读者-写者问题

老师同学们大家好，我们是第7小组，我们组研讨的题目是使用信号量解决读者-写者问题的读写公平、写者优先算法，我是张俊雄，我先开始我这部分的介绍。

记录型信号量

用户进程可以通过使用操作系统提供的一对**原语**来对**信号量**进行操作，从而很方便的实现了进程互斥、进程同步。

原语是一种特殊的程序段，其执行只能**一气呵成，不可被中断**。

记录型信号量除了一个代表资源数目的整形变量外，还要增加一个进程链表用于链接所有等待该资源的进程。

p操作：相当于申请资源，当 `S.value < 0` 时，表示该资源已经分配完毕，需要调用block()原语使进程从运行态变为阻塞态，并把当前进程挂到信号量S的等待队列当中。

v操作：相当于释放资源，当 `S.value ⩽ 0` 时，表示当前仍有等待该资源的进程被阻塞在信号量S的等待队列当中，需要调用wakeup()原语将S.L中的第一个进程从阻塞态变为就绪态。

用信号量实现互斥和同步

当两个进程对同一临界资源进行访问时，就会出现互斥操作。

1. 设置互斥信号量mutex，并设初值为1。
2. 在访问临界资源之前，执行p(mutex)., 把mutex的值-1，表示当前有进程正在访问临界资源。
3. 在结束访问之后，执行v(mutex)，把mutex的值+1，表示访问临界资源的进程访问结束。

> 要实现对一个操作的互斥，只需要在该操作前后分别执行p操作和v操作。

当我们为了保证并发进程代码的执行顺序时，就需要同步操作。

例如，在进程p1中的代码x必须要在进程p2中的代码y之前执行。

1. 设置同步信号量s，初值为0。
2. 在先要执行的代码之后执行v操作。
3. 在后执行的代码之前执行p操作。

下面由胡才郁介绍具体的读者写者算法

这一部分和前面一个小组内容查不多，我就简单过一下。

读者写者问题的几个基本要求是一样的，主要是要把握他们之间的互斥关系，写者与写者、写者与读者之间都是互斥的，而读者之间不互斥。对于这样比较复杂的互斥关系，实现方式是设置一个整数变量readcount记录当前同时访问文件的读进程数量。并且第一个读进程负责加锁，最后一个读进程负责解锁。**（翻页）**

对于写进程而言，要做的就是不断的写文件，当写进程访问临界资源文件的时候需要使用PV操作把临界区夹起来。当第一个写进程进入后，通过P操作，第二个写进程到来时会被卡在P(rw)这里了，直到第一个写进程写完文件后，对rw进行V操作，释放rw，第二个写进程才可以继续访问临界资源。

读者进程相对而言复杂一点。在一个读进程读文件之前，需要先检查有无其他的读进程正在进行读操作。如果检查到没有其他读进程，也就是判断readcount等于0时，那么它作为第一个读进程，对文件进行加锁操作，并且计数变量加1。当它读完文件之后，计数变量减1，并且判断此时这个读进程是不是唯一的读进程，如果是唯一的读进程，需要对文件进行解锁。这样设置的话，当第一个读进程到来时，它对文件加锁，使得写进程无法访问，同时其它的读进程的readcount不等于0，也可以直接访问文件。**（点击）**

但是这样设置有一个弊端，如果两个读进程并发执行，假设他们同时到达判断readcount是否为0的语句，1号读进程来了判断为true，接着执行P(rw)，但是它没有来得及做readcount++操作，此时时间片分给了2号读进程，由于readcount还没来得及修改，2号读进程此时的readcount也是0，它所以也会进行P(rw)操作，这两个读进程先后执行P(rw)，从而使得第二个读进程出现阻塞。**（翻页）**

对于这种现象的解决方法是设置read_mutex互斥信号量，readcount成为临界资源，用来保证每一个读进程对于readcount的访问是互斥的，此时。还是刚刚那种情况，对于reader_mutex执行P操作只允许一个读进程访问readcount，当2号读进程到来时会等到1号读进程释放readcount时才能访问readcount。**（翻页）**

不过这样还有一个问题，由于读进程之间没有互斥的限制，哪怕此时写进程排队在读进程之前，设想一个极端情况，当读进程一个接着一个到来时，由于没有最后一个读进程解锁，写进程还是无法访问文件。导致只有无论排队顺序，只有当没有读进程在排队的时候，写进程才可以访问。这样的算法为读者优先，读进程优先级高于写进程，只有当不存在读进程时，写进程才可以访问文件。**（翻页）**

