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：	张丹　　学 号：　1414010513
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，我的任务是利用记录型信号量实现读/写者和哲学家就餐问题。

1-1.读者写者问题

问题描述：

读者写者问题，是指保证一个 writer 进程必须与其他进程互斥地访问共享对象的同步问题。读者写者问题可以这样的描述，有一群写者和一群读者，写者在写同一本书，读者也在读这本书，多个读者可以同时读这本书，但是，只能有一个写者在写书，并且，读者必写者优先，也就是说，读者和写者同时提出请求时，读者优先。当读者提出请求时需要有一个互斥操作，另外，需要有一个信号量 S 来当前是否可操作。

1-2.哲学家就餐问题

问题描述：

哲学家的生活就是思考和吃饭，即思考，就餐，再思考，往复循环。要求是：每一个哲学家只有在拿到位于他左右的刀叉后，才能够就餐；哲学家只能先拿一把刀或叉，再去拿另一把刀或叉，而不能同时去抓他旁边的两把餐具，也不能从其他哲学家手中抢。

2. 总体设计

2-1. 读者-写者的读写限制（包括读者优先和写者优先）

将所有的读者和所有的写者分别放进两个等待队列中，当读允许时就让读者队列释放一个或多个读者，当写允许时，释放第一个写者操作。读者写者问题的定义如下：有一个许多进程共享的数据区，这个数据区可以是一个文件或者主存的一块空间；有一些只读取这个数据区的进程（Reader）和一些只往数据区写数据的进程(Writer)。

我们需要分两种情况实现该问题：

读优先：要求指一个读者试图进行读操作时，如果这时正有其他读者在进行操作，他可直接开始读操作，而不需要等待。

写优先：一个读者试图进行读操作时，如果有其他写者在等待进行写操作或正在进行写操作，他要等待该写者完成写操作后才开始读操作。

程序由三部分组成：

1)读者模块：1）读-写互斥，即不能同时有一个读者在读，同时却有一个写者在写；

2）读读允许，即可以有 2 个以上的读者同时读。

2)写者模块：写-写互斥，即不能有两个写者同时进行写操作。

3)主模块：主控模块实现对读者写着模块进行调用。

2-2.**哲学家就餐问题**

设计一个程序，能够显示当前各哲学家的状态和桌上餐具的使用情况，并能无死锁的推算出下一状态各哲学家的状态和桌上餐具的使用情况。即设计一个能安排哲学家正常生活的程序。

为哲学家设计 3 种状态，即“等待”“进餐”“思考”。每个哲学家重复进行“等待”->“进餐”->“思考”的行动循环。其中：

“等待”->“进餐”：只有一个哲学家处于等待进餐状态，且左右手两边的餐具都处于“空闲”状态时，可以发生这种状态改变。此状态改变发生后，哲学家拿起左右手两边的餐具。

