1. **需求分析**

## （一）、实验相关概念

在某些进程并发运行的过程中，可能会频繁的访问共享资源，那么不同的读写执行顺序一般来说可能会导致不同的结果，换句话说，并发通常会导致数据的同步问题。

其中最著名的问题之一就是读者-写者问题，该问题的定义如下：存在一个多线程共享的数据区（临界区），有些进程（读进程）只读取这个数据区中的数据，有些进程（写进程）只往数据区中写数据，当两个或以上的读进程同时访问共享数据时不会产生副作用，但若某个写进程和其他进程（读进程或写进程）同时访问共享数据时则可能导致数据不一致的错误（数据同步问题）。因此要求：

1. 任意数量的读进程可以同时读这个文件。
2. 一次只有一个写进程可以写文件。
3. 若一个写进程正在写文件，则禁止任何读进程读文件

也就是说，读进程不需要排斥其他读进程，而写进程需要排斥其他所有进程，包括读进程和写进程。

## （二）、实验任务概括

在读者-写者问题中，有多种解决方案：读者优先、写者优先和公平竞争。

读者优先策略中，读进程具有优先权，也就是说，当至少有一个读进程在读时，随后的读进程就无需等待，可以直接进入；在这过程中，写进程会被阻塞，直到所有的读进程读完（写进程可能会饥饿）。

本次实验主要针对读者优先这个解决方案展开，利用多线程模拟读者与写者对临界区的互斥访问。

## （三）、测试数据要求

本实验的测试数据保存在文件中，测试数据文件包括 n 行测试数据，分别描述创建的n个线程是读者还是写者，以及读写操作的开始时间和持续时间。每行测试数据包括四个字段，各个字段间用空格分隔。第一字段为一个正整数，表示线程序号。第二字段表示相应线程角色，R 表示读者，W 表示写者。第三字段为一个正数，表示读写操作的开始时间：线程创建后，延迟相应时间(单位为秒)后发出对共享资源的读写申请。第四字段为一个正数，表示读写操作的持续时间。当线程读写申请成功后，开始对共享资源的读写操作，该操作持续相应时间后结束，并释放共享资源。测试数据随机生成，为简化起见，假设每个线程只执行一次读或写操作，之后运行结束。

下面是一个测试数据文件的例子：

线程序号 角色 何时开始读写 读写持续时间

1 R 3 5

2 R 4 5

3 W 5 2

4 R 10 3

5 W 11 3

6 R 13 4

…… …… …… ……

1. **概要设计**

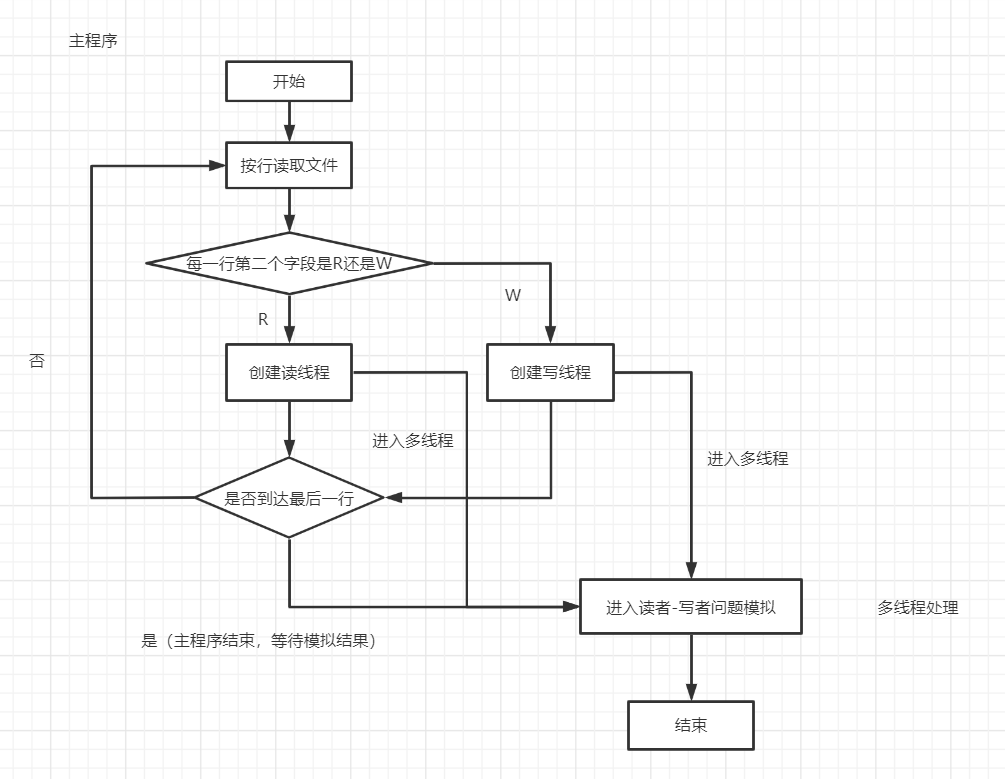
## （一）、抽象数据类型

在并发的问题中，关键的是要解决互斥问题。在互斥的解决方案中，通常采用信号量、管程等工具。在本次实验中，我采用的是信号量这个不要求忙等的同步互斥工具。由于要实现互斥功能，所以应该将相应的信号量初始化为1。

