



操作系统

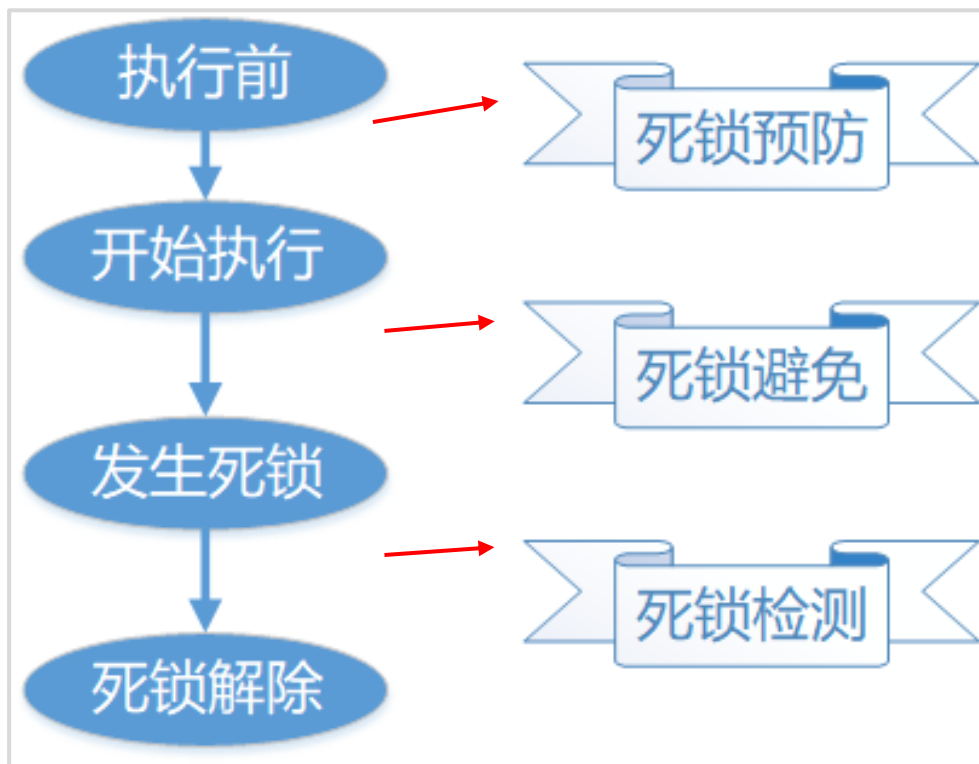
L12 死锁2

胡燕
大连理工大学 软件学院

死锁处理方法

Handling Deadlocks

01



如何防患于未然?



死锁预防思路1：进程必须获取工作所需所有资源，才能开展工作

进程：运行前申请所需全部资源

系统：对于任意一个进程的资源申请，如果能够满足，则一次性全部分配；否则，让进程等待

Each process try to acquire all the resources it needs before the process run

破坏 “hold-and-wait” 条件

缺点：

- 资源利用效率低
- 由于系统很难同时满足多个进程的一次性资源需求，可能出现大量进程等待现象



- 图中交通拥堵画面（不和谐）
- 制定规则：禁止绿灯跟进
- 规则仍需要司机主动遵守

试分析：禁止绿灯跟进，是否与此死锁预防思路一致？

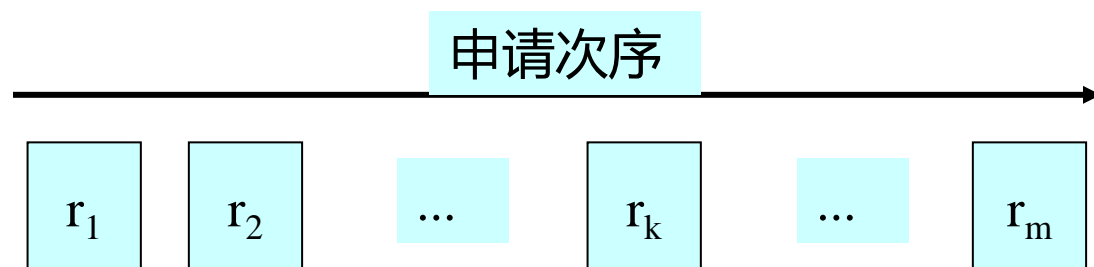


死锁预防思路2：要求进程按序使用资源

对资源进行编号，约定进程按照编号大小顺序对资源进行分配

资源集： $R = \{r_1, r_2, \dots, r_n\}$

函数编号： $F: R \rightarrow N$ (为资源定一个级别)



进程 p_i 可以申请资源 r_j 中的实例 $\Leftrightarrow \forall r_l, p_i$ 占有 r_l , 有 $F(r_l) < F(r_j)$

例如： $R = \{\text{scanner}, \text{tape}, \text{printer}\}$

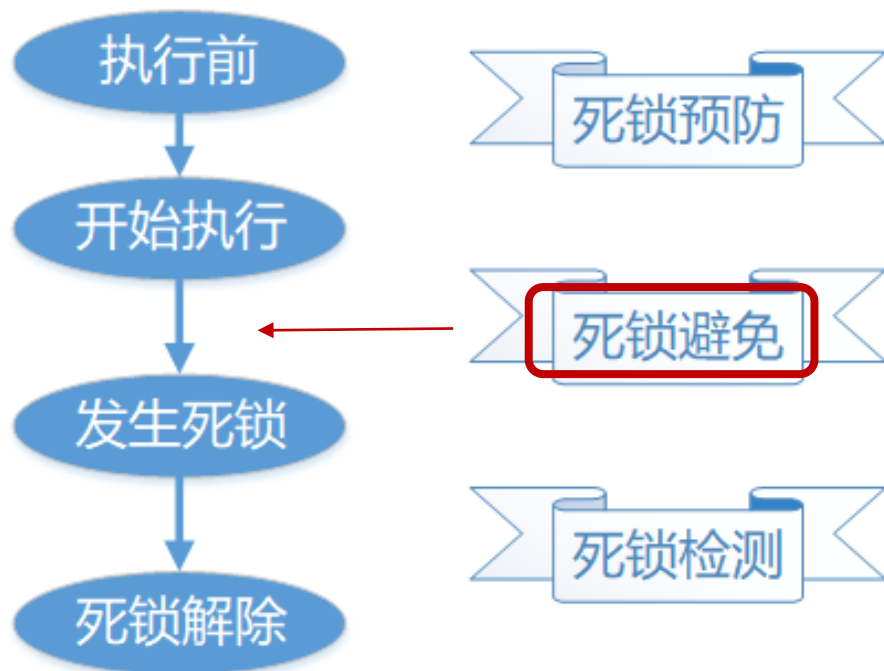
$F(\text{scanner}) = 1;$

$F(\text{tape}) = 2;$

$F(\text{printer}) = 3;$

要求： 进程代码中申请资源必须按照先申请scanner,再申请tape, 最后申请printer的原则进行

破坏 “循环等待” 条件



Deadlock Avoidance

How to avoid deadlock?

死锁避免的基本思路

好比在悬崖上跳舞，每一次脚悬空准备落下来时，都得面对一个问题

安全吗？会不会踩空掉下去啊？

若不安全：就把本来要踏出去脚再收回来



关键：准确判断safety

建立安全感

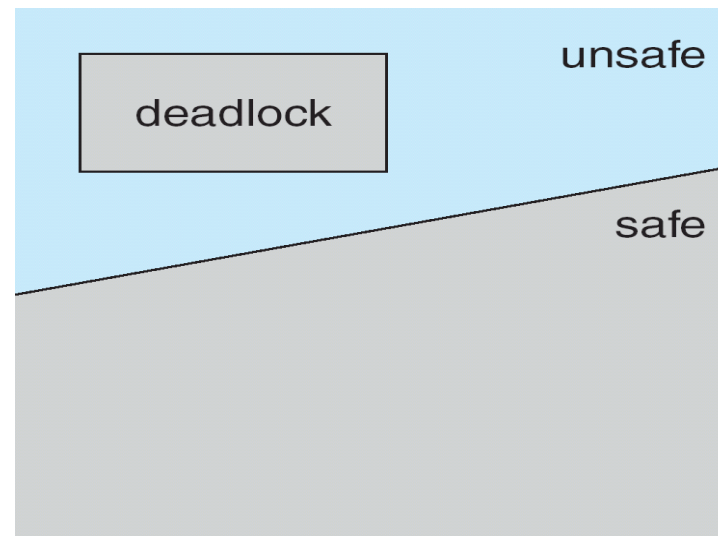


死锁避免的基本思路

对每次进程资源申请，都做严格的审查

- 确定安全，则允许分配
- 不能确定安全，则不允许分配，进程等待

目标：保证系统始终处于安全的资源使用状态



死锁避免的关键要素：对安全状态的定义

最终：

safe状态 – 不会死锁

Unsafe状态 – 存在死锁风险



死锁避免中的安全状态

死锁的特性:

