成绩	

廣東工業大學

操作系统课程设计

学生学院 计算机学院

专业班级 19 级软件工程（1）班

学　　号 3119005028

学生姓名 魏耀辉

指导教师 李　敏

2021 年 06 月 16 日

<table>
<tr><th></th><th>评价标准</th><th>分数比例</th><th>成绩</th></tr>
<tr><td>课程设计报告</td><td>课程设计报告结构包含：
1、设计目的：课程设计要求达到的目的。
2、设计内容：课程设计要求完成的任务。
3、设计步骤：
(1) 需求分析。
(2) 概要设计：说明课程设计的设计思想、用到的所有抽象数据类型的定义、主程序的流程以及各程序模块之间的层次(调用)关系。
(3) 详细设计：实现概要设计中定义的功能；画出函数的调用关系图；画出各程序流程图。
(4) 调试分析：调试中遇到的问题和问题的解决方式。
(5) 系统测试：列出所有系统功能的测试结果，测试数据应该完整、严格，应该至少包含需求分析中所列。
(6) 使用说明：如何使用系统，要求详细列出每一步的操作步骤。
4、经验与体会：对系统设计与实现的回顾、讨论和分析总结等。
5、重要数据结构或源程序中的疑难部分说明，必须有详细注释。
注意：这里不要附源代码，源代码要求在光盘中提交。
请按上述要点进行课程设计报告，缺一个要点扣一定的分数。
评价标准如下：
优秀：课程设计报告内容完整、结论正确。
良好：课程设计报告内容完整、结论正确。
中等：课程设计报告内容完整、结论基本正确。
及格：课程设计报告内容基本完整。
不及格：（有下列情况之一者，课程设计成绩评为不及格）
1、 不能完成最基本的课程设计操作；
2、 课程设计报告马虎、内容不全、无数据处理过程或数据处理过程不完整、课程设计无结论等；
3、 课程设计报告有抄袭现象；
4、 严重违反课程设计规章制度并造成不良后果。
5、 未做课程设计，不能补做。</td><td>40%</td><td></td></tr>
<tr><td>课程设计检查</td><td>优秀：1、较好地完成了课程设计的任务，有加分点；
2、答辩时，熟练答辩内容，能正确回答老师提出的问题；
良好：1、完成了课程设计的任务，没有加分点；
2、答辩时，熟练答辩内容，能正确回答老师提出的问题；
中等：1、能完成课程设计的任务，没有加分点；
2、答辩时，能回答老师提出的问题，有些问题不够清楚；
及格：1、基本完成课程设计的任务，完成地较为简单；
2、答辩不熟练，基本能说清楚设计的过程和方法；
不及格：（有下列情况之一者，课程设计成绩评为不及格）
1、 不能完成最基本的课程设计操作；
2、 课程设计报告马虎、内容不全、无数据处理过程或数据处理过程不完整、课程设计无结论等；
3、 课程设计报告有抄袭现象；
4、 严重违反课程设计规章制度并造成不良后果。
5、 未做课程设计，不能补做。</td><td>60%</td><td></td></tr>
<tr><td>总评成绩</td><td>☐ 优</td><td>☐ 良</td><td>☐ 中</td><td>☐ 及格</td><td>☐ 不及格</td></tr>
</table>

广东工业大学课程设计任务书

学生姓名	**魏耀辉**	**专业班级**	**19 级软件工程 1 班**	**学号**	**3119005028**
题　目	仿真模拟操作系统中的“读者-写者（写者优先）”				
指导教师	**李敏**		**题目编号**		**24**
主要内容	多进程模拟“读者-写者（写者优先）”问题				
基本任务要求	“读者-写者（写者优先）”问题： 1、读读同时； 2、读写互斥； 3、写写互斥； 4、写者优先与其后面到达的读者。				
参考文献	[1] 计算机操作系统， 汤小丹等 ，西安电子科技大学出版社 [2] 操作系统实验指导书，傅秀芬，广东工业大学（自编） [3] 计算机操作系统教程 (第二版)， 张尧学、 史美林，清华大学出版社 [4] 现代操作系统，A.S.Tanenbaum 著，陈向群等译机械工业出版社				
审查意见	**指导教师签字：** **系主任签字：**　　**年　　月　　日**				