“进餐”->“思考”：此状态改变发生后，哲学家放下左右手上的餐具。餐具状态由“使用中”转变为“空闲”。

“思考”->“等待”：哲学家思考结束后，无条件转入等待状态。

程序由四部分组成：

1)筷子模块：调用锁模块的是否锁状态。

2)哲学家模块：设置哲学家的进餐－思考－等待状态，并在不同状态间调用获取或释放左右筷子的方法来转化状态。

3)锁模块：设置锁的状态。

4)主模块：给哲学家分配筷子分调用相应的开始 运行 结束方法。

3．详细设计

3-1 **实现读/写者问题**

数据结构

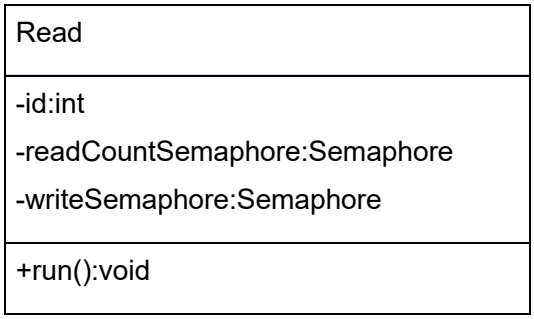


图 3-1-1 读者类的 UML 图

程序中定义一个哲学家类，包含三个私有对象和一个公有方法。

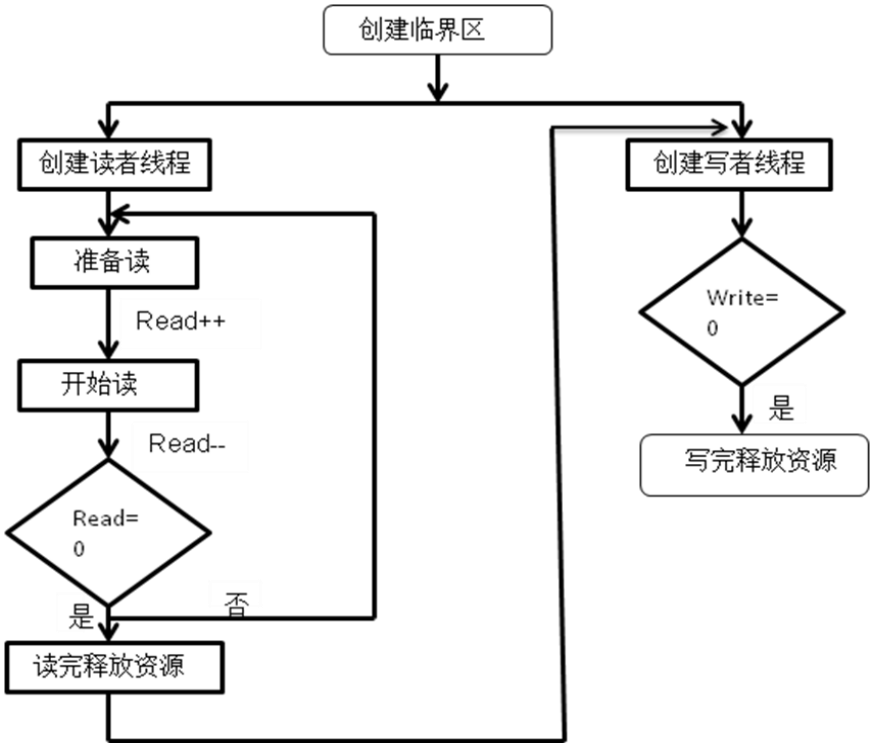
id 对象：读者 id

readCountSemaphore 对象：读者数量信号量

writeSemaphore：写者信号量

run() 方法：读者优先

读者优先：



Writer
-id:int -writeSemaphore:Semaphore
+run():void

图 3-1-2　写者类的 UML 图

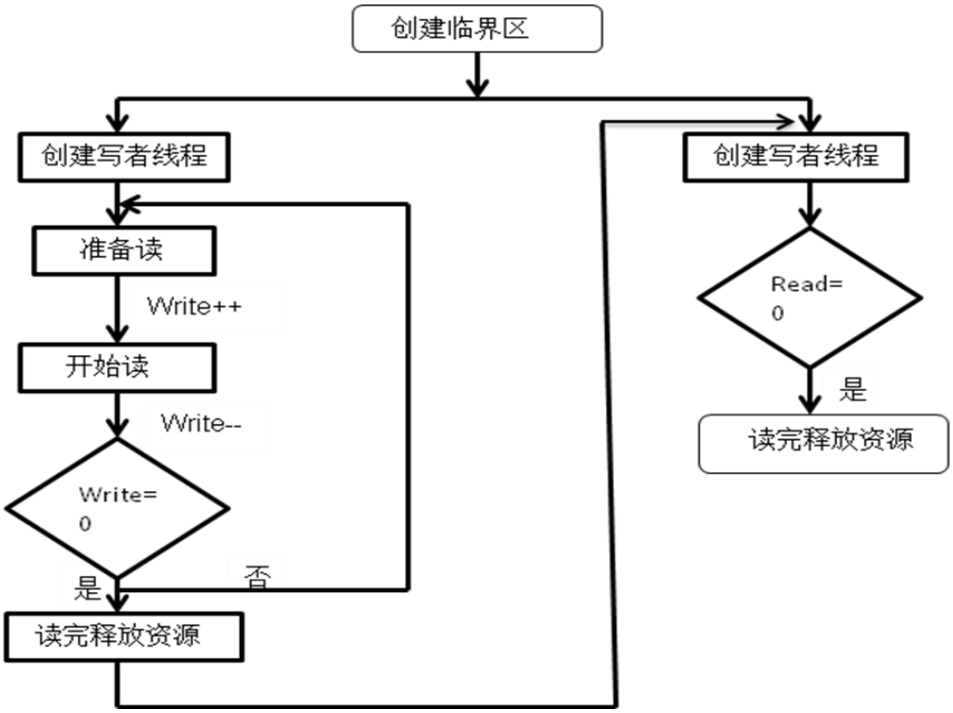
程序中定义一个哲学家类，包含两个私有对象和一个公有方法。

id 对象：读者 id

writeSemaphore：写者信号量

run() 方法：写者优先

写者优先:


创建临界区
创建写者线程
创建写者线程
准备读
Write++
开始读
Write--
Write=
0
是
否
读完释放资源
Read=
0
是
读完释放资源

运行结果


Problems
Javadoc
Declaration
Console
TestReadWrite [Java Application] D:\java_dev\Java\jre1.6.0_05\bin\javaw.exe (2017年1月3日 下午5:22:22)
写者8不可以写。。。
有写者在进行写操作，读者9等待读
写者7不可以写。。。
写者0写完了。。。
读者1我正在读哦。。。
读者1读完了。。。
写者3我正在写哦。。。
写者3写完了。。。
写者2我正在写哦。。。
写者2写完了。。。
写者5我正在写哦。。。
写者5写完了。。。
写者6我正在写哦。。。
写者6写完了。。。
写者8我正在写哦。。。
写者8写完了。。。
写者7我正在写哦。。。
写者7写完了。。。
读者4我正在读哦。。。
读者4读完了。。。
读者9我正在读哦。。。