参与死锁的进程，每个都持有部分资源，并且待申请的其他的资源都在死锁进程集中的其他进程手里，从而所有参与死锁进程都无法继续推进

死锁避免的安全状态判定:

如果能够找到一个顺序，使得所有进程可以依次获得进一步执行所需的资源，从而依次执行结束，那么可以判定系统状态安全（不会死锁）

安全序列: p_1, p_2, \dots, p_n , 使得依次

- 让 p_i 获得所需全部资源，使得 p_i 执行结束，并随后释放 p_i 原本持有的资源回资源池，接着 p_{i+1} 可以获得所需全部资源，并得以计算结束 (i从1到n-1)

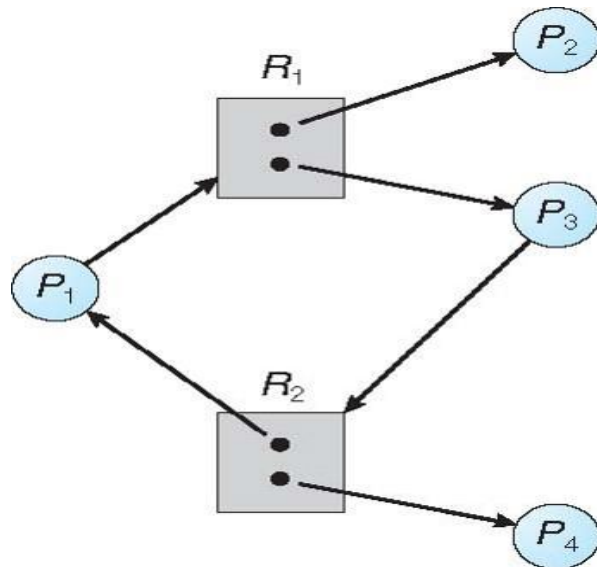
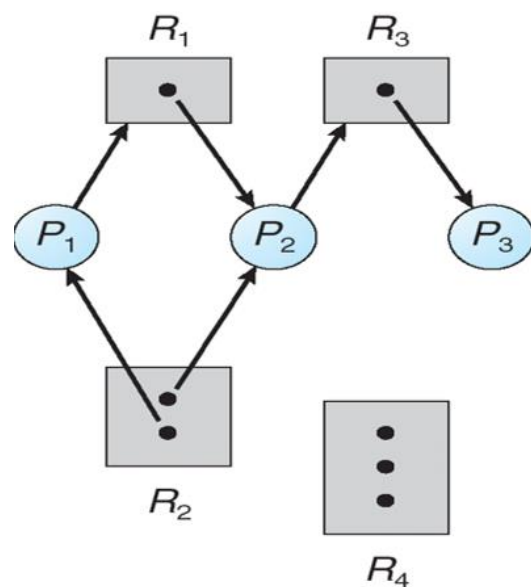
Safe状态 \Leftrightarrow 可找到安全序列

