



操作系统

Operating system

孔维强

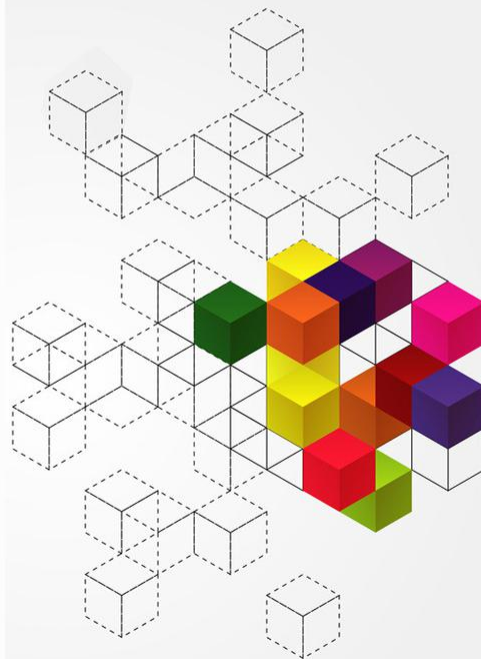
大连理工大学

一、两进程解法-尝试1

二、两进程解法-尝试2

三、Peterson算法

四、多进程软件解法



一、两进程解法-尝试1

两进程P0,P1

Try1: 使用一个共享变量 $turn$,
初值为0(或1)

$i=0$ 或 1
 $j=1-i$

```
do{  
    <进入区>  
    <临界区>  
    <退出区>
```

```
    <剩余区>  
}while(1);
```

进程 P_i :

```
do{
```

```
    while( $turn \neq i$ );
```

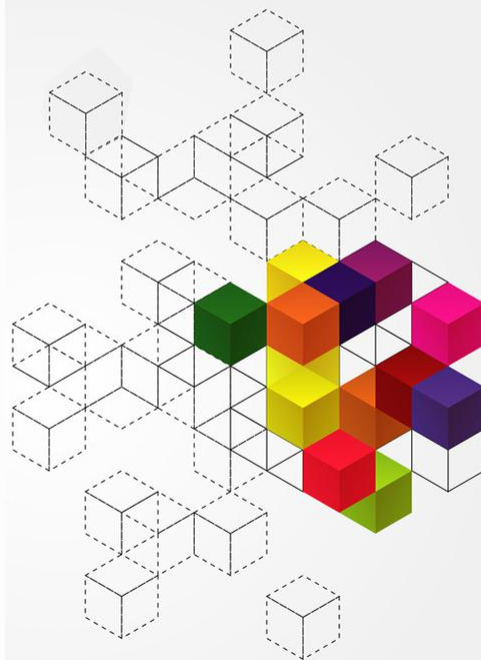
```
    <临界区>
```

```
     $turn = j$ ;
```

```
    <剩余区>
```

```
}while(1);
```

问题: 如果 P_0 循环体执行速度快于 P_1 , 可能会出现 P_0 处于 $turn=1$, 但是 P_1 实质是在剩余区
(P_0 空闲不让进)



二、两进程解法-尝试2

两进程P0,P1

进程Pi:

```
do{
```

```
    while(turn!=i);
```

```
    <临界区>
```

```
    turn=j;
```

```
    <剩余区>
```

```
}while(1);
```

i=0或1

j=1-i

Try2: 用共享布尔数组flag[2]
来替代共享变量turn

进程Pi:

```
do{
```

```
    flag[i]=true;
```

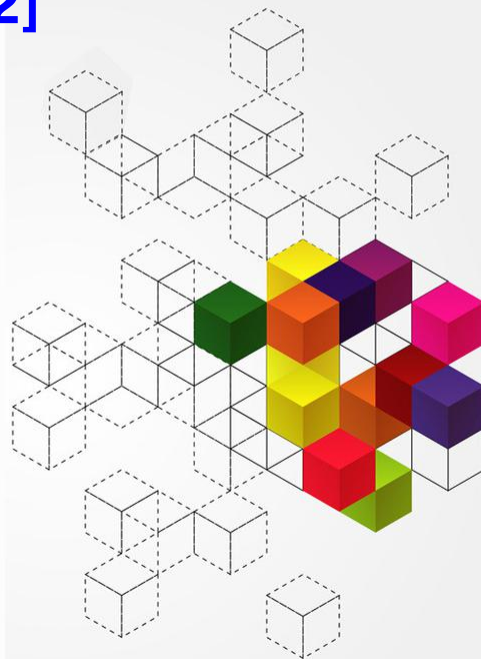
```
    while(flag[j]);
```

```
    <临界区>
```

```
    flag[i]=false;
```

```
    <剩余区>
```

```
}while(1);
```



二、两进程解法-尝试2

两进程P0,P1

Try2解决方案

进程Pi:

```
do{
```

```
    flag[i]=true;  
    while(flag[j]);
```

```
    <临界区>
```

```
    flag[i]=false;
```

```
    <剩余区>
```

