中北大学
操作系统课程设计
说 明 书

学　院、系：	软件学院
专　　业：	软件工程
学 生 姓 名：	和泽　　学 号：　1414010824
设 计 题 目：	基于 Windows 的线程控制与同步
起 迄 日 期:	2016 年 12 月 21 日- 2017 年 1 月 4 日
指 导 教 师:	李玉蓉

2016 年 12 月 20 日

1.需求分析

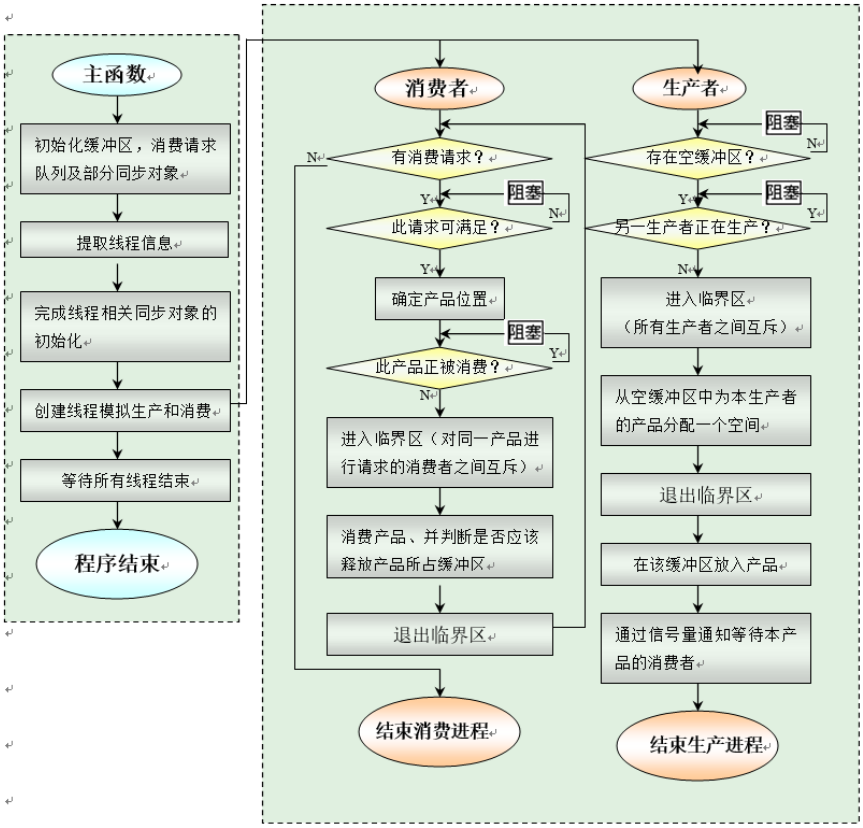
我们的题目是基于 Windows 的线程控制与同步，要求理解和掌握 Windows 中线程控制与同步机制的相关 API 函数的功能，能够利用这些函数进行编程。问题共有三个，实现生产者-消费者问题，实现读/写者问题，实现哲学家就餐问题。针对问题，我们小组两个人进行了不同的分工，我的任务是利用管程机制实现生产者－消费者问题。

问题描述：

生产者消费者问题，也称有限缓冲问题，是一个多线程同步问题的经典案例。该问题描述了两个共享固定大小缓冲区的线程--即所谓的"生产者"和"消费者"--在实际运行时会发生的问题。生产者的主要作用是生成一定量的数据放到缓冲区中，然后重复此过程。与此同时，消费者也在缓冲区消耗这些数据。该问题的关键就是要保证生产者不会在缓冲区满时加入数据，消费者也不会在缓冲区中空时消耗数据。

2.总体设计

2.1 程序流程图



2.2 设计说明

在这次实验中定义的多个缓冲区不是环形循环的，并且不需要按序访问。其中生产者可以把产品放到某一个空缓冲区中，消费者只能消费被指定生产者生产的产品。本实验在测试用例文件中指定了所有生产和消费的需求，并规定当共享缓冲区的数据满足了所有有关它的消费需求后，此共享才可以作为空闲空间允许新的生产者使用。

本实验在为生产者分配缓冲区时各生产者之间必须互斥，此后各个生产者的具体生产活动可以并发。而消费者之间只有在对同一个产品进行消费时才需要互斥，它们在消费过程结束时需要判断该消费者对象是否已经消费完毕并释放缓冲区的空间。

