

哈工大操作系统-L17对信号量的临界区保护

哈工大操作系统-L17对信号量的临界区保护

- 1.共同修改共享的信号量会有问题
- 2.竞争条件(Race Condition)
- 3.引出临界区和原子操作
- 4.临界区(Critical Section)
- 5.临界区代码的保护三原则
- 6.进入临界区设计的尝试
 - 6.1轮换法
 - 6.2普通标记法
 - 6.3非对称标记法
- 7.两进程的Peterson算法
- 8.多进程的面包店算法
- 9.上述软件方法复杂, 需要硬件方法
 - 9.1阻止中断
 - 9.2原子指令

靠临界区保护信号量, 靠信号量实现进程的同步。

1.共同修改共享的信号量会有问题

共同修改信号量引出的问题

初始情况

`empty = -1;`

这是什么含义?

```
Producer(item) {  
    P(empty);  
    ...  
}
```

一个可能的执行(调度)

```
P1.register = empty;  
P1.register = P1.register - 1;  
P2.register = empty;  
P2.register = P2.register - 1;  
empty = P1.register;  
empty = P2.register;
```

生产者 P_1

```
register = empty;  
register = register - 1;  
empty = register;
```

因为并发执行的进程, 执行的顺序是无法预测的, 所以每个进程都有可能任意一个地方开始切换。

生产者 P_2

```
register = empty;  
register = register - 1;  
Empty = register;
```

因此, 设计需要保证无论在什么地方切换都保证不出错, 同时规定一些不能切换的地方。

最终的empty等于多少?对吗?

最终应该是-2, 因为P1的修改被P2覆盖了

- 这就是为什么信号量需要保护的原因。
- 内核中所有的共享数据都需要保护。

2.竞争条件(Race Condition)

竞争条件(Race Condition)

■ **竞争条件:** 和调度有关的共享数据语义错误

第i次执行

```
P1.register = empty;  
P1.register = P1.register - 1;  
P2.register = empty;  
P2.register = P2.register - 1;  
empty = P1.register;  
empty = P2.register;
```

第j次执行

```
P1.register = empty;  
P1.register = P1.register - 1;  
empty = P1.register;  
P2.register = empty;  
P2.register = P2.register - 1;  
empty = P2.register;
```

- 错误由多个进程并发操作共享数据引起
- 错误和调度顺序有关，难于发现和调试

问题：右面的图这两个人的方法有效果吗？

- 竞争条件: 和调度有关的共享数据语义错误
- 错误由多个进程并发操作共享数据引起
- 错误和调度顺序有关，难于发现和调试

3.引出临界区和原子操作

解决竞争条件的直观想法

■ 在写共享变量**empty**时阻止其他进程也访问**empty**

仍是那个执行序列

```
P1.register = empty;  
P1.register = P1.register - 1;  
P2.register = empty;  
P2.register = P2.register - 1;  
empty = P1.register;  
empty = P2.register;
```

生产者P₁

检查并给**empty**上锁

```
P1.register = empty;  
P1.register = P1.register - 1;
```

生产者P₂

检查**empty**的锁 ?

生产者P₁

empty = P₁.register;

给**empty**开锁

生产者P₂

检查并给**empty**上锁

```
P2.register = empty;  
P2.register = P2.register - 1;  
empty = C.register;
```