而写者优先算法是指，当写者到来时，队列中此写者之前的读者仍继续读操作，但之后到来的读进程都阻塞，直到队列中不存在写进程在排队。之后到来的写进程会“插队“，即使后续读进程排队在前面，也必须等到队列中没有写进程时才可以继续读文件。实现写者优先需要多设置一些变量，额外设置了3个变量，分别是记录写进程排队的数量、对于写进程互斥访问的信号量、以及实现写进程优先互斥的信号量。读者进程相比于读者优先算法，只是多了一步在进入前申请read权限。而写者进程比较复杂，当写者进程到来时，先判断此时还有没有其他写者，如果存在其他写者的话，那么就限制了其他读者的权限。此时已经有权限的读者进程就不管了，但之后再来的读者进程就会被堵塞。就像疫情封校限制跨片区流动，已经跨了片区的同学就不管了，再来的同学除非有蓝码，否则就不放行。具体到伪代码就是当第二个写者进程来的时候**（点两下）**，对read进行P操作，此后所有的读进程再来就被卡在外面。**（点三下）**当写者进程离开的时候，如果写者队列为空就给予读权限,对read进行V操作。此时读者进程就可以来读了。

至于为什么要设置writer_mutex，实际上是和设置reader_mutex原理是一样的，为了保护writercount计数变量，防止writercount来不及修改时时间片用完，两个写进程都if判断成true了，P(read)执行了两次。**（点一下）**

接下来模拟一种情况。假设1号写者正在读，1号写者来了，由于writecount为0，不会对read进行P操作，而1号读者来了readcount等于0，并且1号写进程已经对rw做了P操作，所以1号读进程会堵塞在if(readcount == 0) P(rw)这里。此时2号写者来了，writercount不为0，会对read进行P操作，限制了读取权限。此时再来一个2号读者，他进行P(read)时被阻塞。如果这个时候来了一个3号写者，按照正常的先后顺序，应该是2号读者比3号写者先执行，因为是2号读者先来的嘛，但是由于此时2号读者没有read权限，还被卡在P(read)这里。所以3号写者比2号读者优先，并且这个权限只有当队列中没有写

者才会赋予，只要有写者在排队，读者就都无法读。这就是写者优先算法。**（看代码）**

下面我们来看一下读者优先算法和写者优先算法的代码，由于时间原因我们就粗略过一下，然后看我们的实验结果。我们用c语言的线程库和信号量库模拟PV操作，实际上就是把刚才的伪代码跑起来。**之后自由发挥**

读写无优先

> 1. 写者和写者之间互斥
> 2. 读者和读者之间不互斥
> 3. 写者和读者之间互斥
>
> 在前面的读优先代码，存在的问题：只要有读进程还在读，写进程就要一直阻塞等待，可能“饿死”。

> 1. 写者先执行时，其余写者和读者就不可能执行。
> 2. 读者先执行时
>   1. 另一个读者也执行，但只有先后执行完p(w)和v(w)后才能同时执行读文件操作；
>   2. 写者此时不可能执行。

代码部分

信号量、控制变量设置

```
semaphore rw = 1; // 用于实现对文件的互斥访问 表示当前是否有进程访问文件
int readcount = 0; // 记录当前有几个读进程在访问文件
semaphore reader_mutex = 1; // 用于保证对readcount变量的互斥访问
```

读进程优先

写者

```
writer(){
  while(1){
    P(rw);
    写文件....
    V(rw);
  }
}
```

读者

```
reader(){
  while(1){

    P(reader_mutex); // 各读进程互斥访问reader_count
    if(readcount == 0)
      P(rw);  // 第一个读进程负责加锁
    readcount++; // 读者在读数+1
    V(reader_mutex);

    读文件...

    P(reader_mutex); // 各读进程互斥访问count
    readcount--; // 读者在读数-1
    if(readcount == 0)
      V(rw);  // 最后一个一个读进程负责解锁
    V(reader_mutex);
  }
}
```

写进程优先

写者

```
writer(){
  while(1){
    P(writer_mutex); // 各写进程互斥访问writercount
    if(writercount != 0) // 如果当前写者队列不为空则申请读者优先
      P(read);
    writecount++; // 写者数量+1
```

```
    V(writer_mutex);

    P(rw);
    写文件....
    V(rw);

    P(writer_mutex); // 各写进程互斥访问writercount
    writecount--; // 写者数量-1
    if(writecount == 0) // 如果写者队列为空则允许读者进行操作
      V(read);
    V(writer_mutex); // 各写进程互斥访问writercount
  }
}
```

读者

```
reader(){
  while(1){
    P(read);  // 申请读取权限

    P(reader_mutex); // 各读进程互斥访问reader_count
    if(readcount == 0)
      P(rw);  // 第一个读进程负责加锁
    readcount++; // 读者在读数+1
    V(reader_mutex);

    读文件...

    P(reader_mutex); // 各读进程互斥访问count
    readcount--; // 读者在读数-1
    if(readcount == 0)
      V(rw);  // 最后一个一个读进程负责解锁
    V(reader_mutex);
  }
}
```