目录

一、设计目的

读者与写者问题作为经典的多线程并发问题，在操作系统中占着十分重要的地位。读者与写者问题的实现在计算机领域十分普遍，数据库的读写分离也是为了解决读者与写者问题加锁带来的并发问题。可见，通过解决读者与写者问题能为实际的软件开发中提供帮助，同时加深对于信号量机制的 P 操作和 V 操作以及对应的读者与写者问题算法。

二、设计内容

（一）要求完成的任务：

- 读读同时，多个读者进程可同步进行
- 读写互斥，读者进程与写者进程同时只能有一个进行
- 写写互斥，多个写者进程同时只能有一个写者进程进行
- 写者优先，在阻塞队列中有写者时写者进程优先

（二）额外完成的任务：

- 自定义进程数量、进程到达时间、进程运行时间
- 新增加选择菜单选项，选择对应算法
- 在写者优先的基础上，增加了读者优先和公平竞争的算法

三、设计步骤

（一）需求分析

本次课程设计是对读者与写者问题的详细实现。因为读者与写者问题是操作系统中经典的多线程问题，所以在编程语言上先选择了多线程编程较为友好的 Java 语言进行开发设计。为了照顾程序使用者的需求及习惯，程序应该设置菜单，让使用者可以进行自主的选择需要的算法进行读者与写者问题的模拟。在此基础上，还应该让使用者自主的进行对线程总数量，线程类型，线程到达时间，线程运行时间的输入而不是采用随机数的模式。本次课程设计的基本要求是完成（1）读者进程与读者进程不互斥，即在一个读者进程运行期间内有其他读者到来时可以直接阅读，不用阻塞在等待队列。（2）读者进程与写者进程互斥，即在读者进程运行期间内有写者到来，该写者必

须等待所有正在阅读的读者进程结束才能进行写操作。反之在写者进程运行期间内有读者到来，读者也必须等待写者进程结束后才能进行读操作。（3）写者进程与写者进程互斥，即在同一时间里只能有一个写者进程在运行（4）算法设计分为：1、读者优先 2、写者优先 3、公平竞争

（二）概要分析

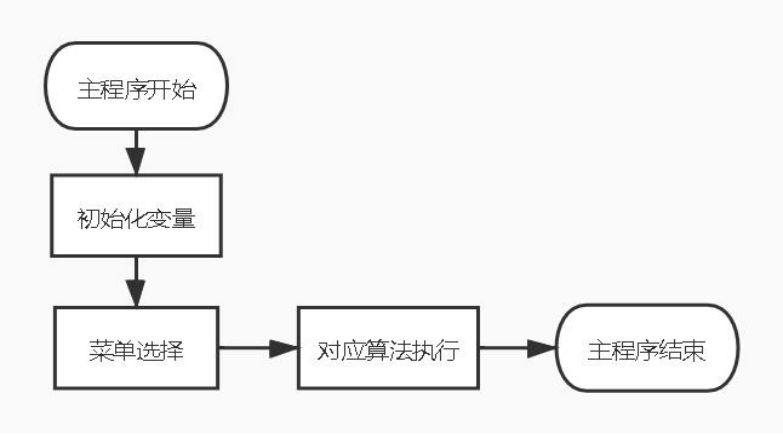
1. 设计思想

读者与写者问题为多线程，创造多个读者类与写者类实现 Java 中的 Thread 类中 Runnable 接口来创建多线程。通过对类构造器的重载将信号量参数和用户类传入对应的读者类或写者类。重写 Run 方法来对线程进行操作。读者与写者问题还设计到了信号量机制和对应的 P 操作（申请）和 V 操作（释放），所以创建一个 Semaphore 类来对信号量进行定义，并通过 P 方法和 V 方法来实现 P/V 操作。