给**empty**开锁

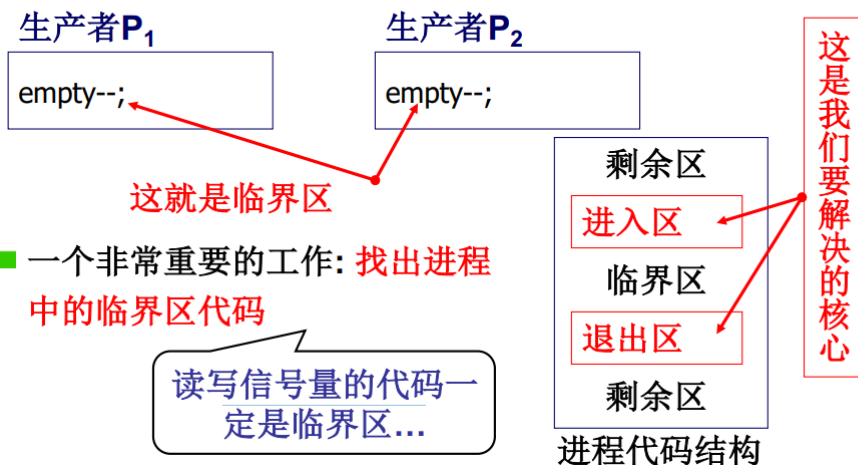
一段代码一次只允许一个进程进入

- 原子操作：原子操作内的代码必须完整的被执行，中途允许被中断打断或者切换出去

4.临界区(Critical Section)

临界区(Critical Section)

■ **临界区**: 一次只允许一个进程进入的该进程的那一段代码



- **临界区**: 一次只允许一个进程进入的该进程的那一段代码
 - 即, 如果一些进程都有访问共享数据的代码, 那么为了保证这些共享数据的正确性, 在一个进程执行访问共享数据的代码的时候, 别的进程是不能执行访问共享数据的代码的
 - 把每个进程中访问共享数据的代码的段, 称为**临界区**
 - **读写信号量的代码一定是临界区...**
- 一个非常重要的工作: 找出进程中的临界区代码
- 我们的目的就是: **合理设计临界区的进入和退出机制, 保证共享数据的正确性**

5.临界区代码的保护三原则

临界区代码的保护原则

■ **基本原则: 互斥进入**: 如果一个进程在临界区中执行, 则其他进程不允许进入

■ 这些进程间的约束关系称为**互斥(mutual exclusion)**

■ **这保证了是临界区**

■ **好的临界区保护原则**

■ **2. 有空让进**: 若干进程要求进入空闲临界区时, 应尽快使一进程进入临界区

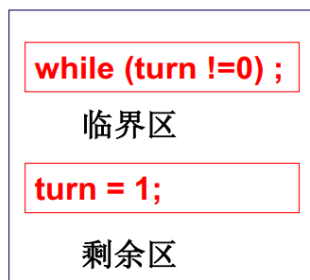
■ **3. 有限等待**: 从进程发出进入请求到允许进入, 不能无限等待

- **互斥进入**--基本原则
 - 互斥进入: 如果一个进程在临界区中执行, 则其他进程不允许进入。
 - 这些进程间的约束关系称为互斥(mutual exclusion)
 - 互斥进入保证了共享数据的正确性
- **有空让进**: 临界区空闲就可以进入
- **有限等待**: 不能让一些进程无限等待, 防止造成饥饿

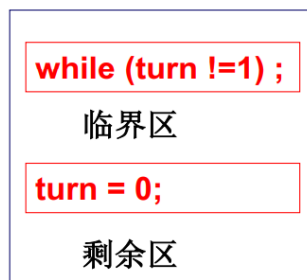
6.进入临界区设计的尝试

6.1 轮换法

进入临界区的一个尝试 – 轮换法



进程 P_0



进程 P_1

- 问题: P_0 完成后不能接着再次进入, 尽管进程 P_1 不在临界区...(不满足有空让进)

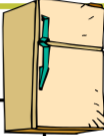
■ 满足互斥进入要求

- 满足互斥的要求
- 但不满足有空让进
 - P_0 进入退出临界区之后, 必须等 P_1 进入退出临界区一次, 才能轮到 P_0
 - 如果 P_1 一直不执行到临界区, 临界区一直空闲, P_0 也没法进入

6.2 普通标记法

进入临界区的又一个尝试

- 似乎没有任何头绪... 可借鉴生活中的道理



时间	丈夫	妻子
3:00	打开冰箱, 没有牛奶了	
3:05	离开家去商店	
3:10	到达商店	打开冰箱, 没有牛奶了
3:15	买牛奶	离开家去商店
3:20	回到家里, 牛奶放进冰箱	到达商店
3:25		买牛奶
3:30		回到家里, 牛奶放进冰箱