死锁避免的实现思路

在确定当前状态安全的基础上，对每一次进程资源申请，都做一个预判：
即，如果通过进程的此次申请，系统到达的状态是否还安全

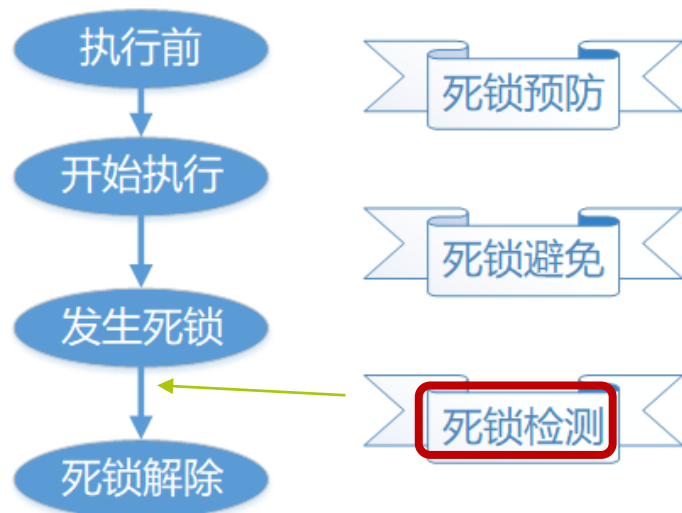
如何实现每一步的状态安全判断，是死锁避免的关键

如何实施呢？



在资源分配图模型下，找出安全序列，近似于在选取图中的一个**拓扑排序**

练习：试找出上述图示状态中的安全序列



基本思路：不采取任何预防或避免死锁的措施，当死锁发生时再决定如何处理

解决方案：

- (1) 检测死锁，并解除
- (2) 不处理，忽略（鸵鸟机制）

考虑到死锁检测的高昂代价，鸵鸟机制被多数通用操作系统采用

死锁避免

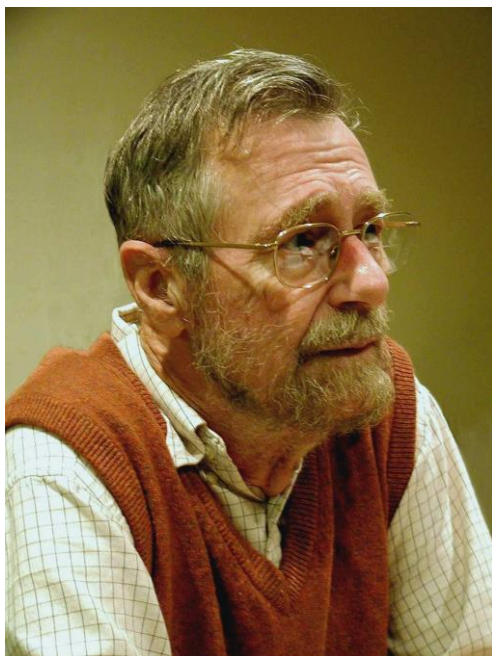
Deadlock Avoidance

02

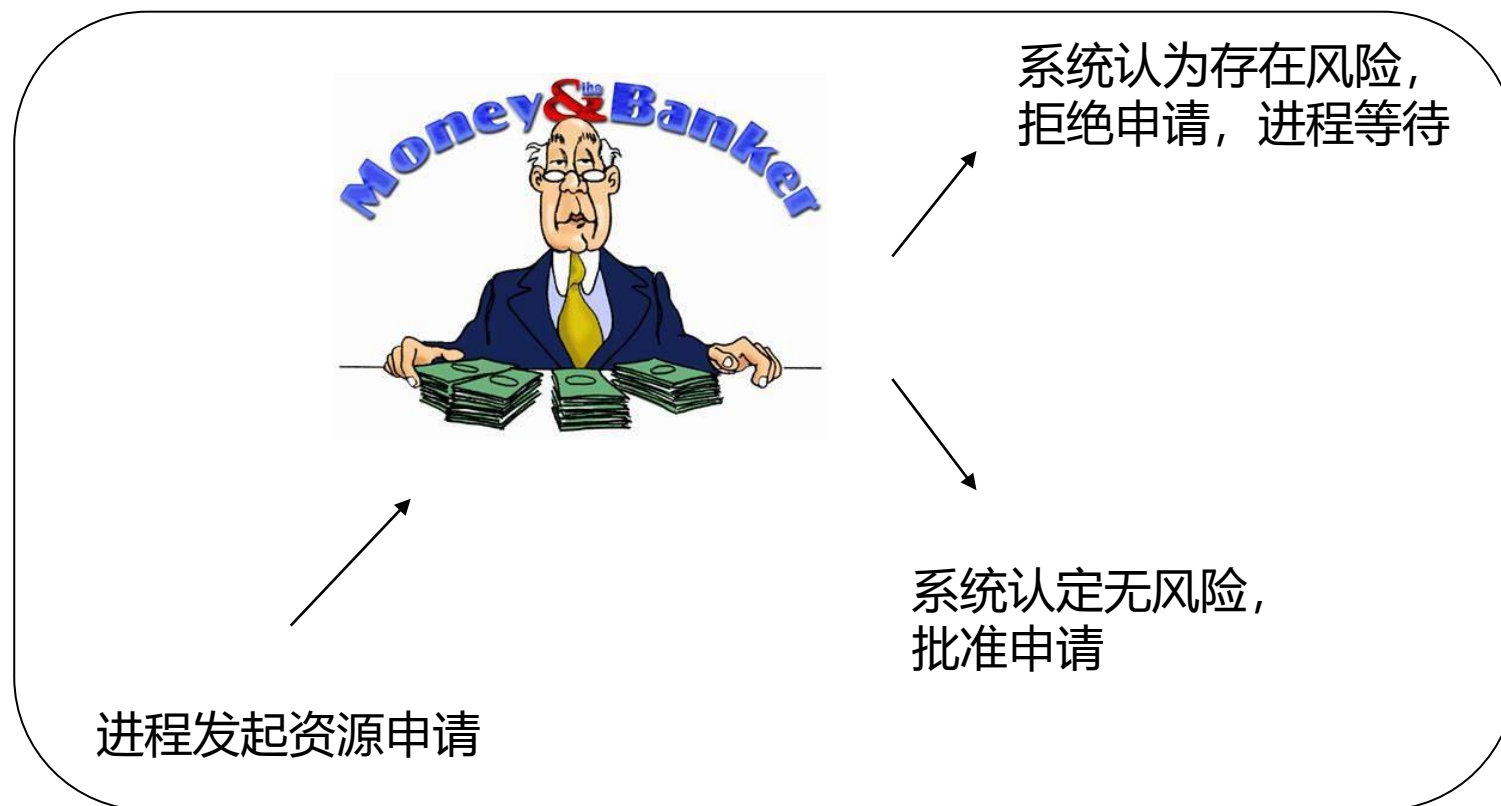
在程序运行起来后，通过对每次资源分配请求进行系统审核，通过拒绝为不安全的资源请求进行分配，来达到避免死锁风险的目的

方法的核心要素：安全性判定

死锁避免算法：银行家算法



dijkstra





银行家算法的基本数据结构

假设系统中有 n 个进程: P_1, P_2, \dots, P_n
有 m 类资源: R_1, R_2, \dots, R_m

- (1) 每个进程明确声明所需最大资源量
- (2) 记录每个进程已获分配的资源量
- (3) 记录每个进程尚需的资源量
- (4) 记录系统剩余可用的资源量
- (5) 描述进程的当前资源申请请求



Max: array[1..n,1..m] of integer;

Allocation: array[1..n,1..m] of integer;

Need: array[1..n,1..m] of integer;

Available: array[1..m] of integer;

Request: array[1..n,1..m] of integer;

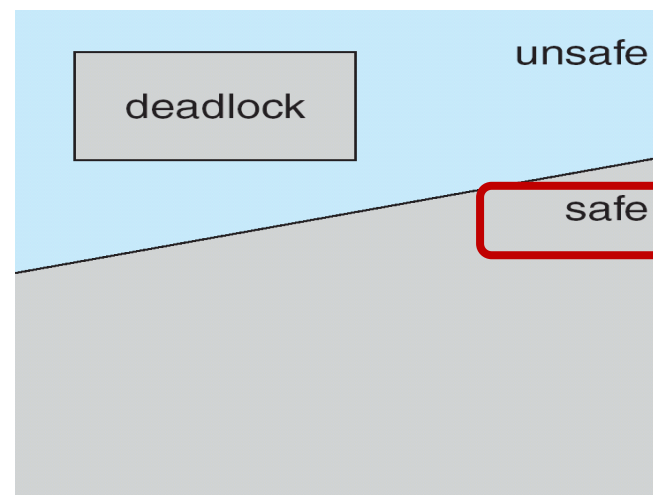


银行家算法：安全判定过程

Local Data Structures:

- Work: vector[1..m], initialize Work to Available
- Finish: vector[1..n], initialize to false for each process i

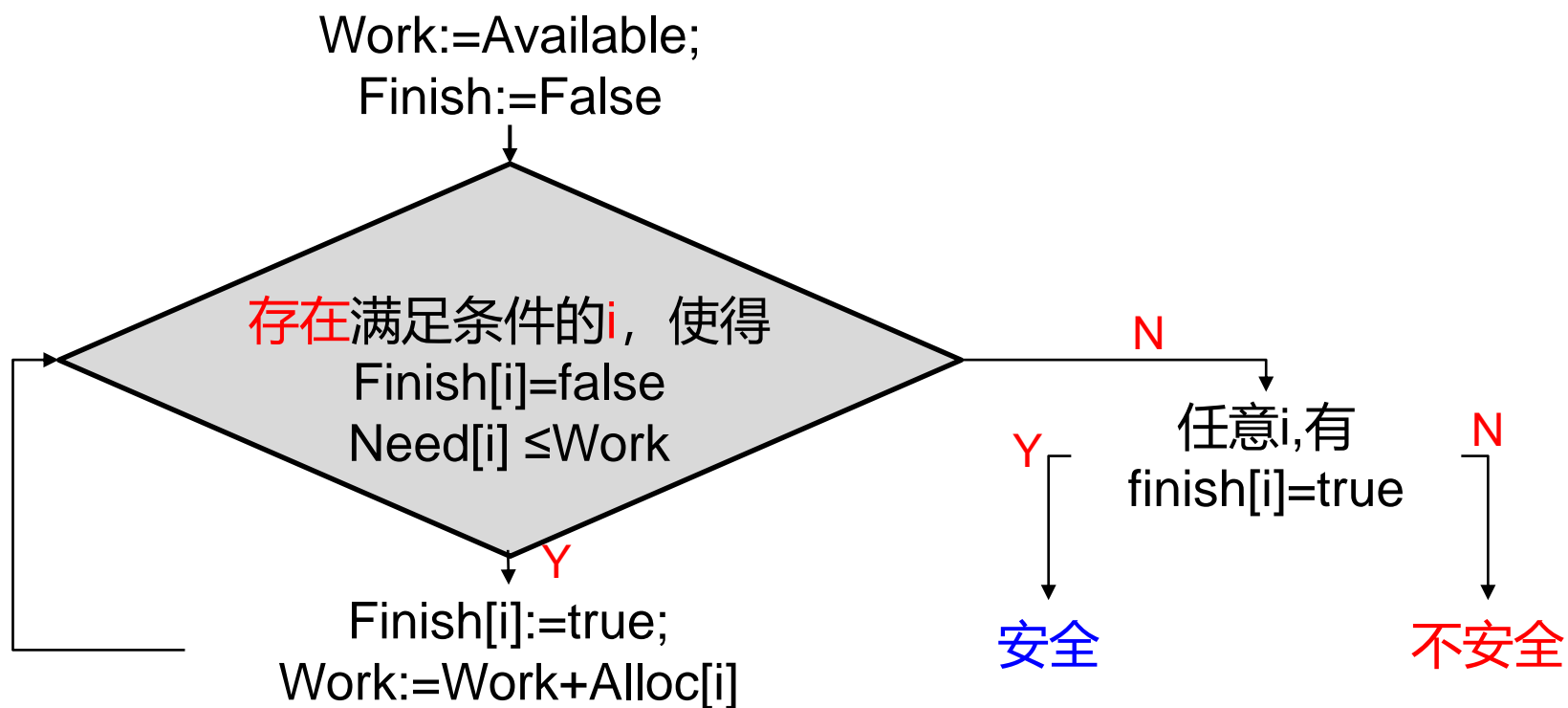
1. Find process i such that $\text{Finish}[i] = \text{false}$ and $\text{Need}[i] \leq \text{Work}$,
if i exists do
 $\text{Work} := \text{Work} + \text{Allocation}$
 $\text{Finish}[i] := \text{true}$
 Go back to 1
2. (if no such i exists any more)
 if $\text{Finish}[i] = \text{true}$ for all i in 1..n, then the system is in a safe state.
 otherwise, the processes whose index is false may potentially be involved in a deadlock in the future.