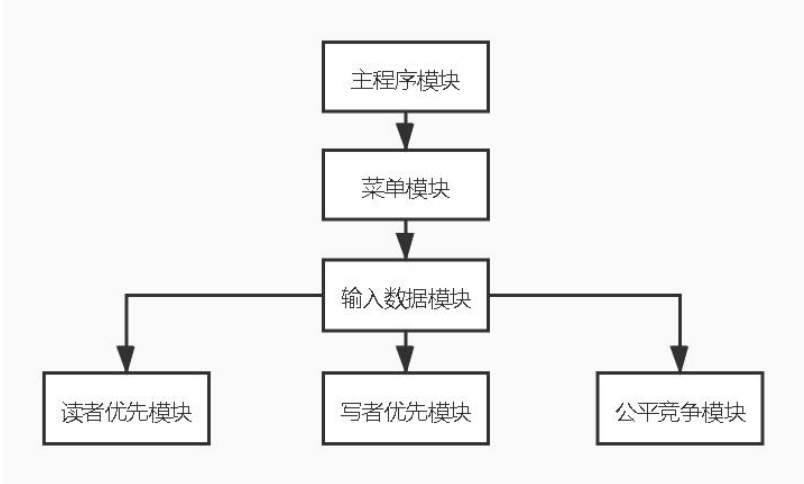
2. 抽象数据类型定义

本次课程设计中未涉及到抽象数据类型的定义

3. 主程序流程



4. 各程序模块之间调用关系



（三）详细设计

1. 实现概要设计的功能

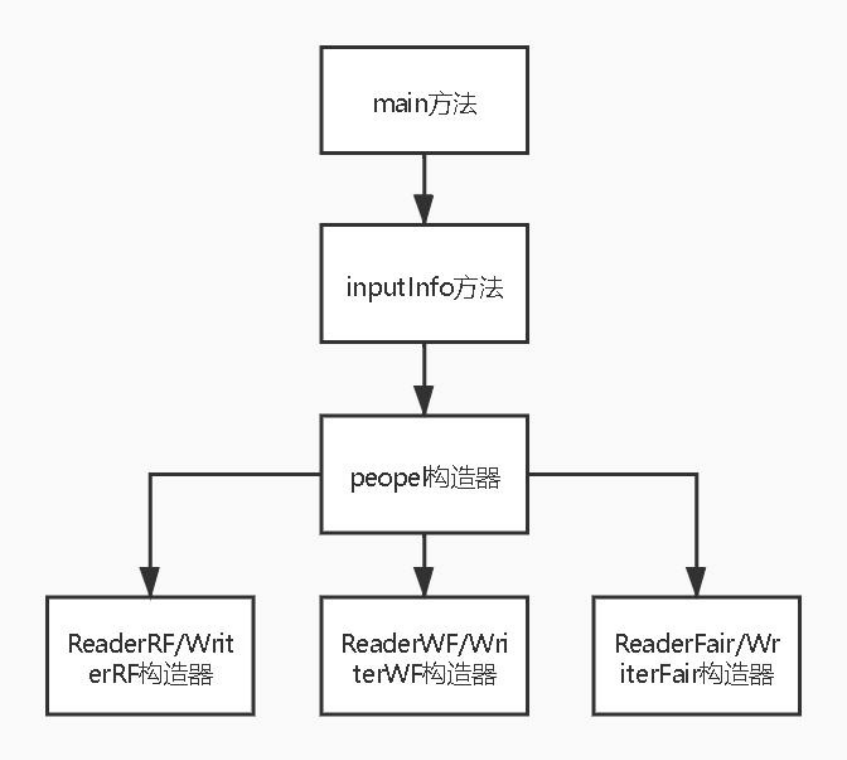
```
package ReaderAndWriter;

import java.util.Scanner;

public class ReaderAndWriter {...}
class People{...}
class ReaderWF implements Runnable{...}
class ReaderRF implements Runnable{...}
class ReaderFair implements Runnable{...}
class WriterWF implements Runnable{...}
class WriterRF implements Runnable{...}
class WriterFair implements Runnable{...}
```

