

本节内容

进程互斥的软件实现方法



- 学习提示:
1. 理解各个算法的思想、原理
 2. 结合上小节学习的“实现互斥的四个逻辑部分”，重点理解各算法在进入区、退出区都做了什么
 3. 分析各算法存在的缺陷（结合“实现互斥要遵循的四个原则”进行分析）

单标志法

算法思想：两个进程在访问完临界区后会把使用临界区的权限转交给另一个进程。也就是说每个进程进入临界区的权限只能被另一个进程赋予

`int turn = 0;` //turn 表示当前允许进入临界区的进程号

| | | | |
|---------------------------------|---|---------------------------------|---------|
| P0 进程: | | P1进程: | |
| <code>while (turn != 0);</code> | ① | <code>while (turn != 1);</code> | ⑤ //进入区 |
| <code>critical section;</code> | ② | <code>critical section;</code> | ⑥ //临界区 |
| <code>turn = 1;</code> | ③ | <code>turn = 0;</code> | ⑦ //退出区 |
| <code>remainder section;</code> | ④ | <code>remainder section;</code> | ⑧ //剩余区 |

turn 的初值为 0，即刚开始只允许 0 号进程进入临界区。
若 P1 先上处理机运行，则会一直卡在 ⑤。直到 P1 的时间片用完，发生调度，切换 P0 上处理机运行。
代码 ① 不会卡住 P0，P0 可以正常访问临界区，在 P0 访问临界区期间即时切换回 P1，P1 依然会卡在 ⑤。
只有 P0 在退出区将 turn 改为 1 后，P1 才能进入临界区。

因此，该算法可以实现“同一时刻最多只允许一个进程访问临界区”

单标志法

算法思想：两个进程在访问完临界区后会把使用临界区的权限转交给另一个进程。也就是说每个进程进入临界区的权限只能被另一个进程赋予

`int turn = 0;` //turn 表示当前允许进入临界区的进程号

| | | | |
|---------------------------------|---|---------------------------------|---------|
| P0 进程: | | P1进程: | |
| <code>while (turn != 0);</code> | ① | <code>while (turn != 1);</code> | ⑤ //进入区 |
| <code>critical section;</code> | ② | <code>critical section;</code> | ⑥ //临界区 |
| <code>turn = 1;</code> | ③ | <code>turn = 0;</code> | ⑦ //退出区 |
| <code>remainder section;</code> | ④ | <code>remainder section;</code> | ⑧ //剩余区 |

turn 表示当前允许进入临界区的进程号，而只有当前允许进入临界区的进程在访问了临界区之后，才会修改 turn 的值。也就是说，对于临界区的访问，一定是按 P0 → P1 → P0 → P1 →这样轮流访问。
这种必须“轮流访问”带来的问题是，如果此时允许进入临界区的进程是 P0，而 P0 一直不访问临界区，那么虽然此时临界区空闲，但是并不允许 P1 访问。
因此，单标志法存在的主要问题是：违背“空闲让进”原则。

双标志先检查法

算法思想：设置一个布尔型数组 `flag[]`，数组中各个元素用来标记各进程想进入临界区的意愿，比如“`flag[0] = true`”意味着 0 号进程 P0 现在想要进入临界区。每个进程在进入临界区之前先检查当前有没有别的进程想进入临界区，如果没有，则把自身对应的标志 `flag[i]` 设为 `true`，之后开始访问临界区。

```
bool flag[2];           //表示进入临界区意愿的数组
flag[0] = false;
flag[1] = false;        //刚开始设置为两个进程都不想进入临界区
```

P0 进程: P1 进程:

```
while (flag[1]); ① while (flag[0]); ⑤ //如果此时 P0 想进入临界区, P1 就一直循环等待
flag[0] = true; ② flag[1] = true; ⑥ //标记为 P1 进程想要进入临界区
critical section; ③ critical section; ⑦ //访问临界区
flag[0] = false; ④ flag[1] = false; ⑧ //访问完临界区, 修改标记为 P1 不想使用临界区
remainder section; remainder section;
```

若按照 ①⑤②⑥③⑦... 的顺序执行，P0 和 P1 将会同时访问临界区。因此，双标志先检查法的主要问题是：违反“忙则等待”原则。原因在于，进入区的“检查”和“上锁”两个处理不是一气呵成的。“检查”后，“上锁”前可能发生进程切换。

王道考研/CSKAOYAN.COM

双标志后检查法

算法思想：双标志先检查法的改版。前一个算法的问题是“先检查”后“上锁”，但是这两个操作又无法一气呵成，因此导致了两个进程同时进入临界区的问题。因此，人们又想到先“上锁”后“检查”的方法，来避免上述问题。

```
bool flag[2];           //表示进入临界区意愿的数组
flag[0] = false;
flag[1] = false;        //刚开始设置为两个进程都不想进入临界区
```

P0 进程: P1 进程:

```
flag[0] = true; ① flag[1] = true; ⑤ //标记为 P1 进程想要进入临界区
while (flag[1]); ② while (flag[0]); ⑥ //如果 P0 也想进入临界区, 则 P1 循环等待
critical section; ③ critical section; ⑦ //访问临界区
flag[0] = false; ④ flag[1] = false; ⑧ //访问完临界区, 修改标记为 P1 不想使用临界区
remainder section; remainder section;
```

