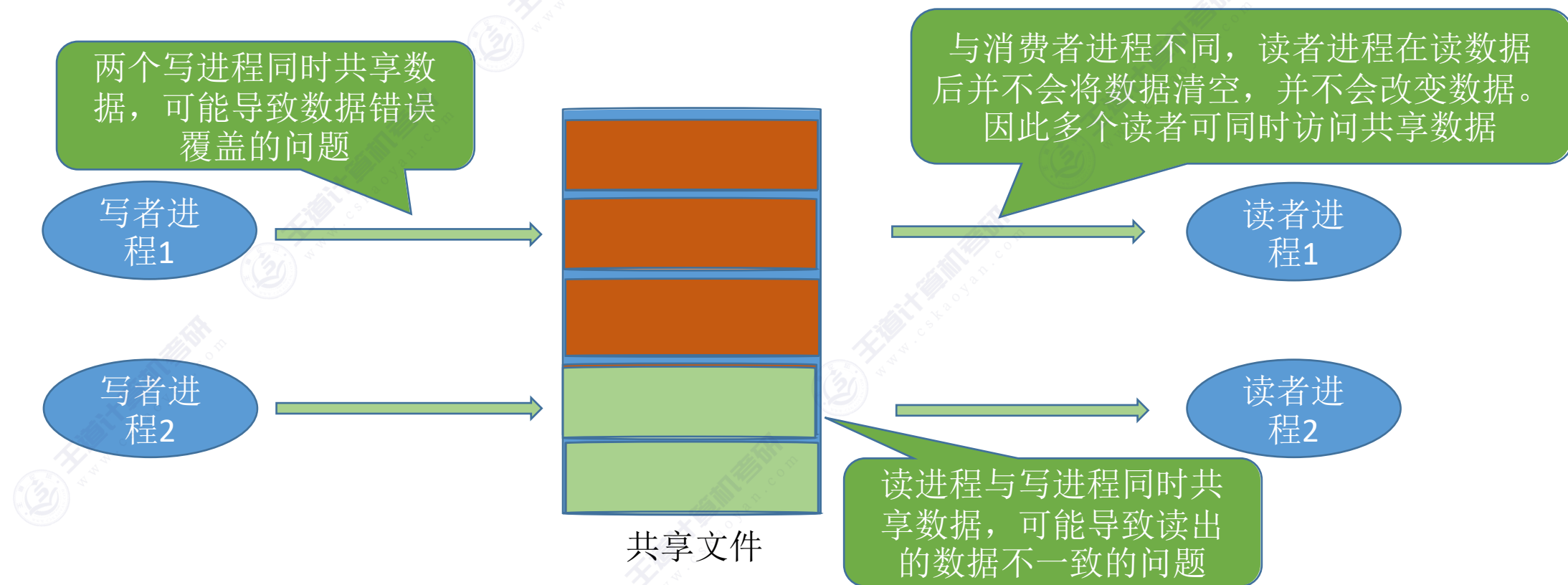


本节内容

读者-写者 问题

问题描述

有读者和写者两组并发进程，共享一个文件，当两个或两个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。因此要求：①允许多个读者可以同时的文件执行读操作；②只允许一个写者往文件中写信息；③任一写者在完成写操作之前不允许其他读者或写者工作；④写者执行写操作前，应让已有的读者和写者全部退出。



问题分析

有读者和写者两组并发进程，共享一个文件，当两个或两个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。因此要求：①允许多个读者可以同时同时对文件执行读操作；②只允许一个写者往文件中写信息；③任一写者在完成写操作之前不允许其他读者或写者工作；④写者执行写操作前，应让已有的读者和写者全部退出。

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序
3. 设置信号量。设置需要的信号量，并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初始值要看对应资源的初始值是多少）

两类进程：写进程、读进程

互斥关系：写进程—写进程、写进程—读进程。读进程与读进程不存在互斥问题。

如何实现

```
semaphore rw=1;          //用于实现对共享文件的互斥访问
int count = 0;           //记录当前有几个读进程在访问文件
semaphore mutex = 1;     //用于保证对count变量的互斥访问
```

```
writer () {
    while(1) {
        P(rw);    //写之前“加锁”
        写文件...
        V(rw);    //写完了“解锁”
    }
}
```

思考：若两个读进程并发执行，则 `count=0` 时两个进程也许都能满足 `if` 条件，都会执行 `P(rw)`，从而使第二个读进程阻塞的情况。
如何解决：出现上述问题的原因在于对 `count` 变量的检查和赋值无法一气呵成，因此可以设置另一个互斥信号量来保证各读进程对 `count` 的访问是互斥的。

潜在的问题：只要有读进程还在读，写进程就要一直阻塞等待，可能“饿死”。因此，这种算法中，读进程是优先的

```
reader () {
    while(1) {
        P(mutex);    //各读进程互斥访问count
        if(count==0) //由第一个读进程负责
            P(rw);    //读之前“加锁”
        count++;      //访问文件的读进程数+1
        V(mutex);
        读文件...
        P(mutex);    //各读进程互斥访问count
        count--;      //访问文件的读进程数-1
        if(count==0) //由最后一个读进程负责
            V(rw);    //读完了“解锁”
        V(mutex);
    }
}
```

如何实现

```
semaphore rw=1;      //用于实现对共享文件的互斥访问
int count = 0;        //记录当前有几个读进程在访问文件
semaphore mutex = 1;  //用于保证对count变量的互斥访问
semaphore w = 1;      //用于实现“写优先”
```

分析以下并发执行 **P(w)** 的情况:

读者1→读者2

写者1→写者2

写者1→读者1

读者1→写者1→读者2

写者1→读者1→写者2

结论: 在这种算法中, 连续进入的多个读者可以同时读文件; 写者和其他进程不能同时访问文件; 写者不会饥饿, 但也并不是真正的“写优先”, 而是相对公平的先来先服务原则。

有的书上把这种算法称为“读写公平法”。

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

```
reader () {
    while(1) {
        P(w);
        P(mutex);
        if(count==0)
            P(rw);
        count++;
        V(mutex);
        V(w);
        读文件...
        P(mutex);
        count--;
        if(count==0)
            V(rw);
        V(mutex);
    }
}
```

知识回顾与重要考点

读者-写者问题为我们解决复杂的互斥问题提供了一个参考思路。

其**核心思想**在于设置了一个**计数器 count** 用来记录当前正在访问共享文件的读进程数。我们可以用 count 的值来判断当前进入的进程是否是第一个/最后一个读进程，从而做出不同的处理。

另外，对 count 变量的检查和赋值不能一气呵成导致了一些错误，如果**需要实现“一气呵成”**，自然应该想到用**互斥信号量**。

最后，还要认真体会我们是如何解决“写进程饥饿”问题的。

绝大多数的考研PV操作大题都可以用之前介绍的几种生产者-消费者问题的思想来解决，如果遇到更复杂的问题，可以想想能否用读者写者问题的这几个思想来解决。



公众号：王道在线



b站：王道计算机教育



抖音：王道计算机考研