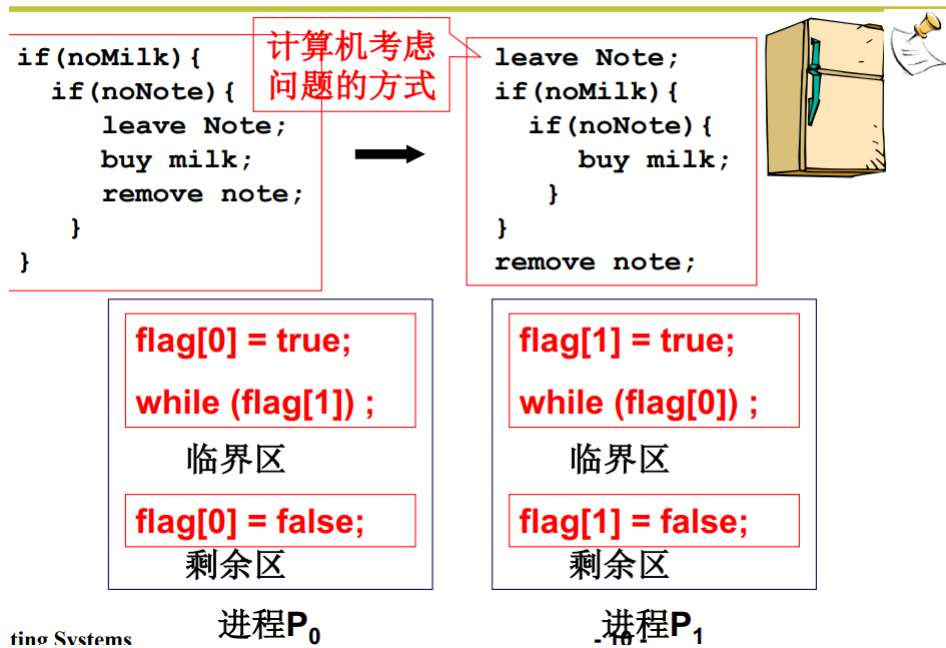
- 上面的轮换法类似于什么? 值日

- 更好的方法应该是立即去买, 留一个便条

许多复杂的道理往往就埋藏在日常生活中!