ReaderAndWriter 类实现主程序功能，包含 main 方法和程序输入 inputInfo 方法。People 类定义了一个读者或写者进程的编号、类型、到达时间和运行时间。ReaderRF 类和 WriterRF 类实现读者优先算法的进程调度。ReaderWF 类和 WriterWF 类实现读者优先算法的进程调度。ReaderFair 类和 WriterFair 类实现读者优先算法的进程调度。

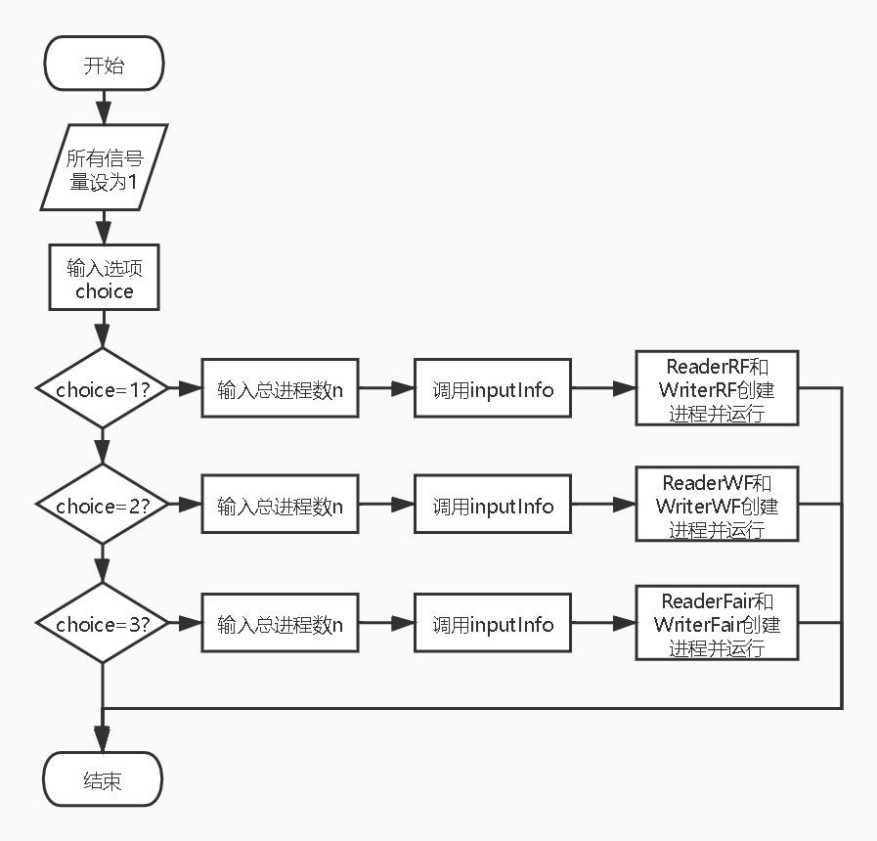
```java
package ReaderAndWriter;

public class Semaphore {//信号量Semaphore的PV操作类
    int count=0;//计数器初始化为0
    public Semaphore(int count) { this.count=count; }
    public synchronized void p(){...}
    public synchronized void v(){...}
}
```

Semaphore 类实现信号量的定义和 P 操作、V 操作的具体实现

2. 函数（方法）调用图



3. 各程序流程图



（四）调试分析

1. 调试中遇到的问题

①一开始打算用比较熟悉的 C 语言进行编写，但后来发现 C 语言针对多线程的部分比较难以实现，并不能达到一个很好的效果。②在使用 Java 后对多线程的使用也不太熟悉，在线程创建的时候不能很好的使用 Java 自带的方法。③信号量的使用每次都要使用 Java 中对线程的 wait（）方法和 notify（）方法很麻烦。