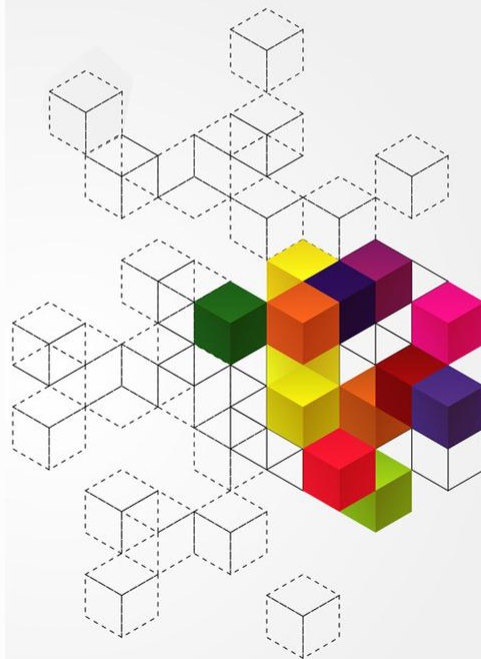
```
}while(1);
```

i=0或1

j=1-i

问题：考虑到进程P0、P1执行时的异步并发特性，可能会出现P0和P1的准入标志flag[0],flag[1]均为true，使得进程P0，P1陷入死循环的问题

(P0，P1空闲不让进)



三、两进程解法-Peterson算法

两进程P0,P1

Try2解决方案

进程Pi:

```
do{
```

```
    flag[i]=true;  
    while(flag[j]);
```

```
    <临界区>
```

```
    flag[i]=false;
```

```
    <剩余区>
```

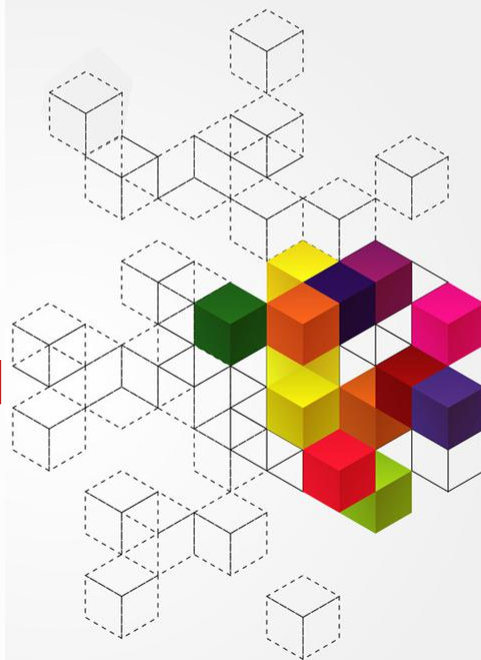
```
}while(1);
```

i=0或1

j=1-i

如果解决了死循环问题，两
进程互斥的问题即可解决

Peterson给出的方案：
共享变量turn + 数组flag[2]



三、两进程解法-Peterson算法

两进程P0,P1

Try2解决方案

进程Pi:

```
do{
```

```
    flag[i]=true;  
    while(flag[j]);
```

```
    <临界区>
```

```
    flag[i]=false;
```

```
    <剩余区>
```

```
    }while(1);
```

i=0或1

j=1-i

Try3 (Peterson算法)

进程Pi:

```
do{
```

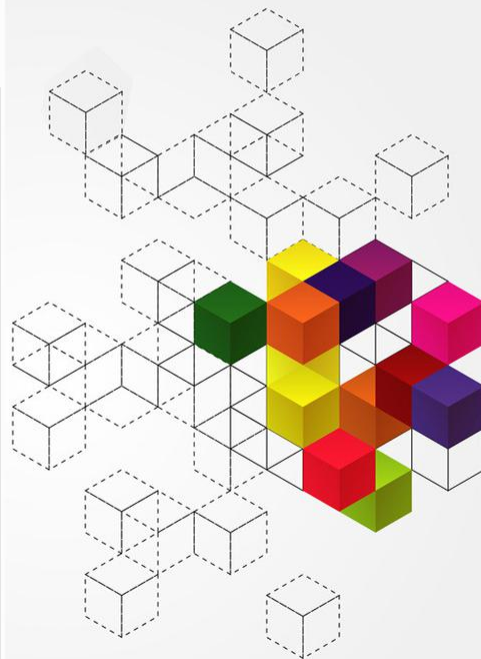
```
    flag[i]=true;  
    turn=j;  
    while(flag[j]  
          &&turn==j);
```

```
    <临界区>
```

```
    flag[i]=false;
```

```
    <剩余区>
```

```
    }while(1);
```