程序由四部分组成：

1)生产者模块：调用缓冲区的生产函数。

2)消费者模块：调用缓冲区的消费函数。

3）缓冲区模块：具体的消费生产过程。

4）主模块：声明生产者消费者对象，并开始执行。

3. 详细设计

算法和数据结构

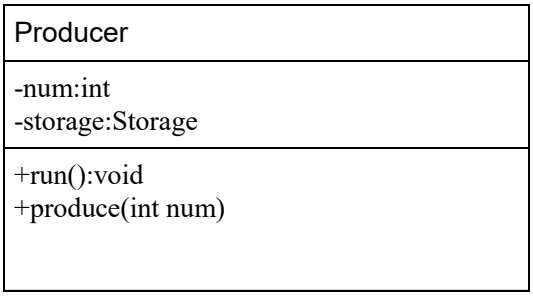


图 2-1　生产者类的 UML 图

程序中定义一个生产者类，包含两个私有对象和一个公有方法和一个私有方法。

num 对象：每次生产的产品数量。

storage 对象：所放位置的仓库 Philosopher(int num)方法：哲学家类构造函数，参数 num 表示哲学家编号

run():void 方法：线程 run 函数

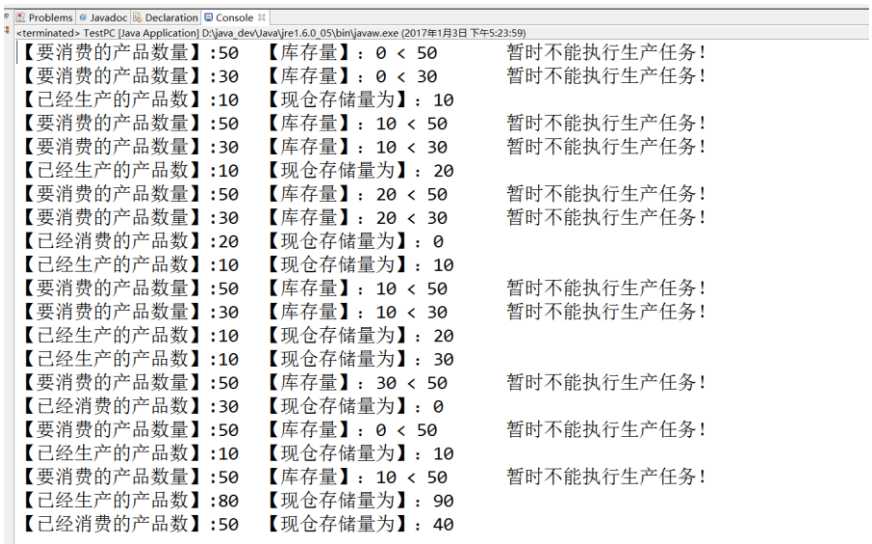
produce(int num)方法：调用仓库的生产函数

Consumer
-num:int -storage:Storage
+run():void +consum()(int num)

仓库

Storage
-MAX_SIZE:int -lists:LinkedList -lock:Lock -full:Condition -empty:Condition
+produce()(in num) +consum()(int num)

运行结果：



4. 心得体会

本次的实验，让我对生产者消费者的问题有了更加深刻的认识，特别是对 P 和 V 操作的了解。还有就是 Storage 临界区，Mutex 量得运用，以及 Semaphore 信号量这个同步对象的熟悉。Semaphore 信号量这个同步对象，在课堂上了解的很少，但通过这次实验，参考资料，以及和两位两个互斥量得对比，有了更加深刻的了解。

附录：

核心代码

```
package nuc.hz.os.pc;
import java.util.LinkedList;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
//仓库类 Storage 实现缓冲区
public class Storage {
    //仓库的最大存储量
  private final int MAX_SIZE=100;
  //仓库存储的载体
  private LinkedList<Object> list=new LinkedList<Object>();
  //锁
  private final Lock lock=new ReentrantLock();
  //仓库满的条件变量
  private final Condition full=lock.newCondition();
  //仓库空的条件变量
  private final Condition empty=lock.newCondition();
  //生产 num 个产品
  public void produce(int num) throws InterruptedException{
      //获得锁
      lock.lock();
          //如果仓库剩余容量不足
          while(list.size() + num > MAX_SIZE){
              int a=MAX_SIZE- list.size();
              System.out.println("【要生产的产品数量】：" + num + "\t【仓库剩余
量】：" + a + " > "+ num + "\t 暂时不能执行生产任务！");
              //由于条件不满足，生产阻塞
              full.await();
          }
          //生产条件满足情况下，生产 num 个产品
          for(int i=1;i<=num;++i){
```

```
                list.add(new Object());
            }
            System.out.println("【已经生产的产品数】:" + num + "\t【现仓存储量为】:
" + list.size());

            //唤醒其他所有线程
            full.signalAll();
            empty.signalAll();

            //释放锁
            lock.unlock();

        }
        //消费 num 个产品
        public void consume(int num) throws InterruptedException{

            //获得锁
            lock.lock();

                //如果仓库存储量不足
                while(list.size() < num){
                    System.out.println("【要消费的产品数量】:" + num + "\t【库存量】:"
+ list.size()+" < " + num +"\t 暂时不能执行生产任务！");
                    //由于条件不满足，生产阻塞
                    empty.await();
                }
                //消费条件满足情况下，消费 num 个产品
                for(int i=1;i<=num;++i){
                    list.remove();

                }
                System.out.println("【已经消费的产品数】:" + num + "\t【现仓存储量为】:
" + list.size());

                //唤醒其他所有进程
                full.signalAll();
                empty.signalAll();
```

```
                //释放锁
                lock.unlock();
    }
public LinkedList<Object> getList() {
        return list;
}
public void setList(LinkedList<Object> list) {
        this.list = list;
}
public int getMAX_SIZE() {
        return MAX_SIZE;
}

}
```