2. 问题的解决方式

①转用对多线程编程更为友好的 Java 语言，只需要编写相应的类来实现线程的 Runnable 接口就可以实现了②上网查阅了线程 Thread 类中的 sleep 方法来控制开始和运行时间③编写 Semaphore 的信号量类并写相应的 P 操作方法和 V 操作方法，每次使用时只需要使用 Semaphore.P()；或者 Semaphore.V()；就可以了

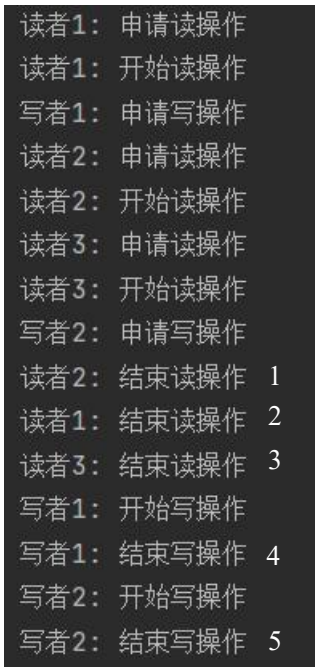
（五）系统测试

1. 测试用例①（读者与写者均匀分布）

线程类型	编号	到达时间	运行时间
R	1	3	5
W	1	4	5
R	2	5	2
R	3	6	5
W	2	7	4

结果分析：①读者优先算法中，R1 的结束时间是第 8 秒，此时等待的有 W1、R2、R3、W2，根据读者优先和读读不互斥原则，R1 运行期间 R2 和 R3 在到达后也会运行，最终结束的顺序应该为：R2->R1->R3->W1->W2②写者优先算法中，因为 W1 在第 4 秒时到达，阻塞后到的 R2、R3 进程，W1 在第 13 秒结束，此时 W2 已经到达所以 W2 先开始写操作继续 R2 和 R3，W2 结束后 R2 和 R3 才开始读操作，最终结束的顺序应该为 R1->W1->W2->R2->R3③公平竞争，读者与写者优先级一致，最终结束的顺序应该为：R1->W1->R2->R3->W2

读者优先：



写者优先：

公平竞争：

2. 测试用例②（读者多）

线程类型	编号	到达时间	运行时间
R	1	2	5
W	1	3	5
R	2	4	4
R	3	5	5
R	4	6	3

结果分析：①读者优先算法中，最终结束的顺序应该为：R1->R2->R4->R3->W1，可以看出来在读者优先算法中，如果一直有读者到来，一直进行读者的优先，造成写者饥饿②写者优先算法中，最终结束的顺序应该为R1->W1->R4->R2->R3③公平竞争，读者与写者优先等级相同，最终结束的顺序应该为：R1->W1->R4->R2->R3

读者优先：

写者优先：

公平竞争：

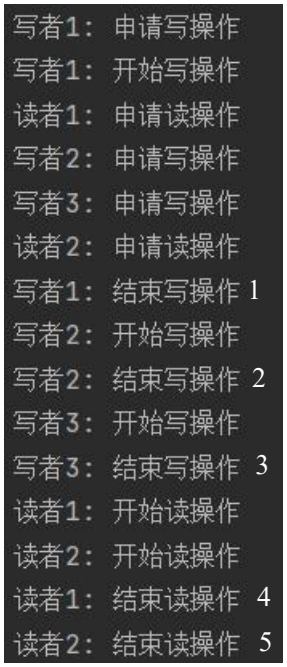
3. 测试用例③（写者多）

线程类型	编号	到达时间	运行时间
W	1	1	4
R	1	2	2
W	2	3	4
W	3	4	5
R	2	5	3

结果分析：①读者优先算法中，最终结束的顺序应该为：W1->R1->R2->W2->W3，可以看出来在读者优先算法中，如果一直有读者到来，一直进行读者的优先，造成写者饥饿②写者优先算法中，最终结束的顺序应该为W1->W2->W3->R1->R2③公平竞争，读者与写者优先等级相同，最终结束的顺序应该为：W1->R1->R2->W2->W3

