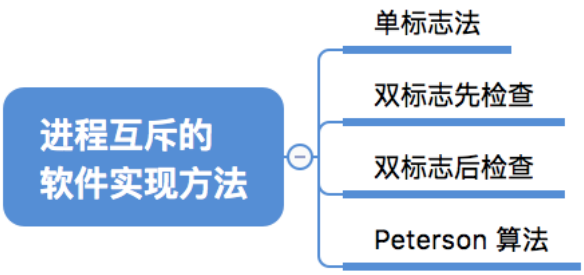
本节内容

进程互斥的软件实现方法

1

知识总览



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）

2

如果没有注意进程互斥？

进程A、进程B在系统中并发地运行

```
进程A:
{
    其他代码;
    使用打印机;
    其他代码;
}
```

```
进程B:
{
    其他代码;
    使用打印机;
    其他代码;
}
```

先调度A上处理机运行
当A在使用打印机的过程中，分配给它的时间片用完了，接下来操作系统调度B让它上处理机运行
进程B也在使用打印机

结局：A、B 的打印内容混在一起了

如何实现进程互斥？

3

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
```

turn 变量背后的逻辑：表达“谦让”

```
P0 进程:                          P1进程:
while (turn != 0);      ①         while (turn != 1);      ⑤ //进入区
critical section;       ②         critical section;       ⑥ //临界区
turn = 1;               ③         turn = 0;               ⑦ //退出区
remainder section;      ④         remainder section;      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”

4

单标志法

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

turn 变量背后的逻辑：表达“谦让”

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

①是否轮到自己用？（检查）
②访问临界资源，啦啦啦~
③下次让老渣用（表达谦让）
④做其他事情


小渣


马桶


老渣

⑤是否轮到自己用？（检查）
⑥访问临界资源，拉拉拉~
⑦下次让小渣用（表达谦让）
⑧做其他事情

只能按 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。

5

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] = ture”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
```

理解背后的含义：“表达意愿”

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;		

进入区

①老渣是否想用（检查）
②我小渣要用（表达意愿，上锁）
③访问临界资源，啦啦啦~
④小渣不用了（解锁）


小渣


马桶


老渣

⑤小渣是否想用（检查）
⑥我老渣要用（表达意愿，上锁）
⑦访问临界资源，拉拉拉~
⑧老渣不用了（解锁）

6

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] = ture”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
```

理解背后的含义：“表达意愿”

```
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;
```

进入区

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。

执行完①之后 进程切换

7

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
```

理解背后的含义：“表达意愿”

```
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;
```

进入区

①我小渣要用（表达意愿，上锁）
②老渣是否想用（检查）
③访问临界资源，啦啦啦~
④小渣不用了（解锁）

小渣

马桶

老渣

⑤我老渣要用（表达意愿，上锁）
⑥小渣是否想用（检查）
⑦访问临界资源，拉拉拉~
⑧老渣不用了（解锁）

8

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;
```

理解背后的含义：“表达意愿”

进入区

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。

①⑤②⑥ 死执行顺序

王道考研/CSKAOYAN.COM

9

Peterson 算法

算法思想：结合双标志法、单标志法的思想。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”（谦让）。做一个有礼貌的进程。

```
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;
```

背后的含义：“表达意愿”

表达“谦让”

进入区


小渣


马桶


老渣

①我小渣要用（表达意愿）
②我愿意优先让老渣用（谦让）
③老渣想用？且最后是自己表示了谦让？我就等待（检查）
④访问临界资源，啦啦啦~
⑤我小渣不想用了

⑥我老渣要用（表达意愿）
⑦我愿意优先让小渣用（谦让）
⑧小渣想用？且最后是自己表示了谦让？我就等待（检查）
⑨访问临界资源，拉拉拉~
⑩我老渣不想用了

动手推导：
按不同的顺序穿插执行会发生什么？
①②③⑥⑦⑧...
①⑥②③...
①③⑥⑦⑧...
①⑥②⑦⑧...

王道考研/CSKAOYAN.COM

10

Peterson 算法

算法思想：结合双标志法、单标志法的思想。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”（谦让）。做一个有礼貌的进程。

```
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 //表示自己想进入临界区
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;
```

背后的含义：“表达意愿”

表达“谦让”

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”

谁最后说了“客气话”，谁就失去了行动的优先权。
Eg: 过年了，某阿姨给你发压岁钱。

场景一
阿姨：乖，收下阿姨的心意~
你：不用了阿姨，您的心意我领了
阿姨：对阿姨来说你还是个孩子，你就收下吧
结局...

11

Peterson 算法

算法思想：结合双标志法、单标志法的思想。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”（谦让）。做一个有礼貌的进程。

```
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 //表示自己想进入临界区
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;
```

背后的含义：“表达意愿”

表达“谦让”

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”

谁最后说了“客气话”，谁就失去了行动的优先权。
Eg: 过年了，某阿姨给你发压岁钱。

场景二
阿姨：乖，收下阿姨的心意~
你：不用了阿姨，您的心意我领了
阿姨：对阿姨来说你还是个孩子，你就收下吧
你：真的不用了阿姨，我已经成年了
结局...

12

Peterson 算法

算法思想：结合双标志法、单标志法的思想。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”（谦让）。做一个有礼貌的进程。

```
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”

动手推导：
按不同的顺序穿插执行会发生什么？
①②③⑥⑦⑧...
①⑥②③...
①③⑥⑦⑧...
①⑥②⑦⑧...

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。

13

知识回顾与重要考点

turn 谦让

flag 争取

- 进程互斥的软件实现方法
  - 单标志法
    - 在进入区只做“检查”，不“上锁”
    - 在退出区把临界区的使用权转交给另一个进程（相当于在退出区既给另一进程“解锁”，又给自己“上锁”）
    - 主要问题：不遵循“空闲让进”原则
  - 双标志先检查
    - 在进入区先“检查”后“上锁”，退出区“解锁”
    - 主要问题：不遵循“忙则等待”原则
  - 双标志后检查
    - 在进入区先“加锁”后“检查”，退出区“解锁”
    - 主要问题：不遵循“空闲让进、有限等待”原则，可能导致“饥饿”
  - Peterson 算法
    - 在进入区“主动争取—主动谦让—检查对方是否想进、己方是否谦让”
    - 主要问题：不遵循“让权等待”原则，会发生“忙等”

未释放CPU

14

bilibili

@王道论坛

新浪微博 weibo.com

@王道计算机考研备考

@王道咸鱼老师-计算机考研

@王道楼楼老师-计算机考研

抖音 短视频

@王道计算机考研

知乎

@王道计算机考研

微信视频号

@王道计算机考研

微信公众平台

@王道在线

15