由一开始的需求分析可知，在读者-写者问题的临界区使用中，读者与读者之间不存在互斥问题，写者与写者之间存在互斥问题，写者与读者之间也存在互斥问题。所以需要设计一个信号量wr来实现一个写者与其他读者或写者之间的互斥。

到这里就完成了基于信号量的读者-写者问题的抽象数据类型定义，这个抽象类型在读者优先策略和写者优先策略都能用到。

## （二）、程序实现流程



1. **详细设计**

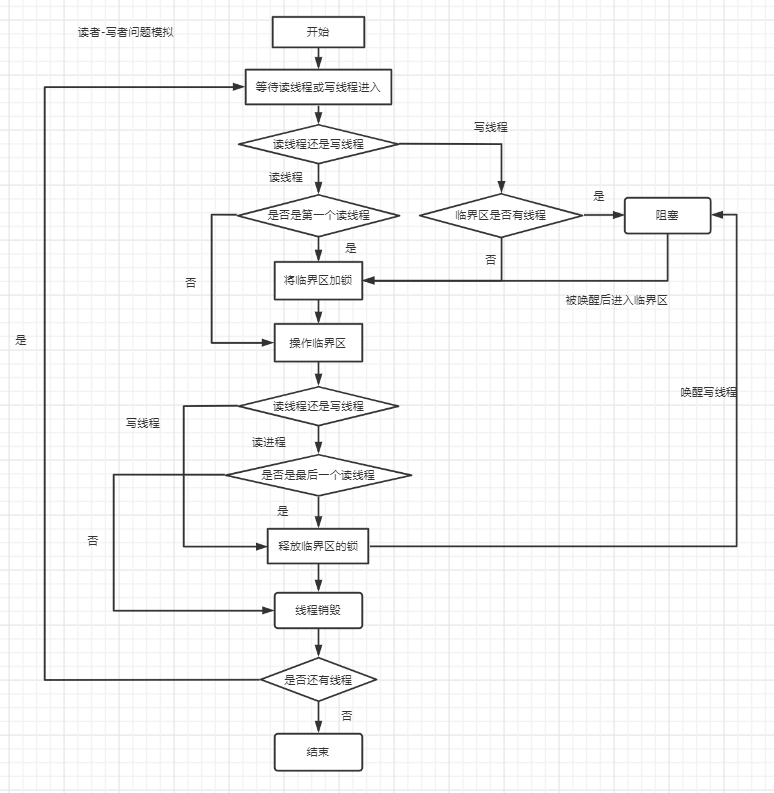
## （一）、读者优先策略实现

在本次实验中，由于采用的是读者优先策略，所以要针对这个策略来进行进一步详细的设计。

在读者优先策略中，当至少有一个读进程在读时，随后的读进程就无需等待的要求，因此需要设计一个记录读者数量的变量readCount。当有读者进入临界区时，readCount就自动加1；当有读者离开临界区时readCount就自动减1。所以当readCount为1时，说明第一个读者进入临界区，此时可以对临界区进行加锁（即将信号量wr递减），这就使得后来的写者无法进入临界区，而随后的读进程无需进行临界区加锁，可以直接进入临界区；只有当所有的读者离开后，即readCount为0时，才释放临界区的锁（即将信号量wr递增），这时被阻塞的写者才能进入临界区，并进行加锁操作。

虽然在临界区中，读者与读者并不存在互斥问题，但是在多个读者修改readCount变量时，可能导致数据同步问题。所以还应该设计一个信号量x来确保readCount变量互斥访问。

流程图如下（可能难以理解，其中是多线程操作）：



实现上述操作的伪代码如下（用semWait和semSignal模拟加锁、解锁操作）：

1. Reader(){
2. semWait x
3. readcount++
4. **if** readcount == 1
5. semWait wr
6. semSignal x
7. read() //读操作
8. semWait x
9. readcount--
10. **if** readcount == 0
11. semSignal wr
12. semSignal x
13. }
15. Writer(){
16. semWait wr
17. write() //写操作
18. semSignal wr
19. }

在这次实验中，我采用了java语言来进行代码的实现。由于之前学习过java的并发编程，我知道在java中有一个现成的信号量类Semaphore（Semaphore是synchronized的加强版，作用是控制线程的并发数量。——摘自百度）。它将基本的信号量操作都进行了封装，使用起来也很方便。由于实验的重点是实现读者-写者问题，并没有要求要自行定义和封装信号量这个类，所以我打算直接引用这个类，方便后面代码的实现。