读者优先：

写者优先：



公平竞争：

（六）使用说明

```
***欢迎进入读者与写者模拟程序***
|   作者：19级软工(1)班魏耀辉   |
|          1.读者优先           |
|          2.写者优先           |
|          3.公平竞争           |
*****************************
请输入选项：3
请输入读者与写者总人数：5
请输入用户类型(R/W)：W
请输入读者或写者的编号：1
请输入读者或写者到达时间(从0时刻开始,单位为秒)：1
请输入读者或写者操作时间(从0时刻开始,单位为秒)：4
请输入用户类型(R/W)：R
请输入读者或写者的编号：1
请输入读者或写者到达时间(从0时刻开始,单位为秒)：2
请输入读者或写者操作时间(从0时刻开始,单位为秒)：2
请输入用户类型(R/W)：W
请输入读者或写者的编号：2
请输入读者或写者到达时间(从0时刻开始,单位为秒)：3
请输入读者或写者操作时间(从0时刻开始,单位为秒)：4
请输入用户类型(R/W)：W
请输入读者或写者的编号：3
请输入读者或写者到达时间(从0时刻开始,单位为秒)：4
请输入读者或写者操作时间(从0时刻开始,单位为秒)：5
请输入用户类型(R/W)：R
请输入读者或写者的编号：2
请输入读者或写者到达时间(从0时刻开始,单位为秒)：5
请输入读者或写者操作时间(从0时刻开始,单位为秒)：3
```

①根据提示输入对应算法的选项②输入读者与写者总人数③依次输入每个用户的类型、编号、到达时间、操作（运行）时间，直到所有用户输入完毕

四、经验与体会

本次的课程设计来说总体难度不大，在设计的前期由于第一次接触多线程编程而需要学习的成本导致前期的进度比较缓慢。回顾本次的课程设计，关键点在于Semaphore类信号量的P操作方法和V操作方法的实现,P操作使用的是wait();的原子语句，使其他线程被阻塞，达到申请到临界区资源的目的。而V操作使用的是notify();原子语句来唤醒被阻塞的线程，达到释放临界区资源的目的。之后对读者优先、写者优先、公平竞争的算法编写就是直接使用P操作和V操作。

本次课程设计加深了我对读者与写者问题的思考与学习，对其中的读写互斥、读读互容、写写互斥等加深了理解，并了解了读者与写者问题在现实中的实现意义。另外，通过本次课程设计，学习到了Java的一些多线程的使用方法。

五、重要数据结构及疑难说明

信号量Semaphore类：

```
public class Semaphore {//信号量Semaphore的PV操作类
    int count=0;//计数器初始化为0
    public Semaphore(int count){//构造器为带count数据的
        this.count=count;
    }
    public synchronized void p(){//编写信号量类下的p操作方法
        count--;//计数器-1
        if(count<0){//等于0则表示又一个进程进入了临界区
            try{
                this.wait();//进程进入阻塞队列，等待唤起
            }
            catch (InterruptedException e){
                e.printStackTrace();
            }
        }
```

```
    }
    public synchronized void v(){//编写信号量类下的 v 操作方法
        count++;//计数器+1
        if(count<=0){//如果有进程阻塞，则唤起该进程
            this.notify();
        }
    }
}
```

用户 People 类：

```
class People{//定义人的类，包含编号、类型、到达时间和运行时间
    double arriveTime;//到达时间
    double operateTime;//运行时间
    int number;//编号
    String type;//类型
    public  People(int  number,String  type,double  arriveTime,double operateTime){//有参构造器
        this.number=number;
        this.type=type;
        this.arriveTime=arriveTime;
        this.operateTime=operateTime;
    }
}
```