3-2.**实现哲学家就餐问题**。

数据结构

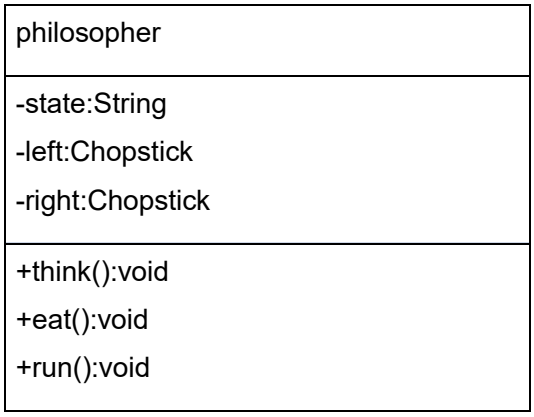


图 3-2-1 哲学家类的 UML 图

程序中定义一个哲学家类，包含三个私有对象和三个公有方法。

state 对象：哲学家的状态

left 对象：左筷子

right 对象：右筷子

think()方法：哲学家思考

eat()方法：哲学家就餐

run()方法:哲学家的活动，状态的改变体现了活动

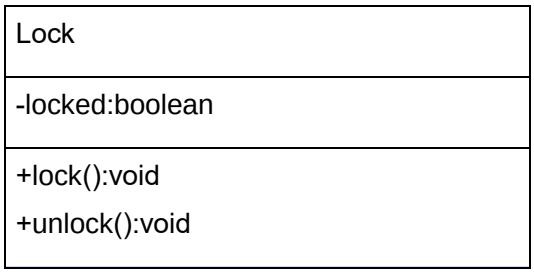


图 3-2-2 锁类的 UML 图

程序中定义一个哲学家类，包含一个私有对象和两个公有方法。

locked 对象：实现进程间同步

lock()方法：筷子正被使用

unlock()方法：筷子可被调用

Chopstick
-id:char -lock: Lock
+pickup():void +putdown():void

图 3-2-3　筷子类的 UML 图

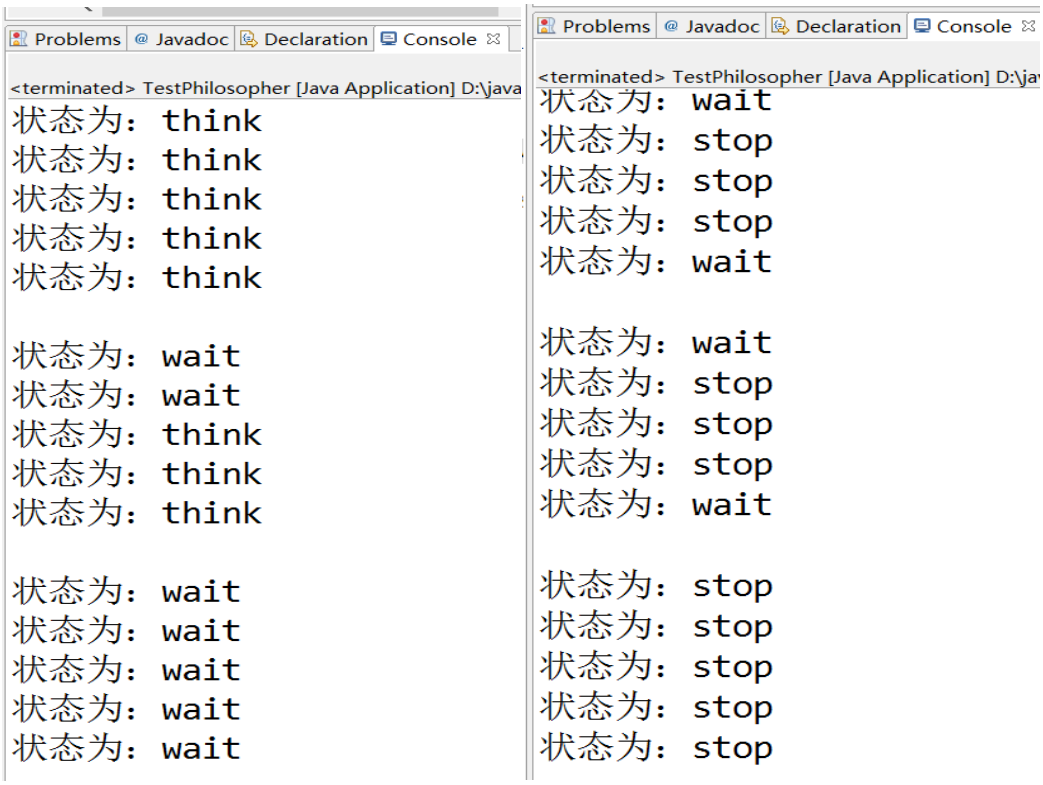
程序中定义一个哲学家类，包含两个私有对象和两个公有方法。

id 对象：每支筷子的标识符

pickup()方法:取筷子后加锁

putdown()方法:释放筷子后解锁

运行结果：



4.心得体会

通过此次课程设计，使我更加扎实地掌握了有关“ 读者-写者问题 ”实现的技巧。“ 读者-写者问题 “的编写，花的时间很多，也学到很多东西。了解支持多道程序的并发操作系统设计中解决资源共享时进程间的同步与互斥的信号量机制。几天的试验，虽然难度有点大，但只要自己花时间去学习，还是会攻克困难的。在设计过程中虽然遇到了一些问题，但经过一次又一次的思考，一遍又一遍的检查，终于找出了原因之所在，也暴露出了前期我在这方面知识欠缺和经验不足。实践出真知，通过亲自查阅资料和动手设计，使我掌握的知识不再是纸上谈兵。过而能改，善莫大焉，在课程设计过程中我不断发现错误不断改正，不断领悟，不断获取。在设计过程中，我不仅巩固了以前所学过的知识，而且在实践的过程中学到了很多在书本上没有学到过的知识，更让我懂得了理论与实际相结合是很重要的，只有这样才能更好地提高自己的实际动手能力和独立思考的能力。