进入临界区的又一个尝试 – 标记法



标记法能否解决问题？

■ 考虑下面的执行顺序

因为是可以随意执行的，如果我们按照1~4的顺序执行标记，多进程每个进程是看不见别人的会产生死锁

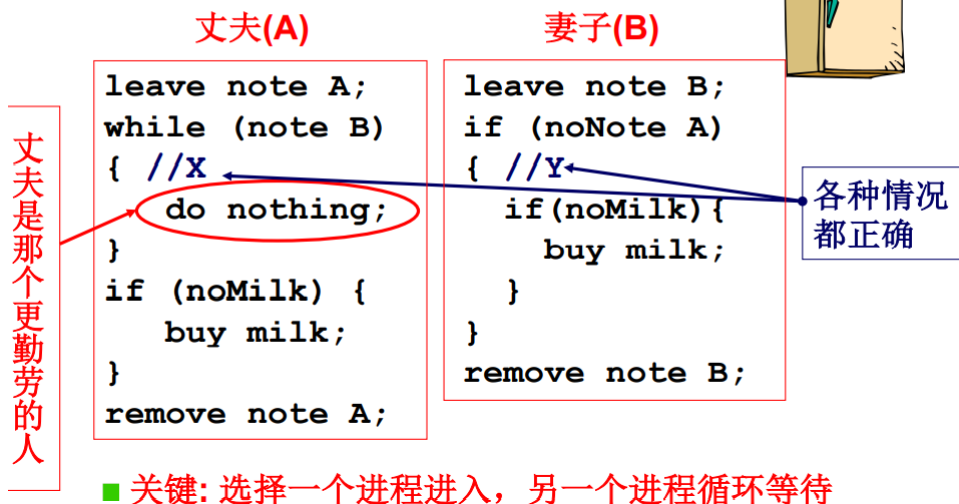


- 满足互斥
- 不满足有空让进
 - 普通标记会造成死锁

6.3非对称标记法

进入临界区的再一次尝试 – 非对称标记

- 带名字的便条 + 让一个人更加勤劳



7.两进程的Peterson算法

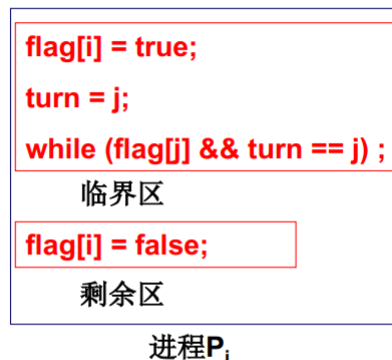
进入临界区Peterson算法

- 结合了标记和轮转两种思想



Peterson算法的正确性

- 满足互斥进入:
如果两个进程都进入, 则 $flag[0]=flag[1]=true$, $turn==0==1$, 矛盾!
- 满足有空让进:
如果进程 P_1 不在临界区, 则 $flag[1]=false$, 或者 $turn=0$, 都 P_0 能进入!
- 满足有限等待:
 P_0 要求进入, $flag[0]=true$; 后面的 P_1 不可能一直进入, 因为 P_1 执行一次就会让 $turn=0$ 。



• Peterson算法概述

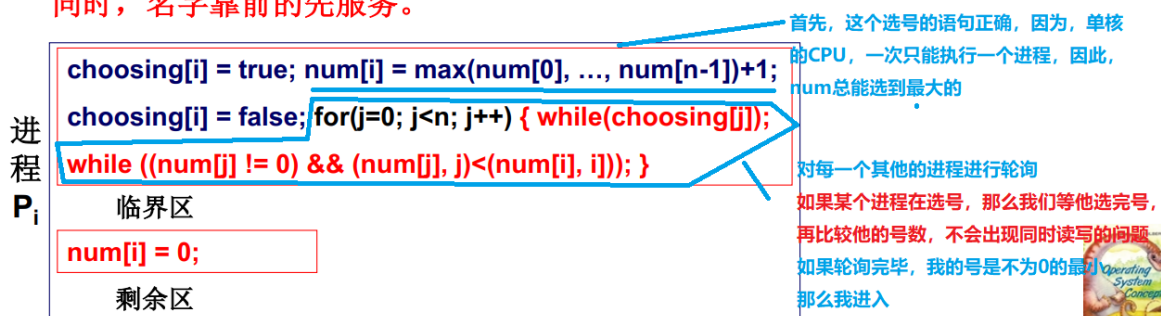
- 进入前先标记自己想进入
- 然后，把turn让渡给另一个人
- 如果 别人没有标记想进入&&turn也不是轮到别人的话，自己就进入
- 为何满足有空让进：
 - 如果一个进程不在临界区有三种状况
 - 执行到turn之后但还未进入临界区，让给别人执行，此时turn是别人，别人可以进入临界区
 - 执行到turn之前flag为ture，但还未进入临界区，让给别人执行，此时别人也访问不了，等待。但是时间片过后，就又到我们自己了，我们自己是可进入临界区的，不存在死锁的问题
 - 执行到flag为false之后，别人都可以进入临界区
- 为何满足有限等待：
 - 因为一个进程执行过后，会让渡turn给别人，如果别人要求执行，那么别人肯定能进入临界区执行
 - 如果别人不要求执行，我们也还能继续执行
 - 但这个turn的作用就是，我执行过了，别人如果突然要求执行，别人的优先级高。但如果无人要求执行，我还是可以继续执行
- 要一直询问几个标记的值，也是一种忙等的行为