以写者优先下的读者类和写者类进行说明

写者优先下的读者类：

```
class ReaderWF implements Runnable{//通过实现 Runnable 接口来模拟写者优先的读者线程
    private Semaphore read,readCountSignal,fileSrc;
    static int readCount=0;//初始化静态的读者数量
    People people;//People 类的 people 对象
    public ReaderWF(Semaphore read,Semaphore readCountSignal,Semaphore fileSrc,People people){//写者优先模式下，读者线程的构造器
        this.read=read;
        this.readCountSignal=readCountSignal;
        this.fileSrc=fileSrc;
        this.people=people;
    }
    @Override
    public void run() {//重写 run 方法
        try{
            Thread.sleep((int)(1000* people.arriveTime));//线程睡眠，模拟到达时间
        }
        catch (InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("读者"+people.number+": 申请读操作");
        read.p();//read 信号量的 p 操作（没有写者阻塞）
        readCountSignal.p();//readCountSignal 信号量的 p 操作
        if(readCount==0){
            fileSrc.p();//fileSrc 信号量的 p 操作
        }
```

```
            readCount++;//读者数量+1
            readCountSignal.v();//readCountSignal 信号量的 v 操作
            read.v();//read 信号量的 v 操作
            //临界区代码块
            System.out.println("读者"+people.number+": 开始读操作");
            try{
                Thread.sleep((int)(1000*people.operateTime)); //线程睡眠,
模拟运行时间}
            catch (InterruptedException e){
                e.printStackTrace();
            }
            System.out.println("读者"+people.number+": 结束读操作");
            readCountSignal.p();//readCountSignal 信号量的 p 操作
            readCount--;//读者读完离开，读者数量-1
            if(readCount==0){
                fileSrc.v();//fileSrc 信号量的 v 操作
            }
            readCountSignal.v();//readCountSignal 信号量的 v 操作
        }
    }
```

写者优先下的写者类：

```
class WriterWF implements Runnable{//通过实现 Runnable 接口来模拟写者优先的写者线程
    Semaphore writeCountSignal,fileSrc,read;
    static int writeCount=0;//初始化静态写者数量为 0
    People people;//People 类的 people 对象
    Public WriterWF(Semaphore read,Semaphore writeCountSignal,Semaphore fileSrc,People people){//写者优先模式下，写者线程的构造器
        this.read=read;
        this.writeCountSignal=writeCountSignal;
        this.fileSrc=fileSrc;
        this.people=people;
    }
    @Override
    public void run() {//重写 run 方法
        try{
            Thread.sleep((int)(1000* people.arriveTime));//线程睡眠，模拟到达时间
        }
        catch (InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("写者"+people.number+": 申请写操作");
        writeCountSignal.p();//信号量 writeCountSignal 的 p 操作
        if(writeCount==0){
            read.p();//信号量 read 的 p 操作
        }
        writeCount++;//写者数量+1
```

```
            writeCountSignal.v();//信号量 writeCountSignal 的 v 操作
            fileSrc.p();//信号量 fileSrc 的 p 操作
            System.out.println("写者"+people.number+": 开始写操作");
            try{
                Thread.sleep((int)(1000* people.operateTime));//线程睡眠,
模拟运行时间
            }
            catch (InterruptedException e){
                e.printStackTrace();
            }
            System.out.println("写者"+people.number+": 结束写操作");
            fileSrc.v();//信号量 fileSrc 的 v 操作
            writeCountSignal.p();//信号量 writeCountSignal 的 p 操作
            writeCount--;//写者数量-1
            if(writeCount==0){
                read.v();//信号量 read 的 v 操作
            }
            writeCountSignal.v();//信号量 writeCountSignal 的 v 操作
        }
    }
```