

本节内容

用信号量机制实现  
进程互斥、同步、前驱关系

王道考研/CSKAOYAN.COM

知识总览

信号量机制

实现进程互斥

实现进程同步

实现进程的前驱关系

Tips: 不要一头钻到代码里，要注意理解信号量背后的含义，一个信号量对应一种资源

信号量的值 = 这种资源的剩余数量（信号量的值如果小于0，说明此时有进程在等待这种资源）

P(S) —— 申请一个资源S，如果资源不够就阻塞等待

V(S) —— 释放一个资源S，如果有进程在等待该资源，则唤醒一个进程

王道考研/CSKAOYAN.COM

理解：信号量 mutex 表示  
“进入临界区的名额”

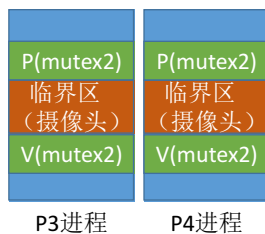
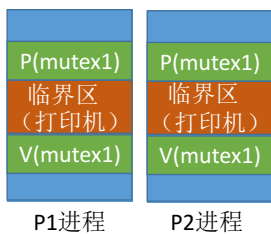
## 信号量机制实现进程互斥

1. 分析并发进程的关键活动，划定临界区（如：对临界资源打印机的访问就应放在临界区）
2. 设置互斥信号量 mutex，初值为 1
3. 在进入区 P(mutex)——申请资源
4. 在退出区 V(mutex)——释放资源

注意：对不同的临界资源需要设置不同的互斥信号量。

P、V操作必须成对出现。缺少 P(mutex) 就不能保证临界资源的互斥访问。缺少 V(mutex) 会导致资源永不释放，等待进程永不唤醒。

要自己定义记录型信号量，但如果题目中没特别说明，可以把信号量的声明简写成这种形式



```
/*记录型信号量的定义*/
typedef struct {
    int value;           // 剩余资源数
    struct process *L;   // 等待队列
} semaphore;
```

```
/*信号量机制实现互斥*/
semaphore mutex=1; // 初始化信号量

P1(){
    ...
    P(mutex);       // 使用临界资源前需要加锁
    临界区代码段...
    V(mutex);       // 使用临界资源后需要解锁
    ...
}

P2(){
    ...
    P(mutex);
    临界区代码段...
    V(mutex);
    ...
}
```

王道考研/CSKAOYAN.COM

## 信号量机制实现进程同步

进程同步：要让各并发进程按要求有序地推进。

```
P1(){
    代码1;
    代码2;
    代码3;
}

P2(){
    代码4;
    代码5;
    代码6;
}
```

比如，P1、P2 并发执行，由于存在异步性，因此二者交替推进的次序是不确定的。

若 P2 的“代码4”要基于 P1 的“代码1”和“代码2”的运行结果才能执行，那么我们就必须保证“代码4”一定是在“代码2”之后才会执行。

这就是进程同步问题，让本来异步并发的进程互相配合，有序推进。

王道考研/CSKAOYAN.COM

## 信号量机制实现进程同步

用信号量实现进程同步：

1. 分析什么地方需要实现“同步关系”，即必须保证“一前一后”执行的两个操作（或两句代码）
2. 设置同步信号量  $S$ ，初始为 0
3. 在“前操作”之后执行  $V(S)$
4. 在“后操作”之前执行  $P(S)$

技巧口诀：前V后P

理解：信号量  $S$  代表“某种资源”，刚开始是没有这种资源的。P2 需要使用这种资源，而又只能由 P1 产生这种资源

/\*信号量机制实现同步\*/

semaphore  $S=0$ ; // 初始化同步信号量，初始值为 0

```
P1(){
    代码1;
    代码2;
    V(S);
    代码3;
}
```

```
P2(){
    P(S);
    代码4;
    代码5;
    代码6;
}
```

释放资源

保证了 代码4 一定是在 代码2 之后执行

若先执行到  $V(S)$  操作，则  $S++$  后  $S=1$ 。之后当执行到  $P(S)$  操作时，由于  $S=1$ ，表示有可用资源，会执行  $S--$ ， $S$  的值变回 0，P2 进程不会执行 block 原语，而是继续往下执行代码4。

若先执行到  $P(S)$  操作，由于  $S=0$ ， $S--$  后  $S=-1$ ，表示此时没有可用资源，因此P操作中会执行 block 原语，主动请求阻塞。之后当执行完代码2，继而执行  $V(S)$  操作， $S++$ ，使  $S$  变回 0，由于此时有进程在该信号量对应的阻塞队列中，因此会在  $V$  操作中执行 wakeup 原语，唤醒 P2 进程。这样 P2 就可以继续执行 代码4 了

王道考研/CSKAOYAN.COM

## 信号量机制实现前驱关系

进程 P1 中有句代码 S1，P2 中有句代码 S2，P3 中有句代码 S3 ..... P6 中有句代码 S6。这些代码要求按如下前驱图所示的顺序来执行：

其实每一对前驱关系都是一个进程同步问题（需要保证一前一后的操作）因此，

1. 要为每一对前驱关系各设置一个同步信号量
2. 在“前操作”之后对相应的同步信号量执行  $V$  操作
3. 在“后操作”之前对相应的同步信号量执行  $P$  操作

```
P1() {
    ...
    S1;
    V(a);
    V(b);
    ...
}

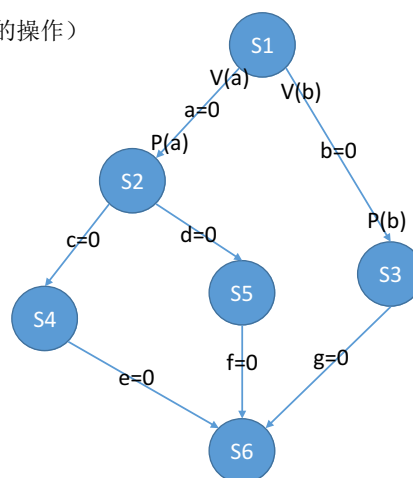
P2() {
    ...
    P(a);
    S2;
    V(c);
    V(d);
    ...
}

P3() {
    ...
    P(b);
    S3;
    V(g);
    ...
}

P4() {
    ...
    P(c);
    S4;
    V(e);
    ...
}

P5() {
    ...
    P(d);
    S5;
    V(f);
    ...
}

P6() {
    ...
    P(e);
    P(f);
    P(g);
    S6;
    ...
}
```



王道考研/CSKAOYAN.COM