因此，所有的全局变量定义如下（java代码）：

1. **private** **final** Semaphore x = **new** Semaphore(1);
2. **private** **final** Semaphore wr = **new** Semaphore(1);
3. **private** **int** readCount = 0;

## （二）、其他函数、方法实现（太过基础，代码略）

在读者类、写者类的实现中，应该实现Runnable接口，并重写run方法（在run方法中具体实现读者或写者的读或写操作），这样就能实现用多线程来模拟读者、写者过程。

由于本次实验要读取测试数据文件，并根据每一行的数据生成响应的线程，其中每个线程要含有该线程的序号、对临界区的申请操作时间、对临界区的执行时间。所以应该将这三个变量绑定到每个线程上（作为类的成员变量），并在线程中利用Thread.sleep这个函数来模拟读、写操作和申请操作的时间延迟。

另外，实验还要求在每个线程创建、发出读写操作申请、开始读写操作和结束读写操作时分别显示一行提示信息，这个也能很容易的实现。

1. **调试分析**

## （一）、调试过程

在测试几组数据之后，我发现了一个和我想象中结果不一致的问题。就是当一个写者在写的时候，后面有写者和读者在等待。我一直以为，当写者结束操作释放信号量wr后，下一个访问临界区一定是读者（毕竟叫读者优先策略），所以一直在检查前面分析的步骤，并且查阅资料（可惜百度并没有这方面的资料）。

最后还是请教了老师，然后才知道下一个访问临界区的对象，是要通过抢占信号量wr来决定谁是下一个访问临界区。

测试文件数据如下：

1 W 2 4

2 R 3 5

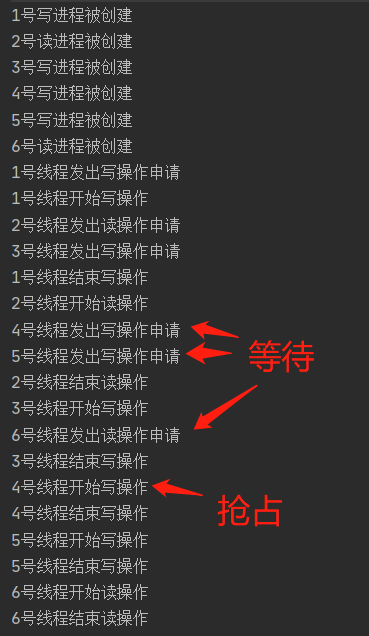
3 W 5 4

4 W 8 5

5 W 10 3

6 R 12 3

测试结果如下：



## （二）、设计过程的经验和体会

由于先前学习过java并发编程，对多线程的同步互斥比较熟悉，因此这次实验对我来说并没有想象中那么困难。

这个程序我认为唯一有缺陷的地方就是：所有线程并不是同时创建的（本该同时申请操作的线程可能会有一个先后顺序），而是在对文件逐行扫描时创建，因此对实验中这个要求可能无法完美实现（线程创建后，延迟相应时间(单位为秒)后发出对共享资源的读写申请）。每个线程的创建之间可能会有几微秒的延迟，在文件行数少的情况下完全可以忽略。但如果文件行数是百万级别的，可能这个延迟就会被放大，第一个线程和最后一个线程创建时间可能相差几秒，这对结果可能会产生影响。在我目前的水平来说，并没有办法完美解决这个问题，通过百度也得知，几乎无法在让线程绝对的同时创建，这是我比较遗憾的一个地方了。

在实验中我也发现我很容易进入思维的定势。就如同调试过程中提及的一样，很容易被文字的字面意思所误导。这要求我应该反思自己，并改正这个不良的习惯。

1. **用户使用说明**

按照需求分析里的要求创建一个测试数据文件，将其命名为test.txt并放到源代码目录下的src目录（与主要代码同级）；或者直接修改源代码目录下的src目录中的test.txt文件，输入想要测试的数据，之后运行Main程序即可看到结果（按照java方式运行）。

1. **测试与运行结果**

## （一）、第一次测试

测试文件数据如下：

1 R 3 5

2 R 4 5

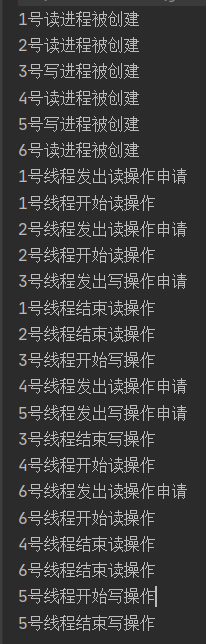
3 W 5 2

4 R 10 3

5 W 11 3

6 R 13 4

测试结果如下：



## （二）、第二次测试

测试文件数据如下：

1 W 2 4

2 R 3 5

3 R 5 2

4 W 7 4

5 R 11 3

测试结果如下：