寻找安全序列之旅



多资源实例条件下的死锁避免：银行家算法





银行家算法: Resource Request

Request[i] – request vector

初始化:

- For each process P_i
- Need[i] = Max[i]
 - Allocation[i] = 0

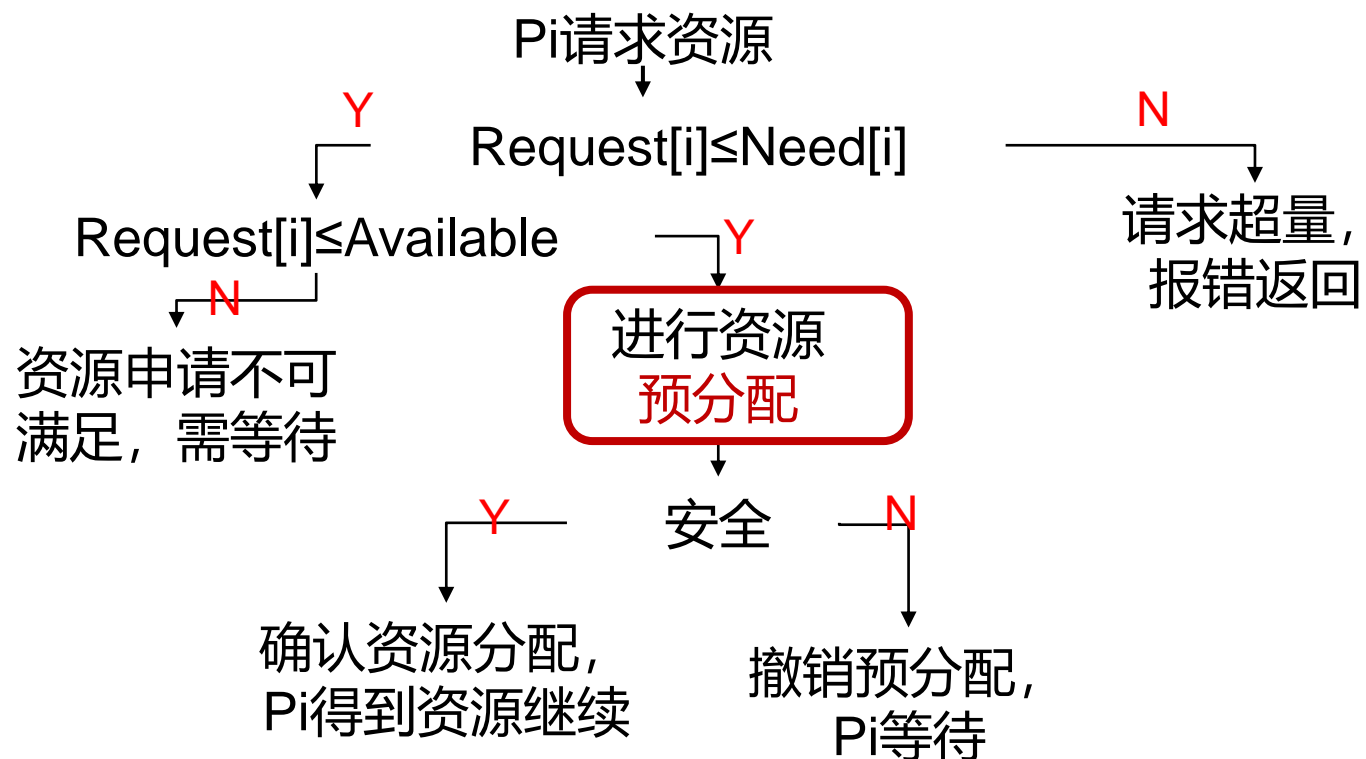
1. If not (Request[i] \leq Need[i]) raise an error
2. If not (Request[i] \leq Available) process i must wait
3. **Tentatively allocate** the requested resources to process i:
 Available := Available - Request[i]
 Allocation[i] := Allocation[i] + Request[i]
 Need[i] := Need[i] - Request[i]
4. Check safety of state. If safe, the resources are allocated.
 If not safe then cancel the tentative allocation and process i must wait.

如果进程请求超过其声明的允许最大量, 则报错

如果进程请求超出了当前系统剩余资源量, 则需等待



资源申请处理流程

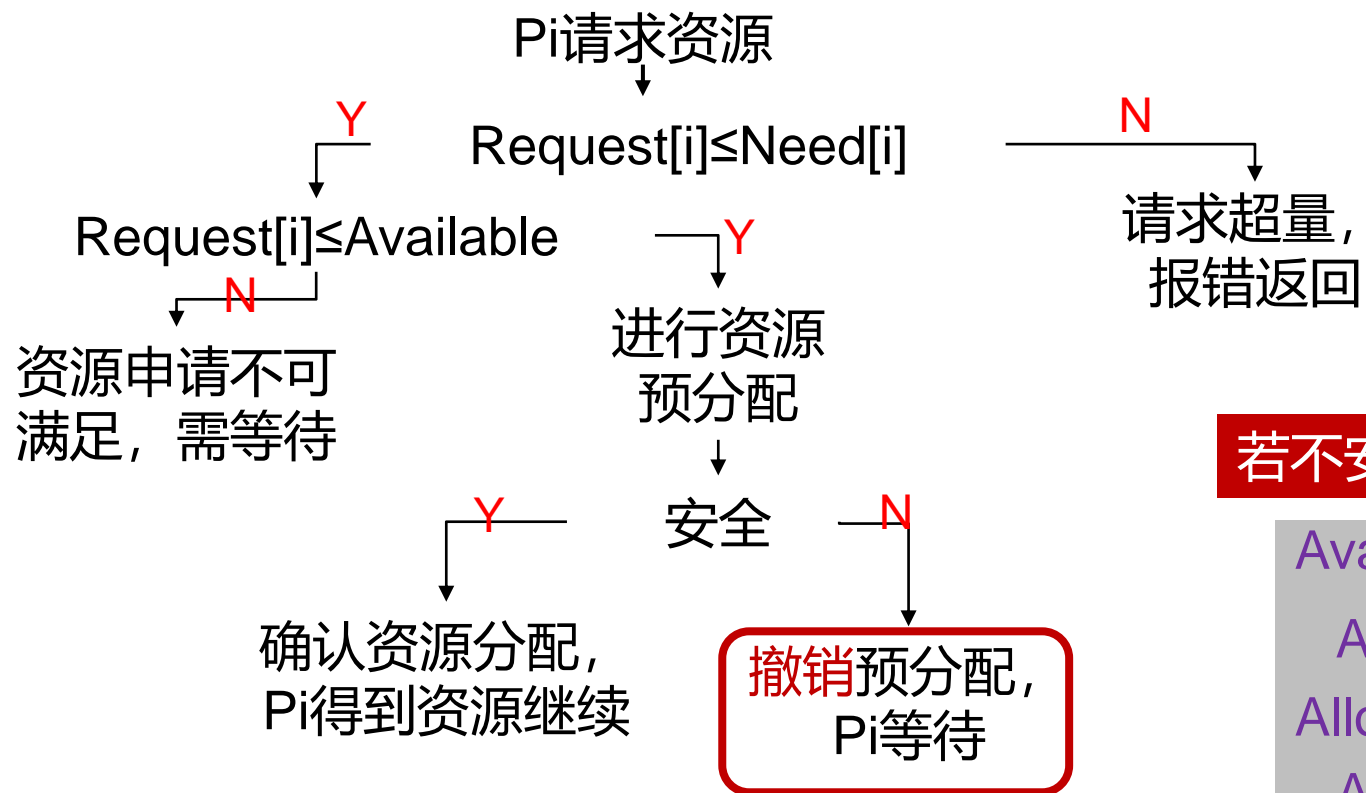


预分配的具体操作如下:

```
Available := Available - Req[I]
Alloc[I] := Alloc[I] + Req[I]
Need[I] := Need[I] - Req[I]
```