若按照 ①⑤②⑥... 的顺序执行，P0 和 P1 将都无法进入临界区。因此，双标志后检查法虽然解决了“忙则等待”的问题，但是又违背了“空闲让进”和“有限等待”原则，会因各进程都长期无法访问临界资源而产生“饥饿”现象。两个进程都争着想进入临界区，但是谁也不让谁，最后谁都无法进入临界区。

王道考研/CSKAOYAN.COM

Peterson 算法

算法思想：双标志后检查法中，两个进程都争着想进入临界区，但是谁也不让谁，最后谁都无法进入临界区。Gary L. Peterson 想到了一种方法，如果双方都争着想进入临界区，那可以让进程尝试“孔融让梨”，主动让对方先使用临界区。

```
bool flag[2];           //表示进入临界区意愿的数组, 初始值都是false
int turn = 0;           //turn 表示优先让哪个进程进入临界区
```

P0 进程:

```
flag[0] = true; ①
turn = 1; ② 进入区
while (flag[1] && turn==1); ③
critical section; ④
flag[0] = false; ⑤ 退出
remainder section;
```

P1 进程:

```
flag[1] = true; ⑥ //表示自己进入临界区
turn = 0; ⑦ //可以优先让对方进入临界区
while (flag[0] && turn==0); ⑧ //对方想进, 且最后一次是自己“让梨”, 那自己就循环等待
critical section; ⑨
flag[1] = false; ⑩ //访问完临界区, 表示自己已经不想访问临界区了
remainder section;
```

两种双标志法的问题都是由于进入区的几个操作不能一气呵成导致的。我们可以推理验证在 Peterson 算法中，两个进程进入区中的各个操作按不同的顺序穿插执行会发生什么情况：

①②③⑥⑦⑧...
①⑥②③...
①③⑥⑦⑧...

王道考研/CSKAOYAN.COM

Peterson 算法

算法思想：双标志后检查法中，两个进程都争着想进入临界区，但是谁也不让谁，最后谁都无法进入临界区。Gary L. Peterson 想到了一种方法，如果双方都争着想进入临界区，那可以让进程尝试“孔融让梨”，主动让对方先使用临界区。

```
bool flag[2];           //表示进入临界区意愿的数组, 初始值都是false
int turn = 0;           //turn 表示优先让哪个进程进入临界区
```

P0 进程:

```
flag[0] = true; ①
turn = 1; ②
while (flag[1] && turn==1); ③
critical section; ④
flag[0] = false; ⑤
remainder section;
```

P1 进程:

```
flag[1] = true; ⑥
turn = 0; ⑦
while (flag[0] && turn==0); ⑧
critical section; ⑨
flag[1] = false; ⑩
remainder section;
```

进入区: 1. 主动争取; 2. 主动谦让; 3. 检查对方是否也想使用, 且最后一次是不是自己说了“客气话”

该算法的思想其实我们在生活中也会使用... 举一个有味道的栗子.....



香香想用马桶

臭臭想用马桶

可以让臭臭先用，可以让香香先用

经检查发现香香也要使用，但最后是臭臭说了“客气话”，因此臭臭循环等待

按①②③...的顺序

按①⑥②⑦⑧...的顺序



香香



臭臭

王道考研/CSKAOYAN.COM

Peterson 算法

算法思想：双标志后检查法中，两个进程都争着想进入临界区，但是谁也不让谁，最后谁都无法进入临界区。Gary L. Peterson 想到了一种方法，如果双方都争着想进入临界区，那可以让进程尝试“孔融让梨”，主动让对方先使用临界区。

```
bool flag[2];           //表示进入临界区意愿的数组，初始值都是false
int turn = 0;           //turn 表示优先让哪个进程进入临界区
```

P0 进程:

```
flag[0] = true;          ①
turn = 1;                ②
while (flag[1] && turn==1); ③
critical section;        ④
flag[0] = false;         ⑤
remainder section;
```

P1 进程:

```
flag[1] = true;          ⑥
turn = 0;                ⑦
while (flag[0] && turn==0); ⑧
critical section;        ⑨
flag[1] = false;         ⑩
remainder section;
```

进入区: 1. 主动争取; 2. 主动谦让; 3. 检查对方是否也想使用, 且最后一次是不是自己说了“客气话”

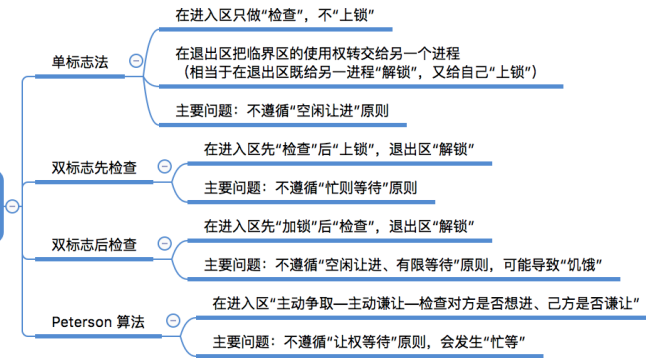
Peterson 算法用软件方法解决了进程互斥问题, 遵循了**空闲让进、忙则等待、有限等待**三个原则, 但是依然**未遵循让权等待**的原则。

Peterson 算法相较于之前三种软件解决方案来说, 是最好的, 但依然不够好。

王道考研/CSKAOYAN.COM

知识回顾与重要考点

进程互斥的软件实现方法



王道考研/CSKAOYAN.COM