三、两进程解法-Peterson算法

```
var flag: array[0..1] of boolean;  
    turn: 0..1;  
procedure P0;  
begin  
    repeat  
        flag[0] := true;  
        turn := 1;  
    while (flag[1] and turn=1) { };  
        <critical section> ;  
        flag[0] := false;  
        <remainder>  
    forever  
end;  
procedure P1;  
begin  
    repeat
```

```
        flag[1] := true;  
        turn := 0;  
    while (flag[0] and turn=0) { };  
        <critical section> ;  
        flag[1] := false;  
        <remainder>  
    forever  
end;  
begin  
    flag[0] := false;  
    flag[1] := false;  
    turn := 1;  
    parbegin  
        P0; P1  
    parend  
end.
```



三、两进程解法-Peterson算法

- 可证明Peterson算法满足临界区的3个条件:

1. 互斥

P_i enters CS only if:

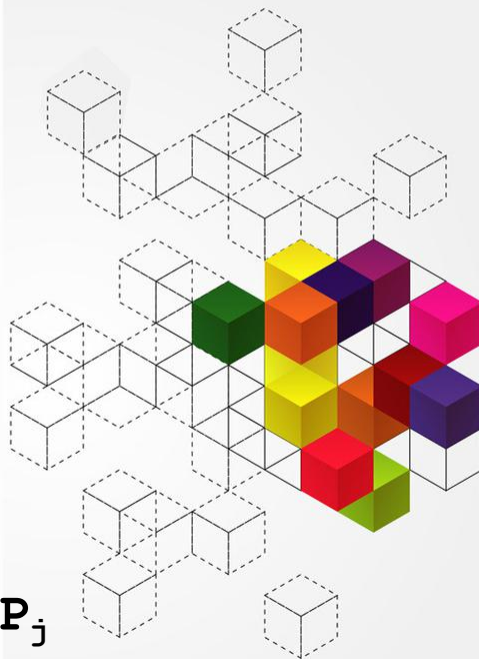
either `flag[j] = false` or `turn = i`

2. 前进

P_i will enter CS

3. 有限等待

P_i will enter at most one entry by P_j



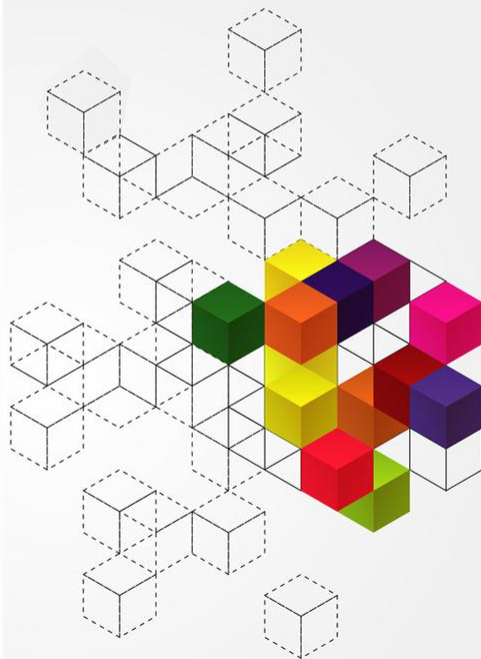
四、多进程软件解法

多进程互斥（面包店算法）

```
do{
    choosing[i]=true;
    number[i]=max(number[0],...,number[n-1])+1;
    choosing[i]=false;

    for(j=0; j<n; j++){
        while(choosing[j]);
        while((number[j]!=0) &&(number[j], j)<(number[i], i));
    }
    临界区
    number[i]=0;
    剩余区
}while(1);
```

取时间戳



四、多进程软件解法

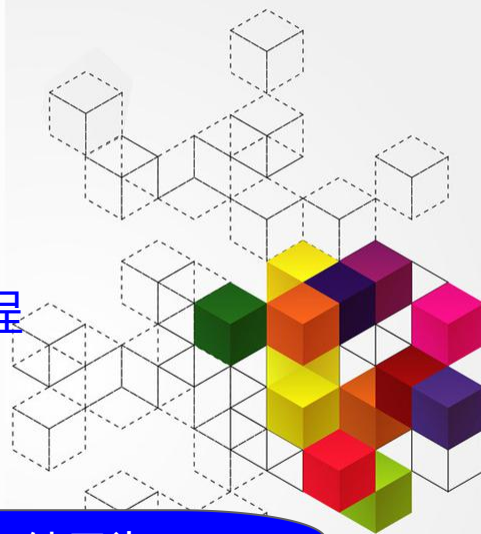
● 多进程互斥（面包店算法）

```
do{
    choosing[i]=true;
    number[i]=max(number[0],...,number[n-1])+1;
    choosing[i]=false;

    for(j=0;j<n;j++){
        while(choosing[j]);
        while((number[j]!=0) &&(number[j],j)<(number[i],i));
    }
    临界区
    number[i]=0;
    剩余区
}while(1);
```

看有没有更“老”的进程

当 $\text{number}[i] < \text{number}[j]$, 结果为true;
当 $\text{number}[j] == \text{number}[i]$, 且 $j < i$, 结果为true



本讲小结

- 两进程解法-尝试1
- 两进程解法-尝试2
- 两进程解法-Peterson算法
- 多进程软件解法