8.多进程的面包店算法

多个进程怎么办？ – 面包店算法

■ 仍然是标记和轮转的结合

- **如何轮转**: 每个进程都获得一个序号，
- **如何标记**: 进程离开时序号为0，不为0的序号最小的进入
- **面包店**: 每个进入商店的客户都获得一个号码，号码最小的先得到服务；号码相同时，名字靠前的先服务。



• 面包店算法概述

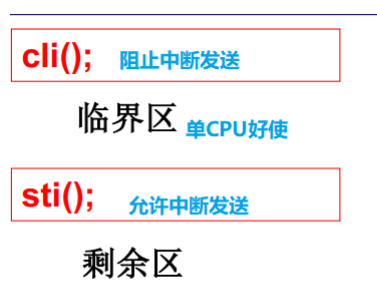
- 想进临界区先取号。
 - 取号前告诉别人我在取号。作用是让别人等我取完号再跟我比较。
 - 然后取一个比最大的都大1的号。
 - 告诉别人我取完号了
- 逐个逐个问别人号数多少
 - 先看别人是否在取号，是，就等他取完再比较，否，就直接比较
 - 如果我号数是所有人里面最小的，我进入临界区；否则我一直轮询跟别人比
- 是一种忙等的行为

9. 上述软件方法复杂，需要硬件方法

9.1 阻止中断

临界区保护的另两类解法...

- 再想一下临界区：只允许一个进程进入，进入另一个进程意味着什么？
- 被调度：另一个进程只有被调度才能执行，才可能进入临界区，如何阻止调度？



■ 什么时候不好使？

■ 多CPU(多核)...

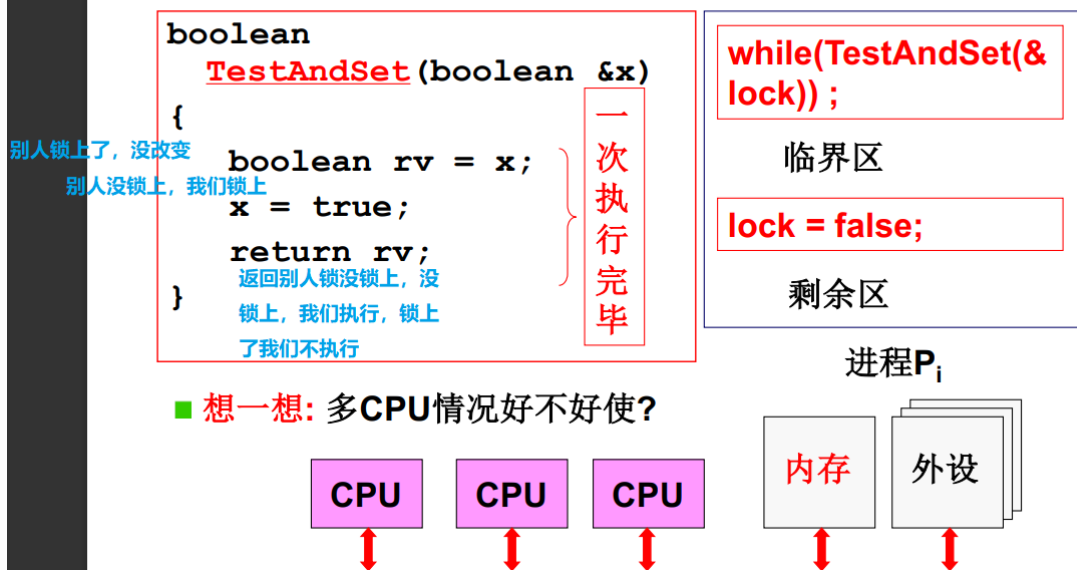
问题：为什么不好使？

没办法控制其他的CPU的中断

- 单CPU好使
- 多CPU不好使

9.2 原子指令

临界区保护的硬件原子指令法



■ 想一想：多CPU情况好不好使？

- 用锁对共享资源保护
- 用原子指令设置只能单人开锁和关锁，那么就实现了对锁的保护