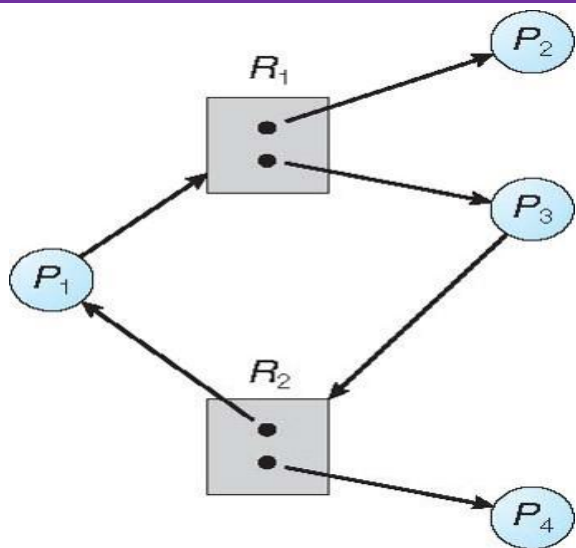
资源申请处理流程



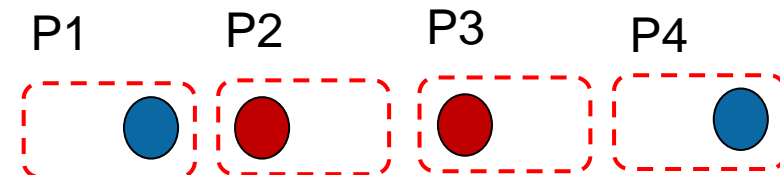
若不安全, 则撤销预分配, 具体操作如下:

```
Available :=  
    Available + Req[I]  
Alloc[I] :=  
    Alloc[I] - Req[I]  
Need[I] :=  
    Need[I] + Req[I]
```

安全判定过程示例



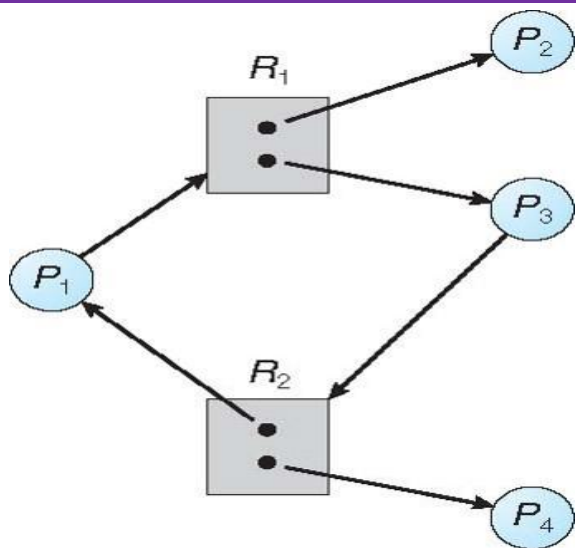
呈现一下安全序列的流程？



初始情况: P2,P4已经获得运行所需全部资源

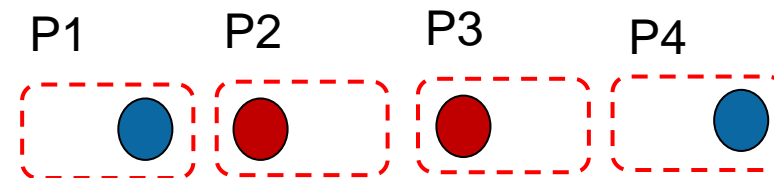
| | <u>Max</u> | | <u>Alloc</u> | | <u>Need</u> | | <u>Available</u> | | <u>Work</u> | | <u>Finish</u> |
|-----|------------|----|--------------|----|-------------|----|------------------|----|-------------|----|---------------|
| | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | |
| P1: | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |
| p2: | 1 | 0 | 1 | 0 | 0 | 0 | | | | | |
| p3: | 1 | 1 | 1 | 0 | 0 | 1 | | | | | |
| p4: | 0 | 1 | 0 | 1 | 0 | 0 | | | | | |

安全判定过程示例



呈现一下安全序列的流程？

初始状态：

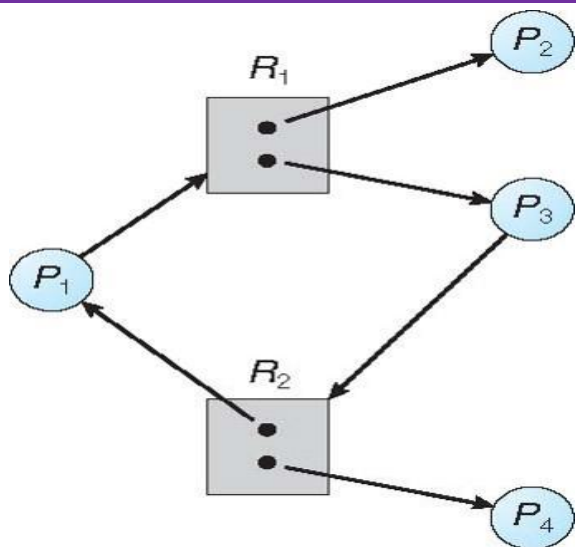


Step1: **let P2 finish**

| | <u>Max</u> | | <u>Alloc</u> | | <u>Need</u> | | <u>Available</u> | | <u>Work</u> | | <u>Finish</u> |
|-----|------------|----|--------------|----|-------------|----|------------------|----|-------------|----|---------------|
| | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | |
| P1: | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |
| p2: | 1 | 0 | 1 | 0 | 0 | 0 | | | | | |
| p3: | 1 | 1 | 1 | 0 | 0 | 1 | | | | | |
| p4: | 0 | 1 | 0 | 1 | 0 | 0 | | | | | |

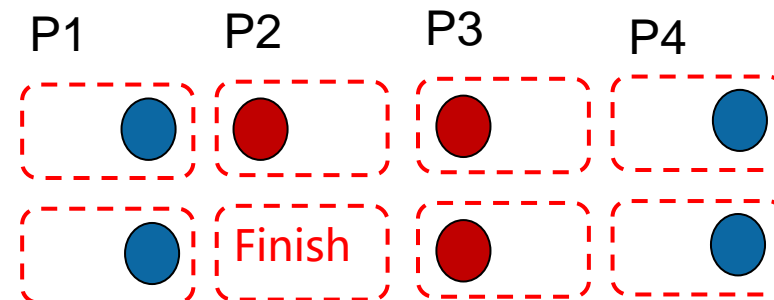
| | | | |
|--|--|--|--|
| | | | |
|--|--|--|--|

安全判定过程示例



呈现一下安全序列的流程？

初始状态:



Step1:

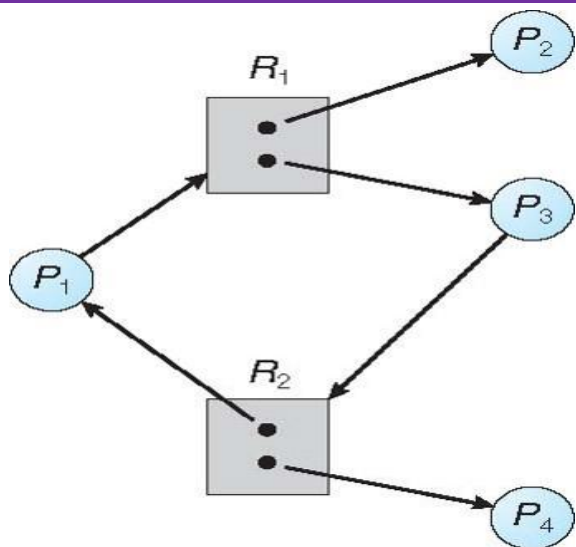
Step2: let P1 finish

| | Max | | Alloc | | Need | | Available | | Work | | Finish |
|-----|-----|----|-------|----|------|----|-----------|----|------|----|--------|
| | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | |
| P1: | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | F |
| p2: | 1 | 0 | 0 | 0 | 0 | 0 | | | | | T |
| p3: | 1 | 1 | 1 | 0 | 0 | 1 | | | | | F |
| p4: | 0 | 1 | 0 | 1 | 0 | 0 | | | | | F |

| | | | |
|----|--|--|--|
| P2 | | | |
|----|--|--|--|

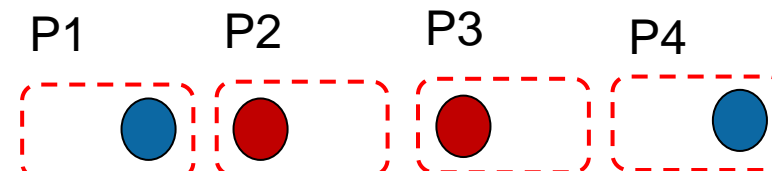
Work: ●

安全判定过程示例



呈现一下安全序列的流程？

初始状态:



Step1:



Step2:



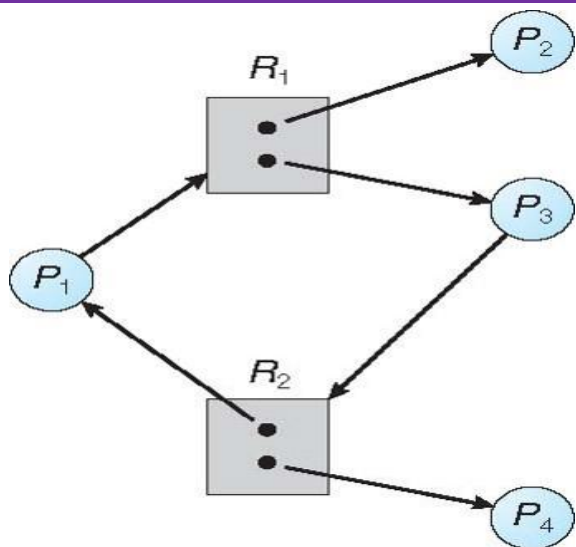
Step3: let P3 finish

| | Max | | Alloc | | Need | | Available | | Work | | Finish |
|-----|-----|----|-------|----|------|----|-----------|----|------|----|--------|
| | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | |
| P1: | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | T |
| p2: | 1 | 0 | 0 | 0 | 0 | 0 | | | | | T |
| p3: | 1 | 1 | 1 | 0 | 0 | 1 | | | | | F |
| p4: | 0 | 1 | 0 | 1 | 0 | 0 | | | | | F |

| | | | |
|----|----|--|--|
| P2 | P1 | | |
|----|----|--|--|

Work:  

安全判定过程示例



呈现一下安全序列的流程？

初始状态:

Step1:

Step2:

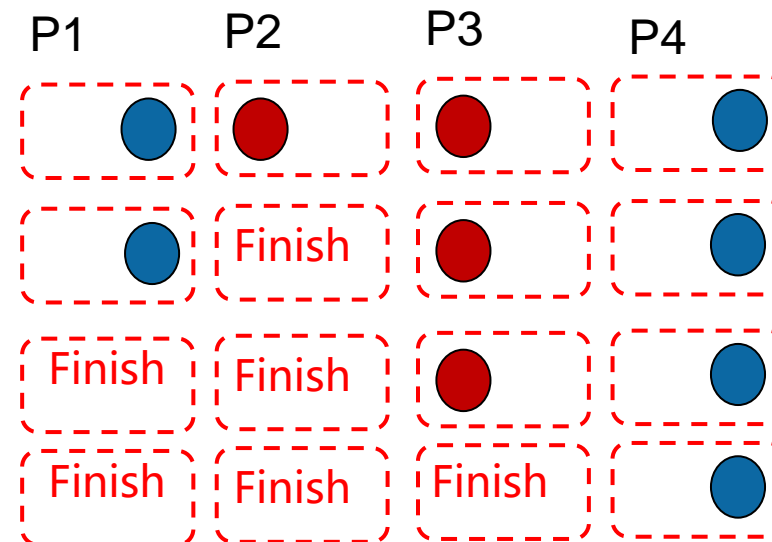
Step3:

Step4: let P4 finish

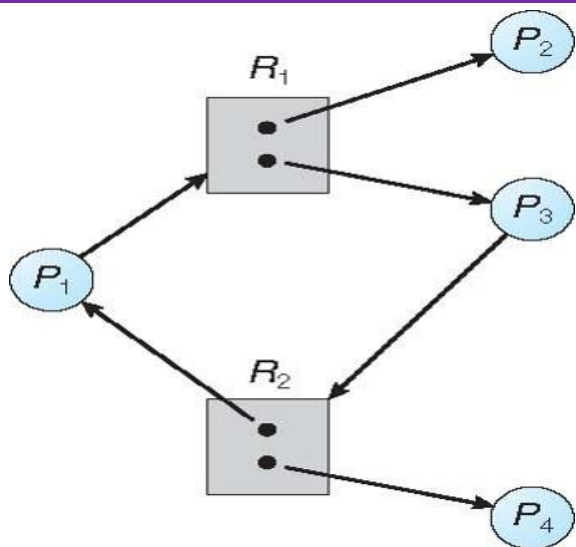
| | Max | | Alloc | | Need | | Available | | Work | | Finish |
|-----|-----|----|-------|----|------|----|-----------|----|------|----|--------|
| | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | |
| P1: | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 1 | T |
| p2: | 1 | 0 | 0 | 0 | 0 | 0 | | | | | T |
| p3: | 1 | 1 | 0 | 0 | 0 | 1 | | | | | T |
| p4: | 0 | 1 | 0 | 1 | 0 | 0 | | | | | F |

| | | | |
|----|----|----|--|
| P2 | P1 | P3 | |
|----|----|----|--|

Work: ● ● ●



安全判定过程示例



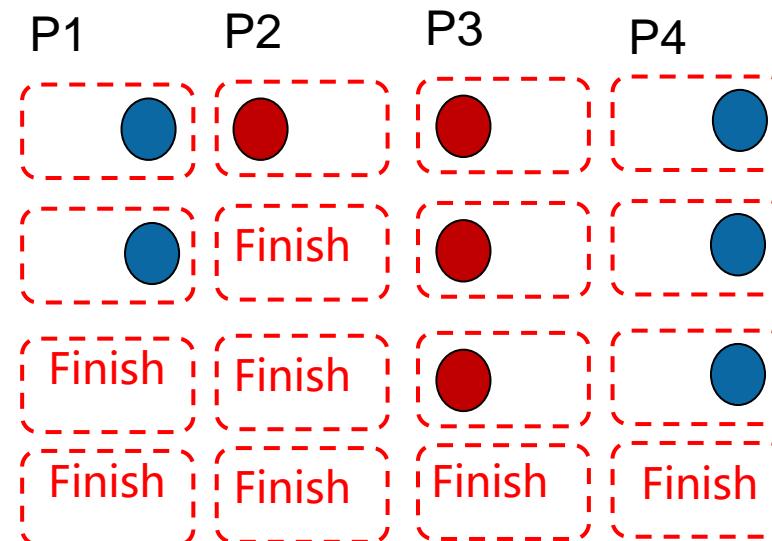
呈现一下安全序列的流程？

初始状态:

Step1:

Step2:

Step3:







Done!

成功找到安全序列!



| | Max | | Alloc | | Need | | Available | | Work | | Finish |
|-----|-----|----|-------|----|------|----|-----------|----|------|----|--------|
| | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 | |
| P1: | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 2 | T |
| p2: | 1 | 0 | 0 | 0 | 0 | 0 | | | | | T |
| p3: | 1 | 1 | 0 | 0 | 0 | 1 | | | | | T |
| p4: | 0 | 1 | 0 | 0 | 0 | 0 | | | | | T |

| | | | |
|----|----|----|----|
| P2 | P1 | P3 | P4 |
|----|----|----|----|

Work:    



资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| P0: | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 | | | | |
| p1: | 3 | 2 | 2 | 2 | 0 | 0 | 1 | 2 | 2 | | | | | | | |
| p2: | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | |
| p3: | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | |
| p4: | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | |

进程P1提出资源申请Request[1]=(1,0,2)
系统应如何处理

步骤1: 判定当前状态是否安全



步骤2: 实施资源预分配



步骤3: 预分配后状态是否安全



资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| P0: | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 | 3 | 3 | 2 | F |
| p1: | 3 | 2 | 2 | 2 | 0 | 0 | 1 | 2 | 2 | | | | | | | F |
| p2: | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | F |
| p3: | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | F |
| p4: | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | F |

步骤1: 判定当前状态是否安全

Step1: 从满足 $need \leq work$ 的进程中选一个, 我们选 p1



资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p0,p1,p2,p3,p4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| P0: | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 | 5 | 3 | 2 | F |
| p1: | 3 | 2 | 2 | 2 | 0 | 0 | 1 | 2 | 2 | | | | | | | T |
| p2: | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | F |
| p3: | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | F |
| p4: | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | F |

