PrefixSum.java 程序，运行结果为:



3. Try to derive T(n) = O($\log^2$ n) from T(n) = O(log n) + 1T(n/2)

用代入法求解递归式：
假设 T(n) <= c $\log^2$ n 则 T(n/2) <= c $\log^2$ (n/2) = c (logn-1)*(logn-1) (1)
将(1)代入递归式 T(n) = O(log n) + 1T(n/2)可得：
T(n) <= c0logn + c (logn - 1)*(logn – 1) = c0logn + c$\log^2$ n - 2clogn + c <= c$\log^2$ n
对任意 n，只要 c0 < c 时，必然成立，所以 T(n) = O($\log^2$ n)。

4. Write a program where multiple threads read and write to a sequential data structure (e.g. ArrayList) using java.util.concurrent.locks framework. Write the same program, but now using Synchronized instead of the locks framework. Play with the number of threads (readers and writers) and note differences in the performances.

这里选择了生产者消费者问题进行分析：

为了方便重用以及比较，首先建立一个生产消费资源的抽象类:

```
 abstract class Resource<T> {
   ArrayList<T> buffer = new ArrayList<T>();  //连续资源
   final int COUNT = 10; //资源空间最大容量
   abstract  void put(); //放置一个资源
   abstract  void get(); //取用一个资源
 }
```

针对抽象类直接写出生产者，消费者类:

```
class Producer<T> implements Runnable {
        private Resource<T> resource;
        long round;

        public Producer(Resource<T> resource, long round) {
                this.resource = resource;
                this.round = round;
        }
        @Override
        public void run() {
                for(int i=0; i < round; i++) {
                        try {
                                Thread.sleep((long) (Math.random() * 1000));
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                        resource.put();
                }
        }
}


class Consumer<T> implements Runnable {
        private Resource<T> resource;
        long round;

        public Consumer(Resource<T> resource, long round) {
                this.resource = resource;
                this.round = round;
        }
        @Override
        public void run() {
          for(int i=0; i < round; i++) {
                        try {
                                Thread.sleep((long) (Math.random() * 1000));
                        } catch (InterruptedException e) {
                                e.printStackTrace();
```

```
                }
                resource.get();
            }
        }
}
```

这样做的好处一是不必对 Sync, Lock 重写生产者，消费者，二是方便对 Sync，Lock 进行控制比较，Sync 的 Resource 实现如下:

```
//Sync
class ResourceS<T>  extends Resource{
        public synchronized void put() {
                while (buffer.size() == COUNT) {
                        try {
                                wait();
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
                T  item = (T)new Object();
                buffer.add(item);
                System.out.println("Producer an item, buffer size:  " +
                buffer.size());
                notify();
        }

        public synchronized void get() {
                while (buffer.size() == 0) {
                        try {
                                wait();
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
                buffer.remove(0);
                System.out.println("Consumer an item, buffer size:  " +
                buffer.size());
                notify();
        }
}
```

Lock 的 Resource 实现如下：

```
//Lock
```

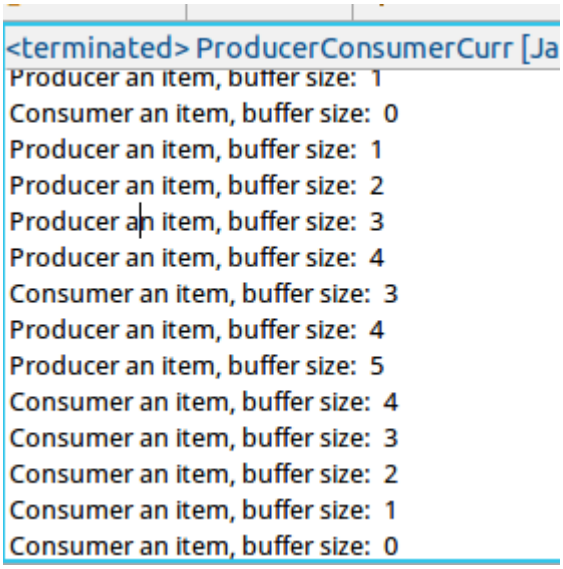
```
class ResourceL<T>  extends Resource{
   private final Lock lock = new ReentrantLock();
   private final Condition full = lock.newCondition();
   private final Condition empty = lock.newCondition();
        public void put() {
                lock.lock();
       try {
          while (buffer.size() == COUNT) {
             System.out.println("It's full, can't produce.");
             full.await();
          }
          T item = (T)new Object();
          if (buffer.add(item)) {
                System.out.println("Produce an item, buffer size:  " +
        buffer.size());
             empty.signal();
          }
       } catch (InterruptedException ie) {
          System.out.println("producer is interrupted!");
       } finally {
          lock.unlock();
       }
       }

        public void get() {
                lock.lock();
       try {
          while (buffer.size() == 0) {
             System.out.println("It's empty, can't consume.");
             empty.await();
          }
          if (buffer.remove(0) == null ) {
                System.out.println("Consume an item, buffer size:  " +
buffer.size());
             full.signal();
          }
       } catch (InterruptedException ie) {
          System.out.println("consumer is interrupted!");
       } finally {
          lock.unlock();
       }
     }
}
```

测试驱动程序为：

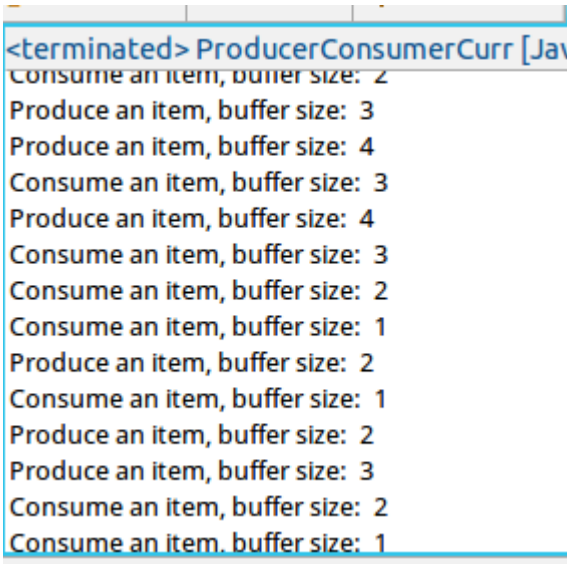
```
public class ProducerConsumerCurr {
```

```
        public static void Test(Resource resource) {
                new Thread(new Producer(resource,100)).start();
                new Thread(new Consumer(resource,100)).start();
                new Thread(new Producer(resource,1000)).start();
                new Thread(new Consumer(resource,1000)).start();
                new Thread(new Producer(resource,1000)).start();
                new Thread(new Consumer(resource,1000)).start();
        }
        public static void main(String[] args) {
                Resource<Object> rs = new ResourceS<Object>();
                Test(rs);
//              Resource<Object> rl = new ResourceL<Object>();
//              Test(rl);
        }
}
```

Sync 方式的结果：



Lock 运行结果：



Sync vs Lock 性能比较

(PS: 由于电脑为单核，而且程序中有许多输出语句和随机 sleep 语句，对两种方式进行时间对比得到的结果很难做到绝对客观。)

查阅资料和 Web，得到如下信息：在资源竞争不是很激烈的情况下，Synchronized 的性能要优于 ReetrantLock，但是在资源竞争很激烈的情况下，Synchronized 的性能会下降几十倍，但是 ReetrantLock 的性能能维持常态。