附录：

核心代码一：

```
package nuc.os.readWrite;
import java.util.Random;
import java.util.concurrent.Semaphore;
public class Read extends Thread {
    public int id;//读者 id
    public Semaphore readCountSemaphore;//读者数量信号量
    public Semaphore writeSemaphore;//写者信号量
    public Read(int id, Semaphore semaphore, Semaphore semaphore2) {
        this.id = id;
        this.readCountSemaphore = semaphore;
        this.writeSemaphore = semaphore2;
        this.start();//开始读
    }
    //读者优先
    public void run(){
        //没人写
        if(writeSemaphore.availablePermits()>0){//可以读
            System.out.println("读者"+id+"可以读。。。");
        }
        else{
            System.out.println("有写者在进行写操作，读者"+id+"等待读");
```

```
                }
                readCountSemaphore.acquire();//等待着读
            if(TestReadWrite.readCount==0){//如果第一个读者，那么要考虑是否有写者，没有写者，直接读；有写者，等待写者
                TestReadWrite.readCount++;//已经具备读的条件了 读者数量加 1
                writeSemaphore.acquire();//等同于  wait(wmutex);
            }
            readCountSemaphore.release();  //signal(rmutex);
            readCountSemaphore.acquire();  //此时允许其他读者读数据
            //可以读了
            System.out.println("读者"+id+"我正在读哦。。。");
            Thread.sleep((long)(new Random().nextFloat()*2000));
            System.out.println("读者"+id+"读完了。。。");
            TestReadWrite.readCount--;  //读者完了  读者数量减 1
            if(TestReadWrite.readCount==0){
                writeSemaphore.release();  //没有读者了 可以写了
            }
            readCountSemaphore.release();  //释放读者信号量
        }
    }
```

核心代码二：

1）

```
package nuc.os.Philosopher;
public class Lock {
    private boolean locked; //实现进程间的同步
    public Lock() {
        locked=false;//筷子可用  不处于锁状态
}//同步方法，每个进程取筷子之前都要 locked=false 才行//只能被一个进程使用 修饰符
    public synchronized void lock() throws InterruptedException{
        while(locked)//此时正被用
            wait();//等待 locked=false;
            locked=true;//加锁，防止别的进程取同一支筷子
    }
    public synchronized void unclock(){//解锁
        locked=false;
        notify();//唤醒别的进程
```

```
    }
}
```

2）

```
package nuc.os.Philosopher;
public class philosopher extends Thread {
    //state 代表哲学家的状态  初始化为思考状态
    public String state="think";
    private Chopstick left,right;
    //构造函数   传进当前哲学家用的左右筷子编号
    public philosopher(Chopstick left, Chopstick right) {
        super();
        this.left = left;
        this.right = right;
    }
    protected void think() throws InterruptedException{
        sleep((long)(Math.random()*1.0));
    }
    protected void eat() throws InterruptedException{
        sleep((long)(Math.random()*1.0));
    }
    //run 函数开始一个线程  代表当前哲学家的活动
    public void run(){
        int i;
        for(i=0;i<100;i++){
            //每个哲学家循环 100 次后结束
            state="think";//思考
            think();
            state="wait";    sleep(1);     left.pickup();//取左筷子
            state="wait";     right.pickup();//取右筷子
            state="eat";
            eat();//就餐
            left.putdown();
            right.putdown();
        }
        state="stop";//当前线程结束  当前哲学家停止一起活动
    }
}
```