安全序列: p1

Work变量的变化: 3 3 2 \rightarrow 5 3 2

步骤1: 判定当前状态是否安全

Step1: 从满足 $need \leq work$ 的进程中选一个, 我们选p1 (p1预计获取所有资源结束)

Step2: 从满足 $need \leq work$ 的进程中选一个, 这次选p3



资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| p0: | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 | 7 | 4 | 3 | F |
| p1: | 3 | 2 | 2 | 2 | 0 | 0 | 1 | 2 | 2 | | | | | | | T |
| p2: | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | F |
| p3: | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | T |
| p4: | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | F |

安全序列: p1 p3

Work变量的变化: 3 3 2 \rightarrow 5 3 2 \rightarrow 7 4 3

步骤1: 判定当前状态是否安全

Step1: 从满足 $need \leq work$ 的进程中选一个, 我们选 p1

Step2: 从满足 $need \leq work$ 的进程中选一个, 这次选 p3 (p3预计获取所有资源结束)

Step3: 从满足 $need \leq work$ 的进程中选一个, 这次选 p4

资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| P0: | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 | 7 | 4 | 5 | F |
| p1: | 3 | 2 | 2 | 2 | 0 | 0 | 1 | 2 | 2 | | | | | | | T |
| p2: | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | F |
| p3: | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | T |
| p4: | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | T |

步骤1: 判定当前状态是否安全

Step1:从满足 $need \leq work$ 的进程中选一个, 我们选p1

Step2:从满足 $need \leq work$ 的进程中选一个, 这次选p3

Step3:从满足 $need \leq work$ 的进程中选一个, 这次选p4 (p4预计获取所有资源结束)

Step4:从满足 $need \leq work$ 的进程中选一个, 这次选p2

安全序列: p1 p3 p4

Work变量的变化: $\boxed{3 \ 3 \ 2} \rightarrow \boxed{5 \ 3 \ 2} \rightarrow \boxed{7 \ 4 \ 3} \rightarrow \boxed{7 \ 4 \ 5}$

资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|---------|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| p_0 : | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 | 10 | 4 | 7 | F |
| p_1 : | 3 | 2 | 2 | 2 | 0 | 0 | 1 | 2 | 2 | | | | | | | T |
| p_2 : | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | T |
| p_3 : | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | T |
| p_4 : | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | T |

安全序列: $p_1 \ p_3 \ p_4 \ p_2$

Work变量的变化: $\boxed{3 \ 3 \ 2} \rightarrow \boxed{5 \ 3 \ 2} \rightarrow \boxed{7 \ 4 \ 3} \rightarrow \boxed{7 \ 4 \ 5} \rightarrow \boxed{10 \ 4 \ 7}$

步骤1: 判定当前状态是否安全

Step1: 从满足 $need \leq work$ 的进程中选一个, 我们选 p_1

Step2: 从满足 $need \leq work$ 的进程中选一个, 这次选 p_3

Step3: 从满足 $need \leq work$ 的进程中选一个, 这次选 p_4

Step4: 从满足 $need \leq work$ 的进程中选一个, 这次选 p_2 (p_2 预计获取所有资源结束)

Step5: 从满足 $need \leq work$ 的进程中选一个, 这次选 p_0

资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| P0: | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 | 10 | 5 | 7 | T |
| p1: | 3 | 2 | 2 | 2 | 0 | 0 | 1 | 2 | 2 | | | | | | | T |
| p2: | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | T |
| p3: | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | T |
| p4: | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | T |

安全序列: p1 p3 p4 p2 p0

Work变量的变化:



步骤1: 判定当前状态是否安全

成功找到安全序列，确认状态安全！

Step1:从满足 $need \leq work$ 的进程中选一个，我们选p1

Step2:从满足 $need \leq work$ 的进程中选一个，这次选p3

Step3:从满足 $need \leq work$ 的进程中选一个，这次选p4

Step4:从满足 $need \leq work$ 的进程中选一个，这次选p2

Step5:从满足 $need \leq work$ 的进程中选一个，这次选p0 (p0预计获取所有资源结束)



资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| P0: | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 | | | | |
| p1: | 3 | 2 | 2 | 2 | 0 | 0 | 1 | 2 | 2 | | | | | | | |
| p2: | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | |
| p3: | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | |
| p4: | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | |

进程P1: Request[1]=(1,0,2)

步骤1: 判定当前状态是否安全



步骤2: 实施资源预分配



资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| P0: | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 2 | 3 | 0 | | | | |
| p1: | 3 | 2 | 2 | 3 | 0 | 2 | 0 | 2 | 0 | | | | | | | |
| p2: | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | |
| p3: | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | |
| p4: | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | |

进程P1: Request[1]=(1,0,2)

步骤1: 判定当前状态是否安全



步骤2: 实施资源预分配



步骤3: 预分配后状态是否安全

课堂练习: 请尝试看看能否找出一个安全序列



资源申请处理过程示例

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

| | <u>Max</u> | | | <u>Alloc</u> | | | <u>Need</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|------------|---|---|--------------|---|---|-------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | A | B | C | |
| P0: | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 2 | 3 | 0 | | | | |
| p1: | 3 | 2 | 2 | 3 | 0 | 2 | 0 | 2 | 0 | | | | | | | |
| p2: | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | | | | | | | |
| p3: | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | | | | | | | |
| p4: | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 | | | | | | | |

进程P1: Request[1]=(1,0,2)

找到安全进程序列: <p1,p3,p4,p0,p2> ➡ 安全

步骤1: 判定当前状态是否安全



步骤2: 实施资源预分配



步骤3: 预分配后状态是否安全



银行家算法的保守型：示例

例子： $R=\{A(1),B(1)\}$, 申请A, B; 释放 \bar{A} , \bar{B}

$P=\{p1,p2\}$, $p1: A B \bar{A} \bar{B}$; $p2: B \bar{B} B A \bar{A} \bar{B}$

| | <u>Claim</u> | | <u>Allocation</u> | | <u>Need</u> | | <u>Available</u> | | <u>Work</u> | | <u>Finish</u> |
|-----|--------------|---|-------------------|---|-------------|---|------------------|---|-------------|---|---------------|
| | A | B | A | B | A | B | A | B | A | B | |
| p1: | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | | | |
| p2: | 1 | 1 | 0 | 0 | 1 | 1 | | | | | |

$Request[1]=(1,0)$, 安全, 分配。



银行家算法的保守型：示例

p1: a b \overline{a} \overline{b} ; p2: b \overline{b} b a \overline{a} \overline{b}

分配后：

| | <u>Claim</u> | | <u>Allocation</u> | | <u>Need</u> | | <u>Available</u> | | <u>Work</u> | | <u>Finish</u> |
|-----|--------------|---|-------------------|---|-------------|---|------------------|---|-------------|---|---------------|
| | A | B | A | B | A | B | A | B | A | B | |
| p1: | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | | | |
| p2: | 1 | 1 | 0 | 0 | 1 | 1 | | | | | |

若P2此时提出Request[2]=(0,1),
则银行家算法认定不安全，不分配

(实际上针对此示例，此时满足P2的申请并不导致死锁)

死锁检测

Deadlock Detection

03

Deadlock



死锁真实发生



检测死锁



解除死锁

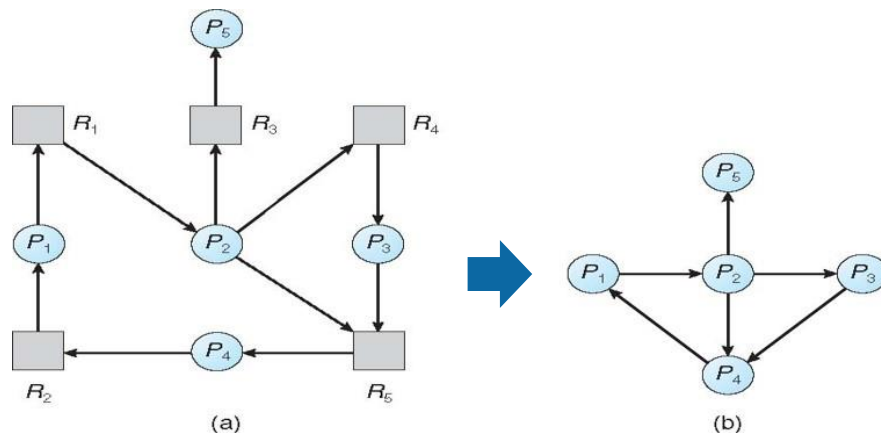
单资源实例的死锁检测算法

多资源实例的死锁检测算法

实施方式：在系统运转过程中，定期进行死锁检测，判断是否有死锁发生

单资源实例条件下的死锁检测

基本方法：以资源分配图为基础，构建进程等待图



(a)资源分配图

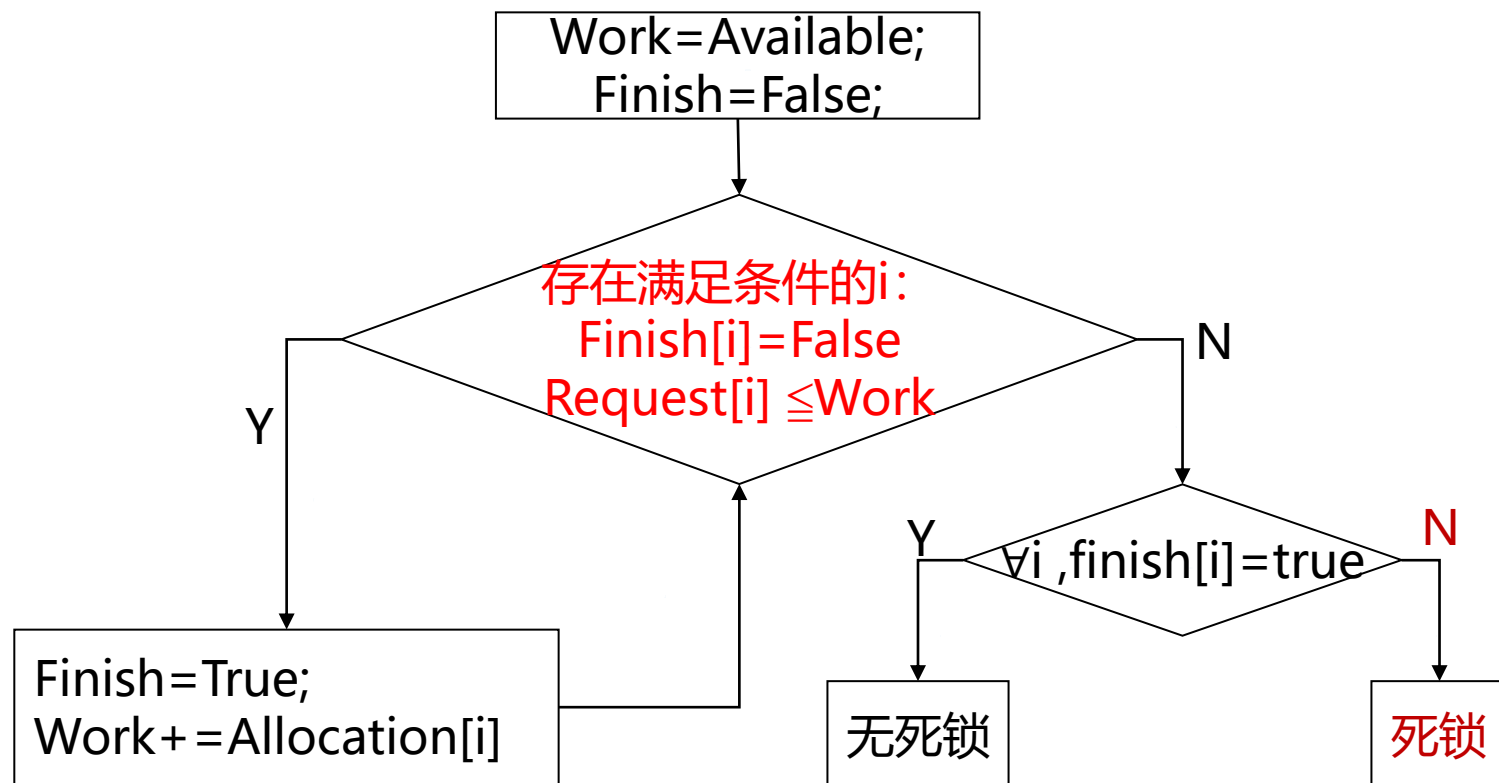
(b)进程等待图

死锁检测问题转化为对进程等待图的环路检测问题

实施方式：定期启动对进程等待图的环路检测，如果发现等待图中有环，那么报告死锁发生



多资源实例条件下的死锁检测





多资源实例条件下的死锁检测

示例: $R=\{A(7),B(2),C(6)\}$; $P=\{p_0,p_1,p_2,p_3,p_4\}$

| | <u>Allocation</u> | | | <u>Request</u> | | | <u>Available</u> | | | <u>Work</u> | | | <u>Finish</u> |
|-----|-------------------|---|---|----------------|---|---|------------------|---|---|-------------|---|---|---------------|
| | A | B | C | A | B | C | A | B | C | A | B | C | |
| p0: | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | |
| p1: | 2 | 0 | 0 | 2 | 0 | 2 | | | | | | | |
| p2: | 3 | 0 | 3 | 0 | 0 | 0 | | | | | | | |
| p3: | 2 | 1 | 1 | 1 | 0 | 0 | | | | | | | |
| p4: | 0 | 0 | 2 | 0 | 0 | 2 | | | | | | | |

未死锁。

若此时, p2提出Request[2]=(0,0,1), 死锁, 参与死锁进程{p1,p2,p3,p4}



由于死锁检测算法的开销较大，所以不可能频繁进行死锁检测

检测时机：

1. 当进程阻塞时，系统实施检测
2. 定时检测
3. 系统资源利用率下降时检测死锁



必须用解除死锁方法，与死锁检测方法配套

死锁解除手段：

1. 重新启动
2. 撤销进程
3. 剥夺进程资源
4. 进程回退



视而不见

- Pro:
 - ▶ 工程师观点
 - 死锁发生频率 < 其它故障引起的系统瘫痪的频率
 - 死锁处理 constant overhead > 危害
- Cont:
 - ▶ 数学家观点: 必须处理, 无论代价如何

目前通用桌面操作系统实际采取**鸵鸟策略**



小结:  死锁处理方法

 死锁避免

 死锁检测



死锁检测的成本过高。



在一个繁忙的十字路口，每个方向只有一个车道，如果车辆只能向前直行，而不允许转弯和退后，并且没有任何方式进行交通管理。

- (1) 该十字路口是否会产生交通死锁？若可能，请画出交通死锁的示意图。
- (2) 说明产生死锁的4个必要条件在此处成立。
- (3) 提出一个避免死锁的简单规则。



某系统采用了银行家算法，则下列叙述正确的是（ ）。

- A. 系统处于不安全状态时一定会发生死锁
- B. 系统处于不安全状态时可能会发生死锁
- C. 系统处于安全状态时，可能会发生死锁
- D. 系统处于安全状态时，一定会发生死锁



若有10个同类资源供三个进程分配使用，下表列出了这三个进程目前已占资源和最大需求量的情况，现在这三个进程P1、P2、P3又分别申请1个、2个、1个资源，请问：能否先满足P2的要求？

P1: 已占资源数 (3) , 最大需求量 (8)

P2: 已占资源数 (3) , 最大需求量 (8)

P3: 已占资源数 (2) , 最大需求量 (3)

系统中有5个进程P0,P1,P2,P3,P4，竞争使用3类资源R1、R2、R3，资源总数分别为18,6,22。当前时刻的资源分配情况如表所示。（ ）是此时存在的一个安全序列。

| 进程 | 已分配资源 | | | 资源最大需求 | | |
|----|-------|----|----|--------|----|----|
| | R1 | R2 | R3 | R1 | R2 | R3 |
| P0 | 3 | 2 | 3 | 5 | 5 | 10 |
| P1 | 4 | 0 | 3 | 5 | 3 | 6 |
| P2 | 4 | 0 | 5 | 4 | 0 | 11 |
| P3 | 2 | 0 | 4 | 4 | 2 | 5 |
| P4 | 3 | 1 | 4 | 4 | 2 | 4 |

- A. P0, P2, P4, P1, P3
- B. P1, P0, P3, P4, P2
- C. P2, P1, P0, P3, P4
- D. P3, P4, P2, P1, P0



谢谢!
Thank you!